C38xx CPU Utilities Specification

Document No. 417-001245-X00

Revision 1.0
November 5, 1991

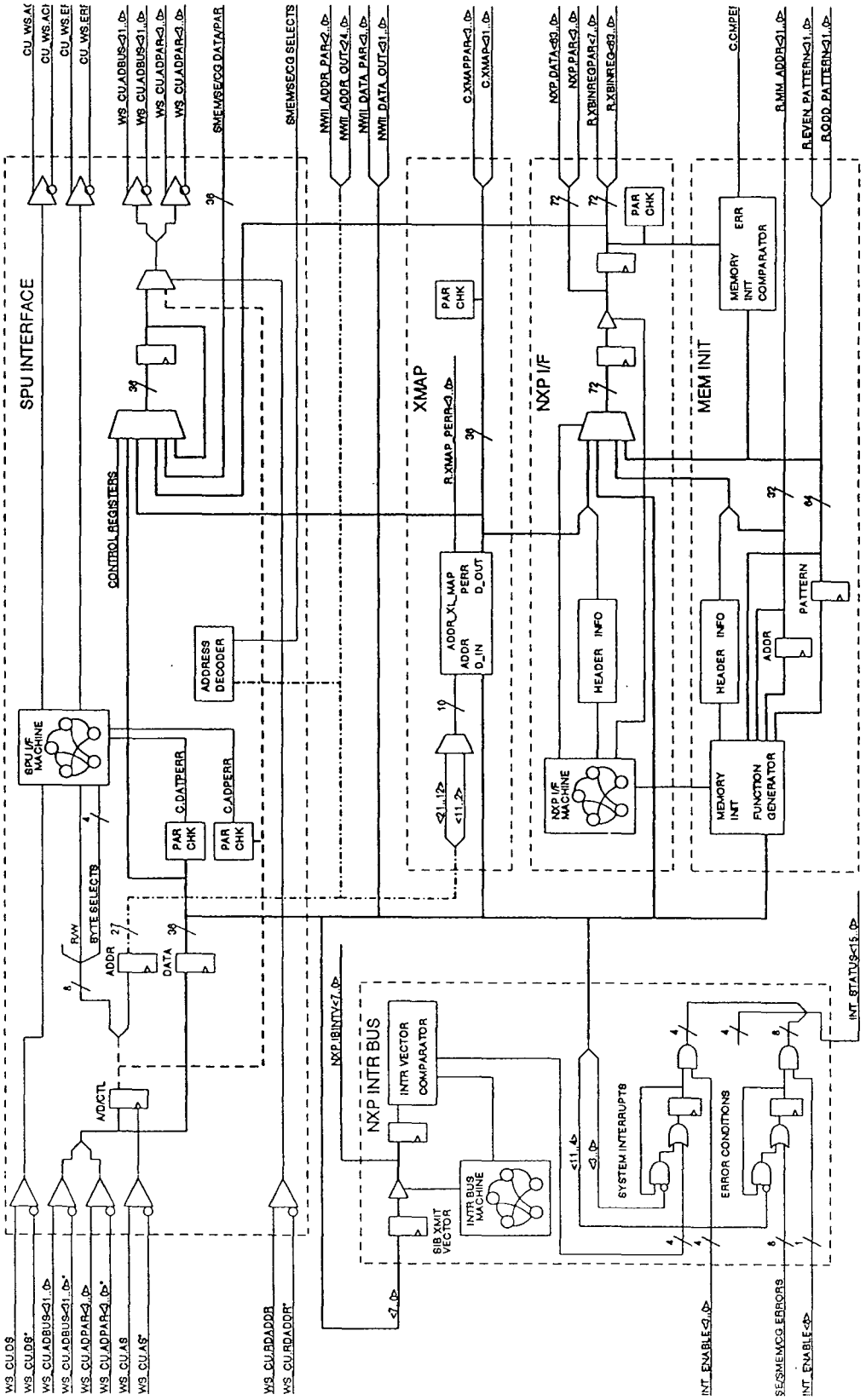
INTERNAL USE ONLY

PROPRIETARY DISCLAIMER

List of Figures

List of Tables

Figure 2-1 NWII Block Diagram



2.2 SPU Interface

2.2.1 NWI-NCU Control Signals

As previously mentioned, the NWI drives several control signals to the NCU. The address strobe (WS_CU.AS) clocks the ADBUS into the ADCTL register. The data strobe (WS_CU.DS) initiates a transaction. The read address strobe (WS_CU.RDADDR) puts the data path into a diagnostic mode. In addition to these, the SPU also provides a reset line (WS_CU.RESET), which resets the NWII portion of the NCU, putting it into a known state prior to the first SPU access.

In the other direction, the NWII drives three control signals on the cable. The first two, CU_WS.ACK and CU_WS.ERR, indicate successful and unsuccessful transactions to the SPU, respectively. These signals are handshake lines with the SPU. The other control signal is an interrupt request to the NWI (CU_WS.INT). Assertion of this last signal indicates to the SPU that one or more of the NWII interrupts is currently asserted and enabled.

2.2.2 Typical Transaction Sequence

A typical transaction is preceded by the loading of the ADCTL register, and begins with the synchronization of the data strobe from the NWI. The synchronized data strobe (R.DS3) drives the PAL equations which implement the SPU I/F state machine. Upon recognizing a data strobe from the NWI, the SPU I/F state machine checks the parity of the information stored in the ADCTL register. A parity error results in a transition from the IDLE state to the BERR state. Otherwise, there is a transition from the IDLE state to either the READ or WRITE states, determined by the read/write bit from the ADCTL register (bit 28).

In the WRITE state, the ADBUS is loaded into the SPU_DIN register and parity is checked. A parity error results in a transition to the BERR state. As soon as the write has been completed, the state machine moves to the ACK state. Except for accesses through the NXP interface, all writes are for a fixed number of clock cycles. For NXP interface accesses, the state machine must wait until the NXP transfer to the NIA has completed.

In the READ state, the contents of the SPU_DIN register remain unchanged. NXP interface reads must wait until the NXP transfer from the NIA has completed. All other reads are for a fixed number of clocks. After completing the read, the SPU I/F state machine proceeds either to the BERR or ACK states, depending on whether or not an error was detected.

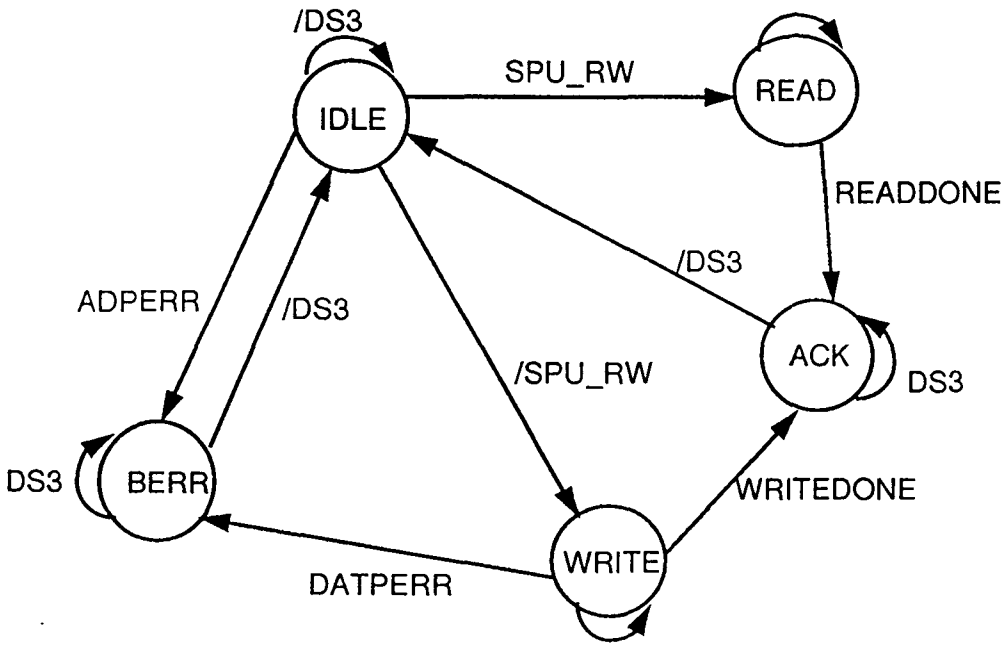
Once in either the ACK or BERR states, the NWII generates a response to the SPU. The response is either an acknowledge (in the ACK state) or an error indicator (in the BERR state). The acknowledge indicator is WS_CU.ACK, and the error indicator is WS_CU.ERR. The NWII remains in the particular state until the data strobe is negated by the SPU. The NWII then returns to the IDLE state to await the next access from the SPU.

Figure 2 shows the transaction sequence. Not all paths are shown, nor are all of the signals/terms provided for every path. Consult the schematics and PAL equations for more detailed information.

2.2.3 Data Strobe Masking

In the case of a write to the Scan Engine Command Register, the data strobe is masked off temporarily. At a suitable point after memory refresh, the scan operation begins, and the data strobe masking is removed. Any pending data strobe is then synchronized, and the next SPU access begins.

Figure 2-2 SPU I/F State Diagram



2.2.4 Diagnostic Transactions

2.2.4.1 Address Only Transfers

This type of diagnostic access does not require any transitions through the SPU I/F state machine. During an Address Only transfer, the NWI places address and control information on the cable, and then asserts the address strobe (WS_CU.AS). The information is clocked into the ADCTL register, and the cycle terminates. Note that their is no data transfer phase associated with this type of access, and there is no handshaking on the cable between the SPU and the NCU. The only effect is to load the ADCTL register, and the SPU does not receive any indication as to the success of this operation.

2.2.4.2 Read Address Transfers

The purpose of this diagnostic access is to return the contents of the ADCTL register to the SPU. To initiate this transfer, the SPU asserts a cable control signal called WS_CU.RDADDR. The assertion of this signal configures the read path through the NWII such that the ADCTL register drives the cable ADBUS. At the same time, the SPU I/F state machine returns an acknowledge to inform the SPU that the cable data is valid. The acknowledge remains asserted until WS_CU.RDADDR is negated.

The combination of an Address Only transfer and a Read Address transfer allows the SPU to pattern test the cable while requiring a minimum of NCU logic to be operating. In particular, these transactions are implemented by combinational logic only, and do not require that clocks be supplied to the NCU.

2.3 NXP Interface

The Neptune Expansion Port (NXP) is an ECL version of the existing PBUS architecture on C1's and C2's. The port can run at up to 250 MB/sec, utilizing a 64-bit data path, and is controlled by the NIA. The NXP I/F portion of the NWII provides access to this port. In conjunction with the NXP Address Translation Map, the SPU can access any location in main memory or I/O space through this interface. In conjunction with the Memory Initialization Logic, the SPU can pattern test main memory at nearly the full bandwidth of the NXP. The Address Translation Map and the Memory Initialization Logic will be discussed in more detail in later sections. This section will focus on how the NXP I/F conducts transfers to/from the NIA across the NXP.

2.3.1 NXP Signals

The NXP is implemented with the following set of signals:

Table 1: NXP Signals

Signal Name	Input/Output	Comments
NXP.DATA<63.0>	Bidirectional	64-bit data bus
NXP.PAR<7.0>	Bidirectional	8-bits parity
XIOP0_IA.BUS_REQ	Output	Bus Request
XIOP_IA.CBUF_AVL	Output	Channel Buffer Available
XIOP_IA.WRT_VAL	Output	Write Valid
IA_XIOP.CLK_SYNC	Input	NXP Clock Synchronizer
IA_XIOP.BUS_ERR	Input	NXP Bus Error Indicator
IA_XIOP0.BUS_GNT	Input	Bus Grant
IA_XIOP.MBUF_AVL	Input	Memory Buffer Available
IA_XIOP.RD_VAL	Input	Read Valid

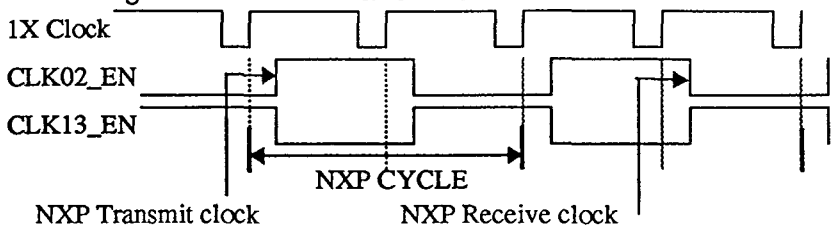
Notice that outputs from the NCU have signal names with the prefix XIOP_IA, while inputs to the NCU have signal names with the prefix IA_XIOP. Several signals have prefixes with XIOP0 instead of XIOP. These signals are routed individually to each node on the NXP, whereas the others are common to all nodes on the NXP.

2.3.2 NXP Clock Synchronization

The NXP I/F makes use of a pair of system 1X clock cycles for an NXP clock cycle. Outputs and bidirectional signals are only allowed to change on every other system clock cycle, while inputs are sampled on the clock cycles between output changes. Figure 2-3 shows the transmit (CLK02_EN) and the receive (CLK13_EN) clocks generated on the NCU and their relationship with the 1X system clock. To make sure that ports on the NXP are synchronized with respect to which clock cycles are for output changes, the NIA provides a signal called IA_XIOP.CLK_SYNC.

Assertion of this signal implies that the next clock is an output change clock edge. This signal is asserted periodically by the NIA

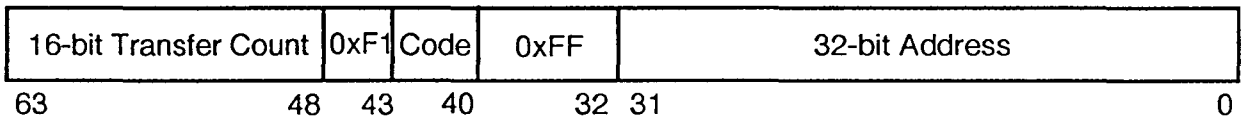
Figure 2-3 . NXP Transmit and Receive Clocks



2.3.3 NXP Transfer Sequencing

A basic NXP transfer consists of 4 states, *Header, Dead, Transfer* and *Done* states. To begin a transfer on the NXP, the NCU asserts the bus request signal, and then waits for the bus grant signal to be asserted by the NIA. Next, the NCU enters the Header state and sends a 64-bit header on the data bus, specifying the details of the requested transfer. This information includes the address to be accessed, the number of bytes to be transferred, and the type of operation being performed. The header fields are shown below:

Figure 2-4 NXP Header Fields



The *Transfer Count* can range from 1 to 64K, in bytes. SPU accesses to the NXP are never more than 4 bytes at most, but the Memory Initialization Logic can transfer a full 64 KB at one time. The *Code* field determines the operation to perform as follows:

Table 2: NXP Transfer Codes

Binary Code	Transfer Operation
000	Memory Write
001	Memory Read
010	TAS - Test and Set
011	TAC - Test and Clear
100	Memory Scrub
101	Illegal
110	I/O Read
111	NOP

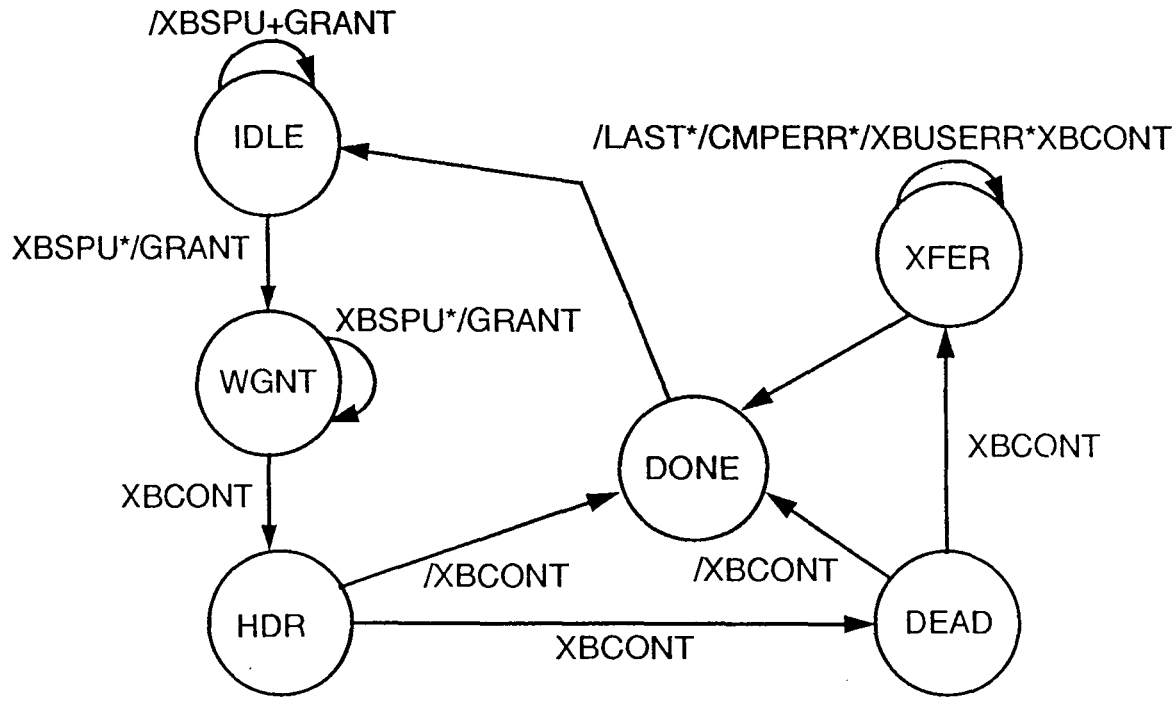
When the SPU accesses the NXP directly, the NXP Header fields are generated from information in the NXP Address Translation Map, combined with the read/write control bit from the ADCTL

register. Access by the Memory Initialization Logic can only result in *Memory Read* or *Memory Write* operations. The *Address* field can contain any valid memory or I/O space address.

After transferring the Header on the NXP, the next state is a *Dead* state which lasts one NXP clock cycle (an output change clock followed by an input sampling clock). Then begins the first data *Transfer* state. If the data on the bus is valid, it will be indicated by the assertion of either the *Read Valid* or *Write Valid* strobes as appropriate. If the receiving port is ready to accept the data, the *Channel Buffer Available* or *Memory Buffer Available* strobes are asserted as appropriate. A data transfer is valid if and only if it is accompanied by one of the *Valid* strobes and one of the *Available* strobes. A data transfer state which is not valid is followed by another data transfer state.

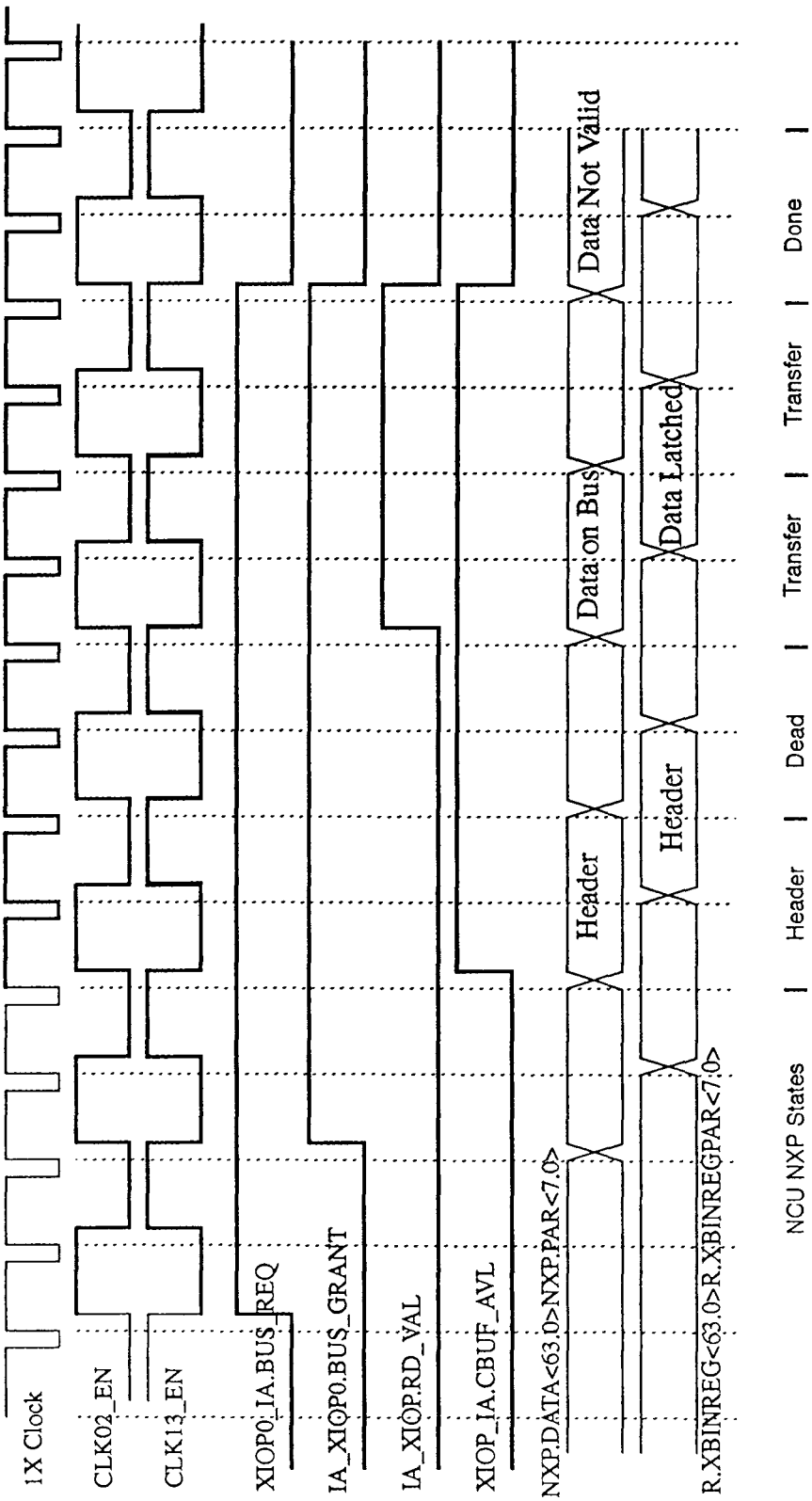
After a valid data transfer, the next state is determined by the Transfer Count. If not all the data has been transferred, the next cycle is another transfer state. If all the data has been transferred, the cycle is complete. On the NCU, the NXP I/F state machine then proceeds to the *Done* state. If the transfer was a write, the IA_XIOP.BUS_ERR signal must be checked to ensure that transfer was successful. The NXP I/F state machine then returns to the *Idle* state.

Figure 2-5 NXP I/F State Diagram



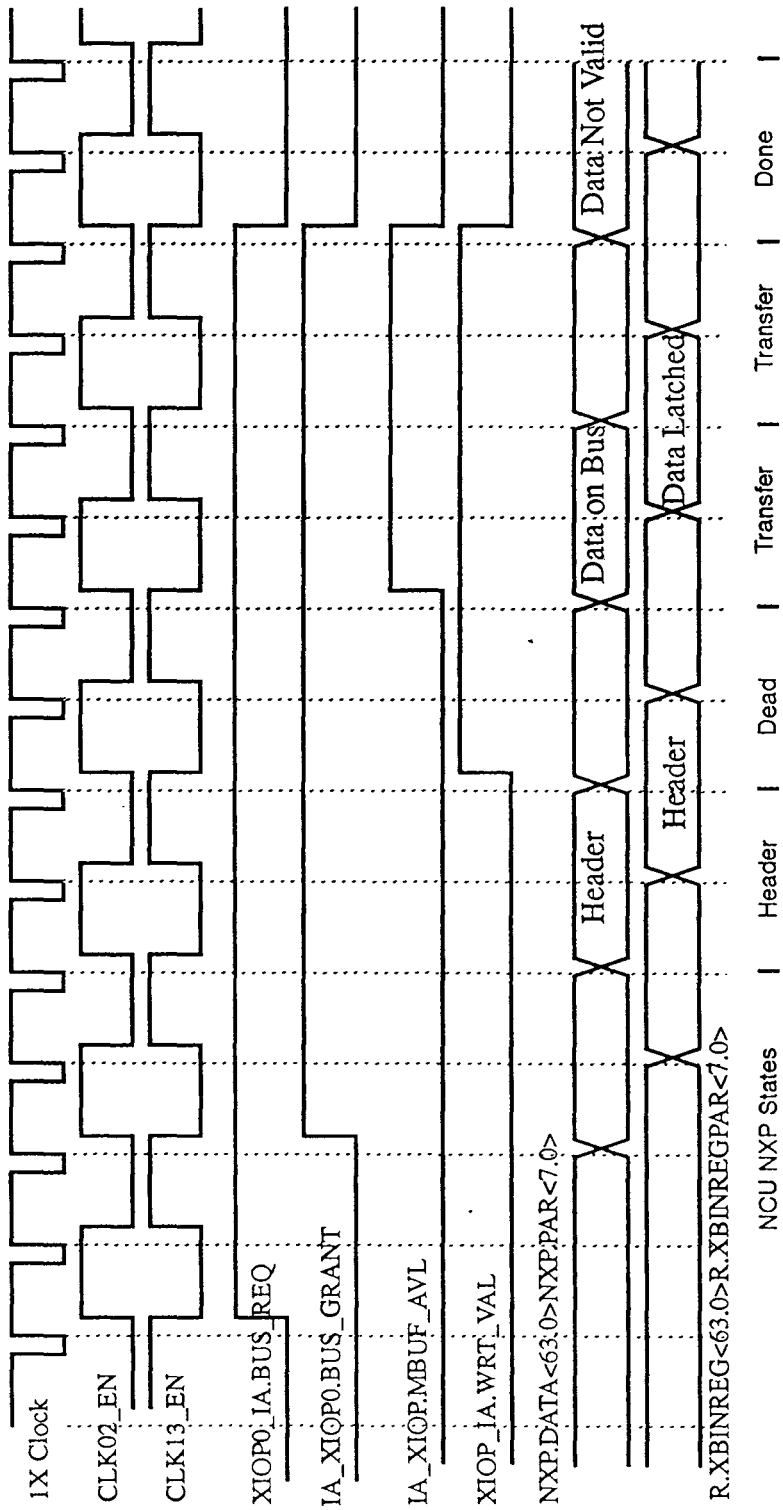
The above diagram shows the NXP I/F state diagram. Note that not all paths are shown, nor are all the signals/terms shown for any given path. Refer to the schematics and PAL equations for more detailed information. Figure 2-6 and Figure 2-7 show the timing of a NCU read and write cycle, and the various signals associated with that cycle.

Figure 2-6 .NCU NXP Read cycle



NXP Bus signals showing an NCU read transfer of 2 long words.

Figure 2-7 NCU NXP Write cycle



NXP Bus signals showing an NCU write transfer of 2 long words.

2.3.4 NXP Error Conditions

Many different situations on the NXP Bus could result in one of three error bits getting set. Some of these conditions will actually result in multiple bits being set. To help further determine what the exact cause of the error is, it will often be necessary to examine the state of the NIA which is the master of the NXP Bus. Each of the three error bits that relate to the NXP are described below.

2.3.4.1 NXP Busy

When the Memory Test/Initialization (MTST) logic is active, it assumes ownership of the NXP I/F to access main memory. During this time, the NXP I/F is not available to the SPU for direct access to main memory or the I/O space. Should the SPU attempt to access the NXP while the MTST is active, the NCU will return an error. At the same time, the NBSY bit of the Error_Log register (bit 3), will be set, indicating that the NXP was busy during the last SPU access.This bit remains set until the Error Log register is written by the SPU. When using the MTST logic, the SPU should poll the GO bit of the MTST_CTL register (bit 16) to determine when the current MTST operation has finished, and then proceed with direct accesses to the NXP.

2.3.4.2 NXP Bus Errors

There are a three conditions which can produce bus errors on the NXP port, illeagal header, parity errors and PCM errors. For all types of accesses, the NIA checks the header information for both content and parity. An invalid header for either reason results in a bus error (NIA asserts IA_XIOP.BUS_ERR).

In the case of a write operation, data driven from the NCU is parity checked on the NIA. As in the case of an invalid header, a parity error in the write data results in a bus error.

In the case of a read operation, data driven from the NIA is parity checked on the NCU. A parity error results in a bus error, although the IA_XIOP.BUS_ERR signal is not asserted by the NIA.

When the NIA detects a PCM error the IA_XIOP.BUS_ERR signal will be asserted by the NIA and the NBUS bit will be set. The NIA will have to be examined to determine if there is has been PCM error.

All of these bus error conditions result in an error being reported to the SPU, and the NBUS bit of the Error_Log register (bit 4) will be set. This bit remains set until the Error Log register is written by the SPU. The parity error bits for the 8 byte data word in question (data or header) are saved in the Error_Log register (bits 31-24) as well.

2.3.4.3 NXP Sequencing Errors

The NXP I/F state machine expects a definite sequence of states during a valid NXP access. If, for some reason, the sequence is not correct, an error results. For instance, if the grant form the NIA is negated part way through a cycle, or if the SPU access in progress terminates as a bus time-out, the NXP I/F logic detects a sequencing error. This results in an error being reported to the SPU, if possible, and the NSEQ bit of the Error_Log register is set (bit 5).This bit remains set until the Error Log register is written by the SPU. Other conditions that can cause this bit to be set are by specifying more than one of the following operations, TAC, TAS, I/O and Scrub. Note that these operations are mutually exclusive and are only used in the XMAP interface to main memory. See Sec.2.4.2 and Sec. 2.4.3 for more information.

2.4 NXP Address Translation Map (XMAP)

The XMAP is a block of RAM which implements a set of 1024 32-bit registers. These registers are accessed in two different modes, direct and indirect. Both modes require a collection of address bits from the address supplied by the SPU. The NWII block diagram shows a mux for these two address bit slices, the output of which is used to address the XMAP.

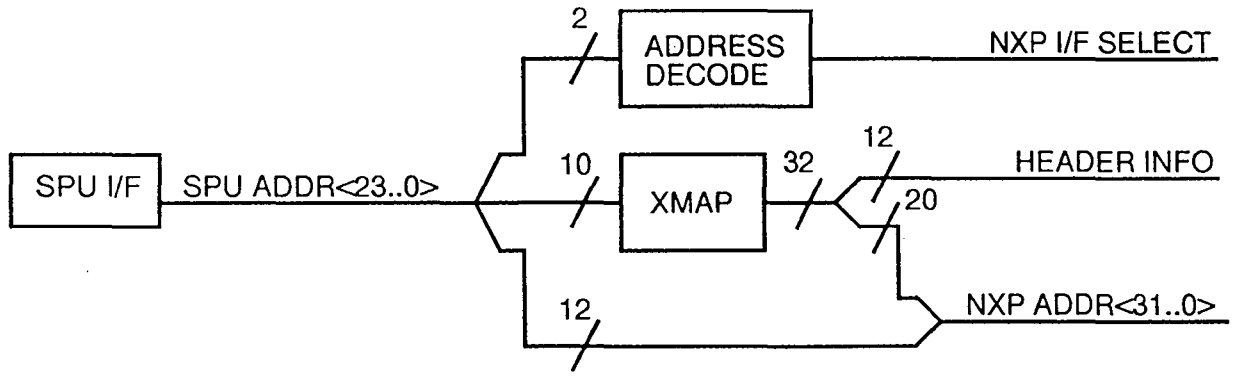
2.4.1 SPU XMAP Access

The SPU can read and write the RAM contents directly by addressing the XMAP registers. In this mode, the XMAP functions as a 1K x 32-bit RAM. Parity is checked on both reads and writes. If a parity error is detected, the XMP bit of the Error Log (bit 2) is set, and an error is reported to the SPU.This bit remains set until the Error Log register is written by the SPU. The parity error bits are also stored in the XPERR bits of the Error Log (bits 19-16) until the next XMAP access. Each parity error bit is from a separate bank of RAM, thus isolating parity errors to a particular RAM chip.

2.4.2 Using the XMAP with the NXP

The XMAP is also referenced by any SPU accesses which select the NXP. The XMAP is responsible for providing the window mapping information used to translate SPU addresses to Neptune main memory addresses, as well as determining the type of operation to be performed. A simplified diagram of the address mapping mechanism for NXP transfers is shown below:

Figure 2-8 NXP Address Translation



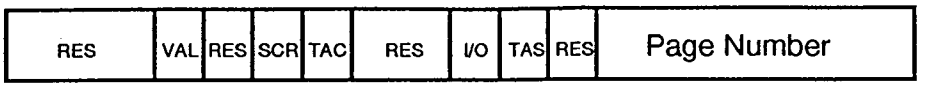
2.4.3 XMAP Registers

The XMAP register structure is shown below. The address offsets shown are those supplied on the cable by the SPU. The actual addresses used by system software on the HP workstation are dependent on the configuration of the NWI. The window numbers correspond to the 1024 logical windows which can be used to map into main memory.

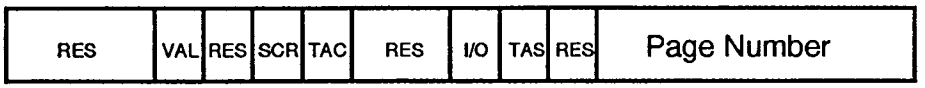
Figure 2-9 XMAP Registers

Bits 31, 30, 29, 27, 24, 23, and 20 are reserved. These bits are implemented as read/write in the XMAP registers, and are also part of the scan chain.

Bit 28, the **Valid** bit, indicates that the information in the rest of the register is valid and that the workstation may access the corresponding page of memory. An access to a page which does not have the **Valid** bit set results in an error being reported to the SPU. In this case, the NBUS bit of the Error_Log register would be set (bit 4).



NXP Window 0x3FF
Offset 0x3F5FFC



NXP Window 0x000
Offset 0x3F5000

Bit 26, the **Scrub** bit, is used to inform the NIA to scrub the referenced location in memory to correct single bit ECC errors. The NIA translates the scrub transaction into a word write with no zones selected. This has the effect of causing the memory system to read and correct the corrupted word, then write the corrected word back with no new write data merged in. Any SPU write reference to a word with the **Scrub** bit set results in an NXP transaction which is word aligned and 4 bytes long.

Bit 25, the **TAC** (*Test and Clear*) bit, is used to perform test and clear operations on semaphores in main memory. TAC operations are only allowed on bytes. Any access larger than a byte to a page with the **TAC** bit set results in an error being reported to the SPU. A TAC operation is performed by reading the desired byte with the **TAC** bit set in the map register. The contents of the memory location are returned and the location is set to 0x00.

Bit 22, the **I/O** bit, causes SPU NXP reads and writes to access I/O space instead of main memory space.

Bit 21, the **TAS** (*Test and Set*) bit, is used to perform test and set operations on semaphores in main memory. TAS operations are only allowed on bytes. Any access larger than a byte to a page with the **TAS** bit result in an error being reported to the SPU. A TAS operation is performed by reading the desired byte with the **TAS** bit set in the map register. The contents of the memory location are returned and the location is set to 0xFF.

Bits 19 through 0 of the map register contain the actual address mapping information. These bits become the upper 20 bits of the NXP address during an NXP transaction. The lower 12 bits of the NXP address come directly from the lower SPU I/F address lines (bits 0 and 1 are actually derived from the spu interface byte selects, adctl reg 27..24).

The **Scrub**, **TAC**, **I/O**, and **TAS** bits define mutually exclusive operations. Accesses made to a page with more than one of these bits set results in a NSEQ (bit 5 of Error_Log) error being reported to the SPU.

2.5 Memory Test/Initialization (MTST) Logic

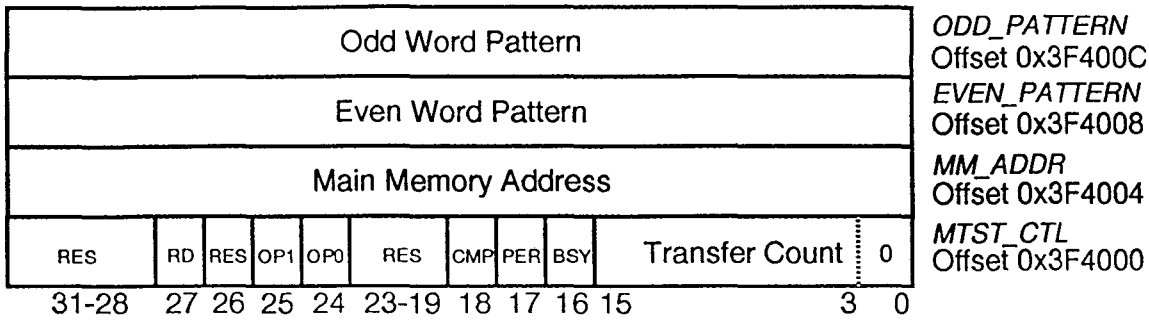
2.5.1 Purpose

For those who spend a lot of time in the lab or on the test floor initializing and booting machines, one of the slowest of all operations is initializing main memory form the SPU. This is due to the SPU's low main memory access bandwidth of about one megabyte per second on a good day. This low bandwidth also makes pattern testing main memory from the SPU very painful. Both of these operations can be dramatically speeded up using the MTST logic.

2.5.2 Control Registers

The MTST logic is controlled by four 32-bit registers on the NCU, as shown below:

Figure 2-10 Memory Test/initialization Control Registers



The *Even Word Pattern* and *Odd Word Pattern* registers contain the next pattern to be written to memory on a write, or the next pattern expected to be read back on a read. The registers are updated based on the function selected by the opcodes. In the case of an NXP error on a write operation, the pattern registers will contain the next pattern to be written to memory, not the pattern associated with the error. Errors during read/compare operations will result in these registers containing the pattern associated with the error.

The *Main Memory Address* register contains the current NXP transfer address. Since the interface only transfers longwords, the least significant three bits of this register are always forced to zero.

The *MTST_CTL* register contains fields that specify what operation to perform, and that monitor for the completion of the operation. The register is broken down as follows:

Bits 31-28, bit 26, and bits 23-19 are reserved.

Bit 27, the **Read** bit, controls whether the MTST hardware writes to main memory or does a read/compare with main memory. Setting this bit to one selects a read/compare operation.

Bits 25 and 24, the **Opcode** bits, determine the type of pattern generated by the even and odd word pattern generators. Four opcodes are currently defined as shown in the following table:

Table 3: Pattern generator Opcodes

Op1	Op0	Opcode/Mode	Function
0	0	0	Hold
0	1	1	Increment
1	0	2	Shift Left
1	1	3	Complement

In mode 0 (Opcode equals 0), the even and odd pattern words remain unchanged for the length of the transfer. This mode is used to initialize memory to a constant (such as zero). In mode 1, each pattern word is incremented by two at the end of every longword transfer on the NXP. This mode is used to put an incrementing pattern of address equals data in memory (on word

boundaries). In mode 2, the two pattern words are linked to form a 64-bit shift (left) register. This mode is used to generate patterns of walking ones and zeroes. This mode is also used to implement scan mode operation of these registers. In mode 3, each pattern word is complemented at the end of every longword transfer on the NXP. This mode is used to create/test various noisy or abusive situations. Only one mode/function can be performed at a time, and the mode should not be changed while the BUSY bit is set (see below).

Bit 18, the **Compare Error** bit, indicates that the 64-bit pattern read from the NXP did not match the pattern in the Even and Odd pattern words. When this error occurs, the transfer is stopped, the Main Memory Address register contains the address of the failing longword, and the pattern words contain the expected pattern. The failing pattern can be obtained by reading main memory at the failing address.

Bit 17, the **NXPBUS Error** bit, is set when an NXP BUS error is reported by the memory system during a transfer. For both read and write operations, the memory address register points to the address associated with the NXP BUS error.

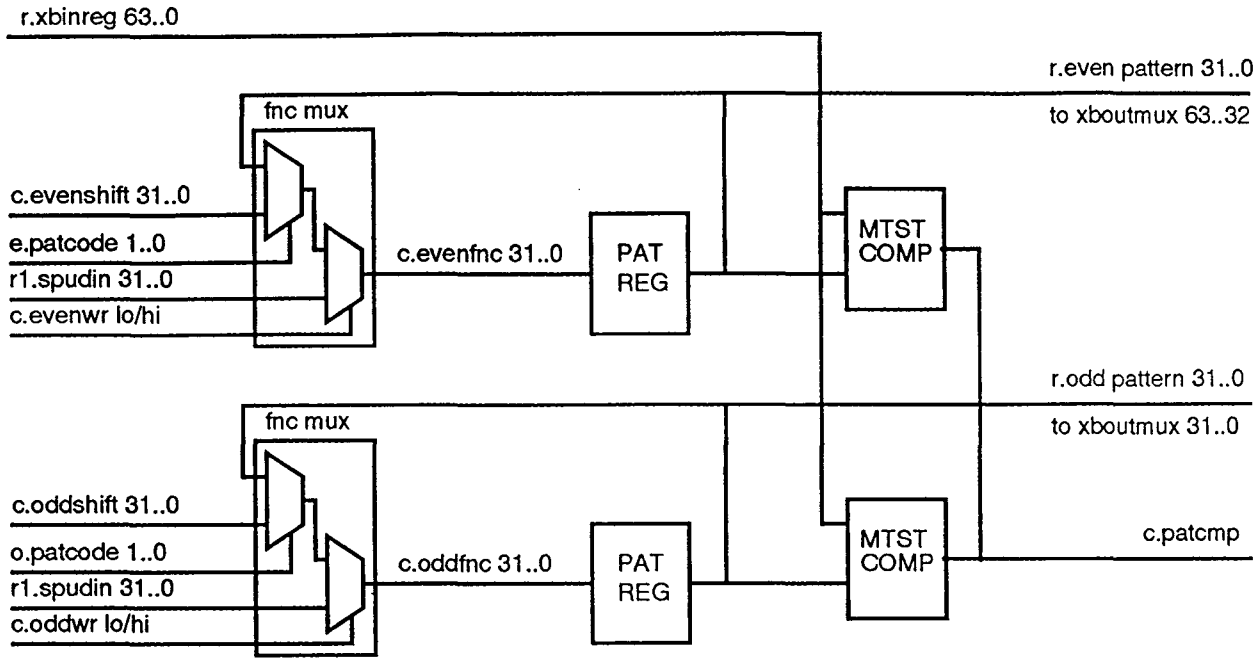
Bit 16, the **BUSY** bit, is used to indicate that a write or read/compare operation is in progress. This bit is set when the Transfer Count is written. At this point, the MTST hardware requests the NXP and transfers patterns until either the transfer count goes to zero or an error occurs. When the operation completes (successful or not), the MTST logic resets the **BUSY** bit back to zero. The SPU can poll this bit to determine the status of a transfer. No other indication of completion (interrupt, etc.) is provided. There is no time-out mechanism either, but the SPU can terminate a transfer in progress by writing a one to this bit (which clears it as the transfer terminates).

Bits 15 through 0, the **Transfer Count**, are the actual byte count used in the NXP header. A count of zero results in a 64 KB transfer. Since the interface only transfers longwords, the least significant three bits of this register are always forced to zero.

2.5.3 MTST Hardware

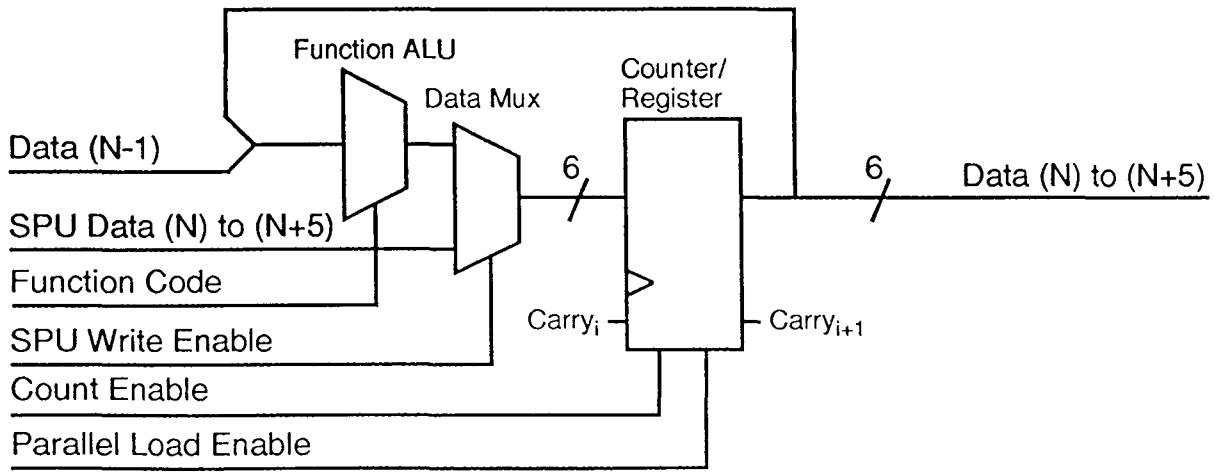
Figure 2-11 shows the posititon of the date function generators with respect to 3 data paths. Data from the spu written to the generators. Data received from the NXP interface and compared with the expected data and data patterns generated for the NXP interface during the MTST operations. The address register is implemented with the same structure so that when a compare error occurs the address of the failing location is known.

Figure 2-11 MTST Function generators in Data Path



The bulk of the hardware required to implement the MTST functionality is used to create and update the pattern and address registers. Not only are these registers are read/write capable, but they must also be updated during MTST operations. Furthermore, these registers must be scannable.Given the large size of these registers, the most reasonable approach was to develop circuitry to implement a register slice. Slices can then be connected together to create the registers desired, along with the required scan path. A pattern or address register slice has the following general architecture:

Figure 2-12 MTST Pattern/Address Register Slice



Each slice is 6 bits wide (or a portion there of). The Counter/Register holds the current value of the register slice. For any operation except Increment, the parallel load enable causes the

Counter/Register to load the output of the Data Mux. This is also the mechanism which allows the SPU to write data to the register slice. Other operations are generated by the Function ALU as specified by the Function Code (derived from the Opcode field of the *MTST_CTL* register). Notice that the most significant bit from the next lower slice is connected to the Function ALU. This provides the scan path and also the Shift Left functionality. For the Increment operation, the Count Enable causes the Counter/Register to count if the $Carry_i$ bit from the preceeding slice is asserted. The $Carry_{i+1}$ bit is asserted as appropriate for use in the next slice.

In the NWII, these slices are constructed with a PAL and a counter/register device. The PAL implements the Function ALU and Data Mux portion of the circuitry. Several additional PALs are required to characterize the Enable and Function Code signals for each register slice.

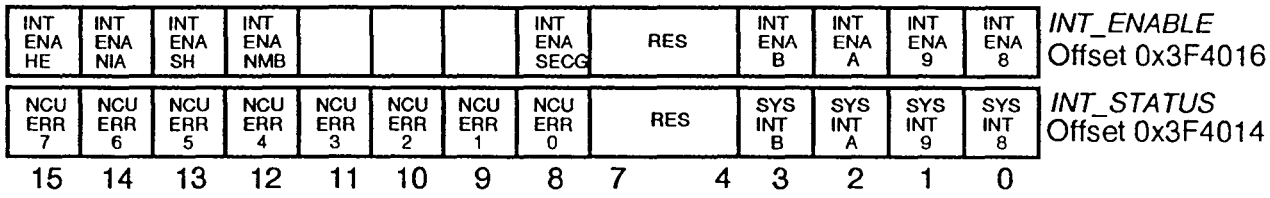
2.6 NWII Interrupts

The NWII portion of the NCU maps NCU errors and System interrupts into an interrupt request line to the SPU. In addition to interrupt mapping, the NWII also provides an interface to the System Interrupt Bus, allowing the SPU to send and receive system interrupts.

2.6.1 Interrupt Mapping

The NWII maps twelve interrupts into the SPU interrupt request line (CU_WS.INT). These interrupts have status and enable bits associated with them as follows:

Figure 2-13 Interrupt Status and Enable Registers



The offset shown are those provided on the cable by the SPU. The actual addresses used on the workstation are dependent on the configuration of the NWI. The *INT_STATUS* register provides information about the current state of interrupts mapped into the SPU interrupt request line. Note that once an interrupt has been asserted, the corresponding status bit remains set until cleared by the SPU. The SPU clears a bit position in the *INT_STATUS* register by writing a one to that position. The individual bits of the *INT_STATUS* register are as follows:

Bit 15 indicates a **Hard Error.**

Bit 14 indicates an **NIA Soft Error.**

Bit 13 indicates a **Scalar Halt.**

Bit 12 indicates an **NMB Soft Error.**

Bit 11 indicates a **Scan Memory Access Error.**

Bit 10 indicates a **Scan Memory Parity Error.**

Bit 9 indicates a **Clock Generator Parity Error.**

Bit 8 indicates an **XCL Parity Error.**

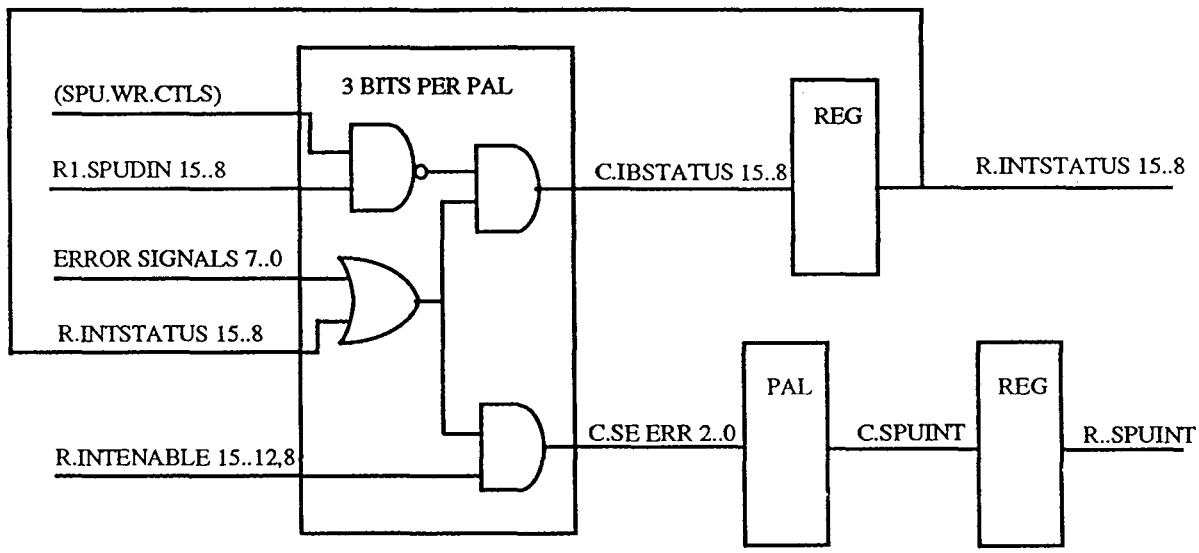
Bits 7-4 Reserved

Bits 3-0 indicate the corresponding system interrupt has been recognized.

The *INT_ENABLE* register provides enables to mask off the above interrupts. The lower 8 bits are enables for the system interrupts. Currently only 4 system interrupts are implemented, however, to increase this number is a simple modification. Bits 7-4 have been reserved for this purpose. All the Scan engine and Clock generator errors, XCL Parity, Clock Generator Parity, Scan Memory Parity and Scan Memory Access, are enabled with bit 8. Bits 12-15 are enables for the corresponding NCU Error interrupts.

Figure 2-14 shows the block diagram of the error interrupt system. Interrupts that are disabled, but active, are visible in the status register. Only if the corresponding bit is enabled will the interrupt be sent to the SPU.

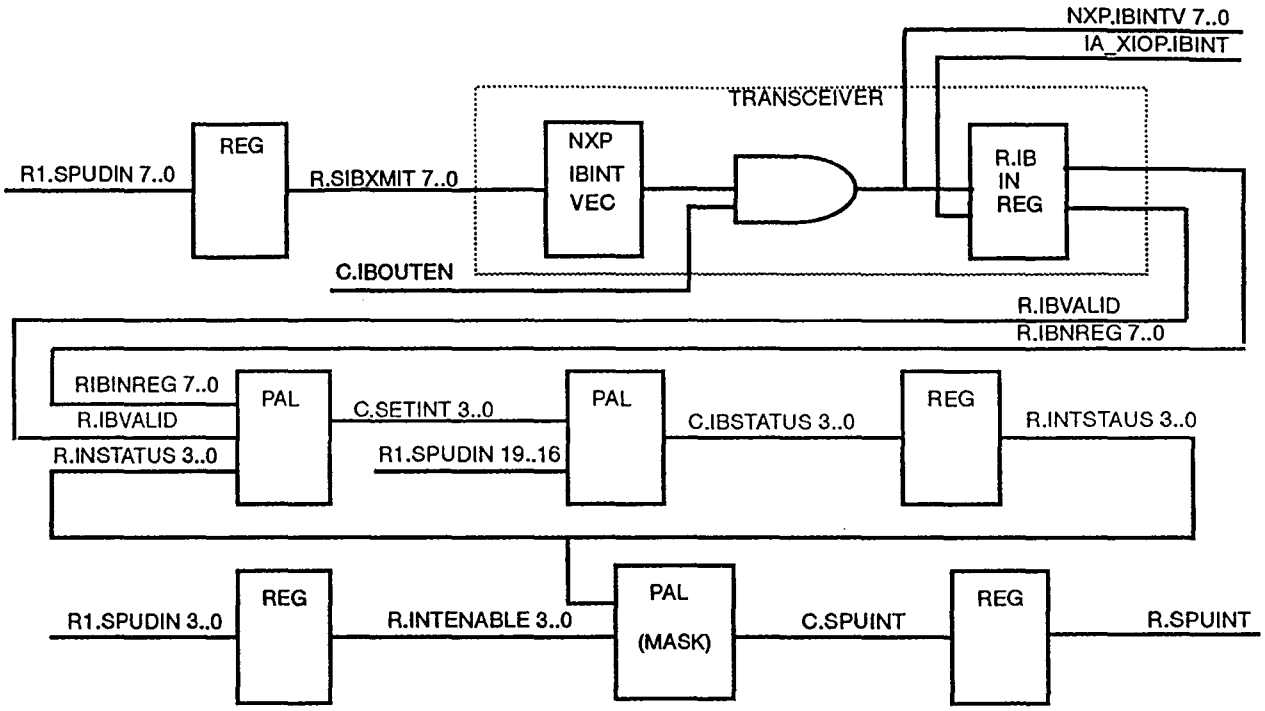
Figure 2-14 NCU Error Interrupt Structure.



2.6.2 System Interrupts

The NIA provides an NXP Interrupt Bus for sending and receiving system interrupts. A block diagram of the System interrupt structure shows how the system interrupts are implemented.

Figure 2-15 System Interrupt implementation.



Both of the interrupt cycles (sending and receiving) are different from a data transfer on the NXP bus. Figure 2-16 and Figure 2-17 show each of these cycles. System interrupts received on the NXP Interrupt Bus are monitored by the interrupt bus interface. When the interface recognizes system interrupts 8, 9, A, or B and the *IA_XIOP.IBINT* signal is set, the corresponding bit in the *INT_STATUS* register is set. If the interrupt is enabled, an interrupt is sent to the spu.

The *SIB_XMIT* register provides the SPU access to the NXP Interrupt Bus for the purpose of sending system interrupts. The NCU Interrupt state machine state enters the Grant state as soon as a vector is written into the *SIB_XMIT* register. This state is maintained until the NIA grants the interrupt bus. The next three states proceed automatically. The Pre-transfer state is used to drive the vector onto the bus. The Transfer state keeps the data on the bus, and at the end of the Post transfer state the vector is removed from the bus.

Figure 2-16 NXP Interrupt Cycle - NCU receiving interrupt

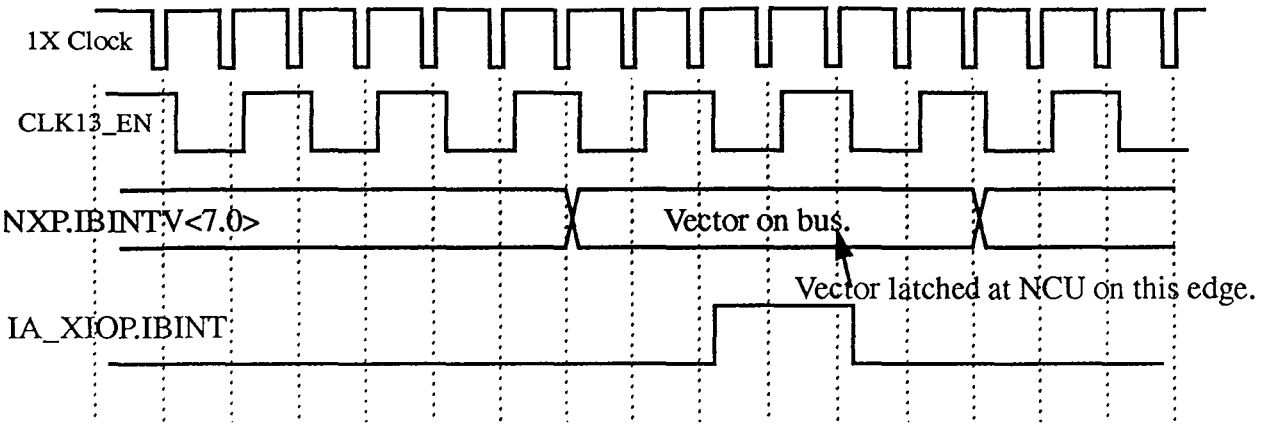
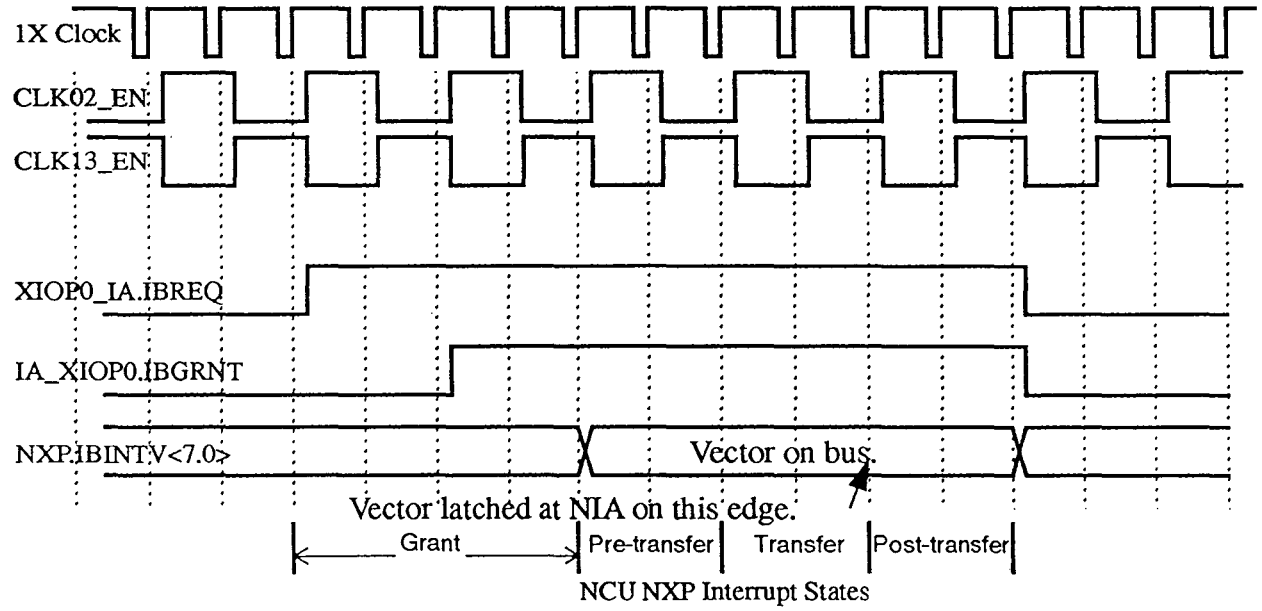
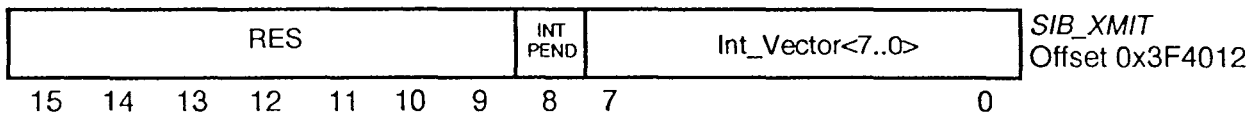


Figure 2-17 NXP Interrupt Cycle - NCU sending an interrupt.



The *SIB_XMIT* register is shown below:

Figure 2-18 SIB_XMIT



Bits 15 through 9 are reserved.

Bit 8, the **Interrupt Pending** bit, is set whenever the Interrupt Vector (bits 7-0) is written by the SPU. Once set, the Interrupt Bus interface attempts to send the vector on NXP Interrupt Bus to the NIA, where it will be forwarded to the rest of the system. When the vector has been successfully sent, the bit is reset to zero. The SPU should poll this bit to determine if the last interrupt has been sent before sending another one. If the SPU attempts to write the vector while

the **Interrupt Pending** bit is still set, the IBSY bit of the Error Log register (bit 6) is set, and an error is reported to the SPU.

Bits 7 through 0 are the **Interrupt Vector** to be sent. This vector remains unchanged after being transmitted on the NXP Interrupt Bus.

2.7 Diagnostic and Utility Registers

The remaining SPU-addressable registers in the NWII portion of NCU are of either a diagnostic or utility nature. The following sections explain the purpose and use of these registers.

2.7.1 Loopback Data Register

The Loopback Data register provides the SPU with a means of testing the data path to the NCU from the NWI. This 32-bit register half-word, or word addressable. The register is physically implemented by the hardware in the data read mux register (RDMUX_REG).

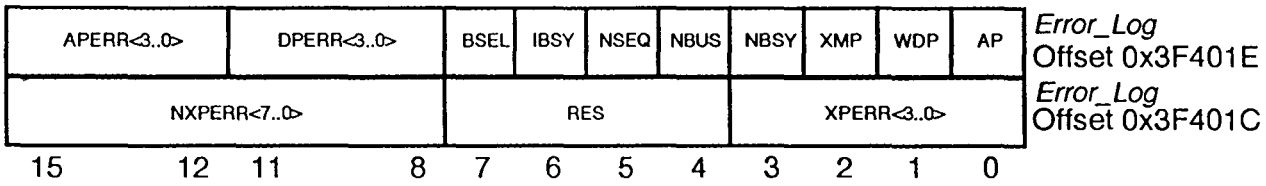
Writing this register involves selecting address offsets 0x3F4018 through 0x3F401A as appropriate to the size of the write access. The appropriate byte(s) of the SPU I/F data bus are clocked into the register, and held until the next SPU read from the NCU, or until the next SPU write to the Loopback Data register, whichever comes first.

To read this register, the SPU supplies the offset for the byte(s) requested. The read mux register maintains its value, and the contents are returned to the SPU.

2.7.2 Error Log Register

All of the bits/fields of the Error Log register have been referred to previously in this section. Each error bit, or collection of parity error bits, are discussed in the context of how such an error condition occurs. The following diagram and definitions are only brief explanations. The reader should refer to the previous sections for more details about error conditions. All bits in the Error Log are frozen when an error occurs. The contents are retained until such time as the SPU writes the Error Log register to clear the error condition. The Error Log register is shown below:

Figure 2-19 Error Log Register



Bits 31-24, the NXPERR bits, indicate which byte(s) had parity errors when a parity error is detected on reads from the NXP. If one or more of these bits is set, the NBUS bit (bit 4) should also be set. Refer to the section on the NXP Interface for more information about this error condition occurs.

Bits 23, 22, 21and 20 are reserved. These bits are not implemented on the NCU, and thus can not be scanned. Writes to these bit positions have no effect.

Bits 19-16, the **XPERR** bits, indicate which bank(s) of the XMAP RAM reported a parity error on

the most recent access to the XMAP. Refer to the section on the NXP Address Translation Map for more information.

Bits 15-12, the **DPERR** bits, are the parity error bits associated with data written from the SPU. One or more of these bits set implies that the WDP bit should also be set. Refer to the sections on Address and Data Flow, and also the SPU Interface for more information.

Bits 11-8, the **APERR** bits, are the parity error bits associated with the contents of the ADCTL register, which is loaded by the SPU. One or more of these bits set implies that the AP bit should also be set. Refer to the sections on Address and Data Flow, and also the SPU Interface for more information.

Bit 7, the Byte Select Error bit indicates that a byte access of the scan engine or scan memory was attempted.

Bit 6, the **IBSY** bit, indicates that the NXP Interrupt Bus had a pending interrupt when the SPU attempted to write a new vector to the SIB_XMIT register. Refer to the section on System Interrupts for more information.

Bit 5, the **NSEQ** bit, indicates that the NXP access resulted in a sequencing error. Refer to the NXP Interface section.

Bit 4, the **NBUS** bit, indicates an NXP bus error, detected either by the NCU or the NIA. Refer to the NXP Interface section.

Bit 3, the **NBSY** bit, indicates that the NXP was in use by the MTST logic when the SPU attempted to access the NXP directly. Refer to the NXP Interface section.

Bit 2, the **XMP** bit, indicates an XMAP Parity error. Refer to the section on the NXP Address Translation Map for more information.

Bit 1, the **WDP** bit, indicates a Write Data Parity error in the data path from the SPU. Refer to the sections on Address and Data Flow, and also the SPU Interface for more information.

Bit 0, the **AP** bit, indicates an Address Parity error in the address/control information from the SPU. Refer to the sections on Address and Data Flow, and also the SPU Interface for more information.

2.7.3 Neptune Serial Number

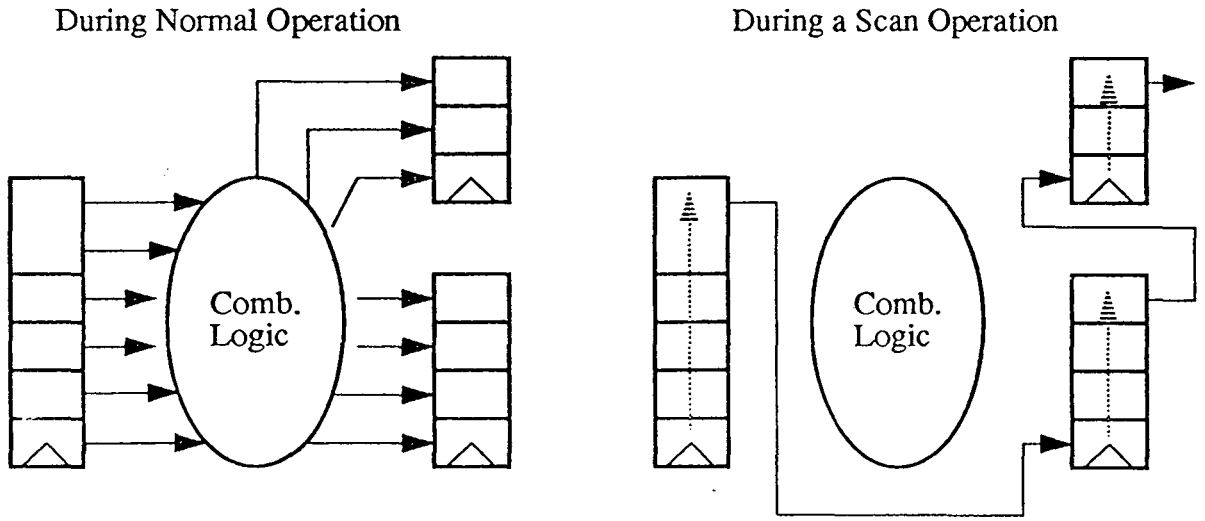
The Neptune Serial Number can be accessed through a register at offset 0x3F4010. This 16 bit register is a read only register, which returns the value of the serial number jumpered on the I/O Bay backplane. Writing this register has no effect. This register is not scannable.

3 Scan Engine and Clock Generator

3.1 Overview of Scan in the Neptune System

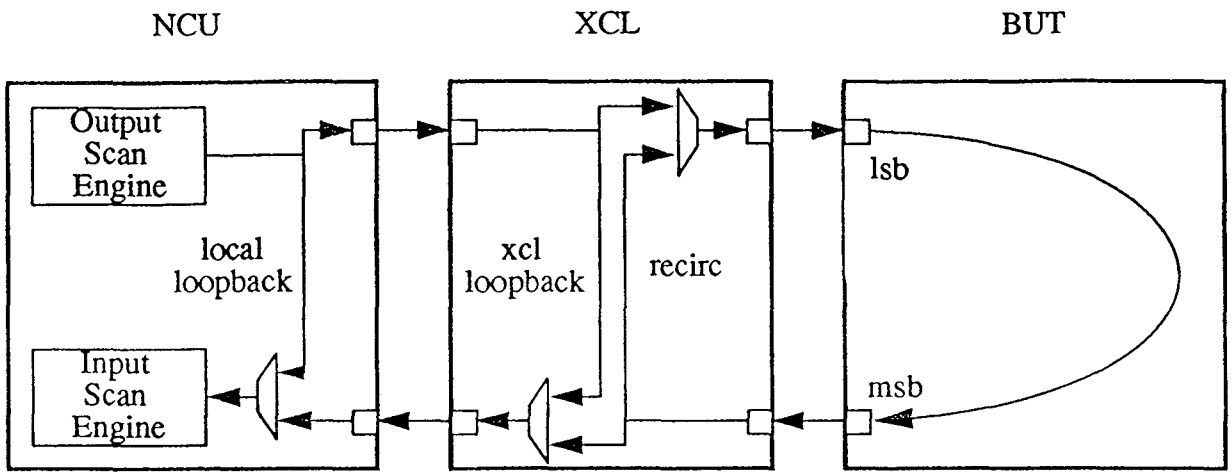
Before going into the detailed operation of the Diagnostic Engine, it is important that the process and importance of board scanning be defined for those who may not be familiar with its use as a diagnostic tool. Board scanning for lack of a better description, is the process of turning all of the register elements contained on a logic board into a large shift register. The continuous path of register elements that make up this shift register is known as the board ring. With the clocks disabled, this ring represents the state of the board at any instance in time. The block of logic responsible for transferring data between this ring and memory contained on the NCU is defined as the Scan Engine.

Figure 3-1: Normal to Scan Operation Logic Conversion



The process of transferring data between the Scan Engine and the Board Under Test (BUT) is known as a Scan operation. Data is tranferred from the least significant bit in any register structure to the most significant bit and is known as a Scan Left Operation. The Scan Input of a board is known as the least significant bit in a ring. The Scan Output of a board is known as the most signficant bit of the ring. During a scan operation, the scan output of the BUT will be the first bit transferred. In the case of a scan write, this bit will pass through every register element contained in the test ring until it reaches its finally destination. Exactly the opposite is true of the scan in bit of the ring. It is the last bit transferred between boards in a scan operation. During a read operation, the Scan in bit data will pass through each register element in the test ring before being transferred to the Scan Engine. The following figure illustrates the location of the ring msb and lsb in reference to the direction of scan and the proximity of the input and output Scan Engines.

Figure 3-2: Scan Engine Serial Data Path

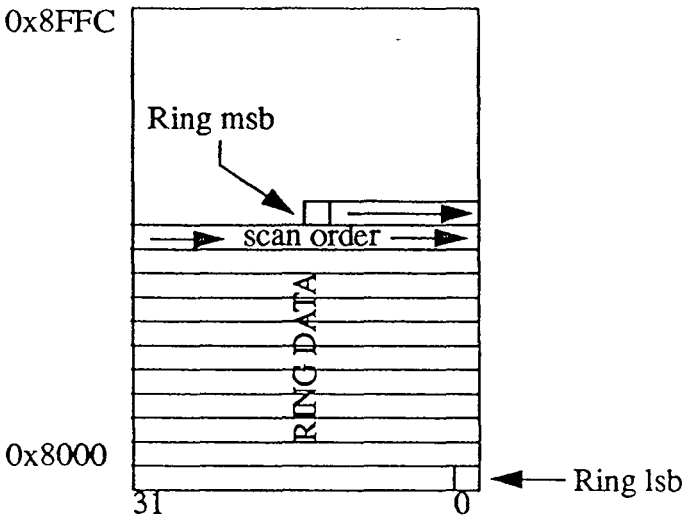


Three loopback data paths are included in the Scan Engine to allow for self diagnostic operations and non-volatile reads of data. The local and xcl loopback allow the Scan Engine to perform loopback data testing in order to verify logical operation of the system and to guarantee data path integrity. Pipeline adjustments in the scan data path allow the data to be properly synchronized so that data arrives at the same time that it would if it were coming from a BUT. The recirculation data path allows the spu to recover the state of a ring without disturbing its state. As data is read into the Input Scan Engine, the same data is also transmitted to the scan input of the BUT. The scan data pipeline is adjusted so that at the end of a scan operation, data will reside in the same register element that it started out in.

Scan data is divided into four types and is stored in the Scan Memory during a scan operation. The four data types are output, mask, compare, and result. Output data is the data that is transmitted to a board during a scan write operation. It is only accessed by the Output Scan Engine. Mask data is used to mask off bits during a data verification operation. When any mask bit is set, it effectively zeroes out the data that will written to the corresponding bit in the result page in memory. Compare Data is used during verification to compare against received data that is not masked off. Mask and Compare data are only accessed by read operations from the Input Scan Engine. Result data is the only data written to the Scan Memory by the Scan Engine. During a read with no verify, the result data written to the Scan Memory will consist of the actual data that was read from the BUT. During a read with verify, the output of the verification logic will be written to the memory.

Data is organized the same in each of the four pages of the Scan Memory. A given register bit at the same relative address within each of the four pages of memory will reference the same bit in a any given scan ring. This occurs because the least significant bit of any scan ring is contained in the lsb of the register at relative address 0 in each memory page. Therefore, as a scan ring gets larger (for different boards), data will occupy higher addresses within the framestore. When a scan ring contains a non-integral number of 32 bits, the partial word containing the left over scan data will be stored in the most significant address of the addresses containing scan data. This partial word will be the first word that is accessed in the scan operation. The organization of scan data within a page of memory is illustrated in table 3.14.

Figure 3-3: Scan Memory Page Organization



3.1.1 Types of Scan Operations

All scan rings in the system can be divided into one of three ring types: Static, Dynamic, or CCU. The ring type determines how data is transferred between the BUT and the Scan Engine, and how long the data pipeline is between the BUT and the scan interface of the CU board. In a Static scan operation, clocks are first disabled and then changed to the 1x clock rate. The 1x clock is then bursted while data is transferred between the Scan Engine and the BUT. In a dynamic scan operation, the board clock to the BUT is neither disabled or changed (assuming that it is running). The scan controls in conjunction with a scan enable to the board, control the clocking and shifting of the scan ring. CCU scans are similar to Dynamic scans because an enable signal is used to control the clocking and shifting of the BUT. However, since the target board is clocked by the PBUS clock , scan entry and exit overhead, and data synchronization between the CU and the BUT, make this operation significantly more involved than a Dynamic scan.

Three basic steps make up each of the above scan operations. These are: Clock disabling, Scan Engine initialization, and data transfer and clock enabling. Of these steps, the first two are common to all of the scan types, while the third will differ greatly depending on the type of scan. The first two steps of the scan process are described below while the third step is described in the sections covering each scan type.

The first step of any scan operation is to disable clocking to the ring which will be scanned. This is accomplished by either shutting the board clock off at the Clock Generator or disabling the ring clock by resetting its enable. In the case of Static scans, the board's clock will be shut off by executing a disable command in the Clock Generator while its enable is set in the Diagnostic Enable register. For Dynamic and CCU scans, the ring clock is disabled by shutting off the ring clock enable to the BUT. This is accomplished by executing a CCU disable in the clock generator while the ring's bit is set in either the rbe, slog/ccu_ctl, or log_sys_ena registers contained in the Scan Engine register space. Once the clock to the ring under test is disabled, the scan engine can be set up and the scan controls to the BUT can be safely changed.

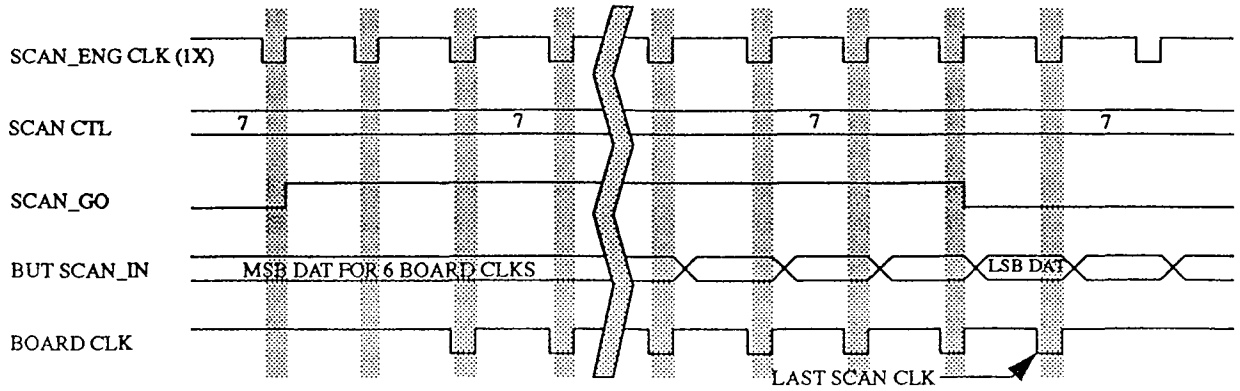
In the second step of a scan, the Scan Memory is loaded with ring data and the scan command is executed in the Scan Engine. Before the control registers of the scan engine are loaded, the software will load the Scan Memory with the data required for the upcoming operation. If a write is executed, the outgoing data page will be loaded into the memory. If a read with verify operation is executed, then the mask and compare data pages will also have to be loaded. Note: A scan read without verification does not require any data to be loaded into the Scan Memory. Once the Scan Memory is loaded, the appropriate control registers in the Scan Engine are loaded. This is completed by writing the command status register of the Scan Engine with the desired operation. The act of writing the Command Status register causes the execution of the command.

Between when the command is written to the Scan Engine and the scan command is executed in the Clock Generator, the Scan Engine will prefetch data for the first word (s) in the upcoming scan operation. In the case of a write operation, the output scan engine will fetch the first two words of the scan and load them into the output shift register and the output buffer register. For a read with verify operation, the input scan engine will prefetch the mask and compare data from memory and load them into their respective registers for the first word received in the read operation. Once the data has been fetched and loaded into the appropriate registers, the Scan Engine will go into idle until the scan command is executed in the Clock Generator, and the scan_go output of the Clock Generator becomes active. This output becoming active indicates that on the next 1X clock cycle that scan clocks (or their ring enable) will be issued to the selected boards and that data transfer can commence.

3.1.1.1 Static Scan Operations

In a Static scan operation, the clocks to a given board are bursted as the 1X rate while data is transferred between the Scan Engine and the board. During the scan operation, the BUT will receive a number of clocks equivilent to the number of bits in the scan ring plus six extra clocks to account for scan data pipelining in the XCL. The data pipeline between the Scan Engine and the BUT is set up so that the last piece of data issued from the Scan Engine coincides with the last clock edge issued by the Clock Generator. Scan data pipelines on the XCL are set up so that the same number of clocks will be issued to the BUT whether scan data transferred to the board comes from the Scan Engine (Scan Write) or from the recirculation pipe (Non-volatile Read) contained on the XCL.

Figure 3-4: Static Scan Operation Timing



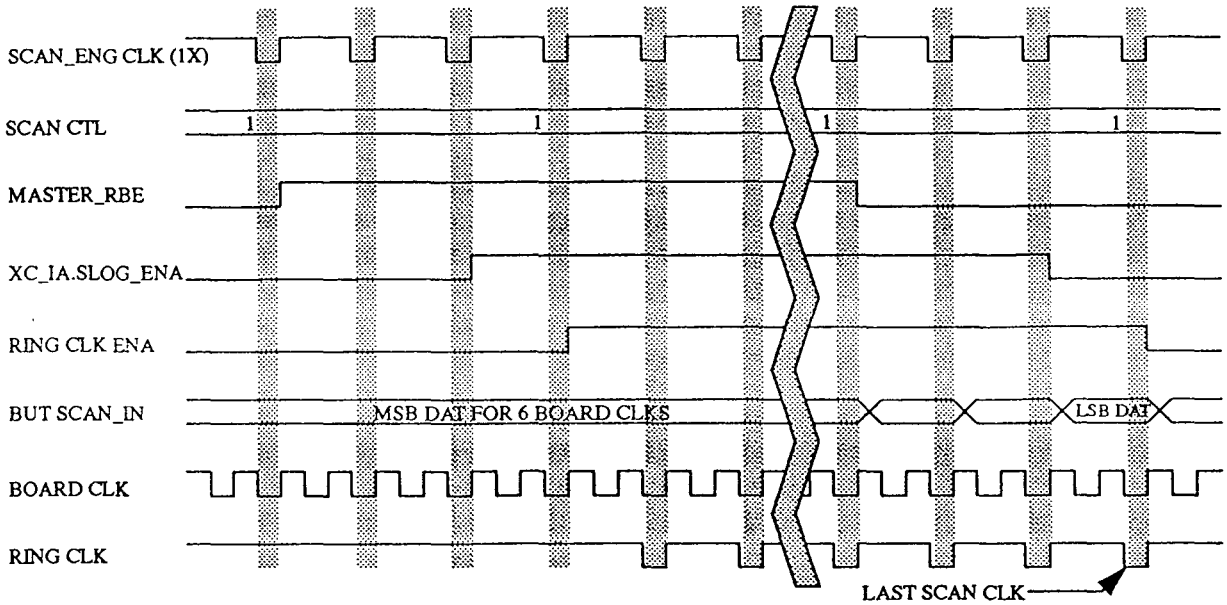
3.1.1.2 Dynamic Scan Operations

At its most basic level, a Dynamic scan operation is very similar to a Static scan operation. Clocks to the desired ring are bursted while data is transferred between the Scan Engine and the BUT. However, once one looks at the implementation of this scan operation, the similarities cease. A dynamic scan is defined as a data transfer that occurs while a certain scan code and a clock enable are present at the interface of a board. Another interesting difference between Static and Dynamic scans is that the board clock for the BUT will remain "free running" for the entire operation at the rate specified by the contents of the Clock Generators frequency control registers. All Crossbar boards (except the XCL), the NIA, and the NMB all support this type of scan mode. Error log recovery and changes to the processor configuration are examples of tasks that are accomplished through Dynamic scan operations.

Unlike Static scans where control of the clocks occurs at the clock source, control for the clocks must pass through pipelining on the XCL alongside the data. The clock enable signal is also registered on the BUT before it is used to control the flow of clocks to the ring. The source of the clock enable is the master_rbe signal which is sourced by the Clock Generator array. This signal when logically anded with the ring's enable contained in the slog/ccu_ctl, or log_sys_ena produces the ring clock enable. When the ring clock enable is set, the ring will shift once on each rising edge of the 1X clock. Since the BUT continues to receive its normal free running clock throughout the scan, the 1X clock driving the ring must be sourced from the BUT's on board 1x phase generator.

Since the new control signal changes the relationship between the ring clock and the scan data, the scan data pipelining must be changed in order for data alignment to be correct. This is accomplished by monitoring the status of the log and sys bits in the Scan Engine Cmd register and the contents of the slog/ccu_ctl register. The log and sys bits indicate to the Scan Engine that a dynamic ring operation is being executed, while a non-zero value in the slog/ccu_ctl register indicates that the dynamic operation is going to an NIA board. Due to implementation differences between board types, the NIA requires an additional pipeline stage for proper data alignment.

Figure 3-5: Dynamic Scan Operation Timing



3.1.1.3 CCU Scan Operations

CCU scans are the mechanism used to scan the IO Processors residing on the PBUS. These scans combine elements of both Static and Dynamic scans. However, because data and control pass through the NIA, and the differences in clock speed, they are more involved. The Scan Engine uses a Run Bit Enable (similar to Dynamic scans) to control the flow of clocks to the board. However, since PBUS clock generation is sourced from the NIA, the Run Bit Enable disables the clock for the CCU under test. The actual clock to the BUT is stopped and bursted for the period of the scan (like a Static scan). Since there are only 3 PBUS clocks for every 20-1x clocks, logic is required to make sure that data is correctly transferred between the Scan Engine (1x clock) and the CCU under test (PBUS clock). A breakdown of the entire CCU scan process is described in the following paragraphs.

The first step of a CCU scan operation is the disabling of clocks to the board under test. This is accomplished by resetting the board's run bit enable. The run bit enable for a given board is used to control the flow of the boards clocks. When set, clocks wil be issued. When the run bit enable is reset, clocks will be disabled just after the falling edge of PBUS clock. A description of how these run bit enables work, can be found in section 3.4.4.4.1.

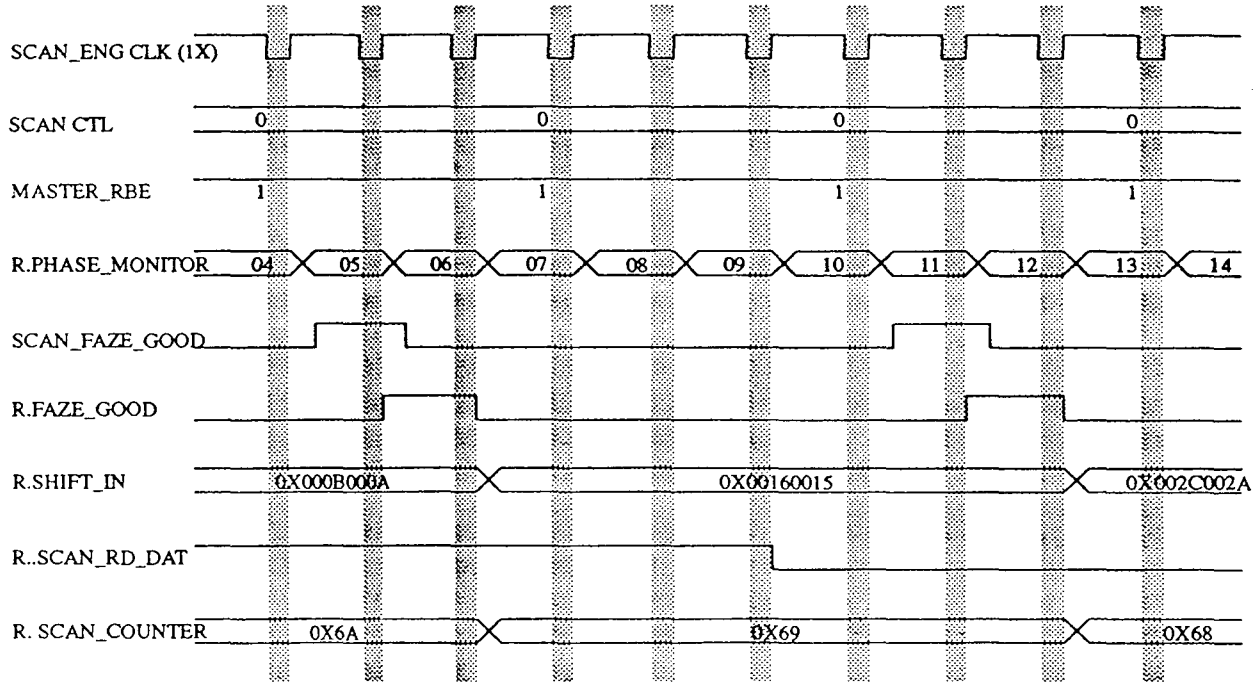
Once the clocks have been stopped to the BUT, the scan controls can be changed. However, unlike boards in the system, (daughter, XBAR) the CCU scan controls are sourced from the NIA and have no connection to the master scan controls contained in the Scan Command register. The scan controls for the CCU under test are sourced from the CCU Ctl ring contained on the NIA that services it. The contents of this ring are changed by performing a Dynamic ring scan on its contents. The act of completing the ring scan (ccu_ctl_ena becoming reset) loads the register that actually drives the scan controls onto the PBUS.

Once the scan controls for the CCU under test have been set, the Scan Memory can be loaded and the Scan Engine initialized. This process is the same as all other ring types in the system. There are no special codes that have to loaded to indicate to the Scan Engine that the upcoming scan is for CCUs. The Scan Engine will decode the state of the Clock Generator Com/Stat register (a shadow copy) to determine whether the operation will involve CCUs or not. Note: If a scan operation is attempted with the CCU bit set(0x7000 = 0x0A) while the Log and Sys bits are reset in the SE Com/Stat register, then a CCU scan is being performed.

Because the BUT receives a different number of clocks than the Scan Engine for the period of the scan, clock calculations are slightly more involved than for a Static or Dynamic scan. Since the CCU being scanned receives one clock for every six clocks (1X) that the Scan Engine receives, the Burst Counter in the Clock Generator is loaded with six times the number of scan ring bits plus twenty clocks to account for Scan Engine pipelining. For example, a 100 bit CCU scan ring would receive: 100 bits * 6 clock/bit + 20 pipeliine clocks = 620 clocks.

Because of the different clock rates between the BUT and the Scan Engine, data can't be transferred every 1X clock cycle. The scan engine controls the flow of data by monitoring the state of the PBUS clock generation monitor. Once every six 1X clock periods, a state is reached which is called a "scannable phase". When this state is detected, the scan engine will sample/transfer data. During this state, valid scan data will be present at the input of the Input Scan Engine, and data transferred to the CCU from the Output Scan Engine will be valid when the CCU receives it. An illustration of how this process works is illustrated in the following diagram.

Figure 3-6: CCU Scan Operation Timing



The final difference between CCU scans and other types of scans is that the board under test will receive a different number of clocks depending on whether a read or write scan operation is being performed. This occurs because scan read operations to CCUs automatically engage on-board recirculation logic which changes the scan data pipeline length. Unlike CCUs, all daughter and crossbar board scan read recirculations occur on the XCL which equalizes the scan read and write data paths. Since a software requirement of scan operations was to issue the same number of clocks for read and write operations, additional hardware was added to the Scan Engine to make these clock differences transparent to software.

A second counter was added to the scan engine to count PBUS cycles and reset the run bit enable to the CCU, so that the scan data would be properly aligned during a scan read. During a CCU scan read, the CCU read counter is loaded with the number of bits in the scan ring minus two. This guarantees that clocks to the CCU are disabled at the proper time while the last scan data from the BUT traverses through the NIA and XCL on its way to the Scan Engine. During a scan write operation to CCUs, this logic is disabled and has no affect on when the board's clock is disabled.

3.2 Scan Engine Modes of Operation

3.2.1 Description of Scan Control Modes

Each board in the system receives three scan control lines that control the scan operation for that board. The scan control lines control the flow of data and clocks within each board. Each slot in the system receives a unique set of lines that are transmitted from the control crossbar board. The unique set of scan control lines is created from the logical ANDing of the master scan control bits with the bit associated with the slot in the scan mask register. If the scan control enable for a particular board is set, then the scan control code contained in the scan command register will issued to the board. If the enable is reset, the board will be placed in normal mode (sctl = 000).

Mode 0 is the default scan mode for boards running in the system. If a board's scan control enable is not set, Normal mode is the code that will be issued to the board. While in mode 0, the BUT will have the same clocking and data flow that is used during system operation. Mode 0 is additionally the code that is sent to the crossbar boards during config load scans.

Mode 1 is used by the IA and MB boards to enable soft error log scans of the board. When this mode is enabled and the board's log ring enable bit is set, the BUT's log ring will shift one each rising edge of the 1X clock.

Mode 2 is interpretted as Last Left Sys mode by the IA and Shift Left Sys by the CU and XCL boards. When this code is presented to the IA, the board will be shifted one final time while the command contained in the STRs is be executed. When the code is presented to the CU and XCL boards, the system rings of each board will shift once on each rising edge of the board clock.

Mode 3 is interpretted as Shift Left Sys by the IA and MB boards and as Last Left Sys by the CU board. On the IA and CU boards, the system ring on the BUT will shift once on the rising edge of the board clock. However, the STRs on the CU board will additionally execute the command contained in their control ring. Since a system ring scan on the MB board is a dynamic scan operation, the MB board will shift on each 1X clock edge only while it's sys_run bit is set.

Mode 4 enables CCU Ctl scans on the IA board. While this control mode is enabled and the CCU Ctl Ena is set, the CCU control ring will shift once on each rising edge of the 1X clock. The MB board interprets mode 4 as the Enter NMC Scan Mode. While in this mode, the board will receive one clock which aligns the NMC clocks for scan.

Mode 5 is used by boards that contain Self Timed Rams and logic that scans like these devices, to execute the command contained within the device's control ring. While in this mode, the scan enable for most logic on the board will be set while the control for the rams will be reset. The single clock that is issued to the board while in this mode, causes the board ring to shift one final time while the ram executes the contents of its control ring. The MB uses this mode to align NMC clocks before the board enters normal mode. A single clock issued while the board is in this mode will properly align the NMC clocks for board clock restart.

Mode 6 is very similar to normal mode with the exception that clock gating in the third level of the clock tree is disabled. Assuming that the phase generators are properly set up, this will allow all devices to be clocked on the same clock edge. This is the scan control mode that is issued to boards during the parallel load phase of testing in the CAST tester.

Mode 7 is the scan mode that is used during full board scans. When a BUT is in this mode, the entire board is converted into a large shift register that will shift once on each clock edge.

The eight scan codes and their associated scan modes are illustrated in the following table

Figure 3-7: Neptune Scan Control Codes

Table 3-4: Scan Control Modes

Scan_Ctl	Scan Function
000	Normal
001	Shift Left Log (MB, IA)
010	Last Left Sys (IA) Shift Left Sys (CU,XCL)
011	Shift Left Sys (IA, MB) Last Left Sys (CU)
100	Load CCU Ctl (IA) Enter NMC Scan (MB)
101	Last Left Shift Exit NMC Scan (MB)
110	Load (CAST)
111	Board Left

3.2.2 Clock Generator Modes of Operation

The Clock Generator is responsible for the generation and control of the clocks and run bit enables that are distributed in the system.

Eight clock commands are available to place any combination of clocks and the master run bit enable (MASTER_RBE) in one of three states: free running, disabled, or diagnostic state. A clock is regarded as free running if the last command executed upon it was a restart command. A second way to verify if a clock is free running, is if its bit in the disabled clocks register is set while the busy bit in the command register is reset. Once a clock is free running, the only way to stop clocks form being issued is either execute a disable command, or if the hard error input to the NCLK becomes set.

A clock is disabled if the last command executed on it was anything other than a restart and the busy bit is reset, or if a clock's bit in the disabled clocks register is reset while the busy bit is also reset. With the exception of the restart command, all commands executed in the Clock Generator will complete with the affected clocks in a disabled state. The state of the busy bit is important in determining a clock's state because during execution (busy bit set) of diagnostic operations, the disabled clocks register will indicate that the clocks are running, even though they will be disabled at the completion of the operation.

A clock is regarded as being in diagnostic mode if the last command executed against it was anything other than a restart or disable and the disabled clocks register bit is set only when the busy bit is set. Note: Diagnostic mode is a transient state and will always terminate (assuming an error does not interrupt the execution) in the disabled state.

Due to a lack of status visibility, there is no bit that can be read by the SPU that will indicate the state of the Master_RBE signal. If the last command executed against it was a restart, then the bit is regarded as free-running (signal is set). If the last commad executed against the Master_RBE was anything other than a restart and the busy bit is reset, then the signal is regarded as disabled (signal is reset). SPU software is responsible for keeping track of commands executed against the Master_RBE in order to determine its state.

3.2.3 Diagnostic Mode

Diagnostic mode is defined as issuing a user programmed or command specific number of clocks to selected boards. The busy bit in the command register must be set when determining if a clock is in diagnostic mode, because diagnostic mode is a transient which will terminate and place the specified clocks in disabled mode. Diagnostic mode is defined into two types of operations, burst operations and step operations. Burst operations allow the user to issue a programmable number of 1x clock periods worth of clocks to the selected board (s). Step operations will issue a single 1x clock period's worth of clocks to the BUT. The number of clocks issued to the selected board will be a function of the operation and the clock frequency enabled to the board. A detailed description of the commands that comprise diagnostic mode follows .

Boards are micro or single stepped by programming the Command Register and Diagnostic Enable register in the Clock Generator. The Command Register specifies the step command that will be executed and the Diagnostic Enable register controls which boards are affected by the operation. During single step operations, the boards under test will ordinarily remain in normal mode (See scan controls). Therefore, the Scan Engine does not need to be accessed in order to

execute these operations. The software selects the boards to be tested by writing a one to the each board's corresponding bit in the Diagnostic Enable Register. When the Command Register is written, the boards selected by the DE register will be issued a specified number of clock pulses based on the contents of the step code field in the command register and the selected boards' fields in the Clock Frequency Control Register.

3.2.3.1 Burst Mode Operations

Burst mode operations allow the software to issue a user programmable number of clocks (specified in numbers of 1X clock periods) to the boards specified in the Diagnostic Enable register.This offers the equivilent functionality of being able to set a break point in time. A forty bit counter determines how many system clocks should be issued before the clocks are disabled. By using the burst mode, a user has the ability of bursting clocks up to a point in time just prior to the occurance of an error and then can restart clocks while capturing the state of the error. This feature significantly simplifies the process of debugging errors that occur minutes or hours into the execution of a process. A burst operation of one clock period is the functional equivilent of executing the single step command. As in the single step operation, the selected clocks will maintain their proper phase relationship during execution.

Two types of burst operations are available to the user. These operations are the burst and scan operations specified by contents of the command field of the Clock Generator Command/Status register. The difference between the two operations is the rate of clocks issued to the boards under test. In Burst mode, the clock frequency issued to the boards is specified by the respective boards' code contained in the Clock Generator's frequency control registers. In a scan operation, all selected boards will be forced to a 1X clock rate regardless of the contents of the frequency control registers. Note: the value of the frequency control register is not changed in this operation. It's effect on the clock frequency is merely overridden by the Scan operation.

Burst mode operations are the only diagnostic operations that can be executed while the CCU bit is set in the Command/Status register of the Clock Generator. If Step operations are attempted while the CCU bit is set, unpredicatable behavior will result. During Burst mode operations, the RBE will start out the operation from a reset state. While the command is being executed, the RBE will be set for the period of 1X clock periods specified by the Burst Counter and will then be reset at the same time that the clocks are disabled. By using this feature, the software has the ability to step the memory and I/O system with the processor and XBAR subsystems. Executing a Scan operation while the CCU bit is set, is the method used to perform Dynamic and CCU scan operations.

The Burst Counter registers must be loaded prior to writing the Command/Status register. The act of writing to the CS register loads the value contained in the Burst Counter register into the free running Burst Counter. The operation will continue until the terminal count for the burst counter is set when the counter rolls over at zero. Note: If a Hard Error or Scalar Halt condition is set in the middle of a burst mode operation, the operation will never complete because the state of the counter will be frozen, preventing the terminal count from ever becoming active.

When the terminal count becomes active, all the selected boards' clocks will be stopped just after the rising edge of the 1X clock and the Busy bit in the CS register will be reset. This indicates to the polling SPU that the Burst operation is complete. The clocks will remain disabled until another command controlling the operation of these clocks is executed . If the restart command is given, the clock generator will first resynchronize the stopped clocks with the free running clocks. Once synchronized, the clocks will begin to free run at the rate specified by the Clock Frequency Control Registers.

3.2.3.2 Step Operations

Single and Micro Stepping allow the user to walk through individual events in the system by issuing a command specific number of clock pulses to a single or group of boards in the system. Between these clock operations, the software can non-destructively scan out the state of the board (s) under test to determine if they are operating properly. Stepping the clocks at speed is also used as the second (of 3) step of the CAST process. After data is scanned into the selected board (s), two clocks are issued to allow the data to pass through the combinational logic that connects register stages.

The first operation, micro-stepping, is regarded as a single board operation where the phase relationship of the stepped clock with respect to the other clocks in the system is unimportant. The hardware has the ability to send one or two clock pulses to the selected board (s). The rate of the clock pulses is determined by the contents of each board's clock rate register. 1x, 2x, and 3x cloc rates are all supported by the microstep function. The four microstep commands are described in detail in the Clock Generator register definition. The disadvantage of this mode is that the 1x, 2x and 3x pulses are all sent out on the same edge. Therefore, if the microstep is repeated more than once, the phase relationship between the clocks will change. This occurs because clock pulses are issued to selected boards by enabling clocks for a portion of a single 1X clock period. Depending on the number and frequency of clocks that will be issued, the window can only exist in certain parts of the period. For this reason, the window for a specific microstep operation will always begin and end at the same point of the 1X period. Because there are no utilities in hardware and software to keep track of individual board phases during the microstepping of multiple boards, great care must be exercised to keep accurate track of these phases.

Single stepping allows one system clock (1x) period worth of clocks to be issued to the boards under test. Therefore, a board receiving a 3X clock would receive 3 clock pulses (at speed), a 2X clock two pulses, and a 1x clock one pulse. The advantage of this mode is that boards operating at different clock rates will maintain their proper phase relationship regardless of how many times the command is executed. Restricting the user to sending out clocks in 1x time increments should not be a problem, because most interaction between boards occurs on 1x (system) clock boundaries.

3.3 Scan Engine and Clock Generator Interfaces

3.3.1 Scan Engine Interfaces

3.3.1.1 NWI Interface

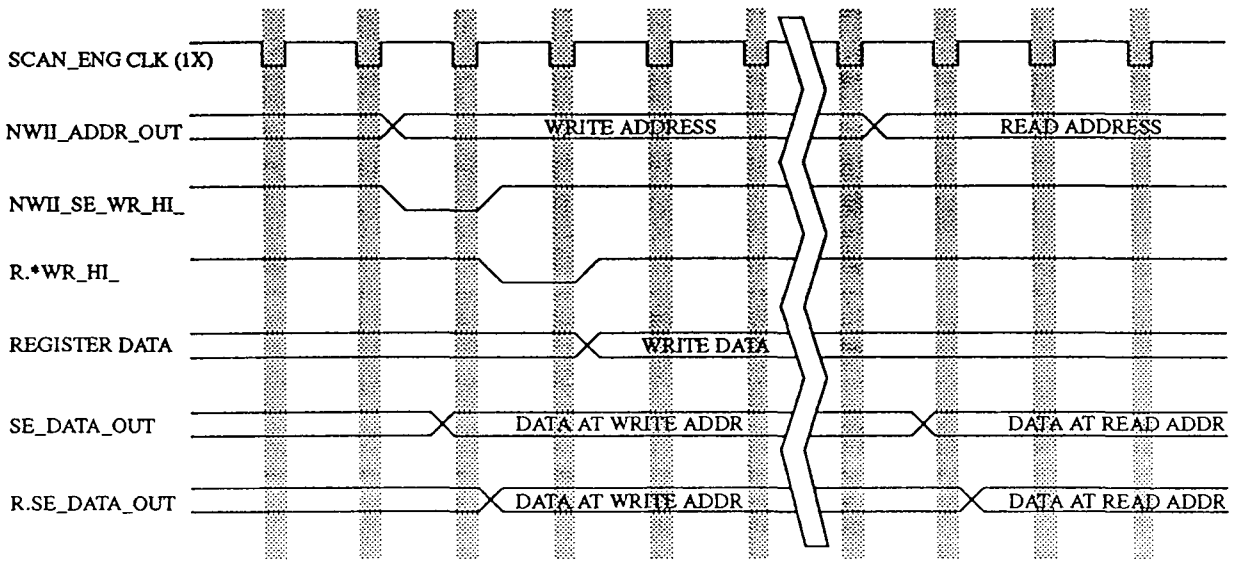
The NWI Interface to the Scan Engine is path used to access the Scan Engine's registers and the Scan Memory. The Scan Memory is accessed through this path because the Scan Memory Bus Master multiplexor is part of the logic of the Scan Engine. Writes and reads are simple accesses that are driven by the NWI. No acknowledge is required for accesses to either the Scan Engine or the Scan Memory. A write cycle is a three clock process and a read is a two clock process.

Table 3-5: NWI Interface Signal Description

Name	Description
nwii_addr_out<15:0> nwii_addr_out2<7:0> nwii_addr_out3<15:0>	Address bus from the NWI Interface. Due to heavy loading, especially 7:0, multiple copies are needed.
nwii_data_out<31:0> nwii_data_out2<31:0> nwii_data_out3<31:0>	Data bus from the NWI Interface. Due to heavy loading, multiple copies are required.
nwii_data_par<3:0>	Parity bus for the above data bus. One bit is assigned to each byte of the 32 bits.
nwii_cg_wr_hi	Clock Generator write strobe for the upper 16 bits of data. Used to load Clk Cmd Shadow register.
nwii_cg_wr_lo	Clock Generator write strobe for the lower 16 bits of data. Used to load Clk Cmd Shadow register.
nwii_se_wr_hi	Scan Engine write strobe for the upper 16 bits of data.
nwii_se_wr_lo	Scan Engine write strobe for the lower 16 bits of data.
nwii_mem_sel_hi*	Scan Memory master chip select for the upper 16 bits of memory.
nwii_mem_sel_lo*	Scan Memory master chip select for the lower 16 bits of memory.
nwii_mem_wr_hi*	Scan Memory master write enable for the upper 16 bits of memory.
nwii_mem_wr_lo*	Scan Memory master write enable for the lower 16 bits of memory.

During the first clock of a write cycle, the address and the data are driven to the Scan Engine. During the second clock, the control signal (s) are asserted and during the third clock, the control signals are reset completing the transfer. All reads in the Scan Engine and the Scan Memory are fall through accesses that require no control signal assertion. During the first clock cycle, the address is driven to the Scan Engine and Memory, which causes the Scan Engine's read mux to select the appropriate register and the Scan Memory to perform a read cycle. The Scan Memory's control lines are set, so that a read is performed every cycle from the location specified by the nwii_addr_out bus. During the second cycle the read data becomes valid and is loaded into the output register. For both types of accesses, the address (and data in the case of a write) will remain valid until the SPU's next access. An example of a write followed by a read is illustrated below.

Figure 3-8: Scan Engine Write and Read Access Timing



3.3.1.2 XCL Data Interface

The XCL Data Interface is the path through which all data transfers between the NWI and XCL occur. With the exception of the write strobes is a registered extension of the Scan Engine's NWI interface. The write strobes are generated by decoding the NWI address and write enables. These decoded signals are then registered before being transferred to the XCL

Write and read accesses are also the same as the Scan Engine. A write access is a four clock process. The address and data are presented to the board on the first clock and are registered on the XCL on the second clock. The XCL's decoded write strobe becomes active on the third clock, and on the fourth clock, the stobe is reset. A read access is a two clock operation. On the first clock, the address is presented to the board. The registered address is used to drive the select lines on the read multiplexor on the second clock, selecting the the data to be read. The read data is registered at the output stage and sent back to the NCU during the third cycle. An illustration of both cycles is illustrated below.

Table 3-6: XCL Data Interface Description

Name	Description
xc_cu.xcl_dato<31:0>	Read data bus coming from the XCL. For ease of implementation, the data is read back through the Scan Engine read mux.
xc_cu.xcl_dato_par<3:0>	Parity bus for the above data bus. Each parity bit is assigned to one byte of data.
cu_xc.xcl_data_in<31:0>	Write data for the xcl. This is a registered copy of the nwii_data_out bus.
cu_xc.xcl_dati_par<3:0>	Parity bus for the above data bus. Each parity bit is assigned to one byte of data.
cu_xc.xcl_addr<5:0>	Address bus from the NWI to the XCL. This is a registered version of nwii_addr_out. All addresses are 32 bit data addresses.
cu_xc.xcl_addr_par	Parity for the above address bus.
cu_xc.xcl_write_hi*	Write enable for the upper 16 bits of data transferred from the NCU to the XCL.
cu_xc.xcl_write_lo*	Write enable for the lower 16 bits of data transferred from the NCU to the XCL.

3.3.1.3 Scan Memory Interface

The Scan Memory interface is the Scan Engine's connection to the Scan Memory which contains the scan data. With the exception of the smem_data_out and smem_perr_data busses, all the signals contained in the interface are driven by the Scan Engine. They are sourced by the fifty four bit two:one multiplexor which selects whether the NWI or the Scan Engine is Master of the Scan Memory interface. When the busy bit is set in the Scan Engine's command/status register (bit 16) , the Scan Engine is master of the bus. Otherwise, the NWI has control over the bus. A description of the interface, is contained in table 4.

Table 3-7: Scan Memory Interface Description

Name	Description
smem_data_out<31:0>	Data bus from the Scan Memory containing read data which will be loaded into Scan Engine registers.

Table 3-7: Scan Memory Interface Description

Name	Description
smem_perr_data<31:0>	Parity error log register for the Scan Memory. For ease of implementation, its contents are read back through the Scan Engine.
smem_addr_in<14:2>	Address bus for the Scan Memory. All addresses are 32 bit addresses.
smem_addr_par	Parity bit for the above address bus.
smem_data_in<31:0>	Data bus for the Scan Memory which contains data to be written to the Scan Memory.
smem_dati_par<3:0>	Parity bus for the above data bus. One parity bit is assigned to each byte of data.
smem_mem_sel_hi*	Scan Memory master chip enable for the upper 16 bits of data.
smem_mem_sel_lo*	Scan Memory master chip enable for the lower 16 bits of data.
smem_mem_wr_hi*	Scan Memory master write enable for the upper 16 bits of data.
smem_mem_wr_lo*	Scan Memory master write enable for the lower 16 bits of data.

3.3.1.4 Clock Generator Interface

Please refer to the Scan Engine Section of the Clock Generator Interface Portion of this document (Sec 3.2.2.3) for the description of the interface.

3.3.1.5 Run Bit Enable Interface

The Run Bit Enable Interface contains the signals that control the clock gating of the CCU and dynamic scan ring clocks on various boards in the system. When one of these bits is asserted, the clocks (to the ring) associated with this bit are running. When the bit is reset, the clocks are disabled. In the case of the CCU run bit enables, the ring clock is the same as the board clock, so toggling the bit will enable and disable the clocks to the CCU specified by the given bit. Since the NMB clocks are regarded as free running, the nmb_rbe bits are additionally used as the vehicle used to step the board.

Each of these bits is the product of the master run bit enable and a status bit in the Scan Engine. The status bit drives the select line to a 2:1 multiplexor which drives the bit out to the interface. The two inputs to the multiplexor are the master run bit enable and the registered version of the multiplexor output. This allows the software to either hold the state of a given run bit enable (status bit reset), or to control its operation through manipulation of the master run bit enable.

Table 3-8: RBE Interface Description

Name	Description
cu_xc.nia_ccu_rbe<39:0>	Run Bit Enables for clock gating to the 40 possible CCUs in a system. 8 are assigned to ia8 (7:0) and 4 to each port in the system (39:8, ports7 to 0).
cu_xc.nia_ccu_cntrl_ena<8:0>	CCU control ring enables for each possible NIA in the system. Bit position corresponds to port assignment.
cu_xc.nia_slog_ena<8:0>	Soft Error Log ring enables for each possible NIA in the system. Bit position corresponds to port assignment.
cu_xc.nmb_rbe<15:0>*	Log and Sys ring enables for the NMB board. Lower 8 are log enables and upper 8 are sys enables. Bit position in the byte indicates port location.
cu_xc.xbar_rbe<5:0>	Configuration ring enables for the 6 XBAR boards (no XCL). See software description for bit assignments.

3.3.1.6 Master Scan Interface

The Master Scan interface contains all the status and data lines necessary for the XCL to set up data pipelines and transfer data between the NCU and the other boards in the system. With the exception of the scan read data line, all the signals in this interface are sourced by the NCU. The scan write data, master scan controls, and the Master_s0 signal are transferred out to the boards in the system and are not used to control scan data flow in the XCL. The remaining signals in the interface are used to inform the XCL what type of scan operation is being attempted and which boards are involved. The XCL uses this status to adjust the length of data pipelines (will change for different scan operations) and to route data between the Scan Engine and the board (s) under test.

Table 3-9: Master Scan Interface Description

Name	Description
cu_xc.scan_ctl<2:0>	Master Scan Control which is gated with each board enable to form a board's scan control lines.
cu_xc.scan_wr_dat	Scan Write data line. This signal is buffered and sent to the scan input of each board in the system.
xc_cu.scan_rd_dat	Scan Read data line. This is the registered output of the XCL's scan read/ loopback multiplexor.
cu_xc.xcl_mux_ctl<5:0>	Scan Read multiplexor select lines used to select which board's scan output is transmitted to the Input Scan Engine.
cu_xc.xcl_loopback	When this line is asserted, scan write data is looped back to the Input Scan Engine. Has priority over mux controls.
cu_xc.recirc	Scan recirculation line, which when asserted causes the board output selected by the read mux to be fed back to all board scan data inputs.
cu_xc.nia_log_scan	NIA log/sys scan indicator. When set, scan data pipes in the XCL are adjusted for proper data synchronization.
cu_xc.log_scan	Log or sys ring scan indicator. When set, scan data pipes in the XCL are adjusted for proper data synchronization.
cu_xc.master_s0	CCU state 0 clock phase signal. The signal is set while the NCLK phase monitor is in state 0. It is buffered and sent to all NIAs to synchronize the CCU clock phase monitors.

3.3.2 Clock Generator Interfaces

The Neptune Clock Generator Array has six unique interfaces that communicate with on-board NCU logic and the other boards in the system. The NWI interface allows commands and status to be passed between the SPU and the Clock Generator. The Error and Control Interface contains the only external signals that have a significant effect on the internal operation of the Clock Generator array. The Scan Engine interface transfers operating status from the Clock Generator to the Scan Engine so that proper data and clock synchronization can occur. The outgoing clock interface provides clocks to all the boards in the system. The Oscillator interface contains the highest speed logic in the Neptune system, providing the clocks which drive the Clock Generator array.

3.3.2.1 NWI Interface

The NWI Interface contains the address, data, and control signals required to read and write the Clock Generator array. The table below shows the breakdown of the interface.

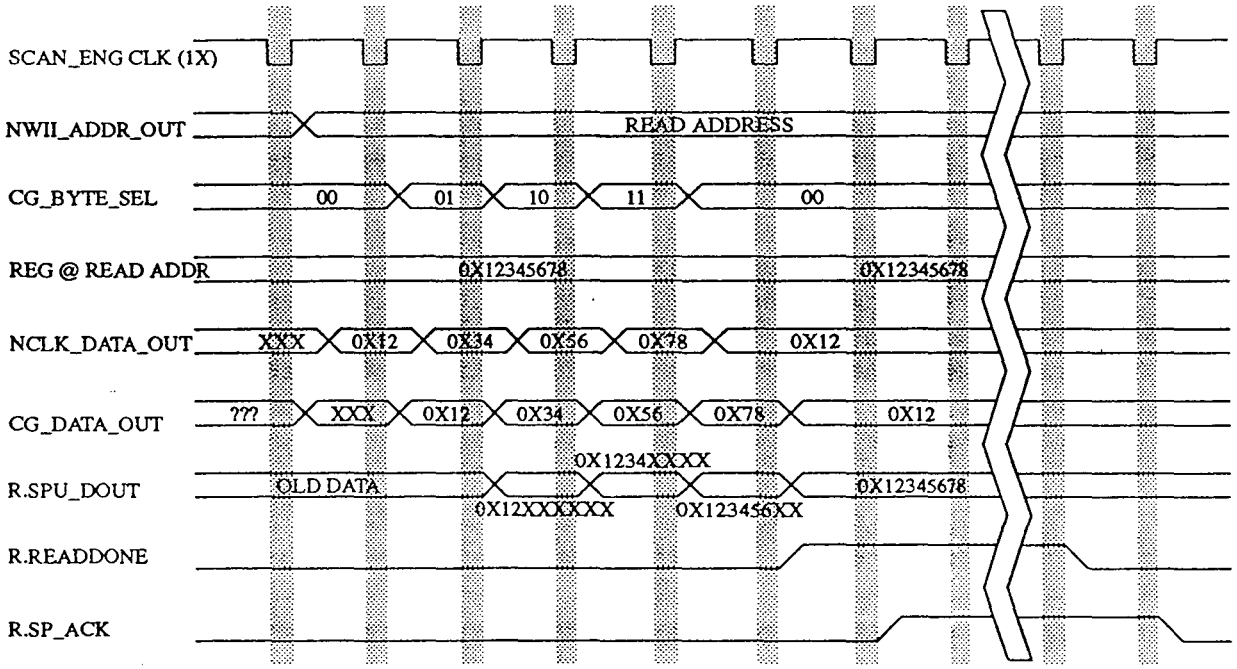
Table 3-10: NWI Interface Signal Definition

Name	Description
nwii_data_out<31:0>	32 bit data bus from NWI Interface
nwii_data_par<3:0>	Parity bits for the above data bus
nwii_addr_out_ga<5:2>	Address bus from NWI Interface.
cg_byte_sel<1:0>	Byte selects used to select which byte of a register is output on the data out lines.
nwii_cg_wr_hi	Write strobe for the upper 16 bits of data
r.cg_wr_lo_gate	Write strobe for the lower 16 bits of data
nclk_data_out<7:0>	Data output from the NCLK to the NWI
nclk_data_par	Parity bit for the above data.
cg_par_err<3:0>	Parity error bus which is set when byte parity errors are detected on NWI data.

A write access to the Clock Generator requires three system clocks, and a read requires five system clocks. During the first cycle of a write, the address and data are presented at the input of the chip. During the second cycle the write line is asserted for one system clock. The rising edge of the write line causes the data to be latched into the control register selected by the address. Data and Address remain valid for one more clock cycle. A read access is significantly more involved than a write access because the NCLK array can only output data a byte at a time, and this data must be reformed into a 32 bit word before being sent back to the SPU. During the first cycle, the address is asserted. During the next four cycles the address will remain asserted while the byte select lines cycle through their four states.

The four bytes are formed into the 32 bit word at the register that receives the output of the SPU Interfaces read data mux. The 32 bit read mix is broken up into bytes, each of which having unique select lines. These select lines allow each 8 bit mux to select either the output of the Clock Generator or the output of the register immediately following the read mux. The mux switches from the Clock Generator to the register as the byte selects cycle through their four states. As the register is selected, this effectively holds the state of each byte until the entire word is formed. Once this accomplished, the readdone signal is asserted which causes the spu acknowledge to be asserted.

Figure 3-9: Clock Generator Read Access Timing



3.3.2.2 Error/Control Interface

The Error and Control interface for the Clock Generator contains the five interface signals that have a significant impact on the internal operation of the NCLK array. The five signals and their descriptions are listed below.

Table 3-11: Error/Control Interface Signal Definition

Name	Description
cg_delay<3:0>	Control bus which specifies the number of 1x clock periods between when the RBE and clocks are issued.
r.hard_err	Error signal which indicates that an unmasked hard error has been asserted in the system.
r.scalar_halt	Error signal which indicates that an unmasked stop_cntr signal has been asserted by a scalar processor.

3.3.2.3 Scan Engine Interface

The Scan Engine interface contains operational status signals that are used by the Scan Engine to properly sychronize data with clocks issued to the board under test. These three signals and their descriptions are listed below.

Table 3-12: Scan Engine Interface Signal Description

Name	Description
dscan_god	Control signal which is asserted 1-1X clock before a scan operation and remains asserted through the operation
dmaster_rbe_raw	Master Run Bit Enable which is used to create clock gate signals for CCUs and boards containing Log/Sys rings.
dmaster_s0_raw	Control signal which is asserted while the nclk phase monitor is in state 0, and is used to sync the phase monitors in the NCLK, Scan Engine and NIA.

3.3.2.4 Outgoing Clocks Interface

The Outgoing Clocks Interface is the largest of the Clock Generator's interfaces, containing the forty differential clocks that are issued to the boards in the system. With the exception of the NCU, NIA, and the XCL boards, each board will receive one of these clock pairs which drives the board's clock distribution tree. The exception boards each receive two sets of clock pairs. These clock outputs are controlled by writing commands to the NCLK's command/status register (through the NWI Interface) while specific bits in the diagnostic enable registers are set (See Clock Generator section of the Software Description).

Each of the above clock outputs is matched in length in order to minimize the skew between any two clock outputs. The clock path is matched from bond on the NCLK die to the via farm on the receiving board. The clock includes not only routed wire, but bond wire in the nclk array, escape etch, and flex circuit in the Augat and Dupont connectors. Although indviduals sections of the clock path may be different lengths, the overall path must be matched in order for clock skew to be at a minimum.

3.3.2.5 Oscillator Interface

The Oscillator Interface contains the six inputs that are used to drive the internal clock logic of the Clock Generator. The six inputs are divided into three pairs of differential ECL 100K signals. The S, N, and F suffixes of these signals (OSCS(*), OSCN(*), OSCF(*)) indicate which of the three oscillator inputs will be selected by the oscillator switching logic and stand for slow, nominal, and fast respectively. The selected oscillator is regarded as the master 6x clock from which all other clocks (1x, 2x, 3x) are derived. The oscillators required to produce the 15 nS, 16.67 nS, and 18 nS 1X periods (fast, nominal, & slow margin) will be 400 MHz, 360 MHz, and 333 MHz respectively.

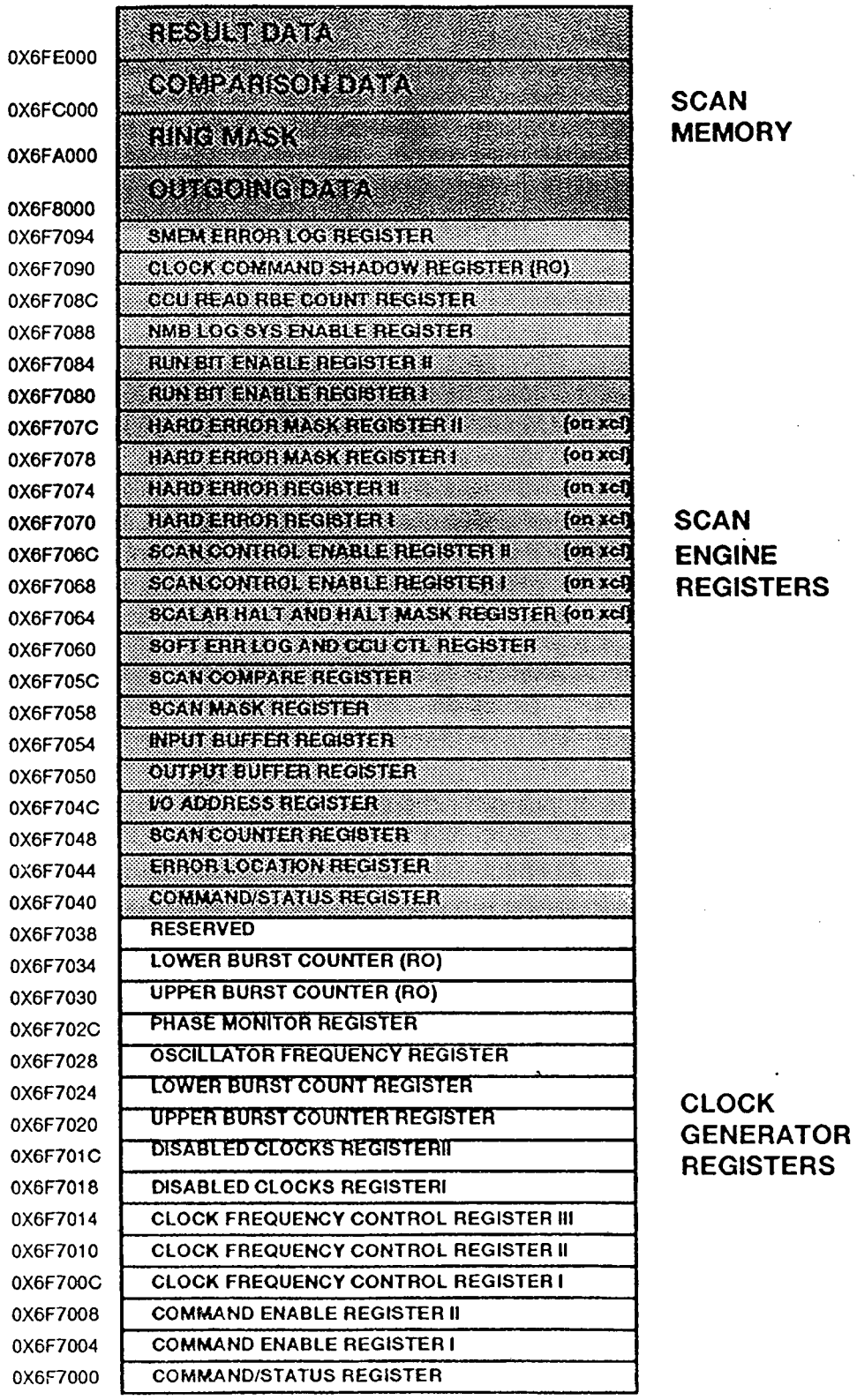
3.4 Software Description

This section covers the software description of the scan engine and clock generator. This perspective is from the software running on the SPU accessing registers on the NCU.

3.4.1 System Memory Map

The Scan Engine and Clock Generator Subsystem are accessed as a block of 8224, 32 bit wide locations in memory. The Block of memory is divided into 3 logical groups, Scan Memory, Clock Generator, and Scan Engine. The Scan Memory occupies the first 8192 words of the memory space. Only half of this will be utilized by the Neptune system. The upper half of the memory space is reserved for future expansion of the scan ring size. The next 16 words in memory contain the control registers of the Clock Generator. The final 16 words contain the Scan Engine control registers. The Memory Map for the Scan Engine and Clock Generator subsystem is illustrated in Figure 3-1.

Figure 3-10: Scan Engine and Clock Generator Subsystem Memory Map



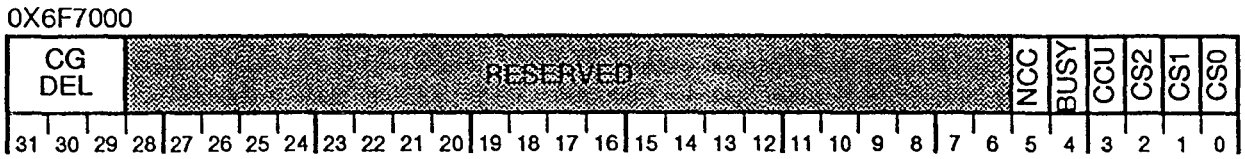
3.4.2 Register Descriptions

3.4.2.1 Clock Generator

3.4.2.1.1 Command/Status Register

The Command/Status Register is a 32 bit read/write register that controls the operation of the clock generator. Of the 32 available bits, five bits are used to decode the desired command and transmit command completion status back to the SPU. The five bits are divided into three fields, the Busy field, the CCU field, and the three bit Command Select field (CS2-CS0). Writing to the Command/Status register causes the command specified by the Command Select field to be immediately executed. The organization of the Command/Status Register is illustrated in Figure 3-11

Figure 3-11: Command/Status Register



The Command field is a three bit read/write field that contains the encoded command for the Clock Generator. By programming this field, the SPU selects one of the eight available commands to be executed. All eight command codes contain valid instruction codes. The eight available commands and their corresponding codes are illustrated in table 10.

The CCU bit acts as a Command Enable for the Clock Generator's run bit enable. When the bit is set, the hardware will include the run bit with the clock outputs that have been selected by the command enable registers. However, unlike the clock outputs, there is only a limited number of legal commands that can be executed as CCU commands. The only legal commands that can be executed with the CCU bit set are Restart, Stop, and Burst. All other commands are not guaranteed to yield expected results.

The Busy bit is a one bit read only field in the CS register that indicates the command execution status of clock generator. After the SPU writes a command to be executed to the CS register, the Control Engine will set this bit (1) indicating to the SPU that the command is in the process of being executed. The Busy bit will remain set until the command is completed, when the Control Engine resets the bit.

The NCC (No Clock in Cable) bit is a one bit field that adjusts the control pipelining of the DSCAN_GOD signal. It is set when the period of the 1x clock cycle is longer than the delay of the clock path. Setting the NCC bit removes one stage of command pipelining so that the Scan Engine will operate properly from the "dead zone" of 1x clock periods until DC. Under normal system conditions (nominal oscillator, production clock cable length) the bit will be reset. Note: this bit is not readable by the SPU.

The CG_DEL field is a three bit field that specifies the number of 1x clock periods that will exist between when a command will execute on a clock output and when it will execute on the Master RBE output of the Clock Generator array. Under normal operating conditions, this field will be set to a binary value of 010. With this value, the Master RBE will execute a clock command 2-1x clock periods before the command is executed on a clock output.

Table 3-13: Clock Generator Command Codes

Command Code	Command Selection
000	Disable Clocks
001	Burst Operation
010	Scan
011	Restart Clocks
100	Step One Clock
101	Step Two Clocks (2X or 3X rate)
110	Step Two Clocks (1X rate)
111	System Step

The Stop command (Code = 000) causes all clocks drivers specified by the Command Enable register to be disabled. The specified clocks will all be disabled on the rising edge of the 1x clock. The selected clocks will remain disabled until a restart command is executed while they are selected.

The Burst command (Code = 001) instructs the Clock Generator to free run the selected clocks for a programmable number of 1X clock periods. The number of clock periods is specified by the contents of the Burst Counter register. After the command is executed, the Control Engine sets the Busy bit in the CS register, and enables the selected drivers. During execution of this command, all of the selected clocks will maintain their correct phase relationship with respect to one another. When the Burst Counter reaches a count of 0, the Busy bit is reset and all of the selected clocks are disabled on the rising edge of the last 1X clock.

The Scan command (Code = 010) instructs the Clock Generator to issue a programmable number of 1X clocks to the slots specified by the Command Enable register. The number of clocks to issue is specified by the contents of the Burst Counter Register. This command overrides, but does not change the contents of the Clock Frequency registers. Once the command is executed, the Control Engine sets the Busy bit in the CS register and enables the drivers. When the Burst Counter reaches a count of 0, the Busy bit is reset and the selected clocks are disabled on the last rising edge of the 1X clock.

The Restart command (Code = 011) causes clocks selected by the Command Enable register to be synchronized to the free running clocks and enabled as free running clocks. This command will be principally used to restart disabled clocks after a diagnostic function has been performed. After the command is executed the phases of the disabled and free running clocks will be properly aligned.

The Step One Clock command (Code = 100) instructs the Clock Generator to issue a single clock to boards selected by Command Enable register. Each board selected will simultaneously receive a low going pulse equal to the period of the oscillator clock (6X). This is equivilent to issuing the clock which occurs just before the rising edge of the 1X clock (see figure 4-1). Since all the selected boards receive their rising clocks at the same time, boards receiving different rate clocks would not maintain their phase relationship for multiple executions of this command. Therefore, this command should be restricted to boards that have the same clock rate.

The Step Two 2X or 3X clocks (Code = 101) causes the Clock Generator to issue two clocks (2x or 3x rate) to the boards selected in the Command Enable register. The type of clock issued is determined by the contents of the Clock Frequency Control Register. The principle use of this command is CAST testing boards which ordinarily receive 2X or 3X clocks in the system. Note: If this command is executed on a board receiving a 1X clock, only one clock period will be issued. The two clock periods that are issued are the periods that occur immediately before and after the rising edge of the 1X clock (see Figure 4-1). Since the same two clock periods are issued for each execution of the command, boards receiving different clock rates will lose their phase relationship during multiple executions of this command. Therefore, this command should be restricted to boards that have the same clock rate.

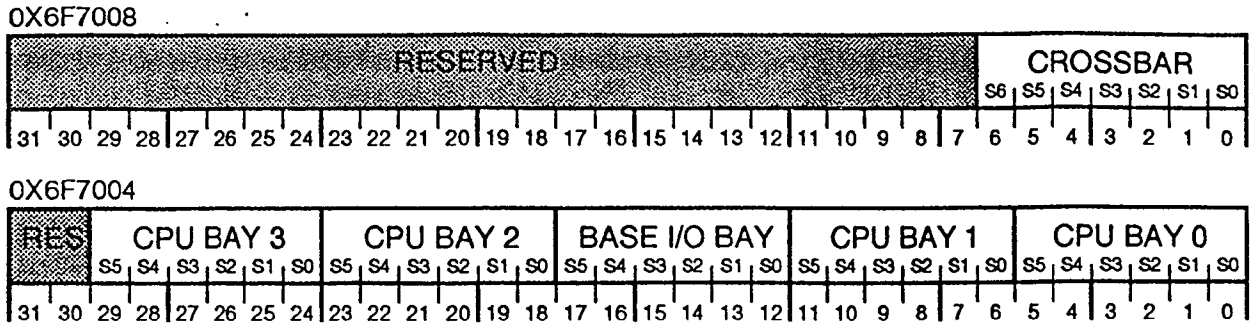
The Step Two 1X clocks (Code = 110) commands the Clock Generator to issue two 1X clocks to boards selected by the Command Enable register. The principle use of this command is CAST testing boards which ordinarily receive 1X clocks in the system. Note: The Control Engine assumes that this command will only be executed on boards that have 1X specified as their clock rate in the Clock Frequency Command register. Boards with 2X or 3X specified as their clock rate will receive 3 and 4 periods of clocks respectively. This occurs because The window that allows clocks to be released starts at the falling edge of the first 1X clock and continues until the rising edge of the second 1X clock.

The System Step command (Code = 111) causes the Clock Generator to issue one system clock period worth of clocks to boards selected by the Command Enable Register. The number of clocks issued is determined by the contents of the Clock Frequency Control register. 3X boards will receive 3 clocks, 2X boards 2 clocks, and 1x boards 1 clock. The phase relationship between the three clock rates is preserved in this command, and may be repeatedly run on boards of different rates.

3.4.2.1.2 Command Enable Registers

The Command Enable Registers are two 32 bit read/write register which are used to place boards into the clock mode specified by the Command Register. One enable bit is assigned to each slot in the system. If a board's bit is set in the Command Enable Register, the board's clock will perform the function specified in the Command Status register. Boards that are not selected will be in either a disabled or free-running state, and will continue in this state during and after the execution of the current command. Because of clock disabling and resynchronization restrictions, it is important that boards are taken in and out of diagnostic mode at the same time. Therefore, the Command Enable register must be programmed before the command register. If the command register is written first, the boards selected by the command enable bits will be placed in diagnostic mode before the SPU has a chance to write the correct values into the enable register. The organization of the Command Enable Registers is illustrated in Figure 3-12.

Figure 3-12 : Command Enable Registers



Please note that this register only controls the operation of the Clock Generator. For example, setting up a board for scan in the Clock Generator only sets up a board to receive the programmed number of 1x scan clocks. To completely set up a diagnostic function, such as scan, the SPU must program both the Scan Engine and Clock Generator registers. The command enables for the Engine and Generator are in different locations because there are times when the Scan Control and Clock Diagnostic enables will be different values for the same slot. Single Step and Burst Mode operation are good examples of this. For these operations, the diagnostic bit in the Clock Generator will be set. However, since the board wishes to operate normally (only slower, or for a certain number of clocks) the Scan Control lines will indicate normal operation. This is most easily accomplished by resetting the diagnostic bit in the Scan Control Enable Register.

3.4.2.1.3 Clock Frequency Control Registers

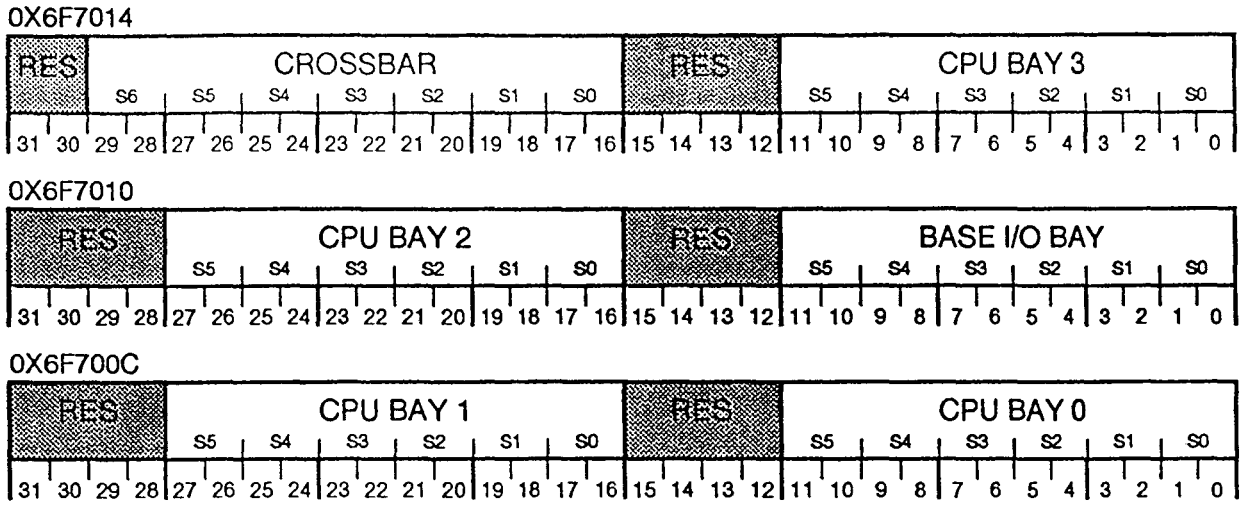
The Clock Frequency Control Registers are three 32 bit, read/write registers that are used to control the frequency of the clocks going to each slot in the system. Two bits are assigned to each slot in a bay. The two bits allow the software to select one of four available clock frequencies: 3x, 2x, 1x, and disabled (Note: these clock rates are relative to the master 6x oscillator frequency). The rate selected determines the frequency output to the selected board during both free-running and diagnostic modes Note: In scan mode, the clock to the selected boards is forced to 1x regardless of the contents of the Frequency Control Register. The four control codes and their corresponding frequencies are shown in the table below.

Table 3-14: Clock Frequency Control Codes

Code	Corresponding Frequency
11	3X Clock (Osc/2)
10	2X Clock (Osc/3)
01	1X Clock (Osc/6)
00	1X Clock (Osc/6)

The clock rate of a specific slot should only be changed if the clock to it has been disabled by executing the stop command while the slot's diagnostic enable bit is set. This is because the output of this register connects to the select lines of a mux that drives the clocks to the output stage. Programming this register will result in immediate switching of the clocks, which may cause glitching at the output. To bring the clocks up, the SPU will write the correct values into the Clock Rate Register followed by executing a restart command in the command register. The organization of the Clock Rate Registers is shown in Figure 3-13.

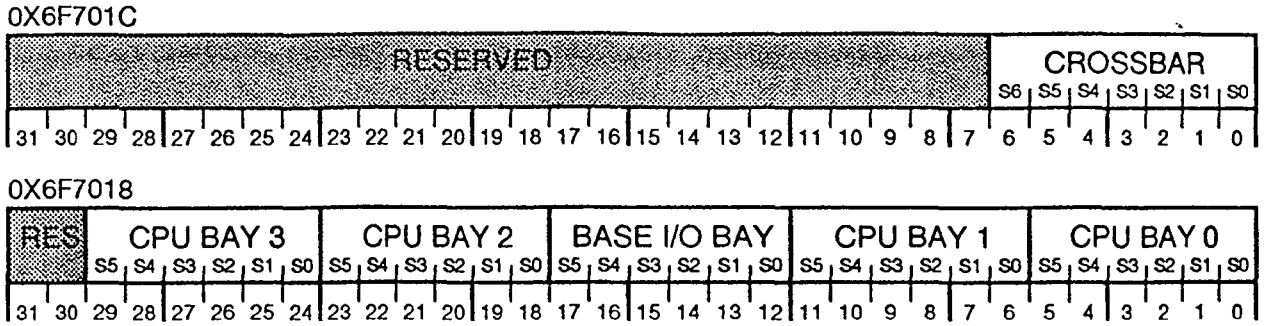
Figure 3-13 : Clock Frequency Control Registers



3.4.2.1.4 Disabled Clocks Registers

The Disabled Clocks Registers are two 32 bit read/write registers that indicate which clock generators have been disabled from transmitting clocks to the system boards. Each slot in the system is assigned a bit in one of the two registers. The register is written each time clocks are enabled or disabled in the system. A one in a given bit position indicates that the clock output associated with that bit is issuing clocks. The only exception to this rule is when the Hard Error input to the Clock Generator is set. When this input is set, the stoppable clocks are stopped two clocks after the set bit is detected. When the clocks are stopped, the Disabled Clocks register is not updated and reflects the state of the clocks at the time that the Hard Error was detected. If clocks are being enabled, the register is loaded with the logical ORing of the current register contents (Clocks that are currently enabled) and the contents of the diagnostic enable register (Clocks about to be enabled). When clocks are being disabled, the register will be loaded with the logical ANDing of the previous register contents with the logical inverse of the diagnostic enable register. This causes the bits associated with the clocks being disabled, to be reset. The organization of the Disabled Clocks Register is illustrated in Figure 3-14.

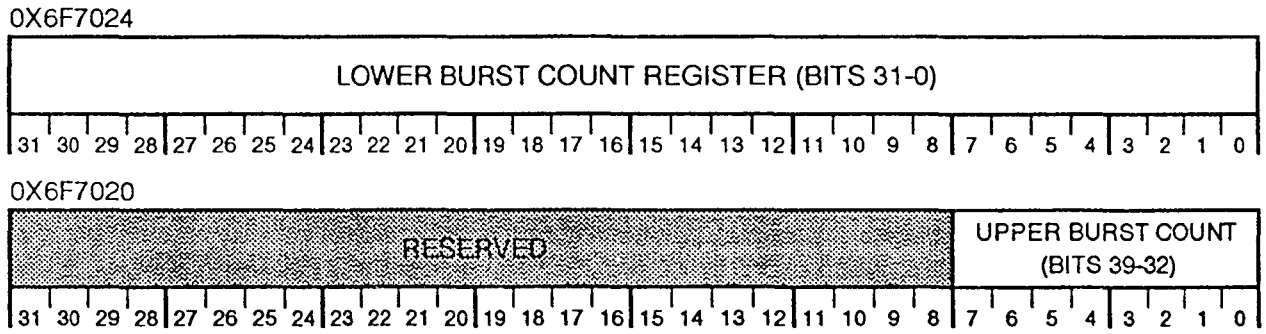
Figure 3-14 : Disabled Clocks Register



3.4.2.1.5 Burst Counter Register

The Burst Counter Register is a 40 bit read/write register that contains the value which is loaded into the 40 bit free running burst counter when a restart, burst, or scan command is executed. The forty bit register occupies two adjacent addresses in the address space of the clock generator. The lower 32 bits in the register are accessed via address 0x6F7024, and the upper eight bits of the register are accessed via 0x6F7020 (see following figure). The value loaded into the counter allows the clock generator to keep track of the number of 1x clock cycles that have passed since execution of the command. The 40 bit counter is free running and decrements by one at the rising edge of the internal 1x clock. During burst and Scan modes, when the counter reaches zero, the terminal count becomes active, which disables the operation and resets the busy flag in the status register.

Figure 3-15: Burst Counter Register



3.4.2.1.6 Spare Clock and Master Oscillator Register

The Spare Clock and Master Oscillator Register is 32 bit read/write register that is programmed in order to select different master oscillators and change the frequency of the free running spare clock. Only four of the bits are used to control these functions. The control bits for the oscillators and spare clock are illustrated below. The oscillator codes control which of the three oscillators are providing the master 6x clock to the clock generator. Programming the osc field of this register allows glitch free transitions between the three oscillators. Changing oscillators is principally used when marginning the machine. The oscillators may be switched while the clock outputs are running or stopped without problems occurring. The codes used to select the three oscillators are illustrated below. The spare clock field of the register allows the software to select the operating frequency of the free running spare clock. The operating frequency select codes follow the same pattern as the controllable clocks and are shown in the following table

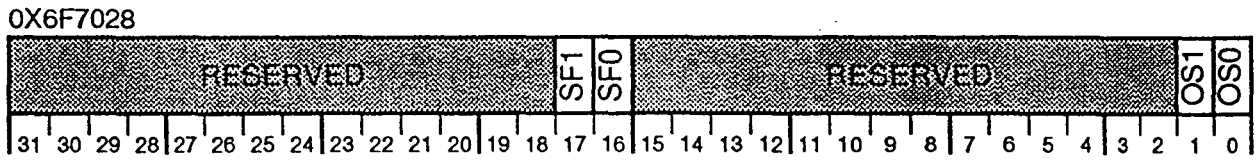
Table 3-15: Master Oscillator Select Codes

Code	Selected Oscillator
00,11	Nominal Oscillator
10	Fast Oscillator
01	Slow Oscillator

Table 3-16: Spare Clock Frequency Select Codes

Code	Corresponding Frequency
11	3X Clock (Osc/2)
10	2X Clock (Osc/3)
01	1X Clock (Osc/6)
00	1X Clock (Osc/6)

Figure 3-16: Spare Clock and Master Oscillator Register

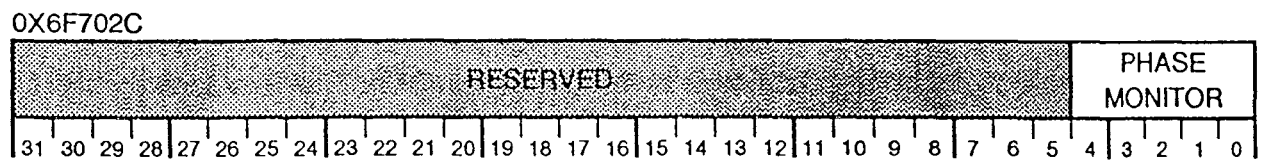


3.4.2.1.7 Phase Monitor Register

The Phase Monitor Register is a 32 bit read-only register that reflects the contents of the phase monitor counter. The value of counter indicates the phase of the ccu clocks. The counter will be either mod 20 or mod 18 depending on the state of the NCLK's CG_DVE input signal. When the CG_DVE signal is set, the counter will be mod 18. The counter is a binary down counter that is reloaded when the counter reaches 0.

The phase counter indicates to the software whether the CCUs are in a scannable phase. The scannable phases for a mod 20 count are 4, 11, and 18. The scannable phases for a mod 18 counter are 3, 9, and 15. The lower 5 bits of the register contain the code that is three states earlier than the NIA and Scan Engine phase monitors, because of pipelining in the Clock Generator. For example, if the phase monitor contains a value of 4, the Scan Engine and NIA phase monitors will be in state 7.

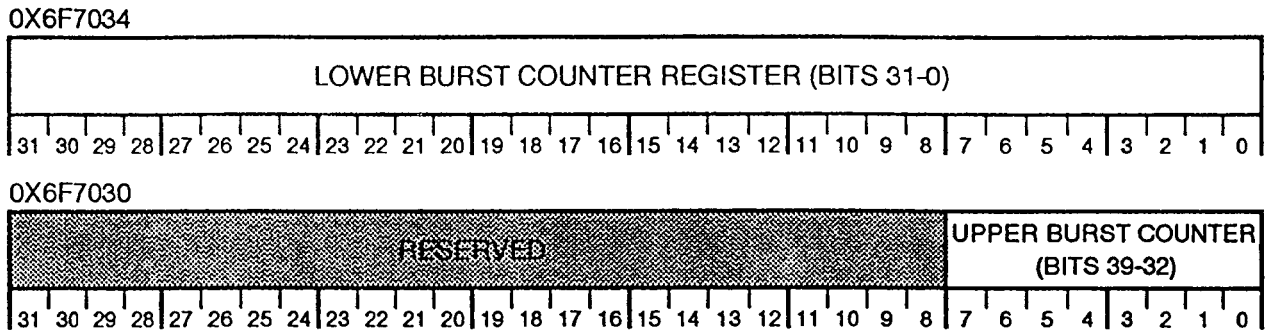
Figure 3-17: Clock Generator Phase Monitor Register



3.4.2.1.8 Clock Generator Burst Counter

The Clock Generator Burst Counter is a 40 bit, read-only register that reflects the state of the "free running" burst counter. The 40 bit register is accessed through two adjacent addresses in the Clock Generator address space. The lower 32 bits of register are accessed through address 0x6F7034, and the upper 8 bits through 0x6F7030. The only useful time to read this register is when the scalar halt or hard error signal is set, because the counter state is frozen during that time.

Figure 3-18: Clock Generator Burst Counter

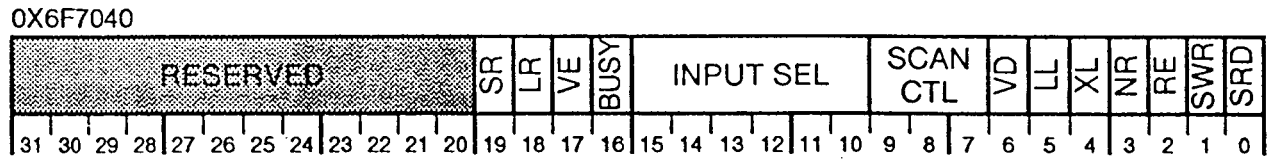


3.4.2.2 Scan Engine

3.4.2.2.1 Command/Status Register

The Command/Status register is a 32 bit read/write register responsible for executing commands and returning completion status to the SPU. The register is divided into 13 fields, 12 of which are commands for the Scan Engine and the thirteenth, Busy, which indicates if the current command has been completed and who is master of the Scan Memory . Writing the commmand register causes the written command to be immediately executed. Therefore, it is important that all enable and counter registers required for the operation are written prior to writing the Command/Status register. The organization of the register is illustrated below.

Figure 3-19: Command/Status Register



The SR bit is the Sys Ring scan enable. This bit is logically ANDed with the master run bit enable and a board's individual sys ring enable to form the NIA's NIA_CCU_CNTRL_ENA signal which is registered and sent off to the NIA under test. The SR bit is also used by the master scan engine to distinquish the difference between a ccu scan operation and a sys ring scan operation.

The LR bit is the Log Ring scan enable. This bit is logically ANDed with the master run bit enable and a board's individual log ring enable to form the NIA's NIA_SLOG_ENA signal which is registered and sent off to the NIA under test. The LR bit is also used by the master scan engine to distinquish the difference between a ccu scan operation and a log ring scan operation.

The VE field is a one bit field that contains a registered version of the logical ORing of the bits in the R.RESULT Register. By checking the state of this bit, the SPU can determine if an error has been detected during the verification of a ring, without having to perform a separate read from the Error Register. Resetting the Error Register will automatically reset this bit.

The Busy field is a one bit field that indicates the Scan Engines command completion status and drives the Scan Memory Multiplexor select line. The Busy bit is read only, being directly driven by the Master Controller. When a command is executed, the Master Controller will immediately set the bit. The SPU can poll the bit to determine if the Scan Engine has completed the programmed operation. When the Scan Operation completes, the Scan Engine will reset the bit, indicating that it is ready to execute another command. When the Busy bit is set, the Scan Engine is master of the Scan Memory address, data, and control buses and may transfer data to the memory. It the SPU attempts to read data while the busy bit is set, unpredictable results will return, because the address and control driven by the SPU are gated off in the Scan Memory interface logic. If a write is attempted the Scan Memory access error flag is set. When the bit is reset, the SPU is master of the bus and may load or analyze the memory contents.

The Input Sel field is six bits wide which controls which board's scan output is connected to the scan engine. The field directly drives the scan engine input multiplexor on the Control Crossbar. Numbers 0-36 are acceptable values to load into the register which map to each slot in the system. Higher values than 36 will result in 0 being transferred to the input of the Scan Engine.

The Scan Ctl field is a three bit field that contains the master Scan Control line that will be output to boards selected by the Scan Control Enable Register. Each element of the bug is logically ANDed with the Scan Control Enable for each board in the system. The output of this AND logic forms an individual board's scan controls that are registered and sent out to each board in the system. Section 3.2.1 of this document provides a detailed description of the Scan Control codes and their corresponding functions.

The VD field is a single bit field that controls whether the Scan Engine performs data verification during a scan or passes the data directly to the Scan Memory, bypassing the mask and compare logic. The bit is connected to the select line of a 32 bit 2:1 mux that drives the result register, that is written to the Scan Memory. When the VD bit is set, the output of the compare logic is written to the result register. When the bit is reset, the contents of the input buffer are written to the result register.

The LL field is a one bit field that forces the scan engine to perform a local loopback. The signal drives the select line on the two input multiplexor that is attached to the front of the input shift register. When the signal is set, a stage of the local loopback pipeline (dependant on type of scan operation) is selected, causing all data shifted out of the scan engine can be scanned back in. When the bit is reset, The scan data output from the XCL is connected to the Scan Engine.

The XL field is a one bit field that forces the scan engine to perform a scan loopback at the data multiplexor on the Crossbar Control board (XCL). The XL bit overrides the input select field of the control register and forces the multiplexor to select the input that is connected to buffered version of the scan data output. This can be used to check scan data path integrity across the two bays.

The NR field drives the reset line that goes to all NIAs. When the bit is set, all NIAs in the system will be reset to a known state.

The RE field controls the recirculation of Scan Ring data during a Scan Read operation. If data is not recirculated back to a board that is being read, the ring data and therefore board state is lost. The RE bit after being registered on the XCL drives the select line to a 2:1 multiplexor which selects either NCU write data or a selected board's output as the data that will be buffered and sent to the scan inputs of all the boards in the system. A board's output is selected by the Input Sel field of the Command Status register. When the bit is set, data from the output of the selected board's scan ring will be fed back to the input of all boards. Note: Even though the scan input of all boards will be toggling, only the boards that are selected to scan will make use of this data. When the bit is reset, the scan output of the NCU is buffered and sent to all boards.

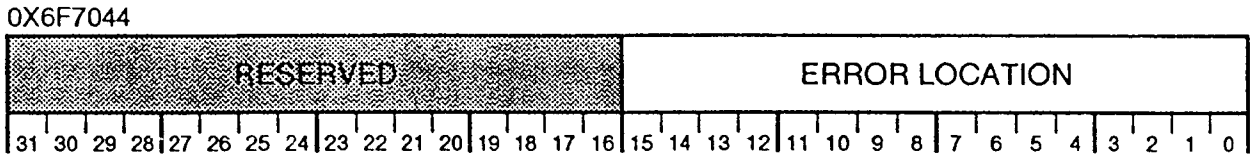
The SWR field is a one bit field that enables the Output Scan Engine allowing system boards to be Scan Written.

The SRD field is a one bit field that enables the Input Scan Engine allowing Scan Reading (verification). If both the SWR and SRD fields are set, the Scan Engine will simulaneously perform a scan read and write operation with the board selected by the Input Sel field. Note: When the SWR and SRD fields are set, the RE field must be reset or new data will not be written to the board.

3.4.2.2.2 Error Location Register

The Error Location register is a 32 bit read/write register that contains the Scan Memory address of the first verification error detected by the input scan engine. When the first error is detected in a scan ring (see Verify Error Register), the contents of the input address counter are written to the Error Location. This will allow the software to quickly find the start of the error data within the Scan Memory. The lower sixteen bits of the register contain the address from the Input Address counter as shown in figure 3-11. The SPU is responsible for clearing the contents of this register.

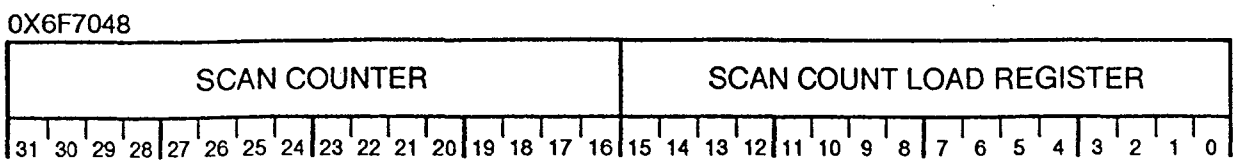
Figure 3-20: Error Location Register



3.4.2.2.3 Scan Counter Register

The Scan Counter Register is a 32 bit read/write register that contains the contents of the Scan Engine's 16 bit Scan Counter and its loading register. Writing to this register loads the Scan Counter's load register with the lower 16 bits of the data bus. The counter is not written at this point. The Scan Counter is written automatically with the contents of the load register when the Command Status register is written. Reading this register outputs the current contents of the Scan Counter on the upper 16 bits and the contents of the load register on the lower 16 bits. Once the Scan operation has been enabled, the contents of this register will decrement when a bit is clocked in and/or out of the Scan Register shift registers. On a normal board scan operation, this will be each 1X clock. On a CCU scan, however, this will occur on either a 7-6-7, 6-6-6 1X clock pattern, dependant on the state of the CG_DVE. The organization of the register is illustrated below.

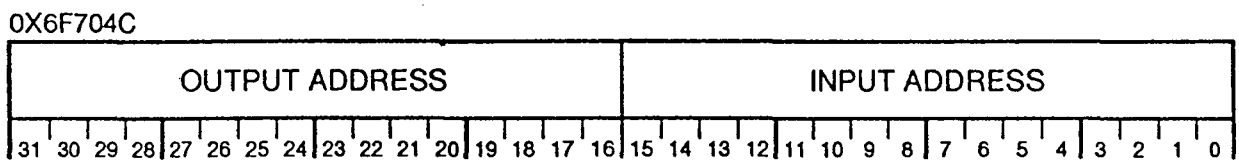
Figure 3-21: Scan Counter Register



3.4.2.2.4 I/O Address Register

The I/O Address Register is a 32 bit read/write register that contains the contents of the Scan Engine's Input and Output address counters. The Output counter is used to access scan data that will be loaded into the output scan buffer and shifted out to boards in the system. The Input counter is used to read the mask and compare data and to write the result data to the Scan Memory. The register is divided into two sixteen bit sections. The Output Counter occupies the upper sixteen bits of the register, and the Input Counter occupies the lower sixteen. Writing data to this register causes the input and output counters to be loaded with input data. Reading this register outputs the current value of both counters. Reading the counters before enabling them will allow the software to verify proper loading of the register.

Figure 3-22 : I/O Address Register



3.4.2.2.5 Output Buffer Register

Address = 0x6F7050

The Output Buffer Register is a 32 bit read/write register containing the scan data that will be shifted out to the boards in the system. During scan write operations, data from this register will be loaded into the output shift register after the 32nd bit has been shifted out of the output shift register. After data has been loaded into the shift register, a bit will be set indicating that the output scan pipe is no longer full. The output scan engine will reload the register from the outgoing page of the Scan Memory, increment the outgoing address counter, and reset the not full flag. The most significant bit in the register contains the data that will be shifted out of the shift register first.

3.4.2.2.6 Input Buffer Register

Address = 0x6F7054

The Input Buffer Register is a 32 bit read/write register that contains the raw scan data loaded into the input shift register. During scan operations, this register is loaded by the Scan Engine after 32 bits of data have been shifted into the input register. Since the raw scan data may contain data fields that are not of interest to the software, the output of this register is input to the mask and compare logic, where fields are stripped off

3.4.2.2.7 Scan Mask Register

Address = 0x6F7058

The Scan Mask Register is a 32 bit read/write register that contains the mask data for Scan Ring Verification. During scan ring verification operations, the Scan Mask register is loaded with mask data from the Scan Memory by the Scan Engine. The contents of the Scan Mask register are logically ANDed with the contents of the input buffer and compare registers. Loading a zero into a given bit position in the Scan Mask register will force the corresponding bit in the Result register to a "0". This effectively masks off certain fields within scanned data and eliminates them from the verification process.

3.4.2.2.8 Scan Compare Register

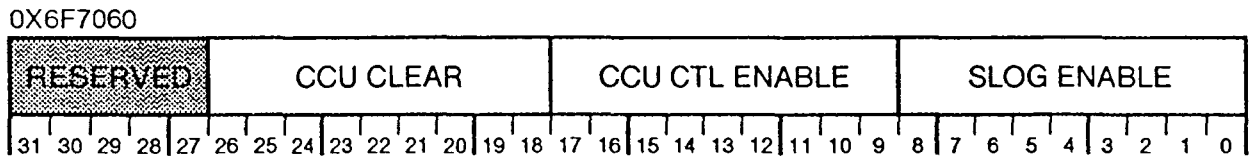
Address = 0x6F705C

The Scan Compare Register is a 32 bit read/write register that contains the comparison data for Scan Ring Verification. This register is ordinarily loaded from the Scan Memory by the Scan Engine. The contents of this register are exclusive ORed with the input scan data after the unwanted fields have been masked off (see Mask Register), and loaded into the result register. The output of the result register is checked for errors (see Verify Error Register) and loaded into the Scan Memory.

3.4.2.2.9 Soft Error Log and CCU Control Ring Register

The Soft Error Log and CCU Control Ring Register contains information that allows individual NIA SLOG and CCU CTL run bits to be controlled, and directly drives the CCU_CLEAR lines. Each field in the register is nine bits wide, one for each port that an NIA may exist in. The lowest significant bit in the field corresponds to port 0 in the system, and the most significant bit to port 8, which is the primary NIA (I/O Bay resident). The SLOG ring enable is used to select which NIA will be performing a Soft Error Log scan operation. An individual NIA's SLOG bit will be set when its enable, the cmd/stat's log ring bit, and the master rbe signal are all set. The CCU CTL ring enable is used to select which NIA will be performing a CCU scan operation. An individual NIA's CCU CTL bit will be set when its enable, the cmd/stat's sys ring bit, and the master rbe signal are all set. The CCU_CLEAR field directly drive the CCU_CLEAR lines which are used by the nia to reset the CCUs attached to it. When a bit in this field is set, the CCUs attached to the selected NIA will all be returned to a known state.

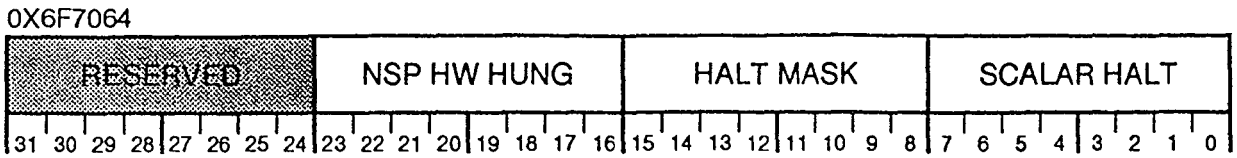
Figure 3-23: Soft Error Log and CCU Control Ring Register



3.4.2.2.10 Scalar Halt and Halt Mask Register

The Scalar Halt and Halt Mask Register is a 32 bit read/write register that is used to read, mask and clear the state of the Scalar Processor Halt signals. The register is divided into three fields: The scalar halt state, its associated mask, and the read-only NSP HW HUNG field. The Scalar Halt state contains the state of the NSP Stop_cntr signals after masking. This field is loaded on every clock cycle until one of the bits becomes set. After that, the register will hold state until it is cleared by a write access from the spu. The Scalar Halt mask is used to mask off individual Scalar Halt signals before the error detection logic. The NSP HW HUNG field reflects the state of the hw_hung signals are received from each scalar processor in the system.

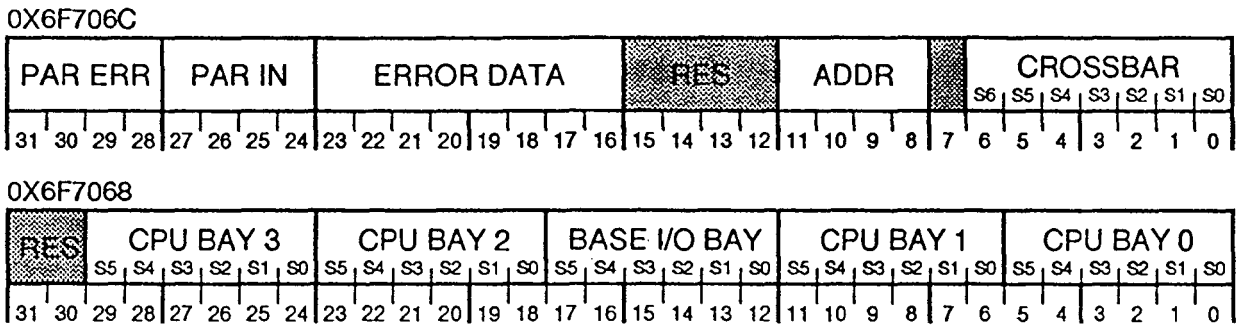
Figure 3-24: Scalar Halt and Halt Mask Register



3.4.2.2.11 Scan Control Enable Registers

The Scan Control Enable Registers are two 32 bit registers that contain the individual board Scan Control line mask bits. Individual bits in the register are assigned to a slot (in a specific bay) and are used to enable the master scan control lines to the selected system boards. If a board is not selected, it be place in the default mode, which is normal mode. The upper 24 bits of the crossbar scan control register contains the parity error log bits for the xcl write parity error. The organization of the Scan Control Enable Registers is shown in Figure 3-25.

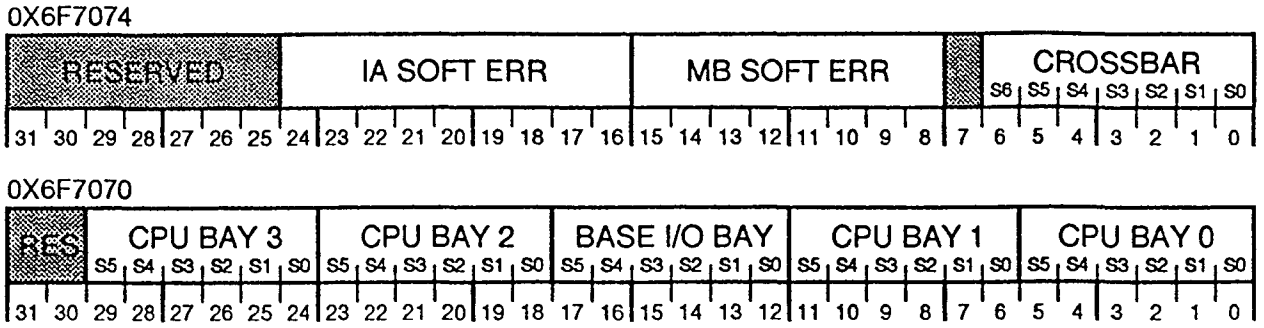
Figure 3-25: Scan Control Enable Registers



3.4.2.2.12 Hard Error Registers

The Hard Error Registers are two 32 bit read/write registers that contain registered versions of the hard error signals output from each of the boards in the system. These registers load the state of the hard error lines on the rising edge of each system clock (1X). If any of the hard error lines is set, a fatal error has been detected and the system must stop. The outputs of this register are logically ORed together (after masking) and input to the clock generator. When this signal becomes active, the clock generator will disable clocks to all of the boards in the system and disable the clock generator's burst counter. The active signal will also cause the register contents to be held until the SPU clears the register. The SPU can then read the contents of the Hard Error Registers to determine which board(s) had an error. The following figure illustrates the organization of the two registers.

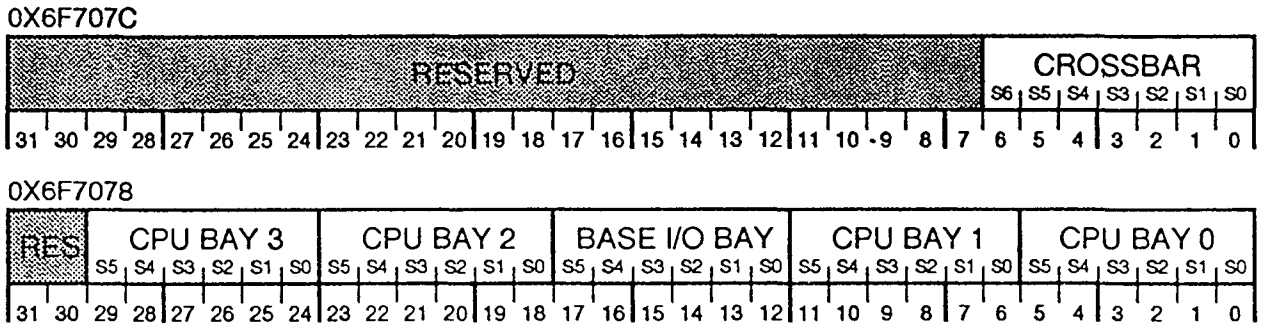
Figure 3-26: Hard Error Registers



3.4.2.2.13 Hard Error Mask Registers

The Hard Error Mask registers are two 32 bit read/write registers that are used to mask off the board hard error signals before error detection. One bit is assigned to each board in the system. Setting a board's bit in this register, zeroes out the hard error signal for that board, and prevents it from setting the hard error flag. The format of the registers is illustrated below.

Figure 3-27: Hard Error Mask Registers

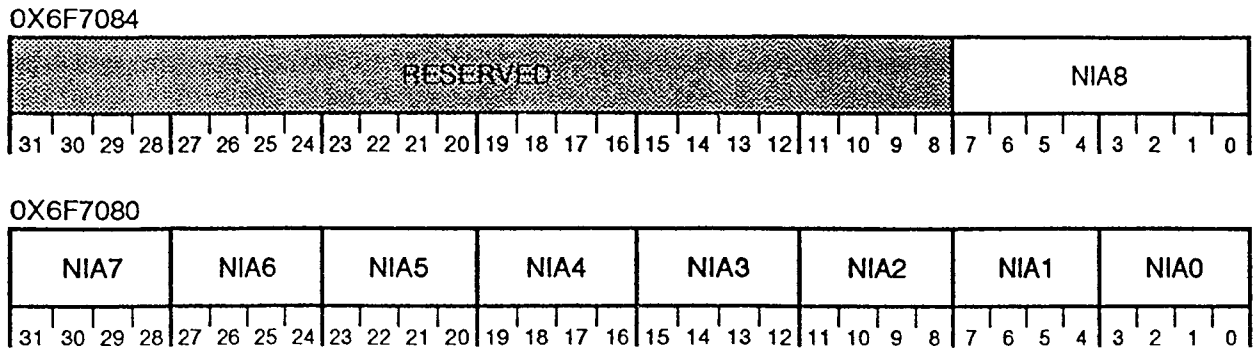


3.4.2.2.14 CCU Run Bit Enable Registers

The CCU Run Bit Enable Registers are two read/write registers that contain the 40 bits which are used to control the run bits for the CCUs in the system. The CCU run bits directly control the stoppable clocks to the CCUs, and are used to step and scan the boards. One bit is assigned to each CCU in the system, and eight CCUs are assigned to each bit in the system. The organization of the two registers is illustrated below. Note: the primary NIA in the system (port 8) is assigned eight CCUs while all other NIAs in the system are assigned four, to yield a maximum total of 40 CCUs in a system.

The run bit enable drives the select line for a 2:1 multiplexor that selects the data that will be sent to the CCU. When the enable is set, the master_rbe from the clock generator drives the CCU run bit. When the enable is reset, the mutiplexor selects a registered version of the multiplexor output, thereby holding the state of the run bit. With this individual control, any combination of CCUs can be in one of the following three states: Running, Diagnostic, Stopped.

Figure 3-28: CCU Run Bit Enable Registers

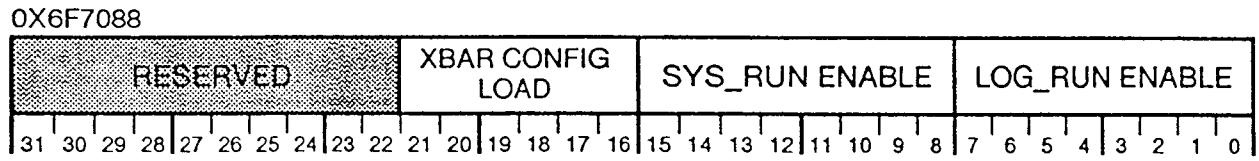


3.4.2.2.15 NMB and Crossbar RBE Enable Register

The NMB and Crossbar RBE Enable Register is a 32 bit read/write register that contains the run bit enables for the NMB Log_run and Sys_run signals, and Crossbar Config_Id signals.. The register is divided into 2 - eight bit fields and 1 - six bit field. The eight bit fields control the log_run and sys_run enables for the NMB. The six bit field controls the operation of the crossbar board's config_Id signals. The organization of the register is illustrated below.

Each run bit enable drives the select line for a 2:1 multiplexor that selects the data that will be driven to the memory or XBAR board. When the enable is set, the master_rbe from the clock generator drives the selected run bit. When the enable is reset, the mutiplexor selects a registered version of the multiplexor output, thereby holding the state of the run bit. Since the NMB signals are active low, the inverse of the multiplexor output is driven to the NMB board. With this individual control, any combination of memory and crossbar boards can be in one of the following three states: running, diagnostic, or stopped.

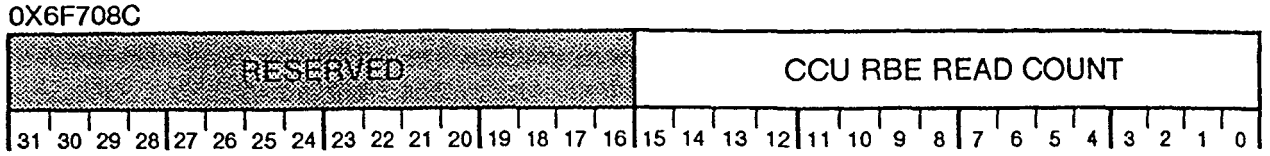
Figure 3-29: NMB Log and Sys RBE Enable Register



3.4.2.2.16 CCU Read RBE Enable Counter

The CCU Read RBE Enable Counter is a 32 bit read/write binary up counter that is used by the Scan Engine to gate off the CCU_RBE signal to the CCU under test during read cycles. During a CCU read operation, the counter will increment by one at the same time that the Scan Counter increments. When the counter reaches 0xFFFFFFFF, the terminal count becomes active, gating off the CCU_RBE signal and disabling the increment signal to the counter. During all other operations, the counter is disabled and is not referenced by the Scan Engine State machine. The organization of the register is illustrated below.

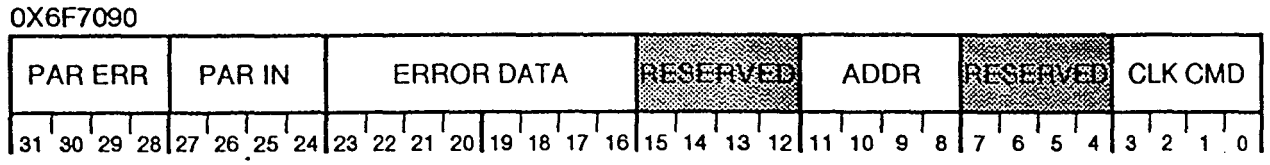
Figure 3-30: CCU Read RBE Count Enable



3.4.2.2.17 Clock Command Shadow Register

The Clock Command Shadow Register is an exact copy of the Clock Generator's Command Status register, coupled with the Clock Generator parity error register. The Command Status portion of shadow register is used by the Scan Logic to initiate scan operations, and to reference which operations are being executed. The scan logic decodes when a scan operation is being executed and checks to see if CCUs are involved. The parity error field of the Shadow register is referenced when a parity error is detected by the Clock Generator. Error State is held until the register is written. Note: Data is not written to the parity error log during the write. Writing to the register, causes the hold term to be removed, allowing new data to be written to the register on each 1x clock, until a new error is detected.

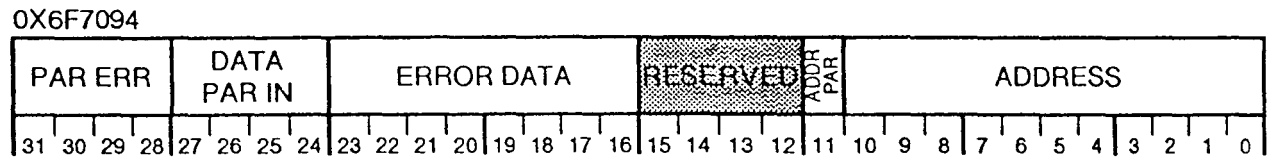
Figure 3-31: Clock Command Shadow Register



3.4.2.2.18 Scan Memory Parity Error Log Register

The Scan Parity Error Log is a 32 bit read/write register that contains error information when a parity error is detected in the Scan Memory. When the parity error flag is reset, the register is free running, capturing addr, data, parity, and error codes. When the flag becomes set, the contents of the register are held, preserving the state from when the error occurred. A write to this location resets the holding term, allowing the register to load data every clock until the parity error flag becomes set again.

Figure 3-32: Scan Memory Parity Error Log Register



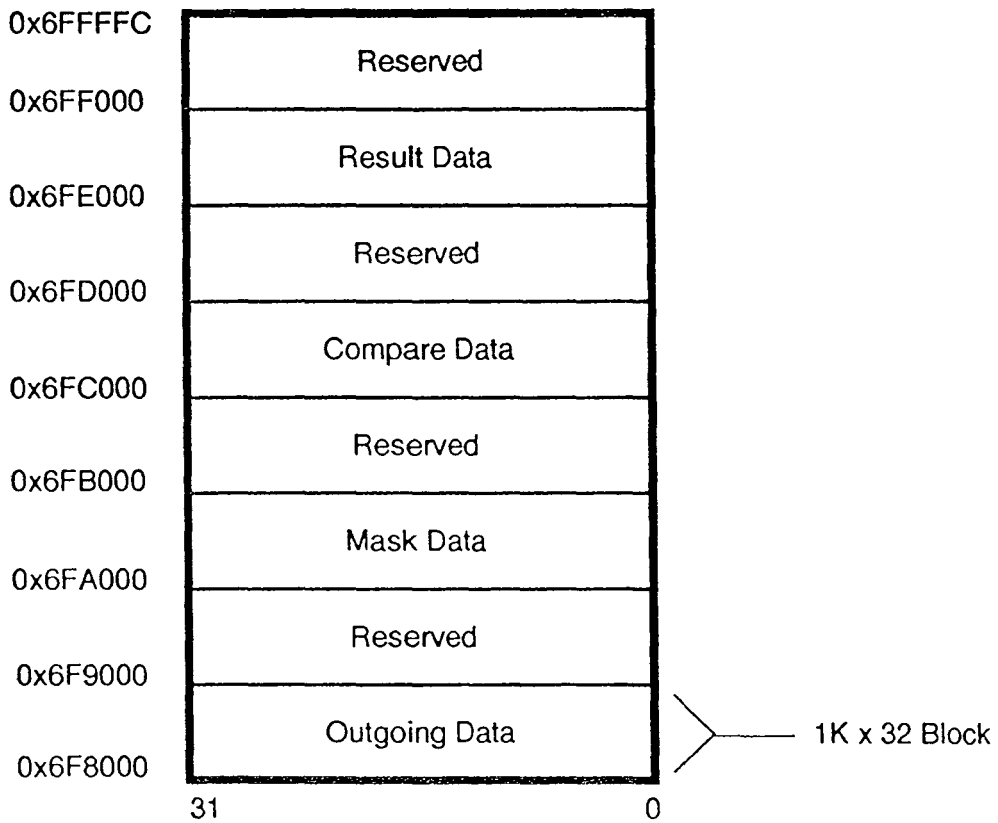
3.4.2.3 Scan Memory

The Scan Memory is a 4K by 36 bit memory array that contains the data used for all scan operations. Thirty two of the thirty six bits contain scan ring data and the remaining four bits contain parity information. The memory is divided into four 1K by 36 bit pages, each containing a unique data type. The four data types are Outgoing, Mask, Comparison, and Result. The Outgoing Data page contains data that will be loaded into the Output Scan Engine to be transmitted to the

boards in the system. The Mask data page contains data that after being inverted, is logically ANDed with the input buffer data and Compare data in order to mask off unwanted fields. The Comparison page contains data that is tested for equivilence against the input buffer data after the mask operation. The result page contains the data output from the Scan Engine Verification logic. Each page contains enough memory cells to support a scan ring of up to 32768 bits.

The SPU and the Scan Engine both have the ability to transfer data to and from the Scan Memory through the memory's 36 bit data interface. The master of the memory interface is determined by a control/data/address multiplexor contained within the of the Scan Engine. The select line of the multiplexor determines whether the SPU or Scan Engine has access to the memory and is controlled by the state of the Scan Engine's Busy bit. If the Scan Engine is in idle mode, the interface is controlled by the SPU. When the Scan Engine is executing a Scan operation, it has mastership of the bus and Scan Memory read or write commands from the SPU will be ignored.

Figure 3-33: Scan Memory Logical Partitioning



3.5 Hardware Description

3.5.1 Overview

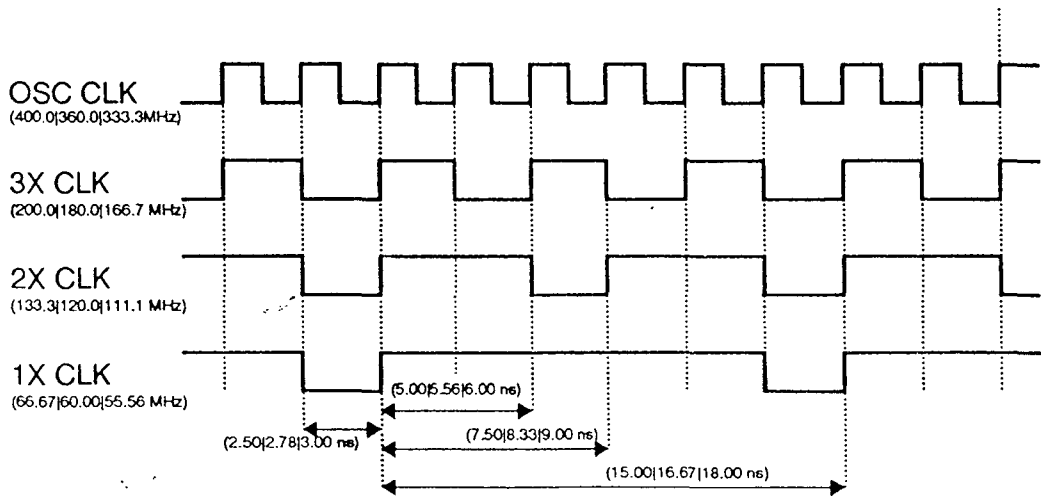
The Neptune Scan Engine and Clock Generation Subsystem (SECG) is responsible for the generation and distribution of the system clocks and the control of scan operations in the system. The SECG subsystem performs these functions based on commands received from the SPU through the Workstation Interface card (SWIP). The SECG is comprised of logic and mechanical hardware (connectors) contained on three different boards in the system: NCU, IO Backplane, and the Crossbar Control Board (XCL). The SECG subsystem is implemented using a single 10K gate array, eight self timed rams, descrete eclips logic, and 40 sets of connectors and differential coax cable for clock distribution.

The SECG subsystem is divided into four logical blocks: Clock Generator, Scan Engine, Scan Memory, and Clock and Scan Distribution. The Clock Generator contains the fastest logic in the Neptune system and is responsible for the generation and control of the system board clocks. The Scan Engine is a state machine responsible for transferring data between the Scan Memory and the system board (s) under test. The Scan Memory is a 4K x 36 bit memory array which stores the ring data transferred to and from the boards by the Scan Engine. The Clock and Scan Distribution section of logic is responsible for the control and distribution of the clocks and Scan Enable to the many descretes that make up the design.

3.5.2 Clock Generator

The Clock Generator is responsible for the generation and control of clocks for the Neptune computer system. It is implemented using a 10K gate array designed to operate at a design margin speed of 500 Mhz (360 Mhz nominal). From the master oscillator clock (6X), the generator produces three controllable clock rates, 3x, 2x and 1x (System Clock). Each of these may be transmitted to any slot in the system and may be operated in one of four modes: Free Run, Burst Run, Single/Micro Step, and Disabled. The clock rate and mode for a given slot are determined by commands written to the Clock Generator by the SPU. The Clock Generator has the ability to assign these rates and modes on a slot by slot basis.

Figure 3-34: Neptune System Clocks



Commands are received from the SPU through the NWI Interface. The transfers are synchronous and occur at the 1x clock rate. Nine control registers determine which function is being executed, and which boards are involved. Once a command is executed (writing to the control register), the clock control engine determines when to disable, switch and enable the selected clocks based on the state of the free running 1x, 2x, and 3x clock generators. Completion of a command execution is detected from the SPU by polling the Command/Status register.

For a detailed description of the operation of the NCLK Clock Generator, please refer to the NCLK functional description, Convex part Number 182-000141-000 . This document contains all the information concerning the functional operation of the Clock Generator, and should be consulted when detailed operational information is required.

3.5.2.1 C3 Clock Distribution

The C3 Clock Distribution scheme is a matched set of clock delay paths that removes the need for clock tuning on the individual system boards. The system is designed so that all clock paths from the Clock Generator Array die to each destination register's clock input (on each board in the system) are matched in length.By matching all clock paths and minimizing skew within these paths, one eliminates the need for "clock tuning" on boards in order to align clock edges in the system.

All board clocks in the system are transmitted as differential ecl 100K level signals. The board clocks are sourced from the NCLK array and routed out to the board edge. The clocks are transmitted through the controlled impedance Augat connector and driven onto the IO Backplane (IOBP). Contained on the IOBP are six connectors that attach to the the triax cable bundles which distribute the clocks to the different bays in the system. The connector on the destination backplane, contains the differential clocks for each slot in the backplane.

The differential clock is received by the destination board through its Augat connector. Once the clock is received by the board, it is distributed to the different register elements on the board. All clock connections on a board are point to point connections. Three levels of 100e111 control skew clock buffers are used to obtain up to 729 clock connections. If that many clocks are not required by a board, the designer may opt to clock devices (such as gate arrays) off of first or second level clock buffers. Clocking off of these earlier stage buffers, will further reduce the clock skew to a board by removing the skew associated with the removed buffer (s). An illustration of the C3 clock distribution scheme is shown in figure 3-26.

The two types of backplane routes (outgoing routes on IOBP, destination routes) equalize the segments that are made up by the Augat connector flex film, the routed wire and the escape etch for the clock connectors. The column of the Augat connector that a signal resides on determines the length of conductor that must be traversed between the connector pins. Table 18 shows the relationship between the connector column and the conductor length. Escape etch for the triax clock cable connector will also vary depending on the pin number that it is attached to. On the IOBP, all clock signals sourced from the NCU and received by the triax connectors are equalized at 7000 mils. On all destination backplanes, all clock signals running between the triax connectors and the destination boards are equalized at 9000 mils.

Table 3-18: Augat Column to Conductor Length Mapping

Board Type	Pin #	Conductor Length
G.A. Right	1xx	0.525 in
G.A. Right	2xx	0.588 in
G.A. Right	3xx	0.638 in
G.A. Right	4xx	0.675 in
G.A. Left	1xx	0.675 in
G.A. Left	2xx	0.638 in
G.A. Left	3xx	0.588 in
G.A. Left	4xx	0.525 in

On Destination boards, sum of the clock buffer delays, routed wire and surface etch are normally equalized to a delay of 10000 pS. However, 10000 pS is really a guideline number to be used, and the deciding factor is really whether board and system level timing will be satisfied with a certain group of clock delays. In the case of boards which have gate arrays, devices will be clocked from second or even first level clock buffers. When devices are clocked from these buffers, the route wire and gate array clock buffering must make up for the missing 100e111 stages. The clocktune program takes this into consideration, modifying the scheduler length file (schlen.dat) so that the routes will be routed to the calculated fixed length.

3.5.2.2 C3 Clock Tree Delay and the SPU Interface

There is a fixed clock delay from the die within the NCLK array to each clock input on registered devices in the system. Although there will be some slight differences in delay due to process variation and manufacturing tolerance, the delay is treated as a fixed delay of 21.655 nS. The delay is set up so that the third level NCU 1X clocks and the NCLK internal 1X clocks will be approximately 180 degrees out of phase from one another over the margin range. This is done so that there will be a large amount of margin between data transfer clocks as the oscillator selection

field is switched from fast to slow.. Since the clock path delay is fixed, as the oscillators are switched from slow to fast margin, the transmitting and receiving register clock edges will move away and closer to on another. One frequency will come close to violating the setup and hold requirements for the SPU to NCLK interface timing, and the other will come close to violating the NCLK to SPU interface timing.

With the total clock delay set at 21.655 nS, the clock interface will operate reliably from 14 to 18 nS. Testing in the lab has shown that the interface will operate up to a 20nS 1X clock rate, but will fail reliably at 22 nS. The timing for the interface has been verified (using TLC) with skew up to a 19.5 nS 1x clock period. As the 1x clock period is increased past the 20 nS mark, the setup and hold requirements for the NCLK to Scan Engine timing start becoming violated, with the DSCAN_GOD signal failing first. The mechanism of failure will be unusual scan related faults as the Scan Engine loses synchronicity with the Clock Generator.

Once the setup and hold for the Scan Engine is violated, the interface enters what is referred to as the "dead zone" for data transfer. The "dead zone" is a range of 1x clock periods that the interface is not guaranteed to work within. For the 21.655 nS clock delay, the "dead zone" of operation is between 19.5 and 25.5 nS for a 1x clock period. Within this range, setup and hold for the two interfaces is violated. Once this range of clock periods is passed, the clock period is now long enough that the clock delay path is less than a 1x clock period. This is an important fact because previously, data would advance through a pipeline stage before a clock could be fired off from the array and received at the destination register. Once the "dead zone" is passed, a clock can be fired off and received before the data passes through a pipeline stage. This means that scan data pipelining must be adjusted in order for the scan logic to work. A better description of this phenomenon is that before the dead zone there were always two clock edges contained in the the clock delay path, and after the zone, there is only one.

The R.NO_CLK_IN_CBL signal is set in order to indicate to the Scan Engine that the 1x clock period is now outside the "dead zone" area, and that there is no longer two clock edges contained in the cable at any given time. When the signal becomes set, the DSCAN_GO that kicks off the Scan Engine state machines and controllers, is issued one clock cycle earlier than when the R.NO_CLK_IN_CBL signal is reset. This allows the Scan Engine to be properly synchronized with the issuing of clocks. If the R.NO_CLK_IN_CBL signal is not set properly, then scans will be misaligned by one bit in one direction or the other (depending on how the bit is incorrectly set).

3.5.2.2.1 Calculating the Proper Clock Cable Length

For each range of margin frequencies that is selected (slow, nominal, fast), there is an optimal cable length that can be selected so that an equivilent amount of margin can be achieved at the slow and fast rates. At first glance, one might believe that selecting a total clock delay that is 1.5 times the average of the slow and fast clock periods (which would put the NCU and NCLK 1x clocks 180 degrees out of phase) would yield the optimal delay. However, differences in net delays, setup and hold requirements, and dealing with a range of frequencies invalidate this assumption. By taking into consideration these differences, one arrives at a more optimal length, which will yield equivilent data transfer margins at the upper and lower oscillator rates. Note: One uses the period of the 1x clock for calculations since that is the rate used to transfer data between the Clock Generator and the interface logic.

Because the clock cable length needs to work for a range of frequencies instead of a single frequency, the calculation for the average delay needed is more involved than multiplying the nominal frequency by 1.5. The average delay needed is the average of the longest single cycle and the shortest double cycle. One must also add in the differences in setup, hold and oscillator to clock edge in order to get an accurate number. Since the setup, hold and output clock times for the NCLK are all referenced from an oscillator edge occuring at time = 0, one must reference all delay calculations from the oscillator edge. The calculations for the maximum single cycle and the minimum double cycle are illustrated below.

Single Clock Calculation:

max. single 1x cycle + NCLK hold + min. osc -> 1x clock

(for 1x Clock = 18 nS)

18 nS	+ 1.70 nS	- 2.96 nS	= 18.86 nS

Double Clock Calculation:

min. double 1x cycle + NCLK setup + max. osc -> 1x clock

(for 1x Clock = 15.5 nS)

31nS	+ 0.14nS	- 3.54 nS	= 27.60 nS

Once these numbers are calculated, the fixed path delays and variances in the clock tree delay must be subtracted in order to get minimum and maximum clock cable lengths that will allow these frequencies to work. Since the 100e111 value used by clocktune is just an average, there will be a minimum and maximum board clock tree delay. These are used to get the final minimum and maximum clock cable values. The calculations for these cable lengths are illustrated below.

Min. Cable Calculation:

max single cycle - fixed wire - board min delay

18.86 nS	- 4.157 nS	- 9.775 nS	= 6.808 nS

Max. Cable Calculation:

min double cycle - fixed wire - board max delay

27.60 nS	- 4.157 nS	- 10.375 nS	= 13.068 nS

By averaging the minimum and maximum cable delays, one arrives at the optimal length for the clock cable, assuming that the worst cast interface signals to the NCLK are of equal length. If this is not the case, then the difference between the two (SE to NCLK, and NCLK to SE) needs to be added to the average cable delay to center the high and low margins. The delay on the NCLK interface is made up of the clock to Q of the driving register, route length, and the setup to the

receiving register. The final calculations for the cable length are illustrated below. The final number indicates the delay required by the cable in order for the system to have equivilent margin at high and low oscillator frequency. By ordering the clock cable by delay instead of length (TDR will indicate the delay) one can remove the problem of varying propagation velocity from cable purchase.

Interface Difference Calculation:

(DSCANGOD wire delay +clk 2 Q + E142 setup) -
(R.CG_WR_LO_GATE wire delay + E142 clk 2 Q + NCLK setup) = Interface Offset

(1.8 nS + 2.0 nS + 0.3 nS) - (0.7 nS + 1.0 nS + 1.9 nS) = 0.50 nS

Final Cable Length Calculation:

((13.068 nS + 6.808) / 2) + 0.50 = 10.438 nS = Optimal Cable Delay for 15.5 ns to 18.0 nS

3.5.2.3 Run After Refresh Logic

The Run After Refresh logic (RAR) is responsible for making sure that clocks are always restarted at the same point in time with respect to the refresh pulse that is issued to the memory boards. The first reason for this is that there is a minimum number of clocks required (by the NMB) between the refresh pulse and when clocks are issued. If this were violated, requests could be issued to busy banks of memory which could result in an errror. Secondly, having a fixed relationship between the refresh pulse and the clocks starting, allows problems to be repeatable because all restarts will always occur at the same point in time with respect to memory refreshes and I/O system transfers. Therefore, a test can be started at the same point in time over and over again which will make non-intermittent bugs repeatable.

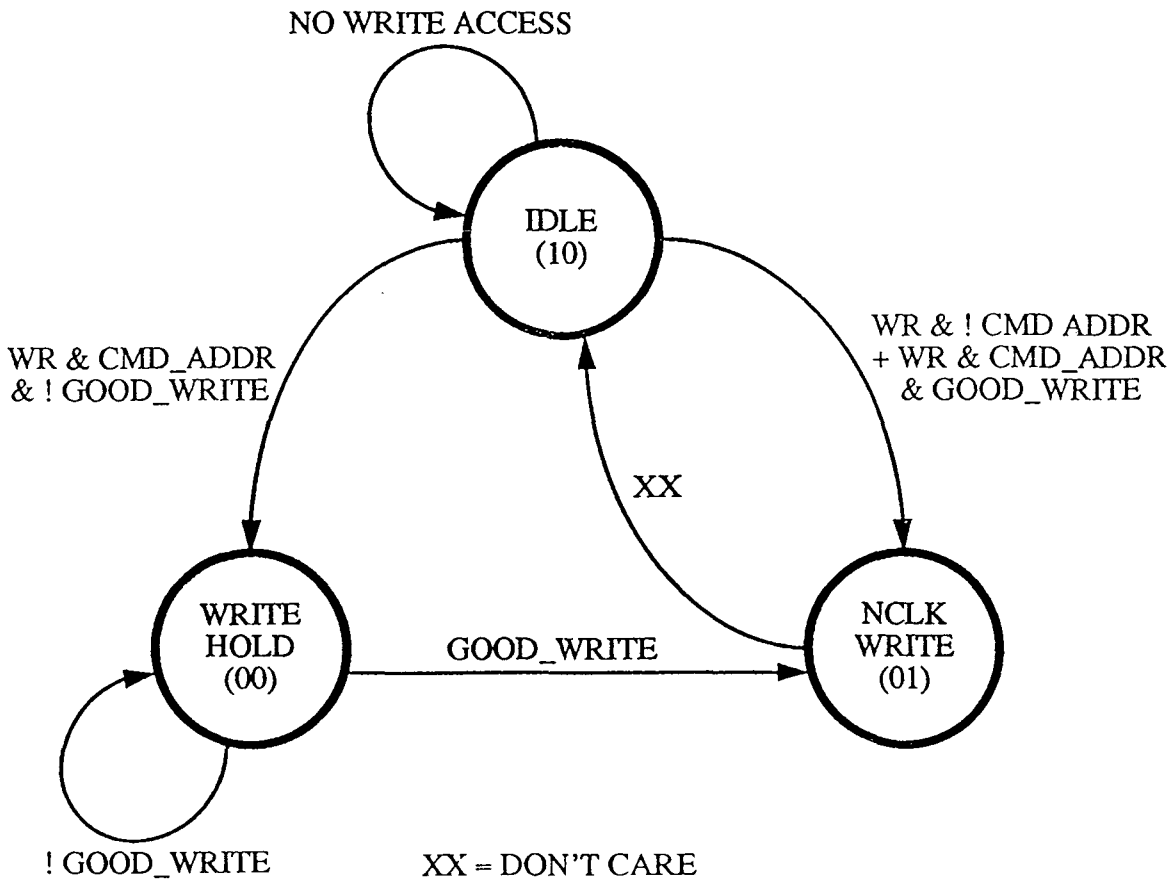
This is accomplished is by coupling the occurance of memory refresh to a specific event in the I/O system clock generation scheme. Since the phase monitor synchronization signal (master_s0) is available on the NCU and is used to synchronize the NIA, NCLK, and Scan Engine, it is the logical selection as the signal that will be used to synchronize the memory refresh to State 0 of the PBUS clock generator. Since the Clock Generator will wait for the stopped and free running phase monitors to align before restarting clocks, now a correlation is established between the memory refresh, I/O system clocking and when clocks restart. A detailed description of how the master_s0 is used to accomplish this can be found in the Memory Refresh logic section of the NCU hardware definition.

The RAR logic is responsible for synchronizing the write access to the NCLK to the memory refresh, which closes the feedback loop and guarantees that clocks will be restarted a specific number of clocks after the refresh is issued to the memory boards. Of the 960 1X clocks that make up a memory refresh period, only one of those clock periods is allowed as the write access period for the NCLK. The RAR logic decodes the state of the memory refresh pulse shaping logic to determine when this cycle occurs. The time period is specifically called out as when the r.refresh_tc3 signal is set and the r.refresh_tc3a signal is reset. This occurs only once each refresh period and guarantees that the clocks will be issued no earlier than 22 clocks after the refresh pulse is issued.

Since the NWI and its interface has no knowledge of this timing requirement, the RAR logic is responsible for synchronizing the NWI access to the valid access period and holding off accesses until the cycle is completed. When a clock generator write is attempted by the NWI, the write strobe is captured while the refresh logic is decoded for the proper state. If the proper state is detected, the pulse will be issued to the NCLK and the access will be completed.

If the proper state is not detected for a write access to occur, the access is held off until the the proper state is detected. When this occurs, the R.CG_WR_LO_HOLD signal is immediately asserted. This stores the strobe and holds off the NWI Interface from acknowledging to the spu. The hold signal is input into pal 181-005449, and is used to hold off the WRITE_DONE signal which is an enable for the SPU acknowledge. The state machine will remain in this state until the proper cycle in the refresh is detected. Once this cycle is detected, the hold signal is reset and the write strobe to the NCLK is issued. This releases the acknowledge to the spu, allowing the next transfer to occur. The state diagram for this process is illustrated in the following figure.

Figure 3-36: Run After Refresh Logic State Diagram.

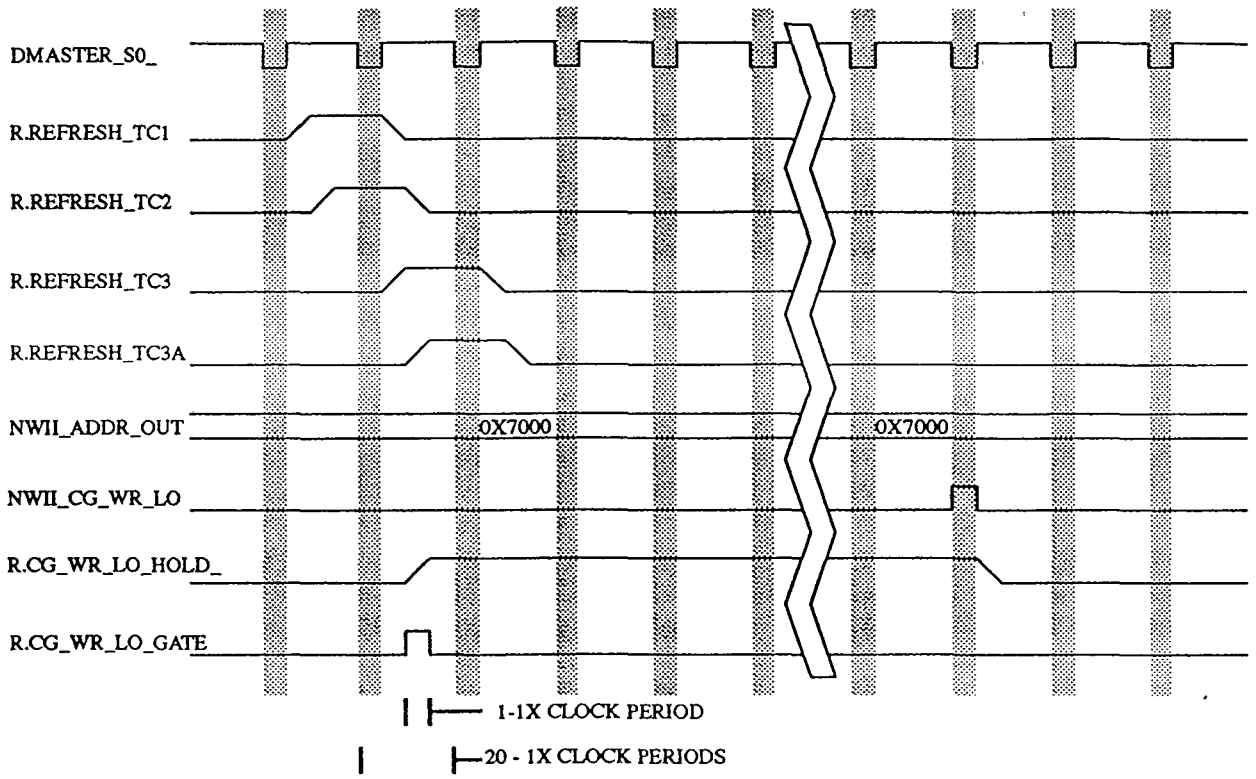


GOOD_WRITE = R.REFRESH_TC3 & !R.REFRESH_TC3A

STATE = R.CG_WR_LO_HOLD_ , R.CG_WR_LO_GATE

Since a write access can be held off for as long as a refresh period (14.40 uS @ mod 18) and performance of the SECG subsystem is a concern, decoding of the NCLK address is included in the logic to minimize the RAR logic's affect on NCLK accesses. Since Command exection is the operation which needs to be synchronized with the refresh, the RAR logic decodes the NCLK address and specifically looks for write accesses being made to the lower sixteen bits of the Command/Status register. If a write access is made to any other register, the access is allowed immediately regardless of the state of the refresh logic. However, if the write access is made to the lower bits of the CS register, the RAR logic will operate as previously described .

Figure 3-37: Run After Refresh Logic Timing



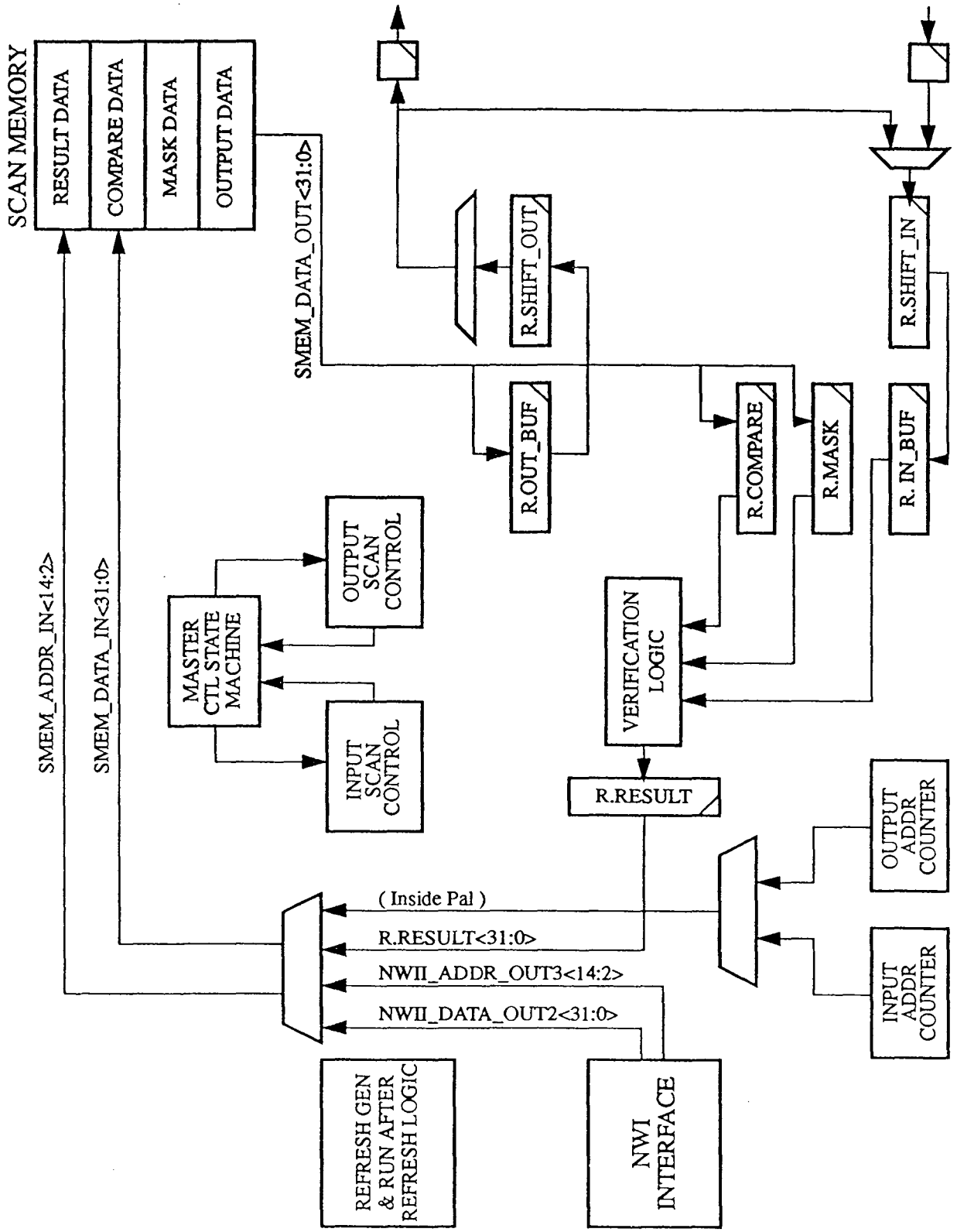
3.5.3 Scan Engine

The Scan Engine controls the transfer of scan data between the Scan Memory and the boards in the system. The Scan Engine has three main functions: Control of the Scan Control Interface, Bi-directional parallel to serial conversion of the scan data, and automatic verification of the received scan data. The Scan Engine receives commands from the SPU via the NWI Interface. It is connected to each board in the system through a six line interface, five unique lines per slot for Scan Control and data input and a common, buffered data output line. Note: The terms input and output are from the BUTs perspective. Write data is transmitted to the XC_XX.SCAN_IN signal of the BUT and data is read from the XX_XC.SCAN_OUT signal. The Scan Engine contains a third interface which interfaces with the Scan Memory. Through this interface, the Scan Engine has the ability to autonomously transfer data between the memory and its own data transfer registers.

The Scan Engine is a state machine based design implemented with Eclips logic and ECL 100K pals. The logic comprising the Scan Engine is divided into six functional blocks: the Master Controller, Memory Interface, Output Engine, Input Engine, Dynamic Ring/ Refresh Logic, and Data Interface logic. The Master Controller is responsible for generating the signals that control the data flow between the Scan Memory and the Input and Output engines. The Memory Interface is responsible for connecting the NWI Interface and the Scan Engine to the Scan Memory. The Output Engine converts words of scan data loaded by the Master Controller into a serial data stream that is shifted out to selected boards. The Input Engine is responsible for the serial to parallel conversion of input scan data, its masking and comparison with expected values and its

storage in the Scan Memory. The Dynamic Ring/ Refresh logic generates the control signals that are used to control Dynamic and CCU ring scans and memory refresh. Additionally,. this block contains the run after refresh logic that synchronizes clock restarts with the memory refresh pulse. The Data Interface logic contains all logic associated with transferring data between the Scan Engine, the NWII and the XCL. The following figure illustrates the data path of the Scan Engine.

Figure 3-38: Scan Engine Dataflow Diagram



3.5.3.1 Master Controller and Support Logic

The Master Controller is responsible for controlling the the data flow between the Input and Output engines and the Scan Memory Interface. The Master Controller changes the state of the control signals for these modules based on inputs received from the control registers and the status lines of the controlled modules. The Command/Status Register informs the Master Controller which of the two engines (possibly both) will be involved in the current scan operation and whether the verification will be involved in the operation. The Master Controller receives a single memory request from the Input and Output Scan Engines. The request from the Output Engine is implicitly a read request. However, since the Input Engine will perform reads and write to the Scan Memory, a second status line is required by the Master Controller to indicate what type of access will occur. The Master Controller gives requests from the Output Engine priority over requests from the Input Engine

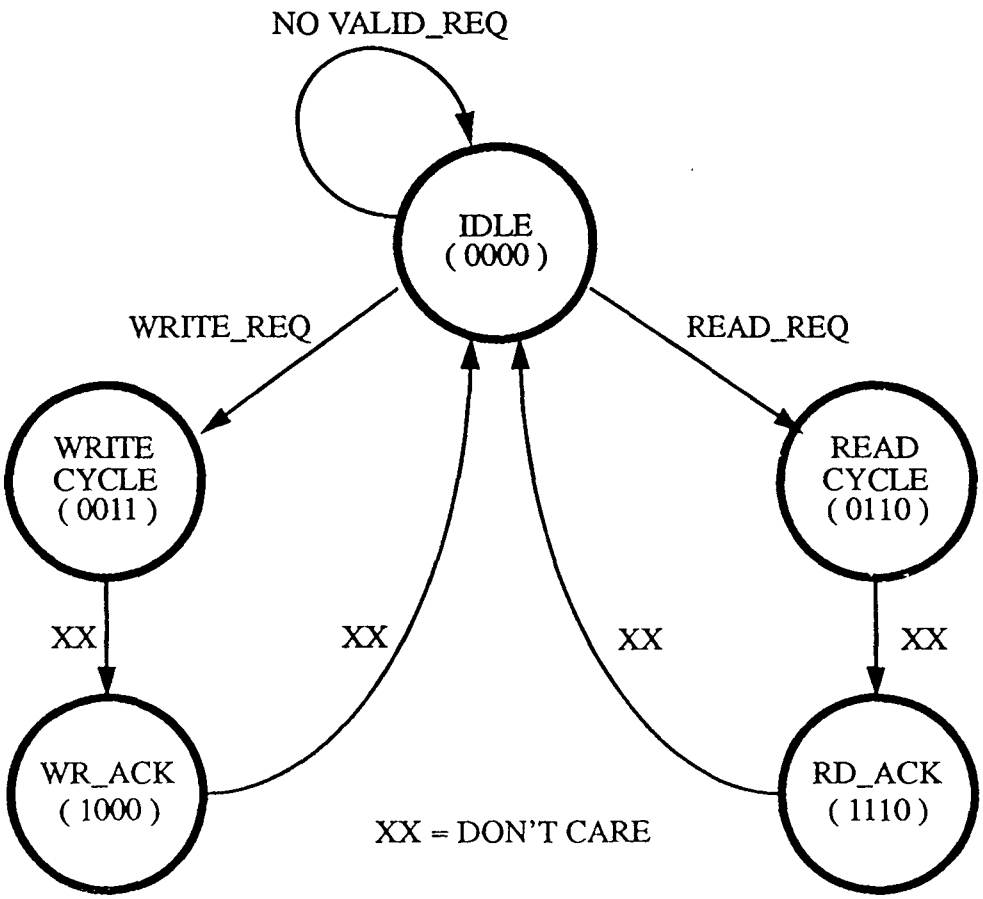
The Master Controller is divided into four logical blocks. These four blocks share status, and work together in order to transfer the data between the Scan Memory and the Input and Output Engines. The four blocks are the Memory Control (MC), Master Arbiter (MA), Input Scan Engine (ISE), and Output Scan Engine (OSE) State machines.

The Memory Control state machine generates the control signals for the Scan Memory and returns an acknowledge when the transfer is completed. The Master Arbiter state machine receives memory access requests from the Input and Output engines and after granting the request, kicks off the Memory Control state machine. The Input Scan Control State Machine generates the requests for the loading of the Mask and Compare registers and the storing of the Result register. The Output Scan Control State Machine generates the requests for the loading of the Output Buffer

3.5.3.1.1 Master Controller: Memory Control State Machine

The Memory Control state machine (MC) generates the control signals for the Scan Memory and returns an acknowledge to the Arbiter state machine when the transfer is complete. The MC state machine remains in the idle state until valid request is sent from the arbiter. Once the request is recognized, the state machine will determine whether a read or write access is required based on the state of the inp_master signal and a status bit from the ISE_SM<4>. In both access cases, the memory cycle will take three 1X clock periods. During the first cycle, the address and data (if necessary) are driven to the Scan Memory. During the second cycle, the control signals are asserted. During the third cycle, the control signals are reset and the transfer acknowledge is set. After the third cycle, the machine returns to the idle state and waits for another valid request.

Figure 3-39: Master Controller: Memory Control State Machine



State := MEM_ACK, MASTER_ENG<2>, MEM_CS*, MEM_WR*

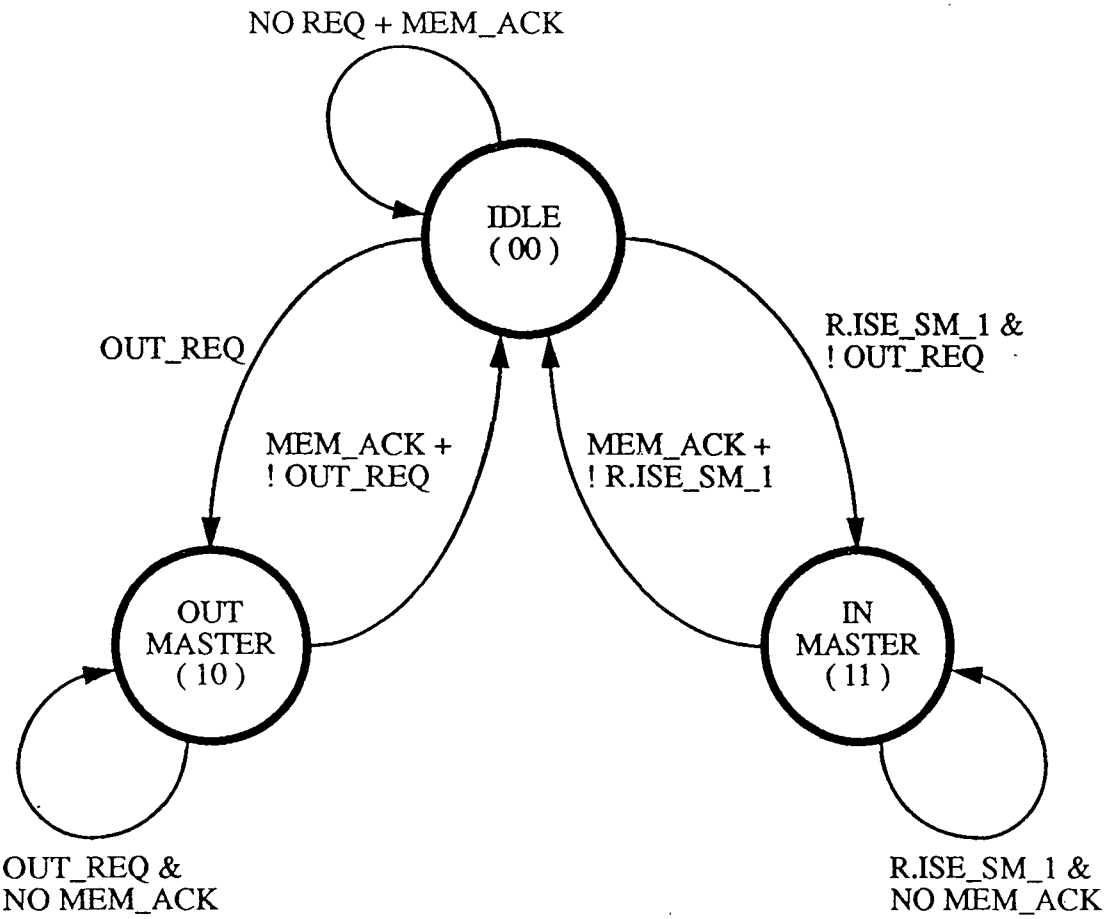
WRITE_REQ = VALID_REQ * INP_MASTER * ISE_SM<4>

READ_REQ = VALID_REQ * INP_MASTER * ! ISE_SM<4>
+ VALID_REQ * ! INP_MASTER

3.5.3.1.2 Master Controller: Master Arbiter State Machine

The MA state machine receives memory access requests from the Input and Output engines, and after granting the request, kicks off the MC state machine. If the OUT_REQ signal is set and the MA state machine is in the idle state, the state machine will transfer to the out_master state. In this state, the valid_req bit is set and the inp_master is reset. Moving to this state will initiate a read operation in the MC state machine. The MA state machine will remain in this state until a memory acknowledge is received from the MC state machine or the output scan engine withdraws its request.

Figure 3-40: Master Controller: Master Arbiter State Machine



If an input request, R.ISE_SM<1>, is set while the output request is reset (output scan engine has priority), the state machine will transition to the in_master state. Moving to the in_master state will kick off a memory access, but the memory access type will be determined by the state of the most significant bit of the ISE state machine. The logic for this state machine is contained in pal 181_005488.

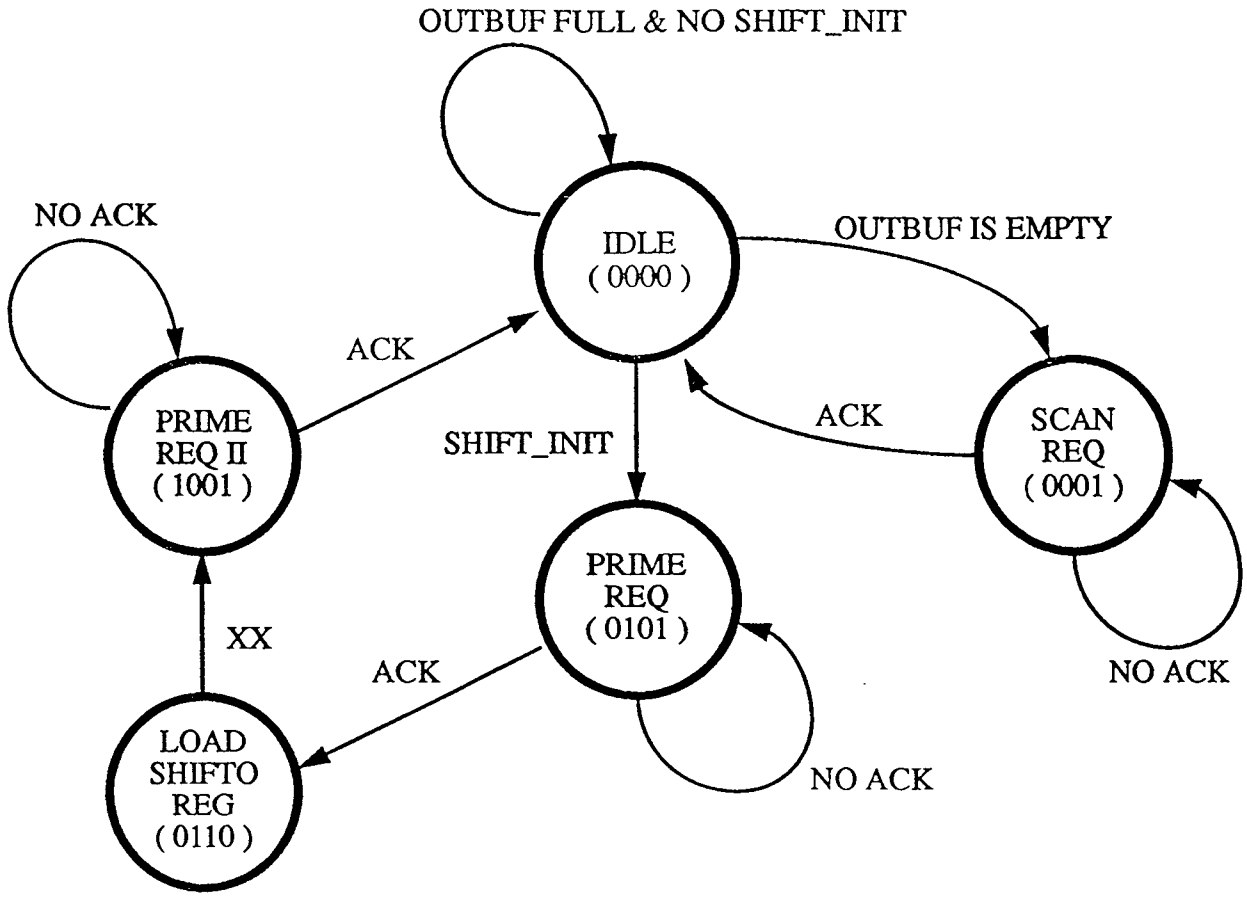
3.5.3.1.3 Master Controller: Output Scan Engine State Machine

The OSE state machine generates the Scan Memory read requests for the loading of the Output Buffer. The OSE state machine is activated when one of two conditions becomes active. These conditions are the r.shifto_init signal becoming set or the resetting of the r.out_buf_wrd_cnt signal. The r.shifto_init is set after the SPU sets the scan write bit in the Command Status Register. This bit indicates to the OSE machine that a scan write operation is about to commence, and therefore, the output buffer and output shift register need to be loaded with data in preparation for the start of the operation. The resetting of the R.OUT_BUF_WRD_CNT signal is an indication that the contents of the output buffer have been loaded into the output shift register, and that the next word needs to be fetched from the output buffer in the Scan Memory.

After the r.shifto_init signal is detected, the OSE will transition to the prime_req state. In this state, the OSE issues a read request to the Master Arbiter. The OSE state machine will remain in this state until an acknowledge is received from the MC state machine. When the ack is received, the state machine will transition through the load_shifto_reg state. In this state, the data is strobed into the Output Buffer and on the following clock into the Output Shift Register. The machine then transitions into the prime_req_II state. In this state, the OSE will issue a second read request to the Scan Memory. The state machine will wait in this state until an acknowledge is received from the MC state machine. When the acknowledge is received, the Output Buffer will be loaded with the data and the state machine will return to the idle state. The state machine will remain in this state until the scan operation is executed in the Clock Generator and a reload of the Output Shift Register is required.

When the r.out_buf_wrd_cnt signal is reset, it indicates that the Output Buffer is empty and needs data loaded from the Scan memory. When this condition is detected and r.shifto_init is reset, the state machine will transition to the scan_req state. The state machine issues a read request to the Scan Memory and will remain in this state until a memory acknowledge is received. When the acknowledge is received, the state machine returns to the idle state.The Out Scan Engine state diagram is illustrated on the following page.

Figure 3-41: Master Controller: Output Scan Engine State Diagram



XX = DON'T CARE
ACK = MEM_ACK & ! INP_MASTER

STATE = OSE_SM<2>, OSE_SM<1>, OSE_SM<0>, OUT_REQ

3.5.3.1.4 Master Controller: Input Scan Engine State Machine

The ISE state machine generates the requests for the loading of the Mask and Compare registers and the storing of the Result register. The state machine will remain in the idle state until the busy and scan_read bits are set in the Com_Stat register. Depending on the state of the verify bit in the Com_Stat register, the state machine will follow one of two paths through the state machine for the duration of the scan operation. If the verify bit is set, the state machine will follow a path that will load the mask and compare registers and then write the result data to the memory once it is received. If the verify bit is reset, the state machine will follow the path that waits for the result data to arrive and then writes it to memory. The verify path through the state machine is a superset of the read path. The logic for this state machine is contained in pal p181_005487.

If the busy, scan_read, and verify bits are set in the Com_Stat register while the ISE state machine is in the idle state, the state machine will cycle through the verify path of the state diagram. When this condition is detected, the machine will transition to the mask_req state. In this state, the state machine issues a read request to the mask data page in the Scan Memory. The state machine will remain in this state until an acknowledge is received from the MC state machine. When the acknowledge is received, the machine transitions to the mask received state.

In the mask received state, the mask register is loaded with the data from memory. The machine, then immediately transitions to the cmpdat_req state where the state machine issues a read request to the compare data page in memory, and will remain in this state until an acknowlege is received from the MC state machine. When the acknowledge is received, the compare data register is loaded and state machine transitions to the scan_din_wait state.

The state machine will remain in this state until the r.in_buf_wrd_cnt signal is set. This signal indicates that the R.IN_BUF register is full and needs to be written to memory. Once the signal is set, the machine will transition to the result_req state. While in this state, the R.IN_BUF data will pass through the mask and verification logic and be stored in the result register. Additionally, the write request for the data will be sent to the arbiter. When the acknowledge is received, the state machine transitions to the memwr_done state.

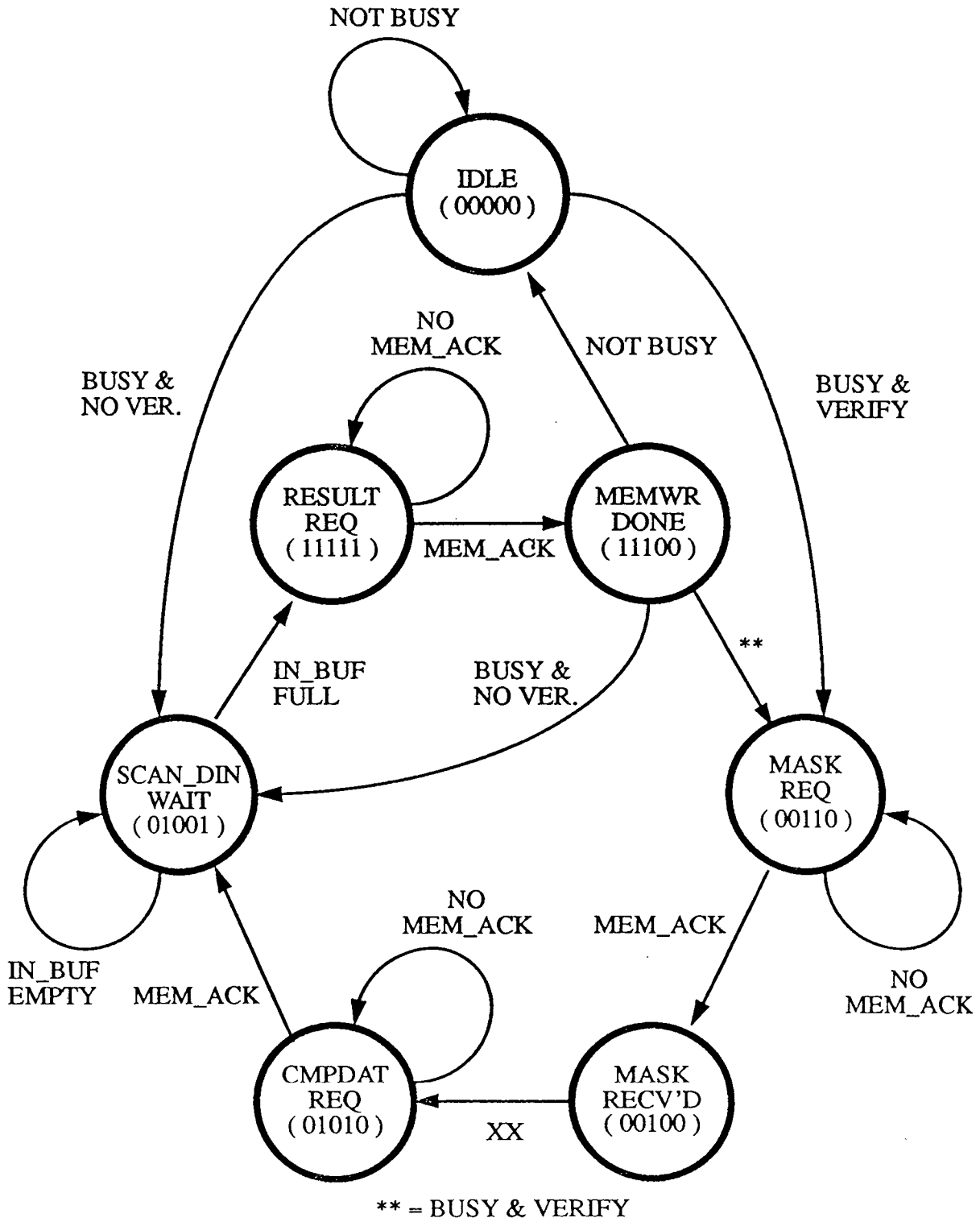
The memwr_done state is very similar to the idle state, except that in the memwr_done state, the input address counter is decremented by one. If there are more words to be scanned (R.CMD_STAT<16> is set) , then the state machine will transition to the mask_req state and the process will repeat. Otherwise, if the scan operation is complete, the state machine will return to the idle state.

If the busy and scan read bits are set while the verify bit is reset, the state machine will cycle through the read path of the state diagram. Once this condition is detected in the idle state, the state machine will transition to the scan_din_wait state. The state machine will remain in this state until the r.in_buf_wrd_cnt signal is set. This signal indicates that the In_Buf register is full and needs to be written to memory before the next word is fully shifted into the Input Shift Register. Once the signal is set, the state machine will transition to the result_req state.

In the result_req state the In_Buf data is stored in the Result Register and a write request is issued to the MA state machine. The state machine will remain in this state until a memory acknowledge is received from the MC state machine. When the acknowledge is received, the state machine will then transition through the memwr_done state.

In the memwr_done state, the write request is reset and the input address counter is decremented by one. If the scan operation is complete, the state machine will then return to the idle state until the next operation is initiated. If the scan operation is not complete, the state machine will return to the scan_din_wait state and repeat the above process until the scan operation is complete.The state diagram for the Input Scan Engine state machine can be found on the following page.

Figure 3-42: Master Controller: Input Scan Engine State Machine



STATE = R.ISE_SM<4>, R.ISE_SM<3>, R.ISE_SM<2>, R.ISE_SM<1>, R.ISE_SM<0>

3.5.3.1.5 Command/Status Register

The Scan Engine Command Status Register is a 20 bit register that contains the status used by the Scan Engine to determine what type of scan operation to perform. All of the bits in the register may be written and read by the SPU. The lower sixteen bits of the register will only change state when a SPU write is performed to the lower half of the word. The upper four bits of the register are loaded on every clock cycle with the output of a multiplexor pal.

A write cycle from the SPU to the register will initiate a Scan operation. When the SPU write enable becomes active, the Scan Counter is loaded with the contents of the Scan Counter Register on the next clock edge. If the Scan Counter is loaded with a non-zero value, the R.SCAN_DUN signal will be reset, which causes the busy bit, R.CMD_STAT<16>, to be set on the next clock cycle. Setting the busy bit in the Command Status Register enables the state machines and controllers of the Scan Engine. If a Scan Write operation is being executed (R.CMD_STAT<1> is set), the R.SHIFTO_INIT signal becomes set as the register is written. This is the signal that enables the prescan loading of the Output Buffer and Output Shift Registers.

The upper four bits of the register, bits 19-16, are loaded each clock cycle with the output of pal 181-005474. The upper two bits, 18 and 19, implement load and hold functionality so that they appear to operate the same as the lower sixteen bits of the register. When R.CMD_WR_HI* is set, bits 18 and 19 will be loaded with the corresponding bits of the NWII_DATA_OUT bus. When the write signal is reset, the bits will reload their current state

Bit 17 of the register is the Verify Error Status for the Scan Engine. This bit needs to be set and held when a verification error is detected during a scan read with verify operation. The R.CMD_WR_HI* bit controls the select line to a 2:1 multiplexor. When the write signal is set, the corresponding bit of the NWII_DATA_OUT bus is loaded into the register. When the write signal is reset, the register will be loaded from the output of the error set and hold logic. The set and hold logic monitors the state of the R.VERIFY_ERR, signal setting the output when the error bit is set in conjunction with bit 6 of the Scan Engine CMD_STAT register. The registered version of the pal output is fed back to the pal so that it can hold state until the clearing term is written by the SPU.

Bit 16 of the register is the Scan Engine Busy bit which is read by the SPU to determine if a scan operation has completed or not. While a scan operation is executing, the bit will be set. The free running register is loaded with the output of a 2:1 multiplexor whose select line is driven the the R.CMD_WR_HI* signal. If the write signal is set, the corresponding bit of the NWII_DATA_OUT bus is loaded into the register. If the write signal is reset, the register monitors the state of the R.SCAN_DUN and R.IN_BUF_WRD_CNT signals, which indicate when a scan operation is complete. A scan write operation is complete when the last bit is shifted out, which is when R.SCAN_DUN becomes set. R.SCAN_DUN is the terminal count of the Scan Counter. A scan read operation is complete when the last scan word is written to the Scan Memory. This is indicated when the R.SCAN_DUN signal is set while the R.IN_BUF_WRD_CNT signal is reset.

The logic for the Scan Engine Command Status Register is contained on pages 1 and 29 of the Scan Engine Schematics.

Because the Parity Error log is limited to 24 bits, not all of the error state can be recovered. The Error Log saves the received parity, the CG_PAR_ERR bits, the Address where the error occurred and the most significant Data byte that had a parity error. The four CG_PAR_ERR bits are priority encoded to create the select lines for the data byte multiplexor. If any single byte is in error, it will be selected as the input to the Log Register. However, if more than one byte is in error, then the most significant byte that is in error will be selected. The most significant byte of the word is defined as byte 3, which contains bits 7 through 0. The Clock Generator Parity Error Log can be found on page 8 of the NCU top level schematics.

3.5.3.1.7 Scan Counter and Scan Counter Register

The Scan Counter and Scan Counter Register, form a 16 bit wide logic block that keeps track of the number of bits remaining in a given Scan Operation. The Scan Counter Register is a 16 bit parallel load register that contains the one's complement of the value that is loaded into the Scan Counter when the Scan Command Status Register is written. The Scan Counter Register can be directly written and read by the SPU. The Scan Counter can be directly read by the SPU, but can only be written from the SPU by first writing the one's complement of the desired load value to the Scan Counter Register followed by a data independent write to the Scan Command Register. During scan operations, the Scan Counter Register is loaded with the ring length before the scan. The one's complement operation is handled in hardware.

The three pals attached to the data inputs of the Scan Counter (all 181-005472) are used to load the one's complement of the Scan Counter Register into the Scan Counter and to allow the Scan Counter to Scan. The three pals together form a 16 bit 2:1 multiplexor, where one of the inputs is inverting. The inverting inputs of the multiplexor are connected to the Scan Counter Register and are always selected unless the NCU board is in a Board Left scan (Scan Modes 101 or 111, and the CAST scan enable set). If the board is in a Board Left Scan mode, the non-inverting input of the multiplexor is selected. The non-inverting inputs of the multplexor are connected to the outputs of the Scan Counter in such a way that a shift left will occur. During the Board Left scan modes, the parallel load signal for the counter is continuously set. This causes the counter to perform a shift left operation on each clock cycle. External hardware for scan is required for the Scan Counter because the devices that implement it (100e016) do not support scan.

When the Scan Engine is scanning a BUT, the Scan Counter will increment in lock step with the shifting of either the Input or Output Shift Register. If the Scan Engine is performing a scan read operation, then the Scan Counter will increment in lock step with shifting of the Input Shift Register. If only a scan write operation is being performed, then the counter will increment in lock step with the shifting of the Output Shift Register. The count enable signal for the Scan Counter is the result of the logical ANDing of the increment signals from the Input and Output Scan Control logic contained on pages 35 and 36 of the Scan Engine schematics. The AND gate is found on page 40.The block diagram for this logic is illustrated in figure 3-43.

During the CCU scan operation, the counter increments by one each time the R.CCU_DIS_GOOD is set. The count enable for the CCU Read RBE counter is output from pal 181-005482, which is found on page 36 of the Scan Engine schematics. Pal 181-005476 on page 40 of the Scan Engine schematics monitors the output of the CCU Read RBE counter and generates the terminal count when the counter reaches a value of 0xFF. The terminal count is used by pal 181-005485 on page 35 to create the R.CCU_READ_RBE_A signal. The R.CCU_READ_RBE signal, is the signal which gates off the CU_XC.NIA_CCU_RBE signals generated on pages 31 and 32 of the Scan Engine schematics.Figure 3-34 illustrates the timing of counter and its relationship to the NIA_CCU_RBE disabling.

3.5.3.1.10 CCU Phase Monitor and Scan Logic

The CCU Phase Monitor and scan logic is an autonomous logic block that is responsible for mirroring the actions of the NIA and NCLK phase monitors and notifying the Input and Output Scan Engines when the CCU clocks are in a scannable phase. There are two outputs from this logic block. The C.SCAN_FAZE_GOOD signal is used to indicate when output scan data can be shifted and input scan data can be sampled. When the signal is set, the data can be reliably shifted and sampled. The C.CCU_DIS_GOOD signal is used as the increment signal for the CCU read RBE counter. When the signal is set, the CCU read RBE counter will increment by one on the next clock. Two pals, 181-005479 and 181-005440, and two eight bit binary up counters make up the entirety of the logic block.

The Phase Monitor uses three signals from the NCLK array to determine and how and when to increment the phase monitor's counter. When the CG_DVE signal is set, the counter will be a mod18 counter, and when it is reset, it is a mod20 counter. By definition, all of the phase monitors (Scan Engine, NCLK, NIA) will increment when their respective rbe is set. In the case of the CCU phase monitor, the DMASTER_RBED signal is monitored to control counter incrementing. When the signal is reset, the counter value is held. The DMASTER_SO signal is used to synchronize the phase monitors on the different boards. When the signal is set, the counter is loaded with the value corresponding to phase 01 on the following clock. If the CG_DVE signal is set (mod18), the value loaded is 0xEF, otherwise, 0xED is loaded into the counter.

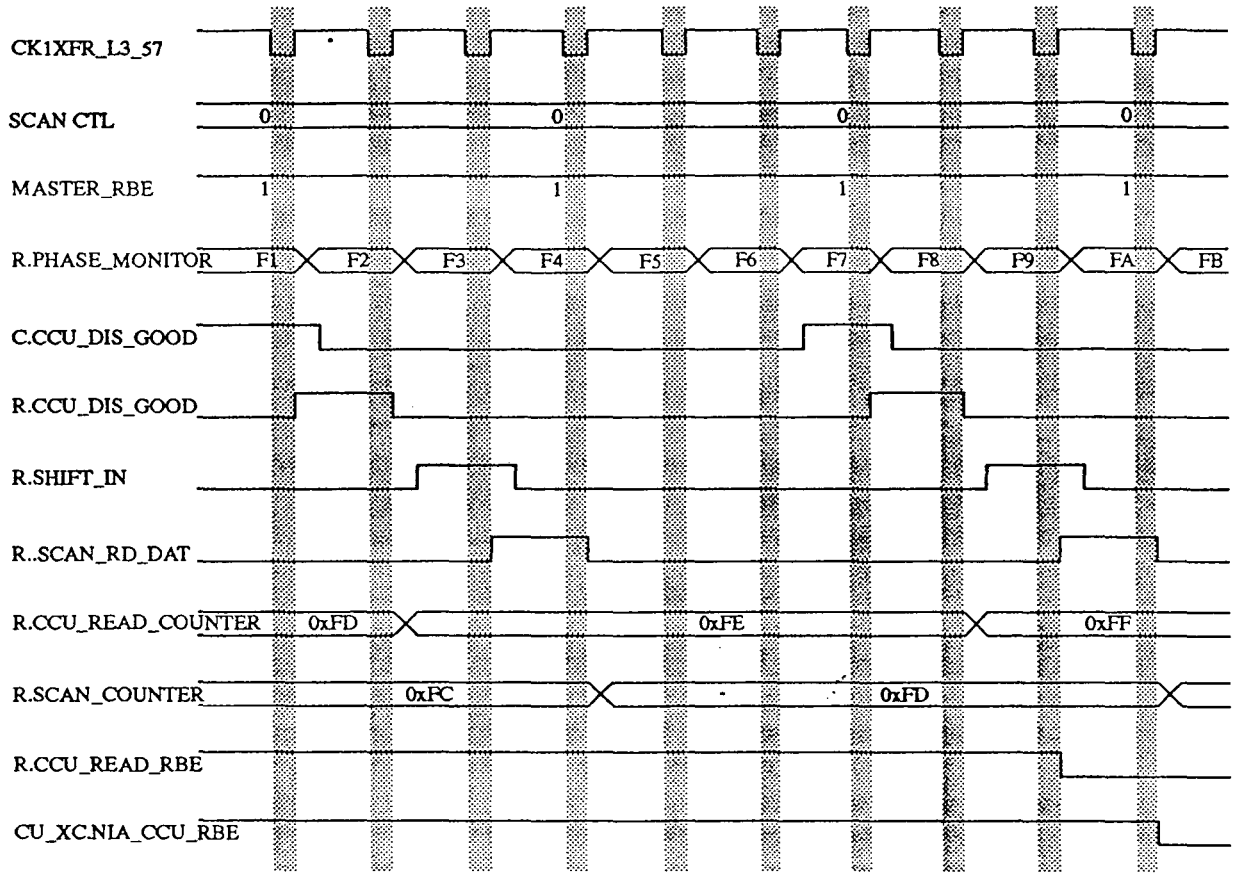
When the counter reaches a value of 0xFF, the terminal count becomes active, and the counter is reloaded with the phase 00 value. When the CG_DVE signal is reset, the reload value is 0xEE, otherwise the value is 0xEC.

Pal 181-005479 is used to decode the state of the counter, generating the C.SCAN_FAZE_GOOD and the counter parallel load and count enables. The CG_DVE signal is monitored because the mod18 and mod20 scannable phases are different. When CG_DVE is set, the scannable counter values are 0xF3, 0xF9, 0xFF. When CG_DVE is reset, the scannable counter values are 0xEC, 0xF2, 0xF9. When these counter values are detected, the C.SCAN_FAZE_GOOD is set. The decoded phases are each one earlier than the scannable phases so that the R.SCAN_FAZE_GOOD, which is used by the Input and Output Scan Engines, is set during the scannable phase.

The C.CCU_DIS_GOOD signal indicates the valid states when the RBE to a CCU (being read) can be disabled. Each time the signal is set, if the RBE to a given CCU is disabled, clocks to that CCU will be disabled just after the falling edge of a PBUS clock. This guarantees that CCU clocks will always be disabled in a scannable phase. When the CG_DVE is set, the valid counter values are 0xF7, 0xFD, and 0xF1. When CG_DVE is reset, the valid counter values are 0xF0, 0xF9, 0xFF.

The CCU Phase Monitor and scan logic can be found on pages 37 and 38 of the Scan Engine schematics.

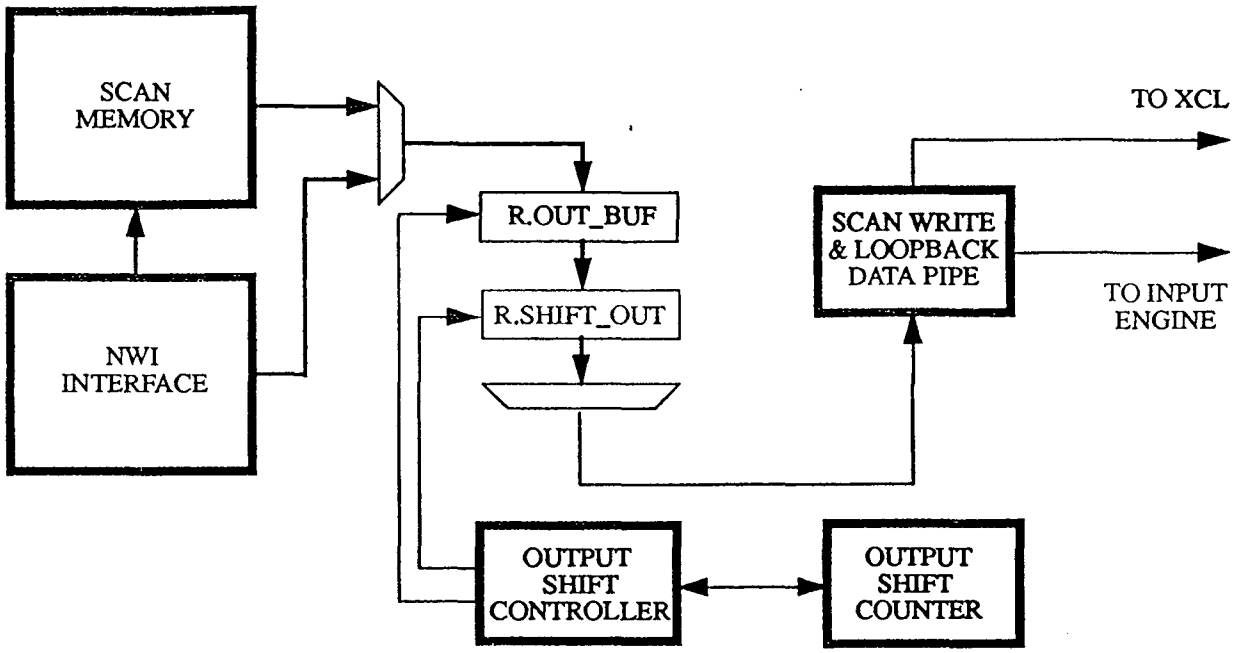
Figure 3-44: CCU Phase Monitor and Scan Logic Timing



3.5.3.2 Output Scan Engine

The Output Scan Engine converts r.out_buf data loaded by the Master Controller into a serial data stream that is output to each board in the system. The Output Scan Engine consists of a buffered, parallel load shift register coupled to a small state controller. The output shift counter which is connected to the state controller keeps track of how many bits have been shifted out and when data must be written to the Output Shift Register to avoid an underrun. The controller also keeps track of whether the buffer register contains valid data. The Output Shift controller will try to keep both registers loaded with valid data at all times. When either register is regarded as "not full", the controller will request new data from the Scan Memory by setting the data request to the Master Controller (R.OUT_BUF_WRD_CNT = 0).

Figure 3-45: Output Scan Engine Datapath Logic Diagram



3.5.3.2.1 Output Shift Controller

The Output Shift Controller is a group of pals and accompanying registers responsible for 1) the loading and shifting of the Output Shift Register, 2) Requesting new data for an "empty" Output Buffer Register 3) Controlling the incrementing of the Output Shift Counter and controlling the incrementing of the Scan Counter during write-only scan operations. The Output Shift Controller controls these operations by monitoring the state of the Output Shift Counter, Master Controller State Machine, Scan Command Register, and Clock Command Shadow Register. The Controller is implemented with pals 181-005481, 181-005480, 181-005482, and 181-005478. The logic for this controller is found on pages 36 and 37 of the Scan Engine schematic.

3.5.3.2.2 Output Shift Counter

The Output Shift Counter is a five bit, binary, up counter that is used to keep track of the number of bits left to be shifted out in the Output Shift Register. When this counter rolls over, the Output Shift Register is loaded with the contents of the Output Buffer Register. When the Scan Command Register is written (start of a scan operation) this counter is loaded with the contents of the Scan Counter Register. This allows the Output Shift Counter to keep track of the partial word that may be transferred (ring length is non-integral multiple of 32 bits) at the start of a scan operation. When the counter rolls over, it is reloaded with 0 since the remaining words loaded into the Output Shift Register contain 32 bits of scan data. The counter increments in lock step with each shift of the Output Shift Register. The Output Shift counter is contained on page 39 of the Scan Engine schematics. The pal that controls the counter's loading and incrementing (181-005481) is found on page 36 of the Scan Engine schematics.

3.5.3.2.3 Scan Write and Local Loopback Logic

The Scan Write and Local Loopback Logic contains the pipeline registers and multiplexors which source scan data to the XCL scan logic and the Input Scan Engine. The output of the Output Shift Multiplexor is input to a sample and hold pal (181-005480) and serial data pipeline that set up the various pipeline delays for the different scan operations. By decoding the scan operation from state of the Scan Command and Clock Command Shadow registers, the write and loopback logic can select a tap point in the outgoing delay pipeline, which is found on page 36 of the Scan Engine schematics. The multiplexor which selects the pipeline tap point is implemented in pal 181-005475, found on page 29 of the Scan Engine schematics.

3.5.3.2.4 Output Buffer Register

The Output Buffer Register is a 32 bit, parallel load register that contains the next word that is to be loaded into the Output Shift Register. The register is fed by a 32 bit 2:1 multiplexor which receives data from the NWI Interface and the Data Output of the Scan Memory. The register is able to be written and read by the SPU.

The Output Buffer Register and its associated multiplexor are implemented in 4 - 100e241 registers and 6 - 100e155 multiplexors. The select lines for the multiplexor (one each for the upper and lower 16 bits of data) are driven by the write enables from the Scan Engine write decode logic. When the SPU performs a write to the Output Buffer, the write enables become set, (low) which selects the NWII_DATA_OUT bus as the input to the register. The logic for the Output Buffer and its multiplexor can be found on page 5 of the Scan Engine schematic.

The write enables for the Output Buffer registers are sourced from the logical ANDing of the (low true) write enables sourced by the Scan Engine write decode logic and the Output Shift Controller. In the case of simultaneous access by both drivers, the decode logic will have priority. If a simultaneous access occurs, the Output Shift Controller access is lost. The write enable AND gates are found on page 40 of the Scan Engine schematics. Since the Scan Engine only makes 32 bit accesses to scan memory, only a single control signal is used to form the upper and lower write enables. C.LOAD_OUT_BUF is sourced from Pal 181-005481 and can be found on page 36 of the Scan Engine schematic.

3.5.3.2.5 Output Shift Register

The Output Shift Register is a 32 bit, parallel load shift register that receives data from the Output Buffer Register and transfers data to the Output Shift Multiplexor. During Static and dynamic scan operations, the Output Shift Register will shift left on each clock. During CCU scan operations, the shift register will only shift when the CCU Phase Monitor detects a scannable phase. During the other clock periods the register will hold its state. The register is not directly accessable by the SPU.

The Output Shift Register is implemented in four 100e241 registers. Since data for the register is solely sourced by the Output Buffer Register, no input multiplexor is required. The shift control for te register is output from pal 181-005481, which is found on page 36 of the Scan Engine schematics. The load signal for the register is the is output from the logical NORing of the R.OSE_ST<0> signal and the C.SHIFTO_REG_LOAD output pal 181-05478. The R.OSE_ST<0> signal is responsible for the loading of the register that occurs prior to the scan. Once the board scan starts, R.OSE_ST<0> will remain reset, and C.SHIFTO_REG_LOAD controls the shift register load line. The logic for the shift register load signal can be found on page 37 of the Scan Engine schematics.

3.5.3.2.6 Output Shift Multiplexor

The Output Shift Multiplexor is a 32:1 multiplexor that selects a bit in the Output Shift Register to output to the Scan Write and Local Loopback Logic. During a scan operation, the mux will select no more than two different bits from the Output Shift Register. The first bit selected is the start position of scan ring in the shift register. Scan rings that are not a multiple of 32 bits will have partial word of data. Since this partial word is the first word loaded into the Output Shift Register, the first valid bit of scan data is will not necessarily be contained in the most significant bit position of the shift register. By loading the registered mux select lines with the same value as the Scan Count Register (at the same time), the mux selects the appropriate bit position in the Output Shift Register. The mux select line register will hold this value until the second word is loaded into the Ouput Shift Register.

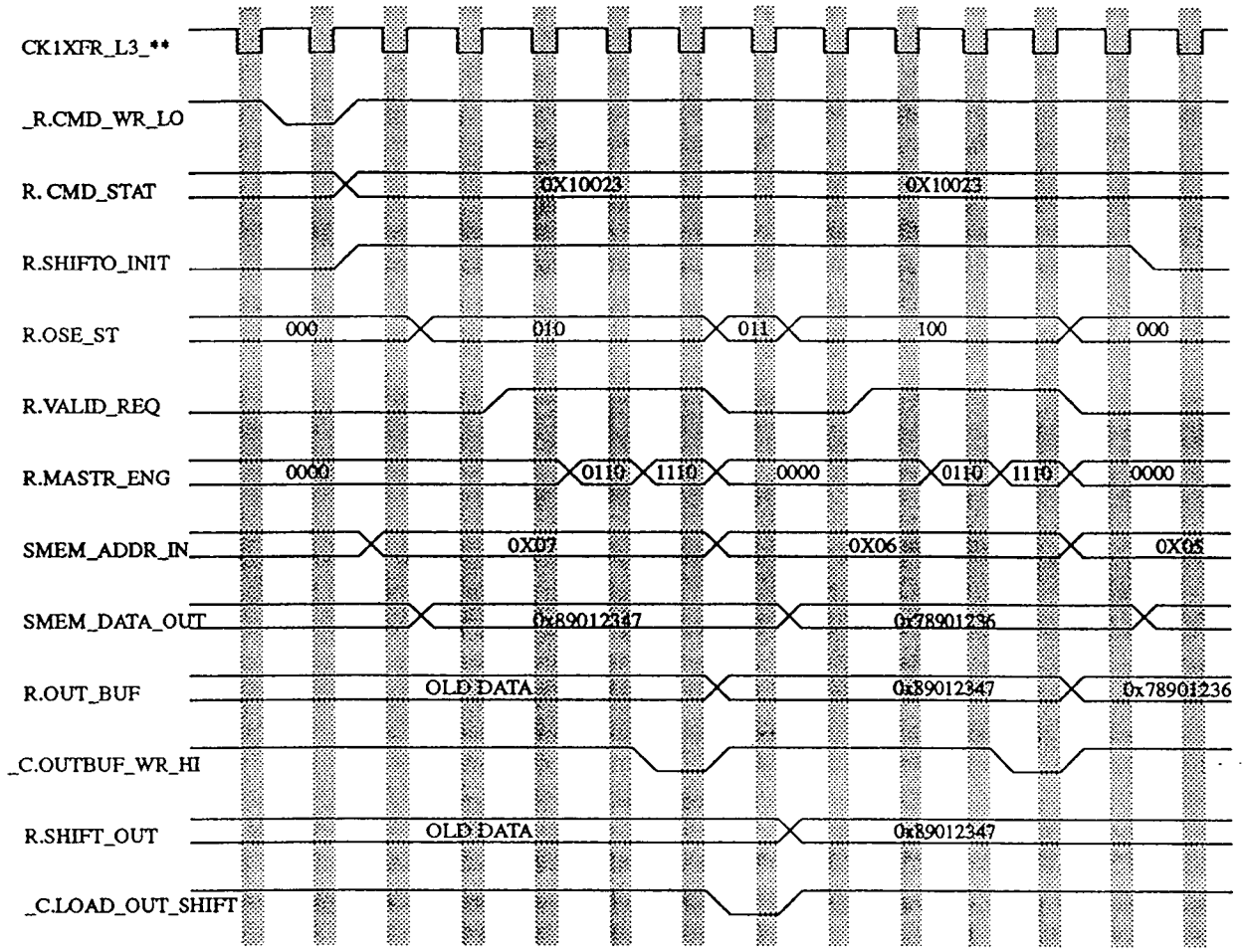
Starting with the second word that is loaded into the Output Shift Register, all remaining words that are loaded will contain full words (32 bits) of scan data. For the remainder of the scan operation, the select line register will contain a value of 0, which selects the most significant bit of the Output Shift Register. The multiplexor select register is loaded at the same time that the Output Shift Register is loaded. The load signal for the mux select line register is sourced from pal 181-005481. Note: If the ring length is an integral multiple of 32 bits, the mux select line register will contain a value of 0 for the entire scan operation.The multiplexor logic can be found on page 28 of the Scan Engine schematics. Pal 181-005481 is contained on page 36 of the Scan Engine schematics.

3.5.3.2.7 Output Scan Engine Data Priming

Between when the Scan Command Register is loaded and the scan command is executed in the Clock Generator, the Output Buffer Register and the Output Shift Register will be loaded with the outgoing data for the scan operation. This is done so that the Output Shift Register can start shifting data out to the BUT as soon as the Clock Generator is ready to start issuing clocks. This process is initiated by writing the Scan Command Register. Setting the R.CMD_WR_LO signal causes the R.SHIFTO_INIT signal to become set. This signal will remain set until the Output Buffer and Output Shift Registers are loaded with outgoing scan data.

The R.SHIFTO_INIT acts as a data request to the Master Control State Machine which fetches the first word of data from the Scan Memory. The fetched data is loaded into the Output Buffer Register and then the Output Shift Register on the following clock cycle. The cycle after the Output Shift Register is loaded with the first word of output data, the second data request is recognized by the Master Control State machine. This second word of data is loaded into the Output Buffer, and will remain there until the scan operation commences. The cycle after the second word is loaded into the Output Buffer Register, the data request is reset. One cycle after the R.VALID_REQ is reset for the second time, the R.SHIFTO_INIT signal is reset and the Output Scan Engine enters idle state until the SCAN_GO output of the Clock Generator becomes set. The timing for the fetching and loading of these first two words of scan data is illustrated in figure 3-46.

Figure 3-46: OSE Prescan Data Pipeline Priming

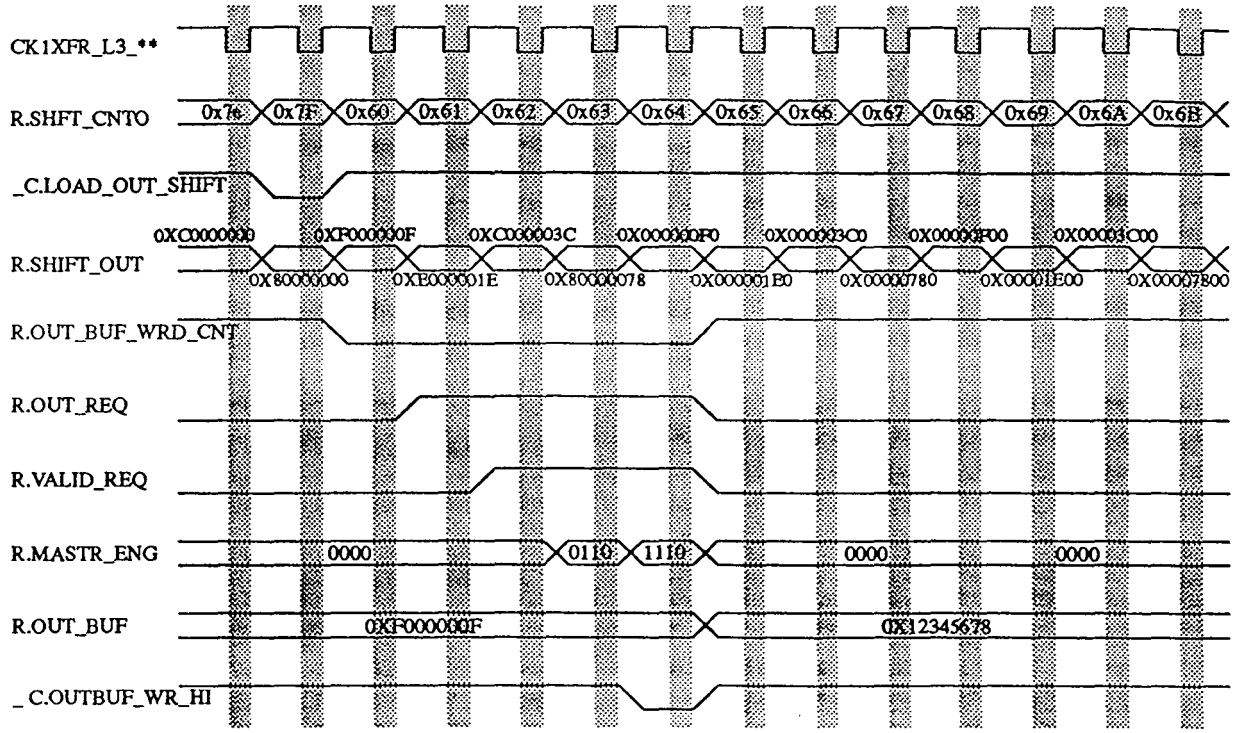


3.5.3.2.8 Output Scan Engine Data Loading (during scan)

As the Output Shift Counter reaches a value of 0x7F, the Output Shift Register contains the last bit of data that needs to be shifted out before new data needs to be reloaded into the register. When this counter value is detected, the _C.LOAD_OUT_SHIFT and _C.SHFT_CNTO_PE signals become set. On the following clock, the Output Shift Register is loaded with the contents of the Output Buffer Register, and the Output Shift Counter is reloaded with0x60.

The cycle that the Output Shift Register is loaded, the R.OUT_BUF_WRD_CNT signal is reset, which indicates that the Output Buffer is now empty. On the following clock cycle when this condition is recognized, a data request is set to the Master Controller State machine. While the data request is being serviced, the Output Shift Register continues to shift during every valid shift cycle. The cycle after scan data is loaded into the Output Buffer Register , the R.OUT_BUF_WRD_CNT signal is set, terminating the reload cycle. This process is illustrated in figure 3-47.

Figure 3-47: Output Shift Register Reload Timing (at word boundary)

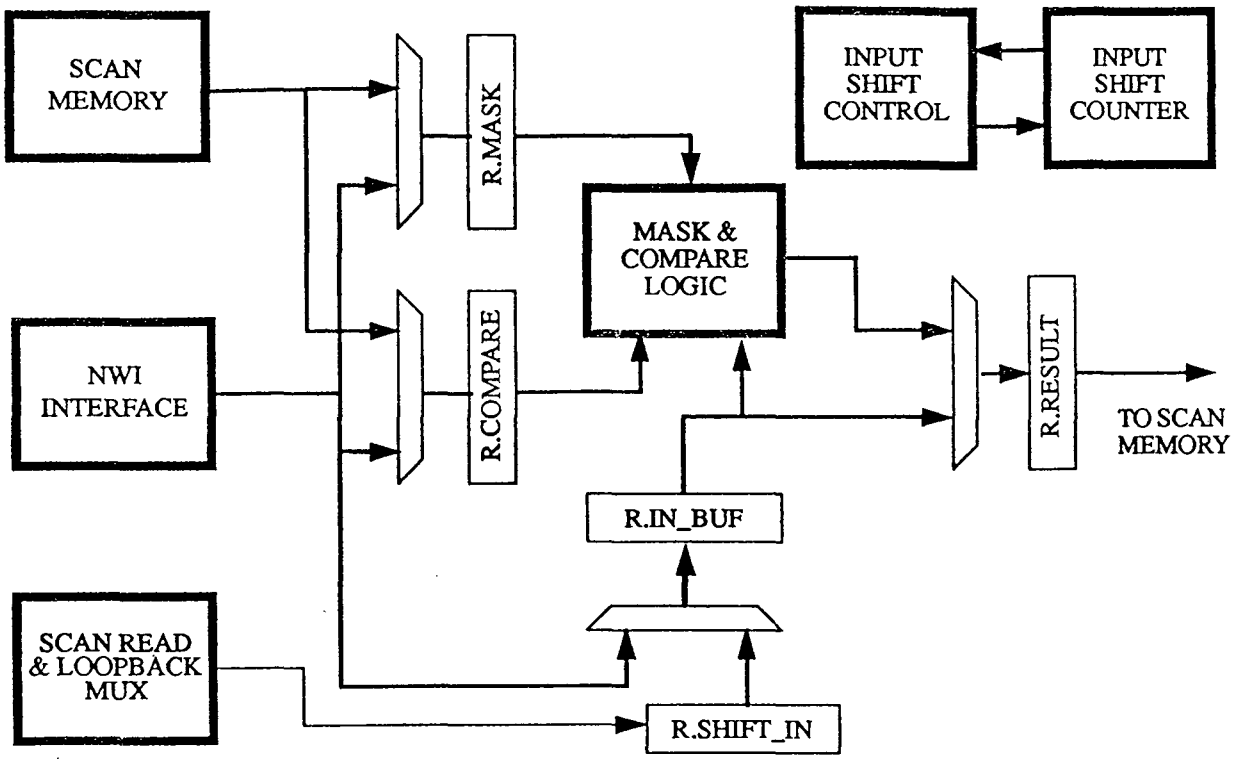


3.5.3.3 Input Scan Engine

The Input Scan Engine converts the serial scan data into words, masks and compares the data against expected values, and then has the data transferred to the Scan Memory. The Input Scan Engine is similar to the output engine because the circuit accomplishes serial to parallel conversion, is single buffered, and contains a small state controller. However, the Input Scan Engine differs from Output Engine in three important areas. First, the interface must make three Scan Memory Accesses for each word scanned in through the interface. Second, the interface can read or write Scan Memory Data. Third, during verification operations, the interface incorporates the logic required to mask and compare the scan data with an expected value, and store the error status and first bad location, if any.

While data is scanning in, the input controller makes two memory requests to retrieve the mask and compare value from the Scan Memory. After each word is read, the input controller modifies the two bit code that selects the page in memory (Mask, Compare, Result). The same offset in different memory pages will contain data that is associated with the same word of scan data . When the Input Buffer register is written, its output is logically ANDed with the output of the Mask Register to strip out certain data fields. The data is then compared against the expected data word. If any of the bits output from this operation are set, an error has been detected, and the VE bit in the Command/Status register is set.

Figure 3-48: Input Scan Engine Block Diagram



3.5.3.3.1 Input Shift Controller

The Input Shift Controller is a group of pals and accompanying registers that is responsible for 1) Controlling the shifting of the Input Shift Register. 2) The loading of the Input Buffer and Result Registers. 3) Incrementing and reloading the Input Shift Counter. 4) Incrementing the Scan Counter during scan read operations. and 5) Controlling the start of the Input Scan operation so that shifting syncs up with the data stream coming in from the board under test. The Input Shift Controller controls the above operations using state from the Clock Cmd Shadow Register, the Clock Generator array, the Scan Command Status Register, and the Input Shift Counter. Pals 181-005483, 181-005484, and 181-005485 make up the controller. The logic for the Input Shift Controller can be found on page 35 of the Scan Engine schematics.

3.5.3.3.2 Input Shift Counter

The Input Shift Counter is a five bit, binary, up counter that keeps track of the number of bits left to be shifted into the Input Shift Register before the data is word aligned. When this counter rolls over, the Input Buffer Register is loaded with the contents of the Input Shift Register. When the Scan Command Register is written (start of a scan operation) this counter is loaded with the lower five bits of the Scan Counter Register. This allows the Input Shift Counter to keep track of the partial word that may be transferred (ring length is non-integral multiple of 32 bits) at the start of

a scan operation. When the counter rolls over, it is reloaded with 0 since the remaining words shifted into the Input Shift Register are full words of scan data. The counter increments in lock step with each shift of the Input Shift Register. The Input Shift counter is contained on page 39 of the Scan Engine schematics. The pal that controls the counter's loading and incrementing (181-005484) is found on page 35 of the Scan Engine schematics.

3.5.3.3.3 Result Register

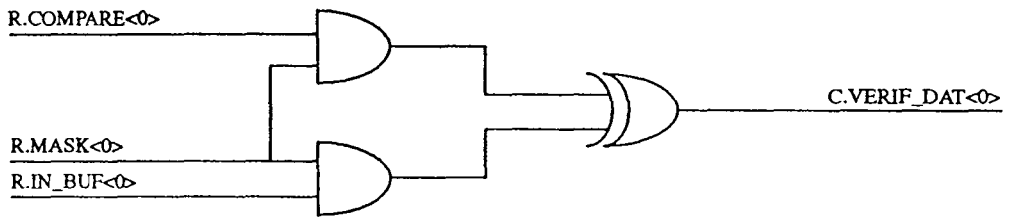
The Result Register is a 32 bit parallel load register coupled with a 32 bit 2:1 input multiplexor that contains the data that is written to Scan Memory during a scan read operation. This register can not be directly written or read from the SPU. The select line on the register's input multiplexor is driven by verify bit contained in the Scan Engine Command Status Register (R.CMD_STAT<6>). When this bit is set, the output of the Mask and Compare Logic is selected as the input to the register. When the bit is reset, the Input Buffer Register is selected as the Result Register's input data.

The load enable for the register is output from pal 181-005487, which contains the Input Scan Engine state machine. The load enable, _C.LOAD_RESULT_REG, will become active when the R.IN_BUF_WRD_CNT is set. This is the same condition that causes a write request to be issued to the Master Controller state machine. On the following clock cycle, the data to be written to the Scan Memory will be contained in the Result Register.

3.5.3.3.4 Mask and Compare Logic

The Mask and Compare logic is used to compare certain expected bit values against what is received from the BUT. The logic receives three 32 bit words from the Mask, Compare, and Input Buffer Registers and outputs a single 32 bit word to the Result Register input multiplexor. Within the logic, 32 - single bit operations are performed. First, the mask bit is logically ANDed with its equivilent bits in the Compare and Input Buffer Registers. The two outputs of these ANDing operations are exclusive ORed together. If the output of this logic is set, then there was a miscompare on the bit in question. The logic for this operation is illustrated below in figure 3-49. The Mask and Compare logic is implemented in seven pals (181-005486) and is found on page 13 of the Scan Engine schematics.

Figure 3-49: Mask and Compare Logic Diagram



3.5.3.3.5 Input Buffer Register

The Input Buffer Register is a 32 bit parallel load register coupled with a 32 bit 2:1 multiplexor, that contains the word aligned data received by the Input Shift Register. The Input Buffer Register can be written and read from the SPU. The SPU write enable for the register controls the select line on the multiplexor, selecting the NWII_DATA_OUT bus as the input to the register when the signal is asserted. During scan read operations, the register will be loaded when the Input Shift Counter reaches the terminal count value, 0xFF. The R.IN_BUF_WRD_CNT signal indicates whether the register contains scan data that needs to be written to the Scan Memory.

The load enable signal for the register is the result of the logical ANDing of the SPU write enable and the Input Scan Engine write enable. Since both are active low signals, if either is set, the register will be written with the output of the multiplexor. The AND gate for the load enable is found on page 40 of the Scan Engine schematics. The SPU write enables are found on page 15 and the Input Scan Engine write enable can be found on page page 35. The ISE write enable is output from pal 181-005484. The Input Buffer Register and its input multiplexor can be found on page 6 of the Scan Engine schematics.

3.5.3.3.6 Compare Data Register

The Compare Data Register is a 32 bit parallel load register coupled with 32 bit 2:1 multiplexor, that contains the Compare data used during Scan Read with Verify operations. The Compare Register can be written and read from the SPU. The SPU write enable controls the select line on the multiplexor, selecting the NWII_DATA_OUT bus as the input to the register when the signal is asserted. During a Scan Verification operation, the Input Scan Engine will try to load this register as soon as possible with the Compare data for the next word to be loaded into the Input Buffer Register.

The load signal for the Compare Register is the result of the logical ANDing of the SPU write enables with the Input Scan Engine write enable. Since both are active low signals, if either becomes active, the register will be written. This logic is found on page 40 of the Scan Engine schematics. The SPU write enables for this logic are sourced from page 15 and the ISE write enable is output from pal 181-005487 found on page 34 of the Scan Engine schematics. The Compare Register and its input data multiplexor can be found on page 3 of the Scan Engine schematics.

3.5.3.3.7 Mask Data Register

The Mask Data Register is a 32 bit parallel load register coupled with a 32 bit 2:1 multiplexor, that contains the mask data used during Scan Read with Verify operations. The Mask Register can be written and read from the SPU. The SPU write enable for the register controls the select line on the multiplexor, selecting the NWII_DATA_OUT bus as the input to the register when the enable is set. During a Scan Verification operation, the Input Scan Engine will try and load this register as soon as possible with the Mask data for the next word to be loaded into the Input Buffer Register.

The load signal for the Mask Register is the result of the logical ANDing of the SPU write enables with the Input Scan Engine write enable. Since both are active low signals, if either becomes active , the register will be written. This logic is found on page 40 of the Scan Engine schematics. The SPU write enables for the register are sourced from logic on page 15 and the ISE write enable is sourced from pal 181-005487 found on page 34 of the Scan Engine schematics.The Mask register and its input data multiplexor can be found on page 4 of the Scan Engine schematics.

3.5.3.3.8 Input Shift Register

The Input Shift Register, R.IN_SHIFT, is a 32 bit, serial in, parallel out register that is used to convert the serial data stream output from the Scan Read & Loopback multiplexor to a 32 bit word that is loaded into the Input Buffer Register. During static and dynamic scan operations, this register will shift left once on each rising edge of each 1x clock. During CCU scans, the Input Shift Register will shift once, one clock cycle after the R.SCAN_FAZE_GOOD is set. Since this register never parallel loads, the load signal is reset so that the only valid operations for the register are hold and shift.The pal that controls the shifting of the register, 181-005484, is found on page 35 of the Scan Engine schematics. The Input Shift Register is found on page 8 of the Scan Engine schematics.

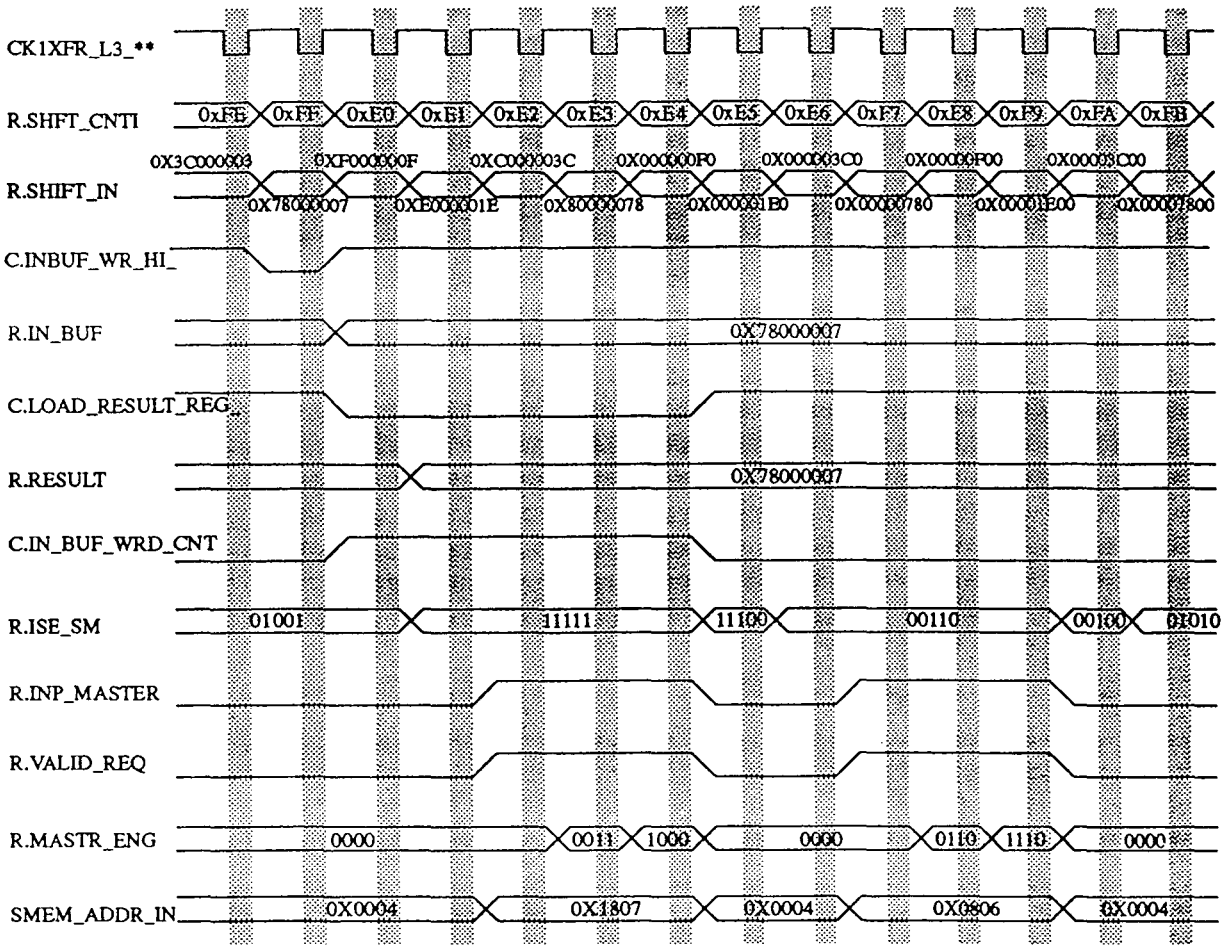
3.5.3.3.9 Input Scan Engine Scan Memory Access

After a word of scan data is completely scanned into the Input Shift Register, the process of writing the received data to memory begins. As the Input Shift Counter reaches a terminal count value of 0xFF, the write enable for the Input Buffer Register is set. On the next cycle, the data from the Input Shift Register is loaded into the Input Buffer Register. While the data is loaded into the buffer register, the Input Shift Register shifts again to start receiving the next word of scan data.

As the data is registered in the Input Buffer Register, the R.IN_BUF_WRD_CNT signal becomes active and the write enable for the Result Register becomes active. These two signals are logical inverses of one another. Depending on the state of the R.CMD_STAT<6> (verify bit) signal, the result register will be loaded with either the output of the Mask and Compare logic or the Input Buffer register. Since the Mask and Compare logic is combinational, timing for the two busses of data will be similar. The cycle after write enable becomes active, the new data will be loaded into the Result Register in preparation for its write to the Scan Memory.

The R.IN_BUF_WRD_CNT signal is interpretted as a Scan Memory write request by the Input Scan Engine state machine. When this signal is set, the ISE state machine transitions from the SCAN_DIN_WAIT to the RESULT_REQ state. This kicks off a request to the Arbiter which causes the Memory Controller to complete the access. The memory write cycle is completed when the Memory Control state machine asserts the memory acknowledge signal, R.MASTR_ENG<3>. When the acknowledge signal is detected, R.IN_BUF_WRD_CNT and C.LOAD_RESULT_REG_ signals are reset. Once the write cycle is completed, the Input Scan Engine starts the read cycle to fetch the Mask data for the next word being scanned in if verification mode is selected. If the verify bit is reset in the Scan Command Status Register, then the Input Scan Engine state machine returns to the SCAN_DIN_WAIT state and remains there until the next word is loaded into the Input Buffer Register. The timing diagram that illustrates the transfer of data from the Input Scan Engine to the Scan Memory is illustrated below.

Figure 3-50: Input Scan Engine Timing (verify mode)



3.5.3.3.10 Completion of a Scan Read Operation

The completion of a scan read operation is an important process to understand because it is the one time during a scan operation that the Input Scan Engine operates while the Clock Generator is idle. One cycle before the last clock is issued to the BUT and the last bit of valid scan data is registered in the Input Shift Register, the Clock Generator resets the SCAN_GO signal. This signal is used by most of the Scan Engine as an enable for Control signals. However, since the last word of scan ring is still contained in the Input Shift Register, most of the Input Scan Engine must remain operational until the last word is finally written to the Scan Memory.

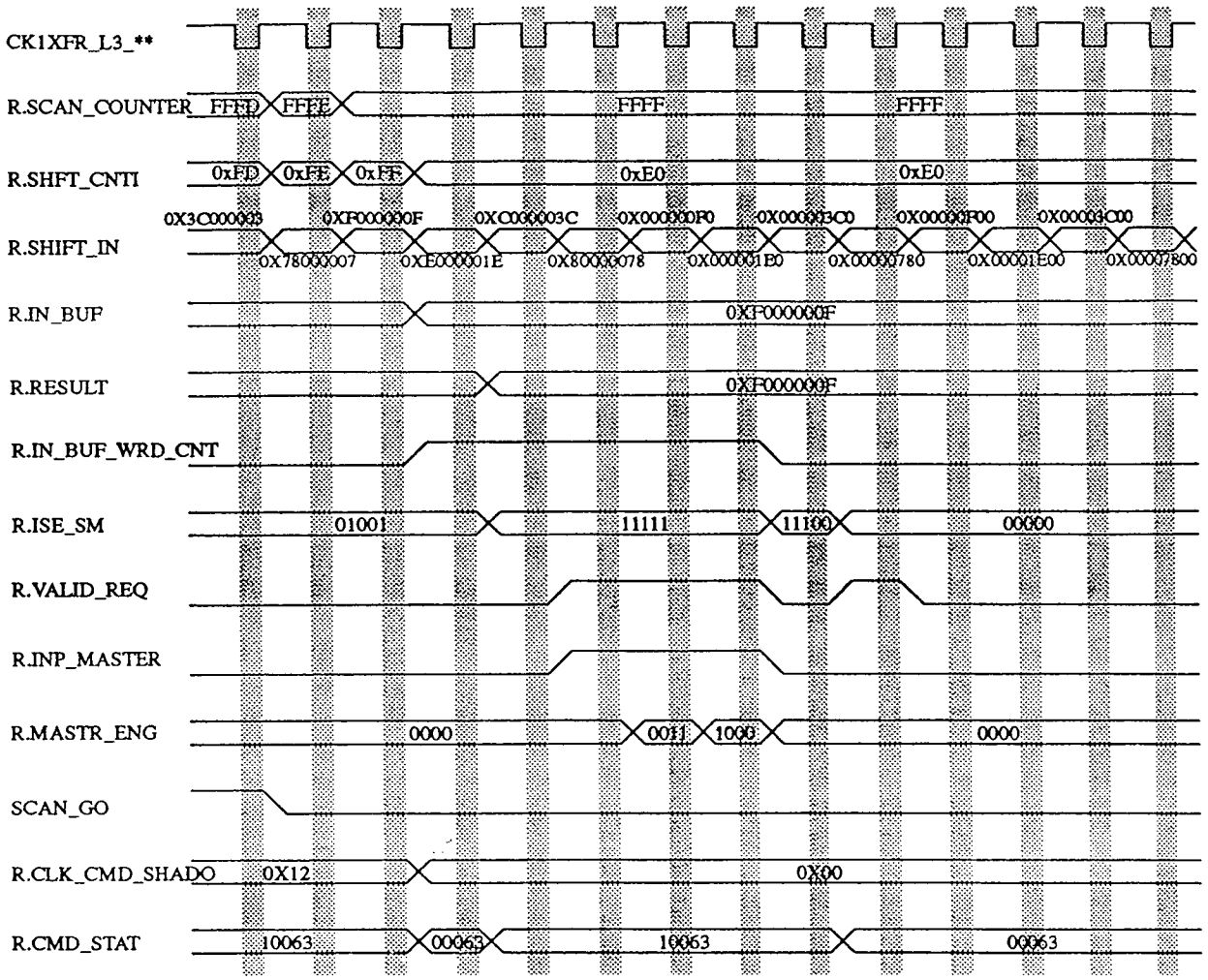
As the last clock is issued to the BUT and the last bit of scan data is clocked into the Input Scan Engine, the Scan Counter and the Input Shift Counter reach terminal count. On the following cycle, the R.SCAN_DUN signal becomes active freezing the Scan Counter at the terminal count value. The Input Shift Counter rolls over to its load value of 0xE0 before freezing its value. The cycle after the Scan Counter reaches terminal count, the Busy bit in the Scan Engine Command Status Register (R.CMD_STAT<16>) is reset for a single clock cycle. This occurs because the Busy bit is a logic ORing of the R.SCAN_DUN and R.IN_BUF_WRD_CNT signals. Since the Memory

Control and Input Scan Engine State Machines are in the idle state during the cycle that this signal resets, no damage is done. However, until the Busy bit becomes set again, the remainder of the write operation is frozen. On the following cycle, the busy bit is set because the R.IN_BUF_WRD_CNT signal is active. The cycle after R.CMD_STAT<16> becomes set, the two state machines can start the write cycle for the data contained in the result register.

The remainder of the write cycle is the same as any other Scan Engine data write to the Scan Memory. The only difference between this write cycle and others is that once the write is completed and the R.IN_BUF_WRD_CNT signal becomes reset, the scan operation is terminated and all state machines return to their idle state the cycle after the Busy bit is reset. When R.IN_BUF_WRD_CNT becomes reset, R.CMD_STAT<16> will become reset on the following cycle. This is a signal to the SPU and the Scan Engine that the operation is complete.

One can see from the timing diagram below that after the final write completes, that the Scan Engine attempts to fetch the Mask data for the next word of scan data. Since there is no more scan data to be received, this is a bogus access. However, this access doesn't get very far because the Busy bit is reset on the following clock cycle. The cycle after the Busy bit is reset, the Master Arbiter, Memory Controller and Input Scan Engine state machines will be forced into the Idle state. This guarantees that the Scan Engine will start from a known good state for the next scan operation. This is the primary reason that the Scan Engine performs a local loopback scan as part of the initialization process. Regardless of the initial state of the machine, the state machines will be in a known good state (idle) at the end of the operation.

Figure 3-51: Scan Completion Timing



3.5.3.3.11 Scan Verification Error Logic

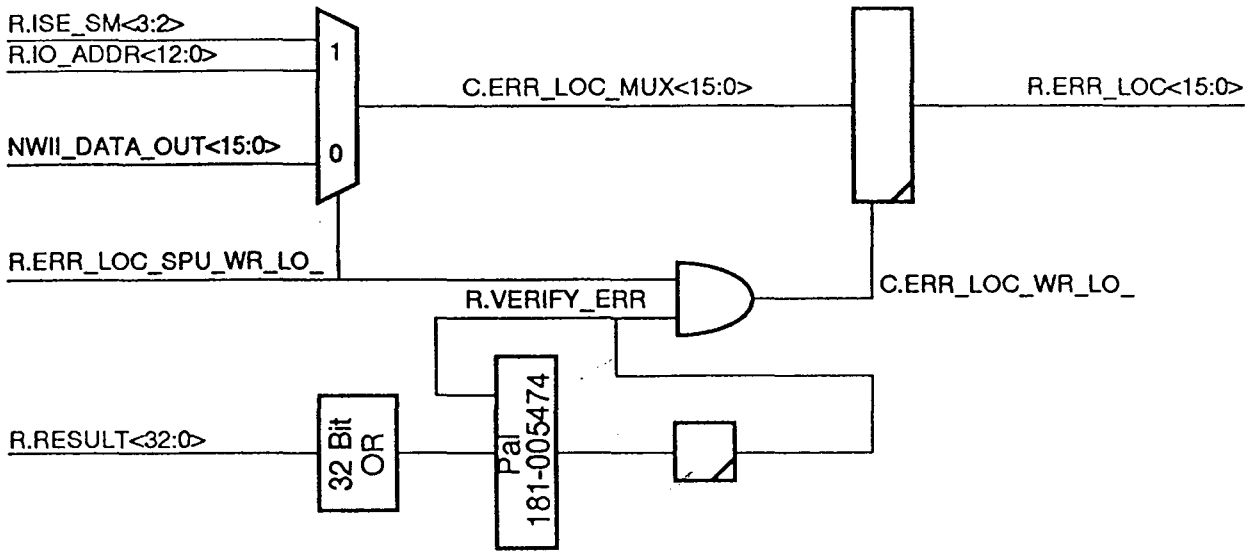
During a Scan Verification operation, the Result Register is loaded with the output of the Mask and Compare logic. If any bit in the register is set, then a verify error has been detected . When this condition is encountered, the Verify Error bit is set and held, and the address where the error occured is stored. Storing the address of the error eliminates the need for software to search through the result page of memory for the first error. The Verify Error bit and the offending address will remain held until a 0 is written to bit 17 of the Scan Command Register.

A group of 100e101s forms a 32 bit OR gate that constantly monitors the output of the Result Register. If one or more bits is set in the register, the C.RESULT_NOZERO signal will become set. During a Verification scan operation, this signal is referenced by pal 181-005474, using it to generate C.COM_STAT<17>, which is the Verify Error bit (VE) in the Scan Engine Command Status Register. The registered verision of this signal, R.VERIFY_ERR, is fed back to the pal and used to hold the state of the bit as long as R.CMD_WR_HI* is reset.

During a Scan Verification operation, the Error Location Register continues to load on every clock until the R.VERIFY_ERR signal becomes set. The Register is loaded with the one's complement of the Result page address presented to memory (the output of the Input Address Counter). When the error signal is set, the register is put in hold mode and will remain in that state until the R.VERIFY_ERR signal is reset. The Scan Verification logic is contained on pages 1,8, and 29 of the Scan Engine schematics. The block diagram for this logic is illustrated in figure 3-43.

Note: Even though a Verification Error is detected, the scan operation will complete.

Figure 3-52: Verification Logic Block Diagram



3.5.3.4 Run Bit Enable Logic

Run Bit Enables are used to gate off certain clocks on a BUT while the remaining logic continues to free run. Boards that are affected by this logic are the NIA, NMB,all CCUs, and all XBAR boards (except the XCL). When a RBE's enable is set, the MASTER_RBE signal output from the NCLK array is used to control the operation of the selected Run Bit Enable. A brief description and illustration of each type of Run Bit Enable follows.

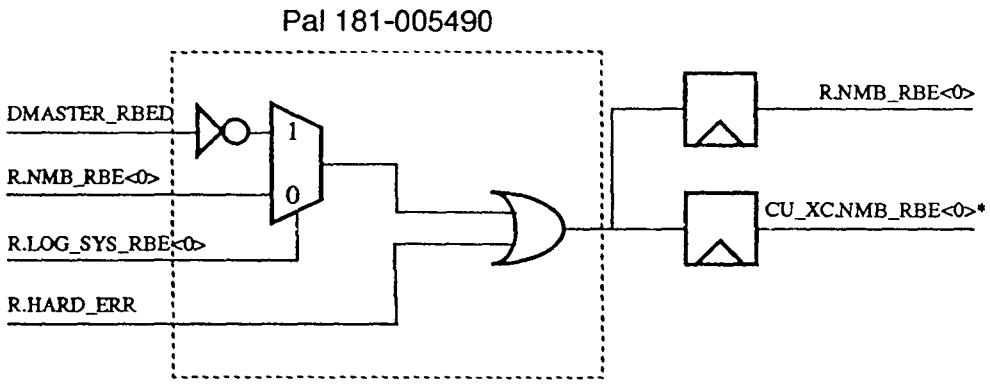
3.5.3.4.1 NMB Log and Sys Run Bits

The NMB Log and Sys Run bits are used by NMB to scan the board's two dynamic rings and to step the board logic. The desired function is selected by the state of the board's scan control lines. If the NMB scan controls select Normal mode (Scan Ctl = 000), then the Log and Sys run bits are used to step the board logic. If the scan controls select Shift Left Log mode (Scan Ctl = 001), the Log Run bit is used to control Soft Error Log scans of the board. If the Shift Left Sys mode is selected (Scan Ctl = 011), the Sys Run bit is used to control system ring scans. A more detailed description of uses of these bits in conjunction with the scan controls can be found in the NMB Functional Specification.

Another function of the Log and Sys Run bits is to disable clocks to the NMB when an unmasked Hard Error is detected. The Log and Sys Run bits are used to disable the clocks on the NMB because during a Hard Error condition, the board clock must still drive certain clocks which will continue to run in order for memory refresh cycles to occur. Figure 3-53 illustrates how the Hard Error signal is used to control the operation of the NMB Run bits. Whenever the R.HARD_ERR signal is set, the output of the OR gate is forced high, forcing the active low Run Bits into a reset state. This disables the clocks to the NMB's dynamic ring logic until the Hard Error condition is cleared

A two input multiplexor selects either the current state of the Run Bit or the Master_RBE as the signal which will be driven to the NMB on the following clock. A board's Run Bit enable contained in the NMB Log and Sys RBE Enable Register (bits 0 thru 15) is used to control the select line on the 2:1 multiplexor. When a board's enable is set, the inverted version of the Master_RBE is passed on to the board. If the enable is reset, the current state of a shadow copy of the run bit is recirculated through the mux, holding the state of the Run Bit. A shadow copy is recirculated because it is a bad design practice to use an outgoing interboard signal as an input to an onboard register. Figure 3-53 illustrates the logic diagram for one of the NMBs' sixteen Run Bits. All of the combinational logic in Figure 3-53 is contained pal 181-005490. Three pals are required to implement the sixteen Run Bits. The NMB Log and Sys Run Bit logic can be found on page 33 of the Scan Engine schematics.

Figure 3-53: NMB Log and Sys Run Bit Logic Diagram

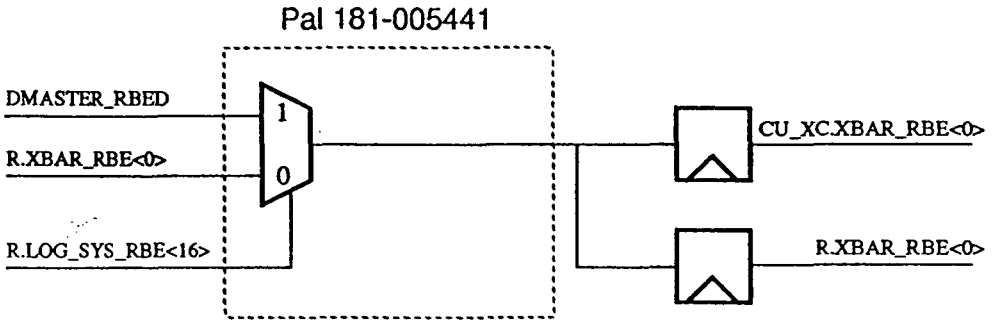


3.5.3.4.2 XBAR Config_load Run Bits

The XBAR Config_load Run Bits are used to scan the six Sytem Configuration (Config)rings in the Crossbar. The XCL is the only XBAR board that doesn't have a Config ring. If a XBAR board's scan controls are in normal mode (Scan Ctl = 000) and its run bit is set, the configuration ring will shift. If the bit is reset, or the scan controls select anything but Normal or Board Left modes, the ring will hold state. A more detailed description of this operation can be found in the XBAR boards Functional specification.

A two input multiplexor selects either the current state of the Run Bit or the Master_RBE as the signal which will be driven to the selected XBAR board on the following clock. A board's Run Bit enable contained in the NMB Log and Sys RBE Enable Register (bits 16 thru 21) is used to control the select line on the 2:1 multiplexor. When a board's enable is set, the Master_RBE signal is passed on to the board. If the enable is reset, the current state of a shadow copy of the run bit is recirculated through the mux, holding the state of the Run Bit. A shadow copy is recirculated because it is considered a bad design practice to use an outgoing, interboard signal as an input to an onboard register. Figure 3-54 illustrates the logic diagram for one of the XBARs' six Run Bits. All of the combinational logic in Figure 3-54 is contained pal 181-005441. A single pal is used to implement the XBAR Run Bits. The XBAR Config_load Run Bit logic can be found on page 33 of the Scan Engine schematics.

Figure 3-54: XBAR Config_load Run Bit Logic Diagram

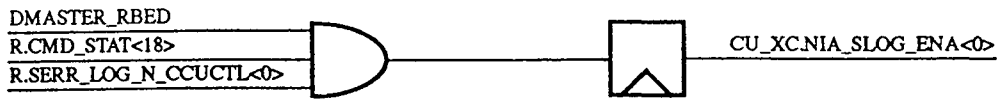


3.5.3.4.3 NIA CCU Ctl and Slog Run Bits

The NIA CCU Ctl and Slog Run Bits are used to enable scans of the NIA's CCU Control and Soft Error Log rings. Each NIA in the system (nine total) will receive a CCU Ctl and Slog Run Bit. When one of the two bits is set in conjunction with the appropriate scan control code, one of the board's dynamic rings will scan. The Soft Error Log ring will scan when the board's NIA_SLOG_ENA bit is set while the scan controls are in Shift Left Log mode (Scan Ctl = 001). The CCU Control ring will scan when the board's NIA_CCU_CNTRL_ENA bit is set while the scan controls are in Load CCU Scan Ctl mode (Scan Ctl = 100). Both Run Bits should not be set at the same time. A more detailed description of these scan operations can be found in the NIA Functional Specification.

A NIA's CCU Ctl or Slog Run Bit will be set if the board's enable, the Master_RBE, and one of the two dynamic ring enables (contained in the Scan Command Register) are set. The Log Ring scan bit (R.CMD_STAT<18>) enables Soft Error Log ring scans, and the Sys Ring scan bit (R.CMD_STAT<19>) enables CCU Control ring scans. The combinational portion of this logic is implemented in both a pal (181-005492) and descrete logic. The NIA CCU Ctl and Slog Run Bit logic is contained on pages 30 and 44 of the Scan Engine schematics.

Figure 3-55: NIA CCU Ctl and Slog Run Bit Logic Diagram



3.5.3.4.4 NIA CCU Run Bit Enables

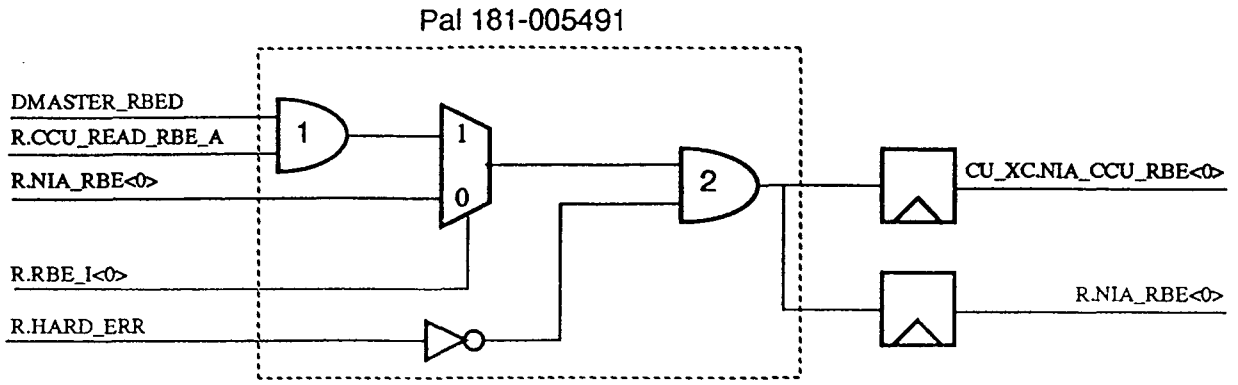
The NIA CCU Run Bit Enables are used to control the clocking of CCUs attached to the different NIAs in the system. There are 40 CCU Run Bit Enables. Eight Run Bits are assigned to the primary NIA in port 8, and four Run Bits are assigned to each of the expansion NIAs that can populate ports 0 thru 7. When a Run Bit is set, clocks to the specified CCU are enabled. The logic diagram for a single NIA CCU Run Bit Enable is illustrated in Figure 3-56.

The registered system hard error signal, R.HARD_ERR, has priority over other controls for the CCU Run Bit logic. When the R.HARD_ERR is asserted, the output of AND gate 2 is forced into reset, and the Run Bits are reset on the following clock. The Run Bits will remain reset until the Hard Error condition is cleared. When the hard error condition is cleared, the output of the 2 to 1 mux will be output to the NIA on the following clock.

A two input multiplexor selects either the current state of the Run Bit or the output of AND gate 1 as the signal which will be driven to the XCL on the following clock. A board's Run Bit enable contained in one of the two CCU Run Bit Enable Registers is used to control the select line on the two to one multiplexor. When a board's enable is set, the output of AND gate 1 is passed on to the board. If the enable is reset, the current state of a shadow copy of the run bit is recirculated through the mux, holding the state of the Run Bit. All of the combinational logic in Figure 3-56 is contained pal 181-005491. Seven pals are used to implement the NIA CCU Run Bits. The NIA CCU Run Bit logic can be found on pages 31 and 32 of the Scan Engine schematics.

The DMASTER_RBED signal is ANDed with R.CCU_READ_RBE_A before the multiplexor so that the CCU Run Bit can be disabled early during CCU scan read operations. R.CCU_READ_RBE_A is the registered output of pal 181-005485 (page 35 of Scan Engine schematics) which, during CCU scan read operations, will become reset when the CCU_READ_RBE counter reaches 0xFFFF. During all other operations, this signal will be set so that DMASTER_RBED will have control over the mux input.

Figure 3-56: NIA CCU Run Bit Enable Logic Diagram



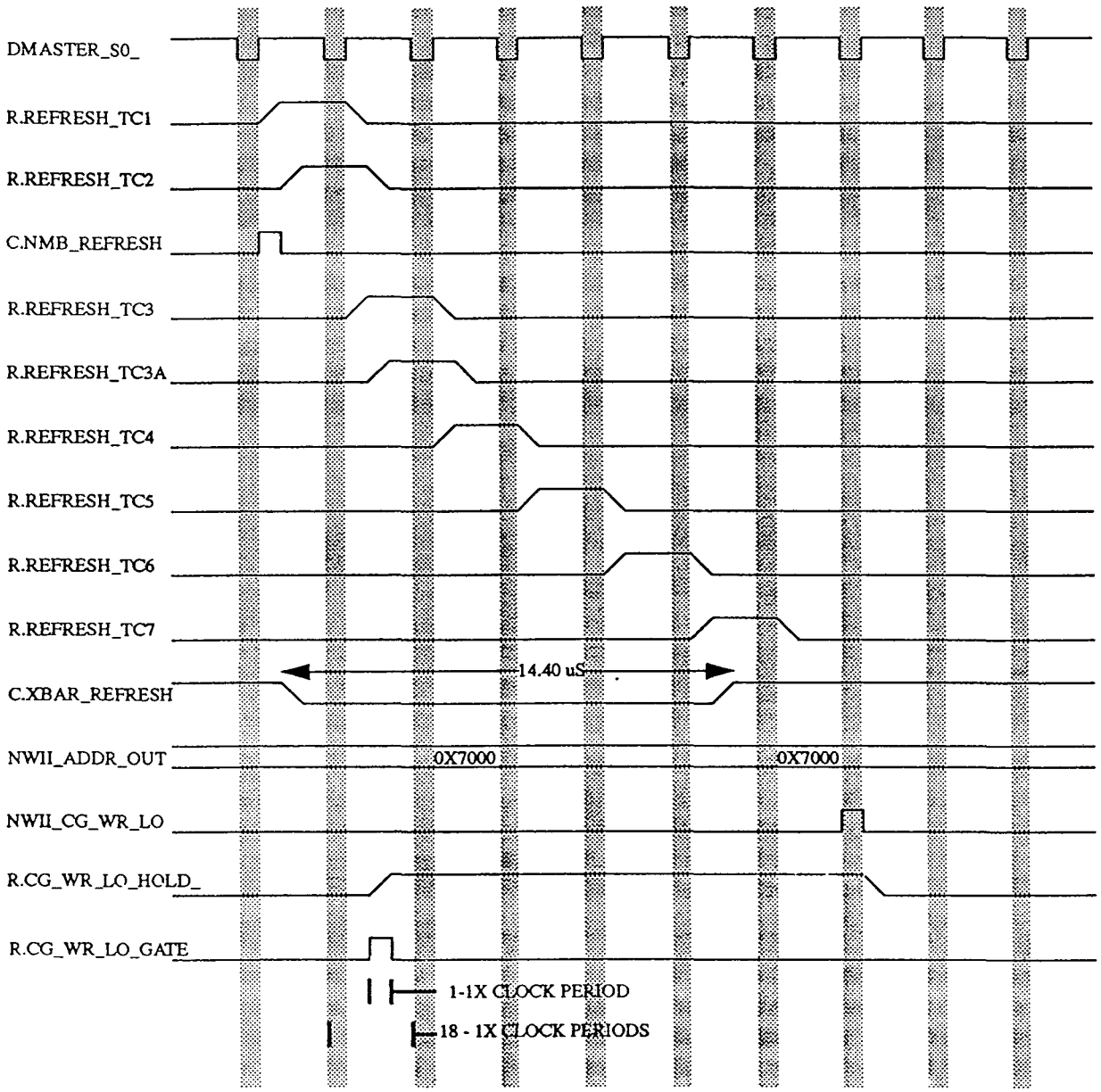
3.5.3.5 Memory Refresh Logic

The memory refresh logic contained within the Scan Engine is responsible for generating the control signals that kick off refresh cycles on the memory board. One signal is sent to the memory boards and the other signal is sent to the crossbar boards. The memory board uses its sys_run signal to determine whether to use the NCU's or the Crossbar's refresh signal as the kick off signal for a memory refresh cycle. When the sys_run bit is set, the memory will use the crossbar refresh signal as the memory refresh kickoff. In either case, however, the memory board's clocks must be running in order for the refresh to occur.

In order to keep all clocks in the system in system synchronized with memory refresh logic (for run after refresh), the refresh logic uses the CCU clock phase signal, master_s0, to control the decrementing of the refresh counter. The memory refresh counter is implemented with a mod 48 binary counter (100e016) clocked at the 1X rate. Each time the master_s0 is set (once every 18 1x clock cycles) the counter will increment by one. When the counter reaches the terminal count value, the terminal count will be set. This starts off the refresh pulse shaping logic and reloads the counter. Therefore, the pulse shaping logic is kicked off every (48 * 18 * 1x period) seconds. At a nominal clock rate of 16.667 nS, a refresh pulse will be sent out every 14.40 uS. The counter logic for the memory refresh logic can be found on page 38 of the Scan Engine schematics.

The setting of the refresh counter terminal count signal kicks off the pulse shaping logic that creates the actual refresh signals for the XBAR and NMB. The NMB receives a refresh pulse once every 864 1x clock periods. The signal is set for a single 1x clock period and reset for the remaining 863 periods. The crossbar refresh signal has timing requirements that define that it must be low for 90 clocks followed by being high for 774 clocks. The XBAR refresh signal is reset the cycle after the NMB refresh is set. The pulse is created by cycling the refresh counter terminal count through a shift register that is shifted once every 20 clocks. DMASTER_S0 signal is used as the shift enable. As the terminal count reaches the sixth stage of the shift register the XBAR refresh signal is set and will remain set until the next time that the refresh counter rolls over. Timing for the pulse shaping logic is illustrated in figure 3-57.

Figure 3-57: Memory Refresh Logic Timing

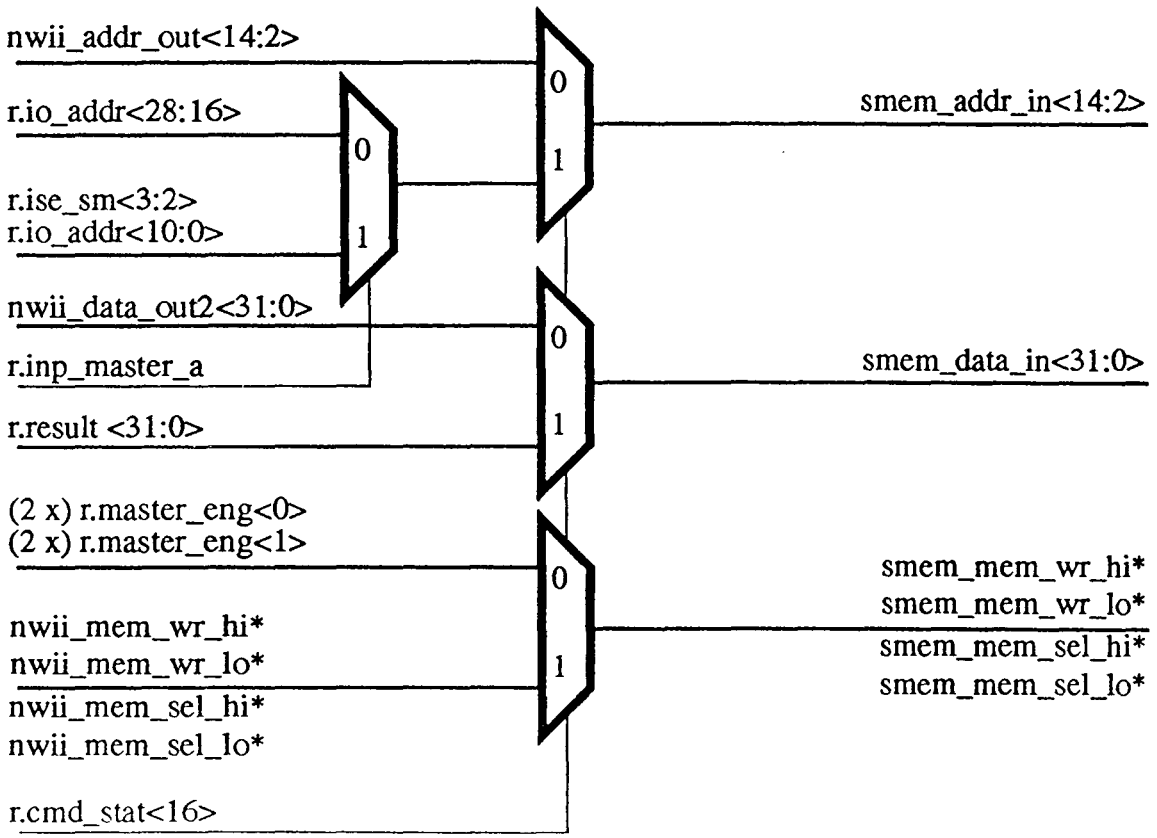


3.5.3.6 Scan Memory Interface

The Scan Memory Interface connects the NWI Interface and the Scan Engine to the Scan Memory. The Scan Memory Interface is a passive logic circuit under the control of the Busy bit (r.cmd_stat<16>) in the Scan Engine Control Register. The Interface is a 2:1 multiplexor that switches the control of the Scan Memory address, data, and control buses between the two potential masters. Ordinarily, the busy bit is reset and the SPU will be the master of the Scan Memory. During a scan operation, the busy bit is set and the Scan Engine can access the memory. If the spu attempts to access the Scan Memory while the Scan Engine is master, the cycle will terminate normally (even though the access never reached the memory) but the Scan Memory Access Error signal will be set.

The Scan Memory Interface is implemented with 3 nS pals and eclips logic. The three to one address multiplexor is implemented with four pals and can be found on page 25 of the Scan Engine schematics. The data and control signal muxes are created with 100e155s and 100e158s. The Data mux can be found on page 26 of the schematics, and the control mux can be found on page 27. The block diagram for the interface is illustrated below, in figure 3-58.

Figure 3-58: Scan Memory Interface Block Diagram



3.5.3.6.1 Scan Memory Access Error Logic

When the Scan Engine is in control of the memory interface, SPU accesses will continue to be acknowledged even though the address, data, and control is gated off at the Scan Memory Interface multiplexor. This occurs because the SPU Interface logic has no interaction with the Scan Memory Interface logic. The Scan Memory Access Error logic generates an interrupt to the SPU which indicates that the last Scan Memory access was ignored. This feature is most usefull during the earliest stages of software debug when Scan Memory accesses are attempted even though the Scan Engine is still busy. During normal system operation, the R.SMEM_ACC_ERR signal should not be set unless there is a problem with the hardware.

The SMEM_ACC logic monitors the state of the Scan Engine's busy bit (R.CMD_STAT<16>) and the SPU Interface's Scan Memory Write enables, using this status to generate the interrupt when an illegal access occurs. An illegal access is defined as the SPU write enables becoming active while the busy bit is set. When this condition occurs, the C.SMEM_ACC_ERR signal will become set and remain set until a data independent writo to the Scan Command Status Register occurs. The C.SMEM_ACC_ERR signal is generated from pal 181-005474 which can be found on page 29 of the Scan Engine schematics.

3.5.3.7 Scan Memory

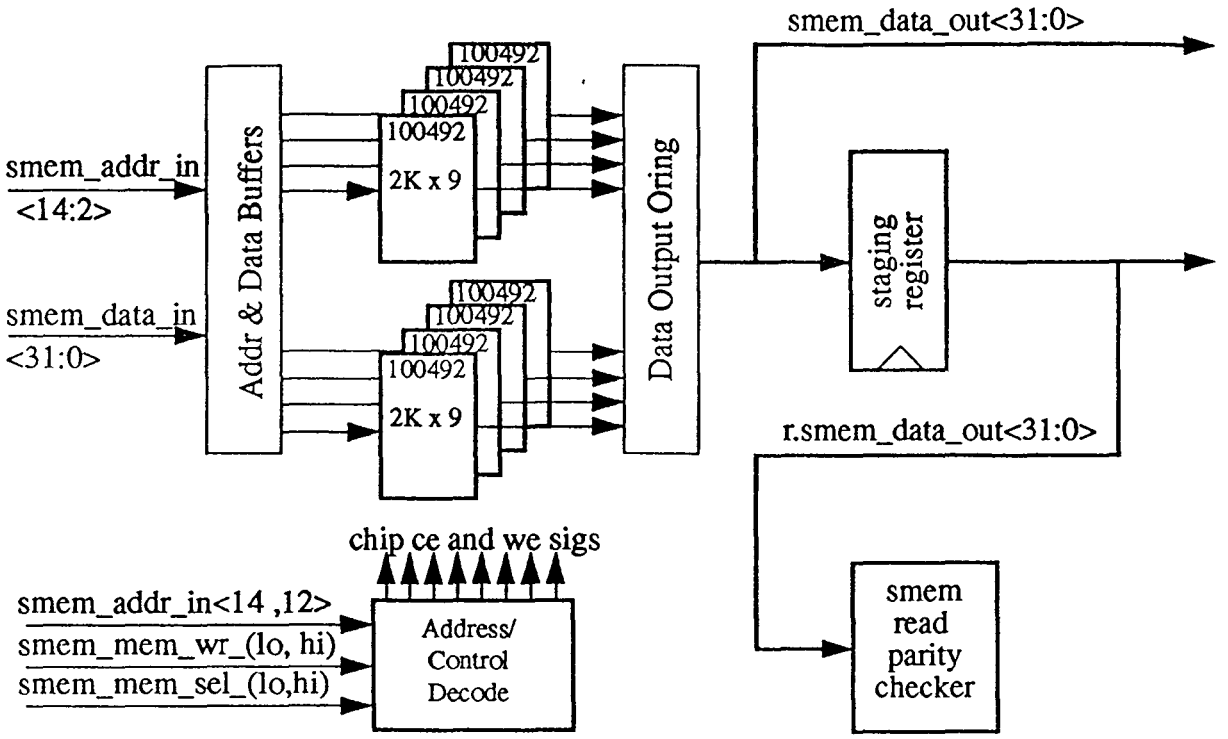
3.5.3.7.1 Memory Array

The Scan Memory is organized as a 4K by 36 bit memory array. Eight Self-timed rams (100492, 2K x 9 bits), formed into two banks of memory, create the array. Each bank is organized as 2K x 36 bits. The 36 bits are divided into four groups, with nine bits assigned to each of the chips that make up the bank. Of the nine bits, the lower eight bits contains memory data and the upper bit contains the parity for the the nine bits.

Two data pages are assigned to each bank (see sections 3.3.3.3) with Outgoing and Mask data being stored in the lower bank and Compare and Result data being stored in the upper bank. The lower bank of the memory array can be found on page 1 of the Scan Memory schematics. The upper bank is found on page 2. The two chips that form a word (16 bits) within a bank receive the same control signals (we_, ce_) from the Address/Control decode pal. Data can only be written and read via 16 and 32 bit accesses.

Address and Input Data for the memory array come from the output of the Scan Engine's Scan Memory Interface logic. To keep the loading on the signals at a reasonable level, the data and address are buffered before reaching the memory chips. Data output from the two banks of memory is logically ORed together (not wire-ORed). The output of the OR logic goes to the Scan Engine and a staging register. The output of the staging register is driven to the NWI Interface and is checked by the Scan Read Parity logic. The dataflow block diagram for the Scan Memory is illustrated in Figure 3-59.

Figure 3-59: Scan Memory Dataflow Block Diagram



3.5.3.7.2 Address/Control Decoding

The Address/Control Decoding receives address and memory control signal information from the Scan Engine's Scan Memory Interface, and converts this into chip and write enable signals for the memory chips contained in both banks of the memory array. Two pals, both with 181-005499 programs, decode the address to determine which bank of Scan Memory the access will go to. One pal decodes the memory chip enables, and the other does the write enables.

SMEM_ADDR_IN bits 14 and 12 are decoded to determine which memory bank is accessed. If both bits are reset, the control signals at the input of the pal will be transferred to the lower bank of the Scan Memory. If bit 12 is set while bit 14 is reset, the control signals will be transferred to the upper bank of memory. All other combinations of address bits will result in the write enable outputs being reset, and the chip enables being directed to the upper bank of memory. This occurs because a read cycle to one of two memory banks must occur each clock cycle (regardless of the address) in order for the read parity checker to work.The Address/Control Decoding logic can be found on page 7 of the Scan Memory schematics.

3.5.3.7.3 Parity Checking

Scan Memory parity is checked for the address and data input to the memory and the data output from the memory. Data and address parity for the memory inputs is checked by parity checkers contained within the Self Timed Ram chips. Both the address and the byte of input data are checked on write accesses to the ram, but only the address is checked during read cycles. The two banks of chips parity errors are ORed together as bytes. The four byte errors are driven to the parity error logger, where they are used for state storage and ORed together to form the R.SMEM_PARITY_ERR signal.

The parity on the Scan Memory output data is checked at the output of the staging register. Since the memory chip enables are permanently set at the Scan Engine's Scan Memory Interface, memory read accesses occur each cycle, so output parity is checked on every clock. The Scan Memory data output parity checking logic can be found on page 13 of the Scan Memory schematics.

3.5.3.7.4 Parity Error Logging

The parity error logger is responsible for holding Scan Memory state when a parity error is detected. This state is used by software to isolate the source of the parity error. The organization of this register is described in section 3.4.2.2.18. The error logging register loads state data on every clock until an error is detected. When an parity error is detected, the registers load enable is reset, holding the state of the register. The load signal is reset by writing (data independent) to the Scan Memory Error Log register.

Due to the limited amount of bits available for parity error state recovery, only eight of the thirty two data bits are captured when the Error Log State is held. A priority encoder is implemented in the scan memory error pal (181-005470) to create the select lines for the muxes that select which byte is loaded into the Error Log. If any single byte is in error, that byte will be loaded into the register. However, if more that one byte is in error, the most significant byte in error is loaded into the register. Bits 31-24 are the most significant byte of the 32 bit word. The Parity Error Log logic can be found on page 11 of the Scan Memory Schematics.

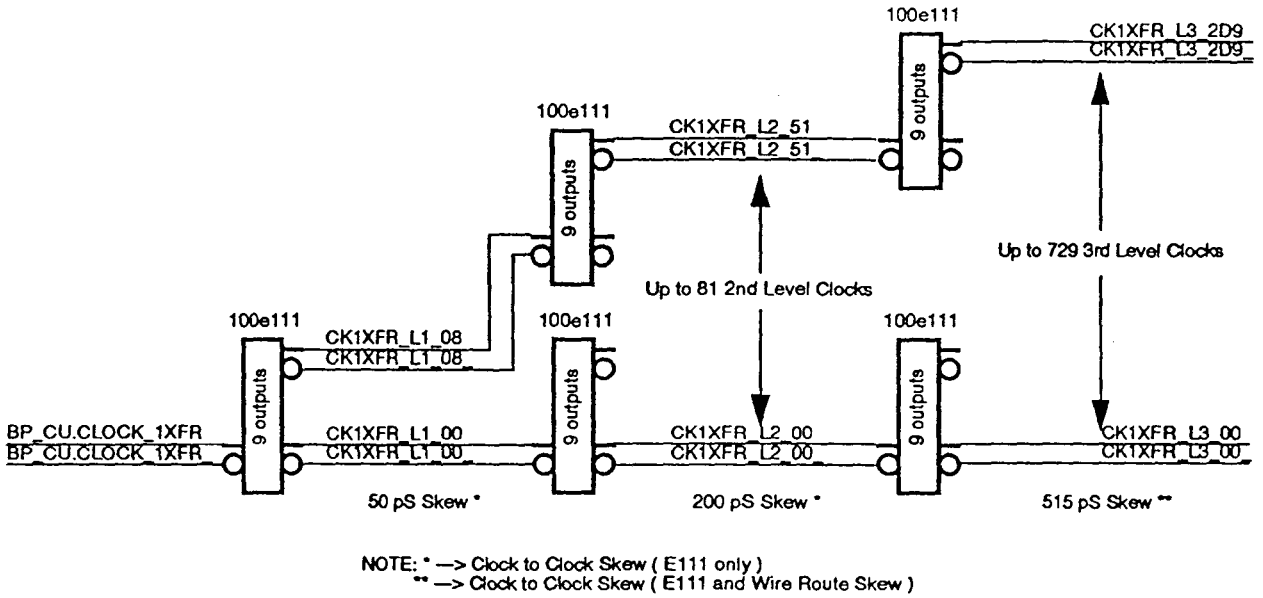
3.5.4 Clock and Scan Distribution

3.5.4.1 Basic Clock Distribution

All of the SECG logic is clocked from a free running 1x clock, BP_CU.CLOCK_1XFR (*), which is received on pins 1_340 and 1_440 of the Augat connector. The SECG portion of the NCU uses 100e111s in order to distribute its clocks. These devices have one differential input and nine differential outputs. These devices are guaranteed to have low skew between any two output pins of the same device, and across any two parts. At the time of this document, the pin to pin skew on the e111 is specified as 50 pS and the device to device skew is 150 pS. Although the Eclips spec may not indicate these numbers, parts are screened in order to meet the aforementioned skew numbers.

Because of the number of descrete devices in the SECG subsystem that require clocks, Three levels of clock distribution are required to create the all the necessary clocks. The first level of clock distribution consists of a single 100e111 which creates the nine differential clocks which drive the second level clock buffers. The third level clock buffers are driven by the 81 differential clock pairs from the second level of clock buffers. This many clock buffers are required because on of the design constraints was that all networks in the clock distribution tree were to be point to point connections. Figure 3-60 illustrates the clock distribution scheme of the SECG subsystem. Note: this is the same clock distribution scheme used on all Neptune daughter and Crossbar boards.

Figure 3-60: SECG Clock Distribution Tree



Controlled skew clock buffers and fixed length wire delays are used to control the clock skew across the board. Because the pin to pin delay of the first level clock distribution and the device to device delay of the second and third level of clock distribution, there is 350 pS of skew contributed by the clock buffers. This is illustrated in Figure 3-51. The wire delay from the Augat connector pads to the clock pin on any device in the sub-system is fixed at 10.00 nS - (3 X (e111 delay)). A nominal delay of 505 pS is used as the e111 delay for the NCU design. A program called clocktune is used to modify the schlen.dat file so that the first, second and third level clock routes add up to this fixed delay. Assuming that up to an inch of variance across the four from-tos that make up the routed clock path is produced by the "fixed" length router (165 pS/in delay), a total of 515 pS of clock skew exists between any two registered devices on the board.

The first and second levels of the clock distribution tree can be found on page 41 of the Scan Engine schematics. Third level of the clock distribution tree can be found on pages 42 and 43 of the Scan Engine schematics and page 12 of the Scan Memory schematics.

3.5.4.2 Scan Enable Distribution

Genrad and Scan based testing are the primary methods used to debug the NCU. However, unlike other boards and logic (CU section of board), scan based testing in a running system is not possible for the SECG subsystem because the Scan Engine is the hardware vehicle used to transfer data. Turning the Scan Engine into a shift register would make scan impossible. However, in the CAST system, a known good NCU can scan a second NCU that is installed in a BUT slot.

Three scan control signals and a board scan enable are decoded in order to determine whether the board will scan. The three scan controls are sourced from the XCL board and are the logical ANDing of the master scan controls and the NCU scan enable. The CAST_SCTL_ENA signal is a signal added between the designs of the KNCU and NCU, which prevents the NCU from accidentally scanning itself in a system (a big problem during initial debug). The enable is logically ANDed with the scan enable for the SECG logic in the scan decode pal, 181-005442. Unless this enable bit is set, a scan of the SECG logic is not possible. The enable is pulled down on the board and is not connected on the IOBP. On the CAST system the connector pin is connected to a ECL 100K driver that will set the pin during scan operations.

When the Board Left (111) or Last Board Left (101) scan codes are presented to the board while the CAST_SCTL_ENA signal is set, the scan decode pal will set the SECG_SCANGO signal. This signal in conjunction with the SECG_STR_SCAN signal, allow the entirety of the SECG logic to scan. The SECG_STR_SCAN is the Scan Memory self timed ram scan enable. This signal will be set during Board Left mode, but is reset during Last Board Left mode. While the board is in Last Board Left mode, a single clock will be issued to the board which causes the ring to shift one final time while the self timed rams in the Scan Memory execute the command contained in their scan rings. During all other scan modes, these scan enables will be reset regardless of the state of the CAST_SCTL_ENA signal.

Because of the large number of descrete parts contained in the SECG portion of the scan ring, 100e122s are used to fanout the SECG_SCANGO signal. There are 22 buffered versions of this signal which are labelled, SCANGO_B**. The loading on these signals is heavier than any other signals on the board. Some of these nets have as many as 12 loads. A number of these signals will fail single clock cycle timing. However, since the enables will not be used for several clock cycles after they are changed, these violations can be regarded as bogus.

The SCANGO_ B** signals are generated at the top level of the NCU schematics and are distributed through the schematics using the global signal attribute. The scan decode pal and the buffers which fan out the scan control signals are contained on page 10 of the NCU schematics.

4 CPU Utilities Implementation

4.1 CPU Utilities Overview

The CPU Utilities are functionally separate from the Scan Engine, Clock Generator and Workstation Interface although it shares a common assembly. It deals with functions that serve the CPUs as a group, coordinating activities during normal operation of the system. This section of logic will be referred to as the CU.

4.1.1 C2 versus C3800

The following subsections define the major differences between the C2 CPU Utilities subsystem and the Neptune CPU Utilities subsystem.

4.1.1.1 Referenced and Modified changes

The reference and modified bits for memory management, located in CPU I/O space implemented in the CPU utilitiy subsystem for C2 systems, have been moved to a dedicated area in main memory for Neptune. The multiported (i.e. simultaneously monitor all memory accesses) implementation used in C2 has been replaced by a scheme in which a possible change in the reference or modified status causes the CPU hardware to perform a transparent (to the program) memory cycle to properly maintain the bits in memory. The reference and modified bits corresponding to the pages currently accelerated in a given CPUs PTE cache are now accelerated and used as an indication that the reference and/or modified bit in memory has already been set to prevent further micro-interrupts for subsequent accesses to the same page. Two new supervisor instructions have been added to support software management of reference and modified cache coherency. The 'PREF' instruction purges the referenced bits in all PTE caches for the entire complex. The 'PMOD' instruction purges the modified bits in all PTE caches for the entire complex (all installed CPUs). The main memory copy of reference and modified bits is not modified by 'PREF' or 'PMOD'. The main memory image consists of a two megabyte block of physically contiguous memory allocated by the SPU on a two megabyte boundary (address bits 20..0 must be zero) at the time the SPU builds the PCM. The SPU scan initializes a hardware base address register in each CPU and passes the same base address to JP-Unix via whatever software convention is preferred by the OS group. Reference and modified bits are allocated one bit per byte (bit position 0) in an interleaved fashion with modified bits occupying even byte locations and reference bits occupying odd byte locations. The CPUs access the memory area at location Scanned_base_address<31..21>::Operand_physical_address<31..12>::'R' where 'R' is '0' for modified and '1' for referenced. Interleaving the bits in this manner allows a CPU to set both bits for a page in a single memory access.

4.1.1.2 Physical Configuration Memory

The PCM, located in CPU I/O space on C2, is an "existance" map used by the CPU hardware to detect illegal physical addresses. The C2 PCM was not accessed by software other than the test which verified that it could be accessed! The SPU software, and the diag calls which allow ConvexOS to access the C2 PCM, all access a copy in SPU memory or on the SPU disk which is initialized at mminit time.

PCM is supported on Neptune by the SPU. The SPU checks the memory configuration via scan at mminit time and keeps a copy of the configuration in SPU memory. The crossbar, CPUs, NIA, and memory boards are initialized by the SPU via scan to work correctly with the memory configuration. PCM as a software readable entity in CPU I/O space, does not exist on Neptune.

4.1.1.3 Time of Century and Interval Timers

After removing R&M and PCM from CPU I/O space, the TOC and interval timer are the only remaining CPU I/O space features. The TOC is the only feature which is accessible (read only) by a user program. This read access is already supported by an instruction due to implementation problems on C2. This means we would have been implementing a fair amount of hardware and a separate protocol to support an I/O address space which contains 16-bytes of counters which are accessible only by the operating system! Instead, these counters have been removed from I/O space and access has been provided by new privileged instructions which access the CU X-space (control registers).

4.1.1.4 Communication Registers Addition of Accounting Timers

Since Neptune supports eight CPUs with sub-complex partitioning where subsets of the installed processors can work on specific processes, the number of communication register sets (CIRs) has been increased from eight to thirty-two. Each set of hardware communication registers (index 0-1F) now contains eight sets of CPU accounting timers instead of the four for C2. The required accounting timers are added in place of eight of the ten hardware reserved locations in the C2 map.

4.1.1.5 Communication Register Access

The C2 system implemented a shared bus structure to allow all four processors to access the common set of communication registers. Shared busses are not a viable technology for the Neptune system. Instead, the communication registers are accessed via the main memory crossbar switch and appear as a form of fixed latency memory board with slightly less latency than the normal memory boards. The crossbar performs all the necessary arbitration and data path switching required for multi-ported access.

In order to minimize hardware impact on the crossbar switch, all CU functions which are initiated from the crossbar pipe at a one clock rate and all return data supplied to the crossbar is returned by the CU at a fixed latency known to the crossbar controllers.

4.1.1.6 Traps, Interrupts, and ASAP

The CU has assumed a new set of functions, previously relegated to CPU microcode, in order to minimize the impact of increased communication register latency.

C2 implemented firmware micro-traps and process deadlock traps via distributed hardware, shared backplane lines, and microcode. These functions are now controlled by the CU subsystem. The CU continuously monitors deadlock status from all CPUs and issues traps to the CPUs of a CIR which is deadlocked. A CPU requests firmware traps and interrupts via the same crossbar command interface used for communication register access and memory access. The CU then coordinates the trap or interrupt sequence by issuing traps to the necessary CPUs and/or I/O devices and by monitoring trap completion.

The CU maintains all the registers associated with the CPU interrupt system, and issues traps to CPUs and I/O devices to perform interrupt services. The interrupt functionality has been expanded to include a broadcast interrupt enable register per interrupt channel (versus one mask for C2) to facilitate sub-complex partitioning of the interrupt system.

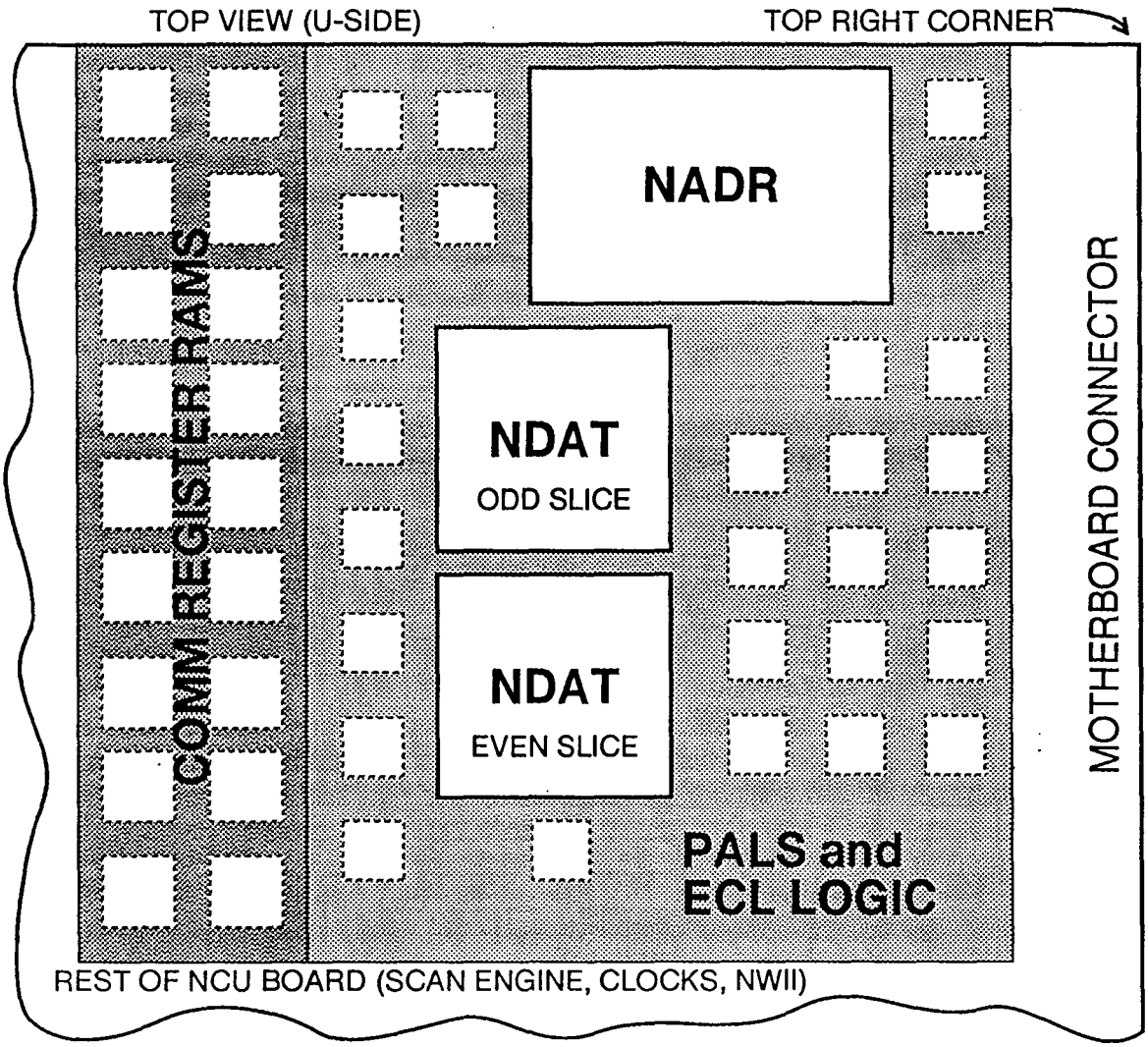
Process deadlock traps, firmware microtraps, and interrupt traps. are all issued to one or more CPUs via a single interface protocol. The CU prioritizes and interlocks these traps to guarantee that any given CPU has no more than one trap pending.

The CU also performs the processing of the communication registers for posted Forks or Spawns. An "idle" CPU in the C2 system was actually executing a microcoded "idle loop", polling communication registers and looking for a posted Fork or Spawn to service. This polling operation would be very time consuming on the C3800 due to the increased communication register latency and the increased number of CPUs vieing for the communication registers. Instead, the "polling" is done in hardware by the CU each time a CPU requests a posted Fork or Spawn to service. The ASAP mechanism has also been expanded by the addition of an 8-bit "CPU Mask" to each CIR. This mask governs which CPUs can participate in a given process to facilitate sub-complex partitioning.

4.1.2 CU Hardware

In addition to an assortment of SSI/MSI logic and PALs, the CU contains two NDAT gate arrays, one NADR, and a bank of 18 self-timed RAMs (STRAMs). The NDAT is implemented in a Vitesse VSC15K GaAs gate array. The NADR is implemented in a Vitesse VSC30K GaAs gate array. The STRAMs are 2Kx9 bits each and contain the CMRs with lock bits and parity. The NDATs serve as the main data path and possess the interrupt arbitration logic. The trap logic is distributed between board level logic and the NADR. The NADR also handles the address paths, the ASAP function, and the lock bit data paths. The remaining logic implements instruction decode, implements the trap state machine, and provides some registers for pipelining and interfacing to the crossbar. The board level logic also provides some simple glue logic and buffering.

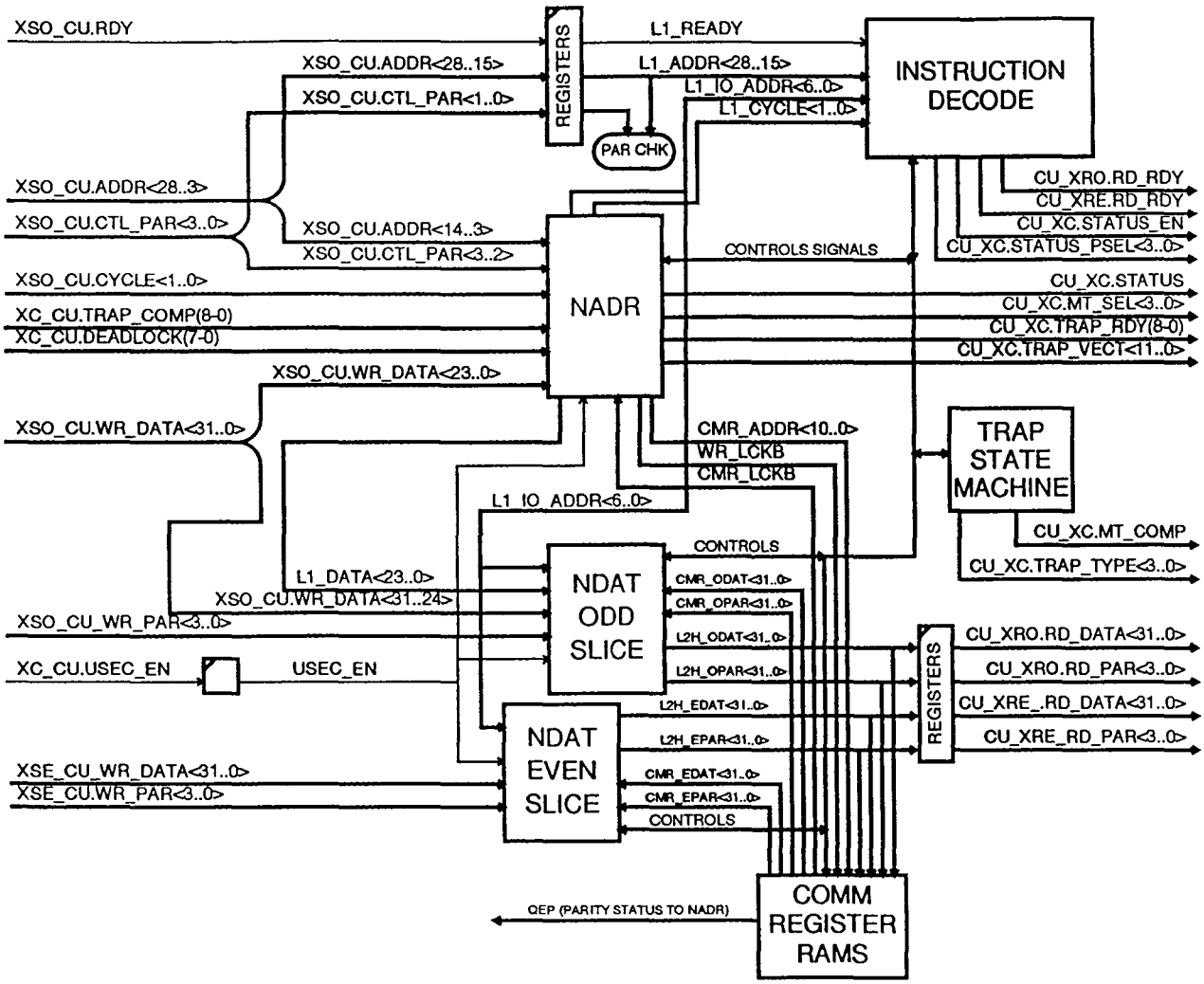
Figure 4-1: CPU Utilities Hardware Layout.



4.1.3 CU Functions

This section summarizes the CU functions. Each area is described in greater detail later in this chapter. Figure 4-2 is a high level block diagram of the CU. It shows all of the crossbar interfaces except scan related interfaces.

Figure 4-2: CPU Utilities Block Diagram.



4.1.3.1 Communication Registers

The CU contains all of the interprocess communication registers. There are 128 sixty-four bit registers for each process CIR. With a total of 32 CIRs, there are 4096 communication registers (CMRs). Some of the CMRs under each CIR are "hardware" CMRs and have dedicated hardware functions. The rest are defined by the software. Refer to the section on Communication registers for an explanation of these functions and an address map of the CMRs.

Each of the CMRs has an associated lock bit. The lock bits are semaphores which allow the processors to know the status of the CMR data. This keeps the data from getting corrupted. The interaction of the lock bits, CMR data, and the returned status is dependent on the microinstruction being used by the processor.

4.1.3.2 ASAP Accelerator

The 'polling' of the communication registers involved in ASAP is handled totally in hardware within the NADR gate array. Specific information in the CMRs are used for ASAP, and this information is shadowed in the NADR by copying the relevant data when those CMRs are changed. This in necessary because of the sequential access nature of the CMR RAMs. With all the ASAP relevant data available at once, the NADR calculates an available CIR thread on-the-fly when the request for a posted fork arrives. The request for a posted fork is a simple register read by the processor. The CPU mask fields in the CMRs are accounted for in the ASAP calculation to allow specific CPUs to be masked from certain CIRs.

4.1.3.3 Interrupts and Traps

The control for system-wide interrupts and microtraps is also contained on the CU. The registers for manipulating the function of the traps and interrupts are in an address space separate from the CMRs referred to as "X space". Other functions in the CU also have X registers associated with them. The X registers used to control traps are introduced in section 4.4 and further illustrated in the text concerning traps.

4.1.3.4 TOC Counter

Another X register in the CU is the Time of Century (TOC) counter. This is initialized by the system and increments every microsecond. With 64 bits of accuracy, this counter can mark more than a century worth of microseconds. [1]

4.1.3.5 Interval Timer Counter

For use by the operating system, the CU provides an interval timer. This timer can be programmed to trigger an interrupt to one of the system-wide interrupt channels. It counts in ten microsecond increments. Once it has gone off, the timer reloads to time a new interval. There is more detail about this in the section below on traps and interrupts.

4.1.4 Pipeline Stages

To the crossbar, the CU looks like a memory with a fixed latency of three 1x clock cycles. On the CU, the first clock is used for address and instruction decode, while the second is used for reading data out of CU registers. This third and final stage either writes data to the CMR or X registers, or returns data to the crossbar. The CU can accept a request every cycle.

The one exception to the three clock pipeline is the INC_C instruction which takes four cycles to complete. The old data in the CMR will return on the third cycle to the crossbar just like a normal read, but the incremented value is written to the communication register STRAMs on the fourth cycle. The crossbar guarantees that there will be one dead cycle after an INC_C so that the following instruction will not get corrupted by the read-modify-write operation of INC_C.

---

1. A century of microseconds can be counted in 35 bits. 64 Bits allows the TOC to time over 50 million years worth of microseconds.

4.2 Crossbar Interface

The crossbar interface provides all communication between the CU, the NIA and the CPUs. The interface, detailed in Table 4-1, is divided into three separate sections: the even word data path, the odd word data path, and the control path. The odd and even data transfers for each access to the CU are synchronized. For a longword write, both data words will arrive at the same time, while a read will deliver both simultaneously. The even address field is not used, and the odd address field contains subfields interpreted as commands

Table 4-1: Crossbar Interface

Signal Name	Description
XSE_CU.WR_DATA<31..0>	Even crossbar send data
XSE_CU.WR_PAR<3..0>	Even crossbar send parity
CU_XRE.RD_DATA<31..0>	Even crossbar return data
CU_XRE.RD_PAR<3..0>	Even crossbar return parity
CU_XRE.RD_RDY	Even crossbar return data ready
XSO_CU.WR_DATA<31..0>	Odd crossbar send data
XSO_CU.WR_PAR<3..0>	Odd crossbar send parity
CU_XRO.RD_DATA<31..0>	Odd crossbar return data
CU_XRO.RD_PAR<31..0>	Odd crossbar return parity
CU_XRO.RD_RDY	Odd crossbar return data ready
XSO_CU.ADDR<28..3>	Odd crossbar send address
XSO_CU.CYCLE<1..0>	Odd crossbar send cycle
XSO_CU.CTL_PAR<3..0>	Odd crossbar control parity
XSO_CU.RDY	Odd crossbar send data ready
CU_XC.STATUS	Return status data to CPUs
CU_XC.STATUS_EN	Return status available to CPUs
CU_XC.STATUS_PSEL<3..0>	Selects which board receives status
CU_XC.MT_SEL<3..0>	Selects which board receives MT_COMP (same as originator)
CU_XC.MT_COMP	Microtrap complete
CU_XC.TRAP_RDY(8-0)	Traps to ports
CU_XC.TRAP_TYPE<3..0>	Trap type (see section on traps)
CU_XC.TRAP_VECT<11..0>	Trap Vectors (see section on traps)
XC_CU.TRAP_COMP(8-0)	Trap completes from up to 9 boards
XC_CU.USEC_EN	Microsecond enable
XC_CU.DEADLOCK(7-0)	Deadlock states from 8 CPUs

The even and odd word data paths function identically to the main memory module interfaces. The CU appears as a fixed latency, non-blocking memory board with a two bit cycle type, a 26-bit address, 32 bits of read and write data, control, and parity. The possible cycle types are show in Table 4-2. The CU differs from a normal memory interface in four important ways.

Table 4-2: Cycle Types

Cycle<1..0>	Action
00	Status
01	Read
10	Write
11	Read-Modify-Write

First, for normal operation, the CU is fully pipelined and non-blocking, such that there is no associated "bank busy" handshake or model. New requests are accepted every clock and return data is provided two clocks following the request.

The second difference, which is a caveat to the first, is that the CU requires a read-modify-write (RMW) cycle to be followed by an empty cycle. The crossbar will automatically insert an "CU busy" after an RMW access. The data and status will still return after two more clock cycles, just like the other read accesses.

The third difference is that all CU operations are "longword" in that the even and odd crossbar datapaths are interlocked. This is by protocol and not by any hardware handshake. This guarantees that the even and odd requests will arrive at the CU on the same clock. The processor issuing the requests waits until "REQ_PEND" goes away to indicate that both sides are clear. The crossbar is designed so that requests which are sent to the queue in lock-step will be dispatched to their destination in lock-step. Data return for the two words to the crossbar from the CU will also occur in lock step. It is possible to make a short word access to the CU, but only in terms of reading and writing the low (odd) word while the even data is ignored.

The fourth difference is that the address field is not simply an address -- but is broken into subfields with different functions. Because the processor's address generation logic may modify the even address field, only the odd address field is used for both even and odd word data streams. Zone bit have no meaning to the CU and are not sent to the CU. The address fields relevant to the CU are noted inTable 4-3 The parity bit assignments are shown in Table 4-4. .

Table 4-3: CU Odd Address Fields

Address Bits	Field Name	Field Description
Addr<28..26>	CMD<2..0>	CU command
Addr<25..22>	PID<3..0>	Port IDs 0 thru 7, CPU/NIA. Port 8, NIA only.
Addr<21..17>	TID<4..0>	Thread ID
Addr<16>	SIZE	0 = Word, 1 = Longword
Addr<15>	SPARE	Unused
Addr<14..10>	CIR<4..0>	Reference_CIR<4..0> = Comm address<11..7>
Addr<9..3>	ADDR<6..0>	Comm/Ctl address <6..0>
Cycle<1..0>	CYCLE<1..0>	Memory Cycle Type

Table 4-4: Parity Bit Assignment For Odd Address

Parity Bit	Address Fields
Bit 0	Addr<28..22>
Bit 1	Addr<21..15>
Bit 2	Addr<14..8>
Bit 3	Addr<7..3> concatenated with Cycle<1..0>

4.3 PROCESSOR INTERFACE

The CU has a set of signals for communicating status and trap information to the CPUs. There are no direct signals between the CU and CPUs; these signals all interface to the XCL board and are registered, routed, and passed on by the XCL. The crossbar interfaces that correspond to the processor interface are shown back in Table 4-1. You may want to refer to the XCL specification for more details.

Some of the accesses performed by the CPUs require that status be returned. This accomplished with CU_XC.STATUS, CU_XC.STATUS_EN and CU_XC.STATUS_PSEL<3..0>. If status is expected, the STATUS_EN will asserted. In this case, STATUS carries the resulting status and STATUS_PSEL<3..0> determines which port the XCL should send the STATUS_EN to.

The CU receives a microsecond enable. This signal goes high for one 1x clock cycle each microsecond. It is originally generated by the clock circuitry on the NCU, but is sent to the XCL for distribution. From there it is sent to the CU logic via XC_CU.USEC_EN, and it is also sent to the scalar processors.

Traps are sent out to 9 possible ports. CPUs or NIAs can occupy ports 0 thru 7, and an NIA is always at port 8. To send out a trap, the CU sets the CU_XC.TRAP_RDY(8-0) signals high that correspond to the ports that the trap should go to. CU_XC.TRAP_TYPE<3..0> identifies the type of trap, and CU_XC.TRAP_VECT<11..0> is the vector for that trap. The ports acknowledge the trap via the XC_CU.TRAP_COMP(8-0) lines. In the case of CTRSG traps, an CU.XC.MT_COMP flag is sent to the processor port identified by CU_XC.MT_SEL<3..0> after all trapped ports acknowledge the trap. Traps are explained in greater detail in Section 4.5.

The CU receives a deadlock status, XC_CU.DEADLOCKx, from each of the 8 possible processors which indicates that the processor may be in a deadlock condition. If all processors sharing a common CIR indicate a possible deadlock, the CU counts how many cycles this condition remains active. If the processors are deadlocked for a certain number of 1X clock cycles, a deadlock trap will be issued. This time-out is determined by an 11 bit scan programmable register.

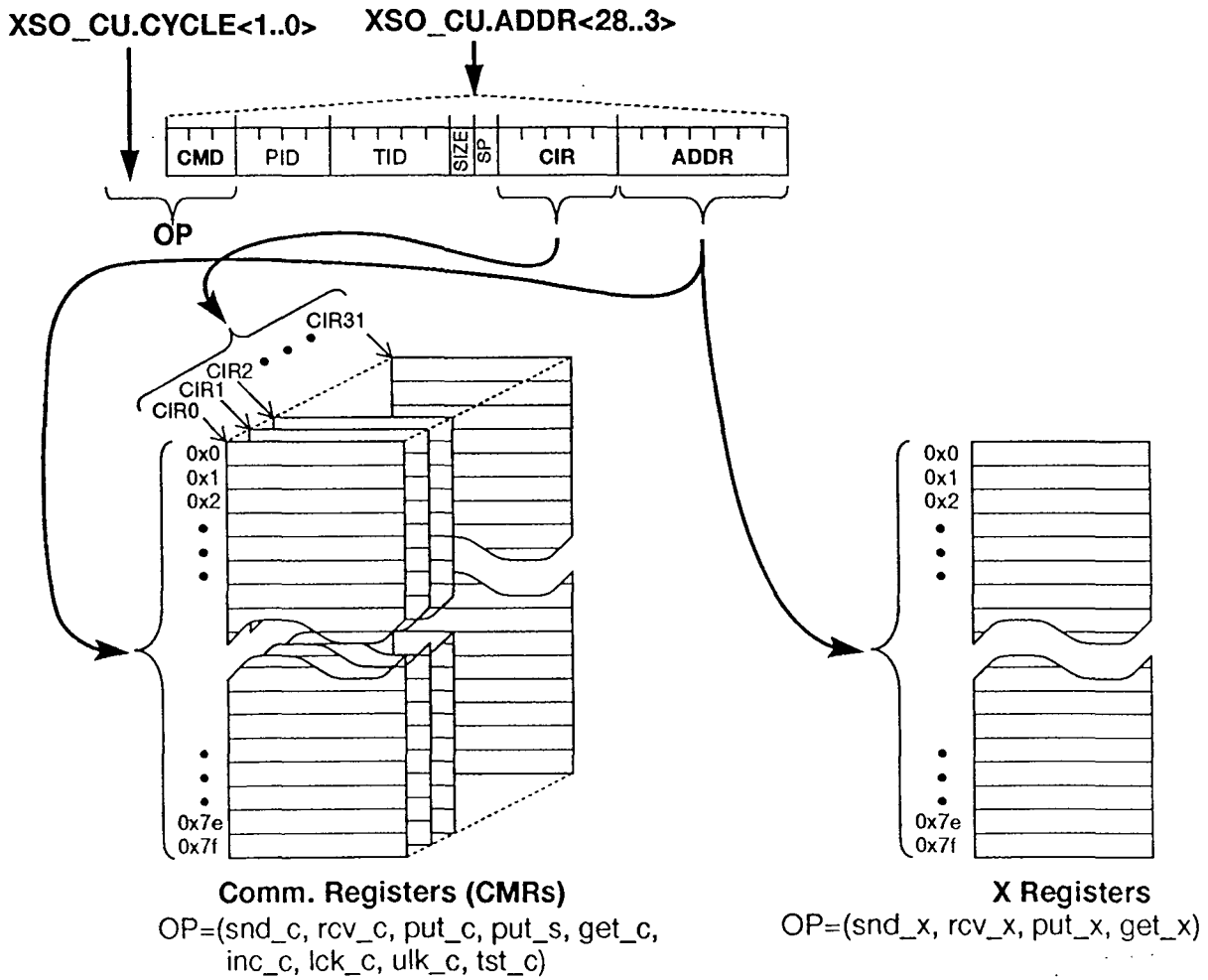
4.4 MICROCODE VIEW OF THE CU

The majority of the CU functions are accessed directly or indirectly by the microcode executing on the scalar processors. To the processor, the CU looks like two sets of registers, the communication registers (CMRs) and the X registers. This section covers the details of the registers and how they are accessed.

4.4.1 Microcode Command Interface

The CU is sent microcode commands which are a combination of a three bit command field, XSO_CU.ADDR<28..26>, and the two bit cycle field, XSO_CU.CYCLE<1..0>. These functions act upon CMRs, X registers and their lock bits. The crossbar performs the arbitration between the processors so that the CU gets the commands one at a time. Figure 4-3 shows how the address fields relate to the register maps. Access to CMRs or X registers is determined by which operation is requested by the CMD and CYCLE bits. The odd and even data for write functions arrive at the CU at the same time as the odd address. The even address is unused for CU accesses. 32 bit word mode is flagged by a SIZE bit of "0"; 64 bit long word mode is flagged by a SIZE bit of 1. PID is the port id which identifies the originating port. TID is the thread ID, and SP is a spare address bit; these are only used for parity checking of the whole address.

Figure 4-3: Register Addressing



4.4.2 Microcode Operations

Table 4-5 is a summary of the microcode operations. A more detail explaination of each function follows the table. Do not use any of the unused functions; they could cause unpredictable results. Further explanation of the CMRs and X register sets follows this section..

Table 4-5: CU Operation Codes

	CMD/ CYCLE	MNEMONIC	OPERATION
STATUS	000 00	lck_c[n]	{if(L_c[n]=0) {L_c[n]=1;status=1;} else {status=0;};}
	001 00	ulk_c[n]	{if(L_c[n]=1) {L_c[n]=0;status=1;} else {status=0;};}
	010 00	eni	{if(ION=0) {ION=1;status=0;} else {status=1;};}
	011 00	dsi	{if(ION=1) {ION=0;status=1;} else {status=0;};}
	100 00	tst_c[n]	{status=L_c[n];}
	101 00	spare	{unused;}
	110 00	spare	{unused;}
	111 00	spare	{unused;}
READ	000 01	rcv_c[n]	{if(L_c[n]=1) {L_c[n]=0;status=1;} else {status=0;};return C[n];}
	001 01	rcv_x[n]	{if(L_x[n]=1) {L_x[n]=0;status=1;} else {status=0;};return X[n];}
	010 01	spare	{unused;}
	011 01	spare	{unused;}
	100 01	get_c[n]	{return C[n];}
	101 01	get_x[n]	{return X[n];}
	110 01	rdcmr[n]	{X[LCK]<<1;X[LCK]_msb=L_c[n];return C[n];L_c[n]=0}
	111 01	spare	{unused;}
WRITE	000 10	snd_c[n]	{if(L_c[n]=0) {L_c[n]=1;status=1;C[n]=data;} else {status=0};}
	001 10	snd_x[n]	{if(L_x[n]=0) {L_x[n]=1;status=1;X[n]=data;} else {status=0};}
	010 10	spare	{unused;}
	011 10	put_s[n]	{C[n]=data;status=L_c[n];}
	100 10	put_c[n]	{C[n]=data;}
	101 10	put_x[n]	{X[n]=data;}
	110 10	wrcmr[n]	{L_c[n]=X[LCK]_lsb;X[LCK]<<1;C[n]=data;}
	111 10	spare	{unused;}
READ-MODIFY-WRITE	000 11	inc_c[n]	{return C[n]; if(L_c[n]=1) {C[n]+=data;status=1;}else {status==0;};}
	001 11	spare	{unused;}
	010 11	spare	{unused;}
	011 11	spare	{unused;}
	100 11	spare	{unused;}
	101 11	spare	{unused;}
	110 11	spare	{unused;}
	111 11	spare	{unused;}

LCK_C[n] CMD:000 CYCLE:00

This command operates on the CMR address specified in the address field, Addr<14..3>. If the lockbit for the CMR is not set, then this command will set it and return a status of 1. Otherwise, a status of 0 is returned.

ULK_C[n] CMD:001 CYCLE:00

This command operates on the CMR address specified in the address field, Addr<14..3>. If the lockbit for the CMR is set, then this command will reset it and return a status of 1. Otherwise, a status of 0 is returned.

ENI CMD:010 CYCLE:00

If the Interrupt ON (ION) bit is not set, this command will set it and return a status of 0, otherwise it will return a status of 1. The ION bit must be set for processor interrupts to occur.

DSI CMD:011 CYCLE:00

If the Interrupt ON (ION) bit is set, this command will reset it and return a status of 1, otherwise it will return a status of 0. Resetting the ION bit inhibits processor interrupts.

TST_C[n] CMD:100 CYCLE:00

This command returns, as status, the state of the lockbit for the CMR specified in the address field, Addr<14..3>.

RCV_C[n] CMD:000 CYCLE:01

This command operates on the CMR address specified in the address field, Addr<14..3>. If the lockbit for the CMR is set, then this command will reset it and return a status of 1. Otherwise, a status of 0 is returned. Regardless of the state of the lockbit, the value contained in the CMR is returned as well.

RCV_X[n] CMD:001 CYCLE:01

This instruction is similar to the RCV_C instruction except that it is intended for use on the X registers. The X registers use only the address bits Addr<9..3>. The X registers also behave differently in many cases because only a few of them have lockbits and in the case of the PCIR, the lockbit is set under special conditions. This is explained in the section on the specific X registers.

If an X register has a regular lockbit, and it is set, then that bit will be reset and a status of 1 will be returned. If not, a status of 0 will be returned. In there is no lockbit, then the status is not guaranteed to be any value. The value of the X register will be returned properly in any case.

GET_C[n] CMD:100 CYCLE:01

This instruction returns the value contained in the CMR addressed in Addr<14..3>. No status is returned or lockbits modified.

GET_X[n] CMD:101 CYCLE:01

This instruction returns the value contained in the X register addressed in Addr<9..3>. No status is returned or lockbits modified.

RDCMR[n] CMD:110 CYCLE:01

This register gets the value of the CMR addressed in Addr<14..3>, exactly as a GET_C. In addition, the lockbit of the CMR is read and shifted into the least significant bit of the LCKB register. The operation of this register is explained later. The RDCMR command is used to dump the state of an entire bank of 64 CMRs. After all the values have been collected with the RDCMR, all of the lockbits will be available in the LCKB register.

SND_C[n] CMD:000 CYCLE:10

This command attempts to write a value to the CMR addressed by the field Addr<14..3>. If the lockbit of the CMR is not set, the value is written, the lockbit is set and a status of 1 is returned. Otherwise, no write is performed and a status of 0 is returned.

SND_X[n] CMD:001 CYCLE:10

If the X register addressed in the field Addr<9..3> has a lockbit, then this command will act similarly to a SND_C. In this case, the lockbit would be tested, and a write would occur if it was not set. It would then be set and a status of 1 returned. If the lockbit had been reset, a status of 0 would be returned.

If the register has no lockbit, it will always write and always return a status of 1. The status returned by a SND_X to the PCIR is not defined, but since the PCIR is a read-only register, this is not a problem.

PUT_S[n] CMD:011 CYCLE:10

The command places data in the CMR addressed by Addr<14..3>. It returns the state of the lockbit as status.

PUT_C[n] CMD:100 CYCLE:10

The command places data in the CMR addressed by Addr<14..3>. It returns no status.

PUT_X[n] CMD:101 CYCLE:10

This instruction places data in the X register addressed by Addr<9..3>. It returns no status.

WRCMR[n] CMD:110 CYCLE:10

This command is the opposite of RDCMR. It writes a piece of data into the CMR address by Addr<14..3>. At the same time, a bit is shifted out of the top of the LCKB X register and written into the CMR's lockbit. The LCKB register could be written with a pattern to initialize the CMRs, but will usually contain a pattern collected from a sequence of RDCMRs.

INC_C[n] CMD:000 CYCLE:11

This command is used to add a value to the CMR address in the field Addr<14..3>. The value is presented along the same path as write data. If the CMR is locked, the current value and the increment value will be added and placed back in the CMR; a status of 1, along with the original value of the CMR will be returned. If the lockbit was not set, the value will not be changed, and will be returned with a status of 0.

On the board, this command requires and extra stage of pipelining. For this reason, it must not be immediately followed by another command. Since it has a cycle type of "Read-Modify-Write" (11), the crossbar should signal "CU busy" for an extra clock after this command is accepted.

While all of the CMR data is stored in the STRAMs, some information on certain specific locations is shadowed in the ASAP accelerator. For this reason, the INC_C command will not function properly on some of the "Hardware" CMRs. The correct value will be in the CMR, but the ASAP mechanism will see corrupted data. Because these particular locations have predetermined functions, an operation such as INC_C would probably not need to be performed on them anyway.

4.4.3 COMMUNICATION REGISTERS (CMRs)

The Neptune system has 32 possible values for the Communication Index Registers (CIRs), with which processes are associated. Each processor has one CIR which is loaded with the CIR index of the process that it is running. More than one processor can be working on the same process and will have the same CIR value loaded. For each CIR index, there are 128 Communication registers which are used by a process. 32 of the 128 CMRs are used by the "hardware level" — and some of them do affect hardware — and 32 are used by execution rings 0 through 3. The last 64 are used at the top level, ring 4. Large sections of virtual address space are reserved for the various types of CMRs, but only a small number, the 128 mentioned, are actually implemented. Table 4-6 shows the allocated and implemented locations within the virtual address space for each CIR index. Also listed is a "Physical" address. The actual address of a CMR within the CU's STRAMs can be derived by concatenating the 5-bit CIR value to the most significant bit of the 7-bit "physical" address number listed; this yields the 12-bit physical location. For example, location 0x5CA, or 10111001010 (binary), is CMR 0x0A (a hardware CMR) for CIR index 0x17.

Table 4-6: CMR Addressing

Allocated	Implemented	Physical	Function
0000-2FFF	0000-001F	CIR::00-1F	Hardware Ring
4000-4FFF	4000-401F	CIR::20-3F	Ring 0-3
8000-FFFF	8000-803F	CIR::40-7F	Ring 4

In order to allow each process to access all CMRs, the 4096 locations are available in a contiguous block from virtual address 3000 to virtual address 3FFF. The mapping in this range is one-to-one where the physical address is merely the lower 12 bits of the virtual address.

While the use of the most of the CMRs is completely up to the software, the "hardware" CMRs have specific functions. A map of these for each CIR index is shown in Figure 4-4. The contents of all the CMRs is kept in the STRAMS, but some information about the contents of locations 0x07, 0x08 and 0x0B is shadowed in the ASAP accelerator in the NADR gate array. For more information about this, consult the section on the ASAP accelerator. The rest of the locations are just addressed storage as far as the CU is concerned.

Each CMR location has a lock bit associated with it for coordinating multiple processor access. The lock bits are also stored in the STRAMs at the same addresses as the CMR data. For every read or write request of a CMR, the lock bit is read out of RAM and affects the next state of that CMR and lock bit. It also affects the status returned to the processor. The relationship of the lock bit to the various microcode operations was covered in section 4.4.2 .

Figure 4-4: Hardware CMRs

Address	64 Bits		
CIR::00	Reserved		
CIR::01	Reserved		
CIR::02	Trap Instruction Register Ring 0		
CIR::03	Trap Instruction Register Ring 1		
CIR::04	Trap Instruction Register Ring 2		
CIR::05	Trap Instruction Register Ring 3		
CIR::06	Trap Instruction Register Ring 4		
CIR::07	Thread Allocation Mask	CPU Mask	Thread Count
CIR::08	fork.FP	fork.AP	
CIR::09	fork.PC	fork.PSW	
CIR::0A	reserved	fork.source_PC	
CIR::0B	fork.type	fork.SP	
CIR::0C	SDR[0]	SDR[1]	
CIR::0D	SDR[2]	SDR[3]	
CIR::0E	SDR[4]	SDR[5]	
CIR::0F	SDR[6]	SDR[7]	
CIR::10	CPU 0 Execution Clock for Rings 0-3		
CIR::11	CPU 0 Execution Clock for Ring 4		
CIR::12	CPU 1 Execution Clock for Rings 0-3		
CIR::13	CPU 1 Execution Clock for Ring 4		
CIR::14	CPU 2 Execution Clock for Rings 0-3		
CIR::15	CPU 2 Execution Clock for Ring 4		
CIR::16	CPU 3 Execution Clock for Rings 0-3		
CIR::17	CPU 3 Execution Clock for Ring 4		
CIR::18	CPU 4 Execution Clock for Rings 0-3		
CIR::19	CPU 4 Execution Clock for Ring 4		
CIR::1A	CPU 5 Execution Clock for Rings 0-3		
CIR::1B	CPU 5 Execution Clock for Ring 4		
CIR::1C	CPU 6 Execution Clock for Rings 0-3		
CIR::1D	CPU 6 Execution Clock for Ring 4		
CIR::1E	CPU 7 Execution Clock for Rings 0-3		
CIR::1F	CPU 7 Execution Clock for Ring 4		

4.4.4 CONTROL SPACE REGISTERS

The CU is home to a variety of control registers – in their own address space – which are referred to as the "X registers". These registers control a variety of functions on the CU. These functions are described in a later sections. The previously described commands ending in "_X" access these registers and the field Addr<9..3> in the odd address data is used as the address of the X register. Table 4-7 is a mapping of the X space addresses. The LCKB and TRPCMD registers are the only two X registers that have lock bits. The PCIR register does not have a lock bit, but it will return a status to indicate if the posted CIR value is valid. If SND_X or RCV_X are used on any other X registers, the status will always return as "good".

Table 4-7: X Space Addresses

Addr<9..3>	Register	Description
00	LCKB	Lockbit shift register + Lock bit
01	TOC	Time of Century Counter
02	TRPCMD	Trap Command Register + Lock bit
03	PCIR	Posted Thread CIR (ASAP) + Status
04	NITC	Next ITC value
05	ITC	Interval Timer Counter
06	ITSR	ITC status register
07	ITIN	ITC Interrupt channel
08	Spare	Unused
09	IO INSTALL	A map of which ports contain NIAs
0A	CPU INSTALL	A map of which ports contain SPs
0B-0F	Spare	Unused
10	P0 CIR	Which CIR SP0 is executing
11	P1 CIR	Which CIR SP1 is executing
12	P2 CIR	Which CIR SP2 is executing
13	P3 CIR	Which CIR SP3 is executing
14	P4 CIR	Which CIR SP4 is executing
15	P5 CIR	Which CIR SP5 is executing
16	P6 CIR	Which CIR SP6 is executing
17	P7 CIR	Which CIR SP7 is executing
18-1F	Spare	Unused
20	P0 IDLE	Idle status of SP0
21	P1 IDLE	Idle status of SP1
22	P2 IDLE	Idle status of SP2
23	P3 IDLE	Idle status of SP3
24	P4 IDLE	Idle status of SP4
25	P5 IDLE	Idle status of SP5
26	P6 IDLE	Idle status of SP6
27	P7 IDLE	Idle status of SP7
28-2F	Spare	Unused
30	GP	Globally Pending Traps

Table 4-7: X Space Addresses(Continued)

Addr<9..3>	Register	Description
31	GE	Global Channel Enables
32	MBP	Memory Base Pointer
33-37	Spare	Unused
38	P0 LE	Local Channel Enable, SP0
39	P1 LE	Local Channel Enable, SP1
3A	P2 LE	Local Channel Enable, SP2
3B	P3 LE	Local Channel Enable, SP3
3C	P4 LE	Local Channel Enable, SP4
3D	P5 LE	Local Channel Enable, SP5
3E	P6 LE	Local Channel Enable, SP6
3F	P7 LE	Local Channel Enable, SP7
40-47	Spare	Unused
48	L0 BE	CPUs enable for broadcast, Channel 0
49	L1 BE	CPUs enable for broadcast, Channel 1
4A	L2 BE	CPUs enable for broadcast, Channel 2
4B	L3 BE	CPUs enable for broadcast, Channel 3
4C	L4 BE	CPUs enable for broadcast, Channel 4
4D	L5 BE	CPUs enable for broadcast, Channel 5
4E	L6 BE	CPUs enable for broadcast, Channel 6
4F	L7 BE	CPUs enable for broadcast, Channel 7
50-7F	Spare	Unused

4.4.4.1 Lockbit Shift Register

The LCKB register is used when dumping, restoring, or initializing a bank of 64 CMRs. When a CMR is read with a RDCMR, its lockbit is shifted into the least significant bit of the LCKB register. When a CMR is written with a WRCMR, the most significant bit of the LCKB register is shifted out and written into the lockbit of the CMR. This register contains 64 bits, one for each lockbit in a CIR-block. This register has a lockbit to prevent corruption when multiple processors are trying to use it. Typically this register would be accessed with SND_X and RCV_X operations.

4.4.4.2 Time of Century Counter

The TOC register is a 64 bit register which counts in microseconds. There is no enable to stop and starts the counting, but it may be written and read at any time.

4.4.4.3 Trap Command Register

The TRPCMD register is a 32 bit register used to issue commands to the trap logic. Refer to the section on traps and interrupts for a full description of this register. This register has a lockbit which behaves differently than the other lock bits.The lock bit is set when the register is written, and it is automatically reset when the contents are processed by the CU. The register may be read at any time and does not clear the lock bit.

4.4.4.4 Posted Thread CIR

The PCIR read-only "register" is the output of the ASAP accelerator. It is a value calculated during the read request and is not really a register. An idle processor typically reads it to find a CIR index which has a thread available for that processor. If the read returns with a status of 0, no threads were availablefor that particular processor. If a status of 1 is returned, the value is the CIR index which has the posted thread available for execution.

4.4.4.5 Next ITC Register

The NITC register contains a 16 bit read/write value. When the Interval Timer Counter (ITC) reaches 0, it is reloaded with the value in the NITC.

4.4.4.6 Interval Timer Counter

The ITC is a 16 bit value that decrements every ten microseconds. When it reaches zero, it may trigger an interrupt. It will automatically reload with the value in the NITC register or it may be written at any time. The ITC will only count when bit 2 of the ITSR is set.

4.4.4.7 ITC Status Register

The ITSR register only has three bits. The most significant, bit 2, may be written, but the lower two are read only. When bit 2 is set, the ITC is enabled to count. Bit 0 is set when the ITC rolls over. Bit 1 is set if the ITC rolls over again. It is used to tell if the ITC has gone off more than once without being serviced. When the ITSR is read, the two least significant bits reset to zero.

4.4.4.8 ITC Interrupt Channel Register

The ITIN register contains an 8 bit interrupt channel number. If the ITC rolls over to zero, it will set an interrupt flag. When the TRPCMD register is available (unlocked), it is loaded with an XMTI trap command with the channel number from the ITIN. For more information on XMTI traps, refer to the section on traps and interrupts.

4.4.4.9 IO INSTALL Register

The IO INSTALL register contains a 9 bit read/write value. Each bit location corresponds to each of the nine ports where an NIA can reside. Ports 0-8 correspond to bits 0-8 in this register. Ports 0-7 can contain CPUs or NIAs; port 8 will always contain an NIA. A bit is set only when an active NIA is attached to the corresponding port. This register is used to determine where XMTI traps above channel 8 can be sent.

4.4.4.10 CPU INSTALL Register

The CPU INSTALL register contains an 8 bit read/write value. Each bit location corresponds to each of the eight ports where a CPU can reside. Ports 0-7 correspond to bits 0-7 in this register. A bit is set only when an active CPU is attached to the corresponding port. This register is used to determine which CPUs can participate in trap dispatches and deadlock checking.

4.4.4.11 Communication Index Registers

There are eight CIRs in the CU, one for each of the eight possible CPUs in the system. The processors maintain these registers to shadow the values in the CIRs on the processors. Each register is 5 bits wide for the 32 possible CIR values. These registers are used for deadlock checking on the CU.

4.4.4.12 IDLE Registers

There are eight IDLE registers, one for each of the eight possible CPUs in the system. Each register is one bit wide. The CPUs maintain the IDLE registers and set a register to a "1" when the corresponding CPU is idle. The IDLE registers are used in the interrupt arbitration.

4.4.4.13 Globally Pending Interrupt Register

The GP register is a 8 bit read-onlyregister where bits 0-7 represents each of the CPU interrupt channels 0-7. Bits are set by XMTI commands being sent to the TRPCMD register for the interrupt channels 0-7. Interrupt channels greater than 7 are sent as XMTI traps to the NIA(s) for the IO system and have no effect on this register. Bits in this register are reset when the corresponding interrupts are dispatched.

4.4.4.14 Global Enables Register

The GE register is an 8 bit read/write register. Bits 0-7 are used to enable interrupts on channel 0-7 respectively. When a bit is set to "1", the corresponding interrupt channel is globally enabled. The interrupt channel is not necessarily enabled for dispatching since the LE and BE registers also control the channels. This register just provides to capability to enable interrupts on a channel by channel basis.

4.4.4.15 Memory Base Pointer Register

The MBP register is a 64 bit read/write register used to store the "Memory Base Pointer", an OS variable. It is only a centrally located storage register and performs no special function in the CU.

4.4.4.16 Local Enable Registers

There are eight LE registers which have 8 bit each. Each register corresponds to each of the eight possible CPUs. Each bit, 0-7, in those registers enables an interrupt channel, 0-7, to issue interrupts to that processor. For example, if P4 LE has a value of 0x31 then channels 4, 5, and 0 may interrupt processor 4; channels 1, 2, 3, 6, and 7 cannot interrupt processor 4 even if these channels are enabled otherwise.

4.4.4.17 Broadcast Enable Registers

There are eight, 8-bit BE registers which correspond to the eight interrupt channels. Usually an interrupt channel only interrupts to one processor even if several are locally enabled on that channel. In the case of broadcast, several processors receive the interrupt when it is dispatched. A channel is placed in broadcast mode by having a non-zero value in its BE register. In this case, The 8 bits in each BE register correspond to the 8 possible CPUs which can receive the interrupt when it is dispatched. For example, if L6 BE has a value of 0x58, then processors 4, 6 and 3 will get a simultaneous interrupt when channel 6 interrupts if these processors are all locally enabled on channel 6; if they are not all locally enabled the interrupt will stay pending.

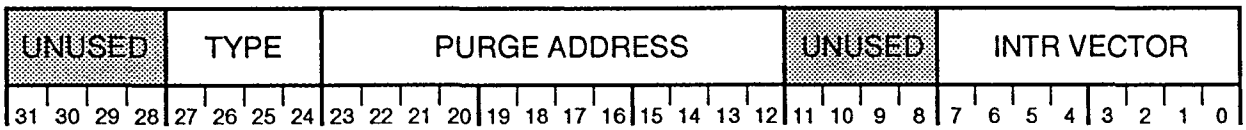
4.5 TRAPS AND INTERRUPTS

Traps allow the CPUs and NIAs to signal each other about events that need to be serviced such as cache coherency and IO transactions. Interrupts are basically traps in the Neptune system. Interrupts to CPUs have some channel arbitration functions, but they are dispatched as traps. The following sections cover the details of the trap functions of the CU.

4.5.1 TRAP TYPES AND VECTORS

In order to issue a trap, a CPU performs a SND_X to the TRPCMD register on the CU. The functional fields in the value written are shown in Figure 4-5. If a status of "1" is returned, the write was successful and the CPU can assume that the trap will be dispatched eventually. If a "0" status is returned, the TRPCMD register is locked, and the CPU must try again to issue the trap.

Figure 4-5: TRPCMD Register Fields



When a trap is dispatched, one or more trap ready (TRAP_RDY) signals are sent to the CPU ports or NIA ports via the XCL. Along with the TRAP_RDYs, a twelve bit TRAP_VECT and a four bit TRAP_TYPE will be sent. You may wish to refer to Table 4-1 on the crossbar interface for a listing of these signals.

The trap type is determined by the value written to the TRPCMD register bits 24-27. The trap type determines what the twelve bit vector will contain. Table 4-8 shows the possible trap type codes and what vectors they use. All of the data will be zero-extended to fill the full twelve bits. The "Trap CIR" refers to the CIR value of the processor issuing the trap, while "Deadlock CIR" refers to the CIR value of the processors which are deadlocked. Deadlock traps are discussed in detail in the later section on the trap state machine.

When a port has finished processing its trap, it sends back a trap complete (TRAP_COMP). This clears its busy bit in the trap state machine. When all of the ports have completed their traps, the CU may return an MT_COMP to the originating processor. Currently only a CTRSG type of trap returns a MT_COMP. An MT_SEL is sent with the MT_COMP so that the XCL can route it to the originator of the trap.

Table 4-8: Trap Types

Code	Type	Vector	Source	Dest	MT_COMP needed?
0000	XMTI	Intr. Vector	CPU/NIA	CPU/NIA	No
0001	Unused				
0010	Unused				
0011	Unused				
0100	PATU	Purge Address	CPU	CPU	No
0101	PMOD	Purge Address	CPU	CPU	No
0110	PREF	Purge Address	CPU	CPU	No
0111	LDSDR	Purge Address	CPU	CPU	No
1000	CTRSG	Purge Address	CPU	CPU	Yes
1001	TRAP	Trap CIR	CPU	CPU	No
1010	Unused				
1011	PATE	Purge Address	CPU	CPU	No
1100	Unused				
1101	DLCK	Deadlock CIR	CU	CPU	No
1110	Unused				
1111	Unused				

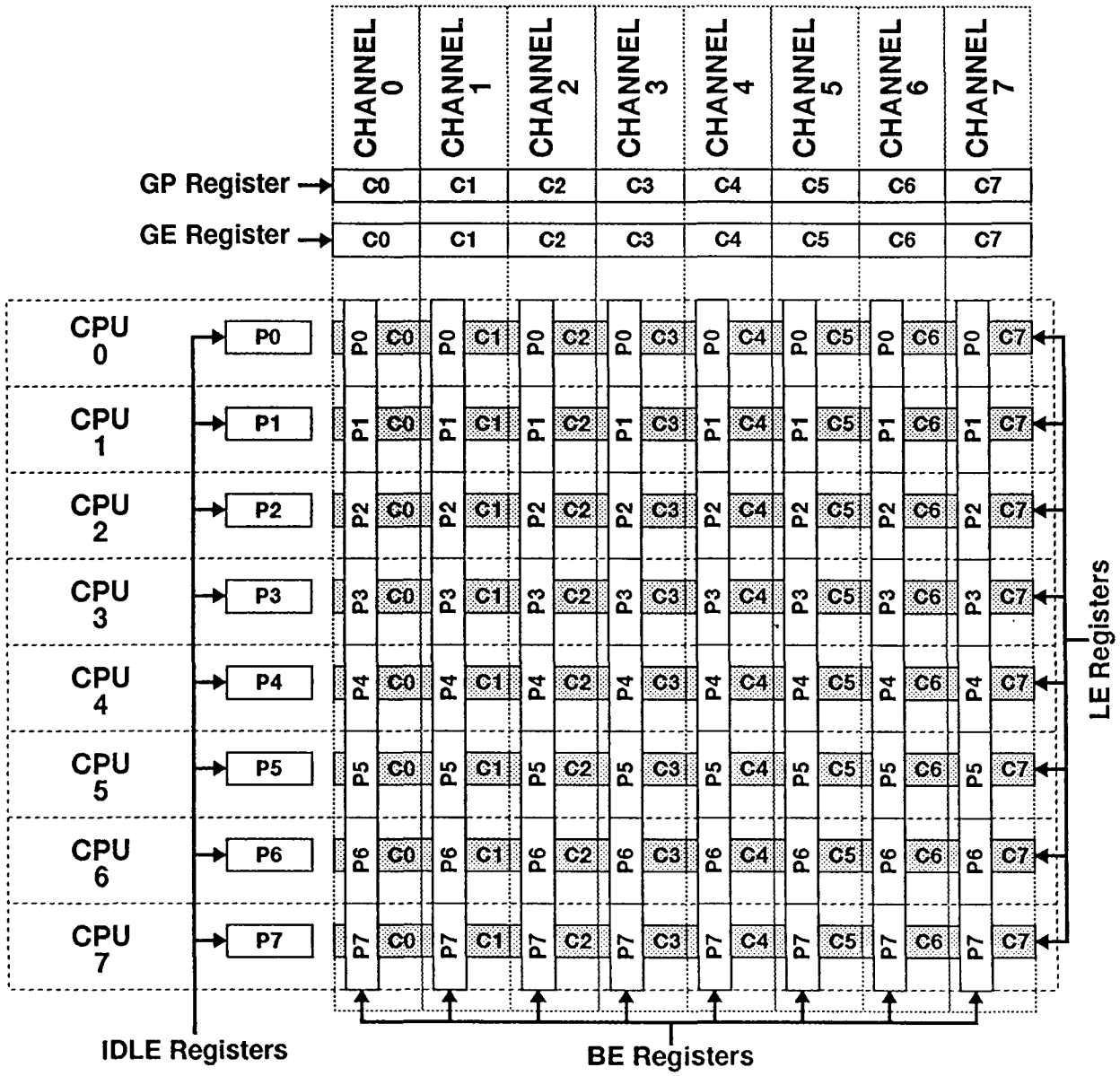
Traps are usually sent to all ports with active CPUs in them except for the port that initiated the trap. The active CPU ports are indentified in the CPU INSTALL register. An exception to this is the DLCK trap which only goes to the deadlocked CPU(s). Another exception to this is the XMTI trap which invokes a complex and flexible arbitration scheme. If the XMTI is sent with a vector of 8 or higher, it is dispatched to all IO ports, which are identified by the IO INSTALL register. If the XMTI has a vector of 0 to 7, it will be sent to an interrupt channel.

4.5.2 CPU Interrupt Arbitration

For the eight CPU interrupt channels, Channel 0 has the highest priority and channel 7 has the lowest. When an XMTI is sent to a CPU interrupt channel via the TRPCMD register, it sets the corresponding bit for that channel in the Global Pending (GP) X register. The channel may be programmed to send the XMTI trap to one of a set of CPUs or all of a set simultaneously.

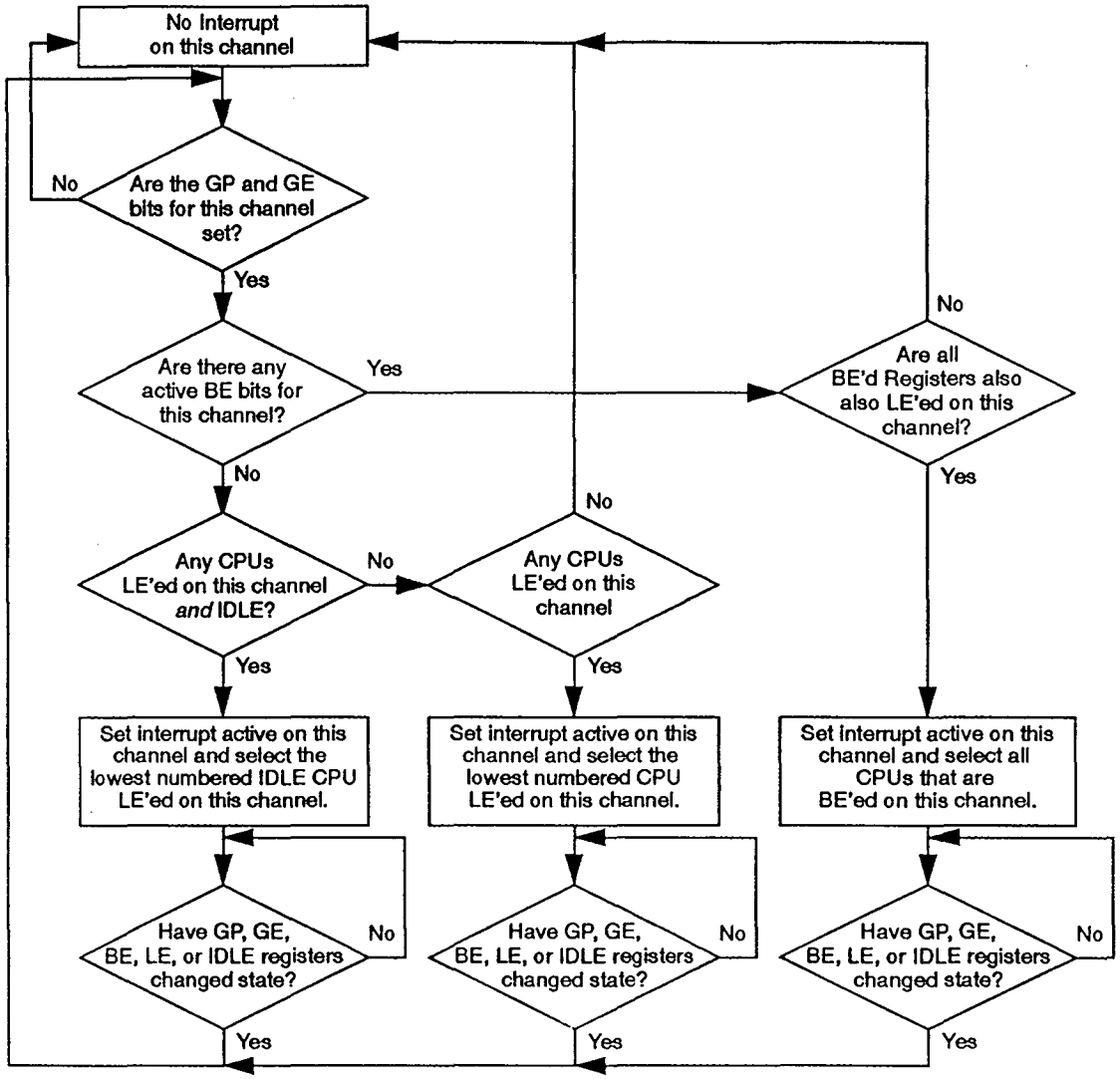
The interrupt arbitration is controlled by the bits in the LE, BE, GP, GE, and IDLE registers in the X space. Figure 4-6 shows how these registers are related to the interrupt channels and CPUs. Each "CPU" row in the figure indentifies which data bits and registers are relevant to the interrupt configuration for that CPU. Each "CHANNEL" column indentifies which data bits and registers are relevant to the configuration for that interrupt channel.

Figure 4-6: Interrupt Register Mapping



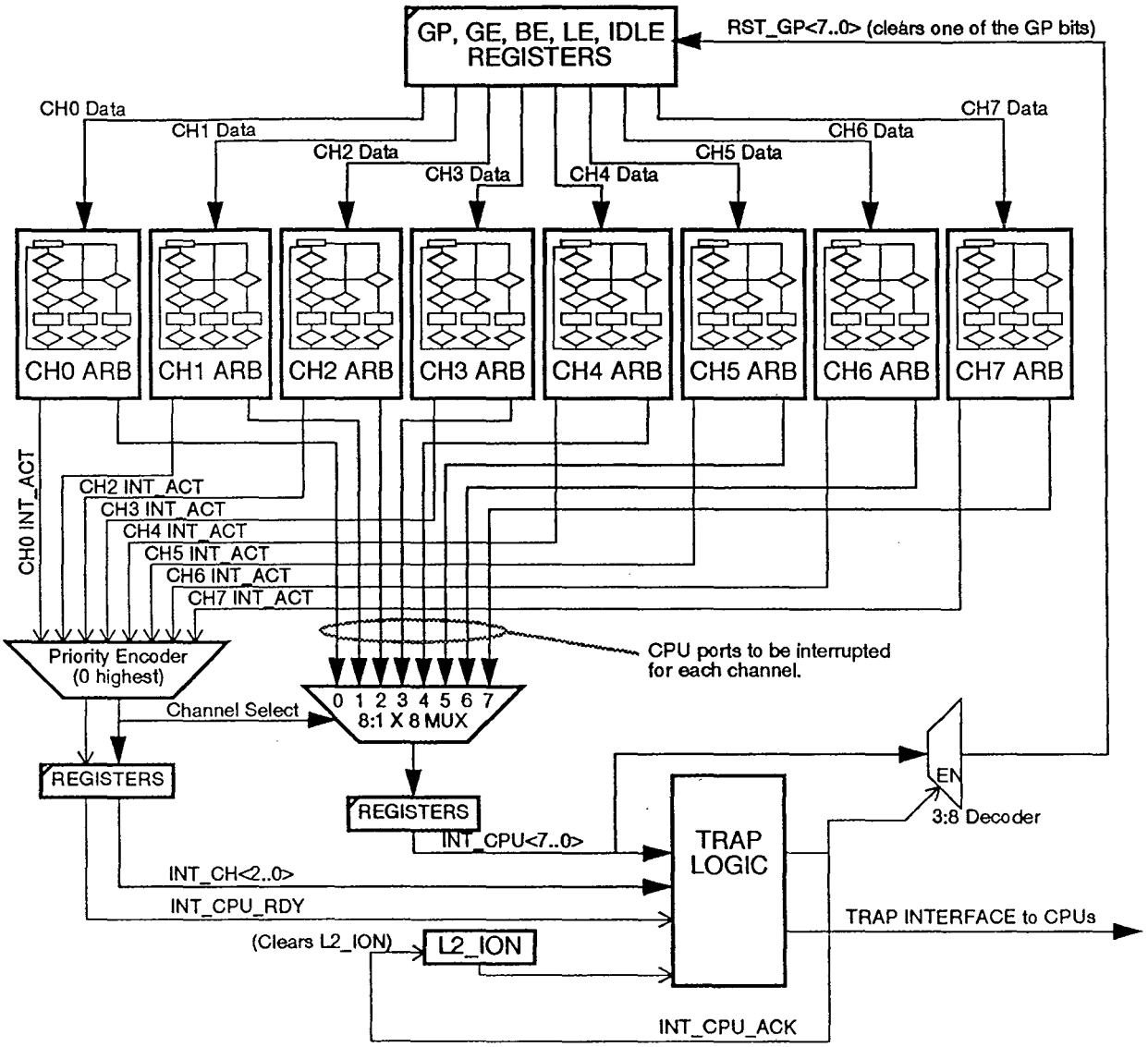
The "CHANNEL" data from the interrupt configuration registers are processed the same for each channel. The IDLE information affects all channels for each CPU. A flowchart of the arbitration process for each channel is shown in Figure 4-7. The flowchart represents the function of combinational logic which contains no registers, so any change in the input data can change the output of that channel immediately. The outputs for each channel are an INT_ACT flag and an 8 bit INT_CPU value . Each bit in the INT_CPU value represents which CPUs are being interrupted by that channel, one bit for each CPU. INT_ACT is set if any of the INT_CPU bits are set.

Figure 4-7: Interrupt Channel Arbitration



The outputs from all the channel arbitrations are proritized and sent to the trap logic as a request. Figure 4-8 shows a functional diagram of the interrupt request process. The data from the interrupt configuration registers are sent to the appropriate channel arbiters as covered above. At this point, more than one channel could have a valid interrupt state; so the INT_ACT signals are priority encoded to select the highest priority channel. Channel 0 is the highest proirity continuing down to channel 7 which is the lowest priority. INT_CH<2..0> is an encoded value registered from the priority encoder. INT_CPU_RDY indicates whether there is any interrupt which validates the INT_CH<2..0> value. The encoded output from the priority encoder is used to select the INT_CPU<7..0> value for the highest priority active channel. INT_CPU<7..0> is a registered value where each bit 0-7 represents each CPU 0-7. The set bits determine which CPUs will get the interrupt (XMTI) trap. The CPU_INSTALL register has no effect on which CPUs will get the XMTI trap, so the operating system must understand which processors are installed and set up the interrupt configuration accordingly.

Figure 4-8: Interrupt Request Arbitration



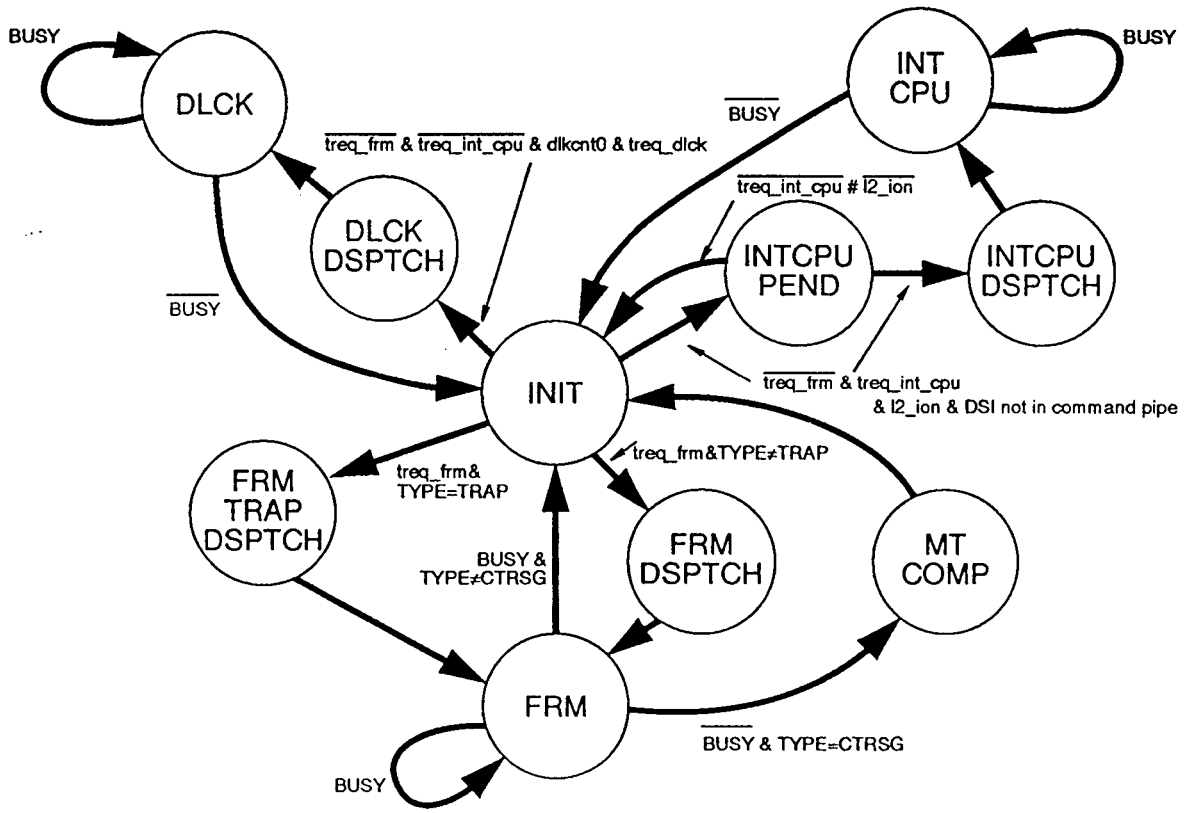
The trap logic receives the INT_CPU_RDY and L2_ION to form an interrupt trap request. L2_ION is the "interrupt on" bit at level 2 of the data pipe. L2_ION must be set to dispatch an XMTI trap to a CPU; it does not affect XMTI traps to NIAs. The interrupt request may not be service immediately if another trap in in progress. INT_CH<2..0> is dispatched as the XMTI vector. L2_ION is set by and ENI operation and can be cleared by a DSI operation or by INT_CPU_ACK which acknowledge that the interrupt has been dispatched. INT_CPU_ACK is also combined with INT_CPU<7..0> to clear the GP bit of the channel that was just serviced by the XMTI dispatch.

It is important to note that the functional diagram in Figure 4-8 is meant to show how the register levels and combinational functions interact at a high level. The real hardware implementation is somewhat different since the interrupt channels are "sliced" across two NDAT gate arrays. The division of the channels causes the priority functions to be a little more distributed between the gate arrays and board level logic. The figure was simplified for clarity. Section 4.8 covers the hardware implementation in greater detail.

4.5.3 TRAP STATE MACHINE

A state diagram of the trap state machine (TSM) is shown in Figure 4-9. The TSM controls the sequencing of the traps. It works in conjuction with logic on the NADR and NDAT gate arrays to accept, dispatch, and resolve traps. When no traps are active, the TSM sits in the INIT state. A firmware trap, which is activated by the "treq_frm" signal, has the highest priority for dispatching. CPU intterupts can be queued in the global pending register (in the NDAT) and can be dispatched if ION is active and if there are no firmware traps pending. "treq_int_cpu" indicates that there is an unmasked interrupt ready for dispatching. There is a "treq_dlck" signal for deadlock traps that starts a deadlock counter when a deadlock condition is detected. When that counter (in the NADR) reaches zero and there are no other traps pending, a deadlock trap is dispatched. The deadlock counter will hold at zero until the TSM can get around to servicing the trap.

Figure 4-9: Trap State Machine



All of the DSPTCH states select the proper proper ports for the traps to be sent to and the vector to go with the trap. Table 4-9 shows the ports and vectors selected for the different DSPTCH states. The trap goes to the crossbar on the clock following the DSPTCH states. CPU interrupts (XMTI trap with vector 0 thru 7) go through an additional CPUINT_PEND state to account for the timing of interrupt masking operations; it prevents false starts. When a CPU interrupt is dispatched (CPU DSPTCH state), the ION bit is cleared and the corresponding channel bit in the GP register is cleared in addition to sending out the trap.

Table 4-9: Trap Dispatch Table

DSPTCH State	Vector	Ports	Comments
INTCPU	0-7 (from command)	Chosen by Arbiter	XMTI with vector 0-7 only
FRMTRAP	CIR of issuing SP	All CPUs except sender.	
FRM	Purge Address	All IO for XMTI. All CPUs except sender.	XMTI with vector 8-255
DLCK	CIR of deadlock	Deadlocked ports	All CPUs with same deadlocked CIR

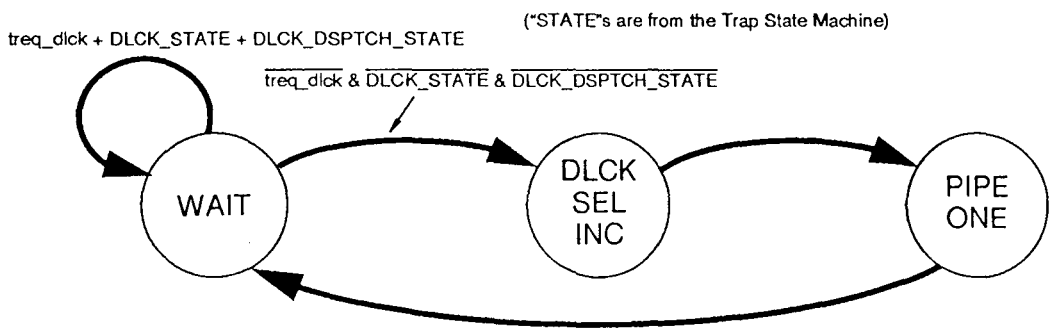
All of the DSPTCH states proceed to a "wait" state which waits for all of the trapped CPUs/ NIAs to acknowledge the traps. The trap will remain "busy" until all the trapped ports have responded. There is a 16 bit, 1usec resolution, scan-programmable watchdog timer that checks for a hang in any of these "wait" states. A harderror is issued if the trap does not resolve in the programmed time. This timer feature can also be disabled via scan.

All traps, except the CTRSG type trap, return to the init state when the trap is complete. The CTRSG trap transitions through the MT_COMP (microtrap complete) state which sends an acknowledge back to the CPU which originally issued the CTRSG trap.

4.5.4 DEADLOCK STATE MACHINE

A state diagram of the deadlock state machine is shown in Figure 4-10. It interacts with the trap state machine and controls the scanning of the processors for a deadlock condition. When the TSM is not in the DLCK_STATE or the DLCK_DSPTCH_STATE, the deadlock state machine cycles between its three states. PIPE_ONE and WAIT are basically dead states used to keep the deadlock circuitry synchronized; they have no effect on any other operations. The deadlock monitor of the NADR steps through the CPUs, checking each CPU's deadlock and CIR versus all of the others. DLCK_SEL_INC causes this monitor to check the next of the 8 possible processors regardless of whether it is installed or not. If all installed processors with the same CIR are each indicating a deadlock condition, "treq_dlck" is set. "treq_dlck" starts the deadlock counter which can initiate a deadlock trap if the counter reaches zero. The deadlock state machine stops at the WAIT state until the "treq_dlck" signal goes away and the TSM is not handling a deadlock trap.

Figure 4-10: Deadlock State Machine



4.5.5 ITC INTERRUPT

As mentioned early, when the ITC reaches zero, it generates an interrupt. It accomplishes this by writing an XMTI trap command to the TRPCMD register. The vector is read from the ITIN register which specifies the interrupt channel. The ITC interrupt is essentially indistinguishable from an XMTI sent from an external source.

4.5.6 TRAP TIMEOUT

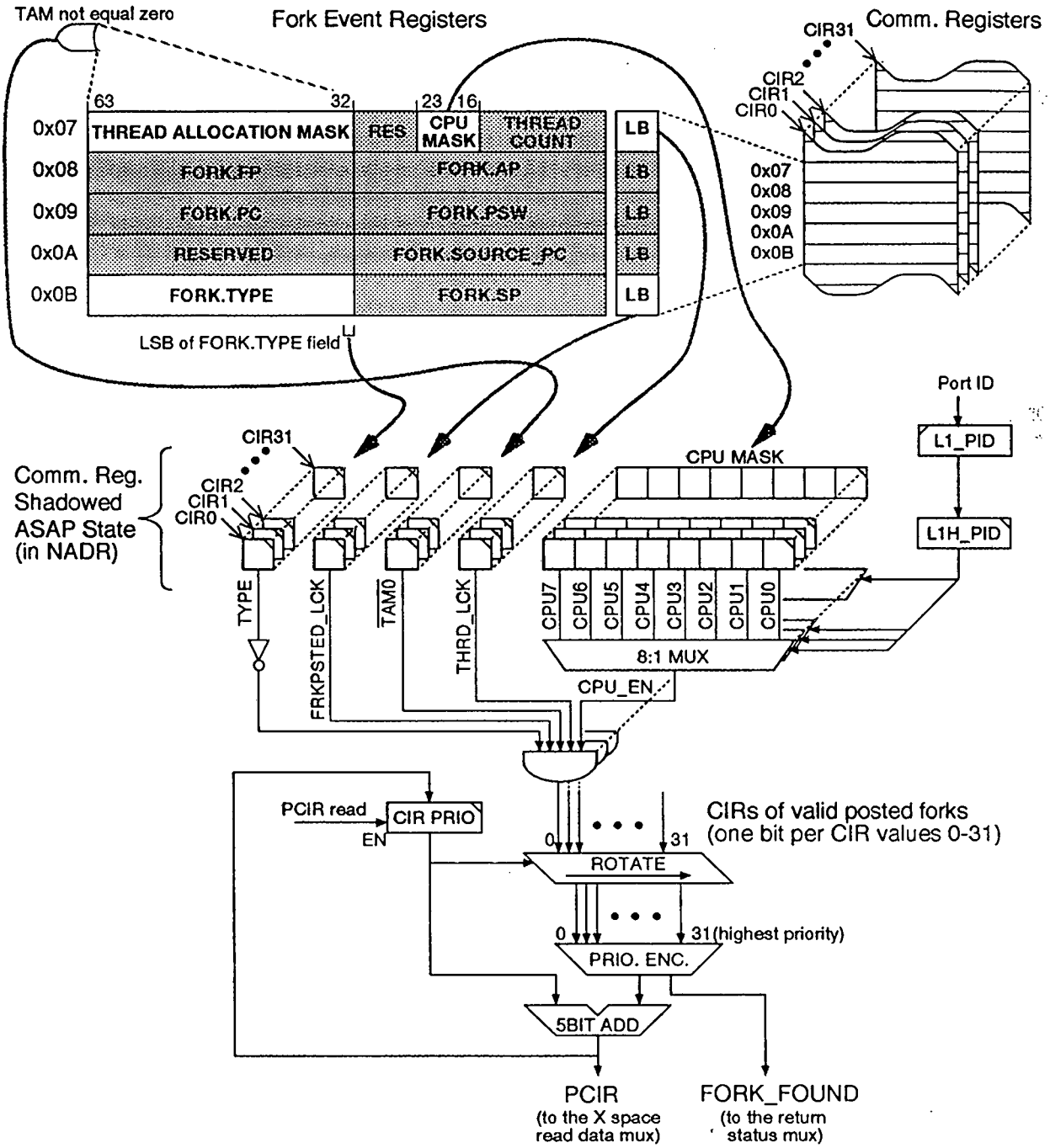
While the TSM is in a "wait" loop waiting for a trap to complete, a time-out counter runs. It is a scan programmable 16 bit counter with a 1 usec resolution. If all trapped ports do not acknowledge the trap within the programmed amount of time, a harderror occurs. This feature can be enabled and disable via scan.

4.6 ASAP ACCELERATOR

This section covers the CU hardware which supports the Automatic Self-Allocating Processors (ASAP) feature of the C3800. When a process running under a certain CIR value wants to start a new thread(s), it sets up information in the "Hardware" CMRs dedicated to ASAP. When all the data is written into these special CMRs, meaning the lockbits are set, it is ready to be read by another CPU so that it can pick up a thread of the process. This will only happen if the thread is not "stopped" (LSB of fork type is 0), if the Thread Allocation Mask (TAM) does not equal zero, and if the requesting CPU is enabled by the CPU mask for that CIR. A more detailed explanation of ASAP at the software level can be found in the *Convex Architecture Reference.*

In earlier Convex machines an idle processor polled the ASAP related CMRs for each CIR value. In the C3800, the number of possible CIR values has gone from 8 to 32 which increases the total number of registers to be polled; and the maximum number of processors has increased from 4 to 8 which increases the possible contention for reading the registers. To decrease the latency of ASAP in the C3800, the ASAP polling function was moved into hardware so that the idle processors can do one read operation without having to poll the CMRs. This is accomplished by the ASAP accelerator function in the NADR gate array shown in Figure 4-11.

Figure 4-11: ASAP Accelerator Block Diagram.



The subset of CMRs at addresses 0x07 thru 0x0B for each of the 32 CIR values are used for controlling process forking. They are called the "fork event registers" for each CIR value. The read/write data and lock bits for these registers are contained in RAMs. The ASAP information relevant to fork validity is highlighted in Figure 4-11. The state of these fields for all CIR values is shadowed in the NADR gate array in a set of write only registers. Whenever one of these fields in the RAM is modified, its shadowed state is updated.

The Thread Allocation Mask is a 32 bit field where each bit represents a thread of a process which can be executed in parallel with another thread. If no bits are set, no threads are available. The shadowed state of this field is one bit which is basically the logical-OR of the Thread Allocation Mask bits. If this shadowed state is "1", then there is at least one thread available. There is one of these shadowed state bits for each of the 32 CIR values.

The CPU Mask is an 8 bit field where each bit represents each of the 8 possible CPUs in the C3800 system. The CPU Mask for a specific CIR value determines whether a requesting CPU can participate in the process with that CIR value. This allows different processes to be assigned to a different sets of CPUs by masking off unwanted CPUs. If a bit in this field is "1", that CPU number can participate pick up a fork from a process with that CIR value. The 32 CPU Mask fields are all shadowed in the NADR.

Presently the Fork Type field can take on three values: STOPPED (0xB), PFORKED(0x0), and SPAWNED(0xA). Refer to the *Convex Architecture Reference* for the detailed meaning of these values. For process forking, the important thing is that the fork type is not STOPPED. The least significant bit of this field is shadowed in the NADR for each CIR value. If this bit is set, the process forking is stopped for that CIR value, and no more processors can pick up threads of that process.

The lock bits of the registers at addresses 0x7 (thread lock bit) and 0xB (fork posted lock bit) indicate whether the data in the fork event registers are ready for acceptance or not. If the two lock bits are set, the data for that CIR value is available; if the two lock bits are not set, the data is in transition or no forks have been posted for that CIR value.

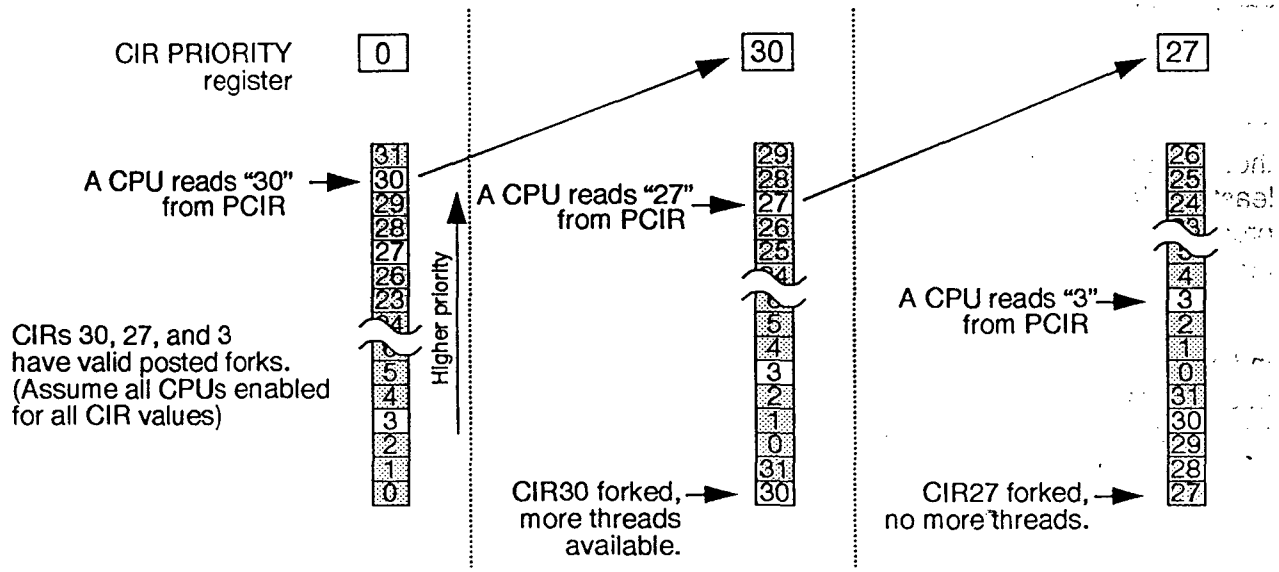
When a CPU requests a read from the PCIR address (typically with RCV_X), the valid posted forks for all of the CIR values are calculated from the shadowed ASAP state registers. The set of valid forks can be different for the different CPUs because of the CPU Mask. The port ID (PID) for that CPU selects the bits out of all 32 CPU Mask fields which enable the allowed CIR values for that CPU. A fork is valid for a CIR value when the fork type lsb is "0", the Thread Allocation Mask is not zero, the forkposted and thread lock bits are "1", and the CPU is enabled for that CIR.

The 32 valid fork status bits are run through a rotating priority function to select one of the valid forks for a CPU if there is a valid fork. A CIR PRIORITY register maintains a 5 bit encoded value of the last CIR that was read from the PCIR. The CIR PRIORITY value rotates the fork status bits so that the last CIR position read for the PCIR is rotated to position "0" before entering a 32 bit priority encoder. This pushes the last posted CIR value to the bottom of the priority so that the ASAP is fair to multiple processes. The priority encoder selects the highest set position and encodes it to a 5 bit value. This value is just a position, so a 5 bit adder adjusts this value by adding the CIR PRIORITY to it to compensate for the rotation. The sum, which can roll over, is the 5 bit CIR value which the requesting CPU reads. The CPU then goes directly to the CMRs for that CIR value to pick up a thread of that process. The reading of PCIR also loads the CIR PRIORITY with the new value.

Figure 4-12 shows an example of how the ASAP prioritizes several posted forks. At first three valid forks are posted. The CPU MASK can affect which forks can be seen by which processors; so for simplicity, assume all processors are masked the same. The process with CIR=30 is at the top of the priority when the first CPU reads the PCIR. That CPU then goes and reads the fork event CMRs for CIR=30. This action temporarily takes the CIR=30 from the posted fork queue since the lock bits get cleared. The CPU takes a thread and updates the fork event

CMRs which sets the lock bits again. In this example, there are still some threads left after the update; so CIR=30 is still a valid fork. In the second frame of Figure 4-12, The CIR PRIORITY register was written with "30" from the last PCIR value that was read; this rotates CIR=30 to the bottom of the priority queue. CIR=27 is now at the top of the queue and is read by the next CPU. In this example, there are no more threads left when the CPU updates the fork event registers for CIR=27. In the last frame of the figure, CIR=27 is now at the bottom of the queue, and CIR=3 is the highest posted fork which is read by the next CPU. Since CIR=30 still has threads available, it will be selected the next time the PCIR is read after CIR=3 is read.

Figure 4-12: ASAP Priority Example



In actual operation, the ASAP configuration is very dynamic. Forks are constantly being posted and unposted, and the CPU MASKs for the CPUs can be different. The PCIR gives a value based on the fork event states at the time it is read. There is a possibility that two or more CPUs could get the same value from the PCIR on consecutive reads. One CPU could be changing the fork event registers of that CIR value before the next CPU gets to the same fork event registers. The locks on the fork event registers will prevent any problems, and later CPU will go back and try to get another fork from the PCIR.

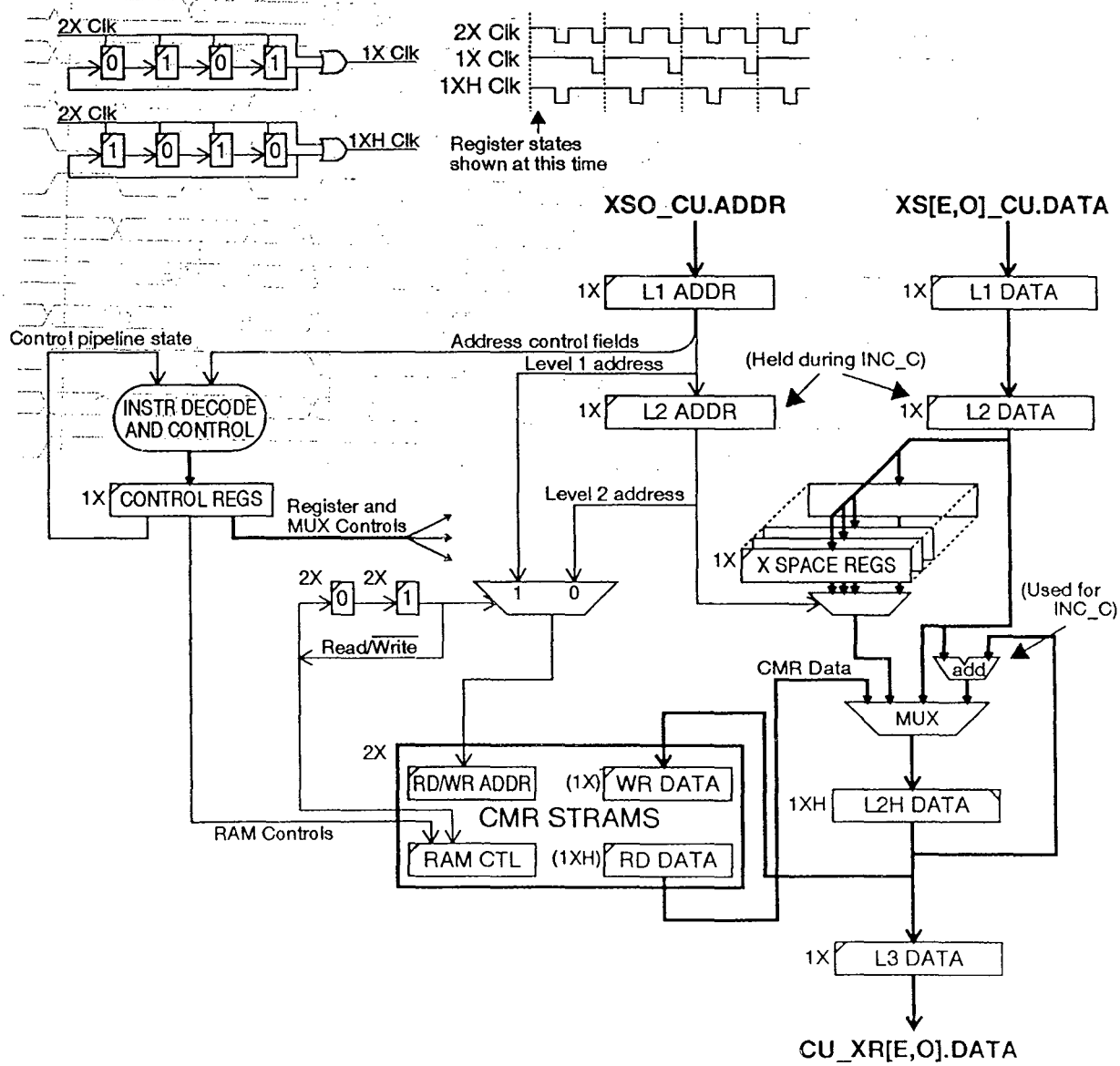
4.7 PIPELINE AND INTERFACE TIMING

This sections covers the operations of the CU address/data pipeline and various trap and status interfaces. All interfaces to the crossbar are clocked by the 1X system clock, and all interfaces to the crossbar are registered with no combinatorial logic between the CU registers and the crossbar interface.

4.7.1 ADDRESS/DATA PIPELINE

All requests to the CU are pipelined, and the CU can accept requests every system clock except for the INC_C instruction which has a dead cycle automatically inserted by the crossbar. Figure 4-14 shows a diagram of the data and address pipeline. The actual hardware implementation is distributed between board level logic and gate arrays, but the diagram is simplified for clarity.

Figure 4-13: CU Address/Data Pipeline



The CU logic receives a 2X clock from the clock generator and internally divides this clock into two 1X phases. One phase is in phase with the 1X system clock, and the other phase (1XH) is 180 degrees out of phase relative to the positive edge placement. The 2X clock and two phases of 1X clocks are distributed to various registers in the pipeline. The behavior of the pipe, as viewed by the system, is as if there are three register stages clocked by the 1X system clock; the 2X and 1XH clocks are only used internal to the pipe.

Figure 4-14: CU Pipeline Timing