OPIC
OFFICE DE LA PROPRIÉTÉ INTELLECTUELLE DU CANADA

CIPO
CANADIAN INTELLECTUAL PROPERTY OFFICE

(12) (19) (CA) **Demande-Application**

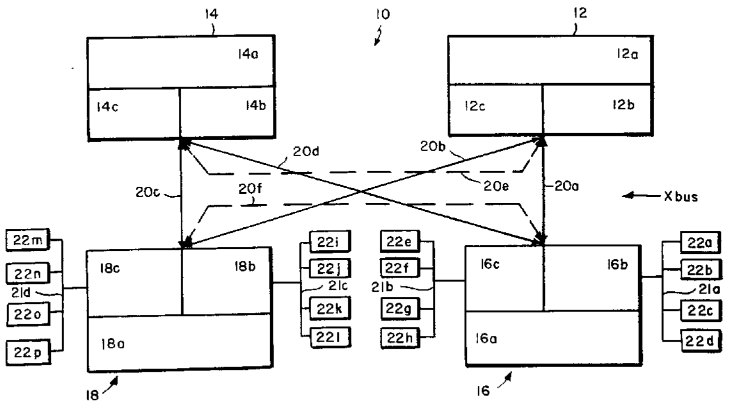
(21) (A1) **2,257,511**
(86) 1997/06/05
(87) 1997/12/11

(72) LEAVITT, William I., US
(72) CLEMSON, Conrad R.., US
(72) SOMERS, Jeffrey S., US
(72) CHAVES, John M., US
(72) BARBERA, David R., US
(72) CLAYTON, Shawn A., US
(71) STRATUS COMPUTER, US

(51) Int.Cl.^{6} G06F 11/16, G06F 11/30, G06F 11/22
(30) 1996/06/05 (08/658,563) US

(54) **PROCEDES ET DISPOSITIF DE TRAITEMENT DE DONNEES NUMERIQUES POUR L'ISOLATION DE DEFAUTS**

(54) **DIGITAL DATA PROCESSING METHODS AND APPARATUS FOR FAULT ISOLATION**



(57) L'invention concerne un dispositif de traitement de données à isolation de défauts comprenant plusieurs unités fonctionnelles (12-18) reliées par plusieurs bus (20a-20d) pour les communications point à point. Ces unités (12-18) contrôlent les bus (20a-20d) auxquels elles sont reliées et s'informent réciproquement en cas d'erreurs de communications au niveau du bus. Elles (12-18) peuvent entrer simultanément dans une phase d'isolation d'erreur, par exemple suite à une erreur de bus signalée par une de ces unités. Outre la signalisation des erreurs de bus, ces unités fonctionnelles indiquent les défauts l'échelon de l'unité (ou de la "carte") si elles décèlent un défaut dans leur fonctionnement propre. On prévoit chaque unité (12 ou 14) des fonctions d'isolation d'erreur signalant un défaut selon les critères suivants: (i) l'unité a indiqué une erreur de signal de retour liée à son fonctionnement propre; (ii) l'unité considérée ou une autre unité a signalé une erreur de bus pendant la phase d'isolation d'erreur; et/ou (iii) toute autre unité fonctionnelle a signalé que son propre fonctionnement était défectueux durant la phase d'isolation d'erreur.

(57) A fault-isolating digital data processing apparatus includes plural functional units (12-18) that are interconnected for point-to-point communications by a plurality of buses (20a-20d). The functional units (12-18) monitor the buses (20a-20d) to which they are attached and signal the other units in the event there are bus communication errors. The functional units (12-18) can simultaneously enter into an error isolation phase, e.g., in response to a bus error signalled by one of the units. In addition to signalling bus errors, the functional units can signal unit-level (or "board") faults when they detect fault in their own operation. Each unit (12 or 14) includes error isolation functionality that signals a fault based on (i) whether that unit (12 or 14) signalled a loopback error with respect to its own operation; (ii) whether that unit or another unit signalled a bus error during the error isolation phase; and/or (iii) whether any other functional unit signalled that it was faulty during the error isolation phase.

CORRECTED VERSION*

PCT

WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 : G06F 11/00, 11/34, 11/08, 11/16, 11/22 A1

(11) International Publication Number: WO 97/46941

(43) International Publication Date: 11 December 1997 (11.12.97)

(21) International Application Number: PCT/US97/09781

(22) International Filing Date: 5 June 1997 (05.06.97)

(30) Priority Data:
08/658,563 5 June 1996 (05.06.96) US

(81) Designated States: AU, CA, JP, European patent (AT, BE, CH, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE).

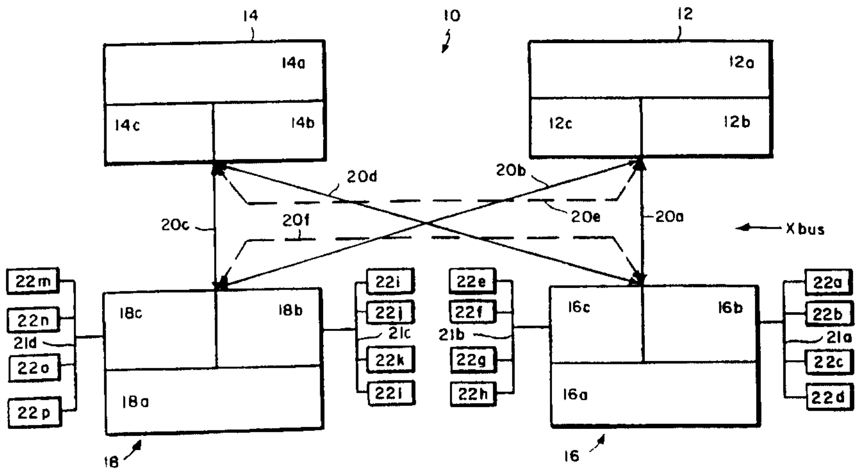
Published
With international search report.
Before the expiration of the time limit for amending the claims and to be republished in the event of the receipt of amendments.

(71) Applicant: STRATUS COMPUTER [US/US]; 55 Fairbanks Boulevard, Marlboro, MA 01752 (US).

(72) Inventors: LEAVITT, William, I.; 180 Grove Street, Lexington, MA 02173 (US). CLEMSON, Conrad, R.; 1 Bowdoin Street, Shrewsbury, MA 01545 (US). SOMERS, Jeffrey, S.; 1 Scott Lane, Northboro, MA 01532 (US). CHAVES, John, M.; 7 Cornish Drive, Hudson, MA 01749 (US). BARBERA, David, R.; 15 Hapgood Road, Worcester, MA 01605 (US). CLAYTON, Shawn, A.; 22 Lee Street #2, Worcester, MA 01602 (US).

(74) Agent: POWSNER, David, J.; Choate, Hall & Stewart, Exchange Place, 53 State Street, Boston, MA 02109 (US).

(54) Title: DIGITAL DATA PROCESSING METHODS AND APPARATUS FOR FAULT ISOLATION



(57) Abstract

A fault-isolating digital data processing apparatus includes plural functional units (12-18) that are interconnected for point-to-point communications by a plurality of buses (20a-20d). The functional units (12-18) monitor the buses (20a-20d) to which they are attached and signal the other units in the event there are bus communication errors. The functional units (12-18) can simultaneously enter into an error isolation phase, e.g., in response to a bus error signalled by one of the units. In addition to signalling bus errors, the functional units can signal unit-level (or "board") faults when they detect fault in their own operation. Each unit (12 or 14) includes error isolation functionality that signals a fault based on (i) whether that unit (12 or 14) signalled a loopback error with respect to its own operation; (ii) whether that unit or another unit signalled a bus error during the error isolation phase; and/or (iii) whether any other functional unit signalled that it was faulty during the error isolation phase.

*(Referred to in PCT Gazette No. 12/1998, Section II)

DIGITAL DATA PROCESSING METHODS AND APPARATUS FOR FAULT ISOLATION

Reference to Related Applications

This application is a continuation in part of United States Patent Application Serial No. 08/309,210, filed September 20, 1994, the teachings of which are incorporated herein by reference.

Reservation of Copyright

Background of the Invention

The invention disclosed and claimed herein relates to digital data processing and, more particularly, to methods and apparatus for fault detection and isolation.

A shortcoming of many conventional digital data processors is their inability to detect and isolate a range of data transfer and operational faults. Personal computers and workstations, for example, are typically configured to detect only a single class of data transfer faults, such as, parity errors. When detected, such a fault can cause the computer or workstation to halt operation, or "crash." Larger computer systems often incorporate error-correcting codes to detect and correct single-bit errors. These systems can be equipped to continue operation even in the event of fault.

Computer systems marketed and described in prior patents to the assignee hereof are capable of both detecting, correcting, and continuing operation after faults. U.S. Patent No. 4,750,177, for example, discloses a fault-tolerant digital data processor having a first functional unit, such as a central processor, with duplicate processing sections coupled to a common system bus for identically processing signals received from other functional units, such as the memory or peripheral device controller. In addition to checking the received data, the first functional unit compares the output generated by the sections while one of them -- the so-called "drive" section -- transmits processed data to the bus. When the section outputs do not agree, the functional unit drives an error signal onto the bus and takes itself off-line. According to the patent, a functional unit such as a central processor can have a redundant partner that is constructed and operates identically to the original. In such a configuration, if one of the partners is taken off-line due to error, processing continues with the partner.

According to U.S. Patent No. 4,931,922, also assigned to the assignee hereof, redundant peripheral control units, each with duplicate processing sections, control the latching of data and signaling of errors vis-à-vis data transfers with attached peripheral devices. For this purpose, data signals applied to the peripheral device bus by either the control units or peripheral devices are captured and compared by processing sections within each control unit. The results of those comparisons are shared between the controllers. If the comparisons indicate that the data captured by both control units agrees, then control units generate a "strobe" signal that causes the data to be latched. If the units do not agree after successive retries, the control units withhold issuance of the "strobe" signal and enter an error-handling state for determining the source of the error.

Co-pending, commonly assigned U.S. Patent Application 07/926,857 discloses, in one aspect, a digital data processing system having dual, redundant processor elements, each including dual, redundant processing sections that operate in lock-step synchronism. Failure detection stages in each section assemble signals generated by that section into first/second and third/fourth groups, respectively. Normally, signals in the first group match those of the third group, while the signals in the second group match those of the fourth group. The groups are routed between the sections along conductor sets separate from the system bus. To detect fault, each stage compares a respective one of the groups with its counterpart. If either stage detects a mismatch, it signals an error to its respective processing section, which can take the corresponding processor element off-line.

While the above-mentioned patents and patent applications describe systems with high degrees of fault detection and fault tolerance, such capabilities are not required for all digital data processing systems. While fault tolerance can be sacrificed in more moderately priced systems, fault detection cannot be. Indeed, in systems that lacking fault tolerance, it is desirable not only to detect fault, but to isolate its source.

An object of this invention, therefore, is to provide digital data processing apparatus and methods with improved fault-detecting capabilities.

A related object is to provide such apparatus and methods with fault isolation capabilities, i.e., that of detecting functional units that are the sources of faults and taking those units off-line.

Further objects of the invention are to provide such apparatus and methods that can be readily implemented without excessive cost.

A related object is to provide such apparatus and methods as can be implemented in high-speed hardware elements, such as ASIC's.

Summary of the Invention

The aforementioned objects are among those met by the invention, which provides in one aspect fault-isolating digital data processing apparatus including plural functional units that are interconnected for point-to-point communications by a plurality of buses. A system with two central processing units (CPU's) and two input/output (I/O) interface units, for example, can employ six bidirectional buses to couple each unit to the others. The functional units monitor the buses to which they are attached and signal the other units in the event there are bus communication errors. Thus, for example, if one of the functional units detects a parity or error correcting code (ECC) error during the transmission of data, that unit can signal a bus error the other units.

The functional units can simultaneously enter into an error isolation phase, e.g., in response to a bus error signalled by one of the units. During this phase, each unit transmits test data (e.g., predetemined patterns of 0's and 1's) onto at least one of its attached buses. Though more than one unit can transmit test data at a time, only one unit can do so on a given bus. For example, in a system with two CPU's and two I/O interface units, both CPU's can simultaneously drive test data over the buses that couple them to the interface units. Later in the phase, both I/O units can drive test patterns to the CPU's back over those same buses.

The functional units continue to monitor the buses and to signal bus errors while the test data is being transmitted. Each unit does this not only when another is transmitting test data on a common, point-to-point bus, but also when the unit itself is driving such test data. The latter affords each unit the opportunity to perform a loopback comparison to match the data that it intended to drive onto a bus with that actually "received" from the bus through monitoring. For example, in a system as described above, each CPU monitors its buses for error while the I/O units are driving test data and, in addition, while the CPU itself is driving such data. Moreover, each CPU compares the test data that it intended to drive on the bus with that which it receives from the bus.

In addition to signalling bus errors, the functional units can signal unit-level (or "board") faults when they detect fault in their own operation. To this end, each unit includes error isolation functionality that signals a fault based on (i) whether that unit signalled a loopback error with respect to its own operation; (ii) whether that unit or another unit signalled a bus error during the error isolation phase; and/or (iii) whether any other functional unit signaled that it was faulty during the error isolation phase. By way of example, in a system as described above, a CPU can signal the others that it is faulty if it detects a loopback comparison mismatch while driving test data onto the buses. By way of further example, that CPU can signal the others that it is faulty if it and the other functional units detect a bus error during a phase when that CPU drives test data, but not when the I/O units drive test data onto that bus. A unit that determines itself fault can, in addition to signalling the other units, take itself off-line.

According to further aspects of the invention, any of the foregoing functional units can include a processing section that generates data for communication to another unit and that includes dual interfaces for applying that data to the associated buses. The interfaces, referred to as the "drive side" and "check side" interfaces, apply to the buses complimentary portions of each datum generated by the processing section. In the foregoing example, a CPU can use dual interface sections to drive one half of each datum generated by its processing section onto the buses.

The drive side and check side interfaces also receive data driven to the buses, whether by that functional unit or another. With respect to data driven by the functional unit, data received by drive and check sides can be matched against the data intended to be driven by the unit for purposes of the loopback comparison. Thus, for example, portions of a word received from the bus by the check side interface can be compared with the portions which had been applied by that interface as part of a "loopback drive check." Likewise, the other portions of a word received from the bus by the check side interface can be compared against those which had been applied by the other interface (i.e., the drive side interface) as part of a "loopback compare check."

Still further aspects of the invention provide methods for fault-isolating digital data processor operation paralleling the processes described above.

These and other aspects of the invention are evident in the drawings and in the description that follows.

Brief Description of the Drawings

A more complete understanding of the invention may be attained by reference to the drawings, in which:

Figure 1 depicts a digital data processing system employing the invention;

Figure 2 depicts the data buses of an "Xbus" according to the invention;

Figure 3 depicts the control buses of an Xbus;

Figure 4 depicts a bit numbering scheme for an Xbus;

Figures 5 and 6 illustrate simple word, read and line write transactions on an Xbus;

Figure 7 depicts the phases of an operation on an Xbus;

Figure 8 depicts the effect of bus BUSYs on the basic bus cycle of an Xbus;

Figure 9 depicts the effect of bus errors on an Xbus;

Figure 10 depicts the basic state machine and state transitions for a bus handler used in connection with an Xbus;

Figure 11 depicts the interconnection between the Xbus and two functional units, or boards, connected therewith;

Figure 12 depicts self-checking parity logic used in connection with an Xbus;

Figure 13 depicts loopback connectivity used in connection with practice of the invention;

Figures 14 and 15 depict the timing and latching associated with the breaking of boards in connection with practice of the invention;

Figure 16 depicts the timing of board breaking in response to loopback errors according to a practice of the invention;

Figure 17 depicts timing of board breaking in response to heuristic or arbitrary breaking in accord with a practice of the invention;

Figure 18 depicts partitioning of the Xbus in accord with a practice of the invention;

Figure 19 depicts circuitry for routing board_not_broken in connection with practice of the invention;

Figure 20 depicts the routing of information lines in accord with practice of the invention;

Figure 21 depicts routing of 3-way voted lines in connection with practice of the invention; and

Figure 22 depicts error checking logic in accord with practice of the invention.

Detailed Description of the Illustrated Embodiment

Figure 1 depicts a digital data processing system 10 according to one practice of the invention. The system 10 includes multiple functional units, namely, central processing unit (CPU) 12, CPU 14, input/output (I/O) unit 16 and I/O unit 18. Each CPU 12, 14 is coupled to each I/O unit 16, 18 via buses 20A - 20D, as illustrated. These buses can include data lines, as well as control lines. CPU 12 is also coupled to CPU 14 via bus 20E, while I/O unit 16 is coupled to I/O unit 18 via bus 20F. These buses include control lines, although they can also include data lines. Each functional unit includes a processing section, 12A, 14A, 16A, 18A, respectively, and two interface sections 12B, 12C, 14B, 14C, 16B, 16C, 18B, 18C, respectively, as shown in the drawing. In a preferred embodiment, each functional unit is contained on a printed circuit board that couples to a bus backplane in the system cabinet (not shown) in the conventional manner.

The processing sections 12A, 14A carry out conventional functions associated with central processing units, e.g., instruction execution. To this end, the processing sections 12A, 14A can include conventional microprocessors or other central execution units (not shown), as well as other conventional elements (e.g., on-board memory) typically employed for those functions. Interface sections 12B, 12C (constituting the drive side and check side interfaces for unit 12) transfer information between associated processing section 12A and buses 20A, 20B, 20E which couple CPU 12 with I/O units 16, 18 and with CPU 14. Likewise, interface sections 14B, 14C (constituting the drive side and check side interfaces for unit 14) transfer information between processing section 14A and buses 20C, 20D, 20E which couple CPU 14 to those same I/O units 16, 18, and to CPU 12.

I/O units 16, 18 serve as bridges to peripheral device buses 21A - 21D and, in turn, to peripheral devices 22A - 22D, as illustrated. In the illustrated embodiment, peripheral device buses 21A - 21D operate in accord with the industry standard PCI Bus Specification. Peripheral devices 22A - 22P are also PCI compatible. I/O units 16, 18 can include processing sections 16A, 18A, respectively, for carrying out administrative tasks, if any, required by the I/O units in their roles as PCI bridges. Interface sections 16B, 16C (constituting the drive and check sides of unit 16) and, 18B, 18C (constituting the drive and check sides of unit 18) provided in each of the I/O units 16, 18, respectively, transfer information between buses 20A - 20D and buses 21A - 21D. This includes converting transferred information to and from the PCI protocol used by buses 21A - 21D. Although each pair of interface sections 16B, 16C and 18B, 18C interface with a respective pair of internal buses 20A, 20D and 20B, 20C, respectively, the sections 16B, 16C, 18B, 18C interface dedicated PCI buses 21A, 21B, 21C, 21D, respectively, as illustrated.

Though the illustrated I/O units 16, 18 serve as PCI bridges, those skilled in the art will appreciate that the invention has application to I/O units capable of interfacing and/or controlling peripherals that communicate via any protocol. More generally, it will be appreciated that, although the illustrated functional units are central processing units and I/O units, the invention can be applied to any class of functional units, e.g., central processing units, memory units, I/O units, communications units, etc.

In operation, the CPU's 12, 14 drive information to I/O units 16, 18 over their respective buses 28A - 20D, and vice versa. Each of the units 12 - 18 also monitor the buses to which they are attached and signal the other units in the event that there are bus communication errors. Thus, for example, if CPU 12 detects a parity error or an error correcting code (ECC) error during a transmission of data to I/O units 16, 18 over data lines in buses 20A, 20B, respectively, CPU 12 can signal that bus error to the I/O units over control lines in buses 20A, 20B. CPU 12 can also signal a bus error to CPU 14 over bus 20E. The other functional units 14 - 18 can likewise signal bus errors in the event they detect erroneous transmissions over their respective buses.

In response to a bus error signal by any of the functional units 12 - 18, each functional unit enters into an error isolation phase. During this phase, each of the units 12 - 18 transmits patterns of test data onto its respective buses 20A - 20F while, at the same time, monitoring those buses for error. For example, in response to a bus error signaled by I/O unit 18, CPU 12 can drive test patterns onto buses 20A, 20B, while CPU 14 simultaneously transmits test patterns on buses 20C, 20D. During these transmissions, each of the functional units

12 - 18 monitor the buses 20A - 20D for parity and ECC errors, while CPU's 12, 14 perform loopback checking on their respective buses 20A, 20B and 20C, 20D, respectively. Once the CPU's 12, 14 have completed their test cycles, I/O units 16, 18 take their turns at driving test data onto the buses.

The functional units continue to monitor their respective buses 20A - 20D and to signal bus errors while the test data is being transmitted. Each unit 12 - 18 does this not only when another unit is transmitting data on a common bus, but also when the unit itself is driving such data. This permits the units to perform loopback comparisons to match the data that it intended to drive onto a bus with that actually received from the bus through monitoring. For example, while CPU 12 is driving test data onto buses 20A, 20B, it simultaneously compares data values received from those buses to ensure that they are identical with the driven data. As during normal operation, the functional units 12 - 18 signal bus errors to one another in the event that they detect communications faults during the error isolation phase.

In addition to signaling bus errors, each of the functional bus errors 12 - 18 can generate a default (or "broken") if it detects that its own operation is faulty. Thus, whenever a functional unit 12 - 18 detects a loopback error, it will signal the others that it is broken. Furthermore, if any of the units 12 - 18 detect a bus error when it drives test data onto the bus, but not when another unit drives such data, the unit can signal the others that it is broken -- so long as none of the other units has, first, signaled broken, e.g., as a result of its own loopback error. By way of example, CPU 12 will signal the other functional units 14 - 18 that it is broken if CPU 12 detects a bus error during a cycle when it is driving test data onto buses 20A, 20B, but not when I/O units 16 - 18 are driving test data onto those buses. On detecting that it is faulty, any of the functional units 12 - 18 can take itself of-line.

A more complete understanding of the construction and operation of the illustrated embodiment may be attained by reference to the section that follows, in which digital data processing system 10 is referred to as "Polo," buses 20A - 20F are referred to as the "Xbus," buses 21A - 21D are referred to as the "Ibuses," CPU 12 is referred to as "CPU1," CPU 14 is referred to as "CPU0," I/O unit 16 is referred to as "PCI Bridge 1" or "PCIB 1," I/O unit 18 is referred to as "PCI Bridge 0" or "PCIB 0."

Bus Naming Convention

The Xbus actually consists of 4 data buses and 12 control buses, which are named as described below.

Data Bus Naming Convention

Figure 2 shows a block diagram of the 4 data buses in the Xbus system. The number for the bus is taken from the CPU slot number shown in the illustration; therefore data buses connected to CPU 0 end in 0 and data buses connected to CPU 1 end in 1. The letter for the bus is determined by whether or not the bus is a crisscross bus (i.e. connects an even slot number to an odd slot number) or a straight bus (i.e. connects an even to an even or an odd to an odd slot). Based on this convention, CPU 0 has connections to data bus A0 and B0. CPU 1 has connections to data bus B1 and A1. PCI Bridge 0 has connections to data bus A0 and B1. PCI Bridge 1 has connections to B0 and A1. In the illustrated embodiment, the PCI bridge cards do not run in lock-step.

Control Bus Naming Convention

Figure 3 shows a block diagram of the control buses in the Xbus system. The control naming convention for the backplane signals uses the signal name followed by the source of the signal and then the destination of the signal. The naming convention for the associated ASIC pins uses the signal name, the direction (in or out), and the connection (n for neighbor, o for opposite, and p for peer). Examples of this naming convention are shown in Figure 3. Table 1 lists the names of all control buses in the Xbus.

Table 1. Control Bus Names

Control Bus Source	Control Bus Destination	Control Bus Name	ASIC Driving Pin Name	ASIC Receiving Pin Name
CPU 0	CPU 1	control_0_1	control_out_p	control_in_p
CPU 0	PCIB 0	control_0_2	control_out_n	control_in_n
CPU 0	PCIB 1	control_0_3	control_out_o	control_in_o
CPU 1	CPU 0	control_1_0	control_out_p	control_in_p
CPU 1	PCIB 0	control_1_2	control_out_o	control_in_o
CPU 1	PCIB 1	control_1_3	control_out_n	control_in_n
PCIB 0	CPU 0	control_2_0	control_out_n	control_in_n
PCIB 0	CPU 1	control_2_1	control_out_o	control_in_o
PCIB 0	PCIB 1	control_2_3	control_out_p	control_in_p
PCIB 1	CPU 0	control_3_0	control_out_o	control_in_o
PCIB 1	CPU 1	control_3_1	control_out_n	control_in_n
PCIB 1	PCIB 0	control_3_2	control_out_p	control_in_p

Bit Numbering

Figure 4 shows a description of the bit numbering scheme used on the Xbus.

Terminology

bus cycle--the 24MHz (~41.67 ns) building block from which all Xbus operations are built. A bus cycle is the time which a valid logic level driven by one board on the backplane is seen by all other boards. Two bus cycles compose a bus phase, 4 bus phases compose a bus operation, and one or more bus operations compose a bus transaction.

bus phase--the 12MHz (~83.3ns, 2 bus cycle) building block from which all bus operations are constructed. There are logically 11 types of bus phases on the Xbus; "Arb", "Info", "Post1", and "Post2" are the phases that occur during normal operation. When an error is detected, the special phases "Error1", "CPU test", "CPU Post", "IO Test", "IO Post1", "IO Post2", and "Error2" are inserted in the protocol for fault isolation. (The error phases are sometimes collectively referred to as "Post3"). During each bus phase it is possible to transmit two sets of information on a physical set of backplane lines (i.e. "double pumping") though this is not done for all bus phases and/or signals.

bus operation--A bus operation is the basic unit of address and data transmission and checking on the Xbus. It is generally composed of at least 4 phases: an Arb phase followed by Info, Post1 and Post2 phases. Bus errors cause the insertion of the error phases after Post2 and increase the number of phases required to complete a bus operation. Bus operations may consist of multiple info transmissions in the case of a block transfer. A bus operation can be thought of as a full one way transfer on the Xbus.

bus sub-operation--A bus sub-operation is an operation initiated by a bus master during subsequent phases of a block transfer. Sub-operations always carry data and BUSYs are ignored. It is generally composed of 4 phases: an Arb phase during which grant inhibit is used, followed by Info, Post1 and Post2 phases. Bus errors may increase the number of phases required to complete a bus sub-operation. A sub-operation is differentiated from an operation in that a bus operation for a block transfer consists of the first transfer plus a number of sub-operations consisting of multiple data transfers.

bus transaction--a complete high level exchange of information on the Xbus. Examples include reads and writes. A read transaction between CPU and PCIB is composed of a minimum of two bus operations; one operation provides the address and function code, and one or more operations provide the return data. A write transaction to a PCIB is composed of a minimum of two operations; one operation provides address and function code, and one or more additional operations provide the data.

Figures 5 and 6 illustrate the terminology surrounding simple word read and line write transactions.

Bus Master--A board that has won arbitration. This board drives the info lines in the info phase of the bus operation. A bus master can be a transaction master or a transaction slave.

Bus Slave--A board that has determined the info lines carry information that it must receive. A bus slave can be a transaction master or a transaction slave.

CD different read--A read in which the C and D side ASICs each provide half the data e.g. when reading error status registers. Loopback checking of the bytes driven by the other side is suppressed.

Cyclops--The Xbus to Ibus interface ASIC on the CPU board in the Polo system.

echo transaction--The second half of a peer-to-peer bus transaction between CPUs. Send and echo transactions are not split; grant inhibit is used to ensure that no other transactions occur between the send and echo. This is to prevent re-ordering.

EFQ--Eviction-Flush Queue. This queue exists only on CPU boards. Refer to the Cyclops (Bus Interface) Specification for details.

EFQ-Freeze State--Set via bit 19 of the Bus Interface State Register. This mode only exists on CPU boards. When in this mode, a board will busy all accesses directed to its EFQ except those from its partner unless it is a data return of any size.

Fast Duplexing--Fast duplexing is a form of duplexing in which no memory is updated. This is done when both boards are coming up for the first time as opposed to updating a new board from a running OS.

Freeze-State--Set via bit 14 of the Bus Interface State Register. This mode only exists on boards that can be duplexed. When in this mode, a board will busy all accesses directed to its RWQ except those from its partner.

Gambit--The Xbus to PCI ASIC on the PCI Bridge card. It interfaces to the Xbus and the PCI bus.

I/O virtual address (IOVA)--An IOVA is an address generated by a PCI card. This address is transmitted across the Xbus to the Cyclops ASIC. Inside the cyclops ASIC, the address is translated into a valid system address. The IOVA is used to provide fault tolerance. It guarantees that a PCI card will generate a correct address range.

loopback checking--This is when an ASIC checks that the value it sees on a pin is equal to the value it thinks should be on the pin during normal operation.

loopcheck operation--This is when the bus ASICs drive test patterns of 55/AAs as part of the error protocol in order to determine the site of a fault.

peer to peer transaction--A two part transaction between two CPUs. The Xbus does not have fully interconnected data buses, and transfers between the two CPU boards must occur in two steps: first a send between the CPU and the PCIBs), then an echo from the PCIB(s) to all of the CPU(s). The requesting CPU drives a complete transaction on its A and B buses. The PCIBs look at the address, and determines whether the transaction is directed to the CPUs. If it is, they buffer the transaction in order to repeat the transaction on their A & B buses once Post2 of the last info phase has passed with no bus errors. In this way, both CPUs see the transaction at the same time. Peer to peer transactions between CPU boards require a minimum of four operations.

RWQ--Read/Write Queue. This queue exists only on CPU boards. Refer to the Cyclops (Bus Interface) Specification for details.

Regurgitated Info--A regurgitated info is a Cougar **[Conrad, what is Cougar?]** generated cycle used during the update process. It is generated by the update-on-line board and transmitted to the update off-line board. It is unique because an update off-line board accepts info cycles even if the base address does not match the base address of the board.

send transaction--The first half of a peer-to-peer bus transaction between CPUs.

single side operation--An operation with data supplied entirely by either the C or D ASIC; e.g. a read from a PCI card. Loopback checking is performed only by the side supplying the data.

TRID--Transaction identifier. This is a unique binary number used to tie together two bus operations during a split transaction and to identify the source for write transactions (TRIDs on write transactions are strictly for debug). A TRID is unique only while a given transaction is still outstanding and will be re-used for later transactions by a transaction-master. Note that **trid** bits 02-00 are used for the slot number of the transaction-master and **trid** bits 06-04 are generated by an on-board master - thus allowing a board to have 8 unique masters with transactions outstanding. Trid bit 03 is a new bit for Polo that indicates the format of the address; a zero indicates a Jetta style system address is being transmitted and a one indicates an IOVA (I/O Virtual Address) format is on the backplane. The IOVA address needs to be translated into a system address via the map RAM.

Transaction Master--The specific resource on a board that generated a transaction. The transaction master is responsible for generating the TRID of the transaction.

Transaction Slave--The board on which a transaction was directed towards, i.e. a board in which the function code and address of a bus operation has decoded to.

Xbus Signal Description

The following section describes in detail the various signals that comprise the Xbus. A functional description is included for each type of signal, though not for every individual signal (for example: there is a TRID field for all 4 buses, however there is only one description that covers both). The following rules were used in creating the signal names:

- all lower-case characters
- the "_" is a delimiter when it is not at the end of a signal name
- the "_" at the end of the signal name indicates that the signal is low-true
- _a indicates A_bus signals
- _b indicates B_bus signals
- _x_, _y_, and _z_ are used for the three low-true signals on a triplicated net
- [n:m] is used to describe a multi-bit field

Signal Description

The bused signals are implemented as four point-to-point bidirectional signals. The ability to drive these signals is controlled through an arbitration network. These signals are protected by a single parity bit that accompanies the bus. The buses are interconnected so that each CPU is connected to both PCI bridge boards and each PCI bridge board is connected to both CPUs. Error recovery is accomplished through the XBus error protocol.

The control signals are organized in a set of point to point unidirectional buses. Each of these buses is ECC protected. These buses carry control signals which are not governed through arbitration. Unlike the bused signals, there is a control bus in each direction between every board. This is necessary in order to ensure the single bus view of the system. For example, if one PCI bridge card sees a bus error, that information must be transmitted to all three other boards in order for the boards to all perform the error sequence.

The bused signals and control signals are double pumped at 24MHz each cycle. That is, they carry different data during the first and second 24Mhz bus cycles that make up a single phase. All buses and control signals are active high on the backplane.

A small number of reset and broken related control signals are buffered by the 26S10 transceiver and replicated for three way voting. These signals are active low on the backplane.

The Info Bus

The following signals are collectively referred to as the info bus. Although there are actually 4 sets of these signals (a0,a1,b0,b1), for simplicity's sake only the a0 version is listed. For example, when the TRID field is described, it should be understood that there are actually 4 TRID buses: trid_a0, trid_b0, trid_a1, and trid_b1.

Table 2. Xbus Bidirectional Buses

signal	width	description
info_a0[31:0]	128 (32x4)	**Xbus info bus** - Info is driven during the info phase of a bus operation by the current bus master. This field may contain either an address (physical address or virtual index with function code) or data, depending on the fund_op control line.
trid_a0[6:0]	28 (7x4)	**Xbus transaction id (TRID)** - The trid lines carry the TRID (transaction ID) during the first cycle of a phase. During the second cycle, it carries the number of phases remaining, the first_op bit, and cache coherency bits.

signal	width	description
func_op_a0_	1	**Xbus func_op** - This line carries the func_op_ signal which indicates that the current information on the info bus contains a function code that should be decoded. This signal is low true so that an idle bus will indicate that a function needs to be decoded, and thus a no-op function. This bit is valid during an info phase and is protected by parity along with the lower half of the info field. During the second cycle, it is unused (driven to logic 0 on the backplane).
parity_a0	1	**Xbus parity** - This parity signal covers all of the bidirectional signals on a bus; info, trid, func_op_.
TOTAL	**158**	

The Control Bus

This section describes the control signals. There are actually 12 control buses, but again only one is described here. The names of the twelve control buses are listed in Table 1, above. For simplicity of documentation, the control bus is identified by bit numbering, similar to the trid field. However, since the meaning of the control bus bits is very significant, each one is described in detail here.

The control buses are protected by a single bit correction, double bit detection ECC code.

Table 3. Control Bus Signals

signal	width	description

control[0]	12 (12x1)	**bus_req and ack** - During the first half of the phase, this bit is used for bus_req. During the second half of the phase, this bit is used for ack. During the first half of the phase, this bit is driven by a board when it is requesting the bus. As described in later sections, the bus uses a distributed arbitration model loosely based on the Golfbus. Each board in the system drives the bus_req and tests all of other boards bus_req signals to determine who will drive the info signals during the next phase. During the second cycle of the phase, this bit is used to acknowledge a bus transaction. Ack is asserted in Post2 by the target board of the transaction. This signal is the result of an address decode, so it is only valid in Post2 of an operation that is transferring an address. Ack provides an indication of whether or not a transaction is progressing. This is relevant in a Polo system, since a PCI card may go away resulting in a no ack for a ping operation. Acks in Polo system also let a PCI Bridge know that a CPU's map RAM has mapped a PCI initiated access to a valid CPU address. If a PCIB's read or write is not Aed, the PCI slot that initiated the access may or may not be set off-line depending on bits in the Gambit's configuration register. Writes from the CPU to the PCIB are acked to facilitate debug, but are otherwise unused. A peer to peer CPU write is not ACKed by the PCIBs, but the echoed operation is

control[1]	12 (12x1)	**grant_inh and main_int** - During the first half of the phase, this bit is used for grant_inh, during the second half of the phase, this bit is used for main_int.
		During the first cycle, the grant inhibit control bit is driven by the current bus master to extend the info cycles when the bus master is moving a block of data. The arbitration logic will not issue a grant to any other board when a board is driving the signal. This ensures that the current bus master will retain ownership of the bus for another cycle.
		During the second half of the cycle, this bit is used to signal a maintenance interrupt. Maintenance Interrupt indicates that some board in the system is requesting attention from the software maintenance process. Any board in the system may drive this signal during the second half of a bus phase regardless of bus mastership. All boards in the system will sample maintenance interrupt and use it to reset their arbitration priority.

control[2]	12 (12x1)	**bus_err_a and busy** - Assertion of this signal during the first half of Post2 signals that a bus error was detected on the info bus associated with this particular control bus (n,o,p). Any operation in Arb, Info, or Post1 will be aborted. Operations in Post2 are suspended while the error protocol runs, and then will return to the Info phase. The CPU initiator of a peer-to-peer operation must track the entire operation to see whether the cycle is errored in either the send or echo portion of the transaction. Assertion of this signal during the second half of Post2 indicates to the bus master that the operation should be aborted and re-tried at a later time. Busy is ignored for the send portion of a peer-to-peer operation. The CPU initiator of a peer-to-peer operation must track the entire operation to see whether the cycle is busied.

control[3]	12 (12x1)	**bus_err_b and funny_state** - Assertion of this signal during the first half of Post2 signals that a bus error was detected on the B bus connected to this board. Any operation in Arb, Info, or Post1 will be aborted. Operations in Post2 are suspended while the error protocol runs, and then will return to the Info phase. The CPU initiator of a peer-to-peer operation must track the entire operation to see whether the cycle is errored in either the send or echo portion of the transaction. During the second half of the cycle, this bit is used to signal that a board has just gone unbroken. The board will continue to assert this signal until it has seen eight phases without a bus error occurring. At that point the board will stop asserting this signal. Then all other boards will treat this board as an active, responding board. This prevents a board that is going unbroken from responding to bus errors in the middle of an error sequence that is already underway.

control[7:4]	48 (12x4)	**checkbits** - The top 4 bits of the control bus are checkbits generated from the lower 4 bits of control signals.
		
		The checkbit algorithm is:
		
		control[4] = control[0]^control[1]^control[2];
		
		control[5] = control[0]^control[1]^control[3];
		
		control[6] = control[0]^control[2]^control[3];
		
		control[7] = control[1]^control[2]^control[3];

TOTAL	**108**	

The Voted Signals

The voted signals are the only signals through 26S10 transceivers. These signals are point to point and terminated at each end, so that insertion of an unpowered board does not disturb the termination (and timing) of a net in use. Only the _x_ versions are listed; there are _y_ and _z_ signals making up the triplet.

Table 4. 3-Way Voted Signals

signal	total	description
reset_0_1_x_, reset_0_2_x_, reset_0_3_x_, reset_1_0_x_, reset_1_2_x_, reset_1_3_x_,	18 (2x3x3)	**reset** - There are separate 3-way voted triplets from each CPU to the other three boards in the system. When a RECC needs to reset the system, all lines go active. When a CPU wants to reset another board, only the lines going to that board are active.
board_not_broken_0_1_x board_not_broken_0_2_x board_not_broken_0_3_x board_not_broken_1_0_x board_not_broken_1_2_x board_not_broken_1_3_x board_not_broken_2_0_x board_not_broken_2_1_x board_not_broken_2_3_x board_not_broken_3_0_x board_not_broken_3_1_x board_not_broken_3_2_x	36 (4x3x3)	**broken status** - This three way voted signal is driven from each board to each other board. It is driven when a board is alive and not broken in the system and is used to determine which buses are active. The C-side ASICs drive the signals and the D-side ASICs drive the output enables for the 26S10s. This organization guarantees that board_not_broken is deasserted whenever either side of the board thinks that the board is broken. The receiving board votes the x, y, and z signals. CPU0 in slot0 drives board_not_broken[0], etc.
sync_x_	3 (1x3)	**sync status** - These signals are on the CPU only and are used when synchronizing a pair of CPUs to enter the duplexed state.
even_online_x_, odd_online_x_	6 (2x3)	**on-line status** - These signals are on the CPU only and are used by the CPU boards to communicate to each other which CPU board(s) are in the on-line state. Even_online_ is asserted when the CPU in slot 0 is on-line; odd_online_ is asserted when CPU in slot 1 is on-line.

signal	total	description
TOTAL	**63**	

Other Control Signals

Table 5. Miscellaneous Control Signals

lot_ida slot_idb	2	**slot id** - The slot ID signals are hard wired on the backplane for each slot. There is one duplicated slot id. Since the slots are dedicated in Polo, it is only necessary to determine if a board is in an even or an odd slot. These two bits will be registered and checked by each ASIC at reset and will not be sampled again. If an error is detected at reset, the board will break and hence will never be capable of being brought on-line.
xb_clk8	1	**system clock** - This is the system clock received by the Sentry clock chip and used to generate the board clocks. It is generated by the backplane clock oscillator. This clock runs at 8MHz, so every board in the system will be in sync with each other and there will be no need for additional synchronization clocks to be passed along the backplane. The clock is pulse width modulated so that 4Mhz can be generated.
slot0_ta_d, slot0_ta_c_, slot1_ta_d, slot1_ta_c_	4	**ta signals** - These "ta" signals are only present on CPU boards and are sent between a duplexed board pair for early detection of the boards going out of lockstep.
TOTAL	7	

Xbus Protocol

Overview

The Xbus is a point-to-point synchronous, pipelined, multiplexed address/data, error detecting bus with split-transaction capabilities. Function-code, address and data are parity and loop-back checked. Control lines are ECC protected and loop-back checked. Three-way voting is implemented on the reset, clock, and broken indicator lines.

The bus supports word accesses and has a block transfer capability for support of cache line accesses. The Xbus has a logical 32-bit wide data/address bus.

Bus Operation

The basic component of all Xbus transactions is the operation. An operation is composed of four phases as illustrated in Figure 7: arb, info, post1, and post2. Two information transfers can occur on the bus during each phase; this is referred to as "double pumping". The double pump frequency is approximately 24MHz. The figure illustrates the logical activity on the bus. All information is directly registered in the ASICs without any external buffers.

The phases are used as follows:

Aarb phase: Boards drive their arbitration request lines during the first half (cycle) of the arbitration phase. During the second half they determine whether they won arbitration and prepare for the info phase.

Info phase: For non-IOVA address transfers, boards drive the virtual index, function code, remote/coherent bits, and byte enables during the first half of the info phase and the physical address during the second half. For IOVA address transfers (IOVA bit in the trid is true), boards drive the IOVA during the first half of the info phase and deterministic data with good parity during the second half; the physical address is gotten from the I/O address map RAM look-up. For data transfers, data is driven during both the first and second halves of the cycle. Note that non-cache consistent address transfers need not supply a virtual index through the driven information must be deterministic and parity will be computed across it.

Post1 phase: During this phase, boards are determining whether any error conditions existed with the info phase and whether there is need to BUSY the operation. CPU boards map the device index portion of the IOVA to obtain the full physical address and virtual index of an I/O board's transfer for IOVA address transfers.

Post2 phase: Any board detecting an error with the info phase drives the error lines during the first half. If a board does get errored, it next goes to the error sequence phases to determine the source of the error. Any board detecting the need to BUSY an address/function code driven during the info phase drives BUSY during the second cycle of this phase. It is also during this phase that accesses are acknowledged.

Bus Busies

Figure 8 illustrates the effect of bus BUSYs on the basic bus cycle. As shown in the figure, BUSY has no effect on a bus operation during any phase except for post2; a BUSY during post2 will cancel the bus operation. Note that busys for multiple cycle bus operations, such as block transfers, have the special rules. Should a cycle be both BUSYed and ERRORed, the ERROR takes precedence.

Bus Errors

Figure 9 shows the effect of bus errors on the basic bus cycle. As shown in that figure, the board that was transmitting during the error automatically gets the first available info cycle following execution of the error protocol. The arbitration is ignored in the previous cycle.

Since the Xbus has no transceivers, the loopcheck phase of the Golfbus error protocol (Post4) has been modified to allow each board an opportunity to verify its transmit and receive capabilities. This has resulted in new states being added to the bus error operation. These states are described below:

Errl: The Errl state is entered on the cycle after a bus error is detected. This state is used to allow for time to turn off the info bus before the loopback checks are performed. A board that is in its info phase during Errl will disable its output enables half way through the phase.

CPUTest: The CPUTest state is used to test the CPU's ability to drive patterns on the Xbus. On the first cycle of the phase the CPU will drive 55 on the info bus, 55 on the trid bus, 1 on the parity line and 0 on the func_op line. On the second cycle of the phase the CPU will drive AA on the info bus, 2A on the trid bus, 0 on the parity line and 1 on the func_op line.

CPUPost: The CPUPost state is used to turn the bus around between the CPU's loopback check and the I/O boards loopback check. This phase is also used as a Post1 cycle for the CPU's loopback pattern.

IOTest: The IOTest state is used to test the I/O board's ability to drive patterns on the Xbus. On the first cycle of the phase the I/O board will drive 55 on the info bus, 55 on the trid bus, 1 one the parity line and 0 on the func_op line. On the second cycle of the phase the I/O board will drive AA on the info bus, 2A on the trid bus, 0 on the parity line and 1 on the func_op line. Bus errors from the CPUTest phase are reported during this phase. This information is used to evaluate the bus, CPU, and I/O board at the end of the error sequence. The last I/O ASIC to drive the data bus drives the bus during the IOTest phase.

IOPost: The IOPost1 state is used to evaluate the IOTest data.

IOPost: The IOPost2 state is used to transmit any bus errors from the IOPost1 state. This information will be used to make an intelligent decision about how to deal with the error.

Err2: The Err2 state is used to evaluate the information from the loopback checks. Bus errors from CPUPost and IOPost2 as well as information shared between the C and D sides of each board are used to determine what course of action to take. This set of actions will be described later in this section.

Figure 10 shows the basic state machine and state transitions for the bus error handler. The key challenge for the bus error algorithm on the Xbus is to diagnose errors so that system operation can continue. Unlike previous systems that use duplicated buses to allow all functional units a guaranteed path for communications, when the Xbus removes a bus from service, it must also remove one of the two units attached to that bus. In some cases, the right thing to do is obvious. In other cases, it is not. The following sections analyze various faults, how they are handled and how they manifest themselves.

Bus Error Broken Conditions

At this point it would be helpful to classify the different types of conditions that cause a board to go broken when a bus is detected bad.

- **loopback on control** - C and D ASICs must always agree on what to drive on the control lines, including whether or not to assert bus error. If one ASIC asserts bus error and the other side does not, the board breaks.
- **loopback on data** - C and D ASICs must always agree on what to drive on the duplicated info lines. When driving CD same data, ASICs compare the data they drive with the data they receive. An ASIC asserts bus error on parity errors when receiving data, and parity errors and loopback errors when driving data. Loopbacks checking is disabled when a board drives "CD different" data, such as the contents of error reporting registers or data from PCI cards. The board breaks if the two sides disagree on which bytes have or do not have errors.

- **arbitrary** - Break the designated board in Err2 when there are bus errors signaled during CPUtest and IOtest and no board has broken by the end of Iopost2. This is called an arbitrary shoot because the fault is most likely on the backplane, so it is arbitrary as to which of the two boards connected to the faulty bus to break. Typically, the CPU is set broken, so that the system can continue with all of its I/O available, but if bit 21 of the Bus Interface State register is set then the PCIB board will be the designated board.
- **heuristic** - a board breaks itself during Err2 if there is a bus error when it drives, but no bus error when the other board drives, and the other board did not break by the end of IOPOST2.

Xbus Fault Analysis

In order to understand various faults and what they can mean, it is important to present a detailed block diagram of the Xbus interconnect. Figure 11 shows the interconnect for a typical Xbus line.

The black dots in figure 11 represent the connectors to the backplane. For fault tolerance and fault isolation reasons, it is important that the boards should be routed so that the etch between the D-side and the C-side runs through the connector connection. This limits the amount of etch on each board that cannot be isolated to a minimum. On the CPU board, one ASIC both drives and receives a given net while the other ASIC only receives that net. On the I/O board, each ASIC can potentially drive every net. The CPU ASICs are always in lockstep and therefore each ASIC is capable of sharing the data out load. However on the I/O board, each ASIC connects to a different PCI bus so a signal ASIC may need to drive the entire Xbus. There are cases in normal operation when only one CPU ASIC will drive the entire bus.

Fault Conditions

The following sections identify all known fault conditions and describe their handling. Refer to Figure 11 to determine the location of the fault site indicated. That figure depicts two functional units (or board), e.g., units 12 and 16, as well as their respective drive and check sides 12B , 12C, 16B, 16C.

CPU Board Faulty Input Circuit - CPU Driving

- fault site 31
- break via loopback on control fault

This fault deals with a fault in the input section of one of the CPU ASICs. In this case, the fault occurred during or just before a cycle in which the CPU drove the info bus. The error is detected when the CPU drives the bus. The ASIC with the faulty circuit will signal a bus error during the Post2 phase of the cycle and the other side ASIC will not. The board will go broken and drive bus error during Err1. The error sequence will be executed, and the operation will be retied by the partner CPU with no error.

CPU Board Faulty Input Circuit - I/O Board Driving

- fault site 31
- break via loopback on control fault

This fault deals with a fault in the input section of one of the CPU ASICs. In this case the fault occurred during or just before a cycle in which the I/O board drove the info bus. If the error is a multi-bit error that evades the parity logic, the error will be caught internal to the CPU board and the CPU board will go broken. If the error is a single bit error the faulty ASIC will detect a bus error during the Post1 phase of the transfer. The ASIC will drive bus error during Post2 of the transfer and the other side of the board will not. The board will break with a loopback on control failure in the next phase. After the error sequence, the operation will be retried by the partner CPU with no error.

CPU Board Different Data C-Side and D-Side

- fault site 32
- break via loopback on data fault

This fault deals with an internal CPU fault that results in different data being driven out of each ASIC. The error is detected when the CPU drives the bus. The C and D ASICs will trade error status and disagree on where the error is during Post1; both C and D sides will see an error on bytes the other side drives but no error on the bytes it drives. The board will go broken and drive bus error during Post2. The error sequence will be executed and the operation will be retried by the partner CPU with no error.

CPU Board Faulty Output Circuit - Buffer to Pad Fault

- fault site 33
- break via heuristic broken

This is a fault in the output section of the CPU ASIC resulting from an output driver circuit fault that blows in a manner that causes an internal open between the output driver and the ASIC pad, while not disrupting the functionality of the input receiver. All ASICs on the bus will detect a bus error during the Post1 phase of the transfer. The ASICs will drive bus error during Post2 of the transfer. All ASICs will detect a bus error during the CPUPost phase. No bus errors are detected during the Iopost1 phase. The CPU board will go broken during Err2 based on the fact that it detected a bus error when it drove the bus, no bus error when the I/O board drove the bus and the I/O board did not go broken after its error sequence.

CPU Board Open - CPU Board Driving

- fault site 34
- break via loopback on data fault

This fault results from an open due to either a broken etch or a lifted pin on he CPU board. The routing is very important for this class of fault. The etch between the C-side and the D-side should be routed through the connector pin. This limits the possibility that an open on the CPU board is mistaken to be an open on he backplane. In this case the fault occurred during or just before a cycle in which the CPU drove the info bus. The error is detected when the CPU drives the bus. During the Post1 phase, the driving ASIC will not see an error but the checking ASIC will signal a compare error. During Post2 the board will go broken and drive bus error. The operation will be retried by the partner CPU with no error after the error sequence.

CPU Board Open - I/O Board Driving

- fault site 34
- break via loopback on control

This fault results from an open due to either a broken etch or a lifted pin on the CPU board. The routing is very important for this class of fault. The etch between the C-side and the D-side should be routed through the connector pin. This limits the possibility that an open on the CPU board is mistaken to be an open on the backplane. In this case the fault occurred during or just before a cycle in which the I/O board drove the info bus. The ASIC with the open between it and the connector will detect a bus error during Post1. During Post2 one ASIC will assert bus error and the other will not, causing the board to break. The operation will be retried by the partner CPU without any error after the error sequence.

CPU Board Short

- fault site 34
- break via arbitrary broken

This fault deals with a short on the CPU board. When the fault occurred and who was driving the info bus during the fault are not relevant to this class of fault. During Post1 of the transfer all ASICs on the bus will detect a bus error. The ASICs will drive bus error during Post2 of the transfer. Both ASICs on the CPU will detect a loopback error during the CPUPost phase and the ASICs on the I/O board will signal a bus error during IOTest. The ASICs will detect a bus error during the Iopost1 phase and signal a bus error during IOPost2. During Err2 the designated board will go broken based on the fact that it has detected bus errors during the error sequence and no other boards went broken after the error sequence.

Backplane Open Etch

- fault site 35
- break via arbitrary broken

When the fault occurred and who was driving the info bus during the fault are not relevant issues for this class of fault. During Post1 of the transfer some ASICs on the bus will detect a bus error and drive bus error during Post2. Both ASICs on the I/O board will detect a bus error during CPUPost. The CPU ASICs will detect a bus error during IOpost1. During Err2, the designated board will go broken based on the fact that it has detected bus errors during the error sequence and no other board went broken during that error sequence.

Backplane Short

- fault site 35
- break via arbitrary broken

When the fault occurred and who was driving the info bus during the fault are not relevant issues for this class of fault. All ASICs on the bus will detect a bus error during Post1 and drive bus error during Post2. All ASICs on the bus will detect a bus error during CPUPost and IOPost1. During Err2, the designated board will go broken based on the fact that it has detected bus errors during the error sequence and no other board went broken during the error sequence.

I/O Board Faulty Input Circuit

- fault site 36
- break via loopback on control

This fault deals with a fault in the input section of one of the I/O Board ASICs. For this particular fault, it is irrelevant who was driving the backplane when the fault was detected. The faulty ASIC will detect a bus error during Post1. During Post2 of the transfer, the faulty ASIC will drive bus error and the other ASIC will not, causing the board to go broken. If it was a CPU initiated request the operation will be retried by the CPU with no error after the error sequence. If it was a request initiated by the I/O board, then the request will be dropped.

I/O Board Output Circuit Fault - Buffer to Pad

- fault site 37
- break via heuristic broken

This is a fault in the output section of the I/O board ASIC. This class of fault results from an output driver circuit fault that blows in a manner that causes an internal open between the output driver and the ASIC pad, while not disrupting the functionality of the input receiver. All ASICs on the bus will detect a bus error during Post1 of the transfer and drive bus error during Post2. No error is detected during the CPUPost phase. The I/O ASICs will detect a bus error during IOpost1. During Err2 the I/O board will go broken based on the fact that it has detected a bus error when it drives the bus, no bus error when the CPU board drove the bus and the CPU board did not go broken after its error sequence.

I/O Single-side Access - ASIC Parity Gen. Fault

- fault site 38
- break via internal parity generator broken

A fault in the parity generation logic during single side accesses (data driven entirely by either the C or D ASIC) could cause the system to bus error forever. The otherside ASIC doesn't know the data, and has no way of checking the parity. For this reason, the info bus parity generation is duplicated and selfchecking inside of the Gambit ASICs.

I/O Single-side Access - ASIC PCI Data Path Fault

- fault site 38
- break each PCI slot due to checksum and may RAM errors

This is a fault within the ASIC's PCI data path. This hardware is not running in lockstep with the other side of the board and therefore is not self-checking. Eventually bad addresses produced by the PCIB will cause map RAM errors and/or data checksum errors in the CPU ASIC. The CPU ASIC noACKs accesses that cause map RAM errors, and the PCIB sets off-line the PCI slot that originated the noACKed cycle based on an option bit in the Configuration register in page zero of SAM compatible I/O space. Eventually, all PCI slots handled by the detective ASIC will be set broke. In the meantime corrupted I/O data is detected by checksums and handled by higher level protocols.

I/O Non-single-side Access, Different C-D Data

- fault site 38
- break via loopback on data

This fault deals with an internal I/O board fault that results in different data being driven out of each Gambit ASIC during a regular (not single-side) read return. Each ASIC will disagree with the data driven by the other side and the board will break with a loopback on data error. The error sequence will be executed and the operation will be retried by the CPU and get noACKed.

I/O Board Open

- fault site 39
- break via loopback on control

This fault deals with an open, either from a broken etch or a lifted pin on the I/O board. The routing is very important for this class of fault. The etch between the C-side and the D-side should be routed through the connector pin. This limits the possibility that an open on the I/O board is mistaken to be an open on the backplane. The driving ASIC sees no errors and the outside ASIC asserts bus error; the board breaks the following phase with a loopback on control fault. After the error sequence the operation will be retried with no error and get noACKed.

I/O Board Short

- fault site 39
- break via arbitrary broken

This fault deals with a short on the I/O board. When the fault occurred and who was driving the info bus during the fault are not relevant issues for this class of fault. During Post1 of the transfer all ASICs on the bus will detect a bus error and then drive bus error during Post2. All ASICs on the bus signal a bus error during CPUpost and IOpost1. During Err2, the designated board will go broken based on the fact that it has detected bus errors during the error sequence and the other board did not go broken after its error sequence.

Transient Fault

- fault site anywhere
- action depending on fault site and timing

This fault deals with a transient fault. A transient fault is defined as a fault that is detected, but cannot be reproduced by the error sequence. If the fault is on the driving board and it is caught by loopback check, that board will go broken. If this is not the case then no board will break until Err2, then the designated board will go broken.

Slow Driver Fault

- fault site 34, 39
- possibly break the wrong board with a loopback on control fault

When an ASIC with a marginally slow driver drives the bus, four ASICs clock in data from the net while the net is changing. Due to differences in speed between ASICs from different lots, its possible that some of the ASICs will detect a bus error, and some won't, resulting in a loopback on control fault. This may result in one or both of the boards on the bus going broken.

Board Not Broken Generation

The board_not_broken_out signal is generated by all boards. If a board is going to break the normal operation is for the board to assert bus error. All boards will enter the error sequence and the board that was going to break will de-assert board_not_broken_out during the error sequence. It is possible that a board could break without asserting bus error. This could only happen if there is a problem with asserting bus error on the d-side gate array (for instance the clock line going to the flip-flop that produces the control signal (bus error) opens).

Arbitrary Breaking

There are a series for cases mentioned before that error logic cannot determine where the fault is. For these cases the goal is to not use the bus that the two boards are connected to. The default is to break the CPU board (designated board). This makes sense because there is a partner running in lock-step so no connectivity is lost. It is possible that the CPU board will break but the fault really lies with the PCIB. The only way this will pose a problem is if the failure in the PCIB effects the other bus that is still connected and is being used. Normally it is the CPU board that is the designated board.

Xbus Fault Tolerance

The goal for fault tolerance on the Xbus is for no single point of failure to cause a system crash or allow a transaction to complete without transmitting the correct data. A single point of failure could be any component, any signal, or any pin on a component.

The Xbus fault tolerance scheme assumes that one of the components connected to the bus is always responsible for hard bus errors. To this end, the design has been simplified around the assumption that when an error occurs, an offending board, or Field Replaceable Unit (FRU), will be removed from service. This extends to include all buses that the FRU is connected. Thus when an Xbus fault occurs the faulted bus service is removed taking at least one FRU with it.

Both sides of the CPU board drive signals to the bus and both sides of the board perform various checks to ensure that the board is functioning normally. This is identical to the Golfbus methodology. The PCIB board behaves differently; both sides of the board drive and check the bus when driving data from Xbus related IO registers, and when driving data originating on a simplexed PCI card, only one ASIC drives and checks the entire width of the bus.

Info Bus Protection

The info bus is protected by parity and loopback checking. However there is no A to B bus cross checking by anyone on the bus other than the bus master. When a PCIB is bus master only one of the two buses attached to a given CPU module will be active because the PCIBs do not duplex and therefore the bus from the PCIB that did not win arbitration must be idle. Likewise the PCIBs cannot know if two duplexed CPU modules are driving the bus or a single simplexed CPU has ownership (two simplexed CPUs could have started arbitration at the same instant). Thus receivers on the Xbus will only listen to the bus that is driven by the module that won arbitration.

Parity

Parity is generated and transmitted across each of the physical buses. The 32 bit bus, trid field, and funcop bits are all covered by a single parity bit. The purpose of the parity check is to protect against etches that open between the transmitter and receiver. Boards in the system check the transmitted parity that they receive with parity that they generate themselves.

A bus error is signaled on any parity error. The bus_err signals are also effected by other error checking that is described later in this document. In an effort to simplify these descriptions the reader may assume that the actual error signals are a logical OR'ing of the various outputs from different checking blocks.

Both the C and the D-sides of the boards will perform the check and compare results individually and decide on any action that must be taken. In the event that the two sides of a board do not agree upon the statute of a bus cycle, the board will break in the following cycle in one of two ways.

- The D-side of the board detected an error and the C-side did not. In this case the D-side will drive a bus error and the C-side will disagree with what the D-side has done and break the board. Here the transaction is errored and repeated, but the broken board should no longer be checking and the transaction will complete.
- The C-side of the board detected an error and the D-side did not. Here the C-side will once again break the board, but this time because it expected to see an error on the bus and didn't.

The transaction will complete normally and the bad board will be removed from service.

CPU boards compute parity in both C and D ASICs and break if a defect arises in one side's parity generators. Most of the transfers from a PCIB board, however, originate from a PCI bus connected to a single Gambit ASIC. When the Gambit drives this single sided data, it drives all the bits on the Xbus, and an error in the simplexed parity generator would hang the system. Fortunately, there is no need to duplicate the entire Gambit ASIC, only the parity generation section.

Referring to Figure 12, a fault at site 41 causes erroneous data to be transmitted with correct parity. This will be detected by higher level checksums. A fault at site 42 will cause the board to break with a data loopback fault. A fault at site 43 will cause the board to break with a parity generator fault. A fault at site 44 or 45 will break the board with either a data loopback fault or a parity generator fault.

Loopback Checking

The loopback correctivity is shown in Figure 13. On normal accesses (non-CD different, non-single sided), each side of the board drives half of each field to the bus and receives all of both busses. Thus there are two checks that each ASIC can do on the data returning from the bus.

1: Does the returning data match the data that was driven from this ASIC. This will be known as the loopback drive check.

2: Does the returning data match the data that was driven by the other ASIC. This will be known as the loopback compare check.

Two error signals are generated inside of each ASIC for each physical bus that it is driving. In order to know the true state of the board each side of the board must know the results of the comparisons on the other side as well as the comparisons it has done. Thus two signals are passed in each direction, as shown in figure 13, so that the each side may correctly determine the state of the board.

Each ASIC will determine whether the data on the bus is faulted or not, as well as if this board is broken or not. Table 4 below shows what combinations of the loopback signals for each bus indicate that the board is broken.

Table 4. Board Broken Matrix for a Single Bus

my drive_err	my comp_err	drive_err from the other side of the board	comp_err from the other side of the board	brd_broken
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1
0	1	0	0	1
0	1	0	1	1
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

A double bus error will be asserted and the not_broken signal de-asserted if a condition exists that should break the board. In addition a bus error will be asserted for any condition where a compare error of any type has occurred and the board has not broken. The bus error will only be on the bus while the error is detected.

Loopback on CD Different Accesses

When an ASIC drives data from a CD different register on the bus, it performs loopback checking only on the trid, func_op_, and parity lines. It asserts bus error if it sees an error, and will go broken according to table 4.

Loopback on Single-side Accesses

When a Gambit ASIC drives single-side data from the PCI it performs loopback checking on all of the bits; it does this only for the purpose of asserting bus error, not to see if it should go broken. It will not break according to table 4, because the cmp_err and drive_err signals from the otherside are invalid. However, if the driving ASIC sees a bus error, and the otherside ASIC does not (parity received good), the board will break due to a loopback on control fault.

Loop_ck_ops

Loop_ck_ops are used during the error sequence to make intelligent choices about the state of various boards and the buses that they are attached to. A loop check op is a cycle in which a board drives an alternating pattern of AA's and 55's on the backplane. The board evaluates the pattern and similar patterns driven by boards on the other end of the bus. Based on this information, bus state is altered.

Control Bus Protection

The Xbus collects control signals into a series of point-to-point unidirectional signals protected by an ECC code. Each board drives a separate control bus to each of the other boards.

There are two completely independent types of checks performed:

1: Every ASIC corrects for any single point failures (shorts, opens, dead drivers or receivers) with a single error correcting, double error detecting ECC code. Single bit failures cause a maintenance interrupt and are logged, but otherwise have no effect.

2: If a driving board detects a double bit error on the outgoing control signals it will attempt to assert bus error and then it will break in the error sequence (it will break regardless of its ability to drive bus error). If a receiving board detects a double bit error on any incoming control signals it will attempt to assert bus error then it will break in the error sequence. If the error was actually on one board and only one board saw the error then the correct board will break. If the error was such that both boards saw it then both boards will break. This is to limit the possibility that data gets corrupted.

Three-way Voted Signal Protection

Some broken and reset related control signals on the Xbus are triplicated and voted. The reset and board_not_broken_out signals are buffered through transceivers because they must be valid during power up or power down events. The CPU signal sync_out is buffered through transceivers to reduce ASIC pin count. The CPU signal my_online_out is sourced directly by the ASIC as a cost reduction.

The D-side ASIC drives signals to the transceivers and to the C-side ASIC. The C-side of the board checks what was driven by the D-side with what it thinks should have been driven. If the C-side disagrees with the D-side the board breaks.

Each ASIC that receives a three-way voted signal will perform a majority voting algorithm on the three signals to determine the value. In normal operation all three lines will be the same. If there is a fault on one of the three lines (either on the backplane or on the board), two of the three lines still will agree.

When a voting error occurs, the device detecting the fault should send a maintenance interrupt and log which transceiver has the non-unanimous vote. The logged information is stored in the Vote Error Transceiver Status register(s) and will not be over-written until the status register is explicitly re-enabled by software.

Error Reporting

A number of errors can be detected by a board's bus interface logic:

1: A loopback error that breaks the board. This specifically refers to loopback errors on the **info, parity, TRID,** and **func_op** signals which are seen by one side of the board and not the other during bus cycles in which the board drove the bus.

2: A disagreement between the two-sides of a board on whether a three-way voted signal should be driven.

3: A parity error on the bus, either on a cycle where the local board was driving the bus or another board was driving the bus.

4: A loopback error on the bus.

5: A three way voter error in which not all three signals are in agreement.

6: Any bus error signaled by any board in the system.

7: A thermal fault (high temperature detected on the board).

Some of the above errors may break a board (e.g. #31 and 32), some may cause a bus error to be generated (e.g. #33,34 and 36), and some have no effect on normal machine operation (e.g. #37). The Xbus interface contains two classes of status registers to record information surrounding the above errors: Broken Status and Error Status. Broken Status registers are only activated when a board goes broken, and are designed to pinpoint the logic that set broken. Error Status registers are for reporting non-fatal errors such as faults on the backplane. All of the Error Status registers are updated simultaneously on any bus error signaled by the system, or a voter or thermal error on the board. Specifically, when all error reporting is enabled, any of the above errors cause:

1: A maintenance interrupt to be generated.
2: The type of error condition to be stored.

In the above case, subsequent error state recording and the resulting maintenance interrupts are disabled until explicitly re-enabled.

It is not always optimal that all the above errors trigger the error registers and generate a maintenance interrupt. The reporting of new errors can be effectively blocked by an excessive number of reports on a known error. Hence, it is possible to turn off error latching due to different sources:

1: Perform error latching and maintenance interrupts on all errors.

2: Inhibit error latching and maintenance interrupts of incoming three-way voter errors.

3: Inhibit error latching and maintenance interrupts on thermal failure related errors.

The error latching disable bits inhibit those faults from triggering the error registers. However, if a disabled error is present and non-disabled error occurs, both errors will be recorded in the error registers.

Producing Errors

The Cyclops and Gambit gate arrays have specific registers which allow software to produce various errors on the Xbus.

The basic logic to produce an error consists of XOR gates on all the info, trid, func_op, parity, control lines going out and all the same lines coming in. One input on the XOR is the data line the other input is the error line and the output is the resultant data. If the error line is high the data coming out will be in error. If the error line is low then the data coming out will be in the form of a loopback error. If the data is errored on the input and the output then the error will be in the form of a bus error since it will not be a loopback error.

Table 5 shows all the error cases and the necessary settings from a hardware point of view to produce the error. 'Receive error' means the XOR gate on the receive side is inverting. 'Transmit error' means the XOR gate on the transmit side is inverting. The category describes if the error occurs only on a particular pattern match or if the error occurs just after the register is written. For many cases it would be possible to produce an error with different cases than those described below. The goal is to only produce the error in one manner.

Table 5. Error Settings

CPU Board Error

Type of Error	Category	D-Side	C-Side
CPU Board Faulty Input Circuit - CPU Driving	Immediate	receive error	no error
CPU Board Faulty Input Circuit - I/O Board Driving	Data Match	receive error	no error
CPU Board Different Data C-Side and D-Side	Immediate	transmit error /receive error	no error
CPU Board Faulty Output Circuit - Buffer to Pad Fault	Immediate	transmit error	no error
CPU Board Open - CPU Board Driving	Immediate	no error	receive error
CPU Board Open - I/O Board Driving	Data Match	receive error	no error
CPU Board Short	Immediate	transmit error	no error

Backplane Error

Type of Error	Category	D-Side	C-Side
Backplane Open Etch	Data Match	CPU and IO board transmit error /receive error	CPU and IO board receive error
Backplane Short	Immediate	CPU and IO board transmit error	CPU and IO board no error

IO Board Error

Type of Error	Category	D-Side	C-Side
I/O Board Faulty Input Circuit	Immediate	receive error	no error
I/O Board Output Circuit Fault - Buffer to Pad	Immediate	transmit error	no error
I/O Single-side Access - ASIC Parity Gen. Fault	Immediate	parity gen. fault	no error
I/O Single-side Access - ASIC PCI Data Path Fault	Immediate	bad address	no error
I/O Non-single-side Access, Different C-D Data	Immediate	transmit error /receive error	no error
I/O Board Open	Immediate	no error	receive error
I/O Board Short	Immediate	transmit error	no error

<u>Transient Error</u>

Transient Fault	This causes a bus error, but when the error sequence is executed there will be no error. This fault will not produce a maint. int. from this board, but all the other boards will assert maint. int.

<u>Bus Busy</u>

Bus Busy	This causes a bus busy.

Board Breaking Timing

Note in the figures Figures 14 - 17, Info_O's shows the time the info bus is driven to zero assuming that the bus is not already tristated from this side, xcever_disable shows the time the drivers from the broken board are disabled from the bus. Bus error is also shown when valid on the Xbus.

Board Breaking and Information Latching

There are several status bits in the Broken Status register on each board to determine the reason a board went broken. These bits are frozen after a broken condition occurs and will remain frozen until after a cold reset. All bits in the common register are straight forward with the exception of **board_specific_set_broken.** Board specific logic may have several reasons to assert this signal and may also have a similar register for the reasons it has asserted this signal. If **board_specific_broken** is set in the common register, the board specific broken register will contain the real reason the board asserted the **board_specific_set_broken** signal. If a board breaks for a reason other than board specific broken, the **board_specific_broken** bit will not be set in the broken status register. However, **board_specific_set_broken** may have been asserted after the original broken condition. This would have caused the board specific broken status register to latch and hold a reason it had set broken. It is important to understand that because the **board_specific_broken** bit is not set in the common status register, this was not the reason the board went broken. The reason the board went broken is logged in the broken status register and the board specific status register may be ignored.

In the timing diagram of Figures 14 and 15, **someone_set_broken** is the logical OR of the following signals:

cold_reset, warm_reset, slot_parity_error_set_broken, control_or_3way_vote_sig_error, asic_specific_set_broken, ecc_dbl_error_in. In addition Gambit contains the following signals: **break_pcib, xb_parity_gen_set_broken.** The board will break if any broken signal is asserted on any phase (i.e. any 48mhz edge). **my_set_broken_reg** is the equivalent of **otherside_set_broken_reg** on the other ASIC. **my_set_broken_reg** is ORed with **otherside_set_broken_reg** to create **set_broken_sync.**

The common broken status register is frozen when latched_broken_status is true. Note, All status signals for the I/O register have the same timing as broken.

Note the bus error driven on the bus at the time the board went broken. This timing is important.

Board Breaking Timing on Info Loopback Error

As shown in Figure 16, if a loopback error occurs that will break the board, bus_error will be asserted in POST2 and the board will break in ERR1. Both ASICs will know of the error because of there are four lines for each info bus that communicate loopback status between the ASICs (cmp_err_in, cmp_err_out, drive_err_in, drive_err_out.

Board Breaking Timing on Heuristic or Arbitrary Broken.

As shown in Figure 17, boards that break due to a Heuristic or an Arbitrary broken will participate in the entire error sequence and then deassert board_not_broken in ERR2.

Xbus Physician Partitioning

This section gives a high level overview of the physical partitioning for the interface ASICs to the Xbus. It describes the components necessary to interface to the Xbus as well as a functional description of how the interface is intended to operate. The fault tolerant strategy is also described.

Info Bus Partitioning

As shown in figure 18, the C and D ASICs on the CPU board each drive half of the bits on the info bus when they are bus master. The C and D ASICs on the PCIB drive either half or all of the bits, depending on the type of transfer. During normal accesses, they drive half the bits as shown; during single-side accesses, they drive all of the bits, including the parity bit.

Control Signal Partitioning

The D-side ASICs drive the control buses, and both C and D-side ASICs check.

Xbus Voted Signal Partitioning

The 3-way voted signals in a Polo system have varying topologies depending on function. In general, each ASIC on a board drives half of the 3-way voted signals originating on a board, and the other side ASIC sits at the end of the net and checks what is driven.

Figure 19 shows the 3-way voted signals for board_not_broken_routing. Each transceiver section (shown as a NAND gate or inverter) of a 3 way voted triplet is in a separate package, to prevent a single failure from disturbing more than one bit of the triplet.

Signal Routing

The bidirectional info buses are routed point to point between boards. On each board, separate traces run from the connector to each ASIC; this allows the error protocol to determine if a fault is in the backplane or in the board. Series resistors are used to control signal quality, nd CMOS levels re used for increased noise margin and reduced susceptibility to cross-talk. Figure 20 shows the routing of info lines.

As shown in Figure 21, the three way voted signals are routed as separate point to point connections. The reset__x,__y__,z__ signals are driven by the CPU boards, wire-ORed on the backplane, and received by all boards. The board __not__broken__ signal is driven by a single board and received by all boards. These signals are drive by normal 4mA drivers of the ASIC, and buffered by 26S10 trancseivers. The voted signals are terminated with a pull-up and isolation diode on both the receiver and driver side. This means that it is possible for there to be one or two pull ups on the line. However, whenever the line is being used for activity signalling, there are exactly two terminators, one at either end of the line.

Functional Description

Figure 22 is a block diagram of the error check logic used by the ASIC's. The Error Checking and Registering logic performs two basic functions. It controls most functions or error checking and it registers incoming and outgoing Xbus signals. The error checking function performs the following tasks:

- ECC generation and checking/correcting
- Parity generation and checking
- Loopback checking
- Three-way voting
- Controls error protocol (includes decisions on breaking board)

- Provides logic for error insertion
- Records error information

The registering logic performs the following tasks:

- registers incoming bus signals
- registers outgoing bus signals
- combines (logical 'or') neighbor, opposite, peer versions of registered ECC corrected control signals coming into the ASIC with the delayed outgoing corresponding signal to produce a combined control signal for internal ASIC module use

The Xbus can be broken into four classes of signals when discussing error checking:

Xbus Signals: these signals are protected by one parity bit, this detects an odd number of errors. During normal accesses the D-side gate array drives approximately half of the signals and the C-side drives the other half. Please refer to Section 11.1 on page 96 for more detail on the signals that are driven by D-side and C-side. The D-side checks the data it has driven (loopback drive check) and the D-side checks the data that the C-side had driven with the D-side would have driven (loopback compare check). During single sided accesses (PCIB only) the driving ASIC drives all of the bus signals. The driving ASIC can do a loopback drive check but the other ASIC cannot do a loopback compare check because it has no knowledge of the exact data being driven.

Control Bus Signals: these signals are protected by four bits of ECC, this detects a double bit error and can correct a single bit error. The D-side drives the signals and the C-side does the loopback compare check. The D-side does the loopback drive check.

Voted signals: these signals are triplicated and protected by three-way voting, this allow for one out of the three lines to be in error. The D-side drives the signals and the C-side does the loopback compare check. The D-side does the loopback drive check. The one exception to D-side driving all signals is the board_not_broken_signal. The C-side will drive the board_not_broken_signal and the D-side will drive the enable for the board_not_broken_signal.

Miscellaneous signals: The slot id signals are duplicated and they are in disagreement the board will break.

Figure 10 shows the Bus Error flow chart, summarizing the foregoing. A further understanding of the invention and of the construction and operation of the illustrated embodiment may be attained by references to the Appendices filed hereiwth.

Summary

Described above is a digital data processing system meeting the aforementioned goals. Those skilled in the art will appreciate that the illustrated embodiment is but one example of the invention and that other embodiments incorporating modifications thereto fall within the scope of the invention, of which we claim:

Polo Programming Guide

JED-00152

"A SAM is a logical concept"

Jeffrey Somers
Gregory Green
Michael Homberg
Conrad Clemson
Will Leavitt
Joe Lamb

26 December 1995

Revision: 3.2

Notice:
Released Version

1. Revision History

Rev 0.0: April 26, 1994

Created.

Rev 1.0: Aug 8, 1994

- Many updates for software

Rev 1.1: Aug 10,1994

- Updates from internal review for sections up to interrupts.

Rev 2.0:

- Release Version

Rev 2.2: March 21, 1995

- FTIO updates

Rev 3.0: July 19, 1995

- Removed obsolete FTIO information

Rev 3.1: September 14 1995

- HSC2 Updates

Rev 3.2: SDecember 26, 1995

- Internal Review HSC2

1.1 Open Issues

Open Polo related issues are also tracked in problem under pci_common. This list reflects open issues that will not be closed for the current release of the specification along with a justification for the issues to remain open.

Open HSC2 related issues are tracked in problem under hsc2_issues. This list tracks unresolved issues during the development cycle of the HSC2 project.

1.1.1 Polo UPS communications

Polo will specify, support and qualify an external UPS solution. The actual vendor selection for the UPS has not occurred yet. Only vendors with an RS-232 interconnect are being considered. When the vendor has been selected, the section on UPS support will be updated to provide the communications specifications.

2. Introduction

This specification is the Polo Programming Guide for the Polo PCIB based system. It provides both the functional model for the hardware and the software detailed design specifications. It is intended to be the primary working model for the programming guide, and it specifies the software visible hardware implementation.

2.1 Applicable Documents

2.1.1 Stratus Applicable Documents

- Cougar Specification, JED-0005
- Xbus Functional Specification, JED-0155
- Cyclops ASIC Functional Specification, JED-0159
- Gambit ASIC Functional Specification, JED-0160
- Polo Product Specification, JED-0151
- Polo Backplane Specification, JED-0158
- Xbus Functional Specification, JED-0155
- Mirage ASIC Functional Specification, JED-01??

2.1.2 Other Applicable Documents

- PCI Local Bus Specification
  Rev-2.0, April 30, 1993
  PCI Special Interest Group
- PCI System Design Guide
  Rev-1.0, September 8, 1993
  PCI Special Interest Group
- PCMCIA 2.1/JEIDA 4.1 Specification

2.2 Terminology

- **Cyclops** - The Polo Ibus to Xbus ASIC. This ASIC contains many of the features of the Gofer ASIC.
- **Gambit**- The PCIB based PCI/Xbus interface ASIC.
- **Mirage** - The HSC2 based PCI/Gbus interface ASIC.
- **GBI** - Golfbus(Xbus) Interface ASIC. This is the Gofer in Jetta systems and the Cyclops in Polo systems.
- **Flash ram** - Flash RAM in this specification refers to a PCMCIA flash ram card. The flash ram card is used for boot functionality.
- **FTIO** - FTIO is the name for the overall I/O subsystem that encompasses the MIO board, the fault tolerant I/O bus, and the SAM and PCI cards.
- **Gofer**- The Jetta based GBI ASIC.
- **HSC2** - The Jetta Golfbus/PCI interface board
- **IOVA** - I/O virtual address.
- **IOBus** - The fault-tolerant bus between the MIO board and the SAM modules.
- **MIO** - The Jetta Golfbus/IOBus interface board.
- **OpenBoot** - This is a proposed standard interface for booting any OS from a PCI card. It is independent of the operating system, the system hardware and the microprocessor. This standard is still under negotiation.

- **Polo** - The new low end system. The Polo system is fully described in the Polo Product Specification.
- **PCI** - Peripheral Component Interface; PC Bus under development by Intel and others as a standard for PCs. The expectation is that peripheral device controller chip sets will be available which can reside directly on the PCI bus
- **PCIB** - The Polo equivalent of a SAM module.
- **PCMCIA** - PC Memory Card International Association. PCMCIA is an industry standard bus which is used for boot functionality on Polo and FTIO.
- **PMI** - Processor/Memory Interface ASIC. This is the Cougar in Jetta and Polo systems
- **SAM** - Stratus Adapter Module - A Jetta-based board which provides the fault-tolerant multi-drop interface to the IOBus. SAM is a generic term for IOBus adapter (not just PCI board adapters). This specification deals only with the SAM/PCI version of the SAM adapter.
- **TRID** - Transaction ID - This binary number provides an identifier for Golfbus and Xbus operations.

3. Functional Overview

Figure 1 below shows a block diagram of a one SAM bucket system that both the FTIO, HSC2, and Polo systems present to software. This diagram is meant to show only the programming model and not the actual implementation.

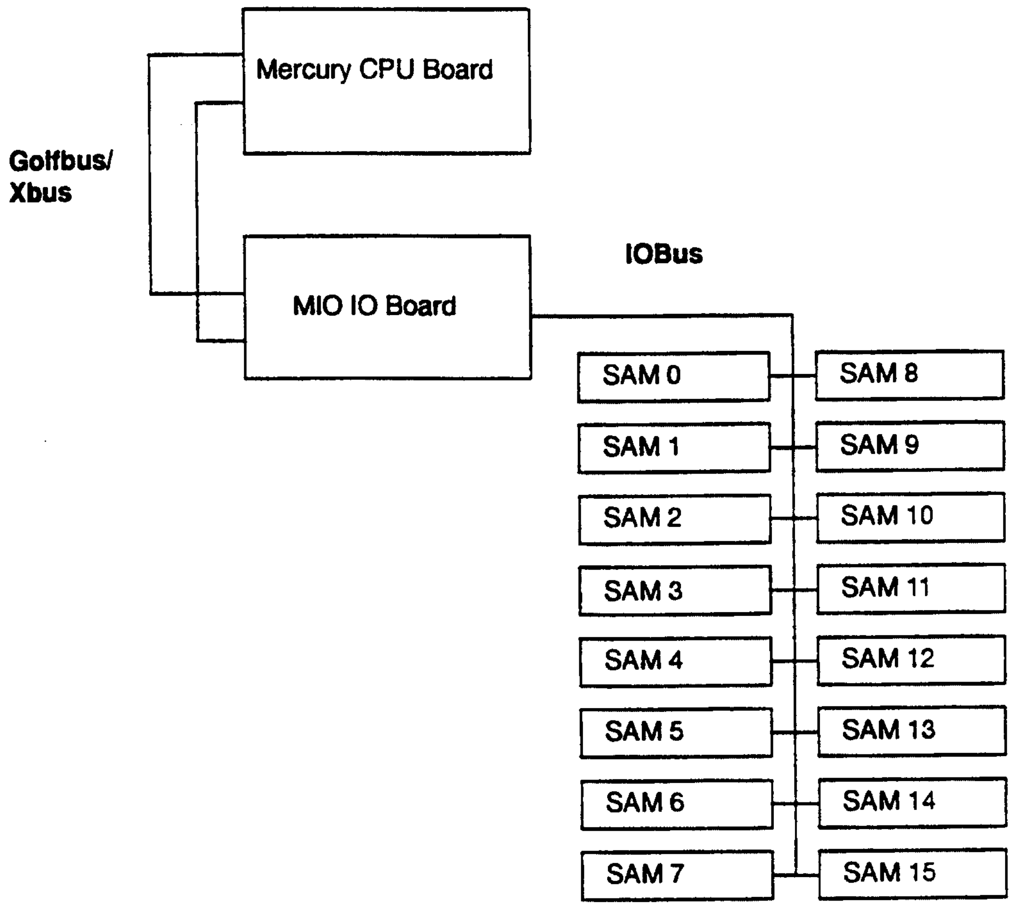
For a Polo system, software should view the system as a 4 slot Golfbus based system. There are Mercury CPUs in slot 0 and 1 and MIO IO boards in slots 2 and 3. The MIOs in a Polo system will never be in a duplexed state.The MIO can connect to one PCI bucket with up to 16 SAMs in it.

For an FTIO system, there can be CPUs in slots 0-3. MIOs can reside in slots 2-11. The system can support multiple MIOs. Each MIO can support 1 to 64 SAM cards residing in 1 to 4 SAM buckets.

For an Jetta based HSC2 system, there can be CPUs in slots 0-3 and MIOs can reside in slots 4-11. The MIOs in a Jetta system will never be in a duplexed state. The MIO can connect to 1 PCI bucket with up to 4 SAMs in it.

Throughout the document, many of these basic blocks are referred to in both Polo and FTIO descriptions. Although mentioned in the terminology section, several of these components are extremely important in understanding of how both systems work and present a compatible view to software. For this reason, these blocks are described below. In some cases, these units are real hardware, in others these blocks are pure abstractions, representative of potential hardware, or merely first instantiations of potential hardware.

Figure 1. Logical High Level Diagram of the Polo and FTIO System



Mercury CPU board: the mercury CPU board is the first CPU board in the Jetta product family. The Mercury CPU board is expected to be the first CPU board in the FTIO product. While a modification of this CPU is used in Polo, it is identical from a software perspective to the Mercury CPU. When the term CPU or Mercury CPU is used in this document it refers to either the Polo or FTIO CPU and any follow on CPU product.

FLASH Card: PCMCIA card containing Flash (non-volatile) memory. Information can be organized as an ATA disk or as memory.

FTIO: Fault-Tolerant IO subsystem consisting of an MIO pair, IOBus, Repeater pair, SAM and SAM backpanel. The subsystem offers fault-tolerant connectivity to an open bus standard, namely PCI.

Golfbus/Xbus/System Bus: FTIO and Polo use different system buses. Golfbus is the Jetta system bus. Xbus is the Polo system bus. Except for M&D these buses are identical from a software perspective. The M&D differences are derived form the different approach the buses must take to bus errors. Golfbus, Xbus, and system bus are used interchangeably in this document to refer to the system bus interconnect.

IOBus: IO Bus is a Fault Tolerant IO bus unique to the FTIO implementation of the system. This bus is transparent to software except for M&D handling of error conditions.

MIO: MIO is the FTIO interface board which connects the system bus to the IOBus. In the FTIO subsystem this is a single board duplexable unit. In the Polo system, the MIO is an abstraction. Although not implemented as a unique board, its software visible aspects are implemented in different components of the system.

PCI Bus: the PCI bus is an Intel designed peripheral bus which is used as the industry standard IO bus for both FTIO and Polo.

PCI Card: A PCI card is a 3rd party card which interfaces to the PCI bus and an I/O specific interface. A PCI card is the I/O adaptor for both systems.

PCIB: A PCIB (PCI Bridge) is unique to Polo. The PCIB performs the functions of the FTIO MIO, IOBus, and SAM. It is invisible to software, except from an M&D standpoint.

PCMCIA: Personal Computer Memory Card Interface Association. Interface Standard used typically for lap tops and notebook pcs for memory and i/o devices. The cards are 54mm wide by 85.6 mm long (credit card size). The cards are available in 3 heights, 3.3mm, 5.0mm and 10.5mm.

SAM: The SAM is the interface and control unit for the PCI cards. In the FTIO system, the SAM is a hot pluggable card which performs isolation and fault tolerance for the PCI cards. In the Polo system, the SAM is an abstraction for the logic and software visible aspects of PCI card control. In this document, the term SAM applies to the software visible features associated with control and fault tolerance with respect to the PCI bus.

SAM Repeater: The SAM repeater is a unique SAM which interfaces to a PCMCIA bus. In the FTIO this card also performs some low level electrical and M&D functions.

3.1 Polo Functional Overview

The Polo Product is a Jetta Mercury and FTIO only based system. From a high level, some of the key features of the CPU board are:

- PA7100 Based Microprocessor, 72 or 96 Mhz, 256KB or 1M data and instruction cache.
- 1 or 2 microprocessors per board
- 128MB, 256MB, 512MB or 1GB of DRAM memory.

This implementation of the MIO subsystem has the following key features:

- Standard I/O Bus support (PCI)
- 1 to 16 PCI cards supported.

3.1.1 Logical Summary

Figure 1 above is a logical diagram of the overall system. As pointed out, many of the specific features, SAM, IOBus, and MIO, are abstracted in the Polo system.

3.1.2 Physical Summary

While the logical view of the system is that of a standard Golfbus machine, the physical view is very different. figure 2 below illustrates an overview of the Polo system.

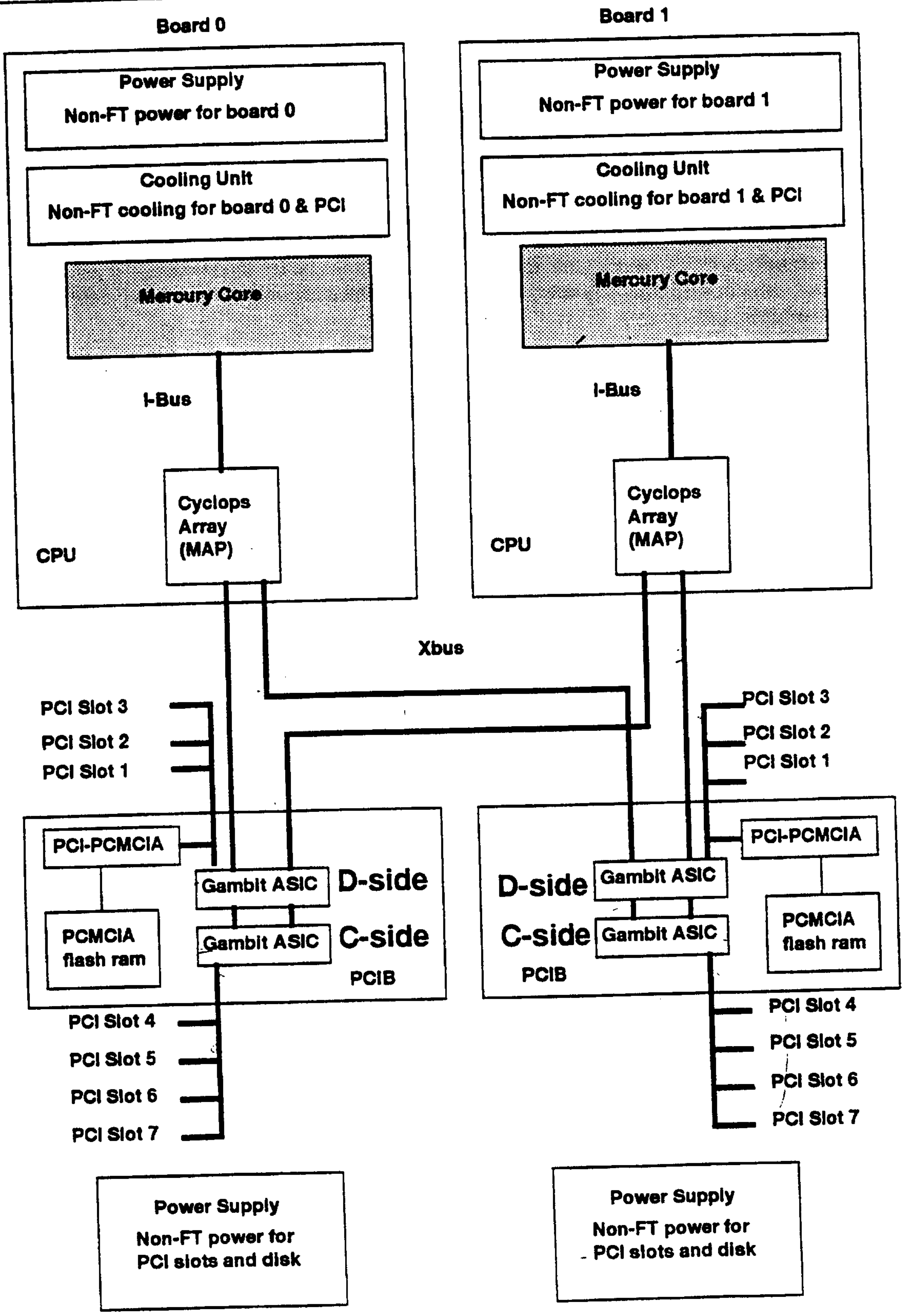
There are two major logical components that will be designed by Stratus in this figure. The first major component is the mother board consisting of the Mercury core (snoop subsystem, cougar, 1 or 2 Lisbon memory cards, and on or 2 logical PA modules) and ReCC. The new ASIC on the motherboard, the Cyclops ASIC, contains all software visible aspects of the Gofer and MIO Golfbus ASICs. From a software perspective, the Golfbus is inside Cyclops. As explicitly defined in later sections, all internal registers are either implemented or will return intelligent responses to software operations.

The second major component is the PCI interface board. This board contains two instances of the CPU-PCI interface ASICs, the Gambit ASIC. Each Gambit acts as the interface between the CPU and one PCI bus. Each Gambit emulates the functionality of 4 SAM cards (one for each PCI slot).

The interconnect between the CPU boards and the PCI interface boards is a set of cross-couple buses. These buses are fault detecting, but not correcting. The buses look like a hybrid Golfbus and are named the Xbus. A complete description of the bus interface can be found in the Xbus Functional Specification. Memory updates, Regurgitated infos, and other CPU to CPU traffic is run across the Xbus.

In Polo, the MIOs will always appear to be simplexed. All accesses can be performed to either slot 2 or slot 3 address space, but must be performed as non-paired accesses. Paired accesses will be responded to as TBD. Just don't do it...

Figure 2. Polo System Overview



3.2 Jetta/FTIO Functional Overview

The FTIO project was cancelled mid-project. Although it is no longer a planned project, there was a tight compatibility goal between polo and FTIO. The remnants of this remain in the design. For this reason, the following sub-sections are included in the specification to provide background to the reader.

This FTIO/MIO/SAM implementation has the following key features:

- Standard I/O bus support (PCI and PCMCIA).
- Up to 64 SAMs (Total of Repeaters plus SAMs)

3.2.1 Logical Summary

The logical view of the FTIO sub-system is consistent with Figure 1.

3.2.2 Physical Summary

FTIO is a standard Jetta Golfbus interface Sub-system. There are four major logical components that will be designed by Stratus in this figure. The first major component is the MIO primarily consisting of the IF ASIC. The IF ASIC provides a bridging function between the Golfbus and the IOBus. There is no processor or memory resident on MIO.

The second major component is the Repeater. The Repeater boards receive the point-to-point differential ECL IOBuses and converts the levels to BTL for use in the multi-drop SAM backplane. The Repeaters provide power for the SAM backplane termination and also re-broadcast the differential ECL IOBuses to allow daisy-chaining up to 4 SAM chassis. (Each SAM chassis contains 2 Repeaters and up to 14 SAMs) Each Repeater consists of ECL and BTL bus transceivers and an IAM ASIC. The IAM ASIC provides the host-bridging function from IOBus to PCI Local bus. The IAM also provides Repeater specific support when used in the Repeater mode. The Repeater uses a 3-way voted clock generation circuit. The Repeaters also support a type II or III PCMCIA interface.

The third major component is the SAM. Each SAM consists of BTL bus transceivers and an IAM ASIC. The IAM ASIC provides the host-bridging function from IOBus to PCI Local bus. Each SAM supports a single 32-bit PCI slot. The IAM provides system memory protection for bus mastering PCI adapters and also SAM specific support when used in the SAM mode (as opposed to repeater mode). The SAM uses a 3-way voted clock generation circuit.

The fourth component is the SAM Backplane/SAM clock board combination. The SAM backplane is a 16 slot backpanel which supports 2 Repeater and 14 SAMs. The backpanel will conduct the bulk power to each SAM and Repeater, eliminating the need for bus bars. The backpanel will also provide the cable attaches for the IOBus cables, bulk power connections, CDC connections and a JTAG port for scanning each slot individually. The backplane will also have connectors for mounting the SAM clock board. The SAM clock board will provide point-to-point triplicated clocks to each SAM and Repeater board. The first iteration of the clock board will be a high-reliability model. A follow-on project can replace the high-reliability clock with a fault-tolerant clock. The high-reliability clock cannot be replaced on-line.

3.3 Jetta/HSC2 Functional Overview

The HSC2 project started during first release of Polo hardware. One of the goals of HSC2 is software compatiblity with Polo for use in Jetta-12 and Jetta-6 configurations. Sections of this document are being updated to highlight differences and give the reader insight

The Jetta/HSC2 implementation has the following key features:

- Standard I/O bus support (PCI and PCMCIA).
- 1 to 4 PCI cards supported per Jetta slot.

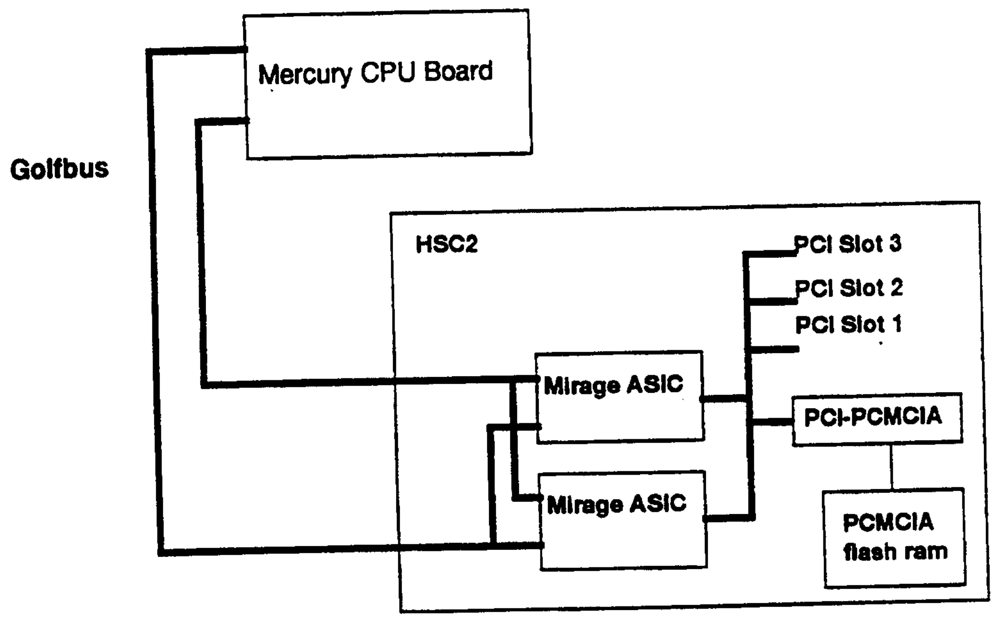
3.3.1 Logical Summary

Figure 1. is a logical diagram of the overall system. As pointed out, many of the specific features; SAM, IOBus, and MIO are abstracted in the Jetta/HSC2 system.

3.3.2 Physical Summary

The logical and physical view of the system is that of a standard Golfbus machine. Figure 3 below illustrates an overview of the Jetta/HSC2 system.

Figure 3. HSC2 System Overview



The major component that will be designed by Stratus is the HSC2 Jetta I/O board. The HSC2 is a simplexed board so all accesses must be performed as non-paired accesses.

The new ASIC on the HSC2 board, the Mirage ASIC, contains all the software visible aspects of the MIO Golfbus ASIC and also emulates the functionality of 4 SAM cards (one for each PCI slot).

4. Xbus/Golfbus to PCI Address Mapping

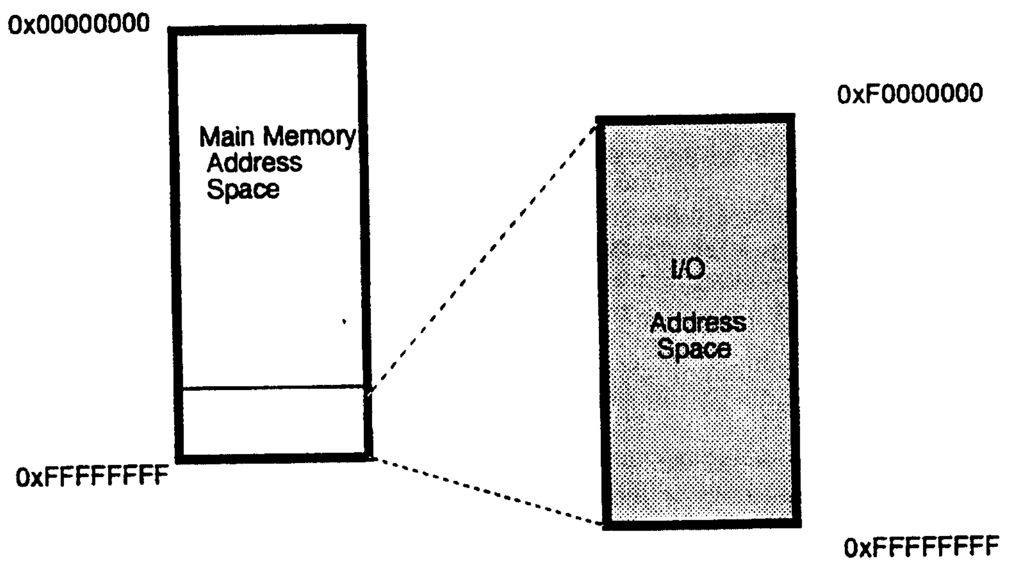
This section outlines many aspects of addressing on the Polo PCIB and Jetta HSC2 systems. The overall high level address map is presented and PCIB/HSC2 specific aspects are outlined. The address translation map is presented and its outline and management are discussed.

4.1 Addressing Overview

The Polo system address map is designed to preserve maximum compatibility with the Golfbus address map. As such, it preserves the support for 15 logical slots, etc. even though Polo is a fixed configuration of two CPU boards and two I/O boards.

The PA7100's address map is broken down into two segments, main memory and I/O space. All of main memory is cacheable and all of I/O space is non-cacheable. There are no limitations as to what virtual address is mapped into what physical I/O space.

Figure 4. PA7100 Physical Address Map



By mapping I/O space into the virtual address space and using the same page translation mechanism as that employed for memory, the PA7100 architecture allows the flexibility for non-privileged users to be given access to some regions of I/O space without compromising system security. All duplexed boards' I/O register definitions can be accessed through local, paired, non-paired, and global space and all simplexed boards' I/O register definitions can be accessed through local, non-paired, and global space. MIO mirrors it's Golfbus registers so that they can be accessed in "freeze space" to allow a CPU controlling the duplexing process to access MIO's registers while MIO is in freeze state.

- **Local Space** - Resource is local (same-board) to the requester and therefore does not require an Xbus/Golfbus cycle. Registers such as CPU interval Timers, Interrupt & Mask Registers are good examples of frequently accessed registers that are optimally handled locally. When a board is off-line it can only access and test I/O on-board through local space.

- **Non-Paired Space** - I/O space is accessed explicitly by slot number. Note that boards that are duplexed are required to go through the motions on non-paired I/O but only the addressed slot will perform the write for write operations or drive data onto the XbusGolfbus for read operations.
- **Paired Space** - I/O space is accessed explicitly by slot number; however the least significant bit, which differentiates the even/odd slot of a duplexed pair, is ignored. There are registers in I/O space that are not allowed to be read through paired space such as a per-board broken status register.
- **Global Broadcast Space** - Global I/O space is defined as write-only space (software needs to enforce this). It's primary purpose is to provide a means of broadcasting synchronous commands to multiple boards in the backplane simultaneously.
- **Freeze Space (Paired and Unpaired)** - This "MIO only" space resides at offset 7FExxxx and allows a CPU that is handling the duplexing of the MIOs to access the Golfbus I/O registers while MIO is in freeze state. Normal paired and unpaired accesses (offset 7FFxxxx) get busied.

4.2 Address Re-mapping

Polo/HSC2 use address re-mapping to protect the system from errors caused by simplexed I/O devices. Note that the I/O Virtual Address or IOVA format used by Polo & HSC2 systems differs from the IOVA format used on BIO systems.

In HSC2 systems the IOVA is translated into a system address inside the Mirage ASIC on the HSC2 board. The Map RAM is also internal to the Mirage ASIC which is self-checking.

In Polo, all system accesses from the I/O ASICs are performed using the IOVA (IO virtual address); these are mapped to the final Xbus style physical/virtual index addresses (CPU addresses) by a map RAM in the Cyclops ASIC. Both IOVA and CPU addresses are passed on the backplane, with a bit in the trid field distinguishing between them.

4.3 Per-Space Ordering of I/O Accesses

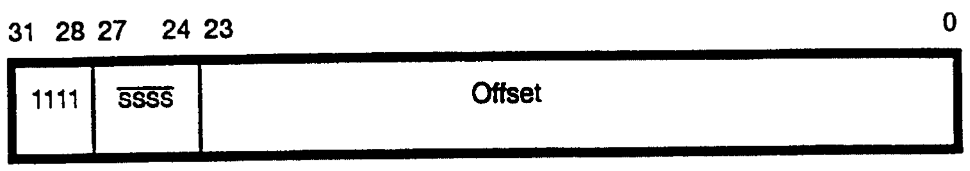
IO accesses requiring the Xbus/Golfbus (accesses to paired, non-paired, or global space) are ordered relative to each other for a given processor. I/O accesses to local space are only ordered relative to other local I/O accesses from the same processor. In particular, if a write to a register in global space is followed by a read of that register in local space, it is possible for the read to occur before the write takes place. Verification of global writes should be performed by non-paired reads.

There is no ordering guaranteed between I/O space accesses and memory space accesses. A mechanism for making I/O accesses sequential to memory accesses is board specific and described in the relevant board specifications.

4.4 Address Decode

The Golfbus I/O address map used by Polo and HSC2 systems is subdivided into 16 segments of 16Mbytes each. The first 15 segments correspond to Slots 0-14 in the backplane; slot 15 is reserved to map local and global I/O Addresses. A Polo system looks to the programmer like a four slot Golfbus backplane. Note that the slot identifier field of the I/O address is inverted to allow local address space to begin at absolute address 0xF0000000 which corresponds to the boot address issued by the PA7100 upon being released from reset.

Figure 5. Physical I/O Address Decode.

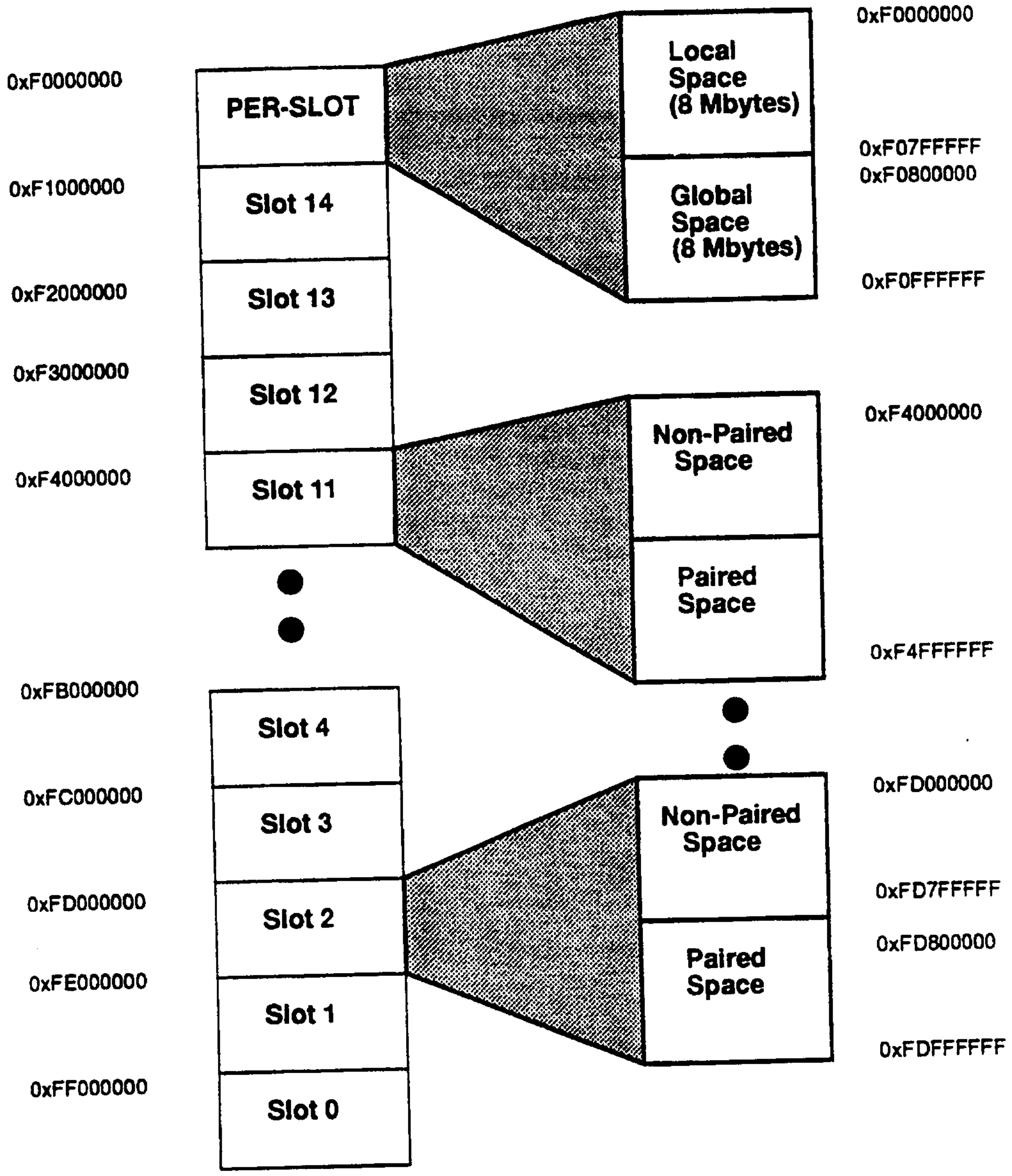


Where: $\overline{ssss}$ is the slot # inverted:

1111 -> Slot #0
0000 -> Slot #15

Each slot with the exception of slot 15 is subdivided into paired and non-paired space which effectively halves their logical address space from 16Mbytes to 8Mbytes. All registers on duplexable boards are addressable through either paired or non-paired space. Accesses to paired space on a non-duplexable board are ignored.

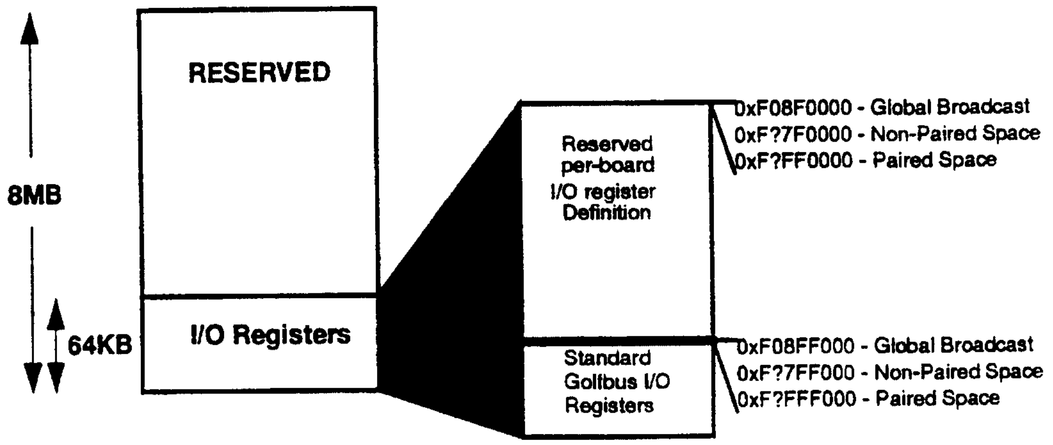
Figure 6. Physical I/O Slot Decode



4.4.1 Per-Slot I/O Space (Local, Paired, Non-Paired)

In general the view of XbusGolfbus I/O space is identical whether it is addressed through local, non-paired, or paired space. All board I/O registers will reside in the last 16 pages of I/O space and each board in the Polo/HSC2 architecture will have a common set of registers that reside on the last physical page of the per-slot I/O segment to simplify the M&D (Maintenance & Diagnostic) interface. Additionally the last 1 Mbyte segment of a board's address space can be touched via global broadcast write operations, which are described in greater detail in the next section.

Figure 7. I/O Space Map

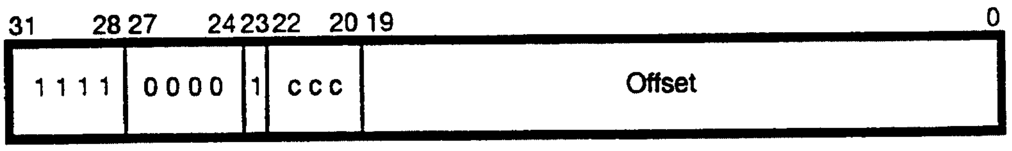

RESERVED
8MB
64KB
I/O Registers
Reserved per-board I/O register Definition
Standard Golfbus I/O Registers
0xF08F0000 - Global Broadcast
0xF?7F0000 - Non-Paired Space
0xF?FF0000 - Paired Space
0xF08FF000 - Global Broadcast
0xF?7FF000 - Non-Paired Space
0xF?FFF000 - Paired Space

? = inverted board slot #

4.4.2 Global Broadcasts

Global broadcasts are write-only and are visible to all boards through slot 15 as illustrated in figure 6. Global broadcasts can be issued to all boards simultaneously in the backplane or to classes of boards such as all CPU/MEM slots. Global broadcast operations must be architected in such a manner that all boards that receive them perform the operation synchronously relative to one another. Figure 8 below indicates the decoding for global broadcasts.

Figure 8. Xbus/Golfbus Global I/O decode



CCC = Command Type

000	= Broadcast to all Boards
001	= Broadcast to CPU/Memory Boards
010-111	= Reserved for Other Board Classifications

4.4.3 Illegal/Abnormal I/O Accesses

Table 1 summarizes the behavior of various types of illegal/abnormal I/O accesses. Note that as described in section 12.1 on page 82, though all common registers are 32 bits, they are spaced at 64 bit (8 byte) intervals to aid future board designs which may make use of 64-bit internal buses.

Also note that the common logic provides an indication to board specific logic that a read access has timed out/no-ACKed, or that a write access has failed (i.e. a simplex source broke after successfully driving info but before driving data). Different boards handle these conditions in different ways. Mercury boards for example:

- return 0's and signal LPMC's (HP-PA low priority machine checks) for timed-out/NACKed reads
- do nothing for failed writes other than to write 0's or nothing (per the table below).

Table 1. Illegal/Abnormal I/O Registers

	Bus ASIC Registers	Non-Bus ASIC registers
32-bit read on an odd 32-bit boundary	return 0's if within a page that has other registers, else NACKed	unsupported
sub 32-bit read	32-bits returned	32-bits returned. Byte enabled bytes guaranteed to be valid, all byes guaranteed to be deterministic.
32-bit write on an odd 32-bit boundary	ignored	unsupported
sub 32-bit write	ignored	ignored for ASIC I/O, performed as specified by byte enables for PCI I/O.
failed write (i.e. a write from a simplex board which breaks after successfully driving address but before driving data)	ignored	Ignored
write to non-existent register	ignored	ignored for ASIC I/O, PCI will Master Abort
read of a non-existent register	return 0's if within a page that has other registers, else NACKed	Can return 0's if the address decodes to a page that has others registers, else, NACKed, PCI will Master Abort

4.5 PCIB MIO/IOBus Compatible System Address Map

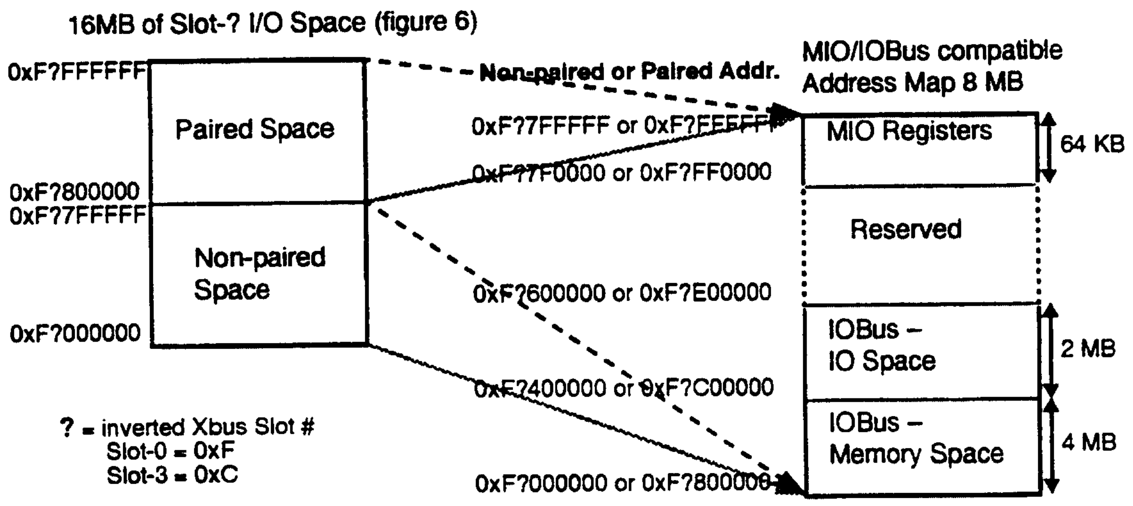
All of the MIO compatible address space can be addressed directly through the Xbus/Golfbus I/O space. Each I/O boards' address space can be accessed directly via set size windows, where the base address of the window can be programmed individually on each board.

The Polo version of the MIO compatible map is a subset of the map presented in the following sections. As previously stated the Polo appears to software as a system with simplexed MIOs in slots 2 and 3.

The Polo PCIB and Jetta HSC2 will only operate in simplexed mode. All IOBus/PCI slots are decoded according to the address map defined bellow.

Figure 9 shows the MIO/IOBus compatible address map in context of the Xbus address space.

Figure 9. MIO/IOBus Compatible Address Map in Xbus/Golfbus Space



The breakdown of the Xbus/Golfbus address bits is illustrated in table 2.

Table 2. Xbus/Golfbus address decoding

Xbus/Golfbus Address bits					Address Space
31:28	27:22	23	22:20	19:16	
4'hF	!slot	P/NP	3'h7	4'hF	MIO Registers
4'hF	!slot	P/NP	3'h7	4'hE	MIO Registers (freeze space) - Mirrored area that allows access while MIO is in freeze state
4'hF	!slot	P/NP	3'h7	4'h0-D	Reserved
4'hF	!slot	P/NP	3'h6	xxxx	Reserved
4'hF	!slot	P/NP	3'h5-4	xxxx	Access to SAM IO Space (32K direct - 16K IAM IO/ 16K PCI IO) (Address bits 31:15 are re-mapped before further usage.)
4'hF	!slot	P/NP	3'h3-0	xxxx	Access to SAM Memory Space (64K direct PCI Memory) (Address bits 31:16 are re-mapped before further usage.)

MIO Compatible Register Space:

- The register space includes all registers of the I/O board, including the standard Xbus/Golfbus I/O registers, the ASIC registers, and the ID PROM space. Many register locations will be commonly defined for all I/O boards. See section 4.5.2 for details.

IOBus Compatible I/O Space:

- This space includes registers and memory on the PCIB/HSC2; a portion of this address space will be turned into I/O cycles on the PCI bus. Only a 32 KB window of the PCIB/HSC2 I/O space can be accessed from the Xbus/Golfbus at a time. The base offset of the window can be changed on the Mirage or Gambit. See section 4.5.3.

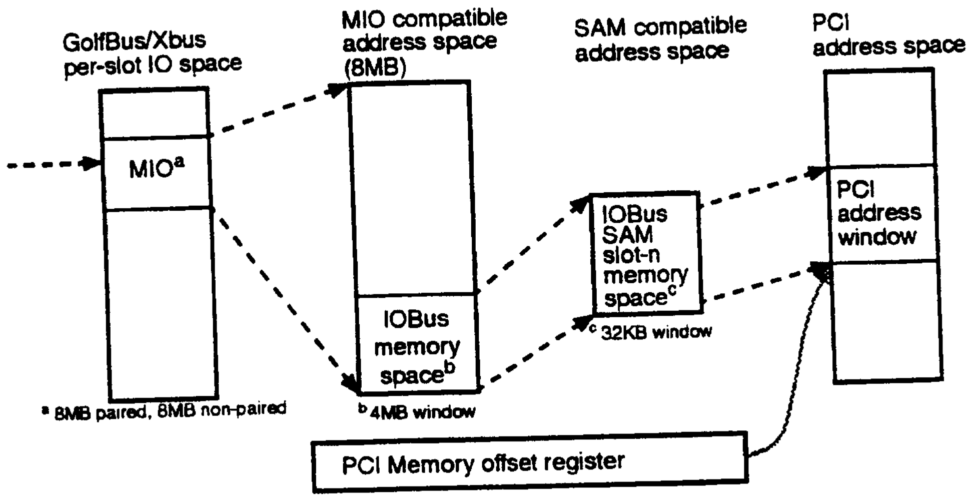
IOBus Compatible Memory Space:

- Access to the memory space portion of the IOBus compatible address space will be turned into memory cycles on the PCI bus. Only a 64 KB window of the PCIB/HSC2 memory space can be accessed from the Xbus/Golfbus at a time. The base offset of the window can be changed on the Mirage or Gambit. See section 4.5.4.

4.5.1 IOBus or Xbus to PCI Local Bus Requests

Base address registers have to be set up in either the Mirage or Gambit ASIC and in the PCI configuration space of the PCI adapter card in order to allow I/O or memory (non-configuration) accesses to the PCI adapter across the PCI local bus from the Goflbus/Xbus. One set of registers (and hence one I/O and memory window) exists for every PCI slot. Figure 10 illustrates a high-level view of this mapping process.

Figure 10. SAM address window scheme

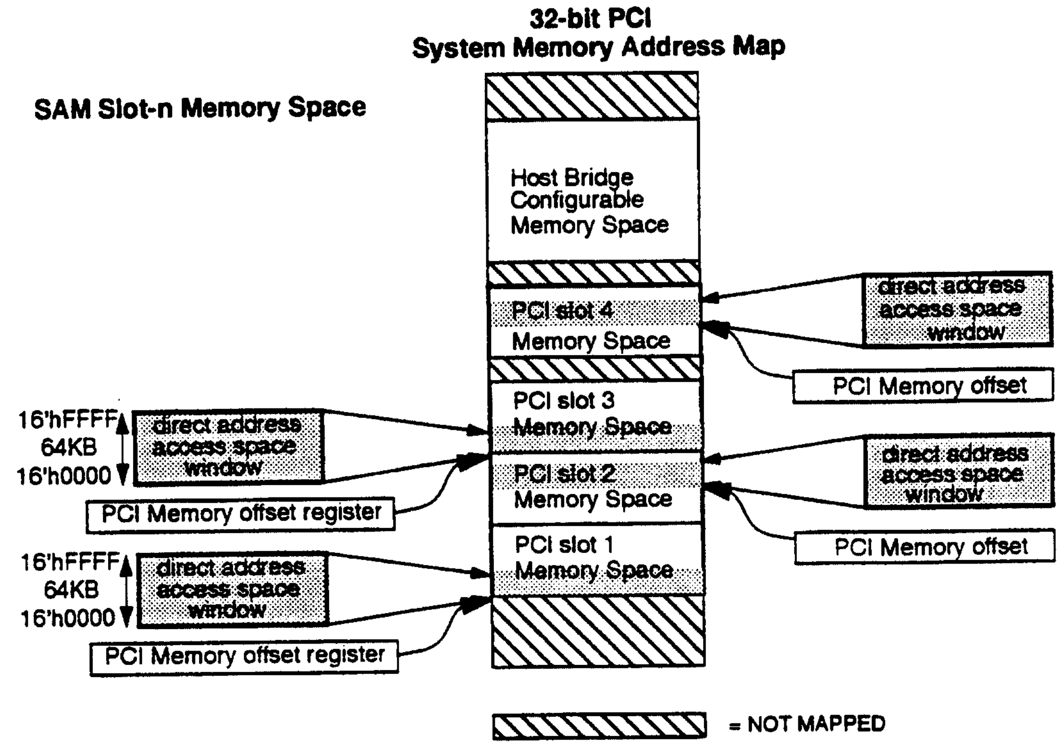

GolfBus/Xbus per-slot IO space
MIO compatible address space (8MB)
SAM compatible address space
PCI address space
MIO^a
IOBus SAM slot-n memory space^c
PCI address window
IOBus memory space^b
^c 32KB window
^a 8MB paired, 8MB non-paired
^b 4MB window
PCI Memory offset register

Direct memory accesses to the PCI adapter must be made through the 64KB SAM memory space window which can be shifted through the entire 32-bit adapter memory address space. I/O accesses to a PCI adapter likewise must be done through the 32 KB SAM I/O space window. The adapter card has to be programmed to respond to a specific memory or I/O area where it will be addressed by the system. Each adapter type may have different memory or I/O range requirements. These requirements can be determined from the base address registers in the adapter's PCI configuration space. The PCI Memory Offset register of the Mirage or Gambit ASIC has to be set to point to the bottom of the selected memory range of the PCI adapter. The memory can start anywhere in the 4GB address range on a 64KB boundary. Also, the PCI IO Space Offset register of the Mirage or Gambit ASIC has to be set to point to the bottom of a 32KB window within the I/O range of the PCI adapter. See the specific PCI adapter card's specification and section 4.5, PCIB MIO/IOBus Compatible System Address Map, for more on this. Thus the procedure for accessing either memory or I/O space on an adapter card is:

- initialize PCI Memory or I/O Space Offset register in the bridge (Mirage or Gambit) to the bottom of a 64KB (memory) or 32KB (I/O) window.
- initialize PCI Memory or I/O Space Offset register in the PCI adapter.
- access memory or I/O window through Golfbus/Xbus I/O space (section 4.5). Any access to the PCI address windows is interpreted as legitimate access using whatever address happens to be in the Offset Registers.

At a minimum the software should guarantee that the PCI slot memory space windows and the host bridge memory space do not overlap for any group of four PCI slots assigned to a single PCIB card. Figure 11 shows an example for a group of four PCI adapters. All four PCI adapters reside on the same PCI bus so care has to be taken in laying out the address windows.

Figure 11. PCI Bus Memory Space Map


32-bit PCI
System Memory Address Map
SAM Slot-n Memory Space
Host Bridge Configurable Memory Space
PCI slot 4 Memory Space
PCI slot 3 Memory Space
PCI slot 2 Memory Space
PCI slot 1 Memory Space
direct address access space window
PCI Memory offset
direct address access space window
PCI Memory offset
16'hFFFF
64KB
16'h0000
direct address access space window
PCI Memory offset register
16'hFFFF
64KB
16'h0000
direct address access space window
PCI Memory offset register
= NOT MAPPED

4.5.2 MIO Compatible Register Space

0xF%6F0000 – 0xF%6FFFFF: Xbus/Golfbus common and MIO specific compatible registers, including access to the ID PROM. For a detailed register address map and description see section 12.

The '%' is equal to the inverted Xbus/Golfbus slot number. The addresses listed are for non-paired space, for paired space addresses change bit 23 from a zero to a one.

4.5.3 IOBus Compatible I/O Space Map

The following IOBus Compatible I/O Device address map allows for 64 addressable I/O slots per IOBus. The address space is divided to I/O Space and Memory Space.

The Polo system only supports slots 0 to 15. The other 48 slots behave as un-populated slots in a

Polo system.

The HSC2 system only supports slots 0 to 3. The other 60 slots behave as un-populated slots in a HSC2 system.

Figure 12. IOBus Compatible I/O Space

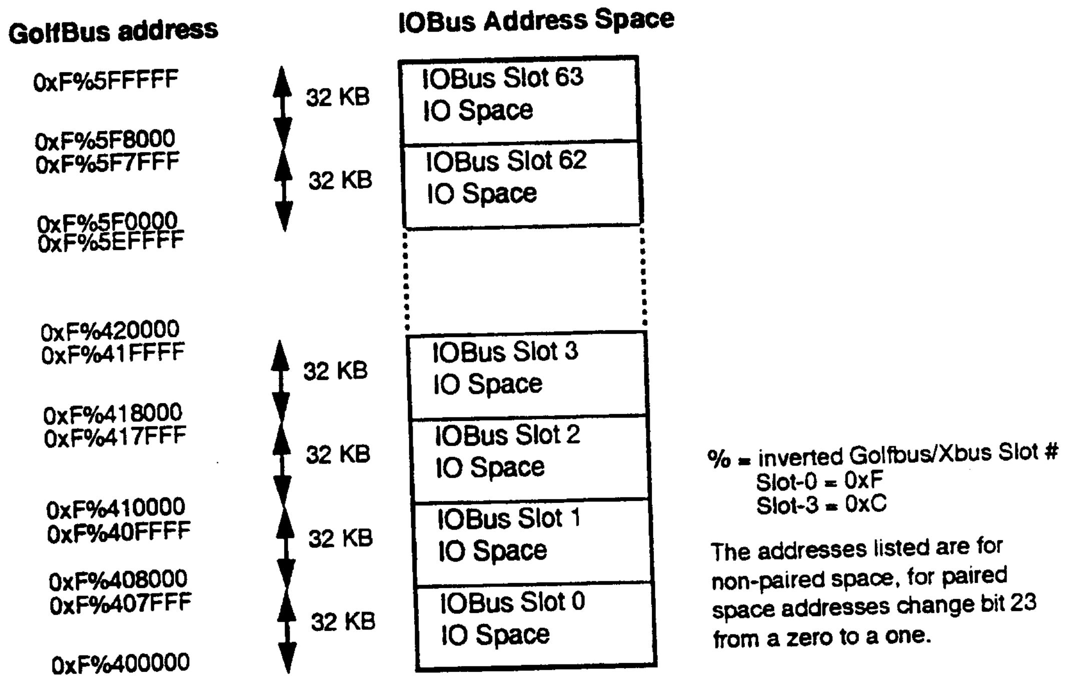


Table 3 illustrates the Xbus/Golfbus address formation for the IOBus compatible I/O space shown in the figure above. The slot field in the address is the inverted Xbus slot number.

Note that SAM/PCI slots 16 to 63 do not exist and the MIO compatible space exists in Xbus slots 2 and 3. Furthermore the Polo PCIB does not support duplexing so it will not respond to paired space accesses.

Note that SAM/PCI slots 4 to 63 do not exist and the MIO compatible space exists in HSC2 Furthermore the Jetta HSC2 does not support duplexing so it will not respond to paired space accesses.

Table 3. IOBus compatible I/O space slot and offset bit positions

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	islot (Xbus)				a	1	0	6 bit slot SAM number						offset into 32K IO Space														

a. paired/ non paired bit

4.5.3.1 SAM I/O Address Space

SAM I/O space is divided up into two spaces (see figure 13).

PCI I/O Space: The PCI Local bus can support a full 4GB of I/O space, but the historical maximum is 64KB due to x86 limitations. Access to the 64KB PCI I/O Space is made through a 16 KB window defined as SAM PCI/IO Space. Accesses to the SAM PCI/IO Space are passed along to the PCI bus with the I/O Read or I/O Write function codes. The position of this 16 KB window can be changed on the16 KB boundary via the PCI I/O Space Offset Register. The base I/O address of the PCI adapter must align with the PCI I/O Space Offset Register of the Mirage or Gambit ASIC. A read that maps to the PCI I/O space of the SAM and generates a PCI cycle that does not map to the I/O space of an adapter will cause a master abort and return zeros. The PCI bus read could also map to the I/O space of the next adapter if the I/O spaces of the adapters are contiguous.

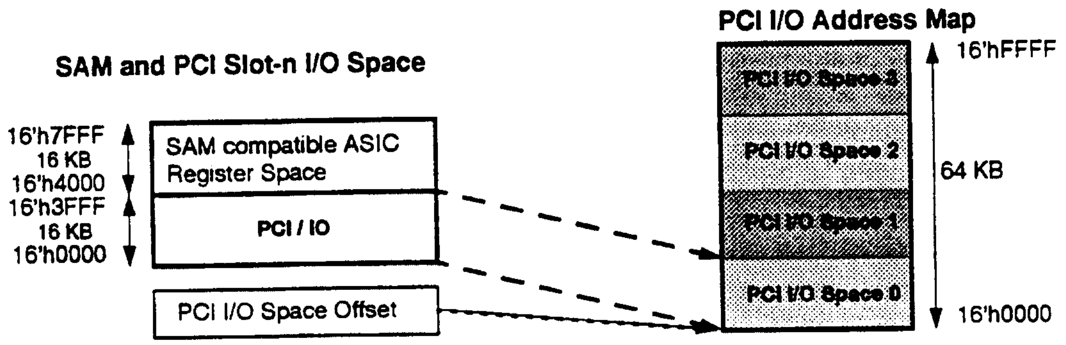
When a sub word access is made to PCI IO space the byte enables are passed through from the read. Since the address from the CPU will have 2'b00 in the address bits [1:0] these two address bits need to be reconstructed from the byte enables as shown in table 4.

Table 4. Lower address bit construction

AD1	AD0	C/BE3#	C/BE2#	C/BE1#	C/BE0#
0	0	X	X	X	0
0	1	X	X	0	1
1	0	X	0	1	1
1	1	0	1	1	1

SAM compatible Register Space: Operations to this 16 KB space provide access to SAM compatible ASIC internal registers. Reads to registers that are not implemented in this space returns zeros.

Figure 13. SAM I/O Space Map



The following table illustrates the formation of system bus addresses within the PCIB/SAM I/O address space.

Table 5. SAM Compatible Register System Address Formation

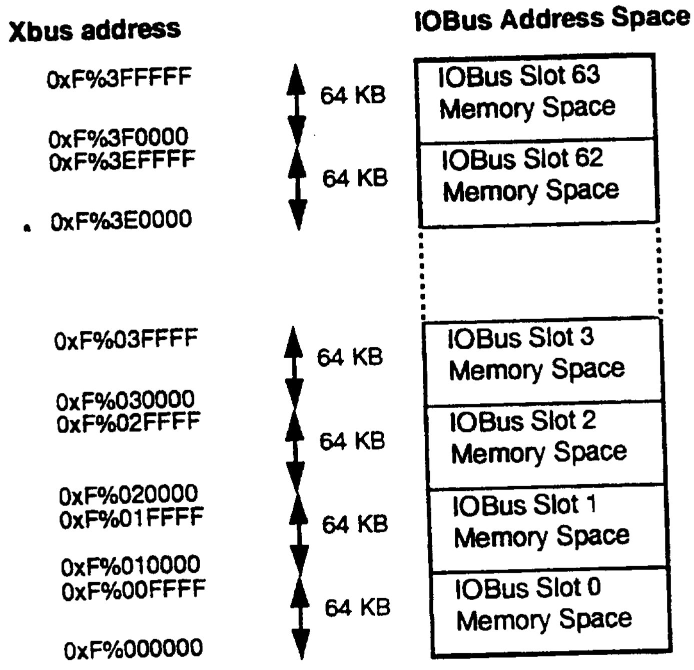
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	System bus slot inverted				a	1	0	SAM/PCI Slot number					b	Offset														

a. paired space -> = 1; unpaired space -> = 0
b. 1 = SAM compatible ASIC register space; 0 = PCI I/O space

4.5.4 IOBus Compatible Memory Space Map

Once again the Polo system only supports 16 slots and the other 48 spaces behave as un-populated slots in a Polo system.

Figure 14. IOBus Compatible Memory Space



% = inverted Golfbus/Xbus Slot #
Slot-0 = 0xF
Slot-3 = 0xC

The addresses listed are for non-paired space, for paired space addresses change bit 23 from a zero to a one.

Table 6 illustrates the Xbus/Golfbus address formation for the IOBus compatible memory space shown in the figure above. The slot field in the address is the inverted Xbus slot number.

Note that in Polo systems, SAM/PCI slots 16 to 63 do not exist and the MIO compatible space exists in Xbus slots 2 and 3. Furthermore the Polo PCIB does not support duplexing so it will not respond to paired space accesses either.

Note that in HSC2 systems, SAM/PCI slots 4 to 63 do not exist. Furthermore the Jetta HSC2 does not support duplexing so it will not respond to paired space accesses either.

Table 6. IOBus compatible memory space slot and offset bit positions

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	islot (Xbus)				a	0	6 bit slot SAM number						offset into 64K memory space															

a. paired/ non paired bit

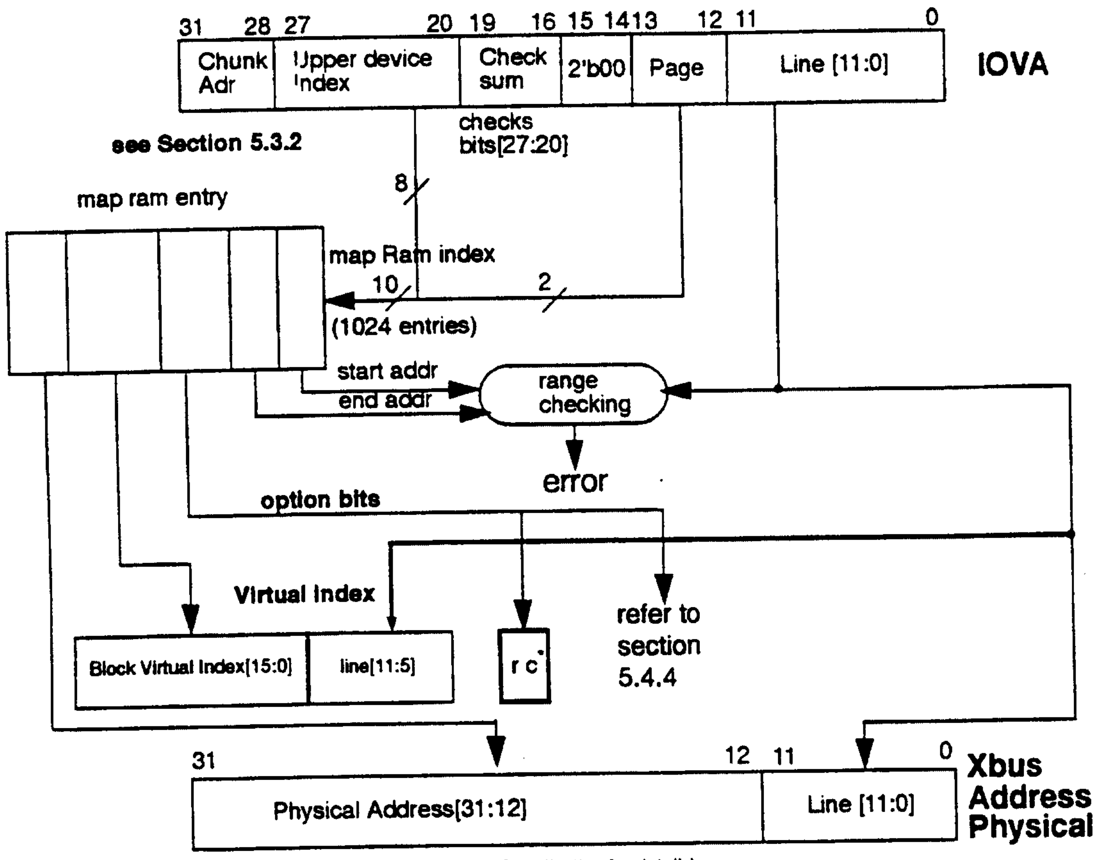
5. Xbus/Golfbus PCI to Xbus/Golfbus Address Mapping

The hardware takes a PCI generated address which we refer to as an IOVA and maps it to a system address.

5.1 Software View of Re-mapping

Figure 15 shows the software view of re-mapping.

Figure 15. IOVA to System Address Mapping



* rc = remote coherant (see Golfbus Functional Specification for details)

5.2 Polo Re-mapping

Polo supports both an addressing scheme compatible with Jetta based processors, 32 address mode, and an addressing scheme compatible with Runway based processors, 40 bit mode. The sequence starts with a PCI Address or IOVA. Gambit takes the IOVA and sends it across the Xbus. Cyclops re-maps the address to a system address and sends it to the CPU via the IBUS. Figure 16 illustrates the address re-mapping for the 32 bit address mode used by Jetta systems and currently used in Polo systems. The line offset shown in the system virtual index of figure 16 is the line offset taken from the Xbus physical address driven during the Xbus IOVA cycle.

HSC2 supports only the 32 address mode compatible with Jetta based processors. The sequence starts with a PCI Address or IOVA. Mirage takes the IOVA and re-maps the address to a system address and sends it to the CPU via the Golfbus. Figure 16 illustrates the address re-mapping for the 32 bit address mode used by Jetta systems and currently used in Polo systems.

Figure 16. Polo System Address generation (32 bit address mode)

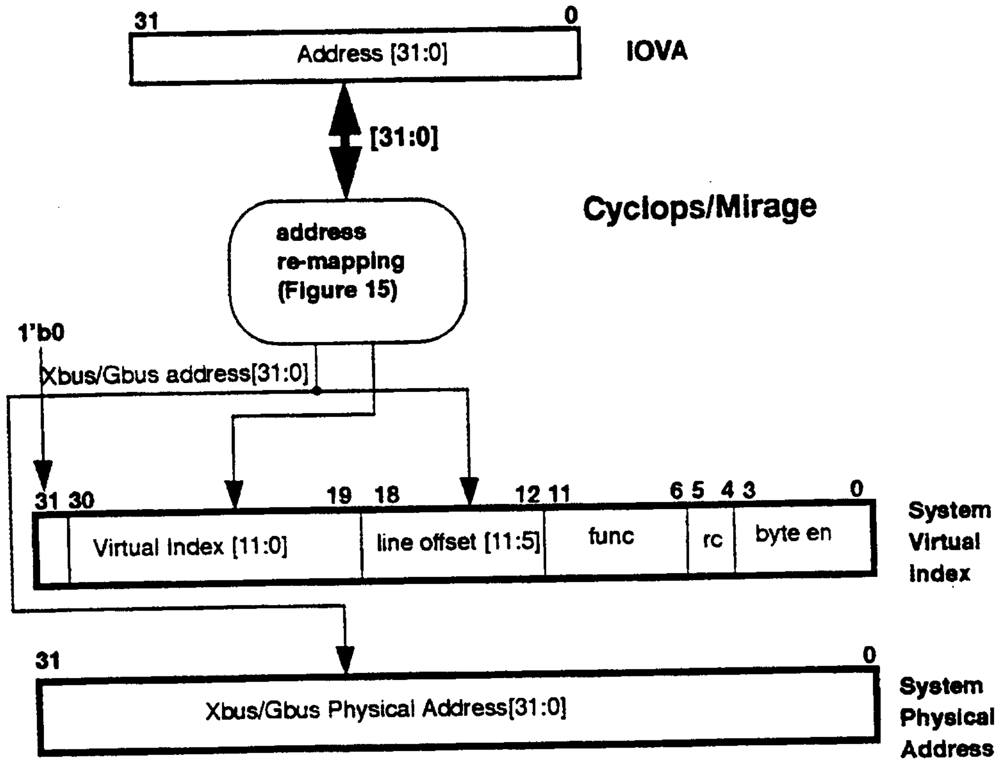
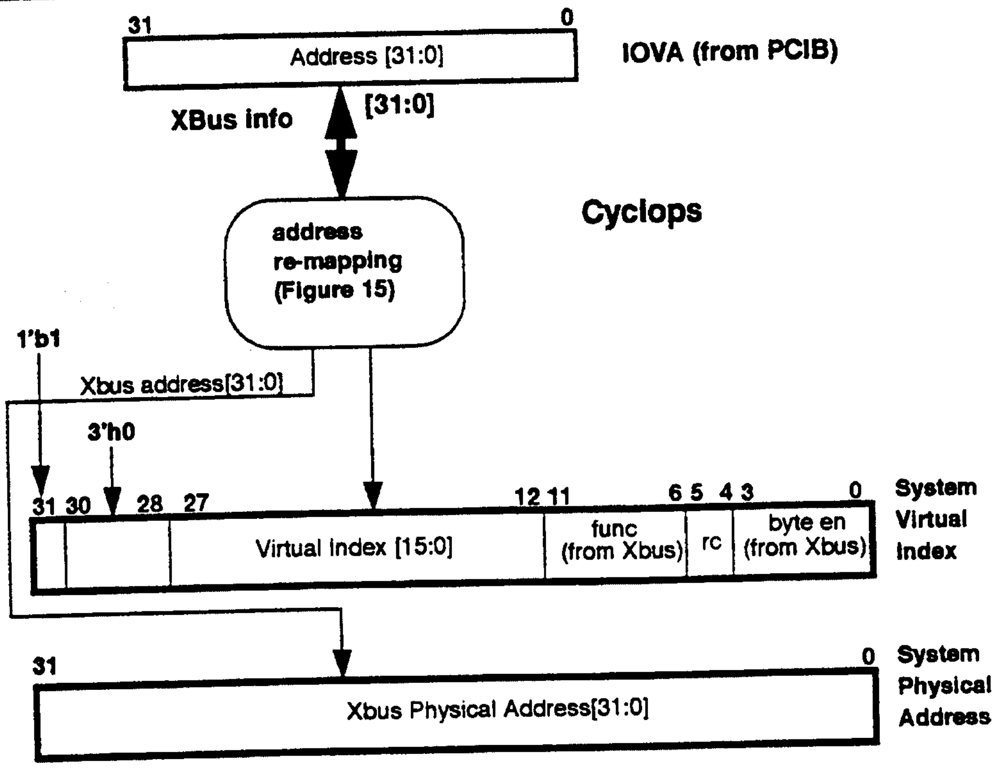


Figure 17 illustrates the address re-mapping used when 40 bit address mode is selected (Polo only). This is the addressing mode that will be used by Runway systems and is included for upward compatibility purposes. Bit 2 of the ASIC specific configuration register within the Cyclops selects between 32 bit and 40 bit address mode. The mode bit is driven on bit 31 of the virtual address to indicate which scheme is being used.

Figure 17. Polo System Address generation (40 bit address mode)



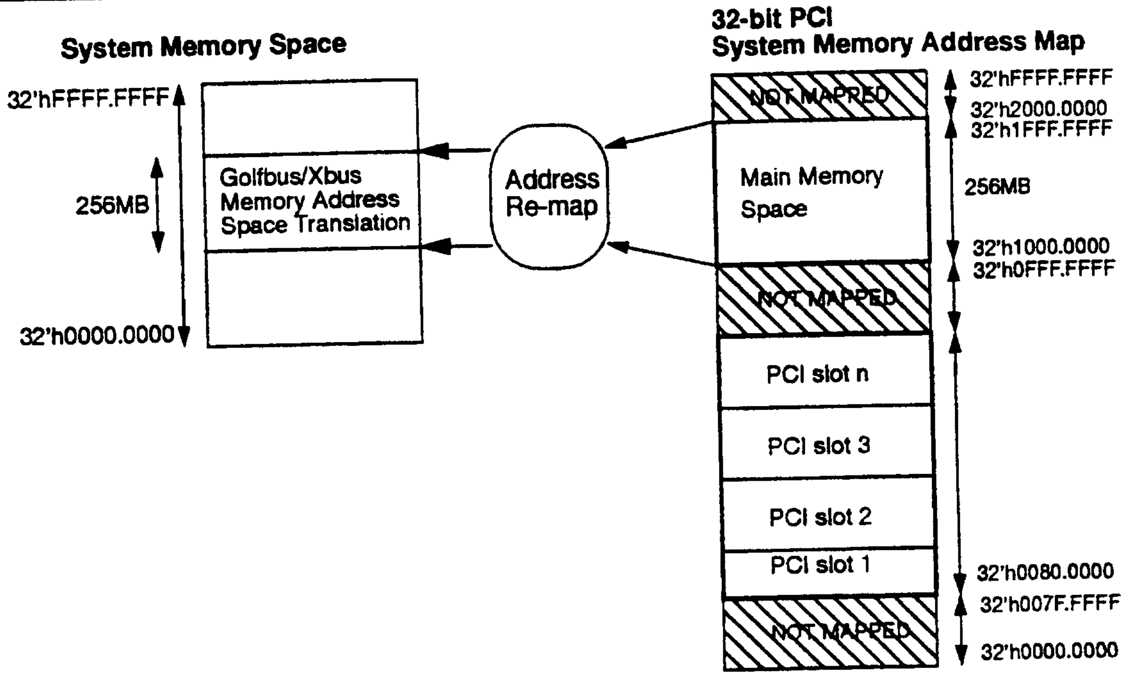
5.3 PCI Memory and I/O Address Mapping

The PCIB/HSC2 has no local memory space. All requests between the Xbus/Golfbus and the PCI adapter are passed through the Gambit/Mirage ASIC. The address mapping between the two buses can be best described according to the direction of the request.

5.3.1 PCI to Xbus/Golfbus I/O Requests

I/O Devices have the same view of the whole 4 GB Address Space as the boards on the Xbus/ Golfbus do. Currently the need for a PCI adapter to generate PCI I/O space accesses is not apparent. The Gambit/Mirage will issue a maintenance attention if a PCI card does an I/O space access. This will happen through the Master Abort mechanism. The Gambit/Mirage will also capture the address on the PCI bus which caused the master abort.

Figure 18. 32-bit PCIB/HSC2 Memory Map



5.3.2 PCI to Xbus Memory Requests

The Gambit or Mirage ASIC only accepts 32-bit PCI memory addresses in a 256 MB window (32'hX000.0000-XFFF.FFFF). The 256 MB block is movable on 256 MB boundaries as defined by the base address registers in the PCI configuration space. The PCI adapter addresses this 256 MB window with the appropriate Chunk Address which is the high 4 bits of the IOVA. Memory accesses created by the adapters that are not within this 256 MB block are terminated through the Master Abort mechanism. There does exist the possibility of accessing a peers memory range on Polo.

The base address registers for Gambit/Mirage and the PCI adapters must not create a conflict in PCI space. All devices and this 256MB block must not overlap in PCI address space. The hardware cannot detect overlaps so the base registers must be set up correctly. The resulting behavior will be unpredictable. The driver must also set up the address translation map using the address table registers described in section 12.6.2 before any access from the PCI adapter can take place.

In a Polo system the I/O virtual address (IOVA) is placed directly on the Xbus. Responding agents on the Xbus are told that the address on the Xbus is an IOVA by the requester's having set a bit in the Xbus TRID field. The responding agents then put the IOVA through their local address translation mechanism to check and generate a standard Xbus address.

In a Jetta/HSC2 system. the IOVA to system address translation is done entirely with the Mirage ASIC. The location of the address translation, whether in the Mirage as in HSC2 or in the responding agent as in Polo, is completely transparent from a software perspective.

Figure 18 shows an example of a PCI memory map with 4 adapters mapped. The Gambit/Mirage PCI base address register is set to 32'h1000.0000 and the limit is set to 32'h1FFF.FFFF. This is the bridges 256 MB window to host memory. There are 1K 16 byte entries in the translation table which are indexed by the I/O Virtual Address generated by the PCI adapter.

5.4 Address Re-mapping

Polo and HSC2 use address re-mapping to protect the system from errors caused by simplexed I/O devices. Note that the I/O Virtual Address or IOVA format used by Polo & HSC2 systems differs from the IOVA format used in BIO and PKIO systems.

In Jetta systems the IOVA is translated into a system address inside the Mirage ASIC on the HSC2 board. The map RAM is also located inside the Mirage ASIC and the Mirage ASIC is self-checking.

In Polo, all system accesses from the I/O ASICs are performed using the IOVA (IO virtual address); these are mapped to the final Xbus style physical/virtual index addresses (CPU addresses) by a map RAM in the Cyclops ASIC. Both IOVA and CPU addresses are passed on the Xbus, with a bit in the trid field distinguishing between them.

The PCI adapters access main memory via the Xbus/Golfbus for command, data and status transfers. The address generated by a PCI adapter originates from a non-self-checking source. It has to be checked to protect self-checking (safe) memories.

5.4.1 System Address Generation

There is an I/O Address Table allocated in internal memory. This memory resides within the Cyclops ASIC for Polo systems and on the Mirage ASIC for HSC2 systems. In the table there is an entry for each block allocated for I/O in system memory. An entry does not assume any addressing method beyond the basic 16KB block size. It contains the starting virtual index, the starting physical address, and the 12 bits of the ending physical address of the block. The next field is the PCI slot number. This field is used to verify that only the expected slot can access a particular IOVA entry. Figure 19 illustrates the encoding of this field. it is checked against the encoded slot information in the TRID generated by the Gambit.

Figure 19. PCI Slot [3:0] IOVA Map Field Definition - Polo only

3	2	1	0
Xbus slot (slot 2 = 2, slot 3 =1) equal to Xbus TRID[0]	**Side D side = 0 C side = 1 Xbus Trid[4]**	**PCI Slot generating Transaction Trid[6:5] slot 0 - 00 slot 1 - 01 slot 2 - 10 slot 3 - 11**	
PCI Bus number			

Note that the HSC2 system only uses the PCI Slot [1:0] bits. PCI Slot [3:2] bits are ignored and are not saved in the Map RAM located inside the Mirage ASIC.

There are another four bits allocated for board specific options and a final bit to indicate validity of the entry. Figure 20 shows an entry example using the Xbus/Golfbus address naming conventions. Each implementation will drop unused bits for this table in hardware to reduce the RAM required.

Figure 20. I/O Address Table Entry/ Block

31							0
Block Starting Physical Address = Physical Cache Tag (31:12), Line Offset (11:2)							
Block Virtual Index (15:0)		Reserved	Block Ending Address (11:2)				
Reserved (2 bits)	PCI Slot[3:0]	Reserved	IC	SW	LK	PR	VA

VA = Valid Entry

OPTION BITS - see section 5.4.4
PR = Data Pre-read
LK = Lock Cycle
SW = Swap 16 or 32 bit Endian
IC = Incoherent Memory Access

This table will be indeterminate at power on. Software should walk through the table and invalidate each entry. Software then must write a valid entry for each device index before any of its respective IOVAs can be used. A maintenance attention will be generated if an entry is used without the valid bit set.

This table is dynamically managed by the I/O driver on a per command basis. The driver has to set up the I/O address table entry for each block before an I/O command that requires memory access can be initiated. The data block can be any size up to 16 KB. Upon command completion the I/O address table entry has to be invalidated by the driver.

There is a map table entry for each 4K page (there can be up to 1024 re-mappable 4K pages). The page bits of the IOVA, bits 12 and 13 are used to address the map RAM. If the programmer wants to allocate a page in PCI memory space larger than 4K with one device index there must be a map entry for each 4K page. This scheme allows up to 4 contiguous 4K pages in PCI memory space without forcing them to be contiguous in PA memory space.

The driver software gives the I/O Virtual Address (IOVA) to a PCI adapter with every command. The IOVA is the combination of the device index to the I/O address table entry, the page within the device index and the starting Page Line Offsets[11:0] into the block. Figure 21 shows the IOVA for a 32-bit PCI adapter card. It allows for a maximum of 1024 Entries in the I/O Address Table. The next two sections describe the step-by-step process used to create the system address.

Figure 21. 32-bit PCI adapter I/O Virtual Address

31 28	27 20	19 16	15 14	13 12	11 0
Reserved	Device Index	Checksum	2'b00	Page	Line Offset[11:0]
	max. 256 indexes			**4 pages**	**4KB blocks**

5.4.1.1 Polo System Address Generation

The Gambit will use the IOVA as is in an Xbus request cycle. The Cyclops in turn will take this information and pass it to the address mapping logic where it applies the appropriate algorithm to perform the checking and address mapping.

The I/O device driver goes through the following steps to set up addresses for an I/O block transfer:

- Finds an available slot in the I/O address table.

- Sets up the entry for the block in the I/O address table.
- Sets up the I/O Virtual Address (IOVA) (including the generation of the checksum).

When a PCI adapter initiates a transfer, the Polo system goes through the following steps of address checking and re-mapping:

- Gambit ASIC gets the I/O Virtual Address.
- Gambit checks that the access is to the Xbus memory range. If there is no match the access is halted and the error state is generated. The Gambit does a master abort and causes a maintenance attention pm the Xbus.
- Gambit checks the checksum within the IOVA and will cause a Target Abort if it is not OK. This will also cause a maintenance attention.
- Gambit issues an Xbus cycle using the IOVA.
- Cyclops checks the IOVA against the I/O address table entry for the device index and page. If the valid bit is set and the IOVA's page and line offset checks out against the entry's start and stop addresses, the combination of the entry and the IOVA's Line Offset[11:0] create the system address. The Cyclops then looks at the system address to determine if the cycle was directed to its memory space and behaves accordingly.

5.4.1.2 HSC2 System Address Generation

The Mirage will use the IOVA to perform the address map look-up. The address generated from the look-up will in turn be used to generate the Golfbus address.

The I/O device driver goes through the following steps to set up addresses for an I/O block transfer:

- Finds an available slot in the I/O address table.
- Sets up the entry for the block in the I/O address table.
- Sets up the I/O Virtual Address (IOVA) (including the generation of the checksum).

When a PCI adapter initiates a transfer, the HSC2 system goes through the following steps of address checking and re-mapping:

- Mirage ASIC gets the I/O Virtual Address.
- Mirage checks the checksum within the IOVA and will cause a Target Abort if it is not OK. This will also cause a maintenance interrupt.
- Mirage checks the IOVA against the I/O address table entry for the device index and page. If the valid bit is set and the IOVA's page and line offset checks out against the entry's starting and ending addresses, the combination of the entry and the IOVA's page and line offset[11:0] create the system address. Otherwise the access is halted and the error state is generated.

5.4.2 IO Address Table Entry Invalidate

Upon completion of the I/O, the driver software has to invalidate the corresponding I/O address table entry in the table. The table's entry is invalidated by simply clearing the valid bit. The entry cache's valid bit can be cleared by using the address table registers described in section 12.6.2. If this index matches the entry cache's index, the valid bit is cleared.

5.4.3 Address Error Checking

Three other address checking features are employed in addition to the index/entry validity check as described in the next two sections.

5.4.3.1 IO Device Address Checksum

Even though the IOVA is re-mapped it is still used to index into the I/O address table. This leaves a small possibility of an erroneously generated IOVA indexing into a valid, but incorrect table entry. To reduce this probability there are four bits of checksum allocated in the IOVA to cover the index. This number is generated by the driver software when setting up the I/O Virtual Address. It is checked by the Cyclops when the I/O device presents the IOVA for data access.

Supporting the 16K page size in PCI memory space impacts the fault tolerance. In order to keep the 4-4K pages that make up the 16K page contiguous in PCI memory space, the 2 pages bits of the IOVA bits [13:12] are not checked under the checksum. In the case of an error this could allow a controller to write to the wrong 4K page within the 16K page. **If you want coverage on these two IOVA bits then only make one 4K page valid in the 16K page. Also note that this reduces the number of usable table entries to 256.**

The algorithm to generate checksum (C) on the index (I) is the following:
C[0] = I[0] ^ I[4] ^IOVA[14]
C[1] = I[1] ^ I[5] ^IOVA[15]
C[2] = I[2] ^ I[6]
C[3] = I[3] ^ I[7]

This code will detect all single bit errors and all adjacent double, triple, and quad bit errors. Over all the code detects 94% of all possible errors in the 8 bit index (table 7). Note: that the logic exclusive ORs IOVA bits 14 and 15 into the check bits since they should be zero, this does not change the outcome but makes insures that the addresses don't increment out of the 16K page.

Table 7. 4 Bit Parity On 8 Bits Fault Detection Summary

# of faults	# of possible fault combinations	# of undetected faults	% of undetected faults
1 bit	8	0	0%
2 bit	28	4	14%
3 bit	56	0	0%
4 bit	70	6	9%
5 bit	56	0	0%
6 bit	28	4	14%
7 bit	8	0	0%
8 bit	1	1	100%
total:	255	15	6%

5.4.3.2 Out of Bound Access Check

The ASIC performing the address re-mapping checks to make sure that the physical address [11:2] of the I/O virtual address is larger or equal to the starting and smaller or equal to the ending physical address [11:2] of the table entry on both read and write accesses. Thus checking is done on 32 bit boundaries. This check is automatic. If you don't want checking, set the starting and ending addresses at their limits.

5.4.4 Control Bits

There are five control bits in the address table entry as illustrated in figure 20 and described in the following sections.

5.4.4.1 Valid Entry bit

The valid entry bit in the table entry indicates whether the entry is valid or not. The table should be fully invalidated by initialization software at power on. This involves resetting the valid bit for each of the 1024 entries in the table. When an IOVA is mapped software must write a corresponding entry into the IO map table with the proper system addresses and set the valid bit. If a device uses an IOVA with an invalid entry the Xbus request will be no-ACKed resulting in the Gambit performing a target abort and generating a maintenance attention.

5.4.4.2 Data Pre-read

This bit is ignored on writes.

In a HSC2 system the Mirage ASIC will perform a data pre-read from the host system if this bit is set and the PCI adapter is performing a read. The Mirage will begin by reading a cache line for the current request and then continue to read successive cache lines until the pre-read is completed. The data pre-read will continue until the pre-read limit in the Mirage is full, the block ending address is reached, or a cycle is issued from the PCI adapter that is not to the block being pre-read. In the event of a non sequential address the pre-read data will be flushed. If this bit is not set then the Mirage will only read the 32 bit memory word requested by the PCI adapter and nothing else until an additional request is made. We may make use of the PCI commands read and read-multiple.

In Polo, all memory accesses through the PCIB initiated by a PCI adapter are issued as 32 bit reads on the Xbus. A full cache line is returned if pre-read is on, lock is off, the starting address is on a cache line boundary, and the ending address of the cache line is less than the block ending address (if the starting address of the access is out of bounds then the access is illegal and not responded to). If all of these conditions are not met on a legal access then the Cyclops will only return 32 bits of data. The Polo system will only use this bit as a qualification for returning either 32 bits or a cache line on a read. The Gambit ASIC will accept either a word or a full cache line back in response to a cache line read request.

5.4.4.2.1 Coherency Restrictions

software note:

Improper use of the pre-read option can result in inconsistent data in the system. It is software's responsibility to guarantee data coherency by insuring that the PCI adaptor never receives "stale" data from a pre-read buffer.

On an HSC2 system, pre-read data sits in the Mirage until one of the following conditions are met:

- The data has been completely read by the PCI adaptor
- The host updates the Map RAM and the map table entry matches the Map cache entry being used by the Pre-read state machine.

- The host explicitly flushes out the pre-read buffer by writing to flush pre-read data bit in the Test Control Register.

Note that PCI adaptor writes while pre-read operations are taking place do not cause data to be flushed from the Mirage even if the write transaction is to the same page as the data pre-read.

5.4.4.3 Lock Cycle

This bit is ignored on writes.

The data pre-read bit is ignored and should be set to zero if the lock cycle bit is set to one.

In a HSC2 system any read access to the block will go out onto the Golfbus with a lock cycle function code (a load/clear operation) if this bit is set. The data pre-read bit is ignored and should be set to zero if the lock cycle bit is set to one.

In a Polo system the I/O address map RAM resides on the other side of the Xbus so the cycle will not have a lock function code when it is on the Xbus. However if the I/O address map look-up reveals that the lock cycle option bit is set then the Cyclops will turn the cycle from a cache line read into a load/clear operation (semaphore lock). The Gambit ASIC will accept either a full cache line or a 32 bit word back on a read access so the Cyclops will only send back the 32 bit word provided by the load/clear. The data pre-read bit is ignored and should be set to zero if the lock cycle bit is set to one.

Note: The swap bit must be turned on when using the lock bit to avoid the possibility of inconsistent swapping between two boards in a duplexed pair.

5.4.4.4 Swap 32 bit Endian

When this bit is set to a one the data associated with this IOVA address will be big/little endian byte swapped. If this bit is set to a zero then there is no byte swapping is performed. Refer to section 5.6 for a detailed description of byte swapping and when this bit should be set.

5.4.4.5 Incoherent Memory Access

This bit allows software the option of generating incoherent (don't snoop, don't set remote tag) as opposed to coherent (snoop, but don't set remote tag) Xbus/Golfbus memory read and write transactions. Refer to the section titled Remote/Coherent bits in the Xbus/Golfbus specification for more details.

- 0 = coherent
- 1 = incoherent

In a HSC2 system, if this bit is set to a one then a read or write operation will go out on the bus with the remote/coherent bits set to indicate an incoherent operation. If this option bit is set to zero then the remote/coherent bits indicate a coherent cycle.

In a Polo system the snoop results will be ignored if this bit is set to a one. All reads and writes will be issued from the PCIB with the remote/coherent bits indicating a coherent cycle. The remote/ coherent bits sent across the Ibus will reflect the state of this bit.

5.5 Software Map Management

The four address table registers that control reading and writing of the I/O address map RAM (see section 12.6.2) will require the use of locks to guarantee exclusive access. There is only one set of these registers accessible from 16 different addresses so a single lock should be used to control them.

5.6 Byte Alignment

The Gambit/Mirage is the interface from the big-endian byte order Xbus/Golfbus to the little-endian byte order PCI Bus. Significant endian issues exist within the PCI Bus as data structures (including SCRIPTs code) are required to be in little-endian format while disk and tape data are required to be in big-endian format (Stratus currently writes to disks (K105/K121) and tapes (K116) in big-endian format).

Many data structures (including 82596 SCB's) intermix 32-bit entities with 16-bit and 8-bit entities which causes further byte ordering conversion issues.

The PCIB/HSC2 system does not attempt to resolve these endian issues, rather it supports a method of moving data into and out of the PCI adapter with the byte swapping under software control. This is configurable on a page by page basis through the option bits in the address map (refer to section 5.4.4.4). It is the responsibility of software to resolve any remaining endian issues.

The Gambit/Mirage will always perform a byte swap on data going from the IOBus/Xbus to the PCI bus. When going from the PCI adapter to the Xbus the byte swapping is software programmable via an option bit in the system IO address table re-mapping entry. Figure 22 represents the byte swapping performed for a little/big endian byte swap. A byte entering a specific byte lane (i.e. byte 0; big-endian) on one side (Xbus or PCI bus) will be sent back out on the same byte lane (byte 0; little-endian) on the other side. Thus when a 32 bit word is byte swapped its format changes from big to little endian or vice-versa so that the data is identical from a software perspective when viewed from either the little endian (PCI) or big endian (Xbus) side of the system. This is the default configuration and the one which should be used when moving data from/to disk and tape devices. Programmable byte swapping is not supported for requests coming in from the Xbus/ Golfbus to the PCI bus - data is always byte swapped so that it appears the same in either format to software.

Figure 22. Little/Big Endian Byte Swap

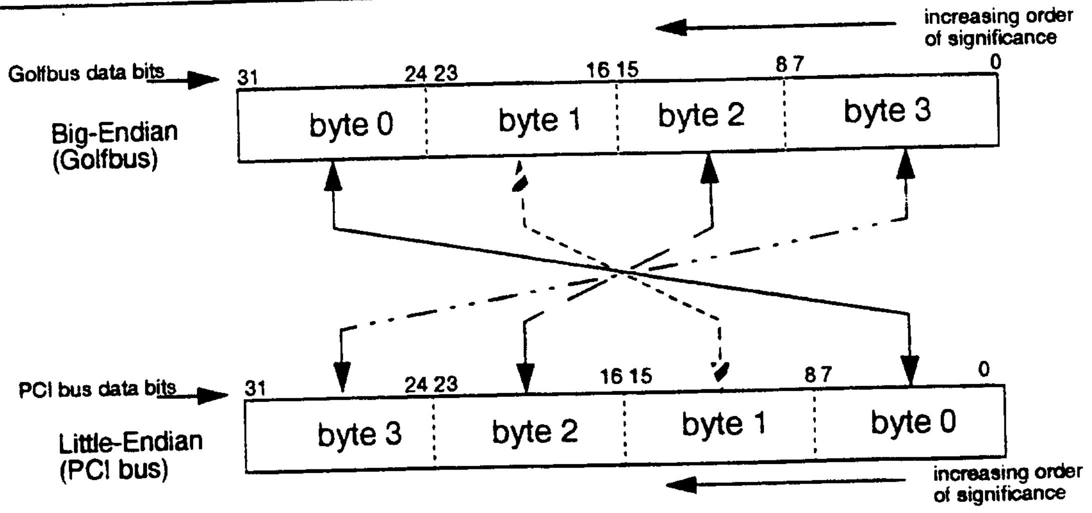
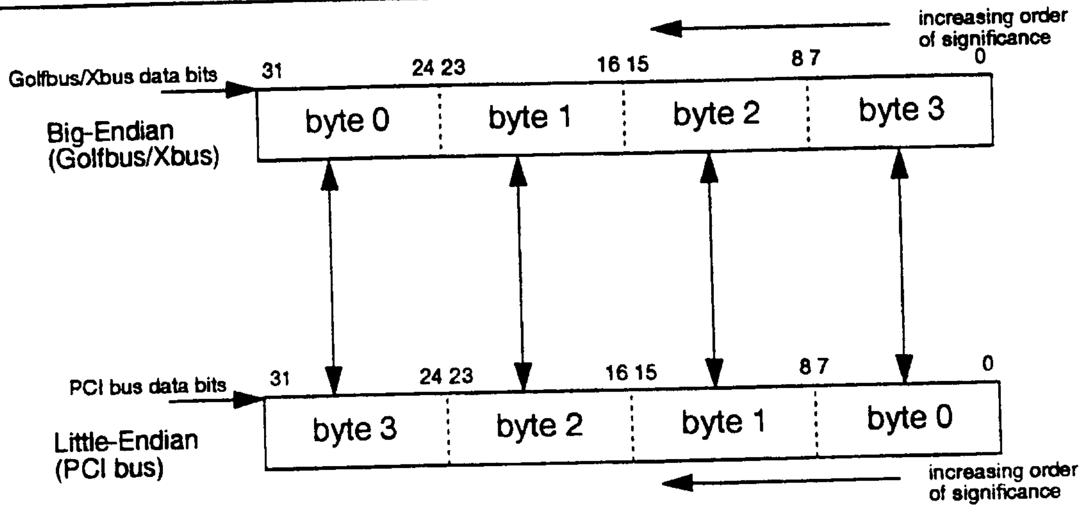


Figure 23 represents the byte swapping performed when the option bit in the address field is configured for no endian swapping. A byte on bits 7:0 traveling from the PCI Bus to the Xbus/Golfbus will remain on bits 7:0. In this mode an integer will hold the same value whether it is in PCI memory or in system memory.

Figure 23. No Little/Big Endian Byte Swap



The following table illustrates the byte written or read based on the system bus byte enables. Note that the big endian point of view is the standard system bus point of view and the one that should be used when referencing the byte enables.

Table 8. Big Endian to Little Endian Byte Swap

Xbus/ Golfbus byte enables	big endian data (Xbus/Golfbus)				little endian data (PCI bus)			
	bits 31....24	bits 23....16	bits 15......8	bits 7........0	bits 31....24	bits 23....16	bits 15......8	bits 7........0
8 bit operations								
1000	0	1	2	3	NC[a]	NC	NC	0
0100	0	1	2	3	NC	NC	1	NC
0010	0	1	2	3	NC	2	NC	NC
0001	0	1	2	3	3	NC	-NC	NC
16 bit operations								
1100	0	1	2	3	NC	NC	1	0
0110	0	1	2	3	NC	2	1	NC
0011	0	1	2	3	3	0	NC	NC
24 bit operation								
1110	0	1	2	3	NC	2	1	0
0111	0	1	2	3	3	2	1	NC
32 bit operations								
1111	0	1	2	3	3	2	1	0

a. NC = no change. writes - this byte holds the previous value; on reads this data is not valid.

Table 9 illustrates how various HSC2/PCIB data cycles are affected by byte swapping.

Table 9. Big/Little Endian Byte Swapping Based on Access Type

access type	swapping performed
MIO/SAM compatible Xbus/ Golfbus registers	no swapping, registers accessible from Golfbus/Xbus only and stored in big-endian format
PCI adapter and bridge configuration space registers	always swapped when read/written, only accessible from Golfbus/Xbus.
Host access of PCI adapter	always swapped when read/written
PCI Access of Host Memory	Swapped (software controlled currently on a page basis, may change on a controller basis)

6. Interrupts

Interrupts generated by cougar, ReCC, and Sable remain unchanged and are well specified in the appropriate specs. This section defines the interrupt structures associated with the PCIB and HSC subsystem.

6.1 PCIB/HSC2 Interrupts

6.1.1 Introduction

The PCI bus specifies the mechanism through which PCI devices can interrupt the host. The PCI interrupt mechanism defines 4 level sensitive interrupts, (INTA, INTB, INTC, and INTD). Three of these interrupts (INTB, INTC, and INTD) can only be used by multi-functional devices. Non-multifunctional devices must use INTA. The interrupt output structures are defined to be open drain signals with no restrictions on grouping together or interrupts. The implementation only supports the one INTA# interrupt (actually a combination of all four lines tied together in hardware). This saves 3 pins and only marginally complicates software for multifunctional device drivers.

The HSC2/Polo interrupt scheme is based around the method that HP uses to implement interrupts on the PA-RISC chip. The HP scheme is a transaction based interrupt architecture. Interrupts are signalled by setting a bit in the EIR (External Interrupt Register) inside the PA chip. In HSC2/Polo, interrupting devices may select to interrupt a specific processor or all processors by writing to either one or all EIR registers.

6.1.2 Implementation goals

The specific goals of the PCIB/HSC2 interrupt scheme are:

- **Directed Interrupt Capabilities**

  It should be possible to select the target host on a per-PCI adapter basis. This allows for selection of either a particular CPU or all CPUs on an individual SAM basis.
- **Minimal host overhead**

  The host should not have to poll SAM modules in order to determine the source of the interrupt.
- **simultaneous interrupt support**

  Since multiple SAMs may share the same EIR bit, any implementation must be able to support simultaneous interrupts. Schemes that depend on only one interrupt being presented until that interrupt is serviced are not acceptable.
- **Simple**

  Complexity in either hardware or software is undesirable.

6.1.3 Hardware Implementation

In Polo/HSC2, there are some severe system limitations on how interrupts are managed within the EIR register of the PA chips. Each PA can have 32 interrupts, but of these only a small number can be allocated for a given slot. Without some kind of additional hardware support, it would be necessary for the interrupt handler to poll all PCI adapters on the interrupt. This would not make for very optimal performance.

For that reason, the hardware must implement more than a write to an EIR register to perform an interrupt. In order to prevent polling of all the PCI adapters, each SAM writes an interrupt into the EIR register and to an OS specified byte (the Host interrupt table) in cacheable memory. The OS can specify any location in memory for this byte. This allows the OS a great deal of flexibility. The OS can then scan the table of interrupts at memory speeds to determine which PCI adapter interrupted it. In the suggested software implementation, some possible architectures for this table are discussed.

In order to implement this scheme, the hardware makes use of a series of registers, operations, and memory tables. The software team should evaluate this proposal to ensure that it is both necessary and sufficient for their needs. Six registers have been implemented on the SAM for hardware support of interrupts. See the corresponding register descriptions in section 12.6.4 on page 129 for a bit-by-bit description.

SAM Interrupt Source Register[0]

This read only register reflects the state of the interrupt pins on the PCI bus. The signal is high true and used to trigger interrupts. This bits reflects the synchronized value of the interrupt signals on the PCI bus.

SAM Interrupt Mask Register[0]

This register is used to enable and disable individual interrupts in the interrupt source register. Asserting a bit in this register enables interrupts in the interrupt source register to cause an interrupt event.

SAM Host Interrupt Table Pointer[47:0]

This 48 bit address points to a location in memory that will be used as a table entry to identify the interrupt source.

SAM Host Interrupt Address Pointer[31:2]

This 32 bit I/O address identifies an EIR register address. The interrupt bit register value will be written into this register.

SAM Host Interrupt Bit Register[5:0]

Bits [4:0] of this register point to the bit to be set in the host interrupt register, EIR and bits 5:0 are written to the interrupt table pointer address. Bit 5 is always a one and functions as a valid bit for the table entry.

Interrupts are generated by either of two conditions:

Condition 1: If there is a zero to one transition on the interrupt bit, and the interrupt mask bit is set to one, an interrupt action will be triggered.

Condition 2: If there is a zero to one transition on the interrupt mask bit, and the interrupt bit is set, an interrupt action will be triggered.

The hardware performs an interrupt operation in the following manner:

Step 1: A byte write is performed to the address indicated by the SAM Host Interrupt Table Pointer. This write is performed as a cacheable memory write, so the software must use a valid memory address for this location, not an I/O address. The data pattern for the write is taken from the least significant byte of the SAM Host interrupt bit register. No validity checks are performed on the write.

Step 2: A 32 bit I/O write is performed to the address indicated by the SAM host interrupt register pointer. The data pattern written is all zeros except for the bit indicated by the SAM host interrupt bit register. It is intended that the software set the address of this write to point to either an individual Cougar EIR register or to all CPUs' Cougar EIR register.

No further interrupts will be transmitted to the host by this SAM until one of the 2 conditions listed above is met.

These two writes are not atomic. Software running on the PA can make no assumptions concerning timing of these two writes. Furthermore, it is possible for two SAMs to perform these operations at the same time. When this occurs, any mixing of write operations is possible between the SAMs. If there is a common interrupt bit shared by the two SAMs, software must ensure that it correctly decodes all of the interrupts. However, the writes are strong ordered on a per SAM basis, that is the two writes from a particular SAM are guaranteed to be seen in the correct order.

For these reasons, it is suggested that software follow the algorithm below for decoding and handling interrupts.

6.1.4 Suggested Software Implementation

As part of the PCI interrupt handling mechanism, the OS must manage the host interrupt table which consists of a one byte addressable piece of information per PCI adapter that is used by the hardware. Since this table is in cacheable memory software must keep the virtual index up to date. This is probably best accomplished through static allocation of the table. Since the SAM register that contains the pointer to this byte is justified down to the byte, the software has a great deal of flexibility in how they organize the host interrupt table. While there are a number of possible organizations, the suggested implementation is to use one host interrupt table per system. This table should be organized as 64 contiguous bytes of PCI interrupt information. This organization allows for the fastest possible scan of the interrupt table. Optimally, all interrupts should be directed to one of the CPUs local to the host interrupt table. That CPU can perform the steps below and then delegate the servicing of the interrupt request to the appropriate CPU.

The order in which various operations associated with the interrupt handling are performed is also important. Deviations from the suggested implementation may result in either extraneous null interrupts or lost interrupts. The basic steps are outlined below:

Step 1: interrupt is received

Step 2: internal interrupts are masked.

Step 3: the appropriate bit in the EIR register is cleared.

Step 4: The host interrupt table is harvested for any pending interrupts.

Step 5: At this point, or later in the sequence, internal interrupts can be unmasked.

Step 6: Gambit or Mirage interrupt mask register is set to 0 (disabling Gambit or Mirage interrupts).

Step 7: PCI card specific actions are performed to clear the interrupt.

Step 8: Gambit or Mirage interrupt mask register is set to 1 (enabling Gambit or Mirage interrupts).

Step 9: The interrupt is serviced.

7. Boot

This section of the spec outlines the boot process for booting an OS on either a Polo or HSC2 machine. The boot process consists of the operation of loading an OS image into main memory and beginning normal OS execution. Before describing the boot methodology in detail, the boot alternatives that have been considered will be described.

7.1 Boot Alternatives

PCI boot is a challenging opportunity for both systems. Since FTIO will be offered in an MIO only configuration (i.e. no BIO) and Polo's only configuration is a PCI only configuration. A methodology for booting the OS in a PCI only configuration must be supported. A number of possible approaches for accomplishing this task are outlined below.

7.1.1 Baby BIO Boot

The Baby Bio boot alternative assumes that Stratus commits to designing a BIO replacement with a PCI interface. If Stratus follows that approach, then the Baby BIO would be the only bootable PCI device and all boot functions would be performed by the Baby BIO.

The major trade-off in considering boot alternatives is the balance of hardware effort and software effort. A baby BIO approach to boot represents a trade-off that minimizes software effort at the expense of hardware. In this approach, a PCI card would be designed that mimics that BIO functionality but has a PCI interface instead of a Golfbus interface. Much of the BEHI and BIO software could be recycled, and changes to the boot prom would be minimized. The baby BIO would respond to a boot command in the same fashion that it does today.

A fundamental problem with this approach is that it probably could not be ready in time for initial power on.

7.1.2 Bio Firmware on Raid Controller Boot

This is a variation on the Baby BIO boot. In this alternative, Stratus would buy an i960 based raid controller with 720 SCSI scripts processors and port the Bio firmware to it. In this approach, most/ all of the firmware would need to be re-written, but the CPU Prom and the BEHI could be maintained.

This approach has the advantage of minimizing the hardware and boot prom effort. However, it is not clear if this card exists. Also, the firmware effort would be very large.

7.1.3 CPU Prom Resident PCI Card Boot

In this boot alternative, the CPU PROM would contain all information and algorithms necessary to perform a boot of either OS off of a given PCI card. Given that PCI cards are not very intelligent, this may be a substantial amount of code. The advantage of this approach is that the system would be able to boot off of any PCI device. There are several disadvantages to this approach. Stratus would have to write PA7100 specific code for each boot device in the system. Every time a new bootable device is released, a new boot loader would have to be written and qualified. Also, every time that a new card is released, a new boot prom would have to be released as well.

7.1.4 Expansion ROM Resident PCI Card Boot

A variation to the CPU Prom resident boot is to use a PCI card resident boot. This method makes use of the expansion ROM on each PCI card. PCI cards have an expansion ROM that can be used to store self-test, boot, and other code. Stratus could write PA7100 boot strapping code for each PCI card that is supported that contains the required boot code. The CPU prom would only need to know how to perform a generic ROM copy and then execute that code. This approach requires a small modification to the CPU boot PROM, but it would require every new bootable PCI card to have a special boot driver written for it. This work is expected to be substantial.

7.1.5 x86 Emulator PCI Card Boot

A third variation on the boot from PCI card theme is to make use of not only the expansion ROM on the PCI card, but the vendor provided boot code as well. This could be done by supporting an x86 emulator in the CPU PROM. This emulator would then run the PCI BIOS code directly. The advantage to this methodology would be that the installed base of PCI card code is primarily x86 code, and this would allow us to make the maximum use of that code. However, it would take considerable work to support the emulator. Also, hardware changes must be made to support such features as the DOS compatibility hole. Finally, the address mapping scheme may have to be disabled to allow for some address limitations in the PCI code.

7.1.6 PCMCIA Flash PROM Based Boot

A totally different variation on boot is to choose to never boot off of a PCI card. In order to accomplish this boot method, a dedicated Flash PROM card would be used instead. This functionality would be duplicated and located on a PCMCIA bus on each repeater in MIO and the 2 PCIB cards on Polo. The CPU Prom would configure the PCMCIA bus and then initiate the copy of the boot-loader or OS into main memory. This approach has the advantage that the number of boot devices that need to be supported is minimized. The boot process is kept simple. It has the disadvantage that it requires some specialized hardware. Also, some new utilities will have to be written to manage the OS images and the flash prom.

7.1.7 Conclusions

Table 10. summary of pros & cons

	Baby Bio Boot	Bio Firmware RAID Boot	CPU PROM Resident PCI Boot	Expansio n ROM Resident PCI Boot	x86 Emulator PCI Boot	Flash Prom Boot
Hardware effort	strong negative	strong positive	strong positive	strong positive	weak negative	weak negative
initial Prom effort	weak positive	weak positive	strong negative	weak negative	strong negative	weak negative
subsequent prom effort	strong positive	weak positive	strong negative	strong positive	strong positive	strong positive

Table 10. summary of pros & cons

	Baby Bio Boot	Bio Firmware RAID Boot	CPU PROM Resident PCI Boot	Expansion ROM Resident PCI Boot	x86 Emulator PCI Boot	Flash Prom Boot
initial driver/ os effort	weak positive	strong negative	strong negative	strong negative	strong negative	weak negative
subsequent driver/OS effort	strong positive	weak positive	strong negative	strong negative	weak positive	strong positive
cost	strong negative	strong negative	strong positive	strong positive	strong positive	weak negative

The table above lists the potential pros and cons of each solution. The standing proposal is to use the flash prom boot. This decision is based on the belief that the 2 closest alternatives to the flash prom boot are the baby bio and the expansion prom boot. The baby bio minimizes the software effort at the extreme expense of the hardware effort. While the expansion prom boot minimizes the hardware effort at the extreme expense of the software effort. The flash prom alternative seems like a good compromise.

7.1.8 Open Boot

A note on Open Boot...

Open boot is a proposed standard to allow for a processor and O/S independent boot process. It is currently unclear what direction this process is taking. We have currently chosen not to count on open boot being available in the initial Polo/HSC2 time-frame. However, should this change, it may be advantageous to reconsider some of our current directions.

7.2 PCI Boot Process

7.2.1 Find and Configure the Boot SAM

Once the PROM has accomplished its normal initialization and is ready to boot the system, it checks the boot list in the ReCC NVRAM. There should be two SAMs identified in the boot list as primary and secondary bootable devices. The SAMs and PCI buses for the devices need to be configured before they are used. The following steps are followed to do this:

1) Set the HSC2/PCIB on-line by writing to the HSC2/PCIB bus interface state register.

2) Set the SAM on-line by writing to the SAM bus interface state register.

With both the PCIB and SAM on-line, PCI configuration cycles can be run to initialize the PCI bus.

The PCI configuration cycles are accomplished by first writing to the PCI Configuration Address register and then writing/reading to/from the PCI Configuration Data register. For Configuration cycles to the bridge, the enable bit (bit 31) should be enabled, the bus number (bits 23:16) should be 0, the device number (bits 15:11) should be 0, the function number (bits 10:8) should be 0, the register number should be set to the register number for the desired configuration register, bits 1 and 0 should be 0.

WARNING: Configuration addressing is little endian. See the endian description for a discussion of how endian works. The summary is that byte operations work with no intervention, 32 bit operations must be byte swapped!

1) Perform a 32 bit read from configuration address 0. Verify the manufacturer ID, 16 bit word at location 00, is 1107h. Verify the Device ID, 16 bit word at location 02, is 0700 for IAM or 0600 for Gambit.

2) Perform a 16 bit write to location 04 of 16'h127. This write disables fast back-to-back enable, enables SERR, disables ECR0, disable address stepping, enables parity error responses, disables VGA palette snooping, disables memory write and invalidate cycles, enables special cycles, enables bus mastering, enables memory space and enables I/O space.

3) Perform a 16 bit write of location 06, the status register, verify that it is 0.

4) Set the memory base address register to 0 by performing a 32 bit write of 0h to address 10h.

5) Set the disconnect timer to 0 by performing a byte write of 0 to 40h.

6) set the retry abort timer to 8 by performing a byte write of 8 to 41h.

7) set the trdy timer 10 20h by performing a byte write of 20h to 42h.

The bridge is now configured.

7.2.2 Configure PCMCIA Bridge Chip

Once the PCI bridge is configured it is necessary to configure the PCMCIA bridge chip by performing a series of configuration reads and writes to the PCIMCI bridge. For these writes, the enable bit (bit 31) should be enabled, the bus number (bits 23:16) should be 0, the device number (bits 15:11) should be 1, the function number (bits 10:8) should be 0, the register number should be set to the register number for the desired configuration register, bits 1 and 0 should be 0. The PCMCIA bridge chip (PPEC) is a multifunctional device. However Polo only uses the first function.

1) Perform a 16 bit read of register 00h. Verify that the Vendor ID is 8086h, intel.

2) Perform a 16 bit read of register 02h. Verify that the device ID is 1221h, ppec chip.

3) Perform a 16 bit write of 14bh to location 04h. This write disables fast back-to-back enable, enables SERR, disables ECR0, disable address stepping, enables parity error responses, disables VGA palette snooping, disables memory write and invalidate cycles, disables special cycles, disables bus mastering, enables memory space and enables I/O space.

4) Perform a 16 bit write of c800h to location 06h.

5) Initialize the PCI-PCMCIA bridge base address register. This register determines the starting address of the PCMCIA index/data socket configuration registers in the PCI I/O space. It is recommended that 0 is used is for the base address. This is accomplished by writing a 1 to location 10h. After boot this base address can be deconfigured or moved out of the way for normal operation.

6) set the PCI Configuration control register. Perform a byte write of 39h to location 40h. This

enables enhanced PCMCIA timing mode, enables PCMCIA read-prefetching buffers, enables write posting buffers, set the PCI clock speed to 25Mhz.

7) set the PCI interrupt routing register. Perform a byte of 3h to location 50h. This routes both of the PCMCIA interrupts to the interrupt line and disables interrupts from all other (unused PCMCIA) sockets.

The PCI side of the PPEC chip should now be configured.

7.2.3 Configure PCMCIA Bus

The PPEC chip is run in mode0. This operational mode provides two fully buffered PCMCIA sockets. Polo and HSC2 only populates socket A so all accesses should be performed to socket A register and windows. Like the PCI side of the PPEC, the PCMCIA side requires some initial set up. The set up is accomplished through performing I/O writes to the PCI bus. The PCMCIA chip uses an index port and a data port to access its registers. The index port is a byte located at the base address loaded into the base address register in step 5 above. The data port is a byte located at base address + 1. If the algorithm was followed as outlined in step 5, then the base address is at 0 in PCI I/O space. If the algorithm was not followed, then you must do your own translation. Writes can be performed to the PPEC chips by performing a 16 bit write with bits 7:0 corresponding to the desired register and bits 15:8 corresponding to the data. Reads may be performed by performing a byte write to the index port and a byte read to the data port.

WARNING: this register and all descriptions are little endian. Byte swapping must be performed if anything other than byte operations are performed.

1) Configure the PCMCIA power. Perform a 16 bit write of 1000h to PCI I/O space 0h. This will disable the output enable for the PCMCIA bus, turn off auto-powering, turn the socket on to 5v, and turn VPP on to 5v.

2) Enable the PCMCIA bus. Perform a 16bit write of 9000h. This write enables the PCMCIA output enables for actual writes to the PCMCIA bus.

3) Set up the I/O window data side. The PCMCIA card has an 8 bit I/O bit data path. This is configured by performing a 16 bit write of 1007h.

4) Setup I/O window0. The PPEC chip contains two I/O space windows. The I/O space is passed directly through the PPEC chip. I/O accesses are used to setup and control component management registers starting at address 4000h. We will setup 4000h through 41ffh as the I/O space for the PCMCIA bus. These may be deconfigured after boot. Set the low byte start address for I/O window 0 by performing a 16 bit write of 4008h. Set the high byte start address for I/O window 0 by performing a 16 bit write of 0009h. Set the low byte stop address by performing a 16 bit write of ff0ah. Set the high byte stop address by performing a 16 bit write of 410bh.

5) Setup Memory Address Mapping Window 0. The PPEC contains 5 memory windows. The memory windows are used for accessing the command register and the flash array itself. For simplicity's sake we will map this address space starting at 0 and running through the full 20MB range. After boot, this address can be deconfigured. The window allows for a 16Mbyte region of PCI address space to be mapped at one time, and for the full 64Mbyte region of PCMCIA space to be mapped. the mapping granularity is a 4K page. We will setup a window of 16Mbytes of PCI space and 16Mbytes of PCMCIA space and then reset the windows when we roll over 16Mbytes. Set the system memory address mapping window 0 start low byte (bits 19:12) by performing a 16 bit write of 0010h. Set the system memory address mapping window 0 start high byte (bits 23:20) by performing a 16 bit write of 8011h. Bit 7 of this write sets the data path to 16 bits, and bits 3:0

set address bits 23:20 to 0. Set the system memory address mapping window 0 stop low byte (bits 19:12) by performing a 16 bit write of ff12h. Set the system memory address mapping window 0 stop high byte (bits 23:20) by performing a 16 bit write of 6f13h. Bits 7:5 set the timing of the card to 150ns. Bits 3:0 set the upper address to f. Setup the PCMCIA card offset address. This will initially be 0. We will bump this up when we roll over the 16Mbyte region. Set the card memory offset address 0 low byte register by performing a 16 bit write of 0014h. Set the card memory offset address 0 high byte register by performing a 16 bit write 0015h.

6) Enable the address mapping windows. Perform a 16 bit write of 4106h. This enables I/O window 0 and memory address window 0.

With these settings:

I/O addresses 0 and 1 map to the PPEC chip.

I/O addresses 4000h - 41ffh map to PCMCIA I/O space

Memory addresses 00000000h - 00ffffffh map to PCMCIA space.

7.2.4 Configure PCMCIA Flash Card.

The PCMCIA flash card is accessible as a slave on the PCMCIA bus. The card is organized into an attribute space accessed through I/O space and a memory space accessed through memory space. The first step is to configure the I/O space to the proper configuration.

1) Configure the voltage register by performing a byte write of 01h to 4000h. This sets VCC to 5v and VPP to undriven.

2) turn on VPP generation by performing a byte write of 01h to register 410ch. This turns on the internal VPP generator in the PCMCIA chip.

The card is now ready to be read for boot. This operation is started by performing a command write. The command write sets the mode of the flash array. Perform a 16 bit write of hex ffff to memory space 0h.

The card is now in read mode and memory reads will be processed directly to the array. The array may be read in 32 bit chunks.

Notes:

This description did not describe setting up the Gambit/Mirage offset window registers to the PCI bus. These must be setup.

This description did not explicitly explain the reprogramming of the card offset register when the prom rolls over the 16Mbyte PCI page.

Writing the PCOM is not described, and will be covered in a later revision of the spec.

8. PCI Configuration

The Polo and HSC2 PCI bridge and adapters have to be configured. This will need to be done at boot time. There are also configuration issues that need to be addresses when new devices are plugged in, pulled out or when they fail.

PCI is defined to allow the PC to do an auto configuration. The PC BIOS discovers which devices are on the PCI bus and configures the bridges and adapters. (There is some user intervention required in some cases where it cannot be completely automatic - see PCI ECR Status document dated 5/11/94.) When a configuration is complete the information is stored in non-volatile memory and is used in subsequent boots as long as the same devices are present.

We need to support a configuration similar to the auto configuration done in PCs. Cards may have swapped since the last boot. We also need to support hot-plug reconfiguration due to board pseudo-hot plugs in the Polo and HSC2 systems.

8.1 PCI Configuration Register Space

Each PCI agent, including the host bridge within the Gambit/Mirage ASIC, has a 256-byte configuration space. The first 64 bytes of this configuration space comprises the required PCI configuration registers. These are a predefined header containing ID, status, command and other information. The optional configuration space from 040h-FFh is available for use by bridges. Gambit/Mirage will not use this area. All PCI adapters/agents must respond to the entire 256 byte configuration space. All writes to undefined areas must have no effect and all reads to undefined areas must return zeros.

The Gambit/Mirage ASIC implements accesses to the configuration space using a method similar to the PC/AT. Accesses to the 32 bit I/O registers, config_addr and the config_data, defined in the Gambit/Mirage I/O register space are used to generate PCI cycles with the Configuration Read or Write function code. First an address is set up by writing to config_addr then a read or write is done to config_data. See configuration section 8.5, PCI Address formation, for specifics on how the config_addr decodes to the individual PCI devices on the PCI bus.

8.1.1 Gambit PCI Configuration Space

The PCI configuration space of Gambit/Mirage is not available to other agents on the PCI bus, but only to Xbus generated requests. Accesses to these registers are made through the 'config_addr' and 'config_data' registers in Gambit IO Register Space. The Gambit PCI Configuration Space is available as 'Device Number 0' for type '00' accesses.

Table 11. Gambit/Mirage PCI Configuration Space Header

Offset	Register Name	R/W	Reset Value	Description
00h	Vendor ID	R	1107h	Unique manufacturer ID - Stratus PCI ID
02h	Device ID	R	0600h	Identifies the device as per vendor

Table 11. Gambit/Mirage PCI Configuration Space Header

Offset	Register Name	R/W	Reset Value	Description
04h	Command			
[15:10]		R	00h	Reserved
[9]		R	0	Fast back-to-back Enable
[8]		R/W	0	Global SERR# Enable. Discrete enables in ECR0.
[7]		R	0	Wait cycle control - Address/Data Stepping
[6]		R/W	0	Parity Error Response Enable
[5]		R	0	VGA Palette Snoop - Not used.
[4]		R	0	Memory Write and Invalidate - Not used.
[3]		R/W	0	Special Cycles - Special Cycles.
[2]		R	1	Bus Master Enable
[1]		R/W	0	Memory Space Enable
[0]		R	0	I/O Space Enable
06h	Status			
[15]		R/W[1]	0	Detected Parity Error.
[14]		R/W[1]	0	Signaled System Error.
[13]		R/W[1]	0	Received Master-Abort.
[12]		R/W[1]	0	Received Target-Abort.
[11]		R/W[1]	0	Signaled Target-Abort.
[10-9]		R	01h	DEVSEL# timing: 00=fast, 01=medium, 10=slow.
[8]		R/W[1]	0	Data Parity Error Reported.
[7]		R	0	Fast Back-to-Back Capable.
[6:0]		R	0	Reserved
08h	Revision ID	R	0000h	IAM Rev number.
09h	Class Code	R	06h 0000h	Bridge Device Host Bridge (Gambit/Mirage)
0Ch	Cache Line Size	R	00h	Gambit/Mirage not supporting cache protocol.
0Dh	Latency Timer	R	00h	No Latency Timer.
0Eh	Header Type	R	00h	Header layout as per PCI spec.
0Fh	BIST	R/W		Built-In Self-Test
10h-27h	Base Address	R/W		Refer to section 6.2.5 PCI Specification Rev 2.0
28h-2Fh	Reserved			
30h	Expansion ROM Base Address	R/W		Refer to section 6.2.5 PCI Specification Rev 2.0
34h-3Bh	Reserved			
3Ch	Interrupt Line	R	00h	Not implemented

Table 11. Gambit/Mirage PCI Configuration Space Header

Offset	Register Name	R/W	Reset Value	Description
3Dh	Interrupt Pin	R	4h	Not implemented
3Eh	MIN_GNT	R	0	Not implemented
3Fh	MAX_LAT	R	0	Not implemented

8.1.2 PCI Adapter Configuration Space

The PCI configuration space of the mated PCI adapter is available to the Gambit/Mirage ASIC across the PCI bus. Accesses to these registers are made through the 'config_addr' and 'config_data' registers in Gambit/Mirage IO Register Space. The PCI Adapter Configuration Space is available as 'Device Number 1' for type '00' accesses. For adapter specific details on the Configuration Space Header, refer to the PCI Specification Chapter 6 and the appropriate Adapter Specification.

Table 12. PCI Adapter Configuration Space Header

Offset	Register Name	Description
00h	Vendor ID	Unique manufacturer ID
02h	Device ID	Identifies the device as per vendor
04h	Command	
06h	Status	
08h	Revision ID	Rev number.
09h 0Ah-0Bh	Class Code Sub-Class/Prog I.F.	Class of device Sub Class / Programming Interface
0Ch	Cache Line Size	Not supporting cache protocol.
0Dh	Latency Timer	Refer to section 6.2.4 PCI Specification Rev 2.0
1Eh	Header Type	Header layout as per PCI spec.
0Fh	BIST	
10h-27h	Base Address	Memory and IO Offset Addresses as per PCI spec.
28h-2Fh	Reserved	
30h	Expansion ROM Base Address	Refer to section 6.2.5 PCI Specification Rev 2.0
34h-3Bh	Reserved	
3Ch	Interrupt Line	Identifies interrupt controller input. Refer section 6.2.4 of the PCI Specification

Table 12. PCI Adapter Configuration Space Header

Offset	Register Name	Description
3Dh	Interrupt Pin	Identifies which INT# signal the PCI device asserts
3Eh	MIN_GNT	Suggested Latency timer value
3Fh	MAX_LAT	Suggested value for tuning the PCI bus.

8.1.3 PCI Configuration space header description

These registers are defined in the configuration space for system bus/PCI host bridge and for each PCI adapter in the system.

The *Vendor ID* field is read in each slot to check for card existence. It is a 2 byte field. The PCI spec. states that if the read returns 16'hFFFF this is an indication that the device does not exist. On Polo/HSC2 the Xbus/Golfbus I/O read will be aborted and cleared out of the PING table in Gambit/Mirage. This will result in a PING timeout which will then cause a data return of zeroes with an LPMC to be sent to the Cougar. The OS should search for available cards and not the driver code. Otherwise the vendor ID should be checked for a valid Stratus supported device.

The next field read should be the *Device ID*. Again it should be checked against known devices for the vendor.

The *Revision ID* may have meaning in some cases. We need to see what specific adapters do here and what significance the number holds.

The *Header Type* identifies the layout of bytes 10h through 3Fh and bit 7 indicates if the device is a multi-function device. Type 00 is a normal card and type 01 is a PCI-PCI bridge header format. The other encodings are reserved. If the device is multi-function the all of the function numbers have to be polled for existence. If it is a PCI-PCI bridge there are additional type 01 configuration cycles required to configure the lower level PCI buses

The *Class Code* field is a 6 byte field, the first two bytes indicate the base class (VGA, Mass Storage, Network, Bridge...) The next two bytes are the Sub-class code (SCSI, IDE, Floppy...). The next 2 bytes are the Programming Interface which indicates a register level programming interface for device independent software.

The *Command* register control the major functions of the PCI card. Writing all zeros to this register logically disconnects the card from the PCI bus.

Bit 9 - Fast Back-to-Back Enable - Reset since Gambit/Mirage will not support it.
Bit 8 - SERR# enable - This bit will be turned on.
Bit 7 - Wait cycle control - Reset to zero if writable.
Bit 6 - Parity Error Response - Set to enable parity checking
Bit 5 - VGA Palette snoop - Not supported
Bit 4- Memory Write and Invalidate - Not supported
Bit 3- Special Cycles - Reset to zero to disable. Hardware is capable if Software wants to turn this on.
Bit 2- Bus Master - Set to 1
Bit 1- Memory Space - Set to 1
Bit 0- IO Space - Set to 1 on adapters, Set to 0 on the bridge.

The *Status* register

Bit 15 - Detected Parity Error
This bit indicates that a device detected a parity error. A Maintenance Interrupt is generated from Gambit/Mirage. An adapter is responsible to report a parity error to the system most likely via an interrupt to its driver.

Bit 14 - Signaled System Error
This bit gets set whenever a device asserts SERR# which is a global error signal directed to the Operating System. SERR# indicates an address parity error, data parity errors on a special cycles or other non parity errors. SERR# will cause a Maintenance Interrupt to be generated from Gambit/Mirage.

Bit 13 - Received Master Abort
Gambit/Mirage causes a Maintenance Interrupt.
Adapter - TBD

Bit 12 - Received Target Abort
Gambit/Mirage causes a Maintenance Interrupt.
Adapter - TBD

Bit 11 - Signal Target Abort
Gambit/Mirage causes a Maintenance Interrupt.
Adapter - TBD

Bit [10:9] - DEVSEL timing
00 - fast
01 - medium
10 - slow
These bits are read only and indicate the speed at which a device will decode accesses to its address spaces. It can be used by the subtractive decode agent to speed up selection. Gambit/Mirage does not do subtractive decode so the Master will have to use the Master Abort Mechanism.

Bit 8 - Data Parity Detected
This bit is only set by a bus master when: 1. When the master asserts **PERR#** (on a read transaction) or when a master samples **PERR#** (on a write transaction) and 2. The Parity error response bit is enabled.

Bit 7- Fast Back-to-Back Capable

The *Cache Line Size* register - This register only needs to be implemented on devices that are participating in the caching protocol which we do not support.

The *Latency Timer* - register must be writable on all master devices that can burst more than two data phases. It may be read only on devices that burst two or fewer data phases but must contain the number 16 or less.

The *Built-in Self Test (BIST)* register

Bit 7 - BIST Capable
Bit 6 - Start BIST
Bits [5:4] - Reserved
Bits [3:0] - Completion code: 0 indicates test passed; non zero device specific failure codes.

The *Interrupt Line* register - Use of this register is TBD. In a PC it corresponds to the IRQ level of the 8259.

The *Interrupt Pin* register - This register indicates which interrupt line the device will use. It will contain 0 if no interrupts are used. Polo will tie all interrupts to **INTA#**.

The *MIN_GNT register* - This register specifies how long a burst period the device needs assuming a 33 MHz clock. This value should be the recommended Latency Timer value or 0 if there are no special requirements.

The *MAX_LAT* register - This register contains the time this device can wait for a grant. This number is used to set the latency timers of other devices. There must be some algorithm/ equation but we don't know it.

8.2 Proposed configuration sequence

1. Boot system - See Section 7.
2. Start OS based config-for each PCI slot and sub-function
   a. Device detection - Configure host bridge if it exists
   b. Match device against requirements - then config the device
      i. Run the post code on the devices Expansion ROM
      ii. or Run the cards initialization code stored in the OS files system
      iii. or Restore the previous configuration (unlikely to be choice, discussed later)
   c. Shut down any unknown/unconfigurable devices
   d. Update host bridge configuration with new address information.
   e. Run any diagnostics on device
      i. Expansion ROM based diagnostics
      ii. or OS based diagnostics
   f. Handle special configuration i.e. dual initiated scsi host ids etc.
   g. Set up device drivers

8.3 OS Based full system Configuration

The PCI config_addr and config_data registers are in Gambit/Mirage and there is one register that responds to the address corresponding to 4 SAMS. They map as follows:

MIO	Polo	HSC2
SAM 0	PCIB, Gambit C-side	HSC2, Mirage
SAM 1	PCIB, Gambit C-side	HSC2, Mirage
SAM 2	PCIB, Gambit C-side	HSC2, Mirage
SAM 3	PCIB, Gambit C-side	HSC2, Mirage
SAM 4	PCIB, Gambit D-side	does not exist
SAM 5	PCIB, Gambit D-side	does not exist
SAM 6	PCIB, Gambit D-side	does not exist
SAM 7	PCIB, Gambit D-side	does not exist

NOTE: To control access to these multi-response register the OS needs to lock the group of 4 SAM registers since they are physically one register.

8.4 PCI device detection

All PCI devices adaptors and bridges must implement the first 4 fields in the header, Vendor ID, Device ID, Command and Status. The additional fields are optional but if used must be at the appropriate memory address.

In a PC all possible PCI slots must be polled to determine the configuration. The polling starts at device 0. If a multi function device is detected the all other functions in the slot must be polled. Our system will avoid polling since each SAM will supply a maintenance interrupt when it powers on or after a reset. When the OS is ready to configure the system it will reset all of the SAM slots. Each existing SAM will supply a maintenance interrupt.

Polo and HSC2 have 4 PCI slots per PCI bus. There will be one maintenance interrupt per PCIB/ HSC2. Software must poll each PCIB/HSC2 slot for the existence of a PCI adapter. The PCI adapters are addressed as though they are pseudo SAMs (see Section 8.5). Polo needs to check each slot for multi-function adaptors see section 8.5.1.

8.5 PCI Address formation

Gambit/Mirage will format configuration addresses from config_addr as explained in the PCI rev 2.0 spec.

8.5.1 Configuration cycle generation

The hardware will compare the "Bus Number" from the address to the bridges secondary bus number which in our system is 0. It will generate a type 0 configuration cycle when the "Bus Number" is zero otherwise it will generate type 1 configuration cycle. When a type 1 configuration cycle is created the lower 2 address bits AD[1:0] need to be changed from 2'b00 to 2'b01.

For electrical reasons we require that the address for a configuration cycle be Pre-driven. Since the address lines will be resistively coupled to the IDSEL lines going to the adapter the IDSEL lines will have a low slew rate. The address must be stable **TBD** clocks before **FRAME#**.

8.5.2 Special cycle generation

When the config_addr register is loaded with the Device and Function numbers of all ones and the Register number is zero the bridge is "Primed" for a special cycle. When our config_data register gets written we generate a special cycle if the bus number equals zero. Otherwise we will send it on as a config_write. Lower level PCI-PCI bridges that match the Bus number will then create special cycles on their secondary buses.

Figure 24. Gambit/Mirage Bridge Translation config_addr type 00 -> PCI AD

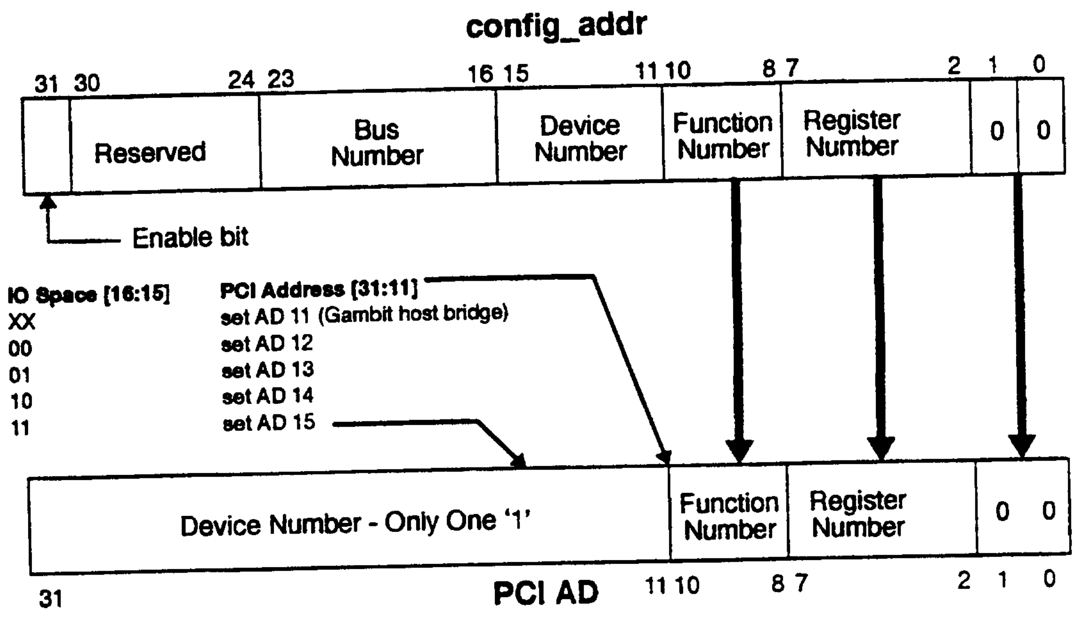
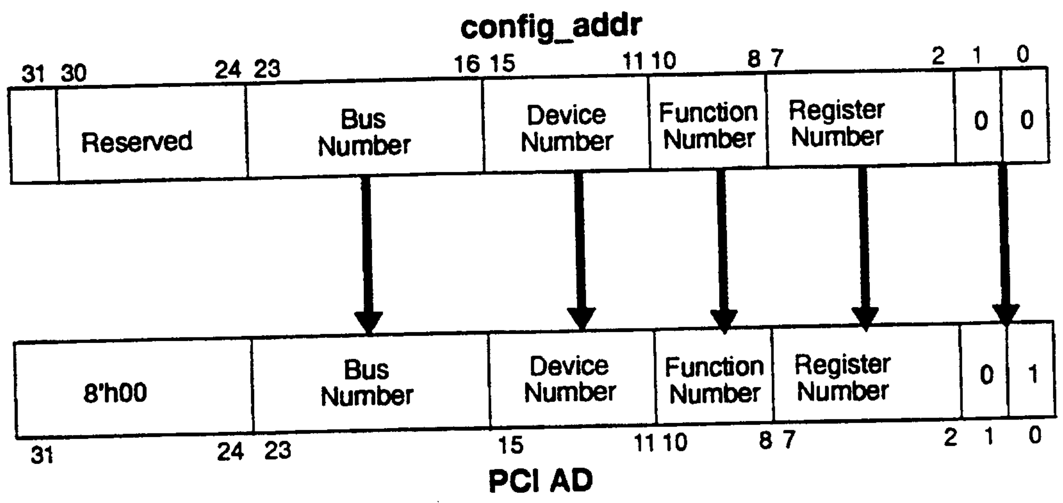


Figure 25. Bridge Translation config_addr type 01 -> PCI AD



8.5.3 Polo/HSC2 Configuration Address Generation

Polo/HSC2 have 4 PCI buses with 4 PCI slots each. The devices will be numbered 1-4 on each PCI bus. For type 00 configuration cycles the Gambit will form PCI address bits [31:11] from the device number in the PCI config_addr register along with bits [16:15] of the Xbus/Golfbus IO Space address from the cycle that accesses config_data to mimic which of the 4 pseudo SAMs is being addressed. If the 'Device Number' equals zero the bridge is accessed if it is non zero device 1 is accessed. Therefore device 1 should always written to config_addr to access an adapter. See example below. The IDSEL[3:0] will be tied to AD[15:12] respectively again see section 3.6.4 of the PCI spec for electrical details. Polo/HSC2 also needs to check each slot for multi-function adaptors see Section 8.1.2.1.

PCI Device Field	IO Space [16:15]	PCI Address [31:11]
0	XX	set AD 11 (Gambit/Mirage host bridge)
!0	00	set AD 12
!0	01	set AD 13
!0	10	set AD 14
!0	11	set AD 15

8.6 Multi-function Adaptors

A PCI adapter can have multiple functions on one card. For example SCSI and enet or multiple SCSI ports. These sub functions are addressed through the function number field in the config_addr register. All cards must use function number 0. If they have multiple functions then bit 7 of the *Header Type* configuration register indicates whether a device is a multi-function device.

If a multi-function device is detected all possible function numbers must be polled for existence. The PCI spec. allows them to occupy any function number after function 0. Each function has its own configuration header and therefore has to be configured into PCI address space as an individual device.

8.7 SAM and PCI Adaptor Configuration

Once the bridge device is detected the bridge can be used for PCI configuration cycles to detect resident PCI devices without any bridge configuration. Software then reads the header information for PCI device 1. The PCI device needs to be mapped into PCI address space. First the software has to determine how much memory and IO address space the device needs. This is done by writing all ones to the base address registers and reading them back. The device will return 0's in all don't care bit positions effectively specifying the address space required.

Note: The typical device will have two base address registers in the header i.e. requiring one memory range and one IO range. The low order bits distinguish between the memory and IO. See PCI local bus spec section 6.2.5.1.

8.7.1 Base Address Registers

The number of high order bit that an adapter implements determines how big its memory space is. For example Gambit/Mirage responds to 256 MB so it implements the 4 high address bits. If software did not know how big the range was it could write all ones to this register and it would read back f000_0000h indicating the size and the fact that it is a memory space base register. See figure 26 and figure 27 for the formats of base registers.

Figure 26. Base Address Register for Memory

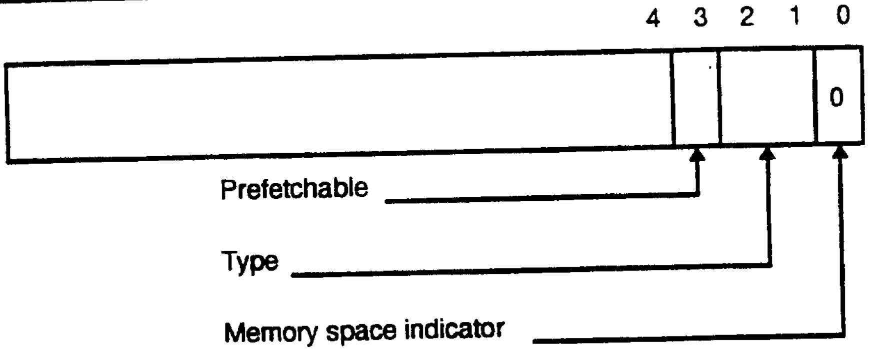
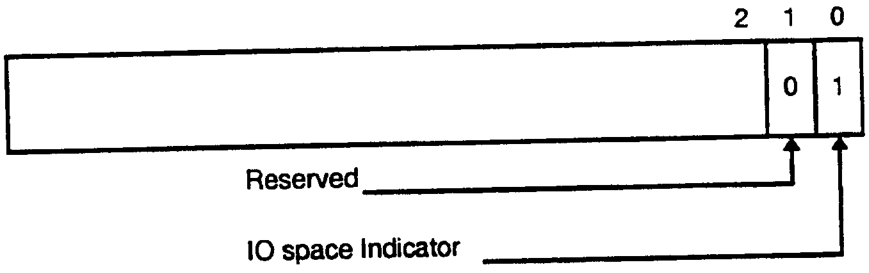


Figure 27. Base Address Register for I/O



8.8 PCI Address Space

Xbus/Golfbus addressing of PCI address space is well defined. Inside each SAMs address space there is a 64KB memory window and a 32 KB IO window from the Xbus/Golfbus. The PCI card's memory may be much larger than these windows and the PCI card needs to mapped into a complete range of PCI address space. Furthermore for Polo we need for these ranges to be non-overlapping from SAM to SAM (remember that there are 4 pseudo SAMs per PCI bus in Polo).

This non-overlapping address space needs to be tracked by the configuration software. There are a number of issues around this PCI address space related to board insertion/removal. When a new board is added to the system its memory and IO ranges have to be mapped so that it does not overlap with any previously mapped devices. When a board is pulled it will leave holes in this PCI address space. This brings up the need to defragment the PCI address range if new devices cannot be mapped into the existing ranges.

(There are other alternative here like a fixed per slot allocation which may not work due to large possible memory and IO ranges. The contiguous range has to work per PCI bus. Four separate ranges can be tracked, one for each Polo PCI bus, or one per bucket if that is easier.)

8.9 PCI Bus Target and Master Aborts

The bus master detects a master abort when no target responds to its request. The master sets bit 13 in the its PCI configuration status register. The Bridge (Gambit/Mirage) also watches the PCI bus for master aborts and sets a bit in its PCI Error register. This condition will cause a maintenance interrupt if it is not masked off in the SAM IO space Configuration register. If the bridge acting as a master detects a master abort it sets bit 13 in the its PCI configuration status register and may cause a maintenance interrupt if it is not masked off in the SAM IO space Configuration register.

A target signals a target abort when it gets an error on a transaction. The target sets bit 11 in its PCI Configuration status register. The master sets bit 12 in its PCI Configuration status register. In Polo and HSC2 the bridge is either the master or target in every transaction. The bridge is aware of the abort and sends a maintenance interrupt again based on the mask bits in the SAM IO space Configuration register.

8.9.1 PCI Bus Fault Detection and Tolerance

See M+D section 9.

8.10 On Line Adds

The PCI configuration will have to be updated when a Jetta/HSC2 or a Polo/PCI card is added to the system. This mechanism is different for HSC2 and Polo.

Although the HSC2 board itself can be inserted, PCI cards themselves cannot as they are connected on the HSC2 board via PMC (PCI Mezzanine Card) Adaptors.

Polo requires the PCI bay to be powered down for a board insertion. When a bay comes back on line after a power down the OS should go through a full configuration sequence.

8.11 On Line Failures

An on-line failure is a condition that breaks the SAM. These include SAM errors and PCI errors. See the error registers in section 12.6.

1. Reset the failed device.
2. Check the failed device.
3. Configure it back into the same place it used to be.

8.12 Board removal

Polo requires the PCI bay to be powered down for a board removal.

Although the HSC2 board itself can be removed, PCI cards themselves cannot as they are connected on the HSC2 board via PMC (PCI Mezzanine Card) Adaptors

8.13 Special Requirements

On reads from bit encoded fields with reserved bits software may not rely on the value in these fields. Software must use the appropriate masks. On writes software must read the register and merge in the reserved bits to the new data to be written then write back the merged word.

We need a mechanism for programming and storing config information for PCI cards with user selectable config. options. For example we need to change the SCSI host adapter id number for the second host on a dual initiated bus. There is additional information on this topic in the PCI ECR Status document dated 5/11/94.

When an expansion prom is decode enabled the software must not access the device through any other base address registers. This is because PCI adapters may save gates and share the address decoders for the expansion ROM with another base register.

See the byte order discussion in section 5.6.

8.13.1 Lock requirements

The page 2 SAM registers (section 12.6.2) need to be locked due to Polo multi registers. See section 5.5 for more information on locking these registers.

9. PCIB Maintenance & Diagnostics

9.1 Overview

The Maintenance and Diagnostics (M&D) subsystem is a major part of the value-added in a Stratus system. It is responsible for managing the configuration of the machine as parts fail and are replaced, and as the machine is upgraded on-line. It handles the reporting and logging of error information, but does not attempt to diagnose the root cause of a system problem by interpreting the error status.

Polo's PCI subsystem consists of 1 to 14 PCI cards arranged in two 7 slot groups. Each group has a controller card (the PCIB) bridging between the Xbus and two multi-slot PCI buses, and a PCMCIA flash card on the PCIB. The HSC2 subsystem consists of at least one (simplexed) HSC2 board with 1 to 4 PCI cards. On Polo and HSC2, the "SAM" is just an abstraction in the address space; 8 "SAMs" are implemented on a single PCIB card, 4 "SAMs" are implemented on a single HSC2 board, and all power up and down simultaneously. The only time new SAMs (PCI cards) appear is when a PCIB/HSC2 is powered up.

9.2 ID Proms

There is a Jetta-style ID prom on the PCIB and HSC2 for tracking board revisions and for error logging. The ID proms on the PCIB and HSC2 should not be programmed with information concerning the PCI card in the SAM's PCI slot.

There are no Stratus ID PROMs on the PCI cards themselves; these boards are identified by reading their PCI-defined configuration space. All PCI cards provide a Vendor ID (assigned by the PCI Special Interest Group to assure uniqueness), Device ID, and Revision ID. The PCI cards will not contain individual serial numbers in an on-line readable format. Most PCI cards have bar-code serialization only.

The Polo backplane ID Prom is accessed through the ReCC, identically to Jetta. Additionally, the Polo I/O power supply has 8 bits of ID information in the register space of the gambit ASIC.

9.2.1 Fault Logging

An important function of the ID proms is the logging of fault information to aid in diagnosis when a defective board is returned from the field. Each ID prom records faults in a 512 byte partition. The partition initially starts out empty, and it is filled in a round-robin fashion as faults occur.

Note that PCI cards do not support memory dumping. Any information from a stuck PCI card must be copied prior to resetting the card. Virtually all PCI cards zero memory on reset.

The following registers are logged for each board type. Note that different boards record different registers.

9.2.1.1 PCIB

A single PCIB is a bridge to 8 PCI slots. Accordingly, the relevant per-slot SAM registers appear 8 times in the fault log. In the following table, 'xxx' takes on the hex values 404, 40C, 414, 41C, 424, 42C,434, 43C. The four config space registers are the first 4 DWORDs of the PCI config space in the PCI card in the corresponding SAM; the M&D performing the logging must access them using the PCI configuration space mechanism. None of the per-bus registers are judged useful for diagnosing failures, and they are not included.

Table 13. PCIB registers logged

Offset	Register Name	size
7FFFE8	Board Reset	8
7FFFE0	Bus Interface State	32
7FFFC8	Slot ID	8
7FFF78: 7FFF60	Gen. Purpose Comm [3:0]	32
7FFF38	Bus Interface Fault Reporting	16
7FFF30	Common Broken Status	16
7FFF28	ASIC Specific Broken Status	16
7FFF20	Bus Info Error Status	16
7FFF18	Misc Error Status	16
7FFF10	Control Bus Error Status	16
7FFF08	Bus Error Byte Status	16
7FFF00	Voter Error Transceiver Status	16
7FF008	PCIB Status	32
7FF000	PCIB Configuration	32
xxxFD8	SAM PCI Error Register	32
xxxFB8	SAM Status	32
xxxFB0	SAM Configuration	32
config space 00	Device ID/Vendor ID	32
config space 04	Status/Command	32
config space 08	Class Code/Revision ID	32
config space 0C	BIST/header/latency timer/cache line size	32

9.2.1.2 HSC2

A single HSC2 is a bridge to 4 PCI slots. Accordingly, the relevant per-slot SAM registers appear 4 times in the fault log. In the following table, 'xxx' takes on the hex values 404, 40C, 414, and 41C. The four config space registers are the first 4 DWORDs of the PCI config space in the PCI card in the corresponding SAM; the M&D performing the logging must access them using the PCI configuration space mechanism. None of the per-bus registers are judged useful for diagnosing failures, and they are not included.

Table 14. HSC2 registers logged

Offset	Register Name	size
7FFFE8	Board Reset	8
7FFFE0	Bus Interface State	32
7FFFC8	Slot ID	8
7FFF78: 7FFF60	Gen. Purpose Comm [3:0]	32
7FFF38	Bus Interface Fault Reporting	16
7FFF30	Common Broken Status	16
7FFF28	ASIC Specific Broken Status	16
7FFF20	Bus Info Error Status	16
7FFF18	Misc Error Status	16
7FFF10	Control Bus Error Status	16
7FFF08	Bus Error Byte Status	16
7FFF00	Voter Error Transceiver Status	16
7FF008	PCIB Status	32
7FF000	PCIB Configuration	32
xxxFD8	SAM PCI Error Register	32
xxxFB8	SAM Status	32
xxxFB0	SAM Configuration	32
config space 00	Device ID/Vendor ID	32
config space 04	Status/Command	32
config space 08	Class Code/Revision ID	32
config space 0C	BIST/header/latency timer/cache line size	32

9.2.2 I/O Power Supply Faults

There are 2 status bits that identify that an I/O Power supply in a Polo system has failed. This information will tentatively be stored in ReCC status registers. Specifically, I/O power supply fault status will be stored in the ReCC bus and Local fault status register (address 0xF0000B). Bit 2 will be used for I/O power supply 1 fault, and bit 1 will be used for I/O power supply 0 fault.

9.3 Insertion & Removals

9.3.1 PCIB devices

After insertion, power-up and release of reset, a PCIB issues a maintenance interrupt to the host CPU. The host then goes through the process of setting the board on-line:

- read the id prom and verify revision compatibility with the rest of the system
- set the yellow led on
- TBD

When the lid to one of the Polo PCI card bays is opened, the power to the PCIB and 8 associated SAM slots (7 physical PCI slots and the flash ROM) is switched off. To software, this is identical to a simplexed MIO being pulled from a system. In theory, if all disks are duplicated on different controllers, and if comm is duplicated in both PCI bays as needed, opening a PCI bay lid on a running system should cause no problems. However, it may be advantageous to include an M&D function that quiesses all operation in a PCI bay in preparation for powering it down. From a signalling standpoint, the yellow LED will be illuminated on the remaining power supply.

9.3.2 Polo PCI devices

In the Polo and HSC2 systems, there are no SAM wrapped around the PCI cards to provide hot plugging. As a result, new PCI cards appear in the system only when a PCIB or HSC2 powers on, and the code to add a PCI card is simply run after the code to add a PCIB/HSC2. In Polo there is a single MAP ram in the CPU board; in Jetta there are separate MAP rams in each HSC2.

9.4 Reset Overview

All Xbus/Gbus boards support at least two levels of reset, a "warm" reset and a "cold" reset.

Cold reset resets as much state as possible on a board. It is always applied to a board by hardware upon first powering it up so that the C/D sides are sufficiently similar to permit operation of the board without breaking.

Warm reset resets as little state as possible. It is intended to unwedge a broken board with destroying as little state as possible so as to permit the diagnosis of fault conditions.

As part of the reset process, hardware first puts a board into the broken state and then about 1 microsecond later unbreaks the board. This has two consequences:

There is an approximately 1 microsecond period during reset that a board will not respond to bus activity.

A board generates a maintenance interrupt when it breaks or unbreaks. This means that a healthy unbroken board will generate two maintenance interrupts as part of reset process and a broken board unbroken by reset will generate a single maintenance interrupt.

This next section will attempt to summarize the effect of reset on the major components of the subsystems. For detailed bit-by-bit descriptions, please refer to Register Definitions on page 82 of this document.

9.4.1 Polo Resets

9.4.1.1 CPU Reset

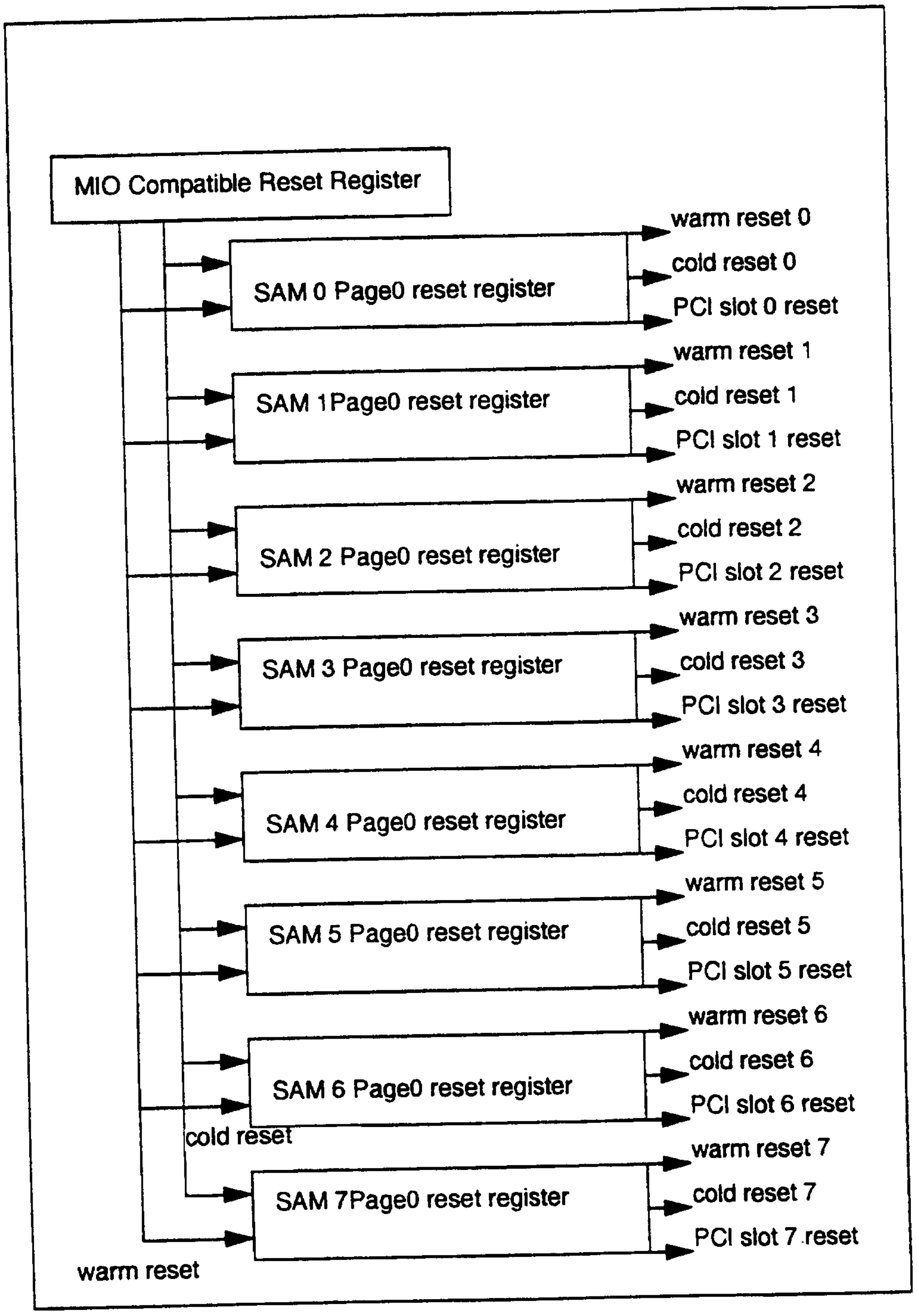
A Polo CPU can be warm or cold reset through two mechanisms. First, an Xbus reset can be issued to a Polo CPU through an I/O write across the Xbus to the CPU's reset register. This will cause a reset only to a non-broken CPU. Since broken boards do not respond to Xbus accesses, a reset to a broken board will not reset it. Polo implements a second reset function designed to reset a board regardless of its state. This is the suggested method for resetting boards in a Polo system, although the Golfbus method is supported for compatibility. This reset is accomplished by performing a local I/O write to the appropriate reset bits in the local Board Reset register. Setting the appropriate bit in this register causes a reset to be performed using a special set of dedicated three way voted lines on the Xbus. For more details on the protocol of this reset line, please refer to the Xbus Functional Specification. This reset does allow for cold and warm resets to be broadcast to any board in a Polo system, including CPUs.

The effect of a reset on a CPU is identical to a reset on a Golfbus system and is documented in the Golfbus Functional Specification.

9.4.1.2 PCIB

In the PCIB subsystem there is a 2 level hierarchy of resets. The two levels are the one MIO compatible Xbus Board Register, and the eight page 0 SAM compatible Reset registers. Figure 28, below shows a block diagram of the Resets relative to Polo.

Figure 28. Polo PCIB reset Hierarchy



The MIO compatible Xbus/Golfbus Board Register, documented in the Golfbus Functional Specification, allows for either a warm or cold reset to be performed. The effect of this reset is to reset the entire PCIB subsystem. From an FTIO standpoint, this reset is the equivalent of resetting an FTIO and every SAM connected to it. Warm and Cold resets leave the PCIB unbroken, off-line, and with only the red LED on. Warm reset leaves most register state (particularly error register state, PCI configuration space and the PCI boards themselves.) intact; Cold reset clears nearly everything. This reset will reset the MIO compatible Xbus/Golfbus registers, all 8 sets of page 0 per slot registers, both per bus registers, the two PCI buses connected to the PCIBs (cold reset only), all PCI devices connected to the PCIB (cold reset only), both sets of PCI configuration space information (cold reset only), and any associated state with the PCIB. This reset is intended to allow for resetting of a broken PCIB. As with the CPU resets, this can be accomplished either across the XBus or from a local CPU write. As with the CPU, the local write is the preferred method.

The page 0 SAM compatible Board Reset Register is supported to allow for resetting an individual SAM. A reset to this register leaves the PCI slot in the Off-line not ready state.This reset resets the pag0 SAM state for that slot, the PCI board in that slot (cold reset only), and any state associated with that slot. This reset is intended to allow software to revive a dead PCI card, recover from a single PCI card hang, and provide any compatibilities required between an FTIO system and a Polo system with regard to reset functionality.

9.4.1.3 PCI cards

PCI cards are provided by third party vendors, and do not support the warm and cold flavors of reset. All PCI cards implement a specified reset behavior for their PCI interface; the effects of reset on the remainder of the card is card-specific.

Typically, resetting a PCI card clears all useful registers and zeros out on-board memory. Please refer to the manufacturers specification for each PCI card.

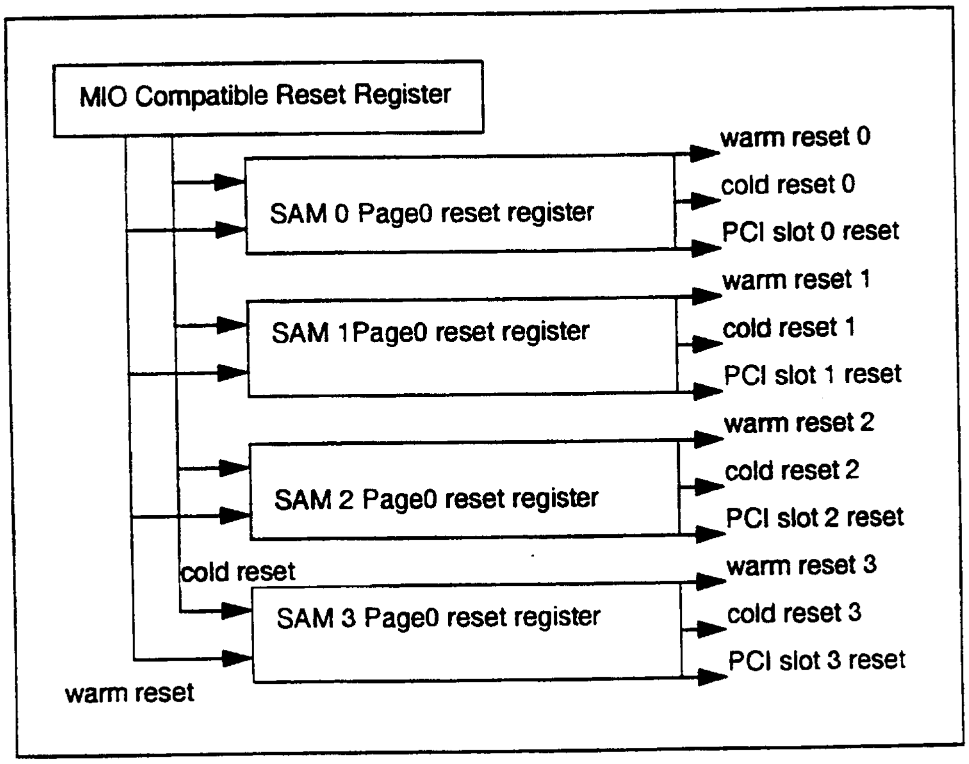
This reset is accomplished by writing a PCI reset to the appropriate page 0 SAM compatible board reset register. It has the effect of only resetting the PCI card and not effecting an SAM compatible or PCIB board state.

9.4.2 HSC2 Resets

9.4.2.1 HSC2

In the HSC2 subsystem there is a 2 level hierarchy of resets. The two levels are the one MIO compatible Xbus Board Register, and the four page 0 SAM compatible Reset registers. Figure 28, below shows a block diagram of the Resets relative to HSC2.

Figure 29. HSC2 PCIB reset Hierarchy



The MIO compatible Golfbus Board Register, documented in the Golfbus Functional Specification, allows for either a warm or cold reset to be performed. The effect of this reset is to reset the entire HSC2 subsystem. From an FTIO standpoint, this reset is the equivalent of resetting an FTIO and every SAM connected to it. Warm and Cold resets leave the HSC2 unbroken, off-line, and with only the red LED on. Warm reset leaves most register state (particularly error register state, PCI configuration space and the PCI boards themselves.) intact; Cold reset clears nearly everything. This reset will reset the MIO compatible Golfbus registers, all 4 sets of page 0 per slot registers, both per bus registers, the PCI bus connected to the HSC2 (cold reset only), all PCI devices connected to the HSC2 (cold reset only), the PCI configuration space information (cold reset only), and any associated state with the HSC2. This reset is intended to allow for resetting of a broken HSC2.

The page 0 SAM compatible Board Reset Register is supported to allow for resetting an individual SAM. A reset to this register leaves the PCI slot in the Off-line not ready state.This reset resets the pag0 SAM state for that slot, the PCI board in that slot (cold reset only), and any state associated with that slot. This reset is intended to allow software to revive a dead PCI card, recover from a single PCI card hang, and provide any compatibilities required between an FTIO system and a Polo system with regard to reset functionality.

9.4.2.2 PCI cards

PCI cards are provided by third party vendors, and do not support the warm and cold flavors of reset. All PCI cards implement a specified reset behavior for their PCI interface; the effects of reset on the remainder of the card is card-specific.

Typically, resetting a PCI card clears all useful registers and zeros out on-board memory. Please refer to the manufacturers specification for each PCI card.

This reset is accomplished by writing a PCI reset to the appropriate page 0 SAM compatible board reset register. It has the effect of only resetting the PCI card and not effecting an SAM compatible or HSC2 board state.

9.5 Determining the source of a Maintenance Interrupt

Maintenance interrupts provide notice that there one or more conditions requiring action from the M&D process. In general, it is impossible for software to distinguish between a single maintenance interrupt, and multiple maintenance interrupts in quick succession. Upon seeing a maintenance interrupt in a system, the following process can be used to determine the source(s) of the interrupt.

If an Golfbus/Xbus board that was in the system is no longer visible to software then that board generated a maintenance interrupt. (The recommended way of determining if an Xbus board is unbroken is by reading the Bus Interface Status Register; it is guaranteed to return non-zero on a unbroken board). To get more detail on what happened to the board, it can be reset and its Common Broken Status and ASIC Specific Broken Status registers examined. If multiple reset attempts have no effect, the board was either removed or is too broken to be read. The OS should then use the board presence information from the RECC to determine if a board has been removed.

Golfbus/Xbus boards that have not broken or been removed from the system will have either the "Bus Interface Maintenance Interrupt" bit or the "Board Logic Maintenance Interrupt" bits set in the their Bus Interface Reporting register if they activated maintenance interrupt.

If the "Bus Interface Maintenance Interrupt" bit is set, then Board Reset register may be read to see if a maintenance interrupt was due to a board reset. Similarly, the Bus Info Error Status and Misc. Error status registers may be read to determine if the maintenance interrupt was generated by any non-fatal errors (the board would be broken otherwise). If no bits are set in either of these two registers, the maintenance interrupt was software generated.

If a "Board Logic Maintenance Interrupt" is set on a PCIB or HSC2, then the interrupt originated from SAM slots associated with this board, or from a disk being inserted or removed from the Storageworks disk shelf. The next step is to read the SAM Maintenance Attention Request register. If any of bits 3-0 in SAM Maintenance Attention Request register is set, all four PCI Bus Interface State register for SAM3 - SAM0 should be read. If any of bits 11- 8 in SAM Maintenance Attention Request register is set, all four PCI Bus Interface State register for SAM7 - SAM4 should be read (Polo only).

In Polo, if a "Board Logic Maintenance Interrupt" is set on a Polo CPU, then the interrupt originated from an IOVA map error and the IOVA Map Error 1 and 2 registers should be examined (section 12.4.3 and section 12.4.2). The TRID can be used to determine the PCI slot that sourced the Xbus IOVA operation. The IOVA map error maintenance interrupt can also be masked from the IOVA Map Error 1 register.

The next step is to examine the SAM Status and Configuration registers for all PCI slots that issued maintenance interrupts. Bits 7:21 specify various sources of maintenance interrupts; only some bits are implemented in Polo & HSC2 systems.

On Polo systems, the final place to check is the Polo Disk Status register. This indicates whether a maintenance interrupt was issued due to a disk swap event. Unfortunately, it does not specify whether a disk was added or removed; only that there was a change of state.

9.6 Faults & Errors

9.6.1 PCI card faults

Typically, there will be no Stratus style fault detection comparison logic on 3rd party PCI cards. Some boards may compute check-sums or utilize parity on memory arrays; support for this level of fault reporting is the responsibility of the card driver. However, there should be an interface for drivers to communicate a text string of fault information to the M&D, so it can be incorporated into the SYSERR log and be available to the CAC.

The presence or absence of a PCI card is indicated in the PCI config space Vendor ID register. The PCI bridge will return a value of all 1s (an invalid vendor ID) on read accesses to configuration space registers of non-existent devices.

9.6.2 PCI bus faults

9.6.2.1 PERR#

The PCI bus is protected by a single Even parity bit over 36 bits of information (32 address/data and 4 C/BE). The parity is driven with the same timing as the address/data, but delayed by one clock. All agents on the PCI bus must generate parity.

Data parity errors are indicated using the **PERR#** signal. Only the device which claims the cycle may drive the **PERR#** signal on the PCI bus. The Gambit/Mirage ASIC will continuously monitor PCI transactions and the state of the **PERR#** signal, regardless of the selected device. Any PCI parity errors observed during non-Special Cycles will result in a maintenance interrupt action back to the host CPU.

PERR# is a non-recoverable condition and causes a state reduction in the Gambit/Mirage ASIC from on-line to off-line, not ready. This reduction in state will cause a maintenance interrupt. It is up to software to test and bring the card back into service if it can.

9.6.2.2 SERR#

System errors which could be catastrophic are indicated on the SERR# signal. Any PCI agent which detects a non-recoverable condition can assert SERR#. The Gambit/Mirage ASIC monitors the state of the SERR# signal and will issue a maintenance interrupt up to the host processor for any assertion of SERR#. The state of the SAM will then be reduced to Off-line, not ready. The Gambit/Mirage ASIC does not generate target-abort for this case.

A PCI adaptor non-recoverable condition includes:

1. Address parity errors on all cycles.
2. Data parity errors on Special Cycles.
3. A catastrophic adapter fault.

SERR# can be disabled via config space on all PCI devices (including the Gambit asics). All PCI devices power-up with SERR# disabled; it is up to M&D software to enable SERR# assertion.

9.6.2.3 Polo(Gambit)/HSC2(Mirage) Error logic

Polo(Gambit) detects several error conditions on PCI bus and takes different actions depending upon error kind. Actions taken for different kind of error is given below.

1. Target Abort

   If the bridge is current bus master, bring the SAM who drove the devsel line to OFFLINE-NOT-READY state. If this is a read access by host, return failed op.

   If the SAM is current bus master, bring the current bus master to OFFLINE-NOT-READY state.

Set the value in PCI Error register and PCI IOVA Error register.

2. Master Abort

If the bridge is current bus master, and this is a read access by host, return failed op.

If the SAM is current bus master, bring the current bus master to OFFLINE-NOT-READY state.

Set the value in PCI Error register and PCI IOVA Error register.

3. Parity Error

If the bridge is current bus master, bring the SAM who drove the devsel line to OFFLINE-NOT-READY state.

If the SAM is current bus master, bring the current bus master to OFFLINE-NOT-READY state.

Set the value in PCI Error register and PCI IOVA Error register.

If the bridge is bus master and is reading from the SAM, failed op returned to the host.

If the bridge is bus master and is writing to SAM, it is up to the SAM to take appropriate actions.

If the SAM is current bus master and is writing to the host, the transactions is dropped. No transaction happens on X bus.

If the SAM is current bus master and is reading from the host, it is up to the SAM take appropriate actions.

4. System Error

If the dont_break_on_serr bit is not set in test control register, bring all SAMs to OFFLINE-NOT_READY state. If the dont_break_on_serr bit set in test control register, nothing happens. On power on, dont_break_on_serr bit is cleared.

Set the value in PCI Error register and PCI IOVA Error register.

If the bridge is bus master and is reading from SAM, failed op returned to the host.

If the bridge is bus master and is writing to SAM, it is up to the SAM to take appropriate actions.

If the SAM is current bus master and is writing to the host, the transactions is dropped. No transaction happens on X bus.

If SAM is current bus master and reading from the host memory, the transaction is dropped. No transaction happens on X bus. The SAM receives target abort.

Note: If the host read of PCI bus gets SERR and retry, but do not get SERR for retried transaction, data is returned to the host. However PCI Error and PCI IOVA Error register is updated.

5. Time Out Error (TRDY timer error)

If the bridge is current bus master, assert PCI reset to the SAM who drove the devsel line. If this is a read access by host, return failed op.

Set the value in PCI Error register and PCI IOVA Error register.

6. Drive Check Error

   Break PCIB/SAM board.

   Set the value in PCI Error register and PCI IOVA Error register.

7. Retry Count Error

   Bring the SAM who drove the devsel line to OFFLINE-NOT-READY state. If this is a read access by host, return failed op.

   Set the value in PCI Error register and PCI IOVA Error register.

8. ~~Disconnect Count~~ IRDY Timeout Error

   Issue PCI reset to the SAM who drove the devsel line. If this is a read access by host, return failed op.

   Set the value in PCI Error register and PCI IOVA Error register.

9. Peer to Peer Error

   Bring the current bus master to OFFLINE-NOT-READY state.

   Set the value in PCI Error register and PCI IOVA Error register.

10. Protocol Error

    If the bridge is current bus master, issue PCI reset to the SAM who drove the devsel line. If this is a read access by host, return failed op.

    If the SAM is current bus master, issue PCI reset to the SAM.

    Set the value in PCI Error register and PCI IOVA Error register.

    Note: If a PCIB is issued a warm reset while PCI traffic is in progress, a protocol error may be detected by Gambit error logic.

    In certain cases of protocol error, write buffer data in the Gambit ASIC goes on X bus even if there is a protocol error. (e.g. A pci adapter does a write to gambit, but does not assert irdy_ and deasserts frame_, causing a protocol error) .

11. Host Request FIFO Time-out Error

    This error is detected when Host Request FIFO (inside Gambit ASIC) requesting to access PCI bus fails to advance by the number of clock ticks specified in Host Request FIFO Time-out Value Register. In this case, PCIB broken so that CPU does not hang, waiting for data to return from PCIB.

    Break PCIB/SAM board.

    Set the value in PCI Error register and PCI IOVA Error register.

    Note: This error happens in some rare catastrophic condition(e.g. frame_ signal stuck low on PCI bus). When this error condition happens, the PCI IOVA Error Register and PCI Error Register may or may contain the info about the transaction that caused this error, since it is impossible to figure out which transaction caused this error.

The PCI error logic in Gambit implements PCI IOVA register and PCI Error register for diagnosing errors. The PCI Error Register is read only by host. This register has arm/rearm control. This register could be armed/rearmed by host. When PCI Error Register is armed, the PCI IOVA register contains the PCI bus address that caused error. When an error occurs, the PCI IOVA register and PCI Error registers are frozen so that it holds the address that caused error. Subsequent errors does not affect the register. It needs to be rearmed before it could store address for next error. For description of PCI IOVA register and PCI Error register, refer to section 13.5.5.

9.6.3 Disk bus faults

The DEC Storageworks disk solution provides the necessary features for Stratus disk M&D. In particular, Storageworks accomplishes live insertion, insertion/removal detection, and fault LEDs. In order to perform these tasks DEC has implemented two basic features. The first one deals with live insertion. Live insertion is accomplished through some current limiting and power sequencing. It seems to work. The latter two features are accomplished through DEC's fault bus. The fault bus consists of a swap interrupt and an LED bus. Both of these features are implemented on Polo.

There are a series of operations which should be performed by the M&D processes associated with the various M&D features of the disk subsystem. These are discussed in the sections below.

9.6.3.1 LED Initialization

LED initialization should be performed at power up time. The LED initialization routine provides a simple way to turn off all of the LEDs. At power up time, the LED state is non-deterministic, and must be set by setting/resetting each LED individually, or as a group by using the broadcast bit. The broad cast is performed by setting (turn on LEDs) or clearing (to turn off LEDs) bit 4 and setting bit 3 (the broadcast bit) in Disk LED Control register. This register is described in section 12.6.1.1 of this document. In a broadcast write to the disk LEDs, the SCSI device target ID (bits 2:0) are not meaningful. For example, to set all of the LEDs in a SCSI bus, a 18x would be written to the register.

9.6.3.2 Disk LED Control

Individual LED control is performed on an as needed basis by the software. Software has control over one of the LEDs, the fault indicator, on the disk. In order to address a particular slot, bit 3 (the broadcast bit) must be clear, bits 2:0 should be set to the SCSI device target ID, and bit 4 should be set (to turn the LEDs on) or cleared (to turn the LEDs off) in the Disk LED Control register. This register is described in section 12.6.1.1 of this document. The SCSI device target ID is the same as the SCSI devices normal SCSI device number. For example, to set the led for SCSI device 2, a 012x would be written to the Register. to clear the LED for the sam device, a 002x would be written to the register.

9.6.3.3 Disk Insertion and Removal

Finally, disk insertion or removal is detected through the swap has occurred bits. For the C-side shelf, this is on bits 1 and 17 and for the D-side it is on bits 9 and 25 of the Disk Status register. If the corresponding interrupt disable bit is cleared, then a maintenance interrupt will be generated due to this event. The interrupt is cleared by clearing the appropriate bit in the register, bit 1 for C-side interrupts, and bit 9 for D-side interrupts. The swap interrupt reflects only that a change has occurred in the state of the disk shelf, the M&D software must perform a SCSI poll to determine the nature of the change.

9.6.4 SCSI bus faults

Faults on the SCSI bus are handled in a manner specific to the PCI card.

9.6.5 PCIB/HSC2 faults

9.6.6 Xbus/Golfbus faults

Xbus faults reported by the PCIB/HSC2 are handled like those from any other Xbus/Golfbus board. Please refer to the Xbus/Golfbus Functional Specification for more information.

9.7 Diagnostics

9.7.1 PCI Card

The level of diagnostic support for the PCI cards will be consistent with the PCI Third-Party Adapter Requirements Specification. Most PCI cards will be very highly integrated (consisting of 1 or a few chips), and all are anticipated to be very reliable. In any case, they are non self-checking components, and the system is protected from incorrect operation by higher level mechanisms: address re-mapping, check-sums, and time-outs.

The system must be able to isolate faults down to a field replaceable unit. If it is not possible to isolate faults to this level via the error reporting registers, specific fault isolation diagnostics will be needed.

9.7.2 PCIB

The PCIB/HSC2 board has no local processor and an extremely low chip count. It will be tested and diagnosed exclusively by scan.

10. ReCC Functionality on Polo

This section describes key changes envisioned for the ReCC firmware on Polo. On Polo, it is a goal to isolate the OS from many changes in the M&D subsystem that are particular to Polo.

10.1 Remote Power control

One feature on Jetta that is not supported on Polo is that ability to cycle power from the ReCC. The Polo machine does not implement house keeping power or the power supply control necessary to perform this function. Therefore it is not supported on Polo. ReCC firmware should probably stub out this command set. It has been suggested that the ability to power down the machine from the ReCC be supported. This feature would require spinning the else gate array on the ReCC and therefor will not be supported.

10.2 RS-485 bus

Polo does not implement the CDC. In the initial release of Polo there is no 485 bus devices. The 485 is in the system for future expansion but with no devices in the 485 system, it will not respond to any polls. Therefore, the ReCC firmware should return intelligent response to M&D requests for 485 information.

10.3 Powerfail

Polo does not support powerfail. Polo's solution to AC failures is accomplished through an interconnect to an external UPS. The interconnection to the UPS will be documented in a later release of this specification. It is expected to be RS-232, and it will be connected to the secondary console when it is used.

10.4 Clockcard, backpanel power supply failures

The clock card and backpanel power supplies are not implemented in Polo, therefore there will be no signalling of these faults.

10.5 I/O power supply failures

The I/O power supply fault signals may be rooted in place of the backpanel power supply signals. If these lines are activated it signals that the I/O power supply has determined that a fault exists in the I/O power supply.

10.6 NVRAM

It would be preferable to replace the full functional EEPROM with flash prom. This may effect the writing algorithm for the PROM.

11. Board States

This section describes the board states for the PCIB and SAM modules in the Polo system and the HSC2 for the Jetta system.

11.1 PCIB Board States

The Polo PCIB module would gain little by supporting duplexed operation for the following reasons:

1. All paired SAM space accesses, whether to memory or I/O must be converted in hardware to non-paired. This is because the Polo PCIB does not have the benefit of going through an MIO before driving the Xbus so the two PCIB boards cannot know the same information.

2. Many of the registers in Gambit could only be accessed through paired space due their definitions already.

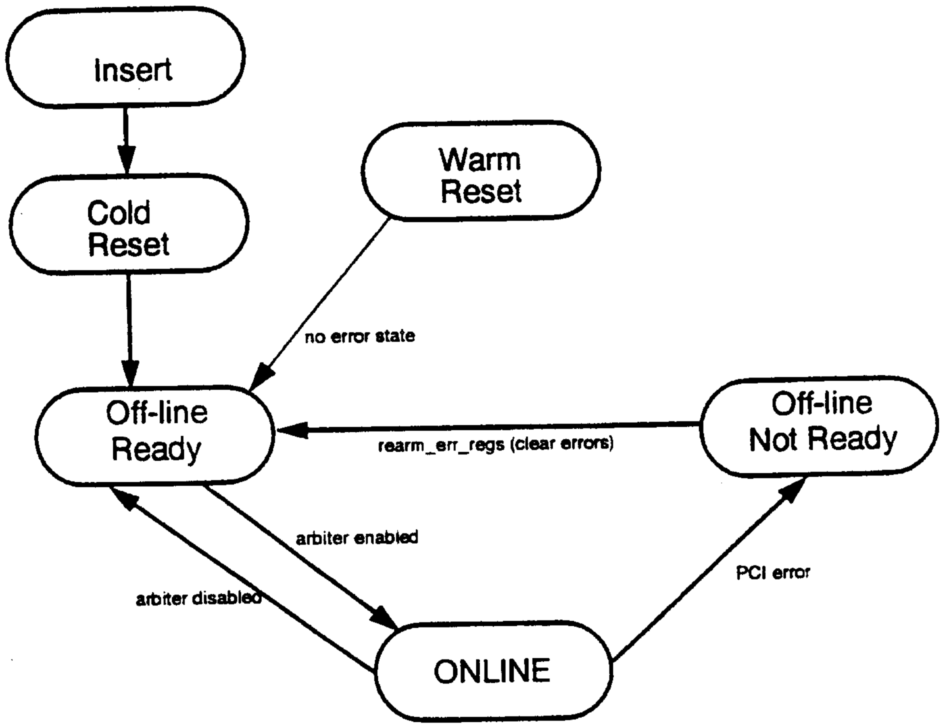
The task of implementing a duplexable PCIB in hardware would add work with little or no gain so the PCIB can only be run in simplexed mode.

11.2 HSC2 Board States

The HSC2 is similar to the BIO in that it does not support duplexed operation. The HSC2 is similar to the Polo PCIB in that it shares the SAM board states model detailed in SAM Board States on page 80.

11.3 SAM Board States

Figure 30. SAM-PCI Board State Transition



Board States:

OFF-LINE - Not Ready

Board capable of receiving and responding to XBus/Golfbus requests, but is incapable of performing PCI adaptor initiated requests. There is non-zero error state in the SAM.

This state indicates the occurrence of a PCI Bus related hardware fault which was detected by the checking logic. This state cannot be entered by a software set off-line command as there must be non-zero error state in the SAM.

OFF-LINE - Ready

Board capable of receiving and responding to XBus/Golfbus requests but is incapable of performing PCI adaptor initiated requests. No error conditions are present.

This state is entered automatically after a cold or warm reset (provided there are no errors detected by the SAM) or when the error state of the chip is cleared via a Rearm Error Command. Also software can enter this state by either a set off-line command or by a clear on-line command.

ONLINE

Board capable of receiving and responding to XBus/Golfbus requests. Board capable of performing XBus/Golfbus initiated PCI requests. PCI adapter enabled to post requests.

Software can enter this state with a set on-line command to the Bus Interface Sate Register provided there is a non-zero error state.

Figure 31. SAM board states.

ON-LINE	READY		RECEIVE SYSTEM BUS COMMANDS	RESPOND TO SYTEM BUS COMANDS	PCI ADAPTER TARGET REQ.	PCI ADAPTER MASTER REQ.
X	X		1	0	0	0
0	0		1	1	1	0
0	1		1	1	1	0
1	1		1	1	1	1

KEY:

• 1	—	State bit true.
• 0	—	State bit false.
• X	—	State bit true or false.

12. Register Definitions

The following section outlines the I/O space registers in the Polo system.

12.1 XBus/Golfbus Bus Interface Registers

All IO registers in the Xbus/Golfbus interface are designed to be accessed as 32-bit registers so as to be accessible by all devices on the bus. Writes to any smaller data size Xbus/Golfbus registers are ignored and reads return 32-bits even if a smaller size datum was accessed. Though all registers are 32 bits, they are spaced at 64 bit (8 byte) intervals to aid future board designs which may make use of 64-bit internal busses.

Writing any non-existent IO register has no effect (i.e. unused addresses do not wrap around and map to used addresses). Reading a non-existent IO register either returns 0's or causes a NACK, depending on the particular register. The read never has any side effects.

There are two types of IO registers supported in the Xbus interface in additional to interrupt registers intended to mimic the HP-PA interrupt scheme:

1) read/write registers

These registers are readable/writable 32 bit registers for which no bit encoding occurs, i.e. assuming hardware events have not altered any state, what is written to bit 0 is read from bit 0, what is written to bit 1 is read from bit 1, etc. For upward compatibility, unused bits should always be written with 0's, though 1's have no effect. Reads of unused bits always return 0's.

2) read/set/clear registers

Read/set/clear registers are used when individual bits need to be atomically updated. Reads return 32 bits; writes specify one of the 32 bits to be either set or cleared. Specifically, writes are performed by selecting one of the 32 read bits bit to be atomically set or cleared with the five least significant bits (4:0) pointing to the bit and using the 8th least significant bit (7) to set (=1) or clear (=0) the selected bit. For example, to set bit 2 of the read register, 00000082x is written to the register. To clear bit 2 of the read register, 00000002x is written.

Unused bits always return 0's and for upward compatibility unused bits should always be written with 0's, although writing 1's has no effect.

3) read/clear registers

Read/clear registers are used when individual bits need to be atomically updated. Reads return 32-bits; writes specify one of the 32 bits to be cleared. Specifically, writes are performed by selecting one of the 32 read bits bit to be atomically cleared with the five least significant bits (4:0) pointing to the bit and using the 8th least significant bit (7) clear (=0) the selected bit. For example, to clear bit 2 of the read register, 00000002x is written. Attempts to set the register will have no effect.

Unused bits always return 0's and for upward compatibility unused bits should always be written with 0's, although writing 1's has no effect.

12.2 Reading Registers with Different C/D Information (Polo Only)

Note: This section does not apply to the HSC2 as there is no board processor to read the error registers in local space.

There are certain C and D registers that can have different values (for example the error registers). In a duplexed pair of boards the register values between board pair may also vary. There must be a way of reading these registers via either the local or a remote processor.

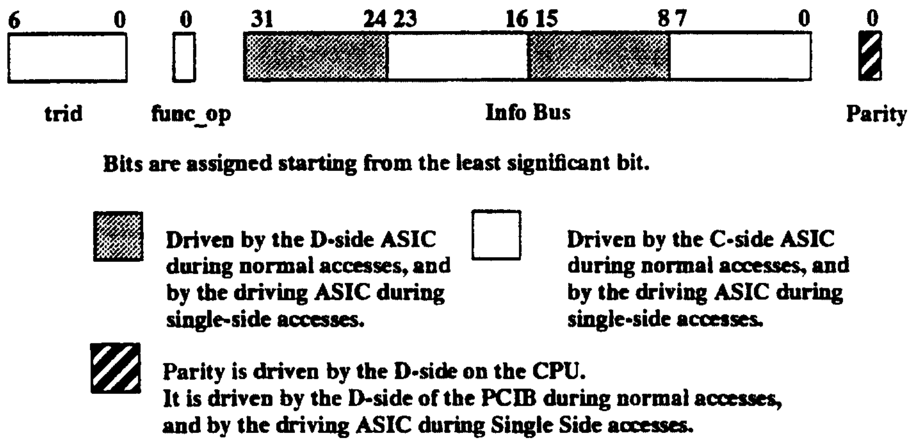
These registers should not be read in local space because this could bring duplexed boards and/ or the C&D board halves out of lockstep when the register values are different.

The access to these registers is similar to unpaired reads in that part of the data checking has to be ignored. During an unpaired read the board not providing the data has to ignore the checking but go through the motions to remain in lockstep. In this case the concept is taken a little further. Each board half (except the board half actually being read) goes through the motions but ignores the error checking on the info; however the trid, func_op, and parity are still checked.

Since some of the bits are driven from the C-side and some are driven from the D-side and the data from the two sides can differ, there is a complication in generating good parity. The solution to this problem is to place the differing error bits in the regions of the info bus that are driven by the ASIC responding to the read (see figure 7). For example C-side error bits have to be in the region 7:0 or 23:16, and the D-side from 15:8 and 31:24. To get good parity each side drives the same data on each of the bytes that the ASIC drives and puts zeros into the bytes that the ASIC does not drive. This will result in an even number of ones in the parity chain.

These registers must be readable when the board is off-line which is not an issue because the board responds to unpaired reads when it is off-line. There is a problem reading these registers if the board is broken. When the board is broken, it cannot be read or written by software. To access these registers on a broken board there must be a warm reset which brings the board to the off-line state.

Figure 32. Bus Driver Definition


6
0
0
31
24 23
16 15
8 7
0
0
trid
func_op
Info Bus
Parity
Bits are assigned starting from the least significant bit.
Driven by the D-side ASIC during normal accesses, and by the driving ASIC during single-side accesses.
Driven by the C-side ASIC during normal accesses, and by the driving ASIC during single-side accesses.
Parity is driven by the D-side on the CPU.
It is driven by the D-side of the PCIB during normal accesses, and by the driving ASIC during Single Side accesses.

12.3 Common Xbus/Golfbus Register Definitions (Polo Only)

Note: HSC2 supports the set of common Golfbus Registers. Refer to the Golfbus Specification for details concerning the common Golfbus Registers for the HSC2 board.

The Xbus supports a set of common registers that are present on all Xbus boards. These are modeled directly after the Golfbus interface registers, and preserve compatibility to the bit level whenever possible. However, due to the different physical partitioning of the Polo system, there are necessarily some changes. To ease the task of porting software to the Polo system, we define four levels of compatibility for each register:

- exact: this register preserves exact read/write functionality with the Golfbus definition, and may be programmed in exactly the same way. Examples include the ID PROM Address Data register.

- usage change: this register is bit for bit identical to the corresponding Golfbus register, but the usage in practice may be different. For example: the performance monitor registers can be programmed to monitor the function codes on the backplane; however, the Xbus is 32 bits and does not use the Golfbus 64 bit backplane function codes.
- subset/super-set: the Xbus version of this register is a subset or super-set of the Golfbus register. It may be programmed in the same way, but bit values that make sense in Golfbus systems may not be supported in Xbus systems or visa-versa. It may also have additional bits that do not exist in a Golfbus board. An example is the Bus Interface State register: the Xbus does not have the equivalent of the bus obey, so setting the "obey_a" bit has no effect.
- incompatible: this register has new meanings and bit definitions in an Xbus system.

The Xbus/Golfbus compatibility column in table 15 indicates the level of compatibility with the corresponding Golfbus register. Descriptions of the registers follow the table

Table 15. I/O Space Map.

Offset	Register Name	Type	C/D different	Xbus/Golfbus compatibility
7FFFF8	ID PROM Instr	Read/Write	no	exact
7FFFF0	ID PROM Address Data	Read/Write	no	exact
7FFFE8	Board Reset	Read/Set/ Clear	no	superset
7FFFE0	Bus Interface State	Read/Set/ Clear	no	subset/superset
7FFFD8	Board Sync Register	Read Only	non-deterministic; data MUST be discarded	exact
7FFFD0	LED Control	Read/Write	no	exact
7FFFC8	Slot ID	Read Only	no	exact
7FFFC0	Read Ping Interval	Read/Write	no	exact
7FFFB8	Set Interrupt/Interrupt Status	Read/Write	no	not impl.
7FFFB0	Clear Interrupt	Write only	no	
7FFFA8	Set Interrupt Mask/Interrupt Mask	Read/Write	no	not impl.
7FFFA0	Clear Interrupt Mask	Write only	no	not impl.
7FFF98: 7FFF60	Gen Purpose Comm [7:0]	Read/Write	no	exact
7FFF58	Memory Size/Location	Read/Write	no	implemented for S/W compatibility, but ignored by H/W
7FFF48	Test Control	Read/Set/ Clear	CD different if difference bit is set	subset
7FFF40	Parity Test	Read/Write	no	not implemented
7FFF38	Bus Interface Fault Reporting	Read/Set/ Clear	yes	incompatible
7FFF30	Common Broken Status	Read Only	yes	incompatible
7FFF28	ASIC Specific Broken Status	Read Only	yes	incompatible
7FFF20	Bus Info Error Status	Read Only	yes	subset

Offset	Register Name	Type	C/D different	Xbus/Golfbus compatibility
7FFF18	Misc Error Status	Read Only	yes	subset
7FFF10	Control Bus Error Status	Read Only	yes	new register
7FFF08	Bus Error Byte Status	Read Only	yes	usage change
7FFF00	Voter Error Transceiver Status	Read Only	yes	incompatible
7FFEF8	Bus ASIC Chip Revision	Read Only	no	new values
7FFEF0	Perf_Counter	Read/Write	no	exact
7FFEE8: 7FFEE0	Perf_Counter_Trig[1:0]	Read/Write	no	exact
7FFED8: 7FFED0	Perf_Counter_Mask[1:0]	Read/Write	no	exact
7FFEC8	Perf_Counter_Control	Read/Set/ Clear	no	exact
7FFEC0: 7FFEB8	Fault Bit[1:0]	Read/Write	no	new register
7FFEB0	Data Match	Read/Write	no	new register
7FFEA8	Error Control	Read/Set	no	new register

General Warning: Many of these registers have bits which can be toggled during self-test and left in varying legitimate states. Prior to or as part of duplexing, all bits must be ensured consistent.

Second General Warning: Several of the error reporting registers are "CD different" and can legitimately contain data that is different between C and D sides of the board. They must only be read via unpaired space, or they could break the board or hang the system.

The format of each register description is as follows. The paragraph header for each description provides the register name. The bold faced text immediately following contains the register type, address(es), reset states, and a listing of boards types it is present on.

12.3.1 ID Prom Instr

Type: Read/Write **Offset [22:0]: 7FFFF8**
Present on all Xbus boards
Warm and Cold have no effect on this register
Compatibility: exact

Writing to this register scans an instruction into the external JTAG ID Prom logic. Reading this register returns the status of the scan. It is only used for initializing the ID Prom logic after power-up or reset.

31:8 Reserved

7:0 JTAG instruction/status for the external ABT18245 scan register.
Refer to the next section for usage. Valid instructions are

```
SAMPLE_PRELOAD     = 82x
EXTEST             = 00x
```

Valid return status

```
OK_STATUS          = 81x
```

12.3.2 ID PROM Address Data

Type: Read/Write **Offset [22:0]: 7FFFF0**
Present on all Xbus boards
Warm and Cold have no effect on this register
Compatibility: exact

31:28 ID prom opcode. This controls the WE_, CE_, and OE_ lines of the ID Prom

```
id_READY          = 6x
id_PRESENT_DATA   = Ex
id_WRITE          = Ax
id_OFF            = 7x
id_READ           = 4x
```

27:20 ID Prom Data. On writes, this is the data to be written into the id prom. On reads, it is the data returned

19:11 Reserved

10:0 ID Prom Address.

Every Xbus board has a 2Kx8 writable ID PROM. The PROM is intended to store various board status information such as serial number, eco rev level, PCB rev level, etc.

The actual PROM used is the X2816C 2Kx8 EEPROM from Xicor. This part is accessed though a JTAG compatible serial scan interface to save pins on the bus ASIC and to allow the part to be accessible via the board's scan test logic. A Texas Instruments ABT18245 scannable buffer is used to translate from the scan interface to the ID Proms broadside pins. More detail on the use of this part is available in section 11.3.1 on page 98 (ID PROM Partitioning) in the Xbus Functional Specification.

Logically, the ID PROM appears as two virtual 32-bit registers, an instruction register for sending initialization commands to the ABT18245 scan chip, and an address_data register for reading and writing address/data pairs to the id prom itself. These two virtual registers are actually implemented as a single 44 bit scan register in the ASIC. Writing to the instruction register initiates an 8 bit instruction scan to the ABT18245, and writing to the address_data register initiates a 44 bit data scan which drives and samples the pins of the id prom. Reading the address_data register returns the bits of the scan register corresponding to the address, data, and opcode fields. Reading the instruction register returns an 8 bit value corresponding to the results of the instruction scan. Note that there is only one scan register; a read of id prom instruction reg should only follow a write to the id prom instruction, and a read of the id prom address data reg should only follow a write to that address. Otherwise, deterministic gibberish will be returned. Once initialized, reads of the instruction and address_data registers will always return non-zero data; if a zero is ever returned, it is due to the target board going broken or a simplexed fault in the ID prom hardware.

The ID Prom hardware needs to be initialized after power-up or a warm or cold reset. Once it has been initialized, the prom can be read or written. The ID prom is a fully random access device, with no sectors or erase modes. When a single byte is written to the X2816C, the device begins an internal programming cycle that takes ~5mS. While the programming cycle is in progress, bit[7] of the byte written to the part will appear inverted when read; software needs to poll the device to determine when the program cycle is finished. The ID prom has an endurance of 10,000 writes.

To initialize the ID Prom hardware:

```
ID_prom_instr = SAMPLE_PRELOAD;          //(0000_0082x)
ID_prom_instr = EXTEST;                  //(0000_0000x)
```

```
if(ID_prom_instr != OK_STATUS);        //(0000_0081x)
     "hardware failure in ID prom"
ID_prom_address_data = {id_OFF,28'h0000000}
```

To read the ID prom addresses start_addr to end_addr

```
ID_prom_address_data = {id_READ,8'h00,start_addr}
for (i = start_addr; i<= end_addr; i++)
     {
     ID_prom_address_data = {id_READ,8'h00,i+1}
     read_data[i] = ID_prom_address_data[27:20]
     }
ID_prom_address_data = {id_OFF,28'h0000000}
```

To write mybyte to the ID prom address start_addr

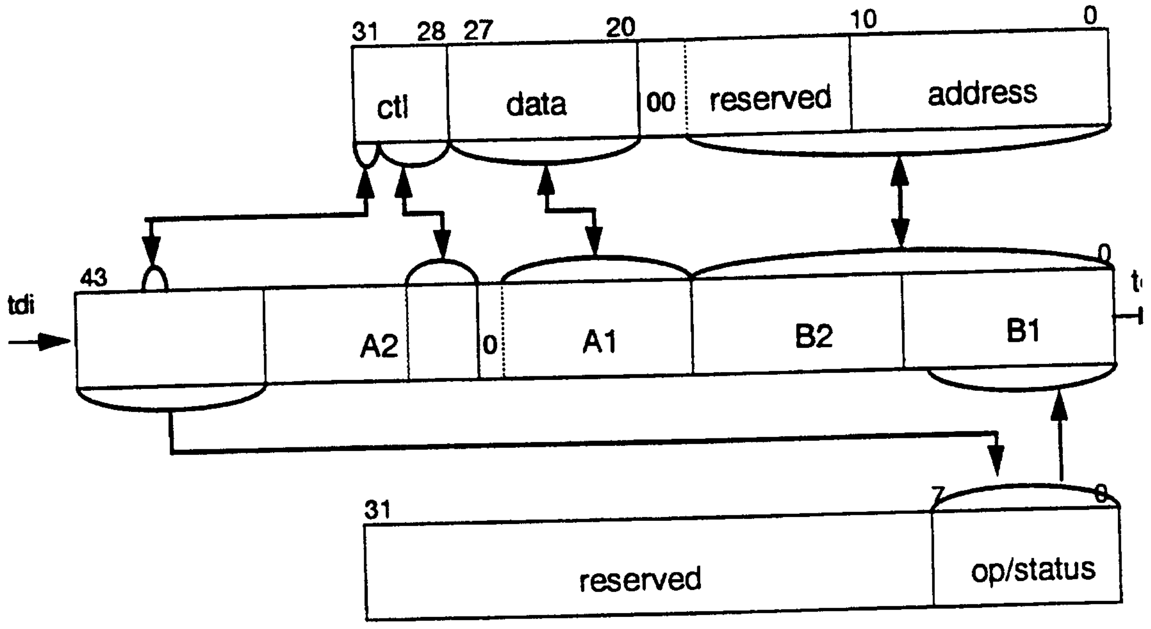
```
ID_prom_address_data = {id_PRESENT_DATA,mybyte,start_addr}
ID_prom_address_data = {id_WRITE,mybyte,start_addr}
ID_prom_address_data = {id_PRESENT_DATA,mybyte,start_addr}

/* poll to wait for programming cycle to complete */
ID_prom_address_data = {id_READ,8'h00,start_addr} /* once */
ID_prom_address_data = {id_READ,8'h00,start_addr} /* again */

isbyte= ID_prom_address_data[27:20]
while (isbyte[7] != mybyte[7])
     {
     wait 1 mS
     ID_prom_address_data = {id_READ,8'h00,start_addr}
     isbyte= ID_prom_address_data[27:20]
     }
ID_prom_address_data = {id_OFF,28'h0000000}
```

Note that two successive writes of the id_READ opcode are required before read data can be examined. This is because the boundary scan ring in the ABT18245 acquires the data on its pins before applying the new data. The first write acquires old data, then applies the id_READ opcode and address to the ID Prom. The second write acquires the results of the read opcode and first address, then applies the id_READ opcode (and optionally a new address).

Note that ID PROMs differ in contents from board to board. Boards operating in lock step should always read and write them in an unpaired fashion. An unpaired read to an ID PROM belonging to one of a duplex board pair causes both board's state machines to perform the actual read, though only one board actually drives the bus with the read data. An unpaired write causes both board's state machines to go through the motions, though TCK is inhibited on the board not being written.

This JTAG scan occurs at 12 MHz, hence each write of these registers takes about 5 usec. The bus ASIC circuitry automatically delays a subsequent access if an earlier initiated write is still in progress. When the ID Prom is accessed via global, paired or non-paired space, other IO accesses can take place while an ID prom access is in progress. When the ID prom is accessed via Local space, all other IO space access to Bus Interface registers are held off until the access completes.

Figure 33. Internal Implementation of ASIC scan register



Warning: Because multiple processors/processes simultaneously accessing the ID PROM may interfere with each other, software must manage access to the ID PROM to ensure only a single processor/process is accessing it at a time.

Warning: The ID PROM is a simplexed (non error protected) component. A checksum or other similar protection scheme should be utilized by software to ensure the integrity of the data read from the ID PROM. Also, because ID PROMs on partnered boards will have different contents, ID PROM Data registers should always be read in an non-paired fashion.

12.3.3 Board Reset

Offset [22:0]: 7FFFE8

Type: Read/Set/Clear
Present on all Xbus boards
Cold and warm reset affect this register as documented below
Compatibility: super-set

Polo boards have two resets, a cold reset applied to the board at power up (or via software or ReCC) and a warm reset applied via software or the ReCC once the machine is running. Cold reset brings all registers to a known deterministic state, warm reset generally resets state machines and cancels operations in progress but leaves as much of the register state untouched as possible to aid in debug.

The activation of any of the bits in this register (excluding test cases) mean that a reset pulse was earlier driven on the board, not that a reset line is currently asserted. It should be noted that as part of the normal reset process, the board goes broken for the duration of the reset pulse, and then unbroken when the reset de-asserts.

Cold resets clear all bits in this register except for the bit identifying the source of the cold reset, warm resets leave bits in their current state (except for activating the bit that identifies the source of the warm reset).

This is a Read/Set/Clear register, commands are performed by writing to this register with data bits [4:0] pointing to a bit to set or clear and bit 7 determining whether the bit is set (=1) or cleared (=0). Reading this register returns 32-bits. Setting either "Software initiated Cold Reset" or "Software initiated Warm Reset" actually causes a reset, even if the bit was already set. Setting (to test the register read path) the other bits cause the bits to go to 1, but no reset takes place.

Polo is different from Golfbus systems in that writing the Board Reset register of a broken board across the bus has no effect. For this reason, Polo has 4 additional reset bits that when activated send a reset pulse to the corresponding slot. This reset is interpreted as a cold reset if it is three 4 Mhz periods long and as a warm reset if the pulse is two 4Mhz pulses long. Software Initiated Cold Reset and Software Initiated Warm Reset still work as before when performed locally, and work across the bus when the target board is unbroken.

31:13 These read bits/write encodings are reserved for additional board specific resets.

12 Cold Reset Slot 3

Writing to set one of these bits sends a cold reset pulse to the board in the corresponding slot. These bits return 0 when read. These bits are only implemented in the CPU.

11 Cold Reset Slot 2

10 Cold Reset Slot 1

9 Cold Reset Slot 0

8 Warm Reset Slot 3

Writing to set one of these bits sends a warm reset pulse to the board in the corresponding slot. These bits return 0 when read. These bits are only implemented in the CPU.

7 Warm Reset Slot 2

6 Warm Reset Slot 1

5 Warm Reset Slot 0

4 Software initiated Warm Reset

When reading this register, this bit =1 indicates that the board has received a software generated warm reset since this bit was last cleared. Writing a 00000084x to this register generates a warm reset pulse on the board and sets this bit. Writing 00000004x to this register clears this bit. The activation of either of the cold reset bits also clears this bit.

3 Reset line initiated Warm Reset

When reading this register, bit 3 =1 indicates that the board has received a warm reset from the reset line since this bit was last cleared. This bit is cleared by writing 00000003x to this register. The activation of either of the cold reset bits also clears this bit. This bit can also be set by writing 00000003x to this register.

2 Reset line initiated Cold Reset

When reading this register, bit 2 =1 indicates that the board has received a reset line initiated reset when the board was broken and therefore treated it as a cold reset. Writing 00000002x to this register clears this bit, as does a power up initiated or software initiated cold reset. When this bit is set by a reset line reset all other bits in this register are cleared. This bit can also be set by writing 00000002x to this register.

1 Software initiated Cold Reset

When reading this register, bit 1 =1 indicates that the board has received a software generated cold reset since this bit was last cleared. Writing a 00000081x to this register generates a cold reset pulse on the board, sets this bit, and clears all other bits in this register. Writing 00000001x to this register clears this bit, as does a power up initiated cold reset. This bit is NOT cleared by any warm reset.

0 Cold Reset due to Power up

When reading this register, bit 0 =1 indicates that the board has received a power-up generated cold reset since this bit was last cleared. This bit is cleared by writing 00000000x to this register or by a software initiated cold reset. When this bit is set by a power up reset all other bits in this register are cleared. This bit is NOT cleared by any warm reset.

12.3.4 Bus Interface State

Offset [22:0]: 7FFFE0

Type: Read/Set/Clear
Present on all Xbus boards
Cold and warm reset affect this register as documented below
Compatibility: subset/super-set

This is a Read/Set/Clear register; commands are performed by writing to this register with data bits [4:0] pointing to a bit to set or clear and bit 7 determining whether the bit is set (=1) or cleared (=0). Reading this register returns 32-bits.

Note that since this register always returns 1s in bit positions [3:1], reading this register will always produce a non-zero result assuming the board is not broken. Stratabus boards used the ID PROM for this purpose. It is recommended that this register be used for that purpose on Xbus systems due to the large latency of accessing the IDPROM,.

31:23 These read bits/write encodings are reserved for additional board specific purposes. These bit positions always return 0's when read unless assigned a board specific purpose.

22 Force Peer-to-Peer Mode - When this bit is set the operations in the second column of table 10, Peer to Peer and Non Peer-to-Peer Operations, will be turned into peer-to-peer cycle. Refer to section 6.8 for details. Warm or cold reset clears this bit.

21 Break PCIB board on backplane failure

When a failure on the backplane brings down a bus, it is necessary to break one of the two boards connected to it. The default is to break the CPU. When this bit is set, the PCIB board is the one designated for breaking. This Polo feature is described in full detail in section 6.4, Bus Errors, on page 44. This bit is cleared by cold reset.

20 Enhanced EFQ/RWQ Nesting Disable - this mode is not implemented in Polo (Polo doesn't support enhanced nesting) - writes are ignored, reads return zeros.

19 EFQ Freeze State

This bit only exists on CPU boards. When set, the board busies any Xbus transaction decoded for the Cyclops EFQ (Eviction Flush Queue) EXCEPT if it is a data return of any size. When read, this bit will not appear set until the Cyclops's EFQ is empty or is currently being emptied onto the Ibus. It is set in the following cases:

Set when an Xbus or local write of 00000093x is performed AND the board's Freeze State bit is NOT set.

Set when an Xbus or local write of 00000093x is performed AND the board's Freeze State bit is set AND all Cyclops and Cougar incoming R/W pipes are empty.

This bit is cleared by writing 00000013x, or by warm or cold reset. It is not cleared as a result of the board going broken.

18 Break Even Board on TA Failure

This bit only exists on duplexable boards. Setting this bit causes the even board to break when a TA failure is detected; otherwise the odd board breaks. Software should set this bit so that the most recently added board of a pair is the one that breaks during a TA failure. The bit is set by writing 00000092x and cleared by writing 00000012x to this register. Warm reset has no effect on this bit, cold reset clears it. This bit always returns 0 on those boards which do not implement it.

17 Simplexed mode - this mode is not implemented in Polo - writes are ignored, reads return zeros.

16 Nested OCU Op Disable - this mode is not implemented in Polo (Polo doesn't support nested OCU operations) - writes are ignored, reads return zeros.

15 On-line For Dumping

This bit only exists on some boards which have system memory. Setting this bit permits the memory from an otherwise off-line board to be read. The bit is set by writing 0000008Fx and cleared by writing 0000000Fx to this register. Cold and Warm reset clear this bit. This bit always returns 0 on those boards which do not implement it.

14 Freeze State

This bit only exists on some boards which can be duplexed. On CPU boards, it is used in conjunction with the EFQ Freeze Mode bit. As outlined in detail in table 15 on page 72 in the Xbus Functional Specification, writing to set this bit (0000008Ex) causes a board to busy all accesses directed to it except those from its partner until the board passes through the sync point. Writing 0000000Ex, Warm and cold reset cold reset clear this bit. This bit always returns 0 on those boards which do not implement it.

13 Update Mode

This bit only exists on boards which have system memory and lock cycles. As outlined in detail in table 15 on page 72 in the Xbus Functional Specification, when set this bit indicates that the board is the "update" mode, a mode used during the board synchronization procedure to update memory on a new board. When in this mode the on-line board sends its otherwise local writes to the Xbus and the new partner board (in the off-line/update mode) updates its memory from these writes. Writing to set this bit (0000008Dx) puts a board in update mode. Writing 0000000Dx, Warm and cold reset cold reset clear this bit. This bit always returns 0 on those boards which do not implement it.

12 Any CPU On-line

This bit indicates that some CPU board in the system is on-line. It reflects the state of the cpu_online backplane signal and exists on all cpu/memory boards in the system. This bit always returns 0's on IO boards. This bit is cleared whenever no CPUs are on-line (e.g. a single on-line CPU going off-line or a by a system-wide cold reset that brings all boards off-line). Writing to set (0000008Cx) or clear (0000000Cx) this bit has no effect.

11 First CPU On-line

This bit exists on CPU boards only and is set if this board becomes the first in a system after a cold reset to gain mastership by winning an arbitration cycle and being the first to pass through the sync point. This bit is cleared by both cold reset warm reset. This bit always returns 0 on non-CPU boards.

10 Leave Sync Point Jump Switch

Writing to set ant one of these bits (00000086x, 00000087x, 00000088x, 00000089x, or 0000008Ax respectively) clears the other bits and instructs the board what state it should leave the sync point in. Regardless of the state of these bits, any CPU board which is the first on-line will leave the sync point simplex and clear all these bits. Except for the first on-line case, if none of these bits are set, a board will not leave the sync point.

These bits can be cleared by writing 00000006x, 00000007x, 00000008x, 00000009x, or 0000000Ax respectively. Warm reset has no effect on these bits, cold reset clears them. Bits 7 and 8 exist only on boards which can be duplexed and always returns 0s on all other boards, bit 9 exists only on boards with system memory that can be put on-line as just memory boards and always returns 0s on other boards, bit 10 exists only on boards which can "jump switch" and always returns 0s on all other boards.

A board reaching the sync point with bit 6 (Leave Sync Point Simplex) set will leave the sync point simplex. Bits 7 and 8 (Leave Sync Point Fast Duplex and Leave Sync Point Full Duplex) functionally are the same. A board will leave the sync point in a duplexed state only if its partner is also at the sync point with one of these bits set. Two bits exist to provide software a differentiation between a "fast" duplex (one where two initialize but off-line boards are duplexed) and a "full" duplex (one where an off-line board is being synchronized to an on-line board). Bit 9 (Leave Sync Point On-line For Dumping) is for a board to leave the sync point with only its memory on-line, say for a memory dump. Note the Xbus architecture puts the system at risk of crashing when any simplex system memory is utilized. Bit 10 (Leave Sync Point Jump Switch) is to support the jump switch feature wherein the on-line board goes off-line and is replaced by the previously off-line board.

Fore more information on these bits and there effect on board state see the Board Sync Register on page 93.

9 Leave Sync. point on line for dumping - not implemented in Polo systems.

8 Leave Sync Point Full Duplex

7 Leave Sync Point Fast Duplex

6 Leave Sync. point simplex - not implemented in Polo systems.

5 Duplexed

This bit only exists on duplexable boards. As outlined in detail in table 15 on page 72 in the Xbus Functional Specification, when set this bit indicates that the board and its partner are in lockstep. This bit is set as a result of the boards leaving the sync point in lockstep. The bit is cleared by cold and warm board resets, T/A failure, or by a board or its partner going Off-line or Broken. Writing this bit (00000085x or 00000005x) has no effect. This bit always returns 0s on no-duplexable boards.

4 On-line

This bit equals 1 if the board is on-line. As outlined in detail in table 15 on page 72 in the Xbus Functional Specification, an on-line board is fully functioning from the system perspective. Among other things, this means the board responds to all bus transactions directed toward it, including paired reads. Writing to set this bit (00000084x) or passing though the sync point (in some cases) brings a board on-line. Writing to clear this bit (00000004x) and the board breaking bring it off-line. Both warm reset and cold reset clear this bit.

3 Obey Both

The Xbus changes obeys based on the source of a transaction, not in response to errors. There is no need to set the obey state via software control. To maintain backwards compatibility with the Xbus, writes to the obey bits have no effect and reads always return 1.

2 Obey B - not implemented, returns 1 on reads.

1 Obey A - not implemented, returns 1 on reads.

0 Broken

This is a status bit used to indicate a hardware fault which has caused the two sides of the board to behave differently. This bit is set automatically by hardware or by the writing 00000080x to this register. This bit, as well as the diagnostic broken bit in the Test Control register, are cleared at the end of a cold or warm reset if the two sides are equal or by writing 00000000x to this register. Resets are the preferred method for unbreaking a board. When this bit is set the board cannot drive any external, (Xbus or C-connector), signals. The board will still respond to writes addressed to it in I/O space, (if capable), and will continue to check data and address information which would normally be transmitted to the bus. Whenever this bit is set by hardware a double bus error will be issued to the bus to abort any potentially bad information that may have been driven by this board. The act of clearing this bit generates a maintenance interrupt.

12.3.5 Board Sync Register

Type: Read Only **Offset [22:0]: 7FFFD8**
Present only on some duplexable Xbus boards
Cold and warm reset do not directly affect this register
Compatibility: exact

WARNING: **Reading this register may return non-deterministic data and can be different side-to-side or board-to-board; it is important to discard whatever data is returned as soon as possible. Data left in registers can be pushed onto stack frames thus causing system crashes if different.**

A board reads this register to pass through the sync point. This data return may be delayed by hardware and may have side effects. Reads of this register locally will cause the board to wait at the sync point. Reads to this register from Xbus IO will return 0s immediately and no sync procedure will take place.

It is possible to perform non-local IO accesses to/from a board waiting at the sync point.

The specific actions resulting from a board passing through the sync point (i.e. reading this register locally) depend on the state of the board and system involved:

1 **First CPU On-line:** The first CPU board in a system to try to read the sync register (or a the first CPU to win arbitration should multiple CPUs simultaneously attempt this) will pass through the sync point (i.e. have data returned to it when reading this register) and go on-line in a simplex state. Hardware then causes that board to assert the cpu_online backplane signal, thereby notifying other CPU boards that one CPU board has passed through the sync point.

Should the cpu_online backplane signal deactivate (say by the only on-line CPU breaking), another CPU board sitting at the sync point will be released in a similar fashion.

Note that a board passing through the sync point in this fashion will leave the sync point in a simplexed fashion regardless of the state of the "Leave Sync Point Simplex", "Leave Sync Point Fast Duplex", "Leave Sync Point Full Duplex", "Leave Sync Point On-line For Dumping", and "Leave Sync Point Jump Switch" bits in the Bus Interface State register. Should any of those bits have been set, they are cleared in this case, thereby providing a mechanism for software to detect that the sync point was passed through as first cpu on-line.

2 **Duplexed:** Should a board's "Leave Sync Point Full Duplexed" or "Leave Sync Point Fast Duplexed" bit in its Bus Interface State register be set, it will leave the sync point in lockstep with its partner when its partner reaches the sync point (assuming the partner's not already there) and a read is performed of the sync register and that board's "Leave Sync Point Full Duplexed" or "Leave Sync Point Fast Duplexed" bits are set. If a board is waiting at the sync point and its partner breaks, it will leave the sync point On-line/Simplexed. (Actually, the duplexed bit will be set and then cleared immediately when the hardware detects the partner is not on-line). Any other condition and the board will remain stalled at the sync point.

A pair of boards can pass through the sync point duplexed when both are new to the system (a "Fast Duplex") or when one board is already part of the system and a new board is being synchronized to it (a "Full duplex"). In the former case, the boards were both in the off-line state prior to passing through the sync point (but there was another CPU on-line, allowing the boards to stall at the sync point), in the latter case the "old" board was in the On-line/Update/Freeze state and the "new" board was in the Off-line/Update/Freeze state.

It is possible to put a board on-line duplexed when a board is already waiting at the sync point. This is done by setting the "Leave Sync Point Full Duplexed or Leave Sync Point Fast Duplexed" bit on while the board is stalled at the sync point.

3 **On-line/Simplexed:** Should a board's "Leave Sync Point Simplexed" bit in its Bus Interface State register be set, it will leave the sync point, on-line but never duplexed. If it's partner is on-line, It will stall until its partner is off-line. Note, If a board and its partner both perform a go-on-line-simplex at approximately the same time they will both go on-line but not duplexed. lockstep is not guaranteed after leaving the sync point. This condition is not supported on duplexable boards and Xbus transactions will conflict with each other because they will both be using the same TRID.

If a board's partner is at the sync point with a the "Leave Sync Point Full Duplexed" or "Leave Sync Point Fast Duplexed" or "Leave Sync Point Jump Switch" bit set, the board will pass through the sync point simplexed and not effect its partner in any way. (The partner will remain at the sync point).

It is possible to put a board on-line simplexed when a board is already waiting at the sync point. This is done by setting the "Leave Sync Point Simplexed" bit on while the board is stalled at the sync point.

4 **On-line For Dumping:** Should a board's "Leave Sync Point On-line For Dumping" bit in its Bus Interface State register be set, it will leave the sync point immediately with its memory on-line but otherwise simplex and off-line. It is planned that this mode be used to support memory dumps. A board and its partner can be both "on-line for dumping" but Xbus transactions initiated by these boards are not supported

It is possible to put a board On-line For Dumping when a board is already waiting at the sync point. This is done by setting the "Leave Sync Point On-line For Dumping" bit on while the board is stalled at the sync point.

Jump Switch: Should a board's "Leave Sync Point Jump Switch" bit in its Bus Interface State register be set, it will either leave the sync point on-line and simplex or off-line, depending on whether it was on-line or off-line prior to passing through the sync point. (If it was on-line it leaves off-line and if it was off-line, it leaves on-line.) It is planned that this mode be used to support on-line upgrade of board with different processor speeds or memory sizes.

12.3.6 LED Control

Type: Read/Write **Offset [22:0]: 7FFFD0**
Present on all Xbus boards
Cold and warm reset affect this register as documented below
Compatibility: exact

This register contains status/control of the LEDs on the board. There are three LEDs (red, yellow, green) arranged like a stoplight on each board handle. The following table indicates their states (disks and dumb devices are included for completeness). For more information on the LEDs, their purpose and software model, see the appropriate sections of the maintenance and diagnostic specification.

Table 16. LED States

Unit State	Red LED Service	Yellow LED Do not Pull	Green LED In Operation
No Power	OFF	OFF	OFF
Testing	Cycle	Cycle	Cycle
Simplexed	OFF	ON	ON
Broken	ON	OFF	OFF
Duplexed	OFF	OFF	ON
Lamp Test	ON	ON	ON
Disk Drive Inserted	OFF	OFF	BLINK
Disk Drive Spun Down	OFF	OFF	OFF
Dumb device OK	OFF	N/A	ON
Dumb Device Faulted	ON	N/A	OFF
Dumb Device No Powers	OFF	N/A	OFF

31:3 Reserved

These bits are reserved for future use. Writes to the register will have no effect. Reads of this register will return zeros in these bits.

2 Red LED State

The red LED is turned on by hardware whenever the board is in the broken state. When the board is not broken, writing this bit can turn on (write a 1) or off (write a 0) the LED. Note that hardware will not clear the LED when the board transitions from broken to not broken; that must be done by software. The red LED is set by warm and cold reset.

2 Yellow LED State

The yellow LED on is used to indicate that a board may not be removed. It is transition-driven: on duplexable boards, it is turned on by hardware when the board transitions from off-line to on-line-simplexed, or from on-line-duplexed to on-line-simplexed (due to its partner breaking). It is turned off by hardware on the transition to duplexed or broken; it is not cleared by the on-line->off-line transition. On non-duplexable boards, the LED is set when the go on-line command is issued. The yellow LED may be set or cleared by software at any time. The yellow LED is cleared by warm and cold reset.

0 Green LED State

The green LED is turned on by hardware when the go on-line command is issued, and is turned off when the board goes off-line or broken. These actions are state transition driven. Only the transition causes hardware to set or clear the light. At any time, the software may also set or clear the green led. The green LED is cleared by warm and cold reset.

12.3.7 Slot ID

Type: Read Only **Offset [22:0]: 7FFFC8**
Present on all Xbus boards
Cold and Warm Reset have no effect on this register
Compatibility: exact

This register contains the 4 bits that indicate the physical slot position of this board. This is the slot number address that this board will respond to for non-global I/O accesses. Note that this register is readable locally, when not duplexed (e.g. during diagnostics).

The 4 slot id bits are located in bits 3:0 of the slot id register. Bits 31:4 are unused and will return zeros when read.

12.3.8 Read Ping Interval

Type: Read/Write **Offset [22:0]: 7FFFC0**
Present on all Xbus based boards that can issue I/O reads
Cold reset clears this register, warm reset has no effect
Compatibility: exact

31:1 Reserved

0 Read Ping Interval

The value loaded into this bit determines the amount of time a read will be outstanding before a "Ping" bus transaction is sent on the Xbus to determine if there is still a target that intends to respond to the request. Writing a one sets the read ping interval to 31 bus phases, or 2.667 microseconds. This value is provided for diagnostic and simulation use only. Writing a zero sets the interval to 99.416 microseconds, or 1,193 bus phases (the default). Cold reset clears this register; in normal operation there is no need to touch this register.

This register is not implemented on PCIB boards (since they cannot issue I/O reads). When a read request is committed on the Xbus, i.e. the request is not terminated with BUSY or ERROR, a time-out counter which was pre-loaded with either 31 or 1,193 is enabled. There is a single counter that times the duration of the oldest request from a board that is outstanding; if a response is not received before this counter reaches terminal count, a ping transaction is sent on the bus. Should no acknowledge to the ping be received, the bus interface returns zeros and an error indicator to the requestor. This error indication is a low priority machine check on the CPU board. If an acknowledge to the ping operation is received, the time-out counter is reloaded according to the contents of this register and the timeout count is restarted. If a response is returned before the counter reaches terminal count, data is returned in the usual fashion.

A read of this register provides the most recently written data. Bits 31:1 are unused and will return zeros when read.

12.3.9 General Purpose Communications [7:0]

Type: Read/Write **Offset [22:0]: 7FFF98:7FFF60**
At least 4 present on all Xbus based boards, up to 8 on some.
Cold and warm reset affect these registers as documented below
Compatibility: exact

These registers are provided for any general communication desired by any resource on or off the board. Because boards may have different needs that are not defined at this time, eight General Purpose Communication registers are provided in IO space. Not all 8 registers are necessarily implemented on all bus ASICs, but each board is required to have at least the first four (3:0).

Table 17. General Purpose Registers

Name	Offset [22:0]	Cold Reset	Warm Reset
GPR0	7FFF60	cleared	cleared
GPR1	7FFF68	cleared	no change
GPR2	7FFF70	cleared	no change
GPR3	7FFF78	cleared	no change
GPR4:GPR7 (unimplemented)	7FFF80:7FFF98	cleared	no change

12.3.10 Memory Size/Location

Type: Read/Write **Offset [22:0]: 7FFF58**
Present only on Boards with Global System Memory
Cold reset clears this register, warm reset has no effect
Compatibility: implemented for software compatibility only

31:8 reserved

Writes have no effect; reads return zeros.

7:0 These bits are implemented for software compatibility with Jetta only. The bits are read/write but have no effect on operation.

12.3.11 Test Control

Type: Read/Set/Clear **Offset [22:0]: 7FFF48**

Present on all Xbus boards
Cold reset clears this register, warm reset has no effect
Compatibility: subset

This is a Read/Set/Clear register, commands are performed by writing to this register with data bits [4:0] pointing to a bit to set or clear and bit 7 determining whether the bit is set (=1) or cleared (=0). Reading this register returns 32-bits.

31:7 Reserved.

6 D-side Difference Bit

When the D-side Difference Enable is set, reading this bit returns a 1 from the D side hardware, and a 0 from the C side. This C-D difference can be used for testing comparators internal to the board. This bit is cleared by cold reset.

5 D-side Difference Enable

Setting this bit enables the D-side difference bit. When it is enabled, the D-side difference bit becomes a one on the D side bus ASIC, and remains a 0 in the C-side ASIC. This bit is cleared by cold reset.

WARNING: Setting this bit and then reading this register will return different data on C vs. D sides of the board (via bit 6 below). This will break you if not read locally.

4 Diagnostic_Broken

This bit provided to facilitate self-test. This bit is set whenever a condition that would set the Broken bit of the Bus Interface State register occurs. However, it may be cleared independently of the Broken Bit by writing 0000004x to this register. This permits testing of various board features while the board remains Broken, preventing the testing from polluting the Golfbus. This bit is also cleared by cold reset. Warm reset sets this bit, because it causes the board to go broken for the duration of the reset. Writing 00000084x (which normally would set a bit) have no effect on this register. Clearing Broken (by writing to the Bus Interface State register) does not clear Diagnostic Broken.

Setting this bit enables the D-side difference bit. When it is enabled, the D-side difference bit becomes a one on the D side bus ASIC, and remains a 0 in the C-side ASIC. This bit is cleared by cold reset.

3:0 Reserved.

12.3.12 Bus Interface Fault Reporting

Offset [22:0]: 7FFF38

Type: Read/Set/Clear
Present on all Xbus boards
Bit 1 set by cold and warm reset;
for other bits cold reset clears this bit, warm reset has no effect
CD Different: read via unpaired space
Compatibility: incompatible

This register controls the error latching and reporting around bus faults. It also is used to rearm the broken registers. There are five error registers: Bus Info Error Status, Misc. Error Status, Bus Error Byte Status, Control Bus Error Status, and Voter Error Transceiver Status. All are latched when an error is detected in the system. This register provides a means of disabling certain faults from causing the "Error Latching" to occur, as a way of allowing new faults to be detected in the presence of known, continuous faults. Maintenance interrupts are generated on the latching of a fault; if a fault is disabled from causing error latching, it is also disabled from causing a maintenance interrupt. An overview of the information supplied and how it is intended to be used is provided in section 9.4, Error Reporting, on page 98.

Note that this register is both Read/Set/Clear and CD different. A single write to set or clear a bit affects the C and D side bits, thus to set bits 3,11,19, and 27, it is only necessary to write 00000083x. It is not possible to write only the C or D side.

These bits indicate whether the various classes of faults cause the Error Registers to record state ("Error Latching") and generate a maintenance interrupt. The default is all zeros, i.e. any fault causes a maintenance interrupt and all of the Error Registers to Update. Error Latched on Thermal Faults Disabled suppresses Error Latching and the resulting maintenance interrupt due to Thermal faults. However, if another class of fault occurs and a thermal fault is present, the thermal fault will be present in the updated error register. Likewise, Error Latched on Incoming 3-Way Vote Fault Disabled, Error Latched on Control Bus Fault Disabled, and Error Latched on Clock Faults Disabled determine whether 3-way voter, Control bus errors, or clock faults trigger Error Latching and maintenance interrupts.

*Error Latched on TA Problem Disabled (bits 5,21,13,29) is named "Error Latched on non-OBEYed Bus Fault Disabled" on Golfbus systems. In Golfbus systems, this bit has two functions: disabling error latching due to errors on the non-obeyed bus, <u>and</u> due to TA problem. In the Xbus system, only the TA problem functionality is retained, so the name of the bit has been changed for clarity.

31,15 D side Error Latched on Control Bus Faults Disabled

30,14 D side Error Latched on Clock Faults Disabled - CPU only

Clock faults only occur on CPU boards.

29,13 D side Error Latched on TA Problem Disabled* - CPU only

TA faults only occur on CPU boards.

28,12 D side Error Latched on Incoming 3-Way Vote Fault Disabled

27,11 D side Error Latched on Thermal Faults Disabled - CPU only

Thermal faults only occur on CPU boards.

26,10 D side Board Logic Maint Int

This bit being set indicates that a maintenance interrupt was issued by the logic (excluding the other bus interface) on this board. The appropriate on board register(s) should be examined to determine the source of the interrupt. This bit is cleared by cold reset or by writing to clear this bit (00000002x). This bit may be set (and a maintenance interrupt generated) by writing 00000082x. **Note: For Gambit ASIC, whenever the Gambit specific logic sets this bit, bits 18 and 2(C side Board Logic Maint int) also get set.**

25,9 D side Bus Interface Maint Int

This bit being set indicates that a maintenance interrupt was issued from this bus interface (including the clock chip and board logic maint int). Writing to set this bit causes a maintenance interrupt to be pulsed on the bus though the read bit in this register remains set until cleared. This bit is cleared by writing 00000001x to this register. Since both warm and cold reset break and then unbreak a board (events which generate maintenance interrupts), either warm or cold reset leave this bit set.

24,8 D side Error Data Recorded/Rearm Error and Broken Registers

When this bit =1, the following registers contain new information: Bus Info Error Status, Misc. Error Status, Bus Error Byte Status, Control Bus Error Status, and Voter Error Transceiver Status. Writing a 0 to this bit clears these registers and "rearms" them so that they will capture the state of the next applicable error condition. Clearing this bit also "rearms" the Common Broken Status and ASIC Specific Broken Status registers. Writing a 1 to this bit has no effect.

23,7 C side Error Latched on Control Bus Faults Disabled

22,6 C side Error Latched on Clock Faults Disabled - CPU only

21,5 C side Error Latched on TA Problem Disabled* - CPU only

20,4 C side Error Latched on Incoming 3-Way Vote Fault Disabled

19,3 C side Error Latched on Thermal Faults Disabled - CPU only

18,2 C side Board Logic Maint Int **Note: For Gambit ASIC, whenever the Gambit specific logic sets this bit, bits 26 and 10(D side Board Logic Maint int) also get set.**

17,1 C side Bus Interface Maint Int

16,0 C side Error Data Recorded/Rearm Error and Broken Registers

12.3.13 Common Broken Status

Offset [22:0]: 7FFF30

Type: Read Only
Present on all Xbus boards
Cold reset clears this register, warm reset has no effect
CD Different: read via unpaired space
Compatibility: incompatible

This register contains status bits from the broken logic comparators that are common to all Xbus boards. It is cleared by cold reset (unless the board leaves cold reset broken). The contents of this register are frozen when a board first goes broken, so that the only comparators that caused the broken condition are flagged. This is so that comparators that miscompare further down the line (after the error has rippled through the logic) do not obscure the cause of the original failure. This register is re-armed via the Re-arm Error registers bit in the Bus Interface Fault Reporting register.

31,15 D-side saw control bus set broken

30,14 D-side saw DLL out-of-lock

29,13 D-side saw drive miscompare error on a or b bus (i.e. A C to D side info out miscompare)

28,12 D-side saw C-side bus ASIC set broken

27,11 D-side board logic (external to the bus ASIC) generated broken

26,10 D-side slot parity error

25,9 D-side ASIC saw drive miscompare error on 3-way voted line.

24,8 D-side Software set broken by command

23,7 C-side saw control bus set broken

22,6 C-side saw DLL out-of-lock

21,5 C-side saw drive miscompare error on A or B bus (i.e. a C to D side info out miscompare)

20,4 C-side saw D-side bus ASIC set broken

19,3 C-side board logic (external to the bus ASIC) generated broken

18,2 C-side slot parity error

17,1 C-side ASIC saw drive miscompare error on 3-way voted line.

16,0	C-side Software set broken by command

12.3.14 ASIC Specific Broken Status

Type: Read Only **Offset [22:0]: 7FFF28**
Present on all Xbus boards
Cold reset clears this register, warm reset has no effect
CD Different: read via unpaired space
Compatibility: incompatible

This register contains status bits from the broken logic comparators that are specific to a particular Xbus ASIC. It is cleared by cold reset (unless the board leaves cold reset broken). The contents of this register are frozen when a board first goes broken, so that the only comparators that caused the broken condition are flagged. This is so that comparators that miscompare further down the line (after the error has rippled through the logic) do not obscure the cause of the original failure. This register is re-armed via the Re-arm Error registers bit in the Bus Interface Fault Reporting register.

CPU ASIC (Cyclops):

31,15	Reserved
30,14	D-side Fan broken
29,13	D-side Sable broken
28,12	D-side Cougar broken
27,11	D-side TA broken
26,10	D-side IM OK broken
25,9	D-side Arbitrary broken.
24,8	D-side Heuristic broken
23,7	Reserved
22,6	D-side Fan broken
21,5	C-side Sable broken
20,4	C-side Cougar broken
19,3	C-side TA broken
18,2	C-side IM OK broken
17,1	C-side Arbitrary broken.
16,0	C-side Heuristic broken

PCIB ASIC (Gambit):

31,15	Reserved
30,14	Reserved
29,13	Reserved
28,12	Reserved
27,11	D-side Gambit specific
26,10	D-side Xbus Parity generator fault
25,9	D-side Arbitrary broken.

24,8	D-side Heuristic broken
23:7	Reserved.
22:6	Reserved.
21:5	Reserved.
20:4	Reserved.
19,3	C-side Gambit specific
18,2	C-side Xbus Parity generator fault
17,1	C-side Arbitrary broken.
16,0	C-side Heuristic broken

12.3.15 Bus Info Error Status

Type: Read Only **Offset [22:0]: 7FFF20**
Present on all Xbus boards
Cold reset clears this register, warm reset has no effect
CD Different: read via unpaired space
Compatibility: subset

Any bus error signaled by the system, or a voter, control bus, thermal, or clock error on this board causes this register (and the other Error Latching registers) to freeze with state recorded for the offending info phase. This includes both non-fatal and fatal (i.e. board breaking) errors. All bits in this register correspond to the same info phase. Cold reset clears this register.

31,15	D-side some other board drove a the Bus B Error backplane signal and I did not.
30,14	reserved
29,13	D-side The bus error whose info was recorded was signaled by a different board
28,12	reserved
27,11	D-side ASIC saw parity error on B bus
26,10	D-side ASIC saw parity error on A bus
25,9	D-side ASIC saw loopback fault on the B bus on the info bits the D-side drove.
24,8	D-side ASIC saw loopback fault on the A bus on the info bits the D-side drove.
23,7	C-side some other board drove a the Bus A Error backplane signal and I did not.
22,6	reserved
21,5	C-side This board was bus master for the info phase recorded
20,4	reserved
19,3	C-side ASIC saw parity error on B bus
18,2	C-side ASIC saw parity error on A bus
17,1	C-side ASIC saw loopback fault on the B bus on the info bits the C-side drove.
16,0	C-side ASIC saw loopback fault on the A bus on the info bits the C-side drove.

This register reports the various bus and compare faults separately for the C side Bus ASIC and the D-side ASIC.

This register will not accurately reflect the state of the errors if the board is in the "funny" state. Since boards in the "funny" state do not enter the error sequence, it is not possible to track the errors.

All bits are updated upon the detection of an error and remain at the same state until the Error Data Recorded/Rearm Error Registers bit of the Bus Interface Fault Reporting Register is cleared.

12.3.16 Misc. Error Status

Type: Read Only **Offset [22:0]: 7FFF18**
Present on all Xbus boards
Cold reset clears all bits except the Thermal faults bits.
Thermal fault is unaffected by any resets.
Warm reset has no effect on this register.
CD Different: read via unpaired space
Compatibility: exact

The bit definitions for this register are identical to the Golfbus Specification definitions with one difference: thermal and clock faults in Polo only occur on CPU boards so the bits associated with those faults are CPU only.

28,12	D-side T/A problem (CPU/MEM board ONLY.)
27,11	D-side thermal fault bit 0 - under temperature (CPU/MEM board ONLY.)
26,10	D-side clock recovery chip status bit 1 (CPU/MEM board ONLY.)
25,9	D-side clock recovery chip status bit 0 (CPU/MEM board ONLY.)
24,8	D-side ASIC saw a three way voter failure
23,7	Reserved.
22,6	Reserved.
21,5	Reserved.
20,4	C-side T/A problem (CPU/MEM board ONLY.)
19,3	C-side thermal fault bit 1 - over temperature (CPU/MEM board ONLY.)
18,2	C-side clock recovery chip status bit 1 (CPU/MEM board ONLY.)
17,1	C-side clock recovery chip status bit 0 (CPU/MEM board ONLY.)
16,0	C-side ASIC saw a three way voter failure

There are two clock recovery chips on every board; each recovery chip has two pins which indicate the status of the Backplane fault tolerant clock. The various combinations of these status bits are indicated in table 18

The clock status bits will change off the falling edge of 24 MHz, therefore it is necessary for the Bus Interface ASIC's to latch the status bits on the rising edge of 24 Mhz (phase12_3) to insure proper setup and hold time.

Table 18. Clock Status Definition

Clk Status<1:0>	Definition
00	no fault
01	Clock recovery chip detected a CLK A fault
10	Clock recovery chip detected a CLK B fault
11	Clock recovery chip detected a CLK C fault

Any change on a Thermal fault bit results in a maintenance interrupt (unless disabled) and freezes all registers. These two bits are from the thermal sensor on the board.

bit[0] Represents the under temperature pin(6) on the thermal sensor. (etch run to D ASIC.)

bit[1] Represents the over temperature pin(7) on the thermal sensor. (etch run to C ASIC.)

Table 19. Thermal fault Status Definition

Thermal Fault <1:0>	Definition
01	Board is within recommended operating temperature.
00	Board is warm, above recommended operating temperature.
10	Board is hot, customer should consider a shutdown.
11	Bad board hardware is causing this condition (illegal).

The T/A problem bit (Cyclops only bit) when set, represents that there is a non-fatal problem in the T/A Hardware. The board or backplane could be at fault. This condition should be cleared up. If left un-fixed, REAL T/A problems will not be handled correctly. This bit is masked off by the "Error Latched on non-OBEYed Bus Fault Disabled" bit.

12.3.17 Control Bus Error Status

Type: Read Only **Offset [22:0]: 7FFF10**
Present on all Xbus boards
Cold reset clears this register, warm reset has no effect
CD Different: read via unpaired space
Compatibility: New Register

This register records the control bus error status to aid in fault isolation of a system experiencing single or multiple bit errors on the control buses. All bits are updated upon any bus error signaled by the system, or a voter, control bus, thermal, or clock error on this board, and remain at the same state until the Error Data Recorded/Rearm Error Registers bit of the Bus Interface Fault Reporting register is cleared. Cold reset clears these registers.

The bits are set for both single bit (correctable) and double bit (detectable, causes board broken) errors. Check the Common Broken Status register to see if the error caused board broken.

31,15 Reserved

30,14	Reserved
29,13	D-side saw error on control_out_p
28,12	D-side saw error on control_in_p
27,11	D-side saw error on control_out_o
26,10	D-side saw error on control_in_o
25,9	D-side saw error on control_out_n
24,8	D-side saw error on control_in_n
23,7	reserved
22,7	C-side arbitration out of sync with D-side. This bit is set when the C side checking logic sees an unexpected value on the bus_req control line driven by the D-side.
21,5	C-side saw error on control_out_p
20,4	C-side saw error on control_in_p
19,3	C-side saw error on control_out_o
18,2	C-side saw error on control_in_o
17,1	C-side saw error on control_out_n
16,0	C-side saw error on control_in_n

12.3.18 Bus Error Byte Status

Type: Read Only **Offset [22:0]: 7FFF08**
Present on all Xbus boards
Cold reset clears this register, warm reset has no effect
CD Different: read via unpaired space
Compatibility: usage change

This register records the bus error status on a per-byte basis to facilitate debug. All bits are updated upon any bus error signaled by the system, or a voter, control bus, thermal, or clock error on this board, and remain at the same state until the Error Data Recorded/Rearm Error Registers bit of the Bus Interface Fault Reporting register is cleared. Cold reset clears these registers.

The bits are interpreted as follows: If the board was the bus master (indicated in the Bus Info Error Status register) at the time of the error, this register indicates which bytes had a loopback error. If the board was not a bus master at the time of the bus error, these bits will be zero.

31,14	Reserved
30,14	Reserved
29,13	D-side saw error on trid or func_op
28,12	D-side saw error on parity
27,11	D-side saw error on info[31:24]
26,10	D-side saw error on info[23:16]
26,9	D-side saw error on info[15:8]
24,8	D-side saw error on info[7:0]

23,7	Reserved
22,6	Reserved
21,5	C-side saw error on trid or func_op
20,4	C-side saw error on parity
19,3	C-side saw error on info[31:24]
18,2	C-side saw error on info[23:16]
17,1	C-side saw error on info[15:8]
16,0	C-side saw error on info[7:0]

12.3.19 Voter Error Transceiver Status

Type: Read Only **Offset [22:0]: 7FFF00**
Present on all Xbus boards
Cold reset clears this register, warm reset has no effect
CD Different: read via unpaired space
Compatibility: incompatible

These register records the status of the 3-way voting logic. All bits are updated upon any bus error signaled by the system, or a voter, thermal, or clock error on this board, and remain at the same state until the register is rearmed by the Error Data Recorded/Rearm Error Registers bit of the Bus Interface Fault Reporting register. Error latching and maintenance interrupts due to voter errors can be suppressed by the Error Latched on Incoming 3-Way Vote Fault Disabled bit in the Bus Interface Fault Reporting register.

For a list of which signals go through each transceiver, please see Xbus Voted Signal Partitioning on page 96.

CPU ASIC (Cyclops):

31,15	Reserved, read as zero.
30,14	Reserved, read as zero.
29,13	D-side saw error on online_in
28,12	D-side saw error on reset
11, 27	D-side saw error on sync_in
26,10	D-side saw error on board_not_broken_p
25,9	D-side saw error on board_not_broken_o
24,8	D-side saw error on board_not_broken_n
23,7	Reserved, read as zero.
22,6	Reserved, read as zero.
21,5	C-side saw error on online_in
20,4	C-side saw error on reset
19,3	C-side saw error on sync_in
18,2	C-side saw error on board_not_broken_p ⟵
17,1	C-side saw error on board_not_broken_o

16,0	C-side saw error on board_not_broken_n

PCIB ASIC (Gambit):

31,15	Reserved, read as zero.
28,12	D-side saw error on reset_o
27,11	D-side saw error on reset_n
26,10	D-side saw error on board_not_broken_p
25,9	D-side saw error on board_not_broken_o
24,8	D-side saw error on board_not_broken_n
23,7	Reserved, read as zero.
22,6	Reserved, read as zero.
21,5	Reserved, read as zero.
20,4	C-side saw error on reset_o
19,3	C-side saw error on reset_n
18,2	C-side saw error on board_not_broken_p
17,1	C-side saw error on board_not_broken_o
16,0	C-side saw error on board_not_broken_n

12.3.20 Bus ASIC Chip Revision

Type: Read Only **Offset [22:0]: 7FFEF8**
Present on all Xbus boards
Cold and Warm reset have no effect
Compatibility: new values

This register returns a revision number of the ASIC. Note: this information should also be available from the ID PROM; it is included here to simplify tracking parts in the lab when the ID PROM is not stable.

Table 20. Bus ASIC Revision Numbers

Board	chip	Bus ASIC Chip Revision	VLSI part number
CPU	Cyclops 1st pass	00002???	VY12???
PCIB	Gambit 1st pass	00002???	VY12???

12.3.21 Performance Counter

Type: Read/Write **Offset [22:0]: 7FFEF0**
Present on CPU/MEM boards
Cold and warm reset Clear this register.
Compatibility: exact

A read of this register will return the value of this 32 bit performance counter. The counter may be set to any value by writing to this register. This counter will increment every time the trigger condition (as defined in perf_counter_trig, perf_counter_mask, perf_counter_trig_not_equal and perf_counter_trig_enable) is encountered. This counter will never increment past the value of 0FFFFFFFFx. If the counter were set to zero and the trigger condition was set to always increment, the counter would reach the value of 0FFFFFFFFx in 358.1 seconds (5.97 minutes). This register may be read and written while the counter is enabled.

12.3.22 Performance Counter Trig[1:0]

Offset [22:0]: 7FFEE8:7FFEE0

Type: Read/Write
Present on CPU/MEM boards
Cold and warm reset clear these registers.
Compatibility: exact

This register defines the trigger condition that will increment the performance counter. A trigger condition is evaluated every 83.381361ns (one ~12MHZ Xbus phase). All Evaluations occur using data that is "lined up" with the given Xbus bus operation. That is, function codes, physical address, busy, error, ACK (etc.) are all delayed the correct amount so that all the information presented to the trigger circuitry have meaning for that Xbus operation. Any bit may be masked to a don't care using the perf_counter_mask registers (below).

See Perf_Counter_Mask for bit definitions.

12.3.23 Performance Counter Mask[1:0]

Offset [22:0]: 7FFED8:7FFED0

Type: Read/Write
Present on CPU/MEM boards
Cold and warm reset clear these registers.
Compatibility: exact

This register works in conjunction with the perf_counter_trig register above. A bit that is a one '1' in this register signifies you "don't care" to evaluate this bit as part of the trigger. A bit that is a zero '0' in this register signifies you wish to evaluate this bit as part of the trigger.

Perf_Counter_Trig[0]: (7FFEE0) OR Perf_Counter_Mask[0]: (7FFED0)

31:26	Reserved.
26	xbus_error: Some board drove A or B bus error for this Xbus operation or this operation was aborted due to a previous bus error. All above data could be wrong/invalid.
25	my_busy: This board drove busy for this Xbus operation.
24	xbus_busy: Some board drove busy for this Xbus operation.
23	my_ack: This board drove ACK for this Xbus operation.
22	xbus_ack: Some board drove ACK for this Xbus operation.
21	i_drove_bus: This board was bus master for this Xbus operation.
20	first_op: This is the first operation of a block transfer.
19	func_op: The info bus contains a valid function code, byte enables, RC and physical address, else it is data.
18:12	trid[6:0]: The 7 bit trid used on this bus operation.
11:6	function_code [5:0]: Needs qualification with func_op (bit 19).
5:4	remote coherent bits [1:0]: Needs qualification with func_op (bit 19).
3:0	byte_enables [3:0]: Needs qualification with func_op (bit 19).

Perf_Counter_Trig[1]: (7FFEE8) OR Perf_Counter_Mask[1]: (7FFED8)

31:0 physical_address[31:00]: Needs qualification with func_op.

12.3.24 Performance Counter Control

Type: Read/Set/Clear **Offset [22:0]: 7FFEC8**
Present on CPU/MEM boards
Cold and warm reset clears this register
Compatibility: exact

This is a Read/Set/Clear register, commands are performed by writing to this register with data bits [4:0] pointing to a bit to set or clear and bit 7 determining whether the bit is set (=1) or cleared (=0). Reading this register returns 32-bits.

31:2 Reserved.

1 Perf_Counter_Trig_(on)_Not_Equal

This bit returns a 1 if a board's performance counters are incrementing when the trigger condition is not equal, otherwise it returns a 0. Writing to set this bit (00000081x) will cause the counter trigger circuitry to increment the perf_counter whenever the trigger condition (as defined in perf_counter_trig, perf_counter_mask) is NOT seen. Writing to clear this bit (0000001x) will cause the perf_counter to increment when the trigger condition is seen. This bit is cleared on cold and warm reset.

0 Perf_Counter_Enable

This bit returns a 1 if a board's performance counter is enabled, otherwise it returns a 0. Writing to set this bit (00000080x) will enable the performance counters and the performance counters will increment every time the trigger condition (as defined in perf_counter_trig, perf_counter_mask and perf_counter_trig_not_equal) is encountered. Writing to clear this bit (0000000x) disables any incrementing of the counter. This bit is cleared on cold and warm reset.

12.3.25 Fault Bit[1:0]

Type: Read/Write **Offset [22:0]: 7FFEC0**
Present on all Xbus boards **7FFEB8**
Cold reset clears this register, warm reset has no effect
Compatibility: new register

This register is used to generate errors on the Xbus or local to the board depending on the setting of the Error Control register. Each bit in these registers corresponds to a bit on the info bus and the control bus. Setting that bit in the register will cause the corresponding bit on the bus to be incorrect (inverted). The timing of when the bit is incorrect is based on the Error Control register setting.

Fault Bit[0]: (7FFEC0)

31:0 Fault corresponding bit on info bus 31:0 for the first half of the phase

Fault Bit[1]: (7FFEB8)

24 Fault parity for the first half of the phase

23 Fault func_op for the first half of the phase

22:16 Fault trid[7:0] for the first half of the phase

15:12 Fault corresponding to checkbits 7:4 for second half of the phase

11 Fault unused control[3]

10	Fault busy
9	Fault maint_int
8	Fault ACK
7:4	Fault corresponding to checkbits 7:4 for first half of the phase
3	Fault bus_err_b
2	Fault bus_err_a
1	Fault grant_inh
0	Fault bus_req

12.3.26 Data Match

Type: Read/Write **Offset [22:0]: 7FFEB0**
Present on all Xbus boards
Cold reset clears this register, warm reset has no effect
Compatibility: new register

This register is used for the data matching capability of the error generation. This allows the user to set a trigger and the error will not occur until a match is seen on the incoming info bus.

31:0	Data pattern to match against incoming info 31:0 for the first half of the phase

12.3.27 Error Control

Type: Read/Set **Offset [22:0]: 7FFEA8**
Present on all Xbus boards
Cold reset clears this register, warm reset has no effect
Compatibility: new register

Note: The Error Control register is a set/clear register, software can set one bit at a time, but cannot clear any bits. Bits are cleared by hardware once the error has been generated; also once bit 0 is set no other bits can be changed by software because the error has already been initiated

This register is used to control the different types of errors seen on the system. There are three modes of operation. The first is normal mode, no error will be produced on the system. The second mode is immediate trigger. This causes an error after this register is written to and an access is directed to the Xbus. The last mode is data match. This allows a trigger pattern to be set in the Data Match register. The error will not occur until the pattern is seen on the incoming info bus. The matching is done on the first half of the info phase. Note: It is important to set up the Fault Bit[1:0] register and the Data Match register before this register is set. The types of errors that can be produced corresponds directly to the cases described in section 6.4, Bus Errors, on page 44.

To produce an error, the first step is to decide which type of error is desired from table 8, Error Types. Then, if the error is a Data Match error the Data Match register must be initialized along with the Fault Bit[1:0] register. If the error is immediate then the Fault Bit[1:0] register must be initialized. If a transient error or a bus busy is desired the Data Match register must be initialized but the Fault Bit[1:0] register does not have to be initialized. After necessary registers are initialized the bit in the Error Control register should be set which corresponds to the error the user is looking for. Then bit 0 of the Error Control register is set. If it is an immediate error then as soon as the Xbus is accessed the error will occur. If the error is a data match error then it will not occur until the data pattern is seen on the incoming info bus.

Once bit 0 is set and the error has occurred hardware will clear the bit that indicates the error and it will also clear bit 0. The error will be asserted for as long as necessary to produce the particular case. Therefore some cases require the error to occur during both CPUTest and IOTest where as other cases will only require the fault to happen for one phase. Only one error is allowed with the exception of the transient fault. There can be a transient fault on both buses. If more than one bit is set, the lowest bit number will have priority. All bits will be cleared after the error is produced.

Table 21. Error Types

Bit	Type of Error	Case	Category
0	0: normal operation 1: produce error	——	Normal
1	CPU Board Faulty Input Circuit - CPU Driving	Case 1	Immediate
2	CPU Board Faulty Input Circuit - I/O Board Driving	Case 2	Data Match
3	CPU Board Different Data C-Side and D-side	Case 3	Immediate
4	CPU Board Faulty Output Circuit - Buffer to Pad Fault	Case 4	Immediate
5	CPU Board Open - CPU Board Driving	Case 5	Immediate
6	CPU Board Open - I/O Board Driving	Case 6	Data Match
7	CPU Board Short	Case 7	Immediate
8	Backplane Open Etch	Case 8	Data Match
9	Backplane Short	Case 9	Immediate
10	I/O Board Faulty Input Circuit	Case 10	Immediate
11	I/O Board Output Circuit Fault - Buffer to Pad	Case 11	Immediate
12	I/O Single-side Access - ASIC Parity Gen. Fault	Case 12	Immediate
13	I/O Single-side Access - ASIC PCI Data Path Fault	Case 13	Immediate
14	I/O Non-single-side Access, Different C-D Data	Case 14	Immediate
15	I/O Board Open	Case 15	Immediate
16	I/O Board Short	Case 16	Immediate
17	Transient Fault Bus A	Case 17	Data Match
18	Transient Fault Bus B	Case 17	Data Match
19	Bus Busy Bus	Case 18	Data Match

The following lines describe behavior that might not be expected with some of the error insertion cases.

Case 17: the transient bus fault will not cause the faulting board to freeze its error registers, all other boards will freeze their error registers

Case 18: busy will be asserted for two phases.

12.4 CPU Specific Xbus Register Descriptions

This section deals with the ASIC specific Xbus/Golfbus registers found in the Polo/HSC2 ASICs. All of the registers in the board specific space with the exception of the I/O Address Map Error register are non-privileged registers. A description of all CPU specific Xbus registers can be found following the table.

Table 22. Cyclops/Mirage Specific Register Map

Offset [22:0]	Register Name	Type	HSC2
7FF830	ASIC Specific Configuration	Read/Set/Clear	Not Impl.
7FF828	I/O Address Map Error 2	Read Only	Impl.
7FF820	I/O Address Map Error 1	Read Only	Impl.
7F&[a]818	Jiffy Control	Read Only	Not Impl.
7F&810	Master Jiffy Counter	Read Only	Not Impl.
7F&808	Quicktime	Read Only	Not Impl.
7F&800	Time of Day	Read Only	Not Impl.

a. privileged space & = F; non-privileged space, & = E

Cyclops and Cougar (the other large ASIC on the CPU board) share the privileged page 7FF and the non-privileged page 7FE with Cougar registers in the lower half of the pages in 000->7F8 and Cyclops registers in the top half in 800 -> FF8. Table 23 illustrates how the addresses for either CPU Xbus specific or CPU Xbus common registers are formed.

Table 23. CPU Register System Address Formation

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	islot (Xbus)				a	Offset from table 15 or table 22																						

a. paired space -> = 1; unpaired space -> = 0

12.4.1 ASIC Specific Configuration

Type: Read/Set/Clear Only in privileged page 7FF Offset [22:0]: 7FF830
Cold reset clears this register, warm reset has no effect

31:2	reserved
2	IOVA address mode 1 = 40 bit mode, 0 = 32 bit mode, refer to section 5.2.
1	Fan Fault set broken disabled
0	Maintenance Interrupt disabled for IOVA errors.

12.4.2 I/O Address Map Error 1

Type: Read Only in privileged page 7FF Offset [22:0]: 7FF828
Cold reset clears this register, warm reset has no effect

31:0 IOVA Address - refer to figure 21.

12.4.3 I/O Address Map Error 0

Type: Read Only in privileged page 7FF Offset [22:0]: 7FF820
Cold reset clears this register, warm reset has no effect

31:10	reserved.
10	IOVA Out of bounds Access Error
9	IIOVA Illegal Slot Error - PCI slot accessing (from Xbus TRID) disagrees with PCI slot field in the IOVA map entry.
8	IOVA Checksum Error.
7	IOVA Invalid Map Entry Error.
6:0	IOVA Trid (Xbus trid during 1st bus cycle of IOVA info phase).

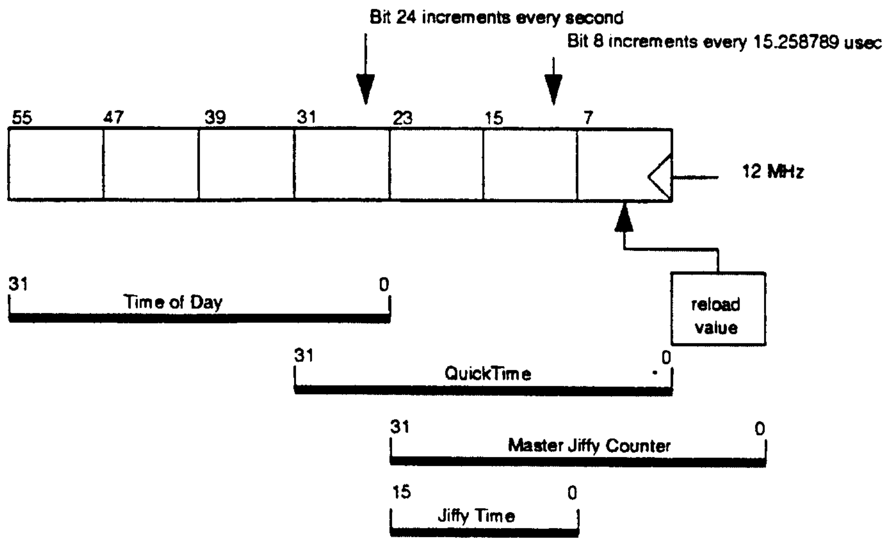
12.4.4 Time Of Day

Type: Read/Write in Privileged page 7FF Offset [22:0]: 7FF800
Read only in non-Privileged page 7FE Offset [22:0]: 7FE800
Cold reset clears this register, warm reset has no effect

This register (and the following three) implement the user accessible time of day function on Mercury. These registers are initialized through Global writes to page 7FF, and are read-only in the user visible non-privileged page 7FE.

The time of day is maintained in a single 56 bit master counter that increments at 12MHz. For compatibility with existing Stratus software, various slices of this counter are referred to by different names. The time of day refers to the most significant 32 bits of the master counter: it gives the current time in seconds.

Figure 34. Cyclops Master Counter



This register should only be written when the counter is stopped (Master Counter Enable bit in the Jiffy Control register is off). This register is cleared by cold reset.

12.4.5 Quicktime

Type: Read Only in Privileged page 7FF Offset [22:0]: 7FF808
Type: Read Only in non-privileged page 7FE Offset [22:0]: 7FE808
Cold reset clears this register, warm reset has no effect

A read-only view of bits 31:0 of the master counter. Bit 24 of quicktime increments once per second.

12.4.6 Master Jiffy Counter

Type: Read/Write in Privileged page 7FF Offset [22:0]: 7FF810
Read only in non-Privileged page 7FE Offset [22:0]: 7FE810
Cold reset clears this register, warm reset has no effect

31:8 Jiffy Counter/Master Counter
This field contains bits 23:0 of the master 12MHz counter. Bits 31:16 are often referred to as the Jiffy Counter. A Jiffy interval is 15.258789 usec, or 2^{-16} seconds.

7:0 Reload Value
This is the counter reload value. The least significant 8 bits of the master counter get reloaded with the contents of this register when they roll over. The reload value is programmable to allow accurate clock operation when the machine is running with a different speed crystal; i.e. 12MHz is no longer really 12MHz. The reload value should be set to 256 minus the number of 12MHz clock ticks in 15.258789 usec. In systems with a standard clock this works out to 49 hex (73 decimal).

This register should only be written when the counter is stopped (Master Counter Enable bit in the Jiffy Control register is off). This register is cleared by cold reset.

12.4.7 Jiffy Control

Type: Read/Set/Clear in Privileged page 7FF Offset [22:0]: 7FF818
Read only in non-Privileged page 7FE Offset [22:0]: 7FE818
Cold reset clears this register, warm reset has no effect

31:1 Reserved

0 Master Counter Enable
When this bit is set, the master counter (and thus the Time of Day, Jiffy, and Quicktime) increments normally. Clearing this bit stops the counter.

12.5 PCIB/HSC2 Specific Xbus Register Descriptions

These registers are the PCIB/HSC2 specific Xbus/Golfbus registers that reside in a PCIB/HSC2 board. Registers described here are implemented in the Gambit ASIC on the PCIB board. Refer to section 4.5, PCIB MIO/IOBus Compatible System Address Map, for details on the address mapping.

Table 24. PCIB/HSC2 Specific Xbus/Golfbus Register Map.

Offset [22:0]	Register Name	Type
7FFEC0 - 7FF088	Reserved	
7FF080	Gambit Maint. Attention Req. [31:0]	Read/Set/Clear
7FF078 - 7FF010	Reserved	
7FF008	IOBus Status Register	Read/Clear
7FF000 - 7F0000	Reserved	

Table 25 illustrates the formation of system I/O addresses for the Xbus specific and Xbus common registers found on the PCIB board. Note that the slot number is inverted. Polo PCIB boards do not run duplexed so they will not respond to paired accesses.

Table 25. MIO Compatible Register System Address Formation

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	Islot				0[a]	**Offset from table 15 or table 24**																						

a. paired space -> = 1; unpaired space -> = 0, must be non-paired

Below is a register by register description of the Gambit/Mirage specific Xbus/Golfbus registers. Note that registers are restricted to 32 bits and are spaced at 32 bit boundaries. The actual register width varies according to its definition. Reads to any undefined holes in the address space return undefined, but deterministic data. For writes, there is no effect.

The format of this information is either a bit number or a bit encoding and a functional name, depending on the configuration of the register, followed by a description of the function of that bit or bit encoding.

All registers are reset to zero unless otherwise noted.

12.5.1 Gambit Maintenance Attention Request [31:0]

Type: Read/Set/Clear- C/D different **Offset [22:0]: 7FF080**
Cold reset clears this register.
Warm reset and going through sync point have no effect.

This CD different register indicates which PCI slot (including the Gambit and Mirage host bridges) caused a maintenance interrupt. **Note: In Polo, this is a C/D different register, but it differs from other C/D different registers that bits 8, 9, 10, 11 and 12 are cleared by writing**

32'00000008, 32'h00000009, 32'h0000000A,32'h0000000B and 32'h0000000C respectively.

31,15	Reserved
30,14	Reserved
29,13	Reserved
28,12	Host_bridge (D-side)
27,11	PCI slot 3
26,10	PCI slot 2
25,9	PCI slot 1
24,8	PCI slot 0
23,7	Reserved
22,6	Reserved
21,5	Reserved
20,4	Host_bridge (C-side) - Polo Only
19,3	PCI slot 7 - Polo Only
18,2	PCI slot 6 - Polo Only
17,1	PCI slot 5 - Polo Only
16,0	PCI slot 4 - Polo Only

12.5.2 IOBus Status

Offset [22:0]: 7FF008

Type: Read/Clear
Cold reset clears this register.
Warm reset and going through sync point have no effect.

31:22	Reserved
21	Xbus/Golfbus Maintenenace Interrupt asserted due to PCIB/HSC2 Attention Request. See the SAM Maintenance Attention Request Register for details.
20:0	Reserved

12.6 SAM Interface Register Descriptions

These registers are the I/O registers that reside on a SAM board from a software perspective. For Polo, these registers exist either in the Cyclops ASIC on the CPU board or in the Gambit ASIC on the PCIB board. For HSC2, these registers exist in the Mirage ASIC located on the HSC2 board. Refer to section 4.5, PCIB MIO/IOBus Compatible System Address Map, for details on the address mapping.

The SAM compatible ASIC's register space is divided as follows. The 16k space is divided into four 4k pages to simplify the decode of this region in the Cyclops Polo ASIC and Gambit Polo ASIC.

Polo implements a single Map RAM in the CPU, four sets of per bus registers (one set for each of four PCI buses) and 16 sets of per slot registers (one set per each of 16 possible PCI slots). HSC2 implements a single Map RAM in the Mirage ASIC, one set of per bus registers and 4 sets of per slot registers (one set per each of 4 possible PCI slots).

All accesses alias to the appropriate physical register, although software locking is required to prevent conflicts. See section 5.5 for software lock restrictions on these registers.

Table 26. SAM Compatible ASIC Register Map

Offset [13:12]	Page	Polo	HSC2
2'h3	System Bus slot registers	Gambit	Mirage
2'h2	Map RAM registers	Cyclops	Mirage
2'h1	System Bus registers	Gambit	Mirage
2'h0	PCI slot registers	Gambit	Mirage

Note that registers are restricted to 32 bits and are spaced at 64 bit boundaries. The actual register width varies according to its definition. The byte significance of these registers is big-endian. Reads to any undefined holes in the address space return zeros. For writes, there is no effect. The following table illustrates the formation of a SAM compatible I/O register space address on the system bus.

All SAM registers are accessed through Xbus slots 2 and 3 in Polo systems. Accesses to registers associated with PCI slot numbers greater than seven will not be responded to in Polo systems.

Table 27. SAM Compatible Register System Address Formation

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	**System bus slot inverted**				[a]	1	0	**PCI Slot number**						1	[b]		**register offset from table 29, table 30, or table 31,**											

a. paired space -> = 1; unpaired space -> = 0
b. page offset from table 26

A bit by bit description follows (bit 31 being most significant and bit zero being the least significant bits). The format of this information is either a bit number or a bit encoding, depending on the configuration of the register, and a functional name, followed by a description of the function of that bit (bit encoding). All registers are reset to zero unless otherwise noted.

12.6.1 SAM Interface Registers - Page 3

These register reside in the Gambit and Mirage ASIC. For this group of registers, there is only a single register for the entire Polo and HSC2 system to which all 16 PCI (Polo) or 4 PCI (HSC2) slot addresses map.

Table 28. SAM Compatible I/O Space Map - Page 3

Offset [11:0]	Register Name	Type	HSC2
FF8	Disk LED Control	Read/Write	Supported
FF0	Disk Status (Polo is CD different)	Read/Write	Supported
FE8	Power supply LED	Read/Set/Clear	Not Supported
FE0	Power supply status	Read only	Not Supported
FD8	Fan speed control	Read/Set/Clear	Not Supported

Note that unlike a Polo/Gambit, the C-side and D-side Mirage's bridge a single PCI bus to the system. Therefore a HSC2 system does not have support for multiple disk shelves (SCSI port 0 & 1) and will react to accesses to SCSI Port 1(analogous to Gambit D-side) only. Accesses to SCSI port 0(analogous to Gambit C-side) are ignored.

12.6.1.1 Disk LED Control

Type: Read/Write **Offset [11:0]: FF8**

Cold reset as documented below, warm reset has no effect

This register is used to control the amber LEDs on the front of the Polo disks via the StorageWorks specified "Disk fault bus." Although this register is read/write, it should be treated as write-only by software. The reason is that the PCIBs in Xbus slots 2 and 3 share the fault bus lines. Writes to this register in either of the PCIBs set the disk LEDs, but only the most recently written PCIB will return a value from this register that is guaranteed to correspond with the actual state of the disk LEDs.

31:6 reserved.

5 Disk shelf select bit. When writing this register and this bit is a 0, SCSI port 0 is selected (C side Gambit). When writing this register and this bit is a 1, SCSI port 1 is selected (D side Gambit). This bit has no effect when reading this register.

Note: This bit is not supported on HSC2.

4 LED on bit. This bit is set to a 1 to turn on the amber device fault LED for the device whose SCSI TID is specified by bits 2:0, or all devices when bit 3 is set to a one. Setting this bit to 0 turns off the LED(s).

3 LED address test bit. This bit is used in conjunction with bit 4 to test the amber device fault LEDs on all disks in the self. Setting bit 3 to a one overrides the TID set by bits 2:0.

2:0 These bits define the SCSI device Target IDentification (TID).

12.6.1.2 Disk Status

Type: Read/Set/Clear **Offset [11:0]: FF0**

Cold reset as documented below, warm reset has no effect

CD Different (Polo Only): read via unpaired space

This register displays information based on the Storageworks disk subsystem SHELF_OK and SWAP_L signals. There are 4 SCSI ports available... C side LOCAL, C side REMOTE, D side LOCAL and D side REMOTE.

31:15 reserved

30:14 reserved

29:13 reserved

28:12 reserved

27,11 D side REMOTE Swap Interrupt Disable - Setting this bit by writing 0000008Bx suppresses the maintenance interrupt that normally is issued when a REMOTE swap on the D side occurs. Disabling the interrupt does not disable the setting and clearing of the REMOTE Swap has occurred bit. This bit is cleared by cold reset, or by writing 0000000Bx.

26,10 D side REMOTE Swap has Occurred. This bit is set when a disk has been inserted or removed from the D side disk shelf. It stays set until explicitly cleared by writing 0000000Ax, or until it is cleared by cold reset. Attempting to set this bit by writing 0000008Ax has no effect. When this bit is set, a maintenance interrupt is issued unless masked by D side REMOTE Swap Interrupt Disable bit.

25,9 D side Swap Interrupt Disable - Setting this bit by writing 00000089x suppresses the maintenance interrupt that normally is issued when a swap on the D side occurs. Disabling the interrupt does not disable the setting and clearing of the Swap has occurred bit. This bit is cleared by cold reset, or by writing 00000009x.

24,8 D side Swap has Occurred. This bit is set when a disk has been inserted or removed from the D side disk shelf. It stays set until explicitly cleared by writing 00000008x, or until it is cleared by cold reset. Attempting to set this bit by writing 00000088x has no effect. When this bit is set, a maintenance interrupt is issued unless masked by D side Swap Interrupt Disable bit.

23,7 reserved

22,6 reserved

21,5 reserved

20,4 reserved

19,3 C side REMOTE Swap Interrupt Disable - Setting this bit by writing 00000083x suppresses the maintenance interrupt that normally is issued when a REMOTE swap on the C side occurs. Disabling the interrupt does not disable the setting and clearing of the REMOTE Swap has occurred bit. This bit is cleared by cold reset, or by writing 00000003x.

This bit is not implemented in a HSC2 system.

18,2 C side REMOTE Swap has Occurred. This bit is set when a disk has been inserted or removed from the C side disk shelf. It stays set until explicitly cleared by writing 00000002x, or until it is cleared by cold reset. Attempting to set this bit by writing 00000082x has no effect. When this bit is set, a maintenance interrupt is issued unless masked by the C side REMOTE Swap Interrupt Disable bit.

This bit is not implemented in a HSC2 system.

17,1 C side Swap Interrupt Disable - Setting this bit by writing 00000081x suppresses the maintenance interrupt that normally is issued when a swap on the C side occurs. Disabling the interrupt does not disable the setting and clearing of the Swap has occurred bit. This bit is cleared by cold reset, or by writing 00000001x.

This bit is not implemented in a HSC2 system.

16,0 C side Swap has Occurred. This bit is set when a disk has been inserted or removed from the C side disk shelf. It stays set until explicitly cleared by writing 00000000x, or until it is cleared by cold reset. Attempting to set this bit by writing 00000080x has no effect. When this bit is set, a maintenance interrupt is issued unless masked by the C side Swap Interrupt Disable bit.

This bit is not implemented in a HSC2 system.

12.6.1.3 Power Supply LED

Type: Read/Write **Offset [11:0]: FE8**
Warm and Cold reset have no effect on this register.
Polo Only Register

31:3 Reserved.

2 Red LED bit. Writing a 1 to this bit position turns the Red LED on. Writing a 0 turns the Red LED off. This bit will also be set if a local power supply fault is detected.

1 Yellow LED bit. Writing a 1 to this bit position turns the Yellow LED on. Writing a 0 turns the Yellow LED off. This bit will also be cleared if a power supply fault is detected on either the local or remote power supplies.

0 Green LED bit. Writing a 0 to this bit position turns the Green LED on. Writing a 0 turns the Green LED off. This bit will also be cleared if a local power supply fault is detected.

12.6.1.4 Power Supply Status(Vanguard powe supply)

Type: Read Only **Offset [11:0]: FE0**
Warm and Cold reset have no effect on this register.
Polo Only Register

31:10 reserved.

9 Local power supply fault bit. This bit is set if the local power supply reports a fault.

8 Remote power supply fault bit. This bit is set if the remote power supply reports a fault.

7:0 Local power supply ID.

12.6.1.5 Fan Speed Control

Type: Write Only **Offset [11:0]: FD8**
Cold reset clears this register.
Polo Only Register

This register is used to control the fan speed. On power on, this register is cleared, there by forcing the fan to run at normal speed. Writing oxh80 to this register sets this register, there by forcing the fan to run at higher speed. Writing oxh00 to this register clears this register, there by forcing the fan to run at normal speed.

12.6.2 SAM Interface Registers Page 2

These registers reside in the Cyclops ASIC in Polo systems, **however they are addressed through Xbus slot 2 or 3**. In a HSC2 system, these registers reside in the Mirage ASIC and should be addressed through the normal Golbus slot.

In a Polo system, there is only a single set of registers to which all 16/4 PCI slot addresses map.

In a HSC2 system, there is a set of registers for each PCI bus (Dune).

Table 29. SAM Compatible I/O Space Map - Page 2

Offset [11:0]	Register Name	Type
FF8	Address Table Command/IOVA	Read/Write
FF0	Address Table Data 2	Read/Write
FE8	Address Table Data 1	Read/Write
FE0	Address Table Data 0	Read/Write

12.6.2.1 Address Table Command and IOVA Register

Offset [11:0]: FF8

Type: Read/Write

Cold reset clears this register, warm reset has no effect.

This register, combined with the three Address Table Data registers, is used to access the map rams for IOVA translation. See section 5. for a complete description of the address mapping function. To write the map rams, first load up the Address Table Data registers with the desired write value then write the Address Table Command and IOVA Register with the proper device index and page and bit 31 equal to 1. In order to read a map ram simply write to this register with the proper device index and page and bit 31 set to zero. Follow this initial write with a read of any of the four registers to retrieve the data.

31 Command
0 = read
1 = write

30-28 Reserved

27-20 Device Index

19-14 Reserved

13-12 Page

11-0 Reserved

12.6.2.2 Address Table Data 2 Register

Offset [11:0]: FF0

Type: Read/Write

Cold reset clears this register, warm reset has no effect.

Refer to section 12.6.2.1, Address Table Command and IOVA Register, for a description of this register.

31 - 30 Reserved

29 -26	PCI Slot Number Note: In HSC2, the most significant bit of the PCI slot number (bit [29]) is not used due to space limitations in the Mirage Map RAM. Since a HSC2 only supports 4 PCI devices this should not be considered an issue.
25-5	Reserved
4	Option bit – Xbus/Golfbus Incoherent Memory Access
3	Option bit - Swap 16 or 32 bit Endian
2	Option bit – Xbus/Golfbus Lock Cycle
1	Option bit – Data Pre-read
0	Entry valid bit

12.6.2.3 Address Table Data 1 Register

Type: Read/Write **Offset [11:0]: FE8**

Cold reset clears this register, warm reset has no effect.

Refer to section 12.6.2.1, Address Table Command and IOVA Register, for a description of this register.

31:16	Block Xbus/Golfbus Virtual Index
15:12	Reserved
11:2	Block Xbus/Golfbus Ending Physical Address
1:0	Reserved

12.6.2.4 Address Table Data 0 Register

Type: Read/Write **Offset [11:0]: FE0**

Cold reset clears this register, warm reset has no effect.

Refer to section 12.6.2.1, Address Table Command and IOVA Register, for a description of this register.

31-2	Block Xbus/Golfbus Starting Physical Address 31-12 Physical Cache Tag 11-2 Line Offset
1:0	Reserved.

12.6.3 SAM Interface Registers- Page 1

Each Gambit/Mirage ASIC has one of each of these registers for the one PCI bus (up to four controllers per bus) it controls.

Table 30. SAM Compatible Common IO Space Map Page 1

Offset [11:0]	Register Name	Type
FF8	PCI config_addr	Read/Write
7F0	PCI config_data	Read/Write
FE8	Test Control	Read/Set/Clear
FE0	PCI Error Register	Read/Set/Clear
FD8	PCI IOVA Error Register	Read only
FD0	SAM Status	Read/Set/Clear
FC8 - FB8	Reserved	
FB0	Arbitration Freeze Counter Max. Value	Read/Write
FA8	Host Request FIFO Timeout Value Register	Read/Write

12.6.3.1 PCI config_addr

Offset [11:0]: FF8

Type: Read/Write

Cold reset clears this register, warm reset has no effect.

IO reads and writes to this register cause no PCI cycles. IO reads and writes to the 'PCI config_data register' use the information in the 'PCI config_addr register' to perform a configuration cycle on the PCI bus except for device zero i.e. the bridge which is accessed internal to Gambit/Mirage using this same mechanism. Thus configuration reads and writes to the host bridge's configuration space will not cause PCI cycles but will instead be handled internal to the host bridge.

Special cycles will be generated when the 'Bus Number' equals zero i.e. the bridges secondary PCI bus, and the 'Device Number' is all ones, and the 'Function Number' is all ones and the 'Register Number' is all zeros.

31	Enable config_data - As per the PCI spec config_data is disabled if this bit is a 0.
30-24	Reserved - read only return zeros
23-16	Bus Number - encoded value to select 1 0f 256 PCI buses in a system.
15-11	Device Number - encoded value to select 1 of 32 device on a given bus.
10-8	Function Number - encoded value to select 1 of 8 functions on a specific device.
7-2	PCI Configuration Register DWORD Index - specific bytes from BEs.
1-0	2'b00 - Read only

The configuration type will be added in by hardware based on the 'Bus Number' - type 00 config to attached bus. type 01 config passed to 'Bus Number'. Refer to the PCI Local Bus Specification for more information on type 00 and type 01 commands.

12.6.3.2 PCI config_data

Type: Read/Write **Offset [11:0]: 7F0**

Cold reset clears this register, warm reset has no effect.

IO Reads and Writes to this register result in PCI configuration cycles on the PCI bus as defined by the address in 'config_addr' except for device zero i.e. the bridge configuration space is accessed internal to Gambit/Mirage using this same mechanism. The PCI configuration data is little-endian. The Xbus and Golfbus are big-endian. Reads and writes to the PCI configuration space of either the host bridge or the mated PCI adapter are always byte-swapped to provide 'byte address consistency' rather than 'MSByte consistency'. Where significance of bytes is rearranged by the swapping mechanism, software will have to re-order to restore significance. Refer to section 5.6 for a discussion of byte ordering (big endian versus little endian) on configuration cycles.

Setting bit 31 in the 'PCI config_addr register' will cause accesses to the this register to be enabled. If bit 31 of 'PCI config_addr register' is 0 writes will be ignored, and reads will return 0's and a failed_op status.

31-0 Configuration data

12.6.3.3 Test Control

Type: Read/Set/Clear **Offset [11:0]: FE8**

Cold reset clear this register.

31:16 Reserved

15 Disable Protocol error signal generation

This bit is set by software to disable Protocol error signal generated by PCI core logic. When this bit is set, the PCI core logic does not assert protocol_error signal to rest of Gambit/Mirage logic. When this bit is set, bit 5 of Polo PCI Error register will not be set. This bit is cleared by cold reset, or by writing 0000000Fx. It is set by writing 0000008Fx.

14 Disable Peer to Peer error signal generation

This bit is set by software to disable Peer-to-Peer error signal generated by PCI core logic. When this bit is set, the PCI core logic does not assert peer_to_peer_error signal to rest of Gambit/Mirage logic. When this bit is set, bit 6 of Polo PCI Error register will not be set. This bit is cleared by cold reset, or by writing 0000000Ex. It is set by writing 0000008Ex.

13 PCI read return ahead enable (Polo Only)

This bit is implemented in Rev. 1 Gambit only. When this bit is set, the Gambit sends read return data to the PCI bus as soon as data is available on post_qual bus. This will improve the PCI bus read performance for cacle line read. If this bit set to to zero, the Gambit waits till last peace of data is available on post_qual bus for cache line read before it sends data to the PCI core. This bit is cleared by cold reset, or by writing 0000000Dx. It is set by writing 0000008Dx.

12 Disable PCI drive/check error signal generation

This bit is set by software to disable PCI drive/check error signal generated by PCI core logic. When this bit is set, the PCI core logic does not assert drive_check_error signal to rest of Gambit/Mirage logic. When this bit is set, bit 9 of the Polo PCI Error register will not be set. This bit is cleared by cold reset, or by writing 0000000Cx. It is set by writing 0000008Cx.

11 Disable System error signal generation

This bit is set by software to disable system error signal generated by PCI core logic. When this bit is set, the PCI core logic does not assert system_error signal to rest of Gambit/Mirage logic. When this bit is set, bit 28 of Polo PCI Error register will not be set. This bit is cleared by cold reset, or by writing 0000000Bx. It is set by writing 0000008Bx.

10 Disable Parity error signal generation

This bit is set by software to disable parity error signal generated by PCI core logic. When this bit is set, the PCI core logic does not assert parity_error signal to rest of the Gambit/Mirage logic. When this bit is set, bit 27 of Polo PCI Error register will not be set.This bit is cleared by cold reset, or by writing 0000000Ax. It is set by writing 0000008Ax.

9 Reserved.

8 Flush Mirage Write Buffer Map Entry Caches (HSC2 Only)

Writing to set this bit will invalidate the two write buffer entry caches. This will insure that Mirage will go to the Map RAM for the next PCI write access and allows software the ability to gurantee consistent data. Refer to the Mirage Specification for more details.

Writing to clear this bit has no affect. This bit will always return a zero when read.

7 Inhibit Broken SAM on SERR

This bit is set by software to inhibit PCI error logic from removing a SAM from service when an error is intentionally caused for diagnostic testing purposes.This bit is cleared by cold reset, or by writing 00000007x. It is set by writing 00000087x.

6 Error Data Recorded/Rearm Error and Broken Registers

This bit is set by hardware when an error condition that requires logging in any of the error registers occurs. Writing a 0 to this bit clears the error state of the chip and "re-arms" them so they will capture the state of the next applicable error condition.This bit is cleared by cold reset, or by writing 00000006x. This bit can not be set It is set by writing 00000086x.

5 Force PCI data parity error

This bit is set by software to intentionally cause a PCI data parity error for diagnostic testing purposes.This bit is cleared by cold reset, or by writing 00000005x. It is set by writing 00000085x.

4 Force PCI address parity error

This bit is set by software to intentionally cause a PCI address parity error for diagnostic testing purposes.This bit is cleared by cold reset, or by writing 00000004x. It is set by writing 00000084x

3:0 Reserved

12.6.3.4 PCI Error Offset [11:0]: FE0

Type: Read Only

Cold reset clears this register, warm reset has no effect. IO Reads and Writes to this register cause no PCI cycles.

This register is valid if Status[16] is set (PCI Error detected).

31-29 PCI Grant Slot#(3-0) indicating mastership during registered error. This information is encoded in the following manner:

0: Slot 0

1: Slot 1

2: Slot 2

3: Slot 3

4: Host

28	Bridge detected SERR# on PCI Bus.
27	Bridge detected PERR# on PCI Bus
26	Bridge detected Master Initiated Termination
25	Bridge detected Target Initiated Termination
24	Reserved.
24	Host request FIFO timeout error. **Note: When this error occurs, other info in this register may or may not represent correct info about PCI bus transaction that caused this error. Refer to section 9.6.2.3 for details.**
23:20	Reserved.
19-16	PCI Command.
15-12	Reserved
11	IDLE timeout after GNT# PCI master failed to start an access within 16 clocks after GNT# asserted. Note that the PCI command field and IOVA values are not applicable.
10	Time Out Error (see PCI Common Logic Spec.)
9	Drive Check Error (see PCI Common Logic Spec.)
8	Retry Count Error (see PCI Common Logic Spec.)
7	Disconnect Count Error (see PCI Common Logic Spec.)
6	Peer to Peer Error (see PCI Common Logic Spec.)
5	Protocol Error (see PCI Common Logic Spec.)
4-0	Reserved

12.6.3.5 PCI IOVA Error

Type: Read Only **Offset [11:0]: FD8**

Cold reset clears this register, warm reset has no effect.

IO Reads and Writes to this register do not cause PCI cycles.

This register is valid if Status[16] is set (PCI Error detected).

31-0 Xbus/Gbus IOVA for the PCI transaction which got an error.

12.6.3.6 SAM Status

Type: Read only **Offset [11:0]: FD0**

Cold reset clears this register, warm reset has no effect.

State Reduction (Off-line, Not-Ready):

The contents of the state reduction, are frozen when the SAM first goes off-line or broken, so that the comparators that caused the state reduction are flagged. This is so that comparators that miscompare further down the line (after the error has rippled through the logic) do not obscure the cause of the original failure.

Writing to clear any of the state reduction bits will have no effect. The state reduction bits are re-armed via the Rearm Error register bit in the Test Control register.

31:17 Reserved.

16 SAM Maintenance Interrupt issued due to **PCI Error/Abort** condition.
Refer to the PCI Error register and the PCI Configuration Space Status registers for details.

15:14 Reserved

13 Software Set Off-line Command.
This bit is set by writing 00000081x to the page 0 Bus Interface State register.

12 Map Error (HSC2 Only)
This bit is set when an I/O Address Map Error is detected. Refer to the CPU Specific Xbus Register Descriptions on page 112 for further details about the error.

11 Failed Op (HSC2 Only)
When set, this bit indicates that the return for a transaction has timed-out and zeros were returned to prevent the device from hanging. Status[10:9] indicate which slot (Sam) was responsible for issueing the read request. Note that this Sam's state will have changed from online to online - not ready.

10-0 Resesved

12.6.3.7 Arbitration Freeze Count Max. Value

Offset [11:0]: FB0

Type: Read/Write

Cold reset clears this register, warm reset has no effect on.

After certain types of PCI errors, arbitration logic is "frozen" and a PCI_reset is issued to the affected PCI slot. Writing an 8 bit value to this register determines how many 24Mhz clocks the arbitration logic will remain "frozen" after PCI_reset is de-asserted.

12.6.3.8 Host Request FIFO Timeout Value Register

Offset [11:0]: FA8

Type: Read/Write

Cold reset sets this register to 32'h0100_0000; warm reset has no effect.

This register contains Timeout value for host request FIFO. This register is initialized to 32'h0100_00 on power on. After a host request for PCI bus access is sent, if the request is not completed by number of 12MHz clock ticks specified in this register, the PCIB or HSC2 is broken and host request FIFO timeout error bit in the PCI error register is set. Setting this register to 32'h0000_0000 disables the host request FIFO timeout error.

12.6.4 SAM Interface Registers - Page 0

There is a copy of each of these registers for each slot that a given ASIC controls. Thus the Gambit and Mirage have four copies of each register.

Table 31. **SAM Compatible I/O Space Map - Page 0**

Offset [11:0]	Register Name	Type
FB8	Board Reset	Read/Set/Clear
FB0	LED Control	Read/Set/Clear
FA8	not implemented	
FA0	SAM Host Interrupt Bit Register	Read/Write
F98	SAM Interrupt Mask Register	Read/Write
F90	SAM Interrupt Source Register	Read Only
F88	SAM Host Interrupt Address Pointer	Read/Write
F80	SAM Host Interrupt Table Pointer#2	Read/Write
F78	SAM Host Interrupt Table Pointer#1	Read/Write
F70	SAM Configuration	Read/Write
F68	PCI IO Space Offset	Read/Write
F60	PCI Memory Offset	Read/Write
F58	Bus Interface State	Read/Set/Clear

12.6.4.1 Board Reset

Type: Read/Set/Clear **Offset [11:0]: FB8**

Cold and warm reset affect this register as documented below

This register behaves almost identically to the Xbus Board Reset register (or Golfbus Board Reset register) listed in section 12.3.3. except that PCI reset is also activated through this register. PCI reset is asserted by an explicit PCI reset access to this register, or by a power-up reset. A software initiated cold or warm reset also asserts PCI reset. NOTE: A PCIB hardware or software cold reset (as indicated in the Xbus Board Reset register) has no effect on this register.

31:7 These read bits/write bits are reserved for additional board specific resets.

6 Clear SAM entry cache valid bit (HSC2 only)
Writing to set this bit generates a pulse which will invalidate the entry cache in the reader and flush any read data in the read RAM. This will insure that the reader will go to the Map RAM for the next PCI access and allows software the ability to guarantee consistent data when using the pre-read option.

Note that the wite buffer caches are unaffected by this operation. Write buffer caches can be invalidated through the Test Control Register (page 1).

Writing to clear this bit has no effect. Reading this bit will always return a 0

Software Note: Clearing the entry cache during a PCI initiated read transaction can cause unpredictable results.

5 Software initiated PCI Reset
These read bits/write bits are reserved for additional board specific resets.hen read-

ing this register, this bit =1 indicates that the board has received a software generated PCI reset since this bit was last cleared. Writing a 00000085x to this register will freeze PCI arbitration logic, generate a PCI reset pulse on the board and set this bit. Writing 00000005x to this register clears this bit. The activation of either of the cold reset bits also clears this bit. For SAM, PCI Reset clears the internal PCI macro, including the configuration space and state machines. It also causes the assertion of a PCI reset to the attached PCI adapter

Software Note: On Mirage, PCI resets also cause the the Reader hardware to reset similar to a warm_reset. Software initiated PCI resets during a PCI initiated read transaction can cause unpredictable results.

4 Software initiated Warm Reset
When reading this register, this bit =1 indicates that the board has received a software generated Warm Reset since this bit was last cleared. Writing a 00000084x to this register generates a Warm Reset reset pulse on the board and sets this bit. Writing 00000004x to this register clears this bit. The activation of either of the cold reset bits also clears this bit. For SAM, Warm Reset Clears all bus state machines, including PCI state machines, clears all non-persistent error conditions and causes a PCI Reset.. All pending bus activities are cleared. The effects of Warm Reset on individual registers is specified in the heading of each register.

3:2 Reserved

1 Software initiated Cold Reset
When reading this register, this bit =1 indicates that the board has received a software generated Cold Reset since this bit was last cleared. Writing a 00000081x to this register generates a Cold Reset reset pulse on the board and sets this bit. Writing 00000001x to this register clears this bit. The activation of the Power-up cold reset bit also clears this bit. For SAM, a Software Cold Reset clears all error reporting registers in addition to clearing everything effected by a Warm Reset and PCI Reset. The effects of Cold Reset on individual registers is specified in the heading of each register.

0 Cold Reset due to Power up
When reading this register, this bit = 1 indicates that the board has received a power-up generated Cold Reset since this bit was last cleared. This bit is cleared by writing a 00000000x to this register or by a software initiated Cold Reset. When this bit is set by a power-up reset, all other bits in this register are cleared. This bit is NOT cleared by Warm Reset.

12.6.4.2 LED Control

Type: Read/Write **Offset [11:0]: FB0**

Cold and warm reset affect this register as documented below

Provided there is a PCI card present in the specified slot, this register functions identically to the register described in section 12.3.3. with the addition that PCI reset will set the red LED and clear the yellow and green LEDs. Refer to section 12.6.4.1 for more information on PCI reset. If no PCI card is present, all LEDs will remain off.

12.6.4.3 SAM Host Interrupt Bit

Type: Read/Write **Offset [11:0]: FA0**

Cold reset clears this register. Warm reset has no effect.

Bits 4:0 of this register points to the bit to be written in the host interrupt register, EIR and the least significant byte is also the pattern to be written to the table in host memory.

31:6	Reserved.
5	1, this is the "set" bit for the read/set/clear register
4:0	Point to the bit location to be set in the host interrupt register.

12.6.4.4 SAM Interrupt Mask

Type: Read/Write **Offset [11:0]: F98**

Cold reset clears this register. Warm reset has no effect.

These bits are used to individually mask or enable the four interrupt source signals, INTA# - INTD# All four interrupt lines are tied together in hardware and must be masked/unmasked as a group.

31:1	Reserved.
0	interrupt mask - (1=enabled, 0=masked)

12.6.4.5 SAM Interrupt Source

Type: Read Only **Offset [11:0]: F90**

Cold reset clears this register. Warm reset has no effect.

This bit is used to indicate if any of the four interrupt source signals, INTA# - INTD#, are active with interrupts pending. Most single function PCI devices only use INTA#. All four interrupt lines are tied together in hardware and can only be checked as a group.

31:1	Reserved.
0	interrupt status - (1=active, 0=inactive)

12.6.4.6 SAM Host Interrupt Address Pointer

Type: Read/Write **Offset [11:0]: F88**

Cold reset clears this register. Warm reset has no effect.

This register contains the IO address of the EIR register in one or all cougars.

31:2	Bits 31:2 point to the location of the interrupt register in host memory.
1:0	Reserved.

12.6.4.7 SAM Host Interrupt Table Pointer #2

Type: Read/Write **Offset [11:0]: F80**

Cold reset clears this register. Warm reset has no effect.

These bits are the upper 16 bits of the address of the interrupt table entry in host memory.

31:16	Reserved.
15:0	Bits 47:32 of the table base address pointer.

12.6.4.8 SAM Host Interrupt Table Pointer #1

Type: Read/Write **Offset [11:0]: F78**

Cold reset clears this register. Warm reset has no effect.

These bits are the lower 32 bits of the address of the interrupt table entry in host memory.

31:0 Bits 31:0 of the table base address pointer.

12.6.4.9 SAM Configuration

Offset [11:0]: F70

Type: Read/Write

Cold reset clears this register, warm reset has no effect.

31-0 Reserved.

12.6.4.10 PCI IO Space Offset

Offset [11:0]: F68

Type: Read/Write

Cold reset clears this register, warm reset has no effect.

IO reads and writes to this register cause no PCI cycles. IO reads and writes to the 16KB window known as the SAM PCI/IO space window use the information in the PCI IO Offset register to map the base address of the 16KB IO space window. Refer to section section 4.5.1 for further information on the use of this register. This register is big endian.

31-14 Upper 18 bits of 32-bit PCI address for accesses to the 16KB PCI IO space window. PCI only defines use of the first 64KB of IO space, but the interface supports IO space accesses over the entire 32-bit address.

13-0 Reserved

12.6.4.11 PCI Memory Offset

Offset [11:0]: F60

Type: Read/Write

Cold reset clears this register, warm reset has no effect.

IO reads and writes to this register cause no PCI cycles. Memory reads and writes to the 64KB window known as the SAM Memory space window use the information in the PCI Memory Offset register to map the base address of the 64KB memory window. Refer to section 4.5.1 for further information on the use of this register. This register is big-endian.

31-16 Upper 16 bits of 32-bit PCI address for accesses to the 64KB PCI memory window. A 64KB window to memory can be put anywhere in the 32-bit PCI memory range. The PCI base address register in the configuration register space of the PCI adapter must be set to align with this window.

15-0 Reserved

12.6.4.12 Bus Interface State

Offset [11:0]: F58

Type: Read/Set/Clear

Cold reset clears this register; warm reset effect is documented below.

All bits remain cleared if there is not a PCI card in the specified slot.

These bits reflect the current state of the SAM board.

31:3 Reserved

2 Off-Line - Not Ready

Board capable of receiving and responding to Xbus/Golfbus requests, but is incapable of performing Xbus/Golfbus initiated PCI requests. There is non-zero error state in the Gambit.

This status bit is used to indicate the occurrence of a non Xbus/Golfbus related hardware fault which was detected by the checking logic. This bit is set automatically by hardware. Writing a 00000082x to set, or a 00000002x to clear will have no effect. This bit is cleared by a cold reset, or by writing the Test control register with a Rearm Error Command.

1 Off-Line - Ready

Board capable of receiving and responding to Xbus/Golfbus requests but is incapable of performing Xbus initiated PCI requests. No error conditions are present.

This state is entered automatically after a cold or warm reset (provided there are no errors detected by the Gambit or Mirage) or when the error state of the chip is cleared via a Rearm Error Command. Writing a 0000081x to set, causes SAM Status[13] to get set and a Attention Maintenance Interrupt to be generated. Clearing this bit has no effect. (Note if the SAM is broken you won't be able to read this.)

0 On-line

Board capable of receiving and responding to Xbus/Golfbus requests. Board capable of performing Xbus initiated PCI requests. PCI adapter enabled to post requests.

Writing a 00000080x to set, bit causes the PCI arbiter to be enabled provided there is a non-zero error state. Writing a 00000000x to clear causes the Gambit/Mirage to go Off-line - Ready but a Attention Maintenance Interrupt is not generated. Note if the SAM is broken you won't be able to read this.)

Xbus Functional Specification

JED-00155

"If You Want Fault Tolerance, Buy A Jetta"

Will Leavitt (editor)
Conrad Clemson
Jeffrey Somers
John Chaves
David Barbera

21 July 1995

Revision: 1.3

Appendix II

Table of Contents

List of Figures

1. Introduction

This document provides a detailed description of the Polo Xbus. Contained in this document are the signal descriptions, functional protocol descriptions, functional timing diagrams, and physical bus descriptions, including pinouts and clocking methodology. The Xbus protocol is directly based on the Stratus Gbus, and this specification is itself derived from the Gbus Functional Specification.

1.1 Applicable Documents

1.1.1 Stratus Documents

Cougar Specification
Rev 1.2, 9/9/92, or later
/auto/jetta/doc/CPU/cougar/...

Polo Programmer's Guide
Rev 3.0 or later
/auto/polo/doc/func_spec/...

Gofer Specification
Rev 2.0 or later
/auto/jetta/doc/CPU/gofer/...

GolfBus Specification
Rev 3.1 or later
/auto/jetta/doc/gbus

Mercury Functional Specification
/auto/jetta/doc/cpu/board/...

Polo Cyclops Specification
Rev 1.0 or later
/auto/polo/doc/CPU/cyclops/...

Polo Backplane Specification
Rev 1.0 or later
/auto/polo/doc/backplane/...

Polo Clocking Specification
Rev 1.0 or later
/auto/polo/doc/clock/...

1.1.2 Other Documents

IEEE Standard 1149.1: IEEE Standard Test Access Port and Boundary Scan Architecture
February 15, 1990
Test Technology Technical Committee of the IEEE Computer Society
The Institute of Electrical and Electronics Engineers, Inc.
345 East 47th Street, NY, NY 10017-2394

1.2 Revision History

Revision 1.0:

- Changes to almost every section in order to complete document.

2. Functional Overview

2.1 Introduction

The Xbus performs a number of functions:

- It supplies a means to transfer information between Polo CPU boards and PCI Bridge cards.
- It supplies a means to transfer information between two Polo CPU boards.
- It provides fault detection and isolation for the bus and the boards on the bus. However, it is only fault detecting, not fault tolerant.

This Xbus must have a number of functional characteristics. First, it must provide enough throughput so that it is not the system bottle neck. Second, it must be inexpensive to implement. Third, it must be able to support both PCI accesses, 32 bits of address and multiple words of data, and PA-RISC accesses, 48 bits of address and multiple words of data. It must also be able to support Cougar/PA-RISC functionality such as regurgitated infos, flushes, and purges.

There are a number of possible interconnect structures that could be used for this bus. The two that were carefully investigated are a fully interconnected bus, such as in Jetta or StrataBus based systems, and a series of 4 point to point buses as shown in figure 1. In order to determine which of these interconnects would best meet the Polo implementation requirements, several key areas were examined.

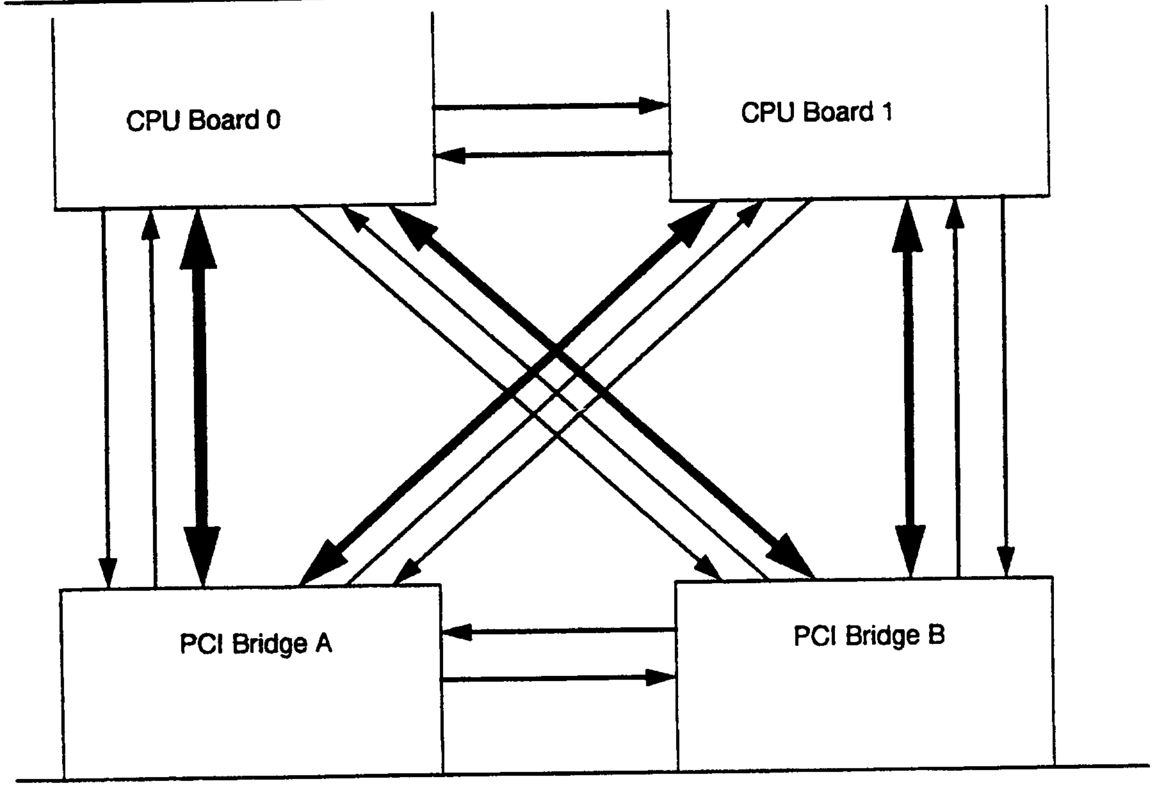
An important part of any Stratus design is live insertion. In order to accomplish live insertion on a fully connected bus, it is necessary to insert a board into the system without perturbing any activity. This is accomplished through the use of special bus transceivers. The bus transceivers are expensive, power hungry parts. Further, these transceivers have some strict requirements associated with them. First, they require a bias voltage of 4 to 5 volts. This voltage must be applied before the signals make connection with the backplane. This places constraints on the power architecture. Consideration must be given to where the bias voltage is generated, and how it is delivered? Second, the BTL transceivers require a termination voltage of 2.1 volts. This voltage must be very tightly controlled. It could be probably be generated by the suitcase supplies, but that would increase the cost of each supply by over $400. Even though the suitcases can provide a source for the power for the termination voltage, in order to regulate it to the tight constraints of the BTL transceivers, it would be necessary to add backpanel power supplies. ?

? In a po~~int to point solu~~tion, buses are not used when both units are not in the system, not broken and 100% functional. This eliminates the need for liv~~e insertion~~ as we know it today. By removing this requirement, the need for the BTL transceivers is eliminated, thereby removing the need for bias voltage and 2.1 volt generation.

Also, with the point to point solution, termination becomes much simpler. Instead of parallel termination series termination is used. This change in termination removes the need for generating a termination voltage and the need for backpanel power supplies.

There are some secondary considerations such as bus isolation, and complexity of implementation, but the live insertion issue is the real driver for the technology choice.

Figure 1. Xbus interconnect



KEY

◀━▶	Bidirectional, point-to-point data bus.
──▶	Unidirectional, point-to-point control bus.

2.2 Architectural Alternatives

There are 4 fundamental choices for implementation of this bus. The bus could be Golfbus based, IBus based, PCI based, or totally new. These possibilities are explored in the next sections.

2.2.1 New Bus

In order to justify inventing a new bus, there must be a good reason. If performance, functionality, or cost goals cannot be met by an existing structure, then it is necessary to invent one. This is not the case for Polo. Therefore, this possibility is discarded.

2.2.2 PCI Based Bus

A PCI based bus is one possible implementation alternative. It meets performance and cost goals. It could be modified to meet the addressing and Cougar functionality. The PCI is probably not quite optimal from a performance stand point due to the fact that it is not split transaction. This is clearly a negative point for a PCI based solution. This design probably simplifies the Gambit ASIC at the expense of the Cyclops ASIC.

If a PCI based bus is used, the PCI protocol will have to be modified to support the virtual index of the PA chip. This is required to support CPU to CPU transfers. The PCI protocol will also have to be modified to support the fault detecting nature of the interconnect. Finally, some new function codes will have to be added to the bus in order to allow it to support the sync functionality of the CPU boards.

For these reasons, a PCI based solution is discarded.

2.2.3 Ibus based Bus

The Ibus based implementation has quite a few advantages over the previous two possibilities. The Ibus is already defined and has a performance and cost point that meet the needs for the Xbus. It already supports PA virtual indexes, and a split transaction protocol. It will require modification to support PCI based non virtual index transfers, but there are spare bits in the virtual index transfer phase that can be used for this purpose. So inventing that cycle should not be difficult.

There are some disadvantages to an Ibus based solution. First, an error protocol will have to be invented that can support the fault detecting needs of this bus. This will require reworking some of the central state machines of the Ibus. Second, although the Ibus has some similarities to the Golfbus, there are enough differences that it is unlikely that the RTL recycling inside of Gofer will be large. The tie between the Ibus and the Golfbus inside of Gofer is tight, therefore, removing the Golfbus may require a substantial reworking of the Gofer internal interfaces.

For these reasons, although the Ibus based solution is a reasonable one, it is not considered to be as good as the next possibility.

2.2.4 Golfbus Based Bus

Although at first glance a Golfbus based solution for the Xbus may seem like overkill, it has a number of significant advantages even over the Ibus based solution. The Golfbus maintains all of the Ibus advantages. However, the Golfbus also has the advantage that it already includes a robust error handling protocol. It has an interface that is well integrated into the existing Gofer RTL logic. Hopefully, a Golfbus based solution will allow for a maximal sharing of RTL code between Gofer and Cyclops. Additionally, the Golfbus logic has already been used in a common logic fashion. This may allow for good sharing of logic between the Cyclops ASIC and the Gambit ASIC.

Unfortunately, the Golfbus requires a large number of external transceivers and parallel backplane termination, which do not meet the cost objectives of the polo project. For these reasons, the Polo Xbus will use a Golfbus-based protocol running on a newly designed, point to point interconnect.

2.3 Golfbus Logical Overview

The Golfbus is a single logical split transaction multiplexed address/data bus. It is duplicated to provide fault tolerance in a manner similar to the Stratabus. Major features of the bus include the ability to support 32 and 64 bit bus interface widths, completely synchronous operation, cache consistency support, and a single logical ASIC interface.

In its initial implementation, the bus supports:

- 2 logical (4 physical) CPU/memory boards, with a 64 bit bus.

- 10 physical IO boards, with a "32 bit" bus.

Peak supported bandwidths are 128MBytes/sec between CPU/Mem boards (assuming all line transfers) and 76.8MBytes/sec to/from IO boards. All information transfer across the Golfbus occurs synchronously to/from the bus transceivers at ~24MHz.

A single logical bus is used for both address and data transfer. The bus contains up to 64 bits of information (data or address and function code), 7 bits of tag, and a single bit which indicates whether the information lines are carrying data or address and function code. The IO boards in the initial implementation will only connect to 32 bits of the information bus. (Special function codes will be used to interface to and from the IO boards.) The information bus, tag, and func_op bit are covered by 8 or 16 bits of parity for "32 bit" interfaces and "64 bit" interfaces respectively.

Control signals and arbitration lines are protected by a three-way voting algorithm. This provides the ability to tolerate a single control signal failure within each three-way voted control line.

The Golfbus interface logic provides full checking between the C-side and D-side of each Golfbus board via loopback, thereby providing board level fault detection. The Golfbus interface is able to isolate itself from the Golfbus, even in the event of clock failures, thereby providing board level fault isolation.

Because of the split nature of the bus, all boards in the system must be able to arbitrate for the bus and initiate a bus operation.

2.4 Golfbus Physical Summary

The Golfbus has an extremely efficient physical implementation. Support of its information transfer and fault detection/isolation capabilities require (in addition to clock generation circuitry) only:

- Connectors
- Transceivers
- 1 logical (2 physical) ASICs
- 1 logical (2 physical) MSI 20-pin register component

Support of board ID PROMs and board indicator circuitry (card edge LEDs) requires some additional MSI circuitry.

The 32-bit version of the bus interface requires fewer than 150 backplane signal connections and the 64-bit version of the bus interface requires fewer than 250 backplane signal connections. 64-bit and 32-bit interfaces coexist in the same backplane, though backplane slots are wired to support one or the other. The AMP SL-100 connector is used for the Golfbus backplane connections.

The transceiver used on the Golfbus is the FB2033, a Stratus specified bicmos device provided by Signetics Corp. and Texas Instruments Inc. This is a Futurebus compatible device with controlled edge rates, capable of delivering 100mA on the bus side. These octal devices are implemented as a three port device, i.e. each bit has a board side input port, a board side output port, and a bus side bidirectional port. The transceiver is used in a clocked mode for both inbound and outbound data flow. A testability feature, which allows internal loopback checking, i.e. a path that includes the entire part except the bus side output driver and input buffer, is also utilized to facilitate testing. The transceiver supports live insertion and removal.

"64-bit" Golfbus interfaces can be supported via 2 391-pin ASICs and "32-bit" Golfbus interfaces can be supported via 2 304-pin ASICs, though for the first implementation the former will be used for both.

The Golfbus interface requires three clocks to transfer information across the backplane, a ~24MHz transmit clock, a ~24MHz receive clock, and a ~12MHz phase information clock qualifier. In addition, first generation Xbus interfaces will require 2 ~24MHz clocks (really OE signals) for transferring information from the bus transceivers to the bus ASICs.

2.5 Polo XBus Logical Overview

Polo implements a subset of the of the Golfbus protocol. The following discussion outlines key Golfbus features not supported in the Polo implementation.

Bus widths are limited to 32 bits. Since there are no CPU boards with remote memory on the Polo Xbus, there is no need to use the optional 64 bit widths of the Golfbus.

The configurations are limited to 4 devices. Two CPU boards and two PCI bridge cards.

Although the physical implementation uses 4 point to point buses instead of 2 fully connected buses, the protocol is still implemented with a single bus view. On any given cycle, there is no more than one transaction on the interconnect. This transaction may be on all 4 buses for the case of a duplexed operation or on only 2 of the buses for a simplexed operation. It is the responsibility of the arbitration network to ensure this functionality.

Like the Jetta implementation, the signals are divided into two categories, transaction based bused signals and control signals. However, due to the point to point nature of the signals, the implementation is very different. The bused signals are implemented as 4 bidirectional point to point buses. These are shown as the heavy lines in figure 1. The control signals are implemented as separate unidirectional buses. These are shown as the lighter lines in figure 1. The bused signals are protected through the map ram, parity, and loopback checking. The control signals protected through ECC and loopback checking.

One fundamentally new feature of the Xbus is the error protocol. Unlike the Golfbus, the Xbus attempts not only to detect bus errors, but also to diagnose the source. This aspect of the Xbus is covered in a later section in detail.

A second fundamentally different feature is the mechanism by which two CPUs communicate. A CPU to CPU transaction is accomplished by means of a peer-to-peer operation in which the transaction is broken into two separate transactions. In the first, the CPU transmits the transaction from the CPU to the PCIB. In the second, the PCIB transmits the information from the PCIB to the CPU. This is explained in detail in the section on peer-to-peer transactions.

2.6 Polo XBus Physical Overview

The Polo XBus physical implementation makes use of a direct ASIC to ASIC connection technology. The bus interface is accomplished through the use of 2 391 pin PGA package ASICs. The Xbus uses two different styles of connectors. The CPU boards (suitcases) use the SL-100 connector from AMP. This connector was chosen for its ruggedness, built in stiffeners, and grounding scheme. The PCIBs use the multiple sourced futurebus connector. These connectors, while less rigid, provide good signal integrity, and are inexpensive. A single 24Mhz clock is used for clocking data and control onto and off of the bus.

3. Detailed Functional Description

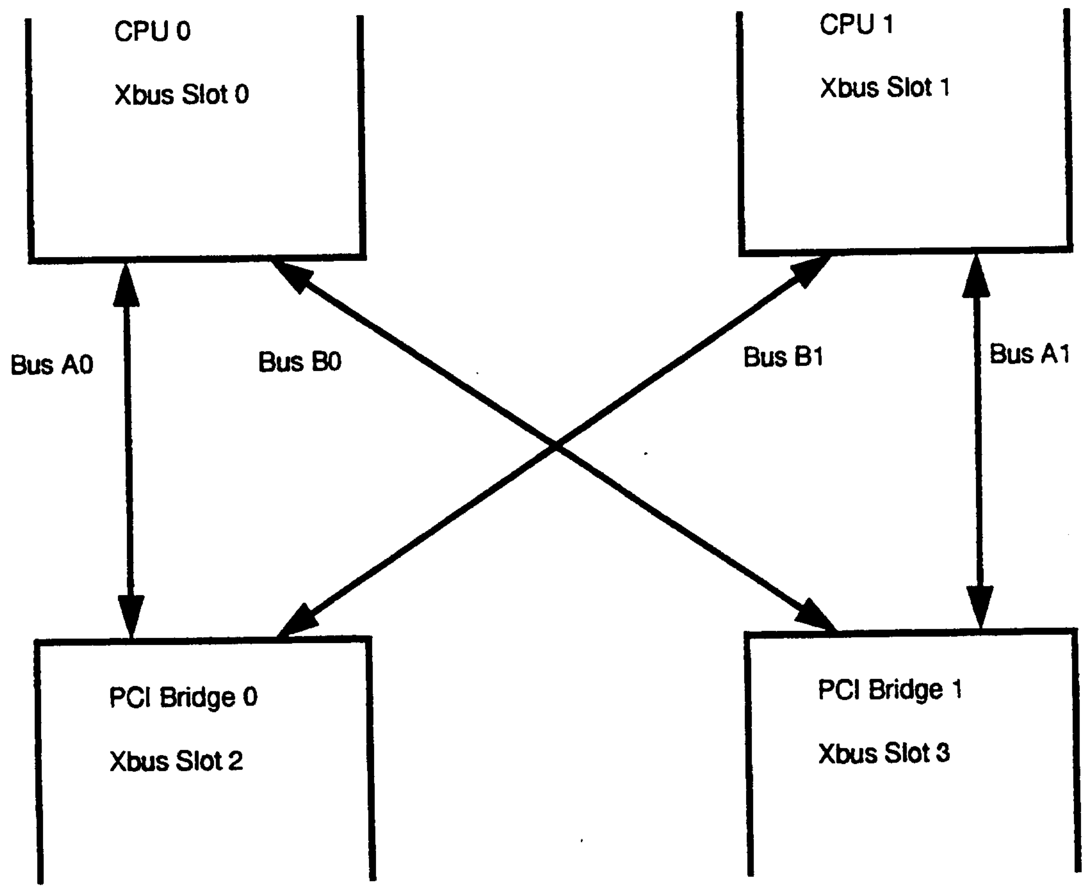
3.1 Bus Naming Convention

The Xbus actually consists of 4 data buses and 12 control buses. With so many buses in the system, it is important to have a concise intuitive naming convention.

3.1.1 Data Bus Naming Convention

Figure 2 below, shows a block diagram of the 4 data buses in the Xbus system.

Figure 2. Xbus data Buses

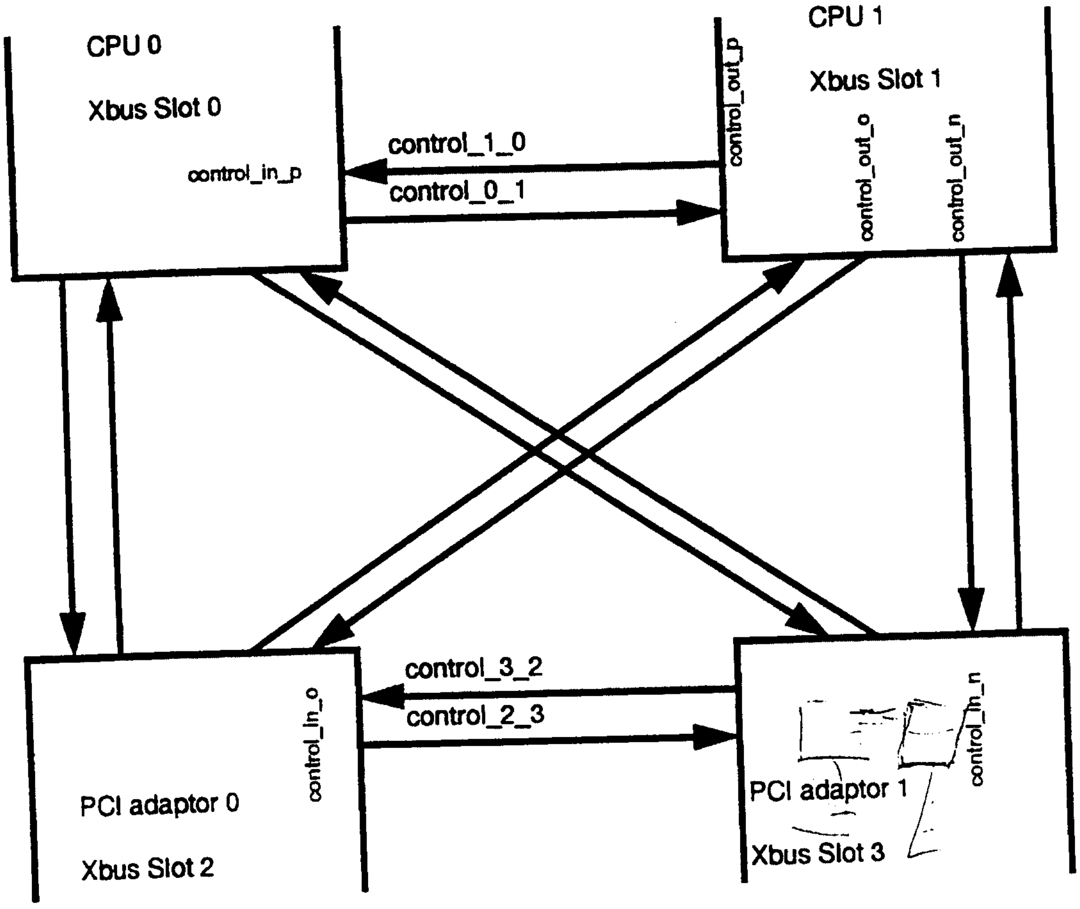


The number for the bus is taken from the CPU slot number; therefore data buses connected to CPU 0 end in 0 and data buses connected to CPU 1 end in 1. The letter for the bus is determined by whether or not the bus is a crisscross bus (i.e. connects an even slot number to an odd slot number) or a straight bus (i.e. connects an even to an even or an odd to an odd slot). Based on this convention, CPU 0 has connections to data bus A0 and B0. CPU 1 has connections to data bus B1 and A1. PCI Bridge 0 has connections to data bus A0 and B1. PCI Bridge 1 has connections to B0 and A1. Note that unlike traditional Stratus boards, the PCI bridge cards do not run in lock-step.

3.1.2 Control Bus Naming Convention

Figure 3 below shows a block diagram of the control buses in the Polo system.

Figure 3. Xbus Control Buses



The control naming convention for the backplane signals uses the signal name followed by the source of the signal and then the destination of the signal. The naming convention for the associated ASIC pins uses the signal name, the direction (in or out), and the connection (n for neighbor, o for opposite, and p for peer). Examples of this naming convention are shown in Figure 3. Table 1 lists all of the control bus names.

Table 1. Control Bus Names

Control Bus Source	Control Bus Destination	Control Bus Name	ASIC Driving Pin Name	ASIC Receiving Pin Name
CPU 0	CPU 1	control_0_1	control_out_p	control_in_p
CPU 0	PCIB 0	control_0_2	control_out_n	control_in_n
CPU 0	PCIB 1	control_0_3	control_out_o	control_in_o
CPU 1	CPU 0	control_1_0	control_out_p	control_in_p

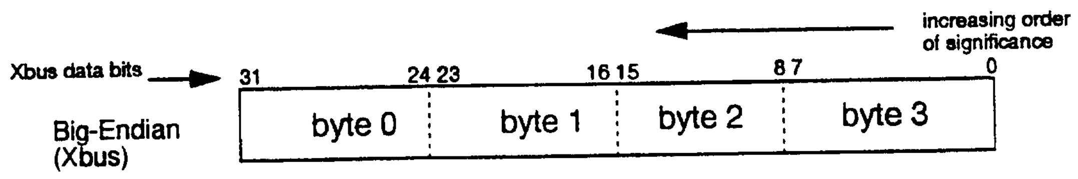
Table 1. Control Bus Names

Control Bus Source	Control Bus Destination	Control Bus Name	ASIC Driving Pin Name	ASIC Receiving Pin Name
CPU 1	PCIB 0	control_1_2	control_out_o	control_in_o
CPU 1	PCIB 1	control_1_3	control_out_n	control_in_n
PCIB 0	CPU 0	control_2_0	control_out_n	control_in_n
PCIB 0	CPU 1	control_2_1	control_out_o	control_in_o
PCIB 0	PCIB 1	control_2_3	control_out_p	control_in_p
PCIB 1	CPU 0	control_3_0	control_out_o	control_in_o
PCIB 1	CPU 1	control_3_1	control_out_n	control_in_n
PCIB 1	PCIB 0	control_3_2	control_out_p	control_in_p

3.2 Bit Numbering

The Xbus and HP-PA7100 use different bit numbering conventions. Refer to Figure 4 for a description of the bit numbering scheme used on the Xbus. Refer to the Mercury Functional Specification for a description of the HP-PA7100 bit numbering scheme. Refer to the MIO/Polo PCI programming specification for the PCI bit numbering scheme.

Figure 4. Xbus Bit Numbering Convention



3.3 Terminology

bus cycle — the 24MHz (~41.67 ns) building block from which all Xbus operations are built. A bus cycle is the time which a valid logic level driven by one board on the backplane is seen by all other boards. Two bus cycles compose a bus phase, 4 bus phases compose a bus operation, and one or more bus operations compose a bus transaction.

bus phase — the 12MHz (~83.3ns, 2 bus cycle) building block from which all bus operations are constructed. There are logically 11 types of bus phases on the Xbus; "Arb", "Info", "Post1", and "Post2" are the phases that occur during normal operation. When an error is detected, the special phases "Error1", "CPU test", "CPU Post", IO Test", IO Post1", "IO Post2", and "Error2" are inserted in the protocol for fault isolation. (The error phases are sometimes collectively referred to as "Post3"). During each bus phase it is possible to transmit two sets of information on a physical set of backplane lines (i.e. "double pumping") though this is not done for all bus phases and/or signals.

bus operation — A bus operation is the basic unit of address and data transmission and checking on the Xbus. It is generally composed of at least 4 phases: an Arb phase followed by Info, Post1 and Post2 phases. Bus errors cause the insertion of the error phases after Post2 and increase the number of phases required to complete a bus operation. Bus operations may consists of multiple info transmissions in the case of a block transfer. A bus operation can be thought of as a full one way transfer on the Xbus.

bus sub-operation — A bus sub-operation is an operation initiated by a bus master during subsequent phases of a block transfer. Sub-operations always carry data and BUSYs are ignored. It is generally composed of 4 phases: an Arb phase during which grant inhibit is used, followed by Info, Post1 and Post2 phases. Bus errors may increase the number of phases required to complete a bus sub-operation. A sub-operation is differentiated from an operation in that a bus operation for a block transfer consists of the first transfer plus a number of sub-operations consisting of multiple data transfers.

bus transaction — a complete high level exchange of information on the Xbus. Examples include reads and writes. A read transaction between CPU and PCIB is composed of a minimum of two bus operations; one operation provides the address and function code, and one or more operations provide the return data. A write transaction to a PCIB is composed of a minimum of two operations; one operation provides address and function code, and one or more additional operations provide the data.

The following diagrams illustrate the terminology surrounding a simple word read and line write transaction.

Figure 5. Terminology - Simple Word Read Transaction

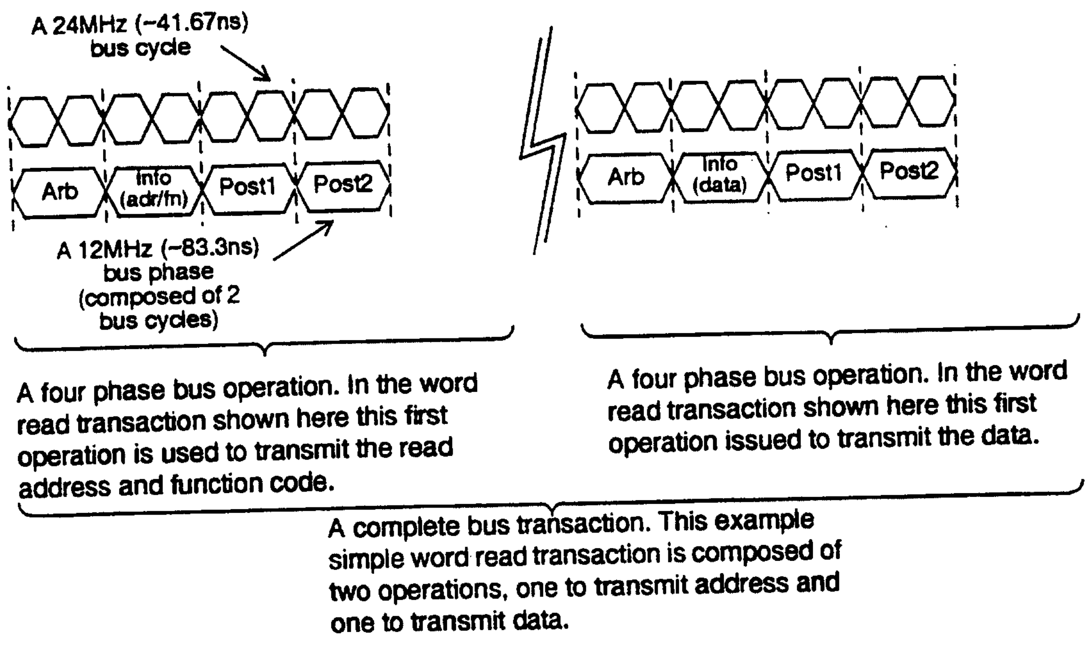
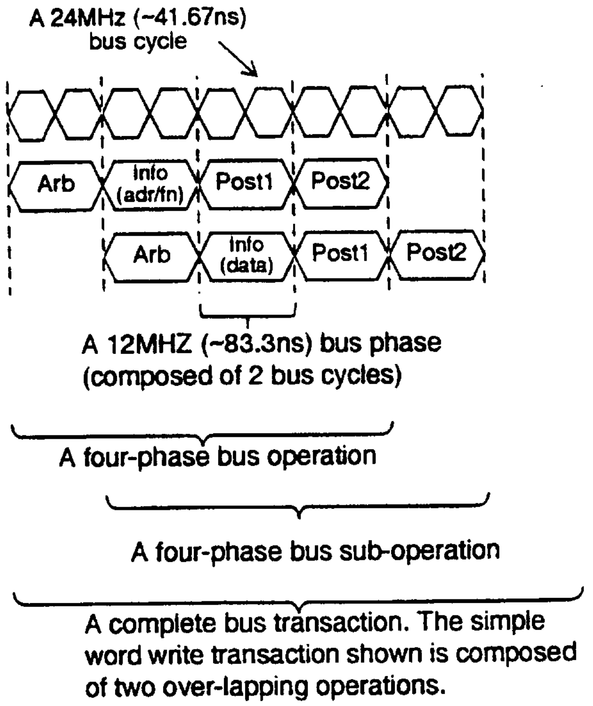

A 24MHz (~41.67ns) bus cycle
Arb
Info (adr/fn)
Post1
Post2
A 12MHz (~83.3ns) bus phase (composed of 2 bus cycles)
Arb
Info (data)
Post1
Post2
A four phase bus operation. In the word read transaction shown here this first operation is used to transmit the read address and function code.
A four phase bus operation. In the word read transaction shown here this first operation issued to transmit the data.
A complete bus transaction. This example simple word read transaction is composed of two operations, one to transmit address and one to transmit data.

Figure 6. Terminology - Simple Line Write Transaction


A 24MHz (~41.67ns) bus cycle
Arb
Info (adr/fn)
Post1
Post2
Arb
Info (data)
Post1
Post2
A 12MHZ (~83.3ns) bus phase (composed of 2 bus cycles)
A four-phase bus operation
A four-phase bus sub-operation
A complete bus transaction. The simple word write transaction shown is composed of two over-lapping operations.

Bus Master — A board that has won arbitration. This board drives the info lines in the info phase of the bus operation. A bus master can be a transaction master or a transaction slave.

Bus Slave — A board that has determined the info lines carry information that it must receive. A bus slave can be a transaction master or a transaction slave.

CD different read — A read in which the C and D side ASICs each provide half the data, e.g. when reading error status registers. Loopback checking of the bytes driven by the other side is suppressed.

Cyclops – The Xbus to Ibus interface ASIC on the CPU board in Polo systems.

echo transaction — The second half of a peer-to-peer bus transaction between CPUs. Send and echo transactions are not split; grant inhibit is used to ensure that no other transactions occur between the send and echo. This is to prevent re-ordering.

EFQ – Eviction-Flush Queue. This queue exists only on CPU boards. Refer to the Cyclops (Bus Interface) Specification for details.

EFQ-Freeze-State – Set via bit 19 of the Bus Interface State Register. This mode only exists on CPU boards. When in this mode, a board will busy all accesses directed to its EFQ except those from its partner unless it is a data return of any size.

Fast Duplexing – Fast duplexing is a form of duplexing in which no memory is updated. This is done when both boards are coming up for the first time as opposed to updating a new board from a running OS.

Freeze-State– Set via bit 14 of the Bus Interface State Register. This mode only exists on boards that can be duplexed. When in this mode, a board will busy all accesses directed to its RWQ except those from its partner.

Gambit – The Xbus to PCI ASIC on the PCI Bridge card. IT interface to the Xbus and the PCI bus.

I/O virtual address (IOVA) – An IOVA is an address generated by a PCI card. This address is transmitted across the Xbus to the cyclops ASIC. Inside the cyclops ASIC the address is translated into a valid system address. The IOVA is used to provide fault tolerance. It guarantees that a PCI card will generate a correct address range.

loopback checking— This is when an ASIC checks that the value it sees on a pin is equal to the value it thinks should be on the pin during normal operation.

loopcheck operation— This is when the bus ASICs drive 55/AAs as part of the error protocol in order to determine the site of a fault.

peer to peer transaction — A two part transaction between two CPUs. The Xbus does not have fully interconnected data buses, and transfers between the two CPU boards must occur in two steps: first a send between the CPU and the PCIB(s), then an echo from the PCIB(s) to all of the CPU(s). The requesting CPU drives a complete transaction on its A and B buses. The PCIBs look at the address, and determines whether the transaction is directed to the CPUs. If it is, they buffer the transaction in order to repeat the transaction on their A & B buses once Post2 of the last info phase has passed with no bus errors. In this way, both CPUs see the transaction at the same time. Peer to peer transactions between CPU boards require a minimum of four operations.

RWQ – Read/Write Queue. This queue exists only on CPU boards. Refer to the Cyclops (Bus Interface) Specification for details.

Regurgitated Info – A regurgitated info is a cougar generated cycle used during the update process. It is generated by the update-on-line board and transmitted to the update off-line board. It is unique because an update off-line board accepts info cycles even if the base address does not match the base address of the board.

send transaction — The first half of a peer-to-peer bus transaction between CPUs.

single side operation — An operation with data supplied entirely by either the C or D ASIC; e.g. a read from a PCI card. Loopback checking is performed only by the side supplying the data.

TRID — Transaction identifier. This a unique binary number used to tie together two bus operations during a split transaction and to identify the source for write transactions (TRIDs on write transactions are strictly for debug). A TRID is unique only while a given transaction is still outstanding and will be re-used for later transactions by a transaction-master. Note that **trid** bits 02 - 00 are used for the slot number of the transaction-master and **trid** bits 06 - 04 are generated by an on-board master - thus allowing a board to have 8 unique masters with transactions outstanding.Trid bit 03 is a new bit for Polo that indicates the format of the address; a zero indicates a Jetta style system address is being transmitted and a one indicates an IOVA (I/O Virtual Address) format is on the backplane. The IOVA address needs to be translated into a system address via the map RAM. Refer to section 6.10.1 for a complete description of the trid field.

Transaction Master — The specific resource on a board that generated a transaction. The transaction master is responsible for generating the TRID of the transaction.

Transaction Slave — The board on which a transaction was directed towards. i.e. a board in which the function code and address of a bus operation has decoded to.

4. Xbus Signal Description

The following section describes in detail the various signals that comprise the Xbus. As there are 4 buses in the Polo implementation, a new naming convention has been developed for the buses. This naming convention is described in section 3.1, Bus Naming Convention, on page 15.

A functional description is included for each type of signal and not for every individual signal (for example: there is a TRID field for all 4 buses, however there is only one description that covers both). Power control and JTAG signals are not described here.The signal names are designed to be compatible with the Verilog coding standards so that the specification will match the code exactly. The following rules were used in creating the signal names:

- all lower-case characters
- the "_" is a delimiter when it is not at the end of a signal name
- the "_" at the end of the signal name indicates that the signal is low-true
- _a indicates A_bus signals
- _b indicates B_bus signals
- _x_, _y_, and _z_ are used for the three low-true signals on a triplicated net
- [n:m] is used to describe a multi-bit field

4.1 Signal Description

As described earlier, the Polo Xbus implements the control and bused signals very differently.

The bused signals are implemented as 4 point-to-point bidirectional signals. The ability to drive these signals is controlled through the arbitration network described later. These signals are protected by a single parity bit that accompanies the bus. The buses are interconnected so that each CPU is connected to both PCI bridge boards and each PCI bridge board is connected to both CPUs. Error recovery is accomplished through the XBus error protocol.

The control signals are organized in a set of point to point unidirectional buses. Each of these buses is ECC protected. These buses carry control signals which are not governed through arbitration. Unlike the bused signals, there is a control bus in each direction between every board. This is necessary in order to ensure the single bus view of the system. For example, if one PCI bridge card sees a bus error, that information must be transmitted to all three other boards in order for the boards to all perform the error sequence.

The bused signals and control signals are double pumped at 24MHz each cycle. That is, they carry different data during the first and second 24Mhz bus cycles that make up a single phase. All buses and control signals are active high on the backplane.

A small number of reset and broken related control signals are buffered by the 26S10 transceiver and replicated for three way voting. These signals are active low on the backplane.

4.2 The Info Bus

The following signals are collectively referred to as the info bus. Although there are actually 4 sets of these signals (a0,a1,b0,b1), for simplicity's sake only the a0 version is listed. For example, when the TRID field is described, it should be understood that there are actually 4 TRID buses:

trid_a0, trid_b0, trid_a1, and trid_b1.

Table 2. Xbus Bidirectional Buses

signal	width	description
info_a0[31:0]	128 (32x4)	**Xbus info bus-** Info is driven during the info phase of a bus operation by the current bus master. This field may contain either an address (physical address or virtual index with function code) or data, depending on the func_op control line.
trid_a0[6:0]	28 (7x4)	**Xbus transaction id (TRID)** - The trid lines carry the TRID (transaction ID) during the first cycle of a phase. During the second cycle, it carries the number of phases remaining, the first_op bit, and cache coherency bits.
func_op_a0_	1	**Xbus func_op** - This line carries the func_op_ signal which indicates that the current information on the info bus contains a function code that should be decoded. This signal is low true so that an idle bus will indicate that a function needs to be decoded, and thus a no-op function. This bit is valid during an info phase and is protected by parity along with the lower half of the info field. During the second cycle, it is unused (driven to logic 0 on the backplane).
parity_a0	1	**Xbus parity** - This parity signal covers all of the bidirectional signals on a bus: info, trid, func_op_.
TOTAL	**158**	

4.3 The Control Bus

This section describes the control signals. There are actually 12 control buses, but again only one is described here. The names of the twelve control buses are listed in table 1. For simplicity of documentation, the control bus is identified by bit numbering, similar to the trid field. However, since the meaning of the control bus bits is very significant, each one is described in detail here.

The control buses are protected by a single bit correction, double bit detection ECC code.

Table 3. Control Bus Signals

signal	width	description
control[0]	12 (12x1)	**bus_req and ack** - During the first half of the phase, this bit is used for bus_req. During the second half of the phase, this bit is used for ack. During the first half of the phase, this bit is driven by a board when it is requesting the bus. As described in later sections, the bus uses a distributed arbitration model loosely based on the Golfbus. Each board in the system drives the bus_req and tests all of other boards bus_req signals to determine who will drive the info signals during the next phase. During the second cycle of the phase, this bit is used to acknowledge a bus transaction. Ack is asserted in Post2 by the target board of the transaction. This signal is the result of an address decode, so it is only valid in Post2 of an operation that is transferring an address. Ack provides an indication of whether or not a transaction is progressing. This is relevant in a Polo system, since a PCI card may go away resulting in a no ack for a ping operation. Acks in a Polo system also let a PCI Bridge know that a CPU's map RAM has mapped a PCI initiated access to a valid CPU address. If a PCIB's read or write is not ACKed, the PCI slot that initiated the access may or may not be set off-line depending on bits in the Gambit's configuration register, described in section 12.6.4.9, SAM Configuration, on page 127 in the Polo Programming Guide. Writes from the CPU to the PCIB are acked to facilitate debug, but are otherwise unused. A peer to peer CPU write is not ACKed by the PCIBs, but the echoed operation is acked by both CPUs; again, this is only for assisting debug. Ack is ignored for the send portion of a peer-to-peer operation. The CPU initiator of a peer-to-peer operation must track the entire operation to see whether the cycle is acknowledged.

Table 3. Control Bus Signals

signal	width	description
control[1]	12 (12x1)	**grant_inh and maint_int** - During the first half of the phase, this bit is used for grant_inh, during the second half of the phase, this bit is used for maint_int. During the first cycle, the grant inhibit control bit is driven by the current bus master to extend the info cycles when the bus master is moving a block of data. The arbitration logic will not issue a grant to any other board when a board is driving this signal. This ensures that the current bus master will retain ownership of the bus for another cycle. During the second half of the cycle, this bit is used to signal a maintenance interrupt. Maintenance Interrupt indicates that some board in the system is requesting attention from the software maintenance process. Any board in the system may drive this signal during the second half of a bus phase regardless of bus mastership. All boards in the system will sample maintenance interrupt and use it to reset their arbitration priority.
control[2]	12 (12x1)	**bus_err_a and busy** - Assertion of this signal during the first half of Post2 signals that a bus error was detected on the info bus associated with this particular control bus (n,o,p). Any operation in Arb, Info, or Post1 will be aborted. Operations in Post2 are suspended while the error protocol runs, and then will return to the Info phase. See section 6.4 on page 30 for more information on bus_err. The CPU initiator of a peer-to-peer operation must track the entire operation to see whether the cycle is errored in either the send or echo portion of the transaction. Assertion of this signal during the second half of Post2 indicates to the bus master that the operation should be aborted and re-tried at a later time. Busy is ignored for the send portion of a peer-to-peer operation. The CPU initiator of a peer-to-peer operation must track the entire operation to see whether the cycle is busied.

Table 3. Control Bus Signals

signal	width	description
control[3]	12 (12x1)	**bus_err_b and funny_state-** Assertion of this signal during the first half of Post2 signals that a bus error was detected on the B bus connected to this board. Any operation in Arb, Info, or Post1 will be aborted. Operations in Post2 are suspended while the error protocol runs, and then will return to the Info phase. See section 6.4 on page 30 for more information on bus_err. The CPU initiator of a peer-to-peer operation must track the entire operation to see whether the cycle is errored in either the send or echo portion of the transaction. During the second half of the cycle, this bit is used to signal that a board has just gone unbroken. The board will continue to assert this signal until it has seen eight phases without a bus error occurring. At that point the board will stop asserting this signal. Then all other boards will treat this board as an active, responding board. This prevents a board that is going unbroken from responding to bus errors in the middle of an error sequence that is already underway.

control[7:4]	48 (12x4)	**checkbits** - The top 4 bits of the control bus are checkbits generated from the lower 4 bits of control signals.
		
		The checkbit algorithm is:
		
		control[4] =control[0]^control[1]^control[2];
		
		control[5] = conrtol[0]^control[1]^control[3];
		
		control[6] = control[0]^control[2]^control[3];
		
		control[7] = control[1]^control[2]^control[3];

TOTAL	**108**	

4.4 The Voted Signals

The voted signals are the only signals driven through 26S10 transceivers. These signals are point to point and terminated at each end, so that insertion of an unpowered board does not disturb the termination (and timing) of a net in use. Only the _x_ versions are listed; there are _y_ and _z_

Table 4. 3-Way Voted Signals

signal	total	description
reset_0_1_x_, reset_0_2_x_, reset_0_3_x_, reset_1_0_x_, reset_1_2_x_, reset_1_3_x_	18 (2x3 x3)	**reset** - There are separate 3-way voted triplets from each CPU to the other three boards in the system. When a RECC needs to reset the system, all lines go active. When a CPU wants to reset another board, only the lines going to that board are active.
board_not_broken_0_1_x board_not_broken_0_2_x board_not_broken_0_3_x board_not_broken_1_0_x board_not_broken_1_2_x board_not_broken_1_3_x board_not_broken_2_0_x board_not_broken_2_1_x board_not_broken_2_3_x board_not_broken_3_0_x board_not_broken_3_1_x board_not_broken_3_2_x	36 (4x3 x3)	**broken status** - This three way voted signal is driven from each board to each other board. It is driven when a board is alive and not broken in the system and is used to determine which buses are active. The C-side ASICs drive the signals and the D-side ASICs drive the output enables for the 26S10s. This organization guarantees that board_not_broken is deasserted whenever either side of the board thinks that the board is broken,. The receiving board votes the x, y, and z signals.CPU0 in slot0 drives board_not_broken[0], etc.
sync_x_	3 (1x3)	**sync status** - These signals are on the CPU only and are used when synchronizing a pair of CPUs to enter the duplexed state.
even_online_x_, odd_online_x_	6 (2x3)	**on-line status** - These signals are on the CPU only and are used by the CPU boards to communicate to each other which CPU board(s) are in the on-line state. Even_online_ is asserted when the CPU in slot 0 is on-line; odd_online_ is asserted when CPU in slot 1 is on-line.
TOTAL	**63**	

See Fig 5 & 6 for voting

signals making up the triplet.

4.5 Other Control Signals

Table 5. Miscellaneous Control Signals

slot_ida slot_idb	2	**slot id** - The slot ID signals are hard wired on the backplane for each slot. There is one duplicated slot id. Since the slots are dedicated in Polo, it is only necessary to determine if a board is in an even or an odd slot. These two bits will be registered and checked by each ASIC at reset and will not be sampled again. If an error is detected at reset, the board will break and hence will never be capable of being brought on-line.
xb_clk8	1	**system clock** - This is the system clock received by the Sentry clock chip and used to generate the board clocks. It is generated by the backplane clock oscillator. This clock runs at 8MHz, so every board in the system will be in sync with each other and there will be no need for additional synchronization clocks to be passed along the backplane. The clock is pulse width modulated so that 4Mhz can be generated.
slot0_ta_d, slot0_ta_c_, slot1_ta_d, slot1_ta_c_	4	**ta signals** - These "ta" signals are only present on CPU boards and are sent between a duplexed board pair for early detection of the boards going out of lockstep. These explicitly named backplane signals are used in place of the Golfbus pair_comm[7:4] signals, which carry the ta information in Golfbus systems.
TOTAL	7	

5. Xbus I/O Interface Registers

A complete description of the Polo register set can be found in the Polo Programming Guide.

6. Xbus Protocol

6.1 Overview

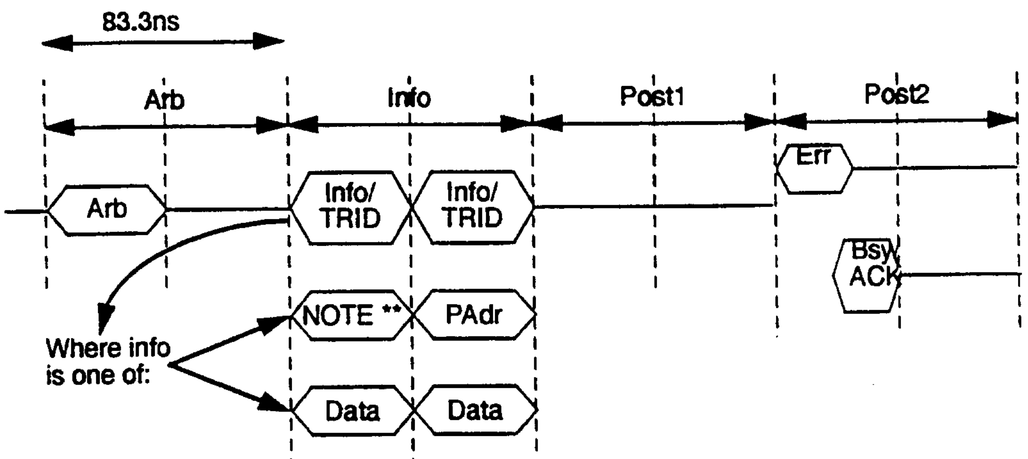
The Xbus is a point-to-point synchronous, pipelined, multiplexed address/data, error detecting bus with split-transaction capabilities. Function-code, address and data are parity and loop-back checked. Control lines are ECC protected and loop-back checked. Three-way voting is implemented on the reset, clock, and broken indicator lines.

The bus supports word accesses and has a block transfer capability for support of cache line accesses. The Xbus has a logical 32-bit wide data/address bus.

6.2 Bus Operation

The basic component of all Xbus transactions is the operation. An operation is composed of four phases as illustrated in figure 7: arb, info, post1, and post2. Two information transfers can occur on the bus during each phase; this is referred to as "double pumping". The double pump frequency is approximately 24MHz. The figure below illustrates the logical activity on the bus. All information is directly registered in the ASICs without any external buffers.

Figure 7. Basic Xbus Cycle



** virtual index, function code, remote/coherent bits, and byte enables or I/O

The phases are used as follows:

Arb phase: boards drive their arbitration request lines during the first half (cycle) of the arbitration phase. During the second half they determine whether they won arbitration and prepare for the info phase.

Info phase: For non-IOVA address transfers, boards drive the virtual index, function code, remote/coherent bits, and byte enables during the first half of the info phase and the physical address during the second half. For IOVA address transfers (IOVA bit in the trid is true), boards drive the IOVA during the first half of the info phase and deterministic data with good parity during the second half; the physical address is gotten from the I/O address map RAM look-up. For data transfers, data is driven during both the first and second halves of the cycle. Note that non-cache consistent address transfers need not supply a virtual index though the driven information must be deterministic and parity will be computed across it.

Post1 phase: During this phase, boards are determining whether any error conditions existed with the info phase and whether there is need to BUSY the operation. CPU boards map the device index portion of the IOVA to obtain the full physical address and virtual index of an I/O board's transfer for IOVA address transfers.

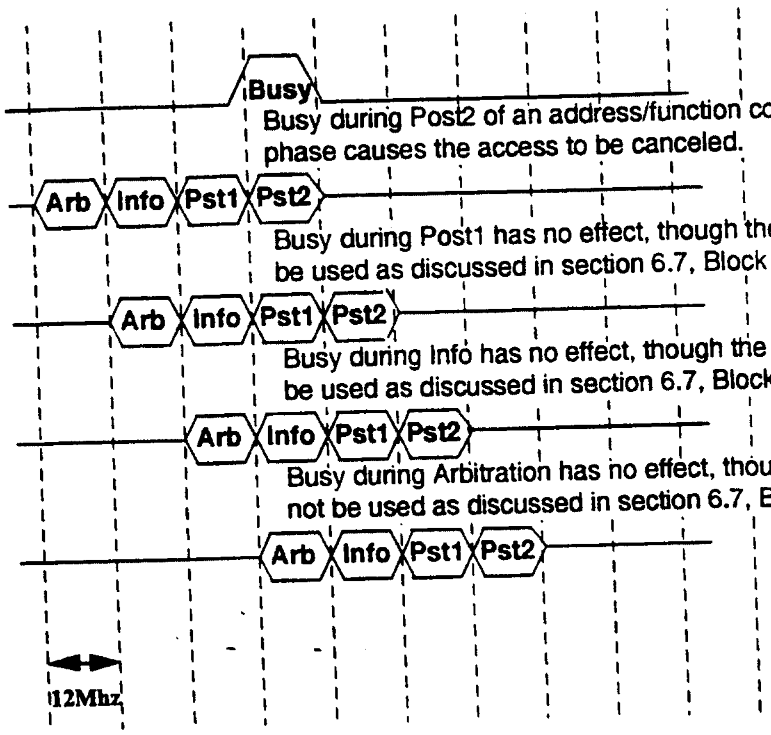
Post2 phase: Any board detecting an error with the info phase drives the error lines during the first half. If a board does get errored, it next goes to the error sequence phases to determine the source of the error. Any board detecting the need to BUSY an address/function code driven during the info phase drives BUSY during the second cycle of this phase. It is also during this phase that accesses are acknowledged (this is described in greater detail in someothersection).

6.3 Bus Busies

Figure 8 illustrates the effect of bus BUSYs on the basic bus cycle. As shown in the figure, BUSY has no effect on a bus operation during any phase except for post2; a BUSY during post2 will cancel the bus operation. Note that busys for multiple cycle bus operations, such as block transfers, have special rules and are described in section 6.7.4 on page 49.

Should a cycle be both BUSYed and ERRORed, the ERROR takes precedence.

Figure 8. Basic Bus Busy Operation

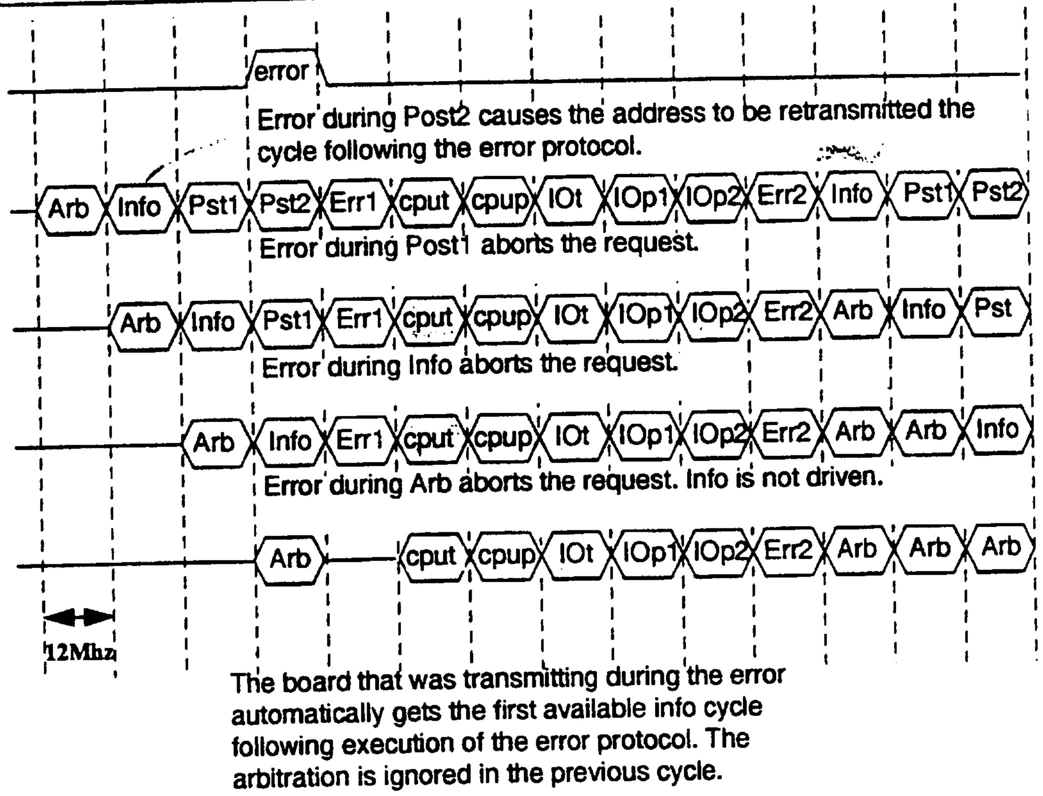

Busy
Busy during Post2 of an address/function code info phase causes the access to be canceled.
Arb
Info
Pst1
Pst2
Busy during Post1 has no effect, though the info may not be used as discussed in section 6.7, Block Transfers,
Busy during Info has no effect, though the info may not be used as discussed in section 6.7, Block Transfers,
Busy during Arbitration has no effect, though the info may not be used as discussed in section 6.7, Block Transfers,
12Mhz

6.4 Bus ~~Errors~~ Protocol

This section covers bus errors on the info bus, for information on the control bus see Section 9.2 on page 84.

Figure 9 shows the effect of bus errors on the basic bus cycle.

Figure 9. Basic Bus Error Operation


error
Error during Post2 causes the address to be retransmitted the cycle following the error protocol.
Arb
Info
Pst1
Pst2
Err1
cput
cpup
IOt
IOp1
IOp2
Err2
Error during Post1 aborts the request.
Pst
Error during Info aborts the request.
Error during Arb aborts the request. Info is not driven.
12Mhz
The board that was transmitting during the error automatically gets the first available info cycle following execution of the error protocol. The arbitration is ignored in the previous cycle.

Since the Xbus has no transceivers, the loopcheck phase of the Golfbus error protocol (Post4) has been modified to allow each board an opportunity to verify its transmit and receive capabilities. This has resulted in new states being added to the bus error operation. These states are described below:

Err1: The Err1 state is entered on the cycle after a bus error is detected. This state is used to allow for time to turn off the info bus before the loopback checks are performed. A board that is in its info phase during Err1 will disable its output enables half way through the phase.

CPUTest: The CPUTest state is used to test the CPU's ability to drive patterns on the Xbus. On the first cycle of the phase the CPU will drive 55 on the info bus, 55 on the trid bus, 1 on the parity line and 0 on the func_op line. On the second cycle of the phase the CPU will drive AA on the info bus, 2A on the trid bus, 0 on the parity line and 1 on the func_op line.

CPUPost: The CPUPost state is used to turn the bus around between the CPU's loopback check and the I/O boards loopback check. This phase is also used as a Post1 cycle for the CPU's loopback pattern.

IOTest: The IOTest state is used to test the I/O board's ability to drive patterns on the Xbus. On the first cycle of the phase the I/O board will drive 55 on the info bus, 55 on the trid bus, 1 on the parity line and 0 on the func_op line. On the second cycle of the phase the I/O board will drive AA on the info bus, 2A on the trid bus, 0 on the parity line and 1 on the func_op line. Bus errors from the CPUTest phase are reported during this phase. This information is used to evaluate the bus, CPU, and I/O board at the end of the error sequence. The last I/O ASIC to drive the data bus drives the bus during the IOTest phase.

IOPost: The IOPost1 state is used to evaluate the IOTest data.

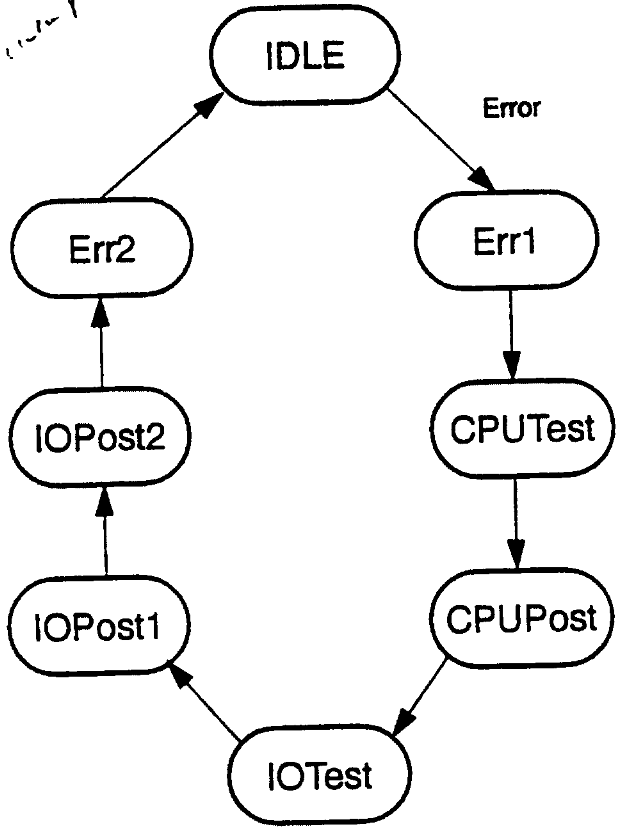
IOPost2: The IOPost2 state is used to transmit any bus errors from the IOPost1 state. This information will be used to make an intelligent decision about how to deal with the error.

Err2: The Err2 state is used to evaluate the information from the loopback checks. Bus errors from CPUPost and IOPost2 as well as information shared between the C and D sides of each board are used to determine what course of action to take. This set of actions will be described later in this section.

Figure 10 shows the basic state machine and state transitions for the bus error handler.

Figure 10. Bus Error Operation Flow Chart

Transceivers [illegible] been eliminated - have duplexed bus to look at so have to implement other [illegible] protocol.



The key challenge for the bus error algorithm on the Xbus is to diagnose errors so that system operation can continue. Unlike previous systems that use duplicated buses to allow all functional units a guaranteed path for communications, when the Xbus removes a bus from service, *it must also remove one of the two units attached to that bus.* In some cases, the right thing to do is obvious. In other cases, it is not. The following sections analyze various faults, how they are handled and how they manifest themselves.

6.4.1 Bus Error Broken Conditions

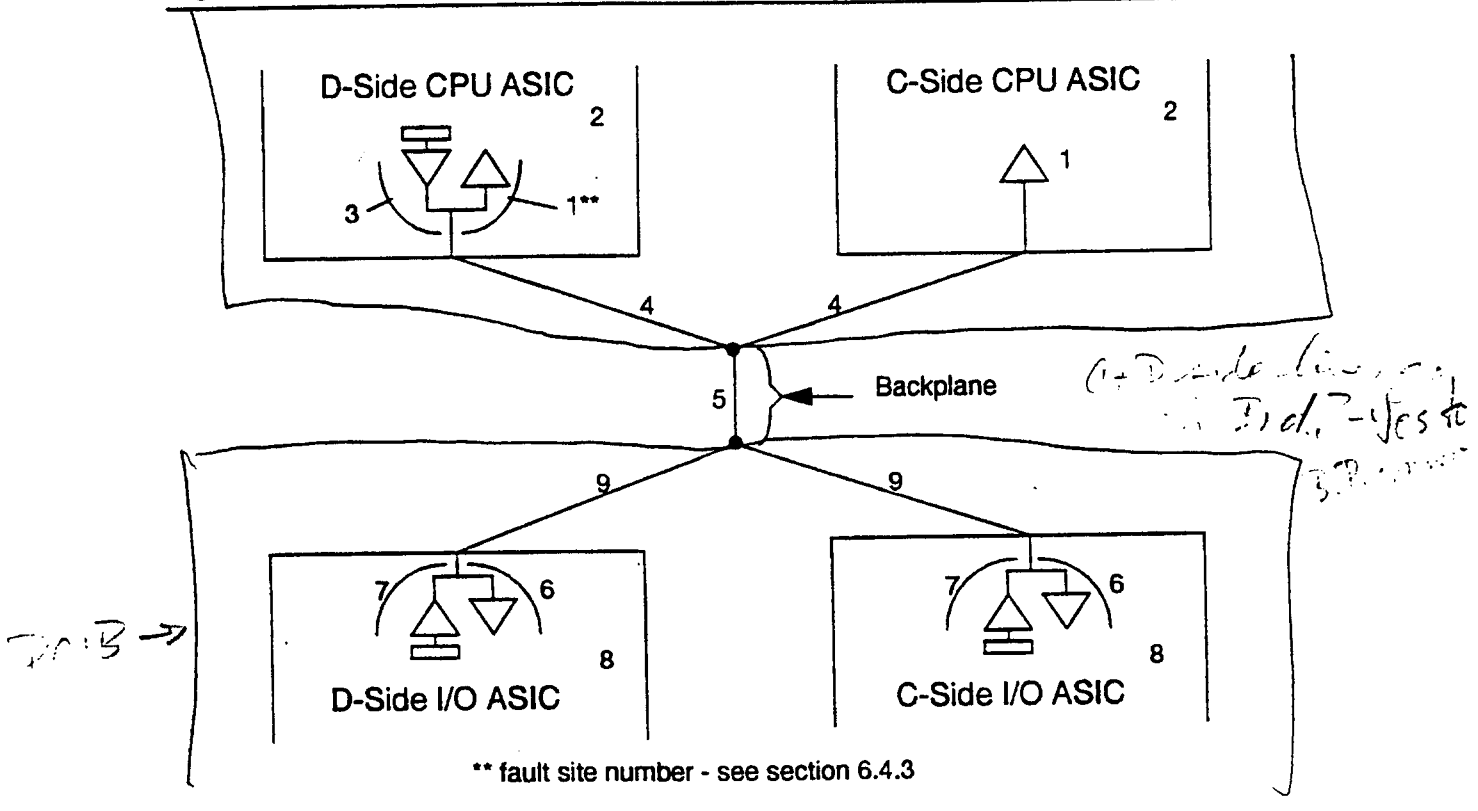
At this point it would be helpful to classify the different types of conditions that cause a board to go broken when a bus is detected bad. For a full discussion of the broken logic, see Section 9. on page 80.

- **loopback on control** - C and D ASICs must always agree on what to drive on the control lines, including whether or not to assert bus error. If one ASIC asserts bus error and the other side does not, the board breaks.
- **loopback on data** - C and D ASICs must always agree on what to drive on the duplicated info lines. When driving CD same data, ASICs compare the data they drive with the data they receive. An ASIC asserts bus error on parity errors when receiving data, and parity errors and loopback errors when driving data. Loopback checking is disabled when a board drives "CD different" data, such as the contents of error reporting registers or data from PCI cards. The board breaks if the two sides disagree on which bytes have or do not have errors,.
- **arbitrary** - Break the designated board in Err2 when there are bus errors signaled during CPUtest and IOtest and no board has broken by the end of IOpost2. This is called an arbitrary shoot because the fault is most likely on the backplane, so it is arbitrary as to which of the two boards connected to the faulty bus to break. Typically, the CPU is set broken, so that the system can continue with all of its I/O available, but if bit 21 of the Bus Interface State register (Section 12.3.4 on page 87 in the Polo Programming Guide) is set then the PCIB board will be the designated board.
- **heuristic** - a board breaks itself during Err2 if there is a bus error when it drives, but no bus error when the other board drives, and the other board did not break by the end of IOPOST2.

6.4.2 Xbus Fault Analysis

In order to understand various faults and what they can mean, it is important to present a detailed block diagram of the Xbus interconnect. Figure 11 show the interconnect for a typical Xbus line.

Figure 11. Xbus Interconnect



** fault site number - see section 6.4.3

The black dots in figure 11 represent the connectors to the backplane. For fault tolerance and fault isolation reasons, it is important that the boards should be routed so that the etch between the D-side and the C-side runs through the connector connection. This limits the amount of etch on each board that cannot be isolated [illegible] to a minimum. On the CPU board, one ASIC both drives and receives a given net while the other ASIC only receives that net. On the I/O board, each ASIC can potentially drive every net. The CPU ASICs are always in lockstep and therefore each ASIC is capable of sharing the data out load. However on the I/O board, each ASIC connects to a different PCI bus so a signal ASIC may need to drive the entire Xbus. There are cases in normal operation when only one CPU ASIC will drive the entire bus.

6.4.3 Fault Conditions

The following sections identify all known fault conditions and describe their handling. Refer to figure 11 to determine the location of the fault site indicated.

6.4.3.1 CPU Board Faulty Input Circuit - CPU Driving

- fault site 1
- break via loopback on control fault

This fault deals with a fault in the input section of one of the CPU ASICs. In this case, the fault occurred during or just before a cycle in which the CPU drove the info bus. The error is detected when the CPU drives the bus. The ASIC with the faulty circuit will signal a bus error during the Post2 phase of the cycle and the other side ASIC will not. The board will go broken and drive bus error during Err1. The error sequence will be executed, and the operation will be retried by the partner CPU with no error.

6.4.3.2 CPU Board Faulty Input Circuit - I/O Board Driving

- fault site 1
- break via loopback on control fault

This fault deals with a fault in the input section of one of the CPU ASICs. In this case the fault occurred during or just before a cycle in which the I/O board drove the info bus. If the error is a multi-bit error that evades the parity logic, the error will be caught internal to the CPU board and the CPU board will go broken. If the error is a single bit error the faulty ASIC will detect a bus error during the Post1 phase of the transfer. The ASIC will drive bus error during Post2 of the transfer and the otherside of the board will not. The board will break with a loopback on control failure in the next phase. After the error sequence, the operation will be retried by the partner CPU with no error.

6.4.3.3 CPU Board Different Data C-Side and D-side

- fault site 2
- break via loopback on data fault

This fault deals with an internal CPU fault that results in different data being driven out of each ASIC. The error is detected when the CPU drives the bus. The C and D ASICs will trade error status and disagree on where the error is during Post1; both C and D sides will see an error on bytes the other side drives but no error on the bytes it drives. The board will go broken and drive bus error during Post2. The error sequence will be executed and the operation will be retried by the partner CPU with no error.

6.4.3.4 CPU Board Faulty Output Circuit - Buffer to Pad Fault

- fault site 3

- break via heuristic broken

This is a fault in the output section of the CPU ASIC resulting from an output driver circuit fault that blows in a manner that causes an internal open between the output driver and the ASIC pad, while not disrupting the functionality of the input receiver. All ASICs on the bus will detect a bus error during the Post1 phase of the transfer. The ASICs will drive bus error during Post2 of the transfer. All ASICs will detect a bus error during the CPUPost phase. No bus errors are detected during the IOpost1 phase. The CPU board will go broken during Err2 based on the fact that it detected a bus error when it drove the bus, no bus error when the I/O board drove the bus and the I/O board did not go broken after its error sequence.

6.4.3.5 CPU Board Open - CPU Board Driving

- fault site 4
- break via loopback on data fault

This fault results from an open due to either a broken etch or a lifted pin on the CPU board.The routing is very important for this class of fault. The etch between the C-side and the D-side should be routed through the connector pin. This limits the possibility that an open on the CPU board is mistaken to be an open on the backplane. In this case the fault occurred during or just before a cycle in which the CPU drove the info bus. The error is detected when the CPU drives the bus. During the Post1 phase, the driving ASIC will not see an error but the checking ASIC will signal a compare error. During Post2 the board will go broken and drive bus error. The operation will be retried by the partner CPU with no error after the error sequence.

6.4.3.6 CPU Board Open - I/O Board Driving

- fault site 4
- break via loopback on control

This fault results from an open due to either a broken etch or a lifted pin on the CPU board. The routing is very important for this class of fault. The etch between the C-side and the D-side should be routed through the connector pin. This limits the possibility that an open on the CPU board is mistaken to be an open on the backplane. In this case the fault occurred during or just before a cycle in which the I/O board drove the info bus. The ASIC with the open between it and the connector will detect a bus error during Post1. During Post2 one ASIC will assert bus error and the other will not, causing the board to break. The operation will be retried by the partner CPU without any error after the error sequence.

6.4.3.7 CPU Board Short

- fault site 4
- break via arbitrary broken

This fault deals with a short on the CPU board. When the fault occurred and who was driving the info bus during the fault are not relevant to this class of fault. During Post1 of the transfer all ASICs on the bus will detect a bus error. The ASICs will drive bus error during Post2 of the transfer. Both ASICs on the CPU will detect a loopback error during the CPUPost phase and the ASICs on the I/O board will signal a bus error during IOTest. The ASICs will detect a bus error during the IOpost1 phase and signal a bus error during IOPost2. During Err2 the designated board will go broken based on the fact that it has detected bus errors during the error sequence and no other boards went broken after the error sequence.

6.4.3.8 Backplane Open Etch

- fault site 5
- break via arbitrary broken

When the fault occurred and who was driving the info bus during the fault are not relevant issues for this class of fault. During Post1 of the transfer some ASICs on the bus will detect a bus error and drive bus error during Post2. Both ASICs on the I/O board will detect a bus error during CPUPost. The CPU ASICs will detect a bus error during IOpost1. During Err2, the designated board will go broken based on the fact that it has detected bus errors during the error sequence and no other board went broken during the error sequence.

6.4.3.9 Backplane Short

- fault site 5
- break via arbitrary broken

When the fault occurred and who was driving the info bus during the fault are not relevant issues for this class of fault. All ASICs on the bus will detect a bus error during Post1 and drive bus error during Post2. All ASICs on the bus will detect a bus error during CPUPost and IOpost1. During Err2, the designated board will go broken based on the fact that it has detected bus errors during the error sequence and no other board went broken during the error sequence.

6.4.3.10 I/O Board Faulty Input Circuit

- fault site 6
- break via loopback on control

This fault deals with a fault in the input section of one of the I/O Board ASICs. For this particular fault, it is irrelevant who was driving the backplane when the fault was detected. The faulty ASIC will detect a bus error during Post1. During Post2 of the transfer, the faulty ASIC will drive bus error and the other ASIC will not, causing the board to go broken. If it was a CPU initiated request the operation will be retried by the CPU with no error after the error sequence. If it was a request initiated by the I/O board, then the request will be dropped.

6.4.3.11 I/O Board Output Circuit Fault - Buffer to Pad

- fault site 7
- break via heuristic broken

This is a fault in the output section of the I/O board ASIC. This class of fault results from an output driver circuit fault that blows in a manner that causes an internal open between the output driver and the ASIC pad, while not disrupting the functionality of the input receiver. All ASICs on the bus will detect a bus error during Post1 of the transfer and drive bus error during Post2. No error is detected during the CPUPost phase. The I/O ASICs will detect a bus error during IOpost1. During Err2 the I/O board will go broken based on the fact that it has detected a bus error when it drives the bus, no bus error when the CPU board drove the bus and the CPU board did not go broken after its error sequence.

6.4.3.12 I/O Single-side Access - ASIC Parity Gen. Fault

- fault site 8
- break via internal parity generator broken

A fault in the parity generation logic during single side accesses (data driven entirely by either the

C or D ASIC) could cause the system to bus error forever. The otherside ASIC doesn't know the data, and has no way of checking the parity. For this reason, the info bus parity generation is duplicated and selfchecking inside of the Gambit ASICs. See section 9.1.1, Parity, on page 80 for details.

6.4.3.13 I/O Single-side Access - ASIC PCI Data Path Fault

- fault site 8
- break each PCI slot due to checksum and map RAM errors.

This is a fault within the ASIC's PCI data path. This hardware is not running in lockstep with the other side of the board and therefore is not self-checking. Eventually bad addresses produced by the PCIB will cause map RAM errors and/or data checksum errors in the CPU ASIC. The CPU ASIC noACKs accesses that cause map RAM errors, and the PCIB sets off-line (breaks?) the PCI slot that originated the noACKed cycle based on an option bit in the Configuration register in page zero of SAM compatible I/O space. Eventually, all PCI slots handled by the defective ASIC will be set broken. In the meantime corrupted I/O data is detected by checksums and handled by higher level protocols.

6.4.3.14 I/O Non-single-side Access, Different C-D Data

- fault site 8
- break via loopback on data

This fault deals with an internal I/O board fault that results in different data being driven out of each Gambit ASIC during a regular (not single-side) read return. Each ASIC will disagree with the data driven by the other side and the board will break with a loopback on data error. The error sequence will be executed and the operation will be retried by the CPU and get noACKed.

6.4.3.15 I/O Board Open

- fault site 9
- break via loopback on control

This fault deals with an open, either from a broken etch or a lifted pin on the I/O board. The routing is very important for this class of fault. The etch between the C-side and the D-side should be routed through the connector pin. This limits the possibility that an open on the I/O board is mistaken to be an open on the backplane. The driving ASIC sees no errors and the otherside ASIC asserts bus error; the board breaks the following phase with a loopback on control fault. After the error sequence the operation will be retried with no error and get noACKed.

6.4.3.16 I/O Board Short

- fault site 9
- break via arbitrary broken

This fault deals with a short on the I/O board. When the fault occurred and who was driving the info bus during the fault are not relevant issues for this class of fault. During Post1 of the transfer all ASICs on the bus will detect a bus error and then drive bus error during Post2. All ASICs on the bus signal a bus error during CPUpost and IOpost1. During Err2, the designated board will go broken based on the fact that it has detected bus errors during the error sequence and the other board did not go broken after its error sequence.

6.4.3.17 Transient Fault

- fault site anywhere
- action depending on fault site and timing

This fault deals with a transient fault. A transient fault is defined as a fault that is detected, but cannot be reproduced by the error sequence. If the fault is on the driving board and it is caught by loopback check, that board will go broken. If this is not the case then no board will break until Err2, then the designated board will go broken.

6.4.3.18 Slow Driver Fault

- fault site 4,9
- possibly break the wrong board with a loopback on control fault

When an ASIC with a marginally slow driver drives the bus, four ASICs clock in data from the net while the net is changing. Due to differences in speed between ASICs from different lots, its possible that some of the ASICs will detect a bus error, and some won't, resulting in a loopback on control fault. This may result in one or both of the boards on the bus going broken.

6.4.3.19 Board Not Broken Generation

The board_not_broken_out signal is generated by all boards. If a board is going to break the normal operation is for the board to assert bus error. All boards will enter the error sequence and the board that was going to break will de-assert board_not_broken_out during the error sequence. It is possible that a board could break without asserting bus error. This could only happen if there is a problem with asserting bus error on the d-side gate array (for instance the clock line going to the flip-flop that produces the control signal (bus error) opens). Please refer to figure 50, Board Breaking Timing and latching (arrived at phase_12_1) on page 89 for detailed broken timing diagrams.

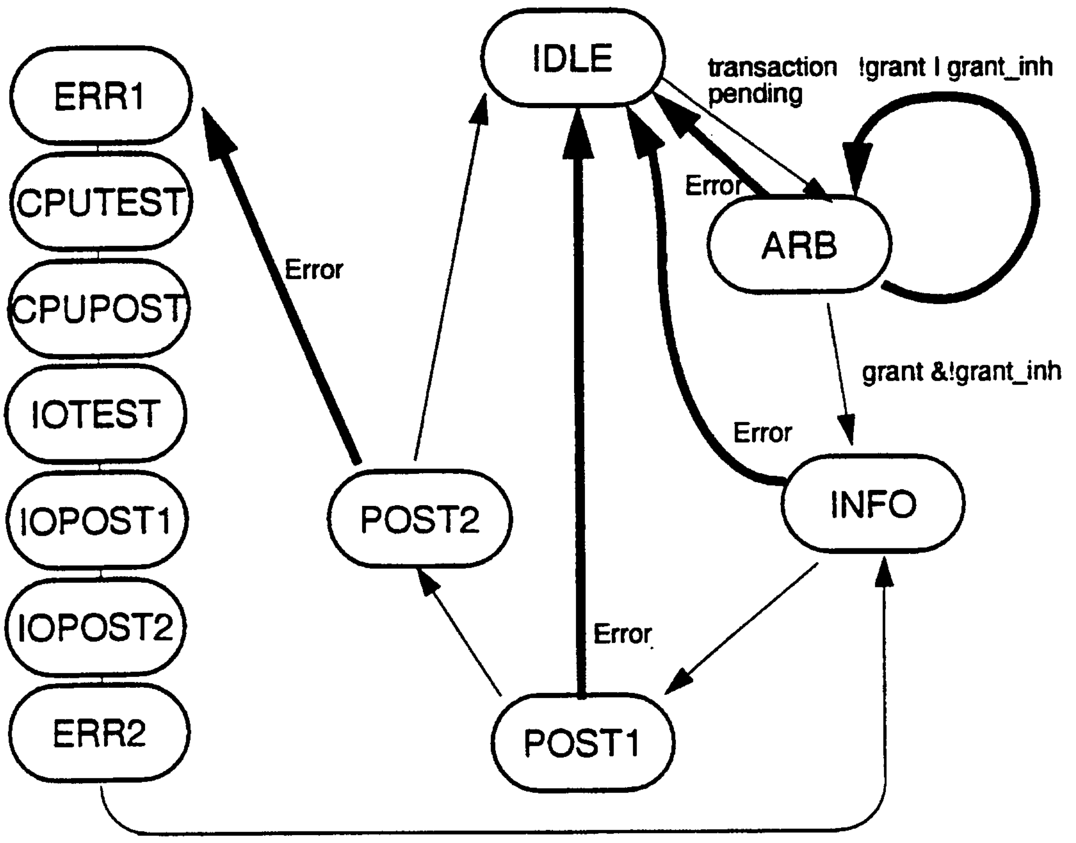
6.4.3.20 Arbitrary Breaking

There are a series of cases mentioned before that error logic cannot determine where the fault is. For these cases the goal is to not use the bus that the two boards are connected to. The default is to break the CPU board (designated board). This makes sense because there is a partner running in lock-step so no connectivity is lost. It is possible that the CPU board will break but the fault really lies with the PCIB. The only way this will pose a problem is if the failure in the PCIB effects the other bus that is still connected and is being used. Normally it is the CPU board that is the designated board, but if bit 21 of the Bus Interface State register is set then the PCIB board will be the designated board. THIS IS ONLY INTENDED FOR LAB DEBUG. UNDER NORMAL OPERATION THE DESIGNATED BOARD MUST BE THE CPU BOARD. IF THIS IS NOT ADHERED TO IT IS POSSIBLE TO HAVE BOTH BOARDS BREAK OR NO BOARDS BREAK.

6.5 Summary Bus State Diagram

A flow chart for the basic bus operation described in the previous sections is shown in figure 12. The thinner state transition lines represent "normal" operation and the thicker lines represent exception conditions (for the sake of this diagram, losing arbitration is considered an exception case).

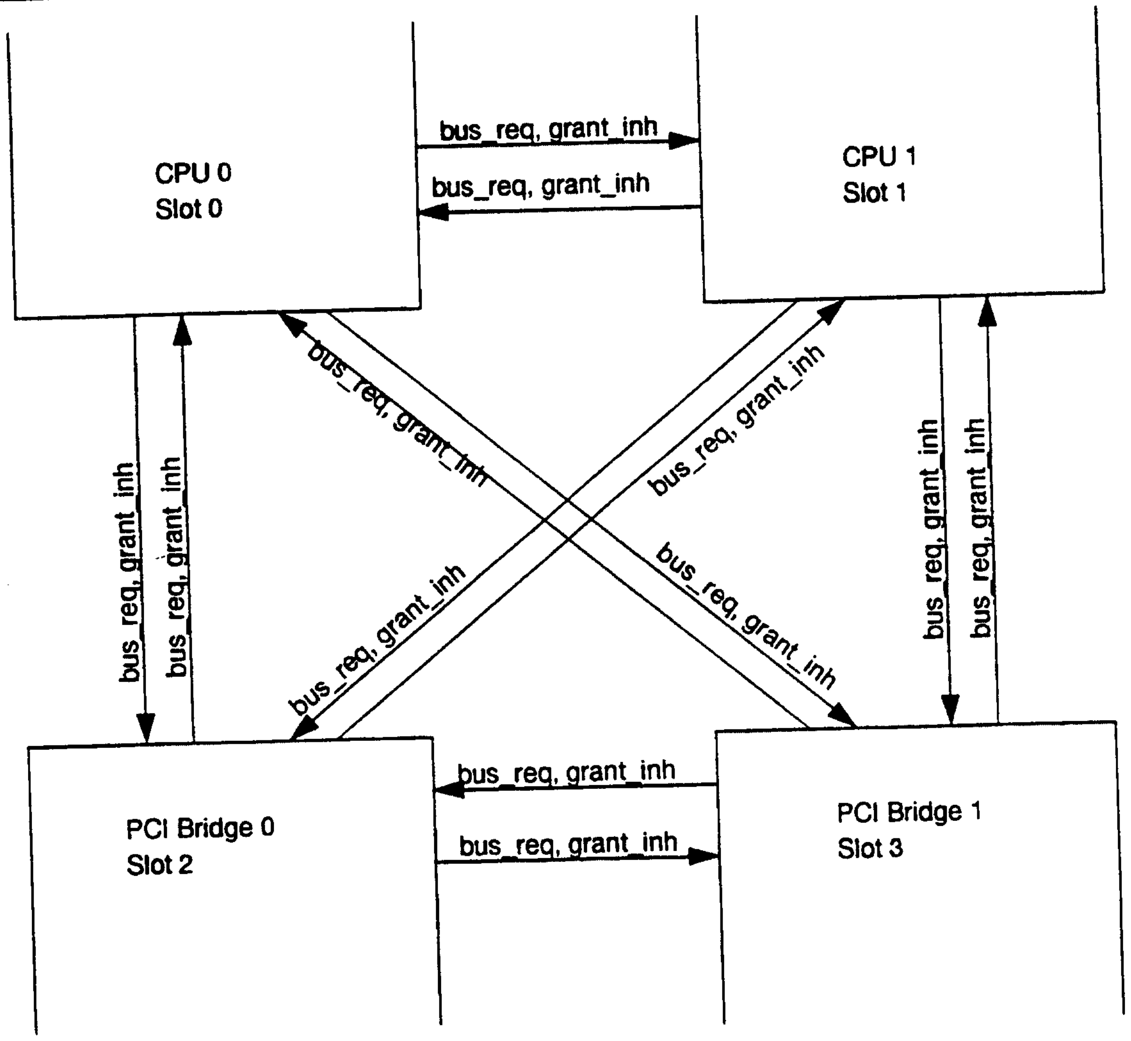
Figure 12. BASIC Bus Operation Flow Chart



6.6 Arbitration

The bus arbitration mechanism on Polo is the key to enforcing the synchronization of the four buses in the Polo architecture. The arbitration algorithm maintains the appearance of one functional bus. This is accomplished through a series of point to point request lines that are part of the control buses. Figure 13 shows the interconnect for the arbitration network.

Figure 13. Inter-CPU Bus Arbitration Network



This arbitration network is a break from previous Stratus systems. Unlike the binary search arbitration network used on Jetta and Stratabus based systems, Polo uses a distributed arbitration based on the request signals that each board drives.

As in Jetta, the determination of a bus grant depends on more than the request lines. Bus Grant can be inhibited by any board about to enter the data phase of a block transfer, or by an error on the bus. Thus any arbitration phase where grant inhibit or a bus error is asserted will not result in a grant for any board. The grant inhibit interconnect is also shown in Figure 13. Grant inhibit is also part of the control bus. A board driving grant inhibit will drive all three sets of its grant inhibit lines. This guarantees that all boards see the same grant inhibit and will make a correct arbitration decision.The **gmt_inh** signal is the mechanism used to guarantee that a block transfer will occur in successive clock ticks. gmt_inh is also use as the mechanism to enforce a dead cycle on the

Xbus to turn the bus around. A bus master will always drive gmt_inh when it is actively driving the bus. This guarantees that no device will be granted the Xbus the cycle after someone has driven info or data on the Xbus. A functional timing diagram which illustrates the use of **grnt_inh** is shown in figure 15, High Level Block Transfer on page 45.

Table 6. Arbitration Priority Rotation

Initial And Maintenance Interrupt Arbitration Value			
CPU 0	CPU 1	PCIB 0	PCIB 1
0	1	2	3
1	0	3	2
2	3	0	1
3	2	1	0

Table 6 shows the sequence of arbitration addresses that a board will follow when rotating it's address priority. Arbitration value zero has the highest priority and arbitration value 3 has the lowest priority. The sequence is such that adjacent boards are always assigned partner addresses, and that all boards will have each priority level at one time or another. For example, in a four slot system where all boards rotate arbitration priority together, every board will have the highest priority 1/4th of the time, the lowest priority 1/4th of the time, and each priority in between for (a total of) 2/4ths of the time in every cycle of address rotation.

The system maintenance interrupt signal is used to reset the rotating arbitration address on each board to the board's slot id. For example, when a new board is inserted into the system, all boards must have their arbitration addresses reset to be their slot addresses, since the new board has no means of knowing what its arbitration address must be other than its own slot address. When the progression through the addresses shown in Table 6 resumes, all boards will continue to arbitrate with unique arbitration addresses. Note that the assertion of maintenance interrupt has no effect on the actual arbitration period, therefore a node's priority at the beginning of the arb period may be different than at the end due to a maintenance interrupt. However, every node will always modify their arbitration priority, either from a maintenance interrupt reset or an end or arb period increment, synchronously on the same clock edge. Since a maintenance interrupt is issued when a board leaves reset, it is guaranteed that no board will try to arbitrate before it's arbitration address has been synchronized with the remainder of the system.

Note: There are other rotation sequences that could be argued to be "more fair" in that the priority will jump from a high priority to a low priority instead of progressing within a grouping of either high or low values, but in an effort to maintain simplicity this rotation scheme has been chosen.

The duplicate slot id signals on each board are latched at reset inside each ASIC and verified that they are identical. A miscompare between the two signals will break the board while it is in reset.

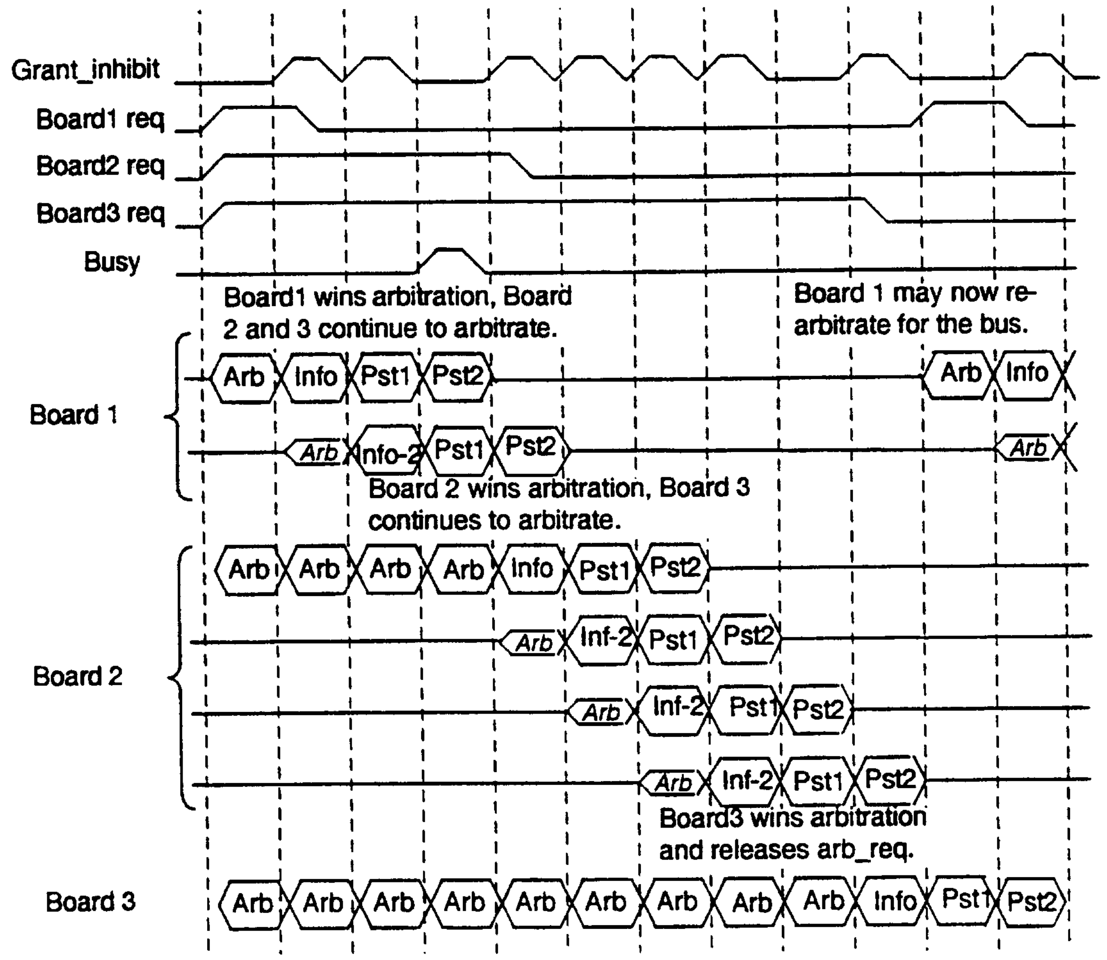
The actual arbitration address may be implemented with a simple two bit up-counter that is XORed with the slot id. This counter will increment at the end of each arbitration period on the bus. An arbitration period is the time when at least one board has a bus_req held active. A board can only win the bus once during any arbitration period. To clarify: some number of boards are arbitrating for the bus, with one board winning. The losing boards re-arbitrate, with one of those winning. This process repeats itself until all boards win arbitration. During this whole time, at least one bus_req is asserted. Any board that wins arbitration cannot arbitrate again until all bus_req signals have been de-asserted. Any boards that haven't arbitrated during this period may join the remaining boards. If all bus_req signals have been negated then all boards in the system have had the chance to be bus master once during that arbitration period. The arbitration period is now over and arbitration priority rotates.

A board always arbitrates against the other three boards in the system. If the CPU boards are duplexed, the duplexed pair arbitrates simultaneously. If either of the CPU boards wins the arb, both act as though they have won it, and both drive their info on the bus. This means that the duplexed CPUs will win half roughly of the arb cycles, and each PCIB will win one quarter of the arb cycles. When the CPUs are simplexed, each board will win roughly one quarter of the cycles.

During a bus error operation, as described in section 6.4, Bus Errors, on page 30, all nodes will de-assert their request and grant inhibit control signals. This will insure that the priority remains in tact during Xbus error recovery.

Figure 14 illustrates a case where **arb_req** prevents a board from re-arbitrating until each board that is requesting the bus gets a chance to be the bus master. Assume here that there are only three boards in this system. In this case board 1 has the highest priority, board 2 has the next highest, and board 3 has the lowest priority. All three boards begin to request the bus in the same cycle (although it should be noted that this is not a requirement for this type of situation to occur). Here board 1 wins the arbitration and its access is BUSYed, thus it must re-arbitrate for the bus but cannot until the current arbitration period has ended. Notice that at least one bus_req is held active until Board3 has been granted the bus, which guarantees that Board3 will get a chance to be busmaster before any other board returns to arbitrate.

Figure 14. Arbitration After Being Busied


Grant_inhibit
Board1 req
Board2 req
Board3 req
Busy
Board1 wins arbitration, Board 2 and 3 continue to arbitrate.
Board 1 may now re-arbitrate for the bus.
Board 1
Arb
Info
Pst1
Pst2
Info-2
Board 2 wins arbitration, Board 3 continues to arbitrate.
Board 2
Inf-2
Board3 wins arbitration and releases arb_req.
Board 3

Note that there is always a dead cycle between two different boards' info cycles. It is necessary to allow one cycle to turn the bus around to avoid unwanted bus fight.

6.6.1 Bus Mastership Upon Bus Grant

If a pair of duplexed boards arbitrate for the Xbus then when the higher priority peer wins the grant, both boards will assume mastership of the Xbus. However, only the higher priority board will actually drive the Xbus because only the Xbus lines driven by the board winning the grant are watched. This is due to the fact that a bus receiver cannot know whether or not the requesters are duplexed so it will not know if valid data will be present on the two buses sourced by the non-winning board. Therefore these two buses will be left tristated.

6.6.2 Special Arbitration Rules for Non-Paired Read Returns

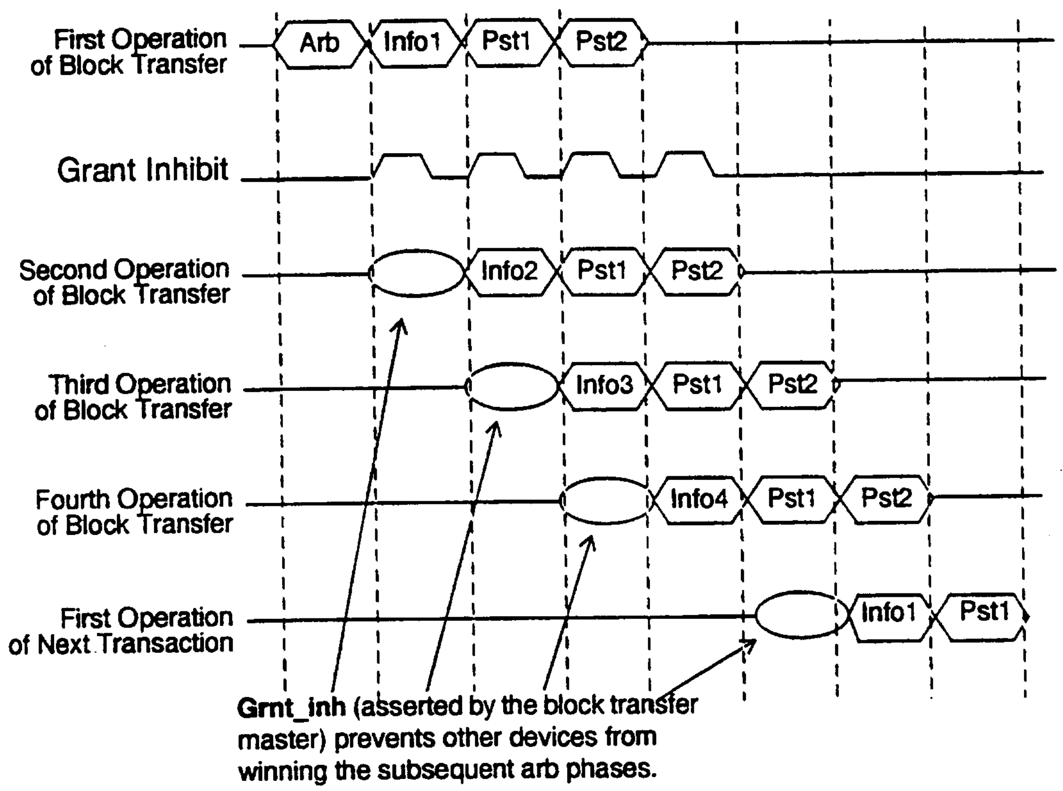
A duplexed board whose partner gets a non-paired register read request must go through the motions of returning the data in order to remain lock-stepped. However the non-target duplexed board in a non-paired read return must not arbitrate for the bus and instead wait for its partner to win the bus. According to Xbus only the buses attached to a board that won an arbitration phase are looked at. Only the target board of a non-paired access will drive bus request in order to guarantee that the correct data is looked at for a non-paired read return. The non-target board will continue to go through the motions on the read return but it will not drive the bus since only the board winning arbitration needs to drive.

6.7 Block Transfers

The Xbus permits boards to hold bus mastership for a block transfer via the grant inhibit (**grnt_inh**) signal. This does not allow for multiple block transfers. A board wanting to perform a block transfer asserts the **grnt_inh** signal the arbitration cycle after it first wins arbitration just as if it was doing a single cycle transfer. Then the signal is activated for as many additional arbitration phases as needed for the block transfer. This signal gives the board highest priority regardless of other arbitration activity, thereby permitting it to retain bus mastership for additional transfers. Note that the **grnt_inh** signal is used for all cycles, not just block transfers.

Figure 15 below provides a high level conceptual diagram of a block transfer which retains bus mastership for 4 bus operations.

Figure 15. High Level Block Transfer


First Operation of Block Transfer
Arb
Info1
Pst1
Pst2
Grant Inhibit
Second Operation of Block Transfer
Info2
Pst1
Pst2
Third Operation of Block Transfer
Info3
Pst1
Pst2
Fourth Operation of Block Transfer
Info4
Pst1
Pst2
First Operation of Next Transaction
Info1
Pst1
Grnt_inh (asserted by the block transfer master) prevents other devices from winning the subsequent arb phases.

6.7.1 Bus Errors

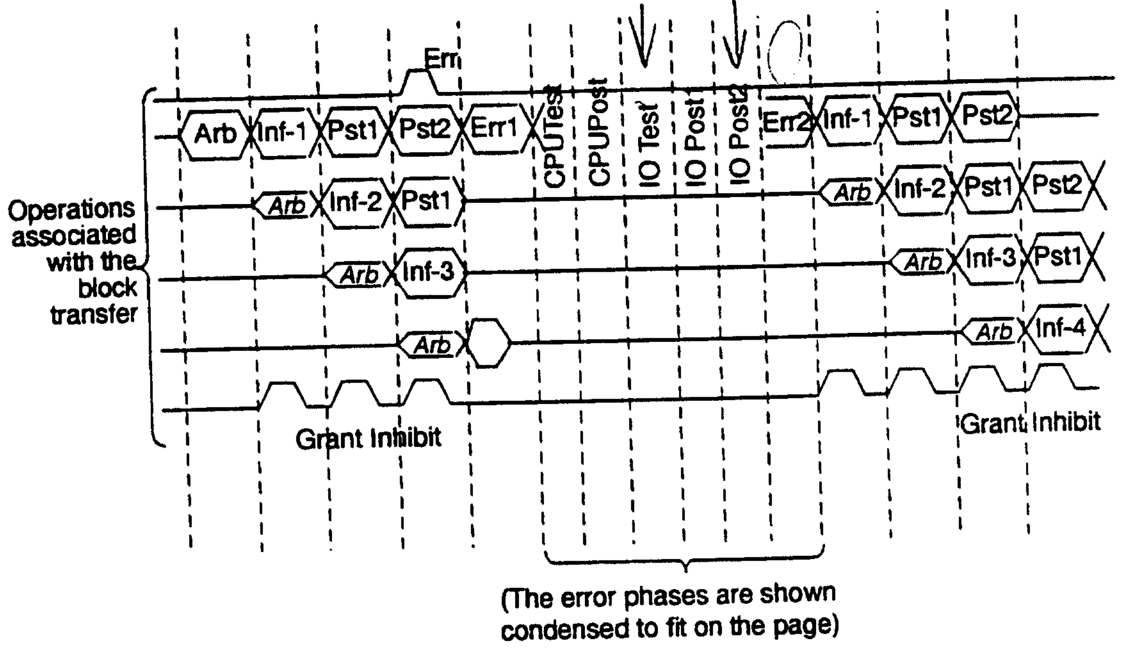
The Xbus block transfer error recovery mechanism is architected so that the same mechanism is applicable to block transfers of any length. The burden of error recovery is placed on the initiator of the transfer. All bus nodes will be responsible for de-asserting their **request** and **grant inhibit** control lines to insure that the arbitration priority remains in tact during the Xbus error operation. Figure 16 below illustrates what this means for a bus error during the first operation of a four operation block transfer.

- the master must retransmit any data associated with the block transfer maintaining correct order

- The **grt_inh** signal must be reactivated in time to keep the entire block transfer contiguous

As discussed later in this section, there are also situations wherein a bus error causes a block transfer to be canceled, requiring it to later be restarted from scratch, i.e. requiring it to rearbitrate for the bus.

Figure 16. Error During First Operation of a Quad Block Transfer


Err
Arb
Inf-1
Pst1
Pst2
Err1
CPUTest
CPUPost
IO Test
IO Post
IO Post2
Err2
Operations associated with the block transfer
Inf-2
Inf-3
Inf-4
Grant Inhibit
(The error phases are shown condensed to fit on the page)

Similarly, errors during the second, third, and fourth operation for a quad block transfer are illustrated in the figures that follow.

Figure 17. Error During Second Operation of a Quad Block Transfer

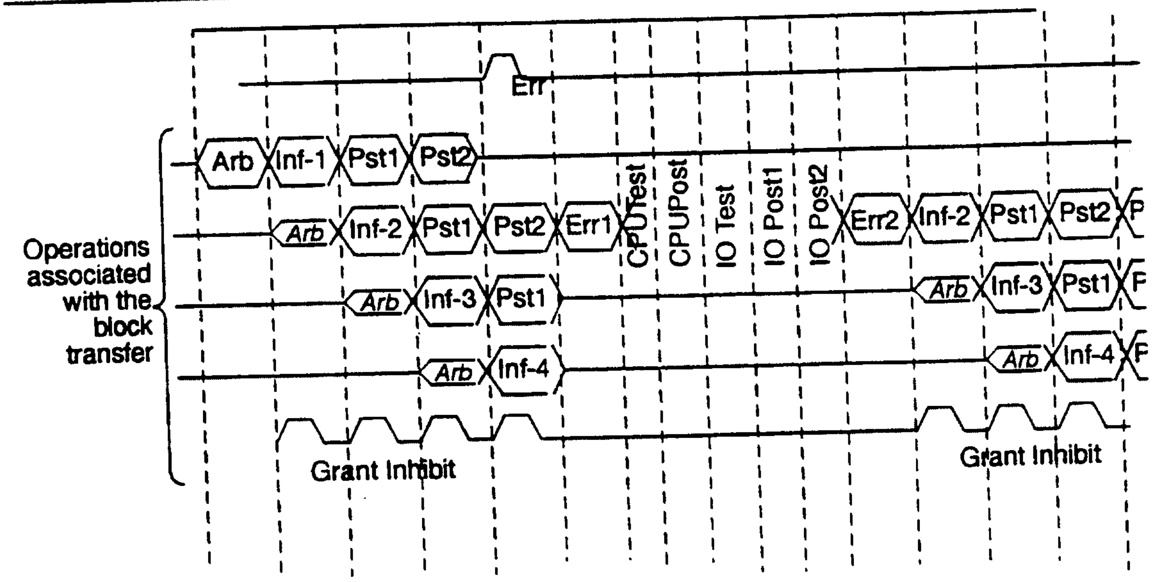


Figure 18. Error During Third Operation of a Quad Block Transfer

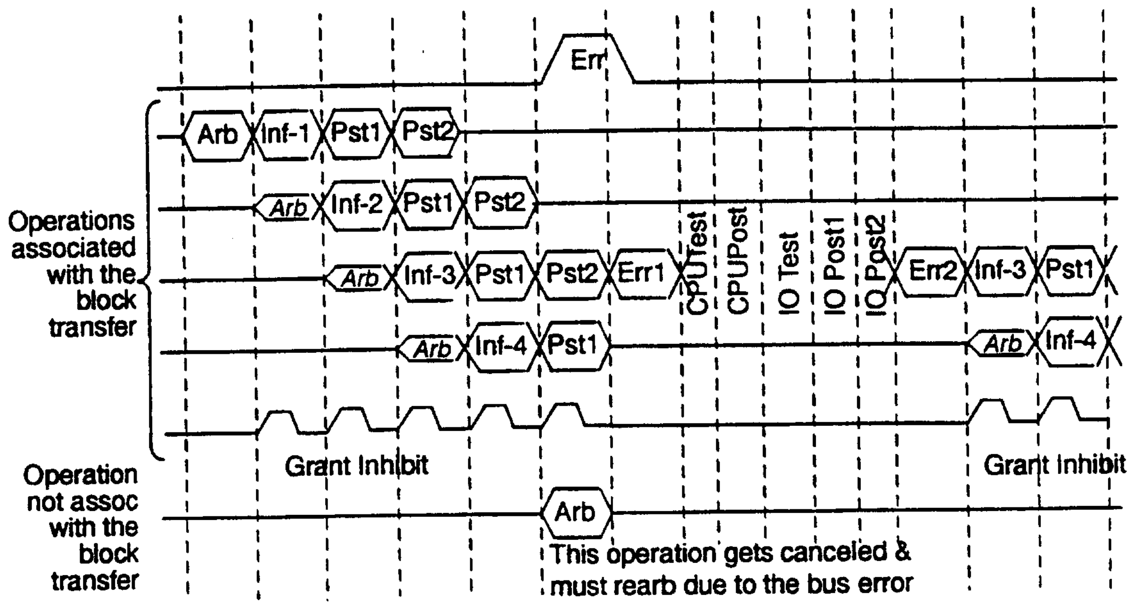
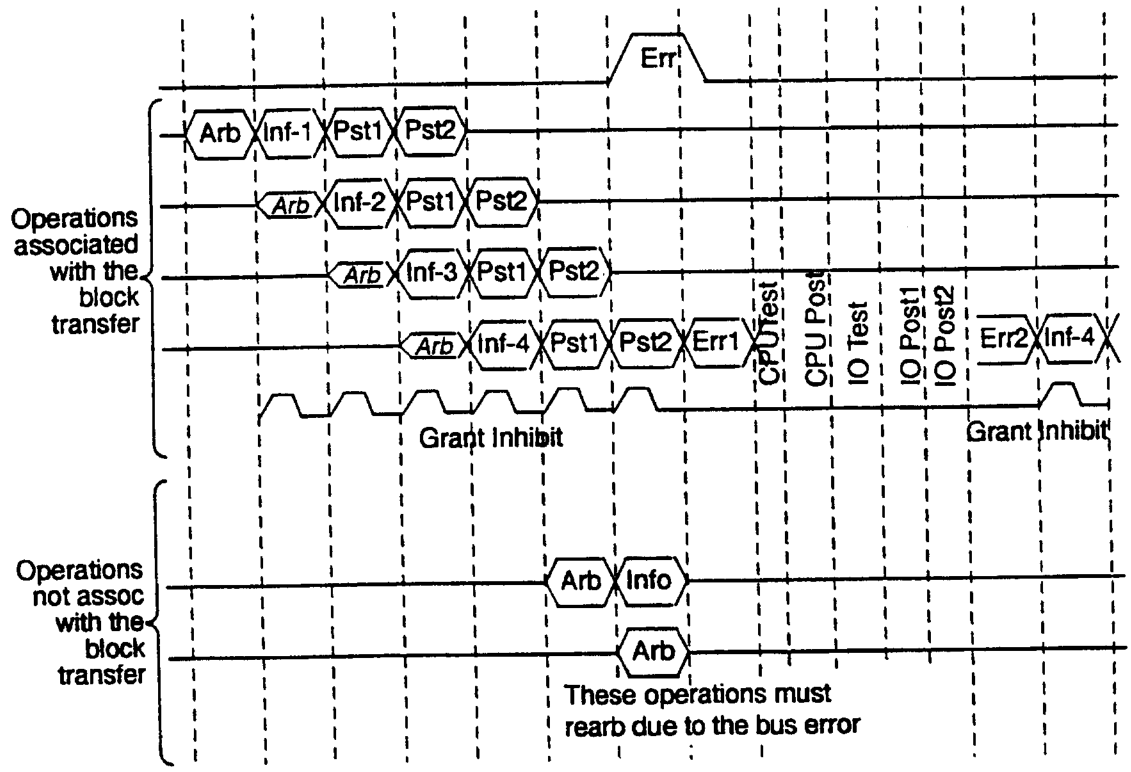


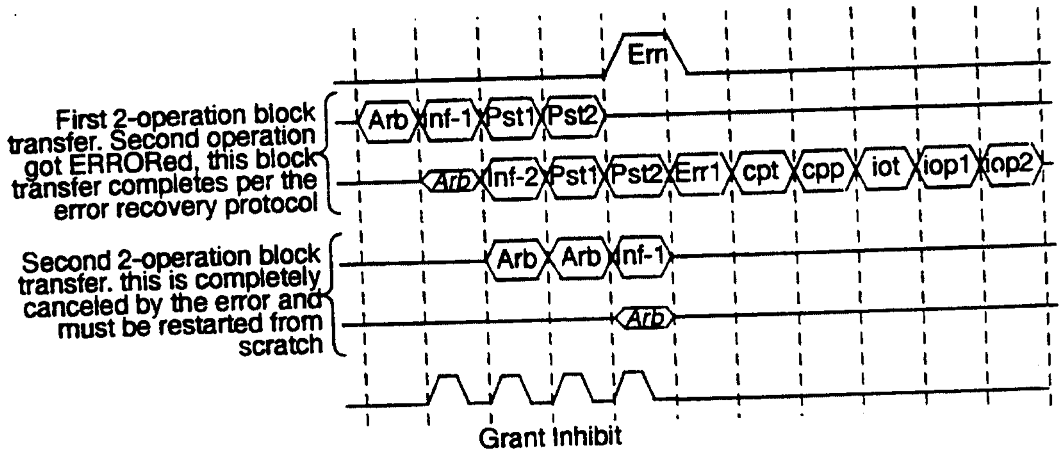
Figure 19. Error During Fourth Operation of a Quad Block Transfer



The four cases in the preceding diagrams illustrated error recovery when an information cycle associated with a particular block transfer is ERRORed. Should a bus error occur which causes all the accesses associated with a block transfer to be aborted, that block transfer is effectively canceled. The initiator must rearbitrate for the bus and start from scratch. The net effect is that at any point in time, only one block transfer is using the error recovery protocol.

This is illustrated in figure 20 below. Two 2-operation block transfers are shown. The second information transfer of the first transfer gets ERRORed. This block transfer still completes using the recovery mechanisms just discussed. All cycles associated with the second block transfer on the other hand are completely aborted as a result of the error. This block transfer is then completely canceled and its initiator at some later time must rearbitrate for the bus and restart the transfer. Had the second block transfer been long enough to have grant inhibit asserted at the time the first block transfer is ERRORed, it deasserts **grt_inh** upon seeing the bus ERROR.

Figure 20. Error causing entire block transfer to be canceled


Err
First 2-operation block transfer. Second operation got ERRORed, this block transfer completes per the error recovery protocol
Arb
Inf-1
Pst1
Pst2
Arb
Inf-2
Pst1
Pst2
Err1
cpt
cpp
iot
iop1
iop2
Second 2-operation block transfer. this is completely canceled by the error and must be restarted from scratch
Arb
Arb
Inf-1
Arb
Grant Inhibit

6.7.2 Global Writes

Global write operations are a variation of peer-to-peer transactions. During the echo transaction of a global write, the PCIB clocks in data at the same it is sending data to the CPU. In this way, all boards in the system see the global write synchronously. It also means that only a CPU can issue global writes.

6.7.3 Board Breaking

It is possible for a simplexed board to be performing a block transfer and then break part of the way through the transfer, resulting in a never to be completed partial transfer. For this reason, all boards targeted by a block transfer must watch the TRIDs of the entire block transfer. Should the TRID change before a block transfer was supposed to have completed, the bus ASIC needs to pad the block with 0's to replace the missing data.

Though padding the block with 0's to replace the missing data may at first appear undesirable (no data is better than bad data), a few factors must be kept in mind:

1) This is only an issue for simplex I/O sources such as disk and ethernet. The software for these boards must tolerate the board "disappearing" and hence must have recovery mechanisms for incomplete blocks (0's in the memory are no worse than whatever random information would have been there prior to being overwritten by the I/O board).

2) The Stratabus based systems exhibit the same characteristic.

3) To not do this has a potential performance impact: bus ASICs may have to otherwise hold the entire block prior to passing onto the board rather than passing good data as it is received.

4) This situation should only occur on CPU initiated transfers. On the CPU, an LPMC is returned coincident with the read data.

Individual bus interface designs may choose to provide a notification mechanism should this occur.

It should be noted that the breaking board will drive a double bus error when it breaks. In the case described above, the errored information should get re-transmitted, but never will because the board has broken.

6.7.4 Bus Busys

A block transfer may only be BUSYed during the first phase of the transfer, i.e. BUSY may be asserted during the Post2 phase associated with the first information transfer. This single BUSY has the effect of canceling the entire block transfer. In practice, it is too late to prohibit the transfer of the subsequent sub-operations on the Xbus so the information in these sub-operations is simply ignored. The transfer initiator must re-arbitrate for the bus and repeat the block transfer.

Note that bus error takes precedence over busy. The only reason a subsequent cycle will ever have BUSY asserted is if a board misinterpreted the information within the cycle or is about to break. In both these cases, ERROR will be asserted concurrently with BUSY and should take precedence.

6.8 Peer to Peer transactions

Peer to peer transactions between the two CPUs are based on a simple store and forward protocol. A CPU sends a transaction on any of its two buses attached to a non-broken PCIB. Once any errors have been resolved (the Post2 phase of the last operation has passed without error), the PCIB(s) echo the transaction back up to the CPUs. The send and echo transaction are not split; the driving CPU holds grant_inhibit until the complete send-echo pair is complete. This is to prevent re-ordering problems arising from eviction/flush queue (EFQ) and read/write queue (RWQ) operations passing each other. Read transactions are split between the request and the data return. An indivisible send-echo carries the read request. Other operations may intervene while the read is performed; then the data is returned in an indivisible send-echo data return. The CPU initiator of the peer-to-peer operations must track the entire operation to see whether the cycle is acknowledged, busied, or errored.

Acknowledge and Busy are ignored for the first half (send portion) of a peer-to-peer transaction. A busy response on the echo will terminate the transaction and it will be retried at a later time beginning with the send portion. The send cannot be busied. The echoing board needs to track the error line during the echo so that an error on the echo portion will cause the normal seven phase error protocol followed by a re-echo of the transaction starting from the first phase that was errored and then proceeding forward. The send portion is not repeated in the event of an echo error.

The initiating CPU needs to keep track of whether a cycle will be peer-to-peer or not and indicate it on the Xbus via bit 4 of the second half TRID. Table 7 lists the Xbus operations that the CPU will mark as peer-to-peer by using bit 4 of the second half TRID. The cycles listed under non peer-to-peer operations will not be echoed. At a minimum the peer-to-peer initiator must drive grant inhibit from the first info phase of the send until one phase prior to the first info phase of the echo. The board performing the echo must handle grant inhibit from the first phase of the echo until the completion of the cycle. The PCIB will only echo operations marked as peer-to-peer via the TRID bit. The peer-to-peer bit in the TRID is only set for the send, the echo sets it back to zero since a

Table 7. Peer to Peer and Non Peer-to-Peer Operations

Peer-to-Peer Operations	Peer-to-Peer Operations during update mode or force peer-to-peer mode[a] (non peer-to-peer otherwise)	Non Peer-to-Peer Operations
non paired I/O operations to slot 0 or slot 1	all load clear operations	non paired I/O operations to slot 2 or slot 3
all memory reads and memory writes	all special cache line operations	paired I/O operations to slot 2 or slot 3
all global I/O operations	paired I/O operations to slot 0 or slot 1	data returns to slot 2 or slot 3
all global CPU broadcast I/O operations		PINGs resulting from non peer-to-peer I/O reads.
data returns to slot 0 or slot 1		
PINGs resulting from peer-to-peer I/O reads.		

a. bit 22 of the Bus Interface State Register

cycle with this bit set will not be responded to.

Only CPU's can issue peer to peer transactions, because only PCIBs have the buffering to perform a send and echo. PCI peer to peer transactions are not supported, and all traffic between PCI cards must occur via main memory.

Note: the following diagrams show A0, B0, A1, B1 info buses, not an idealized overlaying of operations on a single bus (as in the preceding).

Figure 21. Peer to Peer Cycle Initiated by a Simplexed CPU

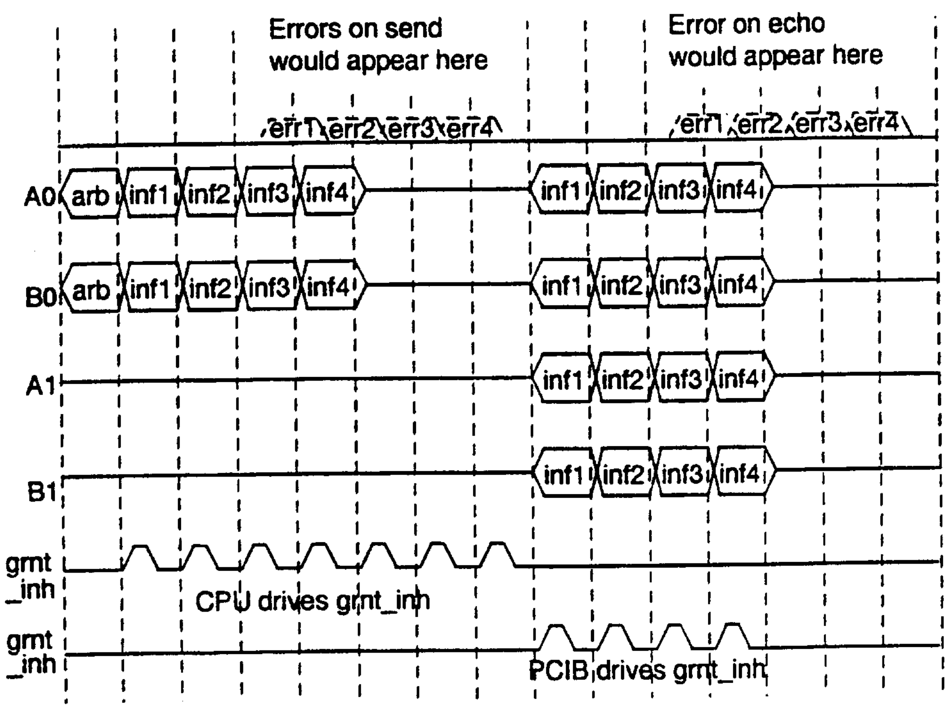
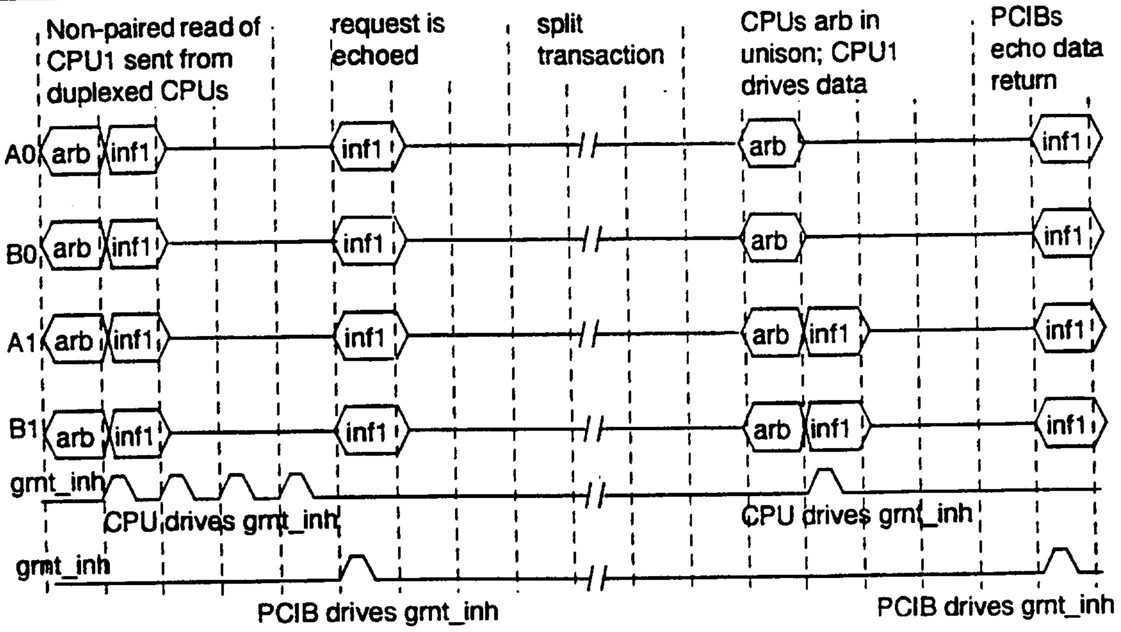


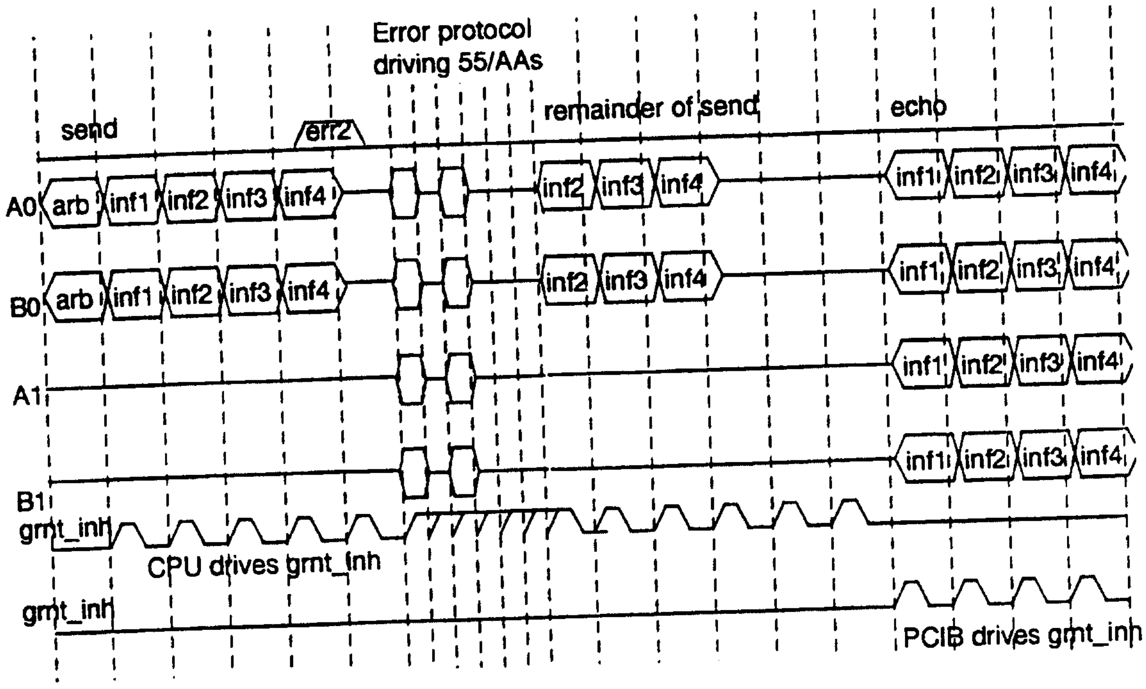
Figure 22. Peer to Peer Non-paired Read from a Duplexed CPU



6.8.1 Bus Errors

A bus error during the send portion of a peer-to-peer cycle is handle identically to any other cycle. Figure 23 illustrates this. A bus error in the echo portion will result in execution of the error sequence followed by the remainder of the echo transmission.

Figure 23. Store and Forward with Error on 2nd Operation of Quad Block Transfer



6.9 Major Transactions

There are three types of transactions that can be performed on the Xbus, (non-split) memory writes, (split) memory reads and (split) flush/purge type transactions.

Non-split transactions are when the bus master does not require any data or response from the slave device to complete the transaction. These transactions supply an address in the first operation and the data in the subsequent sub-operation(s). All data write transactions are non-split and use this protocol.

A split transaction is 'split' into two separate bus operations, a request followed by the response. A block read is a good example of this. The master initiates this transaction by a read request for X bits of data for a given address (the function code, virtual and physical address are all supplied in the info phase). The slave who has decoded the function code and address then perfoms the request on-board. When it gets the data, it returns the data via another bus operation. A transaction id (TRID) is used to match requests with responses and is unique to all transactions pending.

Flush and purge transactions are split. They are unique only because they do not need to see a bus operation returning data to complete the transaction. Any flush or purge transmitted from Cougar to Cyclops is transmitted on the Xbus. It is normally just looped back on the Xbus. However, if the board is in update mode, the transaction is performed as a peer-to-peer transaction. They will always see an INFO RETURN FUNC_OP (or INFO RETURN 256 when returning data) as the return on this transaction. The virtual index and physical address is included in this type of return.

Notes:

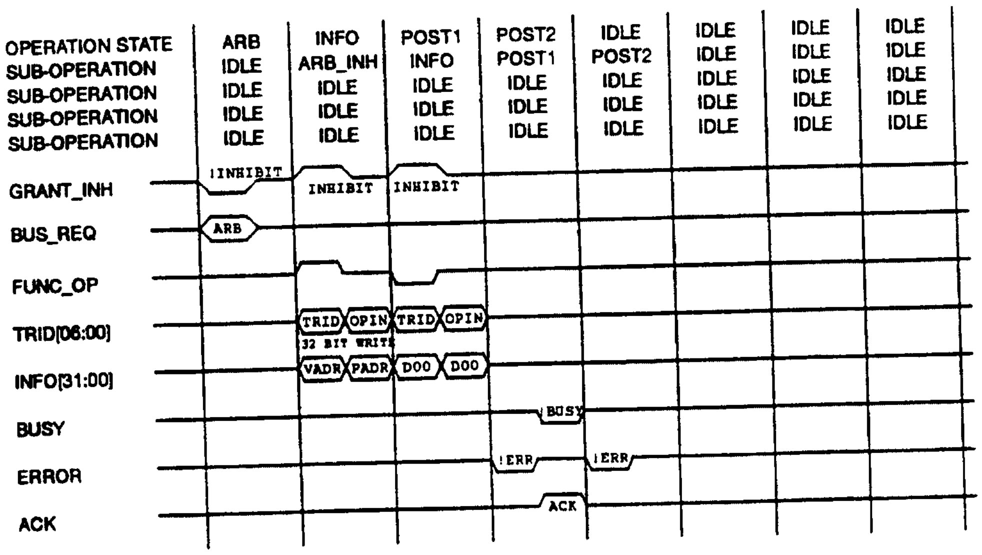
- The figures in this section contain a horizontal line that indicates either a tri-state or don't care condition.
- All signals are drawn active high so they are 'HI' when signals are in their TRUE state.
- bus_req and gmt_inh are driven on both cycles of the ARB phase. Note they are clocked in on the first half of the ARB phase and evaluated for the grant during the second half of the ARB phase.
- When VADR is seen in these figures, it actually represents the function code and virtual index. See section 6.10.2 on page 63 more details and bit definitions.
- When the note 'OPIN' appears on the TRID signals, it represents operation information and transaction defined information that will be passed with the bus operation. see section 6.10.1 on page 62 for more details and bit definitions.
- The Xbus control bus bits have been spilt out for readability. For example, BUS_REQ and ACK are drawn as two separate signals while in reality they are both on controll[0]. Likewise BUSY, ERROR, and GRANT_INH are broken out from the control bus (maint_int is not shown in these diagrams).

6.9.1 Write Transactions

A write transaction is completed in one bus operation with from one to four sub-operations for the data transfer. The initial bus operation supplies the function code, the virtual address, and the physical address in the info phase. The accompanying sub operation(s) are used to supply the data needed for the transaction.

Figure 24 Illustrates a 32 bit write transaction. Note it is up to the bus master to assert GRANT_INH (after receiving a grant) to reserve the info phases for the data transfer(s).

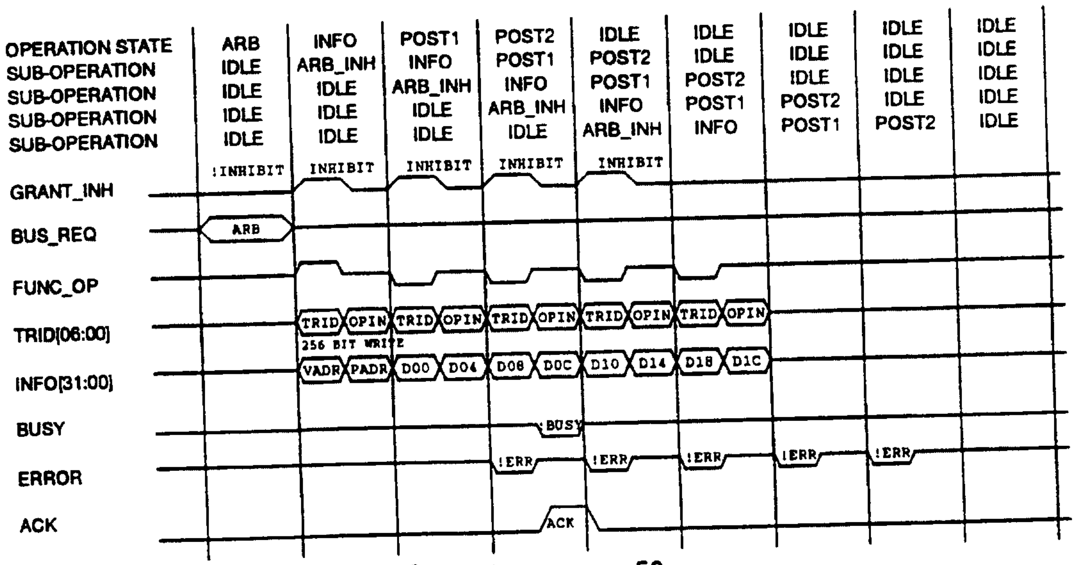
Figure 24. 32 Bit Write Transaction



*See notes on page 53.

A block write transaction is much like a 32 bit write operation. The only difference is that the master must assert GRNT_INH for additional phases to reserve the extra info phases needed to transfer the extra data. Figure 25 illustrates a block write transaction.

Figure 25. 256 Bit Block Write Transaction

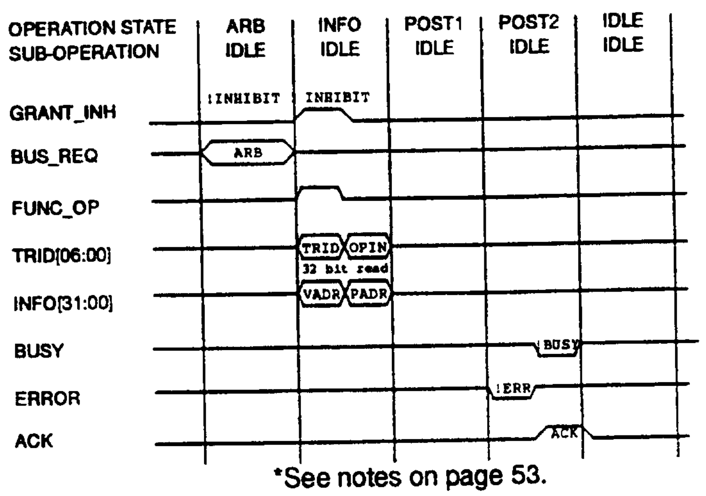


*See notes on page 53.

6.9.2 Read Transactions

A read transaction requires two bus operations: a request operation followed by the response operation. These two bus operations will always have the same TRID. The request operation does not have any accompanying sub-operations, but the response operations will require sub-operations if the amount of data requested cannot be transferred in one info phase. Figure 26 illustrates a basic read transaction with no sub-operations.

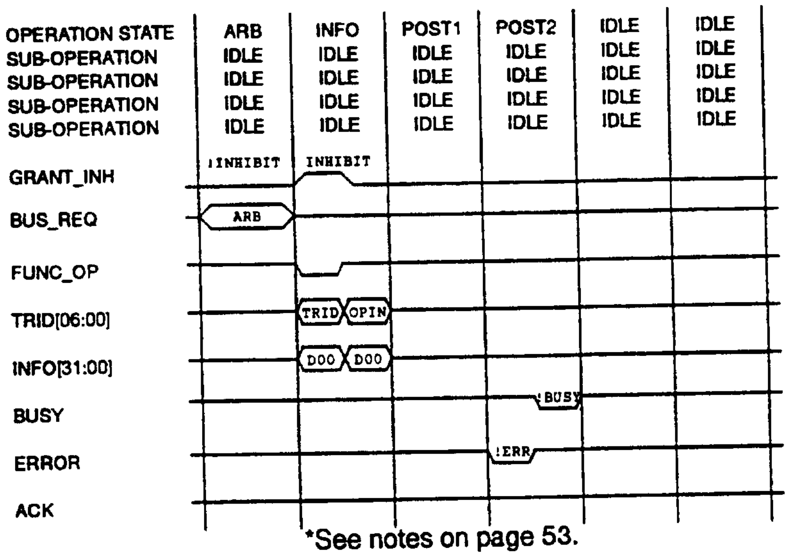
Figure 26. 32 Bit Read Transaction



*See notes on page 53.

The slave then responds with the data as follows:

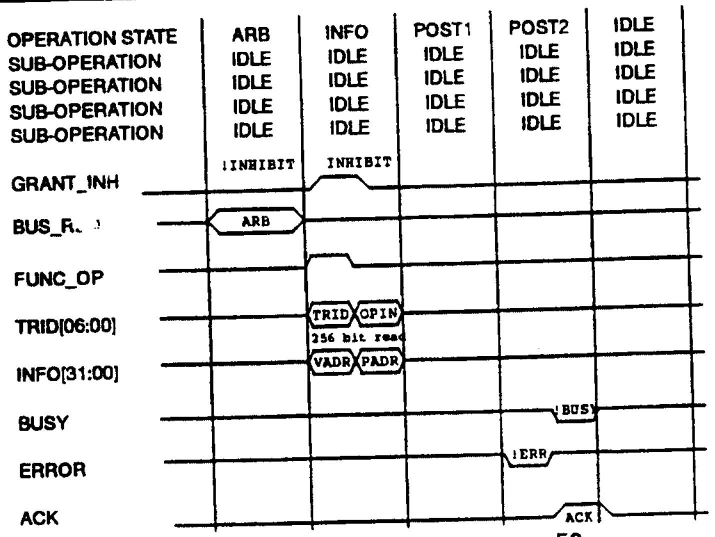
Figure 27. Slave Response to a 32 Bit Read



*See notes on page 53.

Block read transactions are transactions that requires sub-operations on the data return operation. Figure 28 illustrates this type of transaction.

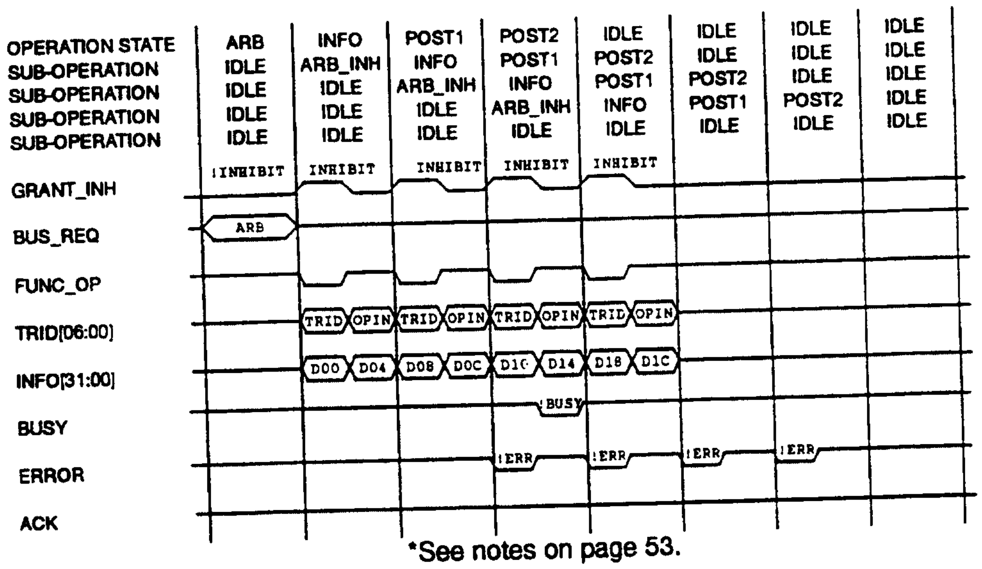
Figure 28. 256 Bit Read Transaction



*See notes on page 53.

The slave then responds with the following.

Figure 29. Slave Response to a 256 Bit Read Transaction



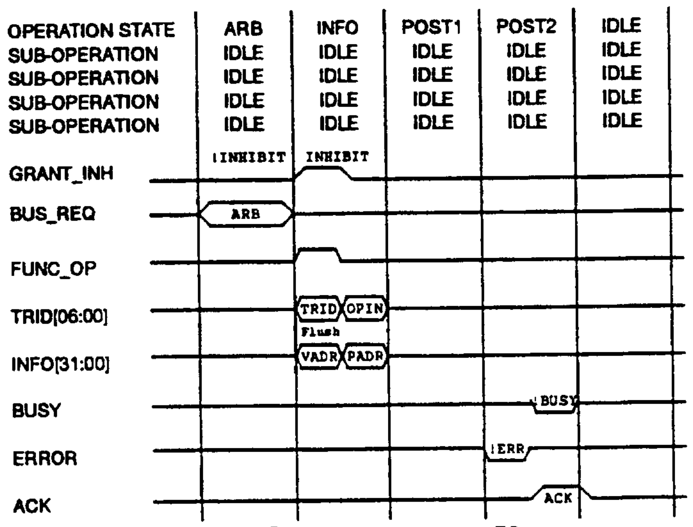
*See notes on page 53.

6.9.3 Flush and Purge Transactions

Flush and purge operations look just like read transactions with the exception that the response may not have data to return. The return for these operations will always have FUNC_OP true with an INFORMATION RETURN (or INFORMATION RETURN 256) function code. The INFORMATION RETURN will have at least one bit of transaction defined information to pass back to the requesting device. The information that is passed here will be hit/miss information on the other board's cache. This information is passed during the second cycle of the info phase and uses the **trid[6:0]** bus lines (OPIN) (see section 6.10.1 on page 62). If returning data, it will use an INFORMATION RETURN 256 function code. Note that the virtual index and physical address are always passed back on the return operation for these transactions.

figure 30, Flush with Modified Data Transaction on page 57 and figure 31, Slave Response to a Flush with Modified Data Transaction on page 57 are examples of a flush operation with a mod-data return. Note this return will always be a cache line and that the return looks just like a cache-line write operation in that it has FUNC_OP true and supplies the virtual and physical address.

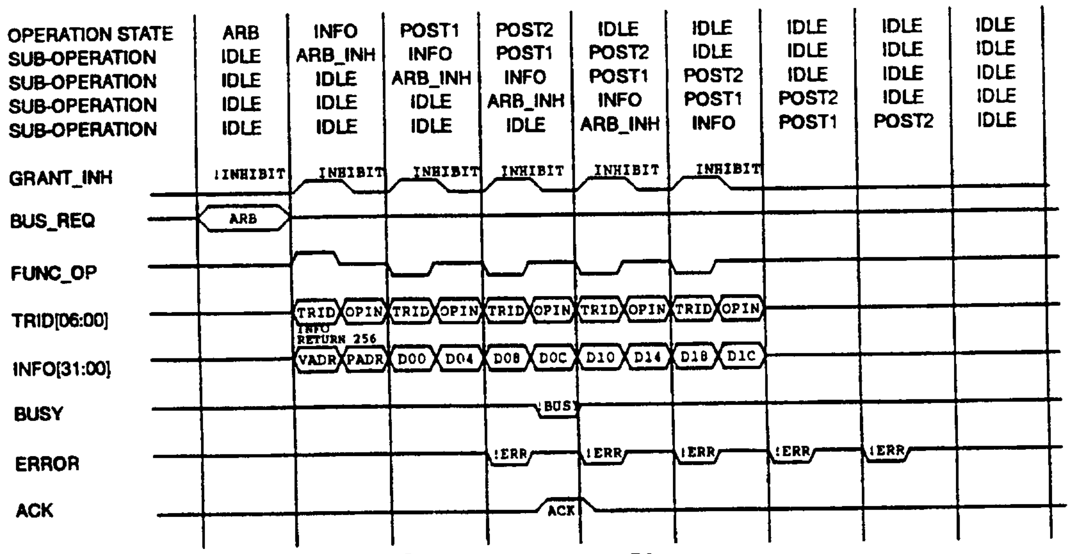
Figure 30. Flush with Modified Data Transaction



*See notes on page 53.

The slave then responds with the following. Note the state of the FUNC_OP signal.

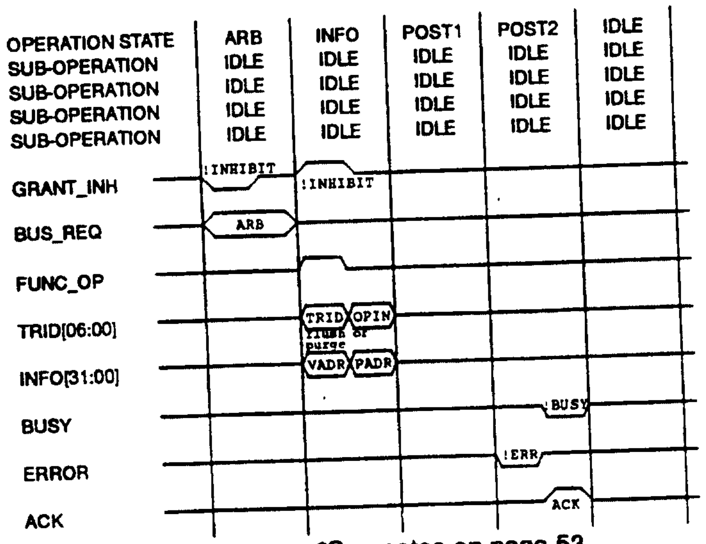
Figure 31. Slave Response to a Flush with Modified Data Transaction



*See notes on page 53.

Figure 32 illustrates a flush transaction with no modified data to return. Note that the return for this transaction uses the info return FUNC_OP as a response. The TLB purge, data purge and instruction flush transactions look exactly like this transaction.

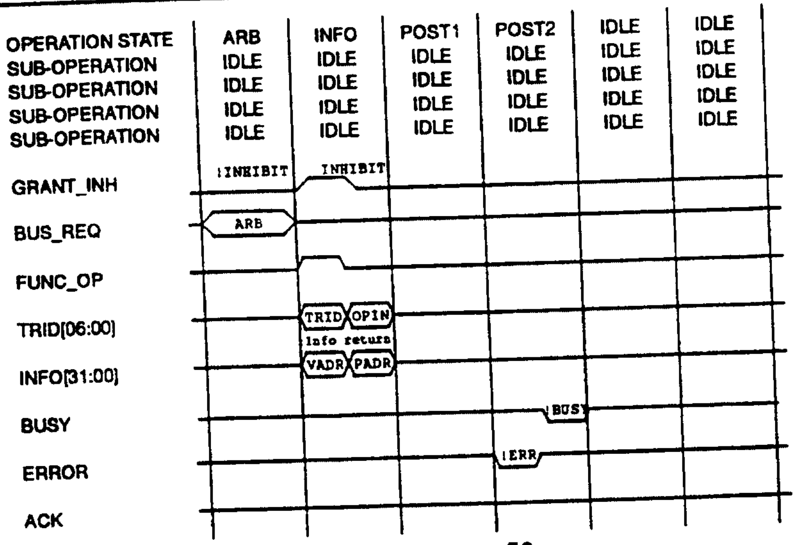
Figure 32. Flush (no modified data) or Purge Transaction



*See notes on page 53.

The slave then responds with the following.

Figure 33. Slave Response to a Flush (no modified data) or Purge Transaction



*See notes on page 53.

6.9.4 Load/Clear Transactions

Load/Clear Xbus transactions are hardware supported lock operations. A lock is won by performing a 32-bit load transaction in which the data received is non-zero. A zero result indicates that the lock is not won. A lock is cleared by issuing a clear transaction or by simply writing non-zero data into the memory lock location.

Load/clear transactions look exactly like a 32 bit read transaction. The only difference other than the function code is that the byte enables are not used. For more information on load/clear transactions, refer to the Mercury Functional Specification.

6.9.5 Ping Transaction

A PING transaction is a single bus operation transaction. Only the TRID is needed in this transaction. The only reason a PING transaction is done is to see if someone drives the ACK signal in POST2 of the bus operation.

6.9.5.1 The Need for PINGs

Because the Xbus uses split transactions, there is a need for the ability to handle cases where simplexed boards break and take themselves off-line while there is still a transaction pending on them. This can result in a device waiting forever for a TRID return that will never come. This condition cannot occur on write operations because they are not split. See section 6.7.3 on page 48 for information on boards breaking during block transfers.

The system will crash when an on-line simplexed CPU/memory breaks so there is no concern with transactions that can only be directed to on-line CPU/memory boards. When CPU boards are off-line testing, there is a need to do I/O reads to them. This is a case where the testing CPU/memory can break and not cause a system crash but still leave transactions incomplete.

I/O boards can be simplexed while on-line or off-line and can break at any time causing the same conditions. The only split transaction directed to I/O boards are I/O reads.

Therefore, the list of transactions that has the potential of causing this condition is limited to I/O reads. I/O reads will require a test operation (called a PING) to gain information on whether or not a given TRID is still being worked on, i.e. if a board has broken or not.

6.9.5.2 Detailed Rules During PING Operations.

After the transaction master sends an I/O read operation it will start a timer. If the timer times-out before the TRID is returned, the transaction master will send a PING operation. This PING operation only contains a valid TRID (the info lines must be deterministic but are considered a don't care). The transaction slave device that is currently active working on this TRID will ACK the PING operation. The transaction master will then set the timer again and continue waiting for the TRID return. If the PING is not ACKed, the transaction master will drop the transaction and assume the transaction has failed and/or the board has broken.

The slave device must keep a copy of all the I/O read TRIDs outstanding on the board. This is so that it may compare the TRID of all PING operations and ACK the PING when the TRID matches.

- The transaction master must ignore results of PINGs if the matching TRID was marked complete. This can occur if the PING and TRID return were both on the bus at the same time (or if due to BUSYs that passed each other on the bus)
- The transaction master should mark TRIDs complete on the phase after POST2 and no-error of the TRID return. If the TRID was complete at the exact instant the timer timed-out, the PING operation can be sent, but if it is sent, the results of the PING must be ignored.
- The transaction master must not start any additional bus transactions on the same transaction master if a PING transaction (operation) is pending. This is so that an old PING is not using the same TRID as a new transaction. This has the potential of occurring when a TRID return was seen before the PING completed it's bus operation.
- The slave must mark a TRID 'working' and start ACKing PINGs on the Xbus before the transaction master timer times-out for the first time. There is ample time but the slave should do this as soon as possible. The cycle after POST2 and no-error of the request is the earliest this could be done.

- The slave should invalidate a 'working TRID' the phase after POST2 and no-error of a TRID return. It could be done later but this only adds complexity. Note that in theory there may be only four phases between POST2 of the return and POST2 of a new request with the same TRID.

Figure 34. PING Transaction

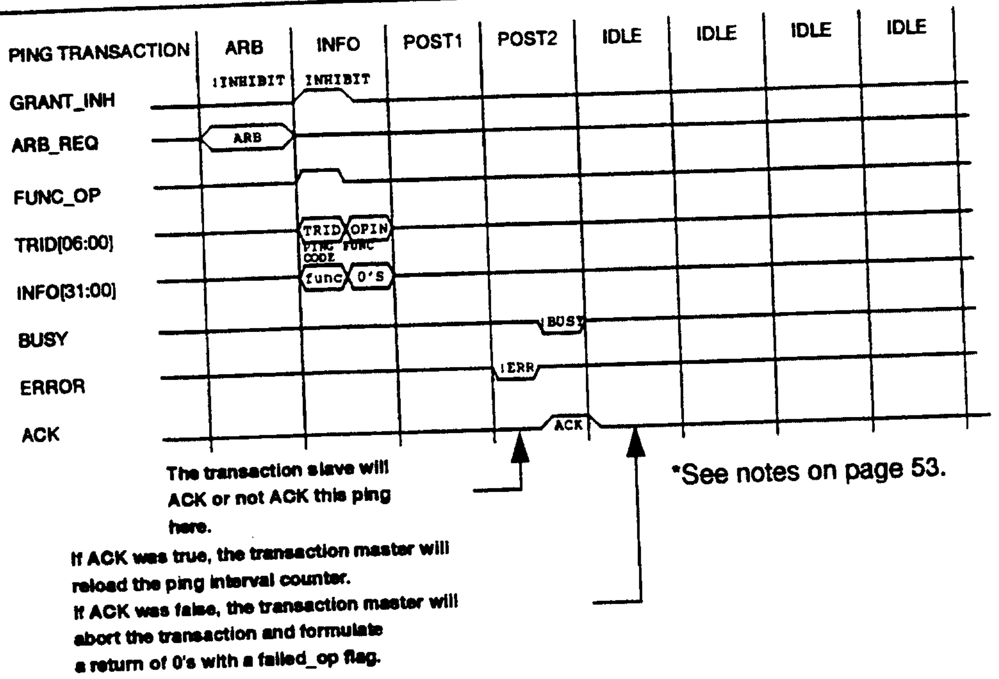

PING TRANSACTION
ARB
INFO
POST1
POST2
IDLE
IDLE
IDLE
IDLE
!INHIBIT
INHIBIT
GRANT_INH
ARB_REQ
ARB
FUNC_OP
TRID[06:00]
TRID
OPIN
PING CODE
FUNC
INFO[31:00]
func
0'S
BUSY
!BUSY
ERROR
!ERR
ACK
ACK
The transaction slave will ACK or not ACK this ping here.
If ACK was true, the transaction master will reload the ping interval counter.
If ACK was false, the transaction master will abort the transaction and formulate a return of 0's with a failed_op flag.
*See notes on page 53.

Figure 35. PING Transactions, Boundary Condition 1

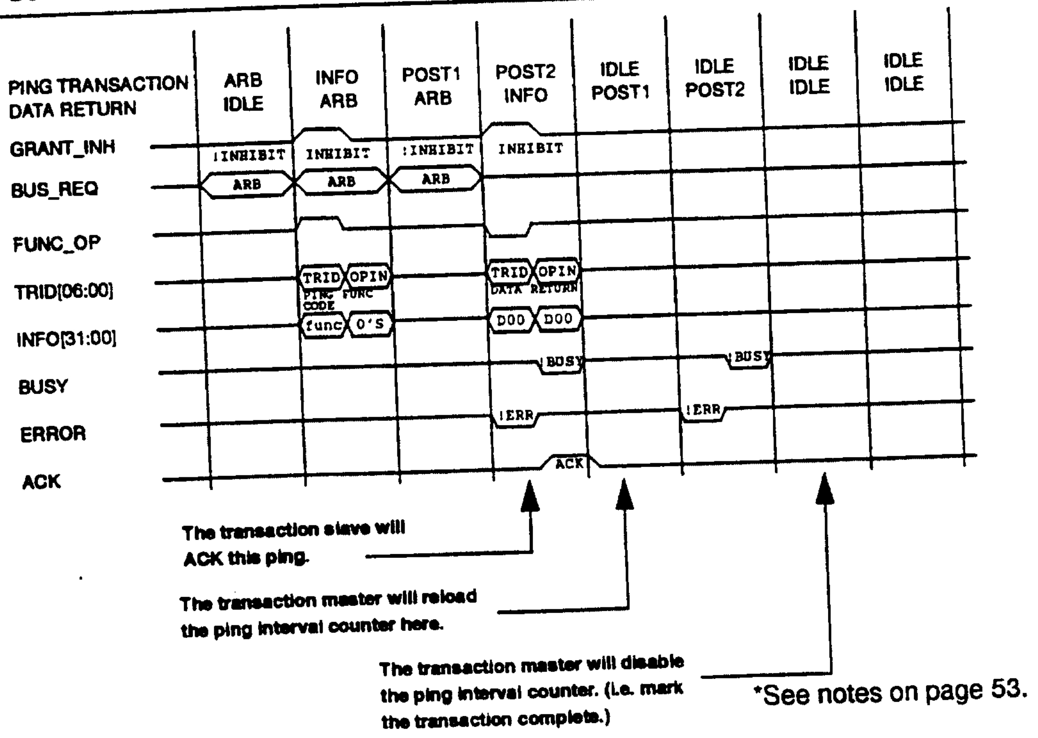
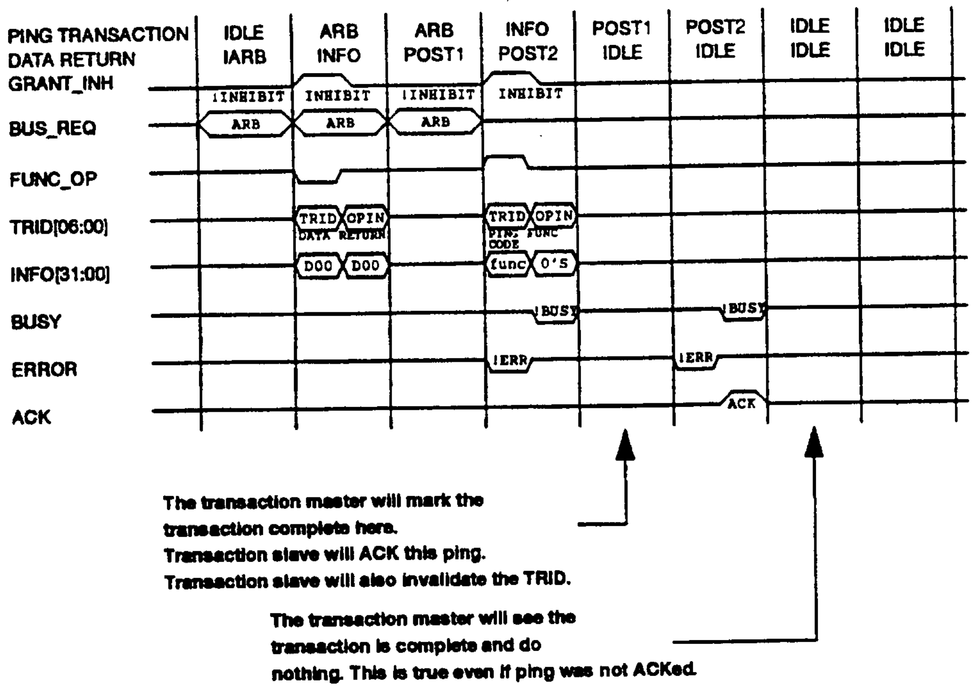


Figure 36. PING Transactions, Boundary Condition 2.


PING TRANSACTION
DATA RETURN
IDLE IARB
ARB INFO
ARB POST1
INFO POST2
POST1 IDLE
POST2 IDLE
IDLE IDLE
IDLE IDLE
GRANT_INH
BUS_REQ
FUNC_OP
TRID[06:00]
INFO[31:00]
BUSY
ERROR
ACK
INHIBIT
ARB
TRID
OPIN
DATA RETURN
PING FUNC CODE
D00
func
0'S
BUSY
ERR
ACK
The transaction master will mark the transaction complete here.
Transaction slave will ACK this ping.
Transaction slave will also invalidate the TRID.
The transaction master will see the transaction is complete and do nothing. This is true even if ping was not ACKed.

*See notes on page 53.

6.9.6 busy, ack and error combinations

There are eight possible combinations of busy, error and ack on the xbus. The table below shows the outcome of the bus operation for the given combinations of busy, ack and error.

Table 8. busy, ack and error combinations.

error	busy	ack	operation results
0	0	0	Non acked operation. Writes are dropped on the floor. CPU reads return 0 and cause an LPMC, if enabled. PCIB reads cause the initiating PCI slot to go off-line.
0	0	1	Normal complete acked operation.
0	1	0	Busied operation, retry.
0	1	1	Busied operation, retry.
1	0	0	Errored operation, follow error protocol.
1	1	0	Errored operation, follow error protocol.
1	0	1	Errored operation, follow error protocol.
1	1	1	Errored operation, follow error protocol.

6.10 Xbus Signal Formats

6.10.1 trid[6:0] and func_op_ Bus Format

A master that has won arbitration will drive these lines along with the **info** lines. Below is a list of signals internal to the ASIC that are multiplexed and transferred on the **trid** and **func_op_** Xbus signals.

trid[6:0]	These are the transaction identification bits.
failed_op	This bit is valid during the second bus cycle of an info phase. It is only true during a peer-to-peer echo resulting from a failed peer-to-peer send. The info for phases that were transmitted without error are sent normally. The info for the phases that were never sent is set to zero and the failed_op bit is set indicating that this data was never transmitted so it has been zero-filled.
first_op	This bit is only true during the first bus operation of a transfer of information, i.e. it is only false during sub-operation and idle phases. Note it must be true during the first bus operation of a data return.
func_op_	This bit when true signifies that the info lines carry a valid function code, address, etc. on this phase.
Hit/Miss	This bit represents the hit/miss status on special cache operations. It is zero on all other transactions.
IOVA	This bit indicates that the address is an I/O Virtual Address and must be translated.
peer-to-peer	This signal indicates that the CPU bus master wants this current cycle to be driven peer-to-peer and it will control the grant-inhibit line to allow for it. This bit only has meaning when driven by a CPU and should otherwise be zero.
remain[2:0]	These bits represent a binary number that indicates how many bus sub-operations remain to complete this transfer of information (after this one). This field is zero for transfers that do not have sub-operations.

The figure below describes the format of the TRID bits (**trid[6:0]**) and the func_op bit on the first cycle of any INFO phase.

Figure 37. **trid[6:0] and func_op_** Format for the 1st Cycle of an Info Phase

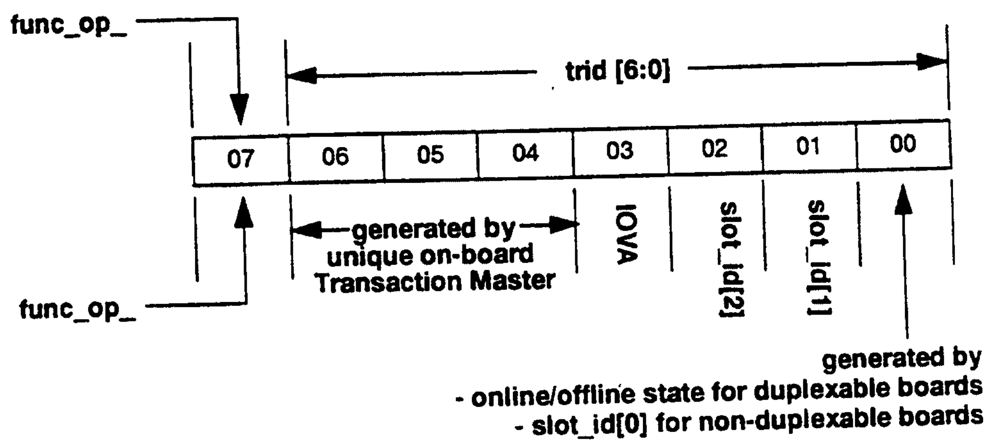
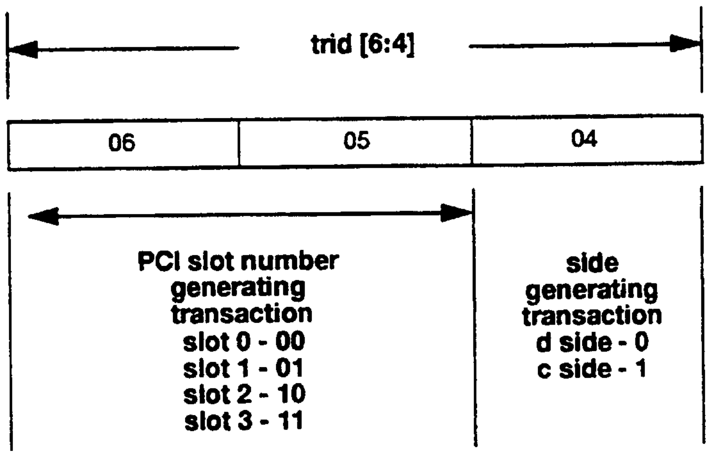

func_op_
trid [6:0]
07
06
05
04
03
02
01
00
generated by unique on-board Transaction Master
IOVA
slot_id[2]
slot_id[1]
func_op_
generated by
- online/offline state for duplexable boards
- slot_id[0] for non-duplexable boards

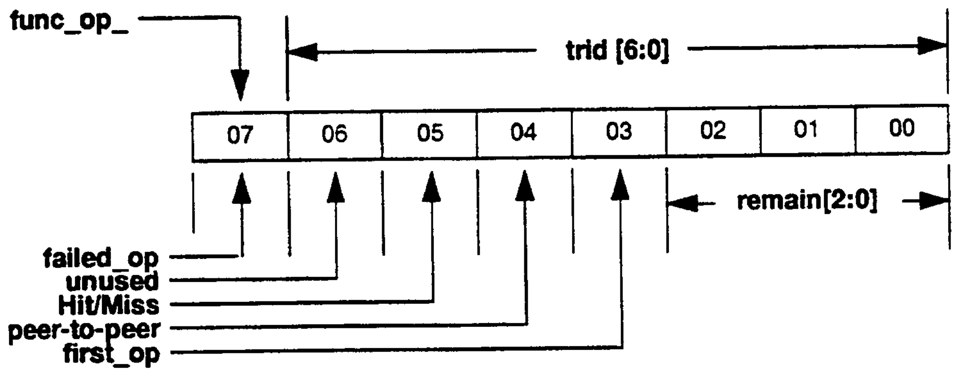
Figure 38 shows how the unique TRID filed is generated on the PCIB board in order to be able to check the PCI slot field in the I/O map RAM look-up (refer to Polo Programming Guide, section 5.4.1, System Address Generation, for details on this checking).

Figure 38. **trid[6:4]** Format for a PCIB for the 1st Cycle of an Info Phase


trid [6:4]
06
05
04
PCI slot number generating transaction slot 0 - 00 slot 1 - 01 slot 2 - 10 slot 3 - 11
side generating transaction d side - 0 c side - 1

The figure below describes the format of the TRID bits (**trid[6:0]**) and the func_op bit on the second cycle of any Info phase. The unused bits should be set to zero.

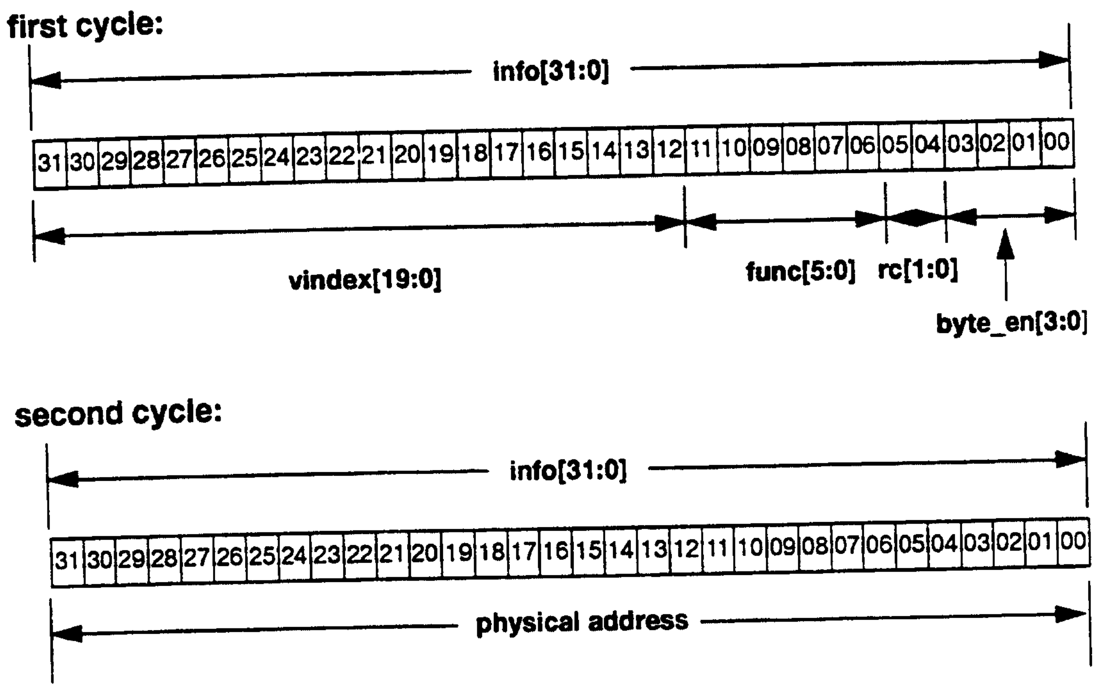
Figure 39. **trid[6:0]** and **func_op_** Format for the 2nd cycle of an Info phase



6.10.2 Info Bus Format

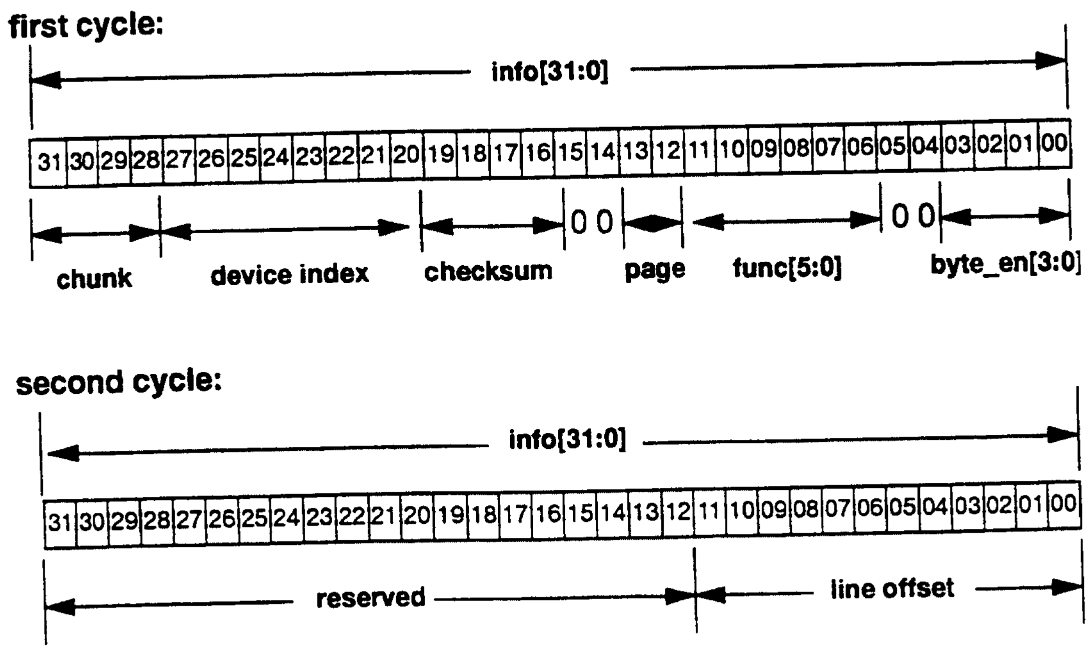
The figure below describes the format of the 32 bit info bus (**info[31:00]**) during the INFO phase of a function code operation (FUNC_OP). This is an illustration of an operation using a CPU address

Figure 40. **Info[31:0]** Format of CPU Address FUNC_OP



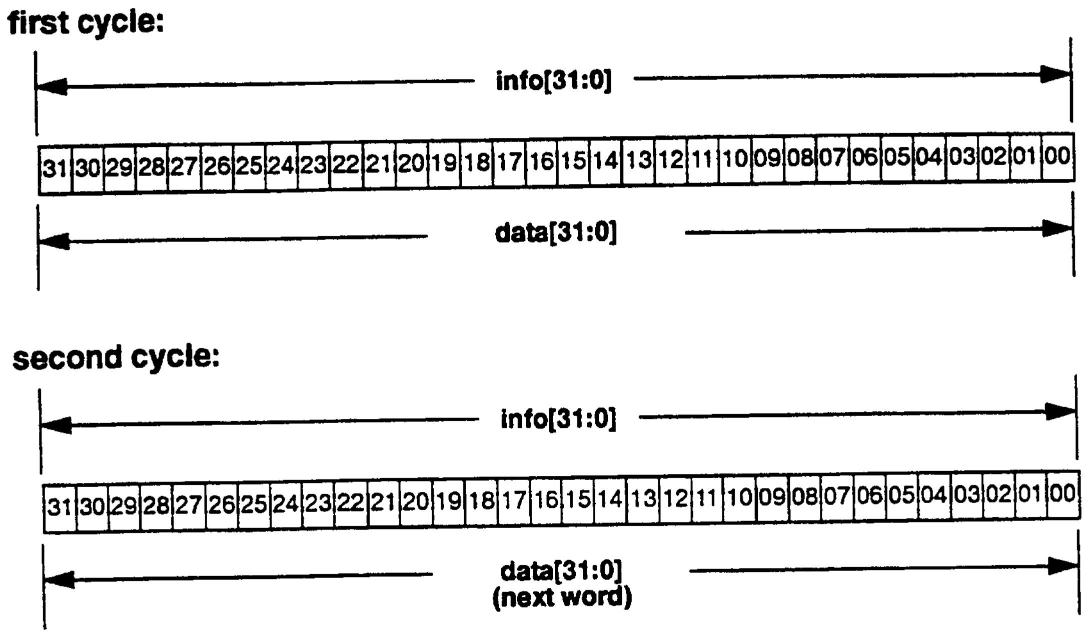
The figure below describes the format of the info bus for the INFO phase during a function code operation (FUNC_OP) when using an IOVA (I/O Virtual Address). Note that the RC bit field is zero; the C bit is stored in the map RAM and looked up during the IOVA to CPU address translation, the R bit is always 0 since there is no remote CPU board.

Figure 41. **Info[31:0]** Format of IOVA Address Bus FUNC_OP



The figure below describes the format of the info bus on the first and second cycle of a of an INFO phase when the FUNC_OP signal is false (i.e. data). It is an example of an operation returning 64 bits of data. If the operation was returning 32 bits of data, the same data would be driven first and second cycle.

Figure 42. **Info[31:0]** Format During Data Transfers



6.10.2.1 Function Code (rc and byte enable) Transfer

The function code bits are used to describe the Xbus operation and/or request. A function code is not passed on a normal data return operation.

The function code bits are transferred on the first cycle of the Info phase. These bits are valid only when the FUNC_OP signal is true. **func[5:0]** use bus lines **info[11:6]** respectively. Below is a list of the basic operations that can be performed on the Xbus. These are defined within **func[5:3]**. Note that **func[2:0]** are re-defined for each basic operation however, the meaning of these bits were kept consistent where possible.

Note that table 9 through table 14 are subsets of the Golfbus function code definitions.

32 bit read operations func[5:3] = 000

Included in this group are all the read operations/requests that only require a single 32 bit return.

func[2:0] These bits represent a specific 32 bit read operation.

Table 9. 32 BIT READ OPERATIONS/REQUESTS

func[5:0]	OPERATION DESCRIPTION	info phases	data bits transfer/ request
000000	IDLE BUS	0	0
000001	reserved		
000010	PING (used to test TRIDs)	1	0
000011	MAIN MEMORY LOAD/CLEAR OPERATIONS	1	32
000100	32 BIT MEMORY READ	1	8-32
000101	32 BIT I/O READ	1	8-32
000110	reserved		
000111	reserved		

32 bit write operations func[5:3] = 001

Included in this group are all the 32 bit write operations/transactions.

func[2:0] These bits represent a specific 32 bit write operation.

Table 10. 32 BIT WRITE OPERATIONS/TRANSACTIONS

func[5:0]	OPERATION DESCRIPTION	info phases	data bits transfer
001000	reserved		
001001	reserved		
001010	reserved		
001011	reserved		
001100	32 BIT MEMORY WRITE	2	8-32
001101	32 BIT I/O WRITE	2	8-32
001110	reserved		
001111	reserved		

Special Cache Line Operations func[5:3] = 010

This group of operations perform special tasks on cache lines.

func[2:0] These bits uniquely identify a bus operation

Table 11. SPECIAL CACHE LINE OPERATIONS

func[5:0]	OPERATION DESCRIPTION	info phases	bits transfer
010000	FLUSH CACHE LINE (data) (may writeback)	1	0 or 256
010001	FLUSH CACHE LINE (instruction) (never a writeback)	1	0
010010	TLB PURGE (data)	1	0
010011	TLB PURGE (instruction)	1	0
010100	reserved		
010101	PURGE (data)	1	0
010110	INFO RETURN	1	0
010111	INFO RETURN 256	5	256

Read Return Function Codes func[5:3] = 011

This group of function codes is used only internal to the bus ASICs. These function codes will never actually get transmitted on the Xbus.

func[2:0] These bits represent how many data words are being returned.

Table 12. READ RETURN FUNCTION CODES

func[5:0]	OPERATION DESCRIPTION	info phases	bits transfer
011000	32 bit return on a 32 bit bus	1	32
011001	128 bit return on a 32 bit bus	2	128
011010	256 bit return on a 32 bit bus	4	256
011011	reserved		
011100	reserved		
011101	reserved		
011110	256 bit return on a 64 bit bus[a]	4	256
011111	reserved		

a. although the Xbus is 32 bits wide this function code is supported for backwards compatibility with the Cougar ASIC.

Block Memory Read Operations/Requests func[5:3] = 100

This group of operations are used to read blocks of data from memory.

func[2] This bit is always zero

func[1:0] These bits represent the number of info phases required to transfer the data on this bus transaction.

Table 13. BLOCK MEMORY READ OPERATIONS/REQUESTS

func[5:0]	OPERATION DESCRIPTION	data phases	bits request
100000	reserved		
100001	READ 128 BIT on a 32 bit bus	1	128
100010	READ 256 BIT on a 32 bit bus	1	256
100011	reserved		
100100	reserved		
100101	READ 256 BIT on a 64 bit bus[a]	1	256
100110	reserved		
100111	reserved		

a. although the Xbus is 32 bits wide this function code is supported for backwards compatibility with the Cougar ASIC.

Block Memory Write Operations/Transactions func[5:3] = 101

This group of operations are used to write blocks of data to memory.

func[2] This bit is always zero.

func[1:0] These bits represent the number of info phases required to transfer the data on this bus transaction.

Table 14. BLOCK MEMORY WRITE OPERATIONS/REQUESTS

func[5:0]	OPERATION DESCRIPTION	data phases	bits request
101000	reserved		
101001	WRITE 128 BIT on a 32 bit bus	3	128
101010	WRITE 256 BIT on a 32 bit bus	5	256
101011	reserved		
101100	reserved		
101101	WRITE 256 BIT on a 64 bit bus[a]	5	256
101110	reserved		
101111	reserved		

a. although the Xbus is 32 bits wide this function code is supported for backwards compatibility with the Cougar ASIC.

Reserved func[5:0] = 11xxxx

6.10.2.2 Remote/Coherent bits (rc[1:0])

These bits were developed to support a MESI CPU caching scheme and the Golf CPU/memory architecture. They give the flexibility to optimize performance and to maintain cache coherency between all boards in the system. The meaning of these bits for a given basic operation is shown below.

For all memory read operations (block or byte specific), **rc[1:0]** represent the following.

00	**/incoherent** - Don't snoop, don't set remote tags. This will be used by CPU code fetches and reads to locations where cache coherency is controlled by software.
01	**/coherent** - Snoop but don't set remote tag. This will be used by boards without a cache when reading cacheable locations.
10	**/remote/shared** - Snoop, set remote tag. An example is a CPU reading shared data.
11	**/remote/exclusive** - Snoop, set remote tag. Information will be passed to the CPU that exclusive rights are required on this cache line. A CPU (a board with a write into cache) will be the only board to initiate this type of operation.

For all memory write operations block or byte specific), **rc[1:0]** represent the following.

00	**/incoherent** - Don't snoop this, write this to main memory. This will be used on writes to locations where cache coherency is controlled by software.
01	**/coherent** - Snoop, Invalidate and writeback any modified data, Don't change remote tag status. (Note: If a writeback is generated from a snoop of this type of write, The writeback will use the remote/exclusive protocol and that will clear the remote tag.) This will be used by I/O board writes that need to maintain cache coherency.
10	INVALID.
11	**/remote/exclusive** - Snoop, clear the remote tag. This will be used on CPU cache line writebacks.

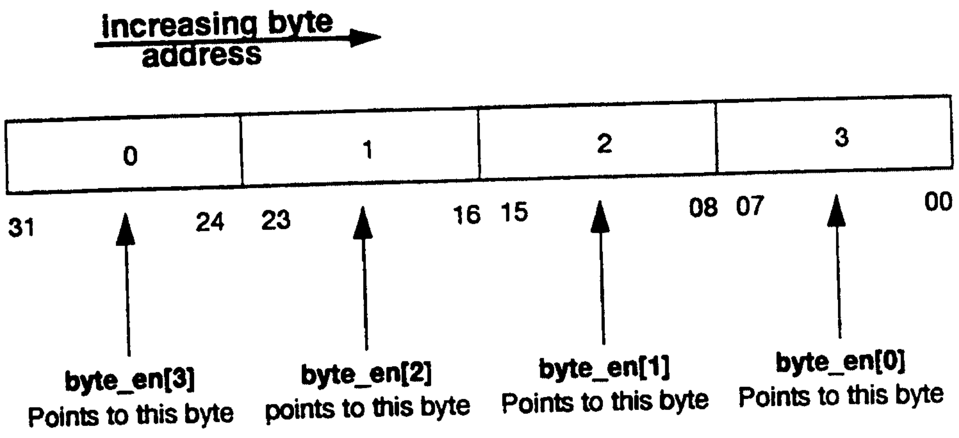
For all other operations, **rc[1:0]** have no meaning and will be = 0. This includes the following:

I/O WRITE OPERATIONS.
I/O READ OPERATIONS.
PURGE OPERATIONS.
FLUSH OPERATIONS.
BOTH INFO RETURN OPERATIONS.

6.10.2.3 Byte Enables **(byte_en[3:0]).**

The byte enables are valid on 32 bit read, 32 bit write, I/O read and I/O write operations. On all other operations, these bits will be false (=0). When these bits are set, It signifies that the operation is requesting/modifying the indicated bytes. Figure 43 identifies each byte and its corresponding byte enable bit.

Figure 43. **byte_en[3:0]** definition


Increasing byte address
0
1
2
3
31
24
23
16
15
08
07
00
byte_en[3] Points to this byte
byte_en[2] points to this byte
byte_en[1] Points to this byte
byte_en[0] Points to this byte

By asserting 1, 2, 3 or all the bytes enables, all 8. 16, 24 and 32 bit operations are possible.

This is shown below.

8 bit operations

```
3210
1000        Request or modify data[31:24] (address[1:0]=0)
0100        Request or modify data[23:16] (address[1:0]=1)
0010        Request or modify data[15:08] (address[1:0]=2)
0001        Request or modify data[07:00] (address[1:0]=3)
```

16 bit operations

```
3210
1100        Request or modify data[31:16] (address[1:0]=0)
0110        Request or modify data[23:08] (address[1:0]=1)
0011        Request or modify data[15:00] (address[1:0]=2)
```

24 bit operations.

```
3210
1110        Request or modify data[31:08] (address[1:0]=0)
0111        Request or modify data[23:00] (address[1:0]=1)
```

32 bit operation

```
3210
1111        Request or modify data[31:00](address[1:0]=0)
```

6.11 Board Synchronization/Board States

Board synchronization is covered in more detail in the ASIC and board specifications for the desired board. This section in the Xbus specification is limited to the board states necessary to perform the task of putting a board on-line simplexed or on-line duplexed. It gives a detailed description of the type of Xbus transactions that are possible in each state.

In Golfbus based systems, a mode known as simplexed mode was implemented for certain debug situations. This mode was never used or tested, and it has been removed on Polo.

6.11.1 CPU/Memory Bus Interface Modes

Below is a description of the board states needed to sync two CPU/memory boards (i. e. bringing a CPU/memory board on-line-duplexed). This is considered a super-set of all the modes necessary for booting and/or bringing a CPU/memory board on-line-simplexed.

The following signals are required on this type of board.

Broken, On-line, Duplexed, Update, Freeze-state (BODUF) (simplexed_mode bit false)

BODMUF	
1XXXX	**Broken:** Board is invisible to the Xbus. The board will ignore all Xbus operations. The board must be reset through the dedicated reset lines before it can be accessed at all.
00000	**Off-line:** The board will respond to non-paired I/O read cycles and will perform non-paired and Global write transactions. It will ignore all other bus operations. The board will arbitrate for the bus to return I/O read data. The board is capable of initiating bus transactions but conceptually, the board should not generate transactions. Software should enforce this where needed.
00010	**Off-line/Update:** The board will respond to non-paired I/O read cycles and will perform all I/O write transactions (Global, Paired and non-paired). The board will NOT respond to paired I/O read transactions. The board will also perform all memory write transactions that index into its memory array. The board is capable of initiating bus transaction and will generate only odd TRIDs regardless of what slot it is in.
01000	**On-line:** This is the normal state of a board that is on-line simplexed. The board will listen to and perform ALL bus transactions directed to him. The board is capable of initiating bus transaction and will generate only even TRIDs regardless of what slot the board is in. In this mode, the board will respond to and drive the Xbus during a return of I/O paired read data.
01010	**On-line/Update:** This is the state of a simplexed on-line board that will broadcast all local memory writes to the Xbus. The board will listen to ALL bus transactions directed toward it. The board is capable of initiating bus transaction and will generate only even TRIDs regardless of what slot the board is in. In this mode, the board will respond to and drive the Xbus during a return of I/O paired read data.
01011	**On-line/Update/Freeze:** This is the state of an on-line simplexed board when it is doing the final copy of state over to it's partner board. The board will ONLY listen to transactions that have the same TRID as its slot number or its partners slot number. (i.e. odd or even TRIDs). The board will BUSY all other new transactions directed towards him. The board may initiate transactions and will use even TRIDs.
00011	**Off-line/Update/Freeze:** This is the state of an off-line board that is getting state copied in via I/O space. The board will listen ONLY to transactions that have the same TRID as it's slot number or it's partners slot number. (i.e. odd or even TRIDs). The board will BUSY all other new transactions directed towards him. The board may initiate transactions and will use odd TRIDs.
01100	**Duplexed:** Entered only after the board went through the Sync point and it was a duplex request. Both boards enter this mode simultaneously and are considered duplexed and running in lock-step. The 'update' and 'freeze-state' bits are cleared upon entering this mode. Odd and even slot boards will generate even TRIDs. On paired I/O reads, the board will perform the read of the register in lock step with it's partner, and both boards will drive the Xbus during the return of the data.This mode is cleared in the result of a T/A failure, or by this board or its partner going Broken.

Online for Dumping, loop-back mode: These modes are unsupported on the Xbus.

Note: States not listed are considered illegal

Table 15 below gives greater detail on operations performed.

Table 15. CPU/Memory Board States[a]

MODE	BROKEN	ON-LINE	DUPLEXED	UPDATE	FREEZE-STATE[b]	NON-PAIRED I/O READ	PAIRED I/O READ	NON-PAIRED I/O WRITE	CPU[c] OR PAIRED I/O WRITE	GLOBAL I/O WR	I/O RD/WR TO PARTNER	MEMORY READS	MEMORY WRITES	LOAD/CLR	FLUSHES	PURGES	INFO RETURN 256	DATA RETURN	GENERATE TRANSACTION?
Broken	1	X	X	X	X	I	I	I	I	I	I	I	I	I	I	I	I	I	N
Off-line	0	0	0	0	0	P	I	P	I	P	I	I	I	I	I	I	I	P	O
Off-line/Update	0	0	0	1	0	P	I	P	P	P	I	I	P	P	I	I	P	P	O
On-line	0	1	0	0	0	P	P	P	P	P	I	P	P	P	P	P	P	P	E
On-line/Update	0	1	0	1	0	P	P	P	P	P	I	P	P	P	P	P	P	P	E
On-line/Update/Freeze	0	1	0	1	1	B	B	B	B	B	I	B	B	B	B	B	B	P	E
Off-line/Update/Freeze	0	0	0	1	1	B	I	B	B	B	I	I	B	B	I	I	B	P	O
Duplexed	0	1	1	0	0	P	P	P	P	P	M	P	P	P	P	P	P	P	E
Duplexed/Update	0	1	1	1	0	P	P	P	P	P	M	P	P	P	P	P	P	P	E
Duplexed/Update/Freeze	0	1	1	1	1	B	B	B	B	B	B	B	B	B	B	B	B	P	E

a. KEY

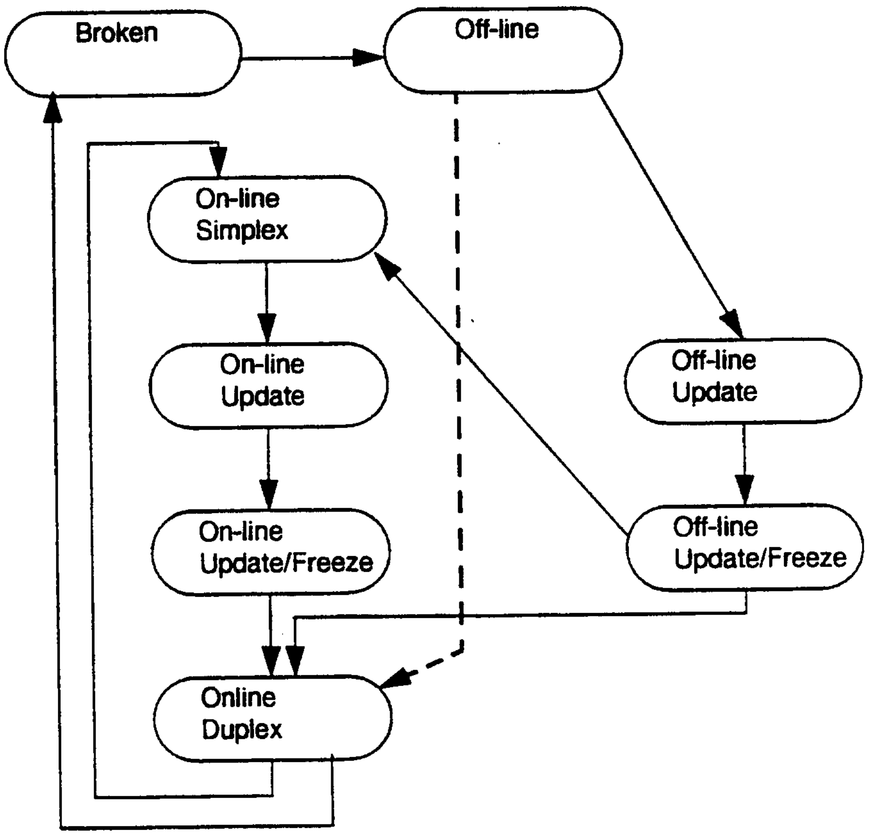
1	State bit true.
0	State bit false.
X	State bit true or false
P	The board will perform this Xbus transaction fully.
M	The board will go through the motions of performing this Xbus transaction but will not drive the Xbus in the info phase of the data return for this transaction.
I	The board will ignore this operation. (i.e. will not drive ACK, BUSY or return data.
B	The board will busy this transaction unless the TRID of the transaction matches its slot number or its partners slot number. Note: the board will busy the transactions until it is released from this mode. EFQ Freeze State busies cycles sent to the EFQ (except data returns), and Freeze State busies cycles sent to the RWQ. Refer to section 3.3 for more details on freeze states, the EFQ, and the RWQ.
N	The board will never initiate a transaction in this mode.
O	The board can initiate all transactions in this mode and will only use odd TRIDs.
E	The board can initiate all transactions in this mode and will only use even TRIDs.

b. Refers to either EFQ Freeze State or Freeze State

c. CPU I/O write refers to an I/O write that is broadcast to all CPUs.

Figure 44 shows the recommended state transitions for CPU/memory boards. Not all possible transitions are shown here, only the ones necessary for software to put a board on-line simplexed or on-line duplexed. A board can transition to the broken state from any state but this is not shown in the diagram to improve readability. The dotted line is the fast duplex path.

Figure 44. CPU/Memory Board state transitions



6.11.2 Simplexed I/O Board Bus Interface Modes

Below is a description of the board states needed to bring a I/O board on-line that has no capability of duplexing. Note, a simplexed I/O board will never respond to (or ACK) paired I/O read transactions.

The following signals are required on this type of board.

Broken On-line (BO)

BO	
1X	**Broken** The board is invisible to the Xbus. The board will only perform I/O writes when possible. If the board is busy performing an I/O transaction, the board will attempt to busy additional I/O transactions. Because the board is broken, it cannot drive the busy Xbus signal. This means that any I/O writes that occur when a broken board is busy fall to the bit bucket and are not performed. The write will also not occur if defective hardware precludes the write from occurring.

00 **Off-line** The board only responds to non-paired I/O read and I/O write transactions. The board will arbitrate for the bus to return I/O read data. This board will never respond to (or ACK) any paired I/O read transactions. The board is capable of initiating bus transactions but conceptually, the board should not generate transactions. Software should enforce this where needed.

01 **On-Line/Simplexed** This is the normal state of a board that is on-line simplexed. The board will listen to ALL bus transactions. The board is capable of initiating bus transaction and will generate TRIDs that are equal to its slot ID. This board will never respond to (or ACK) paired I/O read transactions.

loop-back mode This mode is unsupported on the Xbus.

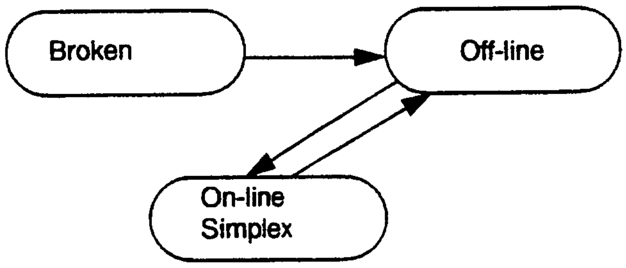
Table 16. Simplexed I/O Board states[a]

board state	BROKEN	ON-LIINE	NON-PAIRED I/O READ	PAIRED I/O READ	NON-PAIRED I/O WRITE	PAIRED I/O WRITE	GLOBAL	I/O RD/WR TO PARTNER	GENERATE TRANSACTION?
Broken	1	X	I	I	T	I	T	I	N
Off-line	0	0	P	I	·P	I	P	I	Y
On-line	0	1	P	I	P	I	P	I	Y

a. KEY:

1	State bit true.
0	State bit false.
X	State bit true or false
T	Will perform if possible (i.e. if not busy performing one just before it)
P	The board will perform this Xbus transaction fully.
I	The board will ignore this operation. (i.e. it will not drive ACK, BUSY or return data.)
N	The board will never initiate a transaction in this mode.
Y	The board can initiate all transactions in this mode and will use the TRID of its slot

Figure 45. Simplex I/O Board State Transitions



6.11.3 Loopback mode

The Xbus does not support Golfbus style loopback testing.

6.12 Board Breaking/Board Removal

6.12.1 Un-breaking a Board

Boards should generally be "unbroken" by resetting them rather than directly clearing the broken bit in the Bus Interface Status register (see the FTIO / PCIB Programmer's Guide). There is at least one (unlikely) scenario where simply clearing the broken bit in the Bus Interface Status register can cause a board to receive multiple read returns, specifically:

- A board acknowledges an Xbus read request targeted to it and then breaks. While it is broken, the original requestor PINGs the read. Since the board is broken, the PING will not be ACKed and the requestor will then return 0's to its on-board processor. If software then simply clears the broken bit in the Bus Interface Status register, the original board may then return the read to the requestor (assuming it took awhile to process the read), thereby giving the requestor a second response on a read it has already passed up 0's to the processor for. Clearing broken via reset in this case would have ensured that the broken board clears the read response out of its state machines prior to unbreaking.

Also, there should be some delay between the time a board breaks and when it un-breaks and starts performing Xbus accesses to other boards. The following (extremely unlikely) danger otherwise exists:

- A board performs a read request, then breaks, and then gets unbroken via a reset. At that time, it immediately generates another read request and happens to reuse the same trid. It's possible that the read data from the first (pre-broken) read (which may be to a completely unrelated register) then comes back and gets grabbed for the second (post-broken) read.

It's difficult to quantify how large of a gap is necessary between breaking and issuing the first read access after unbreaking, but even without precautions, this problem is very unlikely. Normal onboard diagnostics which are executed as part of the unbreaking code are probably long enough in duration so as make the above scenario impossible.

7. Xbus Routing and Xbus Interface Clocking

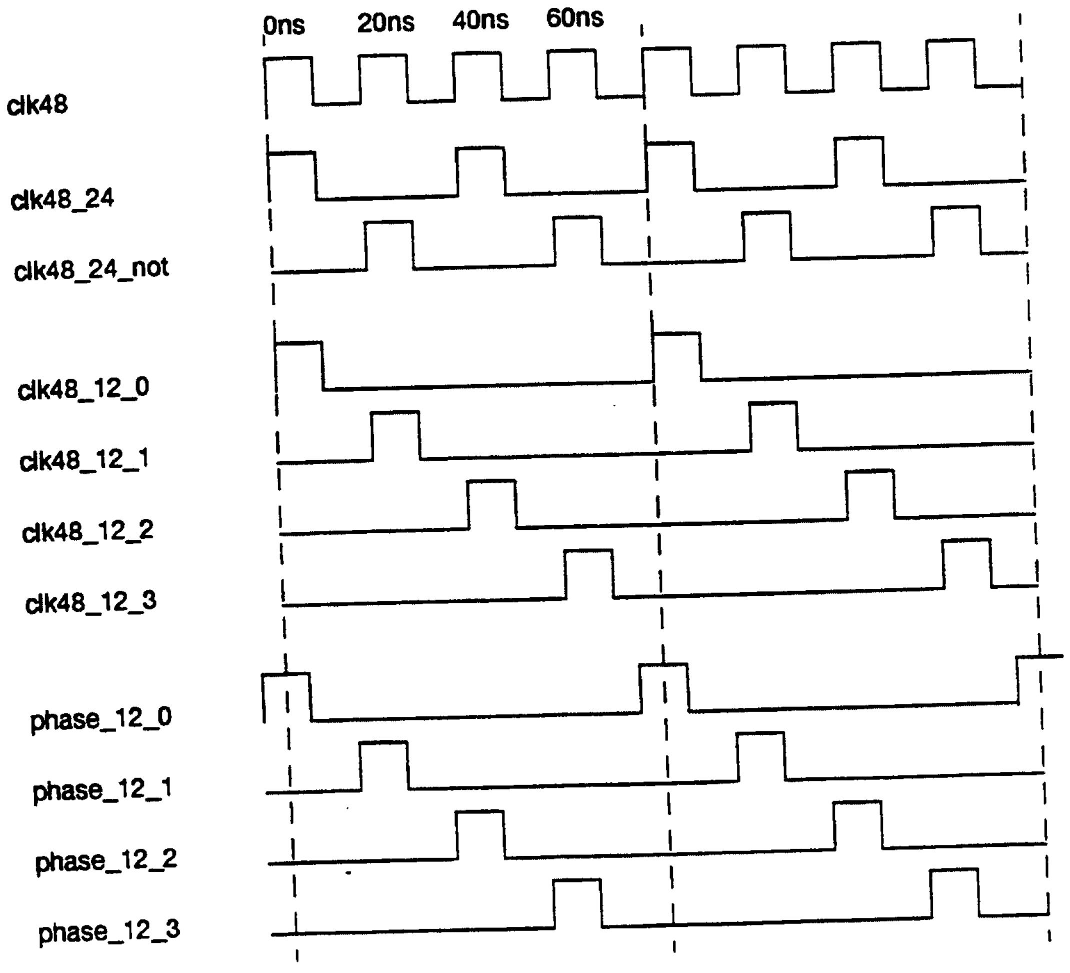
All information transferred across the XBus is synchronously clocked onto and off of the bus at 24MHz. The bused, control and TA signals are driven and received directly by the bus ASICs. The voted signals are buffered by 26S10 transceivers. Clock are generated from a single source located on the backplane and distributed through ECL clock lines.

7.1 On-board Clock Generation

The Polo project reuses much of the Jetta System clock design. The Sentry ASIC receives an 8 MHz clock from the backplane, and generates a 48MHz clock and 4MHz phasor for the bus ASICs. The Jetta 48MHz 0.8 micron DLL (delay locked loop) and active clock spine are used for distributing a single 48MHz clock on the ASICs.

7.1.1 Clock Phases

All flops in the Gambit and Cyclops ASICs are clocked by a single 48MHz clock spine. However, it is often necessary to have PCI and Xbus related logic behave as if clocked at 24MHz or 12MHz. To accomplish this, we define a group of pseudo-clocks that run at 24 and 12MHz, and have rising edges at all of the useful times.

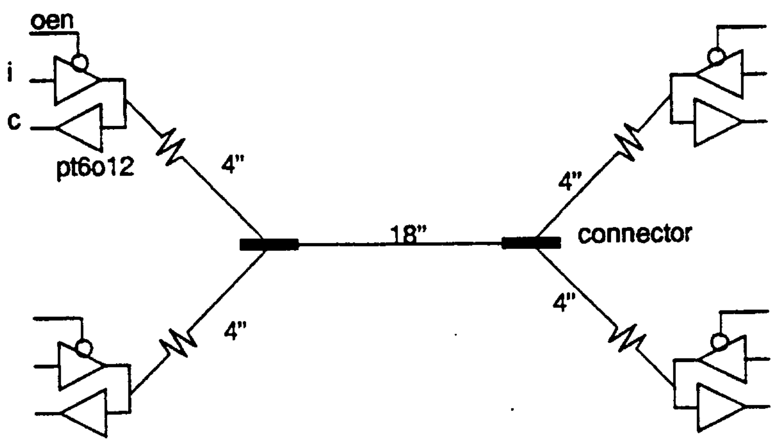


There is really only one clock in the ASICs, clk48. However, by controlling flops with the phase signals on the clock enables, it is possible to make it appear that the design has all of the clocks shown above. Phase_12_0 is used to create clk48_12_0, etc. The flops themselves are a D flop with a mux on the input, where the mux selects between new data and Q output. The phasors are connected to the select line on the mux.

7.2 Signal Routing

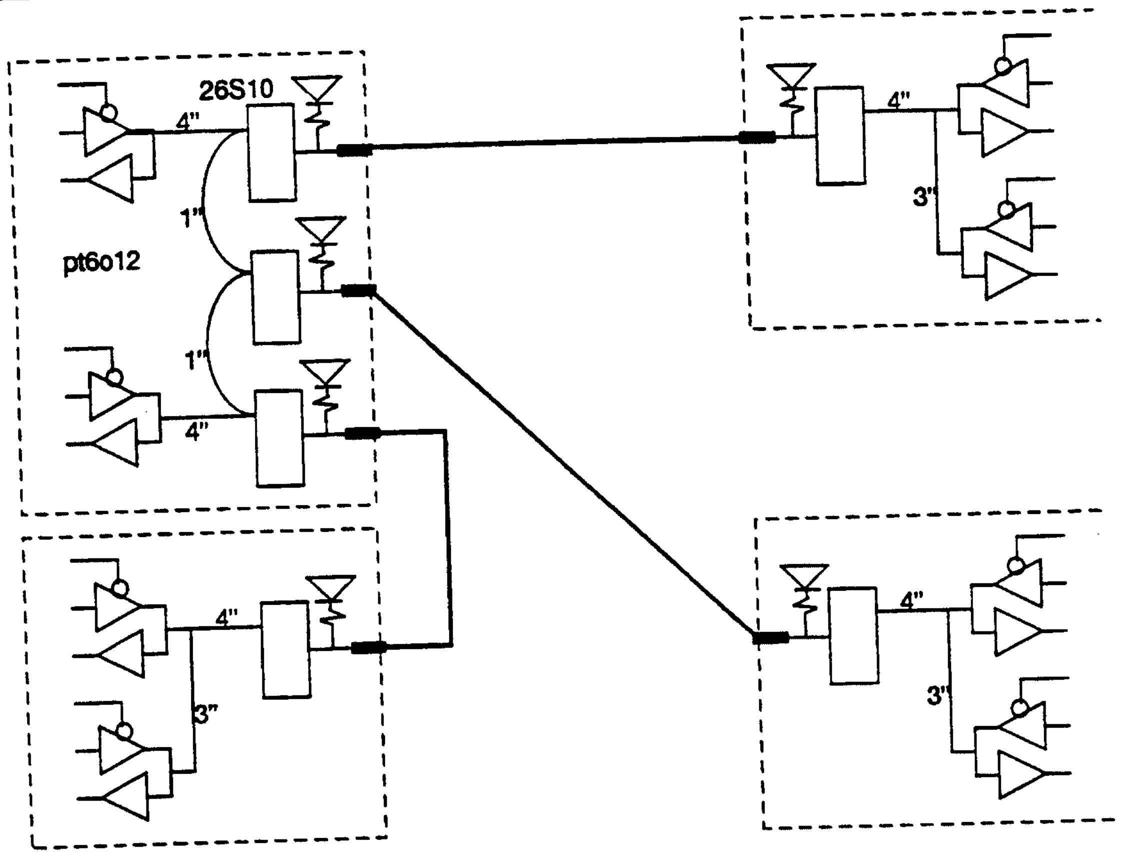
The bidirectional info buses are routed point to point between boards. On each board, separate traces run from the connector to each ASIC; this allows the error protocol to determine if a fault is in the backplane or in the board. Series resistors are used to control signal quality, and CMOS levels are used for increased noise margin and reduced susceptibility to cross-talk.

Figure 46. Routing of info lines



The three way voted signals are routed as separate point to point connections. The reset_x_,y_,z_ signals are driven by the CPU boards, wire-ORed on the backplane, and received by all boards. The board_not_broken_ signal is driven by a single board and received by all boards. These signals are driven by normal 4mA drivers of the ASIC, and buffered by 26S10 transceivers. The voted signals are terminated with a pull-up and isolation diode on both the receiver and driver side. This means that it is possible for there to be one or two pull ups on the line. However, whenever the line is being used for active signalling, there are exactly two terminators, one at either end of the line.

Figure 47. Routing of 3way voted lines



7.3 Clock Fault Tolerance Issues, Outbound Signals.

The Polo system must protect against failures of a clock chip on a board. There are two physical broken lines on a board, one with all clocking from the C-side clock chip and another with all clocking from the D-side clock chip.

The Xbus has all clocking of signals on and off the bus performed within the bus ASICs. If a side of the board loses its clock, the other side will detect data loopback errors and break the board. The three way voted signals are buffered by 26S10 transceivers, and are not susceptible to clocking failures.

In Polo the clock distribution is not triplicated. An analysis on clock line distribution was performed and it was calculated that due to the small number of clock loads system reliability was lowered by adding extra drivers to the system to perform the clock routing triplication.

See the Polo Clock Distribution Specification and the Sentry Clock Recovery Device Specification for a discussion of clock fault tolerance at a system level.

8. Board Insertion and Removal

8.1 Hot Plugging

Polo is the first Stratus produce to interface CMOS ASICs directly to the backplane. As such, it is the first to address live insertion of CMOS parts.

If a CMOS part is supplied significant current through its I/O pins before its power grid has come up, the part will "latch-up". That is, parasitic bipolar transistors at the I/Os behave like an SCR (Silicon Control Rectifier), and the part becomes stuck in a non-functional state. The acceptable limit is one diode drop; as a part powers up, the voltage on any I/O pin must not be more than 0.6v above the VCC inputs. To prevent this, all backplane signals connecting to the Polo bus ASICs must be tristate or driven low while a Polo board is powering up. Whenever the possibility exists that a board is unpowered (as detected via the board_not_broken signals), the other boards must drive any signals connected to that board low. This should be done by driving low for one cycle, then tristating the lines. All nets directly connected to another board's ASIC should be equipped with pull down resistors, instead of the pull-ups used on Jetta. signals buffered by an intermediate part, such as the 26S10s, require no special treatment.

implementation detail (testability)

8.2 Xbus Interface Testing at Board Insertion

The Xbus interface does not include Golfbus style hardware support for bus interface testing. Instead, Xbus systems will rely heavily on scan testing in manufacturing to detect faults in the error detection logic.

[illegible]eems like simple solution. [illegible]

[illegible]?

bipolar channel (separate from Control)

asserted if not ready to talk.

9. Xbus Fault Tolerance

The goal for fault tolerance on the Xbus is for no single point of failure to cause a system crash or allow a transaction to complete without transmitting the correct data. A single point of failure could be any component, any signal, or any pin on a component.

The Xbus fault tolerance scheme assumes that one of the components connected to the bus is always responsible for hard bus errors. To this end, the design has been simplified around the assumption that when an error occurs, an offending Field Replaceable Unit (FRU) will be removed from service. This extends to include all buses that the FRU is connected. Thus when an Xbus fault occurs the faulted bus service is removed taking at least one FRU with it.

Both sides of the CPU board drive signals to the bus and both sides of the board perform various checks to ensure that the board is functioning normally. This is identical to the Golfbus methodology. The PCIB board behaves differently; both sides of the board drive and check the bus when driving data from Xbus related IO registers, and when driving data originating on a simplexed PCI card, only one ASIC drives and checks the entire width of the bus.

The Xbus signals are divided into three classes: info bus signals, control bus signals, and three-way voted signals. The fault tolerance methodologies are different for each of these classes.

9.1 Info Bus Protection

The info bus is protected by parity and loopback checking. However there is no A to B bus cross checking by anyone on the bus other than the bus master. When a PCIB is bus master only one of the two buses attached to a given CPU module will be active because the PCIBs do not duplex and therefore the bus from the PCIB that did not win arbitration must be idle. Likewise the PCIBs cannot know if two duplexed CPU modules are driving the bus or a single simplexed CPU has ownership (two simplexed CPUs could have started arbitration at the same instant). Thus receivers on the Xbus will only listen to the bus that is driven by the module that won arbitration.

9.1.1 Parity

Parity is generated and transmitted across each of the physical buses. The 32 bit bus, trid field, and func_op bits are all covered by a single parity bit. The purpose of the parity check is to protect against etches that open between the transmitter and receiver. Boards in the system check the transmitted parity that they receive with parity that they generate themselves.

A bus error is signaled on any parity error.The bus_err signals are also effected by other error checking that is described later in this document. In an effort to simplify these descriptions the reader may assume that the actual error signals are a logical OR'ing of the various outputs from different checking blocks.

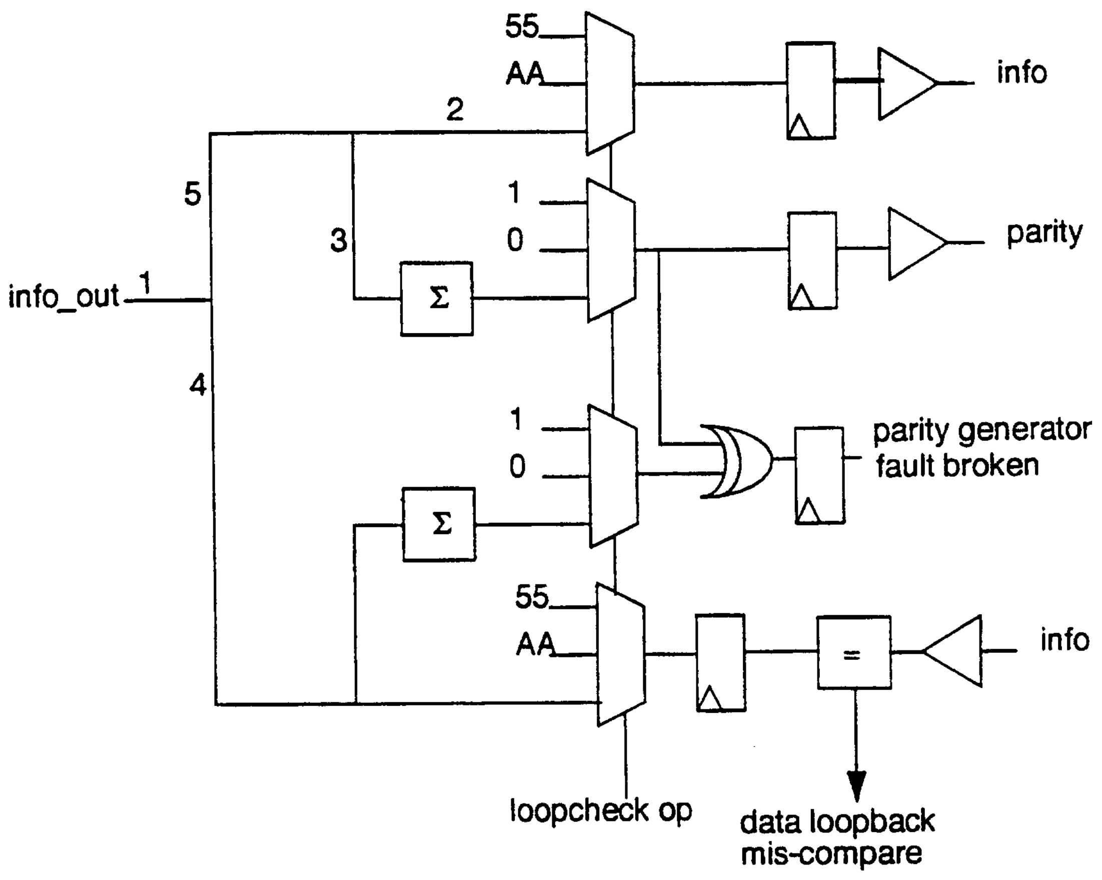
Both the C and the D-sides of the boards will perform the check and compare results individually and decide on any action that must be taken. In the event that the two sides of a board do not agree upon the status of a bus cycle, the board will break in the following cycle in one of two ways.

- The D-side of the board detected an error and the C-side did not. In this case the D-side will drive a bus error and the C-side will disagree with what the D-side has done and break the board. Here the transaction is errored and repeated, but the broken board should no longer be checking and the transaction will complete.
- The C-side of the board detected an errc: and the D-side did not. Here the C-side will once again break the board, but this time because it expected to see an error on the bus and didn't.

The transaction will complete normally and the bad board will be removed from service.

CPU boards compute parity in both C and D ASICs and break if a defect arises in one side's parity generators. Most of the transfers from a PCIB board, however, originate from a PCI bus connected to a single Gambit ASIC. When the Gambit drives this single sided data, it drives all the bits on the Xbus, and an error in the simplexed parity generator would hang the system. Fortunately, there is no need to duplicate the entire Gambit ASIC, only the parity generation section.

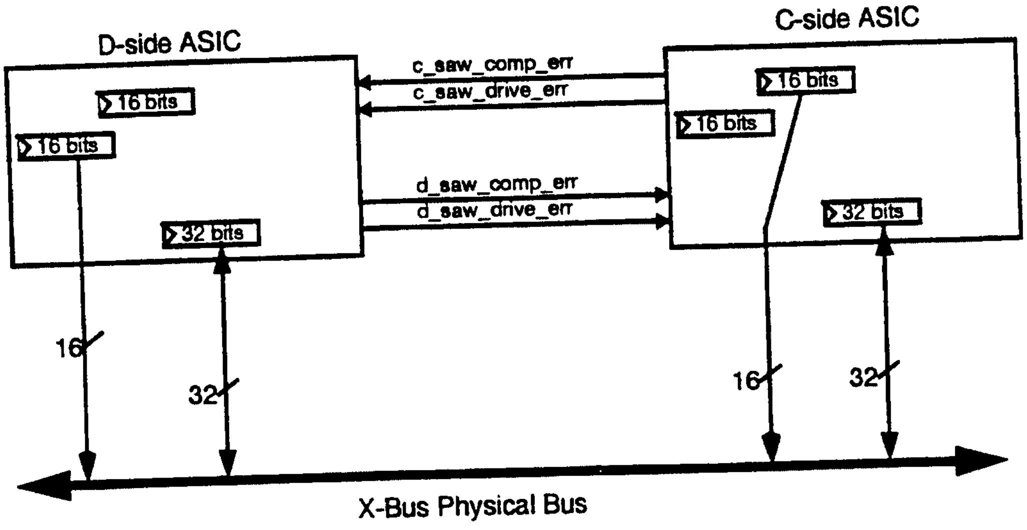
Figure 48. Self Checking Parity Logic.,



- A fault at site 1 causes erroneous data to be transmitted with correct parity. This will be detected by higher level checksums.
- A fault at site 2 will cause the board to break with a data loopback fault.
- A fault at site 3 will cause the board to break with a parity generator fault
- A fault at site 4 or 5 will break the board with either a data loopback fault or a parity generator fault.

9.1.2 Loopback checking

Figure 49. Loopback Connectivity



On normal accesses (non-CD different, non-single sided), each side of the board drives half of each field to the bus and receives all of both busses. Thus there are two checks that each ASIC can do on the data returning from the bus.

1: Does the returning data match the data that was driven from this ASIC. This will be known as the loopback drive check.

2: Does the returning data match the data that was driven by the other ASIC. This will be known as the loopback compare check.

Two error signals are generated inside of each ASIC for each physical bus that it is driving. In order to know the true state of the board each side of the board must know the results of the comparisons on the other side as well as the comparisons it has done. Thus two signals are passed in each direction, as shown in figure 49 above, so that the each side may correctly determine the state of the board.

Each ASIC will determine wether the data on the bus is faulted or not, as well as if this board is broken or not. Table 17 below shows what combinations of the loopback signals for each bus indicate that the board is broken.

Table 17. Board Broken Matrix for a Single Bus

my drive_err	my comp_err	drive_err from the other side of the board	comp_err from the other side of the board	brd_broken
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1

Table 17. Board Broken Matrix for a Single Bus

my drive_err	my comp_err	drive_err from the other side of the board	comp_err from the other side of the board	brd_broken
0	1	0	0	1
0	1	0	1	1
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

A double bus error will be asserted and the not_broken signal de-asserted if a condition exists that should break the board. In addition a bus error will be asserted for any condition where a compare error of any type has occurred and the board has not broken. The bus error will only be on the bus while the error is detected.

9.1.2.1 Loopback on CD different accesses

When an ASIC drives data from a CD different register on the bus, it performs loopback checking only on the trid, func_op_, and parity lines. It asserts bus error if it sees an error, and will go broken according to table 17

9.1.2.2 Loopback on Single-side accesses

When a Gambit ASIC drives single-side data from the PCI it performs loopback checking on all of the bits; it does this only for the purpose of asserting bus error, not to see if it should go broken. It will not break according to table 17, because the cmp_err and drive_err signals from the otherside are invalid. However, if the driving ASIC sees a bus error, and the otherside ASIC does not (parity received good), the board will break due to a loopback on control fault.

9.1.3 Loop_ck_ops

Loop_ck_ops are used during the error sequence to make intelligent choices about the state of various boards and the buses that they are attached to. A loop check op is a cycle in which a board drives an alternating pattern of AA's and 55's on the backplane. The board evaluates the pattern and similar patterns driven by boards on the other end of the bus. Based on this information, bus state is altered.

9.2 Control Bus Protection

The Xbus collects control signals into a series of point-to-point unidirectional signals protected by an ECC code. Each board drives a separate control bus to each of the other boards.

There are two completely independent types of checks performed:

1: Every ASIC corrects for any single point failures (shorts, opens, dead drivers or receivers) with a single error correcting, double error detecting ECC code. Single bit failures cause a maintenance interrupt and are logged, but otherwise have no effect.

2: If a driving board detects a double bit error on the outgoing control signals it will attempt to assert bus error and then it will break in the error sequence (it will break regardless of its ability to drive bus error). If a receiving board detects a double bit error on any incoming control signals it will attempt to assert bus error then it will break in the error sequence. If the error was actually on one board and only one board saw the error then the correct board will break. If the error was such that both boards saw it then both boards will break. This is to limit the possibility that data gets corrupted.

9.3 Three-way Voted Signal Protection

Some broken and reset related control signals on the Xbus are triplicated and voted. The reset and board_not_broken_out signals are buffered through transceivers because they must be valid during power up or power down events. The CPU signal sync_out is buffered through transceivers to reduce ASIC pin count. The CPU signal my_online_out is sourced directly by the ASIC as a cost reduction.

The D-side ASIC drives signals to the transceivers and to the C-side ASIC. The C-side of the board checks what was driven by the D-side with what it thinks should have been driven. If the C-side disagrees with the D-side the board breaks.

Each ASIC that receives a three-way voted signal will perform a majority voting algorithm on the three signals to determine the value. In normal operation all three lines will be the same. If there is a fault on one of the three lines (either on the backplane or on the board), two of the three lines still will agree.

When a voting error occurs, the device detecting the fault should send a maintenance interrupt and log which transceiver has the non-unanimous vote. The logged information is stored in the Voter Error Transceiver Status register(s) and will not be over-written until the status register is explicitly re-enabled by software.

9.4 Error Reporting

A number of errors can be detected by a board's bus interface logic:

1: A loopback error that breaks the board. This specifically refers to loopback errors on the **info, parity, TRID,** and **func_op** signals which are seen by one side of the board and not the other during bus cycles in which the board drove the bus.

2: A disagreement between the two-sides of a board on whether a three-way voted signal should be driven.

3: A parity error on the bus, either on a cycle where the local board was driving the bus or another board was driving the bus.

4: A loopback error on the bus.

5: A three way voter error in which not all three signals are in agreement.

6: Any bus error signaled by any board in the system

7: A thermal fault (high temperature detected on the board)

Some of the above errors may break a board (e.g. #1 and 2), some may cause a bus error to be generated (e.g. # 3, 4 and 6), and some have no effect on normal machine operation (e.g. #7). The Xbus interface contains two classes of status registers to record information surrounding the above errors: Broken Status and Error Status. Broken Status registers are only activated when a board goes broken, and are designed to pinpoint the logic that set broken. Error Status registers are for reporting non-fatal errors such as faults on the backplane. All of the Error Status registers are updated simultaneously on any bus error signaled by the system, or a voter or thermal error on the board. Specifically, when all error reporting is enabled, any of the above errors cause:

1: A maintenance interrupt to be generated

2: The type of error condition to be stored.

In the above case, subsequent error state recording and the resulting maintenance interrupts are disabled until explicitly re-enabled.

It is not always optimal that all the above errors trigger the error registers and generate a maintenance interrupt. The reporting of new errors can be effectively blocked by an excessive number of reports on a known error. Hence, it is possible to turn off error latching due to different sources:

1: Perform error latching and maintenance interrupts on all errors

2: Inhibit error latching and maintenance interrupts of incoming three-way voter errors

3: Inhibit error latching and maintenance interrupts on thermal failure related errors

The error latching disable bits inhibit those faults from triggering the error registers. However, if a disabled error is present and non-disabled error occurs, both errors will be recorded in the error registers.

Details on the operation of this error reporting circuitry are provided in the detailed IO register definition section of the Polo Programmer's Guide. The specific registers that implement the above functionality are the Bus Interface Fault Reporting Registers, the Bus Info Error Status Registers, the Misc Error Status Registers, the Bus Error Transceiver Status[1:0] Registers, and the Voter Error Transceiver Status Registers.

9.4.1 Producing Errors

The Cyclops and Gambit gate arrays have specific registers which allow software to produce various errors on the Xbus. The registers are described in detail in Polo Programmer's Guide. These registers allow the error logic to produce all of the error cases described in section 6.4.3, Fault Conditions.

The basic logic to produce an error consists of XOR gates on all the info, trid, func_op, parity, control lines going out and all the same lines coming in. One input on the XOR is the data line the other input is the error line and the output is the resultant data. If the error line is high the data coming out will be in error. If the error line is low then the data coming out will be the correct data. If data is errored only on the input or only on the output then the error will be in the form of a loopback error. If the data is errored on the input and the output then the error will be in the form of a bus error since it will not be a loopback error.

Table 18 on page 86 shows all the error cases and the necessary settings from a hardware point of view to produce the error. 'receive error' means the XOR gate on the receive side is inverting. 'transmit error' means the XOR gate on the transmit side is inverting. The category describes if the error occurs only on a particular pattern match or if the error occurs just after the register is written. For many cases it would be possible to produce an error with different cases than those described below. The goal is to only produce the error in one manner.

Table 18. Error Settings

Type of Error	Category	D-Side	C-Side
CPU Board Error			
CPU Board Faulty Input Circuit - CPU Driving	Immediate	receive error	no error
CPU Board Faulty Input Circuit - I/O Board Driving	Data Match	receive error	no error
CPU Board Different Data C-Side and D-side	Immediate	transmit error /receive error	no error
CPU Board Faulty Output Circuit - Buffer to Pad Fault	Immediate	transmit error	no error
CPU Board Open - CPU Board Driving	Immediate	no error	receive error
CPU Board Open - I/O Board Driving	Data Match	receive error	no error
CPU Board Short	Immediate	transmit error	no error
Backplane Error			
Backplane Open Etch	Data Match	CPU and IO board transmit error /receive error	CPU and IO board receive error
Backplane Short	Immediate	CPU and IO board transmit error	CPU and IO board no error
IO Board Error			
I/O Board Faulty Input Circuit	Immediate	receive error	no error
I/O Board Output Circuit Fault - Buffer to Pad	Immediate	transmit error	no error
I/O Single-side Access - ASIC Parity Gen. Fault	Immediate	parity gen. fault	no error
I/O Single-side Access - ASIC PCI Data Path Fault	Immediate	bad address	no error
I/O Non-single-side Access, Different C-D Data	Immediate	transmit error /receive error	no error
I/O Board Open	Immediate	no error	receive error
I/O Board Short	Immediate	transmit error	no error

Type of Error	Category	D-Side	C-Side
Transient Error			
Transient Fault	This causes a bus error, but when the error sequence is executed there will be no error. This fault will not produce a maint. int. from this board, but all the other boards will assert maint. int.		
Bus Busy			
Bus Busy	This causes a bus busy.		

9.4.2 Maintenance Interrupts

Each Xbus board generates maintenance interrupts whenever an event occurs that would be of interest to the M&D software. These events fall into three classes: broken/unbroken events, error events, and software commands. As described in the following section, broken/unbroken events include board resets, board insertions, and board removals.

9.4.2.1 Broken Events

Boards transitioning to or from the broken state generate a maintenance interrupt. A board can break due to a detected hardware failure, a cold or warm reset, a board removal, or a software set broken command. A board can unbreak due to a software command, a reset, or an insertion.

Cold or warm resetting a board causes that board to go broken at the start of the reset and unbreak itself at the end of the reset, thereby generating two maintenance interrupts assuming the board was not initially broken (one maintenance interrupt if the board was initially broken). Inserting a board into the system generates a single maintenance interrupt when that board unbreaks itself.

9.4.2.2 Error Events

A number of error events may cause maintenance interrupts. These are reported in the Bus Info Error Status and Misc Error Status registers, defined in the Polo Programmer's Guide, and controlled via the Bus Interface Reporting Register, also defined in the Polo Programmer's Guide. The error events are:

1: A bus error on a backplane bus

2: A voter error on a three way voted backplane signal

3: A thermal fault

4: A failure detected by the clock recovery chip

5: A board specific fault/failure. Note that unlike Stratabus systems, main memory single bit ECC errors do not generate maintenance interrupts on the first generation Polo CPU/ Memory board.

As described in section 9.4 on page 84, it is possible to turn off maintenance interrupts from certain sources so that a known hard failure does not flood the system with maintenance interrupts. Specifically a voter error, a clock error, or a thermal fault may be disabled via the Bus

Interface Fault Reporting register.

9.4.2.3 Software Control

Software may explicitly activate maintenance interrupt by writing bit one of the Bus Interface Fault Reporting register.

9.4.2.4 Determining the Source of a Maintenance Interrupt

Upon seeing a maintenance interrupt in a system, the following process can be used to determine the source of the interrupt.

If a board that was in the system is no longer visible to software then that board generated a maintenance interrupt. To get more detail on what happened to the board, it can be reset and its Common Broken Status and ASIC Specific Broken Status registers examined. If multiple reset attempts have no effect, the board was either removed or is too broken to be read.

Boards that have not broken or not been removed from the system will have either the "Bus Interface Maint Int" bit or the "Board Logic Maint Int" bit set in the their Bus Interface Reporting register if they activated maintenance interrupt.

The "Board Logic Maint Int" is board specific and information on further identifying maintenance interrupts with this bit set is discussed in individual board specs.

If the "Bus Interface Maint Int" bit is set, then Board Reset register may be read to see if a maintenance interrupt was due to a board reset. Similarly, the Bus Info Error Status and Misc Error status registers may be read to determine if the maintenance interrupt was generated by any non-fatal errors (the board would be broken otherwise). If no bits are set in either of these two registers, the maintenance interrupt was software generated.

9.5 Board Breaking Timing

Note in the figures below, Info_0's shows the time the info bus is driven to zero assuming that the bus is not already tristated from this side. xcever_disable shows the time the drivers from the broken board are disabled from the bus. Bus error is also shown when valid on the Xbus.

9.5.1 Board breaking and information latching.

There are several status bits in the Broken Status register on each board to determine the reason a board went broken. These bits are frozen after a broken condition occurs and will remain frozen until after a cold reset. All bits in the common register are straight forward with the exception of **board_specific_set_broken**. Board specific logic may have several reasons to assert this signal and may also have a similar register for the reasons it has asserted this signal. If **board_specific_broken** is set in the common register, the board specific broken register will contain the real reason the board asserted the **board_specific_set_broken** signal. If a board breaks for a reason other than board specific broken, the **board_specific_broken** bit will not be set in the broken status register. However, **board_specific_set_broken** may have been asserted after the original broken condition. This would have caused the board specific broken status register to latch and hold a reason it had set broken. It is important to understand that because the **board_specific_broken** bit is not set in the common broken status register, this was not the reason the board went broken. The reason the board went broken is logged in the broken status register and the board specific status register may be ignored.

In the timing diagram below, **someone_set_broken** is the logical OR of the following signals:

cold_reset, **warm_reset**, **slot_parity_error_set_broken**, **control_or_3way_vote_sig_error**, **asic_specific_set_broken**, **ecc_dbl_error_in**. In addition Gambit contains the following signals: **break_pcib**, **xb_parity_gen_set_broken**. The board will break if any broken signal is asserted on any phase (i.e. any 48mhz edge). **my_set_broken_reg** is the equivalent of **otherside_set_broken_reg** on the other ASIC. **my_set_broken_reg** is ORed with **otherside_set_broken_reg** to create **set_broken_sync**.

The common broken status register is frozen when latched_broken_status is true. Note, All status signals for the I/O register have the same timing as broken.

Note the bus error driven on the bus at the time the board went broken. This timing is important.

Figure 50. Board Breaking Timing and latching (arrived at phase_12_1)

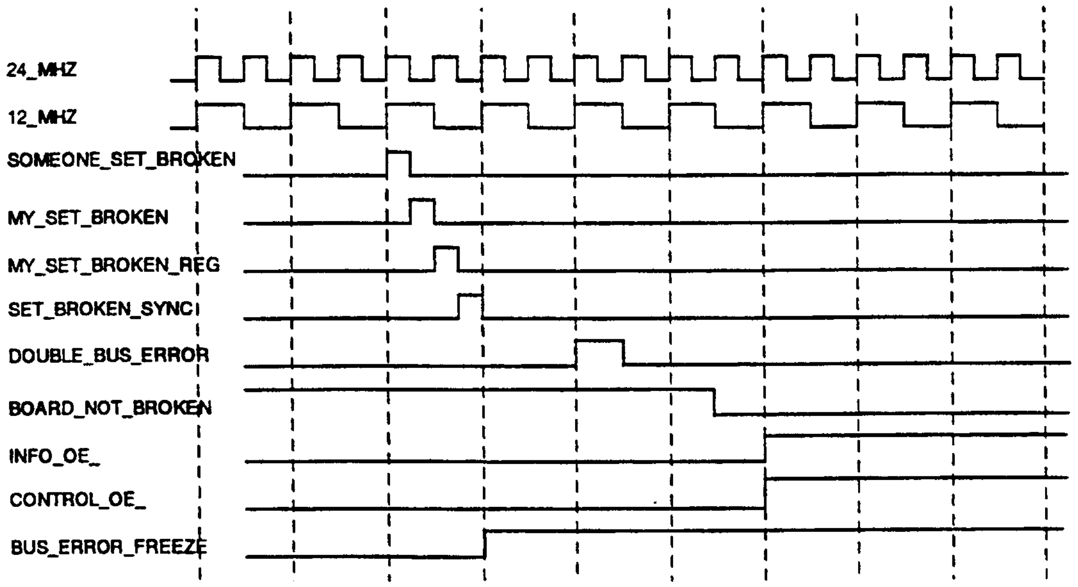
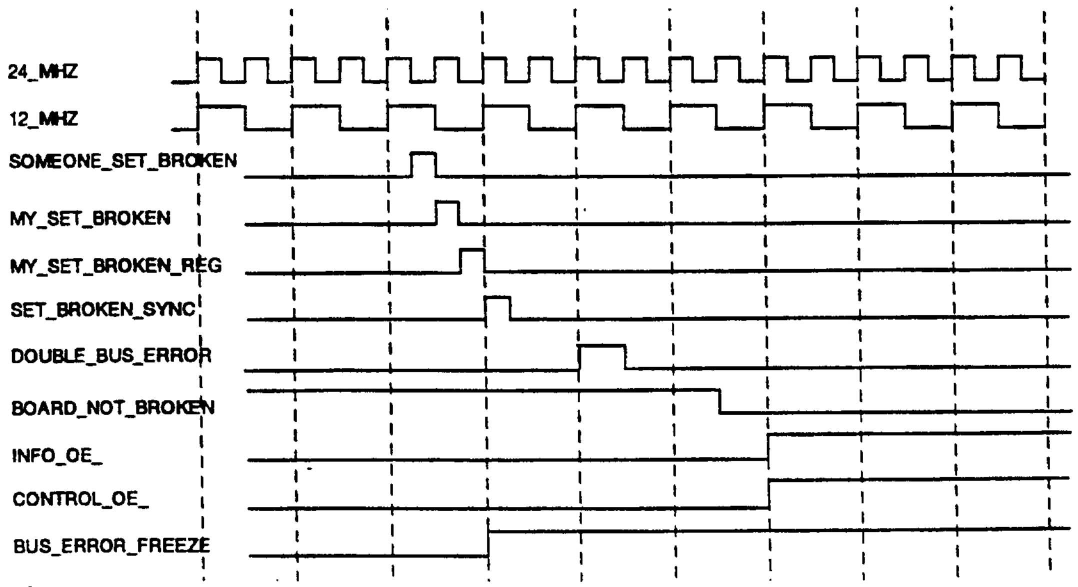


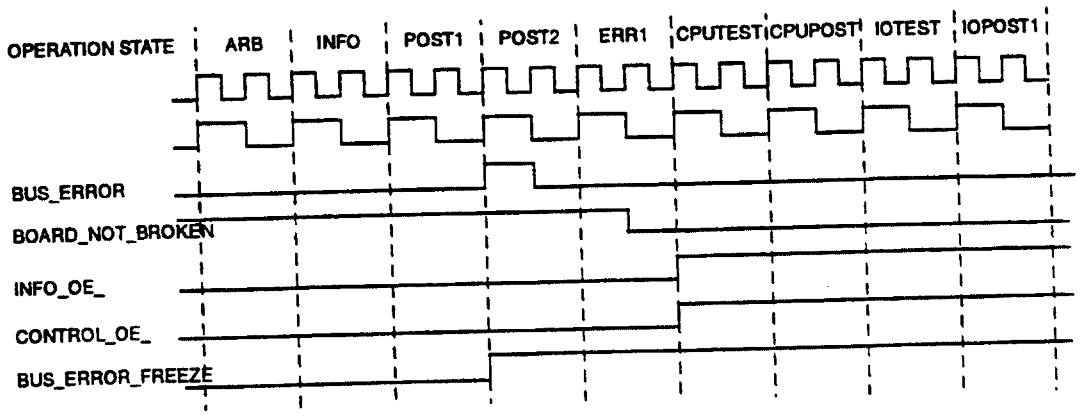
Figure 51. Board Breaking Timing and latching. (arrived at phase_12_2)



9.5.2 Board Breaking Timing on Info Loopback Error

If a loopback error occurs that will break the board, bus_error will be asserted in POST2 and the board will break in ERR1. Both ASICs will know of the error because of there are four lines for each info bus that communicate loopback status between the ASICs (cmp_err_in, cmp_err_out, drive_err_in, drive_err_out).

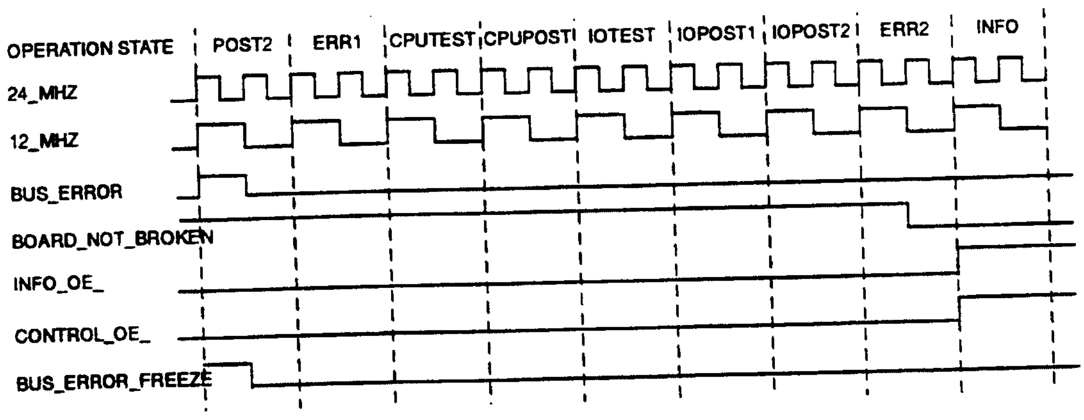
Figure 52. Board Breaking Timing on Info Loopback Error



9.5.3 Board Breaking Timing on Heuristic or Arbitrary Broken.

Boards that break due to a Heuristic or an Arbitrary broken will participate in the entire error sequence and then deassert board_not_broken in ERR2.

Figure 53. Board Breaking Timing on Heuristic or Arbitrary Broken



10. Reset

In the Polo system, there are three kinds of reset. A cold reset is generated through a software command and implemented as a set of dedicated lines on the Xbus. a warm reset uses the same method. Finally, there is a power up/down reset. These resets are described in this section.

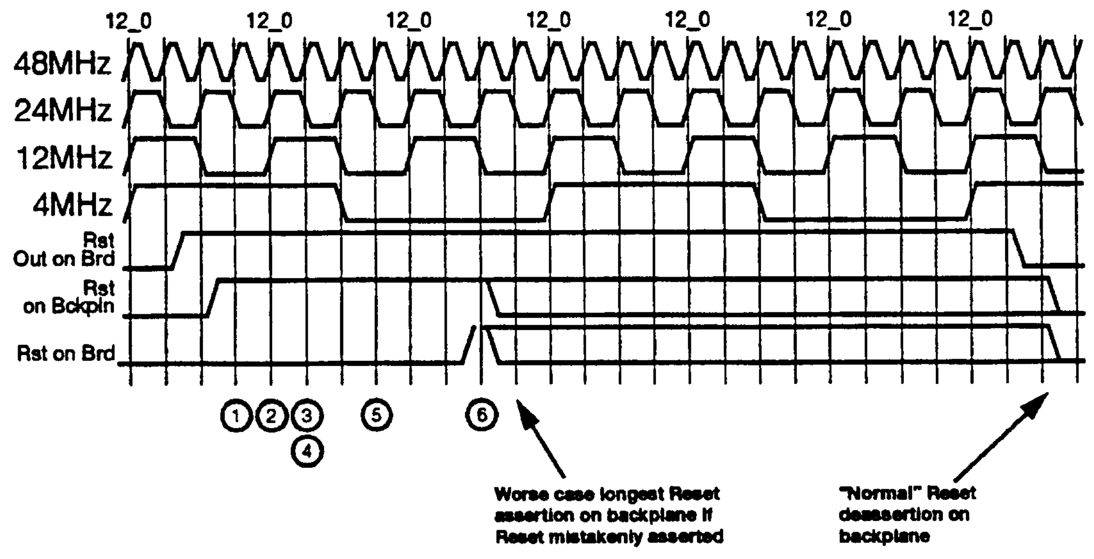
10.1 Xbus Resets

Xbus resets are handled differently on Polo than they are on Golfbus systems. In Polo, an out of service bus cannot be used for transmitting information, like I/O writes to cause warm or cold resets. Instead, Polo implements a set of dedicated lines for this function. The operation of these lines is similar in protocol to the ReCC reset on Jetta.

The three-way voted C/D-side comparison mechanism just described is such that if a board mistakenly drives a three-way voted signal due to a hardware failure, other boards will see that signal before the driving board breaks. For most of the control signals this does not adversely impact system integrity. An exception to this general rule are the warm and cold resets. Should a board mistakenly assert a reset line, it will eventually break itself, but not before potentially resetting one board or the entire system. This section describes how the Xbus addresses the above issues.

Drivers of reset are required to activate reset for 2 4MHz periods (12 24MHz bus phases) in order to cause a warm reset, and 3 4MHz periods in order to cause a cold reset. An example of a warm reset is shown below.

Figure 54. Reset Signal Timing


12_0
48MHz
24MHz
12MHz
4MHz
Rst Out on Brd
Rst on Bckpln
Rst on Brd
1
2
3
4
5
6
Worse case longest Reset assertion on backplane if Reset mistakenly asserted
"Normal" Reset deassertion on backplane

1: At 1st 12_3, the opposite side Cyclops ASIC checks the on-board reset signal

2: At the next 48MHz clock (12_0), the opposite side Cyclops drives a set broken signal to the bus ASIC

3: At the next 48MHz clock (12_1), the bus ASIC receives the set broken signal

4: This 12_1 clock edge is the latest the bus ASIC can receive the set broken signal and meet the illustrated timing for deasserting the signal on the bus.

5: At this 13_3 edge, the bus transceiver drives 0's to all the transceivers so as to synchronously deassert the signals it is driving on the bus.

6: Bus ASIC and external register disable transceivers here (latest reset will be disabled)

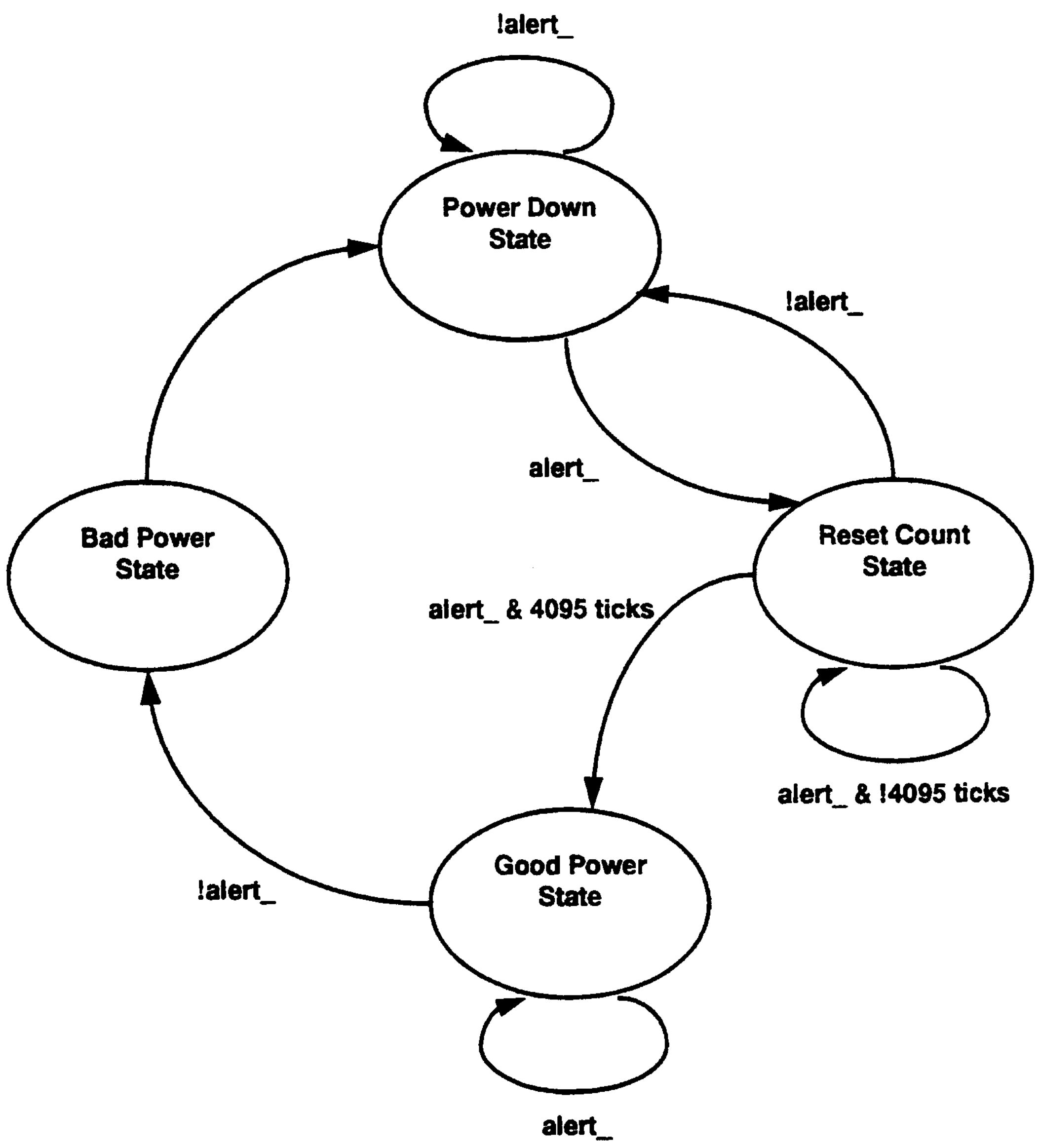
10.2 Power System Generated Reset

Board reset, from power related events, is generated by logic outside of the standard clocking of the ASICs. This is necessary due to the fact that ASIC DLLs have specific reset requirements that must be met. There are three signals associated with the reset sequence. The power supply drives a signal called power_fault_. This signal is asserted when the power supply voltage is out of specification. cc_lock_c_ and cc_lock_d_ are driven by the Sentry clock recovery chips when their clocks are out of specification. When all of these signals are de-asserted, logic high, the board level reset sequence is initiated. This consists of the phases described below.

power_fault_, cc_lock_c_, and cc_lock_d_ are ANDed together to form alert_. alert_ is used to release dumb logic, reset_n_c_, reset_n_d_, and trst_. The deassertion of reset_n(c/d)_ causes the ASIC DLLs to begin their initialization sequence. This can take 4096 48Mhz clock ticks. At the end of this phase, good_power is asserted signalling that the DLLs should be settled. At that point, the bus ASICs can conclude the reset sequence and deassert broken.

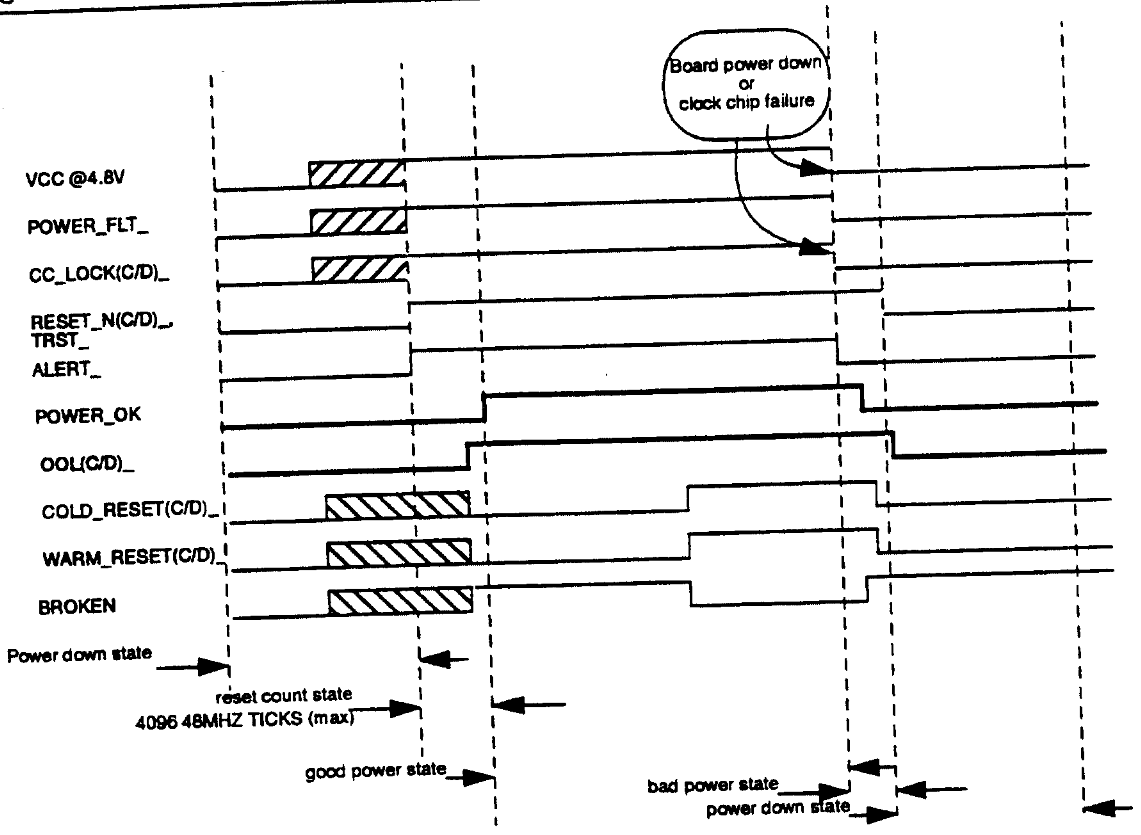
On power down, as soon as power_fault, cc_lock_c_, or cc_lock_d_ is de-asserted, alert_ is asserted and the power down sequence begins. good_power is immediately de-asserted. The bus ASIC must immediately assert broken and reset. 1 4Mhz clock later, trst_ and reset_n(c/d)_ is asserted and the power down sequence is complete. A state diagram for this sequence is shown in the figure below.

Figure 55. Power Reset State Machine



alert_ = power_fault_ & cc_lock_c _ & cc_lock_d_

Figure 56. Board resets.



VCC @4.8v	Asynchronous signal internal to power supply. Represents that the power supply is at 4.8 volts or greater.
POWER_FLT_	Asynchronous signal driven from power supply to reset logic only. represents that the power is within specification.
POWER_OK	Synchronous signal derived from the alert_ and driven to only the bus ASIC.
CC_LOCK(C/D)_	Asynchronous signal from common clock chip driven to the reset logic.
RESET_N(C/D)_	Asynchronous signal from the reset logic and driven to all ASICs with a DLL. This signal must be released before the ASIC's can lock their clocks. At this point it looks like this signal can be the equivalent of the power supply's DUMB signal.
OOL(C/D)_	Internal ASIC signal. Means the clock has achieved lock.
COLD_RESET(C/D/)_	Synchronous signal driven from the bus ASIC and received by all other ASICs. This signal initializes most internal logic.
WARM_RESET(C/D)_	Synchronous signal driven from the bus ASIC and received by all other ASICs. This signal initializes all internal logic.
TRST_	Asynchronous signal derived from DLL_RESET(C/D). Needed on all ASICs to initialize JTAG.
BROKEN	Synchronous signal internal to the bus ASIC. Means the board is broken.

10.3 Fault Tolerant Issues:

POWER OK is a synchronous signal driven from asynchronous ALERT_. It is not possible to remove a single signal run here. However, If POWER_OK is stuck false, the board should never come out of cold reset and never clear broken. If POWER_OK is stuck true, broken will be guaranteed set by the OOL_ signal. When broken is set, it needs a reset complete signal to occur before the board will clear broken. This cannot occur because reset is not active.

The power supply design currently asserts ALERT when CC_LOCK is lost. RESET_N_ should not be driven immediately after power or CC_LOCK_ is lost as this will cause even more clocks to go out of sync. RESET_N_ may be driven however after the board breaks.

An out of lock (OOL) condition on bus ASIC's will break the board but for other ASIC's we will rely on the out_of_lock condition to break the board for other C and D differences. Therefore a failure of reset_n_c_ or reset_n_d_ to a bus ASIC is not a single point of failure. However, a failure of reset_n_(c/d)_ to non-bus ASIC's will not cause an immediate broken condition. On boards that do not have any other bus interfaces (CPU boards), the OOL condition will eventually break the board when bad data is transferred between ASIC's. If the back-end ASIC loses its lock on (I/O) boards that have buses in addition to the Xbus, the board may still compare data correctly but not be able to talk to the bus in question due to timing violations. This should be taken under consideration on a board by board basis. If it is determined that broken should be set on an OOL condition then the boards may do so via the board_specific_set_broken signal.

10.4 ASIC Pins Required For Reset Functions.

The following pins are required for each ASIC.

resetn_	The Dll reset in signal
trst_	JTAG reset.
warm_reset_	Warm reset out
cold_reset_	Cold reset out

10.5 DUMB FET's

The Polo DUMB circuitry is a variation on the Stratabus boards' dumb/alert daughter card. This card provides dumb clamping when the board is in the power down and reset_count states.

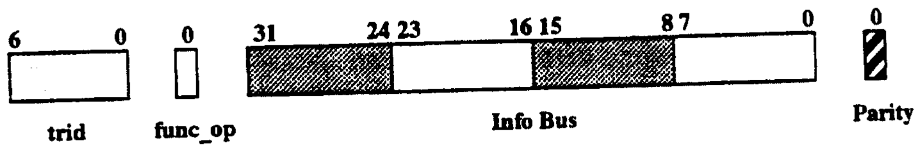
11. Xbus Physical Partitioning

This section gives a high level overview of the physical partitioning for the interface ASICs to the Xbus. It describes the components necessary to interface to the Xbus as well as a functional description of how the interface is intended to operate. The fault tolerant strategy is also described.

11.1 Info Bus Partitioning

The C and D ASICs on the CPU board each drive half of the bits on the info bus when they are bus master. The C and D ASICs on the PCIB drive either half or all of the bits, depending on the type of transfer. During normal accesses, they drive half the bits as shown; during single-side accesses, they drive all of the bits, including the parity bit.

Figure 57. Info Bus Partitioning



Bits are assigned starting from the least significant bit.

▨ Driven by the D-side ASIC during normal accesses, and by the driving ASIC during single-side accesses.

□ Driven by the C-side ASIC during normal accesses, and by the driving ASIC during single-side accesses.

▨ Parity is driven by the D-side on the CPU. It is driven by the D-side of the PCIB during normal accesses, and by the driving ASIC during Single Side accesses.

11.2 Control Signal Partitioning

The D-side ASICs drive the control buses, and both C and D-side ASICs check.

11.3 Xbus Voted Signal Partitioning

The 3-way voted signals in a Polo system have varying topologies depending on function. In general, each ASIC on a board drives half of the 3-way voted signals originating on a board, and the other side ASIC sits at the end of the net and checks what is driven.

In the following diagrams, each transceiver section (shown as a NAND gate or inverter) of a 3 way voted triplet is in a separate package, to prevent a single failure from disturbing more than one bit of the triplet.

Figure 58. board_not_broken_ routing

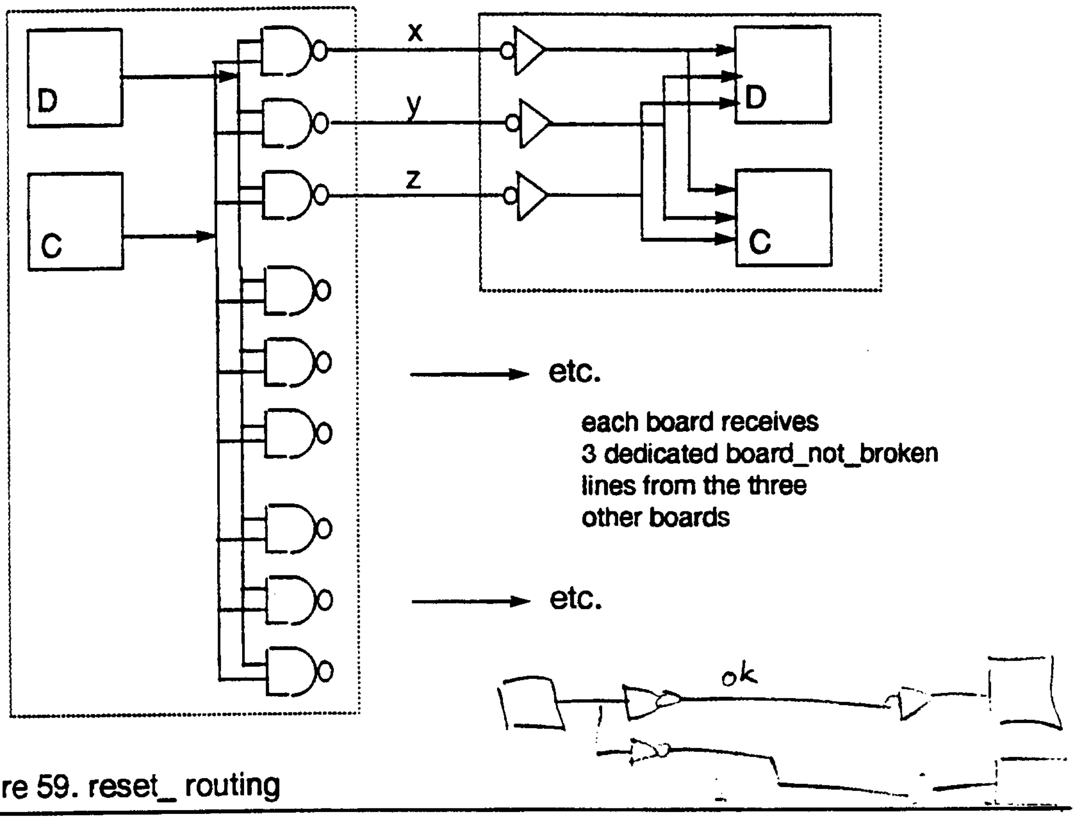
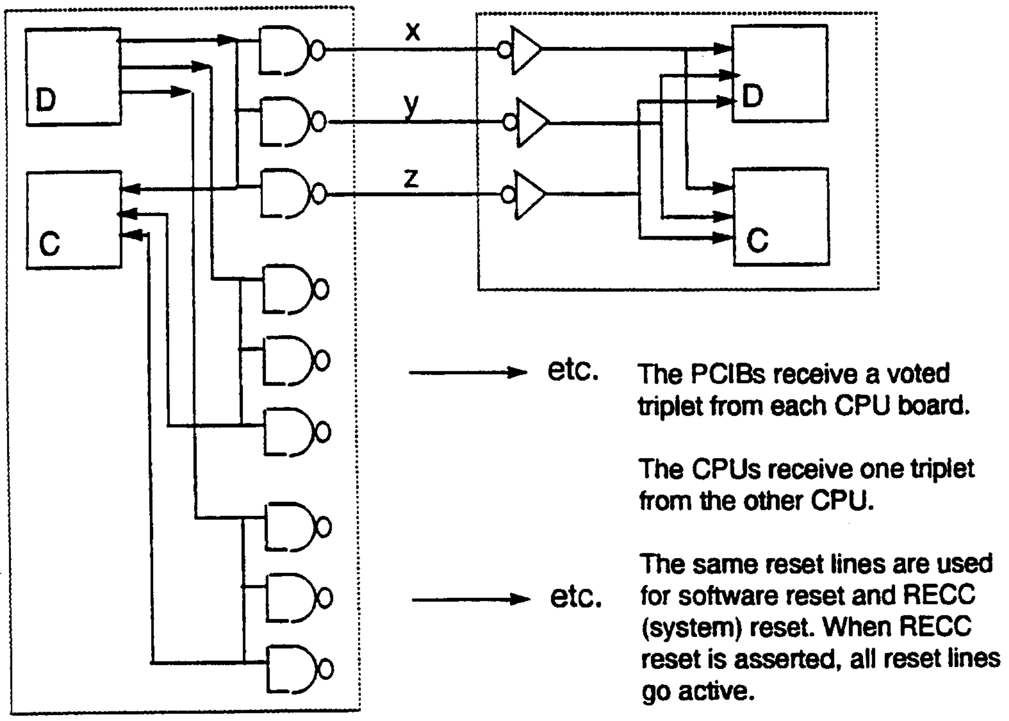


Figure 59. reset_ routing

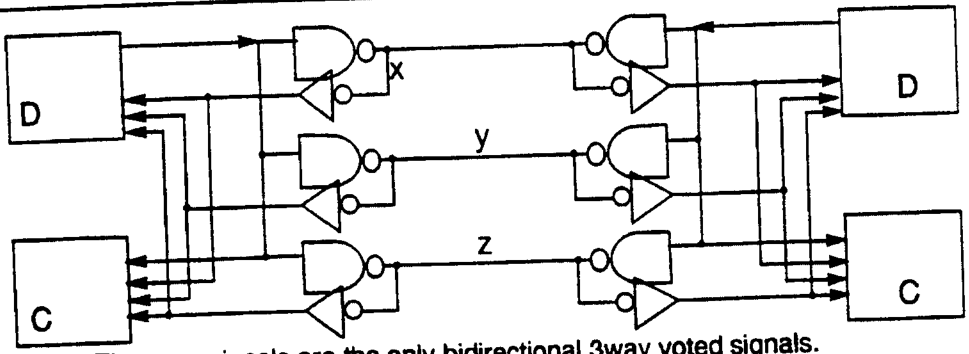


The PCIBs receive a voted triplet from each CPU board.

The CPUs receive one triplet from the other CPU.

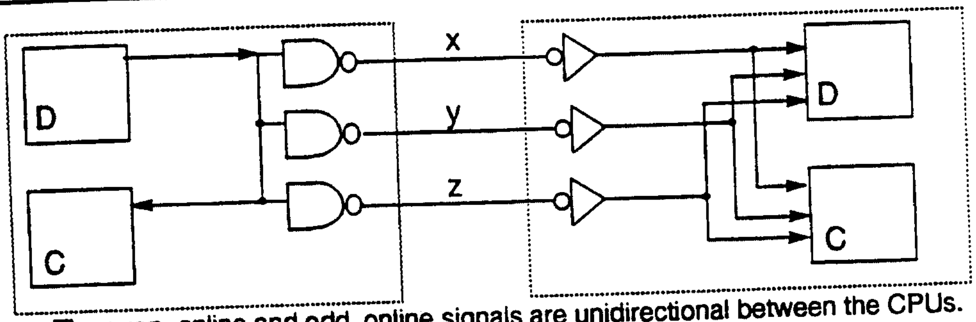
The same reset lines are used for software reset and RECC (system) reset. When RECC reset is asserted, all reset lines go active.

Figure 60. sync_ routing



The sync signals are the only bidirectional 3way voted signals.

Figure 61. online_ routing



The even_online and odd_online signals are unidirectional between the CPUs.

11.3.1 ID PROM Partitioning

Golf/Jetta/Polo systems implement an IDPROM architecture based around a 2Kx8 EEPROM accessed via a JTAG scannable buffer. The Xbus permits ID PROM access under system software control as with the current Golfbus, but also supports:

1) IDPROM access as part of the board level boundary scan chain

2) IDPROM access on an un-powered freestanding board

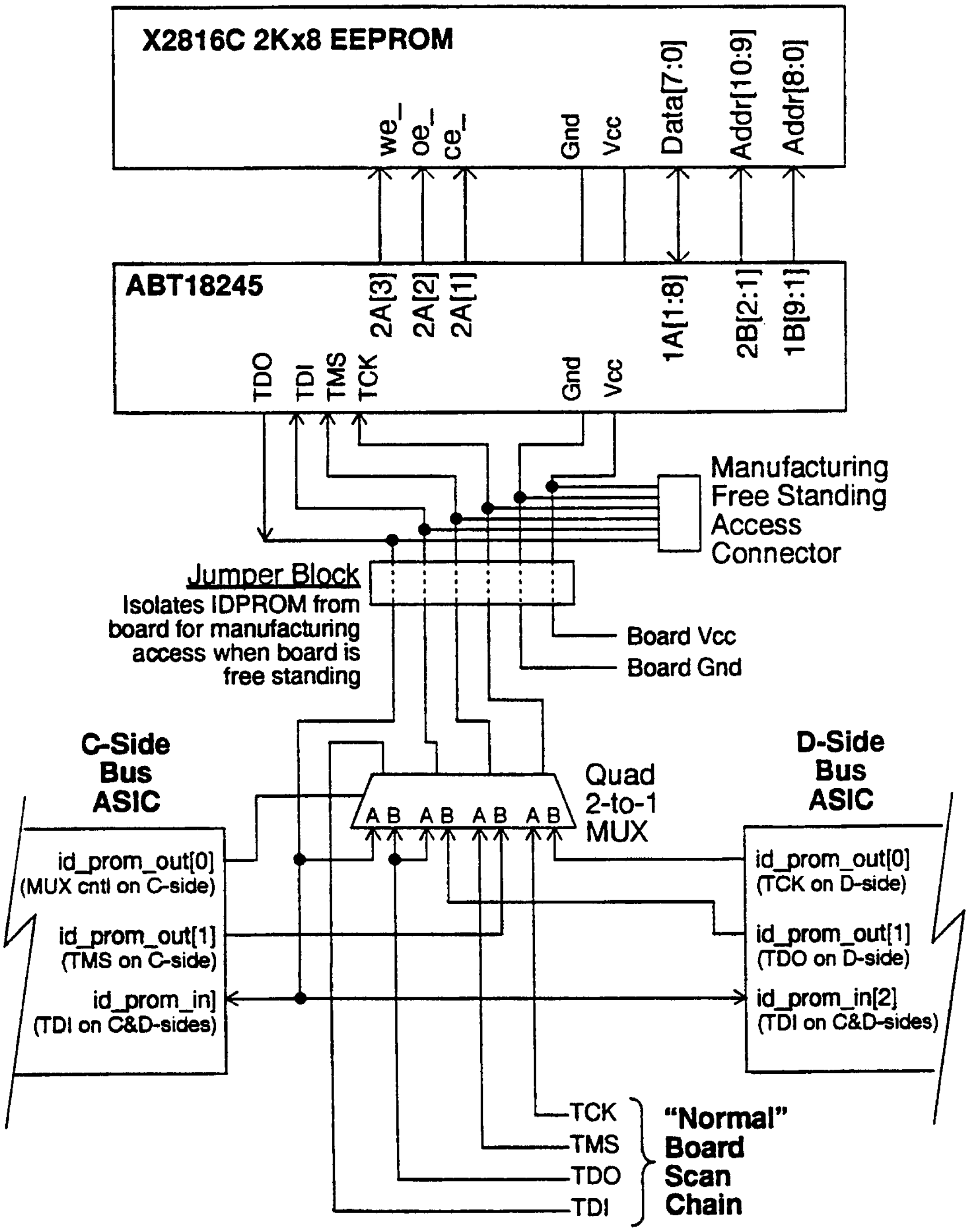
These later two enhancements ease manufacturing's utilization of the IDPROM.

The IDPROM is JTAG 1149.1 accessible. JTAG is an industry standard 4 pin serial bus used for testing the interconnect between components on Polo boards. Initially the test bus will be driven from IBM compatible personal computers equipped with JTAG software/hardware accompanying all Polo debug stations.

A diagram of the ID PROM implementation is provided in figure 62 below. Normally, the illustrated jumper is always present. The jumper block must be removed If manufacturing wants to access the ID PROM without powering up the board (or before the board is tested or fully populated). Once the jumper is removed, the ID PROM can be accessed via the illustrated connector which provides both the JTAG signals and the power and ground to the ID PROM logic.

In normal operation (with the jumper present) the board comes out of the reset state with the IDPROM in the board's scan chain, i.e. the MUX is selecting the A inputs. Once software makes any access to the ID PROM logic through the bus ASICs ID PROM registers, the MUX B inputs are selected, thereby removing the ID PROM logic from the board's normal scan chain and connecting it to the bus ASIC instead.

Figure 62. ID PROM Implementation


X2816C 2Kx8 EEPROM
we_
oe_
ce_
Gnd
Vcc
Data[7:0]
Addr[10:9]
Addr[8:0]
ABT18245
2A[3]
2A[2]
2A[1]
1A[1:8]
2B[2:1]
1B[9:1]
TDO
TDI
TMS
TCK
Gnd
Vcc
Manufacturing Free Standing Access Connector
Jumper Block
Isolates IDPROM from board for manufacturing access when board is free standing
Board Vcc
Board Gnd
C-Side Bus ASIC
id_prom_out[0] (MUX cntl on C-side)
id_prom_out[1] (TMS on C-side)
id_prom_in] (TDI on C&D-sides)
Quad 2-to-1 MUX
A B A B A B A B
D-Side Bus ASIC
id_prom_out[0] (TCK on D-side)
id_prom_out[1] (TDO on D-side)
id_prom_in[2] (TDI on C&D-sides)
TCK
TMS
TDO
TDI
"Normal" Board Scan Chain

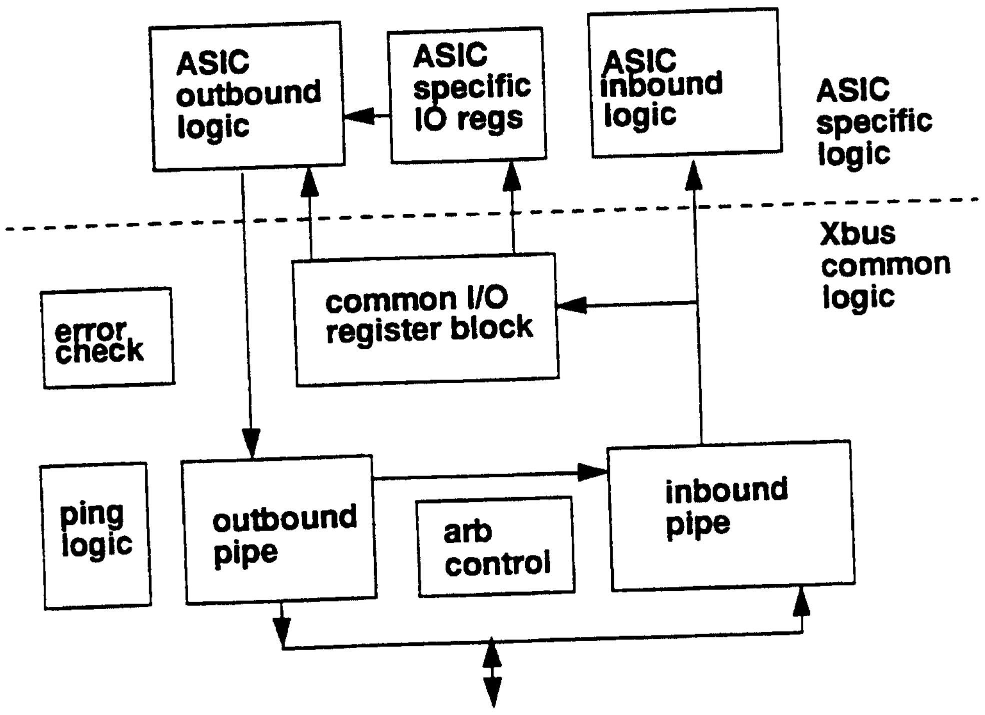
When MUX is in the "A" position, the IDPROM logic is part of the normal board scan chain. When MUX is in the "B" position, the IDPROM logic is removed from the board scan chain and is accessible from the bus ASICs.

Warm or Cold reset place MUX in the A position; any bus ASIC access places it in the B position until the next reset.

12. Xbus Interface Block Diagrams

The Xbus interface portion of the ASICs that are on each board in the Polo system is referred to as the Xbus common logic. A single set of source files is maintained, and the logic is customized by Verilog ifdefs and included into the Gambit and Cyclops designs.

12.1 Xbus Top Level Block Diagram



12.1.1 Port List

The Xbus common logic connects to the ASIC specific logic within the bus ASICs, and to the Xbus side ASIC pins. Also included in the common logic (but not shown above) are miscellaneous support modules: the clock phasor generator, reset generator, DLL, active clock spine, etc.

Port lists for signals entering and leaving the Xbus common logic are discussed in the sections specific to each module.

12.1.2 Functional Description

The Xbus common logic provides all Xbus error checking and board state management (broken, duplexed, on-line, etc.). It contains all control for IO register accesses, although the registers themselves are split between the common logic for bus related registers and the ASIC specific logic for board specific registers. However, the IO register control handshakes with and passes read data through the ASIC specific outbound control unit (OCU); this is because the OCU requirements vary greatly between different board types and is best implemented as ASIC specific logic.

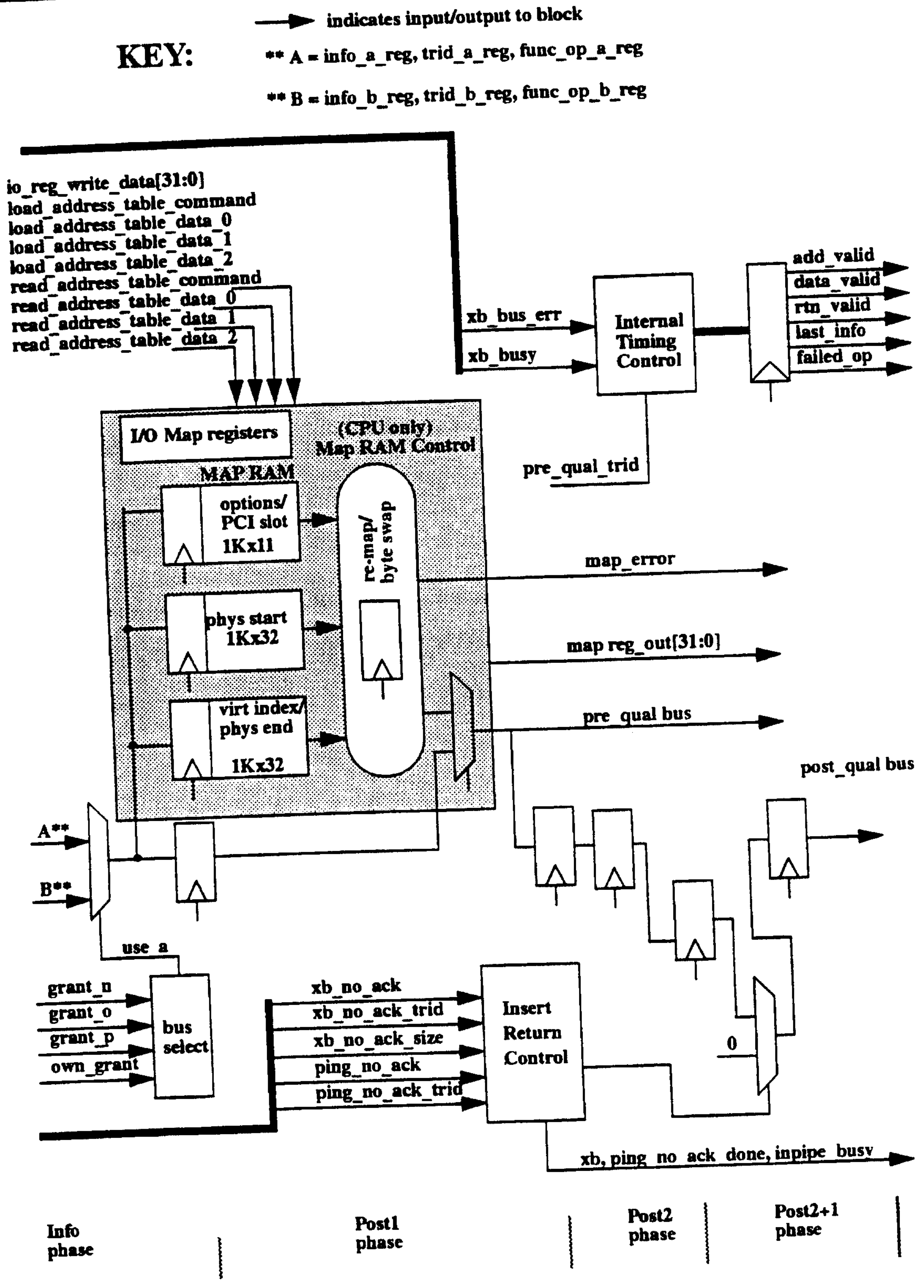
The inbound pipe provides a raw (before bus error and busy are available) version of the Xbus data to the IO registers, ASIC specific inbound pipe, and error logic for ack, busy, and error generation. Two (or more) phases later, it delivers qualified, known good data to the IO registers and ASIC specific inbound pipe for processing. On CPU boards, the inbound pipe also performs the IOVA address re-mapping.

The outbound pipe packages data for transmission on the Xbus. It does not buffer the data before assembly into a bus transaction; rather, it sends mux selects up into the ASIC specific logic as needed to construct the transaction. It stores a copy of the transaction in a recirculation pipe in case the data needs to be transmitted again as a result of a bus error.

12.2 Inbound Pipe Section

The inbound pipe watches the Xbus bus cycles and acknowledges, busies, or errors bus cycles based on information contained within the cycle. The inbound pipe also provides timed versions of the Xbus for use internal to the ASIC. Figure 63 contains a block diagram of the logic contained within the inbound pipe.

Figure 63. Inbound Pipe Block Diagram



12.2.1 Port List

Table 19. Common Logic Inbound Pipe Port List

signal name	width	I/O	description
addr_40_mode	1	I	(Cyclops only). This signal tells the map ram logic to generate 40 bit style virtual address fields on the pre_qual bus (only valid for map RAM IOVA cycles).
board_not_broken_in_n	1	I	Registered not broken signal from neighbor
board_not_broken_in_o	1	I	Registered not broken signal from opposite
bus_oe	1	I	(Gambit only) This signal is used for failed op insert on no-ack cycles and indicates that the current read request was driven by this ASIC.
clk48	1	I	48MHz clock input
cold_reset	1	I	(Cyclops only) cold reset signal - used for resetting the map registers.
err_in_progress	1	I	Signal from error block, true when currently in an error sequence phase.
func_op_a_reg	1	I	Xbus A bus func_op from input registers.
func_op_b_reg	1	I	Xbus B bus func_op from input registers.
ibi_efq_avail	1	I	(Cyclops only) This IBus inbound pipe signal indicates that the EFQ pipe can accept an insert return cycle.
info_a_reg	[31:0]	I	Xbus A bus info from input registers.
info_b_reg	[31:0]	I	Xbus B bus info from input registers.
io_reg_write_data	[31:0]	I	(Cyclops only) From the register control block, the data to be written into a given register on a load_address_xxx signal being asserted.
load_address_table_ command	1	I	(Cyclops only) indicates that the address table command register is being written.
load_address_table_ data	[1:0]	I	(Cyclops only) indicates that the address table data register 2, 1, or 0 is being written.
online	1	I	(Cyclops only) Board online signal from board states logic.
phase_12_0	1	I	12 MHz phase clock 1st half of cycle 0 of phase
phase_12_1	1	I	12 MHz phase clock 2nd half of cycle 0 of phase
phase_12_2	1	I	12 MHz phase clock 1st half of cycle 1 of phase
phase_12_3	1	I	(Cyclops only) 12 MHz phase clock 2nd half of cycle 1 of phase

Table 19. Common Logic Inbound Pipe Port List

signal name	width	I/O	description
phase_24	1	I	24 MHz phase clock
ping_no_ack	1	I	(Cyclops only) This signal from the ping module a PING it sent out on the bus was not acknowledged.
ping_no_ack_trid	[6:4]	I	(Cyclops only) This signal from the outbound pipe indicates the TRID[6:4] of a ping_no_ack cycle.
pu_dside	1	I	(Gambit only) From an ASIC input pin, high = D side ASIC, low = C side
read_address_table_ command	1	I	(Cyclops only) indicates that the address table command register should be placed on the map_reg_out bus for reading
read_address_table_ data	1	I	(Cyclops only) indicates that the address table data register 2, 1, or 0 should be placed on the map_reg_out bus for reading
read_io_map_error	1	I	(Cyclops only) indicates that the I/O map error register should be placed on the map_reg_out bus for reading
rearm_error_regs	1	I	(Cyclops only)
slot_id_latched	1	I	(Gambit only) one bit slot id registered off input pin at reset
trid_a_reg	[6:0]	I	Xbus A bus trid from input registers.
trid_b_reg	[6:0]	I	Xbus B bus trid from input registers.
update	1	I	(Cyclops only) Board update mode signal from board states logic.
warm_reset	1	I	synchronous warm reset to block
xb_bus_err	1	I	Xbus bus error A or B registered - created from three input control buses and one output control bus.
xb_busy	1	I	Xbus bus busy A or B registered - created from three input control buses and one output control bus.
xb_grant_neighbor	1	I	Xbus grant signal from arbitration block, valid one phase prior to ARB phase at 12_3.
xb_grant_opposite	1	I	Xbus grant signal from arbitration block, valid one phase prior to ARB phase at 12_3.
xb_grant_peer	1	I	(Cyclops only) Xbus grant signal from arbitration block, valid one phase prior to ARB phase at 12_3.

Table 19. Common Logic Inbound Pipe Port List

signal name	width	I/O	description
xb_no_ack	1	I	This signal from the outbound pipe indicates a read (I/O or memory) it sent out on the bus was not acknowledged.
xb_no_ack_size	1	I	(Cyclops only) This signal from the outbound pipe indicates the size (high = 256, low = 32) of an xb_no_ack cycle.
xb_no_ack_trid	[6:4]	I	This signal from the outbound pipe indicates the TRID[6:4] of an xb_no_ack cycle.
xb_own_grant	1	I	Xbus grant signal from arbitration block, valid one phase prior to ARB phase at 12_3.
early_xb_p2p_last info	1	O	(Gambit only) This signal indicates that the last xb_p2p_valid is coming.
failed_op	1	O	When true, this signifies this is the return for a transaction that has timed-out (i.e. ping not ACKed) or that a simplexed bus master has 'broken' before completing a transfer involving sub-operations, or that a read issued on the Xbus has not been ACKed. Writes do not cause failed_ops. Note for this condition, the address portion (or first data in the case of a data return) of the transfer completed un-errored and un-busied but the last data has not yet been transferred. This condition cannot occur on bus operations that do not have any sub-operations. If failed_op is true, add_valid, data_valid, return_valid and last_info may also be true. These signals will emulate the completion of the sub-operations. failed_op will remain true until all the sub-operations associated with the transaction have been emulated. When failed_op is true, The post_qual_bus is guaranteed to be all 0's. This signal is clocked by clk48 and is qualified by phase_12_0.
illegal_iova	1	O	(Cyclops only) This signal indicates that the current pre_qual bus contains a cycle with an IOVA fault and therefor should be ignored).
inpipe_busy	1	O	This signal indicates that the current Xbus cycle should be busied because the inpipe is busy servicing an insert return.
insert_return_need_efq_ post2	1	O	This signal indicates that we are currently in post 2 of an insert return and the return will start with the post_qual bus in post 3.

Table 19. Common Logic Inbound Pipe Port List

signal name	width	I/O	description
map_error	1	O	(Cyclops only) This signal indicates that a map look-up error occurred on an IOVA Xbus access. This signal is valid in the first half of Post2 and is sent to the common register logic where it will be stored in the ASIC specific broken status register. This signal is also used internal to the block to inhibit acknowledgment of the cycle. Map_errors do not cause bus errors.
map_reg_out	[31:0]	O	(Cyclops only) This 32 bit bus contains the value from the last map register internal to this block that was read.
ping_no_ack_done	1	O	(Cyclops only) This signal indicates that the insert return resulting from a ping_no_ack has completed and the trid no longer needs to be held valid.
post_qual_func_op	1	O	This is the post_qual bus func_op.
post_qual_hit_status	1	O	(Cyclops only) This signal is valid when an info cycle is on the post_qual bus and is equal to the hit status found in the TRID.
post_qual_info	[31:0]	O	This is a post2+1 (i.e. one phase after post2) version of the info lines on the bus in use. A valid address is present when add_valid is asserted, and valid data when data_valid is asserted. This information has been qualified with busy and error, and is fit to be passed deeper into the system.
post_qual_return_256_32	1	O	(Cyclops only) This signal is valid when a data return is on the post_qual bus and indicates if the return length is 256 (high) or 32 bits (low).
post_qual_trid	[6:0]	O	This is the post_qual bus trid.
pre_qual_ben	[31:0]	O	This is a post1 version of the byte enable lines on the bus in use. Information on these lines should be used for generating busy and ack, and then discarded. The byte enables are valid during the second half of post 1.
pre_qual_func_op	1	O	This is the pre_qual bus func_op.
pre_qual_func	[31:0]	O	This is a post1 version of the function code lines on the bus in use. Information on these lines should be used for generating busy and ack, and then discarded. The function code is valid during the second half of post 1.
pre_qual_info	[31:0]	O	This is a post1 version of the info lines on the bus in use. Information on these lines should be used for generating busy and ack, and then discarded.

Table 19. Common Logic Inbound Pipe Port List

signal name	width	I/O	description
pre_qual_ping	1	O	(Cyclops only) This signal is valid when a ping is on the post_qual bus.
pre_qual_rc	[31:0]	O	This is a post1 version of the remote/coherent lines on the bus in use. Information on these lines should be used for generating busy and ack, and then discarded. The r/c is valid during the second half of post 1.
pre_qual_swap	1	O	(Cyclops only) This signal indicates that the pre_qual bus contains a cycle which should be loaded into the swap table if accepted.
pre_qual_trid	1	O	This is the pre_qual bus trid.
xb_add_valid	1	O	When true, this signifies there is a valid address phase on the post_qual_bus. This is a func_op phase and it is not an idle phase. It has not been busied or errored by anyone on the bus. It does not mean the address is necessarily to my space. This signal is clocked by clk48 and is qualified by phase_12_2.
xb_data_valid	1	O	When true, this signifies that there is valid data on the post_qual_bus. This data is associated with a previous add_valid and has not been errored by anyone on the bus. This signal is not true during data returns. This signal is clocked by clk48 and is qualified by phase_12_2.
xb_last_info	1	O	When true, this signifies that this is the last phase of information associated with this bus operation (It may be the only information needed). This signal is valid for a signal phase. If this is the first piece of information, it has not been busied or errored by anyone on the bus. If this is not the first piece of information, it has not been errored by anyone on the bus. Note, xb_add_valid, xb_data_valid or xb_return_valid can be true while last_info is true. This signal is clocked by clk48 and is qualified by phase_12_2.
xb_next_add_valid	1	O	This is the xb_add_valid signal prior to being registered at 12_2
xb_next_data_valid	1	O	This is the xb_data_valid signal prior to being registered at 12_2
xb_next_return_valid	1	O	This is the xb_return_valid signal prior to being registered at 12_2

Table 19. Common Logic Inbound Pipe Port List

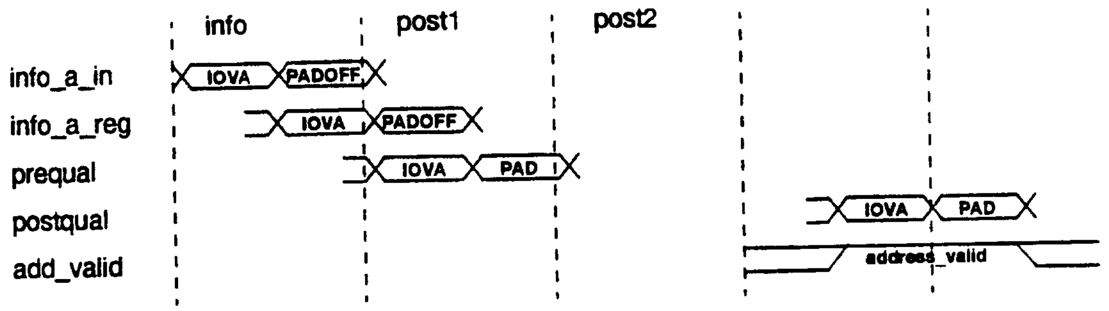
signal name	width	I/O	description
xb_no_ack_done	1	O	This signal indicates that the insert return resulting from a xb_no_ack has completed and the trid no longer needs to be held valid.
xb_p2p_func_op	1	O	(Gambit only) The peer-to-peer send post qual func_op bus sent to the XBO.
xb_p2p_incomplete_send	1	O	(Gambit only) This signal indicates that the p2p send in progress has been dropped and should be discarded.
xb_p2p_info	1	O	(Gambit only) The peer-to-peer send post qual info bus sent to the XBO.
xb_p2p_last info	1	O	(Gambit only) This signal indicates that this is the last xb_p2p_valid and the p2p echo can begin.
xb_p2p_trid	1	O	(Gambit only) The peer-to-peer send post qual trid bus sent to the XBO.
xb_p2p_valid	1	O	(Gambit only) The peer-to-peer send valid sent to the XBO. This signal indicates that the xb_p2p_info, trid, and func_op are valid
xb_return_valid	1	O	When true, this signifies that there is a valid data return on the post_qual_bus. The lower 4 bits of the trid matches this slot address (and duplex/ simplexed state). This signal also indicates that the data has not been busied or errored on the bus. This signal is clocked by clk48 and is qualified by phase_12_2.

12.2.2 Functional Description

The Xbus inbound pipe handles all issues relating to the inbound Xbus. This block generates bus error and the acknowledge (with some help from board specific logic). It also generates busy with input from four different sources: the I/O register block, the ping logic, the board specific logic, and the map RAM logic internal to this block.

The info and trid buses are registered and made available to the rest of the logic on the post_qual and pre_qual buses. The post and pre_qual trid, func_op, and info buses are simply referred to as the post_qual or pre_qual bus in figure 63 in order increase readability. The I/O address map resides within this block so that IOVA Xbus addresses can be converted to system addresses before putting them on the pre-qual bus. A map error of any sort will inhibit the bus acknowledge and assert the signal map_error. The timing of the post and pre-qual buses relative to and Xbus cycle is shown in figure 64.

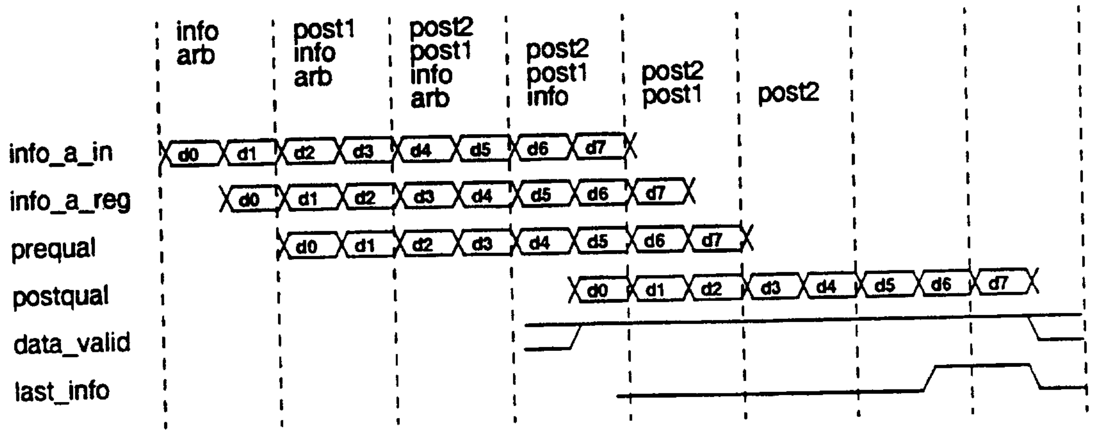
Figure 64. Post-Qual and Pre-Qual Bus Timing



IOVA - I/O Virtual address
PAD - Physical address

The valid signals indicate when the various buses can be used for address or data. Figure 65 illustrates the timing of data valid and last_info for a 256 bit transfer on the Xbus. The timing for return valid is identical to the timing for data_valid.

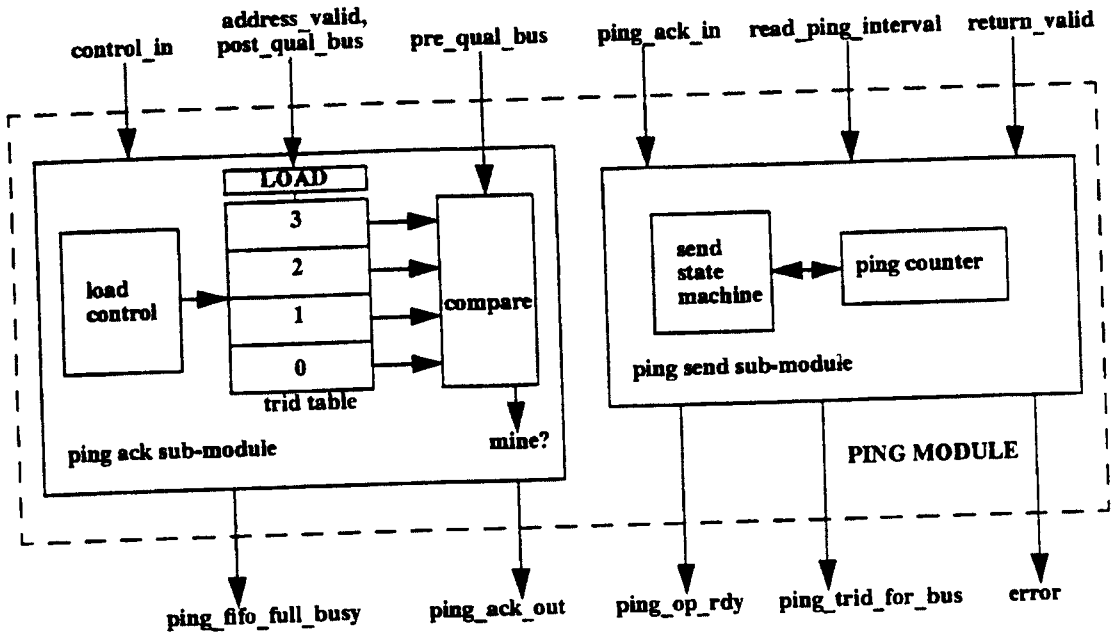
Figure 65. data_valid and last_info Timing



The page 2 SAM registers (IAM ASIC and Cyclops ASIC Map RAM Registers) are also located in this section within the Map RAM Control block (CPU only). These are the registers that control the I/O address table (i.e. Address Table Command, Address Table Data 2,1, and 0 registers). These registers can be written directly from the post_qual bus or read with the data being placed on the map_reg_out bus two bus phases after Post2 and kept there until the next read. It is also possible to read and write the map RAMs via these registers with that control also placed in the map RAM control block. Refer to section 12.6.2 on page 118 in the Polo Programming Guide for details on the registers internal to this block.

12.3 PING Section

Figure 66. PING Control Block Diagram



The PING block controls generation of PINGs for I/O reads that are waiting for responses and acknowledgment of PINGs coming from other boards.

12.3.1 Port List

Table 20. PING Logic Port List

signal name	width	I/O	description
addr_valid	1	I	output from inbound pipe, used to generate ack for incoming PINGs. Indicates that the post_qual bus has a valid address cycle present. Tied to xb_add_valid of the XBI.
clk48	1	I	48MHz input clock
inc_invalidate	1	I	(Gambit only) Indication to invalidate a ping table fifo entry.
inc_trid_to_invalidate	1	I	(Gambit only) Trid of the entry that is to be invalidated via inc_invalidate.
inc_nodrv_npaired_rd, common_nodrv_ npaired_rd	1 1	I	(Cyclops only) These signals come from the inbound pipe and I/O register blocks and are loaded into the ping table with each outstanding IO request. When true they indicate the current I/O read being loaded into the table is a non-paired request to partner so this ASIC should go through the motions but should not drive the Xbus.
inc_take_cycle ioreg_take_cycle	1 1	I	These signals come form the inbound pipe and I/O register blocks and indicate to the ping logic to load the transaction on the post_qual bus into the ping table
ioreg_cd_different	1	I	This comes from the I/O register block and indicates that the I/O register read that is being placed in the trid table is CD different.
ioreg_read_ping_ interval	1	I	(Cyclops only) This signal comes from the common register block (bit 0 of the read ping interval register). Lo - ping after 100 usec; Hi - ping after 2.667 usec.
online	1	I	(Cyclops only) Board online signal from board states logic.
otherside_ping_trid_ matchi	1	I	(Gambit only) Indicates PING TRID matches other-side ASIC
phase_12_0	1	I	12 MHz phase clock for 1st half of cycle 0 of phase
phase_12_1	1	I	12 MHz phase clock for 2nd half of cycle 0 of phase
phase_12_2	1	I	12 MHz phase clock for 1st half of cycle 2 of phase
phase_12_3	1	I	12 MHz phase clock for 2nd half of cycle 2 of phase
ping_no_ack_done	1	I	(Cyclops only) Indication from inbound pipe that no ack processing is complete (insert return finished)
post_qual_bus	[11:6]	I	The post_qual (after Xbus response) info bus from the XBI.

Table 20. PING Logic Port List

signal name	width	I/O	description
post_qual_return_valid	1	I	(Cyclops only) Output from inbound pipe, used to clear entries from the PING table. Indicates that the post_qual bus has a valid data return cycle present. Tied to xb_return_valid of the XBI.
post_qual_trid	[6:0]	I	The post_qual TRID bus from the XBI.
pre_ping	1	I	The pre_qual PING indication from the XBI.
pre_trid	[6:0]	I	The pre_qual TRID bus from the XBI.
warm_reset	1	I	synchronous warm reset to block
xbus_ack	1	I	(Cyclops only) This is the Xbus acknowledge signal used to determine whether the PING was ACKed
xbus_op_done_ack	1	I	(Cyclops only) This is the Xbus op done from the XBO. It is used to push outgoing I/O reads onto the PING FIFO.
xbus_op_done	1	I	This signal comes form the outbound pipe and indicates the final piece of information went across the bus without a busy.
xbus_op_done_trid	[6:0]	I	This signal comes form the outbound pipe and indicates the trid of the completed transaction
xbus_op_done_func	[5:0]	I	This signal comes form the outbound pipe and indicates the function of the completed transaction
xbus_p2p_issue	1	I	(Cyclops only) This signal comes form the outbound pipe and indicates that the I/O read being issued is peer-to-peer so any PINGs will have to be also.
xbus_trid_for_bus	[6:0]	I	This signal comes form the OCU and indicates the trid of the transaction in progress.
nodrv_npaired_rd_out	1	O	This signal comes from the trid table fifo and indicates that the current read return is a non-paired read. This is to be used by the checking logic as well as the arbitration.
cd_different_out	1	O	This signal comes from the trid table FIFO and indicates that the current read return that we are performing is CD different and the checking logic needs to be aware of that.
error_no_ack	1	O	This signal indicates that we had a no-ACKed ping
error_trid	[6:0]	O	This is the trid of the no-ACKed ping
myside_ping_trid_ matcho	.1	O	(Gambit only) Indicates PING TRID matches this ASIC
ping_ack_out	1	O	sent to the ack logic, indicates ping on bus should be acknowledged, valid 12_0 of post2.

Table 20. PING Logic Port List

signal name	width	I/O	description
ping_fifo_full_busy	1	O	This signal goes to the busy block in the inbound pipe and indicates that this cycle should be busied.
ping_op_ready	1	O	This signal indicates to the OCU that there is a ping ready to go.
ping_trid_for_bus	[6:0]	O	Trid to be used for ping cycle - to the OCU
ping_p2p_issue_for_bus	1	O	This signal is sent to the OCU as an indication to whether the ping is a peer-to-peer transaction

12.3.2 Functional Description

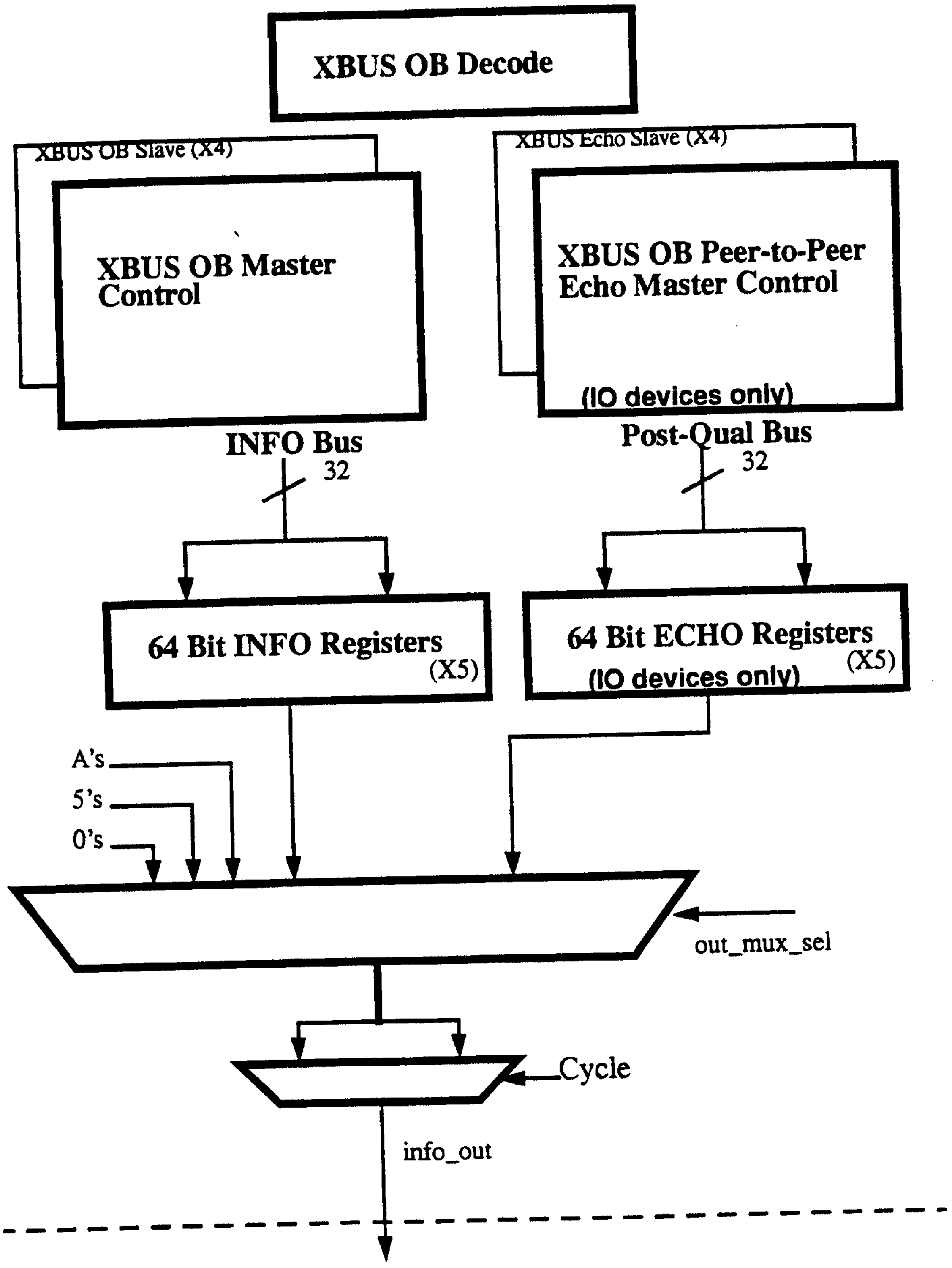
This module has two functional pieces: the send sub-module and the ack sub-module.

The send sub-module keeps track of the outbound read requests and their trids. This block also pings I/O read requests that have timed-out. The ping frequency is determined by the input read_ping interval from the common register block. During normal operation the interval will be 100 usec, however the ping interval can be set to 2.667 usec for testing purposes. The send module has an internal ping count used to keep track of when a cycle should be pinged.

The ack sub-module tracks incoming read requests and their status. It also acknowledges any incoming pings that hit a trid corresponding to an active read response being worked on. A trid table in the ping ack module keeps track of information regarding a read cycle that has been accepted by this ASIC. It maintains this information for acknowledging pings and for use during the read return. Information specific to the read return such as the trid and whether a given cycle is CD different is stored in the trid table. The table can hold up to seven entries; once it is full it sends out the signal ping_fifo_full_busy, indicating that the current transactions to be busied on the Xbus.

12.4 Xbus Outbound Data and Control Unit

Figure 67. Outbound Data and Control Unit



12.4.1 Port List

Table 21 describes the port list to the outbound data path.:

Table 21. Common Logic Outbound Pipe Port List

signal name	width	I/O	description
go_arb	1	O	This signal is driven from the master state machine to the arbitration logic. It is used to initiate an arbitration cycle for the Xbus/
xbus_op_done	1	O	This signal is passed to the OCU. It tells the OCU that the last operation has been sent across the bus (without being busied) and that the information may be overwritten. This signal is qualified with phase_12_1, and thus may clocked by the OCU on phase_12_2.
drv_grnt_inh	1	O	This signal indicates to drive Xbus grant inhibit control signal.
lower_info_sel	[3:0]	O	These lines choose from the 10 possible pieces of information that could be transferred across the lower 32 bits of the Xbus information bus. The encodings are as follows: 1000 - Virtual address (with function code) 1001 - Physical Address 0000 - data word 00 0001 - data word 04 0010 - data word 08 0011 - data word 0c 0100 - data word 10 0101 - data word 14 0110 - data word 18 0111 - data word 1c
drv_xbus	1	O	This signal indicates to xb_err_block to drive the Xbus.
info_out	[31:0]	O	This signal is the info output. It is driven from the data out mux to the transceivers.
trid_out	[7:0]	O	This signal is the trid output as well as the func_op. It is driven from the data out mux to the transceivers.
xb_no_ack	1	O	This signal is driven from the master state machine to the in-pipe for failed_op insertion.
xb_no_ack_trid	[2:0]	O	This signal is driven from the master state machine to the in-pipe for failed_op insertion. These three bits correlate to bits [6:4] of the 1st cycle trid.

Table 21. Common Logic Outbound Pipe Port List

signal name	width	I/O	description
xb_no_ack_size	1	O	This signal is driven from the master state machine to the in-pipe for failed_op insertion. 0 - 32 bit operation 1 - 256 bit operation
retrans_op	1	O	Indication to the OCU that transaction has been busied by the bus and offers an opportunity for switching to a higher priority queue (EFQ)
err_in_progress	1	O	From error block indicating that an Xbus error sequence is in progress.
xact_single_side	1	O	This signal is driven from the master state machine to the error checking logic to signal that loop back checking should be performed on the complete bus width of the current cycle. This signal is used in conjunction with drv_xbus by the error block to determine when and how to do loopback checking
xbus_op_done_ack	1	O	Indication to Ping module that the transaction has completed with an acknowledge
xbus_op_done_trid	1	O	Indication to Ping module of the transaction id for the completed transaction.
xbus_op_done_func	1	O	Indication to Ping module of the func_op for the completed transaction.
op_ready	1	O	This signal is passed to the master control logic. It indicates that all the information needed for a bus transaction is available and ready to go out to the Xbus. Sync. to 12_0 or 12_1 for one phase.
newest_cd_read_return	1	O	CD read return indication qual'ed on a transaction basis
cd_different_out	1	I	CD read return indication from Ping module
newest_nodrv_rd_return	1	I	Indication from Ping module that transaction is a non-paired read return.
read_return_32	1	I	Indication from OCU that the subsequent transaction will be a rd_ret_32.
single_side	1	I	This signal from the an IO device's OCU, when asserted indicates that the current transaction is a single-sided transaction.
bus_oe	1	I	This signal from the an IO device's OCU is used to gate the assertion of drv_xbus for single sided accesses.
info_for_bus	[31:0]	I	From OCU to outbound new data. This is the info bus for the next transaction.

Table 21. Common Logic Outbound Pipe Port List

signal name	width	I/O	description
trid_for_bus	[6:0]	I	From OCU to the outbound new data. This is the trid for the next transaction.
xb_own_grant	1	I	From xbus arb to master state machine indicating that this node has been granted the Xbus.
grant_peer	1	I	From xbus arb to master state machine indicating that this node's peer has been granted the Xbus.
xbus_err	1	I	registered clean version of bus error, used by master state machine.
xbus_ack	1	I	registered clean version of bus_ack, used by master state machine.
xbus_busy	1	I	registered clean version of bus busy, used by master state machine.
xb_err_drv	[1:0]	I	From error block, indicates when and what to drive during error sequence. 00 - Normal operation 01 - Drive 0's 10 - Drive 5's,A's 11 - Drive 0's
no_ack_done	1	I	Indication from in-pipe that failed-op insertion has been completed.
p2p_mode_enable	1	I	From register module indiacating the status of p2p_mode.
ping_p2p_issue_for_bus	1	I	From module, indication whether ping transaction is to be peer-to-peer'ed.
arb_for_cpu_online	1	I	From board states module, arb for 1st cpu online.
cpu_online_grnt_inh	1	I	From board states module, issue grant inhibit if won 1st cpu online.
xb_p2p_info	[31:0]	I	Piped version of the info on the Xbus, needed for Peer-to-Peer echo.
xb_p2p_trid	[6:0]	I	Piped version of the trid on the Xbus, needed for Peer-to-Peer echo.
xb_p2p_func_op	1	I	Piped version of the func_op on the Xbus, needed for Peer-to-Peer echo.
xb_p2p_last_info	1	I	Indication from in-pipe that the last phase of the transaction is currently posted on the peer-to-peer post_qual bus.
xb_p2p_incomplete_send	1	I	Indication from in-pipe that the current peer-to-peer operation will not complete.

Table 21. Common Logic Outbound Pipe Port List

signal name	width	I/O	description
xb_p2p_valid	1	I	Indication from in-pipe that the information on the p2p_post_qual bus is valid with either an info, data or a return cycle.
xb_add_valid	1	I	Indication from in-pipe that the information on the post_qual bus is valid with either an address/trid
swap_table_load	1	I	Indication from in-pipe that an IOVA map RAM lookup has yielded a byte swap hit on a read.
post_qual_trid_4 post_qual_trid_0	1 1	I I	Delayed version of trid from Xbus used for swap_table load. Bit 4 determines C/D side, bit 0 determines PCIB0 or PCIB1
update	1	I	From board states module, update state.
duplexed	1	I	This signal indicates whether or not the board is duplexed.
hit_miss	1	I	From OCU, to be appended into bit 5 of second half of trid.
p2p_issue	1	I	From OCU, to be appended into bit 4 of second half of trid.

12.4.2 Functional Description

The Xbus outbound data and control unit issues all outbound transactions onto the Xbus including the echo of Peer-to-Peer transactions. The XBO will load all of the necessary information from the OCU upon the assertion of op_ready by the OCU. The XBO then issues the Xbus transaction onto the Xbus following the Xbus protocol. The XBO is comprised of the following functional blocks as illustrated in figure 67:

- XBO Decode - This block decodes the function supplied by the OCU.
- INFO Registers - Five registers, each holding one info phase worth of data.
- XBO Control - Five control sequencers, one master and four slaves. Each sequencer controls one info cycle of the transaction.
- ECHO Registers - Five registers, each holding one echo phase worth of data for Peer-to-Peer echo.
- XBO Echo Control - Five control sequencers, one master and four slaves. Each sequencer controls one info cycle of the Peer-to-Peer echo transaction.
- Outbound Mux - This multiplexer channels the appropriate data onto the info_out bus which is ultimately sent onto the Xbus.
- Swap Table - A four entry table, one entry for each PCI, which stores IOVA read byte swap information for outstanding read requests. This table is loaded following an IOVA Map RAM lookup for a read request. The XBO uses this information when issuing the data return.

Figure 68 illustrates an example of XBO timing and its handshake with the OCU block. This particular illustration is that of a 128 bit write. Figure 69 and figure 70 illustrate the control flows for the master and slave sequencers, respectively. The master sequencer controls the first info phase to be posted on the Xbus, each of the four slave sequencers control the other possible Xbus info cycles

Figure 71 illustrates the issuing of a peer-to-peer transaction onto the Xbus. Note that the issuing controller remains involved in the transaction until the transaction is echoed by its neighbor. The issuing controller checks for error on the info cycles that it transmits but it does not check for busy since it is only valid on the echo. Therefore the issuing controller checks for busy on the echo portion of the transaction and retransmits the complete transaction in that occurrence.

Figure 72 illustrates the echo peer-to-peer timing. The XBO loads the post-qual info into its echo info registers when it receives a peer-to-peer indication in the post_qual_trid. It then transmits the received data onto the Xbus, checking for bus error and driving gmt_inh. In the case of an error during the echo, it retransmits the info. Figure 73 illustrates the echo master and slave control sequencers. Similar to the outbound sequencers, the master controls the first info phase and the slaves each control the other possible info phases of the echo.

Figure 68. Outbound Timing Diagram

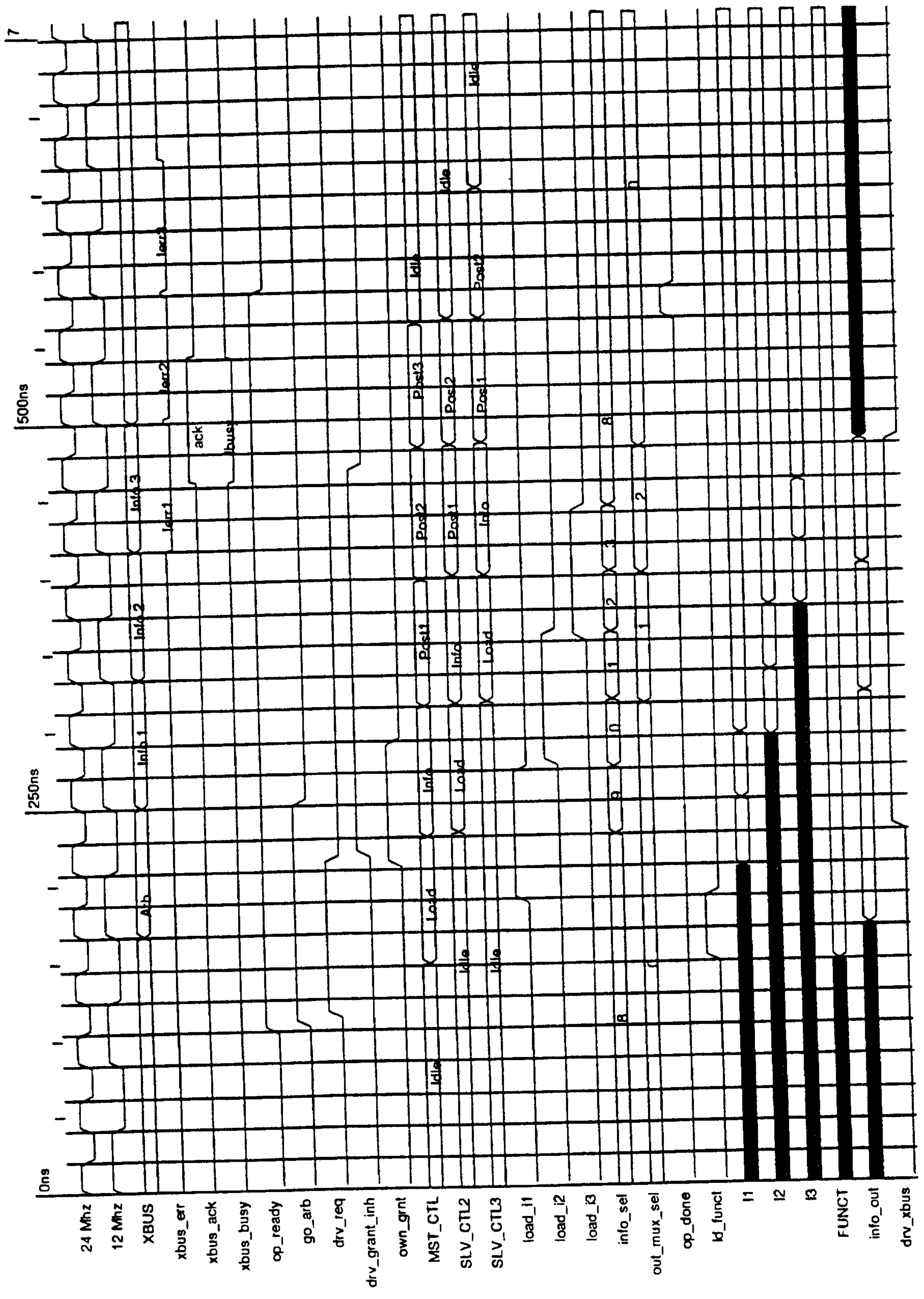
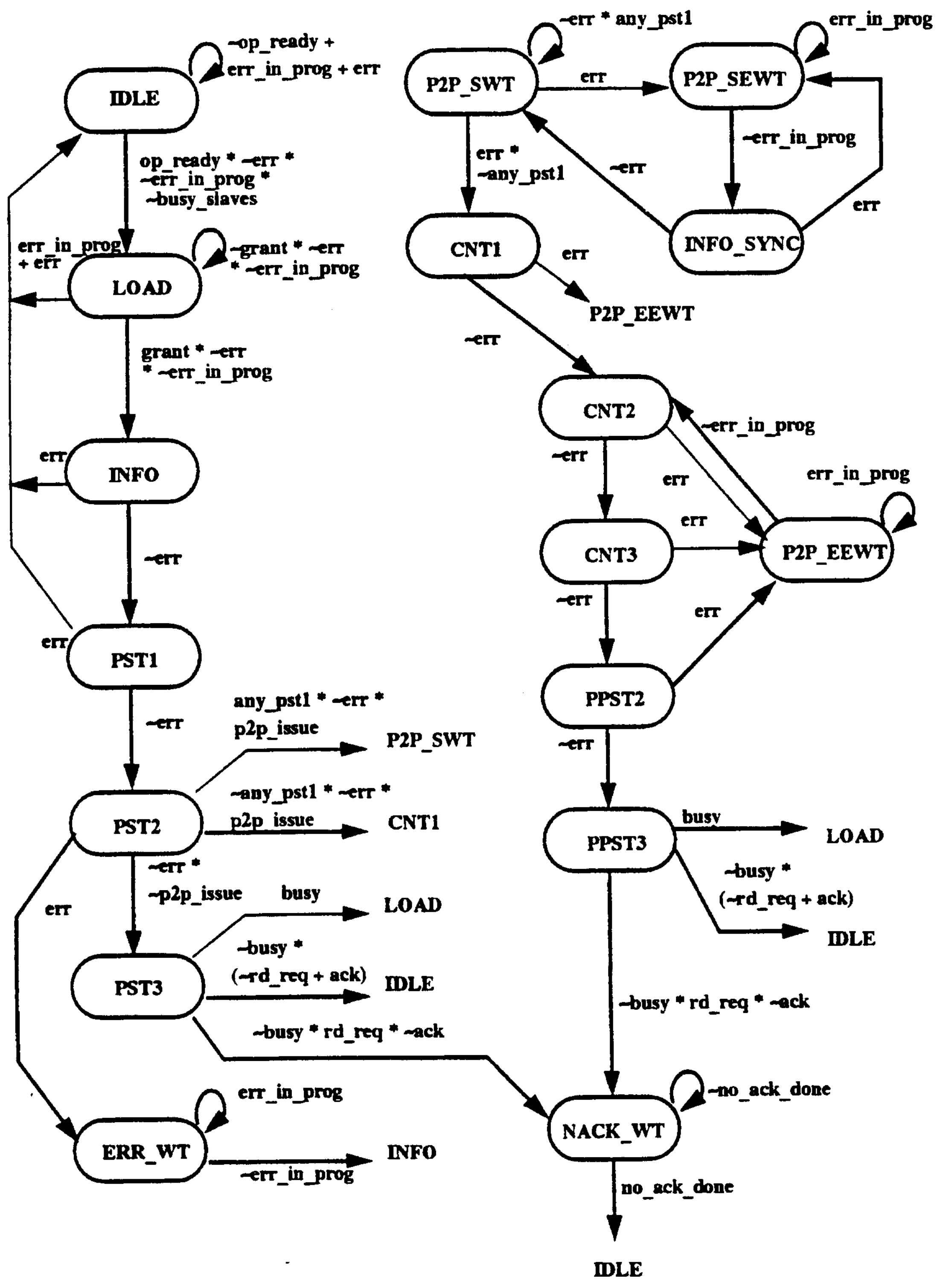
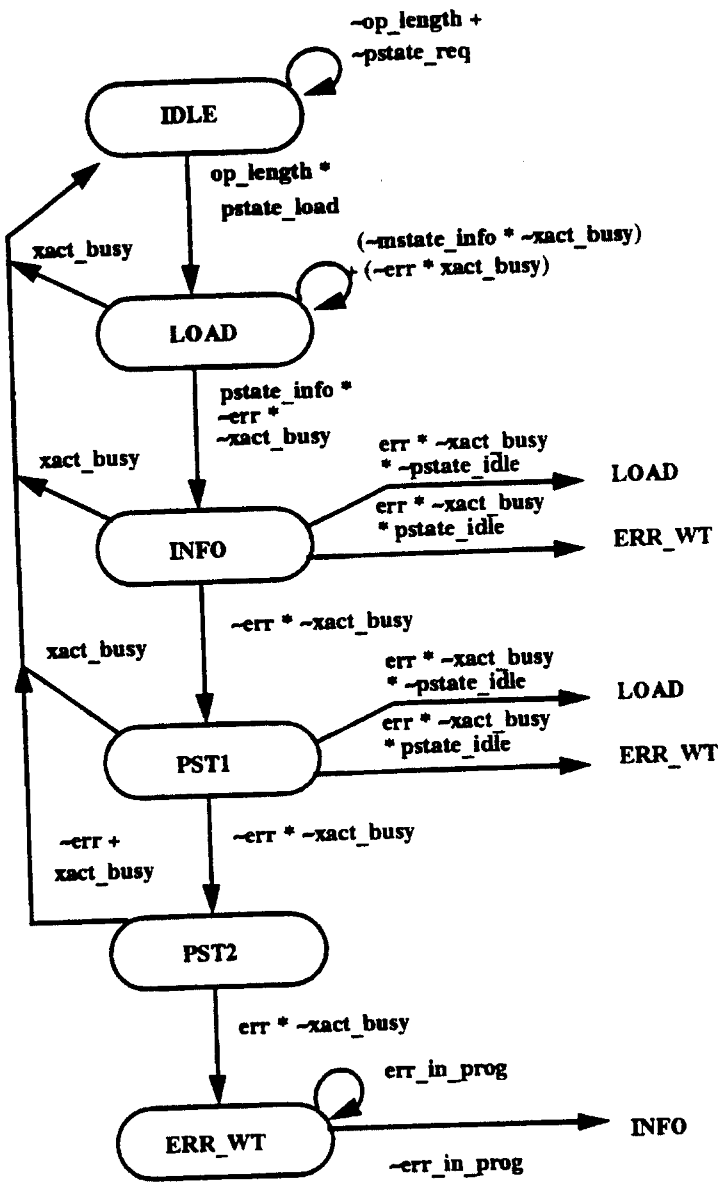


Figure 69. Outbound Master Control



MASTER CONTROL

Figure 70. Outbound Slave Control



SLAVE CONTROL

Figure 71. Outbound Peer-to-Peer Timing

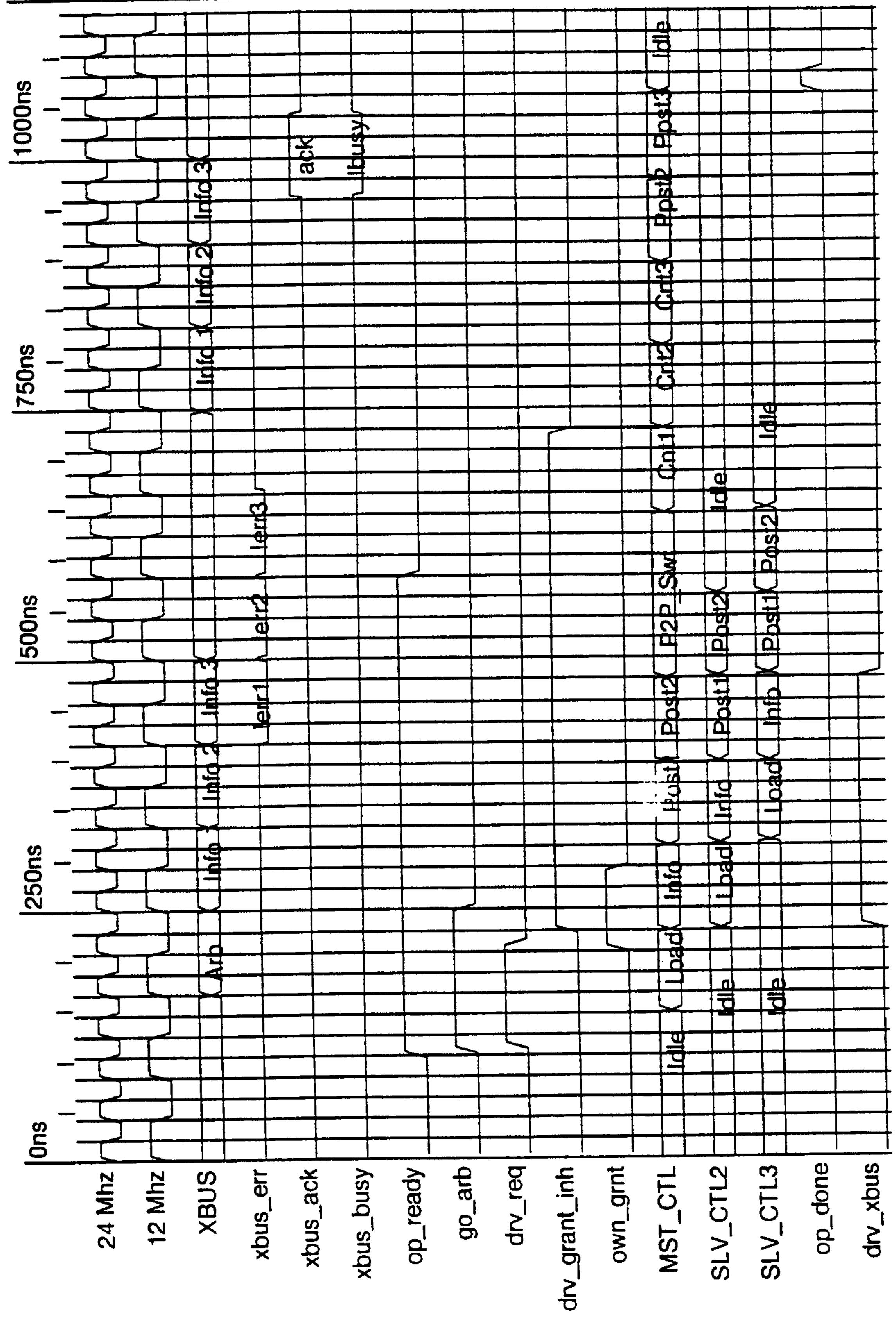


Figure 72. Echo Peer-to-Peer Timing

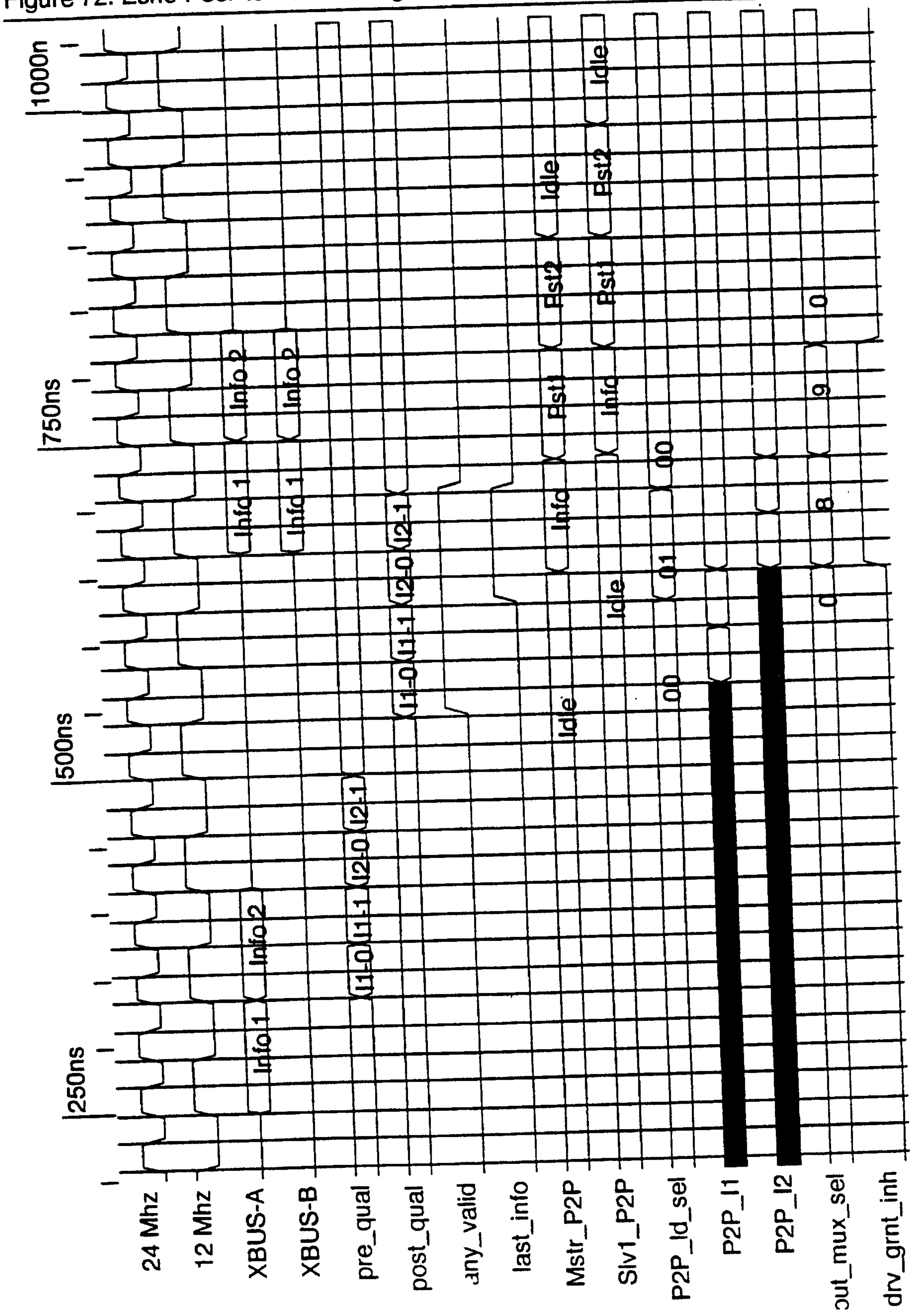
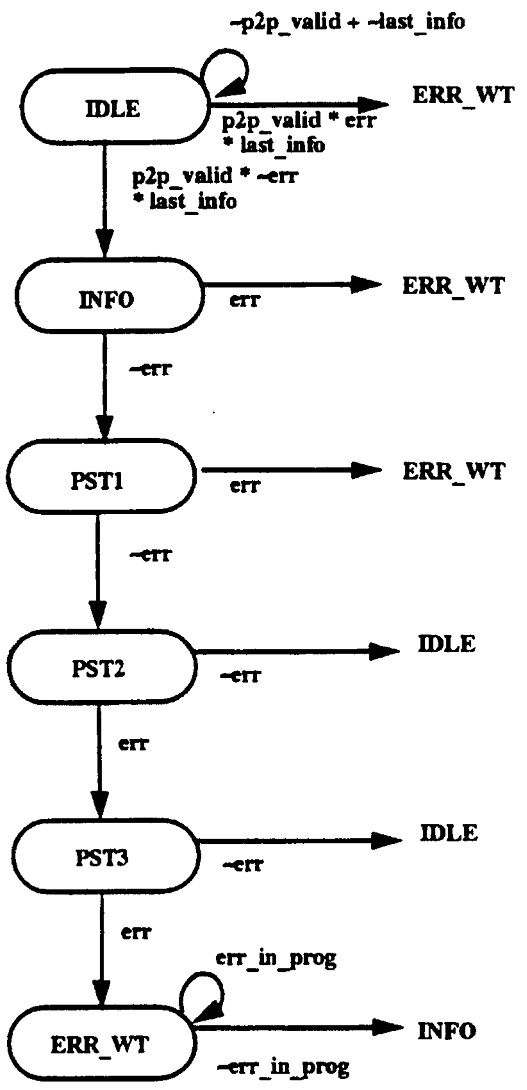
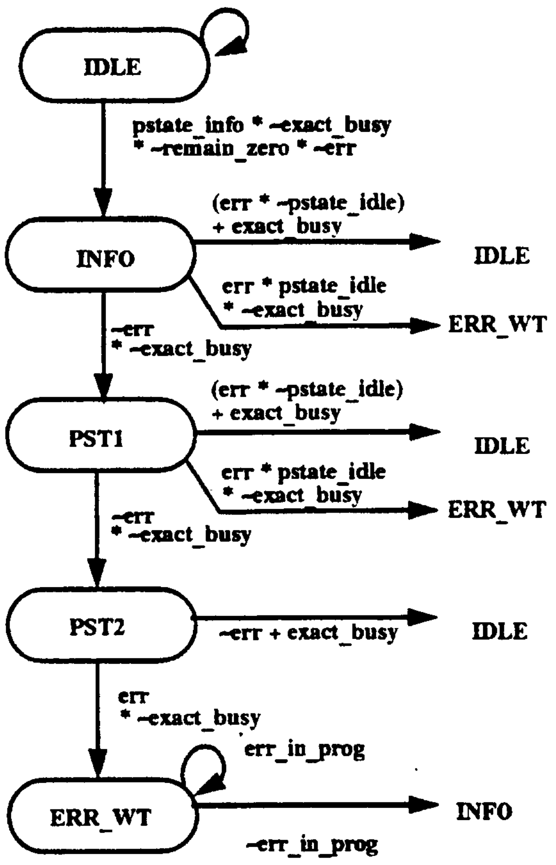


Figure 73. Outbound Peer-to-Peer Control



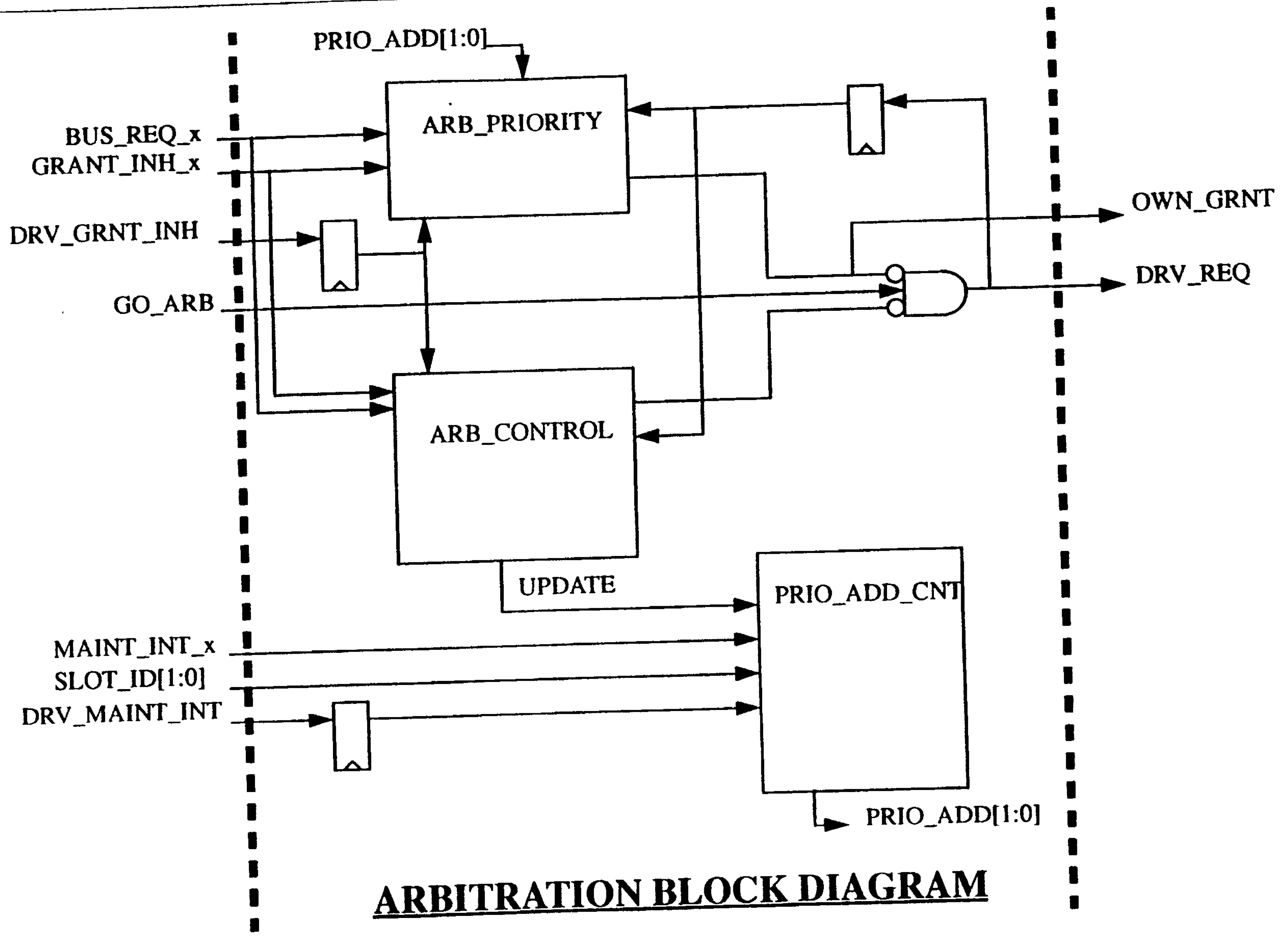
P2P ECHO
MASTER CONTROL


IDLE
pstate_info * ~exact_busy * ~remain_zero * ~err
(err * ~pstate_idle) + exact_busy
INFO
IDLE
err * pstate_idle * ~exact_busy
ERR_WT
~err * ~exact_busy
(err * ~pstate_idle) + exact_busy
PST1
IDLE
err * pstate_idle * ~exact_busy
ERR_WT
~err * ~exact_busy
PST2
~err + exact_busy
IDLE
err * ~exact_busy
err_in_prog
ERR_WT
INFO
~err_in_prog

P2P ECHO
SLAVE CONTROL

12.5 Arbitration Logic

Figure 74. Arbitration Block Diagram



317

12.5.1 Port List

Table 22. Arbitration Functional Block Port List

signal name	width	I/O	description
xb_req_p	1	I	Xbus request control input bus from peer.
xb_req_n	1	I	Xbus request control input bus from neighbor.
xb_req_o	1	I	Xbus request control input bus from opposite.
xb_maint_int	1	I	Xbus maintenance interrupt control input bus
drv_maint_int	1	I	This signal is an input from the error block indicating that the maintenance interrupt control line will be driven in following clock phase.
xb_grant_inh	1	I	Xbus grant inhibit control input bus.
drv_grant_inh	1	I	This signal is an input from the outbound pipe block indicating that the grant inhibit control line will be driven in following clock phase.
go_arb	1	I	This signal comes from the outbound pipe. It is asserted when the outbound pipe wants to arbitrate for the Xbus.
own_grnt	1	O	This signal goes to the outbound pipe to notify that arbitration for the Xbus was successful.
drv_req	1	O	This signal is used to issue the Xbus request on the following clock phase.
slot_id[1:0]	2	I	The slot ID signals are hard wired on the backplane for each slot. The slot ID is used to reset the initial arbitration value.

12.5.2 Functional Description

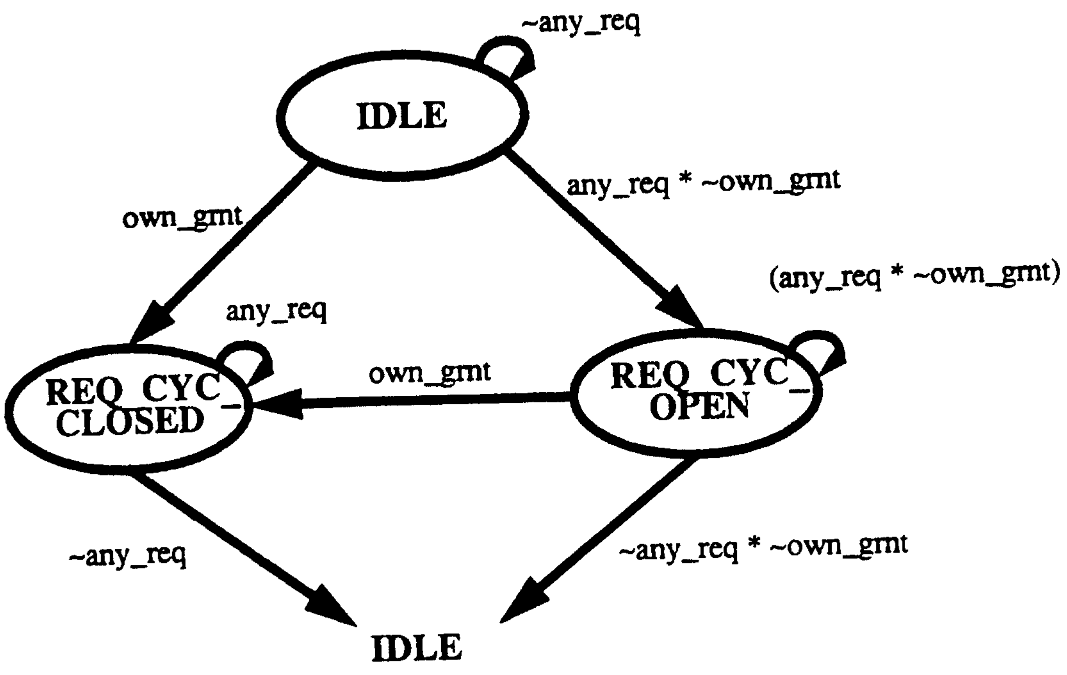
Polo uses a distributed arbitration system. Each CPU and each PCIB has its own arbitration controller. Each board also has connections to the other three boards' control signals and is aware of the arbitration rules. This means each board can keep track of who is arbitrating and who should win. For more detail please refer to section 6.6 on page 40.

The main function of this logical block is to determine when the requesting node captures the Xbus. To do this it maintains a priority address which consistent with that of each of the other three nodes. The arbitration logic also determines when to block the node's request from being issued onto the Xbus. The arbitration control block is comprised of the following functional blocks:

- PRIO_ADD_CNT - This block maintains the priority address for the resident node, this value is determined by the slot_id. The priority address is initialized to a value identical to that of its slot_id whenever a maintenance interrupt occurs.
- ARB_PRIORITY - This block determines whether or not the resident node should be granted the XBUS based on the outstanding requests as well as the current priority address.
- ARB_CONTROL - This block maintains the necessary state information to delineate arbitration cycles. This block also initiates the updating of the priority address in between

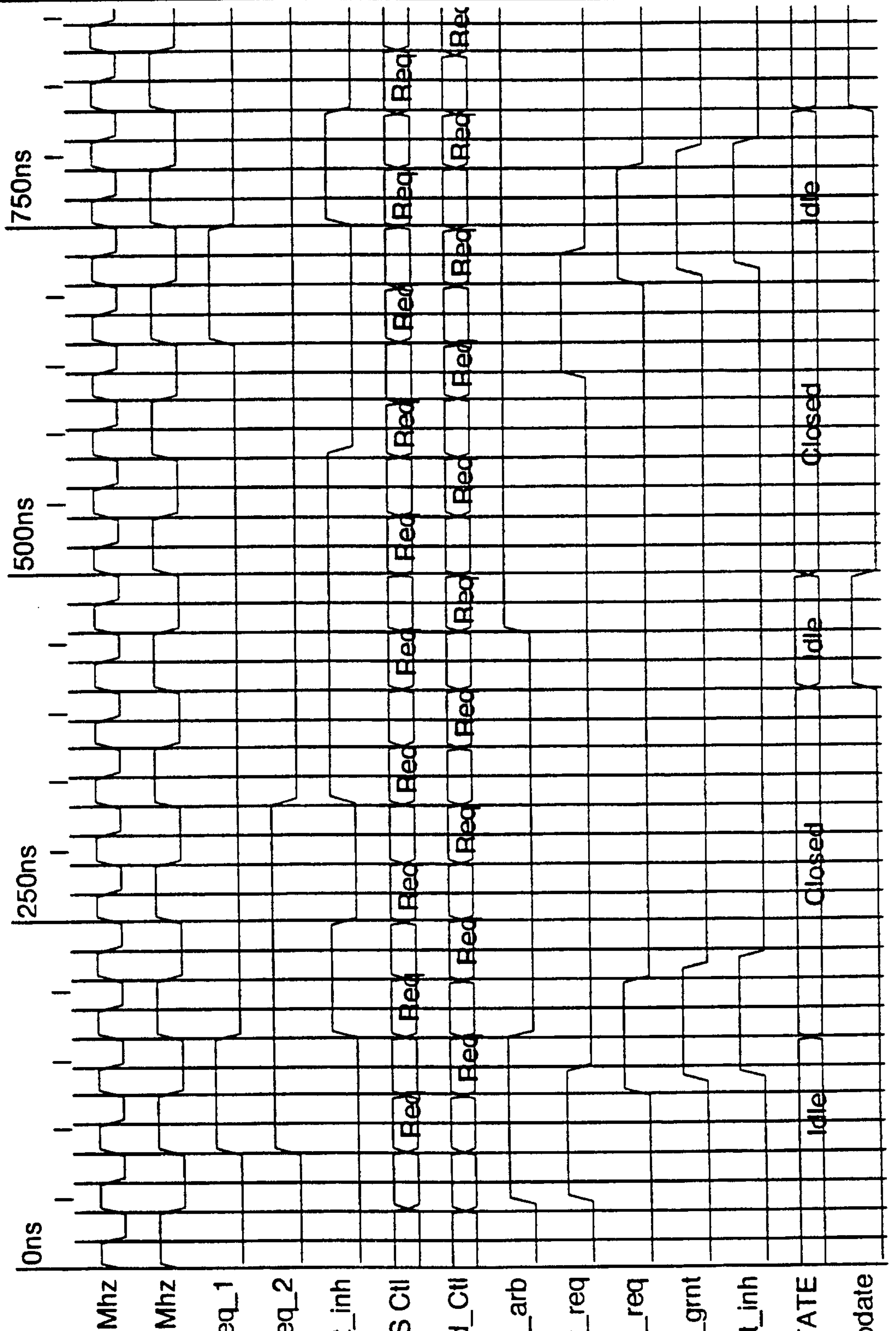
arbitration cycles.

Figure 75. Arbitration State Machine



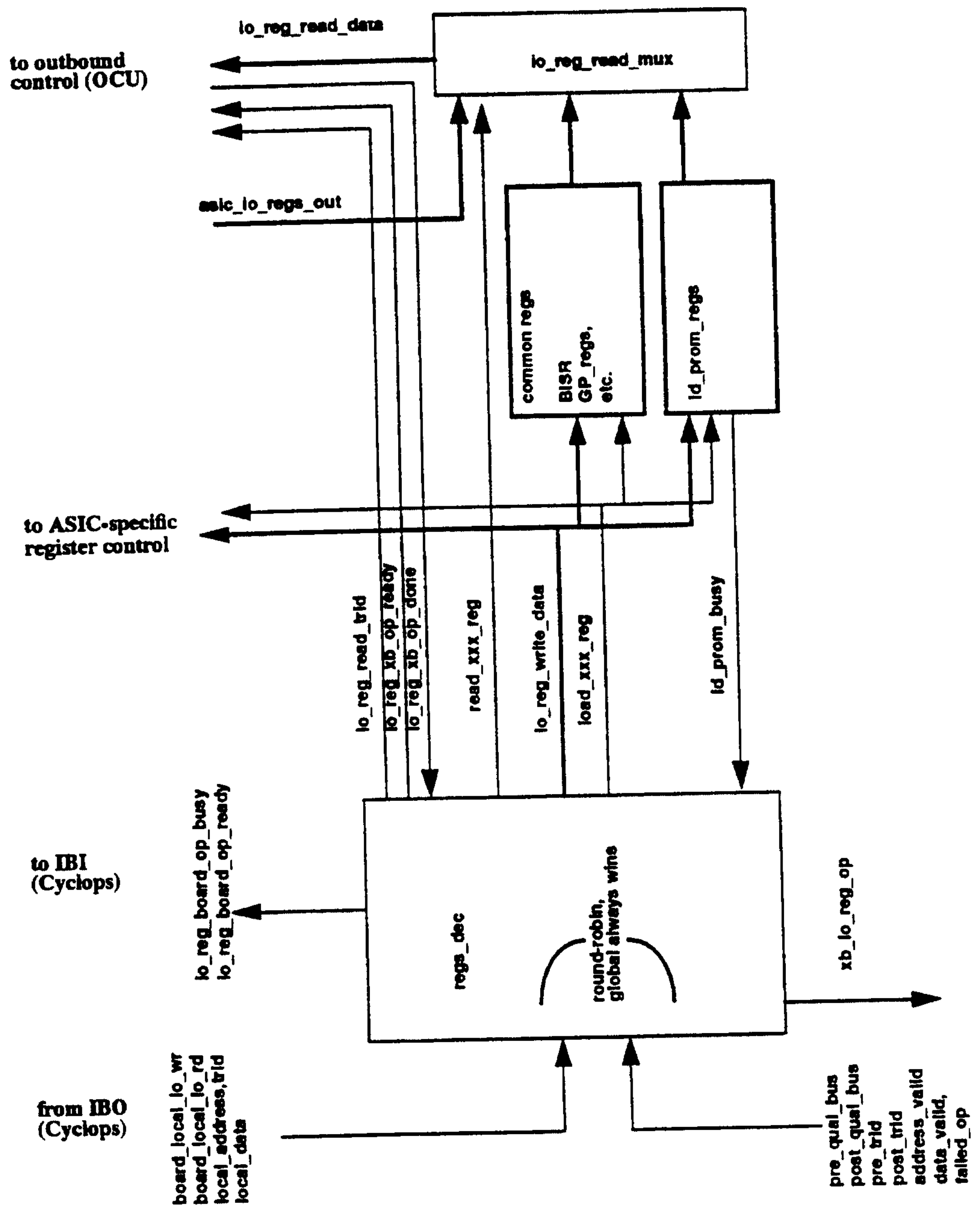
ARBITRATION CONTROL

Figure 76. Arbitration Timing Diagram



12.6 IO Register Section

Figure 77. IO Register Block Diagram



12.6.1 Port List

Table 23. Common Logic Register Control Port List

signal name	width	I/O	description
any_cpu_online	1	I	(Cyclops only) from board states logic, indicates at least one CPU in the system is on-line
asic_broken_status	3:0	I	The ASIC specific broken status register
asic_io_regs_out	31:0	I	The ASIC specific register read output
board_local_bus	31:0	I	(Cyclops only) The local I/O register read/write address/data bus from the IBO.
board_local_byte_enable	3:0	I	(Cyclops only) The local I/O register read/write byte enable bus from the IBO.
board_local_io_rd	1	I	(Cyclops only) Indicates that the board local bus contains an I/O write
board_local_io_wr	1	I	(Cyclops only) Indicates that the board local bus contains an I/O read
board_local_trid	6:0	I	(Cyclops only) The local I/O register read/write trid bus from the IBO.
board_reset	4:0	I	The board reset register
broken	1	I	Broken signal from board states logic
clk48	1	I	48MHz clock input
cold_reset	1	I	cold reset signal
common_broken_status	7:0	I	The common broken status register
diagnostic_broken	1	I	bit 4 of the test control register
duplexed	1	I	(Cyclops only) Duplexed signal from board states logic
drv_xbus	1	I	Indicates that this ASIC drove the Xbus for the current cycle, used by the performance monitoring registers
efq_freeze_state_sta	1	I	(Cyclops only) EFQ Freeze state stable signal from the IBI
freeze_error_regs	1	I	Bit 0 of the bus fault report register
freeze_state	1	I	(Cyclops only) Freeze_state signal from board states logic
map_regs_out	31:0	I	(Cyclops only) The map ram register read output
misc_err_status	4:0	I	The misc. error status register
id_prom_in	1	I	serial input from the ID PROM

Table 23. Common Logic Register Control Port List

signal name	width	I/O	description
io_reg_xbus_op_done	1	I	From the ocu, indicates that the register read return started by io_reg_xbus_op_ready has completed.
io_reg_local_op_done	1	I	(Cyclops only) From the IBI, indicates that the local register read return started by io_reg_local_op_ready has completed.
led_control	2:0	I	The board led control register
online	1	I	Board on-line signal from board states logic.
own_ack	1	I	indicates the assertion of ACK on the Xbus by this controller.
own_busy	1	I	indicates the assertion of BUSY on the Xbus by this controller.
own_main_int	1	I	Indicates that this ASIC issued a maintenance interrupt on the Xbus
phase_12_0	1	I	12 MHz phase clock 1st half of cycle 0 of phase
phase_12_1	1	I	(Cyclops only) 12 MHz phase clock 2nd half of cycle 0 of phase
phase_12_2	1	I	12 MHz phase clock 1st half of cycle 1 of phase
phase_12_3	1	I	12 MHz phase clock 2nd half of cycle 1 of phase
phase_24	1	I	24 MHz phase clock
post_qual_func_op	1	I	Post qual func_op bus
post_qual_info	[31:0]	I	Post qual (after Xbus response) info bus
post_qual_trid	[6:o]	I	Post qual trid bus
pre_qual_func	5:0	I	Pre qual function code bus
pre_qual_func_op	1	I	Pre qual func_op bus
pre_qual_info	[31:0]	I	Pre qual (before Xbus response) info bus
pre_qual_rc	1:0	I	Pre qual RC bus
pre_qual_trid	[6:o]	I	Pre qual trid bus
pu_dside	1	I	hi = D-side ASIC, low = C-side ASIC
set_first_cpu_online	1	I	(Cyclops only) Sets bit 11 of the bus interface state register
set_xbus_int	1	I	(Gambit only) bit 2 of the bus fault report register
sync_complete_dlyd	1	I	(Cyclops only) From the board states logic, indicates that a sync register read has completed

Table 23. Common Logic Register Control Port List

signal name	width	I/O	description
update	1	I	(Cyclops only) Board update mode signal from board states logic.
update	1	I	(Cyclops only) Board update mode signal from board states logic.
warm_reset	1	I	synchronous warm reset to block
xb_ack	1	I	indicates the assertion of ACK by one or more Xbus controllers
xb_add_valid	1	I	indicates that the post qual bus contains a valid address (valid from 12_2 of post3 till 12_2 of post4)
xb_bus_err	1	I	indicates the assertion of Bus Error by one or more Xbus controllers on either the A or B bus
xb_busy	1	I	indicates the assertion of Busy by one or more Xbus controllers
xb_data_valid	1	I	indicates that the post qual bus contains valid data (valid from 12_2 of post3 till 12_2 of post4)
xb_io_reg_failed_op	1	I	Tied to failed_op signal from XBI. When true indicates that the data signaled valid by xb_data_valid is the result of a failed_op.
asic_specific_broken_config	1	O	The ASIC Specific Broken Configuration register
board_local_data_sel	1	O	Tells the IBO to place data on the board local bus for writing to the register currently indicated on the board local bus
board_local_done	1	O	(Cyclops only) Indicates to the IBO that the current local I/O access has completed
break_even_board_on_ta	1	O	Bit 18 of the Bus Interface State register
break_pcib_on_err	1	O	Bit 21 of the Bus Interface State register
bus_fault_report	1	O	The bus fault report register
clr_broken_command	1	O	Un-break the board
clr_diag_broken	1	O	Clear bit 7 of the Test Control register
clr_efq_freeze_state	1	O	(Cyclops only) Clears EFQ Freeze State
clr_freeze_state	1	O	(Cyclops only) Clears Freeze State
clr_online	1	O	Clears On-line mode
clr_udpate	1	O	(Cyclops only) Clears Update mode
id_prom_out	[1:0]	O	Output to the external ID PROM

Table 23. Common Logic Register Control Port List

signal name	width	I/O	description
io_reg_cd_diff	1	O	Indicates that this register access is CD different
io_reg_i_drive	1	O	(Gambit only) Indicates that the Gambit will drive the Xbus on this I/O register access.
io_reg_local_op_ready	1	O	(Cyclops only) Indicates to the IBI that a local I/O register read return is ready for the IBus
io_reg_write_data	[31:0]	O	The data to be written into a given register on a load_address_xxx signal being asserted.
io_reg_read_trid	[6:0]	O	The TRID for the Xbus register read return signalled by io_reg_xbus_op_ready
io_reg_single_side	1	O	(Gambit only) Indicates that this register access is a single sided return.
io_reg_take_cycle	1	O	Tells the PING module to the current Xbus read is being respond to so the PING table has to be loaded
io_reg_xbus_op_ready	1	O	Indicates to the OCU that a register read return is ready for the Xbus
io_reg_write_data	[31:0]	O	The data to be written into a given register on a load_address_xxx signal being asserted.
leave_sync_fast_duplex	1	O	(Cyclops only) Bit 7 of the Bus Interface State register
leave_sync_full_duplex	1	O	(Cyclops only) Bit 8 of the Bus Interface State register
leave_sync_point_jump_ switch	1	O	(Cyclops only) Bit 10 of the Bus Interface State register
load_xxx	1	O	Load register xxx (one for each writable bus ASIC register not contained with the register control block)
p2p_mode_enable	1	O	(Cyclops only) Bit 22 of the Bus Interface State register
read_xxx	1	O	Read register xxx (one for each writable bus ASIC register not contained with the register control block)
rearm_error_regs	1	O	Set by writing bit zero of the Bus Fault Report register
set_broken_command	1	O	Break the board
set_efq_freeze_state	1	O	(Cyclops only) Sets EFQ Freeze State
set_freeze_state	1	O	(Cyclops only) Sets Freeze State

Table 23. Common Logic Register Control Port List

signal name	width	I/O	description
set_online	1	O	Sets On-line mode
set_udpate	1	O	(Cyclops only) Sets Update mode
software_set_maint_int	1	O	Set by writing bit one of the Bus Fault Report register
xb_io_reg_ack	1	O	Xbus ACK from the register response logic
xb_io_reg_busy	1	O	Xbus BUSY from the register response logic

12.6.2 Functional Description

This block contains a sub-block which controls the reading and writing of all registers within the bus ASICs. In addition, it contains the a sub-block for the control of the ID-PROM registers, a sub-block containing several of the common registers, and a sub-block that is used to mux all of the register read sources together. The timing for a register write is shown in figure 78 and the timing for an Xbus register read is shown in figure 79

Figure 78. Register Write Timing

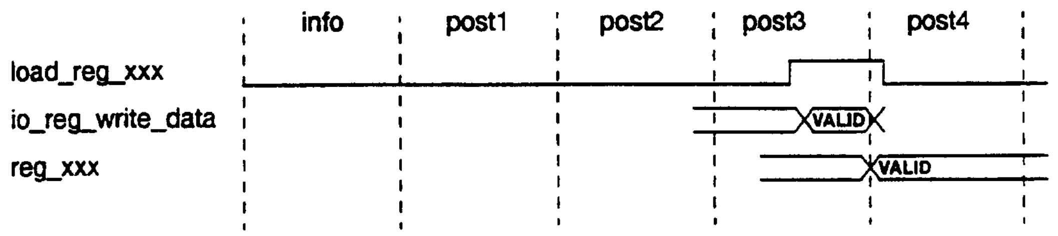
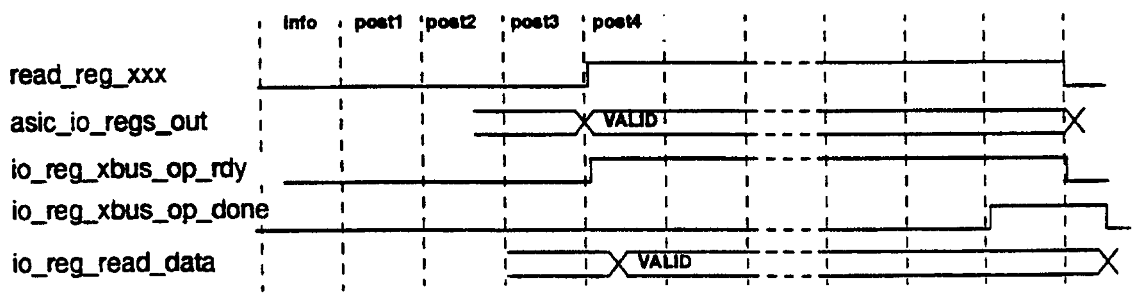
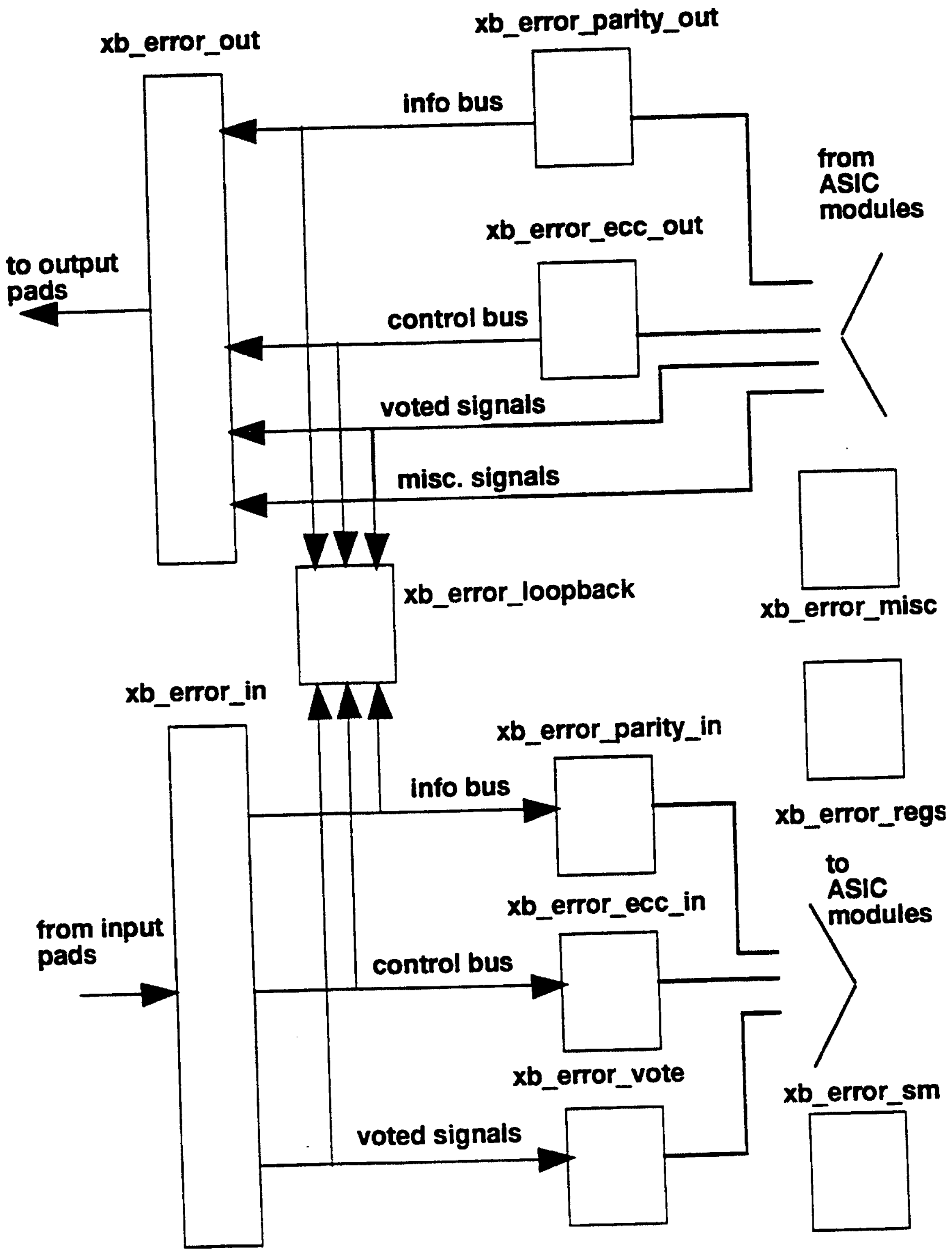


Figure 79. Register Read Timing



12.7 Error Checking and Registering Logic

Figure 80. Error Checking Block Diagram



12.7.1 Port List

Table 24 describes the port list to the error checking and registering logic.:

Table 24. Error Checking and Registering Port List

signal name	width	I/O	description
clk48	1	I	48MHz clock input
clk48_12_0	1	I	48MHz clock valid at 12_0
clk48_12_1	1	I	48MHz clock valid at 12_1
clk48_12_2	1	I	48MHz clock valid at 12_2
clk48_12_3	1	I	48MHz clock valid at 12_3
clk48_24	1	I	24MHZ clock
clk48_24_not	1	I	Inverted 24MHZ clock
phase_12_0	1	I	12MHz phase clock 1st half of cycle 0 of phase
phase_12_1	1	I	12MHz phase clock 2nd half of cycle 0 of phase
phase_12_2	1	I	12MHz phase clock 1st half of cycle 1 of phase
phase_12_3	1	I	12MHz phase clock 2nd half of cycle 1 of phase
phase_24	1	I	24MHZ phase clock
warm_reset	1	I	Used to warm reset the gate array
cold_reset	1	I	Used to cold reset the gate array
alert_	1	I	Signal is asserted when power to the board is not okay, used to set broken
info_ai	[31:0]	I	Info Bus A coming from input pads of ASIC
info_bi	[31:0]	I	Info Bus B coming from input pads of ASIC
trid_ai	[6:0]	I	Trid Bus A coming from input pads of ASIC
trid_bi	[6:0]	I	Trid Bus B coming from input pads of ASIC
func_op_ai_	[31:0]	I	Func Op A coming from input pads of ASIC
func_op_bi_	[31:0]	I	Func Op B coming from input pads of ASIC
parity_ai	1	I	Parity A coming from input pads of ASIC
parity_bi	1	I	Parity B coming from input pads of ASIC
info_ao	[31:0]	O	Registered Info Bus A going to output pads of ASIC
info_bo	[31:0]	O	Registered Info Bus B going to output pads of ASIC

Table 24. Error Checking and Registering Port List

signal name	width	I/O	description
trid_ao	[6:0]	O	Registered Trid A going to output pads of ASIC
trid_bo	[6:0]	O	Registered Trid B going to output pads of ASIC
func_op_ao_	1	O	Registered Func Op A going to output pads of ASIC
func_op_bo_	1	O	Registered Func Op B going to output pads of ASIC
parity_ao	1	O	Registered parity A going to output pads of ASIC
parity_bo	1	O	Registered parity B going to output pads of ASIC
control_in_ni	[7:0]	I	Control Bus (neighbor) coming from input pads of ASIC
control_in_oi	[7:0]	I	Control Bus (opposite) coming from input pads of ASIC
control_in_pi	[7:0]	I	Control Bus (peer) coming from input pads of ASIC
control_out_no	[7:0]	O	Registered Control Bus (neighbor) going to output pads of ASIC
control_out_oo	[7:0]	O	Registered Control Bus (opposite) going to output pads of ASIC
control_out_po	[7:0]	O	Registered Control Bus (peer) going to output pads of ASIC
control_out_ni	[7:0]	I	Looped-back version of control_out_no, used for loopback checking
control_out_oi	[7:0]	I	Looped-back version of control_out_oo, used for loopback checking
control_out_pi	[7:0]	I	Loopbacked version of control_out_po, used for loopback checking
slot_idai	1	I	Slot Id A coming from input pads of ASIC
slot_idbi	1	I	Slot Id B coming from input pads of ASIC
slot_id_latched	1	O	Latched version of slot_id
board_not_broken_in_n_xi	1	I	Board Not Broken (neighbor, x version) coming from input pads of ASIC
board_not_broken_in_n_yi .	1	I	Board Not Broken (neighbor, y version) coming from input pads of ASIC
board_not_broken_in_n_zi	1	I	Board Not Broken (neighbor, z version) coming from input pads of ASIC

Table 24. Error Checking and Registering Port List

signal name	width	I/O	description
board_not_broken_in_o_xi	1	I	Board Not Broken (opposite, x version) coming from input pads of ASIC
board_not_broken_in_o_yi	1	I	Board Not Broken (opposite, y version) coming from input pads of ASIC
board_not_broken_in_o_zi	1	I	Board Not Broken (opposite, z version) coming from input pads of ASIC
board_not_broken_in_p_xi	1	I	Board Not Broken (partner, x version) coming from input pads of ASIC
board_not_broken_in_p_yi	1	I	Board Not Broken (partner, y version) coming from input pads of ASIC
board_not_broken_in_p_zi	1	I	Board Not Broken (partner, z version) coming from input pads of ASIC
board_not_broken_in_n	1	0	Three-way voted version of board_not_borken_in_n(x,y,z) signals, goes to internal ASIC modules
board_not_broken_in_o	1	O	Three-way voted version of board_not_borken_in_o(x,y,z) signals, goes to internal ASIC modules
board_not_broken_in_p	1	O	Three-way voted version of board_not_borken_in_p(x,y,z) signals, goes to internal ASIC modules
board_not_broken_out	1	O	Registered Board Not Broken going to output pads of ASIC
These following signals are specific to the Gambit gate array only			
recc_in_n_xi	1	I	ReCC Reset neighbor (x version) coming from input pads of ASIC
recc_in_n_yi	1	I	ReCC Reset neighbor (y version) coming from input pads of ASIC
recc_in_n_zi	1	I	ReCC Reset neighbor (z version) coming from input pads of ASIC
recc_in_o_xi	1	I	ReCC Reset opposite (x version) coming from input pads of ASIC
recc_in_o_yi	1	I	ReCC Reset opposite (y version) coming from input pads of ASIC
recc_in_o_zi	1	I	ReCC Reset opposite (z version) coming from input pads of ASIC

Table 24. Error Checking and Registering Port List

signal name	width	I/O	description
recc_in	1	O	Three-way voted version of recc_n_in (x,y,z) signals or the Three-way voted version of recc_o_in (x,y,z), goes to internal ASIC modules
single_side	1	I	Asserted when transaction action is single sided
These following signals are specific to the Cyclops gate array only			
recc_in_p_xi	1	I	ReCC Reset peer (x version) coming from input pads of ASIC
recc_in_p_yi	1	I	ReCC Reset peer (y version) coming from input pads of ASIC
recc_in_p_zi	1	I	ReCC Reset peer (z version) coming from input pads of ASIC
recc_in	1	O	Three-way voted version of recc_in_p (x,y,z) signals, goes to internal ASIC modules
drv_reset_out	1	O	The ReCC Reset output (unregistered version)
recc_out_no	1	O	Registered Recc Out (neighbor) going to output pads of ASIC
recc_out_oo	1	O	Registered Recc Out (opposite) going to output pads of ASIC
recc_out_po	1	O	Registered Recc Out (peer) going to output pads of ASIC
recc_out_ni	1	I	Loopbacked version of recc_out_no, used for loopback checking
recc_out_oi	1	I	Loopbacked version of recc_out_oo, used for loopback checking
recc_out_pi	1	I	Loopbacked version of recc_out_po, used for loopback checking
next_sync_out	1	I	Pre-registered version of sync_out, from xb_brdstate_rtl.v
sync_outo_	1	O	registered version of next_sync_out combined with broken
sync_in_xi_	1	I	Sync In (x version) coming from input pads of ASIC
sync_in_yi_	1	I	Sync In (y version) coming from input pads of ASIC
sync_in_zi_	1	I	Sync In (z version) coming from input pads of ASIC

Table 24. Error Checking and Registering Port List

signal name	width	I/O	description
sync_in_	1	O	Three-way voted version of sync_in_zi_(x,y,z) signals, goes to internal ASIC modules
sync_outi_	1	I	Loopbacked version of sync_outo_, used for loopback checking
your_online_in_xi	1	I	On-line from partner (x version) coming from input pads of ASIC
your_online_in_yi	1	I	On-line from partner (y version) coming from input pads of ASIC
your_online_in_zi	1	I	On-line from partner (z version) coming from input pads of ASIC
your_online_in	1	O	Three-way voted version of your_online_in(x,y,z) signals, goes to internal ASIC modules
drv_my_online_out	1	I	Preregistered version of my_online_out coming from internal ASIC module
my_online_outo	1	O	registered version of drv_my_online_out going to output pads of ASIC
my_online_out_xi	1	I	Buffered, loopbacked version of my_online_outo (x version), used for looback checking
my_online_out_yi	1	I	Buffered, loopbacked version of my_online_outo (y version), used for looback checking
my_online_out_zi	1	I	Buffered, loopbacked version of my_online_outo (z version), used for looback checking
These following signals are common to both Cyclops and Gambit gate array			
cmp_err_in_ai	1	I	These inputs come from the other side Cyclops' cmp_err_out. When asserted, it indicates the other side Cyclops detected a miscompare on the bus and returning data did not match the data that was driven by this ASIC.
cmp_err_in_bi	1	I	
cmp_err_out_ao	1	O	These outputs are driven to the other side ASIC's cmp_err_in. When asserted, it indicates this Cyclops has detected a miscompare on the bus and the returning data did not match the data that was driven by the other side Cyclops
cmp_err_out_bo	1	O	

Table 24. Error Checking and Registering Port List

signal name	width	I/O	description
drive_err_in_ai	1	I	These inputs are tied to drive_err_out_a from the other side Cyclops' compare check. When asserted, it indicates the other side Cyclops has detected a miscompare on the bus in that the returning data did not match the data that was driven by the other side Cyclops ASIC.
drive_err_in_bi	1	I	
drive_err_out_ao	1	O	These inputs are tied to the outputs from the other side Cyclops ASIC's compare check. When asserted, it indicates the other side Cyclops has detected a miscompare on the bus in that the returning data did not match the data that was driven by the other side Cyclops ASIC.
drive_err_out_bo	1	O	
cside_ctl_erri	1	I	Signals that c-side gate array control signals differ from dside control signals, this signal is used by the d-side
cside_ctl_erro			Signals that c-side gate array control signals differ from dside control signals, this signal is driven by the c-side
cside_ctl_err_oe_			This signal controls cside_ctl_erro, if it is the c-side gate array this signal is always low (enabling cside_ctl_erro to be driven out), if it is the d-side this signal is always high
info_a_reg	[31:0]	O	Registered version of Info_ai, goes to internal ASIC modules
info_b_reg	[31:0]	O	Registered version of Info_bi, goes to internal ASIC modules
trid_a_reg	[6:0]	O	Registered version of trid_ai, goes to internal ASIC modules
trid_b_reg	[6:0]	O	Registered version of trid_bi, goes to internal ASIC modules
func_op_a_reg_	1	O	Registered version of func_op_ai_, goes to internal ASIC modules
func_op_b_reg_	1	O	Registered version of func_op_bi_, goes to internal ASIC modules
parity_a_reg	1	O	Registered version of parity_ai_, goes to internal ASIC modules
parity_b_reg	1	O	Registered version of parity_bi_, goes to internal ASIC modules
xb_bus_req_corr_n	1	O	Registered, ECC corrected version of control_in_ni[0], valid 12_2 to 12_0, goes to internal ASIC modules

Table 24. Error Checking and Registering Port List

signal name	width	I/O	description
xb_bus_req_corr_o	1	O	Registered, ECC corrected version of control_in_oi[0], valid 12_2 to 12_0, goes to internal ASIC modules
xb_bus_req_corr_p	1	O	Registered, ECC corrected version of control_in_pi[0], valid 12_2 to 12_0, goes to internal ASIC modules
xb_bus_req	1	O	'ored' version of xb_bus_req_corr_ (n,o,p)
xb_ack	1	O	'ored' version of registered, ECC corrected control_in_ni[0], control_in_oi[0], control_in_pi[0] signals sampled at 12_0 and also 'ored' with registered version of drv_xb_ack
own_ack	1	O	'ored' version of all internal acknowledges double registered
xb_maint_int	1	O	'ored' version of registered, ECC corrected control_in_ni[1], control_in_oi[1], control_in_pi[1] signals sampled at 12_0 and also 'ored' with registered version of drv_xb_maint_int
xb_busy	1	O	'ored' version of registered, ECC corrected control_in_ni[2], control_in_oi[2], control_in_pi[2] signals sampled at 12_0 and also 'ored' with registered version of drv_xb_busy
own_busy	1	O	'ored' version of all internal BUSYs double registered
xb_grant_inh	1	O	'ored' version of registered, ECC corrected control_in_ni[1], control_in_oi[1], control_in_pi[1] signals sampled at 12_2 and also 'ored' with registered version of drv_xb_grant_inh
xb_bus_err	1	O	'ored' version of registered, ECC corrected control_in_ni[2], control_in_oi[2], control_in_pi[2], control_in_ni[3], control_in_oi[3], control_in_pi[3] signals sampled at 12_2 and also 'ored' with registered version of drv_xb_bus_err_a and drv_xb_bus_err_b
drv_xb_bus_req	1	I	demultiplexed nonregistered version of control_out[0] coming from internal ASIC module

Table 24. Error Checking and Registering Port List

signal name	width	I/O	description
drv_xb_grant_inh	1	I	demultiplexed nonregistered version of control_out[1] coming from internal ASIC module
drv_xb_bus_err_a	1	I	demultiplexed nonregistered version of control_out[2] coming from internal ASIC module
drv_xb_bus_err_b	1	I	demultiplexed nonregistered version of control_out[3] coming from internal ASIC module
board_specific_ack	1	I	demultiplexed nonregistered term of control_out[0] coming from internal ASIC
xb_io_reg_ack	1	I	demultiplexed nonregistered term of control_out[0] coming from internal ASIC module
drv_xb_maint_int	1	I	demultiplexed nonregistered version of control_out[1] coming from internal ASIC module
board_specific_busy	1	I	demultiplexed nonregistered term of control_out[2] coming from internal ASIC module
inpipe_busy	1	I	demultiplexed nonregistered term of control_out[2] coming from internal ASIC module
The following signal is specific to the Cyclops gate array only			
ping_fifo_full_busy	1	I	demultiplexed nonregistered term of control_out[2] coming from internal ASIC module
These following signals are common to both Cyclops and Gambit gate array			
xb_io_reg_busy	1	I	demultiplexed nonregistered term of control_out[2] coming from internal ASIC module
drv_info	[31:0]	I	nonregistered version of info_o coming from internal ASIC module
drv_trid	[6:0]	I	nonregistered version of trid_o coming from internal ASIC module
drv_func_op_	1	I	nonregistered version of func_op_o_ coming from internal ASIC module
ping_ack_out	1	I	demultiplexed nonregistered term of control_out[0] coming from internal ASIC module

Table 24. Error Checking and Registering Port List

signal name	width	I/O	description
info_a_oe_	1	O	output enable for info_a bus
info_b_oe_	1	O	output enable for info_b bus
trid_a_oe_	1	O	output enable for trid_a bus
trid_b_oe_	1	O	output enable for trid_b bus
func_op_a_oe_	1	O	output enable for func_op_a_ line
func_op_b_oe_	1	O	output enable for func_op_b_ line
parity_a_oe_	1	O	output enable for parity_a line
parity_b_oe_	1	O	output enable for parity_b line
control_out_n_oe_	1	O	output enable for control_out_n line
control_out_o_oe_	1	O	output enable for control_out_o line
control_out_p_oe_	1	O	output enable for control_out_p line
my_online_out_x_oe_	1	O	output enable for my_online_out_x line
my_online_out_y_oe_	1	O	output enable for my_online_out_y line
my_online_out_z_oe_	1	O	output enable for bmy_online_out_z line
reset_out_n_oe_	1	O	output enable for reset_out_n line
reset_out_o_oe_	1	O	output enable for reset_out_o line
reset_out_p_oe_	1	O	output enable for reset_out_p line
sync_out_oe_	1	O	output enable for sync_out line
xb_err_drv	[1:0]	O	Indicates when and what to drive during error sequence, goes to internal ASIC module 00 - Normal operation 01 - Idle condition 10 - Drive 5s then As 11 - Drive 0s
drv_xbus	1	I	This signal comes from the XBO, it is used to control the output buffers on the info bus
early_err_in_progress	1	O	Asserted when error state machine is currently executing the error protocol or the board is in funny_state, goes to internal ASIC modules, it is asserted on 12_1 of Post3 and deasserted on 12_1 of Err2
err_in_progress	1	O	Asserted when error state machine is currently executing the error protocol or the board is in funny_state, goes to internal ASIC modules, it is asserted on 12_2 of Post3 and deasserted on 12_2 of Err2

Table 24. Error Checking and Registering Port List

signal name	width	I/O	description
arb_err_in_progress	1	O	Asserted when error state machine is currently executing the error protocol or the board is in funny_state, goes to internal ASIC modules, it is asserted on 12_2 of Post3 and deasserted on 12_2 of Err2
pu_dside	1	I	This hard-wired input indicates whether this is the C (low) or D (high) side ASIC.
load_error_fault_bit0	1	I	Enables loading of error fault bit0 register
load_error_fault_bit1	1	I	Enables loading of error fault bit1 register
load_error_data_match	1	I	Enables loading of error data match register
load_error_control	1	I	Enables loading of error control register
io_reg_write_data	[31:0]	I	The data bus that is used to write the error registers
read_error_fault_bit0	1	I	Enables reading of error fault bit0 register
read_error_fault_bit1	1	I	Enables reading of error fault bit1 register
read_error_data_match	1	I	Enables reading of error data match register
read_error_control	1	I	Enables reading of error control register
read_bus_info_err_status	1	I	Enables reading of bus info error status register
read_control_bus_err_status	1	I	Enables reading of control bus error status register
read_voter_err_xcvr_status	1	I	Enables reading of voter error transceiver status register
read_bus_err_byte_status	1	I	Enables reading of bus error byte status register
xbus_error_regs_out	[31:0]	O	The data bus that is used to read the error registers
control_bus_error	1	O	Asserted when error logic sees a control bus error, used by xb_brdstate_rtl.v to assert freeze_error_regs
bus_error_freeze	1	O	Asserted when error logic sees a bus error that it asserted, used by xb_brdstate_rtl.v to assert freeze_error_regs
voter_failure	1	O	Asserted when error logic sees a voter error, used by xb_brdstate_rtl.v to assert freeze_error_regs

Table 24. Error Checking and Registering Port List

signal name	width	I/O	description
seen_bus_error	1	O	Asserted when error logic sees a bus error asserted by another board, used by xb_brdstate_rtl.v to assert freeze_error_regs
freeze_error_regs	1	I	From xb_brdstate_rtl.v, used to freeze error registers
break_pcib_on_err	1	I	This signal is used to signify if the CPU or PCIB should break for the arbitrary broken case
broken	1	O	Asserted when board is broken, asserted on 12_0
The following signal is specific to the Gambit gate array only			
gam_spec_broken	[3:0]	O	Replicated versions of broken, used for timing purposes in Gambit
These following signals are common to both Cyclops and Gambit gate array			
set_broken	1	O	Combinatorial preregistered version of broken
broken_12_3	1	O	Asserted when board is broken, asserted on 12_3 after broken is asserted
debug_conn_broken_outo	1	O	Preregistered version of broken qualified with phase_12_0
diagnostic_broken	1	O	Diagnostic version of broken, behaves the same as broken except it can be cleared with clear diagnostics broken command
my_set_broken	1	O	Registered and latched version of someone_set_broken
broken_out_	1	O	Active low version of broken
three_way_loopback_set_broken	1	O	Asserted when there is a loopback error on voted signals (reset, sync, on-line), this signal is 0 for Gambit
loopback_error_set_broken	1	O	Asserted when there is a loopback error on the info bus
slot_parity_error_set_broken	1	O	Asserted when slot_ida and slot_idb disagree
other_cpu_broken	1	O	Inverted version of board_not_broken from the peer board, not used in Gambit
xb_arbitrary_set_broken	1	O	Asserted when board breaks due to arbitrary broken
xb_heuristic_set_broken	1	O	Asserted when board breaks due to heuristic broken

Table 24. Error Checking and Registering Port List

signal name	width	I/O	description
xbus_cntl_err_set_broken	1	O	Asserted when there is a loopback error on the control bus
These following signals are specific to the Gambit gate array only			
xb_parity_gen_set_broken	1	O	Asserted when the duplexed parity generators disagree
break_pcib	1	I	Asserted by internal Gambit ASIC module to break the PCIB
These following signals are common to both Cyclops and Gambit gate array			
broken_out_oe_	1	O	Unused signal
otherside_set_broken	1	I	Other ASIC's my_set_broken
otherside_set_broken_reg	1	O	Registered (48mhz) version of otherside_set_broken
asic_specific_set_broken	1	I	This signal is asserted when the board needs to break for reasons that the error logic does not detect, software set broken command for example or Cougar set broken (CPU only)
clr_broken_command	1	I	Allows software to clear a broken state
clr_diag_broken_command	1	I	Allows software to clear a diagnostics broken state
newest_cd_read_return	1	I	Prevents info loopback checking when it is CD different data
xb_grant_opposite	1	I	From xb_arb_rtl.v, asserted when the opposite board is granted use of the bus, used to prevent looking at the middle of an info cycle when a board first comes up
xb_grant_neighbor	1	I	From xb_arb_rtl.v, asserted when the neighbor e board is granted use of the bus, used to prevent looking at the middle of an info cycle when a board first comes up
xb_grant_peer	1	I	From xb_arb_rtl.v, asserted when the peer board is granted use of the bus, used to prevent looking at the middle of an info cycle when a board first comes up
xb_own_grant	1	I	From xb_arb_rtl.v, asserted when this board is granted use of the bus, used to prevent looking at the middle of an info cycle when a board first comes up
These following signals are specific to the Cyclops gate array only			

Table 24. Error Checking and Registering Port List

signal name	width	I/O	description
funny_state_n	1	O	Asserted when neighbor board is in funny_state, XBI uses the signal to ignore a PCIB if it is in the funny_state
funny_state_o	1	O	Asserted when opposite board is in funny_state, XBI uses the signal to ignore a PCIB if it is in the funny_state

12.7.2 Functional Description

The Error Checking and Registering logic performs two basic functions. It controls most functions of error checking and it registers incoming and outgoing Xbus signals. The error checking function performs the following tasks:

- ECC generation and checking/correcting
- Parity generation and checking
- Loopback checking
- Three-way voting
- Controls error protocol (includes decisions on breaking board)
- Provides logic for error insertion
- Records error information

The registering logic performs the following tasks:

- registers incoming bus signals
- registers outgoing bus signals
- combines (logical 'or') neighbor, opposite, peer versions of registered ECC corrected control signals coming into the ASIC with the delayed outgoing corresponding signal to produce a combined control signal for internal ASIC module use

The Xbus can be broken into four classes of signals when discussing error checking:

Xbus Signals: these signals are protected by one parity bit, this detects an odd number of errors. During normal accesses the D-side gate array drives approximately half of the signals and the C-side drives the other half. Please refer to Section 11.1 on page 96 for more detail on the signals that are driven by D-side and C-side. The D-side checks the data it has driven (loopback drive check) and the D-side checks the data that the C-side has driven with what the D-side would have driven (loopback compare check). During single sided accesses (PCIB only) the driving ASIC drives all of the bus signals. The driving ASIC can do a loopback drive check but the other ASIC cannot do a loopback compare check because it has no knowledge of the exact data being driven.

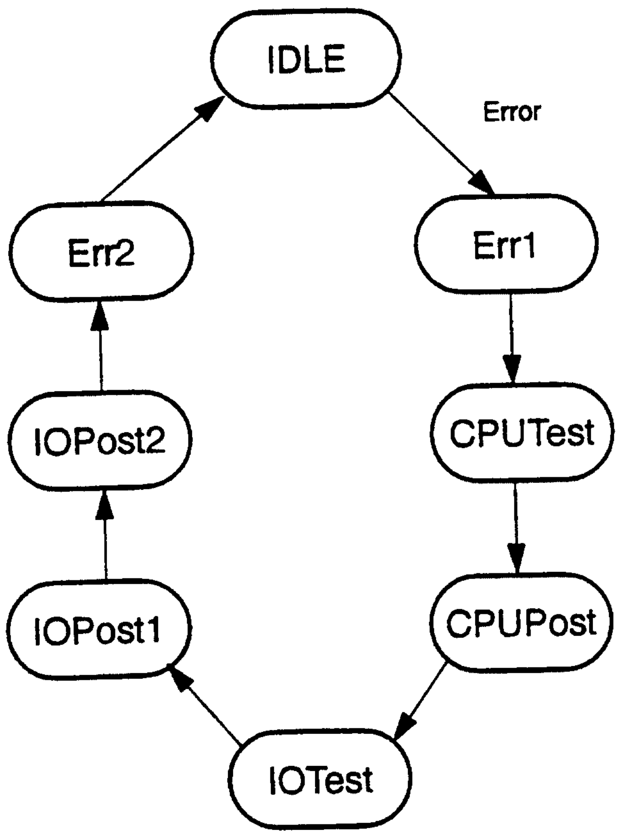
Control Bus Signals: these signals are protected by four bits of ECC, this detects a double bit error and can correct a single bit error. The D-side drives the signals and the C-side does the loopback compare check. The D-side does the loopback drive check.

Voted signals: these signals are triplicated and protected by three-way voting, this allow for one out of the three lines to be in error. The D-side drives the signals and the C-side does the loopback compare check. The D-side does the loopback drive check. The one exception to D-side driving all signals is the board_not_broken_ signal. The C-side will drive the board_not_broken_ signal and the D-side will drive the enable for the board_not_broken_ signal.

Miscellaneous signals: The slot id signals are duplicated and if they are in disagreement the board will break.

Figure 81 shows the Bus Error flow chart, please refer to section 6.4 on page 30 for a detailed explanation of errors.

Figure 81. Bus Error Operation Flow Chart



13. Xbus/Golfbus Differences

Table 25. XBus/Golfbus Differences

Golfbus	XBus
single logical fully interconnected bus, duplicated (2 physical buses), shared by all boards	4 distributed point-to-point buses, not duplicated, each board connects to 2 of the buses, but still maintain a single-bus view
A & B buses are lockstepped	A & B buses are independent
Special bus transceivers to support live insertion	No bus transceivers (except for the 3-way voted signals). To support live insertion, buses to broken boards must not be driven high.
info[63:0]: 32 & 64-bit bus widths	info[31:0]: 32-bit only
Post3 ACK	Post2 ACK
Bus Error: Single-Phase Post4 Error	Bus Error: 7-Phase Error protocol
Arbitration: binary search, implemented with 4-bit counter, no dead cycles	Arbitration: distributed, implemented with 2-bit counter, dead cycle needed between two different boards' info's (implemented via grant_inh)
Fully supported Peer-To-Peer Transactions	CPU-only Peer-To-Peer Transactions
64-bit bus func_op's	unsupported
transceiver loopback mode	unsupported
unsupported	New info[31:00] IOVA Address Format
low_parity[7:0], up_parity[7:0]	info_parity[0]
control signals triplicated, 3-way voted arb arb_req ack_maint_int grnt_inh busy bus_error<a,b><c,d> gb_clk8	control bus, w/check bits control[0]: bus_req & ack control[1]: grant_inh & maint_int control[2]: bus_err_a & busy control[3]: bus_err_b & unused control[7:4]: checkbits xb_clk8

Table 25. XBus/Golfbus Differences

Golfbus	XBus
Other control signals: recc (warm reset) (3-way voted) slot_id[3:0], parity pair_comm[7:4]: ta pair_comm[3:0]: unused btl_term driving_bus	Other control signals: slot_ida, slot_idb reset (3-way voted) board_not_broken (3-way voted) sync (3-way voted) even_odd_online (3-way voted) ta_d, ta_c partners_ta_d, partners_ta_c

func_op/trid[6:0]:
first cycle:
trid[3] = slot_id[3]
second cycle:
func_op = obey_a
trid[6] = obey_b
trid[4] = bus_size
	func_op/trid[6:0]:
	first cycle:
	trid[3] = System/IOVA Address
	second cycle:
	func_op = unused
	trid[6] = unused
	trid[4] = unused

CLAIMS:

1. A fault-detecting and fault-isolating digital data processing apparatus comprising

plural functional units,

plurality of bus means, each connected to and providing point-to-point communications between a respective pair of functional units,

the functional units each including

error phase means, coupled to the respective bus means of that functional unit, for placing the functional unit in an error isolation phase substantially concurrently with the other functional units, and for transmitting test data onto at least one of those bus means during a respective portion of that phase and exclusive, with respect to that phase, of any other unit connected to that bus,

bus error detecting means, coupled to the respective bus means of that functional unit, for detecting a communication error, including any of parity error, an error correction code error and a loopback error, on that bus means, and for signalling a bus error to the other functional units in the event of such communication error,

error isolation means, coupled to the bus error detecting means and to the loopback error detecting means, for signaling the other functional units during the error isolation phase that the respective functional unit is faulty based on (i) whether the bus error means of that functional unit detected a loopback error, (ii) whether the bus error means of another functional unit signalled a bus error in response to test data transmitted during the error isolation phase, and (iii) whether another functional unit signaled that it was faulty.

2. A fault-detecting digital data processing apparatus according to claim 1, wherein the plurality of bus means provide point-to-point communications between each pair of functional units.

3. A fault-detecting digital data processing apparatus according to claim 1, wherein the error phase means are responsive to a bus error signaled by any of the functional units for placing the respective functional units in an error phase.

4. A fault-detecting digital data processing apparatus according to claim 1, wherein the error isolation means includes means for taking off-line a functional unit that signals that it is faulty.

5. A fault-detecting digital data processing apparatus according to claim 1, wherein the bus error detecting means of at least one functional unit comprises loopback error detection means, coupled to the respective bus means and error phase means, for comparing test data transmitted onto that bus means with data received substantially concurrently from the bus means, and for signaling a bus error to the other functional units in the event of discrepancy between the transmitted and received data.

6. A fault-detecting digital data processing apparatus according to claim 5, wherein at least a selected functional unit includes

a processing section for generating a datum for communication to another functional unit,

a drive-side bus interface means, coupled with the processing section, for applying a first portion of that datum to at least the bus means coupling the selected functional unit to that other functional unit, a check-side bus interface means, coupled with the processing section, for applying a second, complementary portion of that datum to at least the bus means coupling the selected functional unit to that other functional unit, and

each of the drive-side and check-side interface means comprise means for receiving a datum driven to at least the bus means coupling the selected functional unit to that other functional unit.

7. A fault-detecting digital data processing apparatus according to claim 6, wherein loopback error detection means comprises a loopback drive check means in each of the drive-side and check-side bus interface means for comparing the portion of a datum generated by the processing section and applied to the bus means by that bus interface means with a corresponding portion of the datum received from the bus means, and for signaling a bus error to the other functional units in the event of disparity between those compared portions.

8. A fault-detecting digital data processing apparatus according to claim 6, wherein each loopback error detection means comprises a loopback compare check means in each of the drive-side and check-side bus interface means for comparing the portion of a datum generated by the processing section and applied to the bus means by the other bus interface means with a corresponding portion of the datum received from the bus means, and for signaling a bus error to the other functional units in the event of disparity between those compared portions.

9. A fault-detecting digital data processing apparatus comprising

plural functional units, including one or more central processing units and one or more input/output interface units, plurality of bus means, each connected to and providing point-to-point communications between a respective pair of functional units,

each central processing unit including dual bus interface means, each of which applies a complementary portion of a datum to the bus means coupling that central processing unit to an input/output interface unit,

each input/output interface unit including dual bus interface means, each of which applies a complementary portion of a datum to the bus means coupling that input/output interface unit to a central processing unit,

the functional units each including

error phase means, coupled to the respective bus means of that functional unit, for placing the functional unit in an error isolation phase substantially concurrently with the other functional units, and for transmitting test data onto the bus means during a respective portion of that phase and exclusive, with respect to that portion, of any other unit connected to that bus,

bus error detecting means, coupled to the respective bus means of that functional unit, for detecting a communication error, including any of parity error, an error correction code error and a loopback error, on that bus means, and for signalling a bus error to the other functional units in the event of such communication error,

error isolation means, coupled to the bus error detecting means and to the loopback error detecting means, for signaling the other functional units during the error isolation phase that the respective functional unit is faulty based on

(i) whether the bus error means of that functional unit detected a loopback error, (ii) whether the bus error means of another functional unit signalled a bus error in response to test data transmitted during the error isolation phase, and (iii) whether another functional unit signaled that it was faulty.

10. A fault-detecting digital data processing apparatus according to claim 9, wherein the input/output interface units transmit information to and from PCI compatible devices.

11. A fault-detecting digital data processing apparatus according to claim 9, wherein the error phase means are responsive to a bus error signaled by any of the functional units for placing the respective functional units in an error phase.

12. A fault-detecting digital data processing apparatus according to claim 9, wherein the error isolation means includes means for taking off-line a functional unit that signals that it is faulty.

13. A fault-detecting digital data processing apparatus according to claim 9, wherein the bus error detecting means of at least one functional unit comprises loopback error detection means, coupled to the respective bus means and error phase means, for comparing test data transmitted onto that bus means with data received substantially concurrently from the bus means, and for signaling a bus error to the other functional units in the event of discrepancy between the transmitted and received data.

14. A method of operating a digital data processor of the type having plural functional units, the method comprising

providing point-to-point communications between respective pairs of functional units, selectively placing each functional unit in an error isolation phase substantially concurrently with each other functional unit,

during the error isolation phase, transmitting test data from each function unit onto a respective bus during a respective portion of that phase and exclusive, with respect to that portion, of any other unit connected to that bus,

with each functional unit, monitoring the respective bus means for detecting a communication error, including any of parity error, an error correction code error and a loopback error, on that bus means, and signaling a bus error to the other functional units in the event of such communication error,

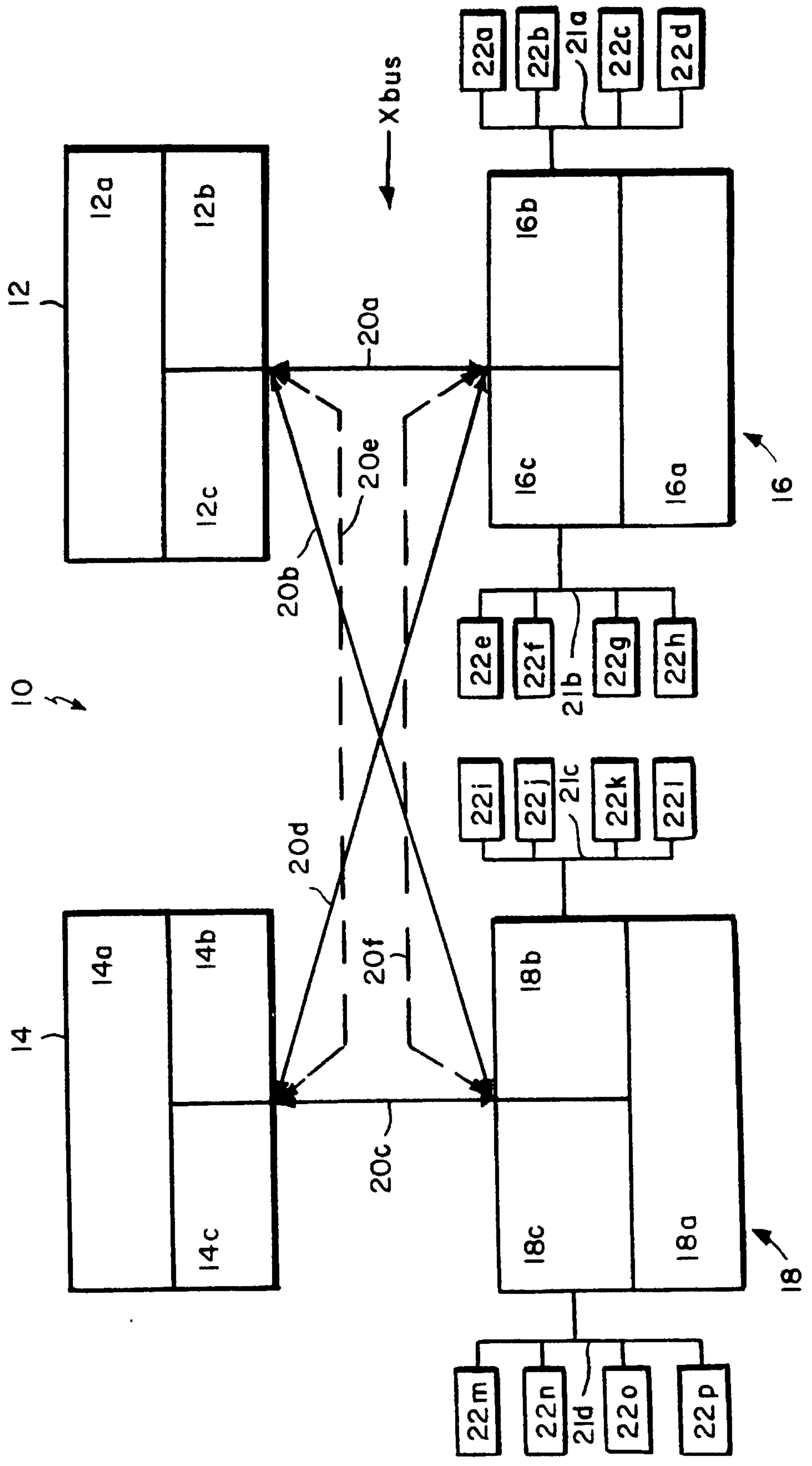
signaling the other functional units that the respective functional unit is faulty based on (i) whether the bus error means of that functional unit detected a loopback error, (ii) whether the bus error means of another functional unit signalled a bus error in response to test data transmitted during the error isolation phase, and (iii) whether another functional unit signaled that it was faulty.

15. A method according to claim 14, including the step of placing each functional unit in the error isolation phase in response to a bus error signaled by any of the functional unit.

16. A method according to claim 14, including the step of taking off-line a functional unit that has signalled that it is faulty.

17. A method according to claim 16, including the step of detecting a loopback error by test data transmitted onto the respective bus means of a functional unit with data received substantially concurrently from that bus means, and for signaling a bus error

to the other functional units in the event of discrepancy between the transmitted and received data.



10
12
12a
12b
12c
14
14a
14b
14c
16
16a
16b
16c
18
18a
18b
18c
20a
20b
20c
20d
20e
20f
Xbus
21a
21b
21c
21d
22a
22b
22c
22d
22e
22f
22g
22h
22i
22j
22k
22l
22m
22n
22o
22p

FIG. 1

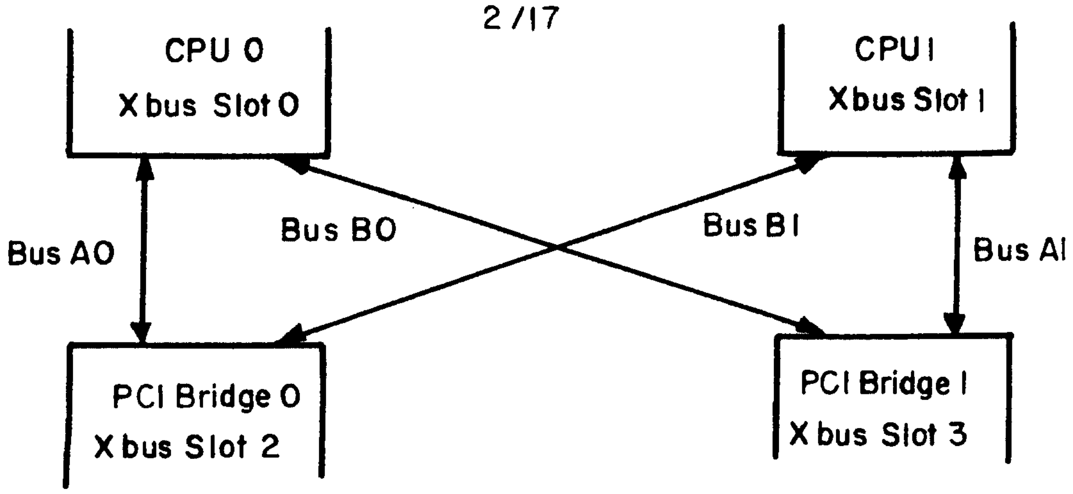

CPU 0
Xbus Slot 0
CPU 1
Xbus Slot 1
Bus A0
Bus B0
Bus B1
Bus A1
PCI Bridge 0
Xbus Slot 2
PCI Bridge 1
Xbus Slot 3

FIG. 2

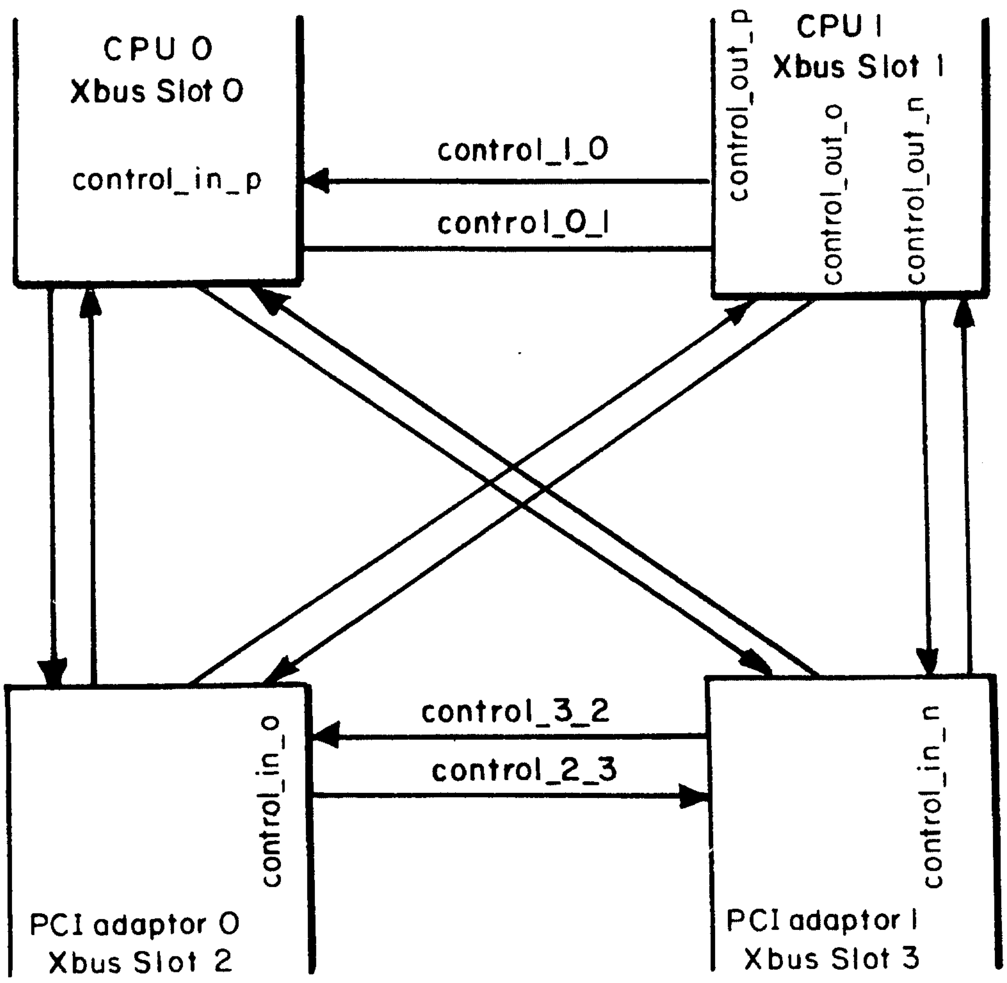

CPU 0
Xbus Slot 0
control_in_p
CPU 1
Xbus Slot 1
control_out_p
control_out_o
control_out_n
control_1_0
control_0_1
control_3_2
control_2_3
control_in_o
control_in_n
PCI adaptor 0
Xbus Slot 2
PCI adaptor 1
Xbus Slot 3

FIG. 3

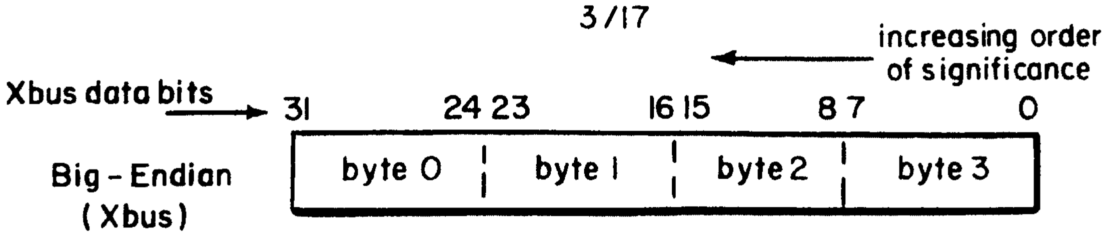

increasing order of significance
Xbus data bits
31
24 23
16 15
8 7
0
Big-Endian (Xbus)
byte 0
byte 1
byte 2
byte 3

FIG. 4

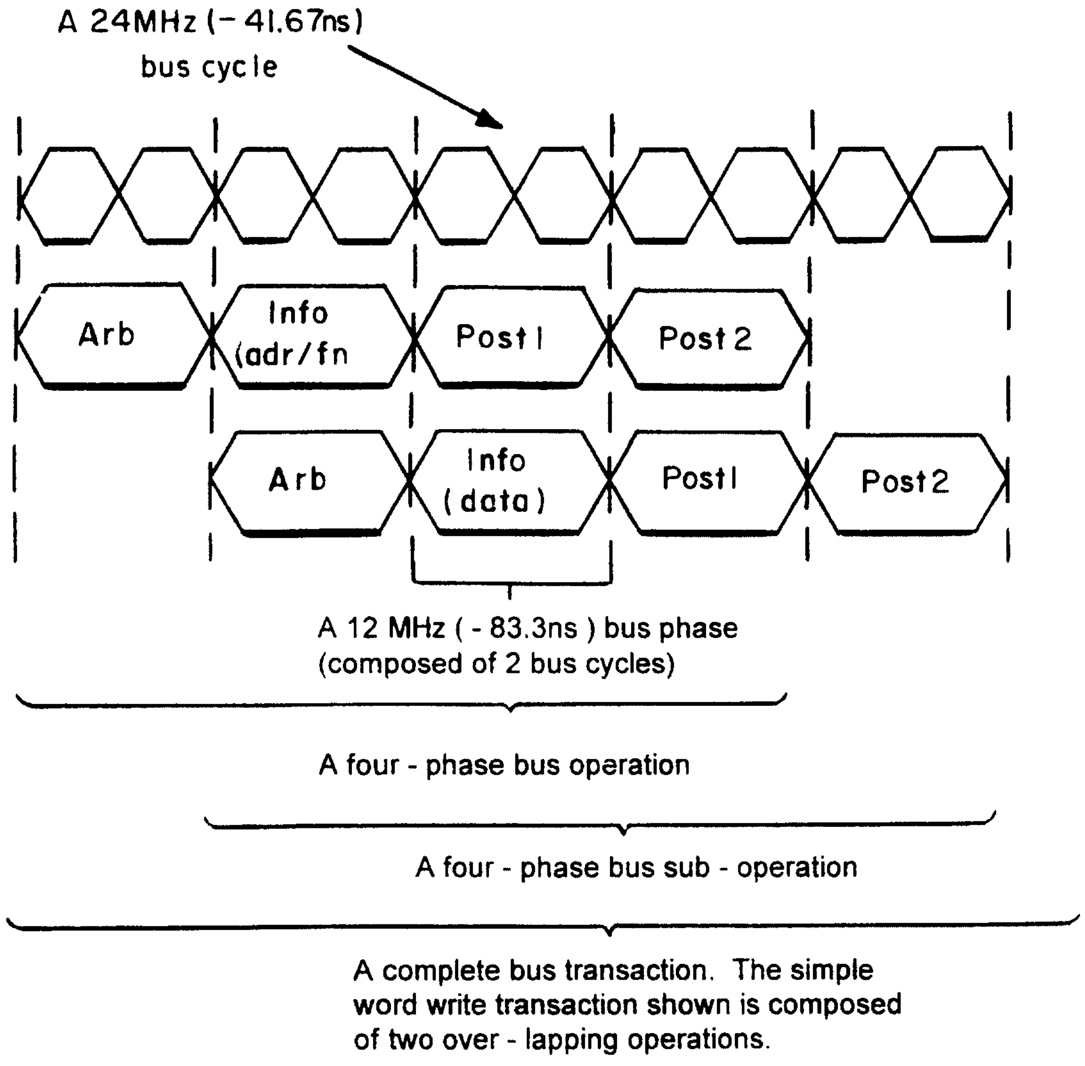
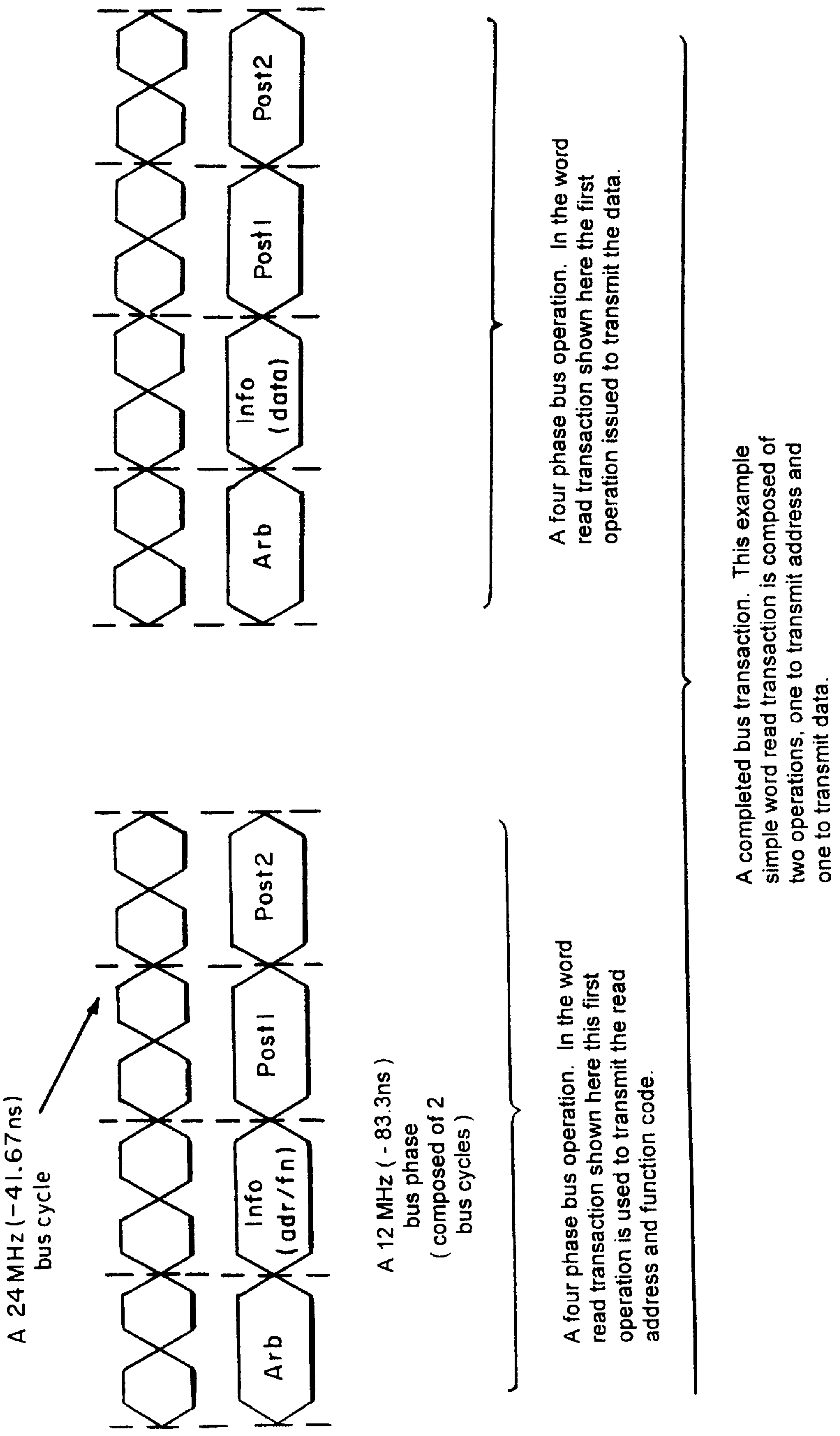
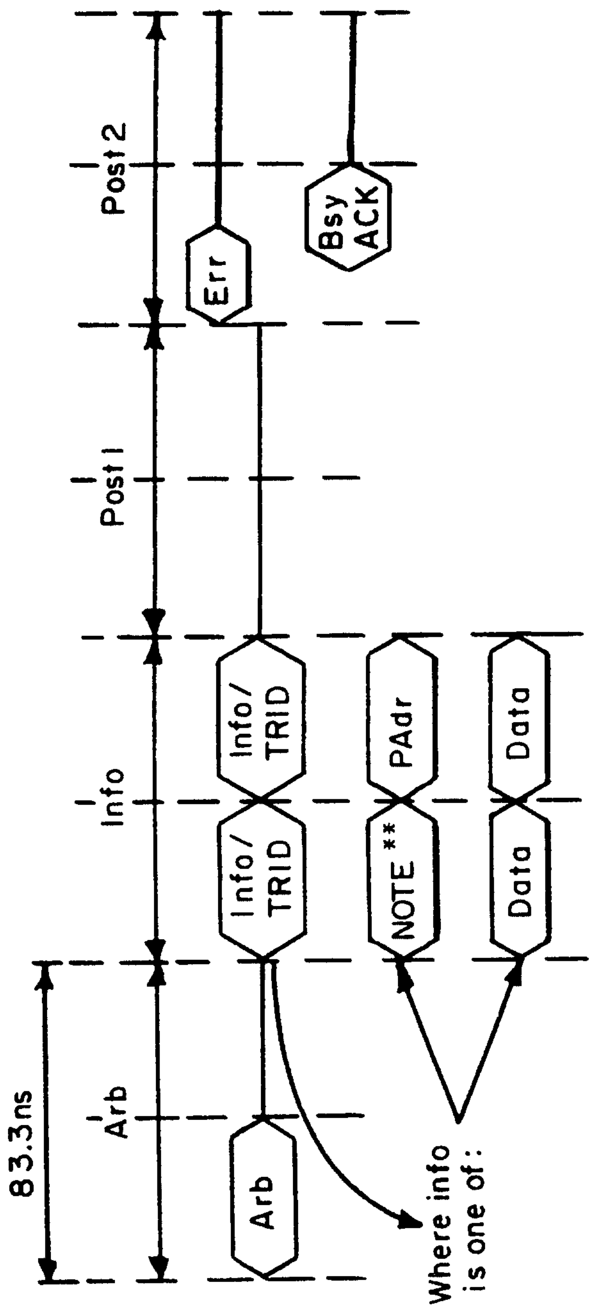

A 24MHz (~41.67ns) bus cycle
Arb
Info (adr/fn
Post 1
Post 2
Arb
Info (data)
Post1
Post2
A 12 MHz (~ 83.3ns) bus phase (composed of 2 bus cycles)
A four - phase bus operation
A four - phase bus sub - operation
A complete bus transaction. The simple word write transaction shown is composed of two over - lapping operations.

FIG. 6



A 24 MHz (~41.67ns) bus cycle
A 12 MHz (~83.3ns) bus phase (composed of 2 bus cycles)
Arb
Info (adr/fn)
Post1
Post2
A four phase bus operation. In the word read transaction shown here this first operation is used to transmit the read address and function code.
Arb
Info (data)
Post1
Post2
A four phase bus operation. In the word read transaction shown here the first operation issued to transmit the data.
A completed bus transaction. This example simple word read transaction is composed of two operations, one to transmit address and one to transmit data.

FIG. 5



** virtual index, function code, remote / coherent bits, and byte enables or I/O

FIG. 7

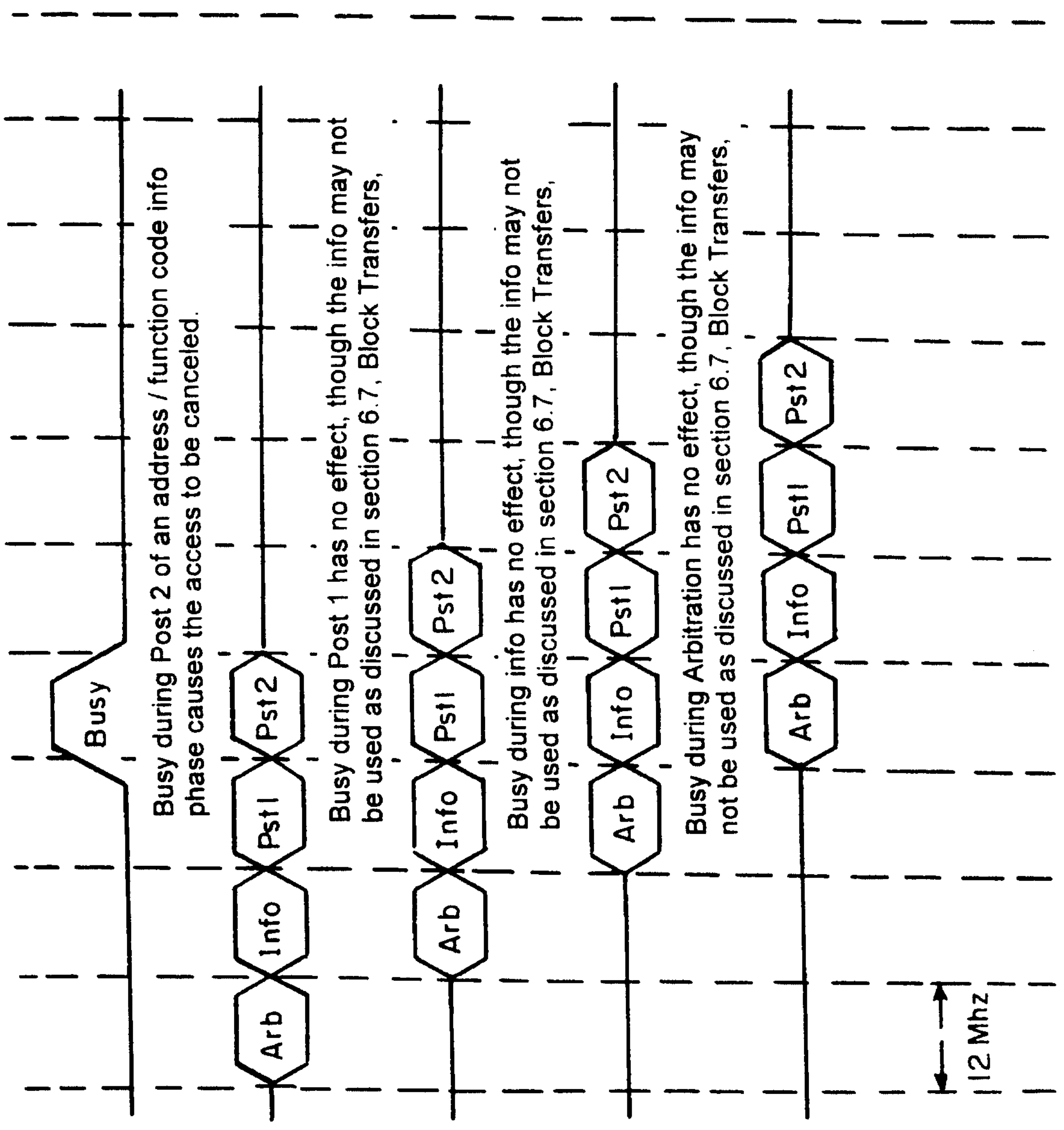

Busy
Arb
Info
Pst1
Pst2
Busy during Post 2 of an address / function code info phase causes the access to be canceled.
Arb
Info
Pst1
Pst2
Busy during Post 1 has no effect, though the info may not be used as discussed in section 6.7, Block Transfers,
Arb
Info
Pst1
Pst2
Busy during info has no effect, though the info may not be used as discussed in section 6.7, Block Transfers,
Arb
Info
Pst1
Pst2
Busy during Arbitration has no effect, though the info may not be used as discussed in section 6.7, Block Transfers,
Arb
Info
Pst1
Pst2
12 Mhz

FIG. 8

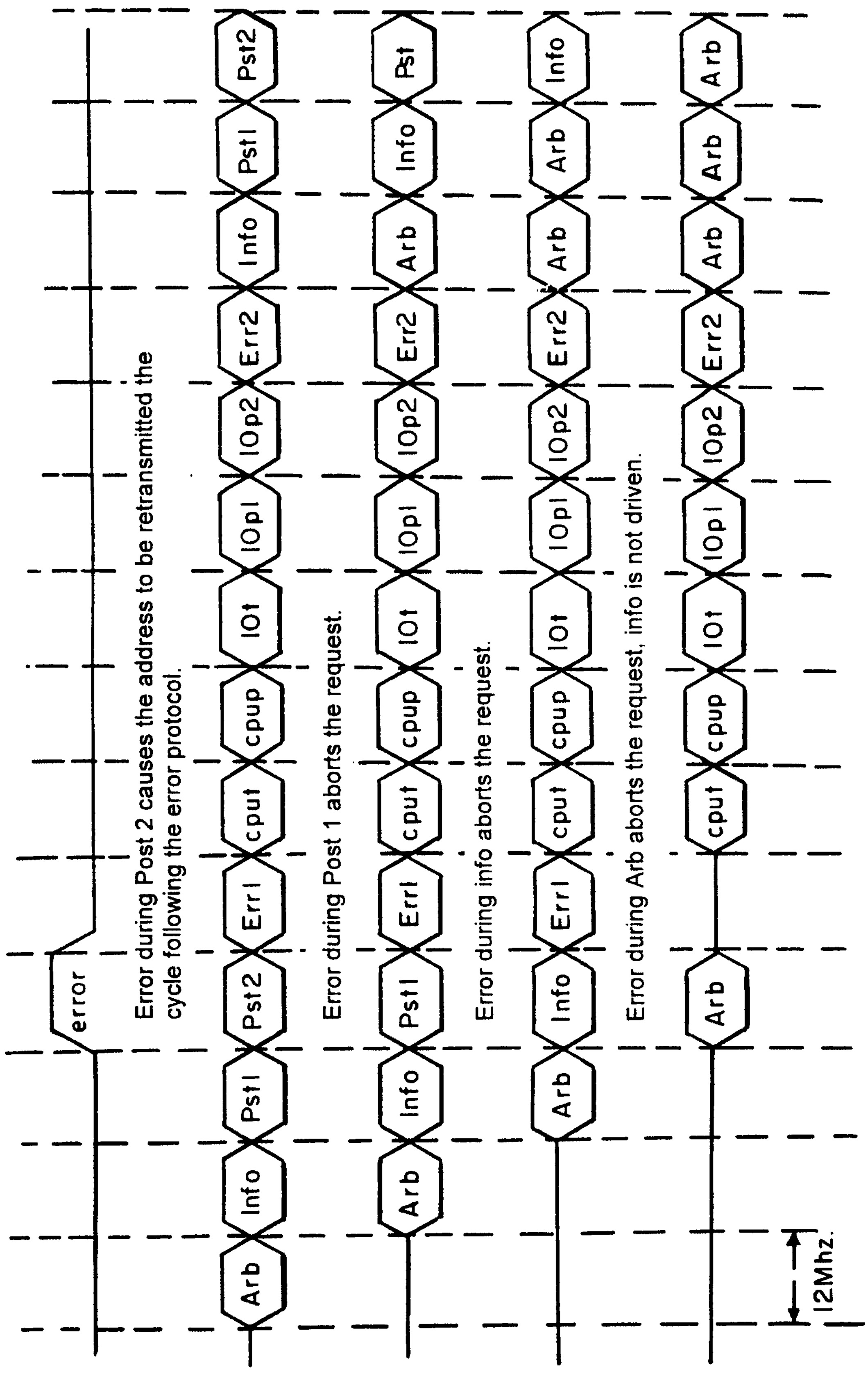

error
Error during Post 2 causes the address to be retransmitted the cycle following the error protocol.
Arb
Info
Pst1
Pst2
Err1
cput
cpup
IOt
IOp1
IOp2
Err2
Info
Pst1
Pst2
Error during Post 1 aborts the request.
Arb
Info
Pst1
Err1
cput
cpup
IOt
IOp1
IOp2
Err2
Arb
Info
Pst
Error during info aborts the request.
Arb
Info
Err1
cput
cpup
IOt
IOp1
IOp2
Err2
Arb
Arb
Info
Error during Arb aborts the request, info is not driven.
Arb
cput
cpup
IOt
IOp1
IOp2
Err2
Arb
Arb
Arb
12Mhz.

FIG. 9

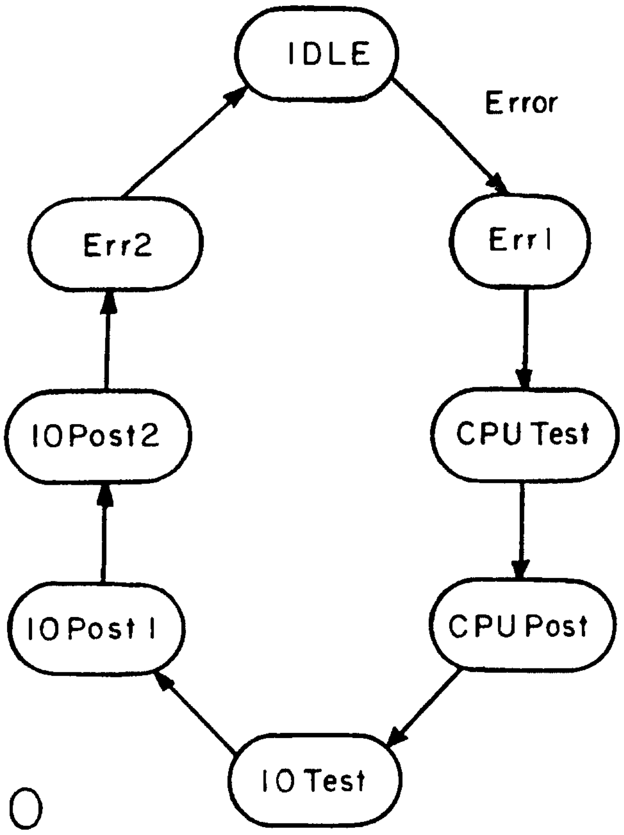

IDLE
Error
Err1
CPU Test
CPU Post
IO Test
IOPost 1
IOPost2
Err2

FIG. 10

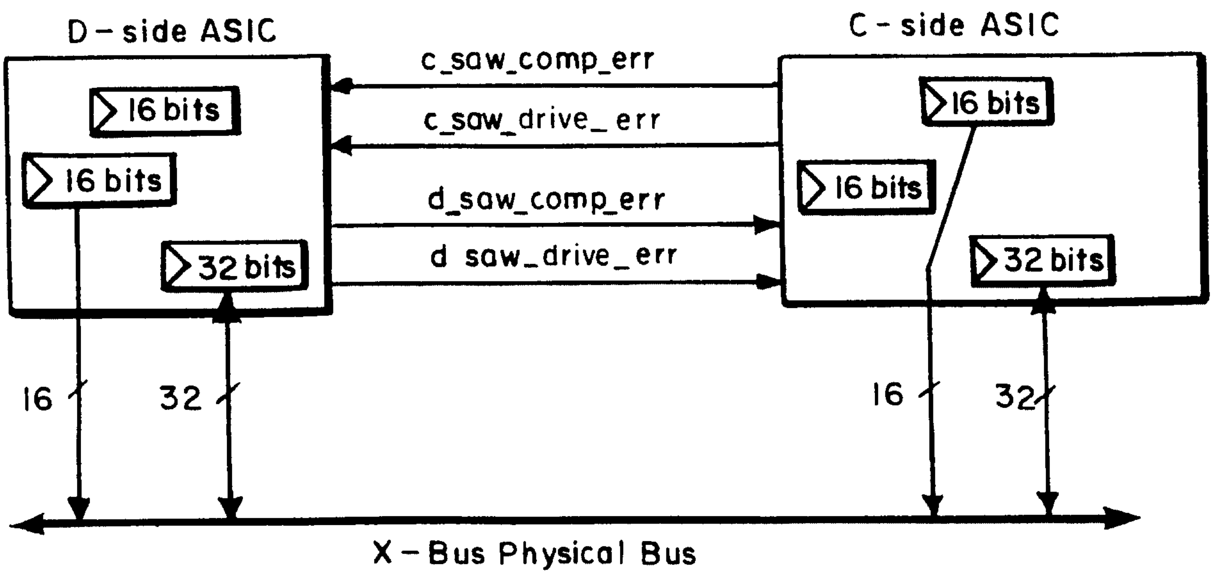

D-side ASIC
16 bits
16 bits
32 bits
c_saw_comp_err
c_saw_drive_err
d_saw_comp_err
d saw_drive_err
C-side ASIC
16 bits
16 bits
32 bits
16
32
16
32
X-Bus Physical Bus

FIG. 13

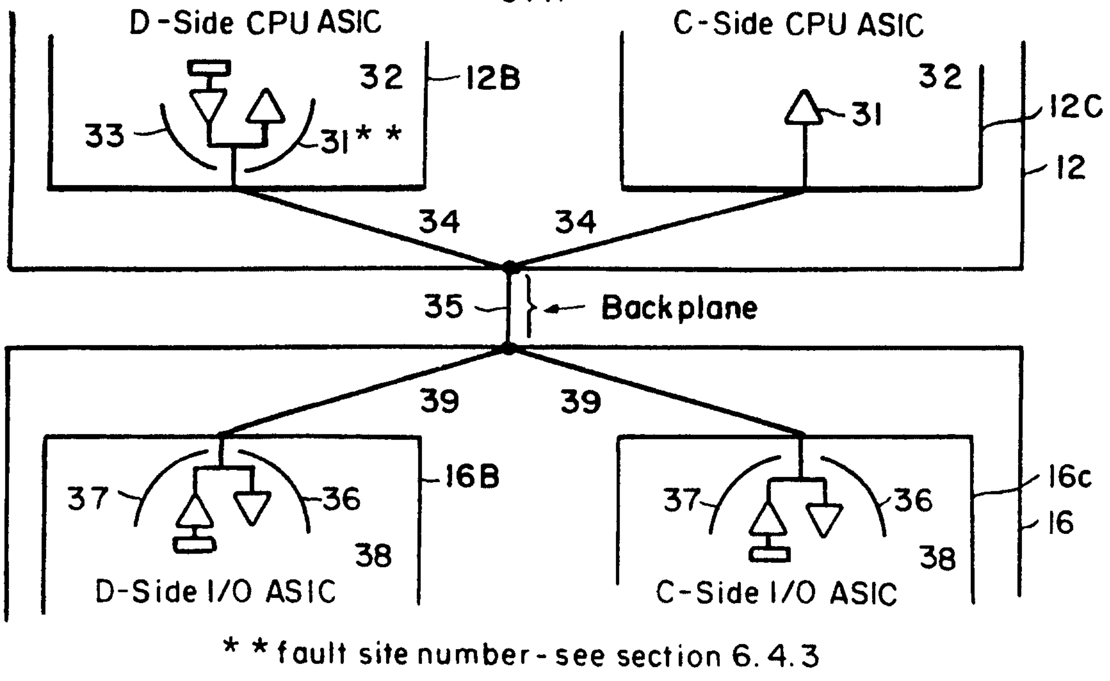

D-Side CPU ASIC
C-Side CPU ASIC
32
12B
32
33
31 * *
31
12C
12
34
34
35
Backplane
39
39
37
36
16B
37
36
16c
16
38
38
D-Side I/O ASIC
C-Side I/O ASIC
* * fault site number - see section 6.4.3

FIG. 11

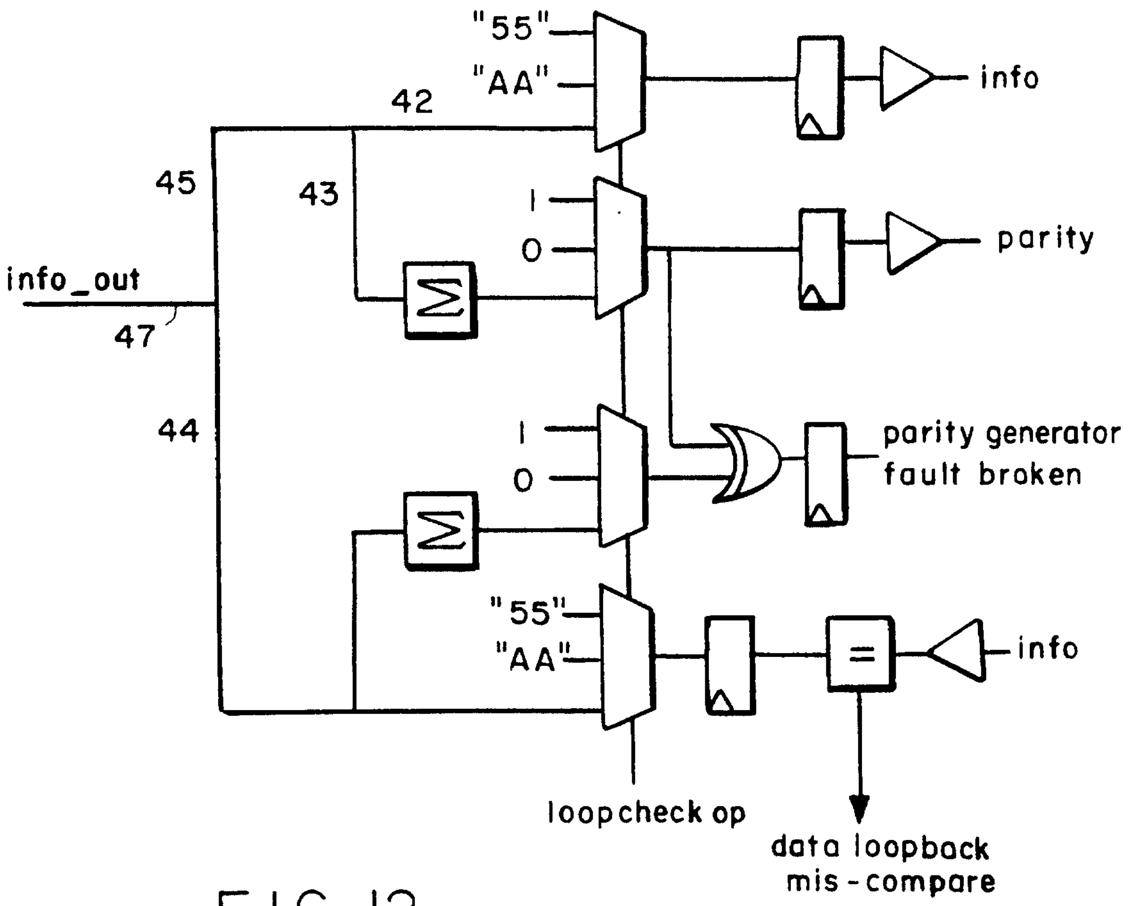

"55"
"AA"
info
42
45
43
1
0
parity
info_out
47
44
1
0
parity generator fault broken
"55"
"AA"
info
loopcheck op
data loopback mis-compare

FIG. 12

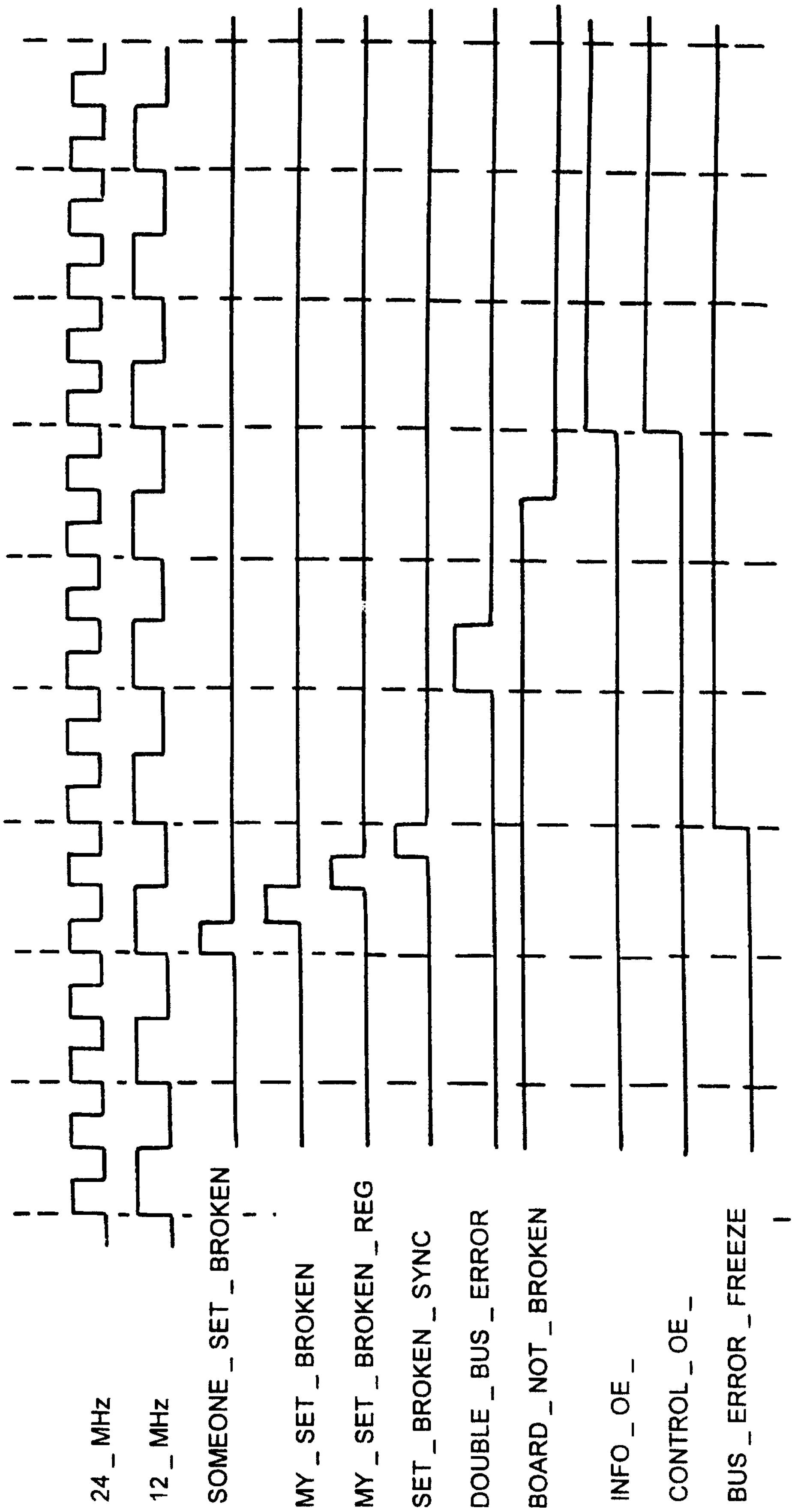

24 _ MHz
12 _ MHz
SOMEONE _ SET _ BROKEN
MY _ SET _ BROKEN
MY _ SET _ BROKEN _ REG
SET _ BROKEN _ SYNC
DOUBLE _ BUS _ ERROR
BOARD _ NOT _ BROKEN
INFO _ OE _
CONTROL _ OE _
BUS _ ERROR _ FREEZE

FIG. 14

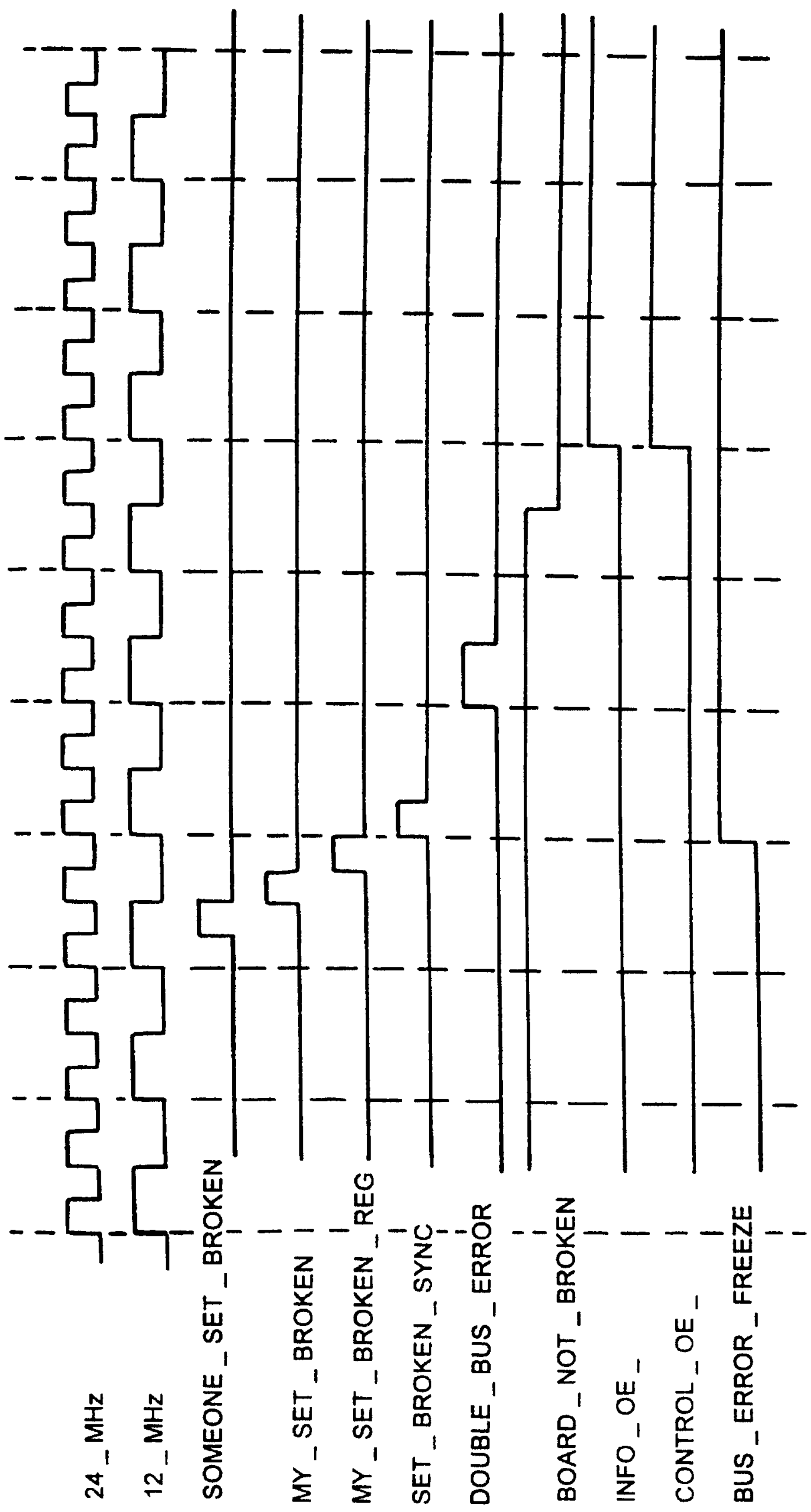

24_MHz
12_MHz
SOMEONE_SET_BROKEN
MY_SET_BROKEN
MY_SET_BROKEN_REG
SET_BROKEN_SYNC
DOUBLE_BUS_ERROR
BOARD_NOT_BROKEN
INFO_OE_
CONTROL_OE_
BUS_ERROR_FREEZE

FIG. 15

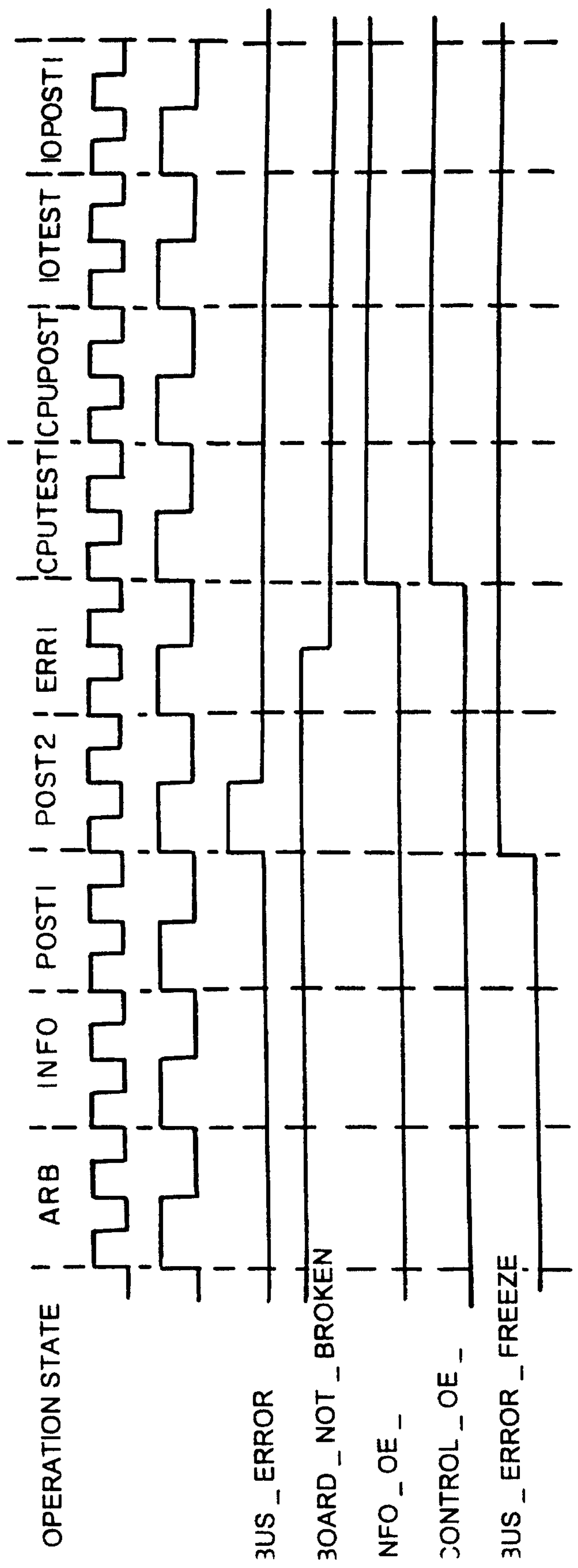

OPERATION STATE
ARB
INFO
POST1
POST2
ERR1
CPUTEST1
CPUPOST
IOTEST
IOPOST1
3US _ ERROR
3OARD _ NOT _ BROKEN
NFO _ OE _
CONTROL _ OE _
3US _ ERROR _ FREEZE

FIG. 16

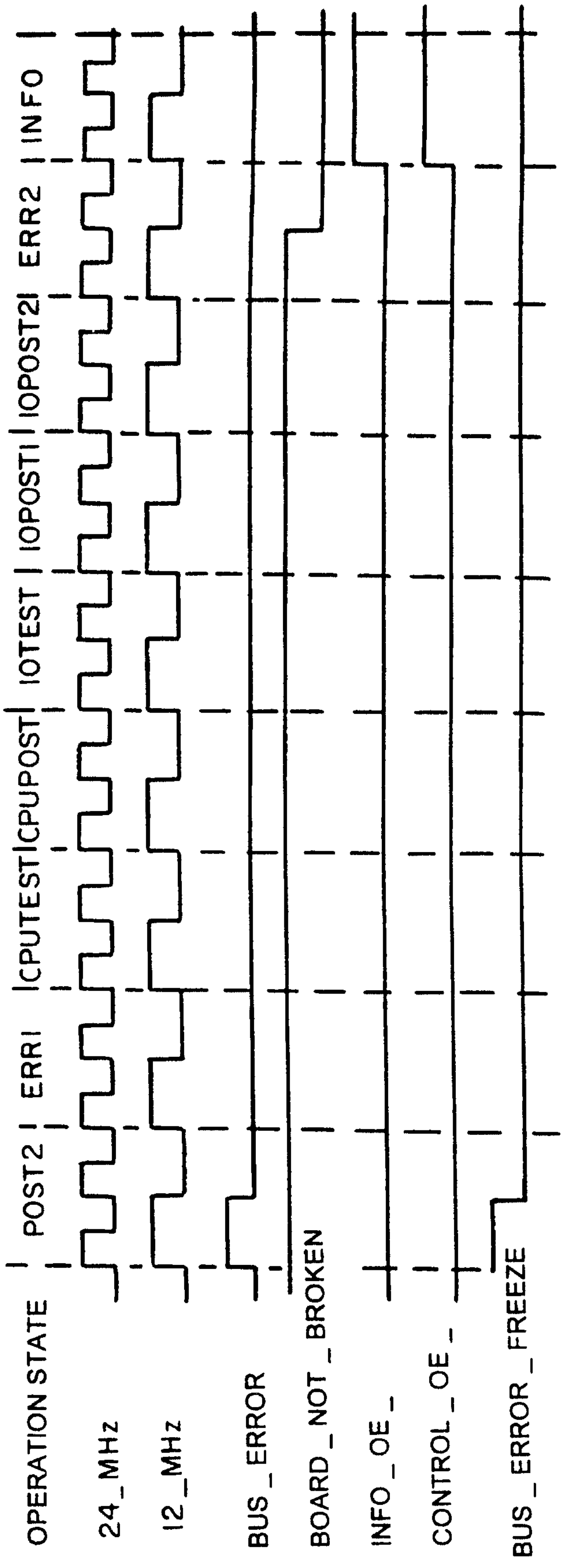
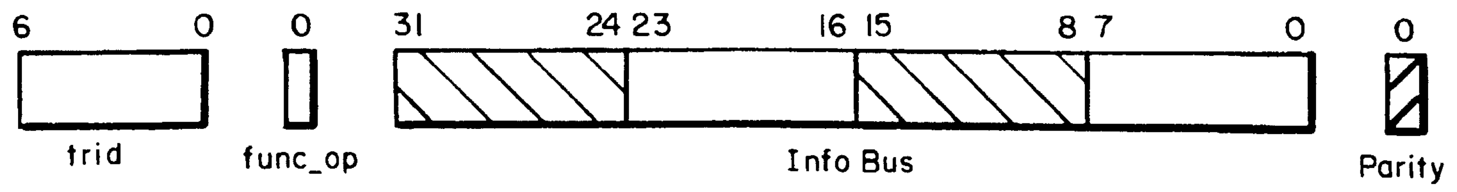

OPERATION STATE
24_MHz
12_MHz
BUS_ERROR
BOARD_NOT_BROKEN
INFO_OE_
CONTROL_OE_
BUS_ERROR_FREEZE
POST2
ERR1
CPUTEST
CPUPOST
IOTEST
IOPOST1
IOPOST2
ERR2
INFO

FIG. 17



Bits are assigned starting from the least significant bit.

▧ Driven by the D - side ASIC during normal accesses, and by the driving ASIC during single - side accesses.

□ Driven by the C - side ASIC during normal accesses, and by the driving ASIC during single - side accesses.

▨ Parity is driven by the D - side on the CPU.
It is driven by the D - side of the PCIB during normal accesses, and by the driving ASIC during Single Side accesses.

FIG. 18

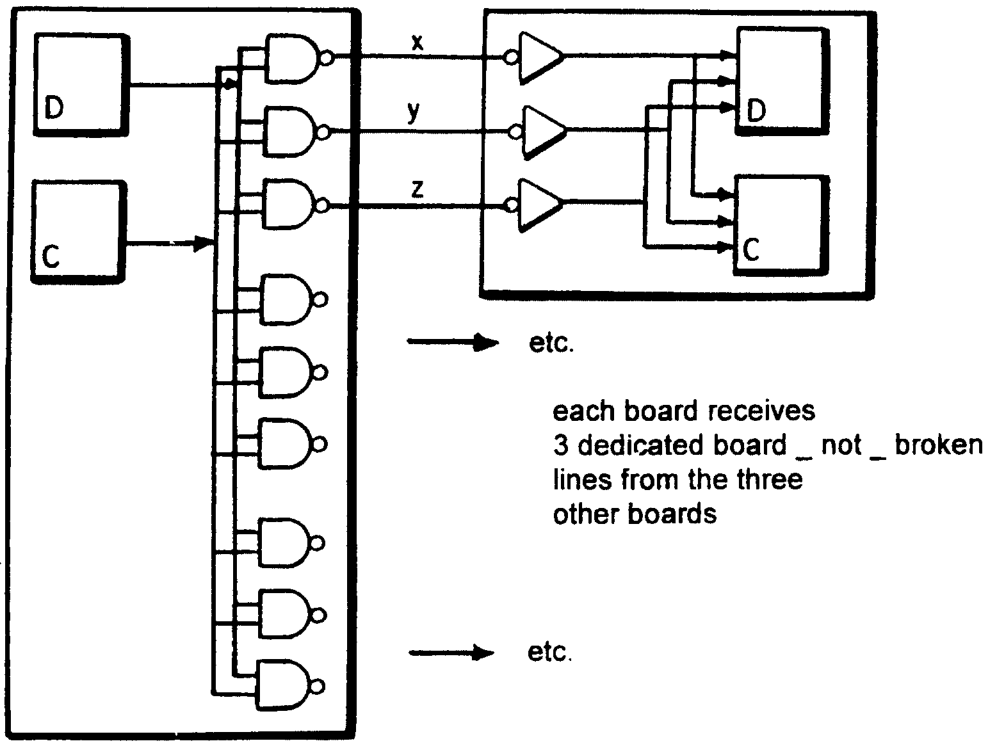

D
C
x
y
z
D
C
etc.
each board receives
3 dedicated board _ not _ broken
lines from the three
other boards
etc.

FIG. 19

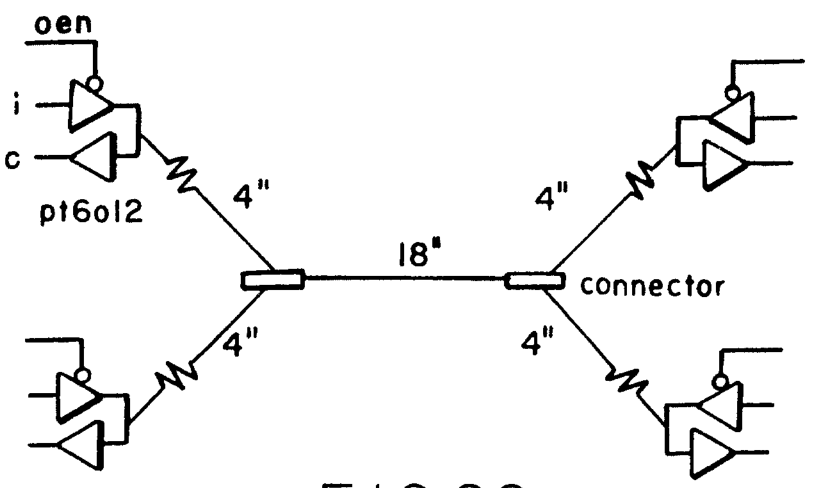


FIG.20

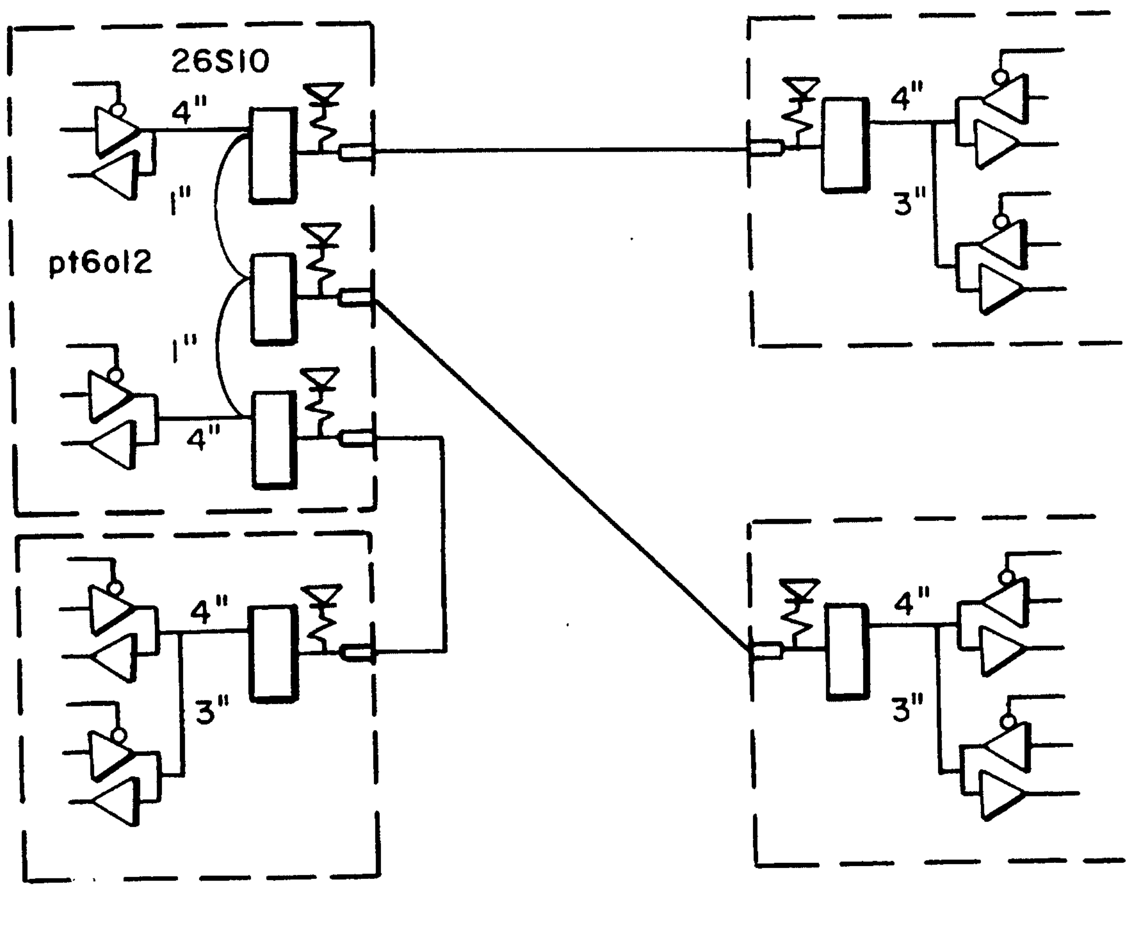


FIG.21

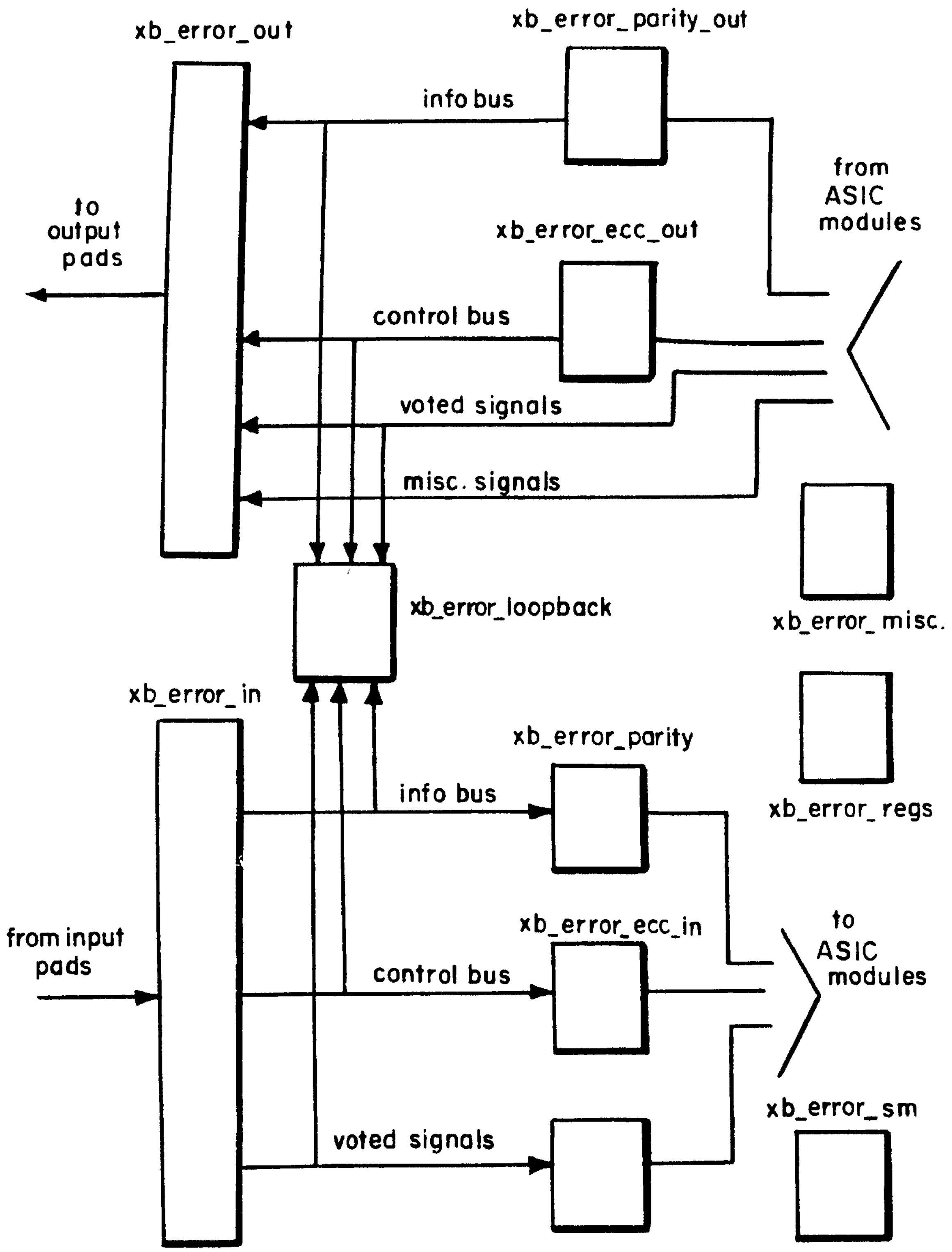
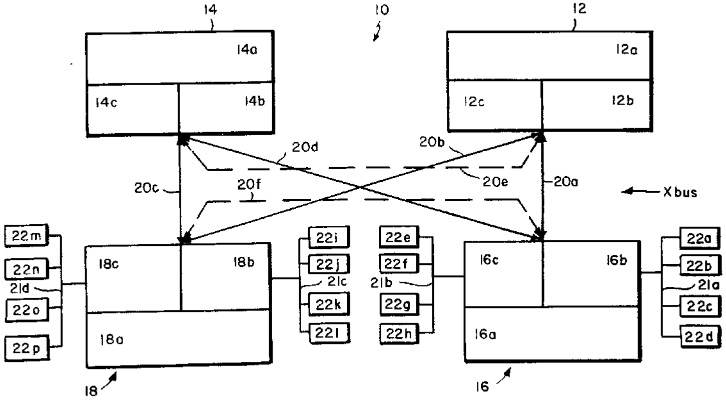

xb_error_out
xb_error_parity_out
info bus
from ASIC modules
to output pads
xb_error_ecc_out
control bus
voted signals
misc. signals
xb_error_loopback
xb_error_misc.
xb_error_in
xb_error_parity
xb_error_regs
info bus
from input pads
xb_error_ecc_in
to ASIC modules
control bus
xb_error_sm
voted signals

FIG. 22



14
10
12
14a
12a
14c
14b
12c
12b
20d
20b
20e
20c
20f
20a
Xbus
22m
22n
21d
22o
22p
18c
18b
18a
22i
22j
21c
22k
22l
22e
22f
21b
22g
22h
16c
16b
16a
22a
22b
21a
22c
22d
18
16