Copy
Steve [illegible]

C38xx Memory Board/Card Functional Specification

Document No. 416-001246-000

Revision 1.1
May 8, 1991

INTERNAL USE ONLY

PROPRIETARY DISCLAIMER

This document is **proprietary.** As such, it is **not** approved for field or customer distribution. It is approved for **internal use only.** Distribution or use outside CONVEX is **strictly prohibited.**

CONVEX Computer Corporation

Table of Contents

1 Introduction 1-1
1.1 Document Organization 1-1
1.2 Physical Description 1-1
1.3 Major Subsystems 1-1
1.4 Conventions 1-2
1.5 Abbreviations 1-3
2 NMB Interfaces 2-7
2.1 External Interfaces 2-7
2.1.1 XSE - Even Cross Bar Interface 2-8
2.1.2 XRE- Even Cross Bar Return Interface 2-15
2.1.3 XSO and XRO Interfaces 2-15
2.1.4 Hard and Soft Errors 2-16
2.1.5 Clock and Scan Interface 2-17
2.2 Internal Interfaces 2-25
2.2.1 NMB Clocks 2-27
2.2.2 NMC Clock Generator 2-34
2.2.3 BCGA to NMC 2-35
2.2.4 ISE to NMC 2-37
2.2.5 NMC to READMUXE to RDEDC 2-37
2.2.6 BCGA to ISE 2-37
2.2.7 BCGA to LDREG, RDEDC and READMUXE 2-38
2.2.8 NMB Scan Rings 2-39
3 Functional Description 3-45
3.1 ECC, Error Check and Correct 3-46
3.1.1 Three and Four Bit Errors 3-50
3.2 The NMC 3-50
3.2.1 NMC/NMB Datapath 3-53
3.3 DRAM Operations 3-54
3.3.1 DRAM Full Writes 3-54
3.3.2 DRAM Reads 3-59
3.3.3 Refreshes 3-63
3.3.4 Read-Modify Write DRAM Operations 3-65
3.3.5 Diagnostic Mode Writes 3-73
3.4 Error Logging and Isolation 3-74
3.4.1 Soft Errors 3-74
3.4.2 Hard Errors 3-78
3.4.3 Crossbar Errors 3-84
3.4.4 Multiple Errors 3-85
3.4.5 Soft Errors During LOG Scan 3-86
3.4.6 Error Enables 3-86
3.4.7 Additional RDEDC Logging Information 3-87

3.5 Clocks and Scan 3-88
3.5.1 NMB_CLOCK 3-88
3.5.2 Scan 3-93
3.5.3 MIM_CTL, the NMC Clock Generator 3-94
3.6 Single Step 3-101
3.6.1 System Clocked/ Free Running Boundaries 3-102
3.6.2 Accessing Memory During Clock Stop 3-107
3.7 Initialization 3-109
4 Programming the NMB 111
4.1 Programming Fields 111
4.2 Timing Parameters 111
4.3 Timing Parameter Values 112
5 Build Procedure 5-113
5.1 Introduction 5-113
5.2 Tools 5-113
5.2.1 Command and Log Files 5-114
5.3 Design Directories 5-114
5.3.1 Links 5-115
5.4 Respin 5-116
5.4.1 Packaging 5-119
5.4.2 Placetool 5-120
5.4.3 Batchdrc 5-120
5.4.4 Netsched 5-120
5.4.5 Clktune 5-121
5.4.6 Itv 5-122
5.4.7 Dwirsif 5-123
5.4.8 Conroute 5-124
5.4.9 Checking the Route 5-125
5.4.10 Tlc 5-126
5.5 Trouble Shooting 5-126
5.5.1 Routing Issues 5-127
5.6 Respinning the NMC 5-127
5.6.1 Placement 5-127
5.6.2 Routing 5-128
5.6.3 Power Planes 5-128
Signal List .... Appendix A
Diagrams .... Appendix B
B.1 Views of the NMB B-149
B.2 **Class Foils B-149**
B.3 Example of NMC Scan B-207
ISP Models .... Appendix C
C.1 Basics C-215
C.2 Topology Files C-215

C.3 Simulation Directory C-217
C.4 Proc.isp C-217
C.5 Clock.isp and Spu.isp C-219
C.6 Xbar.isp C-220
C.7 Mem.isp C-221
C.8 Low Level Memory Models C-221
C.9 Memory Files C-223
C.10 Makefile C-223
C.11 Startup Files C-224
C.12 Running the Simulation C-224
C.13 Simulating Scan C-225
C.14 Trace Files C-225
C.15 ISP Conventions C-226
C.16 Testing the NMB C-226
Related Documents .... Appendix D

List of Figures

Figure 2-1 – NMB Interfaces 2-7
Figure 2-2 – Address bit mapping at NMB 2-9
Figure 2-3 – Parity Error Detect Timing 2-12
Figure 2-4 – Request Timing to the Same Bank 2-12
Figure 2-5 – Refresh, RDY and BANK_DONE 2-13
Figure 2-6 – Example of Suppressed BANK_DONE Because of Refresh 2-14
Figure 2-7 – Example of Simultaneous Refresh and BANK_DONE 2-14
Figure 2-8 – RDY to RD_RDY Timing 2-15
Figure 2-9 – Entering Log Scan 2-19
Figure 2-10 – SCAN_IN/SCAN_OUT in Log Scan 2-20
Figure 2-11 – Entering ALL Scan 2-21
Figure 2-12 – NMB SYS Clocks at Clock Stop 2-24
Figure 2-13 – Requests Across Clock Stop 2-25
Figure 2-14 – Detailed Internal Interfaces 2-26
Figure 2-15 – Hard Error Staging 2-32
Figure 2-16 – LD_EN and RTN_SEL 2-39
Figure 2-17 – Log Ring 2-41
Figure 2-18 – Sys Ring 2-42
Figure 2-19 – All Ring 2-44
Figure 3-1 – Bank Zero, Even Side Logic 3-46
Figure 3-2 – Schematic Representation of the NMC 3-51
Figure 3-3 – Bank Datapath 3-53
Figure 3-4 – DRAM Full Write Operation 3-55
Figure 3-5 – Full Write Dataflow 3-58
Figure 3-6 – DRAM Read Operation 3-60
Figure 3-7 – Read Dataflow 3-61
Figure 3-8 – DRAM Refresh Operation 3-64
Figure 3-9 – DRAM Read-Modify-Write 3-66
Figure 3-10 – Test-and-Modify Dataflow 3-67
Figure 3-11 – DRAM Read-Modify-Write, Error Correct 3-69
Figure 3-12 – Test-and-Modify Dataflow, Correct Cycle 3-70
Figure 3-13 – Diagnostic Write Dataflow 3-73
Figure 3-14 – Soft Error Decode 3-77
Figure 3-15 – Hard Error Decoding 3-81
Figure 3-16 – BCGA Hard Error Decode 3-82
Figure 3-17 – Bank Hard Error State Decode 3-83
Figure 3-18 – Clock Skew 3-88
Figure 3-19 – Basic NMB Clock Tree 3-89
Figure 3-20 – Phase Generator Circuit 3-90
Figure 3-21 – Phase Generator Timing 3-91
Figure 3-22 – NMB Clock Distribution 3-92
Figure 3-23 – MIM_CLK Generator 3-95
Figure 3-24 – MIM_CLK Timing 3-96

Figure 3-25 – State Diagram for NMC Clock Controller 3-98
Figure 3-26 – NMC Clock Controller during ALL Scan 3-99
Figure 3-27 – NMC Clock Controller during LOG/SYS Scan 3-100
Figure 3-28 – Details of LOG/SYS Scan 3-101
Figure 3-29 – System/Free Running Boundaries 3-103
Figure 3-30 – Single Step Example TIming Diagram 3-104
Figure 3-31 – Single Step Example Datapath 3-105
Figure 3-32a – Single Step Example 3-105
Figure 3-32b – Single Step Example 3-105
Figure 3-32c – Single Step Example 3-106
Figure 3-32d – Single Step Example 3-106
Figure 3-32e – Single Step Example 3-106
Figure 3-32f – Single Step Example 3-107
Figure 3-32g – Single Step Example 3-107
Figure 4-1 – Timing Parameters 111
Figure 5-1 – Pre-route Steps for Respin 5-117
Figure 5-2 – Route Steps for Respin 5-118
Figure 5-3 – Post Route Respin Steps 5-119
Figure B-1a – NMC Scan with a Multiple of Eight Bits B-208
Figure B-1b – NMC Scan with a Multiple of Eight Bits B-209
Figure B-1c – NMC Scan with a Multiple of Eight Bits B-210
Figure B-1d – NMC Scan with a Multiple of Eight Bits B-211
Figure B-2a – NMC Scan without a Multiple of Eight Bits B-212
Figure B-2b – NMC Scan without a Multiple of Eight Bits B-213
Figure C-1 – Top Level Organization C-215
Figure C-2 – Sys.t Topology File C-216
Figure C-3 – Mblk.t C-221
Figure C-4 – Bcga.t C-222
Figure C-5 – Rdedc.t C-222
Figure C-6 – Microchannel.t C-223

List of Tables

Table 1-1 – Board Abbreviations 1-3
Table 1-2 – Source-Destination Board Abbreviations 1-4
Table 1-3 – NMB Logic Blocks and Gate Arrays 1-5
Table 2-1 – XSE: Even Cross Bar Send Interface Signals 2-9
Table 2-2 – CYCLE Codes 2-10
Table 2-3 – WR_ZONE to Data Byte Mapping 2-10
Table 2-4 – Request Parity 2-11
Table 2-5 – XRE: Even Crossbar Return Interface Signals 2-15
Table 2-6 – Error Signals 2-16
Table 2-7 – RDY to HARD_ERROR Timing 2-16
Table 2-8 – RDY to SOFT_ERROR Timing 2-17
Table 2-9 – Scan and Clock Signals 2-17
Table 2-10 – SCAN_CTL Codes 2-18
Table 2-11 – NMB_CLOCK Clock Logic Signals 2-27
Table 2-12 – Clock Types Versus Scan Mode 2-28
Table 2-13 – Register Control Signals 2-29
Table 2-14 – ALL_141_CTL 2-29
Table 2-15 – SYS_241_CTL 2-30
Table 2-16 – NMB_CLOCK Error Signals 2-30
Table 2-17 – Error Sources 2-31
Table 2-18 – NMB_CLOCK Scan Signals 2-33
Table 2-19 – NMB_CLOCK Miscellaneous Signals 2-33
Table 2-20 – MIM_CTL Interface Signals 2-34
Table 2-21 – BCGA to NMC Interface 2-35
Table 2-22 – ISE to NMC Signals 2-37
Table 2-23 – NMC to READMUX to RDEDC Signals 2-37
Table 2-24 – BCGA controls to LDREG, RDEDC and READMUXE 2-38
Table 2-25 – LDREG Select Encode 2-39
Table 3-1 – ECC Code 3-48
Table 3-2 – Single Bit Error Decode 3-48
Table 3-3 – Decoding Syndrome 3-49
Table 3-4 – Data and ECC Grouping 3-50
Table 3-5 – Soft Error Scan State 3-75
Table 3-6 – Hard Error Scan State 3-78
Table 3-7 – ISE Hard Error Logging Registers 3-79
Table 3-8 – BCGA Input Staging Registers 3-79
Table 3-9 – BCGA Bank Error Logging State 3-80
Table 3-10 – Crossbar Error Registers 3-84
Table 3-11 – Decoding Multiple Errors At a Bank 3-86
Table 3-12 – Minimum Hard Error Disable 3-87
Table 3-13 – NMB Error Disables 3-87
Table 3-14 – PAL 5303, MIM_CLK Generation 3-95
Table 3-15 – NMC Scan States 3-97
Table 3-16 – Clock State Machine Inputs 3-98
Table 3-17 – Input Decodes 3-98

Table 3-18 – Memory Access During SYS Clock Stop 3-108
Table 4-1 – BCGA Timing Registers 111
Table 4-2 – Timing Field Values 112
Table 5-1 – List of CAD Tools 5-113
Table 5-2 – Design Directories 5-114
Table 5-3 – Incremental Route Steps 5-124
Table A-1 – ALL_141_CTL A-131
Table A-2 – WR_PAR A-132
Table A-3 – B0E_CTL A-132
Table A-4 – HARD_ERROR_E A-136
Table A-5 – LD_EN A-137
Table A-6 – MIM_CMOS_SCAN A-139
Table A-7 – SYS_241_CTL A-144
Table A-8 – Scan Control Codes A-145
Table A-9 – XSE_MB.CTL_PAR A-145
Table A-10 – CYCLE Codes A-146
Table C-1 – Memory Files C-216
Table C-2 – Simulation Directories C-217
Table C-3 – Sample Input to trgen C-218
Table C-4 – Mmk Trace File Generator Input C-219
Table C-5 – Spu Model Mode Bits C-219
Table C-6 – Ucode Input File C-220
Table C-7 – Signal Naming Conventions C-226

1 Introduction

1.1 Document Organization

The NMB functional specification is organized into four chapters and several appendices. Following chapter one, the introduction, is the Interfaces chapter which describes the NMB as it appears to the rest of the system. Chapter three is the functional description chapter, a lengthy chapter which covers the internal details of the NMB. Chapter four contains notes necessary for respinning the NMB. Appendix B contains all the foils used in the NMB training class and is a good overview of the NMB.

Of the four chapters, the Introduction and the Interface chapter, particularly the first section in that chapter dealing with external interfaces, are the most useful chapters for a casual reader. The remaining chapters are only of interest as a reference or to some one respinning the NMB. Appendix B, the Diagrams Chapter, contains the foils used for the NMB training class.

1.2 Physical Description

The Neptune memory system consists of one to eight Neptune Memory Boards (NMBs) and a crossbar complex composed of seven boards, two XS0's, two XS1's, two XRT's and one XCL. The crossbar boards are organized into even and odd triples known as XS0E, XS1E, and XRTE and XS0O, XS1O, XRTO.The crossbar boards handle the switching of data, address and other control signals between the NSPs and NIA, which issue the requests, and the NMBs which service the requests. This document deals primarily with the NMB and its associated daughter board, the Neptune Memory Card (NMC).

An NMB has two independent, thirty two bit data ports. Each port receives requests from and returns read data to one of the two XS0, XS1, XRT triples. One port services even side requests, requests whose address has a zero in bit position two, while the other port services odd requests, requests whose address has a one in bit position two. The NMB can service up to one READ, WRITE or TEST-AND-MODIFY operation per clock per even or odd port.

An NMB can hold from 64 to 512 Megabytes (MB) of memory depending on the number and type of DRAMs installed. The DRAMs and some of the error correction logic physically resides on the thirty two NMCs installed on each NMB. The NMB holds these NMCs, all the logic for controlling the NMCs and the interface logic.

The NMB board is installed in a CPU/memory bay. There are up to four CPU/memory bays in a C38xx system, with each bay containing up to two NMB boards for a total of eight NMBs maximum.

The NMB board assembly consists of thirty two NMCs, the NMB logic board and the NMB power board. The power board receives 300 volts DC and transforms it in to -2.0, -4.5, and +5.0 volts DC.

1.3 Major Subsystems

An NMB contains thirty two banks of data, each of which resides on one of the thirty two NMCs. A bank is a block of memory which can be read or written independently from any other bank. Therefore, with thirty two banks, it is possible for thirty two different operations to be in progress

at any one time.

At the highest level, the NMB breaks down into an even and an odd side and some common logic. The common logic consists of the clock generation logic, the scan control logic, including the scan clock generation for the NMCs and the return data gate array (RDEDC).

The even and odd sides are logical duplicates of each other; that is, they have identical hardware and function. In many places in the document, only one side will be described with the description applying equally well to the other side. Each side consists of four bank control gate arrays (BCGA), sixteen NMCs, input data staging, write data staging and a sixteen to one return data multiplexor.

Each BCGA controls four banks. It generates all the DRAM control signals and all the datapath control signals for requests sent to any of its four banks. Each of the four banks can be independently controlled simultaneously. The BCGA itself, while controlling the data for a bank, does not actually pass data through itself. Data is registered external to the arrays in the input staging registers, the write data staging registers (one set per bank) and the gate array on the NMC.

Return data for each side is passed through a multiplexor which selects data from one of the sixteen banks and sends the data to the RDEDC gate array for error detection and correction. There is only a single RDEDC per NMB; it has an even and odd port for accepting data from both even and odd sides of the NMB.

The NMCs are approximately 1.9" by 8.3" printed circuit cards which contain all the ECL/TTL translation logic, all the DRAMs and the write data error detect and correct (WREDC) gate array. The WREDC holds write data for write operations and write and read data for read-modify-write operations. There is no control logic on the NMC or in the WREDC; all control signals are generated by the NMC's controlling BCGA.

The WREDCs on the NMCs are CMOS devices that can not operate at system clock frequencies. Since they contain state for logging errors on a bank, it is still necessary to scan them even though they can not be scanned individually at the system rate. There exists logic on the NMB for scanning the NMCs at one eighth the system clock rate and multiplexing the scan outs from the NMCs so that the NMCs, as a group, scan at the system clock rate.

1.4 Conventions

The fundamental cycle times for memory operations can be programmed on the NMB by scanning various values into certain fields of the NMB. These fields control how long the read, write and read-modify-write DRAM operations take. Since DRAM cycle times effect various interface timings, most notably when data is ready from the NMB, these times may change if the NMB is programmed differently from what this document assumes.

Times in this document will reflect the times the NMBs will initially be programmed for. These are also the values used in the simulation of the NMB. These values may be altered as a result of debug or the decision to use faster DRAMs. The text will note any cases were the timing may be different if the NMB programming has been altered.

In most of the figures displaying timing with respect to the RDY signal, the request is assumed to be a read or full write operation. Test-and-modify operations and partial writes have somewhat different timing.

Bit ranges are referred to as most significant bit, colon, least significant bit. For example, the thirty two bits of a data word would be DATA<31:0>.

All signals which go between boards are prefixed with a source board abbreviation and destination board abbreviation. For instance, the ready bit between the XSE and NMB is XSE_MB.RDY. Since XSE is listed first, the XSE is the source of this signal and the NMB is the destination.

In many of the figures, the leading source/destination prefixes have been omitted because there is insufficient room for them in the diagram. Generally, it should be obvious which signal is referenced even without the prefix.

The term "bank" is used frequently in this document. A bank is a group of memory which can be accessed independently of other banks. The NMB may have up to thirty-two different requests cycling at any one time: one request on each of the thirty-two banks.

1.5 Abbreviations

All boards in the C3 system have abbreviations by which they are referred to. Table 1-1 shows the board abbreviations for those boards mentioned in this document.

Table 1-1 Board Abbreviations

Abbreviation	Board
NMB	Neptune Memory Board (subject of this document)
NMC	Neptune Memory Card. Plugs into NMB
XS0E	Crossbar board, send data, even side, first of two send boards
XS1E	Crossbar board, send data, even side, second of two send boards
~~XSE~~	Refers to both XS0E and XS1E together Even side, send crossbar
XRTE	Crossbar board, return data, even side
XS0O, XS1O, XSO	Odd side counterparts to XS0E, XS1E and XSE
XRTO	Odd side return crossbar board
XCL	Crossbar Control Logic board. Source of scan control and other control signals.
NCU	Neptune Communications register and Utility board. Source of clocks to NMB.

A truncated board abbreviation is used when specifying board source or destination in signal names.Table 1-2 shows the abbreviations used in signal names,

Table 1-2 Source-Destination Board Abbreviations

Full Name	Signal Name Abbreviation
NMB	MB
NMC	MC
XS0E	XSE
XS1E	XSE
XS1O	XSO
XRTE	XRE
XRTO	XRO
XCL	XC
NCU	CU

For instance, a signal named FOO going from the NMB to the NMC would be called MB_MC.FOO. A signal going from the XS0E to the NMB named FOOBAR would be called XSE_MB.FOOBAR. Note that XS0E and XS1E have the same signal abbreviation.

The NMB schematics are grouped into logic blocks that are referred to throughout this document. These blocks are described in detail in Chapters 2 and 3. For reference purposes the names and a brief description of the blocks are given in Table 1-3.

Table 1-3 NMB Logic Blocks and Gate Arrays

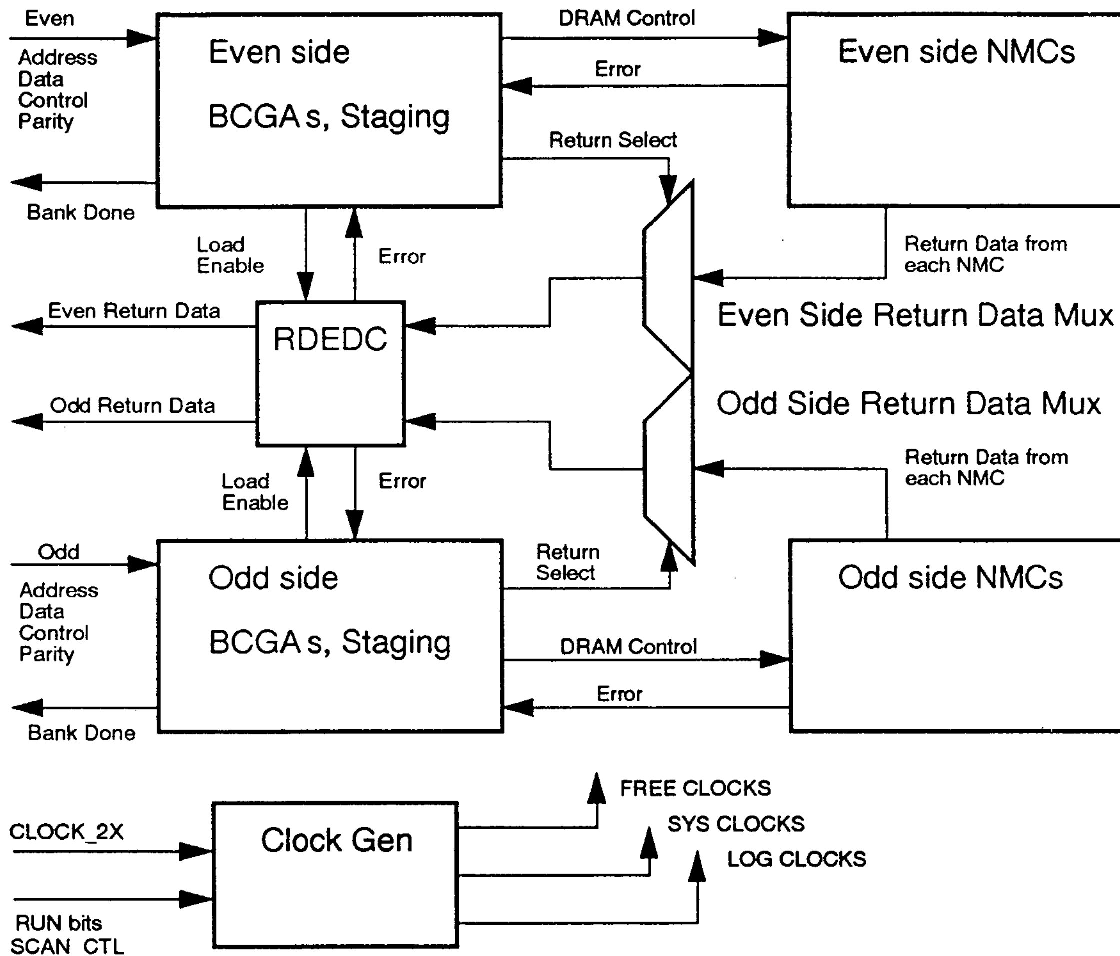
Name	Description
BCGA	Bank Control Gate Array, eight per board.
BC0E	BCGA for even side banks 0 to 3.
BC1E	BCGA for even side banks 4 to 7.
BC2E	BCGA for even side banks 8 to 11.
BC3E	BCGA for even side banks 12 to 15.
BC0O	BCGA for odd side banks 0 to 3.
BC1O	BCGA for odd side banks 4 to 7.
BC2O	BCGA for odd side banks 8 to 11.
BC3O	BCGA for odd side banks 12 to 15.
RDEDC	Return Data Error Detect and Correct gate array.
NMB_CLOCK	Clock buffering logic, error control.
ISE	Input Staging, Even side. Buffers write data, zone and some address for even side NMCs.
ISO	Input Staging, Odd side. Buffers write data, zone and some address for odd side NMCs.
READMUXE	Even side return data multiplexor. Selects data from even side NMCs for RDEDC.
READMUXO	Odd side return data multiplexor. Selects data from odd side NMCs for RDEDC.
LDREG	Decodes selects for REAMUXE and READMUXO. Stages RDEDC control signals.
MIM_CTL	NMC clock generation logic.

2 NMB Interfaces

The NMB board interfaces are presented in this chapter. The chapter starts by describing the interfaces to other boards, followed by interfaces between the major subsystems on the board.

Figure 2-1 shows the interfaces described in this chapter.

Figure 2-1 NMB Interfaces



2.1 External Interfaces

The NMB board services requests presented to it by the XS0E/XS1E and XS0O/XS1O boards. It returns data to the XRE and XRO boards. Control signals for scan and clock operation originate from the XCL board. The clock signals themselves come from the NCU board.

The crossbar boards are physically far enough away that all NMB interface signals only have sufficient time to propagate from their source to their destination and make setup to a register without encountering any levels of logic. As a general rule, all signal propagations between the

NMB and the crossbar boards are register to register. These registers may be in discrete parts on a board or within gate arrays.

The single exception to this rule is RD_DATA discussed in section 2.1.2 on page 15. It goes through one level of multiplexing on the XRE/XRO boards before being registered. See the XRT board specification for details.

2.1.1 XSE - Even Cross Bar Interface

The NMB/XSE interface is a simple one since the arbitration of requests from the different processors is handled by the XSE board. Furthermore, the XS0E keeps track of which banks on the NMB are busy and it is only allowed to send a request to a bank on the NMB when that bank is free. Therefore, the XSE selects the request for the NMB and knows when the NMB can handle the request. It need only inform the NMB that it has sent it a request. No return handshake from the NMB is required.

If the XSE does send a request to the NMB that it can not handle, the NMB will detect that condition and assert hard error.

In addition to sending requests to the even port of the NMB requests, the XS0E also initiates refresh cycles on the NMB when the memory system is in normal operation mode. There are some modes discussed later (section 2.1.5 on page 17) during which refreshes are initiated by a different signal.

All control, address and data signals from the XS0E and XS1E boards except for the RDY signal are protected by parity. Should a parity error be detected, the NMB will directly inform the XS0E and XS1E boards of this event so that they may capture hard error state necessary to determine the source of the error.

Table 2-1 shows all of the signals between the XS0E/XS1E boards and the NMB. Please refer to the XS0 and XS1 specifications to determine whether XSE refers to XS0E or XS1E. In general, WR_DATA, WR_PAR, CTL_PAR and WR_ZONE originate from the XS1E board. All other signals come from or go to the XS0E board unless explicitly defined otherwise.

2.1.1.1 Request Interface

On any clock during which RDY is asserted the NMB will process the starred signals Table 2-1 as a request to the NMB. All signals are registered on the same clock. The NMB can not refuse a

request so there is no return handshake from the NMB to the XSE; the request is simply assumed to be accepted.

Table 2-1 XSE: Even Cross Bar Send Interface Signals

XSE_MB.RDY	Request Ready *
XSE_MB.ADDR<28:3>	Request Address *
XSE_MB.CYCLE<1:0>	Request Type *
XSE_MB.WR_ZONE<3:0>	Bytes to Write for WRITE or TAM cycle *
XSE_MB.CTL_PAR<4:0>	Parity for address, cycle, write zone *
XSE_MB.WR_DATA<31:0>	Data for WRITES and TAMs *
XSE_MB.WR_PAR<3:0>	Parity for WR_DATA *
XSE_MB.REF_REQ	Refresh request from crossbar
MB_XSE.BANK_DONE<15:0>	Indicates a bank is finished with a request
MB_XS0E.SEND_PAR_ERR	Advance hard error signal
MB_XS1E.SEND_PAR_ERR	Advance hard error signal

Address bits are numbered from bit 28 through bit 3 to reflect their original position in the physical address word generated by the NSP or NIA. Bits 31 through 29 of the physical address are used in memory board select generation by the NSP and NIA and are therefore unnecessary at the memory board interface. Similarly, bit 2 of the physical address selects whether the address is to the even or odd port of memory. Bits 1 and 0 together with operand size are used by the processors to specify the zone bits for a write or TAM and are otherwise unnecessary at the NMB. Figure 2-2 shows the address bit mapping at the NMB

Figure 2-2 Address bit mapping at NMB

Address Bit			
28:27	26:25	24:7	6:3
Row Select	4 Meg Address	RAM Address	Bank Address

The Row select and 4 Meg DRAM address bits may not all be valid depending on what type of NMCs are put on the NMB. If the NMCs contain any 1 Meg DRAMs, the 4 Meg Address bits must be zero. If they are not, a hard error will be detected by the NMB. Similarly with the Row select bits, bit 28 can only be non-zero if the NMCs are fully populated (four rows), not half populated (two rows).

The Bank Address field determines which of the sixteen banks on the even side of the NMB the request is for. Since the Bank Address field is the lowest portion of the address, a sequential stream of addresses will access sequential banks.

The CYCLE bits define the operation to be performed by this request. Table 2-2 shows the meaning of each of the four possible CYCLE codes.

NOP stands for *no-operation.* When the NMB receives this operation, the bank selected by the ADDRESS will go busy for the number of clocks it takes to do a READ cycle, but no data will be modified and nothing except for a BANK_DONE will be returned to the crossbar.

Table 2-2 CYCLE Codes

CYCLE<1:0>	Operation
0	NOP
1	READ
2	WRITE
3	Test-And-Modify (TAM)

The READ operation is a request for data. The ADDRESS specifies memory bank and location. WR_DATA, WR_PAR, and WR_ZONE are ignored but are still checked for good parity. Memory data will be returned to the crossbar as described in section 2.1.2 on page 15.

The WRITE operation is a request to modify data in memory. All signals in the request are used, unlike the READ operation. The location to be written is specified by ADDRESS, the data to write by WR_DATA, and which of the bytes in the thirty two bit word to write by the WR_ZONE. Table 2-3 shows the correspondence between WR_ZONE and data byte. This mapping is identical to the parity mapping for data.

Table 2-3 WR_ZONE to Data Byte Mapping

WR_ZONE<0>	WR_DATA<31:24>
WR_ZONE<1>	WR_DATA<23:16>
WR_ZONE<2>	WR_DATA<15:8>
WR_ZONE<3>	WR_DATA<7:0>

Because of the error correcting code used by the memory system, data is maintained as thirty two bit words. Internal to the NMB, a write of less than a word is processed as a merge of the old word with the new bytes and then a write of the new thirty two bit word. This aspect of the NMB means that writes of complete thirty two bit words (full writes) and writes of less than a full word (partial writes) are treated differently by the NMB.

Note that WR_ZONE of all ones is a full write (all bytes in the word are to be changed to what is on WR_DATA). In this case, the write is processed by the NMB as fast as a READ operation, approximately fifteen clocks depending on how the NMB is programmed.

In all other cases of WR_ZONE, the write is only modifying some bytes of an existing thirty two bit word. In these cases, called a partial write, it is necessary to first read the existing word from memory, merge it with the write data and then write the new word into memory. This operation takes longer than the READ or full WRITE, approximately twenty six clocks.

The case of a WRITE with all WR_ZONE bits zero is a second special case. The NMB handles this write as it would any other write whose WR_ZONE bits are not all ones. However, since no bytes are specified to be written, this operation has the effect of simply reading the contents of Memory and then writing that same word back into memory. This operation is useful, though, since any existing soft error at the word will be corrected. This operation is therefore referred to as a *scrub cycle*. It is issued by the service processor to correct memory after the NMB has reported a soft error.

A Test-And-Modify (TAM) is an operation combining a READ and a WRITE. In practise, it is identical to a partial write with the addition that the word read from memory before it is modified by the write operation is returned on the return data interface described in section 2.1.2 on page 15. So for a TAM, data is read and returned to the requester, then it is modified with the WRITE data. The TAM is an *atomic* operation; it can not be divided by any event except stopping the NMB's free running clock. The TAM operation is used for *semaphores* which will not be discussed here.

The NMB supports TAMs with any number of WR_ZONE bits set. In practise, however, only a single bit will be set. Only byte test-and-modify operations are supported by the C Series architecture.

2.1.1.2 Parity Checking on Send Requests

All signals composing a request (the starred signals in Table 2-1) are protected by parity except for the RDY signal. Parity is even parity as defined by the Neptune Scan Package (an even number of ones in data gives a one for parity). A write data value of 0xffffff would then have a WR_PAR of all ones. Table 2-4 shows the parity bit/ data correspondence.

Parity is always checked regardless of the value of RDY.

Table 2-4 Request Parity

Parity Signal	Corresponding Data
XSE_MB.WR_PAR<3>	XSE_MB.WR_DATA<7:0>
XSE_MB.WR_PAR<2>	XSE_MB.WR_DATA<15:8>
XSE_MB.WR_PAR<1>	XSE_MB.WR_DATA<23:16>
XSE_MB.WR_PAR<0>	XSE_MB.WR_DATA<31:24>
XSE_MB.CTL_PAR<4>	XSE_MB.WR_ZONE<3:0>
XSE_MB.CTL_PAR<3>	XSE_MB.ADDR<7:3>, CYCLE<1:0>
XSE_MB.CTL_PAR<2>	XSE_MB.ADDR<14:8>
XSE_MB.CTL_PAR<1>	XSE_MB.ADDR<21:15>
XSE_MB.CTL_PAR<0>	XSE_MB.ADDR<28:22>

If a request is determined to have bad parity, the NMB will of course assert hard error. It will also assert the MB_XS0E.SEND_PAR_ERR and MB_XS1E.SEND_PAR_ERR signals. HARD_ERROR (discussed in section 2.1.4 on page 16) will be asserted on the same clock or one clock later. It is asserted on the same clock if the hard error was in WR_DATA, WR_ZONE or ADDR<28:22>, otherwise it is asserted one clock after the SEND_PAR_ERR signals.

The XSE can not check parity because of physical parts constraints. Instead, it keeps a copy of what it sent to the NMB for three clocks so that if the NMB tells it that there was a parity error on the request, it can freeze its registers and preserve what the data was when it was in the crossbar. By examining the data in the crossbar and at the NMB, it is possible to determine whether the parity error occurred between the XSE and the NMB or between the XSE and the request originator. Because there is a limited number of registers available in the XSE for storing this

information, the SEND_PAR_ERR signals which freeze this information must be quicker than the standard system hard error.

Figure 2-3 Parity Error Detect Timing

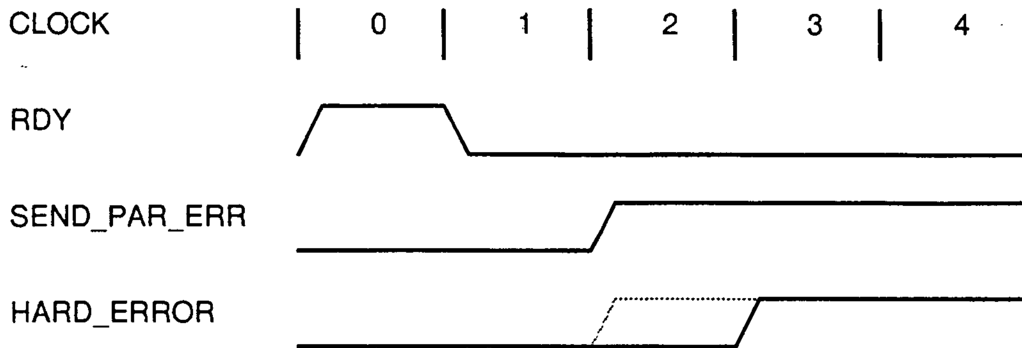


Figure 2-3 shows the timing between the RDY signal and the resulting SEND_PAR_ERR and HARD_ERROR assertions for a parity error detect. SEND_PAR_ERROR goes high two clocks after the RDY with HARD_ERROR following on the same clock or next clock depending on the type of error. The timing shown in this figure is invariant with NMB programming.

2.1.1.3 BANK_DONE

The BANK_DONE signals tell the XSE (specifically, the XS0E) that a bank is no longer busy; that is, it is done. There is one BANK_DONE signal per bank.

Since the XSE/NMB interface has no arbitration, it is assumed that the bank being requested is free. The XSE uses the BANK_DONE to note when a bank frees up and thus is ready to accept a new request.

BANK_DONE will follow RDY by a certain number of clocks depending on how the BCGAs are programmed. This will probably be eleven clocks after a RDY (this is the time the BCGAs will initially be programmed for). The BANK_DONE assertion actually precedes the true completion of the request at the bank so that no bandwidth is wasted: the BANK_DONE precedes the actual bank completion by enough clocks so that the next request can be issued to the bank immediately after the bank controller finishes with the previous request. The number of clocks the BANK_DONE precedes the actual completion at the bank controller takes into account the number of registers between BANK_DONE and the XSE issuing another request and the number of registers in the NMB between RDY and actual start of a bank.

Figure 2-4 Request Timing to the Same Bank

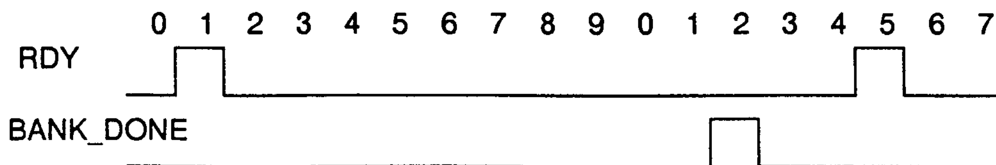


Figure 2-4 shows the relationship between multiple RDYs to the same bank and the BANK_DONE from that bank. In the figure, eleven clocks elapse between RDY and the BANK_DONE. This figure may change slightly if the BCGA is reprogrammed during debug or for faster DRAMs. The relative timing between the BANK_DONE and the earliest allowable next RDY to that bank, three clocks, is hard coded into the registers stages of the XARB and will not change.

2.1.1.4 Refreshes from the XSE

Refreshes are a requirement of Dynamic RAMs (DRAMs). They must be periodically put through a special operation to avoid loss of data. The DRAMs used on the NMB must be refreshed 512 times per 8 milliseconds which is once every 15.3 microseconds.

During normal operation (see section 2.1.5 on page 17 for other modes of operation), refreshes for the even side banks originate from the XS0E board. The signal XSE_MB.REF_REQ is used to initiate the refresh.

When the REF_REQ signal is asserted, each of the even side bank controllers in the even side BCGAs will either immediately initiate a refresh cycle if the bank is idle or begin a refresh cycle immediately after finishing a request in progress. The refresh initiate does not use any of the request signals (RDY, ADDRESS, ZONE, etc.). In fact, it is possible for the XSE to be asserting a REF_REQ and RDY simultaneously. Addresses for refresh cycles are internally generated by the NMB so the ADDRESS bus is not needed for the refresh.

The NMB does not generate the REF_REQ signal internally because the XSE must keep track of the busy status of each of the NMB banks. Since a refresh cycle causes a bank to go busy to service the refresh, the XSE must treat the REF_REQ event as a event which causes all banks on the even side to go busy. A second reason for not having refreshes internally initiated by the NMB is discussed in section 2.1.5.6 on page 22.

Figure 2-5 Refresh, RDY and BANK_DONE

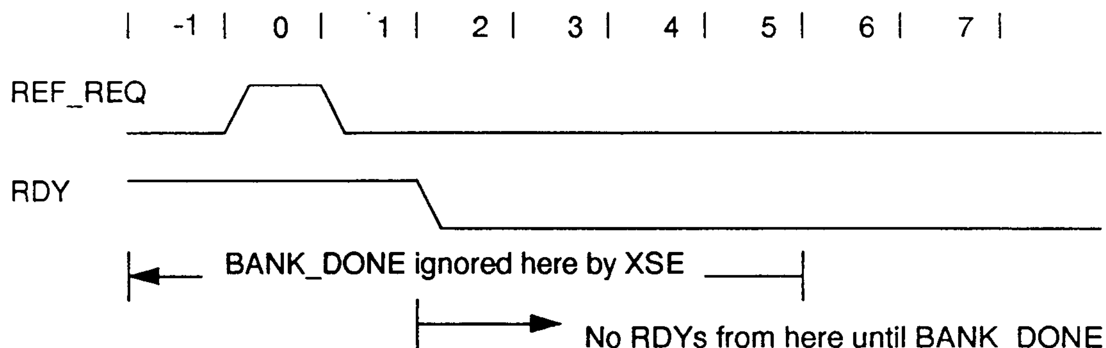


Figure 2-5 shows the relationship between REF_REQ, RDY and the BANK_DONE signals. RDY is valid through the clock after a REF_REQ. After the second clock after a REF_REQ, the XSE has marked all even banks busy because of the refresh and it will therefore not send any bank a request until the bank desired has asserted a BANK_DONE signal. So, from clock number two on, XSE will not be generating RDYs until the bank notifies the XSE that it is no longer busy by asserting BANK_DONE.

If REF_REQ causes a bank to register a refresh request while a bank controller is still processing a request, the bank controller suppresses the BANK_DONE that would have been issued when the bank finished with its current request. Instead, it only issues a single BANK_DONE after both the request *and* the refresh are complete.

Figure 2-6 Example of Suppressed BANK_DONE Because of Refresh

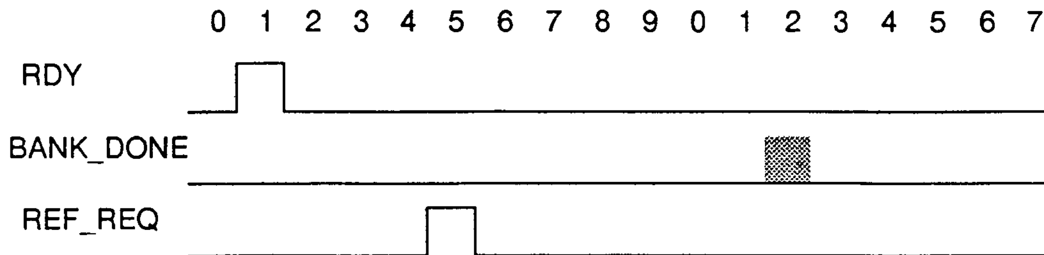


Figure 2-6 is an example of a suppressed BANK_DONE. If there had been no REF_REQ, a BANK_DONE would have been issued at clock 12. However, the bank controller noted the refresh and suppressed this BANK_DONE. Instead a BANK_DONE will be issued after both the original request and the refresh complete, at clock 26 if the request made at clock 1 is a full write or a read.

Of course, the bank control can only suppress the BANK_DONE for a request once it has been informed of the existence of a refresh. Because of pipeline stages in the BCGAs and on the NMB on the refresh initiate signal, a number of clocks occur before the refresh is registered by the bank controller. During this time, the bank controller will still issue BANK_DONEs for requests completing shortly after the REF_REQ was issued since it does not yet know about the refresh. The XSE must therefore ignore BANK_DONEs issued during clocks -1 through 5 in Figure 2-5 to avoid marking a bank as no longer busy prematurely.

Figure 2-7 Example of Simultaneous Refresh and BANK_DONE

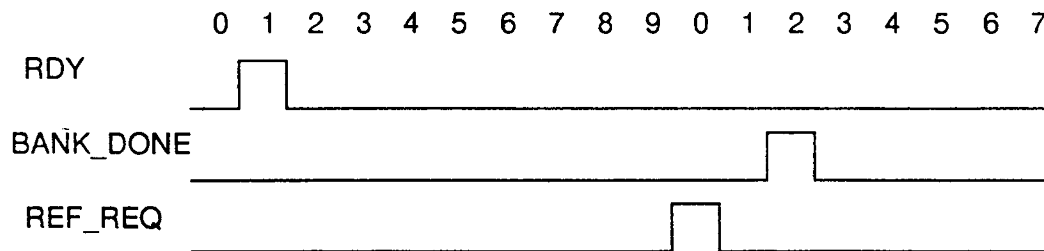


Figure 2-7 is an example of a BANK_DONE that must be ignored by the XSE. At clock 1, a request is issued to some bank, say bank 0. Eleven clocks later bank 0 will issue a BANK_DONE for that request. If the XSE also issues a REF_REQ at clock 10, the bank 0 controller will not yet have seen the refresh and will still issue the BANK_DONE for bank 0. If the XSE did not ignore the BANK_DONE at clock 12, it would mark the bank as unbusy and potentially issue a new request to bank zero while bank 0 was busy with the refresh. This would cause a hard error since a bank controller can not service a request while it is doing a refresh. (It *can* queue a refresh while it is busy with a request as was discussed in Figure 2-6.)

2.1.2 XRE- Even Cross Bar Return Interface

The return data interface to the XRE is a very simple interface. Table 2-5 shows the three signals composing this interface. RD_DATA is the return data resulting from a READ or TAM request, the full thirty two bit word as specified by the ADDRESS passed with the RDY for this request regardless of WR_ZONE bits. That is, even if WR_ZONE was set to something other than all ones, a full thirty two bit word is returned. A single byte can not be requested from the NMB. RD_PAR is mapped similarly to WR_PAR for WR_DATA as shown in Table 2-4.

Table 2-5 XRE: Even Crossbar Return Interface Signals

MB_XRE.RD_DATA<31:0>	READ Return Data Bus
MB_XRE.RD_PAR<3:0>	READ Return Parity
MB_XRE.RD_RDY	READ Return Data Ready

The RD_RDY signal is asserted when valid return data is present on RD_DATA. Since the NMB always returns RD_DATA a fixed number of clocks after RDY (thirteen clocks as currently programmed), the crossbar can and does predict when return data is expected. It is therefore unnecessary, functionally, for RD_RDY to exist. However, RD_RDY is only a single bit and its presence provides some extra error checking: if RD_RDY is asserted by the NMB when the XRE does not expect a RD_RDY, the XRE will assert a hard error. RD_RDY is also useful when debugging NMBs/XREs.

Figure 2-8 RDY to RD_RDY Timing

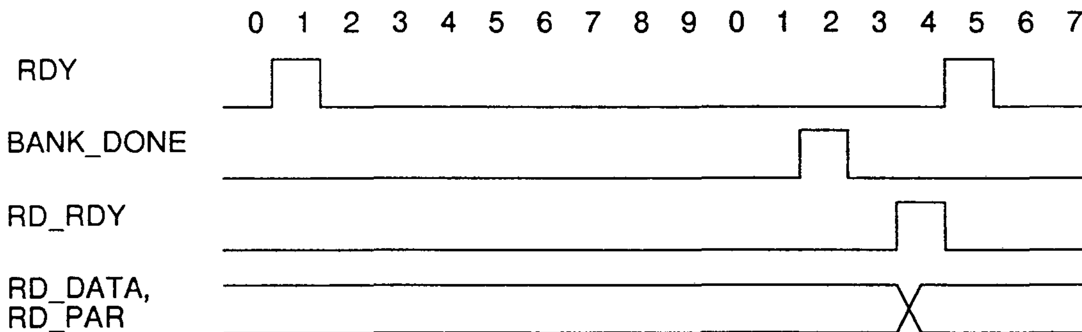


Figure 2-8 shows the RDY to RD_RDY timing for a READ request. Return data timing is identical for a TAM request; however, the BANK_DONE will occur at clock 21 rather than clock 12 for a TAM. As shown in the figure, RD_DATA changes simultaneously with RD_RDY. RD_DATA only changes when new return data is ready. Thus, in the example, it only changes once at clock 14 since only one request was issued at this time. Had there been other request (to other banks, of course), RD_DATA would have changed each time return data from a bank was ready.

The RDY to BANK_DONE, RDY to RD_RDY and RDY to RD_DATA timing is dependent on how the NMBs are programmed. The BANK_DONE to RDY timing is invariant.

2.1.3 XSO and XRO Interfaces

The XSO and XRO interfaces at the NMB are identical to the XSE and XRE interfaces. See section 2.1.1 on page 8 and section 2.1.2 on page 15 for details.

2.1.4 Hard and Soft Errors

The NMB detects a number of fatal, non-recoverable errors called hard errors. It also detects recoverable errors known as soft errors. For both types of errors, the NMB will freeze all necessary state and inform the XCL board of the error. ~~The XCL will then forward the error to the service processor as an interrupt as well as stop system clocks in the case of a hard error.~~ System operation is not interrupted in the case of a soft error.

Table 2-6 shows the error reporting signals. The CAST error signals are for the CAST test fixture. They are asserted for a single clock simultaneous with RD_RDY if any single (soft) or multi bit (hard) error is detected for the data being returned with the RD_RDY. They are not connected to anything in normal system operation.

Table 2-6 Error Signals

Signal	Description
MB_XC.HARD_ERROR	Non-recoverable error
MB_XC.SOFT_ERROR	Recoverable error
M_CST.CERR_E	CAST even side error
M_CST.CERR_O	CAST odd side error

All hard and soft error events on the NMB are triggered by requests. Table 2-7 and Table 2-8 show the number of clocks elapsing from RDY to error signal report. Many of these times are dependent on BCGA programming and are marked with an asterisk.

Table 2-7 RDY to HARD_ERROR Timing

#	Error	# of Clocks to HARD_ERROR
1	Parity Error on Input to Board	2(3)
2	RDY to Invalid Address	3
3	RDY to Busy Bank	4
4	NMC Write Data Parity Error	11
5	Full Read or TAM Multi-bit Error	17*
6	Partial Write Multi-bit Error	38*

Table 2-7 shows the number clocks elapsing from the RDY which initiates a request which will lead to a hard error and the actual assertion of the HARD_ERROR signal. Error number one in the table is a result of bad parity on the request signals (ADDRESS, CYCLE, WR_ZONE, WR_DATA). If a parity error is detected on these signals, the SEND_PAR_ERR signal will also be asserted as discussed in section 2.1.1.2 on page 11. It takes three clocks to generate HARD_ERROR when the error is in ADDRESS<21:3> or the CYCLE bits; otherwise it takes only two clocks.

Error number two is generated when a request has an illegal address. The BCGAs are programmed via scan to check certain address bits depending on what type of memory is installed on the NMB (Figure 2-2). If One Megabit DRAMS are installed and address bits 25 or 26 are set a hard error will be generated. Similarly, if two row (half populated) NMCs are installed and bit 28 is set, a hard error will be generated.

Error number three occurs when the XSE sends a request to a bank that is already busy with a previous request or refresh. Error number four occurs when there is an internal error on the NMB

which corrupts data being staged between the NMB and its daughter card, the NMC. These two errors will generally indicate a manufacturing flaw or noise problem.

Errors five and six result when a word is read from memory that has two or more bits corrupted. This error most often indicates that memory has not been properly initialized.

Table 2-8 RDY to SOFT_ERROR Timing

#	Error	# of Clocks to SOFT_ERROR
1	Full Read or TAM Single Bit Error	17*
2	Partial Write Single Bit Error	18*

Table 2-8 shows the number of clocks elapsing between RDY and SOFT_ERROR assertion. Soft errors only occur when a memory word is found with one bit incorrect. The NMB will return correct data but the service processor must be informed of the error so that it can fix the bit in the memory word and log the error so that a persistent soft error may be corrected by replacing hardware. In some cases, the NMB will fix the corrupted bit itself but the service processor still needs to be informed of the error for logging purposes.

Error number one in Table 2-8 occurs whenever a READ or TAM operation references a data word with one bad bit. Error number two occurs during writes that do not write an entire thirty two bit word. To do a partial write, the NMB must first read the full thirty two bit word which is to be modified; it is when reading this word that it may encounter a soft error.

Once asserted each error signal except for the CAST signals remains asserted until the error state is reset by a scan operation. See section 2.1.5 on page 17 for a discussion of the various scan operations possible.

2.1.5 Clock and Scan Interface

The NMB has a number of scan and clock modes controlled by the signals listed in Table 2-9.

Table 2-9 Scan and Clock Signals

Signal	Description
XC_MB.SCAN_CTL<2:0>	Scan mode select
XC_MB.SCAN_IN	Scan Ring Input
XC_MB.SCAN_OUT	Scan Ring Output
-XC_MB.SYS_RUN	SYS Clock Run Enable, Active Low called -RUN_SYS for short.
-XC_MB.LOG_RUN	LOG Clock Run Enable, Active Low called -RUN_LOG for short.
XC_MB.RAM_RFSH	Raw Refresh
CU_MB.CLOCK_2X	2X system clock
-CU_MB.CLOCK_2X	Complement of Previous Signal

The NMB receives a double frequency clock (CLOCK_2X). From this clock, both double frequency (2X) and system frequency (1X) clocks are derived. Only some of the internal registers dealing with the DRAMs use 2X clocks. All remaining logic and all interface logic is registered off of the

1X clocks. During *all scan* mode (see section 2.1.5.3 on page 21), CLOCK_2X actually runs at a 1X rate. In all other cases it runs at the 2X system clock rate.

The NMB has multiple modes of operation due to the nature of the DRAMs used as main memory in the Neptune system. DRAMs are prone to occasional soft errors. These are errors caused by background radiation or similar mechanisms that cause a bit of memory to become corrupted. The DRAM has not actually failed; once the bit's value is corrected, it will function normally. These errors happen frequently enough that the NMB has been designed with an error correcting code (ECC) that allows a single bit error in a thirty two bit word to be corrected. So, even if a bit goes bad, it is possible to determine what it should have been and, therefore, to proceed with out halting the machine. Two bit errors and some three and four bit errors can be detected but not corrected. These errors are extremely rare and will cause the machine to be halted.

Another feature of DRAMs, the one responsible for the 'D' in DRAM, is that they are dynamic. In return for very high capacity storage, a single transistor structure must be used as a storage element. This element is basically a leaky capacitor. If it is not recharged periodically, it will leak to the point where it is impossible to tell if it was a one or a zero. This recharge operation is known as refresh.

Correctable errors and refresh require that the NMB be treated differently than most other boards in the system (the NIA also has soft errors but for different reasons). Correctable errors require a logging procedure so that they can be recorded and, for certain ones, corrected by the service processor (some soft errors are automatically fixed by the NMB without service processor intervention). This soft error log must be accessible without interfering with normal system operation. Therefore, there exists a set of state on the NMB which can be examined without interrupting normal requests. This is the log state discussed in section 2.1.5.1 on page 19

When system clocks are stopped (for a hard error, debug or system initialization), the refresh requirements of DRAMs requires that some clocks on the NMB remain running so that refresh may be performed. Therefore, the NMB registers are divided into those clocks that remain running for refresh and those clocks that may be examined without effecting refresh. The clocks that stop with the system clocks are known as SYS Clocks. Operations involving these clocks are described in section 2.1.5.2 on page 20. Remaining state is described in section 2.1.5.3 on page 21.

Table 2-10 SCAN_CTL Codes

SCAN CTL	Operation	Section
0	Normal Mode	
1	Log Scan	section 2.1.5.1 on page 19
2	Unused	
3	Sys Scan	section 2.1.5.2 on page 20
4	NMC Normal Clock	section 2.1.5.3 on page 21
5	NMC Scan Clock	section 2.1.5.3 on page 21
6	CAST Load	section 2.1.5.4 on page 21
7	ALL Scan	section 2.1.5.3 on page 21

The various NMB modes of operation are controlled by the SCAN_CTL bits from the XCL. Table 2-10 lists the sections in which each of the various modes are discussed.

2.1.5.1 Log Scan Operation

When the NMB detects a soft error (a correctable error) it logs the address, type of error and corrupted data, returns corrected data to the XRE/XRO and continues its normal operation. The state in which this information is logged is known, simply enough, as log state.

The log state may be scanned by the service processor without effecting NMB operation in any way except for preventing further soft errors from being logged while the log state is being scanned. Soft errors will still be corrected, however.

SCAN_CTL and -RUN_LOG are used to perform the scan of log state. Simply asserting "1" on the SCAN_CTLs will cause the log state to go into scan mode, but because of timing constraints, not all log state will be scanning by the next clock and some log state will be lost or corrupted. To do a log scan, the clocks going to the log state must first be stopped, then SCAN_CTL changed to mode "1", then the log clocks restarted after SCAN_CTL has had time to propagate. When scan is finished, the clocks are again stopped, SCAN_CTL is then put back into normal mode, and then normal operation for log state continues.

All C38XX boards require clocks to be stopped while SCAN_CTL is changing. Since the NMB must be log scanned while performing its other functions, stopping CU_MB.CLOCK_2X is out of the question. Instead, -RUN_LOG is used to gate the NMB log clocks. When -RUN_LOG is low, log clocks are uninhibited. When -RUN_LOG goes high, log clocks stop.

Figure 2-9 Entering Log Scan

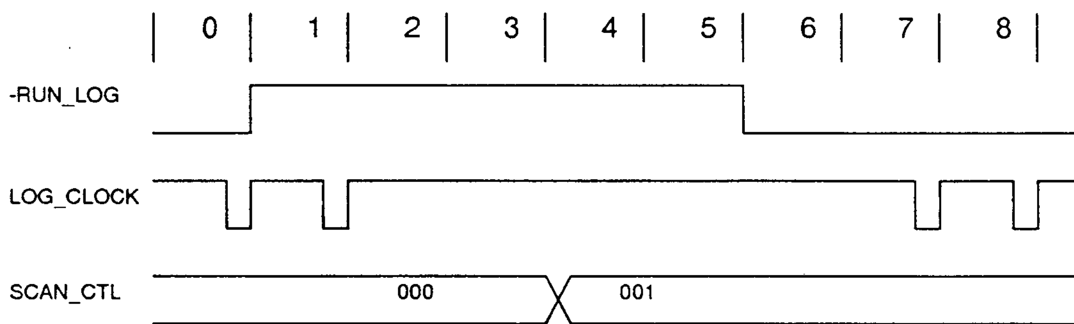


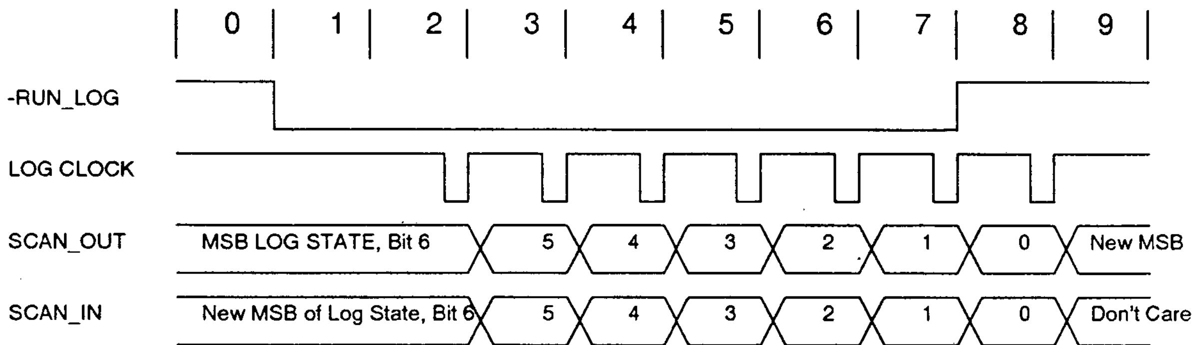
Figure 2-9 is an abbreviated diagram of RUN_LOG and SCAN_CTL being used to enter log scan. In actual system operation, the time from RUN_LOG going high to SCAN_CTL changing will take many more clocks than shown. The re-assertion of RUN_LOG after SCAN_CTL changes will also take more clocks. The actual number of clocks is dependent on the service processor.

In Figure 2-9, LOG_CLOCK represents a generic clock on the NMB clocking log state registers. In normal operation, LOG_CLOCK is a normal, 1X system frequency clock. When RUN_LOG is de-asserted, LOG_CLOCK has one more rising edge, then stops. It does not re-start until the clock following the re-assertion of RUN_LOG. While RUN_LOG was high and the log clocks are stopped, the SCAN_CTL field is changed from 0 to 1. There must be at least three clocks between changing SCAN_CTL and de-asserting RUN_LOG to allow sufficient time for the SCAN_CTL signal to propagate through the NMB. The opposite transition when SCAN_CTL is set back to zero is the reverse of the procedure shown in the figure.

As can be seen from Table 2-10, the SCAN_CTL signal controls scan modes other than log scan. These modes also require several clocks for SCAN_CTL to propagate in order for registers to be properly clocked. If the SCAN_CTL lines were to glitch to some value other than one or zero when it was changing to a one, non-log state registers on the NMB might be forced momentarily into a scan mode which would destroy the NMB state and cause a system crash and loss of memory contents. It is therefore necessary for SCAN_CTL to transition from zero to one without going to any intermediate values. The logic on the NMB that decodes the various modes from SCAN_CTL does so in a manner that prevents momentary glitches as long as SCAN_CTL switches in a glitch free manner.

All scan modes including log scan use the common SCAN_IN/SCAN_OUT signals to and from the XCL to pass scan data. Figure 2-10 shows the various signals needed for a log scan. In this example, the log ring is taken to be only six bits long; it is really about 3200 bits long.

Figure 2-10 SCAN_IN/SCAN_OUT in Log Scan



In Figure 2-10, SCAN_CTL has already been set to the log scan mode before RUN_LOG has gone active at clock one. Note the lag between assertion and de-assertion of RUN_LOG and the actual LOG_CLOCK. Scan data must be synchronized with the NMB log clocks not with RUN_LOG.

2.1.5.2 System Scan Operation

The second scan mode is called sys scan. In operation, sys scan is nearly identical to log scan. The state scanned during a sys scan is the log state plus all the registers which interface with the crossbar, including all hard error state.

Unlike log scan, sys scan is not transparent to normal system operation. The NMB can not service requests when its interface logic is stopped or being scanned. Sys scan will typically be used to examine hard error state or track a request into the NMB.

Sys scan operation is identical to log scan with the addition of the -RUN_SYS signal. This signal is identical in function to the -RUN_LOG signal except that it controls the sys clocks on the NMB. RUN_SYS timing is identical to RUN_LOG timing; LOG_CLOCK timing is identical to SYS_CLOCK timing during sys scan. During sys scan both RUN_LOG and SYS_LOG are asserted and de-asserted together.

During sys scan and setup for sys scan, the NMB's CLOCK_2X keeps running allowing the bank controllers to keep running and keep processing refreshes.

2.1.5.3 Complete Board Scan

A final scan mode exists where all scannable state on the memory board can be scanned. This state includes the registers which control the DRAMs and perform the refreshes so *all scan*, as complete board scan is called, will destroy the contents of memory.

There is a state machine on the NMB which handles generating the clocks for the NMBs. For the previous two scan modes, this machine required no special treatment since it runs on the free running clocks on the memory board. In the case of an *all scan*, CLOCK_2X is stopped prior to and after the scan. The state machine, since it is not receiving clocks at this time, can not function normally. Instead, scan modes four and five are used to force the NMC clock generator into and out of scan. This state machine is initialized for scan operation by SCAN_CTL mode four and forced out of scan operation by SCAN_CTL mode five. This operation effects only the state of the NMC clock generator scan machine. It has no other effect; all other registers on the NMB are held during mode four and mode five. The clock generator is discussed in section 3.5.3 on page 94 of the Functional Description chapter.

The procedure for scanning all the registers on the NMB is to stop NMB clocks, change scan controls to mode four, give the board a single clock, switch to scan mode seven, issue as many clocks as there are bits in the scan ring, stop clocks, switch to scan mode five, issue a single clock, switch to scan mode zero, resume operation.

Figure 2-11 Entering ALL Scan

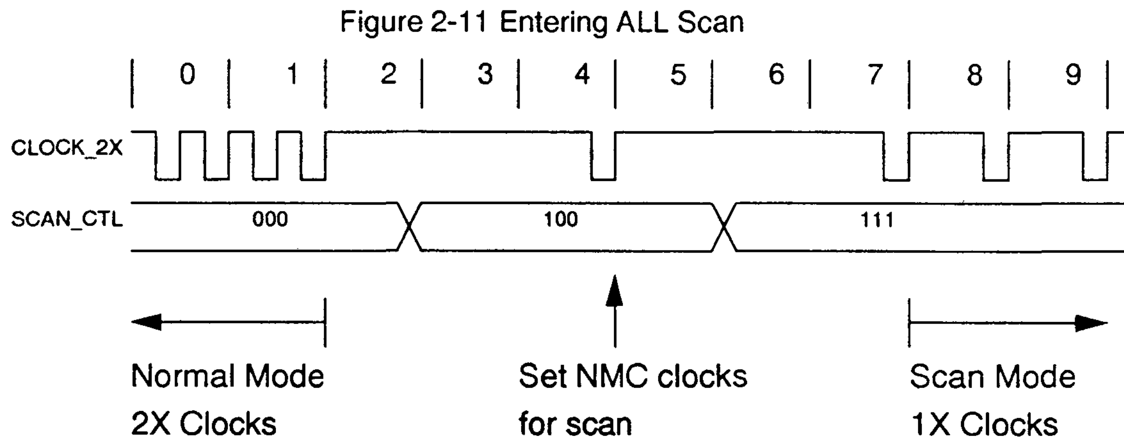


Figure 2-11 shows the sequence necessary to enter *all scan* mode. In real system operation, the number of clocks between SCAN_CTL switches will likely be much greater than what is depicted. Note that once actual scan operations begin, the NMB clock CLOCK_2X is at a 1X rate as opposed to the 2x rate during normal operation. The process for exiting *all scan* mode is the reverse of the given sequence except that mode five is used which sets the NMC clock generator for normal mode (non-scan) clocks.

2.1.5.4 CAST Load

CAST load mode is similar to normal mode except most clock gating operations are inhibited; all registers are forced to clock. The only gating which still occurs is the gating by which the 2x clocks are turned into 1x clocks.

This mode is only used in CAST and SST, the two scan based test packages.

2.1.5.5 SCAN_CTL Mode Transitions

The SCAN_CTL signals effect the gating of all registers on the NMB. Two of these modes, log scan and sys scan, are to be used while CLOCK_2X is still active. Since other clocks are active, momentarily transitioning to a SCAN_CTL mode which effects these registers would corrupt the state of the NMB. Therefore, when turning on log or sys scan, the SCAN_CTL signals must transition to that particular scan mode and no other. In the case of going into SCAN_CTL mode 3, it is permissible to transition through mode 1 or 2 first, since mode 1 effects only log state that is already stopped and mode two effects only sys state.

It is intended that SCAN_CTL only change to modes 4, 5, 6, or 7 (most significant bit set) when CLOCK_2X is not running. In these cases, with the clocks off, it is impossible to corrupt state and the SCAN_CTL signals may change in any manner.

For reasons to be discussed in section 2.1.5.7 on page 23, the free running state on the NMB (that state not stopped by RUN_SYS or RUN_LOG) needs to know when log scan or sys scan is taking place. This means that SCAN_CTL must be registered by the NMB. Therefore the SCAN_CTL going to the NMB is generated on a clock edge boundary and routed so that it can safely meet setup requirements on the NMB. The NMB, internal to itself, then uses both registered and unregistered versions of the SCAN_CTL for various reasons to be discussed in later chapters. For most boards, SCAN_CTL is not generated in such a way that it may be safely registered because it is unnecessary to constrain the SCAN_CTL timing in that way since, in general, clocks are stopped when SCAN_CTL is changed.

2.1.5.6 Refresh

SCAN_CTL modes 5, 6, and 7 effect all the state on the NMB. When these modes are active, or going to be active, the NMB can not perform any of its DRAM refreshing functions and DRAM contents will be lost. For all other modes, it is necessary to maintain refresh so that memory can be examined.

For instance, when a hard error is detected in the system, the system operator may want to examine the NMB's sys state with a sys scan as well as examine memory locations in order to track down the cause of the hard error. At the very least, the hard error will de-assert RUN_SYS as well as the other board clocks to freeze the state of the system. Having memory be corrupted by a system halt would be unacceptable since the operator would then be unable to dump the contents of memory for the crash dump.

The NMB, therefore, performs refreshes under SCAN_CTL modes 0, 1, 2, and 3. It also performs refreshes regardless of the state of RUN_SYS or RUN_LOG as long as the most significant bit of SCAN_CTL is zero. For scan modes 0 and 1 and when RUN_LOG is de-asserted, the normal refresh operation as described in section 2.1.1.4 on page 13 is still usable since log state is simply error logging state and has no effect on refresh.

When RUN_SYS is de-asserted or the sys scan mode is active, the hardware that normally initiates the refresh is either turned off or being scanned. In either case, it is unavailable for refreshes. Instead, the NMB must use a different mechanism to initiate refreshes.

When the NMB detects that RUN_SYS is not active or the sys state is being scanned, it looks to the RAM_RFSH signal from the XCL for the cue to initiate refreshes. RAM_RFSH is simply a copy of the signal that the XCL sends to the XSE telling it to send a refresh to the NMBs. When the

system clocks are stopped, the XSE is not running and can not initiate the refresh. Instead, RAM_RFSH is used to start refresh.

Refreshes initiated by RAM_RFSH while sys clocks are stopped do not cause BANK_DONEs since BANK_DONES are for the XSE and the XSE is not turned on in this case. In all other ways, RAM_RFSH refreshes are identical to XSE initiated refreshes. There is only a single RAM_RFSH line which starts refreshes for both even and odd sides of the NMB.

To avoid the case where the NMB is busy with a refresh when the sys state is re-started (RUN_SYS is re-asserted), sys clocks are re-started only following the completion of a refresh cycle on the NMBs. Logic on the NCU board insures that this constraint is met.

2.1.5.7 Memory System Single Step

During system debug, it is often desirable to stop clocks in order to examine scan state (on any of the boards) or to examine memory. DRAMs, however, can not be stopped in the middle of a read or write; they must be allowed to complete their operations even if the CPUs and crossbar are stopped. Also, as discussed earlier, they require refresh operations.

On the C2 memory boards (MCM), when the system was stopped, the MCM would complete its memory operations and queue up any return data. When clocks restarted, it returned its queued up data for the CPUs. Since the C2 memory function was completely self contained (as opposed to being split between the crossbar and NMB as in C3), it could internally keep track of whether banks were busy and where read data was. In addition, the MCM board had a return data queue whose Neptune analog is on the NSP and NIA.

A side effect of the MCM memory system architecture was that if clocks were stopped during a read request, the request would complete while the clocks were stopped and be available to the requesting processor immediately upon re-start of the clocks. It would appear to the user that the data returned in only two clocks rather than the normal eight clocks that returns usually take. This meant that when single stepping the CPU, data would not return at the same relative clock as when it was free running. It was more difficult to debug the CPU because the machine behaved differently when it was being single stepped versus normally run.

When Neptune clocks are stopped (except for the NMB free running clocks), the crossbar arbitration function which keeps track of busy banks stops as well as the return data queue on the NSPs and NIA. If the NMB were allowed to continue in its normal way when the system clocks were stopped, it would be dropping BANK_DONES and return data on the floor. When clocks restarted, the BANK_DONES and return data that the crossbar and processors would be expecting would never arrive since they occurred while clocks were stopped.

The NMB, therefore, needs a mechanism to detect that clocks have stopped, keep track of any data and signals that should go to the crossbar and replay this information at the appropriate time once clocks restart. The goal of the NMB is to make clock stops completely transparent to the return of data and operation of the memory system. A sequence of code should perform identically whether it was run at once or single step executed.

The operation of this logic on the NMB, called single step logic or SST in the BCGA schematics, is relatively straightforward. When system clocks stop, the NMB allows the requests in progress to be completed, it notes how many clocks elapsed between BANK_DONEs and READ_RDYs and the cessation of clocks and then when clocks are restarted, it waits the appropriate number of clocks to send the BANK_DONEs and RD_RDYs. The read data itself is held in latches on the

NMCs and selected to be returned to the XRE and XRO at the proper time. The process is basically a matter of counting the number of clocks after clock stop that an event occurs and then counting down that many clocks after clocks restart to redo that event.

The internal details of this process will be discussed in the appropriate later chapters. At the NMB interface, it simply looks like the NMB is being stopped with the rest of the system. Internally, the NMB is processing outstanding requests and then continues to do refreshes. Externally, it appears that the NMB has stopped along with the rest of the system.

For this process of NMB single step to function properly, the NMB must have its RUN_LOG and RUN_SYS bits de-asserted synchronously with the cessation of the other clocks in the system. The NMB's CLOCK_2X does not stop since the NMB must actually keep running to do its refreshes and complete any outstanding requests, but its interface registers gated by RUN_SYS must stop with the rest of the system.

Figure 2-12 NMB SYS Clocks at Clock Stop

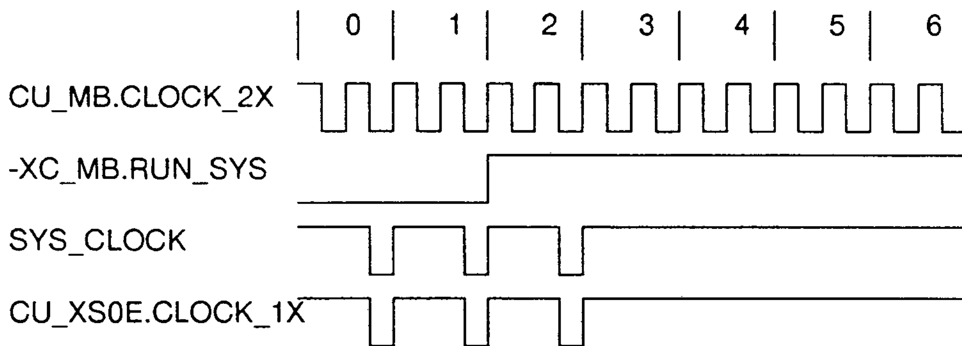


Figure 2-12 shows the synchronization between RUN_SYS and the clocks to the other board. As an example the clock to the XSE has been included. When the system is stopped, all clocks to the XS0E and all the NSPs, NVPs and other crossbar boards will be halted simultaneously. As can be seen from the figure, RUN_SYS will be asserted one clock prior to the clock stop so that the NMB's interface clocks, called SYS_CLOCK in the figure, stop simultaneously with the other

clocks in the system. Note that the NMB's clock, CU_MB.CLOCK_2X, keeps running. The reverse case in which the clocks are being restarted works similarly.

Figure 2-13 Requests Across Clock Stop

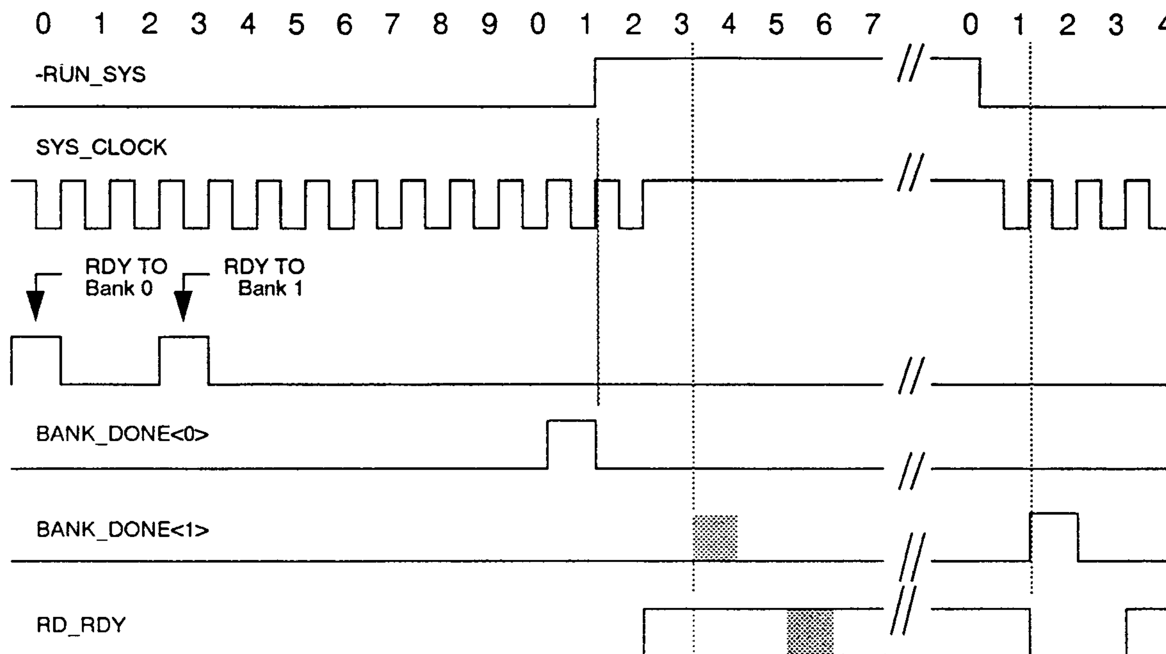


Figure 2-13 is an example of clocks stopping in the middle of two requests being serviced. Both requests are read requests. RUN_SYS is de-asserted on clock 11 but SYS_CLOCK (the NMB's internal clock which runs and stops with the rest of the system) does not actually stop until clock 12. The BANK_DONE for this first request was issued at clock 11. The RD_RDY for this request occurred during clock 13. If the clocks were running, it would have then been de-asserted at clock 14 but since RD_RDY went high on the final SYS_CLOCK edge, it stays high until SYS_CLOCK restarts. The BANK_DONE for bank one would have occurred at clock 14 followed by a RD_RDY for bank one at clock 16. Both of these events do not occur at this time since SYS_CLOCK is halted.

After SYS_CLOCK is restarted, RD_RDY first transitions low, then BANK_DONE<1> goes high at clock 22 followed by RD_RDY at clock 24. To the crossbar, which was also stopped, the BANK_DONE and RD_RDY occur at the proper time. The number of SYS_CLOCKs from RDY to BANK_DONE and RD_RDY is the same as is shown in Figure 2-4. The halting of SYS_CLOCK does not effect the timing of RD_RDY and BANK_DONE with respect to the SYS_CLOCK.

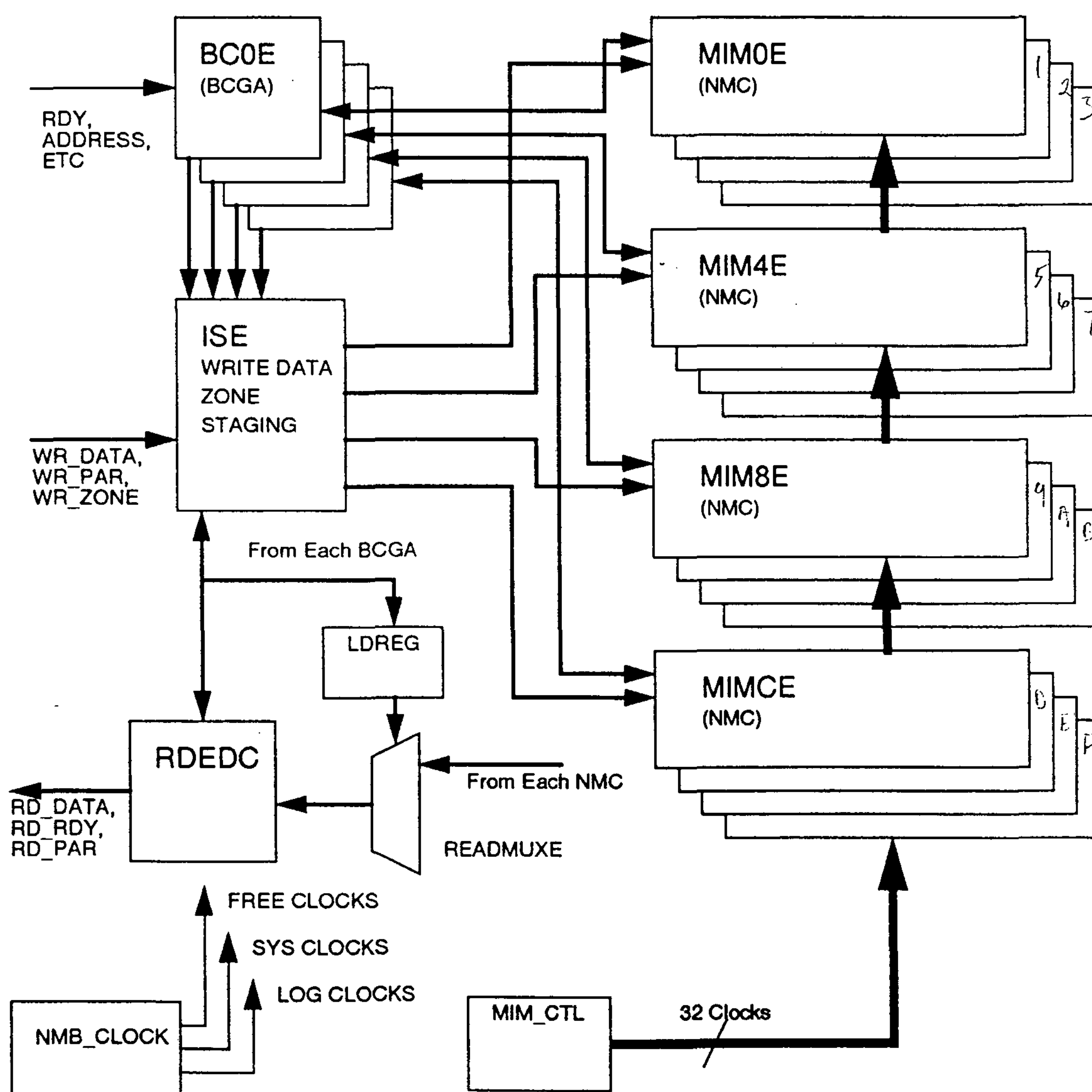
2.2 Internal Interfaces

Figure 2-1 on page 7 is a high level diagram of the NMB board. It shows the basic functional blocks but at a level of detail that is higher than the actual schematic bodies used in the NMB schematics. Figure 2-14 is a more detailed diagram which shows the schematic bodies for the EVEN side logic and all the logic common to both EVEN and ODD sides of the NMB. The ODD side logic has been omitted for clarity but is identical to the EVEN side. In the following sections, each of the sets of signals going between the bodies in Figure 2-14 will be described in detail.

There are four BCGAs on the even side. Each of these gate arrays controls four of the NMCs, thus the grouping in the diagram into four sets of four NMCs. NMCs are number from zero to fifteen in hexadecimal (from MIM0E to MIMFE). The odd side NMCs are numbered as MIM0O to MIMFO. "MIM" is a legacy from the early days of the NMB project and is interchangeable with "NMC" for NMB purposes. (There is actually a C2 card with this name which is why the name changed.)

There are forty eight top level schematics bodies on the NMB schematics: eight BCGAs, the RDEDC, ISE, ISO, READMUXE, READMUXO, LD_REG, MIM_CTL, NMB_CLOCK, and the thirty two NMC connectors representing the NMCs. This figure omits the odd side BCGAs, NMCs, READMUXO, and ISO bodies. The omitted logic is identical to its even side counterparts. NMB_CLOCK, MIM_CTL, LDREG and RDEDC each have a signal instantiation on the NMB schematics.

Figure 2-14 Detailed Internal Interfaces



2.2.1 NMB Clocks

The NMB_CLOCK logic is the catch all body on the NMB. It, of course, generates the internal clocks for the NMB from CU_MB.CLOCK_2X. But it also stages many external signals including the scan in and out bits, hard and soft error signals and the refresh signals. It also performs some of the multiplexing necessary to handle the NMB's three different types of scan rings (LOG, SYS and ALL).

Since the NMB_CLOCK logic performs a number of disjoint functions, the interface signals to it will be treated in separate sections.

2.2.1.1 Clock Generation Signals

All the signals in Table 2-11 are related to the generation of the NMB's clocks. The NMB clocks are a function of the CLOCK_2X signals, the NMB's internal phase generation logic and SYS_RUN, LOG_RUN, and SCAN_CTL. Signals marked GLOBAL use the VALID global signal property and do not appear on the body drawing for NMB_CLOCK. Global signals can be referenced anywhere and at any level in the schematics. Asterisked signals have a corresponding active low version of the signal although many of the active low signals are unused.

Table 2-11 NMB_CLOCK Clock Logic Signals

Signal	Description
CU_MB.CLOCK_2X	2X Clock from NCU
-CU_MB.CLOCK_2X	Complement of Above
-XC_MB.SYS_RUN	SYS_CLOCK enable
-XC_MB.LOG_RUN	LOG_CLOCK enable
-QSYS_RUN<3:0>	Registered SYS_RUN
-QLOG_RUN<3:0>	Registered LOG_RUN
PHASE_CTL	*All scan* mode decode of SCAN_CTL (GLOBAL)
FREE_CK<116:0>	Free running, 1X clocks (GLOBAL)*
FREE_CK2<8:0>	Free running, 2X clocks (GLOBAL)*
SYS_CK<17:0>	System synchronized clocks (GLOBAL)*
FREE_CK_HD<8:0>	As FREE_CK except stopped during Hold Mode (GLOBAL)*

As discussed previously, the NMB has three principle flavors of clocks: log clocks, sys clocks and all or free clocks. The log clocks are 1X clocks gated by RUN_LOG. The sys clocks are 1X clocks gated by RUN_SYS. There are both 1X and 2X free clocks. These clocks only cease when CLOCK_2X from the NCU stops.

SYS_CK is used to clock the input staging part of ISE and ISO as well as the error staging logic. The RDEDC receives a FREE_CK clock and generates its own sys and log clocks from that clock and QSYS_RUN and QLOG_RUN. The BCGAs receive a FREE_CK2 clock, QSYS_RUN and QLOG_RUN and generate internal log and sys clocks as well as 1X and 2X free running clocks. The BCGAs are the only device on the NMB to receive a 2X clock except for some clock generation logic in NMB_CLOCK. The remainder of the NMB logic uses FREE_CKs (thus the large number of them).

Within the NMB_CLOCK block of logic there is a ring of four registers called PHASE_GEN_1X<3:0> running at a 2x clock rate. Every other bit in this ring is initialized to a one.

PHASE_GEN_1X<0> is used to gate the 2X clock to produce all the 1X clocks (every other pulse of the 2X clock is masked leaving a 1X clock rate).

PHASE_CTL is only active during the *all scan* mode. When PHASE_CTL is active the phase generator logic becomes part of the scan ring. When PHASE_CTL is inactive, the phase generator registers form a closed shift ring.

All the clocks are very dependent on SCAN_CTL for their mode of operation. Table 2-12 shows what gates each of the clocks in each of the valid scan modes. For log clock and SYS_CK, gated by RUN_LOG or RUN_SYS implies also being gated by the phase generator logic. It is the phase generator logic that gates the 2X clock into a 1X clock. FREE_CK2 is never gated. (Internal to the BCGAs, the only user of FREE_CK2, it is gated with phase logic similar to that for the 1X clocks. This phase logic is usually all zero so that it does not generally cause the 2X clock to be gated except during CAST mode.) FREE_CK_HD is identical to FREE_CK except that it is stopped during NMC Init mode.

Table 2-12 Clock Types Versus Scan Mode

Scan Mode	CU_MB. CLOCK_2X	Log Clock	SYS_CK	FREE_CK
Normal	2X	Gated by RUN_LOG	Gated by RUN_SYS	Gated by Phase
Log Scan	2X	Gated by RUN_LOG	Gated by RUN_SYS	Gated by Phase
Sys Scan	2X	Gated by RUN_LOG	Gated by RUN_SYS	Gated by Phase
NMC modes	Single Pulse	Stopped	Stopped	Ungated Single Pulse
CAST Load	1X or 2X	Ungated by RUN_LOG	Ungated by RUN_SYS	Gated by Phase
All Scan	1X	Ungated by anything	Ungated by anything	Ungated by Anything

All clocks except for FREE_CK2 are 1X clocks. FREE_CK2 is a 1X clock only during *all scan* when the NCU generates a 1X clock instead of the normal 2X clock for the NMB. Since all clocks are ungated during *all scan*, all clocks will be running at the CLOCK_2X rate which is 1X in this mode.

2.2.1.2 Register Control Signals

NMB_CLOCK generates many of the control signals for the various registers on the NMB. These control signals are derived from SCAN_CTL.

Table 2-13 Register Control Signals

XC_MB.SCAN_CTL<2:0>	Scan Control from XCL
SCAN_CTL.x<2:0>	Buffered SCAN_CTL (GLOBAL)
RSCAN_CTL.x<2:0>	Registered SCAN_CTL (GLOBAL)
ALL_DMODE.x	Registered SCAN_CTL<2> (GLOABL)
ALL_141_CTL.x<1:0>	Controls for the 100141 and 100E141 (GLOBAL)
SYS_241_CTL<1:0>	Controls for the 100E241 (GLOBAL)
HOLD_MODE	Hold mode decode of SCAN_CTL

HOLD_MODE and ALL_DMODE are the simplest of these signals. HOLD_MODE is true only during NMC init mode when SCAN_CTL equals five. During this mode, the only state that should change on the entire NMB is the state dealing with the NMC clock generation logic in MIM_CTL (section 2.2.2 on page 34). HOLD_MODE is used by some registers to inhibit loading of data at this time. HOLD_MODE is also used by MIM_CTL to determine when to initialize the NMC clock generation state machine.

ALL_DMODE is active any time SCAN_CTL bit two is set, i.e. modes, 4, 5 (NMC init), 6 (CAST Load) or 7 (ALL SCAN). It is used to force many of the registers in ISE and ISO to use the ALL_141_CTL signals as register controls rather than other signals internal to ISE or ISO.

ALL_141_CTL is used to control the 100E141s and 100141s on the NMB all of which are clocked by FREE_CK clocks. The 100141s in the ISE/ISO logic ignore ALL_141_CTL when ALL_DMODE is not asserted. Table 2-14 shows the correspondence between SCAN_CTL and ALL_141_CTL and the action taken by the 100141s/100E141s as indicated in the databooks for those parts.

Table 2-14 ALL_141_CTL

SCAN_CTL	ALL_141_CTL	Action
Normal	00	Load
Log Scan	00	Load
Sys Scan	00	Load
NMC Init	11	Hold
CAST Load	00	Load
ALL Scan	10	Shift, lsb to msb

SYS_241_CTL is similar to ALL_141_CTL. It is used to control the 100E241s on the NMB, all of which are clocked by SYS_CK. Table 2-15 shows the value of SYS_241_CTL versus SCAN_CTL

and the action implied. Again, the action is taken from the datasheet for the 100E241. For the 100E241s, bit one is a don't care when bit zero is asserted.

Table 2-15 SYS_241_CTL

SCAN_CTL	SYS_241_CTL	Action
Normal	00	Load
Log Scan	00	Load
Sys Scan	01	Shift, lsb to msb
NMC Init	10	Hold
CAST Load	00	Load
ALL Scan	01	Shift, lsb to msb

The buffered versions of SCAN_CTL are used by the gate arrays and MIM_CTL. Buffering is necessary because of loading on SCAN_CTL and to keep XC_MB.SCAN_CTL as free from electrical glitches as possible because it needs to be registered to generate RSCAN_CTL.

RSCAN_CTL is a registered version of XC_MB.SCAN_CTL. It goes only to the BCGAs and MIM_CTL and is only used when ALL_DMODE is not set. It provides a synchronous copy of SCAN_CONTROL which may be safely registered by logic running at the normal system rate.

2.2.1.3 Error Logic Signals

Table 2-16 NMB_CLOCK Error Signals

HARD_ERROR_E<4:0>	Internal Hard Error Sources, Even Side
HARD_ERROR_O<4:0>	Internal Hard Error Sources, Odd Side
XBAR_ERROR_E<3:0>	Input Staging Hard Error Sources, Even Side
XBAR_ERROR_O<3:0>	Input Staging Hard Error Sources, Odd, Side
SOFT_ERROR_E<4:0>	Internal Soft Error Sources, Even Side
SOFT_ERROR_O<4:0>	Internal Soft Error Sources, Odd Side
MB_XS0E.SEND_PAR_ERR	Advance Error Signal to XS0E
MB_XS1E.SEND_PAR_ERR	Advance Error Signal to XS1E
MB_XS0O.SEND_PAR_ERR	Advance Error Signal to XS0O
MB_XS1O.SEND_PAR_ERR	Advance Error Signal to XS1O
MB_XC.HARD_ERROR	Hard Error report signal
MB_XC.SOFT_ERROR	Soft Error report signal
HARD_ERR_ENABLE	Enable for some of the NMB Hard Errors

All the external interface signals in Table 2-16 have been thoroughly discussed in previous sections. The SEND_PAR_ERR signals are asserted the clock after any bit in their corresponding XBAR_ERROR signals is high. MB_XC.HARD_ERROR is asserted the clock after any HARD_ERROR signal, even or odd, is asserted. MB_XC.SOFT_ERROR is similar in operation.

HARD_ERR_ENABLE is an internal signal used to disable some of the hard error checking on the NMB. It does not disable XBAR_ERROR signals or ISE/ISO generated hard errors so it is of limited utility.

The HARD_ERROR, SOFT_ERROR and XBAR_ERROR signals are the error sources internal to the NMB. Table 2-17 gives the sources of each of the error signal bits for the even side. The odd side signals are similarly mapped.

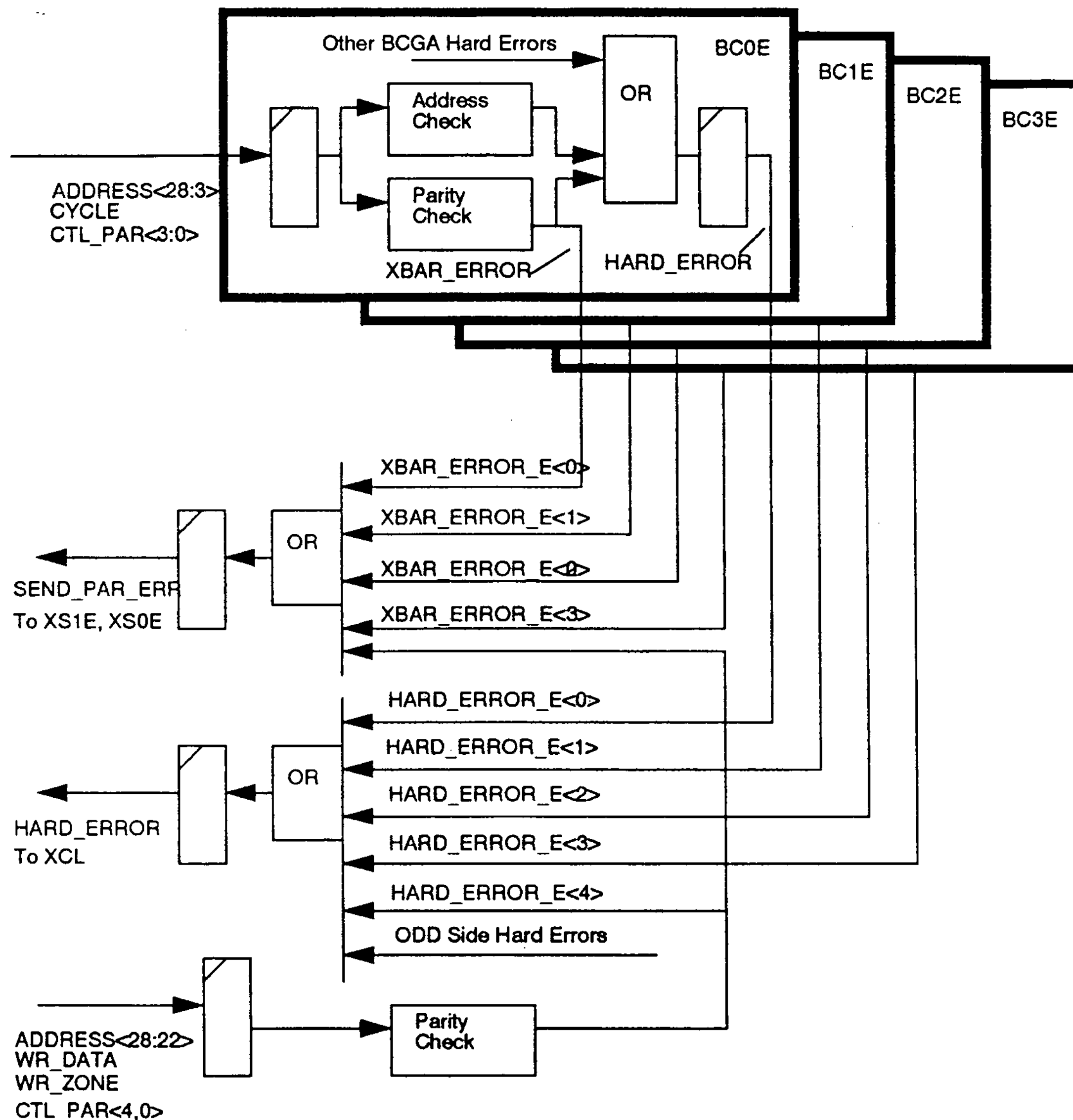
Table 2-17 Error Sources

HARD_ERROR_E<0>	BC0E Hard Error
HARD_ERROR_E<1>	BC1E Hard Error
HARD_ERROR_E<2>	BC2E Hard Error
HARD_ERROR_E<3>	BC3E Hard Error
HARD_ERROR_E<4>	Input Staging, Even Side (ISE) Hard Error
XBAR_ERROR_E<0>	BC0E Advanced Hard Error
XBAR_ERROR_E<1>	BC1E Advanced Hard Error
XBAR_ERROR_E<2>	BC2E Advanced Hard Error
XBAR_ERROR_E<3>	BC3E Advanced Hard Error
SOFT_ERROR_E<0>	BC0E Soft Error
SOFT_ERROR_E<1>	BC1E Soft Error
SOFT_ERROR_E<2>	BC2E Soft Error
SOFT_ERROR_E<3>	BC3E Soft Error

2.2.1.4 NMB Hard Error Staging

The following figure shows the registers used in the hard error generation for the principle hard errors. For simplicity's sake, odd side hard error generation has been omitted.

Figure 2-15 Hard Error Staging



2.2.1.5 NMB_CLOCK Scan Signals

The NMB_CLOCK logic contains scannable state (all registers are either sys or all scan mode scannable) which interact with the rest of the scan ring. It also handles the scan in staging from the XCL board.

Table 2-18 NMB_CLOCK Scan Signals

XC_MB.SCAN_CTL<2:0>	Scan controls
XC_MB.SCAN_IN	Scan input from XCL
ISE_SCAN_OUT	Scan input for NMB_CLOCK All state
ISE_SYS_SCAN_OUT	Scan input for NMB_CLOCK scan multiplexor
BC3E_SCAN_OUT	Scan input for NMB_CLOCK scan multiplexor
NMB_CLOCK_SCAN_OUT	Scan out from NMB_CLOCK for sys state
MIM_CMOS_SCAN_IN	Scan out from NMB_CLOCK for log state
NMB_ALL_SCAN_OUT	Scan out from NMB_CLOCK for all state

XC_MB.SCAN_IN is the input signal for all of the NMB scan rings. Its use is described in section 2.1.5 on page 17. The remainder of the NMB_CLOCK scan signals have to do with the scannable state in this logic body. How they fit into NMB scan rings will be discussed in section 2.2.8 on page 39.

2.2.1.6 Miscellaneous NMB_CLOCK Signals

A number of miscellaneous signals are handled by the NMB_CLOCK simply because it contained registers with spare bits.

Table 2-19 NMB_CLOCK Miscellaneous Signals

XSE_MB.REF_REQ	Even Side, Normal Mode Refresh Request
XSO_MB.REF_REQ	Old Side, Normal Mode Refresh Request
RE_REF_REQ	Registered refresh request for even side BCGAs
RO_REF_REQ	Registered refresh request for odd side BCGAs
XC_MB.RAM_RFSH	Raw Refresh Request for Clock Stop
RRAW_REF_REQ	Registered raw refresh request for all BCGAs

As discussed previously, the XSx.REF_REQ signals are the signals from the XSE and XSO boards for initiating refreshes during normal operation. These signals are registered within the NMB_CLOCK and driven to the even and odd side BCGAs via RE_REF_REQ and RO_REF_REQ.

When RUN_SYS is de-asserted or the NMB is in sys scan, XC_MB.RAM_RFSH is used for refreshes. It is registered and becomes RRAW_REF_REQ which is propagated to all BCGAs. The BCGAs select which of the refresh signals should be used.

2.2.2 NMC Clock Generator

MIM_CTL handles the clocks for the NMC logging logic. During any type of scan mode, it generates the clocks to the NMCs. During normal operation or during CAST load, it passes the clocks generated by the BCGAs.

Table 2-20 MIM_CTL Interface Signals

Signal	Description
B0O_CHECK	NMC log clock generated by BC0O for MIM0O
B0O_MIM_CLK	Log clock to MIM0O
...Other NMC Clock Pairs...	
MIM_CMOS_SCAN_IN	Scan input for NMC log state
NMC_BYPASS	Bypass for NMC log scan
B8E_SCAN_OUT to BFE_SCAN_OUT	Scan outputs from NMC block
HOLD_MODE	True for NMC Clock Init Mode
SCAN_CTL	Unregistered version of SCAN_CTL
RSCAN_CTL	Registered version of SCAN_CTL
-QLOG_RUN	Registered RUN_LOG signal
NMB_CLOCK_SCAN_OUT	Scan input for MIM_CTL all state
MIMO_CMOS_SCAN_IN.0 to MIMO_CMOS_SACN_IN.7	Scan ins for NMC block
SCAN_OUT	Scan out for MIM_CTL all state
MIM_SCAN_OUT	Scan out for NMC log state

There are a total of thirty pairs of Bxx_CHECK/Bxx_MIM_CLK signals, numbered from 0 to F (hexadecimal fifteen) even and odd (E and O). The last signal pair in the set is BFE_CHECK/ BFE_MIM_CLK for fifteenth even side NMC.

The CHECK signals are from the BCGA gate arrays. During the normal course of cycling the DRAMs, the bank controllers in the BCGA will assert their CHECK signals to cause a log state clock on the NMC in order to capture any potential error state. As long as QLOG_RUN is active and log state is not in scan mode, a one on the CHECK signals will cause a one on the following 1X clock for the corresponding MIM_CLK signal.

When QLOG_RUN goes inactive or the NMB is in log scan mode, CHECK signals can not cause a MIM_CLK. Instead, during scan mode, while scan mode is log or sys scan and QLOG_RUN is active or during all scan regardless of QLOG_RUN, logic internal to the MIM_CTL will generate the MIM_CLKs. During scan, every MIM_CLK is pulsed once per eight 1X clocks. Four NMCs receive a MIM_CLK on any one clock.

MIM_CMOS_SCAN_IN is the scan input to the log state for the NMCs. This signal is registered at the NMC MIM_CLK rate and becomes the eight MIMO_CMOS_SCAN_IN.x signals. Signal 0 goes to MIM0O, 1 to MIM1O, and 7 seven to MIM7O.

Since the NMCs are being scanned at one eighth the system frequency, each scan clock, one of the B8E_SCAN_OUT to BFE_SCAN_OUT signals must be selected to produce MIM_SCAN_OUT, the scan out for the NMCs log state.

NMC_BYPASS is used to cut the NMCs out of the NMB scan ring. When this signal is high, the NMC state will be bypassed. Because the NMCs terminate some of the signals used for scan on the NMB, the NMCs must still be plugged in in order for scan to function, even in NMC_BYPASS mode. NMC_BYPASS is available on the NMB backplane interface for access by the CAST tester.

HOLD_MODE is decoded from SCAN_CTL and is true for scan mode four and five, the NMC clock initialize modes. QLOG_RUN, SCAN_CTL and RSCAN_CTL are used by MIM_CTL to determine when it should be generating scan clocks for the NMCs.

SCAN_OUT is the scan output for the MIM_CTL's all state.

2.2.3 BCGA to NMC

Each BCGA controls four NMCs. The control logic for a particular NMC is called a bank controller since each NMC represents one logical bank of memory. Table 2-21 shows the BCGA to NMC interface for the MIM0E bank. As elsewhere, the interface names for the other banks run from 0 to F (hexadecimal fifteen) with an E or O suffix for even or odd. All BCGA to NMC interfaces are identical.

Table 2-21 BCGA to NMC Interface

Signal	Description
-B0E_RAS	Row Address Strobe to DRAM
-B0E_CAS	Column Address Strobe to DRAM
-B0E_WE	Write Enable to DRAM
-B0E_G	Output enable to DRAM
B0E_WR_SEL	Mode Bit for Write Operations
B0E_CHECK_LE	Clock for NMC Input Registers
B0E_DATA_SEL	Select for Corrected Data
B0E_CHECK	NMC Log State Clock and Error State Clock
B0E_ECC_CHECK	Data Strobe for non-DRAM drivers on NMC
B0E_DATA_STR	Latch Enable for NMC Output Latches
B0E_RFSH_ACT	Indicates a Refresh to NMC for RAS Generate
B0E_RAM_ADDR<9:0>	DRAM Address Bus
B0E_MIM_ERROR	Error Signal From NMC

The first four signals in Table 2-21 are the four control signals for commodity "by four" DRAMs. DRAM cycling will be covered in a later chapter. Briefly, the RAS (Row Address Strobe) signal operates as the device select strobe and the clock enable for loading the first part (Row Address) of the two part address to the DRAM. CAS (Column Address Strobe) is the address clock for the second part (Column Address) of the address. WE is asserted to force the DRAM to write. G is asserted to enable the DRAM to drive data out of its bidirectional data pins.

WR_SEL controls how the NMC composes data for a write to the DRAMs. When it is a zero, the NMC loads all bytes of its write data register. When WR_SEL is a one, the NMC uses ZONE information from the ISE or ISO logic to determine whether to load or hold a byte of the write register. WR_SEL is zero in the first part of all types of memory operations when all of write data is being loaded by the NMC. WR_SEL then goes one for read modify writes where the write data word must be composed from bytes of write data and bytes of data from memory.

CHECK_LE is the clock for the NMC's write data and check data registers. These registers are the NMC's data input registers. CHECK_LE is pulsed once, for all cycles, to clock in initial write data from ISE/ISO. It is pulsed a second time for a read-modify-write operation to load the data read from the DRAMs.

DATA_SEL is only used for read-modify-write operations in which a soft error was detected. When DATA_SEL is asserted, the NMC selects corrected data to be written into the DRAMs. Otherwise, uncorrected data is written. Corrected data takes so long to generate that uncorrected data is used first in read-modify-writes. Then if an error was detected, a second operation using the corrected data (DATA_SEL equal one) is executed.

CHECK causes two clocks to be generated for the NMCs. CHECK goes directly to the NMC and becomes the clock for registering error state on the NMCs (ERR_CLK). CHECK also goes to MIM_CTL logic to become MIM_CLK for registering log state on the NMCs. The two clocks are similar in purpose (to record the results of a memory cycle). However, ERR_CLK is not masked by log operations while MIM_CLK becomes the log scan clock during scan.

ECC_CHECK is a multi-purpose signal. It tells the NMC whether it is checking data for good parity, as at the beginning of a cycle when write data and parity are sent to the NMC or whether it is checking the error check and correct code (ECC). ECC_CHECK also determines what devices will be driving the NMC's internal tri-state bus.

DATA_STR is the latch enable for the NMC's output latch. The latch is closed when DATA_STR is high. The signal transitions from low to high at the end of every read cycle so that the NMC holds its last read data on its return data bus to the READMUX. This is done so that when clocks are restarted as described in section 2.1.5.7 on page 23 and RD_RDYs are regenerated for reads that were interrupted by the clock stop, the return data is still available.

RFSH_ACT tells the NMC that the current operation is a refresh. The RASs for the four rows of memory on an NMC are generated on the NMC. The NMC needs RFSH_ACT to know to assert all four RASs for a refresh rather than one of the four as in normal operations.

RAM_ADDRESS is the DRAM address bus. It is used to pass first the row address, then the column address for reads, writes and test-and-modifies. It is also used to pass the refresh address on refresh cycles.

MIM_ERROR is the error signal from the NMC used to report single and multiple bit errors. If MIM_ERROR is one while DATA_SEL is zero, it is reporting an error, either single or multiple bit. If MIM_ERROR is a zero while DATA_SEL is a one, the error previously reported by MIM_ERROR was a single bit error. A MIM_ERROR of one while DATA_SEL is a one corresponds to a multiple bit error which is a hard error. Note that DATA_SEL will only be a one following a read-modify-write during which MIM_ERROR was one while DATA_SEL was zero.

2.2.4 ISE to NMC

The even side input staging logic (ISE) stages write data, write parity, write zone and two of the address bits for the each of the even NMCs.

Table 2-22 ISE to NMC Signals

B0E_BANK_WR_DAT<31:0>	Write Data
B0E_BANK_WR_PAR<3:0>	Write Parity
B0E_ZONE<3:0>	Write Zone
B0E_ROW_ADDRESS<1:0>	Row Select

Table 2-22 shows the signals going from the ISE to the NMC for a particular NMC, in this case, MIM0E. There is a similar set of four signals going to each of the other fifteen even side NMCs. No signals go from the NMC back to the ISE.

The ISE has a free running register which registers write data, zone, parity and row address from the even crossbar every clock and a register for this data for each of the sixteen banks on the even side. The bank registers free run until a request for the bank is received after which they hold the data until the bank is complete. This data is simply driven to the NMC on the signals listed in Table 2-22.

ROW_ADDRESS is bits twenty seven and twenty eight of the request address. These bits are used by the NMC to select one of the four rows of memory.

2.2.5 NMC to READMUXE to RDEDC

The READMUXE logic block is a large multiplexor with some buffering for the select lines. It switches between one of sixteen sources of return data, each source being an NMC. The selected data then goes to the RDEDC on the RTN_DATA_E and RTN_ECC busses.

Table 2-23 NMC to READMUX to RDEDC Signals

B0E_DATA_OUT<31:0>	Return Data from MIM0E NMC
B0E_ECC_OUT<7:0>	Error Correct Code for MIM0E Return Data
RTN_DATA_E<31:0>	Selected Data to RDEDC
RTN_ECC<7:0>	Selected ECC to RDEDC

READMUXE simply selects the proper NMC according to the selects as described in section 2.2.7 on page 38 and drives that NMC's data and ECC onto the RTN_DATA and RTN_ECC bus which go directly to the RDEDC.

2.2.6 BCGA to ISE

The BCGA controls the load/hold of the ISE bank staging registers. A single signal of the form B0E_WR_REG_HOLD runs from each bank (four per BCGA) to the ISE. At the beginning of a request to a bank, the signal is asserted to freeze the DATA, ZONE, PARITY and ROW_ADDRESS state for the duration of the request.

2.2.7 BCGA to LDREG, RDEDC and READMUXE

The BCGAs control the selection of data by the READMUXE and tell the RDEDC when valid data is present from the READMUXE.

Table 2-24 BCGA controls to LDREG, RDEDC and READMUXE

LD_EN_A_E	Load enable from BC0E.
LD_EN_B_E	Load enable from BC1E.
LD_EN_C_E	Load enable from BC2E.
LD_EN_D_E	Load enable from BC3E.
RTN_SEL_A_E<1:0>	Return select from BC0E.
RTN_SEL_B_E<1:0>	Return select from BC1E.
RTN_SEL_C_E<1:0>	Return select from BC2E.
RTN_SEL_D_E<1:0>	Return select from BC3E.
RLD_EN_A_E	Registered load enable from BC0E.
RLD_EN_B_E	Registered load enable from BC1E.
RLD_EN_C_E	Registered load enable from BC2E.
RLD_EN_D_E	Registered load enable from BC3E.
RSELECT_E<3:0>	Select to READMUXE from LDREG.

Each BCGA generates two types of signals for controlling return data, a load enable LD_EN which indicates return data is expected and a return select RTN_SEL which is a two bit code for selecting which of the four banks controlled by a BCGA is expecting return data. Since the NMB can receive only one request per clock on the even side and all requests that return data do so after the same number of clocks, only one of the sixteen banks will have return data on any one clock. Therefore, only one of the four LD_EN signals will be asserted on any one clock. The RTN_SEL for BCGAs whose LD_EN is zero is also zero. Only the RTN_SEL for the BCGA that has read data can have a RTN_SEL of non-zero value.

RTN_SEL and LD_EN are decoded by LDREG into a one of sixteen select and then the select and LD_EN signals are registered. For the select, the upper two bits are a four to two encode of LD_EN and the lower two bits are an OR of all the RTN_SEL signals. RTN_SEL can be OR'd since

only one will be non-zero on any one clock. LDREG then drives these signals to the RDEDC and READMUXE to select the proper return data on the following clock.

Figure 2-16 LD_EN and RTN_SEL

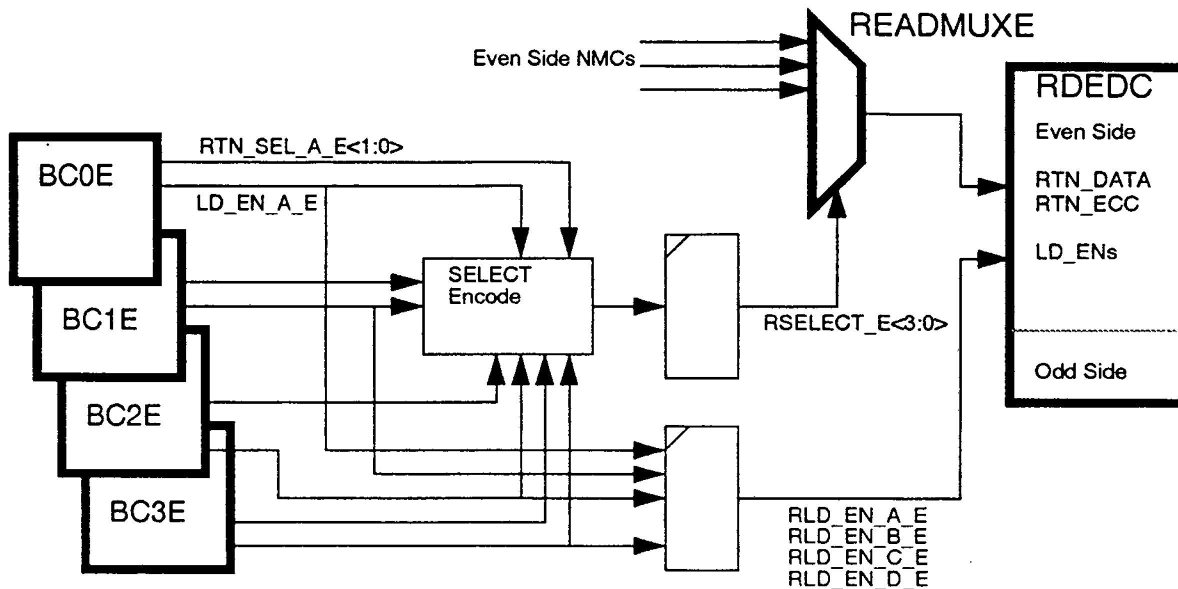


Table 2-25 LDREG Select Encode

SELECT_E<3> = LD_EN_D_E + LD_EN_C_E

SELECT_E<2> = LD_EN_D_E + LD_EN_B_E

SELECT_E<1> = RTN_SEL_A_E<1> + RTN_SEL_B_E<1> +
RTN_SEL_C_E<1> + RTN_SEL_D_E<1>

SELECT_E<0> = RTN_SEL_A_E<0> + RTN_SEL_B_E<0> +
RTN_SEL_C_E<0> + RTN_SEL_D_E<0>

Figure 2-16 shows the BCGA/ LDREG/ READMUXE/ RDEDC interface. Table 2-25 gives the equations for producing the SELECT signals. All signals in Table 2-24 are clocked off of the sys clocks. The RDEDC is entirely clocked by SYS clocks. Therefore all logic depicted in the above figure is synchronous with the crossbar logic and will halt when RUN_SYS is de-asserted.

The RDEDC only returns four signals back to the BCGAs, RTN_MERR_E and RTN_SERR_E for the even BCGAs and RTN_MERR_O and RTN_SERR_O for the odd side. Both the SERR and MERR signals are asserted on the clock following the RD_RDY which corresponds to the data in question. That is, the data for which the SERR and MERR is valid was sent to the crossbar on the previous clock. If RTN_SERR_x is high, a single bit error was detected; if RTN_MERR_x is high, a multiple bit error was detected.

2.2.8 NMB Scan Rings

As discussed earlier, the NMB has three types of state, each of which can be accessed by a different scan mode. The log state is scanned via the log ring. It contains information about soft

errors. The sys state includes the log state plus hard error logging state and the registers on the NMB that interface with the crossbar boards. The *all state* includes the previous state and all remaining scannable state on the memory board.

In the following subsections, the scan ring lengths of various NMB schematic bodies and rings will be given to give a proper idea of relative sizes of the various elements. It is important to remember that these lengths are as of March 26, 1991 and may change because of bug fixes. The final description of the NMB scan rings will always be the NMB scan ring definition files.

2.2.8.1 Log Ring

The log ring is 3156 bits long. It contains address, data, syndrome and type of error information dealing with soft errors.

Figure 2-17 Log Ring

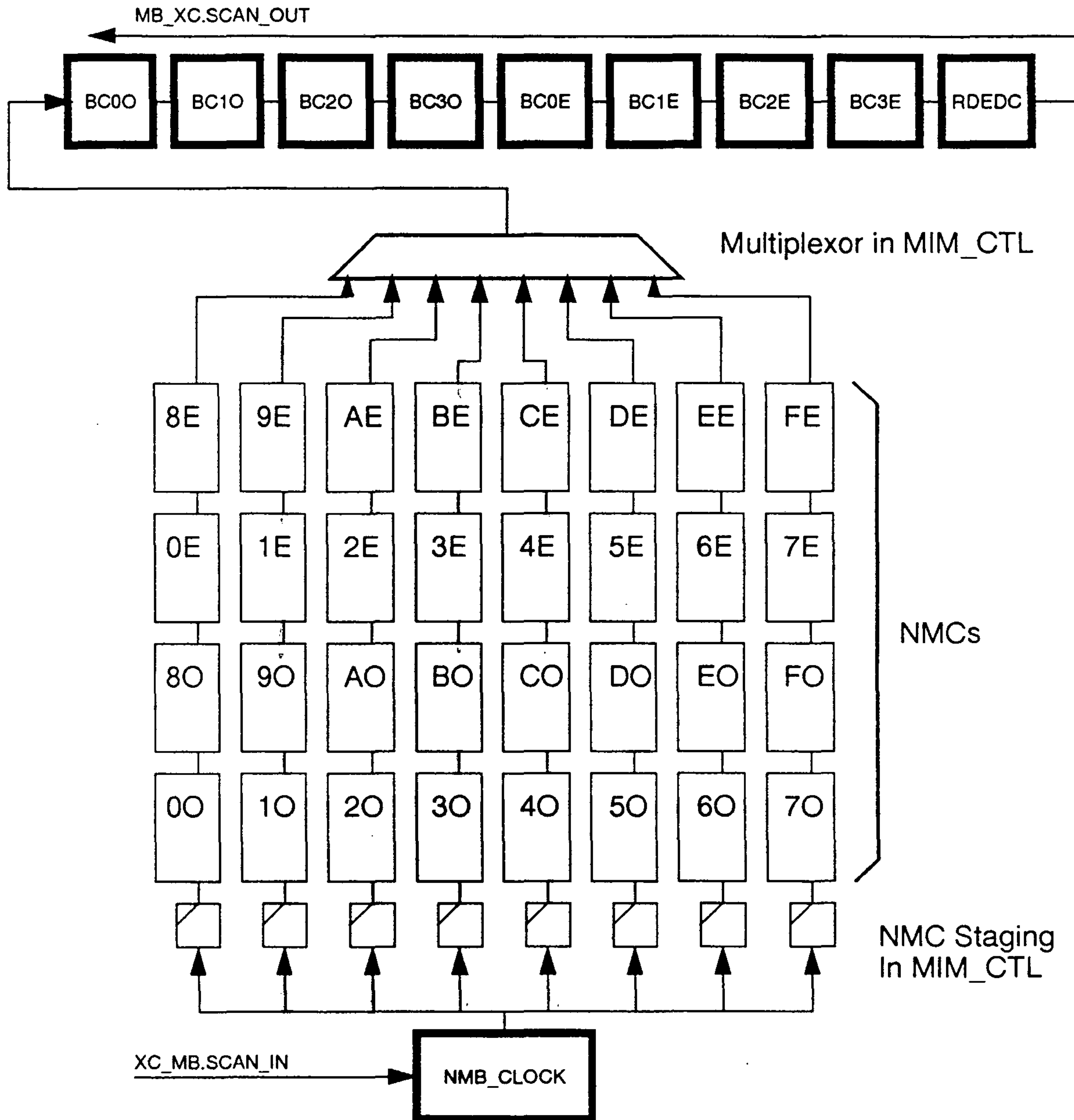


Figure 2-17 shows the way in which the log state is connected. The block labeled NMB_CLOCK is simply input staging for the scan input bit. There is no logging information in the NMB_CLOCK body.

As described in section 2.2.2 on page 34, the NMCs are not scanned at the normal 1X rate. Each NMC, instead receives a 1/8 rate clock. On any given clock during scan only one column of NMCs as shown in the figure a receiving a clock. On the next clock, the column to the right gets its clock

and so on across all eight columns. The multiplexor above the NMCs selects the column receiving the clock on each clock. The registers below the NMCs are staging registers for the NMCs which clock at the same rate that the NMCs are being clocked. They provide the hold requirements that the NMC scan input signal requires.

So, during scan, MIM0O, MIM8O, MIM0E, and MIM8E receive a clock. Then Mim1O, MIM9O, MIM1E and MIM9E get their clock, and so on.

After the NMCs the log scan ring is connected to the BCGAs in the order shown in the figure and then to the RDEDC. The RDEDC drives the scan output bit back to the XCL. For both the RDEDC and BCGA, there is only a single scan input pin and a single scan output pin. Both gate arrays receive SCAN_CTL and internally decode whether they should be scanning their log state, sys state or all state during scan mode.

The NMCs each have 69 bits in their ring for a total of 2208 bits for all the NMCs. The BCGAs have 104 bits of log state. The RDEDC has 102 bits of log state.

2.2.8.2 Sys Scan Ring

The sys ring contains all the state of the log ring discussed in the previous chapter plus some additional state.

Figure 2-18 Sys Ring

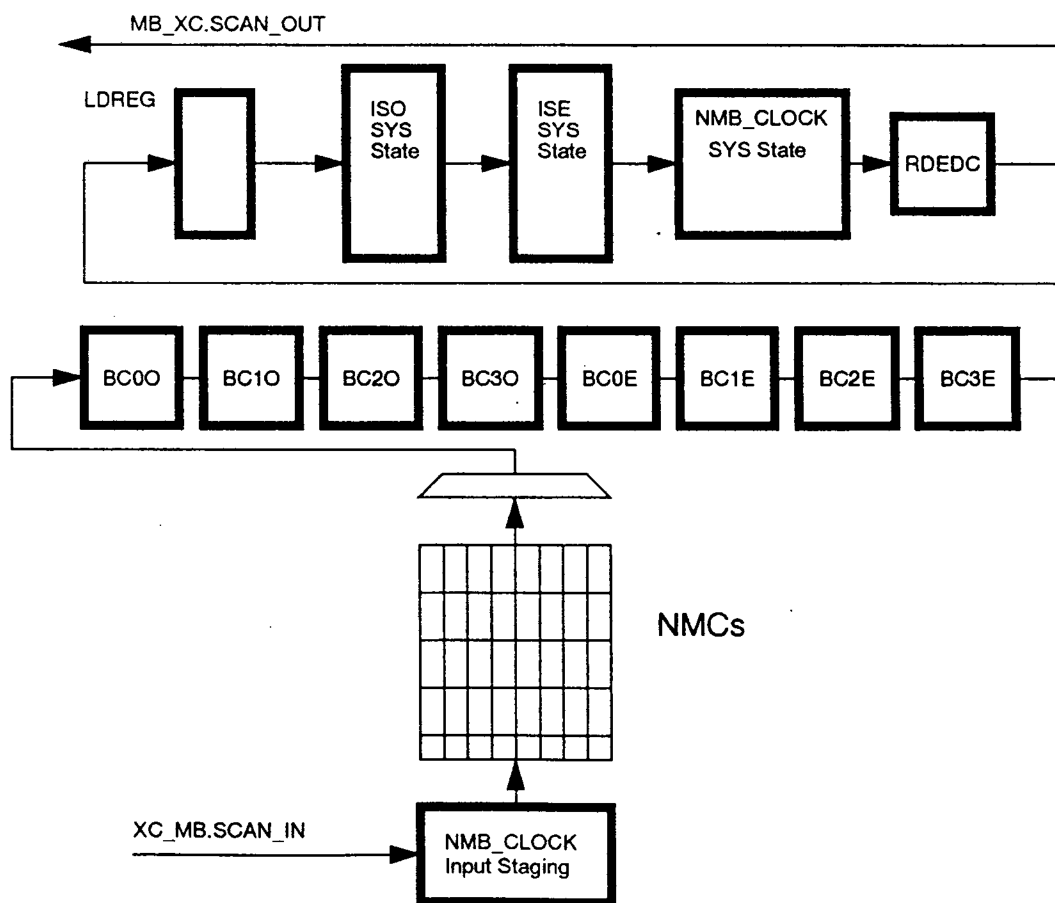


Figure 2-18 shows the connectivity of the sys ring. It is very similar to the log ring except that some sys state has been added between BC3E and RDEDC. This state is all sixteen bits of LDREG, the input staging registers of ISE and ISE, and the hard error state in NMB_CLOCK. Of course, the gate arrays are also scanning out sys state in sys ring scan in addition to the log state the scanned in log ring mode.

The NMCs state in sys mode is identical to that in log mode.

The sys ring has 3978 bits in it.

2.2.8.3 ALL Ring

The *all ring* consists of all the scannable state on the NMB. In this mode, each of the gate arrays now scan out all of its state. (The RDEDC has only log and sys state so it scans no more in *all*

scan than it did in sys scan.) Some additional board logic is also added to the scan ring for this mode. Total number of bits in this scan ring is 9474.

Figure 2-19 All Ring

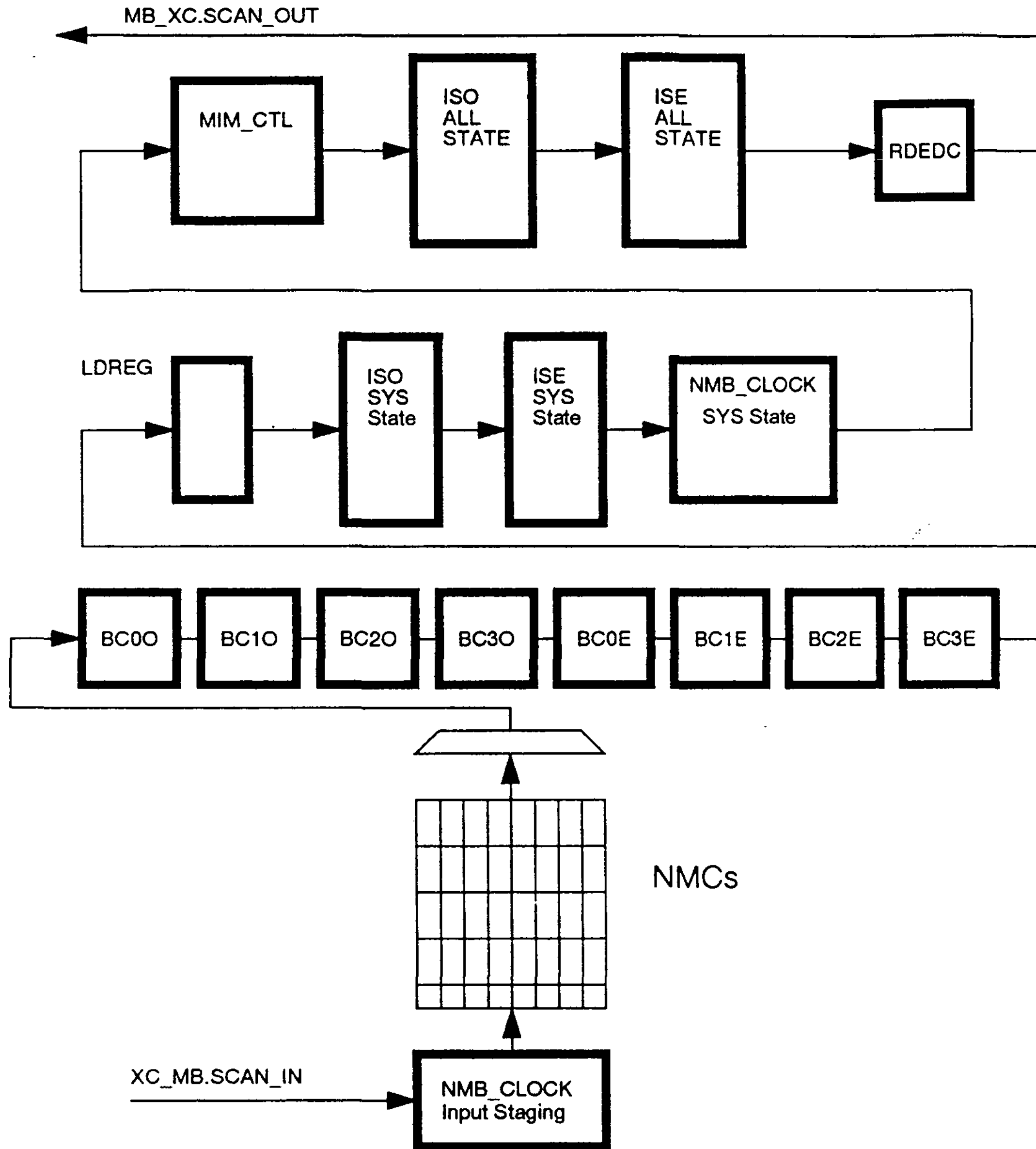


Figure 2-19 differs from Figure 2-18 only in the addition of the MIM_CTL state and the free running state (all state) of the ISE and ISO.

3 Functional Description

This chapter describes the internal operation of the NMB from the operation of read, write, and read-modify-write operations to error checking functions. When necessary, bank 0 zero of the even side, bank 0E, will be used as example. All statements about bank 0E apply equally to the other thirty one banks on the NMB.

As in the previous chapter, this chapter must make certain assumptions regarding how the BCGAs are programmed for DRAM timing. Changes in how the BCGAs are programmed will alter the exact timing relationships between various signals but will not alter the relative timing between signals.

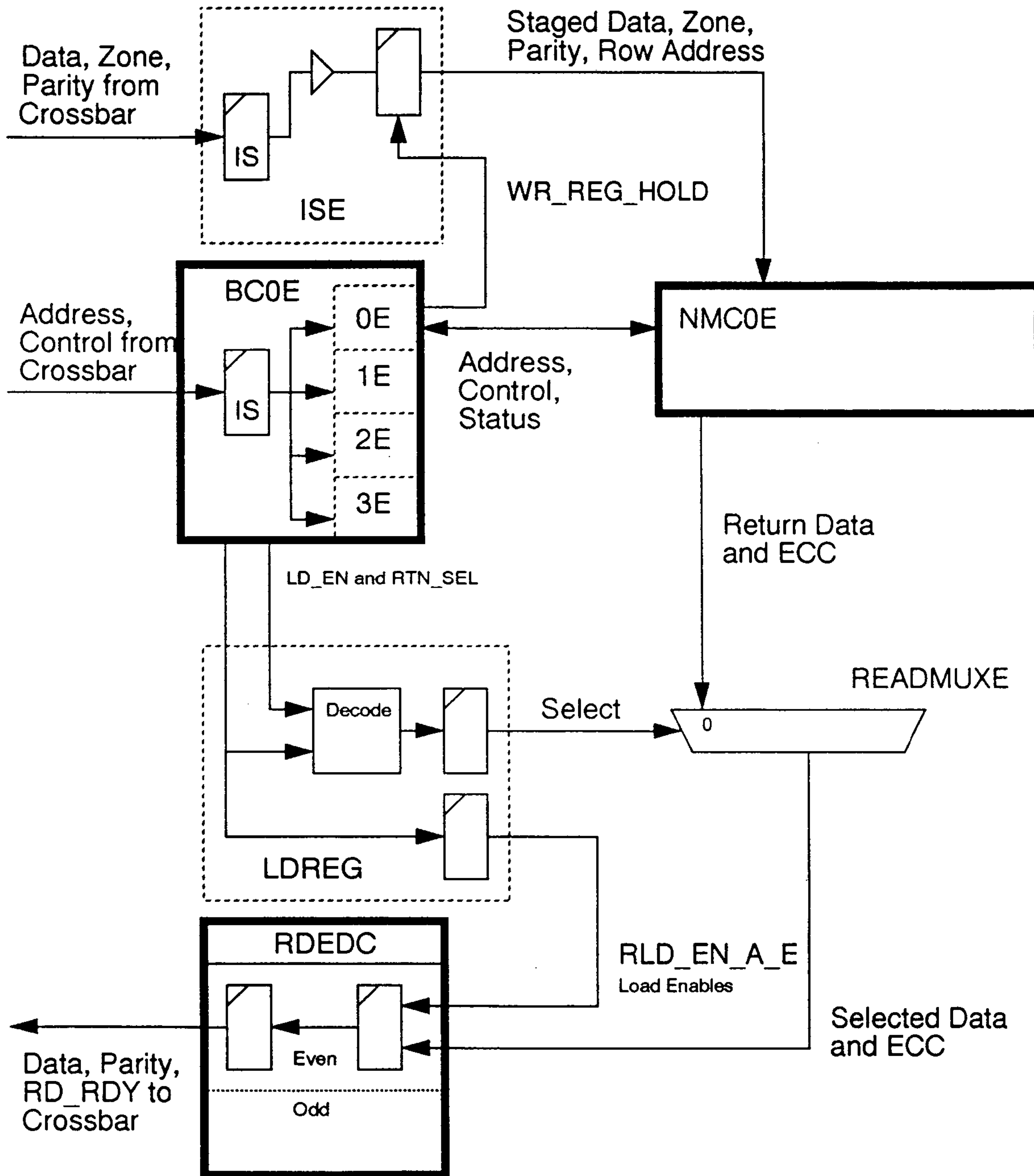
All signal transitions are shown immediately following the clock edge that causes the signal transition. No propagation delays will be shown in the diagrams unless explicitly stated. Note that many propagation delays are significant and may be as much as a complete clock in some cases. However, these delays vary greatly depending on which bank is being examined since some are physically much closer to the BCGAs than others. Showing these delays would also make it more difficult to determine cause and effect relationships with respect to clock edges.

Figure 3-1 shows the logic blocks on the NMB that are used in bank zero, even side (0E) requests, the example bank. All other logic on the board has been omitted. In particular, only one BCGA and one NMC are shown and inputs from other logic into the ISE, LDREG, READMUXE, RDEDC are omitted. Some control and status signals for bank 0E logic is also omitted for clarity.

The registers labeled "IS" are the input staging registers referred to throughout this document. These are the registers that capture data from the crossbar. They represent the edge of the NMB with respect to data from the crossbar.

From the figure it can be seen that the board level logic required for a single bank is fairly small. Any given bank uses a quarter of a BCGA, data staging registers in the ISE/ISO logic, the return multiplexor (READMUX) and a small amount of logic for LDREG.

Figure 3-1 Bank Zero, Even Side Logic



3.1 ECC, Error Check and Correct

ECC is an acronym for Error Check and Correct (or Error Correct Code). The term EDC, Error Detect and Correct, is also used. Because of the nature of DRAMs as discussed elsewhere, the

NMB uses a special code called an ECC code to be able to correct single bit errors and detect certain other errors. As long as only a single bit is corrupted, that bit can be "fixed" using the code.

The ECC uses extra code bits with each word to provide sufficient redundant information that certain errors can be corrected. The NMB uses a modification of the standard 32 bit single bit error correct, double bit error detect (SECDEC) code. The standard code is seven bits for a 32 bit word. The modification consists of adding an eighth bit which allows some three and four bit errors to be detected. By carefully arranging data and code bits within the "by four" DRAMs (each DRAM contains four bits of a data word), it is possible to detect a failure of any given DRAM as well as any double bit error.

Table 3-1 shows the ECC code used. The decode of single bit errors is shown in Table 3-2. The eight bit code is generated by exclusive-ORing the data bits together as specified in the table. For instance, bit six of the code, is the exclusive-OR of bits 31 to 24 and 7 to 0. Three of the bits in the code are the exclusive-NOR of the data bits listed in the table (code bits 2, 3, and 7). As can be seen from the table, bit 7 is the exclusive-NOR of all data bits; it is the parity across all 32 bits.

Table 3-1 ECC Code

	ECC Bit							
Data	7*	6	5	4	3*	2*	1	0
31	X	X	X	X		X		X
30	X	X	X	X				
29	X	X	X		X	X		X
28	X	X	X		X		X	X
27	X	X	X		X			
26	X	X	X			X	X	X
25	X	X	X			X		
24	X	X	X				X	
23	X			X	X	X		
22	X			X	X		X	
21	X			X	X			X
20	X			X		X	X	
19	X			X		X		X
18	X			X			X	X
17	X				X		X	X
16	X				X	X	X	
15	X		X	X		X		
14	X		X	X				X
13	X		X		X	X		
12	X		X		X		X	
11	X		X		X			X
10	X		X			X	X	
9	X		X			X		X
8	X		X				X	X
7	X	X		X	X	X		X
6	X	X		X	X		X	X
5	X	X		X	X			
4	X	X		X		X	X	X
3	X	X		X		X		
2	X	X		X			X	
1	X	X			X		X	
0	X	X			X	X	X	X
* These bits are XNOR'd, remainder are XOR'd								

Table 3-2 Single Bit Error Decode

Syndrome Decode	
Data Bit	Syndrome<6..1>
31	111010
30	111000
29	110110
28	110101
27	110100
26	110011
25	110010
24	110001
23	001110
22	001101
21	001100
20	001011
19	001010
18	001001
17	000101
16	000111
15	011010
14	011000
13	010110
12	010101
11	010100
10	010011
9	010010
8	010001
7	101110
6	101101
5	101100
4	101011
3	101010
2	101001
1	100101
0	100111

Every time data is written into memory, an eight bit code word is generated for this data as specified in the table above. When data is read from memory, the code word is read with it and both are checked for errors. Error checking consists of generating a new ECC code for the data read from memory and comparing that new code to the code read from memory by doing a bitwise exclusive OR of the new and old code. The result of this operation is called the syndrome.

If all bits in the syndrome are zero, the new code and the old code agree and no error is detected. If the syndrome is non-zero, some error has occurred, either a single bit, correctable error or a multiple bit, non-correctable error.

Table 3-3 Decoding Syndrome

Syndrome<7:0>	Single	Multi	Remark
00000000	0	0	No error
10000000	1	0	Error in ECC code<7>
01000000	1	0	Error in ECC code<6>
00100000	1	0	Error in ECC code<5>
00010000	1	0	Error in ECC code<4>
00001000	1	0	Error in ECC code<3>
00000100	1	0	Error in ECC code<2>
00000010	1	0	Error in ECC code<1>
00000001	1	0	Error in ECC code<0>
syndrome<7>=1 and syndrome<6:1> from Table 3-2	1	0	Single bit error, data bit listed in Table 3-2
if odd parity on syndrome<6:1> and:			Multiple bits in error...
XXX1111X	1	1	
XX1X111X	1	1	
XX11XX1X	1	1	
XX000X11	1	1	
XX111XXX	1	1	
XX00X101	1	1	
XX001X01	1	1	
XX010001	1	1	
XX00011X	1	1	
XX00110X	1	1	
Even parity on syndrome<6:1>	1	1	

Table 3-3 shows the procedure for decoding the syndrome. The procedure is complicated by the addition of the eight bit and the need to detect certain triple and four bit errors. If only a single bit is set, there was a single bit error in one of the ECC code bits. Otherwise, if syndrome bit 7 is one and the syndrome is found in Table 3-3, then the error was a single bit error and the bit in error can be found in that table. Remaining errors are multiple bit errors.

Note that during error detect, the hardware will flag a multiple bit error as both a single and a multiple bit error. This is because it is simpler to have the hardware assume any non-zero syndrome is a single bit error and then have it take additional measures if the error was a multiple bit error.

Subsequent sections will discuss when and how the ECC code is used.

3.1.1 Three and Four Bit Errors

As was mentioned earlier, the ECC code uses an extra bit beyond what is normally used for a thirty two bit data word in order to detect the failure of all the bits in a single DRAM. The DRAMs used on the NMCs are four bits wide. If a DRAM failed, perhaps with a failure in its address logic, zero, one, two, three or all four of the bits driven by that DRAM might be incorrect.

The modified ECC code uses an extra bit to cover some of the possible three and four bit errors in a thirty two bit word. By carefully grouping data and ECC bits into the four bit wide DRAMs, it is possible to cover all the three and four bit errors which correspond to the failure of a single DRAM. Not all possible three and four bit errors are detected, just those that correspond to an individual DRAM failing.

Thus, with the eight bit code, it is possible to correct any single bit error, detect any error involving any two bits, and to detect any three or four bit error in bits coming from the same DRAM.

The grouping for the data and ECC bits used in the NMCs is shown Table 3-4

Table 3-4 Data and ECC Grouping

DRAM 0	DRAM 1	DRAM 2	DRAM 3	DRAM 4
Data<29>	Data<30>	Data<25>	Data<22>	Data<19>
Data<28>	Data<27>	Data<23>	Data<21>	Data<18>
Data<1>	Data<26>	Data<16>	Data<7>	Data<15>
ECC<3>	ECC<0>	ECC<2>	ECC<4>	ECC<7>

DRAM 5	DRAM 6	DRAM 7	DRAM 8	DRAM 9
Data<17>	Data<13>	Data<9>	Data<24>	Data<31>
Data<10>	Data<12>	Data<8>	Data<20>	Data<5>
Data<6>	Data<11>	Data<4>	Data<14>	Data<3>
ECC<5>	ECC<6>	Data<0>	ECC<1>	Data<2>

3.2 The NMC

The NMC, the Neptune Memory Card, is the daughter card which plugs into the NMB. One NMC contains all the DRAMs for a single bank of memory. Since there are thirty two banks on the NMB, there are thirty two NMCs per NMB. In addition to the actual memory storage devices, the DRAMs,

the NMCs also contain the translators for converting to and from TTL and a gate array, the WREDC, for doing ECC on write data.

Figure 3-2 Schematic Representation of the NMC

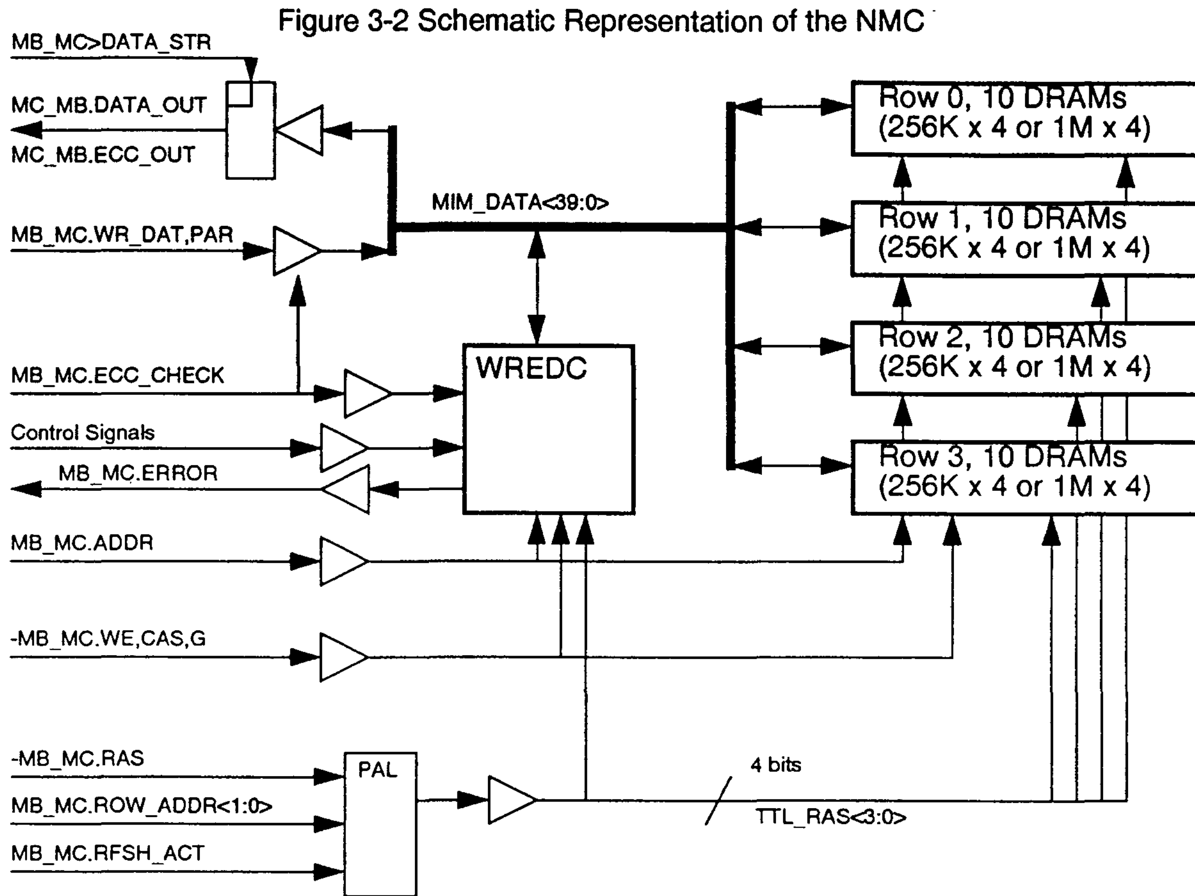


Figure 3-2 is a simplified schematic of the NMC. At the right side are the DRAMs. The forty DRAMs on an NMC are grouped into four sets of ten called rows. The DRAMs are either all 256K by 4 or 1M by 4. Since each DRAM is four bits wide, each row is forty bits wide: thirty two bits of data and eight bits of ECC. The rows share all control, address and data signals except for the RAS signals. Each row has its own RAS so that each row can be independently enabled. Since every other signal is shared, only one row can be used at one time.

The WREDC at the center of the drawing is a 5000 gate, CMOS gate array. It has a bidirectional receiver/driver port for the data bus and a number of control signals. It also receives all the signals going to the DRAM (shown entering the bottom side of the array). These last signals are used solely for diagnostics purposes. They are registered in the array and examined from scan so that the integrity of these signals can be determined.

The WREDC and the DRAMs are all TTL devices. To interface with the ECL signals on the NMB, it is necessary to translate these signal levels. The triangles in the figure represent 100602 and 100603 translators. The 100602s are TTL to ECL translators and the 100603s are ECL to TTL translators. All the translators in both directions have latches built into them. Only the latches on the DATA_OUT and ECC_OUT paths are used, however. The rest are always left transparent and are omitted from the figure for clarity.

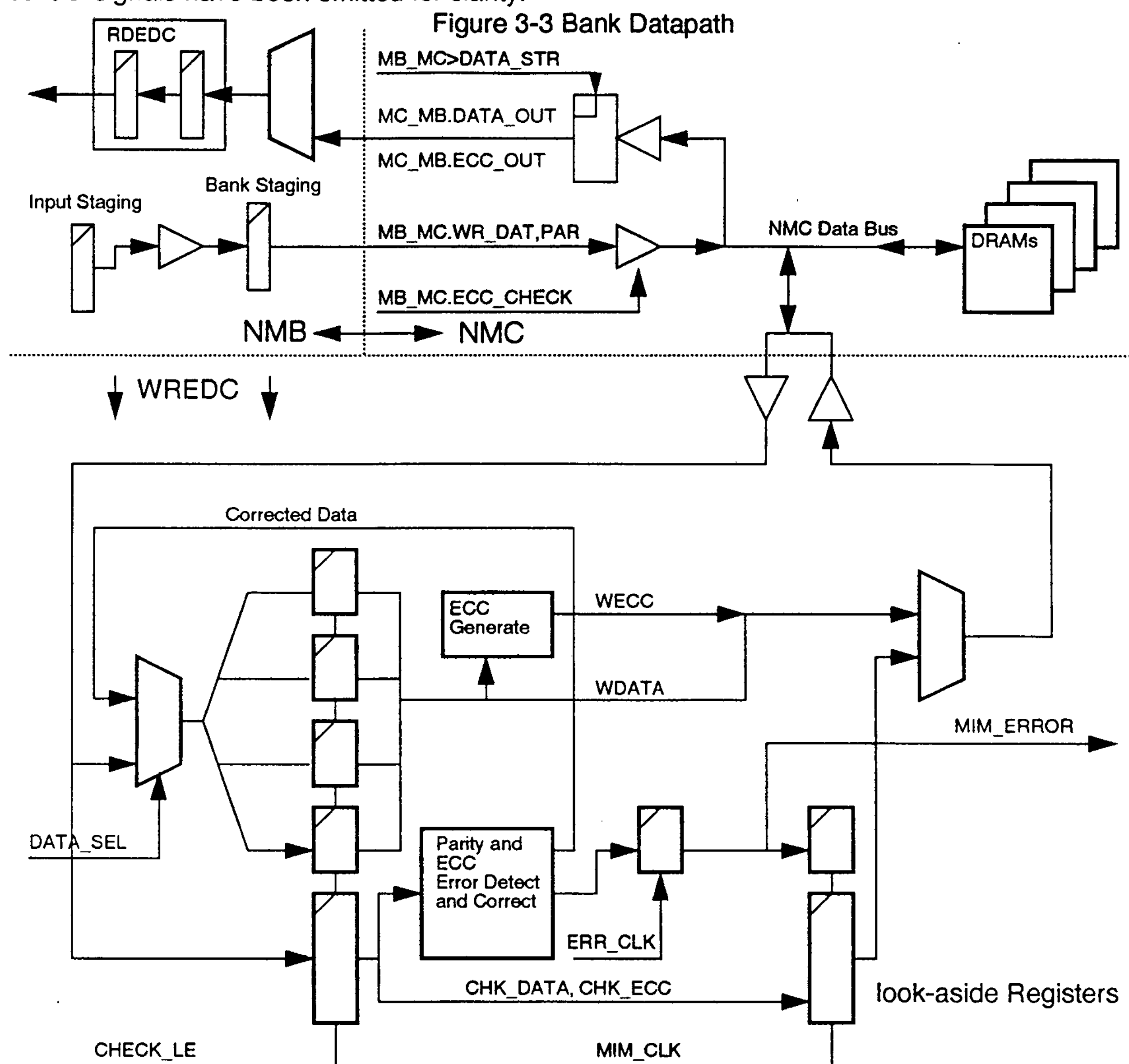
Each bit of the NMC data bus, shown at the top center of the drawing, has seven devices on it: four DRAMs, the WREDC, a 100603 and a 100602. How this bus is used will be discussed in the following sections which cover reads, writes, and read-modify-writes. The lower thirty two bits of this bus are data bits, the upper eight bits are the parity bits.

The individual RAS signals which go to each row of DRAMs is generated on the NMC from the single bit RAS signal, the ROW_ADDR bits and the RFSH_ACT bit. An ECL PAL is used to generate these signals. The RAS signal acts as the enable for the four row RASs. If RFSH_ACT is active, all row RASs go active with RAS. Otherwise, the row RASs selected by ROW_ADDR is asserted and the remainder are inactive. The PAL must be carefully programmed so that no timing hazards can occur and no glitches are generated on the RAS lines.

3.2.1 NMC/NMB Datapath

Figure 3-3 shows the datapath on the NMC and NMB for a particular bank. Address and most control signals have been omitted for clarity.

Figure 3-3 Bank Datapath



The upper left corner of Figure 3-3 shows the NMB logic for a bank. The input staging register and the bank staging register are the two registers shown in the box labeled ISE in Figure 3-1. The two registers in the box labeled RDEDC represent the staging registers in the RDEDC gate array. The multiplexor before the RDEDC is the READMUXE logic. In the upper right hand corner of this figure is the NMC logic exclusive of the WREDC. This logic consists of the translators with the latch in the data out path and the DRAMs.

The largest portion of the figure is the logic within the WREDC. The WREDC has a bidirectional interface on to the data bus on the NMC. The registers on the left side of the WREDC are the input staging registers. The upper four registers are the write data registers. Each register is a

separately writable, eight bit register. The write data registers can hold their data or, through the multiplexor before them, load data from the NMC data bus or load corrected data from within the NMC.

The write data registers are what are used to write the DRAMs. The block labeled ECC generate creates the ECC for the given data. The registers are grouped by bytes for read-modify-writes in which write data from the ISE logic is to be merged with pre-existing data read from the DRAMs.

Both read-modify-writes and corrected data will be discussed in more detail in subsequent sections.

The lower input staging register in the WREDC is the check data register. It holds both data and either parity or ECC depending on whether write data with parity or DRAM data with ECC is being loaded. It is this register on which ECC and parity checks are performed. In the case of an ECC correctable error, this register is used to generate the corrected data.

The MIM_ERROR register holds the result of the error check (either ECC or parity) which is sent back to the BCGA controlling the bank in question. The registers on the far right of the WREDC logic are the lookaside registers accessible from LOG scan. These registers freeze when an error is detected.

3.3 DRAM Operations

There are only a four possible operations that can be performed on the NMB DRAMs. These are reads, full writes, read-modify-writes, and refreshes. Read-modify writes are used for both test-and-modifies and partial writes. A special case of the read-modify-write is the scrub operation in which all zone bits are zero.

3.3.1 DRAM Full Writes

A full write is simply a 32 bit write to a bank. Since ECC is performed on 32 bits, there is no need to read the previous contents of the location being written. That location will be completely

overwritten. In a full write, the full 32 bit write data word received from the crossbar is written into the specified memory location.

Figure 3-4 DRAM Full Write Operation

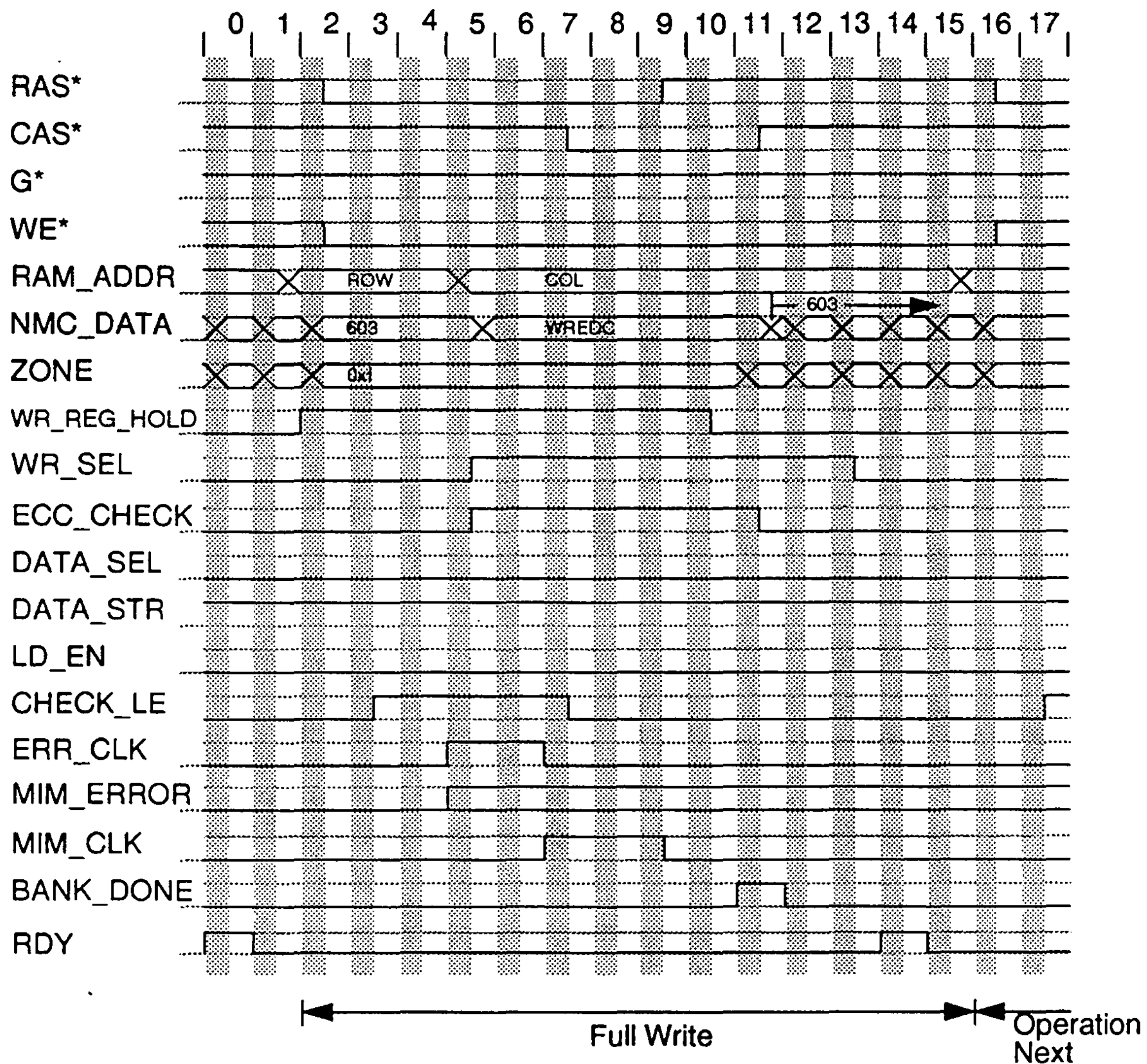


Figure 3-4 shows the timing for the relevant signals for a full write. All signal transitions are shown as instantaneously changing immediately after the clock edge which causes their change; that is, no delays are shown. For some signals, particularly address or data, the real signal propagation delay can be substantial. The rising edge of clock occurs between the unshaded and shaded regions in the figure. For instance, ERR_CLK rises with a clock edge.

RAS*, CAS*, G*, and WE* are the DRAM control signals at the DRAM. Their function will be described below. RAM_ADDR is the 10 bit address that goes to the DRAMs. (256K by 4 DRAMs ignore the tenth bit.) NMC_DATA is the TTL data bus on the NMC. It can be driven by the DRAMs, the WREDC or the 100603 ECL to TTL translators. The driver is shown on the timing drawing.

WR_REG_HOLD is the control signal from the BCGA to the ISE or ISO which causes the write data holding register to freeze. It is shown in Figure 3-1. BANK_DONE is the signal which goes from the BCGA to the XARB on the crossbar which indicates that a bank is finished with its

request. The signal labeled RDY is the RDY signal from the XARB to the BCGA which indicates that a valid request is on the bus from the crossbar.

3.3.1.1 Details of Full Write Timing

The full write cycle shown in clocks 2 through 15 of Figure 3-4 is initiated by the RDY at clock 0. On clock 0, the RDY, address, data, cycle, zone, and parity information is present on the signal lines between the crossbar and the NMB, the signals on the far left of Figure 3-1. On clock 1, this information is now in the registers labeled "IS" in this figure. On clock two, the information is registered in the bank controller the request is for. Write data, parity, zone and row address are also registered in the second register in the block labeled ISE in the figure. At this point, all data is properly staged for the request.

Several events occur at the beginning of the full write before RAS* is asserted. As soon as possible, on the half clock following the clock in which the request from the crossbar is registered in the IS registers, the row address is passed to the NMC. Because of pin limitations, DRAMs receive the address in two pieces. The first piece is called the row address, the second piece is called the column address. Together, they specify the entire address. For 256K by 4 DRAMs, the row and column addresses are each 9 bits. For 1M by 4 DRAMs, the row and column address are 10 bits each. (The two 9 bit addresses yield an eighteen bit address, 2^{18} gives 256K locations. Similarly, two ten bit address yield a twenty bit address, 2^{20} gives 1M locations.). The row address must be stable at the DRAMs before RAS* is asserted to meet setup to the DRAMs.

The NMC_DATA bus from the beginning of the cycle until clock 5 is being driven by the 100603 translators. Until clock 2, the registers driving these translators (the non IS registers in Figure 3-1) have been changing each clock. At the assertion of WR_REG_HOLD, these registers freeze. Thus, from clock 2 until WR_REG_HOLD is released, the inputs to the translators have the write data for the current request. The ZONE bits are also held WR_REG_HOLD. They only go to the WREDC for use in determining which bytes to load or hold during read-modify-writes. Since they are all one in this case, the BCGA determines that this operation is a full write operation and the ZONE bits are not used by the WREDC.

The row address bits used for decoding which of the four DRAM rows are active has timing identical to the ZONE bits for all types of DRAM operations. Row address is not shown in the timing diagram.

At the half clock in clock 2, RAS* is asserted. Asserting RAS* causes the DRAMs to go active; RAS* functions as a device select for the DRAM. RAS* also strobes in the ROW address present on the RAM_ADDR lines. There are four RAS* signals on the NMC, one for each of the four DRAM rows.

WE* is also asserted at the same time as RAS*. WE* is not timing critical, it only needs to make setup to CAS* going active. The half clock of clock two is simply a convenient time to assert the signal. WE* can begin the request either high or low depending on what the previous request was and is therefore shown in both states on the diagram.

The next event is the assertion of CHECK_LE at clock 3.5. CHECK_LE is one of the three clocks to the WREDC. The other two clocks follow CHECK_LE in Figure 3-4 and are ERR_CLK and MIM_CLK. As is shown in Figure 3-3, the Bank Datapath Figure, each of these clocks clocks a different set of registers in the WREDC. CHECK_LE clocks the write and check registers.

WR_SEL, a zero at this time, determines whether the write data registers of the WREDC will be loaded by byte, using the ZONE bits to determine which bytes are written (see Section 3.3.4 on page 65) or whether all bytes are to be loaded. At the beginning of all DRAM operations, all write data bytes are to loaded into the WREDC for the parity check. DATA_SEL determines whether the data bytes are to be loaded from internal to the WREDC or external. It is only set to a one (internal load) during data correct cycles (see Section 3.3.4.2 on page 69). So with DATA_SEL and WR_SEL both zero, the write data registers in the WREDC are loaded with the data and parity present on the NMC data bus which at this time is being driven by the 100603s which are driving the data in the bank staging registers in the ISE.

At the end of clock four, the ERR_CLK is asserted. From Figure 3-3, it can be seen that this clock only clocks the error register, the register which holds the results of the error check. During the time between the rising edge of CHECK_LE and ERR_CLK, the WREDC was performing a parity check on the write data which was registered at the rising edge of CHECK_LE. The WREDC knows to perform a parity check and not an ECC check at this time because the ECC_CHECK signal is unasserted at the rising edge of ERR_CLK. At ERR_CLK rising, the results of the parity check are registered in the register which drives the MIM_ERROR signal back to the BCGA.

On clock seven, MIM_CLK is asserted. MIM_CLK clocks the look-aside registers on the WREDC. These registers save the data and ECC if an error was detected. The registers are loaded each time MIM_CLK is asserted unless an error has previously been detected. If an error has already been detected, the look-aside registers will be frozen.

At the beginning of clock 4, the RAM_ADDR signals were switched by the BCGA from driving the row address to driving the column address. This is not a timing critical transition; column address need only make setup to CAS*.

At clock five and a half, ECC_CHECK is asserted which causes the WREDC to start driving write data. Since CHECK_LE, the WREDC has had time to generate the ECC for this write data. When CAS* is asserted at clock seven and a half, the DRAMs will write the data on the data bus into the address specified by the row and column addresses. At this point, RAS and CAS are asserted, the data is ready at the DRAM, WE* is asserted, indicating a write operation and G* is inactive indicating the DRAMs are not to drive data (since the WREDC must drive the data).

The write of data has now occurred. The remaining signal transitions are present either to return signals to their initial state for the next request or to complete timing requirements for valid DRAM operation.

RAS* is de-asserted at clock 9.5, shortly after CAS* assertion. Once CAS* is asserted, RAS* is no longer needed. It is necessary to de-asserted RAS* as quickly as possible in order to begin pre-charge operations within the DRAMs in preparation for the next DRAM operation (see a DRAM datasheet for details). At clock 16.5, RAS* is ready to be asserted again. The requirements for RAS* low followed by RAS* high are what determine the minimum cycle time for DRAM full writes, reads and refreshes.

CAS* must stay low for a certain period of time to satisfy DRAM timing requirements but this timing is much less critical to DRAM operation. By clock 11.5, this timing has been satisfied and the signal is de-asserted in preparation for the next operation.

WE* is unchanged at the end of all DRAM operations. In the case of a full write, it is left low to be changed only at the beginning of the next operation.

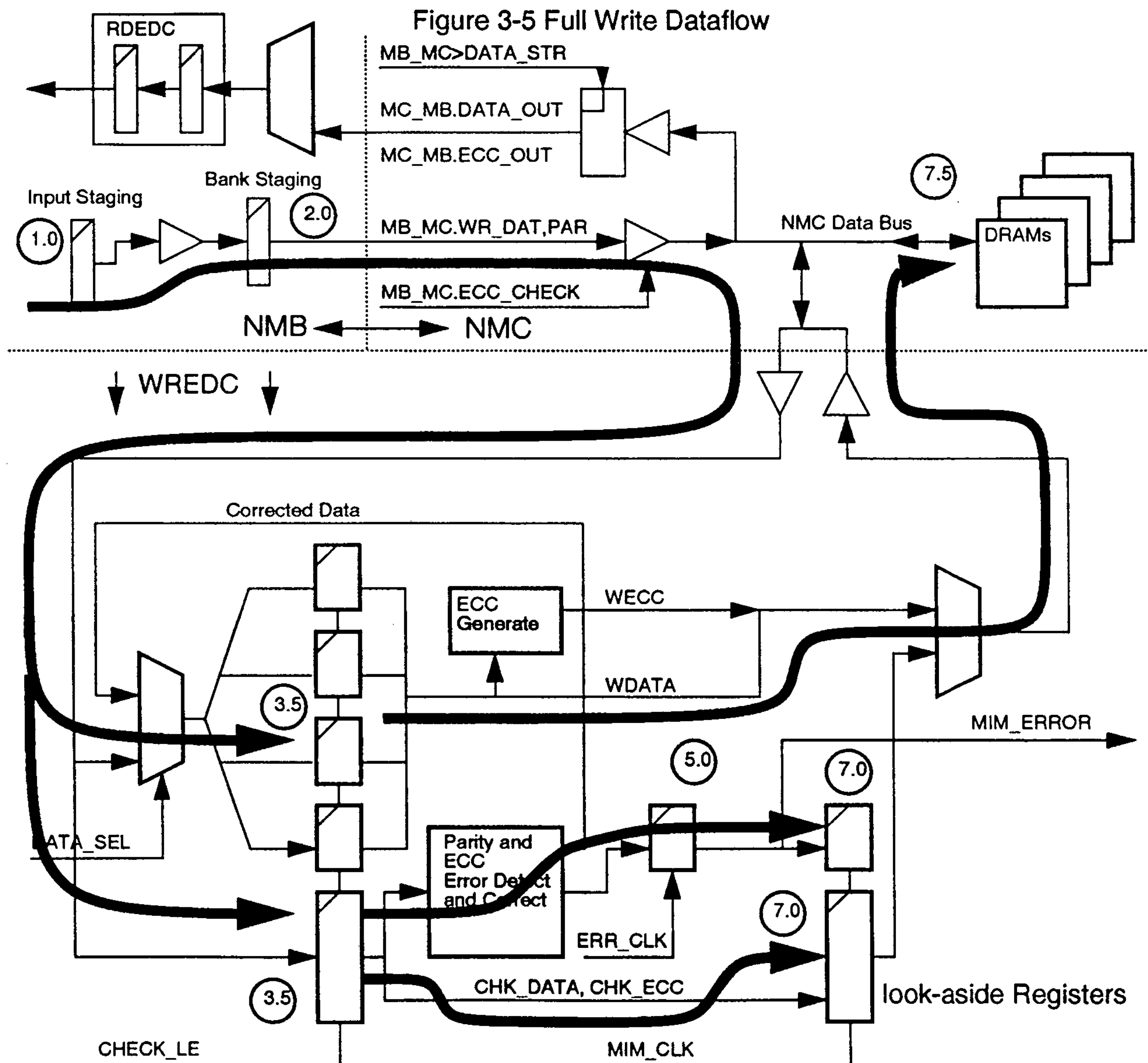
RAM_ADDR begins tracking the address presented by the crossbar at the end of the DRAM operation. It need only satisfy a small hold time after CAS* falling edge but is held for most of the cycle to avoid crosstalk problems.

As with the column address, NMC_DATA need only make a small hold time to CAS* fàlling. About the time of CAS* rising, WR_REG_HOLD and ECC_CHECK are de-asserted and the data bus begins tracking the input staging register. ECC_CHECK was asserted at clock 4.5 in preparation for a read-modify-write cycle. Its assertion has no impact on the full write.

DATA_SEL is only used in read-modify-writes and so is unchanged for this operation. DATA_STR, LD_EN, and G* are only used during reads and read-modify-writes.

The WREDC clocks, CHECK_LE, MIM_CLK and ERR_CLK are all de-asserted at various times. The falling edges of these signals are of no real consequence.

Figure 3-5 Full Write Dataflow



BANK_DONE is asserted as discussed in Section 2.1.1.3 on page 12 of the Interface chapter indicating to the crossbar that another request can be sent to the bank. Notice that the BANK_DONE occurs before the operation is really completed but that the next RDY to the bank does not arrive until just after the RAS* finishes precharging at clock 16. The BANK_DONE is advanced to allow for the pipeline stages between BANK_DONE and the next RDY.

In summary, a full write operation is relatively straightforward. RAS* is asserted to strobe in the row address and begin the cycle. The RAM_ADDR is changed to the column address and that address is strobed in by CAS*. Prior to CAS*, the write data has been registered into the WREDC by CHECK_LE and checked for parity errors. After the data has been driven from the ISE registers into the WREDC, the WREDC becomes the bus driver and drives the data plus the ECC it has generated for the data to the DRAMs. The data is stable at CAS* falling edge and is written into the DRAMs since WE* is asserted. The remainder of the cycle allows the DRAMs to complete their timing requirements and time to restore signals for the beginning of the next operation.

Figure 3-5 high lights the data flow for this operation. This figure is a modified copy of Figure 3-3 on page 53. Circled numbers indicate on which clock or half clock that data is registered in a particular register. In short, data is written into the WREDC, then it is driven from the WREDC to the DRAMs.

3.3.1.2 Full Write Error Checking

For full writes, the only error checking performed is the parity check on the write data registered on clock 3.5 in the WREDC. Error checking in the WREDC consists of registering data in the check register, allowing sufficient time for the error logic propagation and then clocking the error register with ERR_CLK. The error check logic can check for either parity errors or ECC errors as determined by the value of the ECC_CHECK signal at the rising edge of ECC_CHECK. (ECC_CHECK is internally registered in the WREDC; this register is not shown.)

Following the rising edge of ERR_CLK, MIM_ERROR is driven back to the BCGA to be registered in the bank control logic for this bank. The Interface chapter has the timing for asserting a hard error if a parity error is detected at this time.

When any error is detected by the WREDC, a parity error or an ECC error to be discussed later, the data and parity (or ECC) is saved in the lookaside registers. Any subsequent MIM_CLKs are ignored until the lookaside registers are scanned during a LOG scan and the error bit in the lookaside register is cleared.

It should be noted that the clock to output delay on the MIM_ERROR signal is on the order of 25 to 30ns, or one and a half to two clocks. The MIM_ERROR signal is only asserted if an error is detected for the current request and will be set to zero on the next ERR_CLK if no further errors are detected.

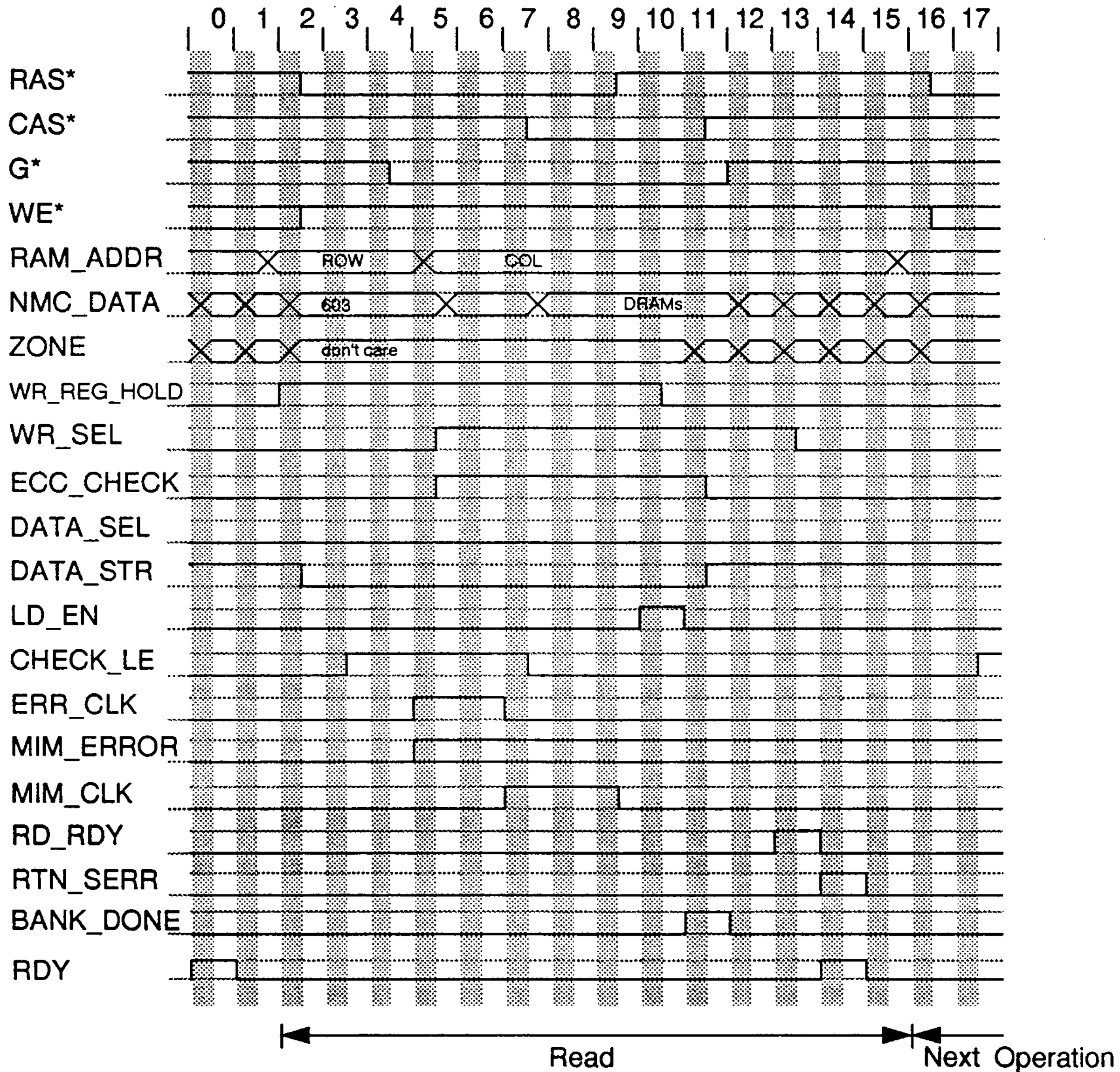
3.3.2 DRAM Reads

The read operation is a request with a cycle code of one. The request is asking for the contents of the DRAM cell specified by the address. It is similar in many ways to the full write operation: both have identical RAS, CAS, RAM_ADDR, BANK_DONE and WREDC clock timing.

As with the write timing diagram, all signal transitions are shown as zero delay. Real signal propagation delays are often very significant.

All signals listed in Figure 3-6 are the same as those in Figure 3-4 except for the addition of RD_RDY and RTN_SERR. The RDY signal is the MB_XRT.RD_RDY signal discussed in Section 2.1.2 on page 15 of the Interface chapter. It indicates to the XRT that the NMB has return data ready. The XRT knows when it should be expecting return data and uses this signal as an error checking signal. It will pull a hard error if it sees a RD_RDY it does not expect or does not see a RD_RDY that it expects.

Figure 3-6 DRAM Read Operation



A read cycle takes the same amount of time as write cycle. The figure above shows a fourteen clock cycle time.

3.3.2.1 Details of READ Operation

All signals in Figure 3-6 are identical in operation to Figure 3-4 on page 55 except for WE*, G*, NMC_DATA, DATA_STR, LD_EN, RTN_SERR and RD_RDY. All other signals operate as for the

write cycle. The RAS, and CAS signals strobe in the row and column address; the WREDC clocks occur at the same time; write data parity error checking is identical; and BANK_DONE timing is identical.

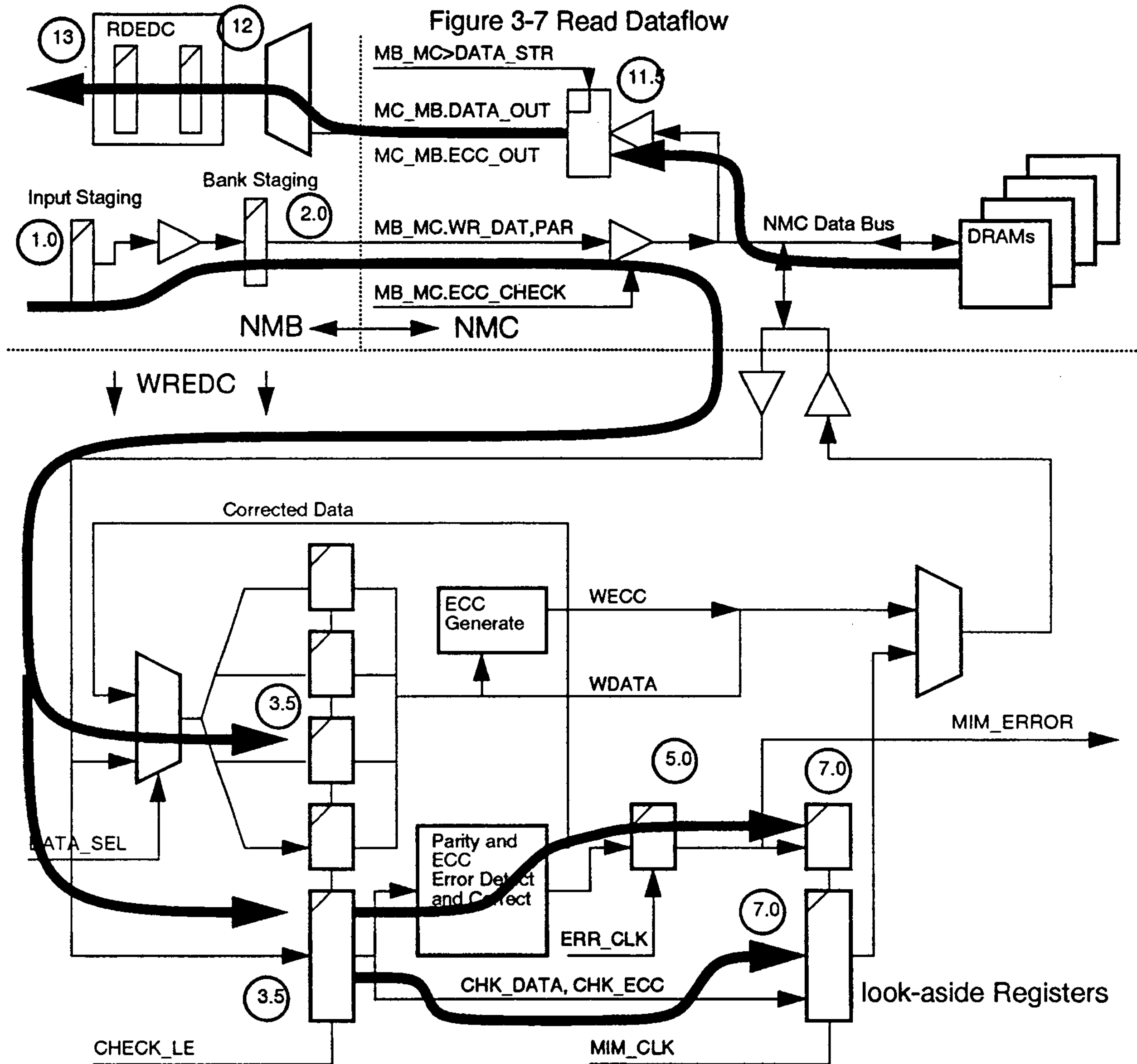


Figure 3-7 Read Dataflow

The differences, obviously, have to do with reading data from the DRAM rather than writing data into the DRAM. At clock 2.5, WE* is set to one whatever it's pervious state may have been. This tells the DRAMs, that the operation initiated by the RAS* will not be a write operation. Write data from the ISE and the WREDC is therefore superfluous for this operation. Nonetheless, the WREDC still receives and checks write data since it is simplest to always have the WREDC perform this function regardless of what operation is being used.

G*, the DRAM output enable is asserted at clock 3.5 to all four rows of DRAMs. Only the row with its RAS* asserted will drive data onto the NMC_DATA bus and only once CAS* is asserted. The NMC_DATA bus is undriven between ECC_CHECK being de-asserted and CAS* being asserted.

AT CAS* falling, the selected DRAM row writes the contents of the cell selected by the row and column address onto the data bus. The data is valid by clock 11.0.

At clock 11.5 DATA_STR is de-asserted. DATA_STR is the latch enable for the latch in the 100602 translators at the top of Figure 3-3. Once the signal goes high, the latches close, holding whatever data was present at the latch before it closed. In this case, that data is the data which was read from the DRAMs. Once the data is captured in the latch, CAS* can be turned off in preparation for the next DRAM operation while the NMC still drives the last read data out of its translators. This data will be held until the next read operation (note that DATA_STR was left one during the WRITE operation). DATA_STR was set to zero at clock 2.5 in preparation for the read.

At clock 10, the BCGA controlling this bank asserted its LD_EN and RTN_SEL signals. As described in Section 2.2.5 on page 37 of the Interface Chapter, these signals tell the RDEDC that data is ready and the READMUX which NMC to select for return data. LD_EN is registered in the LDREG logic block on clock 11. On clock 11, the return data from the NMC is selected by READMUX and clocked into the first register of the RDEDC on clock 12. During clock 12, the data and ECC is checked for single and multiple bit errors. On clock 13, data is sent to the crossbar with good parity. The presence of any errors are noted on the RTN_SERR and RTN_MERR lines from the RDEDC to the BCGA on clock 14. Only RTN_SERR is shown in the diagram but RTN_MERR timing is identical to RTN_SERR.

G* is de-asserted at clock 12.0 in preparation for a possible read-modify-write cycle. (Again, it is simpler to have G* always be de-asserted at this point than to be de-asserted only for a read-modify-write.)

Figure 3-7 shows the data flow for the read operation. The flow into the WREDC registers is identical to that of the full write cycle. However, instead of the WREDC driving into the DRAMs has in the write cycle, the DRAMs drive the NMC_DATA bus. The data bus is latched in the 100602s and then registered in the RDEDC and sent to the crossbar. The propagation delay from the 100602 to the RDEDC is greater than a half clock in most cases. The latching in the 100602s is mostly for single step operation.

3.3.2.2 Read Error Checking

Errors are checked at two points during a read operation. The first point is the write data parity check. This check is identical to the check made during a write cycle; see Section 3.3.1.2 on page 59 for details. The second check is the ECC check performed by the RDEDC.

The RDEDC performs an ECC check as described in the ECC section, Section 3.1 on page 46. The check takes place between the RDEDC's input and output registers. If a single bit error is also detected, the error is corrected at the same time. There is no delay necessary to correct data in the RDEDC as there is in the case of read-modify-writes.

On the clock following the RD_RDY, the RDEDC will assert RTN_SERR if any error was detected and assert RTN_MERR if a multiple bit error was detected. There is a pair of these signals for the even side BCGAs and a pair for the odd side BCGAs. The gate array which asserted the LD_EN for the return data for which an error was detected will check the RTN_SERR and RTN_MERR signals for errors and act accordingly.

In all cases, the data sent to the crossbar should have good parity regardless of RTN_SERR or RTN_MERR.

When the RDEDC detects a soft error, it corrects the error on the data sent to the crossbar as return data. It can not, however, correct the data in the DRAMs. Instead, the NMB asserts soft error as described in Section 2.1.4 on page 16 of the Interface chapter. The service processor then reads the log ring as described in Section 3.4.1 on page 74 of this chapter to determine at what location the error occurred. It then performs a scrub operation to that location to correct the error. The scrub operation is a special type of read-modify-write.

3.3.3 Refreshes

Refreshes are needed to keep dynamic RAMs (DRAMs) from losing the contents of their memory. The refresh restores the charge used to represent a stored bit. Refreshes must occur at the rate of 512 every 8ms for 1M and 4M DRAMs with 512 different addresses being applied during the 8ms period. The BCGA simply generates a count of 512 for refresh.

The refresh cycle is a subset of the full write or read cycle. A refresh can be thought of as one of these to operations without the assertion of CAS*. To keep the hardware simply, many signals are used as they normally are doing reads and full writes even though they have no effect on a refresh.

Figure 3-8 shows the signal transitions for a refresh. CAS* is never asserted, G* is left high and WE* is set high. Only the refresh address is presented; there is no need for a column address. Otherwise, the request looks like a full write. Write data parity is still checked and a hard error can still be detected. The BANK_DONE occurs at the same time as for full writes and reads.

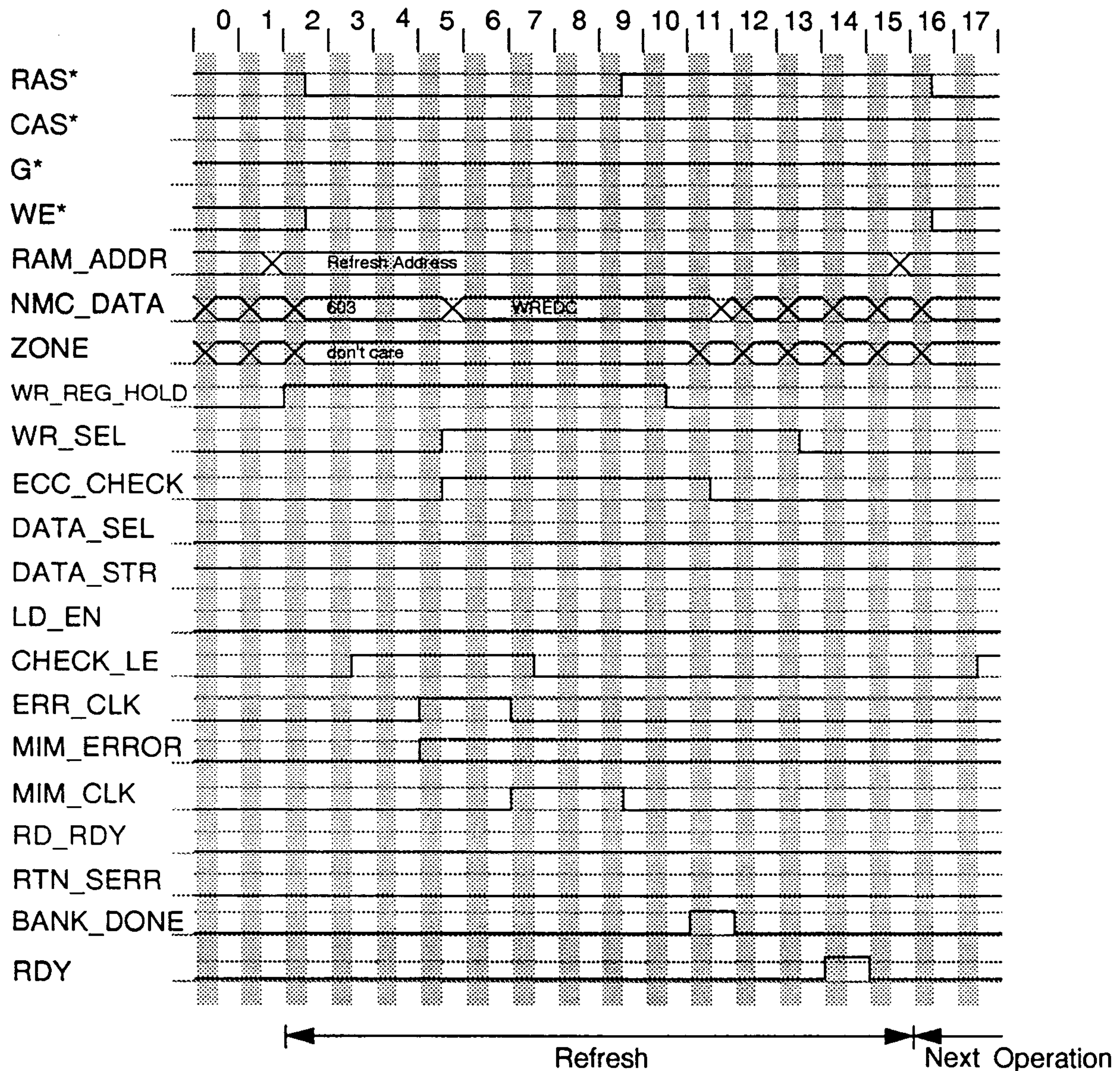
In the case of a refresh all four RAS* signals (one to each row of DRAMs) are asserted unlike the read or write cycle in which only a single RAS* to a particular row as selected by the row address is used.

Error checking on a refresh is identical to that for a full write, see Section 3.3.1.2 on page 59 for details. Data flow consists only of the write data going into the WREDC registers as is shown at the bottom of Figure 3-5.

A request can follow a refresh just as it normally follows another DRAM operation. If the crossbar asserts the refresh request signal while a bank is still busy with another operation, the refresh

follows immediately after completion of the first request just as if one request were following another request.

Figure 3-8 DRAM Refresh Operation



3.3.3.1 Refresh Initiation

Unlike other DRAM operations which are initiated by a one on the RDY lines from the crossbar, refreshes have a different start mechanism.

As discussed in Section 2.1.1.4 on page 13 of the Interface chapter, the XS0E normally initiates a refresh by asserting XSE_MB.REF_REQ. The XS0O board uses XSO_MB.REF_REQ to start the odd side refreshes. Details of the refresh with respect to system timing are discussed in this section. With respect to Figure 3-8, the REF_REQ which would cause RAS* to be asserted at clock 2.5 would occur at clock "-2."

One the -XC_MB.SYS_RUN bit is de-asserted, the crossbar is always halted and the memory board must get its refresh initiation signal from some other source. As described in Section 2.1.5.6 on page 22 of the Interface chapter, this other source is XC_MB.RAM_RFSH. This signal is basically a "raw" refresh that the NMB can use to start refreshes when the XS boards are halted.

In both cases, REF_REQ and RAM_RFSH refreshes, the DRAM operation proceeds identically as shown in the refresh diagram. In the case of a RAM_RFSH refresh, however, the time between the RAM_RFSH and the RAS* assertion for the refresh can vary. Any RAM_RFSH occurring within 64 clocks of SYS_RUN being de-asserted will postponed until 64 clocks have passed since the SYS_RUN event. If 64 clocks or more have transpired, then the RAM_RFSH will occur four and a half clocks before RAS* is asserted. In other words, the RAM_RFSH occurs at clock "-4" in Figure 3-8.

No BANK_DONE is issued for a refresh while SYS_RUN is inactive since the BANK_DONE logic is also inactive at this time.

3.3.4 Read-Modify Write DRAM Operations

The read-modify-write (RMW) cycle is the most complex of the DRAM operations. It is used for both partial writes (writes of less than 32 bits: the ZONE bits are not all ones) and test-and-modifies. In both cases, the contents of the addressed location must first be read before the location can be written The read-modify-write operation uses a special mode supported by all standard DRAMs. It allows a DRAM to be first read then written all during one assertion of RAS*. The read-modify write functions basically as a normal read followed by a write at the point at which RAS* would normally be de-asserted. The dataflow for this operation relies heavily on the WREDC for checking and correcting any errors as well as for generating ECC for write data.

The only difference between a test-and-modify and a partial write is that return data is read from the NMC at the point where it is read in the read operation previously described. Otherwise, the two are identical. In the following description of the read-modify-write, the transitions for the test-and-modify are shown. For a partial write, simply omit the assertion on the return data signals and ignore the RDEDC dataflow.

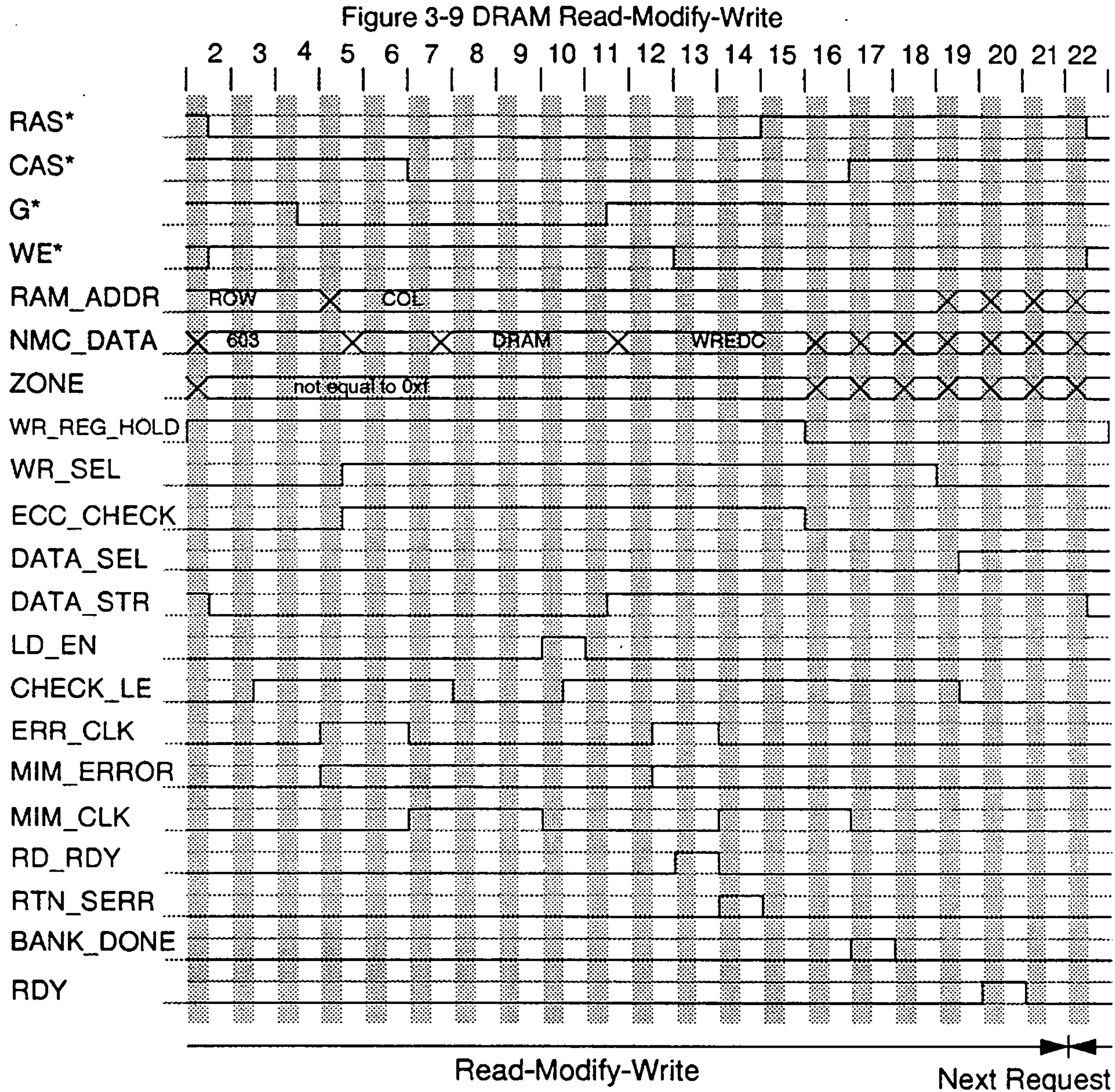
The read-modify-write differs from a full read in the way it handles a soft error. In a full read, the soft error is simply noted and the service processor must read the log ring to find where the error was and perform a scrub operation to fix the error. In a read-modify-write, the WREDC is used to correct the data. It must correct the data because this data is being merged with write data to form a new word with new ECC. Once the new ECC is generated the existence of an error is lost if the correct data is not used.

3.3.4.1 Details of a Read-Modify-Write

The read-modify-write begins exactly as a normal read operation. There is no difference in the read-modify-write until clock 9.5 where RAS* is de-asserted in the read operation but left active in

the RMW operation. Because of space constraints clock 0 and 1 are omitted from Figure 3-9. These two omitted clocks are identical to clocks 0 and 1 in Figure 3-6 on page 60.

Figure 3-9 DRAM Read-Modify-Write



The primary difference in the RMW operation is that RAS* is held low beyond the point at which the data has been read from the DRAM. G* is de-asserted so that the DRAMs are no longer driving data and then WE* is asserted once the WREDC has had time to drive its data onto the NMC data bus. With the assertion of WE*, the read has become a write, the modify portion of the read-modify-write. As with the other operations, RAS* is de-asserted before CAS* to prepare the DRAM for its next request (pre-charge).

At clock 10.5, the WREDC registered the DRAM's data in its write data and check data registers. At this second rising edge of CHECK_LE, WR_SEL is high (since clock 5.5). With WR_SEL asserted and DATA_SEL still zero (DATA_SEL's usage will be discussed later), the write data registers of the WREDC use the ZONE bits to determine which bytes to load. At the rising edge

of CHECK_LE, the write register contains the full 32 bit write data word. Only those bytes whose corresponding ZONE bits are one should be held. The remaining bytes must be loaded with the contents of memory read during the first part of the RMW since the write bytes are to be merged with existing data. After the CHECK_LE edge at clock 10.5, the write data register contains the write data merged with the data which was not to be written and the check register contains the full thirty two bit data word and ECC read from memory.

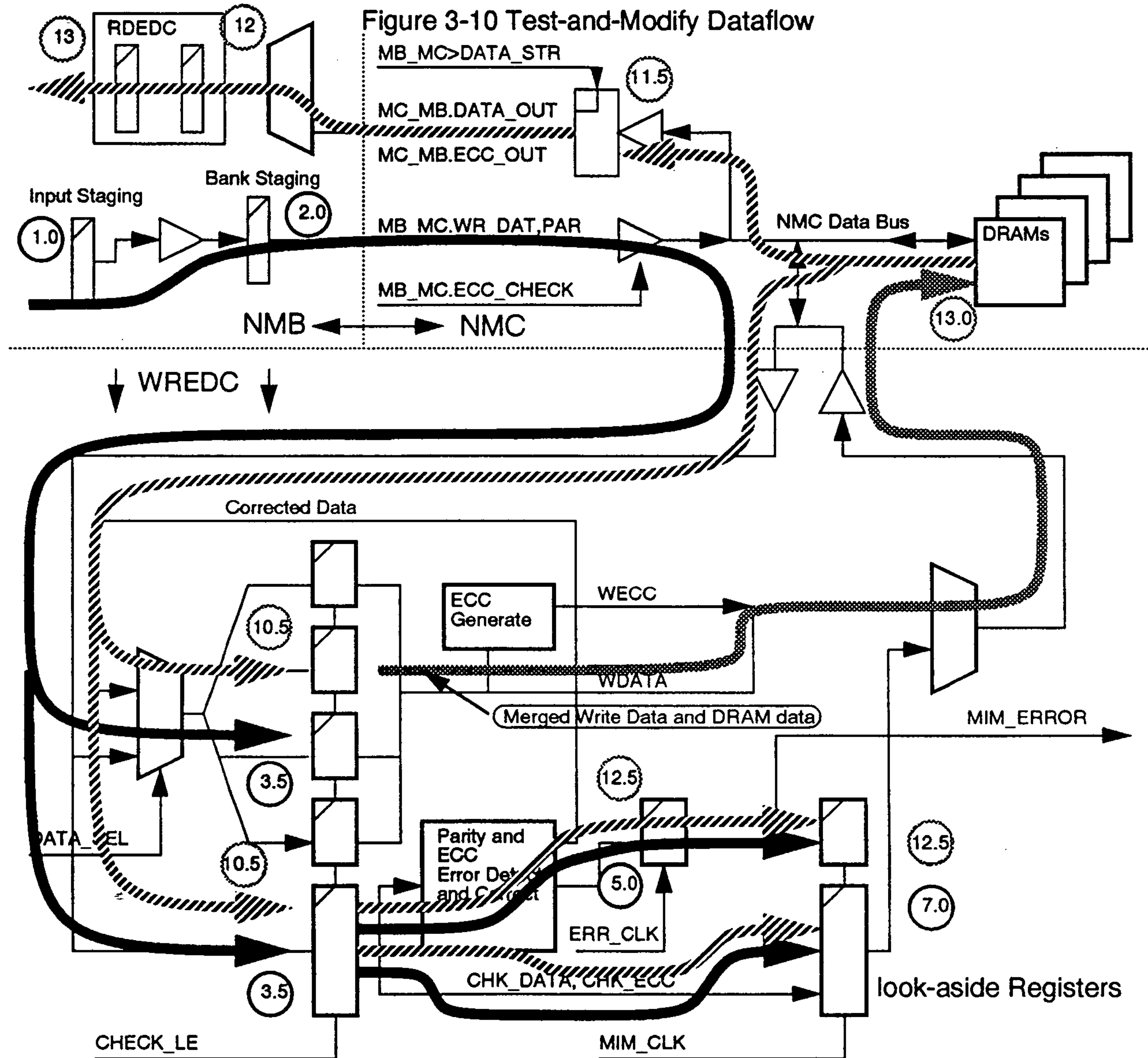


Figure 3-10 Test-and-Modify Dataflow

From clock 10.5 on, the WREDC is generating the ECC for this new write data word and starting at clock 11.5, driving it onto the NMC data bus.

While the write is occurring, the WREDC is checking the ECC on the word read from memory. It has until the rising edge of ERR_CLK at clock 12.5 to complete the ECC check. The WREDC detects for ECC errors rather than parity errors at this time because ECC_CHECK is now asserted, telling it to check for ECC errors.

Notice that the ECC check takes longer than the parity check and there is an extra half clock between CHECK_LE to ERR_CLK at this time. If the WREDC detects any error the MIM_ERROR signal will be asserted following ERR_CLK. The BCGA will see MIM_ERROR and know that some error has occurred. It will not know whether the error is a soft error or a hard error. See Section 3.3.4.2 on page 69 for the details on how an error is handled.

Also note that the data has been written into the DRAMs before it is known whether there was an error in that data word. If an error was detected, the BCGA must redo the RMW operation this time using the corrected data. The reason potentially corrupt data was used rather than waiting for the checked data is that the data check takes many clocks and corrupt data should be a very rare event, perhaps on the order of one piece of bad data per several months. Therefore it is far better on a performance point of view to spend the extra time only when there is an error rather than make every RMW pay the cost of an error. It is assumed that the data was correct and if it was not, the time is taken to fix the error.

As with the earlier parity check, MIM_CLK follows the ERR_CLK so that any error data can be saved in the look-aside registers. At each MIM_CLK rising edge, the look-aside register is loaded with the contents of the check register. At the first MIM_CLK, the check register contains the data being driven from the staging registers in the ISE/ISO logic. At the second MIM_CLK, the check register holds the DRAM read data. The look-aside register also saves the state of the ECC_CHECK signal so it can be determined what type of error caused the data to be saved in the look-aside register.

Since this is a longer operation than read, full write or refresh, the BANK_DONE signal occurs later. For an RMW, BANK_DONE occurs at clock 17 rather than clock 11. The RMW takes six extra clocks. BANK_DONE is only asserted if there was no MIM_ERROR at the second ERR_CLK. If there was an error, the BCGA must correct it. To do so, it repeats the RMW and only issues the BANK_DONE after the second RMW is complete. RMWs therefore take either 20 or 40 cycles depending on whether an error was detected.

Figure 3-10 shows the dataflow for the test-and-modify operation. The read-modify-write operation is very similar except that no data flows through the latch and into the RDEDC. The different stages of dataflow are indicated by different styles of lines in the figure. The bold black line going from the ISE logic into the WREDC at clock 3.5 and on to the error and look-aside registers is the initial write data transfer which is identical to what happens at the beginning of reads, full writes and refreshes.

The next data flow is from the read data being driven by the DRAMs. It is indicated by the hashed, heavy lines. The data goes from the DRAMs, out the translators and into the RDEDC as previously described for a read operation. The timing here is unchanged. The data also goes back to the WREDC into the write data and check registers. From the check register, it then goes to the error register and the look-aside register just as the write data did early in the operation.

In the write register, the byte writable registers in the upper portion of the WREDC, the DRAM read data is merged with the write data transferred from the ISE at the beginning of the cycle. This merged data is indicated by the shaded arrow going from the write register, through the ECC generator and onto the NMC data bus and into the DRAMs.

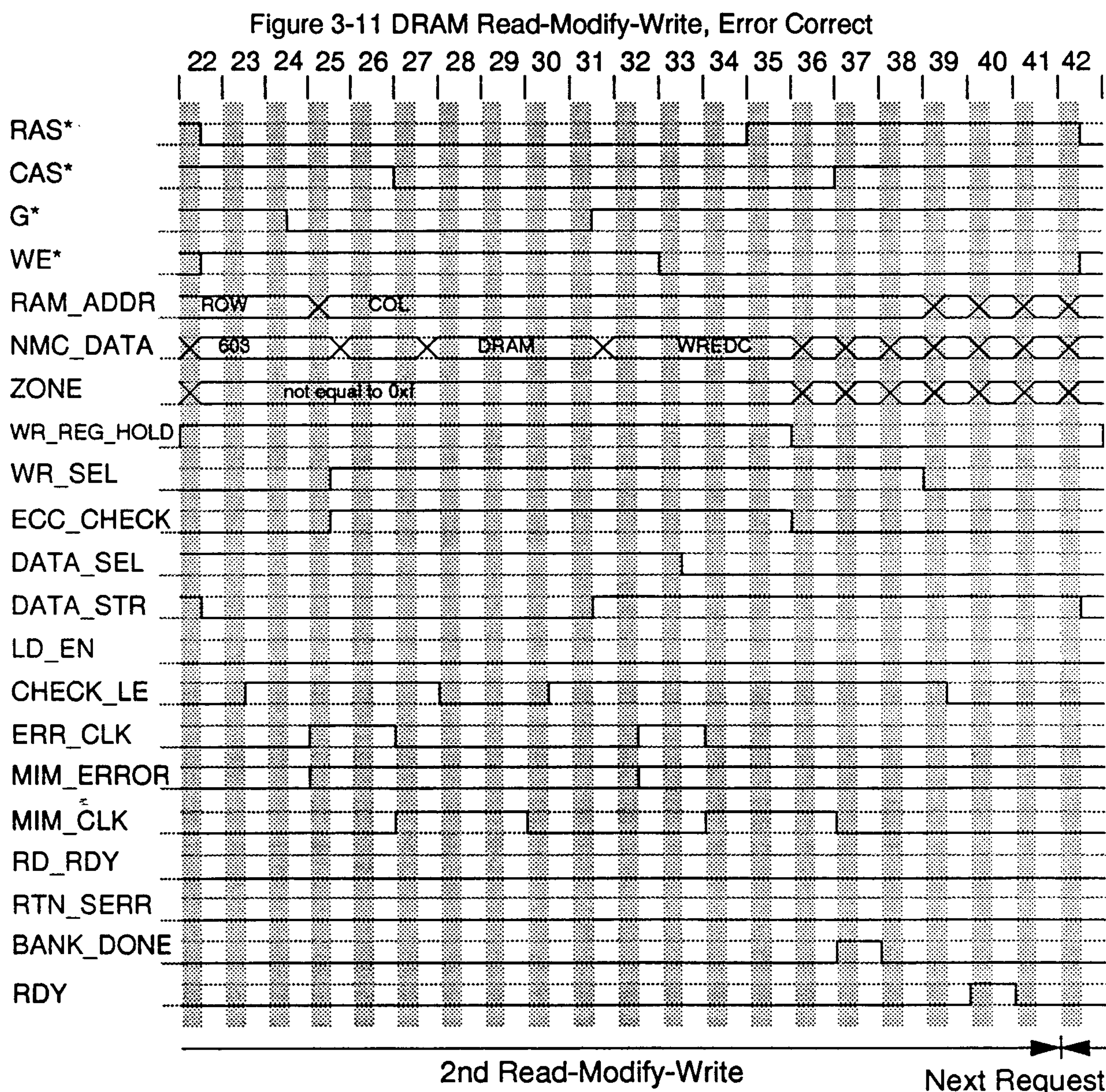
The circled numbers again indicate which clock data is registered. For registers with two flows in the operation, there are two circled numbers. The shaded circle corresponds to the shaded or hashed line and the solid circle corresponds to the solid line.

The DATA_SEL signal is only assert at clock 19.5 if there is a MIM_ERROR following the second ERR_CLK at clock 12.5. Its use is described in the next section.

3.3.4.2 Read-Modify-Write Errors

There are several sources of errors during a read-modify-write. The first source is the same as for any operation: the write data parity check. If MIM_ERROR is asserted at clock 7 following the first ERR_CLK, a write data parity error was detected and the BCGA will assert hard error as it would for any DRAM operation.

Figure 3-11 DRAM Read-Modify-Write, Error Correct



In the case of a test-and-modify where data is being returned to the crossbar through the RDEDC, the RDEDC can detect single bit and multiple bit errors just as in the case of a read operation. Timing for these errors is the same as has been discussed in Section 3.3.2.2 on page 62. It should be noted that if the RDEDC detects an error on a test-and-modify and the WREDC does not or

vice versa, there is a board flaw in the NMB rather than a simple DRAM soft error data corruption. Both WREDC and RDEDC should see the same errors in a read-modify-write. An occasional dropped bit detected by the RDEDC but not the WREDC is an indication of a bad data bit somewhere between the NMC and the RDEDC not a soft error problem.

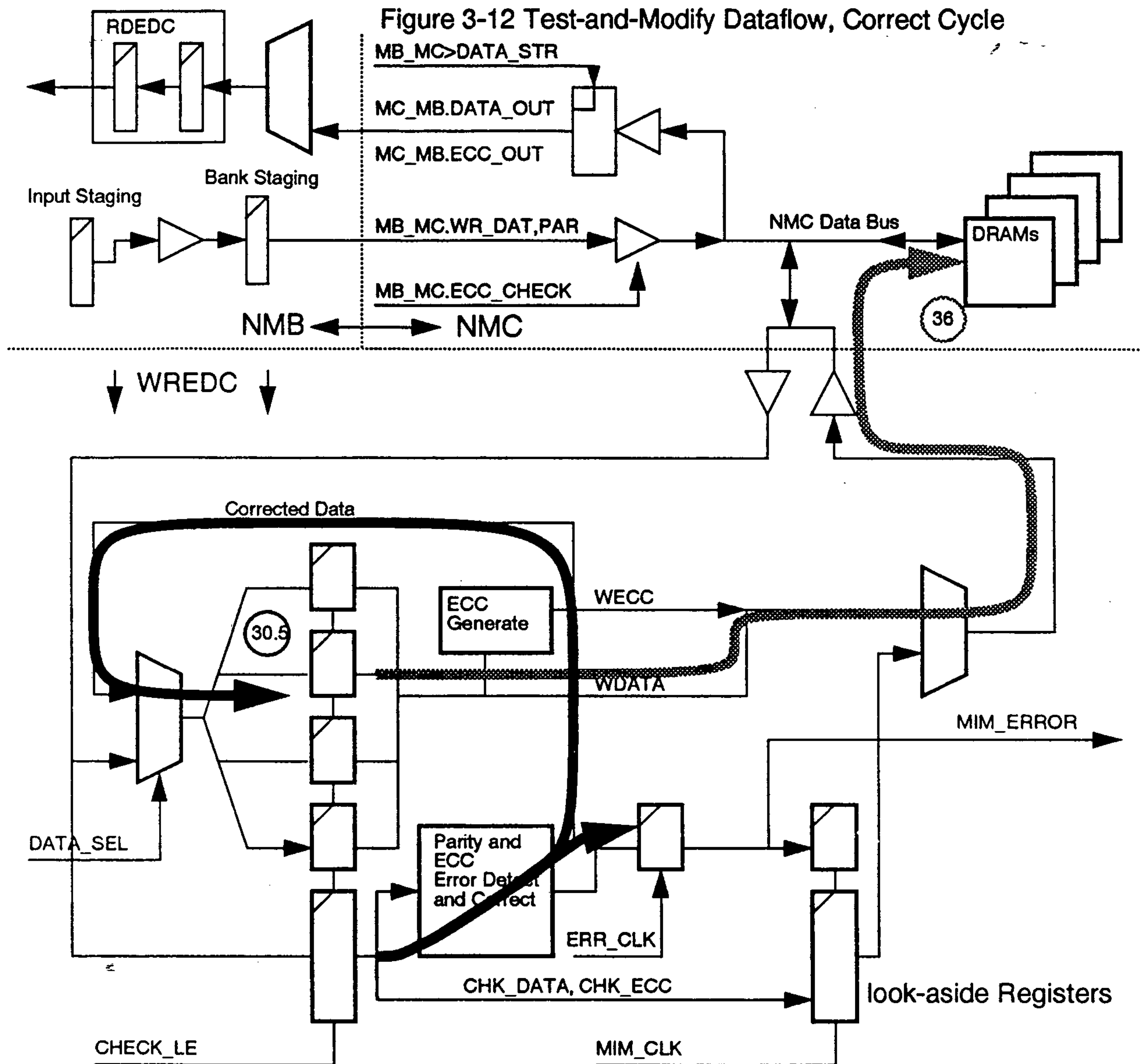


Figure 3-12 Test-and-Modify Dataflow, Correct Cycle

The final source of errors is unique to the RMW. These are errors detected by the WREDC for both partial writes and test-and-modifies. When the DRAM data is registered in the WREDC, it is checked for ECC errors. The error checking is in parallel with the data write. If an error is detected, some action must be taken to overwrite the incorrect data which has already been written into the DRAM with the corrected data. This is done by performing a second RMW operation with a different dataflow through the WREDC.

For this second operation, the write data and the DRAM data is already in the WREDC. The RMW is repeated with the DATA_SEL signal asserted. By asserting this signal, write data is not loaded again from the ISE logic. Instead of loading DRAM data on clock 10.5, the corrected data

generated from the check register at the bottom of the WREDC is used. The check register itself is not loaded at clock 10.5. When the write of the DRAM is executed again on the clock 36, the corrected data is written.

Figure 3-11 shows the timing for the second RMW operation used to correct data. This operation is tacked on to the first operation if MIM_ERROR is asserted after the second ERR_CLK, otherwise this extra operation does not occur. If the MIM_ERROR was asserted, DATA_SEL would be asserted on clock 19.5. With DATA_SEL high, write data is not reloaded at clock at the first rise of CHECK_LE. At the second rise of CHECK_LE, data is taken from the correct path as is shown in Figure 3-12 rather than from the DRAMs as in the first pass.

Aside from the change in dataflow caused by the assertion of DATA_SEL, the second operation is the same as the first. The clocks have been numbered sequentially from the first cycle so that the timing between the two can be compared. From the diagram, it can be seen that the second operation begins where the next request would begin if an error was not detected.

Error checking is different for the second RMW. Any parity error detected by the WREDC at the first ERR_CLK is ignored by the BCGA since the check registers are frozen holding data and ECC which is unlikely to have good parity (good parity is not good ECC).

After the second ERR_CLK, the MIM_ERROR signal is only active if the WREDC has detected a multiple bit error. The BCGA checks this MIM_ERROR signal and if it sees a one, it knows that the error which caused this second RMW operation was a multiple bit error and therefore a hard error. Remember, when MIM_ERROR was asserted in the first RMW operation, it only meant that some ECC error was detected, either a single or multiple bit error. The WREDC uses DATA_SEL to determine whether to assert MIM_ERROR for any error (DATA_SEL equal 0) or MIM_ERROR only for multiple bit errors (DATA_SEL equal 1).

Therefore, a single bit error is indicated by a MIM_ERROR equal one on the first RMW and a zero on the second operation. A multiple bit error is indicated by a one on MIM_ERROR for both RMWs. The BCGA performs the checking of the MIM_ERROR signal.

This double duty on the error signal was done to save pins on the NMC connector and time within the WREDC. By waiting until the second RMW to see whether the error was a multiple bit error, the WREDC has twenty extra clocks to propagate the error signal, more than enough time.

On the second RMW, an error has already been detected so the lookaside registers are already frozen and will not load again until scanned during a LOG scan. The error register will clock again to indicate whether the error was a multiple bit error.

Notice that LD_EN and RD_RDY are not asserted for this correction cycle. The RDEDC has already received the data during the first RMW and performed the error correct and detect. If the error was correctable, the RDEDC would have corrected it and sent the data to the crossbar. Therefore there is no need for the RDEDC to receive data again during the second RMW operation.

3.3.4.3 Scrub Operations

The scrub operation is a special case of the partial write operation. It is used to correct single bit errors which have been detected during a read. Single bit errors detected during a RMW operation are corrected as part of that operation as was described in the previous sections. Single bit errors

detected during reads are left uncorrected until the SPU reads the LOG ring to determine at what address the error occurred then performs a scrub operation on that location.

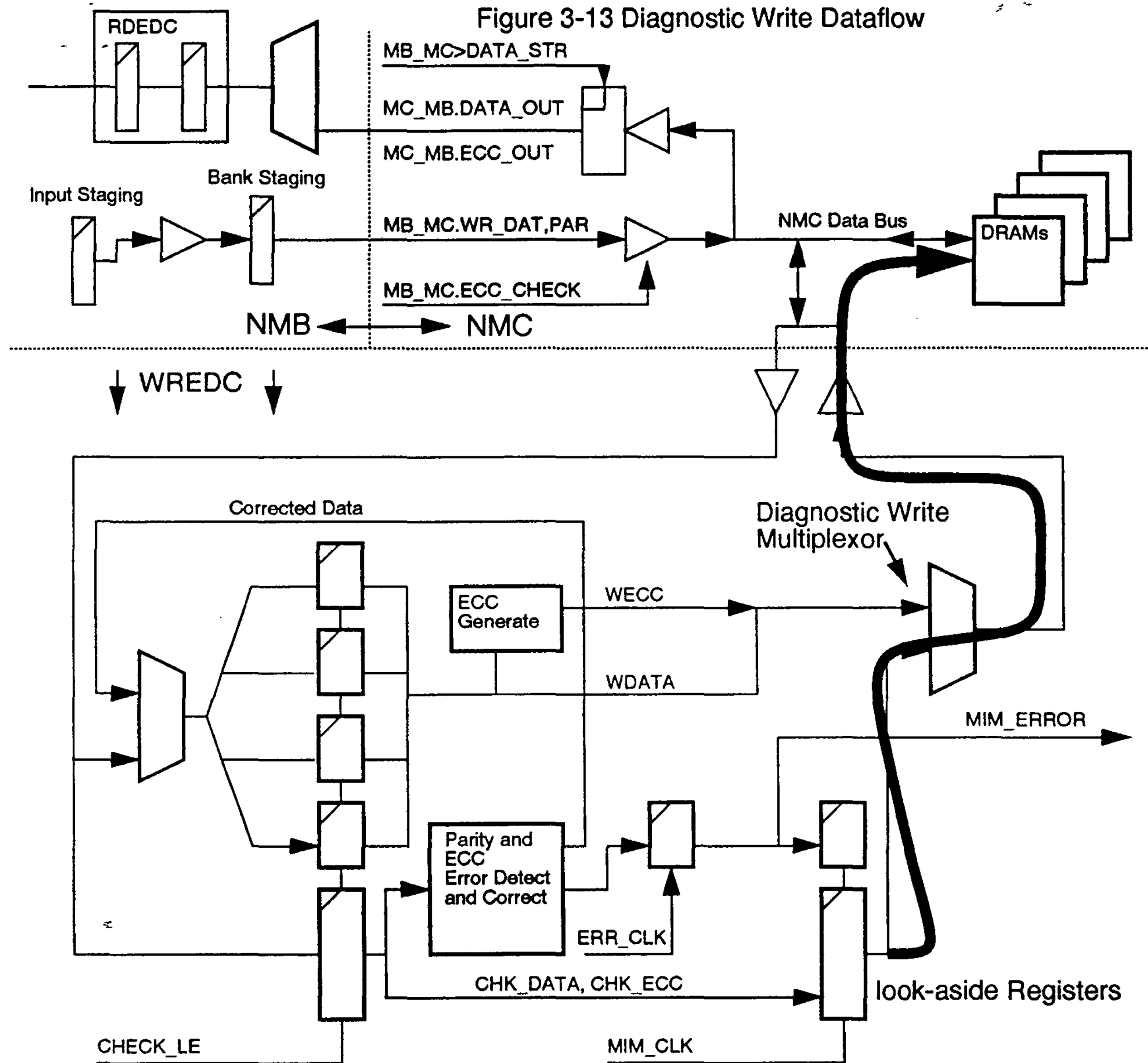
The scrub cycle is simply a write with all the ZONE bits zero. The NMB performs a read-modify-write since it does a RMW for any write in which the ZONE bits are not all ones, but since no ZONE bits are set, memory is not altered in any way. However, during the course of the RMW, the single bit error should be detected again and since RMWs correct single bit errors, the error should be corrected.

It should be noted that the scrub operation will cause a soft error to be generated assuming the error is still present in the data word when the NMB does the scrub operation. The SPU must ignore this second soft error since it occurred while the soft error was being corrected.

3.3.5 Diagnostic Mode Writes

The WREDC supports a special mode for doing writes that allows the write data and ECC to come from the WREDC lookaside registers rather than the normal path for write data. This is the only purpose for the multiplexor on the right side of the WREDC in Figure 3-13.

Figure 3-13 Diagnostic Write Dataflow



This diagnostic mode is set for bank 0e by scanning a one into the nmc0e_diag_data bit. With a one in this bit, all writes at this bank will take their write data from the lookaside register and ignore the normal write data path. This applies to all DRAM write operations: full writes, partial writes and read-modify-writes.

This bit exists solely for diagnostics purposes. It obviously prevents a bank from operating normally. Typical uses are to force bad ECC into a DRAM location for testing the error correct logic. Since this field is in the log ring, it is possible to set it for a write operation, then clear it without affecting the contents of the DRAM as long as no writes are done during the LOG scan

operation. (During LOG scan itself, the bit is masked and can not affect writes but if LOG scan mode was turned off just as a write was being performed, it could affect the timing of write data.)

3.4 Error Logging and Isolation

NMB error logging is localized to the logic which has detected the error and only the state necessary to determine what the error was. There is no global halt of NMB state upon detection of an error. In all error cases, refresh operation of the NMB DRAMs will not be interrupted. All error logging state may be accessed from the SYS ring so that refreshes can also be maintained while error state is examined via scan. All errors except for soft errors can be disabled by clearing enable bits in the NMB scan ring.

There are two main classes of errors detected by the NMB as discussed in the Interface chapter. Soft Errors are single bit, correctable errors that occur when data corruption is detected in the DRAMs. Soft errors may also occur if there is a board or part flaw affecting a single data bit on the return data path. This would be a hard error in the sense that there is a board flaw but it is an error that is still correctable. Hard Errors are non-recoverable (non-correctable) errors. These errors are the parity errors detected on data from the crossbar or data transferred to the NMCs, illegal addresses or requests to busy banks, and multiple bit data corruptions in DRAM data.

The timing at the board interface between error detection and the assertion of the MB_XC.SOFT_ERROR and MB_XC_HARD_ERROR signals has already been discussed in the Interfaces Chapter. This section describes how errors are detected and what registers must be examined to determine what error occurred.

The mechanics of detecting soft and hard errors are covered in the functional sections describing the cycling of DRAMs since it is an integrable part of DRAM operation.

3.4.1 Soft Errors

Soft errors are detected in two places on the NMB: the WREDC on the NMC card and the RDEDC. The RDEDC is used to detect soft errors whenever data is returned to the crossbar: read cycles and test-and-modify cycles. The WREDC is used to detect soft errors whenever a read-modify-write operation is performed on a bank. Read-modify-writes are used during partial writes (a write cycle with the zone bits not all ones) and during test-and-modifies. In the case of a test-and-modify both the WREDC and RDEDC are used to detect soft errors. An error detected by one gate array and not the other generally indicates a flaw in the board rather than a DRAM soft error.

Table 3-5 lists the scan fields used to decode a soft error or errors. Only bank "fe" is listed, other banks are omitted for clarity. Similarly, only the even side of the RDEDC data is listed.

The presence of a soft error is indicated by the bit in the scan ring called he.soft_error. This bit is in the SYS ring and is thus inaccessible from LOG scan. The presence of a soft error and the identity of the bank which detected the soft error is found by examining bc[x].bcga_log.bctly_soft_err where "x" is the number of the BCGA and "bctly" is the bank controller within that gate array. BCGAs are numbered from "0" to "7," with zero controlling odd banks 0 to 3, three controlling odd banks c to f, four controlling even banks 0 to 3 and seven controlling odd banks c to f. Within each gate array, bank controller zero (bctl0) controls the lowest numbered bank and so on.

Also in the BCGA log ring is bc[x].bcga_log.bctly_rdedc_err which indicates whether the error was detected at the RDEDC, bc[x].bcga_log.bctly_row_addr and bc[x].bcga_log.bctly_col_addr which record the row and column address of the request during which the error was detected, and bc[x].bcga_log.bctly_cycle which records the cycle type of this request. A scan ring format called bcx_banky_log_addr formats the row, column and bank number into the address that was presented to the NMB from the crossbar.

The data and ECC for which a soft error was detected is saved in the RDEDC and WREDC logging registers. The RDEDC also records the syndrome for the incorrect data which is the bitwise difference between the expected and read ECC for the data.

Table 3-5 Soft Error Scan State

he.soft_error	Indicates a soft error has been detected. Not visible from the LOG scan ring.
BCGA soft error logging state:	
bc[7].bcga_log.bctl3_soft_err	Bank three of gate array 7 (bank "f" even side) has detected a soft error.
bc[7].bcga_log.bctl3_rdedc_err	The soft error was detected by the RDEDC.
bc[7].bcga_log.bctl3_col_addr	column address of request which caused error.
bc[7].bcga_log.bctl3_row_addr	row address of request.
bc[7].bcga_log.bctl3_cycle	cycle type of request.
bc7_bank3_log_addr	col_addr, row_addr and bank number encoded into 26 bit address.
RDEDC logging state:	
rdedc.log[0].error	indicates the 0 (even) side of the RDEDC has detected an error.
rdedc.log[0].serr	error was a single bit error.
rdedc.log[0].merr	error was a multiple bit error. serr is don't care if merr is set.
rdedc.log[0].error_cmp	syndrome for compare of read versus generated ECC.
rdedc.log[0].error_ecc	ECC of data which caused an error.
rdedc.log[0].error_data	data which caused error.
WREDC logging state:	
nmcfe_ecc	ECC of data which caused error.
nmcfe_data	data which caused error.
nmcfe_rerr	indicates an error has been detected (hard, soft or parity).
nmcfe_rla_ecc_check	if zero indicates a soft or hard error, a one indicates a parity error was detected.

To decode soft error state, each of the thirty two bank soft error registers must be examined. If any of them is a one, then a soft error has been detected. A quick check for a soft error can be made

by examining the he.soft_error bit. If this bit is set one or more soft errors have occurred. However, this bit is not accessible from the LOG ring, the ring usually used to examine soft errors.

For each of the banks that have noted a soft error, further information can be found by examining the address and cycle registers to determine the address and operation type of the request which caused the error. The rdedc_err bit will also indicate whether the error was detected by the RDEDC.

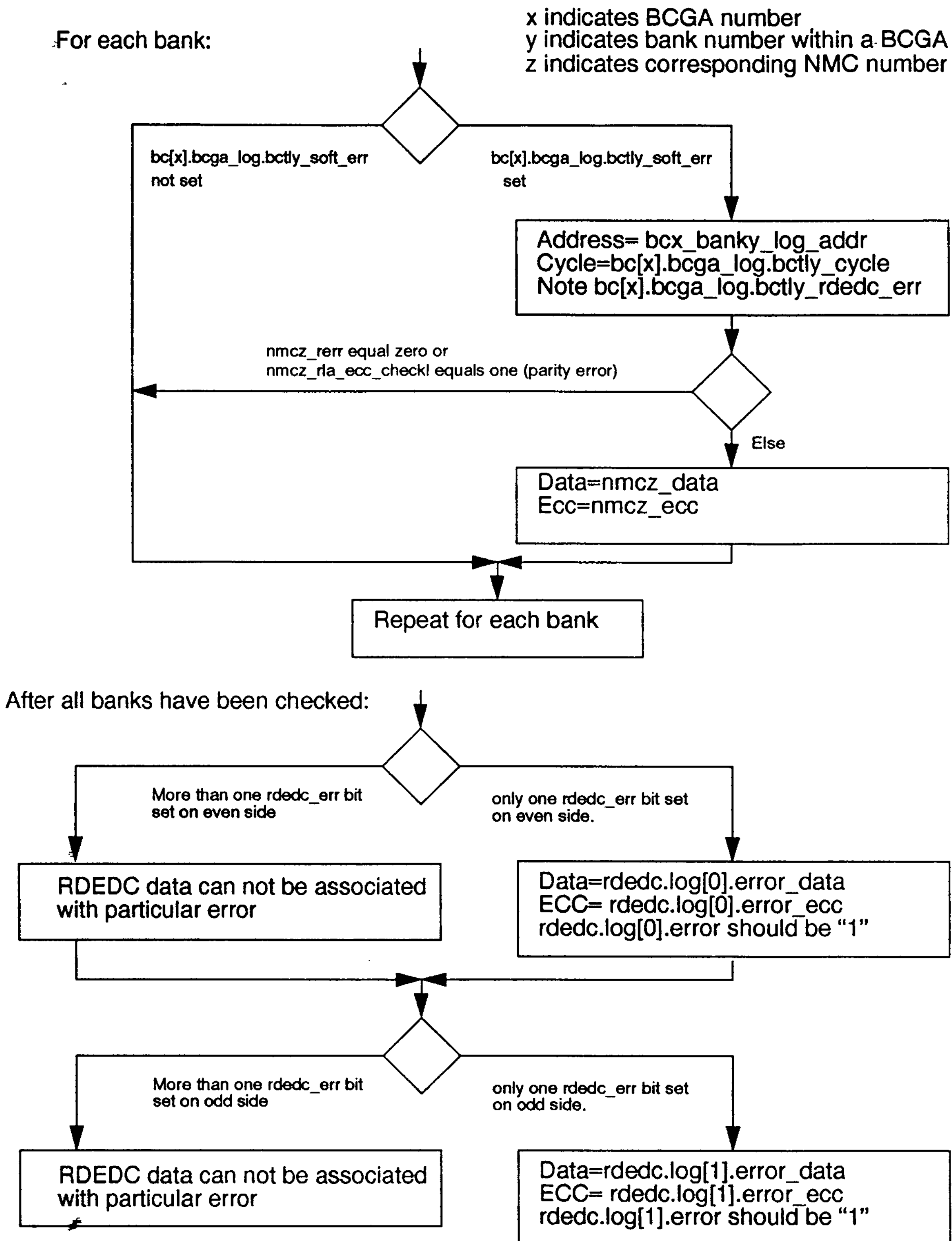
If the error was detected by the RDEDC as indicated by the RDEDC_ERR bit in the BCGA, the data and ECC which caused the error can be found in the RDEDC logging registers. These registers also note whether the error was single or multiple bit error (multiple bit errors are hard errors and are discussed in the next section). Note that the RDEDC can only log two sets of data and ECC, one for the even side and one for the odd side. If more than one error is detected among the even banks or more than one error is detected among the odd banks, the second and later errors can not be logged by the RDEDC since only one error logging register exists for each side in the RDEDC. Furthermore, it can not be determined which of the errors the data corresponds to.

If the WREDC detected the error, the RERR bit in the WREDC will be set and the rla_ecc_checkl bit will be a zero (a one indicates that a parity error on write data from the NMB was detected). The WREDC data and ECC errors will capture the data and ECC which caused the error.

In the case of a test-and-modify operation, both the WREDC and the RDEDC should log the data in error.

Figure 3-14 shows the steps necessary to decode a soft error.

Figure 3-14 Soft Error Decode



3.4.2 Hard Errors

Hard errors are non-recoverable errors which should cause the machine to halt. Halting of the machine is controlled by the NCU and XCL boards which stop clocks when a board reports a hard error and hard errors on that board are not being masked by the XCL.

While soft errors only occur if data read from the DRAMs is corrupted, there are several sources of hard errors in the NMB. Hard errors are produced when data read from DRAMs is so corrupted that it can not be corrected (two or more bits bad). Hard errors can also be caused by parity errors on address, data or control signals from the crossbar, illegal requests to a bank and internal data parity errors between the NMBs and NMCs.

Since there are more sources of hard errors than soft errors, decode of hard errors is more complicated. Table 3-6 lists the bits in the SYS scan ring which are used to decode the presence of an error. The table only lists bits for the even side of the RDEDC and bank "fe" in order to keep the table short. Further bits listed in following tables are used to determine the data or address for that particular error.

Table 3-6 Hard Error Scan State

he.hard_error	indicates that a hard error has been detected.
bc[7].sys.rhard_error and bc[7].sys.rhard_error_b	both should have same value. A one indicates that gate array number "7" has detected a hard error.
BCGA hard error state on request input staging:	
bc[7].sys.rcycle_err	error on cycle or address<7..3>
bc[7].sys.rlow_addr_err	error on address <14..8>
bc[7].sys.rmid_addr_err	error on address <21..15>
bc[7].sys.rhi_addr_err	error on address <28..22>
bc[7].sys._rillegal22	active low bit indicating an illegal address bit 25.
bc[7].sys._rillegal23	active low bit indicating an illegal address bit 26.
bc[7].sys._rillegal25	active low bit indicating an illegal address bit 28.
BCGA hard error state at individual banks:	
bc[7].sys.bctl3.rtn_merr	indicates that a multiple bit data error was detected by the RDEDC for a read by this bank.
bc[7].sys.bctl3_merr_pe	indicates that this bank's WREDC detected a hard error condition, either an ECC or a parity hard error.
bc[7].sys.bctl3_ctl_he	indicates a request was sent to this bank while it was still busy with busy with a previous request or refresh.
NMC hard error logging state:	
nmcfe_rerr	One if the NMC detected an error.
nmcfe_rla_ecc_checkl	Zero if the error was an ECC (DRAM data corruption), one if it was a write data parity error.

To decode the hard error state of the NMB, one begins with the he.hard_error bit. If that bit is not set, no error has been detected by the NMB. If it is set, the next step is to check each of the BCGAs and the input staging logic (ISE/ISO). All hard errors will be detected in the BCGA or the ISE/ISO logic. Some hard error state is logged in the WREDC and RDEDC, but it is not necessary to check these two arrays when looking for the source of the error. Once the error source has been found, these arrays are then useful for determining what was wrong with the data.

The ISE/ISO logic does not contain a register to hold the results of the parity check on its data. Instead, all the input staging data on the even side is held for even side errors and on the odd side for odd side errors. Once a hard error occurs, it is then necessary to examine the data and parity which has been held to determine whether that data is valid or incorrect.

Table 3-7 shows the ISE SYS scan fields which must be examined to determine whether there is a hard error in the ISE logic. These fields are the input staging registers which register the data immediately from the crossbar boards. They therefore register the signals described in the Interfaces chapter, Section 2.1.3 on page 15. The parity to be checked is the standard data and CTL_PAR also described in that section (Table 2-4 in the Interface chapter).

Table 3-7 ISE Hard Error Logging Registers

ise_sys.ise_sys_scan_out	CTL_PAR<0>, the parity bit for address<28..22>
ise_sys.addr	address<28..22>
ise_sys.rctl_par	CTL_PAR<4>, the parity bit for zone<3..0>
ise_sys.re_zone	zone<3..0>
ise_sys.rwre_par	write data parity, four bits.
ise_sys.rwre_data	write data, thirty two bits.

For the BCGAs, the bits in Table 3-6 for each array and bank controller within the array must be examined to determine what was the source of the error. The first errors in the table for the BCGA deal with the input staging logic. Parity errors are noted in the first four error logging bits. Each bit corresponds to an error with a particular CTL_PAR bit and the signals it covers as given in the table. The remaining three bits correspond to requests to illegal addresses. If address bits 25 or 26 are used when only one Megabit DRAMs are installed, these bits are not connected and an error will be detected. If address bit 28 is used when only two row NMCs are installed in the NMB, that bit is also invalid and an error will be noted. The BCGA internally numbers its bits from 0 to 25 rather than 3 to 28 which is why the illegal address bits are numbered 22, 23, and 25 rather than 25, 26, and 28.

Table 3-8 shows the fields to be examined to determine which bits are incorrect for input staging errors detected by the BCGA. The CTL_PAR to address and cycle mapping is the same as is described in Table 2-4 of the Interface chapter. Since all even BCGAs and all odd BCGAs register the same data from the crossbar, they should all have the same data and parity, even and odd side.

Table 3-8 BCGA Input Staging Registers

bc7_sys_addr	address<28..3>
bc[7].sys.is_cycle	input staging cycle<1..0>
bc[7].sys.ris_ctl_par	CTL_PAR<3..0>

For errors at a particular bank, those listed last in Table 3-6, the first two are data errors and the last one is a control error.

Corrupted data errors occur when one or more data bits are incorrect during a DRAM read. Data errors are checked by the WREDC during partial writes and test-and-modifies, and by the RDEDC during test-and-modifies and reads. No DRAM data corruption errors are checked for during full writes, no-ops and refreshes since the DRAMs are not read during these operations. All operations including no-ops and refreshes check write data parity at the WREDC.

For the DRAM data corruption or write data parity errors, the address and cycle type of the request that caused the problem can be found in the same log registers that soft error information is found in. These register are listed again in Table 3-9.

Table 3-9 BCGA Bank Error Logging State

bc[7].bcga_log.bctl3_cycle	cycle type of request.
bc7_bank3_log_addr	col_addr, row_addr and bank number encoded into 26 bit address.

The type of data corruption error is decoded from the rtn_merr and merr_pe bits in the BCGA and from the rerr and rla_ecc_checkl bits in the NMC. If rtn_merr is set, a multiple bit error was detected in read data at the RDEDC. If merr_pe is set, the rerr bit in the corresponding NMC should also be set. Merr_pe indicates that some sort of hard error has been detected by the WREDC on the NMC. To determine whether the error is a write parity error or a DRAM data corruption error, the rla_ecc_checkl bit must also be examined. A one indicates that the error was a write data error, otherwise the error was a DRAM data corruption error.

Data logging in the RDEDC and WREDC for hard errors is identical to the information saved for soft errors as listed in Table 3-5.

The last type of hard error detected by the NMB is the control hard error. This condition is indicated by a one in a ctl_he field in the BCGAs. It indicates that a bank has received a request when it was still busy with a previous request or refresh. The XARB gate array on the XS0 boards is supposed to keep track of when a bank is busy with a request or refresh and only send a request to that bank after the bank has indicated it is no longer busy using the MB_XS0.BANK_DONE signals. If this condition is violated, a hard error will occur. Depending on the severity of the incident, the error may cause some loss of DRAM memory content at the bank in question since the BCGA's operation is not guaranteed during this error. The error should only occur if spurious BANK_DONEs are being generated or refresh's are occurring too close to SYS clocks being restarted. It may also occur of the NMB's run bits are active while a crossbar board is being scanned. No state is saved on this error aside from the ctl_he bit itself.

Figure 3-15, Figure 3-16, and Figure 3-17 show the steps necessary to decode the NMB hard error state.

Figure 3-15 Hard Error Decoding

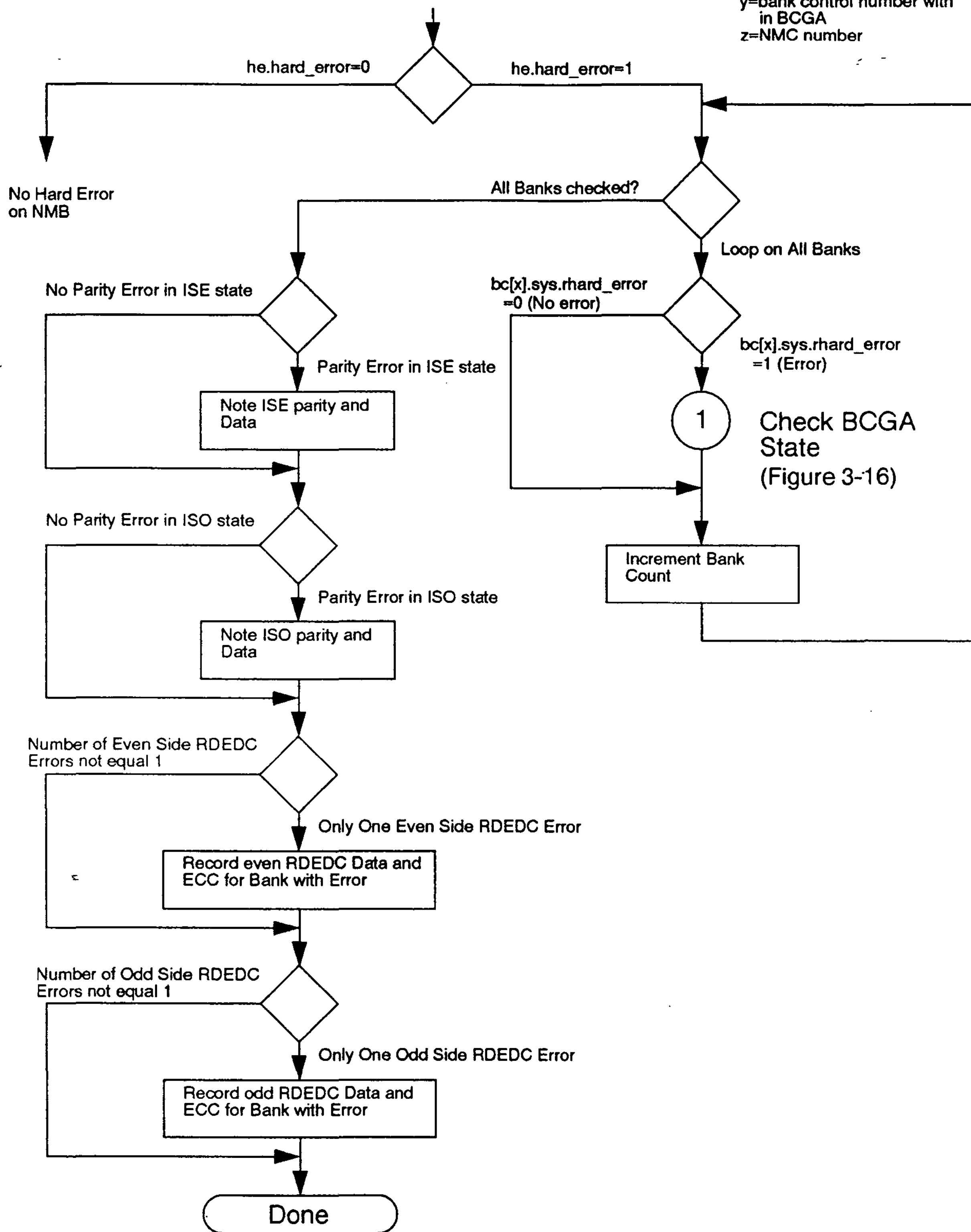


Figure 3-16 BCGA Hard Error Decode

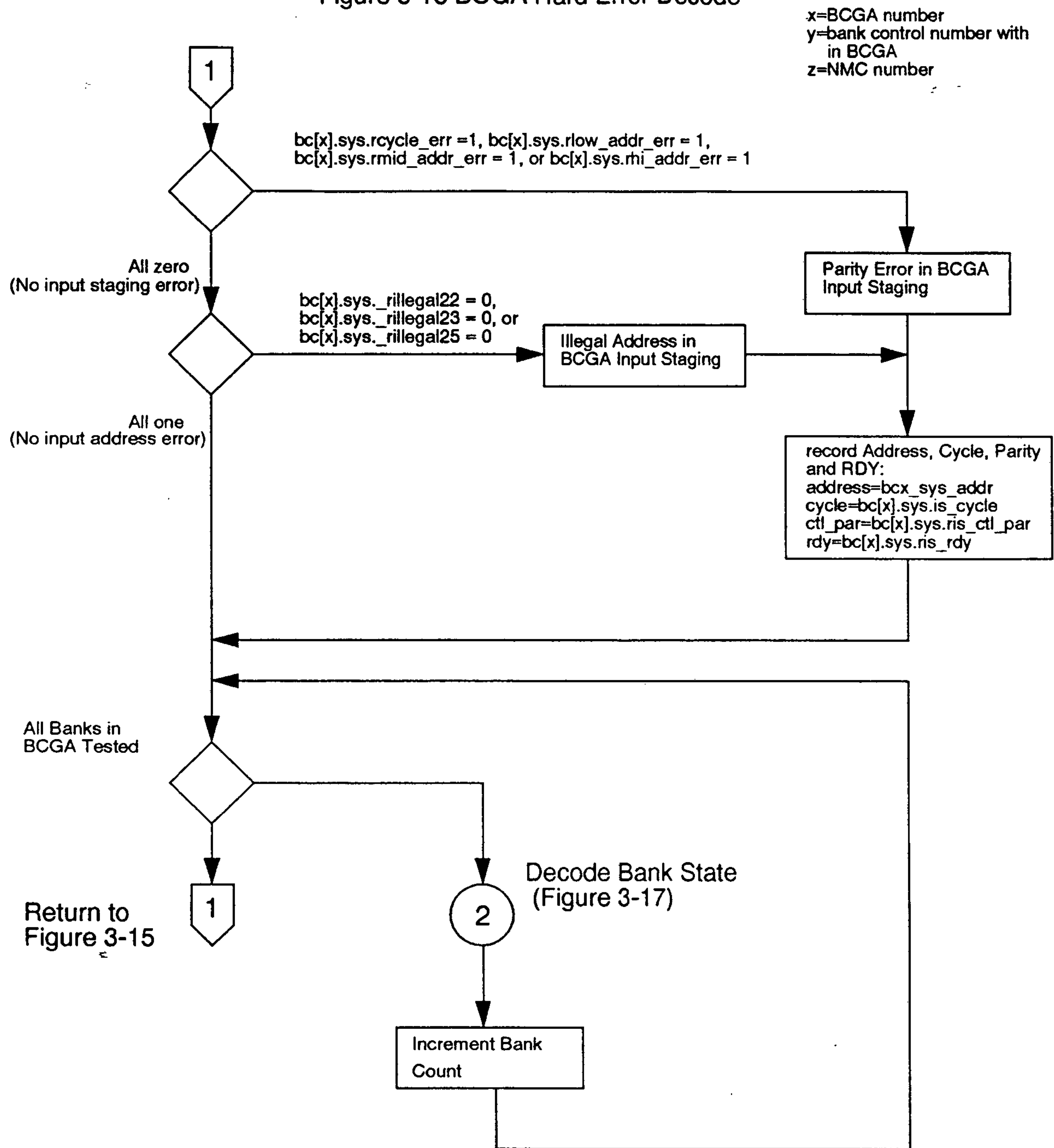
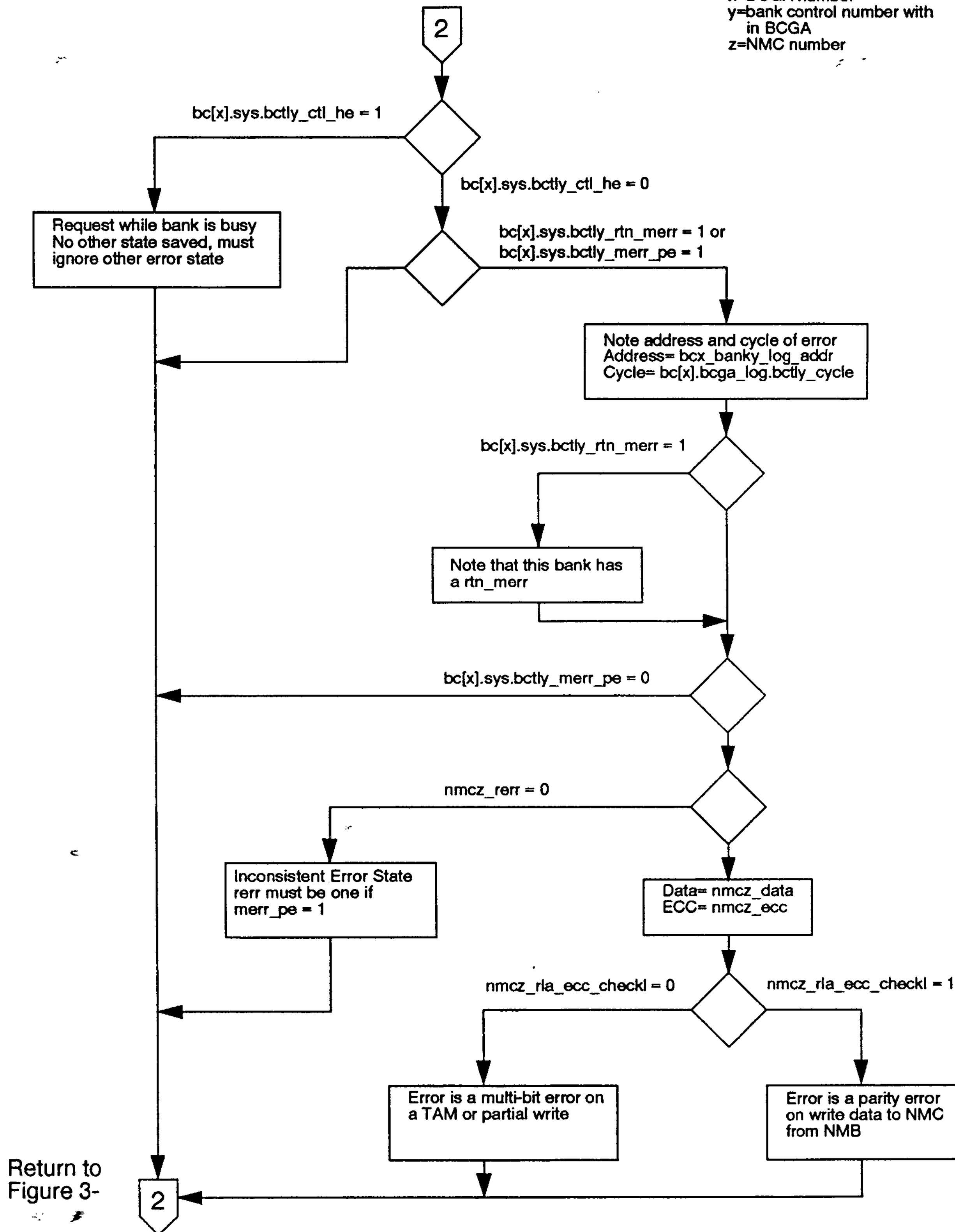


Figure 3-17 Bank Hard Error State



3.4.3 Crossbar Errors

As discussed in the Interface chapter (Section 2.1.1.2 on page 11), parity errors on data from the crossbar cause the SEND_PAR signals to be immediately asserted so that the crossbar can retain its copy of the data that was sent to the NMB. Since parity is not checked in the crossbar, it is necessary to examine the data as it was in the crossbar as well as as it was in the NMB to determine whether the error occurred from processor to crossbar or crossbar to NMB.

Table 3-10 Crossbar Error Registers

Crossbar Field	Purpose or Corresponding NMB field
xs0e:send_par_err	bit M should be set if NMB M sent xs0e a send_par_err signal
xs1e:send_par_err	As above
xrte:r5sel_m[M]	Contains the processor id number of the processor which sent the request which caused the error.
xs1e:wr_data_lsd2[M]	compare to mbM:ise_sys.rwre_data
xs1e:wr_par_lsd2[M]	compare to mbM:ise_sys.rwre_par
xs0e:addr_lsd2[M]	compare to: mbM:bc4_sys_addr, mbM:bc5_sys_addr mbM:bc6_sys_addr, mbM:bc7_sys_addr mbM:ise_sys.addr (bits 28..22)
xs0e:cycle_lsd2[M]	compare to: mbM:bc[4].sys.is_cycle, mbM:bc[5].sys.is_cycle mbM:bc[6].sys.is_cycle, mbM:bc[7].sys.is_cycle
xs1e:wr_zone_lsd2[M]	compare to mbM:ise_sys.re_zone
xs1e:ctl_par_lsd2[M]	compare to: mbM:bc[4].sys.ris_ctl_par (bits 3..0), mbM:bc[5].sys.ris_ctl_par (bits 3..0) mbM:bc[6].sys.ris_ctl_par (bits 3..0), mbM:bc[7].sys.ris_ctl_par (bits 3..0) mbM:ise_sys.ise_sys_scan_out (0) mbM:ise_sys.rctl_par (bit 4)

"M" is the number of the NMB which has detected the error.

Therefore if it is determined that the NMB hard error was caused by a parity error in the ISE, ISO or input staging of the BCGA, the appropriate registers on the crossbar must be examined in order to isolate the error. Table 3-10 shows the fields on the XRTE, XS0E and XS1E boards to be examined and the corresponding NMB field to be compared to. If the crossbar and NMB fields differ then the error occurred between the crossbar and the NMB. If the two fields are identical,

then the data was corrupted between the processor and the crossbar. Similar fields are used for errors detected by the odd side logic.

3.4.4 Multiple Errors

The NMB error logging logic is designed to capture all relevant state for a single error. When multiple errors occur, error information is sometimes reduced. It is always possible to determine which errors have occurred but it is not often possible to determine which data and address is to be associated with which error. (For a control hard error, ctl_he equal one, address and data is never retained.)

The primary case of lost data occurs when the RDEDC detects more than one error on the even or odd side. The RDEDC has only a single look-aside register for saving data and ECC that caused an error. If subsequently to detecting an error, a second error occurs, there is no place to also save the data that caused the second error. Furthermore, there is currently no way to associate data in the error logging register with a particular error at a bank. If two banks record an error detected by the RDEDC, it is not possible to determine which bank's data is held by the RDEDC. For this reason, the flow charts in Figure 3-14 and Figure 3-15 both check to see if there was only one error on the even and odd sides and only then associate the logging state with a particular error.

A control error as indicated by the bc[x].sys.bctly_ctl_he field is caused by a request being sent to a bank that is already busy with a previous request. In this situation the hardware acts unpredictably. It is likely that spurious errors will be generated for this bank. Since these errors are a by-product of the original error, they should be ignored. Only the control error is relevant. No information can be saved for the second request which caused the error. This error will only occur in three cases:

1) The NMB's free running clock is on and the RUN_SYS bit is active when the crossbar boards are scanned. The crossbar boards will generate many spurious requests during scan. Depending on the state of the boards, two false requests might be sent to the same bank.

2) If the -XC_MB.SYS_RUN signal is asserted less than approximately 20 clocks after a refresh, the refresh will still be occurring when the crossbar resumes sending requests to the NMB. Since the crossbar's clocks were not on at the time the refresh was sent, it will not know that the NMB is busy with a refresh and might send requests to busy banks causing a control hard error.

3) If spurious BANK_DONEs are being generated because of noise or a short to a BANK_DONE signal, the crossbar may believe a bank is ready for a new request before the bank is actually finished with the request.

It is possible for a bank to detect more than one error, either hard or soft or both. If the hard error is due to a multiple bit data corruption detected by the WREDC or RDEDC, the soft error bits will also be set. The hard error has precedence over the soft error; therefore, the soft error should be ignored. In all other cases, such as a parity error and a soft error, it is uncertain which error the address and data logging registers correspond to.

Table 3-11.

Table 3-11 Decoding Multiple Errors At a Bank

BCTL state in BCGA			NMC state		Meaning
soft_err	rtn_merr	merr_pe	rerr	rla_ecc_checkl	
1	0	0	X	X	Soft error, data and address as in Figure 3-14.
X	1	0	X	X	Multi-bit error detected by RDEDC, ignore any soft error if cycle was a TAM, WREDC should have detected error also.
X	1	1	1	0	Multi-bit error detected by both WREDC and RDEDC, ignore any soft error
X	0	1	1	0	Multi-bit error detect by WREDC, ignore any soft error. If cycle is a TAM, RDEDC should have detected error also.
X	1	0	X	X	Multi-bit error detected by RDEDC, ignore any soft error
X	0	1	1	1	Write data parity error detected by WREDC.
X	1	1	1	0	RDEDC detected a multi-bit error by WREDC detected a write data parity error. nmcz_data probably holds bad parity

It is only errors detected at a bank controller that can cause confusion in decoding error state. Input staging errors detected in ISE/ISO or the input staging registers of the BCGAs cause the error to freeze and therefore no error state is lost and no subsequent errors can be detected since the register is no longer loading data.

3.4.5 Soft Errors During LOG Scan

Whenever -XC_MB.LOG_RUN is inactive (one), all logging registers are frozen and therefore unavailable for logging soft errors. Similarly, during LOG scan, the logging registers also can not be used for logging errors. At these times, if a soft error occurs, it will still be corrected but no information will remain indicating that there was an error.

3.4.6 Error Enables

All errors on the NMB can be disabled, although there is no single error disable. Error disabling will prevent errors from interfering with the operation of the NMB but it will not prevent errors from being logged and reported to the XCL and crossbar boards. That is, XC_MB.HARD_ERROR, XC_MB.SOFT_ERROR and the SEND_PAR_ERR signals will still be asserted by the NMB. The XCL and XS0/1 boards must be set up to ignore NMB errors if error checking is to be turned off.

Error disables only prevent the NMB from causing non-logging state from interfering with the operation of the NMB. For instance, a parity error on input staging will cause the input staging register to freeze if an error is enabled and detected. If the error is not enabled, the input staging register will not freeze and the operation of the NMB will not be effected by the error. If the errors are enabled, the input register freezes and no further requests can be processed.

Table 3-12 shows the minimum amount of state that must be altered to disable all hard error enables.

Table 3-12 Minimum Hard Error Disable

he.hard_err_enable = 0
he.nmb_clock_scan_out = 1
rdedc.sys[0].he_enable = 0
rdedc.sys[1].he_enable = 0

Table 3-13 lists all the error enables and their functions. Note that some of the enables are only accessible from the ALL ring.

Table 3-13 NMB Error Disables

Enable	Function
he.hard_err_enable	enable all BCGA errors
he.nmb_clock_scan_out	enables ISE/ISO errors, active low
rdedc.sys[0].he_enable	enables even side RDEDC multi-bit errors
rdedc.sys[1].he_enable	enables odd side RDEDC multi-bit errors
rdedc.sys[0].se_enable	enables even side RDEDC soft errors
rdedc.sys[1].se_enable	enables odd side RDEDC soft errors
bc[X].all.four_meg_drams	a one indicates four megabit DRAMs are installed in all the NMCs that BCGA "X" controls. All BCGAs should be set with the same value. When this bit is a one, address bits <26..25> may be non-zero
bc[X].all.four_rows	similar to the previous field. Indicates the NMCs have four rather than two rows and that address bit <28> may be non-zero. All BCGAs should have the same value in this field.
bc[X].all._ctl_err_enable	Active low. If asserted, control hard errors detected by the banks of BCGA "X" will cause a hard error. Control hard errors are caused by a request to a busy bank.
bc[X].all._data_err_enable	Active low. If asserted, multi-bit data corruption and write data parity errors detected by the banks controlled by BCGA "X" will cause a hard error.
bc[X].all._par_chk_ena	Active low. If asserted parity on input staging data is checked and will cause a hard error.

3.4.7 Additional RDEDC Logging Information

The RDEDC logs more than just data and ECC when a soft or hard error is detected. It logs whether it was a single bit or multi-bit error and the bit-wise compare of the actual and expected

ECC, called the syndrome. These fields are:

rdedc.log[0].merr	indicates a multi-bit error on even side
rdedc.log[0].serr	indicates a single bit or multi-bit error on even side
rdedc.log[0].error_cmp	Syndrome bits

There are similar fields for the odd side of the RDEDC.

3.5 Clocks and Scan

The NMB takes the CU_MB.FREE_CLOCK2 signal from the NCU board and buffers and gates this clock into free running 1X and 2X clocks as well as its SYS and LOG clocks. The generation of these clocks is controlled by internal logic, RUN bits from the XCL and the scan control signals.

The clock gating and basic scan control fan out is performed in the NMB_CLOCK logic block in the NMB schematics. The generation of the MIM_CLK signals for the NMCs is performed in the MIM_CLK schematic block.

3.5.1 NMB_CLOCK

The NMB_CLOCK logic block contains most of the logic for generating the NMB's clocks. It also contains some error registering and miscellaneous signal registering which it is not necessary to discuss.

Clock distribution primarily is a matter of creating multiple copies of clocks with a minimum amount of clock skew. Clock skew is the variation between clock edges on the board. Ideally, clock skew is zero: all the clocks line up. In reality, there is some amount of variation between clocks both within a board and from board to board. This variation effects system performance because it, in effect, increases both the setup time in a path and the hold time.

3.5.1.1 Clock Skew

Figure 3-18 Clock Skew

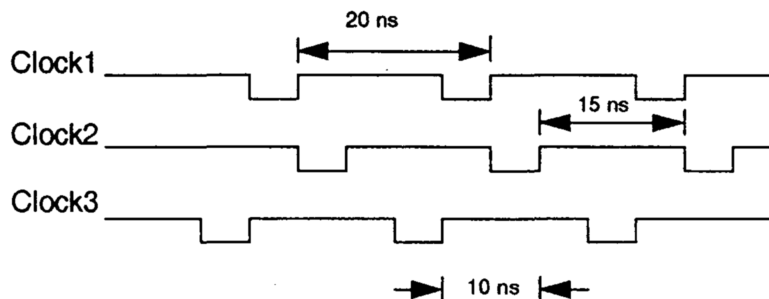


Figure 3-18 is an exaggerated example of clock skew. Ideally, the rising edge of all three clocks should line up. In this figure, however, there is a great deal of variation as to when the clocks really do rise. Clock2 rises 5 ns after clock1 and clock3 rises 5 ns before clock1. The worst case variation is 10ns between clock1 and clock2 which gives a worst case clock skew of 10 ns.

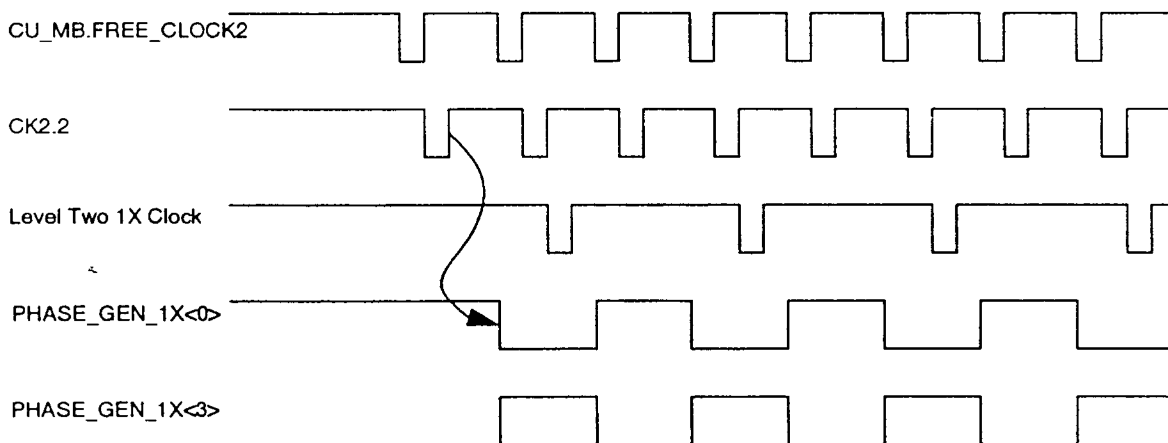
just which pin has the "bubble." The bubbled pin on the first level clock driver is not the bubbled pin on the second and third level drivers.

On the NMB, clock gating is used for two purposes. It is used to create a 1X clock from the 2X input clock and it is used to gate the SYS clocks. Internal to the gate arrays, the clocks are also gated to produce LOG clocks but this is discussed only in the appropriate gate array specification document.

The 1X clock is generated from the 2X clock via the phase generator on the first page of the NMB_CLOCK schematics. This circuit is depicted in a simplified form in Figure 3-20. The clock from the NCU enters at lower left and is buffered by the first 100E111. One of the 100E111 clocks goes to a 100E451 which is used as a phase generator. The 100E451 is simply a six bit register with differential clock inputs. Five of the slices in the 100E451 are shown in the figure. The first four are the phase generator bits numbered from zero to three, left to right. The fifth bit is present only for hold time deskew for scan operation. All five of these bits are in a single 100E451 package.

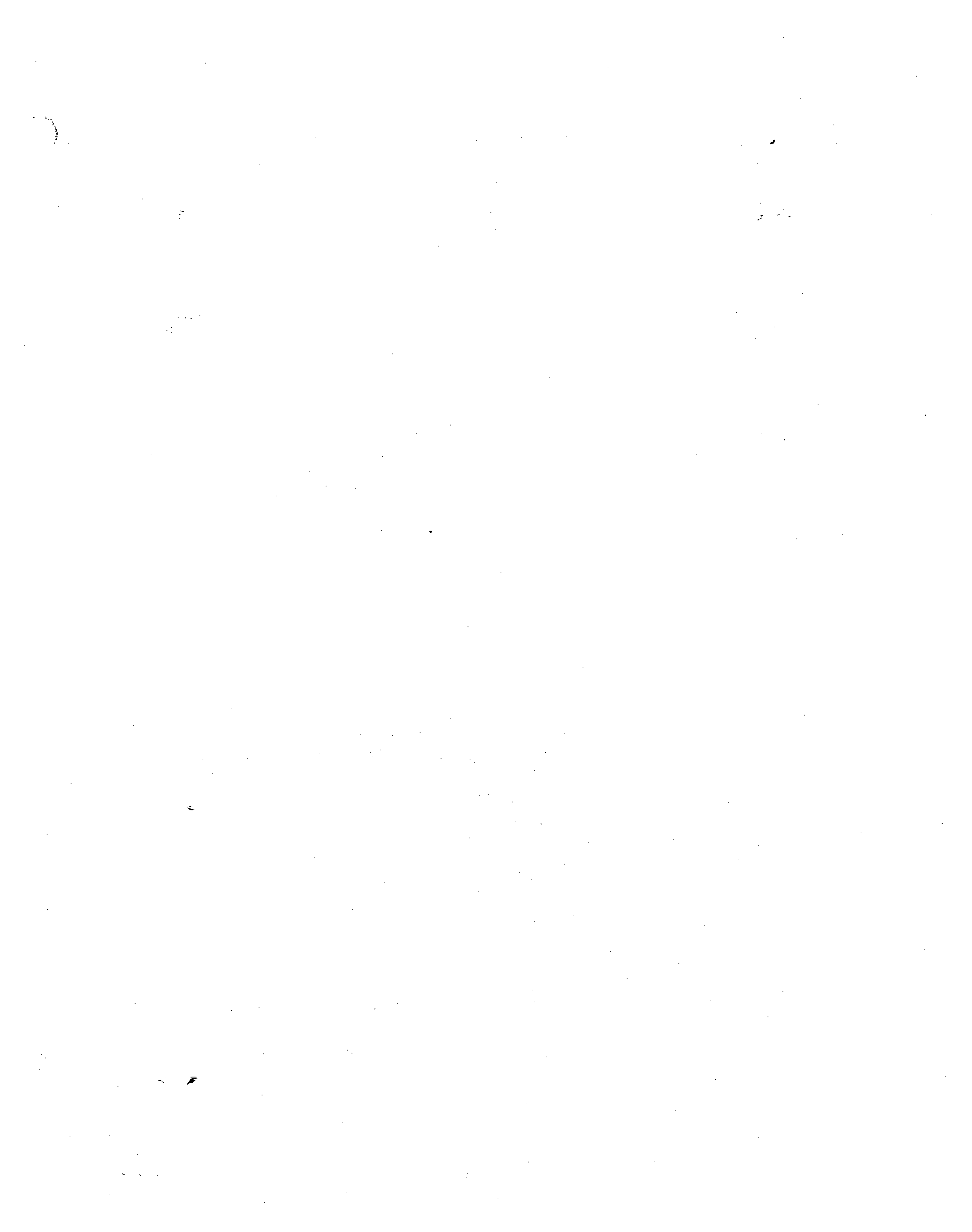
The five bits in the 100E451 can be set from the ALL scan ring. During ALL scan, the multiplexor on the left side selects the scan input to the circuit and the 100E451s can be scanned in a serial fashion. The fifth 100E451 is used as a deskew register. Since the 100E451's clock comes from the level one buffer and the next register in the scan ring is clock off of the level three buffers (as are nearly all registers), there is a large amount of clock skew between the E451 and the next register. A simple way to adjust for this clock skew is to add the fifth register. It does not appear at all in the scan ring definition because it acts only to align the phase generator's scan output with the level three clocks. It in the next bit in the ring are always scanned to the same value.

Figure 3-21 Phase Generator Timing



The phase generator is scan initialized to have a 1010 pattern. (The state of the deskew register is unimportant.) With such a pattern the first bit of the phase generator, bit zero, is a one on every other 2X clock since the 100E451 is clocked by a 2X clock. This bit is used to gate the second level 100E111. Each one masks a 2X clock pulse. Since every other 2X clock the phase generator bit zero is a one, every other 2X clock is masked at the second level buffer leaving as a 1X clock.

Figure 3-21 illustrates the phase generator process. Note that this diagram does show propagation delays since these delays are crucial to the operation of this circuit. CK2.2 is the input to the level two clock buffer shown in Figure 3-20. It's output is the clock labeled "Level Two 1X



Clock skew effects how much time signals have to propagate between registers. For signals going from a register clocked by Clock1 to another register clocked by Clock1, the signal has the full 20 ns to leave the first register, travel to the second register and make setup to that second register. If the first register was clocked by Clock2, however, it would only have 15 ns to travel between the two registers. Since Clock2 does not rise until 5 ns after Clock1, the signal has 5 less nanoseconds to propagate.

When a signal changes too soon at a register, hold times may be violated. If a signal goes from a register clocked by Clock3 to one clocked by Clock2, it may arrive at the destination register before that register has received its clock and then see the data for the current clock rather than the previous clock period.

3.5.1.2 Basic Clock Distribution

The NMB uses 100E111s to distribute its clocks. These devices have one differential input and nine differential outputs. They are guaranteed to have low skew (variation in output delay) between outputs of the same part and between outputs on different parts. At this time, outputs within a 100E111 are within 100ps of each other and between parts are within 150ps.

Because of the number of clocks the NMB requires, three levels of 100E111s are used to fanout the single input clock into all the necessary board level clocks. The first level consists of a single 100E111 from which all other clocks are generated. The other two levels consist of two or more 100E111s. The maximum skew within the board is therefore 100ps from the first level plus two times 150ps for the second and third level clocks for a total of 400ps of skew due to the 100E111s.

Figure 3-19 Basic NMB Clock Tree

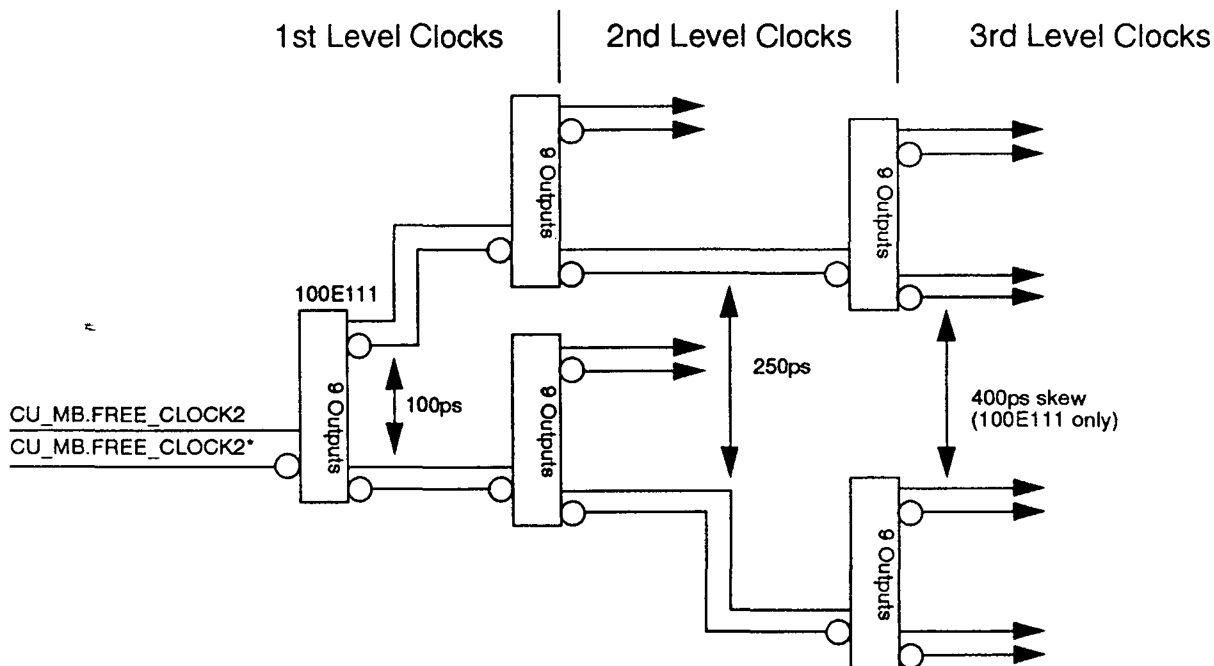


Figure 3-19 shows the basic three level tree with the skew due to each level of 100E111 shown. Note that there are other contributions to skew such as the routed distance of the clock.

All the clock wires on the NMB are carefully routed so that the amount number of wire in each of the clocks is equal for all clocks. It is not possible to have the wires routed precisely to the same length but the skew due to wire length mismatch is small, perhaps 200ps or less. The total skew on a board should then be less than 600ps.

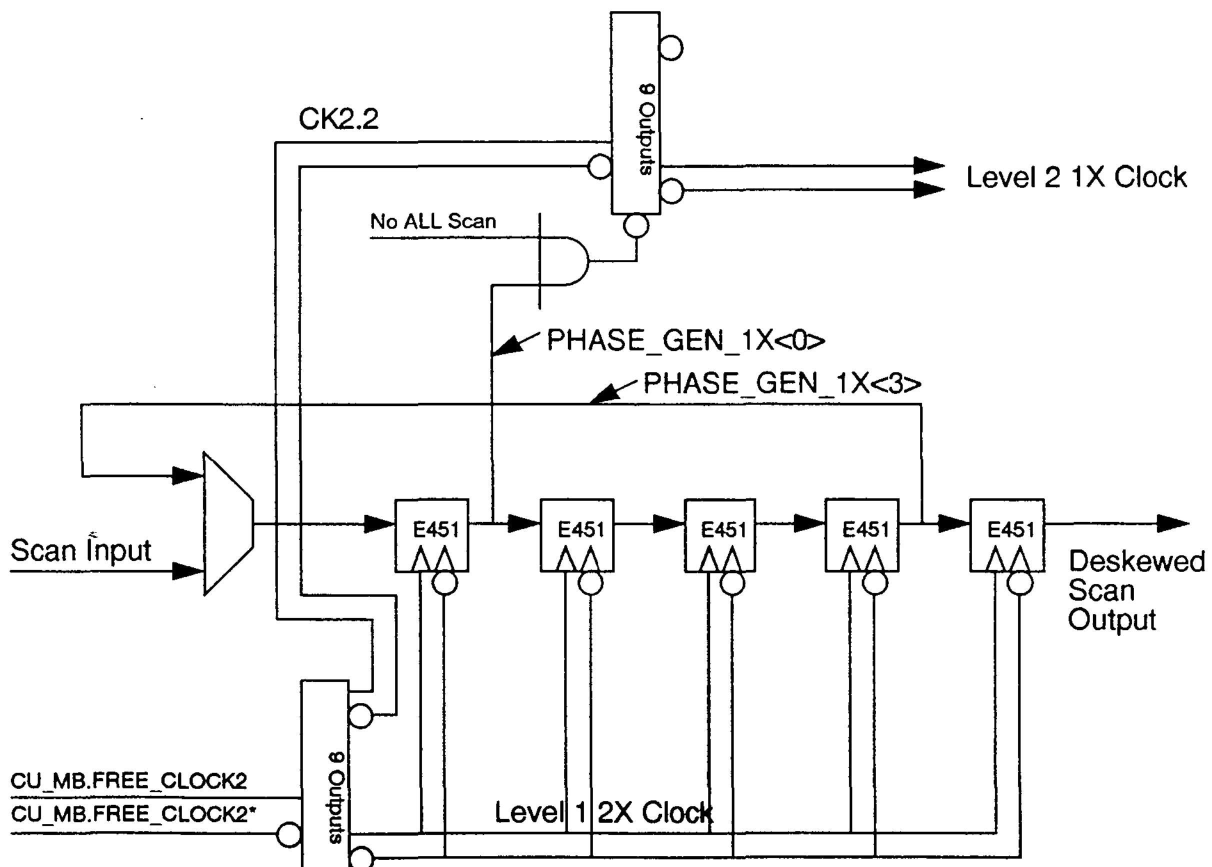
Most clocks only go to a single clock pin. Some of the clocks in the ISE/ISO bank staging logic go to two clock pins.

3.5.1.3 Clock Gating

The 100E111 has an enable input to the device which can be used for clock gating. This enable is an active low enable. A high forces all Q outputs low and all Q* outputs high.

Since all C3 clocks are to be halted in a high state, at level two and three the sense of the inputs the 100E111s is reversed. The active high clock is wired to the active low input of the 100E111 and the active low clock is wired to the active high input. Since the 100E111 provides differential outputs, this has no other affect than to make the enable pin on the 100E111 cause the active high clocks to stop high.

Figure 3-20 Phase Generator Circuit



In the NMB_CLOCK schematics, the 100E111s have been "bubbled" so that the bubble at the input pin is on the signal with the active low board level sense. This is simply a documentation feature of the schematic. Be sure to examine the pin number of the input and output pins and not

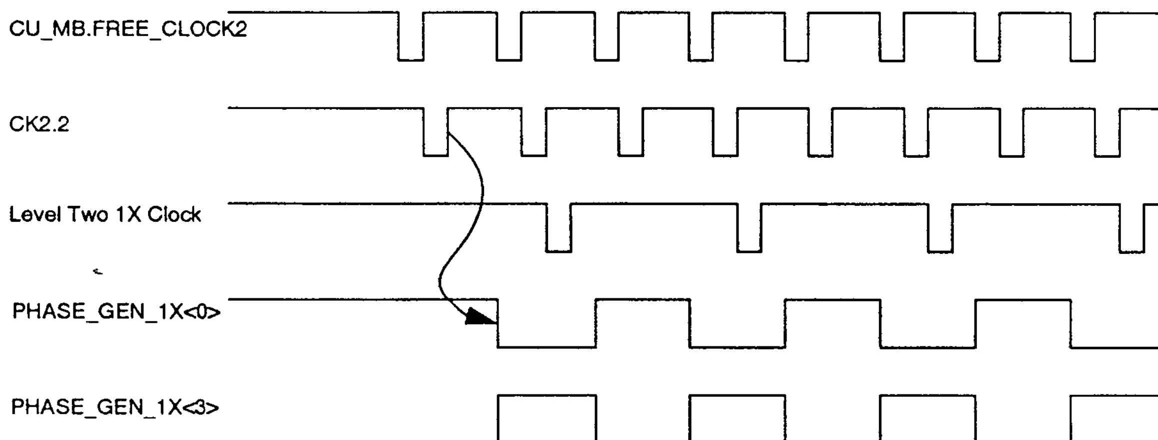
just which pin has the "bubble." The bubbled pin on the first level clock driver is not the bubbled pin on the second and third level drivers.

On the NMB, clock gating is used for two purposes. It is used to create a 1X clock from the 2X input clock and it is used to gate the SYS clocks. Internal to the gate arrays, the clocks are also gated to produce LOG clocks but this is discussed only in the appropriate gate array specification document.

The 1X clock is generated from the 2X clock via the phase generator on the first page of the NMB_CLOCK schematics. This circuit is depicted in a simplified form in Figure 3-20. The clock from the NCU enters at lower left and is buffered by the first 100E111. One of the 100E111 clocks goes to a 100E451 which is used as a phase generator. The 100E451 is simply a six bit register with differential clock inputs. Five of the slices in the 100E451 are shown in the figure. The first four are the phase generator bits numbered from zero to three, left to right. The fifth bit is present only for hold time deskew for scan operation. All five of these bits are in a single 100E451 package.

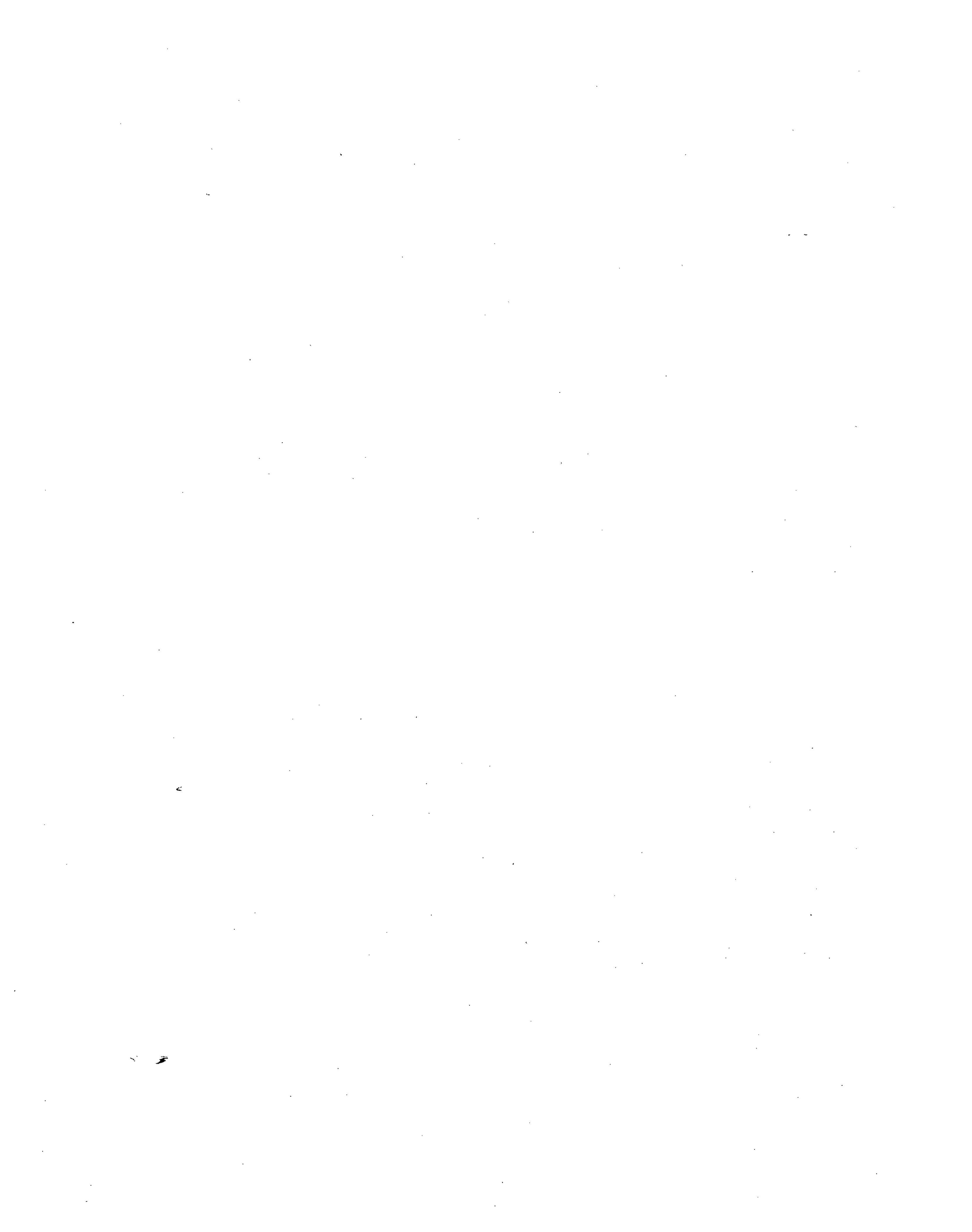
The five bits in the 100E451 can be set from the ALL scan ring. During ALL scan, the multiplexor on the left side selects the scan input to the circuit and the 100E451s can be scanned in a serial fashion. The fifth 100E451 is used as a deskew register. Since the 100E451's clock comes from the level one buffer and the next register in the scan ring is clock off of the level three buffers (as are nearly all registers), there is a large amount of clock skew between the E451 and the next register. A simple way to adjust for this clock skew is to add the fifth register. It does not appear at all in the scan ring definition because it acts only to align the phase generator's scan output with the level three clocks. It in the next bit in the ring are always scanned to the same value.

Figure 3-21 Phase Generator Timing



The phase generator is scan initialized to have a 1010 pattern. (The state of the deskew register is unimportant.) With such a pattern the first bit of the phase generator, bit zero, is a one on every other 2X clock since the 100E451 is clocked by a 2X clock. This bit is used to gate the second level 100E111. Each one masks a 2X clock pulse. Since every other 2X clock the phase generator bit zero is a one, every other 2X clock is masked at the second level buffer leaving as a 1X clock.

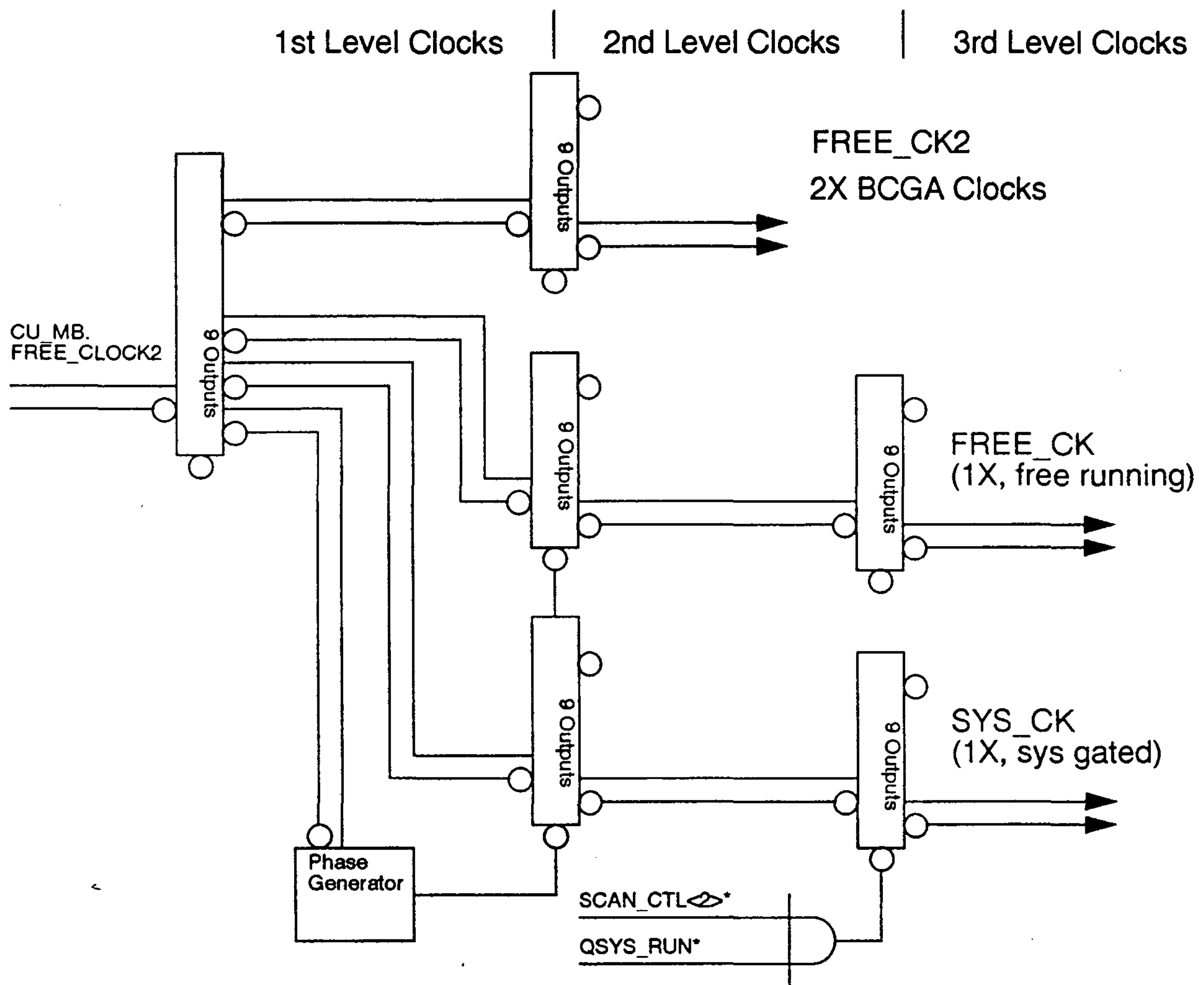
Figure 3-21 illustrates the phase generator process. Note that this diagram does show propagation delays since these delays are crucial to the operation of this circuit. CK2.2 is the input to the level two clock buffer shown in Figure 3-20. It's output is the clock labeled "Level Two 1X



Clock." Every other clock pulse has been masked in this output leaving it at half the frequency of the input clock. This frequency is the 1X system frequency. The PHASE_GEN_1X<0> signal must be properly timed so that it completely masks the 2X clock.

The phase generator is not used to gate clocks during ALL scan. At this time, the NMB receives 1X clocks. It then turns off all clock gating so that all clocks are the ungated 1X clock. Since the phase generators are being scanned during ALL scan, it is impossible to use them to gate clocks.

Figure 3-22 NMB Clock Distribution



The only other type of clock gating performed by the NMB_CLOCK logic is the SYS clock gating. The SYS clocks are used by those registers which are halted in step with the crossbar and the rest of the system. These clocks are called SYS_CK<x> in the schematics where "x" is the bit or id number of the clock. SYS clocks are generated by gating the level two 1X clock with a registered copy of the XC_MB.SYS_RUN* signal at the third level clock buffer to produce a level three, 1X, SYS clock. SYS_RUN* clock gating is disabled whenever SCAN_CTL<2> is set. This bit is set during ALL scan, the NMC clock generator set and clear modes, and during CAST_LOAD mode.

The 1X clocks which are not gated by SYS_RUN* are called free running clocks and are named FREE_CK<x>.

Figure 3-22 shows the full NMB clock distribution network. All first and second level clock buffers are shown; most of the level three clock buffers are omitted. If a 100E111 is shown without a signal going to its enable, that 100E111 is never disabled.

From the drawing, it can be seen that the BCGAs get a 2X clock coming out of only a second level clock buffer rather than the third level clock that all other parts receive.

3.5.1.4 Clock Tuning

In the C2 system, clocks were tuned by using variable delay lines to make all the system clocks line up. The C3 system dispenses with the delay lines. Instead, it uses the high precision (low skew) 100E111s and carefully routed clock lines.

All clocks have been carefully routed so that the total of the wire delay and the total of the buffer delays are matched. For the gate arrays (BCGA and RDEDC), the clocks going to these devices have been adjusted to take into account the clock distribution tree internal to the gate arrays. These clocks therefore have less delay at the device pin than other clocks. The difference is made up inside the gate array.

When reworking a clock line it is important to maintain the propagation delays already on these lines. The details of how the clocks were tuned is given in a later chapter which covers the "recipe" for making the NMB.

3.5.2 Scan

The operation of the various NMB scan modes at the edge of the board has already been described in the Interface Chapter, Section 2.1.5 on page 17. This section only describes some of the details of the input staging for scan operations.

Unless the NMB is scanned for a multiple of eight clocks, the NMCs will not scan out in the same order they scanned in. The NMB uses two taps off of an eight bit shift register to ensure that two of the NMB's three rings contain a multiple of eight bits. Since one ring uses one tap from the shift register and another ring uses a different tap, some bits of this staging register are not in all rings.

The NMB ALL and SYS rings are the two rings with a multiple of eight bits. The LOG ring does not have a multiple of eight bits. The LOG ring must have separate read and write format statements for the NMC fields because, in effect, the NMC fields are in different places for reads and writes. The field name for the LOG read format is identical to the field name for the ALL and SYS ring because reads are the most common operation for the long ring. The write format is the same as the read format with the addition of a "_wr" added to the name. Refer to the NMB LOG ring definition for details.

The scan engine logic clocks a board for an additional six clocks (in some boards' cases, seven clocks) to allow for staging through the crossbar on the scan ring path. These extra six clocks are considered when determining whether the NMB receives a multiple of eight clocks during scan. For instance, the ALL ring is currently 9474 bits long. 9474+6=9480. 9480 is evenly divisible by 8, 9474 is not.

3.5.3 MIM_CTL, the NMC Clock Generator

The NMC's scan clock is the MIM_CLK signal. During normal operation this signal is used to clock the lookaside registers on the WREDC. Clock edges are initiated by the BCGA and occur irregularly as the NMC is used for DRAM operations.

During scan operation in any of the scan modes, the MIM_CLK signal must clock at a regular rate so that the NMC can be a part of the scan ring. Because the WREDC is a CMOS part and incapable of clocking at the system clock rate, the MIM_CLK rate during scan must be some multiple of the system scan clock. The scan MIM_CLK rate was chosen to be one eighth the normal scan clock frequency. Since the NMCs are receiving only one clock in eight, they must be time division multiplexed so that as a group they provide one *scan out* bit per clock and consume one *scan in* bit per clock. This multiplexing is briefly described in Section 2.2.8 on page 39 of the Interface chapter.

The logic to generate this special scan clock and to determine when to use the MIM_CLK generated by the BCGAs is located in the MIM_CTL logic body which this section describes. The MIM_CTL logic block contains a state machine for producing the one eighth rate clock and properly transitioning into and out of scan clock mode. It contains staging for scan data into the NMCs and a multiplexor for selecting scan data from the NMCs. It also contains the logic for determining whether to use the BCGA generated MIM_CLK or the scan clock and logic to ensure that any MIM_CLK to the NMCs satisfies minimum clock pulse width requirements.

3.5.3.1 MIM_CLK Generation

In normal operation the MIM_CLK is generated from the Bxx_CHECK signal from the BCGAs where "xx" is the bank id number. The CHECK signal is simply registered and then sent to the NMC as Bxx_MIM_CLK. The CHECK signal itself also goes to the NMCs (unregistered).

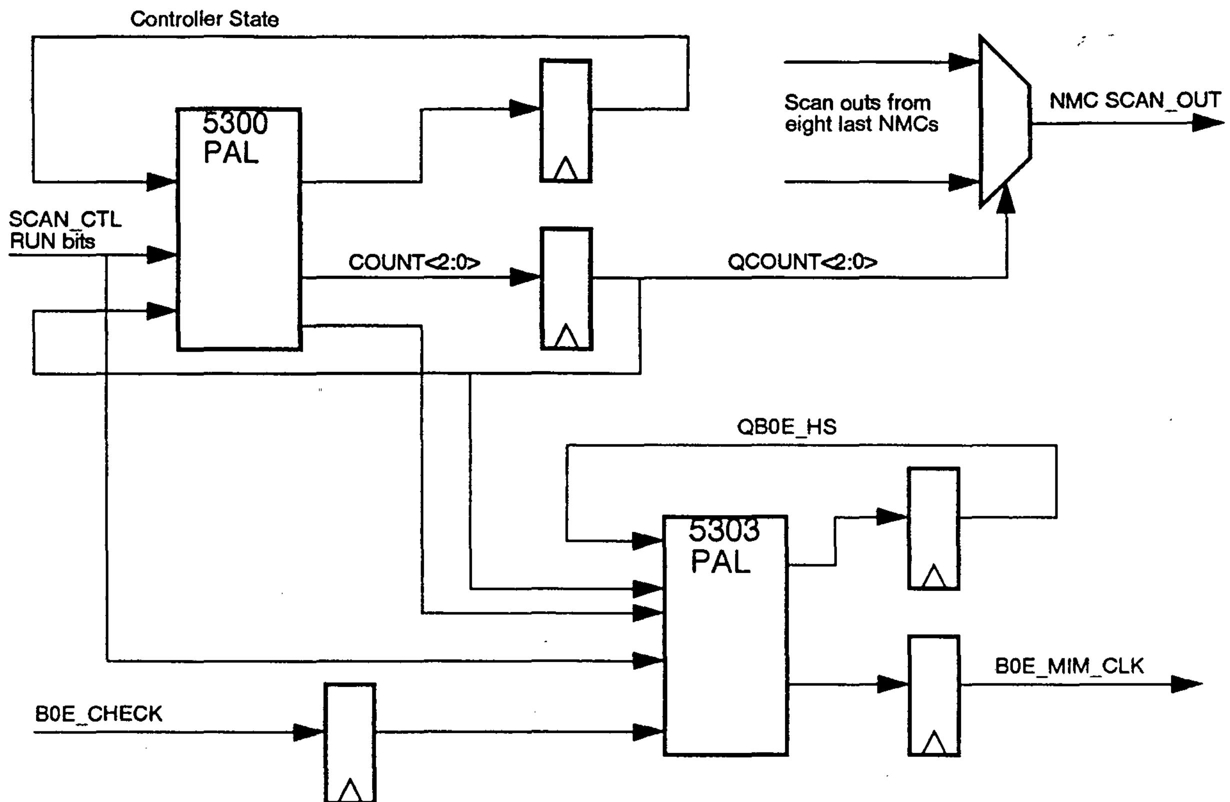
During scan operation, the CHECK signal is ignored and the MIM_CLKs come from the MIM_CTL clock generator state machine. Between normal mode and scan operation, the MIM_CLKs must be gracefully switched from one mode to the other. Clocks can not be truncated to smaller than their minimum pulse width requirements and clocks originated by the CHECK signal can not be run together with scan clocks.

To properly handle the normal, scan and transition clocks, each MIM_CLK has a dedicated, simple state machine consisting of one bit of state, a register for the MIM_CLK signal and part of a PAL for combinational logic. (This logic is found on page two of the MIM_CTL schematics.) This state machine decodes the state of the scan clock controller which is found on page one of the MIM_CTL schematics.

Figure 3-23 shows the logic in MIM_CTL used to generate the B0E_MIM_CLK. There is only one 5300 PAL for all of the MIM_CTL. There are eight 5303 PALs, each controlling four MIM_CLKs. The 5300 generates the QCOUNT which is used both to select which NMC scan out becomes the group scan out and by the 5303 PALs to determine when to generate a MIM_CLK during scan. It

is the 5303 PAL which determines when to ignore the B0E_CHECK signal and assures minimum pulse widths are not violated.

Figure 3-23 MIM_CLK Generator



The 5303 PAL receives several signals to let it know whether it should be generating scan clocks, masking the CHECK signal or generating MIM_CLK from the CHECK signal. These signals are listed in Table 3-14.

Table 3-14 PAL 5303, MIM_CLK Generation

SCAN_CTL<2>	QLOG_RUN*	RSCAN_CTL<0>	ENABLE	Action
1	X	X	0	No MIM_CLKs
1	X	X	1	MIM_CLKs from QCOUNT
0	0	0	X	MIM_CLKs from CHECK
0	1	1	X	No MIM_CLKs
0	0	1	0	No MIM_CLKs
0	0	1	1	MIM_CLKs from QCOUNT

SCAN_CTL<2> is bit two if the scan control signals sent by the XCL. When this bit is set, the board is either in ALL scan mode, preparing or leaving all scan mode or in CAST load mode. At these times, the CHECK bit is ignored. The only clocks which should be generated are the scan clocks.

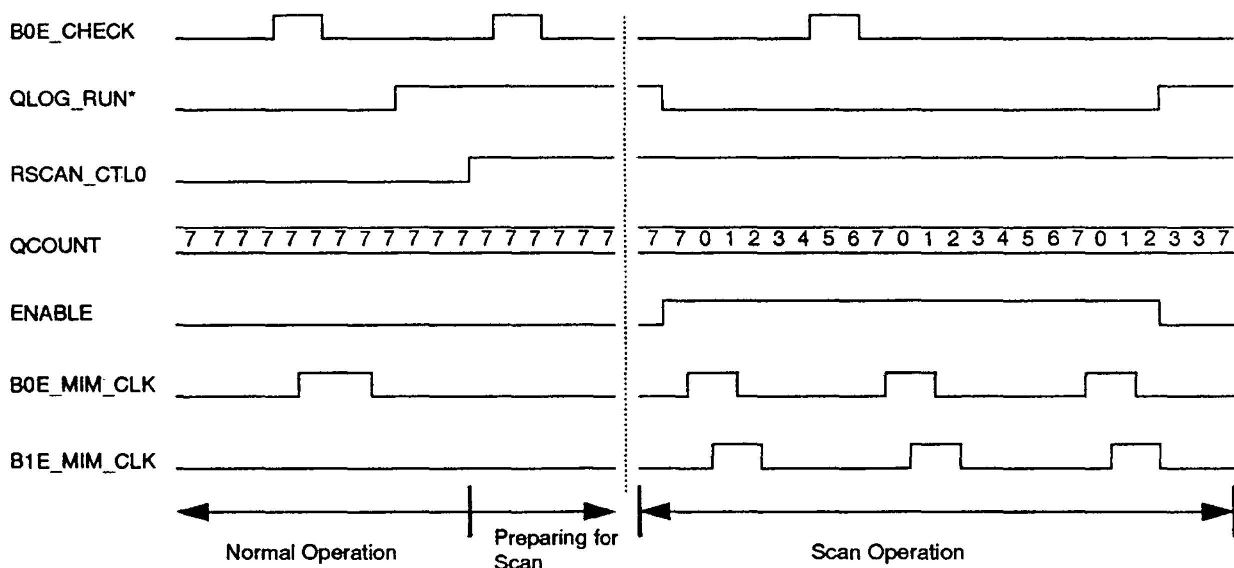
The ENABLE signal comes from the 5300 PAL. It tells the PAL that scan has begun and indicates that the 5303 PAL should begin decoding the QCOUNT signal to generate clocks. The 5300 PAL generates two enables, one called ENABLE03 and one called ENABLE47 for the 5303 PALs. Each goes to half of the 5303 PALs. The two signals are identical in operation.

The QCOUNT signal is a three bit count of zero to seven generated by the 5300 PAL and registered. It is used by the 5303 PALs to determine when each of the clocks they control should be asserted. Every clock is asserted once in eight counts for a frequency of one eighth the system clock rate. One of the 5303 PAL's inputs is a static signal (tied low or high) which tells it whether it is controlling the first four or last four clocks in a group of eight clocks. Using this signal and the QCOUNT, it can then assert each of the four clocks it controls at the proper time.

QLOG_RUN* is the registered XC-MB.LOG_RUN* signal. During SYS and LOG scan it is used to transition to and from scan mode. During ALL scan, it is ignored. When it is de-asserted and SCAN_CTL<2> bit is low, the NMB is either preparing for scan or exiting scan. At this time, there should be no MIM_CLKs to give the NMCs time to change to or from scan mode.

GATED_RSCAN_CTL<0> is a registered copy of XC_MB.SCAN_CTL<0> OR'd with SCAN_CTL<2> so that it is forced high during ALL scan even though the registered SCAN_CTL signal itself will be changing since that register is part of the scan ring. When it is zero, scan is finished. This signal is registered because it can change while clocks are running and therefore must be synchronized to system clocks. SCAN_CTL<2> only changes while the NMB's clocks are completely stopped and therefore does not require synchronization.

Figure 3-24 MIM_CLK Timing



Since QCOUNT is only a three bit count and there are thirty two NMCs, on any one clock of scan four MIM_CLKs will be generated. In Figure 2-17 on page 41 of the Interface chapter, the NMCs are shown in the LOG ring as being in columns of four. These sets of four are the NMCs which receive their clocks simultaneously. These NMCs scan together and basically act as a single register four times as long as a single NMC is by itself.

The 5303 PAL uses its state bit for each clock it is controlling to extend all MIM_CLKs by one additional clock beyond the condition which generated the clock. This means that MIM_CLKs

caused by a CHECK signal are asserted for one more clock than CHECK was asserted. In the case of scan clocks generated from QCOUNT, the MIM_CLK signal is asserted for two clocks, one when the QCOUNT equaled the number of the clock modulo eight and the second clock because the PAL extends all clocks by one. A two clock pulse safely meets the minimum clock pulse widths on the WREDC evening allowing for pulse shortening through the translators.

Figure 3-24 shows the timing for the signals in Table 3-14 for a LOG or SYS scan. Once QLOG_RUN* is de-asserted and RSCAN_CTL0 is asserted, all further B0E_CHECK signals will not cause an assertion of B0E_MIM_CLK. Once scan starts, the B0E_MIM_CLKs are generated each time the QCOUNT is a seven (since MIM_CLK is registered it is decoded from the previous's clocks QCOUNT). B1E_MIM_CLK is also shown for reference. B2E_MIM_CLK would follow the B1E clock and so on up to B8E which would occur simultaneously with B0E_MIM_CLK.

The state machine in the 5303 PAL is a simple state machine. It simply goes true and stays true while MIM_CLK is asserted except for the last clock of MIM_CLK.

The timing for an ALL scan is similar to that shown in Figure 3-24 except that the QLOG_RUN and RSCAN_CTL signals are not used. Instead, clocks are halted on the NMB while it is prepared for scan.

Because of latencies in the service processor interface to the scan engine, the NMB is guaranteed a large number of clocks (several hundred or more) between LOG_RUN being changed and the scan controls being changed (Figure 3-24 shows the LOG_GUN to RSCAN_CTL0 as shorter than it really is). This delay between scan events assures that the NMB has a sufficient amount of time to propagate its scan signals to all destinations.

3.5.3.2 QCOUNT State Machine

The 5300 PAL is the heart of the MIM_CTL. It controls the actions of the 5303 pals and sequences the scan operations. The PAL implements a two bit state machine, the three bit count used for asserting MIM_CLKs, and the ENABLEs that go to the 5303 PALs. All of these signals are decoded from the state machine state. The states of this controller are detailed in Table 3-15.

Table 3-15 NMC Scan States

Name	Code	Action
IDLE	10	Normal operation. No scan.
PREP	01	Preparing for Scan.
SCAN	11	Performing Scan.
TOFF	00	Turn off: preparing for normal operation.

In its simplest operation, the state machine goes sequentially through the states in the order listed in the table and then back to the IDLE state as it goes to and from LOG and SYS scans. The ALL scan throws in a complication because it must go directly to and from the IDLE and SCAN states since clocks are not running during the equivalent of the PREP and TOFF periods.

The state machine has five inputs listed in Table 3-16. Some of these inputs are commonly grouped in the PAL equations. These groups and their meanings are given in Table 3-17.

Table 3-16 Clock State Machine Inputs

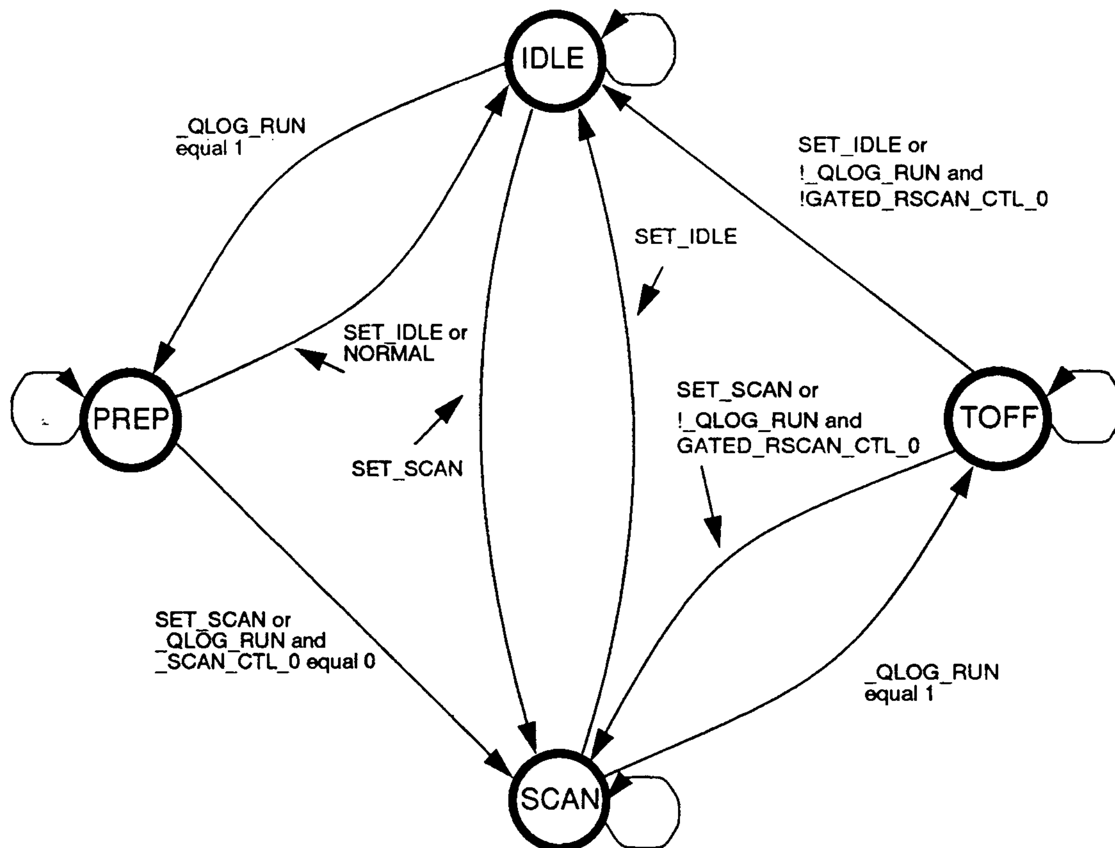
Name	Function
_SCAN_CTL_0	Active low copy of XC_MB.SCAN_CTL<0>
SCAN_CTL_2	XC_MB.SCAN_CTL<2>
GATED_RSCAN_CTL_0	Registered version of XC_MB.SCAN_CTL<0> OR'd with SCAN_CTL<2>
HOLD_MODE	True if SCAN_CTL is four or five.
_QLOG_RUN	Active low, registered version of -XC_MB.LOG_RUN

Table 3-17 Input Decodes

Combination	Condition
HOLD_MODE & _SCAN_CTL_0	SCAN_CTL mode four, SET_SCAN
HOLD_MODE & !_SCAN_CTL_0	SCAN_CTL mode five, SET_IDLE
!_QLOG_RUN & _SCAN_CTL_0	Non-scan log run, NORMAL

The operation of the state machine is specified by the state diagram in Figure 3-25.

Figure 3-25 State Diagram for NMC Clock Controller



To keep Figure 3-25 readable some of the details of the state machine are omitted. In cases where multiple paths are specified, the SET_SCAN and SET_IDLE path always have highest priority. The following figures illustrating how the state machine works during the different scan modes will indicate its normal operation. Consult the source code for the PAL for details.

This state machine can basically be used in two mode: ALL scan and LOG/SYS scan (LOG and SYS scan are identical as far as the NMC clock generation is concerned).

ALL scan was discussed in Section 2.1.5.3 on page 21 of the Interface chapter. To repeat the ALL scan procedure, if clocks were running, they are halted. While clocks are stopped, the SCAN_CTL lines are switched to a value of four. A single clock is issued with SCAN_CTL value. SCAN_CTL is then switched to value seven, ALL scan mode. Clocks are restarted for the number of bits in the scan ring plus XCL staging. After scan stops, SCAN_CTL is switched to mode five, a single clock is issued and then SCAN_CTL is set back to zero for normal mode after which clocks can be restarted.

From Figure 3-25 it can be seen that the clock with SCAN_CTL of four (SET_SCAN) forces the state machine to the SCAN state no matter what state it was currently in. This prepares the state machine for scan operation. After scan, the clock with SCAN_CTLs of five (SET_IDLE) forces the state machine to the IDLE state.

Figure 3-26 NMC Clock Controller during ALL Scan

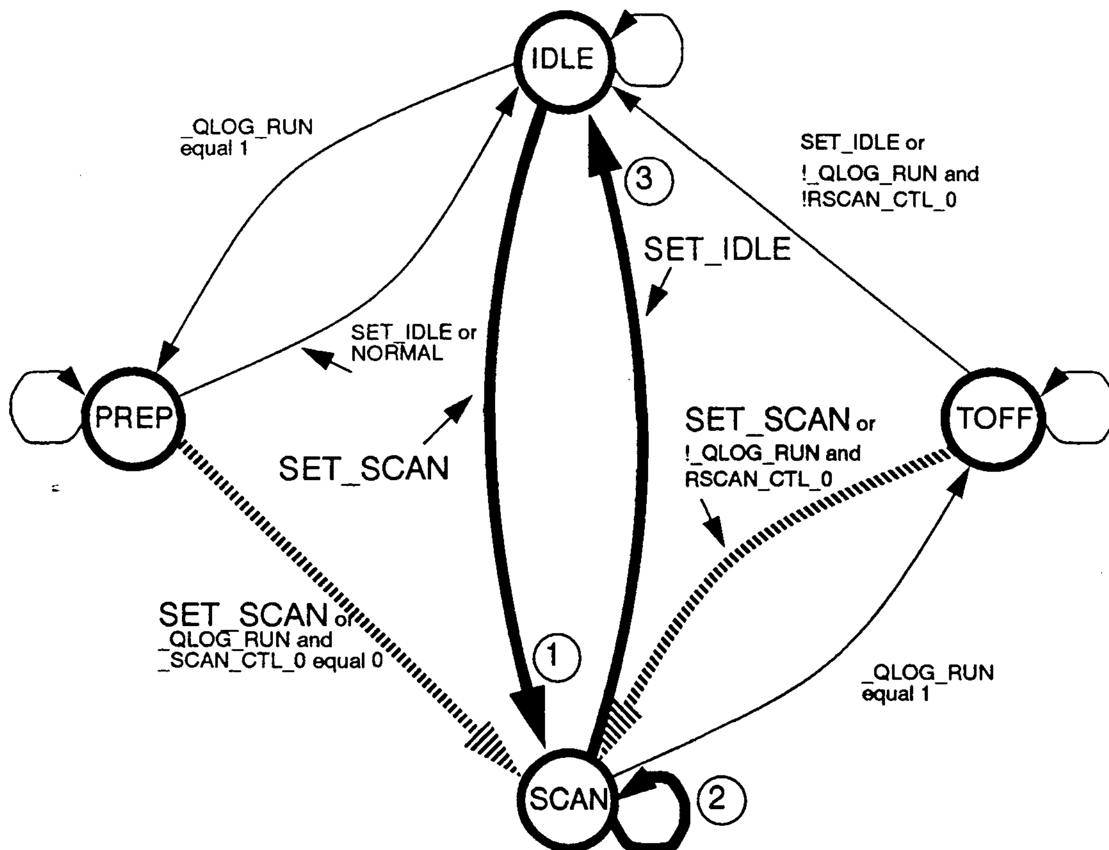


Figure 3-26 is a copy of the previous figure with the arcs used during an ALL scan highlighted. In general, an ALL scan is started with the state machine in the IDLE state. The first transition,

labeled "1" occurs when the SCAN_CTL signals are set to four and the NMB is clocked once (the arc labeled SET_SCAN). The next step is to perform the scan operation. This is the arc labeled "2" which loops back on the SCAN state. The last step is to transition back to the IDLE step on the arc labeled "3". This occurs when the scan controls are put in mode five and another single clock is issued. The emphasized dashed lines indicate alternate paths to the SCAN state on the first arc. If the NMB is in an unknown state as after power up, the state machine will go to the SCAN state no matter where it started.

LOG and SYS scan is more complicated than ALL scan because these scan operations occur while the NMB's free running clocks are still active. Care must therefore be taken that no timing constraints are violated when transitioning from non-scan to scan mode. In ALL scan, the clocks were stopped during these transitions which allowed more than enough time for signals to settle out. The transitioning to and from scan mode occurs in the PREP and TOFF states of the state diagram.

Figure 3-27 NMC Clock Controller during LOG/SYS Scan

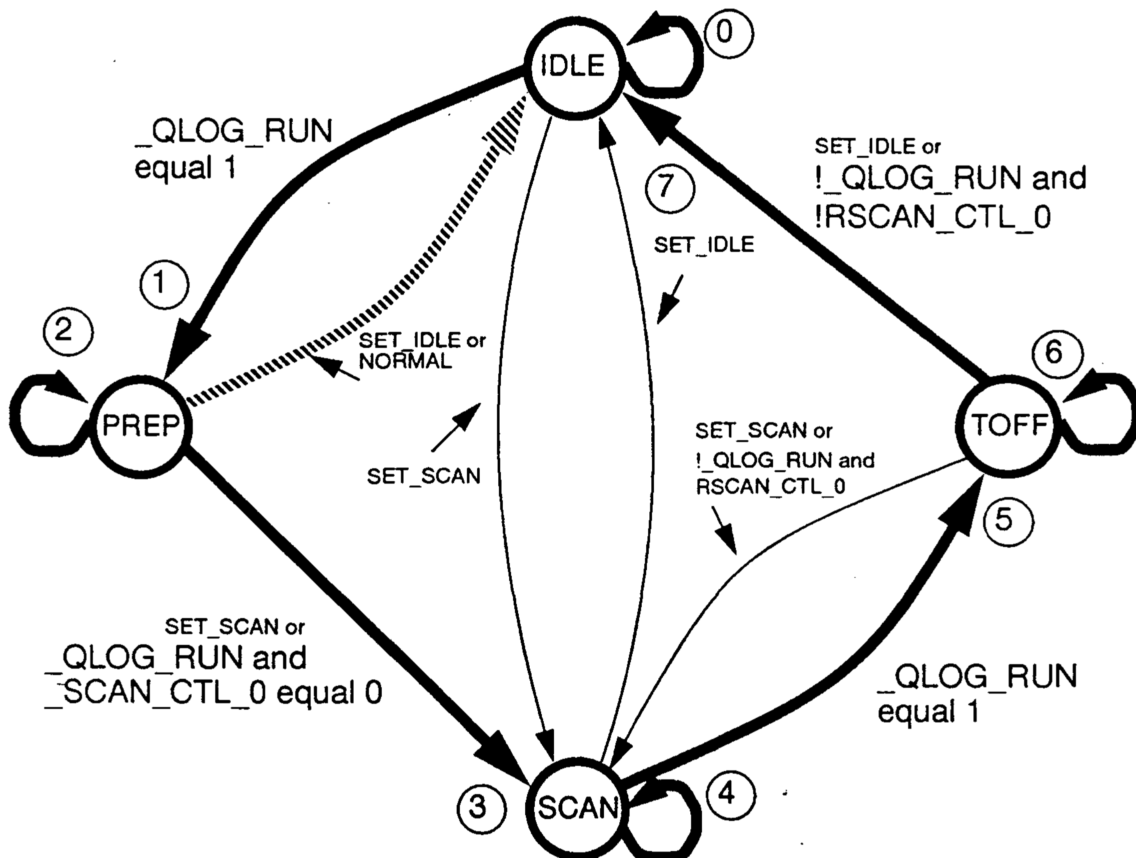


Figure 3-27 shows the steps taken to perform a LOG or SYS scan. The bold arcs are the arcs taken during LOG or SYS scan. The hashed arc is used if the RUN bit is turned off and then back on without ever going into scan mode.

Figure 3-28 Details of LOG/SYS Scan

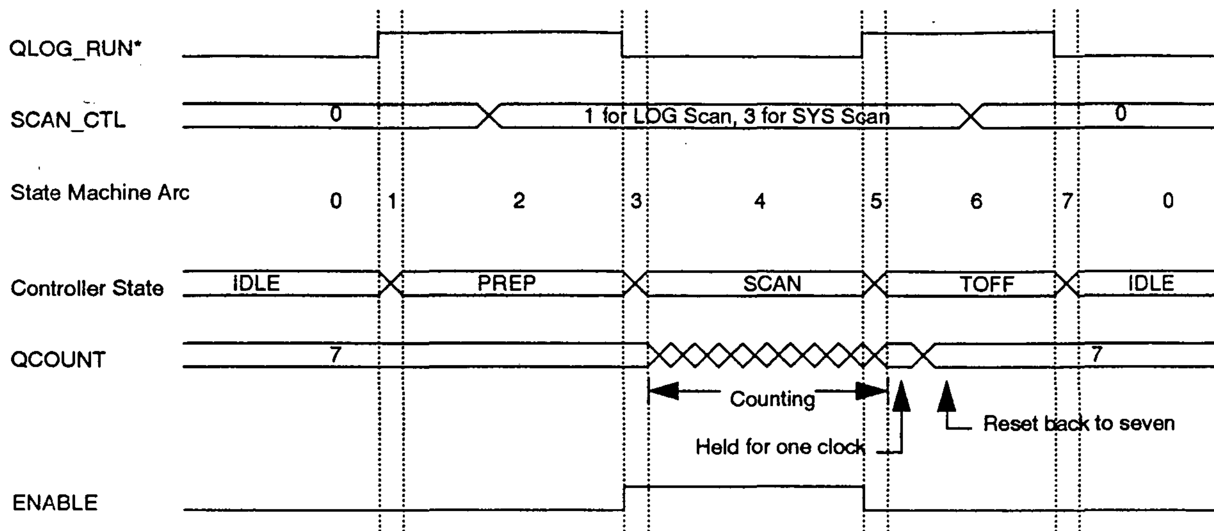


Figure 3-28 relates the inputs used for LOG and SYS scan, QLOG_RUN* and SCAN_CTL, to the arcs on the state diagram, the states and the outputs derived from this state machine.

The state machine progresses in a simple circle during LOG/SYS scan. It goes from IDLE to PREP when QLOG_RUN* is turned off. During PREP, the CHECK signals are inhibited from causing MIM_CLKs. When QLOG_RUN* is re-asserted, the state machine moves to the SCAN state where QCOUNT begins counting and the ENABLE signals allow the 5303 PALs to generate MIM_CLKs from the count. Following the de-assertion of QLOG_RUN*, the state machine moves to the TOFF state where CHECKs are still inhibited from causing MIM_CLKs. WIth the re-assertion of QLOG_RUN* signalling the end of the LOG/SYS scan, the state machine returns to the IDLE state.

The SCAN_CTL signals are only allowed to change while LOG and SYS run bits are disabled. The LOG/SYS scan portion of the state machine uses the registered version of SCAN_CTL, GATED_RSCAN_CTL. The ALL scan portion of the state diagram can not use the registered scan control because the register is part of the ALL scan ring and would be scanned during ALL scan.

3.5.3.3 Scan and Error Logging

Since all soft error state is in the LOG ring and all error state, hard or soft, is in the SYS ring, scanning either of these rings will prevent error state from being saved. That is, if the NMB detects a soft error, for instance, while the LOG ring is being scanned for a previously detected soft error, there is no place to put the error state which is normally saved for a soft error. The soft error will still be corrected but the SPU will never be informed of that error and no information will be retained for that error.

3.6 Single Step

In Section 2.1.5.7 on page 23 of the Interface Chapter, memory system single step was discussed from the perspective of the world outside of the NMB. In brief, the NMB can be made to operate so that it appears that its clocks are being stepped one by one (single stepped), bursted or halted. In reality, the NMB's free running clocks, CU_MB.FREE_CLOCK2, are not halted; only the

interface logic is stopped in step with the rest of the system. The logic which interfaces with the crossbar, the input staging registers, the RDEDC and similar logic, is halted so that it can be run in step with the rest of the system while the logic necessary to maintain refreshes keeps running at all times.

To successfully implement single step logic, there most be some interface between the free running logic and the logic which is lock-stepped with the system logic. The BCGAs control this interface. Details of how the BCGA control single step are covered in the BCGA functional specification document. The interface boundary and the board level flow for single step is covered in the next section.

3.6.1 System Clocked/ Free Running Boundaries

Figure 3-29 is a modified version of Figure 3-1 on page 46 showing the boundaries between the free running and system clocked state. All logic below and to the left of the shaded line is run in lock-step with the system step. All logic above and to the right is free running state.

To keep the system clocked state in lock step with the rest of the system, the NMB's XC_MB.RUN_SYS and -XC_MB.RUN_LOG bits must be asserted and de-asserted with the clocks going to the rest of the system. The proper timing for the RUN bits is described in Section 2.1.5.7 on page 23 of the Interface chapter.

The system clocked state in Figure 3-29 can be divided into two groups: input staging and output staging. Input staging is the registers which hold the request state on the clock it is received from the crossbar. These registers are labeled "IS" in the figure.

The output staging logic consists of the return data logic which is the RDEDC, LDREG logic and the RTN_SEL and LD_EN registers in the BCGA and the BANK_DONE registers.

Any request that enters the input staging registers is immediately processed, even if -RUN_SYS is de-asserted on that clock. Thus, once a request enters the input staging registers, the free running logic takes that request and performs the requested DRAM operation. When clocks are restarted, the request has already been processed, even though it appears that the request was simply sitting in the input staging registers. This occurs, basically, because the NMB can not tell that the SYS clocks have stopped until the clock after they have halted. In the meantime, it has processed the request in input staging.

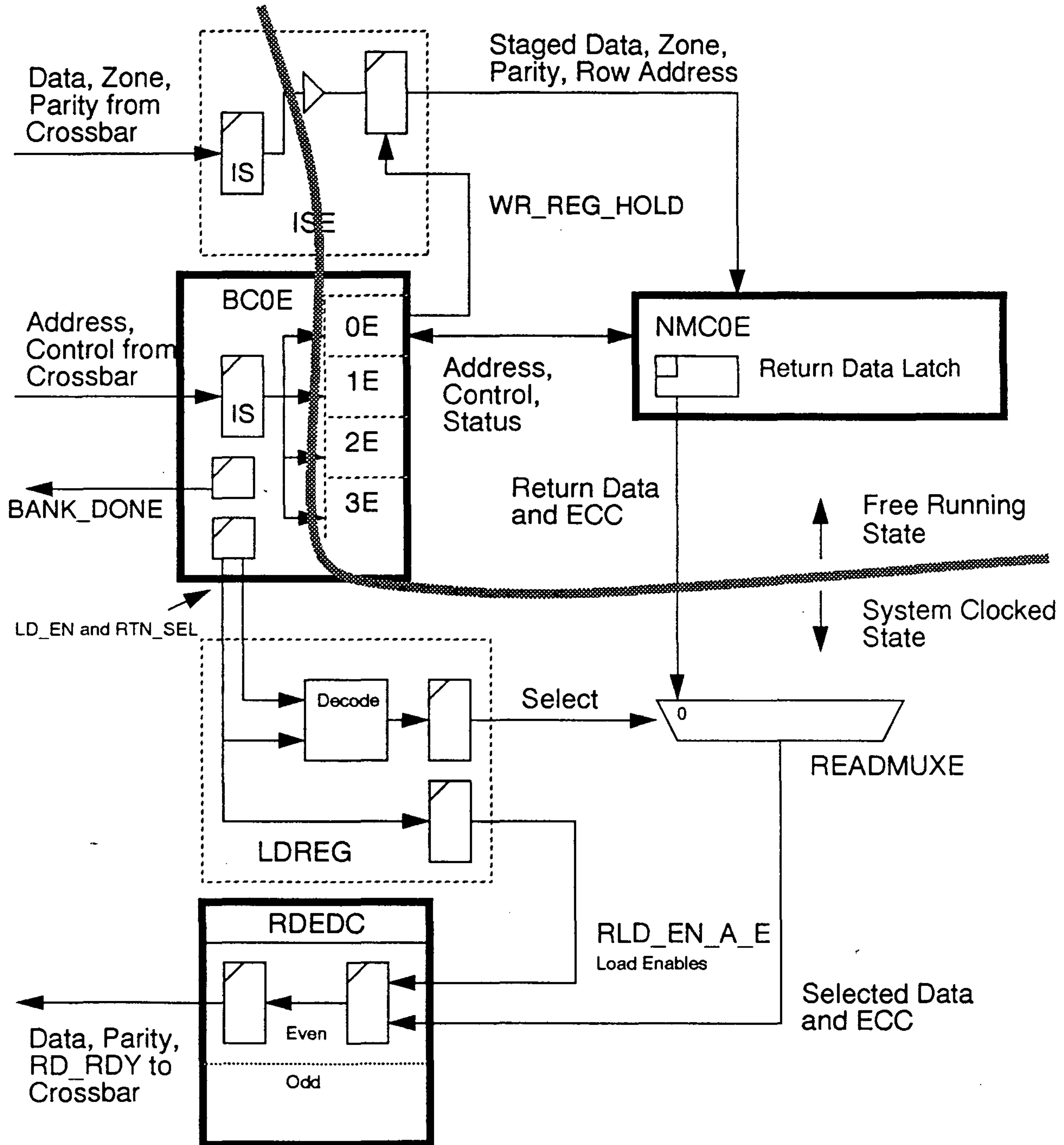
Note: Since a request is processed as soon as it enters the input staging registers, requests in the input staging registers are ignored when clocks are restarted since the NMB assumes it has already processed that request. Thus, it is not possible to scan a request into the input staging registers then have the NMB process that request when clocks are restarted. In order to perform memory operations from scan, the request must be scanned into the crossbar, then clocked into the memory board.

Any requests in input staging or in progress when clocks are halted is allowed to run to completion in the free running logic. Because of refresh and DRAM timing constraints, it is not possible to halt a DRAM in the middle of an operation. It must be allowed to complete whatever it was doing so that it can do refreshes while system clocks are halted.

Even though the DRAM operation has been allowed to complete, to the rest of the system it must look as if the request completed when the proper number of system clocks (SYS clocks) has occurred. The two events which must be synchronized with the SYS clocks are the return of data

and the BANK_DONE signals. These are the only two events which return to the crossbar and result from a request to the NMB.

Figure 3-29 System/Free Running Boundaries



For BANK_DONEs, the BCGAs keep track of when this information should be sent back to the crossbar, regardless of how often and for how long SYS clocks were stopped. When the proper

time occurs, the signal is regenerated and placed into the proper register so that it eventually makes it way back to the crossbar.

Return data is somewhat more complicated in that return data and a RD_RDY must be sent back to the crossbar. Since the read operation completed long before the data is to be returned, the read data must be saved some place. The place data is saved is the latch in the TTL to ECL translators on the NMCs. This latch is shown in all the dataflow diagrams for the DRAM operations. This latch is only opened during a DRAM operation which returns data (reads or test-and-modifies) and is closed at all other times. Therefore, this latch holds the last read data. It will not hold data read from the DRAMs during a partial write.

With the last read data preserved in the NMC latches, the BCGAs need only make sure that the LD_EN and RTN_SEL signals occur at the proper time with respect to the system clocks so that the proper data out of the latches is selected by the READMUX and returned through the RDEDC.

3.6.1.1 Example of Single Step

To clarify the previous material, an example of SYS clock stop and restart will be examined in detail. The example covers a stream of reads to sequential banks beginning at bank 0. The clock stop occurs in the middle of the stream.

Figure 3-30 Single Step Example Timing Diagram

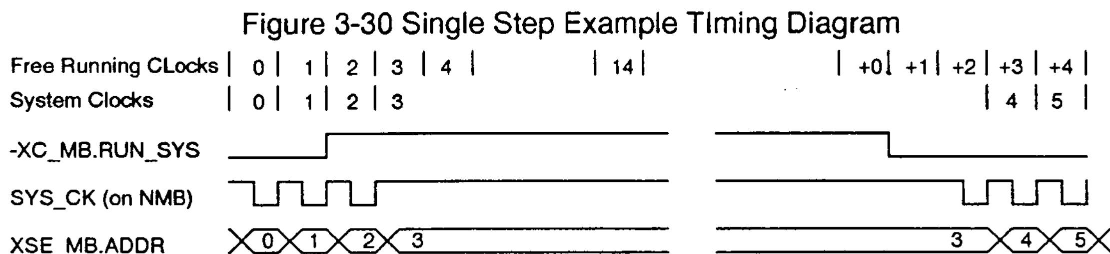


Figure 3-30 shows the timing for the example. The gap in the middle represents an unspecified amount of time between turning and turning off the RUN_SYS bit. The clocks in the figure are numbered with respect to the system clocks and the free running clocks. After the pause the free running clocks are simply numbered +1, +2, etc. since the free running clock number is here the number of clocks elapsed since clock zero (what ever that number is, probably very large) plus one, plus two, etc. The system clocks simply count the number of system clock edges. Since no edges occur between free clock number 3 and "+3," the system clock count does not increase.

SYS_CK represents one of the system clocks on the NMB. These are the clocks gated by RUN_SYS. RUN_SYS is registered before it gates the SYS_CK signal which accounts for the delay between RUN_SYS being de-asserted and SYS_CK stopping. The address line represents

the address on the backplane between the NMB and crossbar. It advances with each SYS_CK in the example.

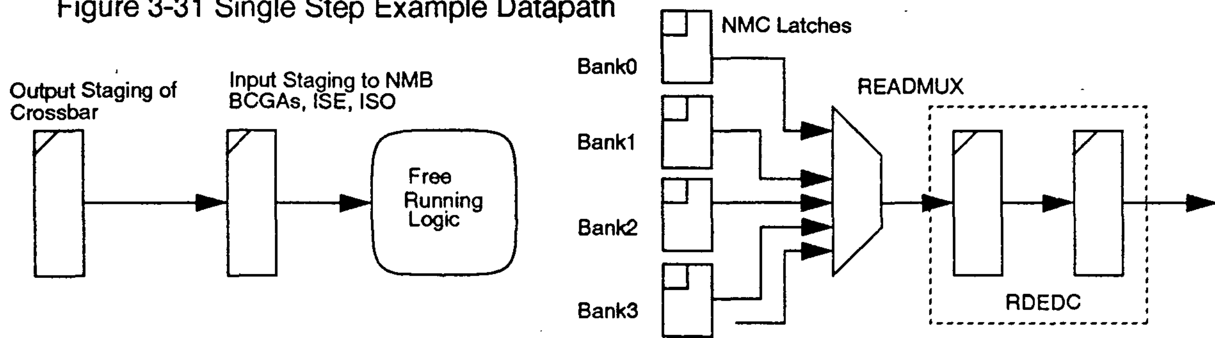


Figure 3-31 Single Step Example Datapath

Figure 3-31 is a very simplified diagram of the NMB datapath for reads. Requests are sent fro m the crossbar output staging register at left into the NMB input staging register. The input staging register represents address and control staging in the BCGAs as well as data and zone staging in the ISE and ISO logic. After the input staging registers, the requests enter the free running logic on the NMCs and in the BCGAs. Return data, as described in Section 3.3.2 on page 59, goes from the DRAMs, into the latches on the TTL to ECL translators on the NMC, through the READMUX and into the RDEDC. All registers shown in the figure are system clocked except for the latches.

Figure 3-32a and following figures walk through the clocks of Figure 3-30. The circled numbers in the figures represent the location of the request to the bank whose number is within the circle. Thus at clock 0, the only request present is a request to bank 0 in the output staging register of the crossbar.

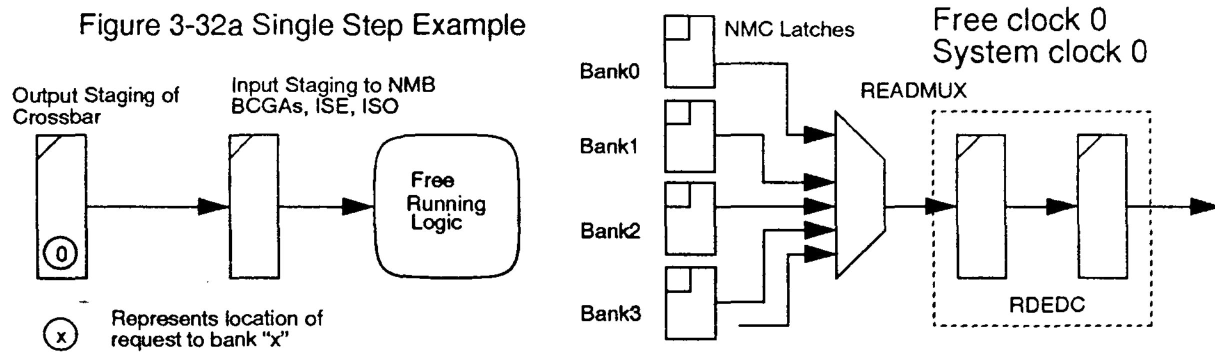


Figure 3-32a Single Step Example

Clock 1:

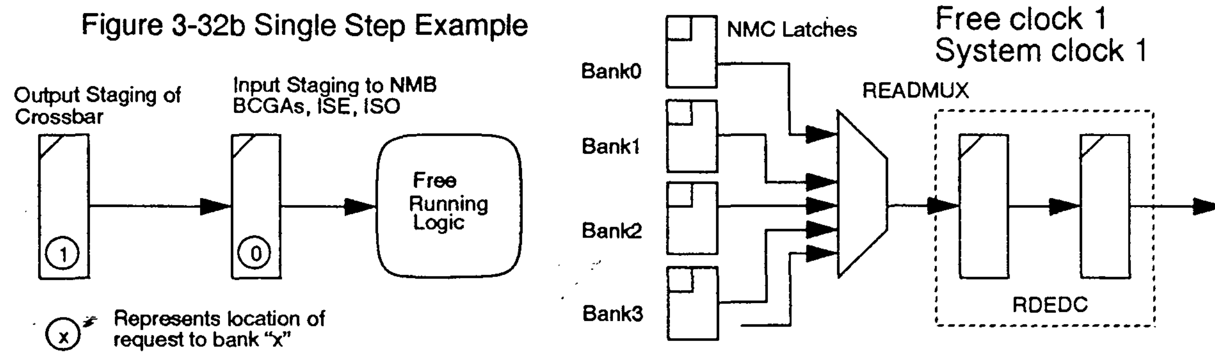
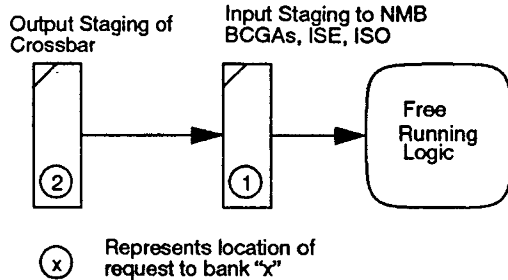
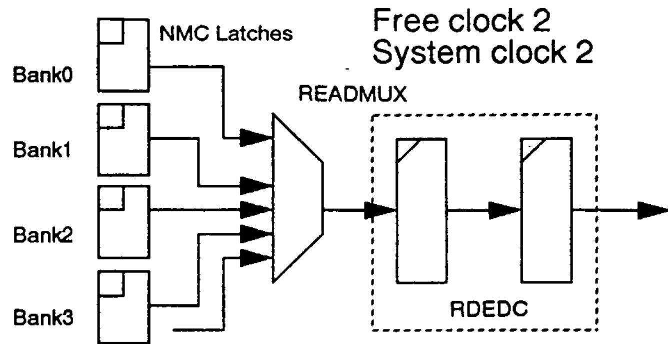


Figure 3-32b Single Step Example

Clock 2:

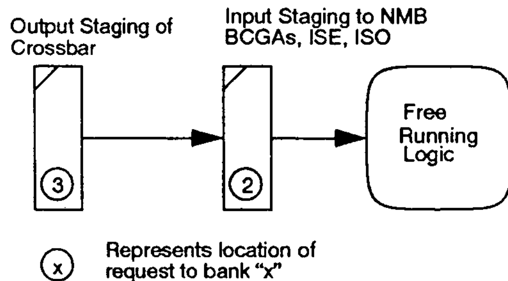
Figure 3-32c Single Step Example

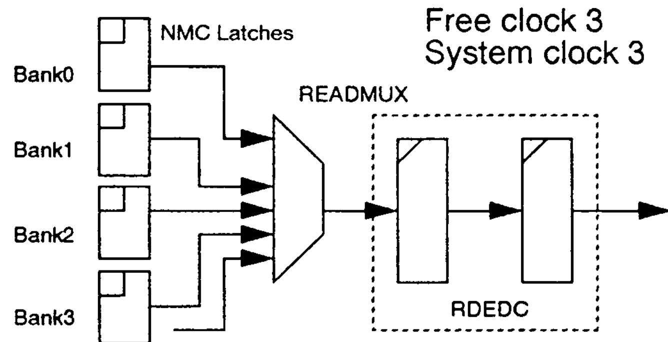




Clock 3. This is the last system clock edge. The requests to bank two is in the input staging registers while the request to bank three will be clocked into the input staging registers when the next system clock occurs.

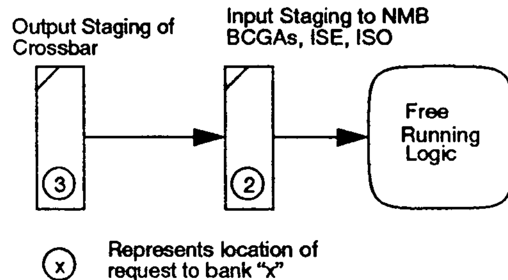
Figure 3-32d Single Step Example

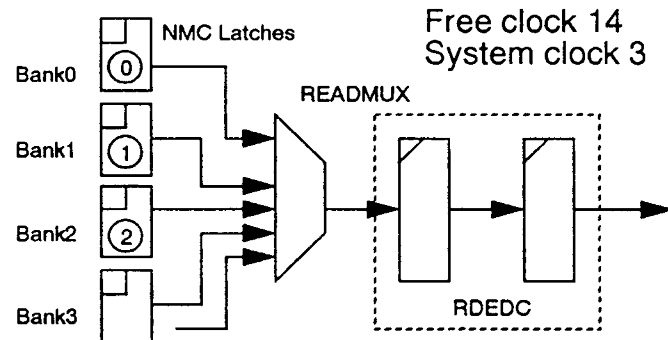




Clock 14. By clock fourteen, all the reads that were started before the system clocks stopped have progressed to the point where their read data has been captured in the NMC laches. The bank numbers inside the latches in the following figure indicate the data that has made it to that point. Note that this includes the data for bank two. Even though the request for bank two is stuck in the input staging register because of the system clock halt, the request was still processed. When the system clocks restart, the data in the input register before the first system clock edge is ignored; it has already been processed.

Figure 3-32e Single Step Example

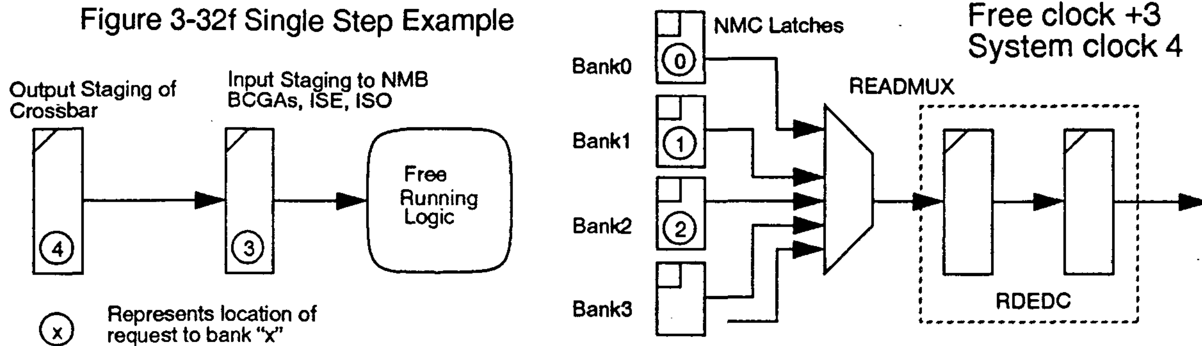




Clock +2. Regardless of how long the clocks have been stopped, the data remains as in Figure 3-32e.

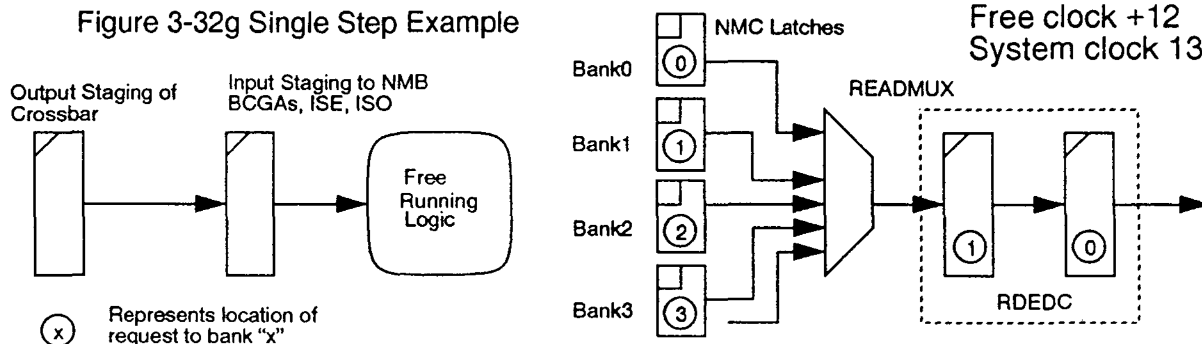
Clock +3. System clocks resume while the data in the latches waits for the proper clock to be selected by the READMUX.

Figure 3-32f Single Step Example



Clock +12. On this clock, the return data for the request to bank zero is finally ready. Note that this occurs on system clock 13, just as it does in Figure 3-6 on page 60. (Clock zero in Figure 3-30 on page 104 is the same as clock zero in Figure 3-6 for the request to bank zero.)

Figure 3-32g Single Step Example



The remaining pieces of return data would follow on subsequent clocks.

3.6.2 Accessing Memory During Clock Stop

During the course of debugging a board using diagnostics, it is often desirable to halt the system clocks and check the contents of memory. On the C2, the memory board would be left running and the SPU could directly access memory through the SPU's direct access to the memory E port. On C3, the SPU must access memory through the crossbar.

Since the SPU must use the crossbar for accesses, it must therefore turn at least some of the system clocks back on to get to memory. Turning the crossbar on is the same as turning the system clocks back on as far as the memory system is concerned which is a potential quandary: to access memory while the clocks are off, the clocks have to be turned back on.

The procedure to deal with this problem requires a large state save. The entire state of the crossbar must be scanned out and saved for later restoration. The NMB's SYS ring state must also be similarly saved. Then, the crossbar must be initialized with all the processors disabled. The NMB's SYS rings must be initialized with the sst_enable bit in each of the BCGAs set zero, the off state.

Once this is done, the crossbar and memory clocks may be restarted and memory freely accessed by the SPU. Of course, any writes to memory will be seen by the processors when they are

restarted. As long as the sst_enable bits are off, the BCGAs will remember the BANK_DONEs and RD_RDYs that correspond to the first time the SYS clocks were halted.

After SPU access are finished, the SPU must decode the NMB SYS state scanned earlier and issue a read request for any reads it found logged in the BCGA's LOG registers. These registers hold the address and cycle type of the last operation to a bank. By redoing any reads, the latches on the NMCs are reloaded with the data they held before the SPU accesses. Now the NMB SYS clocks and crossbar clocks are halted and the state saved earlier is restored including the sst_enable bits which should be one, enabled.

It is not possible from the SYS state to tell whether the operation in the bank logging registers is currently in progress or was completed since the registers will hold a request until the next request comes along.

Note: Since the restore process counts on being able to do a read to a bank in order to restore what was in the bank's data latch before the SPU operations, test-and-modifies operations can not be recovered. The test-and-modify would have returned data which was immediately overwritten by the second half of the test-and-modify. If the SPU did a read to that location, it would find the 'modified' information, not the information that the requestor should not have seen. This could cause a lock byte to be set without any processor ever seeing it clear first. If there are any outstanding test-and-modify operations at a bank and the SPU does an operation to that bank, the memory system can not be completely restored following the SPU transactions.

When the processors, crossbar and NMB sys clocks are resumed, the system should pick up where it left off. Table 3-18 below briefly states the indicated procedure.

Table 3-18 Memory Access During SYS Clock Stop

After all non-free running clocks are halted in the system:

Step 1:
Save the contents of the crossbar, NMB sys ring and any necessary NIA state.

Step 2:
Set the sst_enable bits on the NMBs to zero.

Step 3:
Disable all processors at the crossbar except for the NIA.

Step 4:
Turn on clocks to the crossbar and the non-free running clocks to the NIA and NMBs and make any desired memory requests.

Step 5:
Have the SPU re-issue any reads that it found in the NMB logging state. If any banks had a test-and-modify in their logging state, the restore may not work.

Step 6:
Turn off non-free running clocks.

Step 7:
Restore all state saved in step 1.

Step 8:
Restart clock when ready to continue.

3.7 Initialization

The NMB has no special initialization requirements except for initialization of the DRAMs. Hard errors must be enabled, error state zeroed out, and good parity scanned into the input staging registers and the RDEDC.

It is important to initialize all the NMB phase generators. The 2x phase generators must be zeroed out and the 1x phase generators must be initialized with a 0101 pattern (most significant to least significant).

The DRAMs must at some point be initialized so that all words contain correct ECC. Since full writes do not perform an ECC check (they do not read the contents of the DRAMs), full writes can be used to write all of memory once the board has otherwise been initialized.

Therefore, the initialization sequence is to initialize the scan ring, turn the on the system including the memory board, then write all of memory using long word, aligned writes. Partial writes or reads before writing memory will cause a hard error to be detected.

4 Programming the NMB

4.1 Programming Fields

The NMB can be programmed via scan to operate the DRAMs either faster or slower than currently programmed. A set of four registers in the ALL ring of each of the BCGAs controls how the NMB will cycle the DRAMs. The field names for these registers are given Table 4-1.

Table 4-1 BCGA Timing Registers

Field Name	Current Value	Purpose and Restrictions
bc[x].all.start_sel	4	Three bit field. Determines RAS off, return time.
bc[x].all.done_sel	6	Three bit field. Determines end of cycle after RAS.
bc[x].all.predone_sel	6	Three bit field. Must be identical to done_sel
bc[x].all.rmw_sel	2	Two bit field. Determines extra clocks for RMW.

These fields specify half clock offsets for controlling times in a DRAM cycle. Each unit alters a DRAM timing parameter by half of a system clock (approximately eight nanoseconds). The lower the number in the field, the fewer number of clocks an operation takes. The fields have been set up so that the standard 85ns DRAM at a 14ns clock timing uses values at the upper range of the fields.

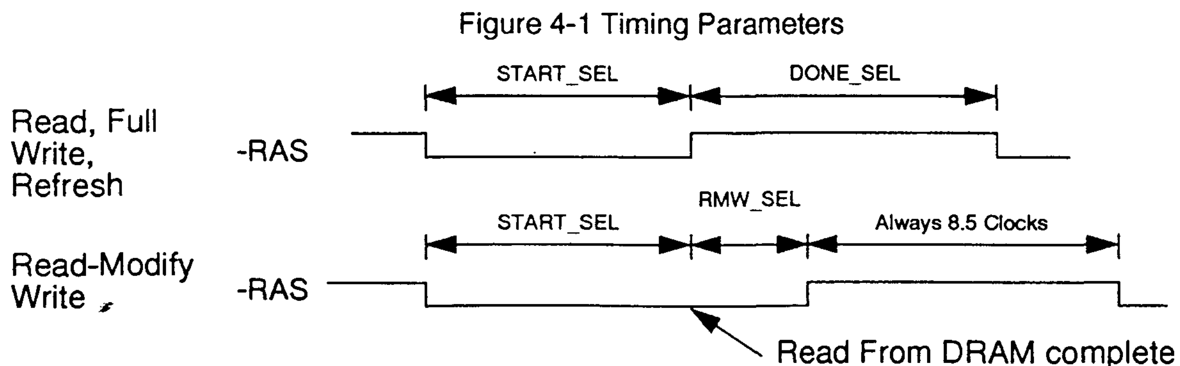
4.2 Timing Parameters

The fundamental parameters controlled by the programming fields listed above are the RAS low time and the RAS high time. The RAS low time specifies the access time of a DRAM. The RAS high time specifies the RAS precharge time. Together, the two times specify the total cycle time for a DRAM full write or read. In modern DRAMs, the other timing parameters, CAS* assertion, address switching and data set up and holds are not timing critical.

The done_sel and predone_sel fields are basically the same field and should be set to the same value. Setting predone_sel to less than done_sel should cause control hard errors (ctl_he equal one) to occur.

The read-modify-write (RMW) operation has a third timing parameter which specifies the length of the write portion of the RMW. The RMW first does a read whose length is specified by start_sel, followed by a write specified by rmw_sel finishing, with the RAS precharge as specified by a fixed period of time. Figure 4-1 shows the timing parameters with respect to RAS.

Figure 4-1 Timing Parameters



4.3 Timing Parameter Values

Each of the timing fields contributes a certain number of clocks to the total memory cycle time. The clocks for each field are listed in the following tables.

Table 4-2 Timing Field Values

start_sel		done_sel		rmw_sel	
Value	Clocks	Value	Clocks	Value	Clocks
0	4.5	0	4.5	0	4.0
1	5.0	1	5.0	1	4.5
2	5.5	2	5.5	2	5.0
3	6.0	3	6.0	3	5.5
4	6.5	4	6.5		
5	7.0	5	7.0		
6	6.5	6	7.5		
7	7.0	7	8.0		

The cycle time for various operation can be determined from these charts. For instance, given the current timing as listed in Table 4-1, the number of clocks needed for a read, full write or refresh is the start timing plus the done timing. The start_sel is currently set for four which gives 6.5 clocks. The done timing is six which yields 7.5 clocks. The sum is 14.0 clocks which is the number of clocks shown for these operations in the other chapters of this document.

A RMW operation adds an additional number of clocks to the cycle time as given in the above table. In addition, RMWs always spend 8.5 clocks in the "done" portion of the cycle regardless of the done_sel value.

Note: The sum of all timing parameters must be a whole number. The sum of the start_sel + done_sel must be a whole number as well as the sum of start_sel + rmw_sel + 8.5.

5 Build Procedure

5.1 Introduction

The Build Procedure chapter covers the tools, steps and pitfalls in order to respin the NMB. It is assumed that the board has been verified at the schematic level either or functional simulation or through a spin check program which verifies that the new schematics matches the old schematics plus the ECNs to the old schematics.

The NMB follows the standard steps for a Neptune style board. Some of these steps, namely the ones that do a preliminary timing verification of the board or other preliminary check, may be omitted in a respin. This chapter will primarily deal with the steps for a respin where a complete reroute and part assignment is not required.

5.2 Tools

A full respin of the NMB involves a number of CAD tools. These tools are identified in Table 5-1.

Table 5-1 List of CAD Tools

Name	Function:
itv	Interactive timing verifier, static timing verification.
n2	ENDOT functional simulator.
cvt_analyze	Creates data files for system interface timing verifier.
mfgphys	Creates database for manufacturing.
compile	VALID schematic compiler. Extracts basic part, section and net information from schematics.
package	VALID packager. Extracts netlists, assigns part (location) numbers. Performs package type checks.
placetool	Used to create and edit placement information for physical parts. Used to view and edit wire routes. Performs some design rule verification.
netsched	Determines order of nodes on nets (scheduling). Performs via and terminator assignments. Generates back annotation files for location numbers. Performs some topology and testability checks. Generates pre-route timing files.
batchdrc	Performs full design rule verification. Much faster than DRC in placetool.
conroute	Convex multiwire router. Generates all wire route information.
dwirsif	Generates from-to files for input to conroute or checks wire files and puts illegal routes back into from-to files.
tlc	Transmission line calculator. Generates timing files for itv based on transmission line effects using routed wire lengths and device driver and load characteristics.
ged	VALID Schematics editor.
clktune	Modifies the scheduler length files so that all clock delays are balanced.

A detailed description of each of these tools can be found by using the cadman utility. Cookbook instructions for some of these tools also exist. The tools will be described in more detail as necessary in the following sections.

5.2.1 Command and Log Files

Most CAD tools used at CONVEX, both those internally produced and those purchased from outside vendors, use the same conventions with respect to the naming of command and log files. Command files are generally called <CAD program name>.cmd. The command file will contain various parameters for the program plus the locations of necessary files, libraries and directories.

The log files are generally called <CAD program name>.log. Log files at a minimum contain all the information printed to the screen (stdout) during program execution. The log file may contain additional information. All warnings, advisories and errors should be listed in the log file.

The VALID tools are one exception. Compile and package produce cmplog.dat and pstlog.dat files respectively.

5.3 Design Directories

A respin should begin with the creation of a working directory for the design taken from the archive tape for the NMB. This directory will contain the following sub-directories:

Table 5-2 Design Directories

Directory	Purpose
artwork	Where the gerber files for the artwork for the multiwire board is created.
c3tv	Timing description of NMB interface for system timing.
mfgphys	Directory for creating data files for manufacturing.
pkg	Where all compiles and packages are performed.
place	Where placement, net scheduling and placement design rule verification is done.
post	A dummy directory with links to the real directories to fool cad tools. Make sure this directory contains only links, no files.
route	Where multi-wire routing is done and wire design rule verification is done.
schem	Schematics files for ged.
ship	Contains final artwork and wire files for release to Hitachi.
time	Where pre-route (estimated) timing verification is done.
time.tlc	Where tlc based (post route, transmission line) timing verification is performed.
tlc	Where the transmission line calculator (tlc) is executed.

The archive tape may contain other directories but only the ones listed in Table 5-2 are relevant for a respin.

5.3.1 Links

The tar utility which was used to create the archive tape may copy links as real files. The NMB directories contain many links that need to be restored by removing the file and restoring the link. If the link is not restored the cad tools will use obsolete and inconsistent data leading to many subtle or frustrating bugs.

5.3.1.1 VALID Links

When checking for links, some simple rules can be followed for most cases.

Files beginning with "pst," such as pstchip.dat, are packager output files. These files should only exist in the pkg directory. All other occurrences of pst files should be deleted and replaced with links to the appropriate file in the pkg directory.

Files beginning with "cmp," such as cmpexp.dat, are compiler output files. Only the cmpexp.dat file is generally needed; the other files such as cmplog.dat are informational files.

There are two types of cmpexp.dat files: a compile for logic or a compile for time. There should only be one of each type with all other occurrences being links. The only place compile for time files are found is in the timing directories. In the two time directories, one of the cmpexp.dat files should be the real file and the other should be a link to it.

The remaining cmpexp.dat files are compile for logic files. The real file should be in the pkg directory with all others being links to this file.

The compiler retains some state information between compiles to minimize the amount of information that must be recompiled when a new compile is performed. This information is maintained in the xshadow directory where the compile is performed. This directory exists only to make compiles faster; it can safely be deleted to save disk space. There need only be one of these files. All other places where compiles occur should have a link to the one xshadow directory. Both compiles for logic and compiles for timing can use the same xshadow directory.

5.3.1.2 CAD Tool Links

Any CAD tool executables, such a file called batchdrc.new, should be deleted or ignored. These were once links to early releases of CAD tools used while a particular CAD tool was still under development. The current CAD tools directories should contain the proper versions of all CAD programs.

5.3.1.3 Scheduler Links

Similarly to the VALID links, there are often links to the scheduler files. These files all begin with "sch," such as schstat.dat. Only the place should contain files beginning with "sch." All other files in other directories should be removed and replaced with links to the place directory.

5.3.1.4 Other Links

Most of the links are covered by the previous sections. A few are specific to certain directories.

The delay.dat file in the time directory should be linked to the schdly.dat file in the place directory.

The delay.dat file in the time.tlc directory should be linked the tlcdly.dat file in the tlc directory.

5.3.1.5 The "post" Directory

The post directory should only contain links to other files. It should not hold any real files. This directory exists solely because certain CAD tools at one time expected certain files to be found in a directory called "post." Even if this restriction is no longer valid, many of the NMB's CAD tool command files reference the post directory. It is therefore simplest and safest to replace all files in the directory with links.

5.4 Respin

Figure 5-1, Figure 5-2 and Figure 5-3 show the steps needed to respin the NMB. The respin procedure assumes that minor changes are being made to the NMB and that an incremental route can be done rather than a route that starts from scratch.

The first step in beginning a respin is to restore the design from the archive tape and modify the schematics to incorporate the ecns. In most cases, a version of the schematics with the ecns already incorporated will exist for sst purposes. These schematics made be used instead.

The modified schematics must be verified against the old schematics and the ecns to the old schematics using the spin check software. This software verifies that the new schematics are identical to the old schematics plus the ecns to the old schematics. The new schematics must pass this check before continuing with the design.

The steps in the following figures sketch out the procedure for a respin. It is beyond the scope of this document to cover all the details of the respin procedure. Sections following the figures do cover some of the details specific to the NMB.

Figure 5-1 Pre-route Steps for Respin

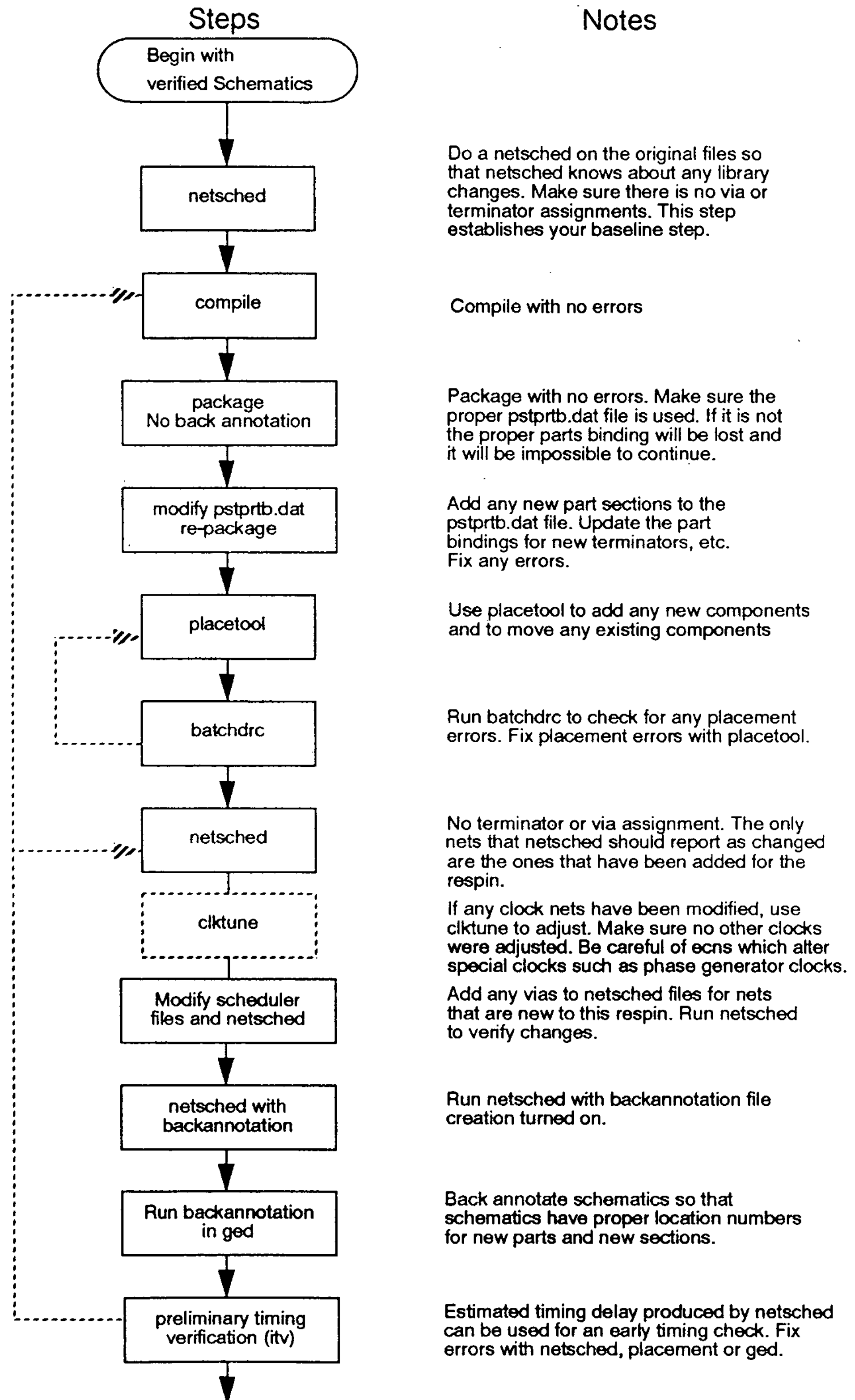


Figure 5-2 Route Steps for Respin

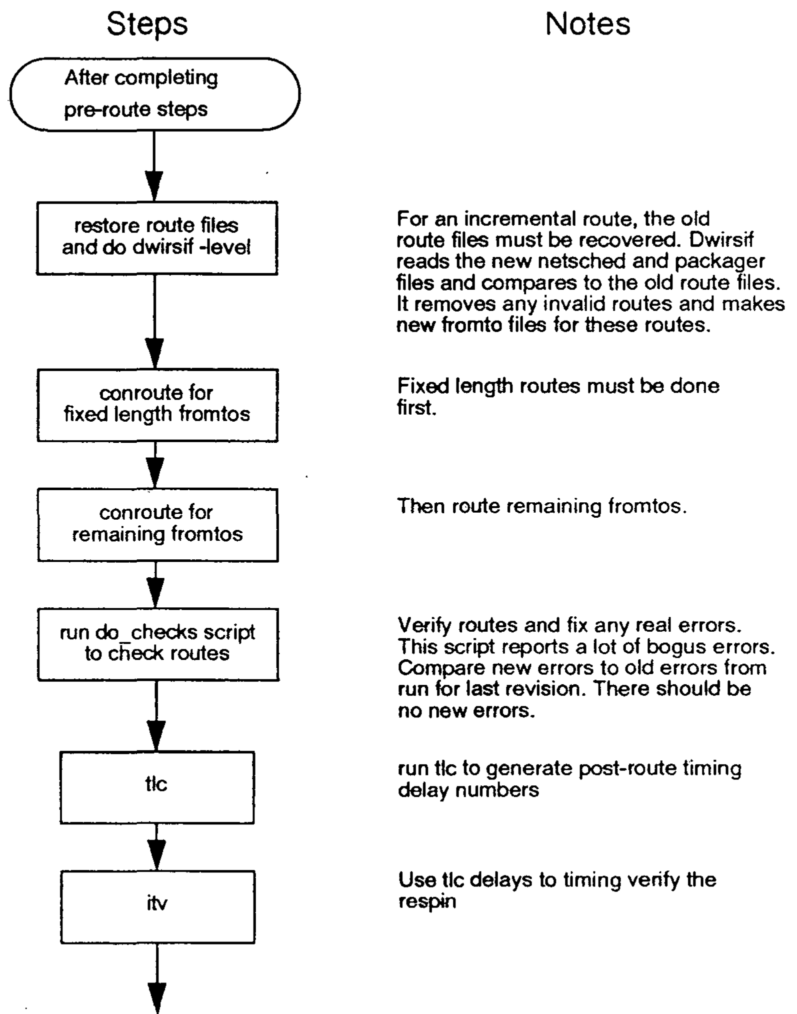
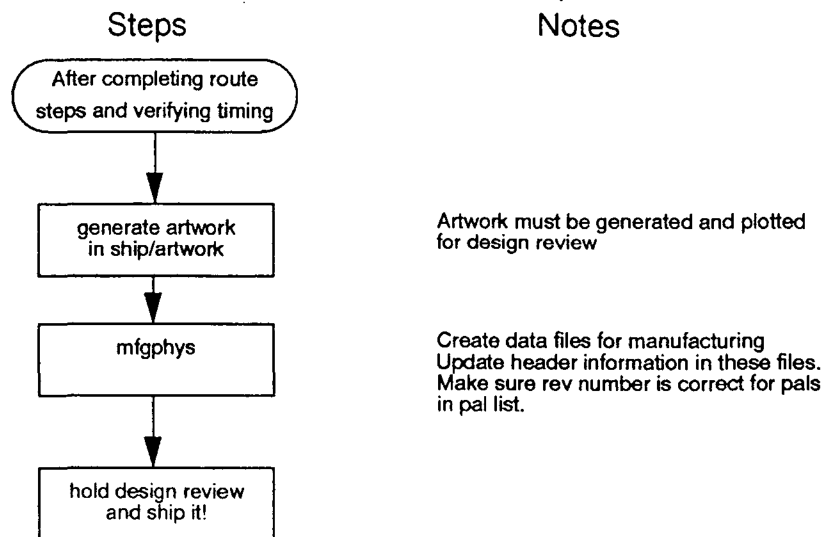


Figure 5-3 Post Route Respin Steps



5.4.1 Packaging

The packager is used to assign logical devices to physical packages as well as extract net information and information for parts used in a particular design. The packager uses the cmpexp.dat file in the directory it is executed in as input. Command options are specified in the packager.cmd file.

The pkg directory contains three packager.cmd files. Packager.cmd.ba is the one to use when doing a backannotate. Packager.cmd.noba is the one to use when not doing a back annotate. Copy either one of these two over the packager.cmd file when packaging.

Back annotates are used to feedback location number information generated by the netsched into the packager state files. It requires a pstback.dat file which is generated by netsched when it is told to generate this file. This is only necessary when new parts are placed on the NMB and these parts do not already have a U number binding in the pstprtb.dat file. For most ecns, the U number should already be specified in the ECN. This U number should be used when the part is added to the pstprtb.dat file.

The pstprtb.dat file is a very important file. It should always be saved somewhere before modifying it. It holds the parts bindings which are the association of a part path name to a physical part number. The physical part number can be any alphanumeric string. However, the location number (U000A0) should be used for the part number. The path name is the hierarchical name including path numbers for a logical part (what is placed in the schematics). Refer to the pstprtb.dat file for an example. The pstprtb.dat defines a logical part by a physical part number and a pin number. For parts with more than one slice per part (such as a terminator package with eight terminators), the pin number is very important for specifying which if the eight terminators is used.

When the first package is performed on the new schematics, the only parts which should change in the pstprtb.dat file are the ones that have been added as a result of ecns. Diff the new and old pstprtb.dat files to make sure nothing else has changed. Before the second package, these new

parts must have their U numbers modified to match what is called out on the ecn. The package will create new U numbers for new parts and randomly assign terminators and sections to parts for old parts. The random assignments are not the proper assignments.

5.4.2 Placetool

Placetool should be executed in the place directory. It produces a placetool.log file which will list any errors. Common errors are missing files, such as a missing pstchip.dat file.

Placetool can be used to modify route files. Unless you want to modify route files make sure the dwirsif locking option is off. When on, the title bar of the window indicates that it is active. The option can be turned off using the status menu item or by modifying the placetool.cmd file.

Placetool uses the place.dat file to save placement information. Without this file, all parts on the NMB would have to be placed again. It is wise to save copies of this file after every change to the placement.

Placetool requires a lot of memory. It may be necessary to delete all other icons when running placetool on a sun under sunview.

5.4.3 Batchdrc

Batchdrc should be run any time the placement is altered. There should be no more errors than are found in the last batchdrc.log file (save it before running a new one). The only allowable errors are those that pertain to power plane shorts in the AUGAT connector region (x coordinate less than about 700). The last batchdrc.log file may contain VSENSE errors. These errors should be fixed by ecns and should not be present on any new NMBs.

Batchdrc should be run the place/drc directory.

5.4.4 Netsched

The net scheduler is a very important tool with a number of functions. It determines the order nodes on a net are routed in (generally the driver first, the loads in nearest order, then the terminator) and does checks on nets to make sure they are scheduled in a way that makes sense. It will also do terminator assignment, via assignment and generate the files necessary to back annotate the U number information into the packager state files and the schematics. Not all of these features are necessary for a respin.

During a respin as outlined in the steps presented in the previous figures, the net scheduler is necessary only to generate the scheduling for nets added during ecns. These nets should already have any necessary vias and terminators since the ECN would have specified these items. The packager state file (pstprtb.dat) should also have been already modified to reflect the U number of the part used in the ECN. The backannotation file for the schematics will have to be used since ged does not allow the user to easily modified section and part information.

There are two master netsched.cmd files in the place directory. Netsched.cmd.noa is used when no terminator assignment is to be done (the normal respin case). Netsched.cmd.ta is to be used when terminators and vias are to be assigned as during a complete remake of the board. Copy the appropriate file to netsched.cmd for use.

The net scheduler uses several state files. These files are very important. If the incorrect or out of date files are used, the net scheduler will completely re-schedule the board. Among the state which will be lost will be the fixed length route information which specifies clock delays and hold time fixes.

The two state files for the scheduler are the schstat.dat file and the schlen.dat file. The first file contains the ordering information for all nets. The order in which nodes of a net are listed in this file is the order in which the net is to be routed.

The second file contains the information for determining how long a node to node segment (a fromto) should be. The distance specification is a pair of numbers representing a distance and a tolerance. In most cases, the distance is "MIN" which means to route the segment in as short a distance as possible. Otherwise a length in mils is given which is the amount of wire which should be on the segment. Sometimes this distance is much longer than the minimum distance. The second parameter specifies how close the router must try to get to the first parameter. The number represents a percent tolerance that the actual wire must fall in with respect to the first distance.

Thus, a wire specified as "MIN 30" must be routed to within 30% of minimum distance, a large amount of tolerance. A wire specified as "10500 0" must be routed as exactly 10.5 inches, with no tolerance. The first case is typical of unimportant wires on the board. The second case is typical of clock nets which must be precisely routed in order to control skew.

Note: It is very important to make sure that the proper schstat.dat and schlen.dat files are used. Copies of these files should be saved to make error recovery easier.

The schstat.dat and schlen.dat files can be hand modified. After the first netsched execution on the new package files, it is necessary to look through the schstat.dat files for the wires that have been changed and make sure they are ordered in the proper manner. The netsched.log file will identify which nets have been changed. Similarly, it is necessary to check the schlen.dat file for the nets that have been altered to check their lengths.

Each time netsched is executed, it creates a netsched.log file which lists the nets and parts that have been modified since the last run. Some of the items which have been modified will only be the result of library changes. Such changes can cause warnings about parts changing or nets changing. The only other changes reported should be the nets which have been modified for ecn purposes. In order to tell the difference between warnings that can safely be ignored and ones that should not be ignored, it is probably wise to run netsched before making any changes and save the log file so that you have a list of what the acceptable errors and warnings are. Even in the case of library changes, a few sample nets should be checked to make sure that no rescheduling (re-ordering) of the net took place.

5.4.5 Clktune

The clktune program is used to automatically modify the schlen.dat file so that the total clock delay is balanced in order to minimize clock skew. The program assumes all clocks should be made to rise at the same time at all clock loads. In cases where this is not correct, the clkadj.dat file can be used to specify offsets from the common rise time. The clktune.cmd file is used to specify wire delay on the board, clock buffer characteristics and the root clock nets.

The old schlen.dat file should be saved and compared to the results of the new clktune produced schlen.dat file. There should be no changes except for the ECN'd nets. If clktune produces too many changes, the causes for the changes must be examined.

The clkadj.dat file is used on the NMB to take into account the extra clock buffering which occurs inside the gate arrays. All the gate array clocks rise early at the gate array pins so that by the time the clock has propagated through the clock tree internal to the gate array, it arrives at approximately the same time as the board clocks at the register clock inputs.

The other group of signals in the clkadj.dat file are the clocks that go to the bank staging registers in the ISE and ISO logic. These clocks each have two loads on them. The clktune program assumes all clocks have only a single load. At the time the NMB was first routed, clktune was confused by multi-load clocks. The offsets in the clkadj.cmd file compensate for the errors that clktune makes with respect to multi-load nets.

Note: Clktune will try to adjust the clocks to the phase generator logic (100E451). The timing for this clock is not the same as the timing for other clocks. The phase generators should be clocked earlier than other clocks and contain a special deskewing register to allow it to scan with the rest of the NMB. As of this writing, the clock to the 100E451 should be a minimum length route. It must not have the extra wire distance that clktune will try to put on it.

5.4.6 Itv

Itv is CONVEX's interactive timing verifier. It is the tool used to insure that a board or gate array meets timing requirements. It is a static timing verifier which means that signals are considered to be stable or changing rather than to have specific logic values.

There are two directories in which NMB timing verification should be performed, the time and time.tlc directories. The only difference in these two directories is the source of the file which specifies the timing delays for the NMB nets. This file is the delay.day file. For the time directory it should be linked to the schdly.dat file produced by the netsched tool in the place directory. The time.tlc directory should use the tlcdly.dat file produced.

Itv's command file is called verifier.cmd. It should not have to be modified. The other file used by itv is the case.dat file. This file specifies the timing of all the inputs to the board plus the timing of some special signals on the board. Since the NMB is verified without the NMCs, the NMC connectors are also considered inputs to the board and must be specified.

The phase generator signals must be specified so that clocks are properly gated. If the phase generator signals are left unspecified, itv will treat the clock signals as a normal logic signal and pass stable and changing values rather than ones and zeroes. Stable is not a valid level for a clock input (itv needs to know where the rising edge is) so all registers receiving such a clock will generate an error.

The time.tlc directory contains a file called norm.case.dat and one called scan.case.dat. Both files can be used as a case.dat file. One specifies normal operation (scan controls are zero), the other file specifies the scan case. Both files should be run by copying them over the case.dat file.

The clock gating of the phase generators should also be checked using the check clock gating mode of itv. When using this mode, remember to use a case file that does not override the phase generator signals as this will affect the clock gating check.

5.4.6.1 Compile for Time

Itv uses a cmpexp.dat which has been compiled for time. (The packager uses one that has been compiled for logic.) The compile for time replaces each of the logical parts (registers, and gates, gate arrays) with a timing model description of the part composed of timing primitives. These primitives are simple bodies which perform a specific check or function. For instance the "SETUP HOLD" primitive checks that an input signal meets specified set up and hold requirements to clock.

Both the time and time.tlc directories can and should use the same compile for time.

5.4.6.2 Gate Array Timing Models

Timing models for standard parts are found in the libraries for these parts. Timing models for gate arrays are generated by the gate array designer. The timing values for these models are generated using itv runs on the gate array's compile for time. The gate array is then characterized using a program called iocheck and a simple model written for the gate array which is used for board level timing.

For reasons which will not be discussed here, the gate array timing uses typical to maximum delay timing numbers rather than the minimum-maximum delay numbers that are desired for board level timing. These typical-maximum numbers must therefore be converted to minimum-maximum numbers for board level timing.

The gate arrays models are written with the typical-maximum numbers extracted from the gate array timing simulation. Then, in the time/minlib directory, versions of these models are created which have the timing numbers scaled to approximate minimum-maximum numbers.

Every time the gate array timing model changes, a "make" should be executed in the time/minlib directory. This will update the timing models for the next compile for time.

5.4.7 Dwirsif

The dwirsif cad tool should be executed in the route directory. This tool can be used in two basic modes. In the simplest mode, it reads the packager and netsched files and creates the fromto files necessary for routing. These files must be concatenated into a smaller set of files. This is done in the dwi script in the route directory which also does some other file intializations and saves a copy of the initial files for later reference. This mode is a complete restart of the routing process and is not necessary for a respin. It is necessary to always group the fromto files created by dwirsif into the files called out in the dwi script.

A fromto is a pair of points specifying a wire segment which must be wired from point X to point Y.

In the second dwirsif mode, the previous route files are specified on the command line (see the cadman page for details). Dwirsif then compares the routed wires to the design as specified in the package and netsched files. Those wires that have changed since the design was routed are removed. Those wires that have not changed are left untouched.

Note: Dwirsif should be run in the second, incremental mode for most respins where only minor changes have been done to the board. The process of a complete re-route is a lengthy and time consuming one. Each full routes takes about two and a half days of C2 CPU time.

Note: Dwirsif splits the fromto files into more groups than are necessary to route the NMB. See the dwi script to see how these files should grouped into a smaller, more manageable set.

Note: The hole schedule file produced by dwirsif is not completely accurate for the NMB. See the dwi script to see how the hole schedule file should be modified to produce the proper file.

The fromtos are split into three basic groups: top, bottom and either fromtos. This split is a result of the use of blind vias on the NMB board. Most vias only go to two routing layers, either the top to or bottom two. Some vias, called through vias, go to all four routing layers. Those fromtos which go from top side to top side blind vias must be routed on the top layers and are placed in the top fromto files. Similarly bottom only fromtos go into the bottom files. Those fromtos going between through vias and can be routed on any of the layers since these vias contact all layers. They are placed in the either files.

5.4.8 Conroute

Conroute should also be executed in the route directory. It takes the fromto files produced by dwirsif and tries to route these fromtos. Conroute uses a command file called conroute.cmd which can contain a large number of parameters. Consult the cadman page on conroute for details. (This man page is a fairly detailed one.)

It is generally impossible to route a board in a single pass. The NMB was routed with a large collection of route files. The script called "route" contains the list of files used to route the NMB. The basic procedures applied in these files are described in a later section.

In the case of an incremental route, it is not necessary to go through the full route procedure. In this case a short sequence of conroute passes should route all the wires. Because of the nature of conroute, fixed length routes must be routed in a separate pass from non-fixed length routes. Table 5-3 lists the command files and order in which these files should be run.

Table 5-3 Incremental Route Steps

Route File	Purpose
cmds.fixes/fixedbot.cmd	Route all fixed length routes on bottom side.
cmds.fixes/fixedtop.cmd	Route all fixed length routes on top side.
cmds.fixes/lastbot.cmd	Route remaining fromtos on bottom side.
cmds.fixes/lasttop.cmd	Route remaining fromtos on top side.
cmds.fixes/lasteit.cmd	Route remaining fromtos which can go on either side.

If any bombs occur at any of these steps, additional command files should be tried immediately until no bombs are reported. Consult the conroute man page for ways to route bombs. Typically, the EXTRA_DIST command is used to allow the router more space in which to route a wire. If nothing can be done to route a wire, see the section on Bombs below.

Normally, there are no fixed length route wires in the either fromto files. If there are, these wires need their own pass of the conroute program to be routed.

5.4.8.1 Full Route

The full route follows several basic steps. First, all the fixed length routes are attempted with diagonal routing turned off (the routes work better this way). Then, any routes that bombed are retried with diagonal routing turned on.

The remainder of the route follows four basic approaches. First, the most difficult area of the board is routed before less difficult areas. Second, strict orthogonal and diagonal routes are tried before the routes that deviate from a straight 90 or 45 degree route. Third, bottom and top fromtos are routed before either fromtos. Finally, orthogonal and diagonal routes are forced to separate layers.

The first three techniques basically follow the principle of routing the most difficult wires before the less difficult wires. Congested areas are routed before less congested areas. In some cases, wire keepouts are used to prevent wires from crossing a congested area when they do not need to do so. Orthogonal fromtos and strict diagonal fromtos are difficult routes because there are fewer ways to route these wires: for two points with the same x or y coordinate, the shortest path is a straight line between the two. Any other path, is greater than minimal distance and runs the risk of violating the route length specified by the scheduler length file. Either fromtos are easier than top or bottom fromtos since they have twice as many layers on which they can be routed.

By routing orthogonal wires on one layer and diagonal wires on another layer, interference between the two types of routes is minimize. The two can interfere because they have their wire cross over points on a different grid and no more than two wires can cross over or near each other.

When the NMB was routed for wire rev A, the "route" script in the route directory was used to route the board. With this script there were still a few bombs which were fixed using the technique described in the next section.

5.4.8.2 Bombs

Should a small number of bombs occur in the route process, it is generally possible to fix these bombs by hand. The technique used is to use placetool to examine the routed wires looking for congestion near the bombed fromto. Generally, either one of the vias on the fromto is boxed in by routes or there is a wire in a routing channel blocking the path the fromto could most easily route in. The offending wire can then be deleted and the route retried. Almost always, the bomb can now route and the deleted wire can be re-routed in some other place.

When deleting a wire, be careful not to delete fixed length routes since these are more difficult to re-route once the board is mostly routed.

See the placetool cad man page for how to edit route files using placetool.

5.4.9 Checking the Route

Routed wires should be checked in the ship/wires directory. The script do_checks will execute the check_dwirsif program which checks the route files for various errors. Some of these errors may be safely ignored, some can not be. The simplest procedure is to save the old check log files and compare new runs to these files. No differences should be noted.

There are two classes of errors which can be safely ignored. The wires between the AUGAT pads and the vias for the AUGAT violate some multiwire routing rules. These violations can be tolerated in this area only. The other errors should all result from the TDR traces which run along the

periphery of the board. These wires run too close to some of the holes for the NMC hold down hardware. However, these wires are only used before the hardware is in place so the error can be tolerated.

5.4.10 Tlc

Tlc is the transmission line calculator. It reads the route files to determine the actual routes on the NMB and performs a transmission line analysis given the particular drivers and loads on the net. The results of the analysis is then turned into a delay.dat file called tlcdly.dat suitable for itv analysis.

The do_tlc script in the tlc directory is used to execute tlc. Tlc is licensed software and must be run on a licensed machine. Errors are logged in tlc.log.

After tlc has been run successfully, the tlc results are translated into delay.day format by executing tlc_if. It should produce a file called tlcdly.dat. This file may now be used in an itv timing verification as described in the itv section.

5.5 Trouble Shooting

The most common way to get into trouble when respinning a board is to use the incorrect files.

Many of the tools discussed previously require state files. These state files describe the NMB. When they are omitted or incorrect, large discrepancies in the design will result. State files should generally be saved after every modification to make backtracking easier.

As discussed earlier, the NMB design directories contain many file links. With the tar tape archive facility, these links can become real files. A link left as a file will not be properly updated and will cause stale data to be used by one of the cad tools. As with bad state files, this will lead to discrepancies. Unlike problems which are the result of missing state files, these errors may be much more subtle. It is very important to check for links as described in Section 5.3.1 on page 115.

The cad tools clearly report any errors they encounter. Unfortunately, some of these errors are acceptable and can be safely ignored while others are very serious. The safest way to determine which errors are acceptable is to compare error logs on the design tape with the ones generated during respin. All differences should be the result of the ecns being incorporated into the board.

Particular attention must be paid to the errors and warnings produced by the netsched, dwirsif and conroute programs. Discrepancies in the database for the NMB will show up in these tools by generating large numbers of errors or warnings. In a board respin, most nets should remain unchanged. There should only be a few differences between the original and new wire revision.

Subtle errors can be introduced into the design if care is not taken with respect to fixed lengths routes. These are routes that are specified to be of a certain length. The fixed length route is used to fix a hold problem by adding extra distance and therefore propagation delay to the wire or it is used to properly line up the clocks on the board. Errors in fixed length routes should appear in the timing verification but it may be unclear as to why the error has appeared.

The clock generator logic has special timing requirements as described in Section 3.5.1.3 on page 90 of the Functional Description chapter. It is very important that the wire delays in this logic be checked both using itv's check clock gating mode and by hand. The precise timing shown in Figure 3-21 on page 91 must be met. In order to meet this timing, the 100E451 in the phase generator

circuit must be clocked well in advanced of the other clocks on the NMB so that it can gate the next clock pulse.

5.5.1 Routing Issues

The NMB route took a great deal of time and effort. The recipe used to route the board is in the script named "route" as has already been mentioned. This script should prove sufficient to do a complete re-route of the board should that prove necessary.

The NMB was routed before the conroute tool was able to use the density routing algorithm. This technique may prove useful on the NMB but it has never been tried.

The primary issue in routing the NMB is routing into the connector side of the board. The worst bottleneck occurs at the boundary between the "0O" bank and the BCGAs. This region is very congested and must be routed first.

Routing through the NMC connectors has been simplified by using a via pattern for the connectors that allows 150 mil spacing between vias in the direction that most of the wires flow. Most of the PLCC28 packages between the connectors are 100141 (or 100341) devices. A special via pattern was used for these devices which also provides 150 mil paths in the route direction. This via pattern is specified by the ALT_JEDEC_TYPE property in the NMB schematics for these parts. The property should be set to PLCC28_100KQ.

For optimum route, the 150 mil gaps in the PLC28_100KQ packages must be aligned with the gaps in the NMC connector packages. All other vias, such as feedthru vias and those on terminators, should be placed to minimize obstructions in these 150 mil gaps. This alignment is less critical as the NMCs move away from the BCGAs.

5.6 Respinning the NMC

NMC schematic capture was done on ged using valid tools as it was for the NMB. Placement and route was done on the Allegro PCB cad tool package. This package requires only the standard packager files as input. As output, it produces all the artwork for the board plus back annotation files to update the schematics and packager files with U number information.

A CAD engineer should be consulted when using the PCB tool package.

5.6.1 Placement

Space is very tight on the NMC card. Space for both placement and routing were the primary restrictions on the NMC cards. Timing issues with respect to wire lengths were not critical and were generally ignored since most wires were routed to within 10% of minimum distances.

Parts were placed so that as many vias as possible could be shared. The DRAMs have common data, address and some control lines. DRAMs placed back to back shared these vias. Similarly, the translators which drove the NMC data bus could share the TTL pin. ECL to TTL and TTL to ECL translators were placed back to back to maximize the number of data bus vias which could be shared.

Via sharing was also done for terminator and connector pins. In some cases, the translator pin for a signal was adjacent to the connector pin for that signal. Where possible, these pins were shared.

When terminator pins could be placed adjacent to translator pins for the same load, these too were shared.

There are several placement keep out areas marked on the board. These are to allow room for the hold down logic. No parts can be placed in these regions.

Placement was done completely by hand for both parts and vias.

5.6.2 Routing

All routing on the NMC was by hand. The primary issue of concern for routing was crosstalk, both between signals of the same logic family and particularly TTL to ECL crosstalk.

Terry Morris ran some Greenfields analysis for the NMC team to determine some basic design rules for crosstalk. As a rule of thumb, signals of the same logic family could run parallel for up to six inches at minimum spacing without exceeding noise margins. Since the NMB is only 8.3 inches long, this rule was not a real constraint.

For TTL to ECL crosstalk, the restrictions were much tighter. A TTL signal could not run over an ECL signal for more than 100 mils. It could not be above and over by 6 mils for more than 250 mils. It could not parallel on the same layer for more than 500 mils.

To minimize crosstalk, ECL and TTL signals were kept on separate layers as much as possible. In the region of the WREDC and DRAMs there are no ECL signals so TTL signals could use all four routing layers. In the region of the translators, both signals could be found. Here, one pair of routing layers was designated for TTL and one pair for ECL. Unfortunately, it was impossible to route all the ECL signals on the ECL layers and vice versa. In the cases were TTL and ECL signals shared the same layers, the signals were hand checked to make sure they did not violate any design rules.

5.6.3 Power Planes

The NMC receives three supply levels, VCC, VEE, and VTT, plus ground and has four power planes for distributing this power.

Because of the potential for TTL switching to cause noise on ECL parts, two ground planes were used, one for TTL and one for ECL. The two planes are joined by the vias at the connector. The TTL ground spans the entire NMC. The ECL ground is really a split plane. It is an ECL ground beneath the translators and a TTL ground underneath the WREDC and DRAMs. The two regions are not completely separate. There is a small neck of copper joining the two near the WREDC.

The TTL power plane (VCC) is a solid power plane spanning the entire NMC. The ECL power plane (VEE, VTT) is split three ways. Beneath the DRAMs and WREDC, the plane is a VCC supply plane. It is only tied to the primary VCC plane through the power vias in that region of the board. It has no direct connection to the VCC vias at the connector so it is primarily an impedance control plane. The other half of the ECL power plane is split between VEE and VTT.

Note: Since the power planes are split, caution must be taken that VTT pins are not hooked to the VEE layer, etc. Under the Allegro tools, a clearance layer is created for each split power plane which contains the clearances to prevent pins from being hooked to improper planes. Adding these clearance to the power plane artwork is a manual step that can not be omitted.

Appendix A Signal List

'(NMB 2NMB_CLOCK1P)ALL_141_CTL'<1:0>

The control signals for the 100141s and 100E141s. Note that the order of the control bits for the 100E141s (but not the 100141s) is reversed with respect to the 100141s. ALL_141_CTL<1> is wired to control bit 0 of the E141s and ALL_141_CTL<0> is wired to bit 1 of the E141s.

Table A-1 ALL_141_CTL

Control Value	Action
00	Load, active during scan mode 0,1,3, and 6
01	Unused (shift msb to lsb)
10	Shift lsb to msb, used during scan mode 7
11	Hold, used during scan_mode 6

'ALL_141_CTL.A'<1:0>
'ALL_141_CTL.B'<1:0>
'ALL_141_CTL.C'<1:0>
'ALL_141_CTL.D'<1:0>
'ALL_141_CTL.E'<1:0>
'ALL_141_CTL.F'<1:0>

Buffered copies of '(NMB 2NMB_CLOCK1P)ALL_141_CTL'<1:0>. Generated in the NMB_CLOCK logic used mostly in ISE/ISO logic.

'(NMB 2NMB_CLOCK1P)ALL_DMODE*'
'ALL_DMODE.0'
'ALL_DMODE.0*'
'ALL_DMODE.1'

All of the above are true when the most significant bit of XC_MB.SCAN_CTL is set. This bit is only set during the various "diagnostics" modes: NMC clock initialization, CAST Load and ALL Scan,

'(NMB 2NMB_CLOCK1P)ALL_EN*'
'(NMB 2NMB_CLOCK1P)ALL_EN.2*'

Gating signal for the second level, 1x clock drivers. These signals are buffered copies of PHASE_GEN_1X<0> at all times except when scan control is in mode seven (ALL scan). These are the signals that mask every other clock pulse of the 2x clock that the NMB receives to generate the 1x signals used by most of the logic on the board.

NOTE: for each of the bank zero, even side signals below beginning with "B0E" there is a corresponding signal for each of the other thirty-one banks. Banks are numbered from "0" to "F" (zero to fifteen in hexadecimal) with a trailing "E" for even side banks and a trailing "O" for odd side banks. For instance, B3E_CAS* corresponds to the CAS signal for bank three of the even side. BDO_CAS* is the CAS signal for bank thirteen (hexadecimal "D"), odd side.

'B0E_BANK_WR_DAT'<31:0>

Write Data from the write data staging register in ISE to the number zero, even side NMC. This data is a registered copy of the input staging data from the crossbar. The data is held in this register when B0E_WR_REG_HOLD is asserted.

'B0E_BANK_WR_PAR'<3:0>

Similar to B0E_BANK_WR_DAT'<31:0> above, this is the parity for the staged write data. This parity is also held when B0E_WR_REG_HOLD is asserted. Table A-2 shows the correspondence between parity bytes and data.

Table A-2 WR_PAR

Parity Signal	Corresponding Data
B0E_BANK_WR_PAR<3>	B0E_BANK_WR_DAT<7:0>
B0E_BANK_WR_PAR<2>	B0E_BANK_WR_DAT<15:8>
B0E_BANK_WR_PAR<1>	B0E_BANK_WR_DAT<23:16>
B0E_BANK_WR_PAR<0>	B0E_BANK_WR_DAT<31:24>

'B0E_CAS*'

Column address for the bank zero, even side DRAMs. This signal comes from bank control gate array BC0E and goes to NMC0E.

'B0E_CHECK'

CHECK originates at BC0E, destination is MIM_CLK where it is used in a state machine to generate B0E_MIM_CLK. CHECK also goes to NMC0E where it becomes a signal called ERR_CLK in the NMC schematics. As ERR_CLK, it causes non-scannable error status state in the WREDC to be registered.

'B0E_CHECK_LE'

Input data clock for the WREDC for the bank zero, even side NMC. On the rising edge of this signal, data is registered in the WREDC. This data is either DRAM data or B0E_BANK_WR_DAT<31:0> depending on ECC_CHECK. Originates at BC0E, destination is NMC0E.

'(NMB 4ISE1P)B0E_CTL'<2:1>

Generated within the ISE logic. This signal controls the ISE 100141s used to stage data, parity, and zone for NMC0E. It is a function of B0E_WR_REG_HOLD, ALL_DMODE and ALL_141_CTL as shown in Table A-3

Table A-3 B0E_CTL

ALL_DMODE	ALL_141_CTL	WR_REG_HOLD	B0E_CTL	Action
0	X	0	0	Load
0	X	1	3	Hold
1	0	X	0	Load
1	1	X	1	Shift right, lsb to msb
1	2	X	2	Unused
1	3	X	3	Hold

'B0E_DATA_OUT'<31:0>

Return data from NMC0E to the READMUXE (even side return data multiplexor). See also B0E_DATA_STR.

'B0E_DATA_SEL'

WREDC data select signal from BC0E to NMC0E. This signal determines whether corrected or DRAM data is loaded into the WREDC on NMC0E. A "1" selects corrected data. This signal is only asserted during a correct cycle after a soft error has been detected during a partial write or test-and-modify operation.

'B0E_DATA_STR'

From BC0E to NMC0E. DATA_STR is the latch enable for the latches in the TTL to ECL translators on the NMC that drive B0E_DATA_OUT<31:0> and B0E_ECC_OUT<7:0>. The latches are transparent when DATA_STR is "0".

'B0E_ECC_CHECK'

From BC0E to NMC0E. ECC_CHECK determines whether the WREDC is checking parity, as at the beginning of a memory operation, or is checking ECC as during the second portion of a partial write or test-and-modify. ECC_CHECK is asserted for the latter case. ECC_CHECK also determines whether the ECL to TTL translators are driving the NMC data bus or whether the DRAMs or WREDC is driving that bus.

'B0E_ECC_OUT'<7:0>

Return ECC from NMC0E to the READMUXE (even side return data multiplexor). See also B0E_DATA_STR. ECC is the error detect and correct code for B0E_DATA_OUT<31:0>

'B0E_G*'

Output enable for the NMC0E DRAMs generated by BC0E. Active low.

'B0E_MIM_CLK'

Error logging clock to the NMC0E WREDC from MIM_CLK. This clock is either a registered version of B0E_CHECK during non-log scan mode operation or a special scan clock operating at 1/8 the normal system clock rate during any scan mode.

'B0E_MIM_ERROR'

WREDC error status signal from NMC0E to BC0E. The type of error detected depends on the value of B0E_ECC_CHECK and B0E_DATA_STR at the time of the error. It can mean a parity error has occurred, a single bit error has occurred or a double bit error has occurred.

'B0E_RAM_ADDR'<9:0>

Address for the NMC0E DRAMs from BC0E. RAM_ADDRESS is used for the row address when RAS* is asserted, the column address when CAS* is asserted and the refresh address (which is RAS* asserted when CAS* is not asserted for the cycle).

'B0E_RAS*'

Row address strobe for the NMC0E DRAMs, generated by BC0E. RAS* is decoded along with B0E_ROW_ADDRESS<1:0> and B0E_RFSH_ACT to generate the RAS* signals for each of the four rows of DRAMs on the NMC.

'B0E_RFSH_ACT'

Indicates the current cycle is a refresh cycle and all DRAMs are to be active. Generated by BC0E. RFSH_ACT is decoded along with B0E_ROW_ADDRESS<1:0> and RAS* to generate the RAS* signals for each of the four rows of DRAMs on the NMC.

'B0E_ROW_ADDR'<1:0>

Selects which of the four rows on an NMC are to be active when RAS* is asserted if RFSH_ACT is "0". RFSH_ACT is decoded along with B0E_ROW_ADDRESS<1:0> and RAS* to generate the RAS* signals for each of the four rows of DRAMs on the NMC.

'B0E_SCAN_OUT'

Scan output from the scannable state in the NMC0E WREDC to MIM_CTL.

'B0E_WE*'

Write enable, active low to the NMC0E DRAMs from BC0E.

'B0E_WR_REG_HOLD'

Holds the NMC0E staged data, zone and parity in ISE when asserted. Generated by BC0E.

'B0E_WR_SEL'

Generated by the BC0E to the NMC0E WREDC. Determines whether the B0E_ZONE bits control loading of data word into WREDC (WR_SEL equal 1) or whether all bytes are loaded regardless of ZONE (WR_SEL equal 0).

'B0E_ZONE'<3:0>

From bank holding registers in ISE to NMC0E WREDC. When B0E_WR_SEL is one, the ZONE bits determine loading of data into the WREDC. When data is being read from the DRAMs to be merged with the write data on a partial write or test-and-modify, DRAM data overwrites write data if the corresponding ZONE bit is zero. Correspondence between ZONE bit and data bytes is identical to parity as in Table A-2.

'BP_MB.LBD_INTL_IN'

This signal is used as a logic board interlock signal. The power pallet uses the signal to make sure the board is plugged into the backplane before power is applied.

'BP_MB.PORTID'<3:0>

These signals allow the power pallet controller to determine which port of the crossbar the board is connected to.

'BP_MB.SLOTID'<3:0>

These signals allow the power pallet controller to determine which slot on the backplane the board is plugged into.

'CLKLEDCLR'

Clear signal from power pallet which resets the scan control and clock test logic.

'CLKLEDON'

Output from the scan control and clock test logic.

'COP_GND'

Ground (power) for the COP chip on the NMB.

'COP_VCC'

Power for the COP chip on the NMB.

'CU_MB.CLOCK_2X'

'CU_MB.CLOCK_2X*'

Active high and low (complementary) master clock to the NMB from the NCU board. Clock is at twice the basic system frequency.

'(NMB 13MIM_CTL3P)ENABLE03'

'(NMB 13MIM_CTL3P)ENABLE47'

These enable signals are internal to the MIM_CTL logic. ENABLE03 gates the MIM_CLKs to NMCs, 0, 1, 2, 3, 8, 9, 10, and 11, even and odd. ENABLE47 gates the remaining NMCs. The ENABLEs were split into two groups for a reason that is no longer valid; the signals are currently identical.

'(NMB 4ISE1P)ERR_E0'

Signal internal to ISE logic. Asserted if there is a parity error on input staging from crossbar between DATA<7:0> and PARITY<3>.

'(NMB 4ISE1P)ERR_E1'

Signal internal to ISE logic. Asserted if there is a parity error on input staging from crossbar between DATA<15:8> and PARITY<2>.

'(NMB 4ISE1P)ERR_E2'

Signal internal to ISE logic. Asserted if there is a parity error on input staging from crossbar between DATA<23:16> and PARITY<1>.

'(NMB 4ISE1P)ERR_E3'

Signal internal to ISE logic. Asserted if there is a parity error on input staging from crossbar between DATA<31:24> and PARITY<0>.

'(NMB 4ISE1P)ERR_E4'

Signal internal to ISE logic. Asserted if there is a parity error on input staging from crossbar between ZONE<3:0> and CTL_PAR<4>.

'(NMB 4ISE1P)ERR_E5'

Signal internal to ISE logic. Asserted if there is a parity error on input staging from crossbar between ADDRESS<28:22> and CTL_PAR<0>.

'(NMB 3ISE1P)ERR_O0'

Signal internal to ISO logic. Asserted if there is a parity error on input staging from crossbar between DATA<7:0> and PARITY<3>.

'(NMB 3ISE1P)ERR_O1'

Signal internal to ISO logic. Asserted if there is a parity error on input staging from crossbar between DATA<15:8> and PARITY<2>.

'(NMB 3ISE1P)ERR_O2'

Signal internal to ISO logic. Asserted if there is a parity error on input staging from crossbar between DATA<23:16> and PARITY<1>.

'(NMB 3ISE1P)ERR_O3'

Signal internal to ISO logic. Asserted if there is a parity error on input staging from crossbar between DATA<31:24> and PARITY<0>.

'(NMB 3ISE1P)ERR_O4'

Signal internal to ISO logic. Asserted if there is a parity error on input staging from crossbar between ZONE<3:0> and CTL_PAR<4>.

'(NMB 3ISE1P)ERR_O5'

Signal internal to ISO logic. Asserted if there is a parity error on input staging from crossbar between ADDRESS<28:22> and CTL_PAR<0>.

'FREE_CK'<116:0>

'FREE_CK*'<116:0>

Generated by NMB_CLOCK. These are the 1X system frequency clocks. They are not affected by SYS or LOG scan. Only SCAN_CTL modes 6, and 7 can alter these clocks.

'FREE_CK2'<8:0>

'FREE_CK2*'<8:0>

Generated by NMB_CLOCK. These are the 2x system frequency clocks. They are not affected by SYS or LOG scan. Only SCAN_CTL modes 5, 6, and 7 can alter these clocks. Clocks 0 to 7 go to the eight BCGAs.

'FREE_CK_HD'<8:0>

'FREE_CK_HD*'<8:0>

These clocks are similar to FREE_CK except that they are held at one when SCAN_CTL is in mode 4 or 5 (NMC CLOCK INIT).

'HARD_ERROR_E'<4:0>

Even side hard error signals. The sources for each of these signals are given in Table A-4. All signals go to NMB_CLOCK where they are OR'd together and

registered to become MB_XC.HARD_ERROR

Table A-4 HARD_ERROR_E

Bit	Source
0	BC0E
1	BC1E
2	BC2E
3	BC3E
4	ISE

'HARD_ERROR_O'<4:0>

Hard error signals for the odd side logic. Sources are similar to HARD_ERROR_E sources.

'HARD_ERR_ENABLE'

From the NMB_CLOCK to the BCGAs. This signal is set via scan of the NMB. When unasserted, it inhibits all hard errors generated by the BCGAs. It does not effect hard errors generated by the ISE or ISO logic (these are disabled by NMB_CLOCK_SCAN_OUT). It also does not affect the XBAR_ERROR signals generated by the BCGAs.

'HOLD_MODE'

Generated by NMB_CLOCK. It is asserted if SCAN_CTL equals 4 or 5, NMC CLOCK INIT mode.

'ISE_SCAN_OUT'

The scan output for the "all" state in ISE. Destination is NMB_CLOCK.

'ISE_SYS_SCAN_OUT'

The scan output for the "sys" state in ISE. Destination is NMB_CLOCK.

'ISO_SCAN_OUT'

The scan output for the "all" state in ISO. Destination is ISE.

'ISO_SYS_SCAN_OUT'

The scan output for the "sys" state in ISO. Destination is ISE.

'(NMB 4ISE1P)IS_CTL_E'<1:0>

Input staging register controls for ISE. IS_CTL is the control signal for the 100E241s used as input staging registers. It is generated internally to ISE. If a hard error is detected by ISE, it holds the input staging registers unless the NMB is in scan mode.

IS_CTL<1> is true if the state is to be held.

IS_CTL<0> is true if the state is to be scanned. IS_CTL<1> overrides IS_CTL<0>.

If both are zero, the registers do a parallel load.

'(NMB 3ISO1P)IS_CTL_O'<1:0>

Similar to IS_CTL_E<1:0> but for ISO logic.

'(NMB 2NMB_CLOCK1P)IS_SCAN_IN.3'

'(NMB 2NMB_CLOCK1P)IS_SCAN_IN.6'

These two signals are internal to NMB_CLOCK logic. The scan input from the XCL is shifted in to an eight bit shift register. The IS_SCAN_IN signals are different taps off of the scan register. The taps are chosen so that there is a multiple of eight bits in the ALL and SYS rings.

'LBD_OVT1'
'LBD_OVT1_GND'
'LBD_OVT2'
'LBD_OVT2_GND'
'LBD_OVT3'
'LBD_OVT3_GND'

Power pallet signals used to supply ground and monitor the output voltage of the temperature sensors on the board.

'LD_EN_A_E'
'LD_EN_A_O'
'LD_EN_B_E'
'LD_EN_B_O'
'LD_EN_C_E'
'LD_EN_C_O'
'LD_EN_D_E'
'LD_EN_D_O'

These eight signals are generated by each of the eight BCGAs. A BCGA asserts its LD_EN when it has a return on the next clock. Only one even and one odd LD_EN can be asserted per clock. All signals go to LDREG.

Table A-5 shows the source BCGA for each of the LD_EN signals.

Table A-5 LD_EN

Signal	Source
LD_EN_A_E	BC0E
LD_EN_B_E	BC1E
LD_EN_C_E	BC2E
LD_EN_D_E	BC3E
LD_EN_A_O	BC0O
LD_EN_B_O	BC1O
LD_EN_C_O	BC2O
LD_EN_D_O	BC3O

'MB_BP.LBD_INTL_OUT'

This signal is used as a logic board interlock signal. The power pallet uses the signal to make sure the board is plugged into the backplane before power is applied.

'MB_XC.HARD_ERROR'

Hard error (non-recoverable error) signal to the XCL board. Generated in NMB_CLOCK.

'MB_XC.SCAN_OUT'

Scan output for all types of scan operations. From the RDEDC gate array to the XCL board.

'MB_XC.SOFT_ERROR'

Soft error (recoverable error) signal to the XCL board from the NMB_CLOCK logic.

'MB_XRE.RD_DATA'<31:0>

Even side return data from RDEDC to the XRTE.

'MB_XRE.RD_PAR'<3:0>

Even side return data parity from the RDEDC to the XRTE. Parity is valid even when RD_RDY is inactive. Parity mapping is as in Table A-2.

'MB_XRE.RD_RDY'

Indicates the NMB has valid return data for the XRTE. Generated by the RDEDC. RD_RDY is not functionally necessary; the XRTE knows when data should be returning. However, the XRTE will generate a hard error if RD_RDY is asserted when data is not expected or vice versa.

'MB_XRO.RD_DATA'<31:0>

Odd side return data from the RDEDC to XRTO.

'MB_XRO.RD_PAR'<3:0>

Odd side return data parity from the RDEDC to the XRTO. Parity is valid even when RD_RDY is inactive. Parity mapping is as in Table A-2.

'MB_XRO.RD_RDY'

Indicates the NMB has valid return data for the XRTO. Generated by the RDEDC. RD_RDY is not functionally necessary; the XRTO knows when data should be returning. However, the XRTO will generate a hard error if RD_RDY is asserted when data is not expected or vice versa.

'MB_XS0E.SEND_PAR_ERR'
'MB_XS0O.SEND_PAR_ERR'
'MB_XS1E.SEND_PAR_ERR'
'MB_XS1O.SEND_PAR_ERR'

Each of these four signals indicates that the NMB has detected a parity error in the data received two clocks ago from the XSE or XSO. The signals are generated by NMB_CLOCK and go to the board indicated in the signal prefix. Only even side errors cause the XS0E and XS1E signals to be asserted. Only odd side errors cause the XS0O and XS1O signals to be asserted.

'MB_XSE.BANK_DONE'<15:0>

The sixteen bank done bits correspond to each of the sixteen banks on the even side of the NMB. They are asserted for one clock each when a bank completes a request. The XS0E board is the destination of these signals. They are sourced by the BCGA controlling the given bank: BC0E sources bits 3:0, BC1E sources 7:4, BC2E sources 11:8, and BC3e sources 15:12.

'MB_XSO.BANK_DONE'<15:0>

The sixteen bank done bits correspond to each of the sixteen banks on the odd side of the NMB. They are asserted for one clock each when a bank completes a request. The XS0O board is the destination of these signals. They are sourced similarly to MB_XSE.BANK_DONE<15:0>.

'MIMO_CMOS_SCAN_IN.0'
'MIMO_CMOS_SCAN_IN.1'
'MIMO_CMOS_SCAN_IN.2'
'MIMO_CMOS_SCAN_IN.3'
'MIMO_CMOS_SCAN_IN.4'
'MIMO_CMOS_SCAN_IN.5'
'MIMO_CMOS_SCAN_IN.6'
'MIMO_CMOS_SCAN_IN.7'

These are the scan inputs for the first eight NMCs, NMC0O to NMC7O. Source is MIM_CTL.

'MIM_CMOS_SCAN.A'
'MIM_CMOS_SCAN.B'
'MIM_CMOS_SCAN.C'
'MIM_CMOS_SCAN.D'
'MIM_CMOS_SCAN.E'
'MIM_CMOS_SCAN.F'
'MIM_CMOS_SCAN.G'
'MIM_CMOS_SCAN.H'

These eight signals are true if the NMCs are to be scanned (scan control modes 1, 3, and 7). Table A-6 shows which NMCs each of the MIM_CMOS_SCAN signals go to.

Table A-6 MIM_CMOS_SCAN

Signal	NMCs
A	0O, 1O, 2O, 3O
B	8O, 9O, AO, BO
C	0E, 1E, 2E, 3E
D	8E, 9E, AE, BE
E	4O, 5O, 6O, 7O
F	CO, DO, EO, FO
G	4E, 5E, 6E, 7E
H	CE, DE, EE, FE

'MIM_CMOS_SCAN_IN'

The scan input to the group of NMCs as a whole. Sourced by NMB_CLOCK. Destination is MIM_CTL.

'MIM_CTL_SCAN_OUT'

Scan output for the "all" state in MIM_CTL. Destination is ISO.

'MIM_SCAN_OUT'

Scan output from the NMCs as a group. Destination is BC0O.

'M_CST.CERR_E'
'M_CST.CERR_O'

These are two signals specifically for the CAST tester. They are asserted with RD_RDY_E or RD_RDY_O if there is any error, single or multi-bit, for the request on the given clock. The signals are sourced by RDEDC. They have no destination except on the CAST backplane.

'NMB_ALL_SCAN_OUT'

Scan output for the NMB_CLOCK logic "all" state. Destination is RDEDC.

'NMB_CLOCK_SCAN_OUT'

Scan output for the NMB_CLOCK logic "sys" state. Destination is MIM_CTL. This signal is used as an active low hard error enable for the input staging registers in the ISE and ISO logic.

'NMC_BYPASS'

This signal is sourced from the NMB connector to MIM_CTL. When asserted, all NMCs are excluded from all NMB scan rings. The NMCs must still be inserted into the NMB for proper scan operation because they are required for some signal terminations.

'PHASE_CTL'

Asserted when scan control is equal to 7 (all scan). Generated by NMB_CLOCK.

It is used both by NMB_CLOCK logic and by the Heart beat logic.

'(NMB 2NMB_CLOCK1P)PHASE_GEN_1X'<3:0>

Phase generation logic signals. Bit zero is used to gate the 2X clocks that are to become 1X clocks.

'PPC_SPI_CLK'

Clock signal from the power pallet controller for the serial EEPROM.

'PPC_SPI_IN'

Scan out signal from the serial EEPROM.

'PPC_SPI_OUT'

Scan in signal to the serial EEPROM.

'PPC_SPI_SELECT'

Chip select signal from the power pallet controller for the serial EEPROM.

'(NMB 13MIM_CTL3P)QB0E_HS'
'(NMB 13MIM_CTL3P)QB0O_HS'
'(NMB 13MIM_CTL3P)QB1E_HS'
'(NMB 13MIM_CTL3P)QB1O_HS'
'(NMB 13MIM_CTL3P)QB2E_HS'
'(NMB 13MIM_CTL3P)QB2O_HS'
'(NMB 13MIM_CTL3P)QB3E_HS'
'(NMB 13MIM_CTL3P)QB3O_HS'
'(NMB 13MIM_CTL3P)QB4E_HS'
'(NMB 13MIM_CTL3P)QB4O_HS'
'(NMB 13MIM_CTL3P)QB5E_HS'
'(NMB 13MIM_CTL3P)QB5O_HS'
'(NMB 13MIM_CTL3P)QB6E_HS'
'(NMB 13MIM_CTL3P)QB6O_HS'
'(NMB 13MIM_CTL3P)QB7E_HS'
'(NMB 13MIM_CTL3P)QB7O_HS'
'(NMB 13MIM_CTL3P)QB8E_HS'
'(NMB 13MIM_CTL3P)QB8O_HS'
'(NMB 13MIM_CTL3P)QB9E_HS'
'(NMB 13MIM_CTL3P)QB9O_HS'
'(NMB 13MIM_CTL3P)QBAE_HS'
'(NMB 13MIM_CTL3P)QBAO_HS'
'(NMB 13MIM_CTL3P)QBBE_HS'
'(NMB 13MIM_CTL3P)QBBO_HS'
'(NMB 13MIM_CTL3P)QBCE_HS'
'(NMB 13MIM_CTL3P)QBCO_HS'
'(NMB 13MIM_CTL3P)QBDE_HS'
'(NMB 13MIM_CTL3P)QBDO_HS'
'(NMB 13MIM_CTL3P)QBEE_HS'
'(NMB 13MIM_CTL3P)QBEO_HS'
'(NMB 13MIM_CTL3P)QBFE_HS'
'(NMB 13MIM_CTL3P)QBFO_HS'

These thirty two signals are used internally to the MIM_CTL logic. They are used by the MIM_CLK state machines for each bank to make sure that each MIM_CLK

clock lasts at least two clocks so that minimum plus width requirements for the CMOS gate array are not violated.

'(NMB 13MIM_CTL3P)QCOUNT'<2:0>

This signal is used internally to MIM_CTL to select which of the eight groups of four NMCs are to receive the next clock during scan.

'(NMB 13MIM_CTL3P)QFIRST_COUNT'

Used internally to MIM_CTL, this signal is true during the first time each of the eight groups of four NMCs are clocked during scan.

'(NMB 13MIM_CTL3P)QLAST_SCAN'

Internal to MIM_CTL. True if the MIM_CTL clock generator was doing a scan on the previous clock. Currently unused by the state machine.

'QLOG_RUN*'<3:0>

Registered copies of XC_MB.LOG_RUN*. These signals, registered in NMB_CLOCK, are used to gate all log clocks. The signals go to each of the BCGAs and the RDEDC.

'QSYS_RUN*'<3:0>

Registered copies of XC_MB.SYS_RUN*. These signals, registered in NMB_CLOCK, are used to gate all sys clocks. The signals go to each of the BCGAs and the RDEDC. QSYS_RUN*<0> also gates the log clock logic internal to NMB_CLOCK.

'(NMB 13MIM_CTL3P)Q_MCS'<1:0>

State bits of the state machine in the MIM_CTL logic which controls the generation of clocks for the NMCs during scan operations.

'(NMB 3ISO1P)RCTL_PAR'

Input staging bit for the parity bit for ZONE in ISE, CTL_PAR<4>.

'(NMB 4ISE1P)RCTL_PAR'

Input staging bit for the parity bit for ZONE in ISO, CTL_PAR<4>.

'RDEDC_TCLR'

RDEDC test structure clear bit. Undriven by the NMB. Only used during test of the RDEDC by the gate array manufacturer.

'RDEDC_TOUT'

RDEDC test structure output bit. Unused by the NMB. Only used during test of the RDEDC by the gate array manufacturer.

'RDEDC_VREF'

Voltage reference signal for the RDEDC. Generated by logic on NMB top level, page 16.

'(NMB 4ISE1P)RE_ADDR'<28:22>

Input staging register for the address from XSE in the ISE logic block. Only bits 28:22 of the address are registered in ISE.

'RE_REF_REQ'

Registered copy of refresh request from XSE. Sourced from NMB_CLOCK to the even side BCGAs. This refresh request is not used when RUN_SYS* is unasserted.

'(NMB 4ISE1P)RE_ZONE'<3:0>

Input staging register for the ZONE bits from XSE.

'RLD_EN_A_E'

'RLD_EN_A_O'

'RLD_EN_B_E'
'RLD_EN_B_O'
'RLD_EN_C_E'
'RLD_EN_C_O'
'RLD_EN_D_E'
'RLD_EN_D_O'

Sourced from LDREG to RDEDC. Registered copies of the LD_EN signals. See Table A-5 for the source of the LD_EN signals.

'(NMB 3ISO1P)RO_ADDR'<28:22>

Input staging register for the address from XSO in the ISO logic block. Only bits 28:22 of the address are registered in ISO.

'RO_REF_REQ'

Registered copy of refresh request from XSO. Sourced from NMB_CLOCK to the odd side BCGAs. This refresh request is not used when RUN_SYS* is unasserted.

'(NMB 3ISO1P)RO_ZONE'<3:0>

Input staging register for the ZONE bits from XSO.

'RRAW_REF_REQ'

Registered copy of the refresh signal directly from the XCL. This refresh signal is used to initiate refreshes when RUN_SYS* is inactive. It goes from NMB_CLOCK to all the BCGAs.

'RSCAN_CTL.A'<2:0>
'RSCAN_CTL.B'<2:0>
'RSCAN_CTL.C'<2:0>
'RSCAN_CTL.D'<2:0>
'RSCAN_CTL.E'<2:0>

Sourced from NMB_CLOCK. Registered copied of XC_MB.SCAN_CTL<2:0>.

'RSELECT_E'<3:0>

From LDREG to READMUXE. Select for the 16 to 1 multiplexor in READMUXE. This select is encoded from the even side LD_EN and RTN_SEL signals and registered in LDREG.

'RSELECT_O'<3:0>

From LDREG to READMUXO. Select for the 16 to 1 multiplexor in READMUXO. This select is encoded from the odd side LD_EN and RTN_SEL signals and registered in LDREG.

'RTN_DATA_E'<31:0>

From READMUXE to RDEDC. This is the even side data selected by READMUXE from the even side NMCs.

'RTN_DATA_O'<31:0>

From READMUXO to RDEDC. This is the odd side data selected by READMUXO from the odd side NMCs.

'RTN_ECC_E'<7:0>

From READMUXE to RDEDC. This is the even side ECC (error correct code) selected by READMUXE from the even side NMCs.

'RTN_ECC_O'<7:0>

From READMUXO to RDEDC. This is the odd side ECC (error correct code) selected by READMUXO from the even side NMCs.

'RTN_MERR_E'

From RDEDC to even BCGAs. Indicates a multi-bit error (hard error) for the data which was sent to the XRTE on the previous clock.

'RTN_MERR_O'

From RDEDC to odd BCGAs. Indicates a multi-bit error (hard error) for the data which was sent to the XRTO on the previous clock.

'RTN_SEL_A_E'<1:0>
'RTN_SEL_A_O'<1:0>
'RTN_SEL_B_E'<1:0>
'RTN_SEL_B_O'<1:0>
'RTN_SEL_C_E'<1:0>
'RTN_SEL_C_O'<1:0>
'RTN_SEL_D_E'<1:0>
'RTN_SEL_D_O'<1:0>

Return selects from each of the BCGAs indicating from which of the four banks each BCGA controls is expecting return data. When no data is expected the return select is forced zero. The correspondence to particular BCGAs are as the LD_EN signals shown in Table A-5.

'RTN_SERR_E'

From RDEDC to even BCGAs. Indicates a single bit error (soft error) for the data which was sent to the XRTE on the previous clock.

'RTN_SERR_O'

From RDEDC to odd BCGAs. Indicates a single bit error (soft error) for the data which was sent to the XRTO on the previous clock.

'(NMB 4ISE1P)RWRE_DATA'<31:0>

Input staging register in ISE for the data from XSE.

'(NMB 4ISE1P)RWRE_PAR'<3:0>

Input staging register in ISE for the data parity from XSE.

'(NMB 3ISO1P)RWRO_DATA'<31:0>

Input staging register in ISO for the data from XSO.

'(NMB 3ISO1P)RWRO_PAR'<3:0>

Input staging register in ISO for the data parity from XSO.

'SCAN_CTL.A'<2:0>

'SCAN_CTL.B'<2:0>

Buffered copies of XC_MB.SCAN_CTL<2:0>.

'SENSE_VCC'
'SENSE_VCC_RET'
'SENSE_VEE'
'SENSE_VEE_RET'
'SENSE_VTT'
'SENSE_VTT_GA'
'SENSE_VTT_GA_RET'
'SENSE_VTT_RET'

These signals are connected to the power planes near the center of the logic area on the board. They are used to sense the voltages on the board for the power pallet regulators.

'SOFT_ERROR_E'<1:0>

Soft error signals from the even side BCGAs to NMB_CLOCK. NMB_CLOCK OR's these signals and registers the result which becomes MB_XC.SOFT_ERROR. SOFT_ERROR_E<0> is the wire-OR of BC0E and BC1E. SOFT_ERROR_E<1> is the wire-OR of BC2E and BC3E. A soft error is a correctable (recoverable) error.

'SOFT_ERROR_O'<1:0>

Soft error signals from the odd side BCGAs to NMB_CLOCK. NMB_CLOCK OR's these signals and registers the result which becomes MB_XC.SOFT_ERROR. SOFT_ERROR_O<0> is the wire-OR of BC0O and BC1O. SOFT_ERROR_O<1> is the wire-OR of BC2O and BC3O. A soft error is a correctable (recoverable) error.

'(NMB 2NMB_CLOCK1P)SYSDIS*'

Signal internal to NMB_CLOCK which is used to prevent the SYS_CK signals from being gated by QRUN_SYS* during CAST load (scan control equal 6) or ALL scan (scan control equal 7).

'SYS_241_CTL'<1:0>

Control signals for the 100E241 parts. All of these parts run on SYS_CK which is the clock gated by QRUN_SYS*. Table A-7 gives the actions taken for the different SYS_241_CTL values. Generated in NMB_CLOCK use in ISE and ISO.

Table A-7 SYS_241_CTL

Control Value	Action
00	Load, active during scan modes 0,1, and 6
01	Shift lsb to msb, used during scan modes 3 and 7
10	Hold, used during scan modes 5 and 6.
11	Unused control value, shift lsb to msb

'SYS_CK'<17:0>

'SYS_CK*'<17:0>

System clocks. These clocks are gated with QRUN_SYS* and are operated in synchronization with the rest of the system (crossbar, NSP, NIA, etc.). FREE_CK keeps running, in most cases, even when other system clocks are stopped so that refreshes may occur. Generated in NMB_CLOCK.

'(NMB 2NMB_CLOCK1P)SYS_EN*'

Signal internal to NMB_CLOCK. SYS_EN* is the enable to the SYS_CK clock buffers. It is active anytime scan controls are 4, 5, 6, or 7 or when QSYS_RUN* is active.

'SYS_MODE'

Used to gate the 100E241 control signals in ISE and ISO; generated by NMB_CLOCK. When true during scan modes 3, 4, 5, 6, and 7, the SYS_241_CTL scan controls are applied to the ISE and ISO 100E241s otherwise these registers are controlled internally to ISE and ISO.

'TEMP_VCC'

Temperature sensor VCC power.

'XBAR_ERROR_E'<3:0>

Even side input staging error signals from the even side BCGAs to the NMB_CLOCK. These signals indicate a parity error was detected by a BCGA in the input staging data received on the previous clock from XSE. Bit zero comes from BC0E, bit one from BC1E, bit two from BC2E and bit three from BC3E.

'XBAR_ERROR_O'<3:0>

Odd side input staging error signals from the odd side BCGAs to the NMB_CLOCK. These signals indicate a parity error was detected by a BCGA in the input staging data received on the previous clock from XSE. Bit zero comes from BC0O, bit one from BC1O, bit two from BC2O and bit three from BC3O.

'XC_MB.LOG_RUN*'

Log clock gating signal from XCL to NMB_CLOCK logic.

'XC_MB.RAM_RFSH'

Raw DRAM refresh signal from XCL to NMB_CLOCK logic. This signal is used to initiate refreshes when XC_MB.RUN_SYS* is inactive.

'XC_MB.SCAN_CTL'<2:0>

Scan control signals from XCL to NMB_CLOCK logic.

Table A-8 Scan Control Codes

Code	Action
0	Normal operation
1	Log ring scan
2	Unused
3	Sys ring scan
4	NMC Clock to Scan
5	NMC Clock to Normal
6	CAST Load
7	ALL ring scan

'XC_MB.SCAN_IN'

Scan input for all scan operations. From XCL to NMB_CLOCK logic.

'XC_MB.SYS_RUN*'

Sys clock gating signal from XCL to NMB_CLOCK logic.

'XSE_MB.ADDR'<28:3>

Address for even side requests. Bit numbering corresponds to physical addresses in NSP. From XSE to even side BCGAs and ISE.

'XSE_MB.CTL_PAR'<4:0>

Parity for even side request address, zone and cycle bits. Table A-9 shows the correspondence between control parity bits and address, cycle and zone. From XSE to even side BCGAs and ISE.

Table A-9 XSE_MB.CTL_PAR

Parity Signal	Corresponding Data
XSE_MB.CTL_PAR<4>	XSE_MB.WR_ZONE<3:0>
XSE_MB.CTL_PAR<3>	XSE_MB.ADDR<7:3>, CYCLE<1:0>
XSE_MB.CTL_PAR<2>	XSE_MB.ADDR<14:8>
XSE_MB.CTL_PAR<1>	XSE_MB.ADDR<21:15>

'XSE_MB.CYCLE'<1:0>

Specifies type of memory operation to perform as in Table A-10.From XSE to even

side BCGAs.

Table A-10 CYCLE Codes

CYCLE<1:0>	Operation
0	NOP
1	READ
2	WRITE
3	Test-And-Modify (TAM)

'XSE_MB.RDY'

Indicates that there is a valid request from the XSE to the even side logic. From XSE to even side BCGAs.

'XSE_MB.REF_REQ'

Indicates that all even side banks are to perform a refresh. From XSE to even side BCGAs.

'XSE_MB.WR_DATA'<31:0>

Write data for even side requests from XSE to ISE.

'XSE_MB.WR_PAR'<3:0>

Write parity for even side requests from XSE to ISE. Parity mapping is as in Table A-2.

'XSE_MB.WR_ZONE'<3:0>

Write zone for even side requests from XSE to ISE. Determines which bytes of the thirty two bit word are to be written on write and test-and-modify operations. A one on a ZONE bit indicates a byte is to be written. Correspondence between zone bits and data bytes are as in parity shown in Table A-2.

'XSO_MB.ADDR'<28:3>

Address for even side requests. Bit numbering corresponds to physical addresses in NSP. From XSO to odd side BCGAs and ISO.

'XSO_MB.CTL_PAR'<4:0>

Parity for odd side request address, zone and cycle bits. Table A-9 shows the correspondence between control parity bits and address, cycle and zone. From XSO to odd side BCGAs and ISO.

'XSO_MB.CYCLE'<1:0>

Specifies type of memory operation to perform as in Table A-10.From XSO to odd side BCGAs.

'XSO_MB.RDY'

Indicates that there is a valid request from the XSO to the odd side logic. From XSO to odd side BCGAs.

'XSO_MB.REF_REQ'

Indicates that all odd side banks are to perform a refresh. From XSO to odd side BCGAs.

'XSO_MB.WR_DATA'<31:0>

Write data for odd side requests from XSO to ISOISO.

'XSO_MB.WR_PAR'<3:0>

Write parity for odd side requests from XSO to ISO. Parity mapping is as in Table A-2.

'XSO_MB.WR_ZONE'<3:0>

Write zone for odd side requests from XSO to ISO. Determines which bytes of the

thirty two bit word are to be written on write and test-and-modify operations. A one on a ZONE bit indicates a byte is to be written. Correspondence between zone bits and data bytes are as in parity shown in Table A-2.

'ZONE_ONES_E'

From ISE to even side BCGAs. True if all bits in XSE_MB.ZONE<3:0> were one on the previous clock.

'ZONE_ONES_O'

From ISO to odd side BCGAs. True if all bits in XSO_MB.ZONE<3:0> were one on the previous clock.

Appendix B Diagrams

B.1 Views of the NMB

The tall, narrow rectangles on the right side of the NMB on the U side drawing are the microchannel connectors for the NMCs. The large squares labeled BC0E to BC3O are the eight BCGA gate arrays. Both the connectors and the gate arrays are placed on the Z side of the board.

There are also parts placed between the microchannel connector vias. These parts have been omitted from the figure for clarity.

B.2 Class Foils

C38XX

Memory Board/ Card

Training Class

Instructor: Marc Quattromani (mail: quattro)

NMB: Neptune Memory Board

- Large board containing many NMCs.

NMC: Neptune Memory Card

- Small card holding memory for a single bank.

First Half of Class:

- Interface
- Internal Dataflow
- Error checking

Second Half of Class:

- Using the NMB
- Debugging the NMB

Physical Layout of Neptune System

Up to Eight NMBs in a four bay system.

Up to Two NMBs per CPU bay.

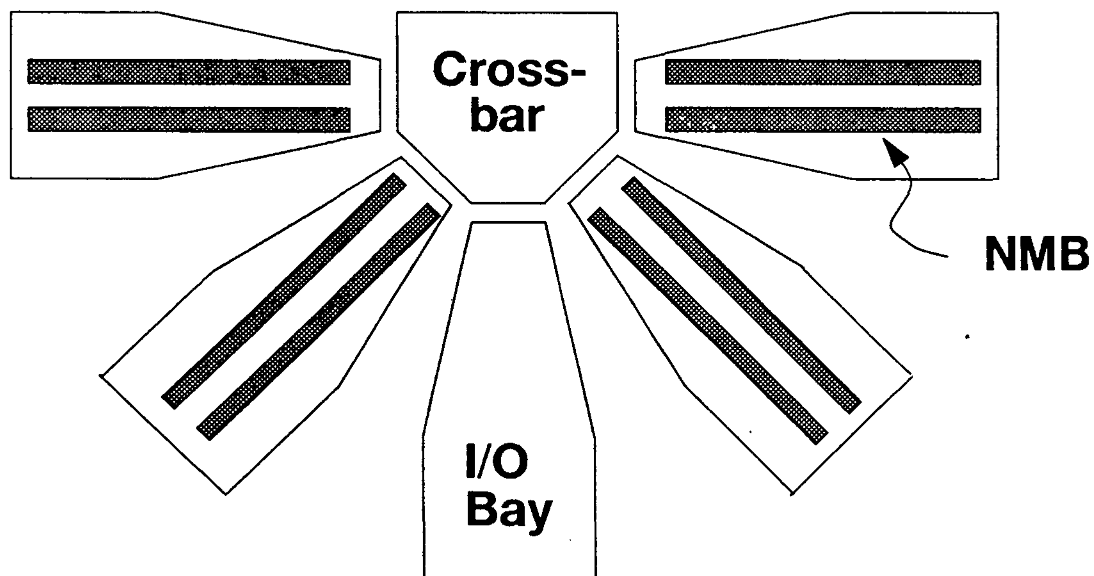
128MB, 256MB or 512MB per NMB.

Maximum of 4GB per C38xx.

32 banks per board, each bank is 32 bits wide.

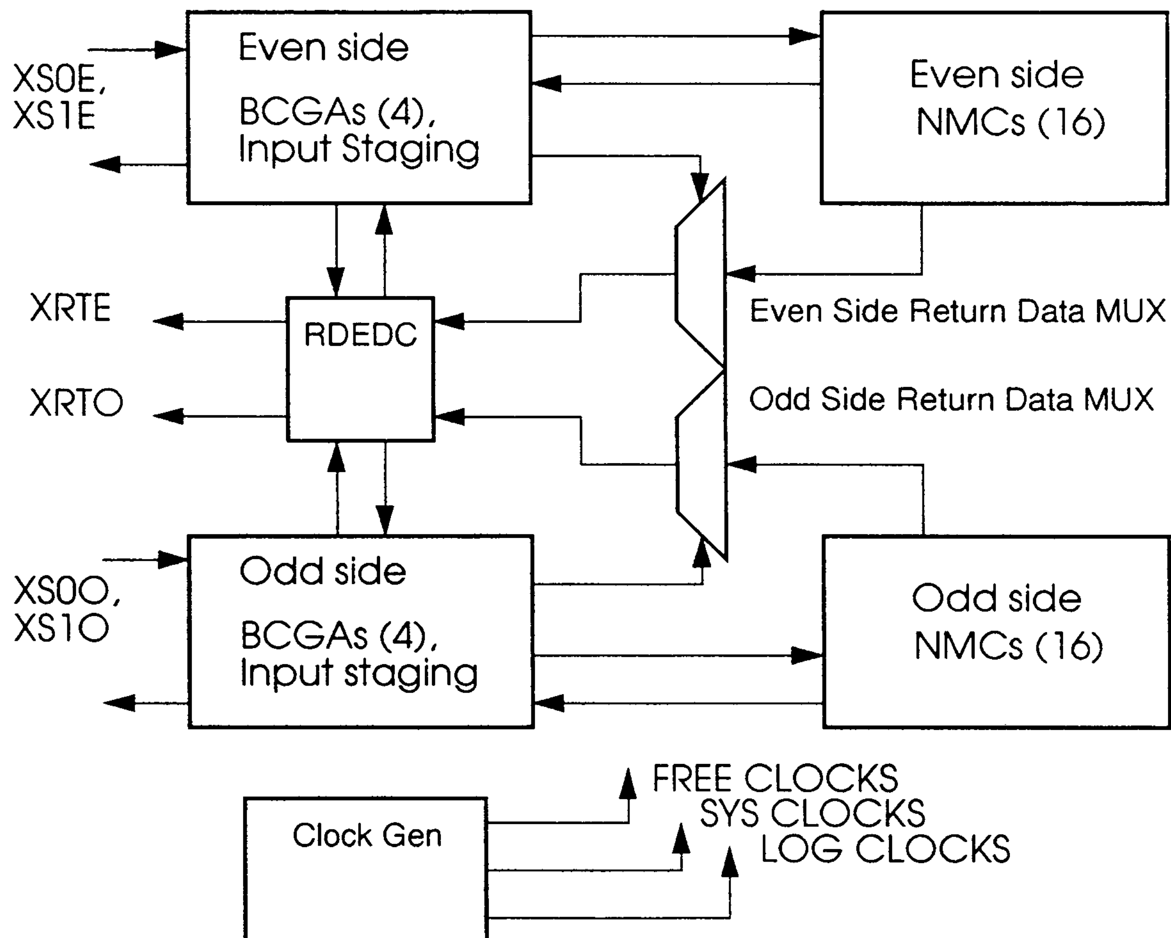
128 way longword, 256 way word maximum system interleaving.

Must have a power of two number of NMBs (1, 2, 4, or 8).



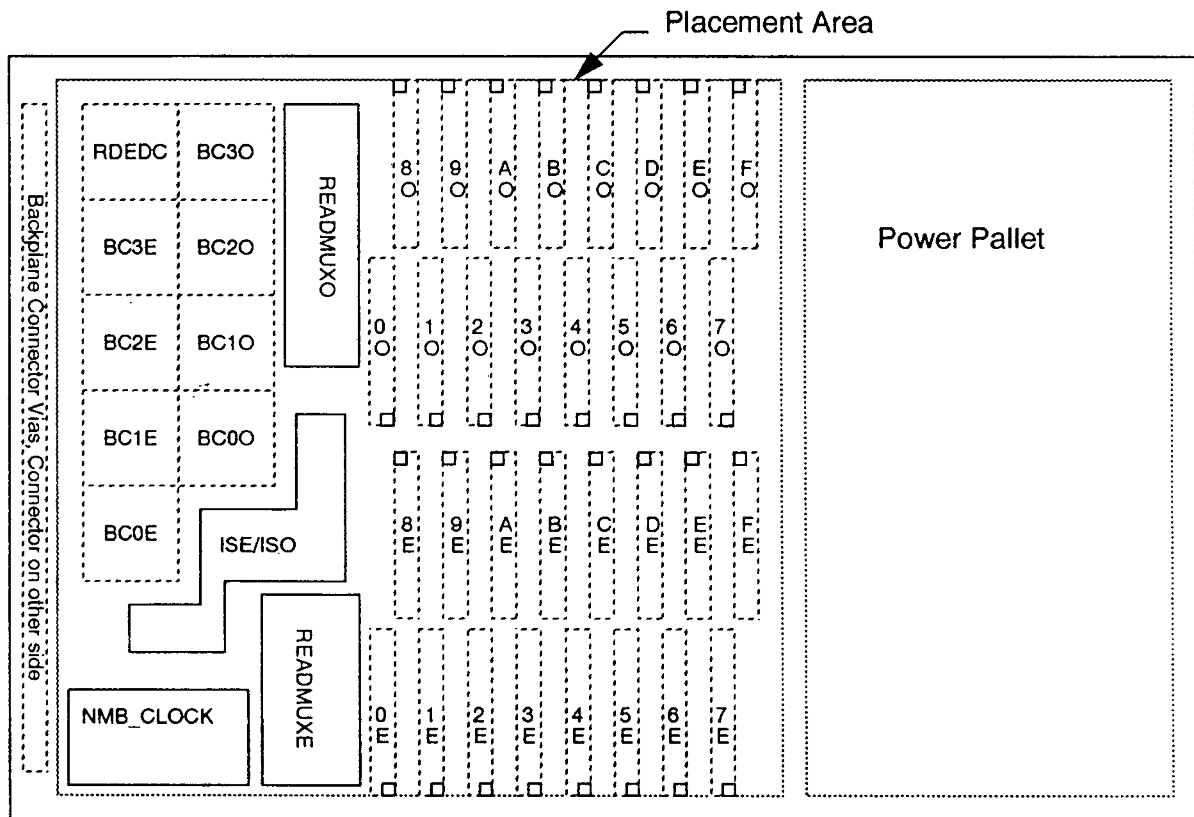
Basic NMB Organization

- Two identical halves, even and odd.
- Each NMC is an independent, 32 bit bank of memory.
- Eight BCGA gate arrays, each controlling four banks.
- One RDEDC for both even and odd return data.

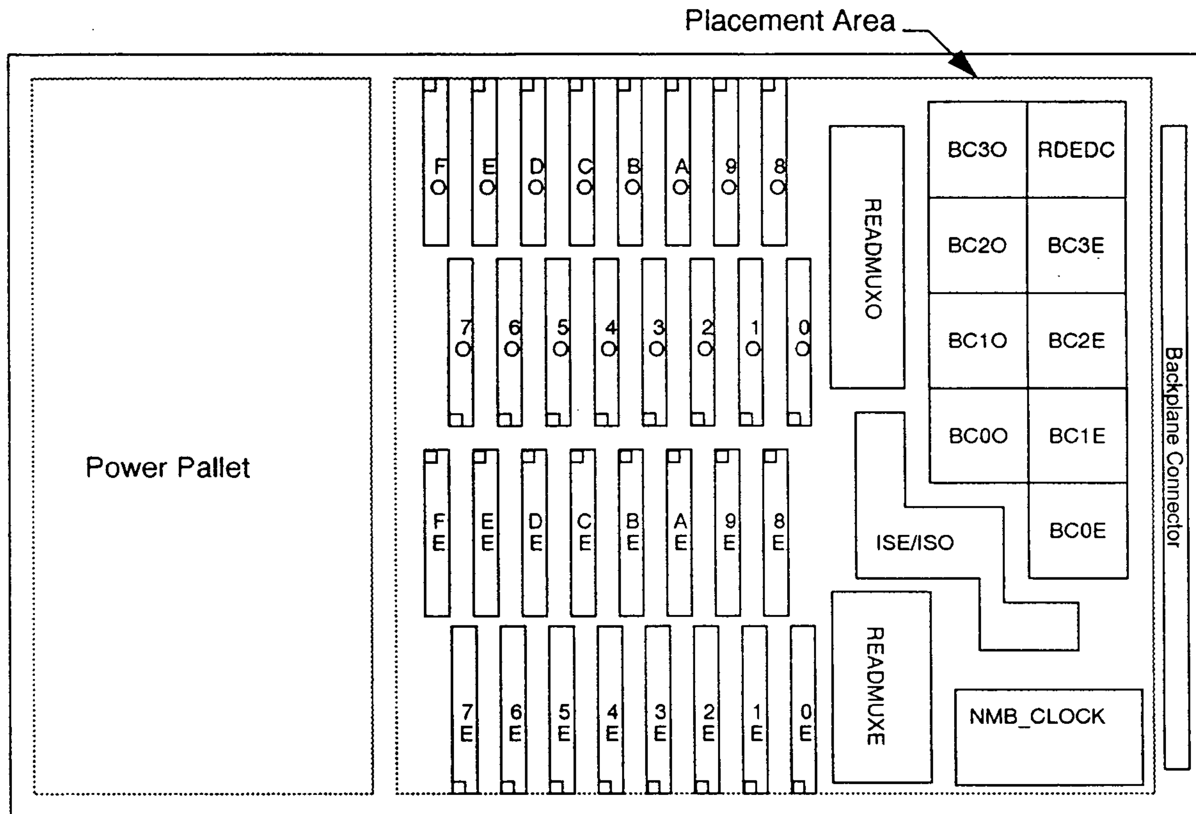


(Simplified Version of Figure 2-1 on Page 2-7)

Sketch of NMB U- side



Sketch of NMB Z- Side



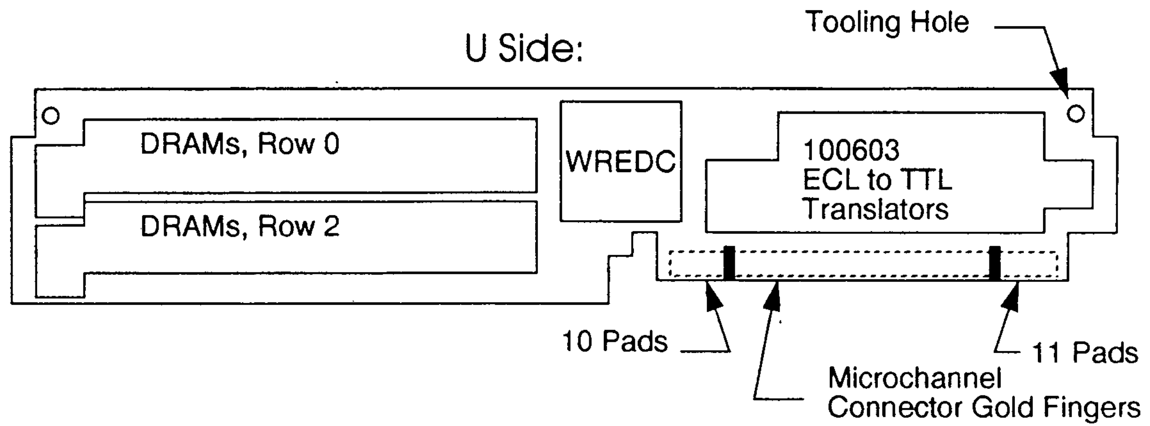
Key:

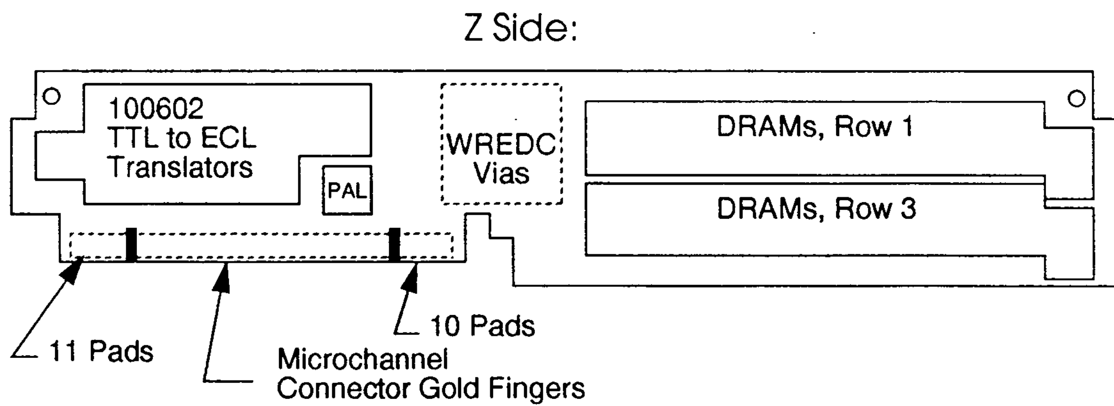
Parts on Z side

Pin One Marker

8 O Bank Number

Sketch of NMC Placement





Even Side Interface to Crossbar

XSE_MB.RDY	Indicates a valid request.
XSE_MB.ADDR	(26 bits). Address of requested operation.
XSE_MB.CYCLE	(2 bits). Type of operation.
XSE_MB.WR_ZONE	(4 bits). Indicates which bytes to write for writes only.
XSE_MB.WR_DATA	(32 bits). Write data.
XSE_MB.CTL_PAR<4:0>	(5 bits). Parity for address, zone and cycle.
XSE_MB.WR_PAR<3:0>	(4 bits). Parity for write data.

(Odd side interface is similar to even side with XSO substituted for XSE.)

Request Decode

Parity is always checked even if RDY is zero.

There is parity on all signals except for the RDY bit itself.

CYCLE and ZONE are decoded to determine exact type of operation:

CYCLE	ZONE	Operation:
00	Don't care	NOP, no-operation.
01	Don't care	Read
10	1111	Full Write
10	0000	Scrub operation
10	Anything else	Partial Write
11	Don't Care	Test-and-modify

Writes:

Full writes write all thirty two bits of a memory word.

Partial writes only modify some of the bytes in a full 32 bit word.

Scrub operations are used to correct soft errors in memory.

Addressing

NMB receives a 26 bit address:

- 2^{26} times 4 bytes per side times two sides = 512 MB.

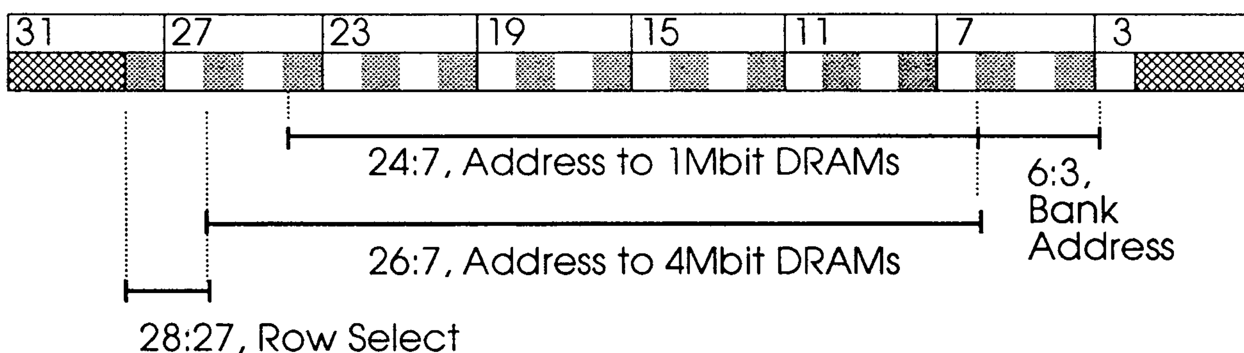
Three flavors of NMCs:

- Four rows, 1 Megabit DRAMs (4 MB per NMC, 128MB per NMB).
- Two rows, 4 Megabit DRAMs (8 MB per NMC, 256MB per NMB).
- Four rows, 4 Megabit DRAMs (16 MB per NMC, 512 MB per NMB).

128MB NMBs need only 24 bits of address.

256MB NMBs need only 25 bits of address.

32 Bit Address Word

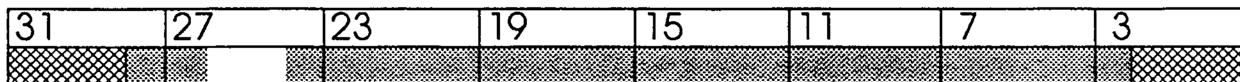


Bits 26 and 25 are unused (zero) when 1Mbit DRAMs are in use.

Bit 28 is unused (zero) when two row NMCs are in use.

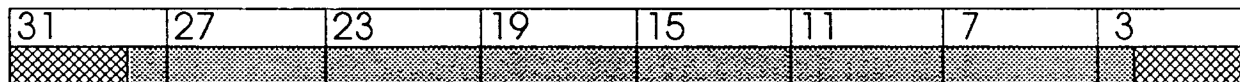
Valid Address Words

4MB NMC: 1Mbit DRAMs, Four Rows



8MB NMC: 4Mbit DRAMs, Two Rows

16MB NMC: 4Mbit DRAMs, Four Rows



Bank Busy

One bank takes 14 to 40 clocks to service a request. It can receive no further requests until it has completed the prior one.

The NMB contains no overrun queues. Any request sent to the NMB is immediately processed.

The XARB gate arrays on the XS0E and XS0O boards keep track of which banks on the NMB are not busy and are available for requests.

- When the XARB sends a request to a bank, it marks that bank as busy.
- When a bank completes a request, it tells the XARB that that bank is now free.

Refresh

All Dynamic Random Access Memories (DRAMs) require refresh.

DRAMs store information in leaky buckets (capacitors). If buckets are not periodically refilled, contents of memory are lost.

Standard DRAMs require 512 refresh operations in an 8 ms period, one every 15.6 microseconds, approximately every 960 clocks.

Normally, the XARB gate arrays tell the NMB when to do a refresh.

When clocks to XARB are turned off, the XCL board tells the NMB when to do a refresh.

For both cases, the NMB internally generates the address for the refresh.

ECC

Error Checking Code

Parity can be used to tell if a single bit was in error but it cannot determine which bit is incorrect.

Standard ECC is used to correct any single bit error and detect any two bit error.

The NMB and MCM3 boards use a modified ECC code which allows some three and four bit failures to also be detected.

- The NMCs use four bit wide DRAMs.
- The NMB ECC code can detect any failure in any single DRAM.

With ECC, the NMB will always correct a single bit error. A processor will not see incorrect data because of a single bit error.

The NMB cannot correct two, three or four bit errors but it can detect them and cause a hard error.

Commodity DRAMs are susceptible to occasional single bit errors. The ECC code keeps these errors from adversely effecting the machine.

NMB Block Diagram

Crossbar Boards:

- XS0E- Even side, send crossbar board. Contains XARB.
- XS1E- Even side, send crossbar board.
- XRTE- Even side return data crossbar board.
- XS0O, XS1O, XRTO- odd side crossbar board.

NMB Logic Blocks

- BCGA- eight bank control gate arrays, each controlling four banks.
- RDEDC- Return data gate array. Performs ECC on return data.
- READMUXE- Selects return data from one of sixteen even banks.
- ISE- Input staging for even side data, zone and row bits.
- READMUXO- Selects return data from one of sixteen odd banks.
- ISO- Input staging for odd side data, zone and row bits.
- LDREG- staging register for return data select signals from BCGAs to READMUX and RDEDC.

NMB Logic Blocks Not Shown on Diagram:

- NMB_CLOCK- Generates 1x, 2x, and SYS clocks for NMB. Also performs miscellaneous signal generation.
- MIM_CTL- Generates clocks for NMC LOG state.

READ Dataflow

Returns the contents of the 32 bit word at the specified address.

ZONE and Write Data are don't care but are still checked for good parity.

READs take 14 clocks.

Read data is returned 13 clocks after the request.

Timing is unaffected by single bit errors.

Possible Errors:

- Input parity error.
- Invalid address.
- Write data staging parity error.
- Single bit read data error (detected in RDEDC).
- Multiple bit read data error (detected in RDEDC).

Full Write Dataflow

A word aligned, 32 bit write. All bytes of a memory word are to be completely overwritten by new data.

ZONE equals 0b1111 (0xf).

No return data generated.

A full write takes 14 clocks.

Possible Errors:

- Input parity error.
- Invalid address.
- Write data staging parity error.

Partial Write Dataflow

A write where ZONE does not equal all ones, including a write where ZONE is all zero (a scrub operation).

- Byte, halfword, unaligned word or longwords writes.

No return data generated.

If a single bit data error is detected, the partial write is repeated using the corrected data.

Partial Write Cycle Timing:

20 clocks if no errors detected.

40 clocks if errors detected.

Possible Errors:

- Input parity error.
- Invalid address.
- Write data staging parity error.
- Single bit data error (detected in WREDC).
- Multiple bit data error (detected in WREDC).

A scrub operation is used to correct a single bit error in memory.

Test-and-Modify Dataflow

A partial write where data read from memory is returned to the requestor before memory is modified.

Except for return data, dataflow is identical to partial write.

As with a partial write, if a single bit data error is detected, the operation is repeated using the corrected data.

Only single byte test-and-modifies are generated by the NSP and NIA although the memory system supports all sizes.

Test-and-Modify Cycle Timing:

20 clocks if no errors detected.

40 clocks if errors detected.

Data returned after 13 clocks.

Possible Errors:

- Input parity error.
- Invalid address.
- Write data staging parity error.
- Single bit data error (detected in WREDC).
- Multiple bit data error (detected in WREDC).
- Single bit data error (detected in RDEDC).
- Multiple bit data error (detected in RDEDC).

Both RDEDC and WREDC should report the same errors.

Refreshes and NOPs

NOPs are used by the NIA to pad out a request.

Refreshes and NOPs do not alter memory.

Basic error checking is still performed.

Both operations take 14 clocks.

Possible Errors:

- Input parity error.
- Invalid address.
- Write data staging parity error.

Soft Errors

Soft errors are correctable errors. They do not cause the machine to halt as with a hard error.

Soft errors can be detected whenever the DRAMs are read:

- Reads
- Partial Writes
- Test and Modifies

Soft errors only effect NMB operation when a partial write or test-and-modify causes an error.

In these two cases, cycle time is doubled.

Soft errors are otherwise transparent to the operation of the machine.

Soft Error Logging

When a soft error occurs, the NMB saves:
- Address
- Data
- Cycle type

The NMB asserts a MB_XC.SOFT_ERROR signal which causes the service processor (SPU) to take an interrupt.

The SPU then scans out the saved error state to examine the error.
- This scan does not effect NMB operation.

Once the address of the error is known, the SPU issues a scrub operation to that address to correct the corrupt data.

NMB operation is not interrupted during this procedure.

Hard Errors

Hard errors are non-recoverable errors. A hard error causes the machine to halt.

Classes of Hard Errors on NMB:

- Input parity errors.
- Internal parity errors.
- Invalid address.
- Multiple bit data errors detected by the WREDC.
- Multiple bit data errors detected by the RDEDC.
- Request to a busy bank.

Coverage:

Data is covered at all steps:

- from crossbar to NMB
- from NMB to NMC
- from NMC to RDEDC
- from RDEDC to crossbar

Address, Cycle and Zone are only covered from crossbar to NMB.

D

LOG, SYS, and ALL State

All registers on the NMB are either LOG, SYS or ALL state.

LOG registers are used to save soft error information.
- These registers are lookaside state.
- LOG registers are not needed for NMB operation.
- LOG registers may be halted or scanned without effecting operation of the NMB.

SYS registers interface with the crossbar.
- SYS registers contain hard error state.
- SYS registers are clocked synchronously with all registers on the crossbar and processors.
- SYS registers may be scanned without effecting refresh or the contents of memory.

ALL registers are the internal registers for the NMB.
- ALL registers are used to cycle the DRAMs.
- Scanning or halting the ALL registers causes the contents of memory to be destroyed.

Each of the three types of state has its own scan mode.

Clocks and Run Bits

NMB receives a single clock from the NCU running at 2X the system frequency.

The 2X clock is gated into a 1X clock by a phase generator circuit which masks every other 2X clock pulse.

Only the BCGAs receive a 2X clock. All other parts except for the phase generator receive a 1X clock.

The clocks to the LOG registers are 1X clocks gated by a RUN_LOG* signal from the XCL.

LOG_RUN*

- RUN_LOG* is active low.
- A "1" causes LOG clocks to stop even though the 2X clock from the NCU is still running.

SYS_RUN*

The clocks to the SYS registers are 1X clocks gated by a RUN_SYS* signal from the XCL. It is similar in function to RUN_LOG*.

LOG Scan

LOG scan scans just the log state.
- Used to examine soft error logging information.
- Does not effect the operation of the NMB.

The 2X clocks must be running for a LOG scan to function properly.

LOG scan procedure:
- RUN_LOG* is de-asserted to turn off LOG clocks.
- SCAN_CTL is set to LOG scan mode.
- RUN_LOG* is turned on for a number of clocks equal to the number of clocks in the LOG ring.
- RUN_LOG* is turned off.
- SCN_CTL is set to normal mode.
- RUN_LOG* is re-asserted and the LOG registers resume normal operation.

Location of LOG registers:
- Each BCGA has address and cycle LOG registers for each bank.
- The RDEDC has LOG registers for even and odd data.
- The WREDCs on the NMCs have LOG registers for data and error type.

The LOG ring contains 3156 bits.

NOTE: Reading the LOG ring automatically clears the ring.

SYS Scan

SYS scan scans the LOG state plus the SYS state.

- Used to examine hard error state.
- Requires the system to be halted.
- Does not interfere with refresh.

SYS scan is very similarly in operation to LOG scan:

- 2X clocks must be running.
- RUN_SYS* and RUN_LOG* gate the clocks for SYS scan.

SYS registers:

- All LOG registers are part of the SYS ring.
- BCGAs' interface to the crossbar and return data registers.
- Remainder of RDEDC registers.
- Input staging registers in ISE and ISO.
- Error output staging.
- LDREG state.

The SYS ring contains 3978 bits.

ALL Scan

ALL scan scans all scannable registers on the NMB.

- Used to initialize the register state of the board.
- ALL scan destroys the contents of the DRAMs. Memory must be re-initialized after ALL scan.

ALL scan requires a specialize initialization clock before and after scan.

- Otherwise, ALL scan is similar to the scans of other boards.
- ALL scan leaves the 2X clock turned off.

Registers NOT in the ALL ring:

- Most WREDC registers except for lookaside.
- Registers used in the state machine that generates the clocks for the WREDCs.

The ALL ring contains 9474 bits.

Single Step

The Neptune system, as a debug feature, has the capability to issue a burst of one or more clocks and then pause.

When clocks are stopped, the NMB DRAMs must still receive refreshes so that the contents of memory are not lost.

The NMB's 2X clock is a free running clock.

- The 2X clock is not stopped when other system clocks stop.
- The NMB's RUN_SYS* and RUN_LOG* are de-asserted instead.

The NMB therefore contains free running logic for refreshes and logic that stops with the rest of the system.

- Free running logic is labeled "A" in the diagram.
- Stopped logic is labeled "S" and "L."

The NMB has dedicated logic to handle the interface between the free running and stoppable logic so that no requests are lost.

Even though the NMB never completely stops during single step, to the crossbar and the rest of the system, the NMB appears to stop and start in step with the rest of the system.

Unlike C2, memory latency is not affected by single step.

Clocks, Run Bits and Refresh

29

The NMB must be performing refresh in order to retain the contents of memory.

For refresh to take place, the NMB must be receiving 2X clocks.

One of the following must also be true:

- The NMB RUN_SYS signal is asserted and the crossbar is initialized and receiving clocks.
- The NMB RUN_SYS signal is de-asserted.

If RUN_SYS is asserted and the crossbar is not receiving clocks, the NMB will look to the crossbar for refreshes but never receive any.

28

SCAN and NMB State

LOG and SYS scan can be done without effecting the contents of memory.

ALL scan destroys the contents of memory.

When the RUN_SYS bit is de-asserted, the NMB ignores the crossbar.

- RUN_SYS must be de-asserted if the crossbar is scanned.
- Otherwise, the NMB will interpret crossbar scan state as requests and probably detect errors.

The ALL ring is referred to as **mb#** where # is the number of the board to be scanned, as in **get mb0:*sig***.

The SYS ring is referred to as **mbs#**, as in **get mbs0:he.hard_error**.

The LOG ring is referred to as **mbl#**, as in **get mbl0:nmc0e_rerr**.

dsh Commands and NMB Clock and RUN bits

27

The NMB's 2X clock is considered a free running clock by the dsh.

- **scnrun** and **halt** do not effect the 2X clock unless the clock is specifically called out.
- The **scnrun** command by itself turns on all non-free running clocks but does nothing to the NMB free running clock.
- **halt** with no parameters will leave the NMB clock on.
- **scnrun mb0:2x** turns on the NMB clock and it's run bits.

An ALL scan of the NMB leaves the NMB's 2X clock turned off and the run bits off.

A LOG or SYS scan:

- does not effect the 2X clock,
- after scan, leaves the RUN_SYS and RUN_LOG off if they were originally off,
- or on if they were on before the scan.
- The scan will not work if the 2X clocks are off.

The NSP, NVP and all the crossbar boards except for the XCL receive non-free running clocks.

96

dsh Command Examples

Turning on system clocks:

scnrun mb0:2x mb1:2x mb2:3x mb3:2x ia8:3x

scnrun

- The first line turns on the free running clocks.
- The second line turns on the rest of the system clocks.

Stopping the system without losing memory or I/O state:

halt

- halt by itself leaves the free running clocks going.
- a scnrun can then be used to continue.

Single step:

scnrun mb0:2x mb1:2x mb2:3x mb3:2x ia8:3x

clock 10

- Free running clocks were turned on then,
- non-free running clocks were given ten clocks.

More dsh Examples

Errors

25

Improper SYS scan:

get mb0:*signature*

get mbs0:he.hard_error

- The first command does an ALL scan which turns 2X clock off.
- The get of the SYS ring fails because the 2X clock is off.

Loss of Refresh:

halt

scnrun mb0:sys or **scnrun mb0:2x**

- The second command turns the NMB's RUN_SYS bit on.
- With RUN_SYS asserted, the NMB looks to the crossbar for refreshes.
- The **halt** turned the crossbar off so the NMB receives no refreshes.
- If the crossbar was now scanned, the NMB would also process bogus requests.

2-4

NMB Diagnostics

Several types of diagnostics are available for testing the NMB.

Following foils discuss these diagnostics in the order they should be executed for bringing up a new board.

spu_4000

23

spu_4000 is used to verify the scan engine as well as the scan rings on each of the boards in the system.

spu_4000, st_601 (610), should be used to verify scan engine functionality any time the functionality of the scan engine is in doubt.

spu_4000, st_601 (611), verifies the functionality of each board's ring or rings.

- A sub-menu selects board and ring.
- The NMB ALL ring should be tested first,
- followed by the SYS ring and the LOG ring.
- One hundred or more iterations should be executed to find any flaky scan ring problems.

The LOG ring is a special case ring. It does not scan out in the same order that it scans in.

- During normal dsh log scan, the ring is filled with zeroes.
- In spu_4000, only certain patterns are executed for the LOG ring test.

All NMB scan rings should be fully functional after these tests.

SST

SST is a scan ring based test which verifies network topologies.

- Since it uses the ALL scan ring, that ring must be working in order for sst to function.

SST should find hard opens or non-functional parts on the NMB. It will probably find most shorts as well.

SST does not check any signals going to or from the NMCs.

- spu_4000 and SST will cover approximately 40-45% of the NMB.

SST can be used to check the NMB to crossbar connectivity. A full crossbar is required for this test, of course.

- The SST tests between boards are known as "partials."

After successfully executing SST:

- The input staging to bank staging registers has been verified.
- Most of the BCGA logic has been verified.
- The return datapath has been verified.

21

mem4000

mem4000 is a comprehensive, functional test of the memory system.

mem4000 has six sections:

- st_50: a scan ring verification of all memory system boards.
- st_100-199: scan based test of the crossbar.
- st_200-299: scan based test of the NMB.
- st_300-349: scan based test of NMB/crossbar complex.
- st_300-399: at speed test of the memory system using the NIA memory test logic and xmap interface.
- st_400-450: CPU based memory tests.

All of mem4000 should be executed to fully verify an NMB.

After mem4000, NMBs have typically been fully functional.

The following foils list the subtest groups in order of execution.

st_50, scan ring verification, may be omitted in favor of spu_4000.

mem4000 st_200-299

Scanned Based NMB Tests

One of the most important mem4000 tests is st_200:

- Scan based verification of the NMB to NMC connections.
- This test covers the remaining 55-60% of the NMB that sst does not cover.
- After sst and st_200, most board opens and non-functional devices should be discovered.
- should be executed any time an NMC flaw is suspected.
- **There is no scan based test of DRAMs.**

st_201, Scan-Based MB PCM test:

checks illegal address checking (PCM: Physical Configuration Map.)

st_202, 203, 204, 206:

checks parity and ECC functionality of NMB.

st_205, Scan-Based MB Single Step:

checks ability of NMB to interface system and free running logic.

mem4000 st_300-349

Scan Based Test of NMB/ Crossbar

Before executing these subtests, sst and mem4000 st_100 to st_105 can be used to scan verify the crossbar boards. In addition to the memory board being tested, the XS0E, XS1E, XRTE, XS0O, XS1O and XRTO are required.

Highlights:

st_301, Scan-Based Request Test:

checks NMB's ability to perform basic reads and writes by scanning requests into the crossbar.

st_303, Scan Based TAC/TAS Test:

checks test and modify operations.

st_307, Scan Based Commreg/ Memory Contention Test:

checks commreg/ memory contention. An NCU with working commregs is required for this sub test.

After completing these subtests, the basic operation of the memory board/ crossbar has been verified. The DRAMs have been partially exercised at this point.

mem4000 st_350-399

NIA Based Memory System Tests.

In addition to the boards required for the previous subtests, a functioning NIA is required for these tests.

st_350-399 verify all memory locations and all memory operations.

- Writes and full reads are verified at speed using the memory test logic on the NIA.
- Partial writes, TAMs, scrubs and NOPs are verified using the xmap hardware.

st_350, Memory Test Logic, address patterns, and st_375, Memory Test Logic, various patterns, can be used to find a bad DRAM or DRAM row.

After these subtest are successfully executed:

- NMB shown to handle many requests at speed.
- NMB has been shown to retain memory for long periods.
- All memory locations have been written with various patterns.
- All DRAMs have been checked.
- At speed partial writes and TAMs have not been checked.

mem4000 st_400-499

CPU Based Memory System Test

17

These tests requiring a functioning CPU in addition to everything needed for the st_350-399 subtests.

st_400-499 covers the same functionality as st_350-399 but is more thorough.

Interpreting Hard Errors

16

The **nmb_errs** script decodes the error state on the NMB.

- Script prompts for hard errors or soft errors check.
- Soft error check scans only the LOG ring.
- Hard error check uses the SYS ring.
- Script reports error type, address, cycle and data for the error.

A persistent soft error is really a hard error.

- Soft errors should be rare.
- A persistent error indicates a flaw in the board.
- The ECC allows the error to be corrected but the board is still flawed.

Different classes of NMB hard errors indicate failed NMB, NMC or even a failed crossbar, processor, backplane or cable.

15

NMB/ Crossbar Errors
Send Path

Input Staging Errors:
- Parity error in address or cycle detected by a BCGA, ISE or ISO.
- Parity error in write data or zone detected by ISE or ISO.

Crossbar does not check request parity.
- Bad parity may have been generated by the processor making the request,
- or corrupted anywhere between the processor and memory, either in the crossbar or in the cabling/ backplanes.

To help isolate the failure:
- The NMB informs the crossbar of the error;
- the crossbar freezes state which holds the request state while it was in the crossbar.
- If the crossbar state contains the error:
  Error occurred between processor and crossbar.
- If the crossbar state is error free:
  Error occurred between the crossbar and memory board.

Diagnostics:

SST partials between the NMB and crossbar or the processor and crossbar should be run to check connectivity between the elements in question.

14

NMB/ Crossbar Errors
Return Path

A similar state saving mechanism is used for return data.

The crossbar saves the return state.

If a processor detects an error on memory return data, the crossbar freezes its return state.

- The error can then be isolated between the NMB and crossbar
- or between the crossbar and processor.

Diagnostics:

SST partials between the NMB and crossbar or the processor and crossbar should be run to check connectivity between the boards in question.

13

ECC Errors on Partial Writes and Test-and-Modifies

Certain ECC errors indicate a bad NMC:

- ECC errors on partial writes (detected by WREDC).
- ECC errors on test-and-modifies detected by both the WREDC and RDEDC.

It is still possible these errors are caused by some sort of NMB flaw.

- However, it is most likely a datapath problem on the NMC.
- NMB problems would be the BCGA controlling the NMC,
- or the MIM_CTL which generates NMC clocks.

An ECC error on a test-and-modify that is detected by the WREDC but not the RDEDC indicates:

- A probable timing error with the WREDC on the NMC.
- Possibly a problem with the NMB (a BCGA or the MIM_CTL).

An ECC error on a test-and-modify that is detected by the RDEDC but not the RDEDC indicates: — WREDC

- A flaw in the data out path from NMC through READMUX to RDEDC.
- Flaw could be NMC or NMB related.

Diagnostics for ECC Errors on Partial Writes and TAMs

mem4000 is the primary tool for finding these flaws.

- st_355, a partial write test, and
- st_360 and st_365, TAM tests, should reproduce these errors.

st_350, an at speed read and write of the entire memory board, can be used to verify the ECC generation by the WREDC and ECC checking by the RDEDC.

- This subtest should find any flaw except for a WREDC ECC check flaw.
- This subtest is more complete than the previous two subtests.
- If st_350 finds the flaw, the error should be treated as a Read ECC error (next foil).

Swapping NMCs for the bank in question should isolate the flaw to the NMC or to the NMB.

ECC Errors on Reads

ECC read errors are only detected by the RDEDC.

- Path being checked is NMC to READMUX to RDEDC.
- Flaw could be on NMC or NMB.

Transient errors are more likely to be NMC failures.

Diagnostics:

mem4000, st_350 or 375, should be able to reproduce the failure.

mem4000, sub_200, will statically check the connectivity from the NMC to the READMUX and RDEDC. Any static flaws should be found by this test. If this test passes, flaw is likely to involve the DRAMs on a bank.

Swapping the NMC for the bank in question with another NMC should isolate the flaw to the NMB or the NMC.

Bank Parity Error

A bank parity error occurs when a WREDC detects a parity error in the write data passed from the bank staging register to the WREDC.

All memory operations including refreshes check for this error.

A bank parity error indicates a flaw in the input staging to bank staging to NMC data path.

Diagnostics:

sst will check the connectivity from the input staging registers to the bank staging registers.

mem4000, st_200, will check the connectivity from bank staging to the WREDC on the NMC.

Swapping the NMC for the bank in question with another NMC should isolate the flaw to the NMB or the NMC.

ECC Errors on Reads

ECC read errors are only detected by the RDEDC.

- Path being checked is NMC to READMUX to RDEDC.
- Flaw could be on NMC or NMB.

Transient errors are more likely to be NMC failures.

Diagnostics:

mem4000, st_350 or 375, should be able to reproduce the failure.

mem4000, sub_200, will statically check the connectivity from the NMC to the READMUX and RDEDC. Any static flaws should be found by this test. If this test passes, flaw is likely to involve the DRAMs on a bank.

Swapping the NMC for the bank in question with another NMC should isolate the flaw to the NMB or the NMC.

Bank Parity Error

A bank parity error occurs when a WREDC detects a parity error in the write data passed from the bank staging register to the WREDC.

All memory operations including refreshes check for this error.

A bank parity error indicates a flaw in the input staging to bank staging to NMC data path.

Diagnostics:

sst will check the connectivity from the input staging registers to the bank staging registers.

mem4000, st_200, will check the connectivity from bank staging to the WREDC on the NMC.

Swapping the NMC for the bank in question with another NMC should isolate the flaw to the NMB or the NMC.

D

9

Illegal Address Errors

These errors are detected when a reference is made to non-existent memory such as row four in a two row NMC system.

Illegal Address errors should be ignored if a parity error was detected on address input staging.

An illegal address error without a parity error indicates:

- Incorrect cop file information.
- A possible software error.
- An error in a processor or NIA.
- Incorrect data in the physical address translation tables:
  - -caused by a software mapping error,
  - -or by memory returning incorrect data during PTE fetches.

Diagnostics:

mem4000, st_201, tests the address checking logic.

CPU and NIA based memory diagnostics can be used to exercise the processor end of requests.

Control Hard Errors

Control hard errors are detected by the individual bank controllers within the gate array when a request is made to a busy bank.

No state is saved when a control hard error occurs.

Possible causes:

Faulty BCGA generating spurious BANK_DONEs.

Faulty XARB gate array on the XS0 board.

Noise on the BANK_DONE, RDY or refresh lines.

Run after refresh problem if problem occurs immediately after restart of system clocks.

Diagnostics:

sst partials between the crossbar and NMB will statically test the BANK_DONE, RDY and refresh signals.

mem4000, st_350-399, will exercise the bank done hardware.

7

Swapping NMCs

Many failures are NMC specific.

As has been mentioned before, an NMC from a suspect bank should be swapped with an NMC from a known good bank to isolate the problem to a bank or the NMB.

Scripts

The NMB debug team has developed several scripts:

mam is useful for decoding addresses into bank, row, page row and page column address.

nmb_errs decodes the NMB hard and/or soft error state.

nmc_test is a script version of mem4000 st_200. It is slower, over all, than mem4000 but can be use to test a particular signal or range of signals and has better looping control.

5

General Debugging Techniques

Address of failure is an important debug clue:

- Use the **mam** script to find any locality in the address.
- Look for failures localized to:
  - a particular bank,
  - a particular row of a particular bank,
  - a particular address range across multiple banks,
  - a particular data bit on a particular bank,
    - a set of eight banks,
    - or all twelve banks on a side.

With an understanding of the logical groupings on the NMB, failure locality will indicate where the flaw is.

The even and odd memory banks on an NMB are almost completely independent.

- Failure on both even and odd requests indicates a non-NMB related problem or
- a scan, clock or major, board-level noise problem.

4

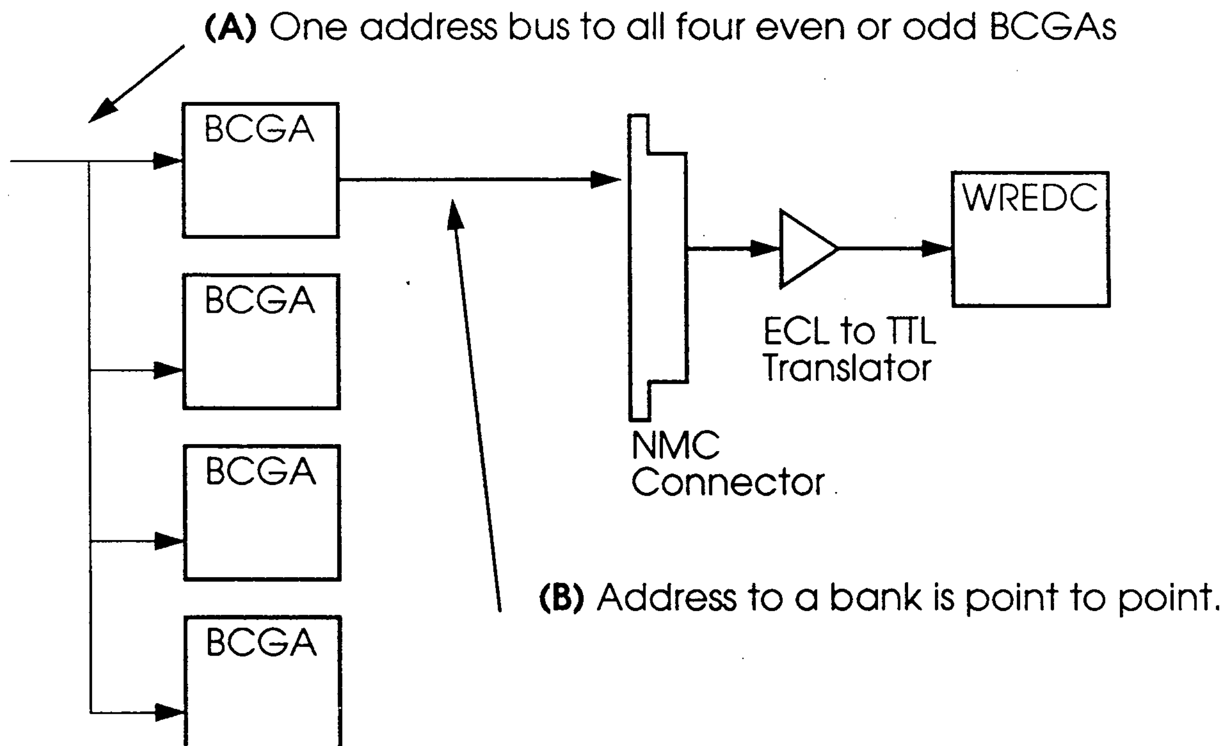
Address Logical Groups

All BCGA's on an even or odd side receive the same address.
- Input staging error should cause parity error.

Address to a bank goes directly from one BCGA to the NMC.

Failures at **(A)** effect all banks on a side.

Failures at **(B)** or beyond effect only a single bank.

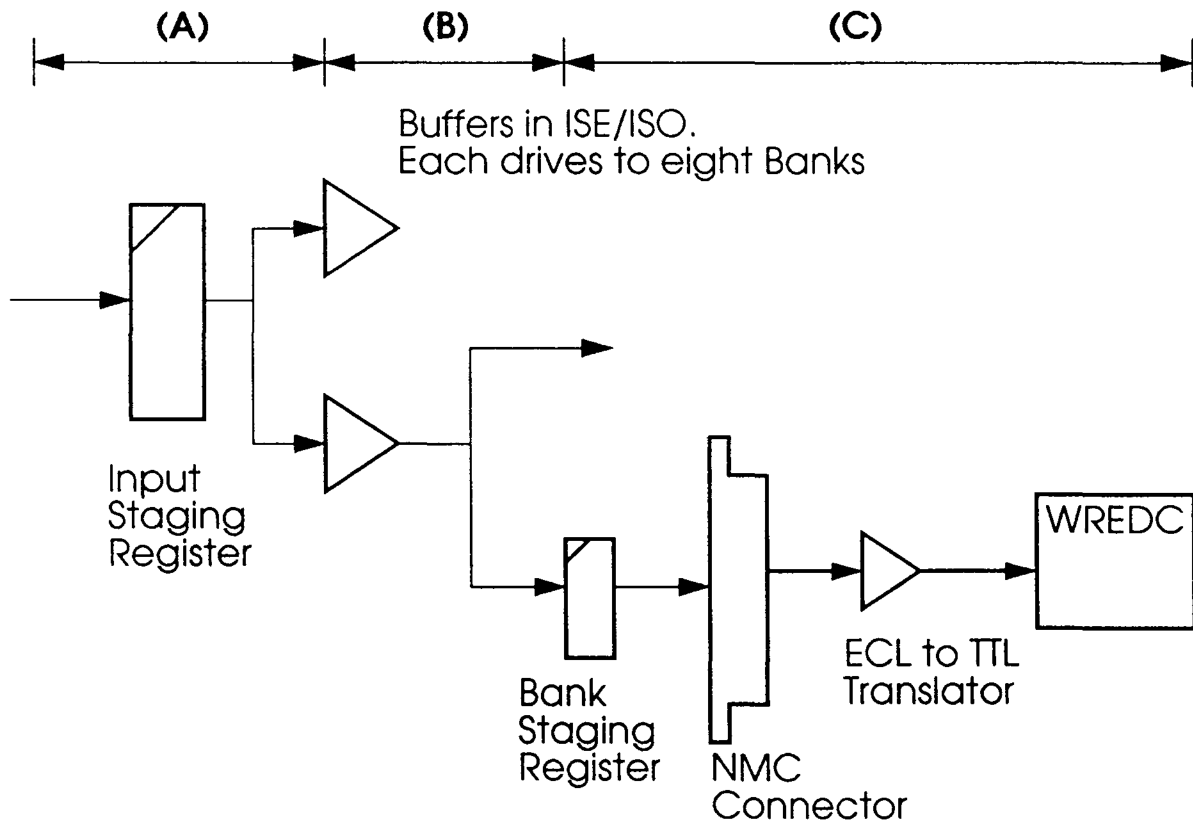


Data Input Logical Groups

A flaw in input staging, (A), would effect all sixteen banks on a side.

A flaw in data fan out, (B), would effect eight banks, 0 to 7 or 8 to 15.

A flaw in bank staging, the NMC connector or the NMC, (C), would only effect one bank.



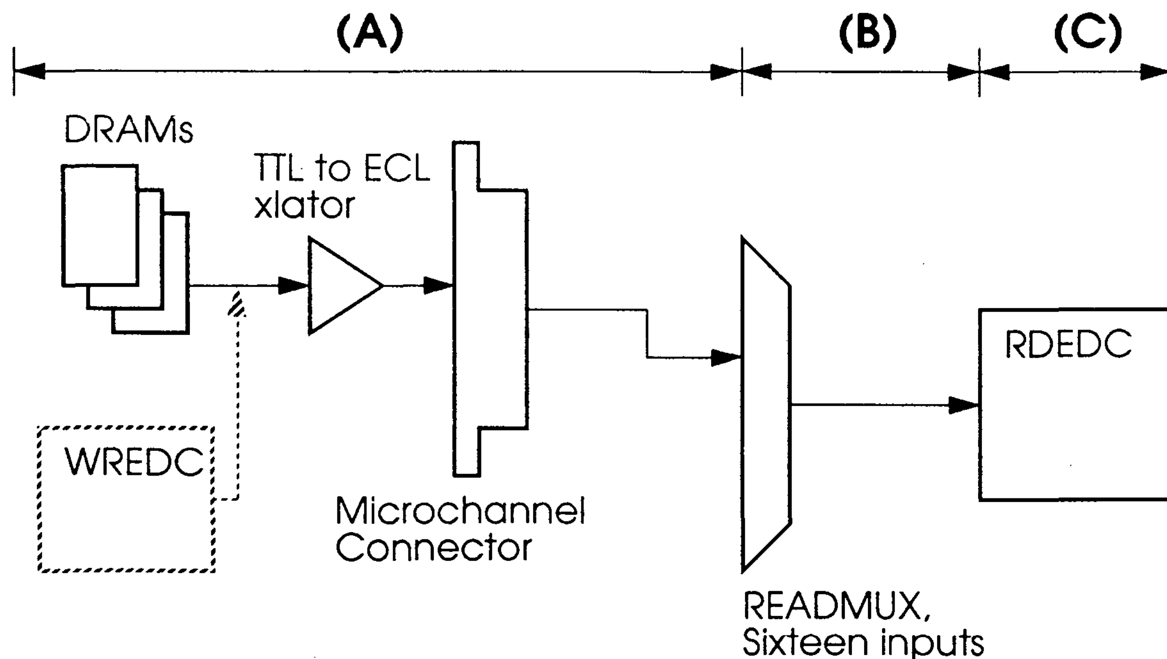
Data Output Logical Groups

Flaws in the NMC to the inputs of the READMUX, (A), would effect only a single bank.

Flaws in the READMUX to the RDEDC inputs, (B), would effect all sixteen banks on a side.

Flaws in the RDEDC itself, (C), could effect all return data since the RDEDC services both even and odd requests.

NOTE: nmc_test and mem4000, st_200, use the WREDC to drive the data out bus. These tests can be used to isolate the DRAMs from the rest of the path under test.



Summary of Logic Groups

BCGAs effect groups of four banks.

Input staging fan out to bank staging effects groups of eight banks.

READMUX effects groups of sixteen banks.

RDEDC effects groups of sixteen or thirty two banks.

B.3 Example of NMC Scan

The NMC scan can be a little confusing since the NMCs are clocked only once every eight clocks during scan operations. Following are two examples of NMC scan. The first represents the case where the total number of bits in the scan ring is a multiple of eight, the second the case where it is not a multiple eight. ALL and SYS scan fall into the first case while LOG scan falls into the second case. In both examples, a truncated version of the scan ring is used to keep the example short.

Figure B-1a NMC Scan with a Multiple of Eight Bits

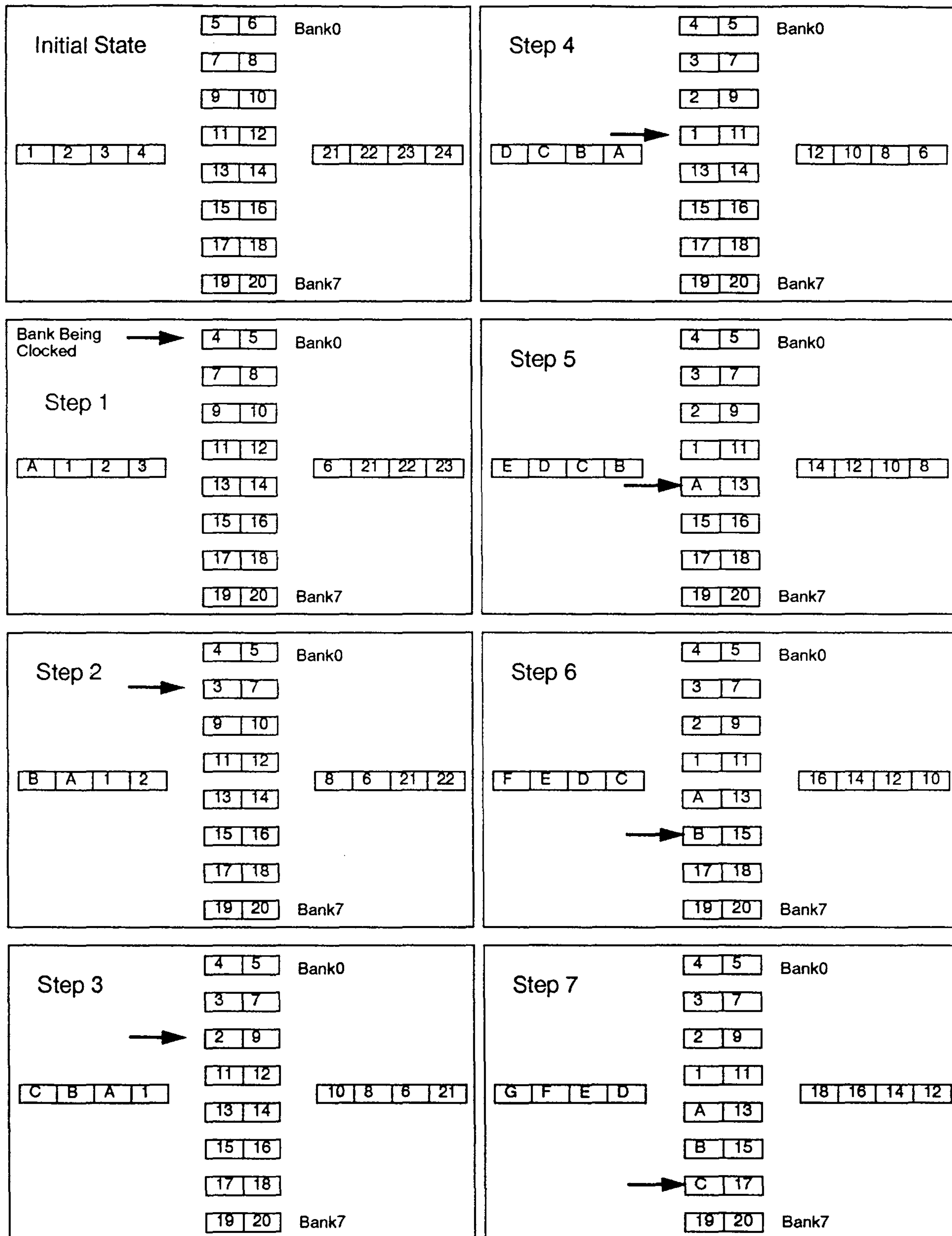


Figure B-1b NMC Scan with a Multiple of Eight Bits

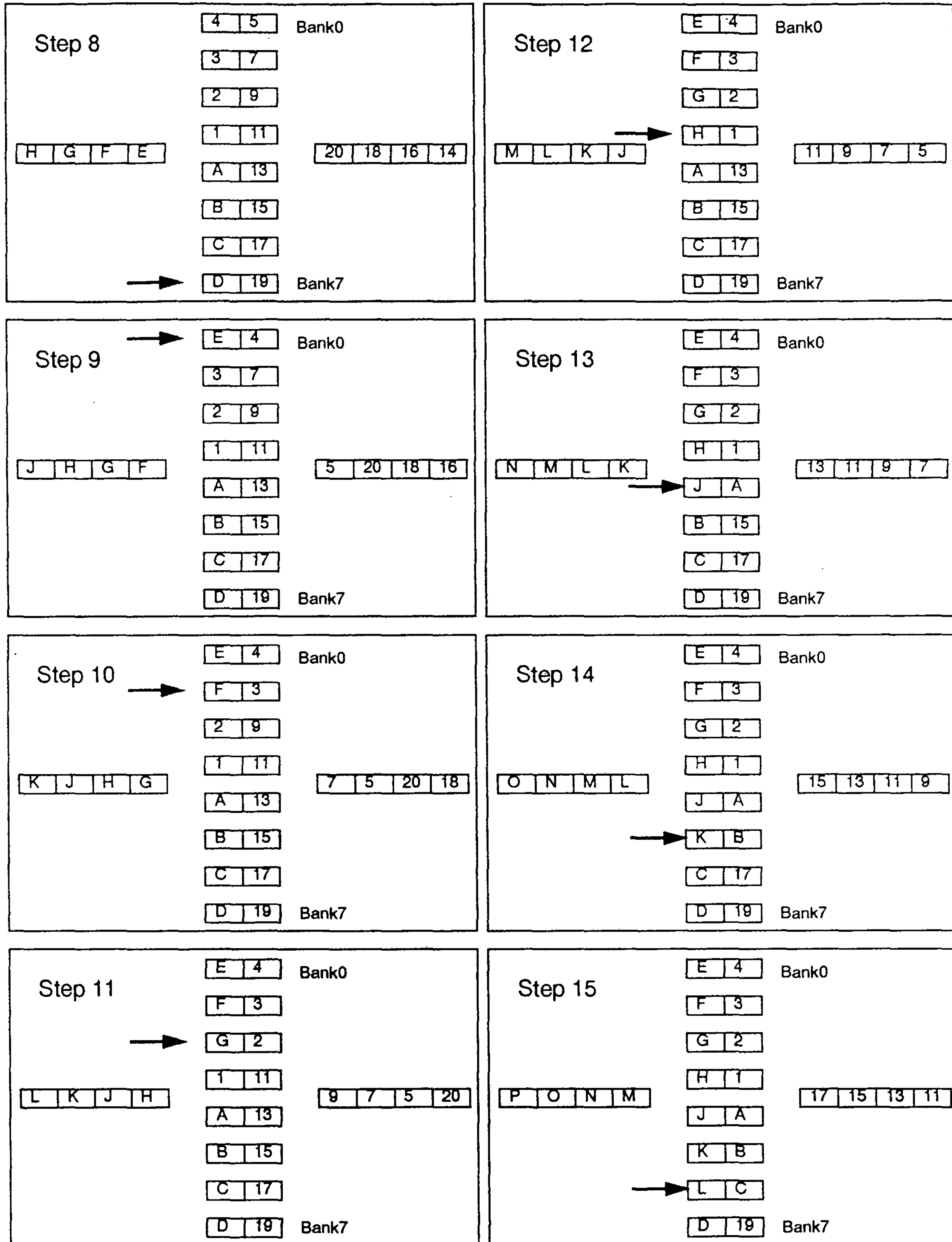


Figure B-1c NMC Scan with a Multiple of Eight Bits

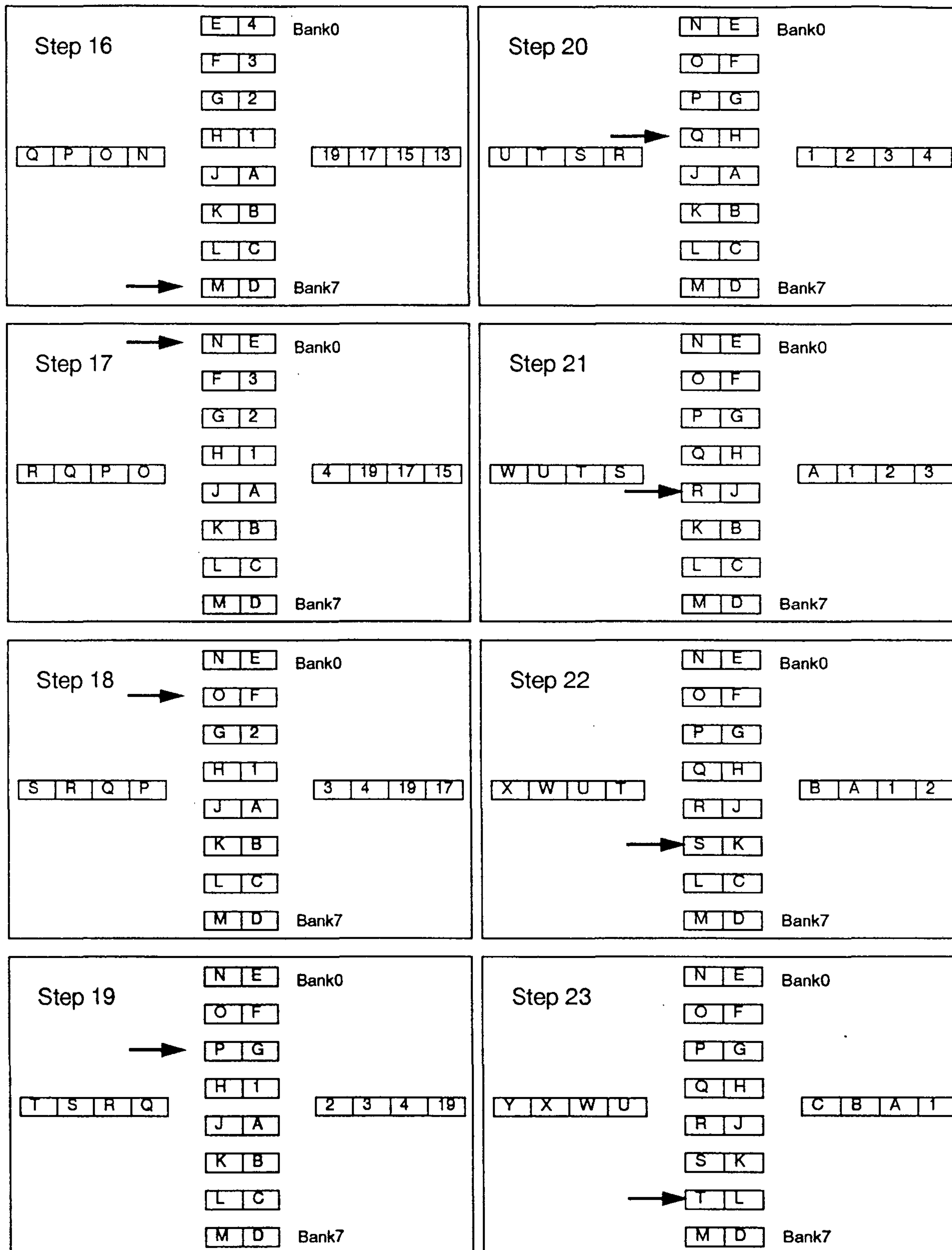
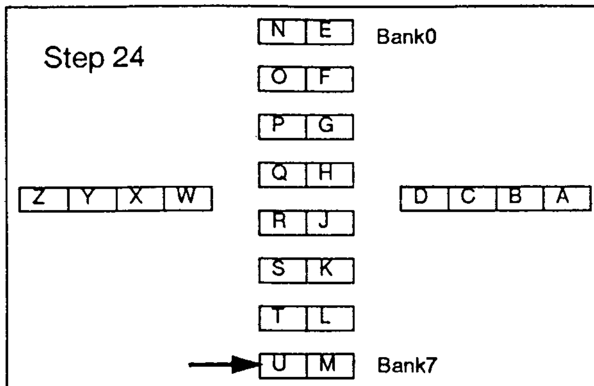
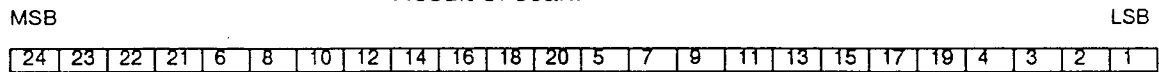


Figure B-1d NMC Scan with a Multiple of Eight Bits



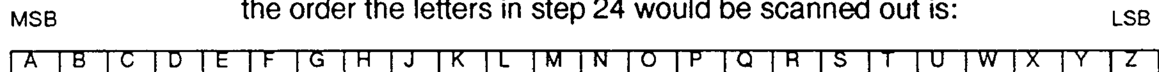
Result of scan:

MSB LSB



By matching letters in step 24 to numbers in the initial state,
the order the letters in step 24 would be scanned out is:

MSB LSB



For instance, the letter E in step 24 is located where the number 6 at the start is. Therefore, if the state at step 24 was scanned out as was the state at the start, the letter E would appear in the scan output in the same position as the number 6 did in the scan example.

Since the order the letters scan out is identical to the order they where scanned in, the scan ring is consistent when the total number of bits is a multiple of eight.

Figure B-2a NMC Scan without a Multiple of Eight Bits

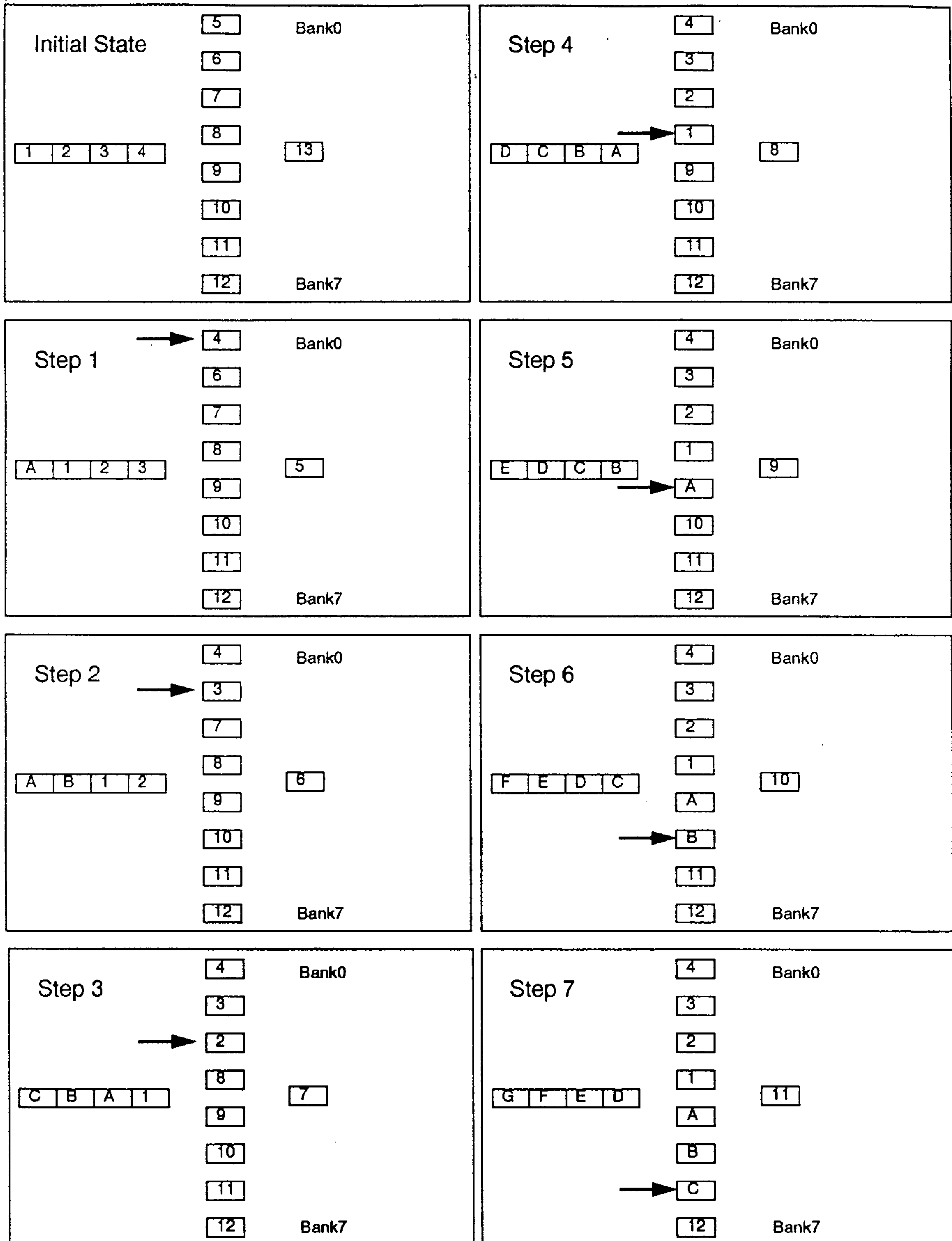
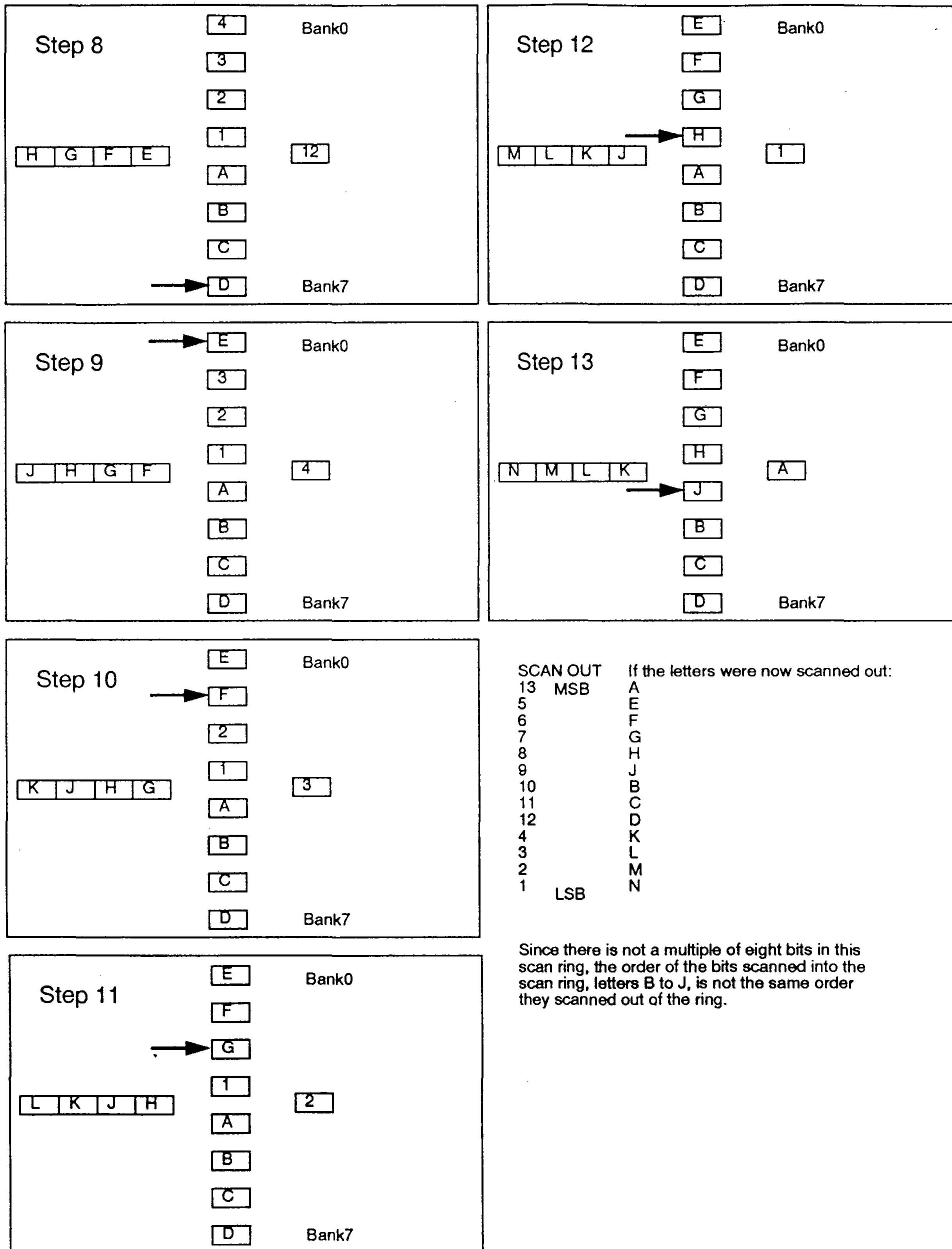


Figure B-2b NMC Scan without a Multiple of Eight Bits

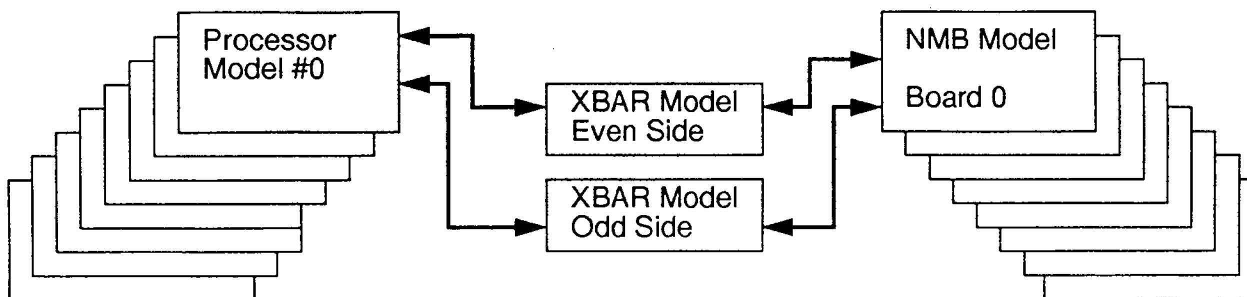


Appendix C ISP Models

C.1 Basics

The NMB was simulated using the ENDOT simulator. The system simulation could simulate from one to eight memory boards with from one to eight processor models. At the top level, the simulation was divided into three basic model types as is shown in Figure C-1.

Figure C-1 Top Level Organization



For NMB simulation, a special processor model was used. This model is described in a following section.

The XBAR model could be either a high level or low level model. The high level model consisted of a single ISP model which implemented most but not all of the crossbar functionality. The low level model was a detailed description of each of the boards in the crossbar and their gate arrays. The low level crossbar model will not be discussed here, consult the crossbar documentation for details.

The NMB model could also be either a simple, high level model or the detailed low level model. The high level model is a small ISP model useful when the NMB does not need to be modeled in detail. The low level model is the gate level description which is used for gate level simulation.

The models used most often for NMB simulation were the processor model, the high level crossbar model and the low level NMB model for one of the NMBs in the simulation with the remainder of the NMBs set to the high level model. High level models were used for three out of four of the NMBs because the low level model is very large and complex. Four low level description would have made simulation very slow.

Typically, four processors and four NMBs were simulated at a time.

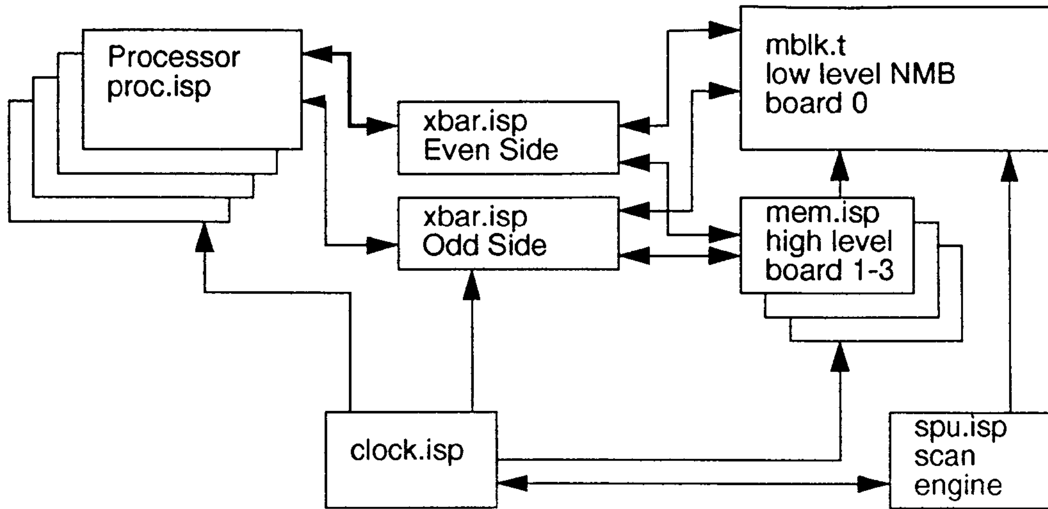
C.2 Topology Files

A topology file is the ENDOT file which determines how models and lower level topology files are hooked together. It can be thought of as a netlist for the simulation. It contains a list of all signals at that level of the topology file plus a series of processor instances. Each processor instance contains a list of that processor's inputs and outputs connected to the signals listed at the beginning of the file. By convention, topology files end in ".t" and are often called ".t" files.

The NMB simulation contains many nested topology files. The top level topology file is the sys.t file. It ties together the models listed in Figure C-1 plus a few support models.

Figure C-2 shows the sys.t file for the standard NMB simulation.

Figure C-2 Sys.t Topology File



The clock.isp model generates the clocks for all the models in the system. The spu.isp model generates the refreshes, scan signals and clock stop sequences for the memory board. It communicates with the clock.isp model when it is simulating clock halts.

Mblk.t is a topology file describing the detailed memory board model. It references a number of models and lower level topology files.

Some of the models in the sys.t file use special memory structures. Memories are very similar to arrays. They differ in how they are initialized and what happens to their contents when the program exits. Arrays are initialized to all zeroes and their contents are lost when the simulation is terminated. Memories are initialized from a particular data file associated with a particular instance of that memory. When the simulation terminates, the contents of the memory structure are written back into the file. Otherwise, the memory and the array act the same.

If a memory file does not exist, the simulator will detect an error and exit. The contents and creation of these files is discussed in the following sections. Table C-1 shows the memory files required for a simulation.

Table C-1 Memory Files

Name	Purpose
ucode.s	Used by the spu for scan simulation
p0_trace.out	Memory trace file for processor p0 Each processor must have a trace file.
.m1e.dat	Memory data file for high level memory model, board 1, even side. Each high level memory board must have an even file and an odd file.
.m0e.b0.dat	Memory data file for low level memory model, board 0, even side, bank 0. Each of the sixteen even and sixteen odd banks must have a memory file.

C.3 Simulation Directory

The NMB simulation was run in the /ncpu2/quattro/sys directory. This directory contains all the files necessary to simulate the NMB except for library models of the standard Eclipse and 100k parts. Table C-2 shows the important sub-directories in the sys directory.

Table C-2 Simulation Directories

Directory	Description
mem	both high level and low level models for memory board
misc	the spu and clock models
n2tr	the files necessary to trace signals during simulation
nmc	the low level description of the NMC boards
proc	the processor model
traces	processor micro-code files
xbar	the high level crossbar model

C.4 Proc.isp

The processor is a simple model which is not meant to look much like a real processor. All the model does is perform various memory operations to memory. For any operation that returns data, it checks to see that it received what it expected to receive.

The processor executes a simple "micro-code" which specifies a series of memory operations to make. This micro-code is the pX_trace.out file in the sys directory. These files are usually links into the traces directory.

The micro-code format is fairly simple. Each word of micro-code contains an address, a cycle type (the normal two bit cycle code), operand size (byte, halfword, word or longword), the amount of time to wait until the next request, the amount of time to wait for return data before halting the processor and a data field. The data field is either write data or expected data for reads. For test-and-modifies which both read and write, write data is assumed to be all ones (a TAS operation) and the data field is the expected data from memory.

The two time parameters allow the processor's memory operation to appear more like a real processor. A request every clock would indicate a vector or instruction fetch operation while gaps between requests might appear more like scalar information. Forcing the processor to wait for data if it is not received in a certain number of clocks reflects the fact that the processor can rarely keep processing while it is waiting for scalar data. On the other hand, vectors are meant to pipeline and therefore those requests which represent vector operations need not cause a wait in the processor.

The trace files can be created in one of two ways. The trgen utility creates one or more traces given a characterization as standard input. The mmk utility can also turn a file containing micro-code in mnemonic form into a trace file.

Trgen is found in /ncpu2/memsys/util. It reads a characterization from stdin of the type of requests and number of processors to make. The simplest way to make a new trace file is to copy an already existing file.

Table C-3 shows a sample trgen input for a single processor run.

Table C-3 Sample Input to trgen

```
test       ! root for trace file names, files will be test0.dat test1.dat
1          ! # of processors
1          ! # of memory boards
20000      ! number of references to generate for each processor
2000       ! size (in bytes) of block allocated to a processor
-1         ! number of clocks per block (-1 means never change blocks)
3.5        ! mean time between references
0          ! % refs to commreg
35         ! % commreg refs that are writes
30         ! % commreg tam refs
100        ! % refs that are scalar
5          ! % scalar refs that are NOPs
35         ! % scalar refs that are READs
50         ! % scalar refs that are writes
10         ! % scalar refs that are TAMs
50         ! % of scalar refs which are to previously referenced memory
40         ! % scalar refs of byte size
40         ! % scalar refs of halfword size
10         ! % scalar refs of word size
10         ! % scalar refs of longword size
30         ! % vector refs that are writes
43         ! % of vector refs which are to previously referenced memory
10         ! % vector refs of byte size
0          ! % vector refs of halfword size
50         ! % vector refs of word size
40         ! % vector refs of longword size
32 50      ! vector length, % vector refs of this length
8 50       ! vector length, % vector refs of this length
-1         ! -1 terminates vector length definitions (only 4 maximum)
8 100 ! vector stride, % vector refs of this stride
-1         ! -1 terminates vector stride definitions (only 4 maximum)
```

The number of references per processor determines the length of the trace file. Block size determines how large a region of memory the processor is allowed to access. At any one time, only one processor is allowed to access one block. The mean time between references specifies the mean time between requests for an exponential distribution.

The remaining values specify what percent of requests go to the communication registers, what the operand sizes are, vector length and stride for vector requests, etc. Scalar references are individual references which cause the processor to halt until the data is returned on read requests. Vector references are sequential accesses.

All appropriate percentages must sum to one hundred. Trgen exits with a failure otherwise. Trgen produces as many traces files as processors were specified. For the example file, the trace files are of the form test0.dat, test1.dat, etc.

Mmk is used to generate small trace files. It takes a file of the format in shown in Table C-4 and

produces a single trace file as output.

Table C-4 Mmk Trace File Generator Input

```
20009000 w b 0 0 ff
20009000 w w 0 0 87654321
20009000 r w 0 0 87654321
20009000 w w 0 0 12345678
20009000 r w 0 0 12345678
20009000 t b 0 0 ff
20009000 r w 0 0 ff345678
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
```

The first parameter is the address of the request. The second parameter is the opcode, "w" for write, "n" for no-op, "r" for read, and "t" for test-and-modify. The third item is the operand size, "b" for byte, "h" for half word, "w" for word and "l" for longword. Next is the number of clocks until the next request, then the number of clocks the processor will continue until waiting for return data from this request. The final parameter is the write data for writes or the read data for reads and test-and-modifies.

A zero in the address field signifies the end of a trace file. Several such lines are required for simulation purposes.

C.5 Clock.isp and Spu.isp

The clock model provides clocks to the entire system. It generates a free running, 2x clock to the NMB and "sys" clocks (clocks gated by run_sys) to the other models.

The spu model is used to provide refreshes, simulate stop/start operation (single step) and to simulate scan. The spu model has several internal state bits which can be set by the user to use the various modes.

Table C-5 Spu Model Mode Bits

Bit	Purpose
x.ref_on	Spu generates refreshes when one.
x.force_clocks_off	Turns processor and crossbar clocks off when one.
x.start_se	Start a scan when a soft error is detected.
x.start_he	Start a scan when a hard error is detected.
x.start_immed	Immediately start a scan.
x.stop_on	Clocks sys clocks to be randomly turned on and off to simulate single step operations.

Table C-5 shows the mode bits for the spu. Generally x.ref_on is left on so that refreshes go to the memory board and the refresh logic can be checked. Refreshes must be period and be timed with the restart of clocks if the spu is in run/stop mode.

The force clocks off mode is never used.

To turn on the stop/start mode, x.stop_on is set to a one. In this mode, the spu will assert clocks for a random amount of time and then halt the clocks. This simulates a user single stepping the system. The clocks are left off for a fixed amount of time to meet the minimum clock off requirements of the memory board.

The remaining bits control the scan simulation. If x.start_he is one and a soft error occurs or x.hard_he is one and a hard error occurs or x.start_immed is one, then the spu begins a scan operation.

The spu can perform LOG, SYS and ALL scans. The type of scan to perform is specified by the ucode.dat file, a memory file for the spu model. This file is generated by the mu utility from the ucode.s file, an ASCII specification of the scan operation.

Table C-6 Ucode Input File

```
log check 01 24 48 noextend
#log check fixed 3158 6316 noextend
all nocheck random 9472 9472 extend
all check random 9472 9472 noextend
run check 01 100 100 noextend
```

Table C-6 shows a sample input file for the mu utility. Lines beginning with a "#" are commented out. The first value specifies the type of scan, log, sys or all. A "run" means to issue normal mode clocks, no scan.

The second line specifies whether the result of the scan should be check (check) or unchecked (nocheck). If check is specified and the data scanned in is not what was scanned out, then the spu detects an error.

Parameter number three determines the type of pattern to scan into a board, random, fixed or alternating 0101 pattern (specify 01). The fixed pattern is a modulo thirty two pattern useful for tracking down breaks in the NMC logic. The next parameter is the length of the scan ring. After that is the number of clocks to scan. This is usually twice the length of the scan ring so that the scan engine scans in a pattern and then scans it back out again.

The last field can be either noextend or extend. Noextend causes the scan engine to put the board back in its normal state after a scan operation. Extend keeps the board in a scan operation. Extend can be used to model a board being scanned, then paused while in scan mode, then scanned again. The last two lines before the line beginning with "run" execute this type of operation. Note that the first line does not check the scan data since the data has just been scanned into the board but not yet scanned out.

Note: Many of the spu model mode bits have corresponding bits which must be set in other models in order for the mode to function properly. For instance, if refreshes are on, the DRAM model must also be told about them so that it knows to check for refreshes. Check the startup.mem.ll file for scripts which properly turn on these modes.

C.6 Xbar.isp

The xbar model provides a simple model of the crossbar. It uses a straight least frequently used arbitration scheme rather than the round robin scheme used by the real crossbar. It also does not allow a processor to burst its requests to a memory board. It is generally sufficient for testing the NMB, however.

C.7 Mem.isp

The mem.isp model is a simple, high level description of the NMB. It is found in the mem directory. It models refreshes and the basic memory operations (read, write, test and modify).

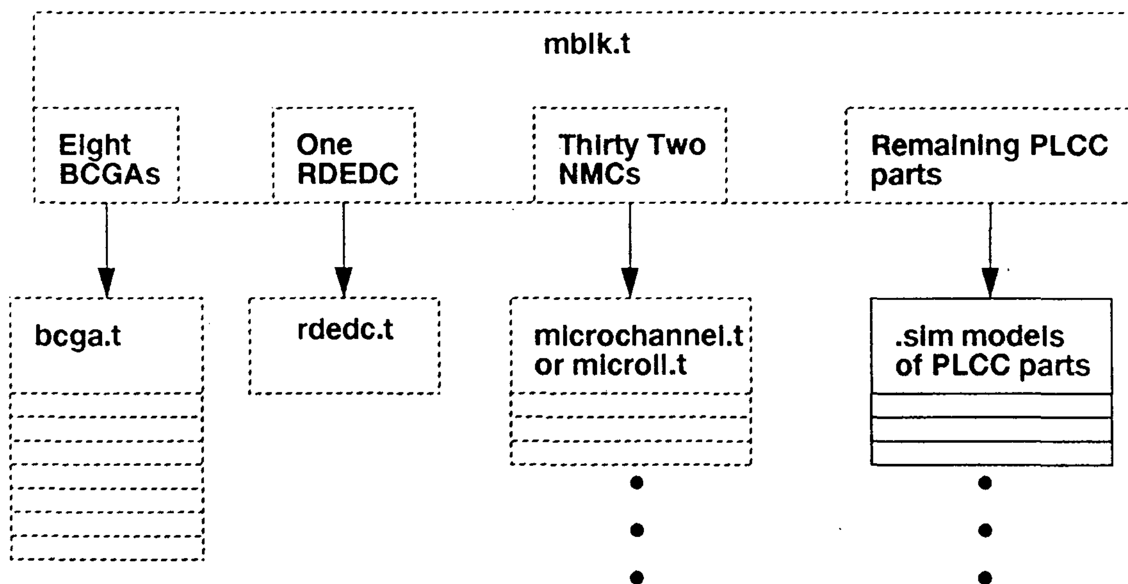
C.8 Low Level Memory Models

The low level memory board model consists of the mblk.t topology file and all the sub-topology files it calls. Mblk.t is generated from the cmpexp.dat file for the NMB using v2e (valid to endot). This tool creates a topology file from a compiler expansion file. A sed script in the mem directory is run over the file that v2e produces to produce a real mblk.t This sed script is called mblk.sed.

Mblk.t contains primitives for the ECLIPSE and 100K logic plus topology files for the NMCs, the RDEDC and the BCGA. An NMC can be modeled either using the microchannel.t file or the microll.t file. The first .t file is a hand generated version of the NMC topology using 4bit DRAM models. The second version is generated from the NMC cmpexp.dat file in the nmc directory. It models the actual NMC circuit but is more complex and is slower. The NMC is relatively simple and need not be simulated unless the NMC has been altered. The selection between microchannel.t and microll.t is made in the mblk.sed file.

Figure C-3 shows the topology of the mblk.t file.

Figure C-3 Mblk.t



The bcga.t is shown in Figure C-4. There are four bank controllers in each BCGA with an sst (single step) logic block associated with each bank controller. The remaining logic provides clock and scan control. This topology partition corresponds to partitions in the BCGA schematics.

The "new" script in the sys directory can be used to update the compiler expansion for the memory board used to generate mblk.t. In order to save space, this cmpexp.dat is compressed in the mem directory. It is also renamed to cmpexp.mblk. The user may manually compress and rename this file or may type "new nmb." The script will compress and rename the cmpexp.dat file found in /ncpu2/quattro/valid/nmb/schem, the directory where compiles for simulation should be done.

Figure C-4 Bcga.t

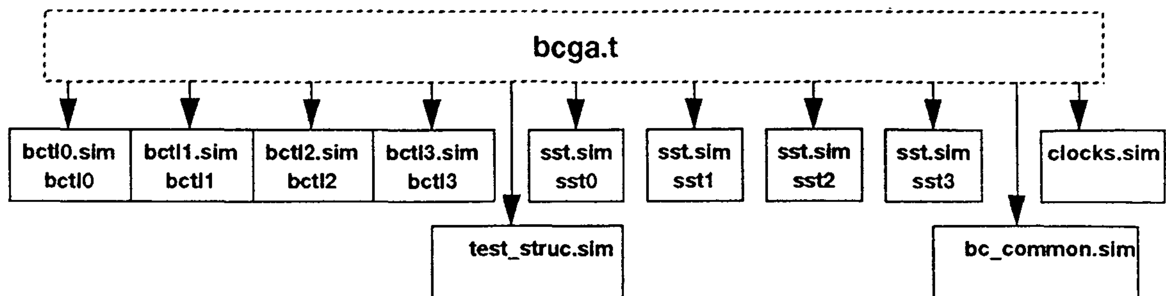


Figure C-5 shows the topology for the RDEDC gate array. The rdedc.sim model is a single model replicated for both the even and odd sides.

Figure C-5 Rdedc.t

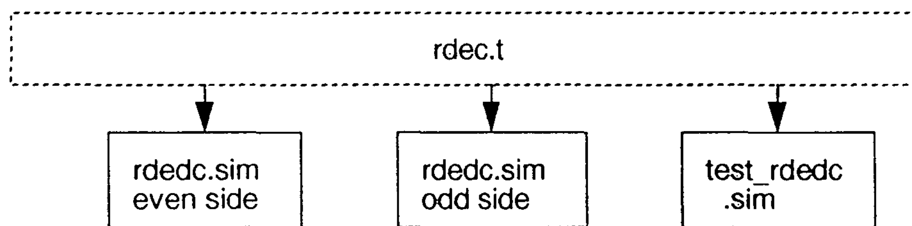
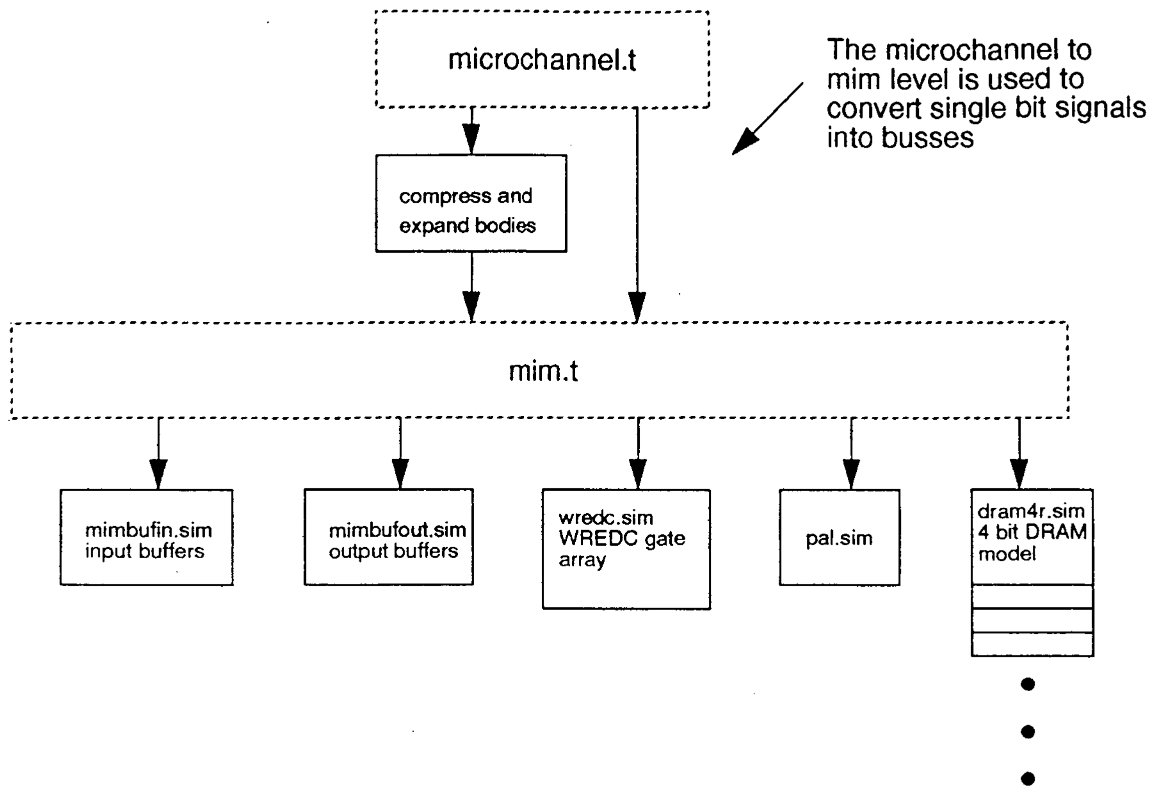


Figure C-6 shows the topology of the microchannel.t file. Microchannel.t contains primitives for converting collections of single bit signals into multi-bit busses. It then calls out mim.t which deals

with bussed signals.

Figure C-6 Microchannel.t



C.9 Memory Files

Both the high level and low level memory models use memory files which represent the contents of memory. These files must be initialized for two reasons. First, proper ECC must be set up in memory otherwise the NMB will detect a hard error. Second, the trgen program used to generate the traces for the proc.isp models assumes memory is filled with a certain pattern. If it is not, the processor models will not receive what they expect from memory and will also cause an error.

The memory files are initialized by the initmem script. This script calls various utilities to generate both the high level and low level models. The script is called with two parameters. The first one specifies the number of memory boards in the system. The second parameter is optional. If it is the letter "l," then the script generates low level memory files for NMB zero, rather than the high level files. Every high level memory board has two memory files. Every low level board has thirty two memory files. The files begin with a dot and therefore are not normally seen when an "ls" is done on the directory.

C.10 Makefile

The entire memory simulation can be readied for simulation by typing "make" in the sys directory. This make file will do a make in all the sub directories, ecologize the entire system, update the memory files for the high and low level NMBs and update and trace files. There should be no errors reported during a make. Warnings about time delays may be ignored.

C.11 Startup Files

The sys directory contains files that should be run when the simulation begins. The startup.n2 file is automatically executed by the simulator at time zero of the simulation. It is set up to execute the startup.comm file and the startup.mem.ll file.

Startup.n2 defines some simple macros such as the log macro for copying output to a file or the "r" macro which is short hand for run. It also deposits a few values to the clock model so that the clock model knows how many processors are active so that it knows when to halt the simulation.

The ENDOT simulator requires all memories used by a simulation to be defined with the "bind" command. The bind command tells ENDOT how to treat memories, either as serial files or random access memories. Unbound memories will not function properly. The startup.n2 file binds all the memories used in a simulation except for the low level model memories. These are bound in startup.mem.ll.

The startup.comm file defines some print functions for printing the state of the memory system. A trigger can be set up that causes this function to be executed each clock but this trigger is usually disabled. This file also sets up some of the error checking commands.

The startup.mem.ll file does a lot of initialization for the low level memory models. The memory system will not function properly without it. It also defines some macros for examining the state of the low level simulation.

The memory files for the low level memory simulation are bound in the startup.mem.ll file. There are two bind commands for each bank. Only one should be active at one time, the other should be commented. One bind command is used if the microchannel.t topology file is used for a bank. The other one is used for the microll.t topology file.

At the end of the startup.mem.ll file, a number of macros can be called. These macros are used to turn on refreshes, cause the DRAMs to generate single bit errors, rotate address bits to exercise all the address lines, or set single step testing. Commenting these lines out prevents these modes from being exercised. Generally, all modes should be run except the single step mode.

C.12 Running the Simulation

The memory simulation is self running. After typing "run," the processor models will begin making request to the memory boards. The processors make requests until their trace file is exhausted after which they assert a signal to the clock model telling that model that it is finished. The clock model will then halt the simulation, either after all processors have finished or after any one processor has finished depending on a mode bit in the clock model.

While the simulation is running, various models will perform error checking. Any hard error detected in the system, parity errors at the processor models or any of the NMB hard errors, will cause the system to halt.

Errors will also be detected if a processor receives data that it did not expect or if the DRAMs are operated incorrectly. When refreshes are enabled, the DRAMs check to make sure that they receive periodic refreshes with sequential addresses. The DRAMs also check to see that the RAS, CAS, G and WE signals are properly asserted.

When an error is detected, the model detecting the error will execute a notify command with a particular numeric value. As long as the "alert notify stop global" command has been executed in the startup.comm file, the simulator will see the notify and stop the simulation. The code can then be looked up in the .isp for the model that detected the error and the error can be isolated.

Note: It is very important to be sure that notifies are detected by the simulator. If notifies are disabled, the simulation will run without error but it will be impossible to determine whether any errors occurred.

It should be noted that the microll.t models for the NMCs significantly decrease simulation speed. They should only be used when necessary.

C.13 Simulating Scan

The spu model can be used to simulate all three scan modes. The simple software for the spu model which tells the spu what type of scan and how long to scan has already been described in section C.5 on page 219.

The scan must be started by some event. Usually a soft error or a hard error is used to start the scan although it can also be forced to start immediately. The actual scan performed is determined by the ucode.dat file not by the type of event which caused the scan.

For the selected type of scan, the spu performs all the scan control, run bit and clock actions to model the real spu's scan operation. The spu model does not yet model the six clock pipeline in the XCL registers.

After the spu has scanned for the length of the scan ring, it prints the contents of its scan registers so that the contents of the scan ring can be examined. After the scan is finished and if the "check" flag was set for the scan, the spu checks to make sure that the data scanned out of the board matches the data scanned in. It must be realized that the board must be scanned for twice the scan ring length in order for the data to go into the board and then back out.

If a mismatch is detected on scan data, the spu asserts a signal indicating the mismatch. A simulation monitor on that signal will cause the simulation to stop.

Once scan is finished, the spu asserts another signal indicating the end of scan. A simulation monitor on that signal will also cause the simulation to stop.

Note: Because the NMCs scan in eight groups of four, any scan pattern of modulo two, four or eight bits is a poor choice for testing the scan ring logic. With such a pattern it may be impossible to detect a break in the NMC scan logic. The random pattern should always be used as a final verification of the scan ring. The "fixed" pattern which is a modulo thirty two pattern is suitable for debugging the scan ring. It is a recognizable pattern yet still exercises all the NMCs.

Note: ALL ring scan destroys the contents of the DRAMs and causes the DRAM control lines to change non-sensically since these control lines are driven from registers in the all ring. During ALL ring scan testing, the alert for notifies should be disabled and the monitor on hard errors should be turned off since these reports will be meaningless.

C.14 Trace Files

Trace files were used in NMB simulation to record the history of selected signals on the NMB. Signals to be traced are specified in a .cmd file in the n2tr directory. The build script in the n2tr directory is then executed on the .cmd file to produce a .inc file which must be included into the simulation in order to trace files and a .bnd file which is used when the trace is formatted for viewing.

Most of the .cmd files in the n2tr directory are automatically updated by the makefile. The startup.n2 file already has include commands for these files which are commented out when not needed. Trace files take a lot of disk space so they should not be used unless needed.

Once a simulation is completed, the signal history for the trace files are written to the sys.d file. The "fv" script in the sys directory can be used to format and view this trace file. Fv takes one to four parameters. The first parameter is the name of the command file to format (the .cmd file without the .cmd). The second and subsequent parameters are optional. They can be used to specify, respectively, the starting time of the format, the interval between samples, and the ending time of the format. The default is one sample per clock (100 time units) with samples starting at 90. Since the NMB has some 2x logic on it, an fv command starting at 40, every 50 time units is often useful.

C.15 ISP Conventions

Certain basic conventions were followed in the isp models.

Table C-7 Signal Naming Conventions

Prefix	Meaning
r.	A register clocked by a 1x clock
r2.	A register clocked by a 2x clock
c.	A combinatorial signal.
p.	A shadow register for an output port.
l.	A latch for simulation deskew.
l2.	A deskew latch for a 2x register.
x.	A special simulation control signal which has no real logic analog.

Table C-7 lists the conventions for naming signals. The first three are self explanatory. The "p." signals are used to improve simulation performance. The ENDOT simulator only evaluates a simulation body when that body's inputs are written. If the inputs are not written, the body is not executed. The "p." signals are used to check the value of a body's output lines. If the value is the same as what the body wants to write to its output, it does not do the write. By avoiding writing what is already on a signal line, the needless execution of the simulation body which receives the signal is avoided.

The "l." and "l2." signals are used in the same way that real hardware latches are used. When simulation registers are clocked by different clocks, there are times when one register changes before the other and a race condition can occur if the two registers are connected one to the other. The "l." signal is used as an intermediate signal functioning as a latch in order to avoid the race condition. The "l." signal does not represent a physical latch. It exists only to avoid a simulation race condition.

The "x." signals are used by various simulation models to control the simulator. For instance, certain "x." signals such as x.ref_on in the DRAM models, determine whether the simulation will attempt to model refreshes. These signals are usually set in the startup files to force different operations.

C.16 Testing the NMB

A complete test of the NMB consists of running many trgen patterns through the NMB during normal mode in order to check out normal operation of the NMB. Following normal operation, the NMB should be simulated in run/stop mode to test its ability to single step. All the scan modes should be simulated using random data. The LOG scan should be executed while the board is performing requests to make sure the log ring operations do not interfere with memory operations.

For all of these tests except for ALL ring scan, refreshes should be turned on.

Appendix D Related Documents

Document Number	Description
182-000181-000	BCGA functional specification.
182-000145-000	RDEDC functional specification.
182-000173-000	WREDC functional specification.
500-001252-000	C38XX Crossbar / Memory System Block Diagram
416-001246-001	C38XX Memory Board / Card Block Diagram