EEEEEE CCCC 0000 E C C 0 0 EEEE C 0 0 E C C 0 0 EEEEEE CCCC 0000	ECO No: EL00057-00-CT003 d i g i t a l TM ENGINEERING CHANGE ORDER

Orig.: Richard Best
Phone: 293-5351
Loc.: BXB1-1
Cost Center No.: 31L

Sheet: 1 of 2
Date Received: 24-Oct-1988
MS: E11
Final Issue: 10-Feb-1989

Unit(s) to be changed: EL-00057-00

Prod. Families Affected: N/A

ECO Type Code: 5 Category: Standard FCC Req.: No
F/S Affected: No FCO Req.: No LOU Code: N/A
Where Used: N/A

Documentation Affected: A-DS-EL00057-00-0, DEC STD 057-0 VAXBI STANDARD

PROBLEM Users had raised questions in several areas.

CORRECTION Update standard per sheet 2 of this document.

BREAK-IN/ EFFECTIVITY Immediately.

APPROVAL NAMES		C/C	Phone No. DTN-Ext.	Date dd-mmm-yyyy
Engineering:	Richard P. Best Richard Best	31L	293-5351	10-Feb-1989
Standards Process Manager:	Eric A. Williams Eric Williams	396	287-3696	10-Feb-1989
Coordinator:	Donald J. Mehaffey Don Mehaffey	3VQ	287-3675	10-Feb-1989

Charge Number: 098-09720 W.O. No: 192659

ECO19-21

EEEEEE CCCC 0000 E C C 0 0 EEEE C 0 0 E C C 0 0 EEEEEE CCCC 0000	ECO No: EL00057-00-CT003 d i g i t a l TM ENGINEERING CHANGE ORDER

CONTINUATION SHEET

ITEM NO.	PART NUMBER	DOCUMENT NUMBER	OLD REV	NEW REV
1	EL-00057-00	A-DS-EL00057-00-0	D	E

DESCRIPTION: Application Note 10 was added to define the use of the Drev Field of Dtype Register. This section includes a table that presents the preferred pairing of revision marking letters and machine-readable VAXBI revision fields.

Clarifications and additional notes were added to the following subheads:

6.2.2.2 and 6.2.3

7.1, 7.2, 7.3, and 7.16

11.1.2, 11.1.6, and 11.1.7

13.1.1

15.5.1.1 and 15.7.3

D.1, D.3, D.8, D.13, D.14, and D.15

DEC STD 057-0 VAXBI STANDARD

DOCUMENT IDENTIFIER: A-DS-EL00057-00-0 Rev E, 10-Feb-1989

ABSTRACT: This standard defines the VAXBI interconnect system. The document provides a complete description of the bus protocol, architectural requirements, and mechanical and electrical requirements.

APPLICABILITY: This standard applies to all nodes and systems using the VAXBI bus.

STATUS: APPROVED 10-Feb-1989; see EL-INDEX-00 for expiration date.

This document is confidential and proprietary. It is an unpublished work protected under the Federal copyright laws.

Copyright (c) Digital Equipment Corporation. 1985, 1986, 1987, 1989. All rights reserved.

This document must not be provided to customers unless they hold a VAXBI license. It must not be provided to vendors.

digital™

TITLE: DEC STD 057-0 VAXBI STANDARD

DOCUMENT IDENTIFIER: A-DS-EL00057-00-0 Rev E, 10-Feb-1989

REVISION HISTORY: Initial release at Rev B, 12-Sep-1985
Rev C, 28-Feb-1986 ECO# LJ001
Rev D, 16-Jul-1987 ECO# LJ002
Rev E, 10-Feb-1989 ECO# CT003

FILES: Published File - EAGLE1::pub$:[biarch.current]s57.mem
Source Files - EAGLE1::pub$:[biarch.current]m379.rno

Document Management Group: MSB Systems Architecture (SHV)
Responsible Department: MSB Systems Architecture
Responsible Person: Richard D. Best
Authors: Richard D. Best, Frank Bomba
Bob Chen, Rick Gillett

APPROVAL: This document has been reviewed and recommended for approval by the General Review Group for its category, which consists of the BI Architecture Review Group.

Richard D. Best

Richard D. Best, Standard Owner

Eric A Williams

Eric Williams, Standards Process Manager

Direct requests for further information to:

Richard Best
BXB1-1/E11

Copies of this document can be ordered from:

Standards and Methods Control
CTS1-2/D4, DTN: 287-3724, JOKUR::SMC

Please provide your name, badge number, cost center, mailstop, and ENET node when ordering.

digital™

1 INTRODUCTION

Footnotes may indicate why certain decisions were made, describe unresolved issues, or provide hardware implementation guidelines. This information is for Digital use only.

1.1 PURPOSE

The purpose of this standard is to define the architecture and requirements for the VAXBI bus so that all hardware design and implementation is compatible and so that software can be written for conforming hardware.

1.2 SCOPE

This standard applies to all nodes and systems using the VAXBI bus.

1.3 RESPONSIBILITIES

1.3.1 MSB Systems Architecture

It is the responsibility of MSB Systems Architecture to:

- Be responsible for the technical accuracy of the specifications.
- Review change requests to the document and accept or reject those requests.
- Grant exceptions as necessary to options that do not conform to VAXBI requirements.

1.3.2 VAXBI Certification Group

It is the responsibility of the VAXBI Certification Group (of Mid-Range Systems Evaluation Engineering) to:

- Certify that options designed to be incorporated into a VAXBI system meet the stated requirements.

2 ECO PROCESS FOR DEC STD 057 VAXBI STANDARD

For information on this see Appendix F of the the VAXBI standard.

CONTENTS

PREFACE

CHAPTER 1 OVERVIEW OF THE VAXBI BUS AND THE BIIC

1.1 DIGITAL'S COMMITMENT TO FUTURE COMPUTING NEEDS . . 1-1
1.2 DESCRIPTION OF THE VAXBI BUS 1-2
1.2.1 Addressing Capability 1-3
1.2.2 Peak Transfer Rate 1-4
1.2.3 VAXBI Signals 1-5
1.2.4 VAXBI Bus Features 1-6
1.3 TRANSACTIONS 1-6
1.4 THE BIIC 1-9
1.5 DESCRIPTION OF THE BCI 1-10
1.5.1 How the BCI Relates to the VAXBI Bus 1-10
1.5.2 BCI Signals 1-11
1.6 SYSTEM CONFIGURATIONS 1-11
1.6.1 Low-End System Configurations 1-12
1.6.2 Multiprocessor System Configurations 1-13
1.6.3 Midrange and High-End System Configurations . 1-15
1.6.4 Clusters and Networking 1-18

CHAPTER 2 VAXBI ADDRESS SPACE

2.1 ALLOCATION OF MEMORY SPACE 2-1
2.2 ALLOCATION OF I/O SPACE 2-2
2.2.1 Nodespace 2-5
2.2.1.1 BIIC CSR Space 2-5
2.2.1.2 User Interface CSR Space 2-5
2.2.2 Multicast Space 2-6
2.2.3 Node Private Space 2-6
2.2.4 Node Window Space 2-6
2.2.5 Assignable Window Space 2-7
2.3 BIIC RESTRICTIONS 2-8

CHAPTER 3 VAXBI PROTOCOL AND CYCLE TYPES

3.1 NODE IDENTIFICATION 3-1
3.2 ARBITRATION 3-1
3.2.1 Arbitration Modes 3-2
3.2.1.1 Setting the Mode 3-2
3.2.1.2 Two Priority Levels 3-3
3.2.1.3 Dual round-robin 3-3
3.2.1.4 Fixed-Low Priority 3-3
3.2.1.5 Fixed-High Priority 3-4
3.2.1.6 Restricted Use of Arbitration Modes 3-4
3.3 TRANSACTION CYCLES 3-4
3.3.1 Command/Address Cycle 3-4
3.3.2 Imbedded Arbitration Cycle 3-5
3.3.3 Data Cycles 3-5

CHAPTER 4 VAXBI SIGNALS

4.1 DATA PATH SIGNALS 4-4
4.1.1 BI D<31:0> L 4-4
4.1.2 BI I<3:0> L 4-4
4.1.3 BI P0 L 4-4
4.2 SYNCHRONOUS CONTROL SIGNALS 4-4
4.2.1 BI NO ARB L (No Arbitration) 4-5
4.2.2 BI BSY L (Busy) 4-6
4.2.3 Use of BI NO ARB L and BI BSY L 4-7
4.2.3.1 Arbitration State 4-8
4.2.3.2 Loopback Transactions 4-9
4.2.4 BI CNF<2:0> L (Confirmation) 4-10
4.2.4.1 Command Responses 4-11
4.2.4.2 Data Responses 4-12
4.3 CLOCK SIGNALS 4-16
4.3.1 BI TIME + and BI TIME - 4-16
4.3.2 BI PHASE + and BI PHASE - 4-16
4.4 ASYNCHRONOUS CONTROL SIGNALS 4-16
4.4.1 BI AC LO L 4-17
4.4.2 BI DC LO L 4-17
4.4.3 BI RESET L 4-17
4.4.4 BI STF L (Self-Test Fast) 4-18
4.4.5 BI BAD L 4-18
4.4.6 BI SPARE L 4-18

CHAPTER 5 VAXBI TRANSACTIONS

5.1 SINGLE-RESPONDER AND MULTI-RESPONDER TRANSACTIONS 5-1
5.2 INTERPROCESSOR COMMUNICATION 5-2
5.2.1 IPINTR Transactions 5-3
5.2.2 VAXBI Requirements for Interlock Transactions . 5-3
5.3 TRANSACTIONS TO SUPPORT DATA TRANSFER 5-5
5.3.1 Address Interpretation 5-5
5.3.1.1 Read-Type Transactions 5-7
5.3.1.2 Write-Type Transactions 5-8
5.3.2 Cacheing and the VAXBI Bus 5-9
5.3.3 Write Mask 5-13
5.3.4 Read Status 5-14
5.3.5 Write-Type Transactions 5-15
WRITE 5-15
WCI (Write with Cache Intent) 5-17
WMCI (Write Mask with Cache Intent) 5-18
UWMCI (Unlock Write Mask with Cache Intent) . . 5-20
5.3.6 Read-Type Transactions 5-21
READ 5-21
RCI (Read with Cache Intent) 5-23
IRCI (Interlock Read with Cache Intent) 5-24
5.3.7 Invalidate Transaction 5-24
INVAL (Invalidate) 5-24
5.4 TRANSACTIONS TO SUPPORT INTERRUPTS 5-27
5.4.1 Device Interrupts 5-27
INTR (Interrupt) 5-29
IDENT (Identify) 5-32

5.4.2 VAXBI Interrupt Vectors 5-35
5.4.3 Interprocessor Interrupts 5-37
IPINTR (Interprocessor Interrupt) 5-37
5.5 TRANSACTION TO SUPPORT DIAGNOSTICS 5-39
STOP 5-39

CHAPTER 6 INITIALIZATION

6.1 VAXBI NODE INITIALIZATION REQUIREMENTS 6-2
6.2 INITIALIZATION MECHANISMS 6-2
6.2.1 Power-Down/Power-Up 6-3
6.2.2 System Reset 6-4
6.2.2.1 Reset Module Requirements 6-4
| 6.2.2.2 Use of System Reset for Down-line Loading Software 6-5
| 6.2.3 Node Reset 6-6
6.3 BI AC LO L 6-8
6.4 BI DC LO L 6-9
6.5 BI RESET L 6-11
6.6 INITIALIZATION TIMING SEQUENCES 6-12
6.6.1 Power-Down/Power-Up Sequence 6-12
6.6.2 System Reset Sequence 6-12

CHAPTER 7 VAXBI REGISTERS

| 7.1 DEVICE REGISTER 7-4
| 7.2 VAXBI CONTROL AND STATUS REGISTER 7-5
| 7.3 BUS ERROR REGISTER 7-9
7.4 ERROR INTERRUPT CONTROL REGISTER 7-14
7.5 INTR DESTINATION REGISTER 7-16
7.6 IPINTR MASK REGISTER 7-17
7.7 FORCE-BIT IPINTR/STOP DESTINATION REGISTER . . 7-18
7.8 IPINTR SOURCE REGISTER 7-19
7.9 STARTING ADDRESS REGISTER 7-20
7.10 ENDING ADDRESS REGISTER 7-21
7.11 BCI CONTROL AND STATUS REGISTER 7-22
7.12 WRITE STATUS REGISTER 7-26
7.13 FORCE-BIT IPINTR/STOP COMMAND REGISTER 7-27
7.14 USER INTERFACE INTERRUPT CONTROL REGISTER . . . 7-28
7.15 GENERAL PURPOSE REGISTERS 7-31
| 7.16 SLAVE-ONLY STATUS REGISTER 7-32
7.17 RECEIVE CONSOLE DATA REGISTER 7-33

CHAPTER 8 CLASSES OF VAXBI NODES

8.1 PROCESSORS, MEMORIES, AND ADAPTERS 8-1
8.1.1 Processors 8-2
8.1.2 Memories 8-3
8.1.3 Adapters 8-3
8.2 REQUIREMENTS ON NODES OF EACH CLASS 8-3
8.2.1 Required Sets of Transactions 8-3
8.2.2 Requirements by Node Class 8-5

8.2.3 Requirements in I/O Space 8-8
8.2.3.1 Read Side Effects 8-8
8.2.3.2 Cacheing in I/O Space 8-9
8.2.3.3 Read-Type Value Equivalence 8-9
8.2.3.4 Write-Type Reaction Equivalence 8-9
8.2.3.5 Longword Data Length Transactions 8-9
8.2.3.6 Quadword and Octaword Data Length Transactions 8-9
8.2.3.7 Locks in I/O Space 8-9
8.2.3.8 Write Masks in I/O Space 8-9
8.2.3.9 Translations of VAXBI Transactions to and from the UNIBUS 8-10
8.2.3.10 Rationale for I/O Space Requirements 8-10
8.3 THE VAXBI AS AN I/O BUS 8-10

CHAPTER 9 VAXBI CONSOLE PROTOCOL

9.1 MASTER CONSOLE 9-1
9.2 RECEIVE CONSOLE DATA REGISTER (RXCD) 9-2
9.3 CONSOLE COMMUNICATIONS 9-3
9.4 THE Z CONSOLE COMMAND 9-5

CHAPTER 10 PERFORMANCE

10.1 VAXBI BANDWIDTH 10-1
10.2 LATENCY 10-3
10.2.1 Extension Cycles 10-3
10.2.2 Effect of Arbitration Modes on Bus Latency . . 10-4
10.2.3 Dual Round-Robin Mode Behavior 10-7
10.2.4 Dual Round-Robin Mode Latency 10-9
10.2.5 Fixed-Low Priority and Dual Round-Robin . . . 10-10
10.2.6 Fixed-High Priority and Dual Round-Robin . . . 10-11
10.2.7 One Fixed-High Priority Node 10-13

CHAPTER 11 ERROR DETECTION AND MAINTAINABILITY

11.1 SELF-TEST OPERATION 11-1
11.1.1 Self-Test Requirements 11-2
| 11.1.2 Self-Test Operation with a BIIC 11-3
11.1.3 Location of the Broke Bit 11-6
11.1.3.1 VAXBI Control and Status Register 11-6
11.1.3.2 Slave-Only Status Register 11-6
11.1.4 Using the BI BAD L Line 11-6
11.1.5 Self-Test Time Limits 11-6
| 11.1.6 Using the VAXBI Bus During Self-Test 11-8
| 11.1.7 Device Type Requirements 11-9
11.2 ERROR DETECTION AND RESPONSE 11-11
11.2.1 Error Detection 11-11
11.2.1.1 Parity Checking 11-11
11.2.1.2 Transmit Check Error Detection 11-12
11.2.1.3 Protocol Checking 11-13
11.2.2 VAXBI Primary Interface Abort Conditions . . . 11-13
11.2.3 Response to Exception Conditions 11-14

11.3 USE OF THE STOP TRANSACTION 11-16

CHAPTER 12 ELECTRICAL SPECIFICATION

12.1 TEST CONDITIONS 12-1
12.2 CLOCK SIGNAL TIMING 12-1
12.2.1 BI TIME +/- Signals 12-1
12.2.2 BI PHASE +/- Signals 12-2
12.2.3 Clock Skew 12-2
12.2.4 Clock Signal Integrity 12-2
12.2.5 Asynchronous Control Signal Timing 12-5
12.3 INTERCONNECT ELECTRICAL CHARACTERISTICS 12-6
12.3.1 Maximum Capacitance Requirements 12-6
12.3.1.1 Data Path and Synchronous Control Lines . . 12-6
12.3.1.2 Asynchronous Control Lines 12-6
12.3.2 VAXBI Backplane Requirements 12-6
12.3.3 VAXBI Extension Cable Requirements 12-7
12.4 DATA PATH AND SYNCHRONOUS CONTROL LINE TRANSCEIVERS 12-7
12.5 ASYNCHRONOUS CONTROL LINE DRIVERS 12-7
12.6 ASYNCHRONOUS CONTROL LINE RECEIVERS 12-7
12.7 CLOCK LINE DRIVER OUTPUT CHARACTERISTICS 12-7
12.8 CLOCK LINE RECEIVER INPUT CHARACTERISTICS . . . 12-7
12.9 VAXBI TERMINATION SCHEME 12-8
12.9.1 Data Path, Synchronous Control, and BI AC LO L and BI DC LO L Signals 12-8
12.9.2 Termination of BI TIME +/- and BI PHASE +/- Clocks 12-8
12.10 MAXIMUM CONFIGURATION 12-8
12.11 INTERFACING AND CONFIGURATION RULES 12-8
12.12 WORST-CASE VAXBI BUS TIMING 12-9
12.12.1 Time Reference Standards 12-9
12.12.2 Timing Parameters Used for Worst-Case Calculations 12-10
12.12.3 Worst-Case VAXBI Setup Time Calculations . . . 12-10
12.12.3.1 Worst-Case VAXBI Hold Time Calculations . . 12-12
12.12.4 Transmission Line Considerations 12-14
12.13 WORST-CASE VAXBI DATA PATH RESISTANCE 12-14

CHAPTER 13 MECHANICAL AND POWER SPECIFICATION

13.1 VAXBI MODULE GENERAL SPECIFICATION 13-2
13.1.1 Physical Requirements 13-2
13.1.2 Border and ESD Requirements 13-4
13.1.3 LED Requirements 13-4
13.1.4 Replaceability Requirements 13-5
13.1.5 VAXBI Backplane Pins and Signals 13-5
13.1.6 VAXBI Voltages and Currents 13-7
13.1.6.1 Voltages Available 13-7
13.1.6.2 Specification of Current and Power for VAXBI Modules 13-8
13.1.7 Worst-Case Current Limits 13-9
13.1.8 I-R Drop and Voltage Tolerance Budgets 13-10

13.1.9 Power Dissipation and Cooling 13-11
13.1.9.1 Power Dissipation Limits 13-11
13.1.9.2 Air Flow 13-12
13.2 VAXBI MODULE WITH A BIIC 13-12
13.2.1 VAXBI Corner Signal Locations 13-13
13.2.2 VAXBI Virtual Connector Signals 13-14
13.2.3 VAXBI Corner Parts List 13-18
13.3 VAXBI CARD CAGE REQUIREMENTS 13-20
13.3.1 General Requirements 13-20
13.3.2 Orientation 13-21
13.3.2.1 VAXBI Cage Orientation A (Preferred) 13-22
13.3.2.2 VAXBI Cage Orientation B (Allowed) 13-23
13.3.2.3 VAXBI Cage Orientation C (Allowed) 13-24
13.3.2.4 VAXBI Cage Orientation D (Allowed) 13-25
13.3.3 Cable Access 13-26
13.4 VAXBI REFERENCE DESIGNATOR SYSTEM 13-27
13.4.1 Vertical Module Mounting 13-27
13.4.2 Horizontal Module Mounting 13-28
13.4.3 Connector Zones 13-29
13.4.4 Connector Pins 13-30
13.4.5 Header Pins 13-31
13.5 VAXBI SLOT INDEPENDENCE 13-32
13.6 VAXBI TERMINATORS 13-33
13.7 OPERATING ENVIRONMENT 13-33

CHAPTER 14 OVERVIEW OF THE BIIC

14.1 BIIC FEATURES 14-2
14.2 BCI AND THE USER INTERFACE 14-5
14.3 BCI FUNCTIONS 14-5
14.3.1 Master Function 14-6
14.3.2 Slave Function 14-6
14.3.3 Interrupt Function 14-7

CHAPTER 15 BIIC SIGNALS

15.1 VAXBI SIGNALS 15-1
15.2 BCI SIGNALS 15-3
15.3 DATA PATH SIGNALS 15-7
15.3.1 Data Lines (BCI D<31:0> H) 15-7
15.3.2 Information Lines (BCI I<3:0> H) 15-7
15.3.3 Parity Line (BCI P0 H) 15-10
15.4 MASTER SIGNALS 15-11
15.4.1 Request Lines (BCI RQ<1:0> L) 15-11
15.4.1.1 VAXBI Transaction Request 15-12
15.4.1.2 Loopback Request 15-13
15.4.1.3 Diagnostic Mode 15-14
15.4.2 Master Abort Line (BCI MAB L) 15-16
15.4.3 Request Acknowledge Line (BCI RAK L) 15-17
15.4.4 Next Line (BCI NXT L) 15-18
15.4.5 Master Data Enable Line (BCI MDE L) 15-19
15.5 SLAVE SIGNALS 15-20
15.5.1 Response Lines (BCI RS<1:0> L) 15-20

15.5.1.1	Use of STALL to Extend VAXBI Transactions	15-21
15.5.1.2	Use of STALL and Loopback Transactions	15-22
15.5.2	Command Latch Enable Line (BCI CLE H)	15-22
15.5.3	Slave Data Enable Line (BCI SDE L)	15-23
15.5.4	Select Line (BCI SEL L)	15-23
15.5.5	Select Code Lines (BCI SC<2:0> L)	15-25
15.6	INTERRUPT SIGNALS	15-27
15.6.1	Interrupt Request Lines (BCI INT<7:4> L)	15-27
15.7	TRANSACTION STATUS SIGNALS	15-28
15.7.1	Event Code Lines (BCI EV<4:0> L)	15-28
15.7.1.1	Summary Event Code Operation	15-28
15.7.1.2	Status Event Code Operation	15-32
15.7.1.3	Special Event Code Operation	15-32
15.7.2	Event Code Windows	15-33
15.7.3	Event Codes	15-33
15.8	POWER STATUS SIGNALS	15-42
15.8.1	AC LO Line (BCI AC LO L)	15-42
15.8.2	DC LO Line (BCI DC LO L)	15-42
15.9	CLOCK SIGNALS	15-44
15.9.1	BCI TIME L	15-44
15.9.2	BCI PHASE L	15-44

CHAPTER 16 BIIC REGISTERS

CHAPTER 17 BIIC DIAGNOSTIC FACILITIES

17.1	SELF-TEST	17-1
17.2	ERROR DETECTION	17-2
17.3	ERROR RECOVERY	17-2

CHAPTER 18 BIIC OPERATION

18.1	POWER-UP OPERATION	18-1
18.1.1	Loading of Configuration Data	18-1
18.1.2	Self-Test	18-2
18.2	RETRY STATE	18-3
18.3	VAXBI TRANSACTIONS AND BIIC OPERATION	18-4
18.3.1	Read-Type Transactions	18-4
18.3.1.1	Master Operation	18-4
18.3.1.2	Slave Operation	18-5
18.3.2	Write-Type Transactions	18-6
18.3.2.1	Master Operation	18-6
18.3.2.2	Slave Operation	18-6
18.3.3	INTR and IDENT Transactions	18-7
18.3.3.1	Master Operation	18-7
18.3.3.2	Slave Operation	18-8
18.3.4	INTR Sequence	18-12
18.3.5	IDENT Sequence	18-14
18.3.6	IPINTR Transaction	18-16
18.3.7	BDCST Transaction	18-16
18.3.8	STOP Transaction	18-16
18.3.9	Invalidate (INVAL) Transaction	18-17

18.3.10 RESERVED Commands 18-17
18.4 TRANSACTION PRIORITY 18-18

CHAPTER 19 PACKAGING INFORMATION

CHAPTER 20 ELECTRICAL CHARACTERISTICS

20.1 ABSOLUTE MAXIMUM RATINGS 20-1
20.2 DC CHARACTERISTICS 20-2
20.3 AC TIMING SPECIFICATIONS 20-5
20.4 LOAD CIRCUITS 20-9
20.5 MEASUREMENT WAVEFORMS 20-11

CHAPTER 21 FUNCTIONAL TIMING DIAGRAMS

CHAPTER 22 CLOCK DRIVER SPECIFICATION

22.1 DC CHARACTERISTICS 22-4
22.2 AC TIMING SPECIFICATIONS 22-8

CHAPTER 23 CLOCK RECEIVER SPECIFICATION

23.1 DC CHARACTERISTICS 23-4
23.2 AC TIMING SPECIFICATIONS 23-7

NOTE 1 TYPES OF ADAPTERS

AN1.1 PROGRAMMED I/O ADAPTERS AN1-1
AN1.2 DIRECT MEMORY ACCESS ADAPTERS, MAPPED ADAPTERS, AND VAX PORT ADAPTERS AN1-2
AN1.3 BUS ADAPTERS AN1-3

NOTE 2 MAIN MEMORY AND CACHE

AN2.1 CACHES AND MULTIPROCESSOR SYSTEMS AN2-3
AN2.1.1 Notify Only After Cached Access AN2-5
AN2.1.2 Notify Upon Read Access AN2-6
AN2.2 REQUIREMENTS FOR VAXBI SYSTEMS WITH WRITE-THROUGH CACHE AN2-8
AN2.3 HOW TO DESIGN VAXBI SYSTEMS WITH WRITE-BACK CACHE AN2-9

NOTE 3 RECOMMENDED USAGE FOR BIIC REGISTERS

AN3.1 ADDRESS REGISTERS AN3-1
AN3.2 BCI CONTROL AND STATUS REGISTER AN3-3
AN3.3 BUS ERROR REGISTER AN3-4

AN3.4 VAXBI CONTROL AND STATUS REGISTER AN3-4
AN3.5 DEVICE REGISTER AN3-5
AN3.6 GENERAL PURPOSE REGISTERS AN3-6
AN3.7 INTERRUPT REGISTERS AN3-6
AN3.7.1 Interrupt Destination Control AN3-7
AN3.7.2 Error Interrupt Vector Control AN3-8
AN3.7.3 User Interface Interrupt Vector Control . . AN3-10
AN3.8 INTERRUPT STRATEGIES AN3-10
AN3.8.1 The "No-Interrupt" Case AN3-11
AN3.8.2 One Interrupt -- With Static Level and Static Vector Control AN3-12
AN3.8.3 Two Interrupts -- With Static Level and Static Vector Control AN3-12
AN3.8.4 2-4 Interrupts -- With Dynamic Level and Static Vector Control AN3-13
AN3.8.5 2-4 Interrupts -- With Static Level and Dynamic Vector Control AN3-13
AN3.8.6 2-4 Interrupts -- With Dynamic Level and Dynamic Vector Control AN3-14
AN3.8.7 2-128 Interrupts -- With Dynamic Level and Dynamic Vector Control AN3-15
AN3.9 INTERPROCESSOR INTERRUPT REGISTERS AN3-16
AN3.10 RESERVED REGISTERS AN3-17

NOTE 4 SELF-TEST

AN4.1 RECOMMENDED GOALS OF SELF-TEST AN4-1
AN4.2 NORMAL AND FAST SELF-TESTS AN4-2
AN4.2.1 Normal Self-Test AN4-2
AN4.2.1.1 Effect of Bus Traffic AN4-3
AN4.2.1.2 Calculation of Self-Test Duration AN4-3
AN4.2.2 Fast Self-Test AN4-4
AN4.3 EXTENDED SELF-TEST AN4-6
AN4.4 VAXBI TRANSACTION DATA PATTERNS AN4-6
AN4.5 MULTIMODULE NODES AN4-7
AN4.6 DETERMINING THE RESULTS OF SELF-TEST AN4-8
AN4.6.1 Using the Broke Bits and the BI BAD L line . . AN4-8
AN4.6.2 Using Only the BI BAD L Line AN4-8
AN4.6.3 Using the Broke Bits and Stored System Configuration Information AN4-9
AN4.6.4 Using Only the Broke Bits AN4-9

NOTE 5 USE OF RETRY RESPONSE CODE

AN5.1 WAYS IN WHICH RETRY IS USED AN5-1
AN5.1.1 Using RETRY as an Alternative to STALL AN5-2
AN5.1.2 Preempting the VAXBI Bus for Another Transaction AN5-2
AN5.1.3 Implementing Interlocks AN5-2
AN5.2 SCENARIOS LEADING TO A RETRY TIMEOUT AN5-3
AN5.2.1 When RETRY Is Used as an Alternative to STALL AN5-3
AN5.2.2 When the VAXBI Bus Is Released for Another Transaction AN5-4

AN5.3 STRATEGIES FOR DEALING WITH RETRY TIMEOUTS . . . AN5-6
AN5.4 RECOMMENDATIONS AN5-8

NOTE 6 USE OF THE CLOCK RECEIVER

AN6.1 VAXBI CORNER LOADING OF RECEIVER OUTPUTS AN6-8
AN6.2 SAMPLE SKEW CALCULATIONS AN6-9
AN6.3 SAMPLE VAXBI NODE CLOCK DESIGN AN6-10

NOTE 7 BCI POWER SEQUENCE TIMING

AN7.1 SIGNAL DESCRIPTIONS AN7-1
AN7.1.1 BCI AC LO L Signal AN7-1
AN7.1.2 BCI DC LO L Signal AN7-1
AN7.2 VAXBI TIMING SPECIFICATIONS AN7-3

NOTE 8 VAXBI BANDWIDTH AND THE BIIC

NOTE 9 USE OF THE RXCD REGISTER IN ROM-BASED DIAGNOSTICS

AN9.1 USE OF THE RXCD REGISTER IN A HOST SYSTEM . . . AN9-1
AN9.2 USE OF THE RXCD REGISTER IN THE NODE UNDER TEST WHEN THE HOST HAS NO RXCD REGISTER AN9-4

NOTE 10 | USE OF DREV FIELD OF DTYPE REGISTER

APPENDIX A BDCST TRANSACTION

APPENDIX B WIRED-OR TRANSMISSION LINE EFFECT

APPENDIX C RESPONSES TO EXCEPTION CONDITIONS

APPENDIX D DIGITAL EXCEPTIONS TO VAXBI REQUIREMENTS

| D.1 EXTENSION CYCLE LIMIT D-1
D.2 INSTRUCTION NOT TO CACHE DATA D-2
| D.3 UNSUCCESSFUL IRCI TRANSACTIONS D-2
D.4 SELF-TEST D-3
D.4.1 Performance of Self-Test D-3
D.4.2 Use of the VAXBI Bus During Self-Test D-3
D.4.3 Setting of VAXBI Registers at the End of Self-Test D-3
D.4.4 Assertion of BI NO ARB L by VAXBI Primary Interface D-4
D.4.5 Deassertion and Reassertion of BI BAD L D-4
D.4.6 Self-Test Time Limit D-4

D.5 RESPONSE TO STOP TRANSACTION D-4
D.6 VAXBI/BIIC PROTOCOL D-5
D.7 RESPONSE TO READ-TYPE TRANSACTIONS D-5
| D.8 MECHANICAL REQUIREMENTS D-6
D.9 OPERATING REQUIREMENTS D-6
D.10 ARBITRATION D-7
D.11 USE OF RETRY RESPONSE D-7
D.12 RESPONSE TO RXCD REGISTER D-7
| D.13 RESPONSE TO IDENT TRANSACTION D-7
| D.14 BIIC MUST ALLOW DISABLING OF STO D-8
| D.15 OPERATION WITH BIIC DTYPE BIT <13> SET D-8

APPENDIX E PASS FOUR BIIC

E.1 NODE RESET DURING SELF-TEST E-1
E.2 PARITY AND BIIC REGISTERS E-2
E.2.1 Interrupt Error Sequences E-3
E.2.2 BER Error Sequences E-6

APPENDIX F ARCHITECTURE MANAGEMENT

F.1 RESPONSIBILITIES F-1
F.1.1 Project, Product, and Engineering Managers . . . F-1
F.1.2 Hardware, Microcode, and Software Engineers . . F-2
F.1.3 Systems Architecture Group and Architects . . . F-2
F.1.4 Architecture-Review-Group Representatives . . . F-3
F.2 SPECIFICATION DISTRIBUTION F-3
F.3 CONFORMANCE AND WAIVERS F-4
F.3.1 Publishing Architecture Discrepancies F-5
F.4 ECO PROCESS F-5
F.4.1 Participants in the ECO Process F-5
F.4.2 ECO Proposals F-6
F.4.3 ECO Proposal Review F-7
F.4.4 ECO Approval And Appeal F-8
F.4.5 Publishing And Distribution of ECO Results . . . F-8
F.4.6 Specification Retirement F-8
F.5 SYSTEMS-ARCHITECTURE-GROUP SPECIFICATIONS F-9

APPENDIX G DEVICE TYPE CODE ASSIGNMENT PROCEDURES

GLOSSARY

INDEX

FIGURES

1-1 VAXBI Bus Address Space 1-3
1-2 VAXBI I/O Address Space 1-4
1-3 Block Diagram of VAXBI Node 1-9
1-4 Small System Configuration with One Processor . 1-12
1-5 Stand-Alone Single Board Computer Configuration 1-13
1-6 Multiprocessor Configuration 1-14

1-7 Multiprocessor SBC Configuration 1-15
1-8 Midrange System Configuration 1-16
1-9 Multiprocessor High-End System Configuration . . 1-17
1-10 Multiprocessor High-End System Configuration with Multiple VAXBI Buses 1-18
1-11 Configuration of a Cluster Node 1-19
2-1 VAXBI I/O Address Space 2-2
2-2 Addressing of I/O Space 2-3
2-3 Nodespace Allocation 2-4
3-1 Node ID and Arbitration 3-3
3-2 Format of VAXBI Transactions 3-4
4-1 VAXBI Signals 4-2
4-2 Transaction Showing BI NO ARB L and BI BSY L . . . 4-7
4-3 State Sequences of BI NO ARB L and BI BSY L . . . 4-8
4-4 Arbitration State Diagram 4-9
5-1 Longword and Byte References in an Octaword Block 5-7
5-2 WRITE and WCI Transactions (octaword length shown) 5-16
5-3 WMCI and UWMCI Transactions (octaword length shown) 5-19
5-4 READ, RCI, and IRCI Transactions (octaword length shown) 5-22
5-5 INVAL Transaction 5-26
5-6 INTR Transaction 5-31
5-7 IDENT Transaction 5-34
5-8 IPINTR Transaction 5-38
5-9 STOP Transaction 5-41
6-1 System Reset Sequence 6-11
6-2 System Reset Timing Diagram 6-13
6-3 "Extended" System Reset Timing Diagram 6-13
8-1 Required Sets of Transactions 8-5
8-2 The VAXBI as an I/O Bus 8-11
9-1 Console Configuration 9-2
9-2 RXCD Protocol 9-4
9-3 Handling of Escape Characters in the Z Console Command 9-6
10-1 Bandwidth Ranges 10-2
10-2 Arbitration Algorithm 10-6
10-3 Example of Dual Round-Robin Mode Behavior . . . 10-7
11-1 Self-Test Flow 11-5
12-1 Clock Timing 12-3
12-2 Edge-to-Edge Skew Between TIME and PHASE 12-4
12-3 Differential Clock Skew 12-4
12-4 Worst-Case Setup Time Configuration 12-11
12-5 Worst-Case Hold Time Configuration 12-13
13-1 VAXBI Module Views 13-2
13-2 VAXBI Corner Signal Locations (component side view) 13-14
13-3 Orientation A (front view) 13-22
13-4 Orientation B (front view) 13-23
13-5 Orientation C (front view) 13-24
13-6 Orientation D (front view) 13-25
13-7 Reference Designation for Vertical Module Mounting 13-27

13-8 Reference Designation for Horizontal Module Mounting 13-28
13-9 Reference Designation for Connector Zones . . . 13-29
13-10 Reference Designation for Connector Pins 13-30
13-11 Reference Designation for Header Pins 13-31
14-1 Block Diagram of a VAXBI Node 14-2
14-2 Block Diagram of the BIIC 14-4
15-1 BCI Signals 15-4
15-2 Pipeline-Request Signal Timing 15-12
15-3 Summary EV Code Timing for Successful Write-Type and BDCST Transactions (Longword Write) 15-30
15-4 Summary EV Code Timing for Successful Read-Type Transactions (Longword Read) 15-31
15-5 Summary EV Code Timing for Successful STOP, INTR, IPINTR, and INVAL Transactions 15-32
15-6 Transaction EV Code Windows 15-33
18-1 IDENT EV Code Timing 18-10
18-2 INTR Sequence 18-13
18-3 IDENT Sequence 18-15
18-4 BIIC Request Prioritization 18-19
19-1 Packaged BIIC with Heat Sink 19-1
19-2 BIIC Pin Grid Array (top view) 19-2
20-1 BIIC AC Timing Symbol Definitions 20-8
20-2 Test Fixturing for VAXBI Driver Measurements . . 20-9
20-3 Test Fixturing for BCI Driver Measurements . . . 20-10
20-4 VAXBI Driver Output Waveforms 20-11
20-5 VAXBI Receiver Setup and Hold Time Waveforms . . 20-11
20-6 VAXBI Driver Minimum Fall Time Waveform 20-12
20-7 BCI Transceiver Propagation Delay Waveforms . . 20-12
20-8 BCI Receiver Setup and Hold Time Waveforms . . . 20-12
20-9 BCI Driver Rise and Fall Time Waveforms 20-13
21-1 Longword Read-Type with a STALL 21-6
21-2 Loopback Longword Read-Type 21-7
21-3 Loopback Longword Read-Type with a STALL 21-8
21-4 Retried Longword Read-Type with a STALL 21-9
21-5 Quadword Read-Type 21-10
21-6 Quadword Read-Type with Pipeline Request 21-11
21-7 Quadword Read-Type with Pending Master 21-12
21-8 Quadword WRITE 21-13
21-9 Quadword WRITE with Pipeline Request 21-14
21-10 Quadword WRITE with a STALL for Each Longword of Data 21-15
21-11 Octaword WMCI with Variable STALLs for Each Longword of Data 21-16
21-12 Octaword WMCI with Variable STALLs for Each Longword of Data and with Pipeline NXT Enable Bit Set 21-17
21-13 Force-Bit Requested Interrupt 21-18
21-14 INT<7:4> Requested Interrupt 21-19
21-15 Master Port Interprocessor Interrupt 21-20
21-16 Force-Bit Requested Interprocessor Interrupt . . 21-21
21-17 IDENT with Internal Vector 21-22
21-18 IDENT with External Vector 21-23
21-19 INVAL 21-24
21-20 STOP 21-25

21-21 STOP with Extension 21-26
21-22 Quadword BDCST 21-27
21-23 Burst Mode WRITEs with Pipeline Request 21-28
21-24 Burst Mode WRITEs with Pipeline Request and with Pipeline NXT Enable Bit Set 21-29
21-25 Special Case 1 21-30
21-26 Special Case 2 21-31
21-27 Special Case 3 21-32
21-28 Special Case 4 21-33
22-1 Pin Assignments (top view) 22-2
22-2 DC Tests 22-10
22-3 Ioz Test 22-11
22-4 AC Test Fixture 22-12
22-5 Propagation Delay Test 22-13
22-6 Tskw1 Test 22-13
22-7 Tskw2 Test 22-14
22-8 Vcc Noise Immunity Test 22-15
23-1 Pin Assignments (top view) 23-2
23-2 DC Tests 23-9
23-3 Tskw1, Tplh, and Tphl Tests 23-10
23-4 Tskw2 Test 23-11
23-5 Vcc Noise Immunity Test 23-12

NOTES

AN2-1 Various Configurations AN2-2
AN2-2 Configurations with Nodes That Do Local Writes . AN2-4
AN2-3 Configuration with Multiple Buses AN2-6
AN5-1 Sample Configuration AN5-3
AN5-2 Configuration with a Nonpended Bus Adapter . . . AN5-5
AN5-3 Configuration with Two Nonpended Buses AN5-5
AN6-1 TIME and PHASE Waveforms AN6-5
AN6-2 Timing Relationships Between TIME and PHASE Signals AN6-6
AN6-3 Skew Between True and Complement Outputs of TIME and PHASE AN6-7
AN6-4 Even Phase Clock Generator Logic AN6-11
AN6-5 Odd Phase Clock Generator Logic AN6-12
AN6-6 VAXBI Node Clock Worst-Case Waveforms AN6-13
AN6-7 Sample Output from Clock Design Aid AN6-14
AN7-1 System Reset Sequence AN7-6
AN7-2 Node Reset Sequence AN7-7
AN7-3 System Reset Timing Diagram AN7-8
AN7-4 "Extended" System Reset Timing Diagram AN7-8
AN8-1 Maximum Bandwidth Obtainable at a BIIC Slave Port for Various Transaction Lengths AN8-1
AN8-2 Single Master Port Maximum Transfer Rates for Various Transaction Lengths AN8-2
AN8-3 Maximum Transfer Rate for Longword Read-Type Transactions (single nonpipeline-request master port) AN8-3
AN8-4 Maximum Transfer Rate for Longword Write-Type Transactions (single nonpipeline-request master port) AN8-3

APPENDICES

A-1 BDCST Transaction A-2
B-1 Circuit to Demonstrate Transmission Line Problem of Wire-ORing B-2
E-1 Interrupt Error Sequence No. 1 . . . . . E-3
E-2 Interrupt Error Sequence No. 2 . . . . . E-5

TABLES

3-1 Arbitration Codes 3-2
3-2 Command/Address Format by Transaction 3-5
3-3 Data Transferred During Data Cycles 3-6
4-1 4-3
4-2 Response Codes 4-10
5-1 VAXBI Command Codes 5-2
5-2 Data Length Codes 5-5
5-3 Read-Type Transaction Address Interpretation . . . 5-8
5-4 Write-Type Transaction Address Interpretation . . 5-9
5-5 Write Mask Codes 5-13
5-6 Read Status Codes 5-14
5-7 Address Interpretation for INVAL Transaction . . 5-25
6-1 Timing Specifications for BI AC LO L, BI DC LO L, and BI RESET L 6-14
7-1 VAXBI Registers 7-2
7-2 Arbitration Codes 7-7
12-1 Timing Parameters Used for Worst-Case Calculations 12-10
12-2 Maximum Data Path Resistance (in milliohms) . . 12-15
15-1 VAXBI Signals 15-1
15-2 BCI Signals 15-5
15-3 Data Length Codes 15-7
15-4 BCI Command Codes 15-8
15-5 Read Status Codes 15-9
15-6 Write Mask Codes 15-9
15-7 BCI Request Codes 15-11
15-8 Diagnostic Mode BCI/BI Signal Assignment 15-14
15-9 Diagnostic Mode Operation Codes 15-15
15-10 Slave Response Codes 15-20
15-11 Select Codes 15-26
15-12 Event Codes by Class 15-29
15-13 Event Codes 15-35
15-14 Correlation of Event Codes and Bus Error Register Bits 15-41
16-1 BIIC Registers 16-2
18-1 BIIC Operation During Self-Test 18-3
22-1 Clock Driver Truth Table 22-4

NOTES

AN6-1 AC Clock System Characteristics AN6-3
AN7-1 Power Sequence Timing Specifications (From Chapter 6) AN7-3

AN7-2 BIIC-Related Power Sequence Timing Specifications (From Section 20.3) AN7-5
AN7-3 Calculated BCI Power Sequence Timing Specifications AN7-5
AN8-1 Maximum Transfer Rates for BIIC-Based Master Port Nodes AN8-5

PREFACE

This document describes and specifies the VAXBI bus. It serves as the reference document for designers who are designing to the bus. The manual defines all aspects of the bus, including protocol, architecture, and bus components.

INTENDED AUDIENCE

The VAXBI Standard provides information needed by all engineering disciplines, from system architecture to mechanical packaging. The Reading Path section at the end of the Preface lists which chapters will be of most interest to particular disciplines.

STRUCTURE OF THIS DOCUMENT

The VAXBI Standard has four major parts:

PART ONE Bus Description and Requirements
PART TWO The BIIC
PART THREE Bus Support Components
PART FOUR Application Notes

Chapter 1 provides an overview of the VAXBI bus and the primary interface to the bus, the BIIC. The chapter describes the major features of the bus and introduces the terms used in this document. A glossary appears at the end of the manual.

The chapters in each part are summarized below.

PART ONE Bus Description and Requirements

Chapter 2 describes the partitioning and use of VAXBI address space.

Chapter 3 describes the VAXBI protocol. The chapter explains how nodes arbitrate for use of the bus and then defines the cycles of a transaction.

digital™

Chapter 4 defines the signals on the VAXBI bus.

Chapter 5 describes the transactions that the VAXBI bus supports and gives requirements for their use. The chapter explains how single- and multi-responder transactions differ and provides background on the rationale for defining the different kinds of transactions.

Chapter 6 defines initialization requirements for systems and for individual nodes.

Chapter 7 defines the VAXBI registers. A subset of these registers is required by all nodes; the use of the other registers depends on the node class.

Chapter 8 provides an architectural framework for how requirements on nodes depend on the node class. The chapter categorizes nodes into three classes: processors, memories, and adapters.

Chapter 9 describes the VAXBI console protocol which provides for communication among processors on a VAXBI bus.

Chapter 10 discusses bus bandwidth and the effects on bus access latency and interrupt latency.

Chapter 11 discusses the following features that contribute to the efficient functioning of VAXBI systems: self-test, error checking, and stopping a node.

Chapter 12 is the electrical specification for the signals on the VAXBI bus.

Chapter 13 defines the physical requirements that VAXBI modules, cages, and other subassembly components must meet.

PART TWO The BIIC

Chapter 14 gives an overview of the BIIC (the bus interconnect interface chip) that serves as the primary interface between the VAXBI bus and the user interface logic of a node.

Chapter 15 deals with the BIIC signals but concentrates on the BCI signals, those that connect the BIIC and the user interface logic.

Chapter 16 gives requirements for the use of BIIC registers. The descriptions of the registers appear in Chapter 7.

Chapter 17 describes the BIIC's diagnostic facilities: self-test, error detection, and error recovery.

digital™

Chapter 18 gives a detailed description of BIIC operation. The chapter explains what the BIIC does on power-up, what the BIIC retry state is, and what the role of the BIIC is in each type of transaction.

Chapter 19 gives BIIC packaging information.

Chapter 20 gives the electrical specifications for the BIIC.

Chapter 21 consists of 28 functional timing diagrams of various types of transactions and sequences as implemented by the BIIC.

PART THREE Bus Support Components

Chapter 22 is the specification for the VAXBI clock driver.

Chapter 23 is the specification for the VAXBI clock receiver.

PART FOUR Application Notes

Some information that appears in these notes is required for the implementation of some functions on the VAXBI bus.

Note 1 describes various types of adapters and the kinds of functions they perform.

Note 2 explains how the VAXBI provides for caching in multiprocessor systems. The VAXBI requirements primarily apply to systems with write-through cache. The note also gives suggestions for designing systems with write-back cache.

Note 3 offers strategies for using the BIIC registers. The strategies should be helpful to node designers and software users of the VAXBI bus.

Note 4 discusses the intended goals of self-test and comments on the implementation of self-test for various lengths of self-test.

Note 5 describes use of the RETRY response code and how to avoid or deal with extraneous retry timeouts.

Note 6 describes the use of the VAXBI clock receiver. It also presents a suggested method of generating a family of clock waveforms for use by VAXBI node logic.

Note 7 discusses the power sequence timing from the BCI viewpoint.

Note 8 discusses the bandwidth that can be achieved by the master port and slave port resident on a node using the BIIC.

Note 9 describes use of the RXCD Register when using diagnostics in read-only memory in a VAXBI node.

Note 10 describes the preferred use for the DREV field of the Device Register.

READING PATH

This document is a compendium of VAXBI system requirements that cover a wide range of engineering disciplines to assure a high level of compatibility between nodes. A thorough knowledge of this entire document by all readers is beneficial to the success of any design. However, some readers may want to concentrate on certain sections of the manual that are of greater importance to their task and their area of expertise. The following list suggests areas of the specifications that may warrant more of an in-depth understanding by engineers in certain fields of expertise:

System architects --
- Chapters 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
- Application Notes 1, 2, 3, 5, 9
- Appendixes C, D, F

Node logic designers --
- Chapters 1, 2, 3, 4, 5, 6, 12, 14, 15, 16, 18, 20, 21, 23
- Application Notes 1, 3, 4, 5, 6, 7, 10
- Appendixes C, E

System programmers --
- Chapters 1, 2, 6, 7
- Application Notes 3, 9, 10
- Appendixes C, D, E

System mechanical engineers --
- Chapter 13

Node module layout designers --
- Chapters 1, 13, 19

Maintainability engineers --
- Chapters 1, 3, 4, 5, 6, 7, 11
- Application Notes 3, 4, 10
- Appendixes C, D, E

digital™

Diagnostic programmers --
Chapters 1, 2, 5, 6, 7, 11, 17, 18
Application Notes 1, 3, 9
Appendixes C, D, E

System power supply designers --
Chapters 1, 6, 13

ASSOCIATED DOCUMENTS

VAX-11 Architecture Reference Manual

VAXBI Designer's Notebook (EK-VBIDS-RM)

VAXBI Options Handbook (EB-27271-49)

Module Layout Database Package. (BLVBI-BA) Includes module contro drawings and magnetic tapes.

digital™

CHAPTER 1

OVERVIEW OF THE VAXBI BUS AND THE BIIC

1.1 DIGITAL'S COMMITMENT TO FUTURE COMPUTING NEEDS

Digital introduced its first 32-bit computer in 1978, the VAX computer. With its 32-bit architecture, the VAX computer quickly became the industry standard for minicomputers. Since then VAX computers have increased in speed, power, and reliability. At the heart of the VAX computer family is its architecture, which offers power and extensibility.

Advancing technology offers new ways of solving computing problems, but Digital will continue to maintain its commitment to customers who have invested in Digital hardware and software. Over the years Digital has adhered to its standard of compatibility, first ensuring that PDP-11 users could migrate into the VAX family and now using the VAX architecture as the foundation for new, more powerful systems.

In developing new systems, Digital formulated an interconnect strategy to offer solutions for the needs of future generations of computers. Part of Digital's overall interconnect strategy was to develop the VAXBI[TM]* bus. The VAXBI system uses state-of-the-art integrated circuit technology and has the flexibility to incorporate the anticipated advances in systems and logic technology.

The VAXBI Standard defines all aspects of VAXBI operation required to ensure compatibility. This includes logical bus protocol, electrical characteristics, mechanical components, and higher-level system architectural requirements.

The VAXBI design provides for the evolution in computing styles. Growth of distributed processing in the next decade will be based largely on progressive development in I/O architectures. Cooperative performance of distributed computing resources and their ability to expand and diversify can be enhanced by the hardware interconnects that link their components at all levels, from individual terminals to networks. In particular, system integrators and designers of I/O

---

*VAXBI is a trademark of Digital Equipment Corporation.

digital™

devices will be looking for superior functionality and compatibility throughout the interconnect hierarchy of distributed processing systems. The VAXBI bus provides for a diversity of computing resources. A single VAXBI bus can accommodate a high-speed processor with a private memory, several processors that may share memory and I/O devices, and single board computers. The VAXBI protocol provides for communication among them all.

Digital has designed a custom integrated circuit that implements the VAXBI protocol, including all required bus error detection and error logging functions. Therefore, instead of developing the complex circuitry required to implement a bus protocol, node designers can devote their efforts to the requirements of their specific application.

The rest of this chapter describes the main features of VAXBI systems and defines the terms used to describe the operation of the bus. Section 1.2 introduces the bus and summarizes the VAXBI addressing capability and the peak transfer rate. Section 1.3 presents the transactions supported by the VAXBI bus. Section 1.4 describes the BIIC, the control chip that is the primary interface to the VAXBI bus. Section 1.5 describes the BCI, the interconnect to the user logic. And, finally, Section 1.6 presents some typical configurations of systems using the VAXBI bus.

1.2 DESCRIPTION OF THE VAXBI BUS

The VAXBI bus is a 32-bit synchronous, wire-ORed bus used to join a processor to I/O controllers, I/O bus adapters, memories, and other processors. Its characteristics are low cost, high bandwidth, a large addressing range, and high data integrity. The VAXBI bus is the interconnect successor to the UNIBUS for VAX computer systems.

Arbitration for use of the VAXBI bus is distributed among all the users of the bus, so no processor needs to be dedicated to controlling bus use. The distributed design of arbitration maximizes the use of multiple processors so systems can be configured to meet a variety of needs. Each user on the VAXBI bus is called a node. A single VAXBI bus can service 16 nodes, which can be processors, memory, and adapters. An adapter is a node that connects other buses, communication lines, and peripheral devices to the VAXBI bus. Each of the 16 nodes can control the bus, and the slot placement has no effect on the relative priority of the node. A node receives its node ID, a number from 0 to 15, from a plug on the VAXBI backplane slot into which the node module is inserted. (Chapter 13 specifies the mechanical characteristics of VAXBI components.)

Arbitration logic, which is distributed among all the nodes, is based on a dual round-robin priority scheme within the system. When all

digital™

nodes arbitrate in dual round-robin mode, over time each node has equal access to the bus and after winning an arbitration can become bus master. The master issues a transaction that is responded to by one or more slaves.

The VAXBI protocol specifies that arbitration for bus mastership can take place during an ongoing transaction. The winner of an arbitration that occurs when a transaction is in process becomes the pending master.

1.2.1 Addressing Capability

The VAXBI bus supports 30-bit addressing capability, which provides one gigabyte of address space. This address space is split equally between memory and I/O space (512 megabytes each) (see Figure 1-1).

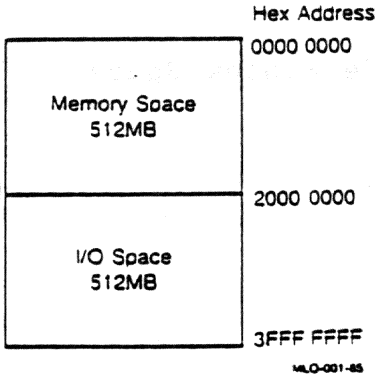


Figure 1-1: VAXBI Bus Address Space

In I/O space, each node has an 8-Kbyte block of addresses known as its nodespace. The first 256 bytes of each nodespace, the BIIC CSR space, are reserved for VAXBI registers. The rest of each nodespace is user interface CSR space. In addition, each node has 256 Kbytes (called its node window) and may have another node-specifiable number of megabytes (called its assignable window) in I/O space for use in mapping addresses to other buses, and so forth (see Figure 1-2).

The basic unit of data that the VAXBI bus handles is the longword (4 bytes), but transactions can transfer from 1 to 16 bytes.

Chapter 2 describes VAXBI address space.

digital™

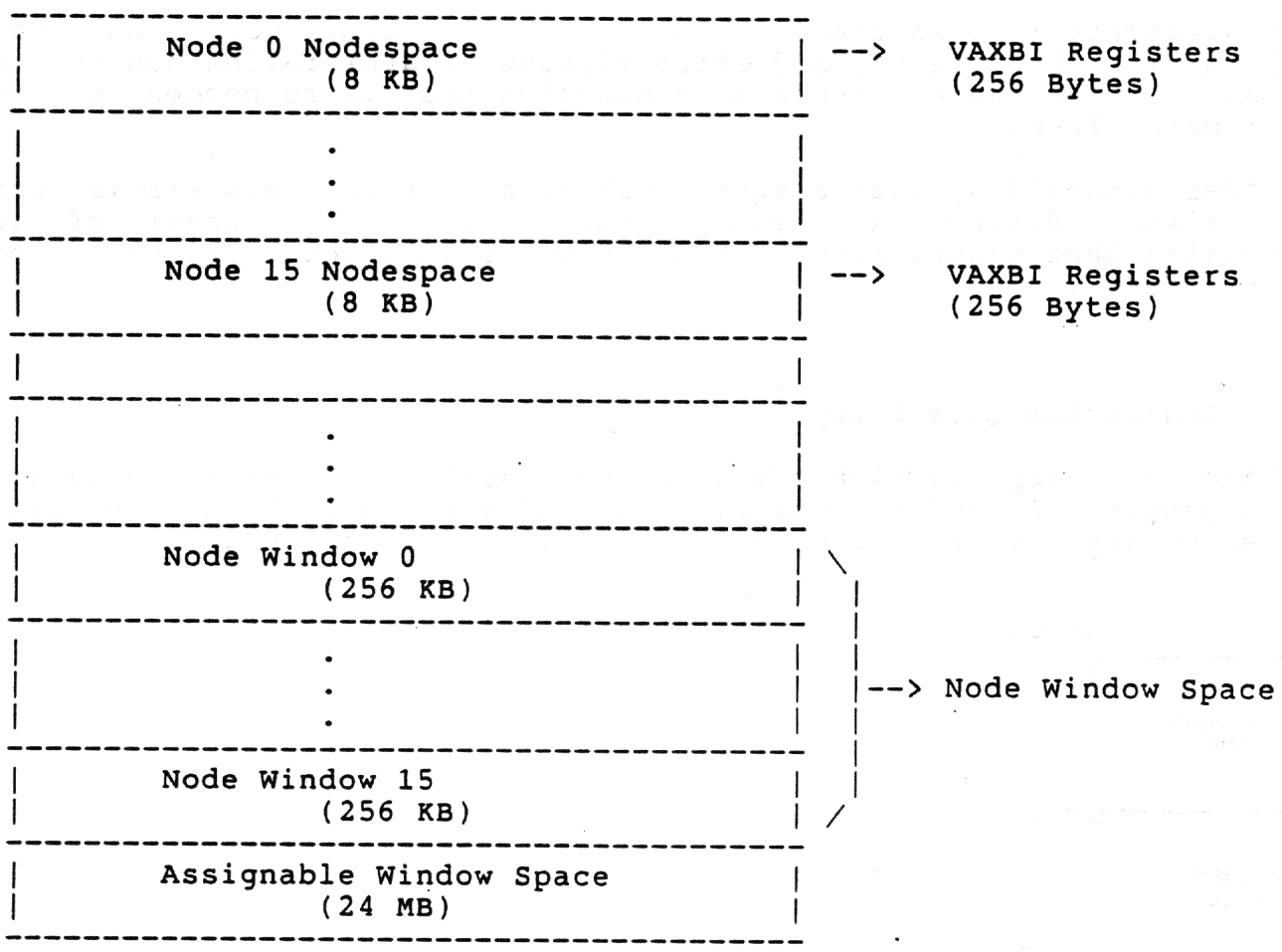


Figure 1-2: VAXBI I/O Address Space

1.2.2 Peak Transfer Rate

Data transmission is at fixed lengths of 4, 8, and 16 bytes (longword, quadword, and octaword lengths) on naturally aligned addressing boundaries. Data transferred within these lengths, however, can be from 1 to 16 bytes in any transaction. As implemented by the BIIC, the maximum data transfer rate on the VAXBI bus, the bandwidth, for 16-byte transfers is 13.3 megabytes per second. For 4-byte transfers the rate is 6.6 megabytes per second. Nodes that are slow responders can stall data cycles of a transaction so that the attempted transfer will be repeated.

Chapter 10 describes VAXBI bus performance, and Application Note 8 explains how the bandwidth is determined for nodes using the BIIC.

digital™

1.2.3 VAXBI Signals

The VAXBI bus has 52 signal lines, 44 of which connect to the BIIC. Four signal lines are for clock signals, and the remaining 4 go elsewhere on the board. The signals can be divided by function into four categories:

- Data Path Signals
  32 data lines
  4 status lines
  1 parity line

- Synchronous Control Signals
  1 no arbitration line
  1 busy line
  3 confirmation lines

- Clock Signals
  4 lines

- Asynchronous Control Signals
  1 AC line
  1 DC line
  1 reset line
  1 self-test fast line
  1 bad line
  1 spare line

The term asserted indicates that a signal line is in the "true" state, while deasserted indicates a "false" state. Assertion is the transition from the false to the true state; deassertion is the transition from the true to the false state. When the absolute level of a signal is specified, the letters H and L indicate a high voltage level and a low voltage level, respectively.

The VAXBI bus is a synchronous interconnect with bus events occurring at fixed intervals. Data is clocked onto the bus at the leading edge of a transmit clock and received and latched with a receive clock at the end of a bus cycle. Information processing occurs during the cycle following the one in which data is transmitted.

Bus arbitration and address and data transmissions are time multiplexed over 32 data lines. Interrupt sequences are performed with VAXBI transactions which may be directed to a single processor or to several processors.

Chapter 4 describes the VAXBI signals.

1.2.4 VAXBI Bus Features

The following list is a summary of key VAXBI features:

- Symmetric and asymmetric multiprocessing.
- Distributed arbitration.
- Slot interchangeability.
- Standardized interface for all designs.
- Address capability of 1 gigabyte.
- Up to 16 full master/slave/interrupt type nodes.
- Bandwidth of 13.3 megabytes per second.
- High degree of data integrity.
- Extensive error logging in all nodes.
- Parity on bus data path.
- Extensive error checking provided on-chip. The BIIC provides for:
  - Checking of parity on the data lines
  - A comparison of data received against data transmitted
  - Protocol checking at all nodes involved in a transaction
- Worst-case design analyzed.
- Power-up self-test in all nodes.

1.3 TRANSACTIONS

The VAXBI bus is a nonpended, bus in that only one transaction can be on the bus at any given time. However, a node can execute a transaction without using the bus. This type of operation, called a loopback transaction, can occur at the same time that the bus is dedicated to an ongoing VAXBI transaction.

Both single- and multi-responder transactions are supported. Every single-responder command is confirmed with a positive acknowledgment for command accepted or command retry, a negative acknowledgment for no responder selected or error detected, or a stall acknowledgment to delay either of the two positive acknowledgments. Multiple responder commands are confirmed as command accepted (by at least one responder) or no responder selected.

Transactions are defined in terms of cycles. The basic bus cycle is 200 nanoseconds.

The VAXBI protocol requires confirmation messages at the end of bus cycles. Depending on the type of cycle and type of transaction, these messages give feedback on errors and slave status. Parity checking monitors the accuracy of data transfer.

The node that gains control of the VAXBI bus for a command transaction is known as the master. The node that responds is the slave.

The first cycle in a transaction is the command/address cycle, during which the node that has gained control of the bus transmits the code for a particular transaction and identifies the node or nodes to respond. The second cycle in any transaction is given over to arbitration. An arbitration cycle that occurs when a transaction is in process is known as an imbedded arbitration cycle. Any nodes other than the current master can negotiate for control of the bus to carry out the next transaction.

During the third cycle of a transaction, the slave sends a command confirmation in response to the command of the first cycle. The slave sends an ACK if it can respond to the request. If the slave can respond and the command was a read or write transaction, then the third cycle is also a data cycle. Data cycles continue until all the data has been transferred. If the slave cannot respond to the command at this time, it issues a STALL or RETRY command confirmation. Upon receipt of a RETRY, the master terminates the transaction and reissues the command at a later time. STALLs delay the continuation of the transaction until the slave can take action. A node returns a NO ACK if the command has not been received successfully.

VAXBI commands are either single-responder commands or multi-responder commands. The single-responder commands include the following:

- o READ
- o RCI (Read with Cache Intent)
- o IRCI (Interlock Read with Cache Intent)
- o WRITE
- o WCI (Write with Cache Intent)
- o WMCI (Write Mask with Cache Intent)
- o UWMCI (Unlock Write Mask with Cache Intent)
- o IDENT (Identify)

The multi-responder commands include:

- o INTR (Interrupt)
- o IPINTR (Interprocessor Interrupt)
- o INVAL (Invalidate)
- o STOP
- o BDCST (Broadcast)

The VAXBI protocol provides for the use of caches, so that reads and writes can be specified depending upon whether data is cached. The terms read-type and write-type are used to describe all the read and write transactions. The transactions "with cache intent" are used when data may be cached. Nodes monitor the bus to see if any transactions with cache intent affect the data that they may have in their cache or in a private memory. If a transaction is specified as a READ or WRITE transaction (in uppercase), this means that data will not be cached. Having the READ and WRITE transactions improves system performance.

The INVAL command is used by processors and adapters to signal other nodes that they may have cached data that is no longer valid. Ordinarily, nodes monitor the bus to see if any transactions with cache intent affect their cached data. However, nodes that do reads or writes to a private memory without performing a VAXBI transaction must have another means of notifying the other nodes that their data may be invalid. The INVAL command meets this need.

digital™

Interrupts are initiated and carried out over the data path. The INTR command is used by nodes to post interrupts. In response, the interrupted processor sends an IDENT command to request vector information from the node that issued the INTR. A processor can also interrupt another processor by sending an IPINTR command.

The BDCST command is reserved for use by Digital. Its operation is described in Appendix A.

Chapter 5 describes the VAXBI transactions.

1.4 THE BIIC

A node's primary interface to the VAXBI bus is the BIIC (bus interconnect interface chip). Figure 1-3 shows a block diagram of a VAXBI node. The BIIC is shown as the VAXBI primary interface (abbreviated VPI) between the VAXBI bus and the user interface logic.

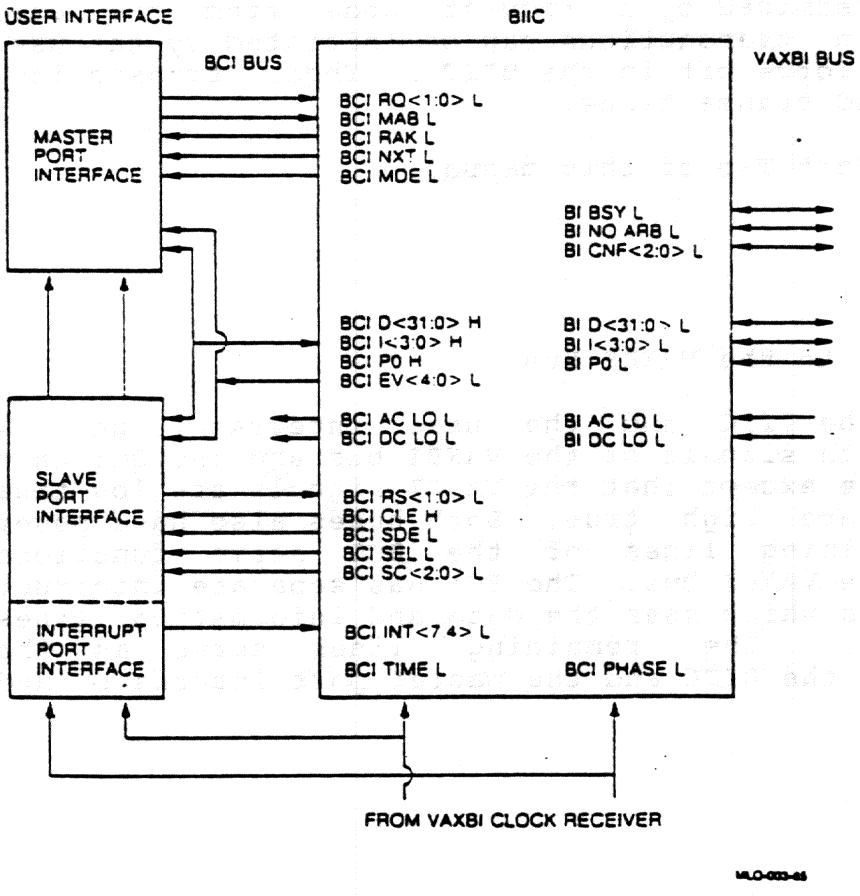


Figure 1-3: Block Diagram of VAXBI Node

The BIIC contains all the logic and registers needed for a node to respond to transactions on the VAXBI bus. In addition, the user interface -- that is, all the logic exclusive of the BIIC -- can request that the BIIC initiate a transaction. The BCI (backplane interconnect chip interface) provides for all communication between the BIIC and the user interface. Since any node can act as either master or as slave, the BCI is represented as having a master port and a slave port. The master port consists of the signal lines used to generate transactions, and the slave port consists of the signal lines used to respond to transactions. The BCI also has an interrupt port, signal lines used in generating interrupt transactions.

Transactions that involve two different nodes are internode transactions, while those that are confined to the same node are intranode transactions. Intranode transactions can be VAXBI transactions (that is, the master port issues a transaction on the VAXBI bus), or they can be loopback transactions (the VAXBI bus is not used). Loopback transactions can occur concurrently with VAXBI transactions.

The type of request is determined by a request code from the user interface logic. Certain transactions can be initiated by the user interface logic setting a force bit in the BIIC. These transactions are known as BIIC-generated transactions.

The BIIC is described in Part Two of this manual.

1.5 DESCRIPTION OF THE BCI

1.5.1 How the BCI Relates to the VAXBI Bus

The BCI, the bus between the BIIC and the user interface, has 64 signal lines. The data path signals of the VAXBI bus and the BCI have a one-to-one correspondence except that the VAXBI signals are low true while the BCI signals are high true. Both buses also have power signal lines, but the remaining lines of the BCI serve functions different from those of the VAXBI bus. The BCI has separate interrupt lines, unlike the VAXBI bus which uses the data and information lines for sending interrupts. The remaining lines serve as the communication path between the BIIC and the master port interface and the slave port interface.

1.5.2 BCI Signals

The BCI has 64 signal lines. The signals can be divided by function into seven categories:

- Data Path Signals
  32 data lines
  4 status lines
  1 parity line

- Master Signals
  2 request lines
  1 master abort line
  1 request acknowledgment line
  1 data ready line
  1 master data enable line

- Slave Signals
  2 response lines
  1 command latch enable line
  1 slave data enable line
  1 select line
  3 select code lines

- Interrupt Signals
  4 interrupt request lines

- Transaction Status Signals
  5 event code lines

- Power Status Signals
  1 AC line
  1 DC line

- Clock Signals
  2 timing signals

Chapter 15 describes the BCI signals.

1.6 SYSTEM CONFIGURATIONS

The VAXBI bus connects processors, I/O controllers, I/O bus adapters, and memory. Because of the potential overlap in functions among processors, adapters, and memory, it is important to know the VAXBI requirements for various classes of nodes. The requirements are designed to ensure that VAXBI nodes will be compatible in the types of configurations for which the VAXBI bus and VAXBI nodes are intended. The descriptions of various types of configurations follow.

digital™

See Chapter 8 for a description of node classes and the VAXBI requirements that each class must meet.

1.6.1 Low-End System Configurations

Figures 1-4 and 1-5 show the VAXBI bus in low-end configurations. The VAXBI can be used in low-end systems either as an I/O bus or as both memory bus and I/O bus.

Figure 1-4 shows a configuration in which the VAXBI bus is used both as memory bus and I/O bus. Such a system would probably include a mass storage adapter for disk storage and a multipurpose communications adapter.

Figure 1-5 shows a configuration in which the VAXBI bus is used only as an I/O bus. With a single board computer (SBC) the processor and memory have a separate memory bus (MB), and the VAXBI bus is used only as an I/O bus. This figure also shows a different approach to mass storage and input/output. A single adapter provides access to disks and tapes and to a local area network (LAN).

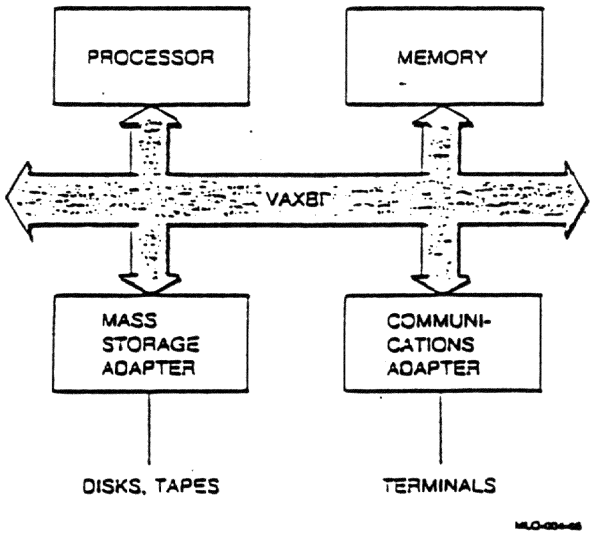


Figure 1-4: Small System Configuration with One Processor

digital™

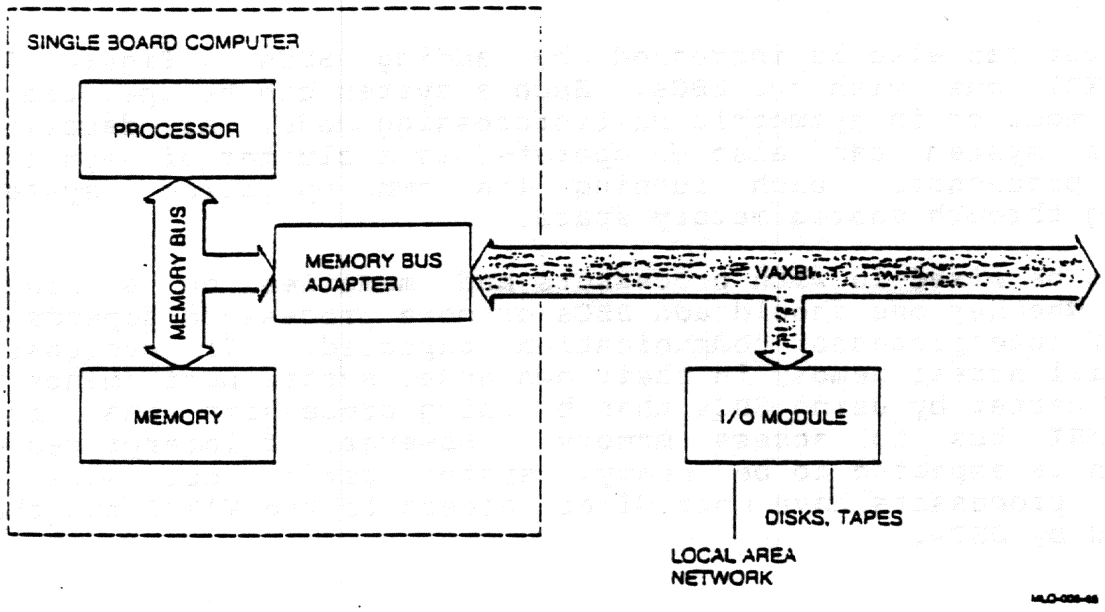


Figure 1-5: Stand-Alone Single Board Computer Configuration

1.6.2 Multiprocessor System Configurations

Figures 1-6 and 1-7 are extensions of the configurations described in the last section; these extensions provide greater computing power.

One way to increase computing power is to add processors. Figure 1-6 shows a multiprocessor system configuration that uses the VAXBI bus as a memory bus. The software may operate this configuration in "master-slave" mode or in "symmetric multiprocessing" mode:

- In master-slave mode, the master processor runs the operating system and manages the other processors, which are called "attached processors." Each processor may have its own console terminal, or there may be a console terminal at the master processor only, which can be used to control any of the processors.

- In symmetric multiprocessing mode, a distributed operating system is used, different parts of which may be executing on different processors at various times. Again, each processor may have its own console terminal, or only one of them may have a console terminal.

digital™

Computing power can also be increased by adding SBCs. Figure 1-7 shows a VAXBI bus with two SBCs. Such a system can be operated in master-slave mode or in symmetric multiprocessing mode, as described above. This system can also be operated as a cluster of separate, independent processors, each running its own operating system, communicating through shared memory space.

SBCs can also be mixed in with processors and memories on a single VAXBI bus. Whether one should add SBCs or more processors depends on the amount of interprocessor communication expected. If processors primarily will access memory in their own node, system performance is likely to be better by using SBCs than by using processors that must use the VAXBI bus to access memory. However, if interprocessor communication is expected to be heavy, system performance will be better when processors have more direct access to the VAXBI bus than that provided by SBCs.

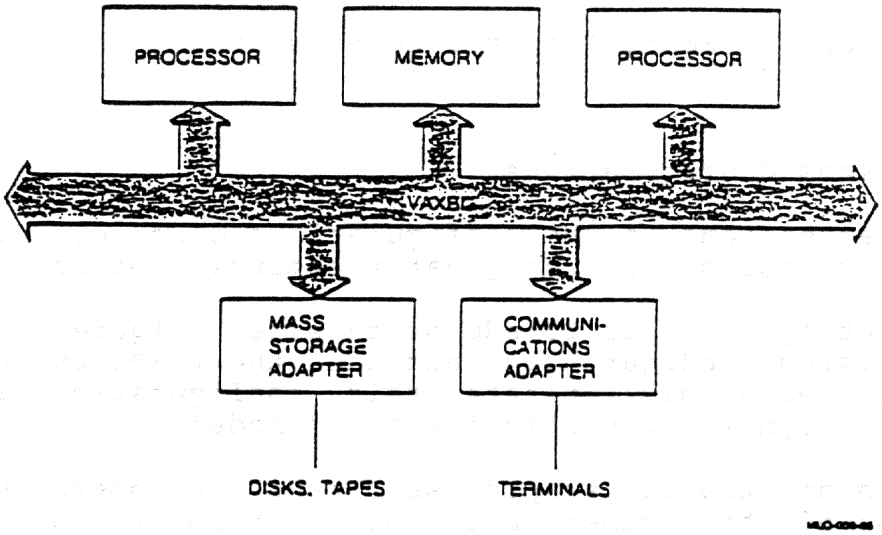


Figure 1-6: Multiprocessor Configuration

digital™

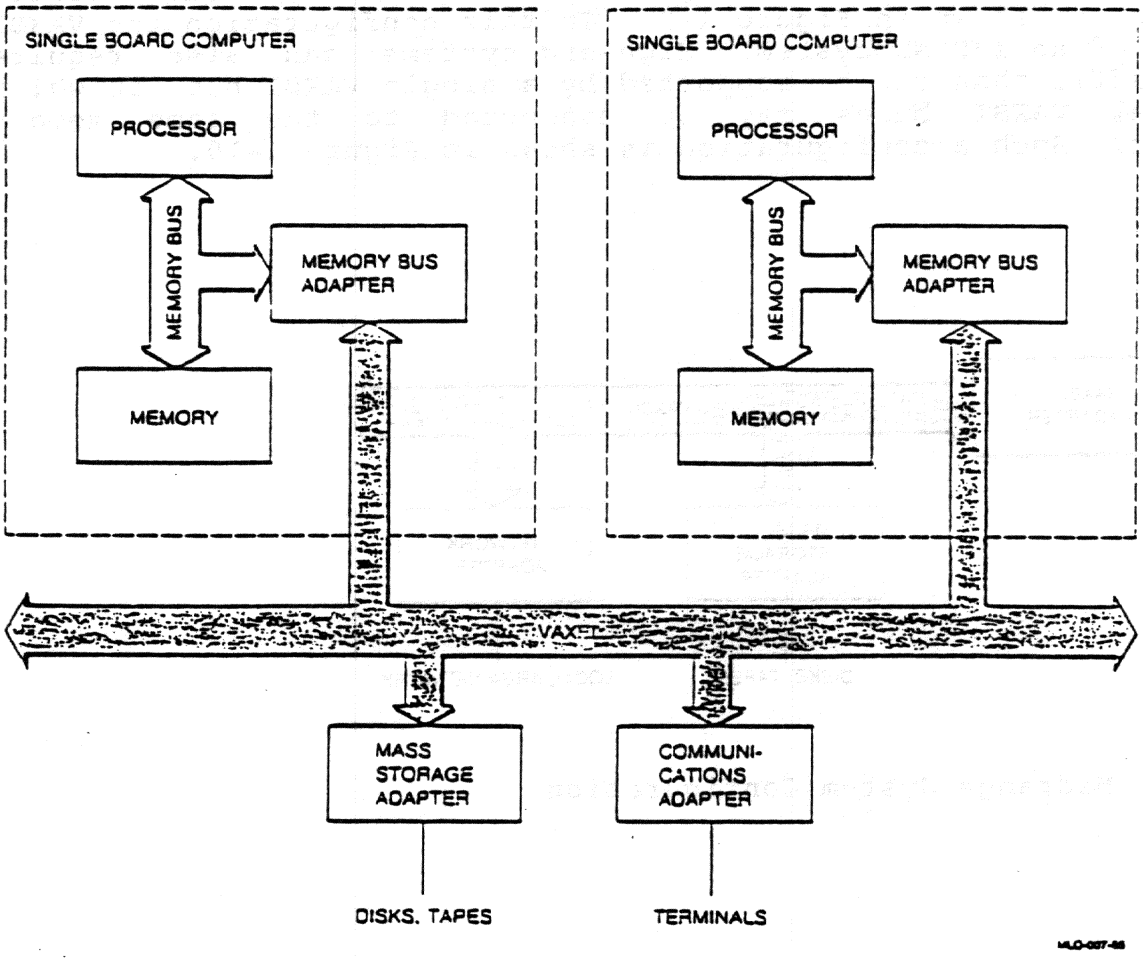


Figure 1-7: Multiprocessor SBC Configuration

1.6.3 Midrange and High-End System Configurations

With more powerful processors, it becomes necessary to improve memory access times by having a dedicated memory interconnect. In such systems the VAXBI bus becomes purely an I/O bus. Such systems can support more input/output than the low-end configurations, and will probably use a local area network adapter instead of the multipurpose input/output adapter. Figure 1-8 shows a midrange system configuration.

digital™

High-end systems can use multiple processors on the memory interconnect as shown in Figure 1-9. In this configuration the VAXBI bus is part of an I/O subsystem. High-end systems can also require more I/O traffic than can be supported by a single VAXBI bus, in which case several VAXBI buses may be connected to the same memory interconnect. Such a configuration is shown in Figure 1-10.

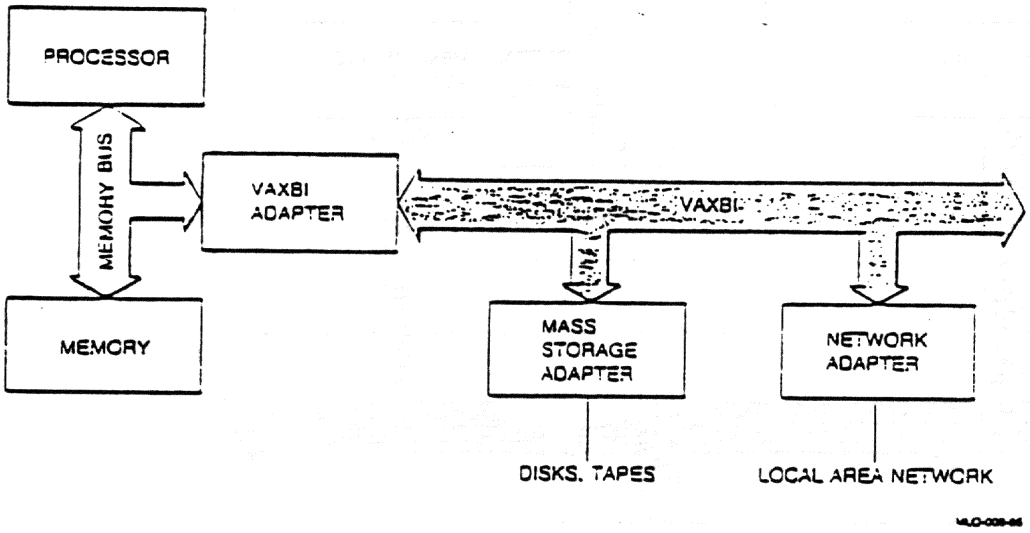


Figure 1-8: Midrange System Configuration

digital™

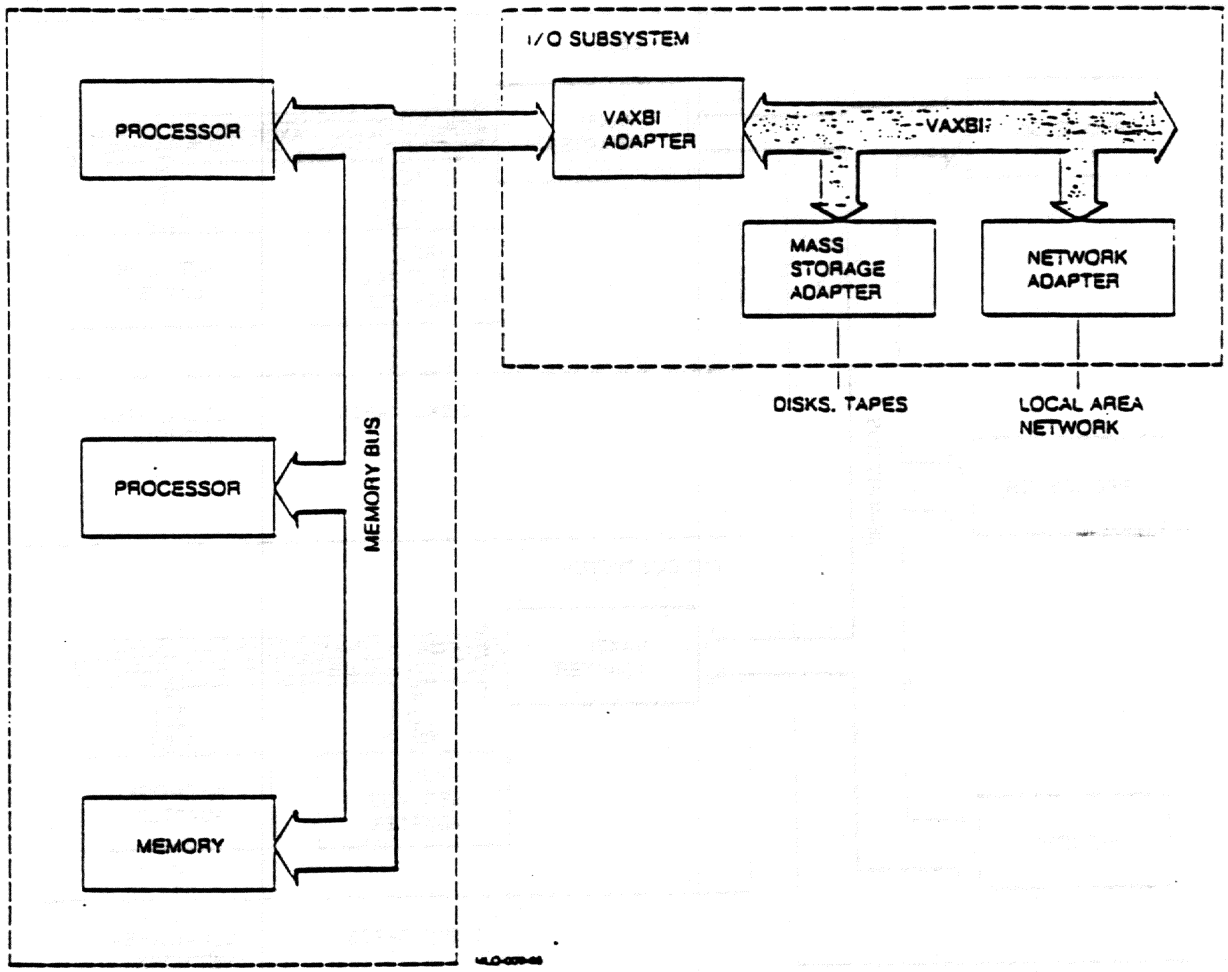


Figure 1-9: Multiprocessor High-End System Configuration

digital™

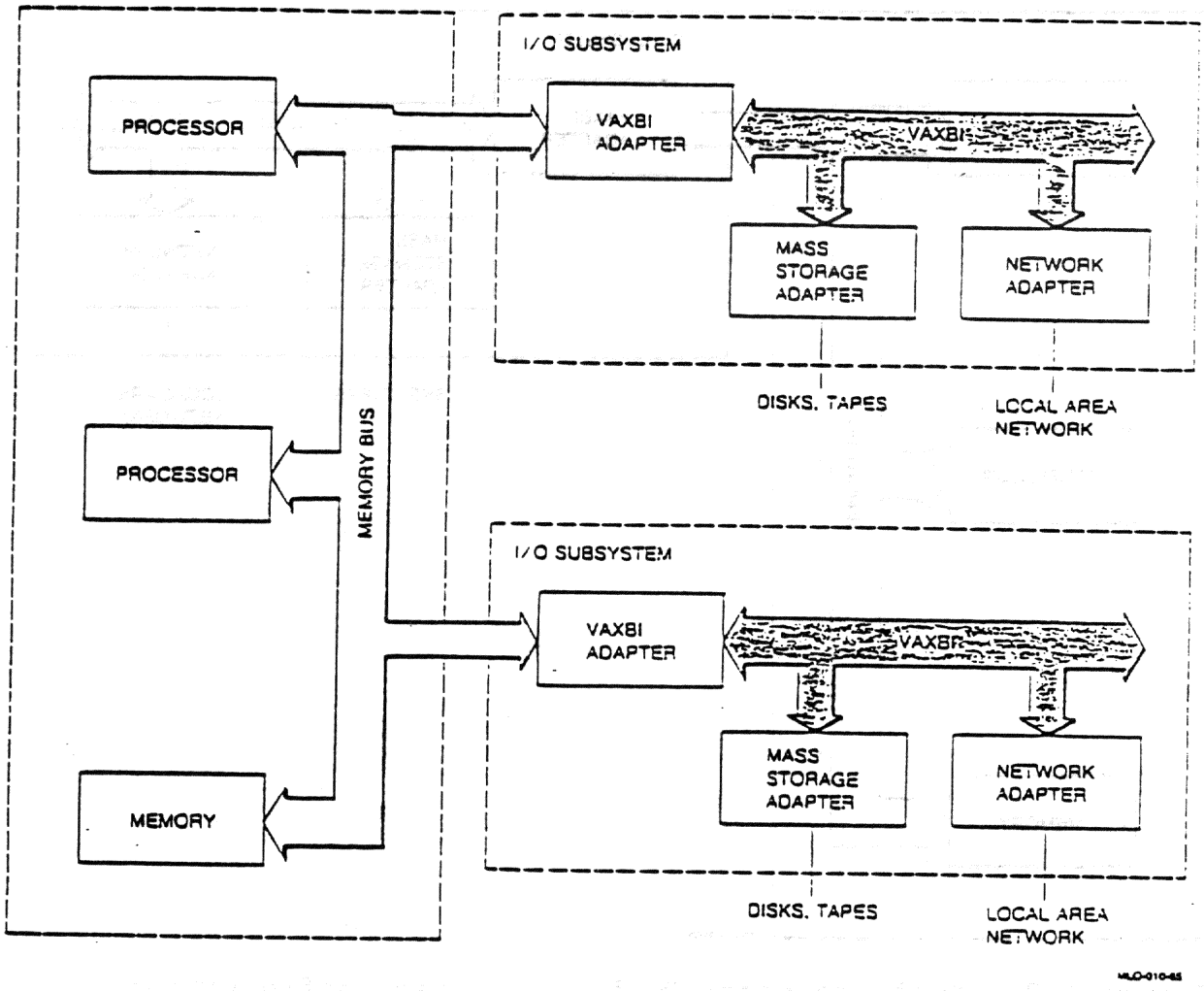


Figure 1-10: Multiprocessor High-End System Configuration with Multiple VAXBI Buses

1.6.4 Clusters and Networking

A system -- whether a low-end system or a high-end system -- can be connected to other systems by a Computer Interconnect (CI), forming a cluster. In Figure 1-11 a VAXBI system with two SBCs is part of a cluster, which might be used in a real-time process control environment, with process control input/output connected by a multifunction adapter. Mass storage facilities are provided solely through the CI.

digital™

A system, at any performance level, can also be connected to other systems into a local area network through a network adapter. For example, in the high-end system shown in Figure 1-10, each VAXBI bus has a local area network adapter for attachment of terminals, servers such as print and file servers, and other computer systems.

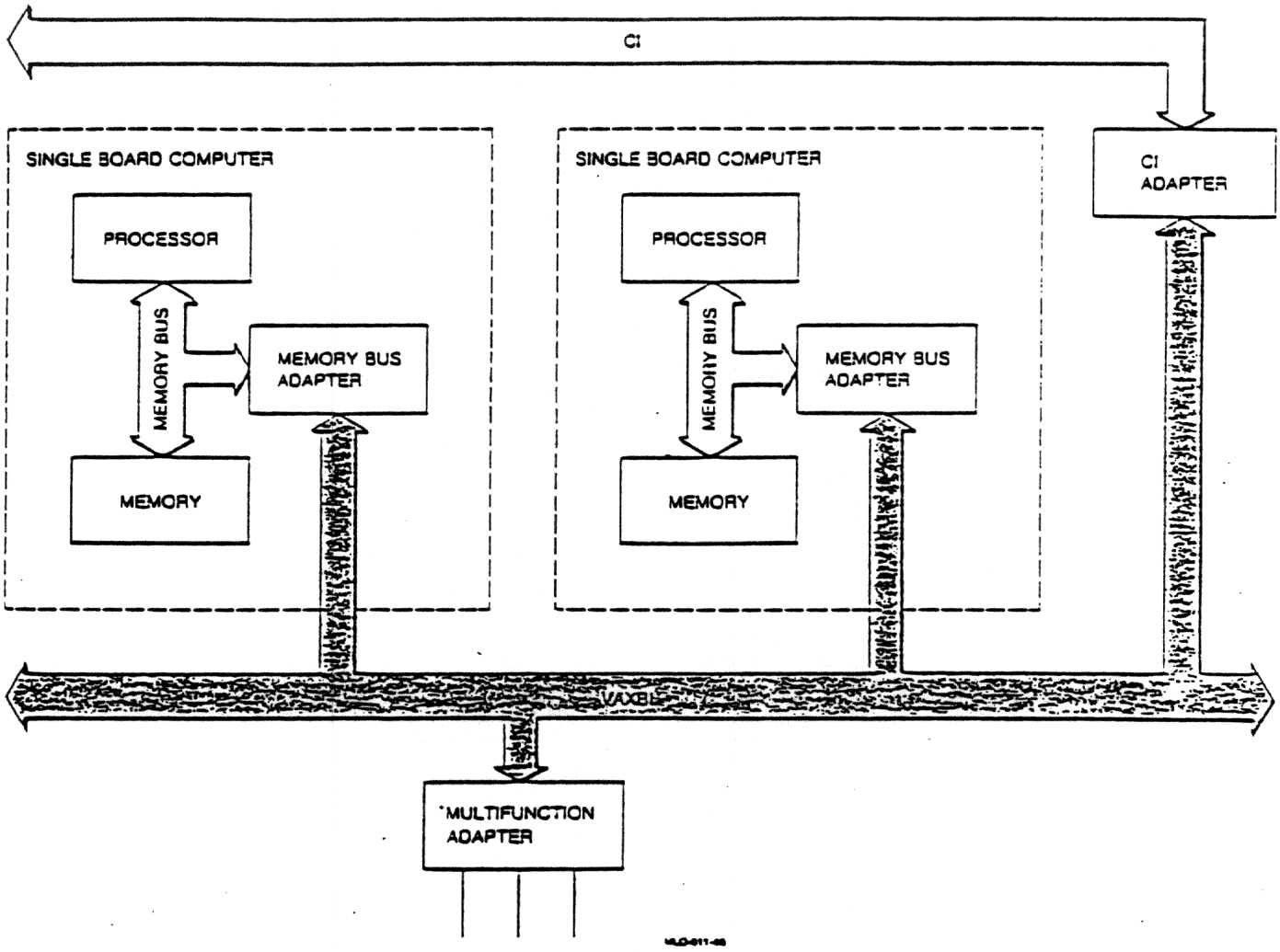


Figure 1-11: Configuration of a Cluster Node

digital™

CHAPTER 2

VAXBI ADDRESS SPACE

This chapter describes the partitioning of VAXBI address space and how to use the space. The VAXBI bus has 2**30 bytes of address space, which is divided into memory space and I/O space. All addresses are given in hexadecimal notation.

During the first cycle of read-type, write-type, and INVAL transactions, a 30-bit physical byte address is transmitted on the BI D<29:0> L lines. We will refer to this address as A<29:0>. When A<29> is a zero, the 512-megabyte memory space is accessed, and when A<29> is a one, the 512-megabyte I/O space is accessed. During the same cycle, D<31:30> indicate the length of the transfer.

2.1 ALLOCATION OF MEMORY SPACE

All memory locations (from 0000 0000 through 1FFF FFFF) are addressed using memory space addresses (A<29> is a zero). Addresses on the VAXBI bus are physical rather than virtual addresses. In other words, any virtual-to-physical translation is performed before the address is transmitted on the VAXBI bus.

Information stored in memory locations can also be stored in a cache and used many times without an access to the actual memory location. Although cache contents may be valid or invalid, memory locations always contain valid information -- they never contain obsolete information (see Application Note 2 for more information on caches).

VAXBI memory is assigned addresses starting at 0000 0000.

digital™

2.2 ALLOCATION OF I/O SPACE

I/O space (from 2000 0000 through 3FFF FFFF) is sparsely filled. In I/O space addresses, A<29> is a one. Figure 2-1 shows the breakdown of VAXBI I/O address space. Figure 2-2 summarizes the addressing of I/O space.

Two blocks of I/O space are partitioned according to node ID. (The node ID is provided by individual ID plugs on the VAXBI backplane.)

- Nodespace. At the low end of I/O space are 16 address blocks of 8 Kbytes each, called "nodespace," one of which is assigned to each node based on its node ID. Each node's nodespace consists of BIIC CSR space (the first 256 bytes) and user interface CSR space (the remainder of the 8K nodespace). The BIIC CSR space contains VAXBI registers (see Figure 2-3).

- Node Window Space. Starting at address 2040 0000 are 16 address blocks of 256 Kbytes each, called "node window space." Node window space can be used by adapters to map VAXBI transactions onto a target bus.

Another region of I/O space starts at address 2080 0000, runs through address 21FF FFFF, and is referred to as "assignable window space". Each node can require that a single n-megabyte block within this address region (n being a positive integer between 1 and 24 inclusive) be allocated to it. This block is referred to as the node's "assignable window." For restrictions governing this allocation, see Section 2.2.5, Assignable Window Space.

The VAXBI architecture defines the use of all of I/O space. There is an addressing convention for multiple VAXBI buses that uses the address bits A<28:25>.* These four bits can define the mapping mechanism for access of up to 16 VAXBI buses. Therefore, these four bits must be cleared before the address is issued on the VAXBI bus. For this reason, the VAXBI address range 2200 0000 through 3FFF FFFF is RESERVED. This allocation also limits the available I/O space to 32 megabytes.

---

*VAX 8800 systems follow this addressing convention.

digital™

VAXBI ADDRESS SPACE

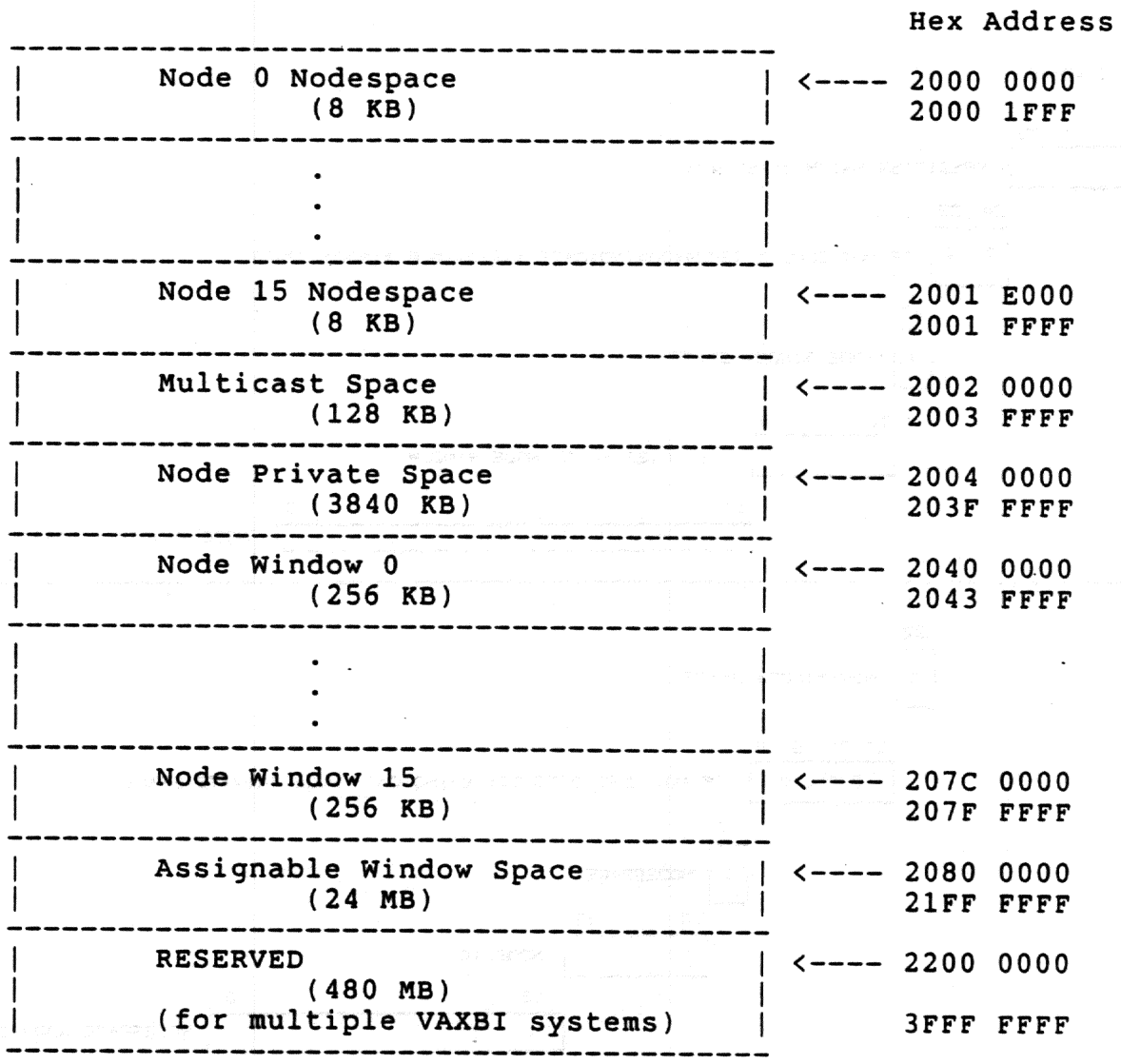


Figure 2-1: VAXBI I/O Address Space

digital™

VAXBI ADDRESS SPACE

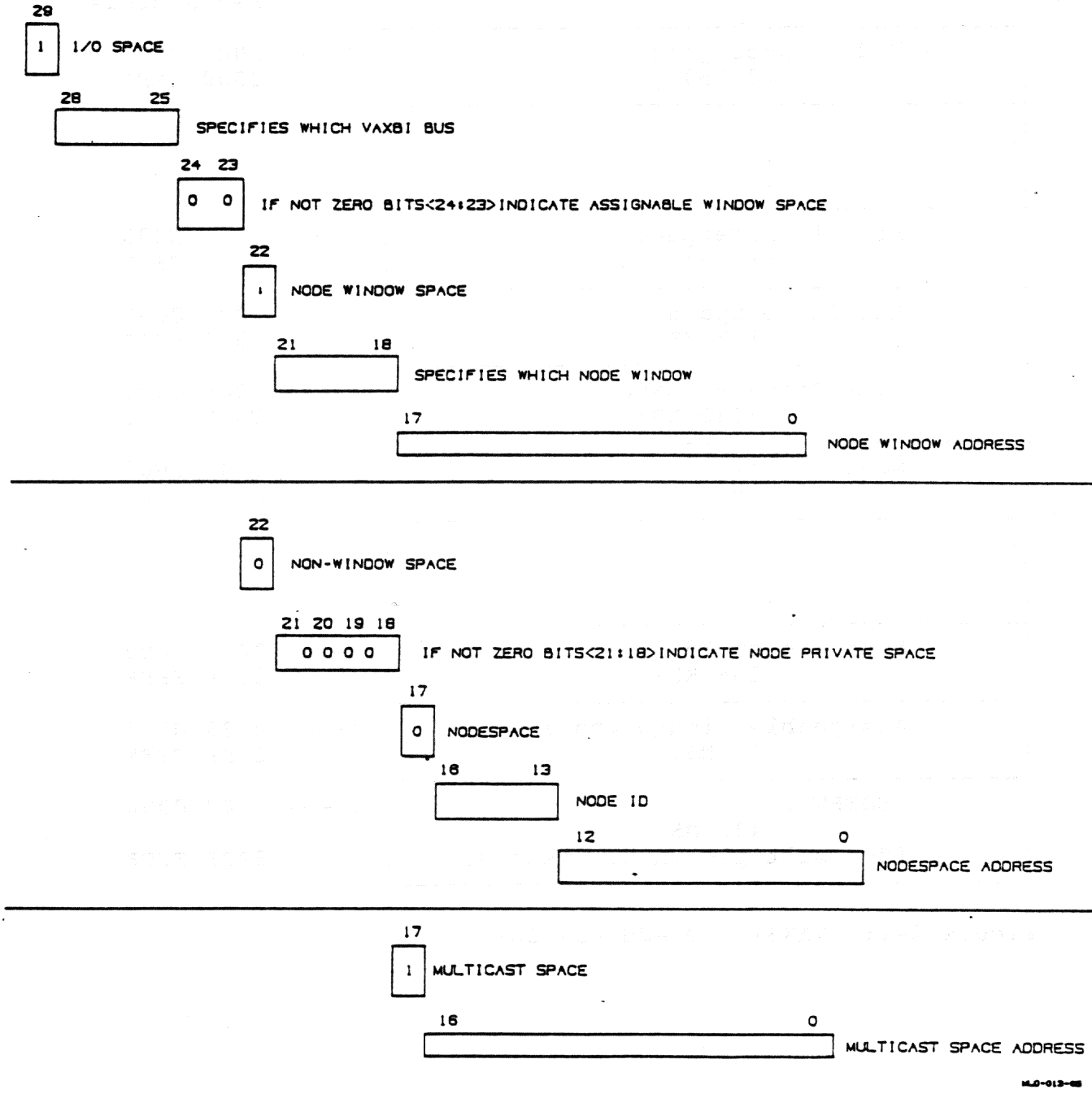


Figure 2-2: Addressing of I/O Space

digital™

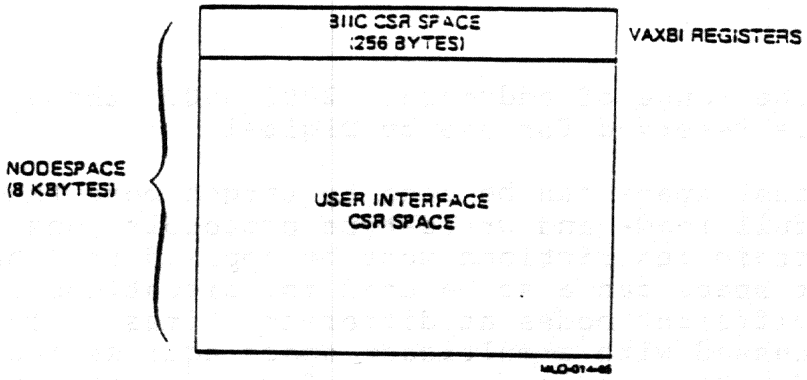


Figure 2-3: Nodespace Allocation

2.2.1 Nodespace

The address range 2000 0000 through 2001 FFFF contains 16 address blocks of 8 Kbytes each. A node's nodespace assignment is based on the node's ID (0 to 15). The starting address of nodespaces for nodes 0 through 15 is 2000 0000 plus 8K times the node ID. We will use "bb" to indicate the base address of a particular node's nodespace.

2.2.1.1 BIIC CSR Space - The first 256 bytes of each node's nodespace are reserved for VAXBI registers.

2.2.1.2 User Interface CSR Space - Within user interface CSR space the use of two locations is defined.

Location bb + 100 is reserved for the Slave-Only Status Register (SOSR), which is used by those nodes that do not implement the Broke bit in the VAXBICSR. This location is not reserved for other nodes. (See Section 7.16 for the register description and Chapter 11 for an explanation of the register's use in node self-test.)

Location bb + 200 is reserved for the Receive Console Data (RXCD) Register. This register must be implemented by nodes capable of performing console terminal transactions. Nodes that do not implement a VAXBI console must respond to reads to that location with either a NO ACK confirmation or a longword in which the RXCD Busy 1 bit is set. (See Section 7.17 for the register description and Section 9.2 for an explanation of the register's use in the VAXBI console protocol.)

2.2.2 Multicast Space

Multicast space consists of the range of addresses 2002 0000 through 2003 FFFF. Multicast space is reserved for use by Digital.

A VAXBI transaction to multicast space can be used to target more than one VAXBI node. Since the full read- and write-type protocols cannot support multiple targets, certain restrictions must be applied to the use of this space. Multicast space can also be used for situations in which a function resides in different nodes at different times. The appropriate node can be accessed with a multicast space address that is node independent. The node that possesses the function at the given time will respond to a transaction that uses that address.

2.2.3 Node Private Space

Node private space consists of the range of addresses 2004 0000 through 203F FFFF. Locations beginning at 2004 0000 are used for storage of bootstrap firmware and software. VAXBI nodes are not permitted to issue or respond to VAXBI transactions targeting locations in node private space. For programmable VAXBI nodes, this restriction may be interpreted as a restriction on the software and/or firmware rather than a requirement for a hardware check that ensures that such accesses cannot happen.

2.2.4 Node Window Space

The address range 2040 0000 through 207F FFFF contains 16 address blocks of 256 Kbytes each, which can be used by bus adapters to map VAXBI transactions onto a target bus.

A node's window space assignment is based on its node ID. A<21:18> in an I/O space address specifies the node window of a particular node. Nodes are not required to implement the address locations in the node window allocated to them.

2.2.5 Assignable Window Space

The address range 2080 0000 through 21FF FFFF can be used by bus adapters to map VAXBI transactions onto a target bus, or for other designer-determined purposes.

A contiguous range of one or more naturally aligned megabytes within assignable window space can be statically allocated to a particular node by the OS (operating system). The range associated with a node is referred to as that node's "assignable window."

Node designers must request the number of megabytes needed by the node for its assignable window; this number can be any integer between 1 and 24 inclusive. This size must be a constant determined by the device type code of the node. So that other nodes that request an assignable window are not crowded out, only the exact number of megabytes needed should be specified (for instance, the number should not be rounded up to a power of 2). However, in predicting whether a given combination of nodes can be configured, it must be assumed that the OS can round this up to a power of 2 anyway. Any node requesting more than 16 megabytes will be allocated the full 24 megabytes of assignable window space.

Any node requesting the allocation of an assignable window must not require a particular starting address, but must allow its starting address to be assigned by the OS. However, the node designer is allowed to require "alignment." When alignment is required, the OS must align the starting address of the assignable window at an address that is a multiple of the smallest k such that:

1. k >= max(the requested size, 1 megabyte), and

2. k is a power of 2.

However, if more than 16 megabytes are requested, the entire 24 megabytes of assignable window space will be allocated to the node.

The purpose of the alignment rule is to allow the node designer some economy in address decoding.

In predicting whether a given combination of nodes can be configured, it must be assumed that the OS can perform the allocation as if alignment has been specified for every requesting node.

An example:

A node designer needs a 5-megabyte assignable window and requires alignment. The OS must then align the starting address of this node on an 8-megabyte boundary. This restricts the possible starting addresses to 2080 0000, 2100 0000, and 2180 0000. The OS can allocate exactly 5 megabytes to the adapter, or it can round up

to the next power of 2 and allocate 8 megabytes. In the latter case, the adapter's assignable window would span one of these three address ranges:

- o 2080 0000 to 20FF FFFF
- o 2100 0000 to 217F FFFF
- o 2180 0000 to 21FF FFFF

The reason for requiring separate specification of window size and alignment is to make possible a set of address assignments that otherwise might be impossible.

An example:

A second adapter (reference previous example) requires a 2-megabyte assignable window with no alignment restriction. The second adapter can be mapped into the last 2 megabytes of the 8-megabyte address range that the first adapter is mapped into.

Note that configuration problems can arise in allocating assignable windows when more than one node requires an assignable window. The likelihood and severity of configuration problems increases with both the size of the requested window and the number of nodes requesting an assignable window.

A node is not required to implement address locations in an assignable window allocated to that node.

2.3 BIIC RESTRICTIONS

The BIIC can be configured to respond to accesses to any combination of the following address spaces:

- o The node's nodespace
- o The space defined by the node's Starting and Ending Address Registers. (For example, this space could be this node's node window or a region in memory space.)
- o Multicast space

Because a BIIC has only one pair of Starting and Ending Address Registers, it cannot be set to allow a node to respond to both its node window (or an assignable window) and a region of memory space.

Response to accesses to multicast space can be disabled through a bit in the BCI Control and Status Register, as can accesses to the user interface CSR space portion of nodespace.

CHAPTER 3

VAXBI PROTOCOL AND CYCLE TYPES

Section 3.1 describes how nodes are identified and the use of the node ID. Section 3.2 describes how nodes arbitrate for control of the bus and explains the priority scheme. Section 3.3 gives an overview of the types of VAXBI cycles.

3.1 NODE IDENTIFICATION

Each node that interfaces to the VAXBI bus has an identification number (0 to 15) called the "node ID." The node ID is provided by individual ID plugs on the backplane. This ID code determines bus priority, interrupt sublevel priority, and the location of the node's registers. Lower number IDs have higher priorities.

3.2 ARBITRATION

Arbitration logic is distributed among all VAXBI nodes. To become bus master, a node arbitrates by asserting one of the 32 data lines during an "arbitration cycle." During this cycle the node determines if there are any lower number data lines asserted. If not, that node wins the bus and may send command/address information when the current bus transaction (if one exists) has completed.

The BI NO ARB L line controls access to the bus data path for arbitration. Arbitration can occur in any cycle following t deasserted state of BI NO ARB L. Arbitration cycles can occur during and outside of bus transactions.

After a master sends the command/address, nodes require the next cycle to decode addresses. The VAXBI protocol allows use of this cycle for arbitration. Within a transaction this cycle is called an "imbedded arbitration cycle." A master cannot arbitrate in the imbedded arbitration cycle of its own transaction.

During imbedded arbitration cycles, the master of the current transaction transmits its encoded ID on the BI I<3:0> L lines; parity is generated by the master for these lines and is checked by all nodes. Nodes use this encoded ID information to calculate arbitration priority.

3.2.1 Arbitration Modes

The VAXBI protocol defines three arbitration modes that a node can be assigned:

- Dual round-robin
- Fixed-high priority
- Fixed-low priority

3.2.1.1 Setting the Mode - These modes are determined by a two-bit field (see Table 3-1) within the VAXBI Control and Status Register (see Section 7.2). Any combination of arbitration modes can coexist among nodes on the VAXBI bus; however, fixed-high and fixed-low priority arbitration modes are reserved for use by Digital. A node's arbitration mode can be changed during system operation. However, all nodes must default to the dual round-robin arbitration mode at power-up time. Arbitration can also be disabled.*

Table 3-1: Arbitration Codes

Bit 5	Bit 4	Meaning
0	0	Dual round-robin arbitration
0	1	Fixed-high priority (RESERVED)
1	0	Fixed-low priority (RESERVED)
1	1	Disable arbitration (RESERVED)

*Arbitration must be disabled on a target node before issuing a node reset to that target node.

digital™

3.2.1.2 Two Priority Levels - Arbitration on the VAXBI bus is performed in two priority levels for each node. During each imbedded arbitration cycle, all nodes are required to update their arbitration priority based on the arbitration mode and the current master's ID.

Low-priority nodes assert the bit corresponding to their node ID on BI D<31:16> L where D<16> corresponds to ID 0, and bit D<31> corresponds to ID 15. High-priority nodes assert the bit corresponding to their node ID on BI D<15:0> L during the arbitration cycle. The relative position within the low- or high-priority word is the same. Figure 3-1 shows the mapping of node ID to arbitration priority on the 32 data lines. At power-up all nodes must default to the low-priority word.

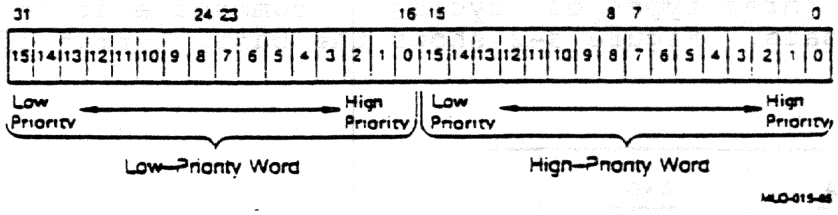


Figure 3-1: Node ID and Arbitration

3.2.1.3 Dual round-robin - The dual round-robin arbitration mode operates as follows. A node will arbitrate on the low-priority word for the next arbitration cycle if its ID is less than or equal (that is, equal or higher priority) to the node ID of the previous bus master. A node will arbitrate on the high-priority word if its node ID is greater (that is, lower priority) than the node ID of the previous bus master.

If all nodes arbitrate in dual round-robin mode, then over time each has equal access to the bus. The dual round-robin mode is important, for example, in multiprocessor configurations. In these systems a fixed-priority scheme could cause extremely long bus latency times for some nodes that were denied bus access by several processors executing in tight instruction loops. (Chapter 10 describes performance differences between a simple round-robin and a dual round-robin.)

3.2.1.4 Fixed-Low Priority - When a node is set to arbitrate at a fixed-low priority, it will win the bus a smaller percentage of the time than with the dual round-robin mode. Since this mode can coexist with other arbitration modes, it may be advantageous to use it in a system with nodes that are relatively latency insensitive.

digital™

3.2.1.5 Fixed-High Priority - When a node is set to arbitrate at a fixed-high priority, it will win the bus a greater percentage of the time than if the arbitration mode were dual round-robin. Nodes with critical access times can benefit by having a fixed-high priority.

3.2.1.6 Restricted Use of Arbitration Modes - The use of arbitration modes other than dual round-robin mode is prohibited. The other modes are reserved for use by Digital.

3.3 TRANSACTION CYCLES

All VAXBI transactions have three types of cycles: command/address, imbedded arbitration, and data. Figure 3-2 shows the basic format of VAXBI transactions.

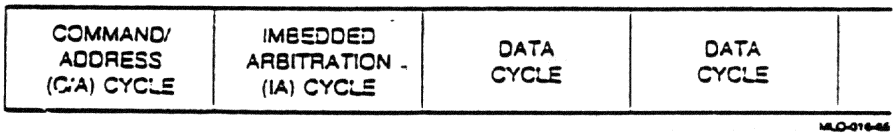


Figure 3-2: Format of VAXBI Transactions

The basic operation of the VAXBI bus is controlled by the BI NO ARB L and BI BSY L lines. These lines are used to detect the occurrence of VAXBI transaction cycles.

3.3.1 Command/Address Cycle

The command/address cycle is the first cycle of all VAXBI transactions. During this cycle the master transmits a 4-bit command code on the BI I<3:0> L lines and information required to select the appropriate slave on the BI D<31:0> L lines. This selection information can take many forms. Each transaction type uses only one form of selection information. The transactions and their corresponding selection information form are shown in Table 3-2.

digital™

One form of selection information is a 30-bit address, accompanied by a 2-bit length code. This form is used by all read-type, write-type, and INVAL transactions. The selection information can also take the form of a 16-bit destination mask in which each bit corresponds to a particular node ID. This form of selection allows for from 1 to 16 slaves to be involved in the transaction. The destination mask is used for all multi-responder transactions (except INVAL). The IPINTR transaction uses the destination mask along with the decoded master ID to select the proper slave(s). The IDENT transaction uses a level field as the slave selection information.

Nodes identify the command/address cycle by detecting the assertion of BI BSY L in a cycle following one in which BI NO ARB L was in the asserted state.

Table 3-2: Command/Address Format by Transaction

Transaction	BI D<31:16> L	BI D<15:0> L
Read-type	Length code and 30-bit address	
Write-type	Length code and 30-bit address	
INVAL	Length code and 30-bit address	
IPINTR	Decoded master ID	Destination mask
INTR	Level	Destination mask
STOP	RESERVED	Destination mask
BDCST	RESERVED	Destination mask
IDENT	Level	RESERVED

3.3.2 Imbedded Arbitration Cycle

The second cycle of a transaction is called the "imbedded arbitration cycle." During this cycle the master transmits its encoded ID on the BI I<3:0> L lines, and the VAXBI data path is available for arbitration by other nodes (except in burst mode).

3.3.3 Data Cycles

Data cycles follow the imbedded arbitration cycle. All transactions include at least one data cycle. A data cycle is a cycle in which the VAXBI data path is reserved for transferring data (such as read or write data, as opposed to command/address or arbitration information) between the master and slave(s). Table 3-3 shows the type of data transferred during data cycles of the different kinds of transactions.

digital™

The number of data cycles in a read- or write-type transaction depends on both the length of the transfer and the number of STALL responses issued by the slave. For IDENT transactions the number of data cycles depends only on the number of STALLs issued by the slave. All multi-responder transactions (except for BDCST) have only a single data cycle (this data cycle is currently a RESERVED cycle). For BDCST transactions the number of data cycles depends only on the length of the transfer. (BDCST data cycles cannot be stalled.)

Table 3-3: Data Transferred During Data Cycles

Transaction	BI D<31:0> L	BI I<3:0> L
Read-type	Read data	Read status
WRITE	Write data	RESERVED
WCI	Write data	RESERVED
WMCI	Write data	Write mask
UWMCI	Write data	Write mask
INVAL	RESERVED	RESERVED
IPINTR	RESERVED	RESERVED
INTR	RESERVED	RESERVED
STOP	RESERVED	RESERVED
BDCST	BDCST data	RESERVED
IDENT	Interrupt vector	Vector status

digital™

CHAPTER 4

VAXBI SIGNALS

The VAXBI bus consists of 52 signals. As shown in Figure 4-1, these lines can be divided by function into four categories:

- o 37 data path signals
- o 5 synchronous control signals
- o 4 clock signals
- o 6 asynchronous control signals

All signal lines except the asynchronous control signals are synchronous and are asserted on a transmit clock's leading edge. Table 4-1 briefly describes the VAXBI signals.

For a given line, each VAXBI open drain or open collector driver is electrically connected. This type of connection produces a wired-OR signal. That is, since VAXBI signals are defined to assert low true, if any VAXBI driver on a particular line asserts, then the corresponding VAXBI signal as observed at every VAXBI node tied to that line is said to be asserted. Conversely, no VAXBI signal can be said to be deasserted unless all drivers on that particular line are deasserted. When no drivers are asserted, a terminator network defaults a line to the deasserted state.

Also included on the VAXBI bus are power and ground lines. Each slot in a VAXBI system provides access to its own unique backplane ID plug.

digital™

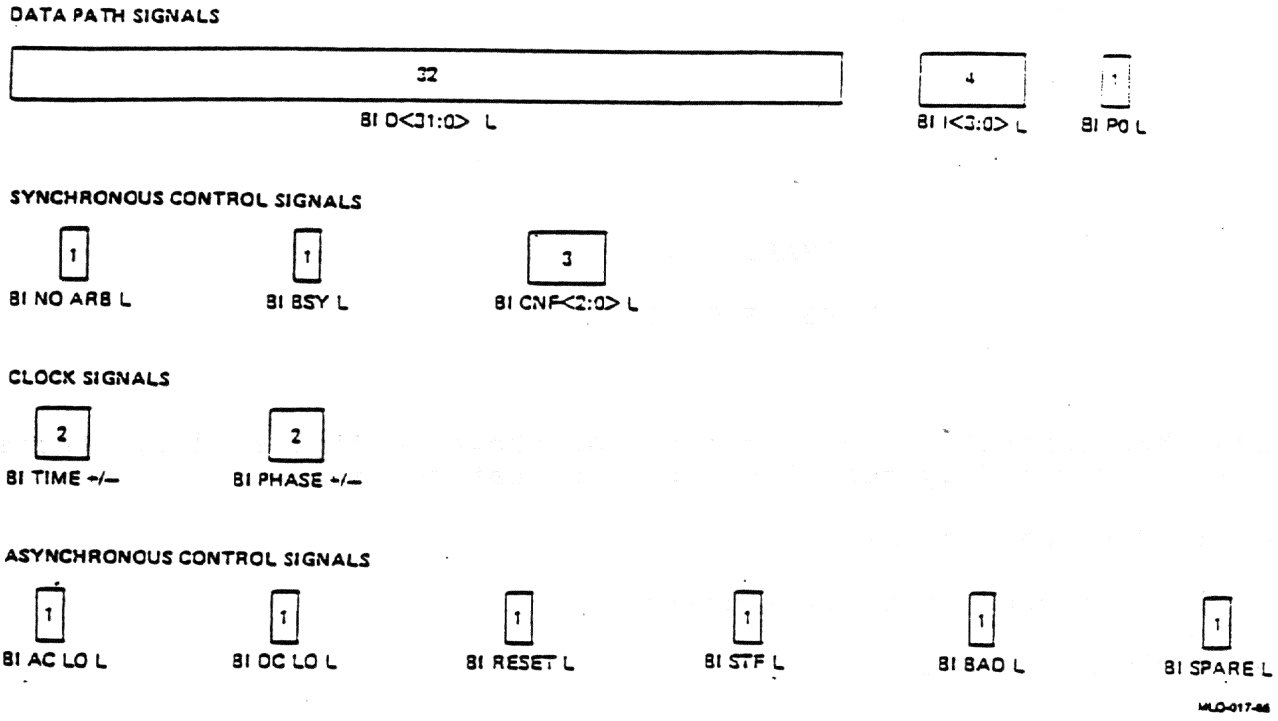


Figure 4-1: VAXBI Signals

digital™

Table 4-1: VAXBI Signals

Signal Name	Number/ Type	Description
BI D<31:0> L	32/OD	Used for the transfer of addresses and data and for arbitration.
BI I<3:0> L	4/OD	Carry commands, encoded master IDs, read status codes, and write masks.
BI P0 L	1/OD	Carries the parity for the D and I lines; asserted if the number of asserted bits on the D and I lines is an even number (ODD parity).
BI NO ARB L	1/OD	Used to inhibit arbitration on the BI D lines; also asserted during BIIC self-test to prevent other nodes from starting transactions until all nodes are ready to participate.
BI BSY L	1/OD	Used to indicate that a transaction is in progress.
BI CNF<2:0> L	3/OD	Used to send responses for command and data cycles.
BI AC LO L	1/OD	Used with BI DC LO L to perform power sequences.
BI DC LO L	1/OD	Used with BI AC LO L to perform power sequences.
BI TIME + BI TIME -	2/DECL	A 20 MHz clock reference used with BI PHASE +/- to generate all required timing signals.
BI PHASE + BI PHASE -	2/DECL	A 5 MHz clock reference used with BI TIME +/- to generate all required timing signals.
BI STF L	1/OC	A static control line used to enable a faster VAXBI system self-test.
BI BAD L	1/OC	Used for reporting node failures.
BI RESET L	1/OC	Used for initiating a VAXBI system reset.
BI SPARE L	1/-	Reserved for use by DIGITAL.

Key to abbreviations:
OD open drain
OC open collector
DECL differential ECL

digital™

4.1 DATA PATH SIGNALS

The VAXBI data path signals include:

- o BI D<31:0> L -- data lines
- o BI I<3:0> L -- information lines
- o BI P0 L -- parity line

All arbitration and transfers of commands, addresses, and data occur over these signal lines. These lines carry different information depending on the particular cycle and transaction type. See Chapter 5 for details on the use of these lines.

4.1.1 BI D<31:0> L

These are the VAXBI data lines. All address and data transfers and arbitration sequences occur on these lines.

4.1.2 BI I<3:0> L

These lines carry commands, encoded master IDs, read status codes, and write masks.

4.1.3 BI P0 L

This signal carries the parity of the BI D<31:0> L and BI I<3:0> L lines. (See Section 11.2.1.1 on parity checking and generation.)

4.2 SYNCHRONOUS CONTROL SIGNALS

The VAXBI synchronous control signals include:

- o BI NO ARB L
- o BI BSY L
- o BI CNF<2:0> L

BI NO ARB L and BI BSY L are the primary control signals on the VAXBI bus. The BI CNF<2:0> L lines carry confirmation codes that provide "handshakes" between the master and slave nodes.

4.2.1 BI NO ARB L (No Arbitration)

The BI NO ARB L signal is used to control access to the VAXBI data lines for arbitration. If BI NO ARB L is asserted in a given VAXBI cycle, then nodes may not arbitrate during the next VAXBI cycle. Nodes monitor the BI NO ARB L signal so that data and command/address information do not contend with arbitration information.

The BI NO ARB L signal is asserted by the following:

- Nodes arbitrating for the bus during the arbitration cycle
- The pending bus master from the cycle after it wins the arbitration until it becomes bus master
- The bus master during the following cycles of its transaction:

Transaction Length	Cycles
Longword	Imbedded ARB
Quadword	Imbedded ARB and following cycle
Octaword	Imbedded ARB through the cycle after the second ACK data cycle

- The slave for all data cycles except the last
- All potential slaves for the third (decoded master ID) cycle of an IDENT command and for the IDENT arbitration cycle of an IDENT command
- Nodes doing loopback transactions (see Section 4.2.3.2)
- The bus master during its command/address cycle to prevent bus arbitration from occurring, so it can start another bus transaction following the current one. This mode of operation, called "burst mode," is reserved for use by Digital.
- Nodes during their power-up self-test, until the VAXBI registers can be accessed

digital™

4.2.2 BI BSY L (Busy)

The BI BSY L signal is used to provide the orderly transition of bus mastership from one node to another. Nodes monitor the BI BSY L signal to determine the action that should be taken during the following cycle. The node that won the last arbitration may become bus master in the cycle following one in which it detects the deasserted state of BI BSY L (deassertion of BI BSY L means a transaction has ended). The new master asserts BI BSY L on the first cycle of the new transaction.

The BI BSY L signal is asserted by the following:

- The bus master during the following cycles of its transaction:

Transaction Length	Cycles
Longword	Command/address, imbedded ARB
Quadword	Command/address, imbedded ARB, and following cycle
Octaword	Command/address, imbedded ARB through the cycle after the second ACK data cycle

- A node to delay the start of the next bus transaction until it is prepared to respond to another bus transaction. A timeout limits any node from extending BI BSY L in this way for more than 127 consecutive cycles. Cycles of this type are referred to as "busy extension cycles." Nodes should not extend BI BSY L for more than 16 consecutive cycles. (Section 10.2.1 explains these requirements.)

- The slave for all data cycles except the last

- Nodes doing loopback transactions (see Section 4.2.3.2)

digital™

4.2.3 Use of BI NO ARB L and BI BSY L

Figure 4-2 shows which nodes assert BI NO ARB L and BI BSY L during each cycle of a transaction. Figure 4-3 shows the state sequences of BI NO ARB L and BI BSY L that can occur.

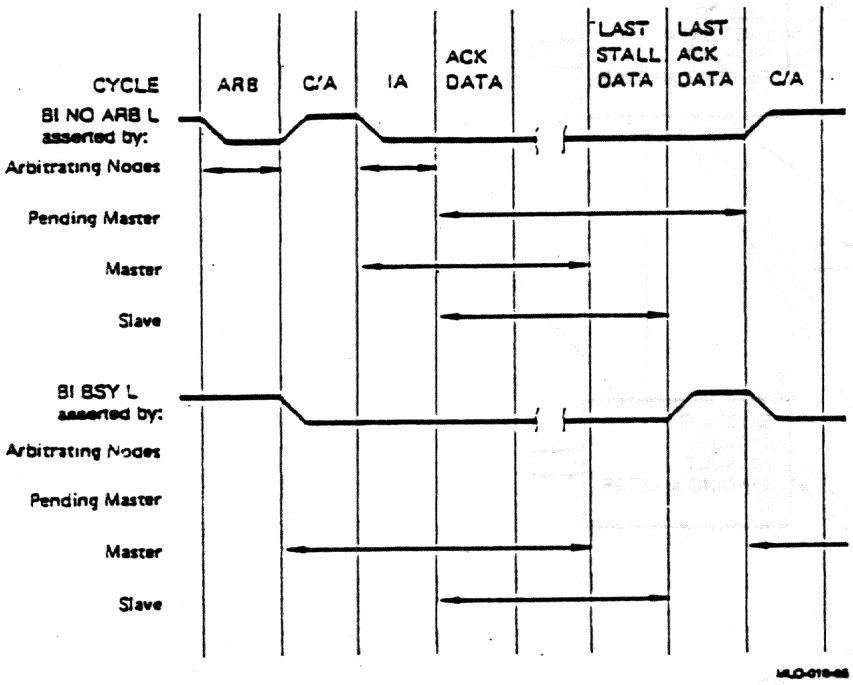


Figure 4-2: Transaction Showing BI NO ARB L and BI BSY L

digital™

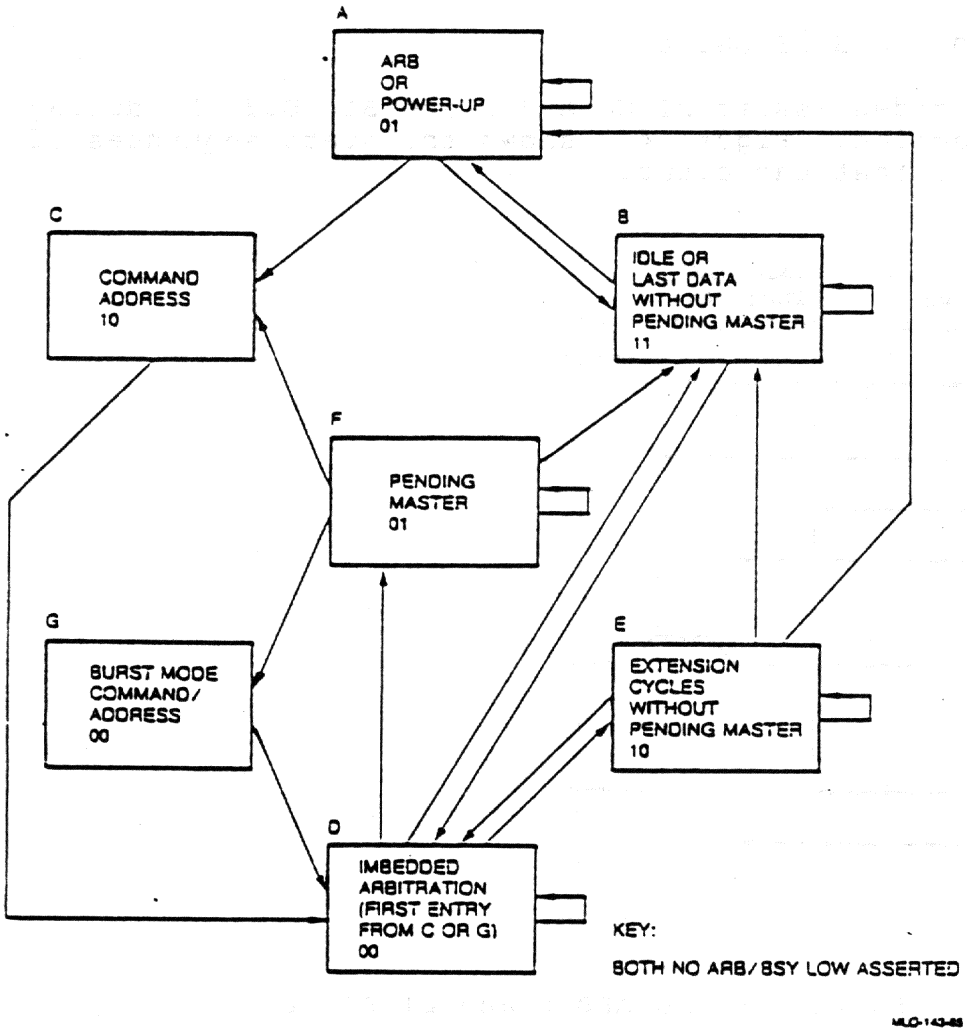


Figure 4-3: State Sequences of BI NO ARB L and BI BSY L

4.2.3.1 Arbitration State - Figure 4-4 shows the state diagram for a node's arbitration control circuitry. Each state represents one bus cycle. The BI NO ARB L signal is asserted in all states except the idle state.

When in the idle state, a node waits for a request to transfer information over the bus. When a request is received, the node enters the arbitration cycle state as soon as the data lines are free, as indicated by a deasserted BI NO ARB L signal. The node then asserts the bit corresponding to its node ID in either the low-priority word or the high-priority word. The node compares the received data lines with the bit that it asserted. If the node is not the highest priority, it returns to the idle state and waits for the next arbitration cycle. If it is the highest priority, and if no bus

transaction is in progress (as indicated by a deasserted BI BSY L signal), it enters the master state. If a bus transaction is in progress, the node goes into the pending master state and waits for the bus to become available.

During the first cycle of a node's bus transaction, the node is in the master state. The BI BSY L signal is asserted, along with the data and information lines that carry the command and address. Control is passed to the master control circuitry. The request condition is cleared during this state.

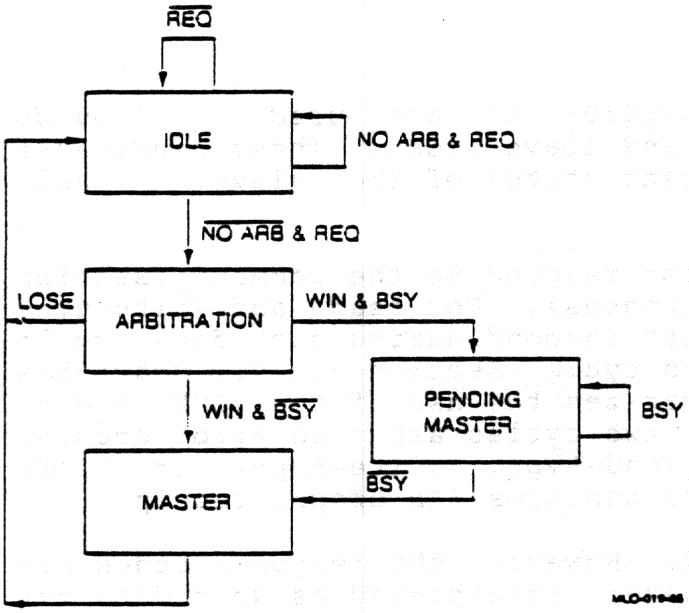


Figure 4-4: Arbitration State Diagram

4.2.3.2 Loopback Transactions - To perform loopback transactions, a node must monitor the state of BI NO ARB L and BI BSY L. A node can start a loopback transaction only if there is no chance that it will be selected by a VAXBI transaction. This is ensured by requiring that nodes only start loopback transactions when the next cycle cannot be a VAXBI command/address cycle.

digital™

In the following cases the next cycle cannot be a VAXBI command/address cycle:

- BI NO ARB L is deasserted (indicates that no node is arbitrating and that there is no pending master).
- BI BSY L is asserted (indicates that a transaction is on the VAXBI bus; as long as the transaction is not targeted at this node, it may initiate a loopback transaction).

See Section 4.2.2 for restrictions on asserting BI BSY L.

4.2.4 BI CNF<2:0> L (Confirmation)

The confirmation signal lines (BI CNF<2:0> L) are used to provide "handshakes" between the master and slave nodes. These handshakes reflect detected errors and the current status of the slave. (Table 4-2 lists the response codes.)

During a transaction a node must first respond to the command (Section 4.2.4.1 describes the command responses). For read- and write-type and IDENT transactions, the slave must respond during each data cycle following the command confirmation cycle (Section 4.2.4.2 describes the data responses). Table 4-3 summarizes the use of the CNF codes. Note that error feedback occurs two cycles after an error occurs. During the two cycles following a read-type, write-type, or IDENT transaction, the receiver of the data confirms its proper receipt.

The CNF lines are not parity checked. However, the response codes are assigned so that bad data is never interpreted as good data for single-bit failure cases (see Table 4-2).

Table 4-2: Response Codes

BI CNF<2:0> L	
2 1 0	Description
H H H	NO ACK
H H L	Illegal
H L H	Illegal
H L L	ACK
L H H	Illegal
L H L	STALL
L L H	RETRY
L L L	Illegal

digital™

4.2.4.1 Command Responses - The ACK, NO ACK, RETRY, and STALL responses are permitted for single-responder commands. The slave sends the command confirmation response during the third cycle of all transactions, except for IDENTs. Command responses for IDENT transactions are sent in the fifth cycle.

Only the ACK and NO ACK responses are permitted for multi-responder commands (INTR, IPINTR, INVAL, STOP, and BDCST). An ACK response indicates that at least one node has responded to the command.

ACK (Acknowledge) Response -- The node selected to respond to a command returns ACK to indicate that it is capable of executing the command at this time. For multi-responder commands, the receipt of an ACK response indicates that at least one node has been selected by the current transaction.

Masters always presume acknowledgment and send data for write-type and BDCST commands.

NO ACK (No Acknowledgment) Response -- The NO ACK response to commands indicates that no node has been selected. Either no node is available or an error has occurred during the command/address cycle. The deasserted state of the three confirmation lines produces the NO ACK code.

RETRY Response -- If a node cannot immediately execute the command sent to it, it returns the RETRY response. A response of this type may be expected from a node:

- That is still locked from an IRCI command
- That has been locked from another port
- That is a bus adapter whose target bus is busy and it is waiting for a transfer path to the VAXBI (the deadlock case)
- That must perform a long internal sequence in response to a STOP command
- That must perform a long internal initialization sequence following the deassertion of BI DC LO L

A node should not return a RETRY response if it will be busy for a short period of time such as during a memory refresh or the completion of a memory write access. The STALL response is the proper action for those cases.

digital™

All masters are required to implement a retry timeout. If a master cannot complete a transaction within 4096 attempts, it must log this as an error condition.

Application Note 5 discusses use of the RETRY response code.

STALL Response -- The STALL response from a node indicates that it needs additional time. It may need more time to acknowledge the command sent to it, to return the first data word on read-type commands or vector data on an IDENT command, or to accept data words on write-type commands. The STALL response is not permitted for multi-responder commands. A node may not send a STALL response to delay the recognition of proper address range. Therefore, a node can use STALL preceding a NO ACK response only when an address allocated to the node does not correspond to an implemented register or memory location.

The STALL response can be sent by the following:

- Nodes that take longer than one cycle to perform a read or a write
- Memories that are selected during a refresh sequence
- Memories that are to receive write data, when their write buffer is full
- Adapters that need to synchronize with the protocol of another bus

An ACK, NO ACK, RETRY, or another STALL response is permitted after a STALL single-responder command confirmation.

All nodes must implement a stall timeout that will force a slave node to release the bus if the node attempts to stall for more than 127 consecutive cycles. At a stall timeout, the slave sets the STALL Timeout bit in the Bus Error Register and deasserts all bus lines. The master interprets the deasserted CNF lines as a NO ACK response and terminates its involvement in the transaction.

4.2.4.2 Data Responses - A slave must transmit an ACK, NO ACK, or STALL response for each data cycle after the command confirmation cycle of data transfer commands (STALL, however, is not permitted for BDCST). During the two cycles following the last data cycle of a data transfer command, the node(s) receiving the data must respond with either an ACK or a NO ACK.

ACK Data Response -- The slave or slaves send the ACK response during data cycles to indicate that no error has been detected and that the cycle is not to be stalled.

An ACK response is also returned by the node receiving the data during each of the two cycles following the last data cycle of a successful transaction. These ACK responses are sent by the master for read-type and IDENT transactions and by the slave(s) for write-type and BDCST transactions.

Receipt of the final ACK response indicates to the node that transmitted the data that the transaction has completed successfully.

NO ACK Data Response -- The NO ACK response indicates that an error has been detected. The response is returned by either the master or slave when an error in a transaction is discovered. A node that detects an error must transmit only NO ACK responses for the remainder of the transaction.

STALL Data Response -- A slave can send a STALL response to delay the transmission of data. The cycle is repeated until the STALL response is removed.

During read-type transactions, a slave can stall any data cycle by returning a STALL response in place of the data. The vector cycle during an IDENT transaction can be stalled in the same manner. For read-type transactions, the master inhibits a parity check on STALL data cycles, since the BI I<3:0> L and BI D<31:0> L lines are UNDEFINED fields during these cycles. For write-type transactions, however, slaves must check parity on STALL data cycles. The STALL response is not permitted for BDCST data cycles.

The master deasserts BI BSY L and BI NO ARB L on the last expected cycle of a bus transaction. It has no way of knowing whether that cycle may be stalled. However, the VAXBI bus will remain dedicated to this transaction, since the slave asserts BI BSY L and BI NO ARB L for all STALL data cycles, as well as all ACK data cycles except the last.

All nodes must implement a stall timeout that will force a slave node to release the bus if the node attempts to stall for more than 127 consecutive cycles. At a stall timeout the slave node sets the STALL Timeout bit in the Bus Error Register and deasserts all bus lines. The master interprets the deasserted CNF lines as a NO ACK response and terminates its involvement in the transaction.

Table 4-3: Meaning and Use of Response (CNF) Code

Multiple-Responder Transactions (except BDCST)

	C/A	IA	D1
Confirmation Type			CR
Error Feedback			C/A
Slave Status			D1
Source			S
Permitted Response			AN

Write-Type Transaction (and BDCST)*

Octaword Length

	C/A	IA	D1	D2	D3	D4		
Confirmation Type			CR	DR	DR	DR	DR	DR
Error Feedback			C/A	NA	D1	D2	D3	D4
Slave Status			D1	D2	D3	D4	NA	NA
Source			S	S	S	S	S	S
Permitted Response			ASRN	ASN	ASN	ASN	AN	AN

Quadword Length

	C/A	IA	D1	D2				
Confirmation Type			CR	DR	DR	DR		
Error Feedback			C/A	NA	D1	D2		
Slave Status			D1	D2	NA	NA		
Source			S	S	S	S		
Permitted Response			ASRN	ASN	AN	AN		

Longword Length

	C/A	IA	D1					
Confirmation Type			CR	DR	DR			
Error Feedback			C/A	NA	D1			
Slave Status			D1	NA	NA			
Source			S	S	S			
Permitted Response			ASRN	AN	AN			

*The CNF codes are used similarly for write-type and BDCST transactions except that the STALL response is not permitted for BDCST transactions.

Abbreviations for each category:
Confirmation Type: CR = command response, DR = data response.
Error Feedback: The cycle for which the error feedback is given; for example, C/A = command/address, D1 = the first data cycle. NA = not applicable.
Slave Status: The cycle for which the slave is reporting its status. NA = not applicable.
Source: S (slave) and M (master) identify the node sending the CNF code.
Permitted Response: A = ACK, N = NO ACK, S = STALL, R = RETRY.

digital™

Table 4-3: Meaning and Use of Response (CNF) Codes (Cont.)

Read-Type Transactions

Octaword Length

	C/A	IA	D1	D2	D3	D4		
Confirmation Type			CR	DR	DR	DR	DR	DR
Error Feedback			C/A	NA	NA	NA	D1-D3	D4
Slave Status			D1	D2	D3	D4	NA	NA
Source			S	S	S	S	M	M
Permitted Response			ASRN	ASN	ASN	ASN	AN(1)	AN(1)

Quadword Length

	C/A	IA	D1	D2				
Confirmation Type			CR	DR	DR	DR		
Error Feedback			C/A	NA	D1	D2		
Slave Status			D1	D2	NA	NA		
Source			S	S	M	M		
Permitted Response			ASRN	ASN	AN(1)	AN(1)		

Longword Length with STALLs

	C/A	IA	STALL D1	STALL D1	ACK D1			
Confirmation Type			CR	CR	CR	DR	DR	
Error Feedback			C/A	NA	D1	NA	NA	
Slave Status			D1	D1	D1	NA	NA	
Source			S	S	S	M	M	
Permitted Response			ASRN	ASRN	ASRN	AN(1)	AN(1)	

IDENT Transaction

	C/A	IA	DMID	IDENT ARB	VECTOR		
Confirmation Type					CR	DR	DR
Error Feedback			C/A	NA	C/A,DMID	NA	VECTOR
Slave Status			D1	D1	VECTOR	NA	NA
Source			S	S	S	M	M
Permitted Response			-	-	ASRN	AN(2)	AN(2)

NOTES

1. The master sends an ACK only if it did not detect a transmit check error during C/A and data cycles.

2. The master sends an ACK only if it did not detect a transmit check error during C/A and DMID cycles.

digital™

4.3 CLOCK SIGNALS

The VAXBI clock signals include:

- o BI TIME + and BI TIME -
- o BI PHASE + and BI PHASE -

See Chapter 12, Electrical Specification, for detailed clock specifications.

4.3.1 BI TIME + and BI TIME -

The BI TIME + and BI TIME - signals are a pair of 20 MHz differential ECL square waves that are input to a clock receiver at each node. These signals and BI PHASE +/- provide the reference for timing at each node.

4.3.2 BI PHASE + and BI PHASE -

The BI PHASE + and BI PHASE - signals are a pair of 5 MHz differential ECL square waves that are input to a clock receiver at each node. These signals and BI TIME +/- provide the reference for timing at each node.

4.4 ASYNCHRONOUS CONTROL SIGNALS

The following control signals are asynchronous to the VAXBI clock signals:

- o BI AC LO L
- o BI DC LO L
- o BI RESET L
- o BI STF L
- o BI BAD L
- o BI SPARE L

These signals are not limited to VAXBI backplane and cable extensions and may be extended off the backplane to other points in the system.

digital™

The BI AC LO L and BI DC LO L signals are used to control power-up and power-down sequences, which guarantee sufficient time for the system to store (on power-down) and then retrieve (on power-up) the parameters required for continued operation. In the descriptions of these signals, the term "DC power" is used to indicate only that DC power which may cause bus control logic, drivers, receivers, and terminators to cease to meet their electrical specifications, thereby rendering the bus inoperable. The DC power tolerance requirements are in Section 13.1.8. The BI RESET L signal is used with the BI AC LO L and BI DC LO L signals to provide the facility for simulating a power-up initialization in the system. (See Chapter 6, Initialization, for a detailed description on the use of these signals.)

The BI STF L signal is used to control the length of self-test. The BI BAD L signal is used to indicate various node failures.

4.4.1 BI AC LO L

The BI AC LO L signal is asserted when the line voltage is below minimum specifications. The deassertion of BI AC LO L indicates that processors and adapters may access memory and begin execution. The full description of BI AC LO L appears in Section 6.3.

4.4.2 BI DC LO L

The BI DC LO L signal warns of the impending loss of DC power and is used for initialization on power restoration. Specifically, a node must use the BI DC LO L signal to force its circuitry into an initialized state. VAXBI node designs must not use other reset methods such as the "RC time constant type." Following the deassertion of BI DC LO L, nodes run their internal self-tests. The full description of BI DC LO L appears in Section 6.4.

4.4.3 BI RESET L

The BI RESET L signal is asserted by nodes that need to initialize the system to the power-up state. BI RESET L is received by a device called a "reset module" which, following the assertion of BI RESET L, sequences BI AC LO L and BI DC LO L just as in the case of a true power-down/power-up sequence. See Sections 6.2.2 and 6.5 for more detail on BI RESET L and its use with reset modules.

digital™

4.4.4 BI STF L (Self-Test Fast)

The BI STF L signal is used to control the length of self-test. If BI STF L is in the asserted state when BI DC LO L is asserted, nodes will execute a fast self-test. (See Section 11.1.5 and Application Note 4 for details on the use of this signal.)

4.4.5 BI BAD L

The BI BAD L signal is used for reporting the failure of a node in a VAXBI system. BI BAD L is asserted by a node if it fails its self-test or if the node fails any time after the power-up self-test.

The BI BAD L signal may be synchronously or asynchronously asserted. BI BAD L is deasserted only when all nodes have passed self-test. (See Section 11.1.4 and Application Note 4 for details on the use of this signal.)

4.4.6 BI SPARE L

The BI SPARE L signal is reserved for use by Digital.

digital™

CHAPTER 5

VAXBI TRANSACTIONS

This chapter describes the transactions that the VAXBI bus supports and gives requirements for their use. Section 5.1 defines the two major classes of transactions. Section 5.2 discusses how the VAXBI bus provides for interprocessor communication. Section 5.3 first presents general information needed for understanding the transactions that perform data transfers. The transactions themselves are then presented in the following order: the write-type transactions (WRITE, WCI, WMCI, and UWMCI) and then the read-type transactions (READ, RCI, and IRCI). The INVAL transaction is included with the data transfer transactions. Section 5.4 discusses the transactions that support interrupts: INTR, IDENT, and IPINTR. Finally, Section 5.5 deals with the STOP transaction, which is used for diagnosing node and bus failures.

5.1 SINGLE-RESPONDER AND MULTI-RESPONDER TRANSACTIONS

VAXBI transactions can be directed at one node -- single-responder transactions -- or at multiple nodes -- multi-responder transactions.

Single-responder transactions cause data to be transferred between a master and a single slave. The master targets a node to be slave by means of a 30-bit address. The node at that address uses other information transmitted during the command/address cycle (command and data length) in determining if it will become slave.

Multi-responder transactions can be directed at more than one node and allow for more than one responder. The master sends a destination mask instead of an address. These multi-responder transactions are INTR, IPINTR, INVAL, STOP, and BDCST. INTRs are generated by means of a command message from an interrupting master to an interrupt fielding slave or set of slaves. The IPINTR command is used to interrupt other processors. The INVAL command is used to notify nodes with cache memory that they may have cached data that is no longer valid. The STOP command is used for error diagnosis. The BDCST command, which is reserved for use by Digital, permits the systemwide broadcast of information (see Appendix A for a description of the BDCST command).

digital™

Table 5-1 lists the VAXBI command codes.

Table 5-1: VAXBI Command Codes

BI I<3:0> L 3 2 1 0	Type	Name	Description
H H H H	--	--	RESERVED
H H H L	SR*	READ	
H H L H	SR	IRCI	Interlock Read with Cache Intent
H H L L	SR	RCI	Read with Cache Intent
H L H H	SR	WRITE	
H L H L	SR	WCI	Write with Cache Intent
H L L H	SR	UWMCI	Unlock Write Mask with Cache Intent
H L L L	SR	WMCI	Write Mask with Cache Intent
L H H H	MR	INTR	Interrupt
L H H L	SR	IDENT	Identify
L H L H	--	--	RESERVED
L H L L	--	--	RESERVED
L L H H	MR	STOP	
L L H L	MR	INVAL	Invalidate
L L L H	MR	BDCST**	Broadcast (RESERVED)
L L L L	MR	IPINTR	Interprocessor Interrupt

*SR = single responder; MR = multi-responder.

**See Appendix A.

5.2 INTERPROCESSOR COMMUNICATION

The interlock transactions (IRCI and UWMCI) and interprocessor interrupts (IPINTRs) support interprocessor communication. Interlock commands allow processors to communicate by exchanging messages deposited in a shared memory. Accesses to the shared memory must be synchronized because one processor's memory accesses may be interspersed in time with another processor's accesses to the same locations. Software-level synchronization is usually achieved with the use of indivisible operations such as the VAX interlock and queue instructions. These operations are implemented by using the VAXBI interlock transactions IRCI and UWMCI.

digital™

5.2.1 IPINTR Transactions

Interprocessor interrupts, in which one processor interrupts the other, is a simpler method of interprocessor communication. A combination of shared memory and interprocessor interrupts can also be used. For example, one processor can deposit a message in a specific area of shared memory and then notify the other processor by sending an IPINTR transaction. (See Section 5.4.3 for details on the IPINTR transaction.)

5.2.2 VAXBI Requirements for Interlock Transactions

Processor nodes and adapters use the VAXBI interlock transactions IRCI and UWMCI to carry out indivisible operations. The interlock feature of these transactions must be implemented for all memory space addresses but may or may not be implemented for I/O space addresses. When the interlock feature is not implemented, IRCI must have the same effect as READ and RCI, and UWMCI must have the same effect as WCI, WMCI, and WRITE (with the possible exception of the write mask).

An IRCI transaction that "locks" a block of addresses must always be paired with a subsequent UWMCI transaction that "unlocks" the block. A node must issue a UWMCI transaction as soon as possible after issuing an IRCI. If another VAXBI node issues an IRCI to a locked location, that node will receive a RETRY response. If the node continues to repeat the transaction and the lock is not cleared, a retry timeout error will occur.

Note that, in the case of VAX queues, a secondary lock exists in the queue header. The secondary lock should be examined with an IRCI and set with a UWMCI before the queue is manipulated. After the secondary lock is set, processing of the queue can be performed without using the interlock transactions. The timing consideration therefore applies only to the time required to set the secondary lock, without waiting to determine if the secondary lock was originally set. This can help reduce the time between the IRCI and the UWMCI.

In memory space, read- and write-type transactions other than IRCI and UWMCI, such as RCI and WMCI, must not be affected by the lock and must be able to proceed unhindered. Since the block is locked only to nodes issuing IRCI transactions, whether a large or small block is locked in general should not affect system operation. In I/O space, whether any transaction type is affected by the lock is implementation dependent.

digital™

The length of a block of data fetched from memory may differ from that requested by a node in an IRCI transaction because of cacheing requirements. For example, a processor with cache may have filled a cache block that is longer than the length of the block to be unlocked by the UWMCI, and the address of the block to be unlocked may be different from that given in the IRCI. The master must comply with the following rules; the slave need not check for compliance.

- The length of a UWMCI transaction can be less than the length of the IRCI transaction, but it cannot be greater.
- The UWMCI transaction must unlock the block of addresses locked by the IRCI transaction.
- The address of the UWMCI transaction must be within the address range of the IRCI transaction; it does not have to be the same.

One processor may use IRCIs of one length while another processor uses IRCIs of a different length.* For all processors to be compatible, the following must be observed:

- In memory space, an IRCI must lock a naturally aligned block that is at least an octaword long.**
- In I/O space, an IRCI locks as little as an aligned longword, except when the location is in a word-accessible or byte-accessible adapter, in which case IRCI locks as little as an aligned word or byte respectively. (See Section 5.3.1, Address Interpretation, for the meaning of word-accessible and byte-accessible adapters.) For example, the UNIBUS adapter is a word-accessible adapter.

The IRCI transaction also sets the Unlock Write Pending (UWP) bit of the VAXBI Control and Status Register (VAXBICSR) at the issuing node. A UWMCI clears this bit. If a UWMCI is issued and the UWP bit is not set (that is, an IRCI had not been issued), the Interlock Sequence Error (ISE) bit is set in the node's Bus Error Register. Setting of the ISE bit generates an error interrupt if the Hard Error Interrupt Enable bit is set in the VAXBICSR register. The UWMCI transaction is

---

*For example, the KA820 processor uses octaword IRCIs (because of its cache) while the KA800 processor uses longword IRCIs.

**In MS820 series memories, the lock will lock the entire memory node.

digital™

carried out regardless of whether the UWP bit is set. IRCI and UWMCI transactions should always be issued in pairs, and such pairs should not be nested, because of the uncertainty as to the extent of the block that is locked by any one IRCI.

VAX interlock instructions are not the only VAX instructions that generate VAXBI interlock transactions: byte- and word-length modify type instructions in I/O space may also generate interlock transactions. All these instructions generate IRCI/UWMCI pairs.

5.3 TRANSACTIONS TO SUPPORT DATA TRANSFER

This section describes the read-type and write-type commands and the INVAL command. During a command/address cycle the data lines specify the number of bytes being transferred (on BI D<31:30> L; see Table 5-2) and a 30-bit address (on BI D<29:0> L). The low address of the block of data transferred is always a multiple of the size of the block of data, in bytes. Note that during read-type transactions, the address supplied during the command/address cycle is not always the low address of the block of data transferred. (See Section 5.3.1.1.) The information lines (BI I<3:0> L) carry the VAXBI command code during the command/address cycle (see Table 5-1).

Table 5-2: Data Length Codes

BI D<31:30> L			
31	30	Data Length	
H	H	RESERVED	
H	L	Longword (LW)	4 bytes
L	H	Quadword (QW)	8 bytes
L	L	Octaword (OW)	16 bytes

5.3.1 Address Interpretation

The following two subsections give rules for data transmission based on the transaction type, address space, data length field, and low-order address bits. Figure 5-1 shows longword and byte references in an octaword block.

digital™

The abbreviations used in Tables 5-3 and 5-4 are explained below:

ALL All address space; includes all of I/O and memory space.

NWS Non-window space; includes all I/O addresses that are not in node window space as well as all memory space addresses.

WS Node window space.

/B The node is byte-accessible; that is, longword read-type commands are treated by the node as reads of single bytes.

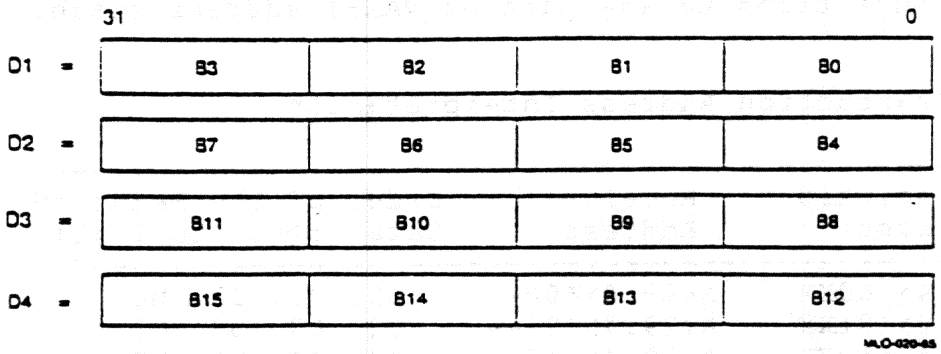
/W The node is word-accessible; that is, longword read-type commands are treated by the node as reads of single words.

/L The node is longword-accessible, that is, the smallest unit that can be read from the node with a VAXBI read-type transaction is a longword. Nodes that are not explicitly specified as byte- or word-accessible are longword-accessible.

X An X in the address field indicates that the master can drive any data on these lines during the command/address cycle.

- A dash in a received address entry indicates bits that the slave must ignore for a particular transaction length. A dash in returned read data indicates bytes that must be ignored by the master (that is, the bytes contain undefined data).

' An apostrophe indicates concatenation.



Address of B0 is:

A<29:2>'00 for longword data length
A<29:3>'000 for quadword data length
A<29:4>'0000 for octaword data length

Figure 5-1: Longword and Byte References in an Octaword Block

5.3.1.1 Read-Type Transactions - Table 5-3 describes the rules for address interpretation for VAXBI read-type transactions. The order in which the longwords of data are returned is shown in the last column.

No masters may generate the RESERVED (H H) data length code, and the response by nodes that receive the RESERVED data length code is implementation dependent.

The slave to a read-type transaction transmits the addressed longword of data first. The way in which the remaining longwords are transmitted depends on the address that was transmitted. In most cases, the address transmitted during a read-type transaction will be data-length aligned (for example, if the transaction is of octaword length and address bits A<3:0> = 0000). In these cases, the remaining longwords (one for quadword and three for octaword length transactions) will be transmitted in ascending address order. However, if the initially addressed longword was not data-length aligned (for example, an octaword transaction with address bits A<3:0> = 1000), then the remaining longwords will be transmitted in ascending address order until the top of the data-length aligned block is reached, at which time a "wrap" will occur, and the next transferred longword will be located at the base address of the data-length aligned block. Longwords are then transferred in ascending address order until the entire block has been transferred. A read-type transaction in which the address is not data-length aligned is called a "wrapped read."

digital™

No slave should rely on masters having the capability to perform quadword or octaword transactions to any part of VAXBI address space.

Table 5-3: Read-Type Transaction Address Interpretation

Data Length	Address Space	Transmitted Address	Received Address	Order of the Returned Data (first to last)
OW	ALL	A<29:4>'00XX	A<29:4>'00--	D1, D2, D3, D4
OW	ALL	A<29:4>'01XX	A<29:4>'01--	D2, D3, D4, D1
OW	ALL	A<29:4>'10XX	A<29:4>'10--	D3, D4, D1, D2
OW	ALL	A<29:4>'11XX	A<29:4>'11--	D4, D1, D2, D3
QW	ALL	A<29:3>'0XX	A<29:3>'0--	D1, D2
QW	ALL	A<29:3>'1XX	A<29:3>'1--	D2, D1
LW	NWS	A<29:2>'XX	A<29:2>'--	D1 (B3,B2,B1,B0)
LW	WS/L	A<29:2>'XX	A<29:2>'--	D1 (B3,B2,B1,B0)
LW	WS/W	A<29:2>'0X	A<29:2>'0-	D1 (XX,XX,B1,B0)
LW	WS/W	A<29:2>'1X	A<29:2>'1-	D1 (B3,B2,XX,XX)
LW	WS/B	A<29:2>'00	A<29:2>'00	D1 (XX,XX,XX,B0)
LW	WS/B	A<29:2>'01	A<29:2>'01	D1 (XX,XX,B1,XX)
LW	WS/B	A<29:2>'10	A<29:2>'10	D1 (XX,B2,XX,XX)
LW	WS/B	A<29:2>'11	A<29:2>'11	D1 (B3,XX,XX,XX)

*The slave must respond with a NO ACK.

5.3.1.2 Write-Type Transactions - Table 5-4 describes the rules for address interpretation for VAXBI write-type transactions. The order in which the longwords of data are transmitted is shown in the last column.

No masters may generate the RESERVED (H H) data length code, and the response by nodes that receive the RESERVED data length code is implementation dependent.

VAXBI writes transmit write data in ascending address order (that is, there are no wrapped writes on the VAXBI bus). As shown in Table 5-5, masters must not issue quadword write-type transactions with A<2> equal to one or octaword write-type transactions with either A<3> or A<2> equal to one.

No slave should rely on masters having the capability to perform quadword or octaword transactions to any part of VAXBI address space.

digital™

Table 5-4: Write-Type Transaction Address Interpretation

Data Length	Address Space	Transmitted Address	Received Address	Order of the Transmitted Data (first to last)
OW	ALL	A<29:4>'00XX	A<29:4>'00--	D1, D2, D3, D4
QW	ALL	A<29:3>'0XX	A<29:3>'0--	D1, D2
LW	NWS	A<29:2>'XX	A<29:2>'--	D1
LW	WS/L	A<29:2>'XX	A<29:2>'--	D1 (B3,B2,B1,B0)
LW	WS/W	A<29:2>'0X	A<29:2>'0-	D1 (--,--,B1,B0)
LW	WS/W	A<29:2>'1X	A<29:2>'1-	D1 (B3,B2,--,--)
LW	WS/B	A<29:2>'00	A<29:2>'00	D1 (--,--,--,B0)
LW	WS/B	A<29:2>'01	A<29:2>'01	D1 (--,--,B1,--)
LW	WS/B	A<29:2>'10	A<29:2>'10	D1 (--,B2,--,--)
LW	WS/B	A<29:2>'11	A<29:2>'11	D1 (B3,--,--,--)

*The slave must respond with a NO ACK.

5.3.2 Cacheing and the VAXBI Bus

In a multiprocessing system in which data from memory is cached, the danger exists that the data in the cache will be "stale," that is, not up to date. In a VAXBI system, it is required that memory must contain the up-to-date copy of the data.

To provide for the cacheing of data, the VAXBI protocol specifies various kinds of reads and writes. The assumption is that most data transfers will involve caches. RCI and IRCI are the read transactions for use with caches, while WCI, UWMCI, and WMCI are the write transactions for use with caches. The INVAL transaction is used to notify nodes that the data in their caches might be invalid. The following discussion describes the way in which these transactions are used.

digital™

To ensure that data in caches that may have become stale are marked invalid, each VAXBI node that caches data must monitor all VAXBI write-type transactions, and if such a transaction is directed to a location that is cached at the node, then the cached data must be marked invalid.* If the node cannot mark the cached data invalid within the time it takes the monitored transaction to complete, the node must extend the monitored transaction by asserting the BI BSY L signal as described in Section 4.2.2.

Because VAXBI nodes are not permitted to cache VAXBI I/O space data, there is no need to provide for invalidating I/O space locations. The discussion below regarding invalidation of cached data therefore pertains only to VAXBI memory space data. To emphasize the difference between those VAXBI nodes that implement memory locations and those that cache data, the former will be referred to as "memory VAXBI nodes" and the latter as "cache VAXBI nodes," respectively, since they are the memory nodes and cache nodes of the data transfer transactions. When memory is located at the cache node, the rules below do not apply to the node as memory node because they assume that the memory node cannot learn of actions at the cache node except through VAXBI transactions.

By monitoring all VAXBI write-type transactions, cache VAXBI nodes can ensure that, should a memory location be updated through a VAXBI transaction, the cached data will always be marked invalid. Should the memory location be updated without the use of a VAXBI transaction, however, this monitoring activity cannot detect the update. Such updating of a memory location is referred to as a "local write." To ensure that stale data is marked invalid, memory VAXBI nodes must issue INVAL transactions for a set of locations when the locations are updated with a local write, provided that the data could have been cached.

To reduce the number of INVAL transactions that have to be issued, cache VAXBI nodes should use the READ and WRITE transactions instead of the RCI and WCI transactions when they read or write data that they do not cache. As long as the READ transaction is not used, VAXBI nodes are allowed to cache any memory space data returned on read-type transactions unless the data is returned with one of the "don't cache" status codes (see Section 5.3.4). If a "don't cache" status code is returned, the memory node will not issue an INVAL transaction when it is updated with a local write, so the data must not be cached.

---

*In fact, the node could update its cache, rather than just invalidate, if the write-type command is with cache intent.

digital™

In contrast to the case of reads, VAXBI nodes are not allowed to cache data when they write data to a memory location unless at the time of the write they have been holding a valid cached copy of the original contents of that location. This is referred to as the "no write allocate" rule. Because of this rule, it is sufficient that "don't cache" be returned only on read-type transactions. Without this rule, the bus protocol would have to support a "don't cache" indication to be returned on write-type transactions. Cache nodes would then have to implement the cache in such a way that the data is not cached in case the "don't cache" indication is returned; this would be awkward since writes are often pipelined.

To take advantage of the READ and WRITE transactions, a memory VAXBI node can implement a "cached bit" for each memory location. The "cached bit" is originally cleared, and is set if the location is read with an RCI or an IRCI transaction, indicating that the data in the location might be cached. The bit is not set on a READ transaction since the data is not cached, and is not set on a write-type transaction because the "no write allocate" rule guarantees that the data is cached only if it had already been cached on an earlier RCI or IRCI transaction and was still valid. If the location is written with a local write while the cached bit is set, the memory VAXBI node must issue an INVAL transaction for the location. If the location is written with a local write while the cached bit is cleared, however, the memory VAXBI node need not issue the INVAL transaction. The cached bit can be cleared if the location is written with a WRITE transaction (but not if the location is written with a WCI, WMCI, or UWMCI transaction), and is cleared when an INVAL transaction is issued for the location.

It may not be feasible to implement a cached bit for each memory location. Instead, a cached bit can be implemented for a large block of memory locations. This bit would be set if any locations in the block are accessed with an RCI or IRCI transaction. (If "don't cache" status is returned on receipt of IRCI transactions, the cached bit need be set only on RCI transactions.) In such a case it is probably not worthwhile to ever clear the bit. Should the bit ever be set, INVAL transactions must be issued on each local write to any memory location in the block of locations. However, if the bit is never set, INVAL transactions need not be issued. Not setting the bit is useful when the node can be used both in configurations where cache nodes occur and in configurations where they do not occur. In the latter case better performance can be obtained because INVAL transactions need never be issued.

digital™

Rather than implement cached bits, memory VAXBI nodes can simply issue INVAL transactions on all local writes. This alternative is desirable when local writes are rare. Or memory VAXBI nodes can avoid ever issuing INVAL transactions on local writes, by returning "don't cache" status on RCI and IRCI transactions (and, optionally, also on READ transactions). This alternative is desirable when defeating the cache at the memory VAXBI node is judged to have a small performance impact. Application Note 2 provides an expanded discussion on the use of the various alternatives.

The rules are summarized below.

- Nodes while not cacheing data should issue READ and WRITE commands on the VAXBI bus to access locations not in local memory (that is, memory which is not part of the node itself).
- Nodes while cacheing data must use cache-intent read- and write-type commands on the VAXBI bus to access locations not in local memory.
- Nodes that cache data must monitor the VAXBI bus; locations designated in write-type commands and INVAL commands must be invalidated.
- Nodes must not cache any data returned with a "don't cache" status.*
- Nodes must not cache data on a write transaction unless, just before the data is written, the cache contained valid data for the location. This rule is known as "no write allocates."
- Nodes that respond to read- and write-type commands to memory space must either (a) issue INVAL commands on writes to local memory or (b) return "don't cache" status to RCI and IRCI commands if writes to local memory are possible to the specified locations. It is optional for these nodes to return "don't cache" status to other read-type commands.
- Reads to I/O space and all interlock reads must not result in cache hits. Even if the data being read has been cached locally, the cached data must be ignored on these transactions and a VAXBI read-type transaction must be generated.

Note that memory nodes accessible only through the VAXBI bus do not have to return "don't cache" status or issue INVAL commands, because local writes are not possible.

---

*The KA820 processor has an exception to this rule.

digital™

When applied to specific cases, the rules listed above can be simplified as follows:

- A node with no local memory (although, of course, it may have a cache) has no need to issue INVAL commands.
- A node with no cache memory should not issue cache intent commands.
- A node with local memory need not record whether RCI or WCI transactions have been issued to locations in the local memory since the last write-type transaction. If this information is not recorded, either all accesses to local memory receive "don't cache" confirmations or all local writes generate INVAL commands.

5.3.3 Write Mask

During data cycles of WMCI and UWMCI transactions, the BI I<3:0> L lines carry a write mask. When a bit in the mask is set to a one, the corresponding byte is to be modified by the contents of the data lines. Table 5-5 shows which byte of the VAXBI data lines is written to when the information lines are asserted.

In memory space, the exact bytes corresponding to the mask bits that are set must be written, for any combination of mask bits. In node window space, byte- and word-accessible nodes ignore the write mask bits for the bytes or word, respectively, not addressed by the low-order address bits. In the rest of I/O space, the effect of the mask bits is implementation dependent. See Section 5.3.1.2 for more details.

The BI I<3:0> L lines are an UNDEFINED field during data cycles of write-type transactions that do not use a mask.

Table 5-5: Write Mask Codes

Signal Asserted	Byte to Be Written
BI I<3> L	BI D<31:24> L
BI I<2> L	BI D<23:16> L
BI I<1> L	BI D<15:8> L
BI I<0> L	BI D<7:0> L

digital™

5.3.4 Read Status

The BI I<3:0> L lines are used to transmit a read status code during each ACK data cycle of read-type and IDENT transactions. The slave sends the code describing the type of data that is being returned to the master. Both the slave and master continue the bus transaction regardless of the status. Table 5-6 lists the read status codes.

Note that there are two versions for each of the three types of status. One version is for data that can be cached, and the other is for data that must not be cached. Because the slave can make this restriction, the number of INVALs can be reduced.

Table 5-6: Read Status Codes

BI I<3:0> L 3 2 1 0	Description
H * H H	RESERVED
H * H L	Read Data
H * L H	Corrected Read Data
H * L L	Read Data Substitute
L * H H	RESERVED
L * H L	Read Data/Don't Cache
L * L H	Corrected Read Data/Don't Cache
L * L L	Read Data Substitute/Don't Cache

*Bit <2> is RESERVED. Slaves must drive this line to H for all status types, and masters must ignore the state of this line.

The Read Data status code indicates that data is being returned without error.

The Corrected Read Data status code indicates that the data being returned has been corrected.

The Read Data Substitute status code warns that the data that was accessed contained an uncorrectable error. If possible, the data lines should contain the uncorrected data.

The master's response to RESERVED status codes should be the same as that to Read Data Substitute.

Parity must be generated, regardless of the data and status returned.

digital™

5.3.5 Write-Type Transactions

WRITE

The WRITE transaction is used to transfer data from a master to a slave when the master does not cache the data (see Figure 5-2*). During the command/address cycle, the master sends the data length (longword, quadword, or octaword) on BI D<31:30> L, the address on BI D<29:0> L, and the command on BI I<3:0> L. Parity is generated by the master and is checked by all nodes. BI BSY L is asserted until the last ACK data cycle. BI NO ARB L is deasserted for the C/A cycle but then is asserted, along with BI BSY L, until BI BSY L is deasserted.

During the second cycle (imbedded ARB cycle), nodes can arbitrate for control of the bus for the next transaction. The present master cannot participate.

The slave sends a command confirmation during the third cycle. This CNF code provides feedback to the master about errors and about the slave's status. Later CNF codes provide information about data cycles of the transaction. The type of feedback depends upon the cycle and the type of transaction. Error feedback occurs two cycles after the cycle being reported. (See Table 4-3 for more information on CNF codes.)

The master sends write data in the third and succeeding data cycles. If a slave cannot receive data at the specified time, it can send a STALL response until it is ready to receive the data. The slave may stall for at most 127 consecutive cycles (see Section 4.2.4.1, STALL response). During data cycles the BI I<3:0> L lines are an UNDEFINED field in WRITE and WCI transactions, whereas in WMCI and UWMCI transactions these lines carry the write mask. During all data cycles the master generates parity, and the slave checks parity.

---

*The following abbreviations are used in the figures in this chapter that show the format of VAXBI transactions:

M	Master node
S	Slave
Ss	Slaves
AAN	All arbitrating nodes
AN	All nodes
APS	All potential slaves (only for IDENT, prior to IDENT ARB selection)

All the CNF codes are listed as they might occur in the specified cycles. A > in front of a CNF code indicates the CNF code that would be transmitted during the transaction illustrated.

digital™

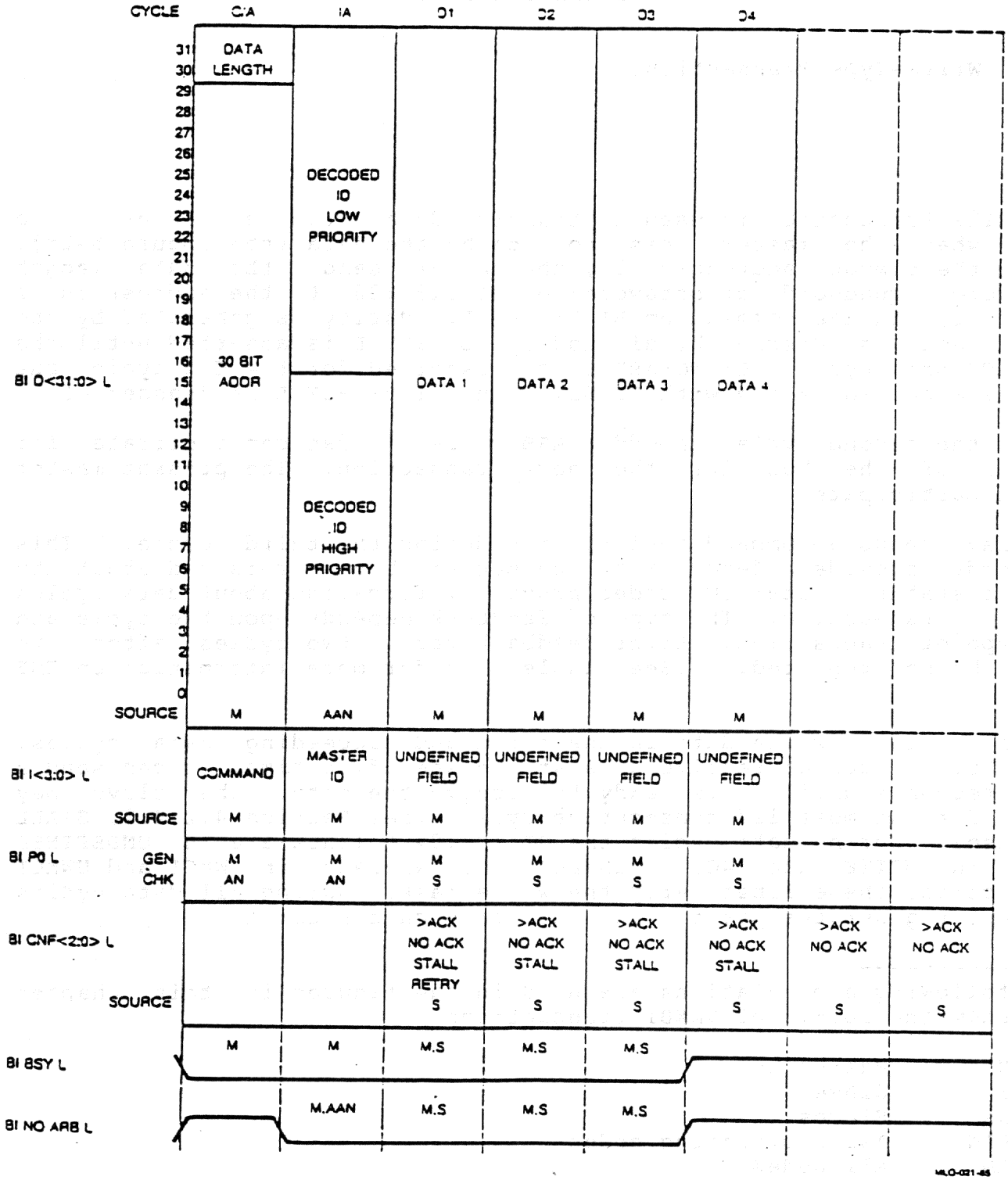


Figure 5-2: WRITE and WCI Transactions (octaword length shown)

digital™

WCI (Write with Cache Intent)

The WCI transaction performs a write but the data transferred may be cached. A node while cacheing must issue a WCI rather than a WRITE to alert the slave that the data is being cached. Note that cacheing can occur only if, just before the write is performed, the cache already contains valid data for the location written to. Should the same location in the slave node be written to later without the use of a VAXBI transaction, the slave must issue an INVAL on the bus. If a cacheing node cannot determine if data is being cached,* then the cacheing node must assume that the data is being cached and issue a WCI rather than a WRITE. For example, in the case of a bus adapter, if a processor on the target bus originated the write, the bus adapter may not be able to determine if the processor cached the data.

The slave's response to a WCI command is the same as that to a WRITE command (see Figure 5-2).

During data cycles the BI I<3:0> L lines are an UNDEFINED field in WRITE and WCI transactions, whereas in WMCI and UWMCI transactions these lines carry the write mask.

*In the sense used here, a cacheing node is any VAXBI node that behaves like a cacheing processor (when looking into the node from the VAXBI bus). For example, a DB88 (VAX 8800 system to VAXBI bus adapter) behaves like a cacheing processor even though it really connects the VAXBI bus to a processor-memory interconnect and not directly to a processor's cache.

WMCI (Write Mask with Cache Intent)

This command is the same as the WCI command except that the master selects which bytes of the addressed location(s) it wants to modify. The write mask is transmitted on the BI I<3:0> L lines during each data cycle. (Note that these lines are UNDEFINED during data cycles in WRITE and WCI transactions.) However, if a master sets all bits to write all bytes, instead of using the WCI command, performance may be degraded (because WMCI execution may require a local read-modify-write operation). The master generates parity for the entire VAXBI data path regardless of which bytes are tagged for modification.

The format for the WMCI transaction is shown in Figure 5-3.

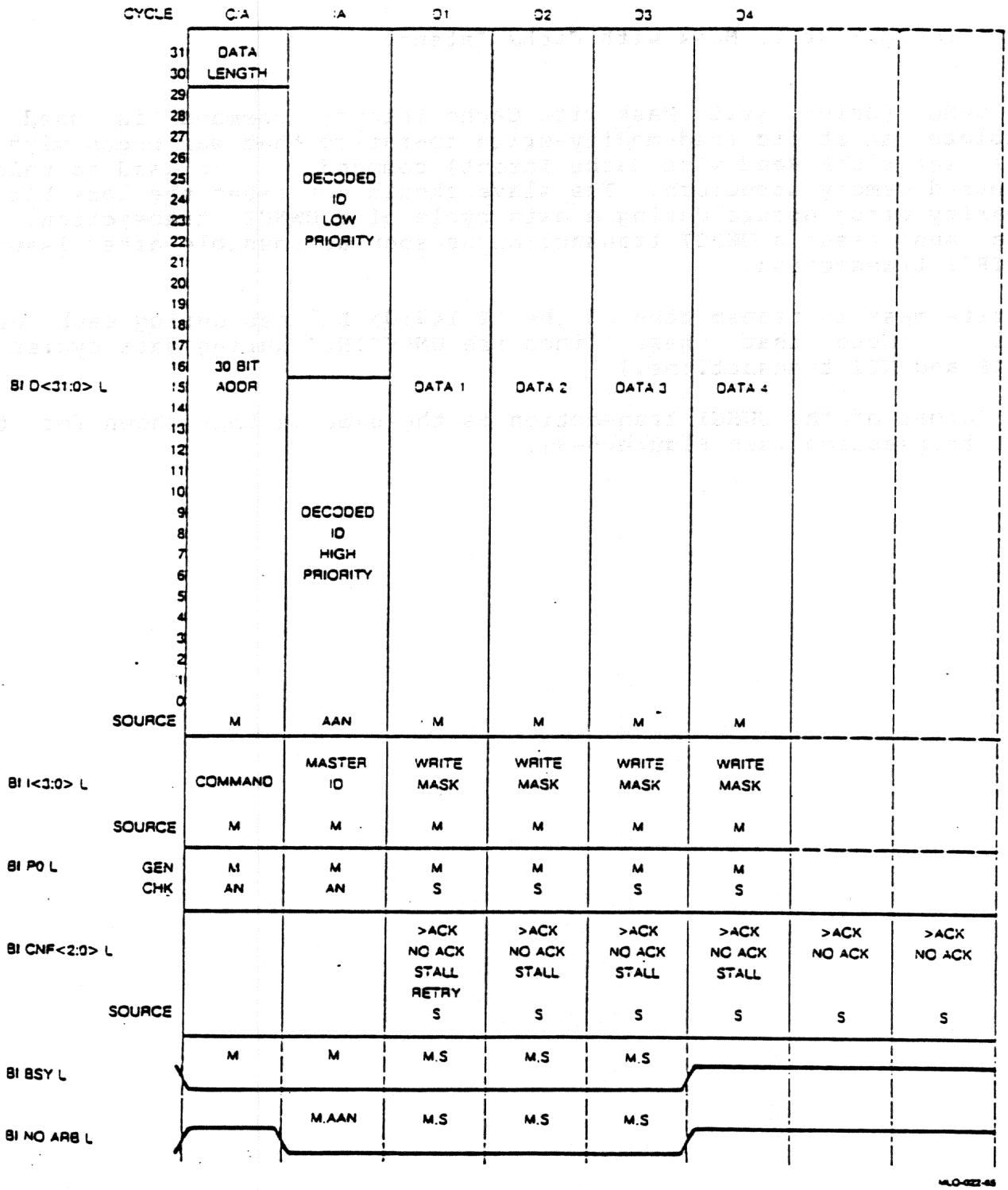


Figure 5-3: WMCI and UWMCI Transactions (octaword length shown)

digital™

UWMCI (Unlock Write Mask with Cache Intent)

The UWMCI (Unlock Write Mask with Cache Intent) command is used to complete an atomic read-modify-write operation that was begun with an IRCI (Interlock Read with Cache Intent) command. It is used to unlock a shared memory structure. The slave should not reset the lock bit if a parity error occurs during a data cycle of a UWMCI transaction. A node must issue a UWMCI transaction as soon as possible after issuing an IRCI transaction.

A write mask is transmitted on the BI I<3:0> L lines during each data cycle. (Note that these lines are UNDEFINED during data cycles in WRITE and WCI transactions.)

The format of the UWMCI transaction is the same as that shown for the WMCI transaction (see Figure 5-3).

5.3.6 Read-Type Transactions

READ

The READ transaction is used to transfer data from a slave to a master when the data will not be cached (see Figure 5-4).

During the command/address cycle, the master sends the data length (longword, quadword, or octaword) on BI D<31:30> L, the address on BI D<29:0> L, and the command on BI I<3:0> L. Parity is generated by the master and is checked by all nodes. BI BSY L is asserted until the last ACK data cycle. BI NO ARB L is deasserted for the C/A cycle but then is asserted along with BI BSY L, until BI BSY L is deasserted.

During the second cycle (imbedded ARB cycle), nodes can arbitrate for control of the bus for the next transaction. The present master cannot participate.

The slave sends a command confirmation during the third cycle. This CNF code provides feedback to the master about errors and about the slave's status. Later CNF codes provide response about data cycles of the transaction. The type of feedback depends upon the cycle and the type of transaction. Error feedback occurs two cycles after the cycle being reported. Read-type transactions differ from write-type transactions in that the CNF code providing feedback for the last two data cycles is sent by the master to the slave. (See Table 4-3 for more information on CNF codes.)

The slave sends read data in the third and succeeding data cycles. If a slave cannot send data at the specified time, it can send a STALL response until it is ready to send the data. The slave may stall for at most 127 consecutive cycles (see Section 4.2.4.1, STALL response). During all data cycles the slave generates parity, and the master checks parity.

During data cycles the slave sends read status on the BI I<3:0> L lines. The read status tells the master the status of the data being sent.

digital™

VAXBI TRANSACTIONS

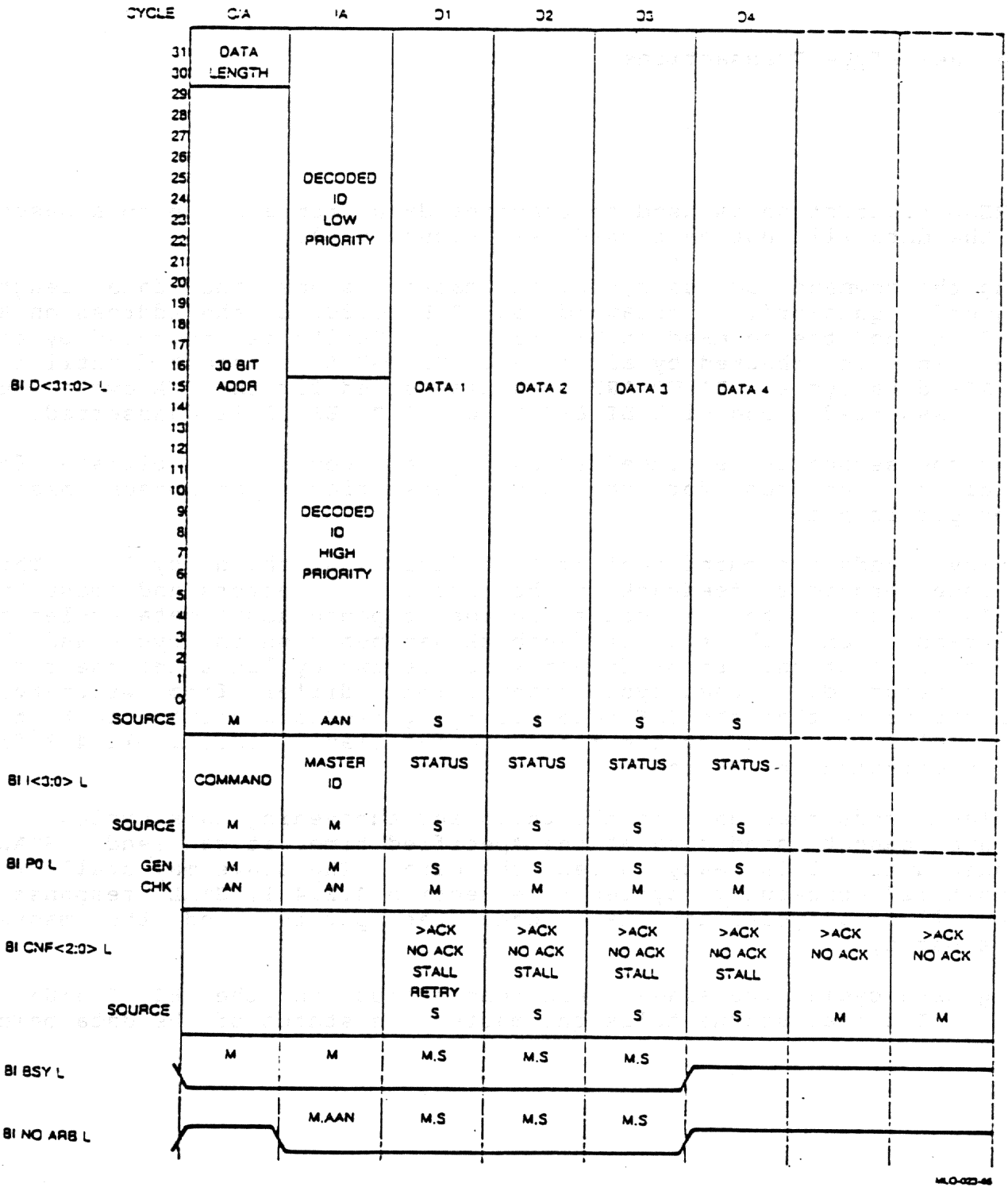


Figure 5-4: READ, RCI, and IRCI Transactions (octaword length shown)

digital™

RCI (Read with Cache Intent)

The RCI transaction performs a read but the data transferred is intended to be cached. However, if the slave returns a "don't cache" status code, the master must not cache the data* (see Section 5.3.4). The slave's response to an RCI command is the same as that to a READ command (see Figure 5-4). The command is used in cached multiprocessor systems. The RCI command is generated on a cache miss by processors to signal the slave that the requested read data will be placed in the master's cache. This permits an efficient mapping mechanism or "cached" bit for efficient generation of INVAL commands. (See Application Note 2, Section 2.1.1.)

---

*The KA820 processor has an exception to this rule.

digital™

IRCI (Interlock Read with Cache Intent)

The IRCI command supports atomic read-modify-write operations and is used with the UWMCI command. The format of the IRCI transaction is the same as that shown for the READ transaction (see Figure 5-4). A node that has been successfully accessed in memory space by the IRCI command must set a "lock bit," which while set, causes subsequent IRCI transactions directed to the same locked address range to be retried. This lock bit must remain set until a successful UWMCI transaction accesses the same locked address range (see Section 5.2.2).

In node window space the minimum size of the address range covered by a single lock bit is a byte. Outside of node window space, the minimum size of the address range covered by a single lock bit is an octaword.

If the master returns a NO ACK during either of the two cycles following the data cycles of an IRCI transaction (for example, as the result of detecting a parity error), the slave must not set the lock bit.

If the Read Data Substitute status code is sent by the slave, the IRCI transaction is considered unsuccessful. The slave, therefore, should not set a lock bit, and the master should not perform the corresponding UWMCI to complete the atomic operation.

If a subsequent IRCI or UWMCI command is received before it is known that no errors have occurred for the prior IRCI transaction, the slave should issue a STALL or RETRY command confirmation until the state of the lock is determined.

A multiport memory will issue a RETRY response to IRCI commands if its lock bit has been set from any port.

An IRCI directed to a UNIBUS adapter from the VAXBI bus will be interpreted as a DATAIP to the UNIBUS. A UNIBUS DATAIP must be translated as an IRCI to the VAXBI.

5.3.7 Invalidate Transaction

INVAL (Invalidate)

The INVAL command is used to signal other nodes that they may have cached data that is no longer valid. This command would be used by processors and other intelligent nodes when they perform a write to a local memory. Since a local write is not visible on the VAXBI bus,

other VAXBI nodes that monitor the VAXBI would not see this transaction. The other nodes then must be notified by the node performing the write. That node issues a VAXBI INVAL transaction. (See Application Note 2, Section 2.1.) The format of the INVAL transaction is shown in Figure 5-5.

During the command/address cycle, the master sends the data length (longword, quadword, or octaword) on BI D<31:30> L, the address on BI D<29:0> L, and the command on BI I<3:0> L. The data length code indicates the number of consecutive addresses to be invalidated. The low-order address bits are RESERVED fields during INVAL commands. Parity is generated by the master and checked by all nodes. BI BSY L is asserted for the first and second cycles. BI NO ARB L is deasserted for the C/A cycle, asserted for the imbedded ARB cycle, and then deasserted.

During the second cycle (imbedded ARB cycle), nodes can arbitrate for control of the bus for the next transaction. The present master cannot participate.

The slave or slaves send a command confirmation during the third cycle. The only valid responses to an INVAL command are ACK and NO ACK.

During the third cycle, the BI D, I, and P lines are RESERVED fields.

To have time to invalidate its cache, a node can delay the start of the next bus transaction. It does so by continuing to assert BI BSY L through the third cycle and until its cache is invalidated.

Table 5-7 describes the rules for address interpretation for the INVAL transaction.

Table 5-7: Address Interpretation for INVAL Transaction

Data Length	Transmitted Address	Received Address
Octaword	A<29:4>'0000	A<29:4>'----
Quadword	A<29:3>'000	A<29:3>'---
Longword	A<29:2>'00	A<29:2>'--

digital™

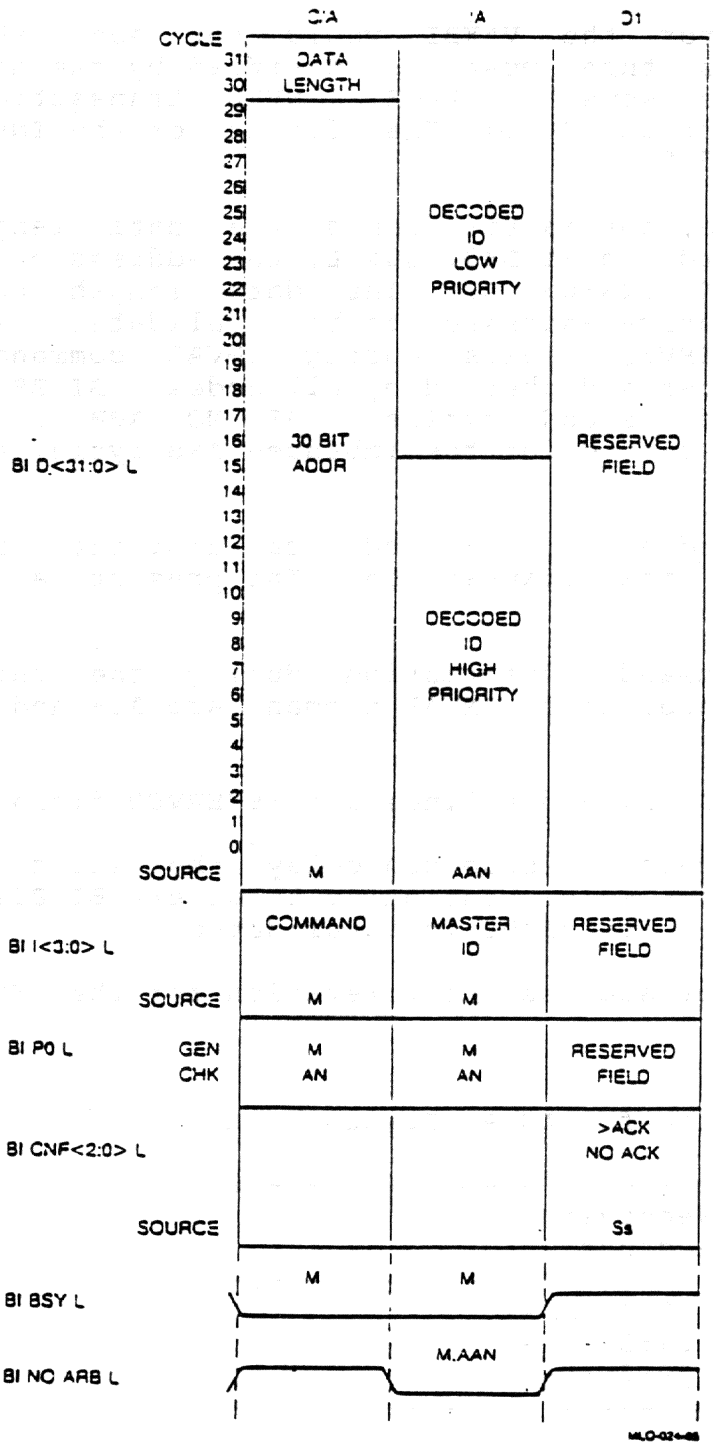


Figure 5-5: INVAL Transaction

5.4 TRANSACTIONS TO SUPPORT INTERRUPTS

The VAXBI bus provides for device interrupts (in the form of INTR and IDENT transactions) and interprocessor interrupts (in the form of IPINTR transactions). With device interrupts, the interrupting device supplies an interrupt vector in response to an IDENT transaction. The vector is specific to the device's node ID. With interprocessor interrupts, the interrupt vector and interrupt level are stored in the receiving node and are the same for all interprocessor interrupts.

5.4.1 Device Interrupts

Each node capable of generating interrupts contains a vector that is used by VAX processors to index into one or more 512-byte pages of memory containing address pointers to interrupt service routines.

During the command/address cycle of the INTR transaction, the D lines <19:16> each correspond to an interrupt level. D<19> corresponds to VAXBI interrupt level 7, the highest priority interrupt level, while D<16> corresponds to VAXBI interrupt level 4, the lowest priority interrupt level. One or more of these lines may be asserted during the command/address cycle to indicate the levels at which the interrupt is issued. VAXBI interrupt levels 7 through 4 correspond to interrupt priority levels (IPLs) 17 through 14 on VAX processors.

When an interrupted node is ready to service the interrupt, it issues an IDENT transaction. In the IDENT Level field, the node specifies the interrupt level it is ready to service. The node must specify only one IDENT level: that is, only one bit in the IDENT Level field should be set. This IDENT level must be the highest priority interrupt level for which the bus master has received an INTR and has not yet responded with a successful IDENT.

All nodes that have an interrupt pending at the IDENT level respond by arbitrating during the IDENT arbitration cycle in the IDENT transaction. The winner of this arbitration returns its interrupt vector during the next cycle. Note that this "VAXBI interrupt vector" is different from the VAX "interrupt vector" as described in the VAX-11 Architecture Reference Manual. In a VAX system, the VAXBI interrupt vector is used as an offset into the system control block (SCB), and the location thus obtained contains the VAX interrupt vector. In this document, "interrupt vector" means "VAXBI interrupt vector."

The interrupt vector 0 and interrupt vectors that are a multiple of 200 (hex) are reserved to indicate a "null interrupt"; that is, no action is needed to service the interrupting device.

digital™

If a node has more than one bit set in its destination mask when it issues an INTR transaction, two or more processors (at different times) will attempt to service this interrupt. Each interrupted processor will issue an IDENT transaction. If this is the only node issuing an INTR, the first processor to respond with an IDENT services the interrupt. The remaining processors issue IDENT transactions, but there will be no contenders during the interrupt arbitration cycle, and no interrupt vector will be returned. This is a case of a "null interrupt," where a servicing processor finds no node waiting to be serviced. The processor must then either cancel the interrupt (so that, as far as the software is concerned, the interrupt never happened) or take the interrupt with a vector of zero. The latter alternative is preferred, as it allows the software to log the occurrence.*

Consider a system where the interrupting node may send its INTR transaction to more than one processor. The interrupting node issues an interrupt at level L. Processor A services this interrupt. Now suppose the interrupting node issues another interrupt at level L. In systems where all interrupts are directed to one processor, an IDENT may not be issued for the second interrupt until the first interrupt has been serviced and the processor's interrupt level has dropped below L. In systems with multiple processors, however, processor B may attempt to service the second interrupt before processor A has completed servicing the first interrupt and dropped its interrupt priority level. If it is important to service these interrupts in sequence, some arrangement has to be made in software, perhaps through the use of semaphores.

---

*The KA88, KA820, and KA800 processors all use this alternative.

digital™

INTR (Interrupt)

The INTR command is used to signal interrupts to one or more nodes on the bus (see Figure 5-6).

During the command/address cycle, the master sends the interrupt level on BI D<19:16> L, the INTR destination mask on BI D<15:0> L, and the command on BI I<3:0> L; BI D<31:20> L is a RESERVED field. Parity is generated by the master and is checked by all nodes. BI BSY L is asserted for the first and second cycles. BI NO ARB L is deasserted for the C/A cycle, asserted for the imbedded ARB cycle, and then deasserted.

During the second cycle (imbedded ARB cycle), nodes can arbitrate for control of the bus for the next transaction. The present master cannot participate.

During the third cycle all slaves respond with an ACK confirmation. During this cycle the BI D, I, and P lines are RESERVED fields.

By transmitting an IDENT command on the bus, an interrupt fielding node solicits a vector from the node that issued the INTR command. Multiple nodes may be targeted to receive the IDENT command, but only one of them is permitted to transmit a vector. Nodes that lose an IDENT arbitration must retransmit an INTR command at the IDENT level.

Note that it is possible to interrupt at more than one priority level during any given INTR command. It is also permissible for an INTR command to contain zeros in the INTR Level field of the command/address cycle. In this case the slave must still respond with an ACK confirmation.

Nodes determine whether they are selected by the INTR command by performing an AND operation for each bit of their decoded ID and the destination code transmitted on BI D <15:0> L during the command/address cycle. If any bit is a one (that is, the interrupt fielding node's decoded ID matches a bit in the destination field), the node is permitted to respond to this interrupt.

Nodes responding to INTR commands must retain sufficient state information to permit them to generate subsequent IDENT commands to solicit vectors for the INTR commands received. The state required is an interrupt pending bit at each of the four INTR levels.

Nodes may inhibit interrupts from other nodes in several ways. They may:

- Respond with NO ACK to INTR commands directed at them.
- Manipulate the INTR Destination Register of the interrupting node (see Section 7.5).
- Manipulate the control register(s) for the node in question.

Nodes may defer interrupt service simply by delaying the IDENT transaction that provides the vector.

Interrupt Priority

A node's interrupt priority is broken down into two groupings -- level and sublevel.

Level is the higher order priority structure and consists of four priorities, 7 through 4. These priorities are used to determine the level at which a processor is interrupted.

For each level, 16 interrupt sublevels can be indicated to sort out which interrupting node will respond with an interrupt vector when polled. During the IDENT ARB cycle of an IDENT command, all potential slaves drive their sublevel on BI D<31:16> L; the node with the lowest number bit asserted is designated to return the vector. Sublevel priority is determined by the node's ID and remains fixed. A node's ID, therefore, affects how quickly that node is serviced.

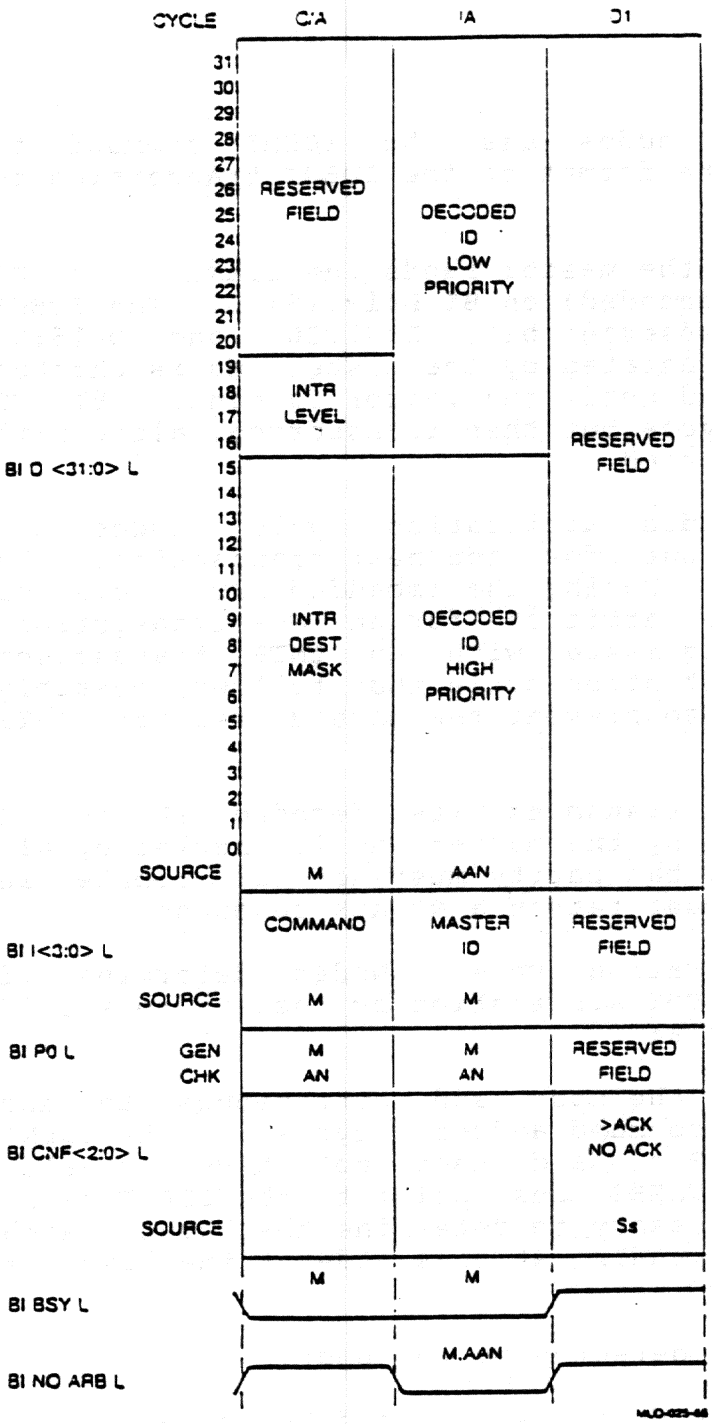


Figure 5-6: INTR Transaction

IDENT (Identify)

In response to the INTR command, nodes use the IDENT command to solicit the interrupt vector. The format of the IDENT transaction is shown in Figure 5-7.

During the command/address cycle, the master sends the command on BI I<3:0> L and the IDENT level (decoded) on BI D<19:16> L. The IDENT Level field can contain only one asserted bit. D<31:20> and D<15:0> are RESERVED fields. Parity is generated by the master and is checked by all nodes. BI BSY L is asserted until the vector is sent. BI NO ARB L is deasserted for the C/A cycle but then is asserted, along with BI BSY L, until BI BSY L is deasserted.

During the second cycle, the imbedded arbitration cycle, nodes can arbitrate for control of the bus for the next transaction. The present master cannot participate. During the imbedded ARB cycle of an IDENT transaction, nodes cannot arbitrate for an INTR transaction. This requirement prevents a pending master with an INTR transaction from taking part in the IDENT arbitration cycle that follows, possibly winning, and necessarily aborting to prevent the already serviced INTR from being reposted.

During the third cycle, the master transmits its decoded ID on BI D<31:16> L. Parity is generated by the master and is checked by all potential slaves. Nodes detecting bad parity must not participate in the IDENT arbitration cycle, and must return a NO ACK response.

The fourth cycle is an IDENT arbitration cycle. Nodes determine if they must participate in this IDENT arbitration by testing to see if they meet all the following criteria:

- An interrupt is pending at the node and corresponds to the level sent during the command/address cycle of the IDENT command. Note that an INTR command need not have actually been sent out on the VAXBI bus prior to the IDENT being received. All that is necessary to determine the level match is that an interrupt be pending at this node at the level of the IDENT command received.
- No Command Parity Error is detected by the node.
- No Master Decoded ID Parity Error is detected by the node.
- A bit match exists between the master's decoded ID and the INTR destination mask.

digital™

If all four criteria are satisfied, the slaves arbitrate by asserting the bit corresponding to their interrupt sublevel priority (that is, their node ID) on BI D<31:16> L. Parity is not checked for the IDENT arbitration cycle. The BI D<15:0> L, BI I<3:0> L, and BI P0 L lines are RESERVED fields during this cycle.

The slave with the highest sublevel priority (lowest ID) wins the IDENT ARB cycle and responds in the next cycle with an ACK, RETRY, or STALL. Note that in Figure 5-7 the slave sends a STALL. For this cycle the BI D, I, and P lines are UNDEFINED fields. Along with the vector (on BI D<13:2> L), the slave sends status on BI I<3:0> L (see Table 5-6 for the read status codes) and a CNF code. If the transfer is unsuccessful because of a parity error, the master sends a NO ACK response two cycles after the attempted transfer by the slave. The master resends the IDENT command at the same level to reattempt the vector transfer. A buffer for the vector may be necessary in some adapters that are unable to resolicit the same vector from a device on the bus they adapt to. Upon receiving the ACK response, with no parity error, the master resets the interrupt pending bit at the IDENT level. BI D<31:14> L and BI D<1:0> L must be zeros at the time the vector is sent. Parity is generated by the slave and is checked by the master for the vector.

During the two cycles after the vector has been transmitted, the master sends ACK confirmations if no parity error has occurred. The responding slave must wait until the final ACK confirmation before assuming that the vector has been correctly received. If a NO ACK or illegal confirmation is received, the slave must retransmit the INTR command and be prepared to retransmit the vector when another IDENT is issued.

Nodes that met the criteria for IDENT selection, but which lost the IDENT arbitration, must reissue the INTR transaction at the IDENTed level. This reissue of INTR ensures that the interrupt fielding nodes do not lose previously posted interrupts at the IDENTed level. It is possible that additional level or destination information will be present when the INTR command is repeated.

The reissuing of interrupts is not expected to cause significant performance degradation in the system. It is assumed that in the typical interrupt service model only a few interrupts are pending at any time.

If the interrupting condition no longer exists or the interrupt has been serviced by another node, nodes return the NO ACK response.

digital™

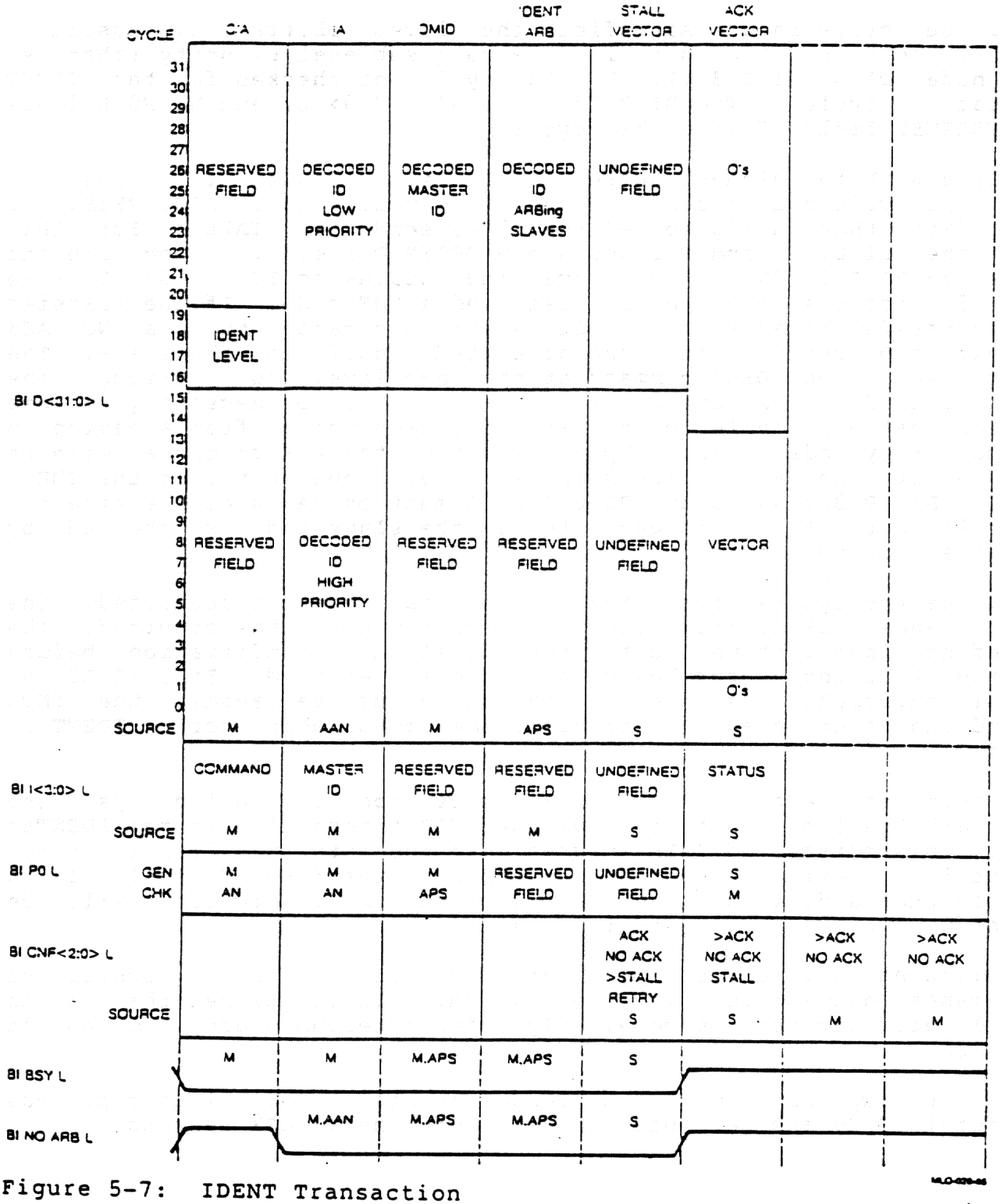
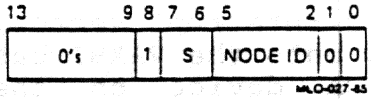


Figure 5-7: IDENT Transaction

5.4.2 VAXBI Interrupt Vectors

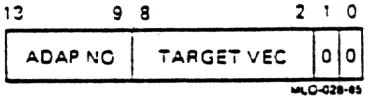
Interrupt vectors issued by VAXBI adapters are of two types. Each adapter is allotted 4 interrupt vectors of the first type and 128 interrupt vectors of the second type.

The first type of vector lies between 100 and 1FF (hexadecimal) and has the form:



Node ID is the node ID of the interrupting node (0 to 15), and S is the interrupt vector number. With the four possible values for S, each node may use up to four different interrupt vectors. It is expected that many types of interrupting nodes will record the interrupting condition in a register in nodespace and then interrupt using one of the four vectors. The interrupt handler will examine the nodespace register to discover the interrupting condition. If two or three of the interrupting conditions require an especially fast response, a separate vector (out of the four possible vectors) may be assigned to each of them, so that the interrupt handling routine need not read the nodespace register to discover the interrupting condition.

The second type of interrupt vector has the following form:



The bit pattern in the target vector field specifies one of up to 128 interrupt service routines. The adapter number, a small, nonzero integer assigned by the operating system software, is stored in a register in the adapter, to be used in constructing the interrupt vector. In a given system, each adapter that issues this second type of interrupt vector has a unique adapter number, and the range of adapter numbers determines the range of possible interrupt vectors in a given system. A target vector of all zeros must indicate a null interrupt, as described in Section 5.4.1.

digital™

On a VAX processor, the first type of interrupt vector, regardless of which VAXBI node issued the vector, uses the first page of the system control block. On the other hand, each VAXBI node that uses the second type of interrupt vector requires an additional page of the system control block. The largest adapter number determines the number of pages in the system control block. Since the system control block must reside in main memory, there is some motivation to keep it small. The VAX-11 Architecture Reference Manual explains the system control block.

All adapters are encouraged to use the first type of interrupt vector. Bus adapters that map interrupts from a target bus onto the VAXBI bus typically use the second type of interrupt vector. A device on the target bus may interrupt a processor on the VAXBI bus and provide an interrupt vector on the target bus when the processor responds with an IDENT transaction. The interrupt vector provided on the target bus is then used for the target vector field of the VAXBI interrupt vector. The UNIBUS adapter, for example, uses the second type of interrupt.

digital™

5.4.3 Interprocessor Interrupts

IPINTR (Interprocessor Interrupt)

The IPINTR command is used by processors to interrupt other processors. The operation of the command is similar to that of the INTR command except that the level and the vector are stored in the receiving node rather than sent. The format of the IPINTR transaction is shown in Figure 5-8.

During the command/address cycle, the master sends its decoded ID on BI D<31:16> L (rather than level information), the command on BI I<3:0> L, and the IPINTR destination mask on BI D<15:0> L. Parity is generated by the master and is checked by all nodes. BI BSY L is asserted for the first and second cycles. BI NO ARB L is deasserted for the C/A cycle, asserted for the imbedded ARB cycle, and then deasserted.

During the second cycle (imbedded ARB cycle), nodes can arbitrate for control of the bus for the next transaction. The present master cannot participate.

Receiving nodes determine if they have been selected by checking their IPINTR Mask Register. Nodes compare the decoded ID received with the corresponding bit position in the IPINTR Mask Register. All slaves respond with an ACK on BI CNF<2:0> L in the third cycle. Since there can be multiple responders, only ACK and NO ACK are valid responses.

During the third cycle, the D, I, and P lines are RESERVED fields. Parity is neither generated nor checked.

With VAX processors, when an interprocessor interrupt arrives at a processor node, the processor receives a VAX interrupt level 14 (hex) interrupt, with an interrupt vector at SCB offset 80 (hex).

To identify the processor that sent the interrupt, the interrupted processor examines the IPINTR Source Register. A set bit in this register indicates that an interprocessor interrupt has been received from a processor with the corresponding node ID. The bits are write-1-to-clear, and they should be cleared after being read.

By the use of the IPINTR Mask Register, a VAXBI node can specify which nodes are allowed to send it interprocessor interrupts.

It is expected that the normal mode of operation is as follows: (a) the interrupting processor deposits an item in an agreed-upon queue in memory, containing the information to be passed to the interrupted processor; then, (b) upon receiving the interprocessor interrupt, the interrupted processor looks in this queue for the information. With this mode of operation, the interrupted processor then does not need to access the Force-Bit IPINTR/STOP Destination and Mask Registers.

digital™

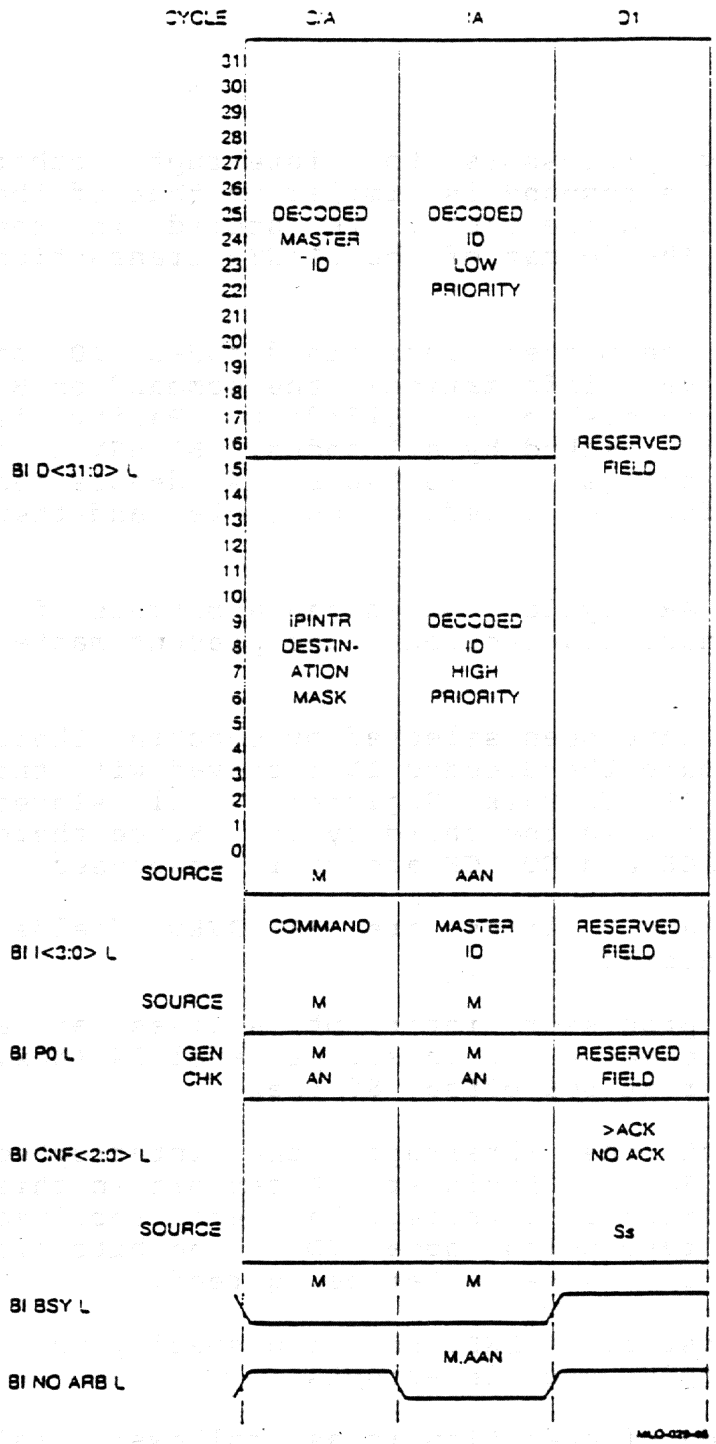


Figure 5-8: IPINTR Transaction

digital™

5.5 TRANSACTION TO SUPPORT DIAGNOSTICS

STOP

The STOP command is used to selectively force nodes to a state (the stopped state) in which they will not issue VAXBI transactions but will retain as much error information as possible. In this state, nodes can be accessed and diagnosed for error information. Whether a node can be restarted by software after receiving STOP without going through a complete power-down/power-up or reset sequence is implementation dependent.

The format of a STOP transaction is the same as that for an INTR transaction, except that no level information is sent (see Figure 5-9). During the command/address cycle, the master sends a destination mask on BI D<15:0> L and the command on BI I<3:0> L. The BI D<31:16> L lines are a RESERVED field. Parity is generated by the master and is checked by all nodes. BI BSY L is asserted for the first and second cycles. BI NO ARB L is deasserted for the C/A cycle but then is asserted, along with BI BSY L, until BI BSY L is deasserted.

During the second cycle (imbedded ARB cycle), nodes can arbitrate for control of the bus for the next transaction. The present master cannot participate.

During the third cycle the slave sends a command confirmation. The only valid responses are ACK and NO ACK. A node must do one of the following while proceeding to the stopped state:

- Respond with RETRY confirmations for subsequent single-responder commands and NO ACKs for subsequent multi-responder commands that it receives.
- Extend the STOP transaction by holding BI BSY L asserted.

The node should not abort any brief operations (such as a memory refresh) that might leave the node in an undefined state.

A node must retain as much state as possible to facilitate error analysis. Node documentation must specify the node registers whose contents could be changed in response to a STOP command.

digital™

All nodes selected by a STOP command are required to do the following:

- Cease issuing transactions as soon as feasible.
- Clear any Sent and Force bits set in the User Interface and Error Interrupt Control Registers to clear any posted interrupts.
- Set the INIT bit in the VAXBI Control and Status Register.

Since the STOP command may be used for maintenance purposes, it must have a low-level implementation so that the node reaches the stopped state as soon and as reliably as possible.

digital™

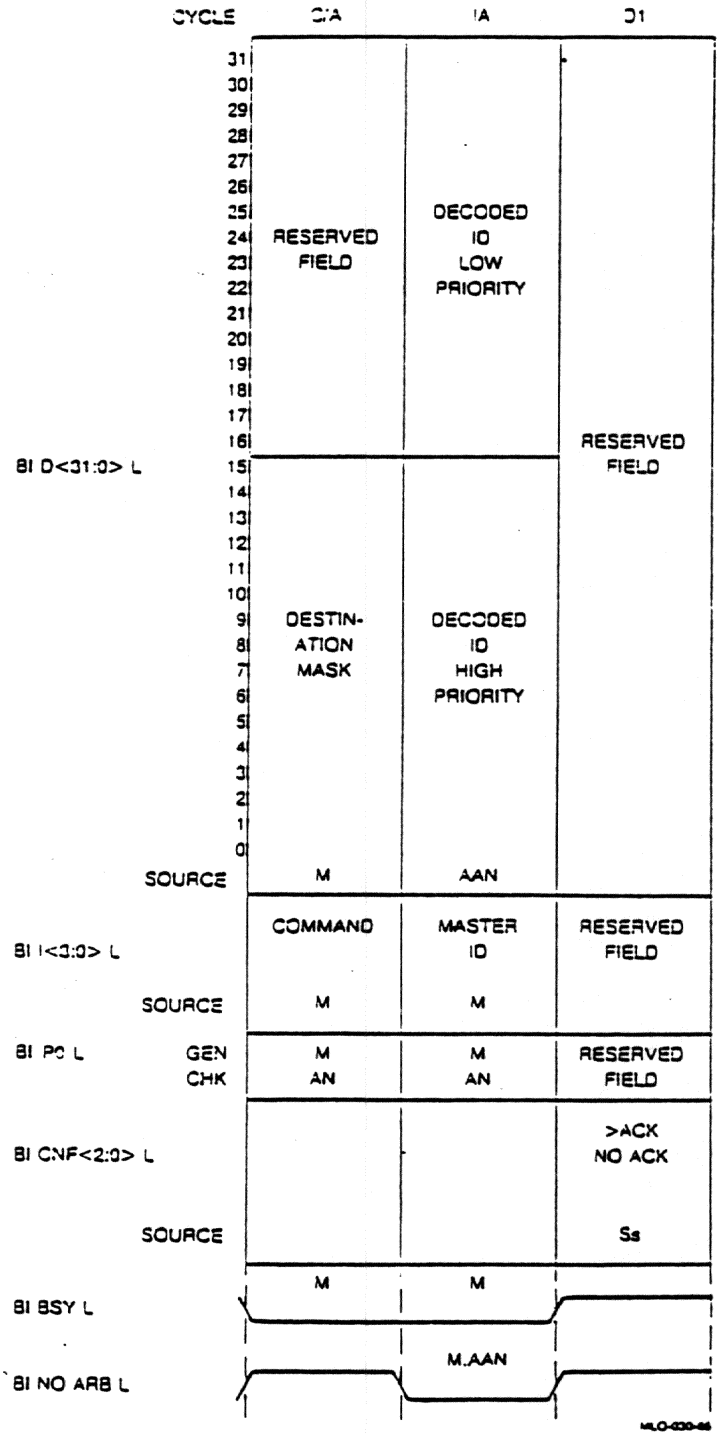


Figure 5-9: STOP Transaction

digital™

CHAPTER 6

INITIALIZATION

The VAXBI bus provides three mechanisms for initialization:

- Power-Down/Power-Up -- On power-up, the BI AC LO L and BI DC LO L lines are sequenced to provide initialization of the system.

- System Reset -- A power-down/power-up sequence can be emulated through the use of the BI RESET L line, which causes the sequencing of BI AC LO L and BI DC LO L in the same way that would occur for a "real" power-down/power-up sequence. In this way the system can be returned ("reset") to its power-up state without actually cycling the power supplies. Note that throughout the system reset sequence the power supply voltages remain in tolerance, whereas in a "real" power-down/power-up sequence the power supplies generally go out of tolerance. (DC power supply output voltages may not drop out of tolerance during brownout conditions.)

- Node Reset -- A single node in a system can be reset without resetting the entire system. This mechanism involves the use of the Node Reset bit in the VAXBI Control and Status Register. Node reset of BIIC-based nodes is discussed in Application Note 7.

Section 6.1 gives the general requirements of a VAXBI node's initialization process. Section 6.2 describes the two system initialization mechanisms: the power-down/power-up sequence and the system reset sequence. The section on system reset also gives requirements for reset modules, which are used to cause a system reset, and discusses how "extended system reset" can be used to down-line load software. Section 6.2.3 discusses node reset. Sections 6.3 through 6.5 provide detailed information on the VAXBI signals that support the initialization mechanisms: BI AC LO L, BI DC LO L, and BI RESET L. Section 6.6 describes the timing for the power-down/power-up and system reset sequences.

6.1 VAXBI NODE INITIALIZATION REQUIREMENTS

Regardless of the method used to cause a node to initialize, the initialization process must consist of the following:

- All node logic must be reset by an asserted BI DC LO L (in BIIC-based nodes BCI DC LO L is used). Whether the initialization of a node causes the initialization of another bus attached to this node is implementation dependent.*
- On the assertion of BI DC LO L, the BI BAD L line must be asserted and the Broke bit must be set. These states must remain until the successful completion of node self-test.
- A complete node self-test must run following the deassertion of BI DC LO L. The specific requirements for this test are discussed in Section 11.1.
- Following a successful self-test, a node must reset its Broke bit (in the VAXBI Control and Status Register or Slave-Only Status Register) and release the BI BAD L line.
- At the conclusion of initialization, all nodespace locations must be in a defined state (as defined in the node specification). The Device Register must be loaded with the appropriate device type (See Section 11.1.7).

6.2 INITIALIZATION MECHANISMS

The VAXBI supports two mechanisms to initialize all nodes in a VAXBI system. These two methods are discussed in Sections 6.2.1 and 6.2.2. A third mechanism, discussed in Section 6.3, permits selective initialization of individual nodes.

---

*The DWBUA asserts UNIBUS INIT during its initialization process.

digital™

6.2.1 Power-Down/Power-Up

In a power outage, first AC power is lost, and then, if it is not recovered quickly, DC power falls below acceptable levels. These two events trigger the following:

- First BI AC LO L is asserted.
- Then BI DC LO L is asserted.

On power-up, these two lines are deasserted in the reverse order:

- First BI DC LO L is deasserted.
- Then BI AC LO L is deasserted.

Power is restored after a delay to ensure that the clock is stabilized and substrate bias voltages are established in integrated circuits. The deassertion of BI DC LO L indicates to VAXBI nodes that they should start their self-test and initialization process; however, VAXBI-accessible memory should not be accessed until BI AC LO L is deasserted. Finally, deassertion of BI AC LO L indicates that software execution can begin, and a warm restart or a cold start (as described in the VAX-11 Architecture Reference Manual) is attempted.

During a power outage, memory nodes with battery backup continue to be supplied with battery power, allowing memory contents to be retained. When power is restored, these memory nodes should not be reinitialized. However, if the power outage is lengthy, battery backup power may be exhausted and backup voltages may drop out of bounds. If this happens, the data in memory is no longer reliable.

What is required, therefore, is an indication, at the time that BI DC LO L is deasserted, whether backup power was maintained during the period that external power was lost. This indication is provided by a "reset module." (See Section 6.2.2.1 for requirements for the reset module.) The reset module monitors the battery backup power voltages. (The reset module also has another function, described below.) If these voltages drop out of bounds, the reset module asserts the BI RESET line before the deassertion of BI DC LO L. Memory nodes with battery backup monitor the BI RESET line at the deassertion of BI DC LO L and perform self-test and initialization only if the RESET line is asserted.

digital™

6.2.2 System Reset

The reset sequence emulates the power-down/power-up sequence of the BI AC LO L and BI DC LO L signals. The reset sequence causes all VAXBI nodes to initialize themselves.

A reset module is used to carry out a reset sequence. The reset module monitors the BI RESET L line and drives the BI AC LO L and BI DC LO L lines. Upon the detection of an asserted BI RESET L line, the reset module begins a reset sequence. Typically, only adapters that provide a remote reset capability and processors can assert the RESET line.

If the reset module finds that the RESET line is asserted while BI AC LO L and BI DC LO L are deasserted, it asserts BI AC LO L and then BI DC LO L; it then deasserts BI DC LO L. These three transitions are subject to the timing constraints given in Table 6-1. In response, all VAXBI nodes perform self-test and initialization. When the RESET line is deasserted, the reset module also deasserts BI AC LO L, completing the emulation of the power-down/power-up sequence. Note that if the RESET line remains asserted until after BI DC LO L is deasserted, then all memory nodes will undergo self-test and initialization, including memory with battery backup. If memory with battery backup should retain its data, the RESET line must be deasserted before BI DC LO L is deasserted.

6.2.2.1 Reset Module Requirements - A VAXBI system must have a reset module if the system is intended to support one of the following:

- VAXBI memory modules that are operated with battery backup power supplies
- VAXBI nodes that require the ability to reset the VAXBI system. Such nodes are referred to as "resetting nodes."

The following requirements apply to the operation of reset modules in VAXBI systems with battery backup:

- BI RESET L must be monitored by any VAXBI memory node with battery backup.
- The RM must monitor the DC outputs of all voltages needed for battery backup and assert the BI RESET L signal if any of these backup voltages goes out of bounds.

  It is not required that the RM physically sense each voltage. Since the purpose is to guarantee power outage sequences with correctly working hardware, an RM can be designed to take advantage of power sequencing or relative power loading to reduce the design and product cost.

digital™

- VAXBI memory modules with battery backup should sample BI RESET L at the deassertion of BI DC LO L. These memory modules should test and initialize their RAMs if BI RESET L is asserted at that time. If BI RESET L is not then asserted, these memory modules must not modify the contents of RAM.

The following requirements apply to the operation of reset modules in VAXBI systems with node reset support:

- BI RESET L must be monitored by the RM.

- Nodes that assert BI RESET L must maintain that assertion until BI DC LO L is asserted.

- If BI RESET L is asserted when BI AC LO L is not asserted, the RM will generate a full sequence of BI AC LO L and BI DC LO L which simulates a power-down/power-up sequence.

- VAXBI nodes that are not asserting BI RESET L may not access VAXBI memory space addresses between the deassertion of BI DC LO L and the deassertion of BI AC LO L. This prevents processors from reading memory, for example, while a resetting node may be modifying memory.

6.2.2.2 Use of System Reset for Down-line Loading Software - A variation of the system reset sequence (called "extended system reset") allows software to be down-line loaded. This is possible because the reset module does not deassert BI AC LO L until the RESET line is deasserted. (As explained later in this section, however, the desired effect may or may not be achieved depending on the system that is being down-line loaded.)

The node that asserts the RESET line is the "resetting node." When the resetting node asserts the RESET line, the reset module asserts BI AC LO L and BI DC LO L and then deasserts BI DC LO L. To attempt to down-line load software, the resetting node holds RESET asserted until after BI DC LO L is deasserted. When BI DC LO L is deasserted, all VAXBI nodes (except perhaps the resetting node) perform self-test and initialization. When these operations are completed, the resetting node can attempt to write the software into memory space, including a restart parameter block.

The reset sequence also causes the BIICs to perform their self-test so that initialization is required, without regard to the state of the RESET line. Thus, Starting and Ending Address Registers are cleared to all zeros and must be set before writing memory. Various control registers are also set to their initial states.

digital™

After the software has been written, the resetting node deasserts the RESET line. What happens at this point is implementation dependent. With some systems the processor will attempt a warm restart. The warm restart finds the restart parameter block that has just been loaded into memory and begins to execute the down-line-loaded software. With other systems, however, the processor will attempt to perform a cold start, in which case the down-line-loaded software will be overwritten. Therefore, although there is provision for down-line loading software, whether this is effective is implementation dependent.*

6.2.3 Node Reset

Node reset causes an individual node to be initialized. Writing a one to the Node Reset (NRST) bit in the VAXBI Control and Status Register of a particular node causes that node to initialize.** (In BIIC-based nodes, setting the NRST bit causes BCI DC LO L to be asserted.) As with the other types of initialization mechanisms, the node will be inaccessible for the duration of its initialization and BI BAD L will be asserted during this time.***

During VPI (VAXBI primary interface) self-test, any access of a targeted node's address space by any other node may fail (the targeted node's VPI will return a NOACK confirmation). This might result in a machine check at the accessing node and might cause the targeted node's self-test to fail.

A target node's address space must not be written to at any time during the target's node self-test (note distinction between node self-test and VPI self-test) by another node. This is because such a write access may disturb node self-test and thus cause the targeted node to fail self-test. Successful completion of node self-test is signaled by the clearing of the BROKE bit in the VAXBICSR or SOSR of the targeted node.

---

*When BI AC LO L is deasserted, both the KA62 and KA88 processors cold start. It is thought that the KA820/KA825 processors would warm start, but the extended system reset capability has never been exercised on these processors.

**Node reset can be performed only after arbitration is disabled on the target node. See the description of the VAXBICSR NRST bit.

***On the VAX 82x0/83x0 and on the VAX 62x0, the fault LED on the control panel will be lit.

digital™

Locations in a target node's address space must not be read at any time during node self-test by another node to avoid disturbing the targeted node's self-test. The only exceptions are:

- The Device Register (DTYPE<14:8> determines the location of the BROKE bit)
- The VAXBI Control and Status Register (contains the BROKE bit if DTYPE<14:8> not equal 0)
- The Slave Only Status Register (contains the BROKE bit if DTYPE<14:8> equals 0)

At the completion of and as a result of a VPI self-test caused by a node reset, the NPE bit of the Bus Error Register in the target VPI might have been set spuriously by the target VPI. It is therefore advisable for the operating system to clear the NPE bit on the target VPI after the node reset, but before causing another operation on the target node. However, NPE must not be cleared until the BROKE bit in the target node's VAXBICSR or SOSR has been cleared by the target node's user interface.

Software drivers that share a node* should agree in advance that a node needs to be reset. In this way lock states can be cleaned up, and the selection limitations can be supported.

To perform a node reset operation on another VAXBI node, a resetting node** must perform the following steps in sequence:***

---

*For example, a DWBUA.
**Here the term "resetting node" means the node writing to the NRST bit on the target node. Note that this is a different usage of "resetting node" than that in sections 6.2.2.2 and 6.3.
***This procedure will work for a uniprocessor or asymmetric multiprocessor system. Symmetric multiprocessor systems will require a different synchronization mechanism from setting processor IPL to ensure that the described sequence executes as a critical section.

digital™

1. Disable interrupts on the resetting node (if applicable).

2. Set the STS and ARB (VAXBICSR<11>, VAXBICSR<5:4>) bits on the target node to 111 (binary). All of these bits must be set by a single longword WRITE or WCI or WMCI (not a UWMCI*). VAX ISP machines should accomplish this with a MOVL or MOVW instruction.

3. Set STS, NRST, ARB (VAXBICSR<11:10>, VAXBICSR<5:4>) on the target node to 1111 (binary). All of these bits must be set by a single longword WRITE or WCI or WMCI (not a UWMCI*). VAX ISP machines should accomplish this with a MOVL or MOVW instruction.

4. Reenable interrupts on the resetting node (if applicable).

This procedure works because the disabling of arbitration prevents the target node from arbitrating for and beginning a transaction right after the NRST bit is set.

Interrupts are disabled on the resetting node to prevent the targeted node from interrupting the resetting node immediately after the ARB bits are set but before the NRST bit is set on the target. This disabling prevents the resetting node from reading a control/status register on the targeted node (for example, as part of an interrupt service routine) when the targeted node may have experienced a Bus Time Out. A resetting node that attempted such a read might experience a machine check.

6.3 BI AC LO L

The BI AC LO L signal is asserted when the line voltage is below minimum specifications. Following the assertion of BI AC LO L, nodes are guaranteed Tbips2(min) + Tbips8(min) of valid DC power (see Table 6-1).

When the BI AC LO L signal is asserted, processors and other intelligent nodes initiate a power-fail routine. Power-fail routines must be designed to complete execution in Tbips2(min).

During power-up a node must not access VAXBI-accessible memory space locations until the deassertion of BI AC LO L; however, memory nodes

---

*The resetting node must not use a UWMCI transaction (such as might result on a VAX ISP machine from a BISL or BISW instruction), since this could cause an Interlock Sequence Error to be declared on the hosts' VAXBI adapter when two host processors attempt to concurrently reset two nodes on the same VAXBI.

digital™

will clear memory locations following the deassertion of BI DC LO L if a cold start was indicated. During a system reset sequence it is permissible for the resetting node to access memory prior to the deassertion of BI AC LO L. As with a normal power-up, no other node may access memory prior to the deassertion of BI AC LO L.

With certain power supplies, during certain brownout power conditions, BI AC LO L may assert and later deassert without an assertion of BI DC LO L.*

BI AC LO L must remain asserted for Tbips3(min) after the deassertion of BI DC LO L to allow a node's internal initialization signals to be removed before a power restart interrupt is raised.

The BI AC LO L and BI DC LO L signals must remain asserted both when power has gone away and when DC power is in transition and not in tolerance.

The power status lines BI AC LO L and BI DC LO L may be driven asynchronously. To guarantee rejection of short, spurious deassertions, nodes must synchronize BI AC LO L and BI DC LO L deassertions to the bus clock and must not recognize deassertions that are less than one cycle in length. (See Appendix B for a description of the transmission line problem that mandates this requirement.) In addition, nodes must synchronize the assertion of BI AC LO L and must not recognize assertions less than one cycle in length. To reject assertion glitches, nodes must not recognize assertions of BI DC LO L of less than 50 nanoseconds.

6.4 BI DC LO L

The BI DC LO L signal warns of the impending loss of DC power and is used for initialization on power-up. Specifically, a node uses the BI DC LO L signal to force its circuitry into an initialized state. VAXBI node designs must not use other reset methods such as "RC time constant type" reset circuits (since VAXBI nodes must be resettable without regard to the state of the power supply outputs). Valid DC power and VAXBI clock signals will be provided prior to the deassertion BI DC LO L.

BI DC LO L must not be asserted until Tbips2(min) after the assertion of BI AC LO L to allow the power-fail routine to save processor state in memory and to halt. The result of any VAXBI transaction in progress when BI DC LO L is asserted is indeterminate.

---

*In a VAX 8200 system, BI DC LO L is always asserted after the assertion of BI AC LO L.

BI DC LO L must be asserted for Tbips8(min) before the loss of DC power so that nodes such as disk controllers can stop certain activities before power is removed.

Tbips9(min) of within-tolerance DC power must be provided prior to the deassertion of BI DC LO L. This allows time for power-up stabilization of components (such as the establishment of proper substrate biases on ICs). The circuitry generating BI TIME +/- and BI PHASE +/- must ensure that these clock signals are valid at least Tbips10(min) prior to the deassertion of BI DC LO L.

There can be no more than Tbips9(max) from valid DC power restoration to the deassertion of BI DC LO L. This helps guarantee a maximum power-fail restart time for all systems.

The BI DC LO L signal must be asserted for Tbips4(min). BI AC LO L will always be in the asserted state when BI DC LO L is asserted, but the opposite is not true. All bus signals except BI AC LO L, BI TIME +/-, and BI PHASE +/- remain deasserted during the assertion of BI DC LO L.

From the deassertion of BI DC LO L, the BIIC asserts BI NO ARB L for up to 5000 cycles while it performs BIIC self-test. Assertion of NO ARB suppresses bus traffic during BIIC self-test. The self-test must be implemented so that no bus signals (including BI NO ARB L) are asserted if the BIIC fails self-test.

The BI AC LO L and BI DC LO L signals must remain asserted both when power has gone away and when DC power is in transition and not in tolerance.

The power status lines BI AC LO L and BI DC LO L may be driven asynchronously. To guarantee rejection of short, spurious deassertions, nodes must synchronize BI AC LO L and BI DC LO L deassertions to the bus clock and must not recognize deassertions that are less than one cycle in length. (See Appendix B for a description of the transmission line problem that mandates this requirement.) In addition, nodes must synchronize the assertion of BI AC LO L and must not recognize assertions of less than one cycle in length. To reject assertion glitches, nodes must not recognize assertions of BI DC LO L of less than 50 nanoseconds.

6.5 BI RESET L

The assertion of BI RESET L initiates a system reset. The signal is received by a reset module (RM), a device that is used to generate a system reset in response to the assertion of BI RESET L.

Regardless of whether there is an RM in the VAXBI system, the proper power-down/power-up sequence of BI AC LO L and BI DC LO L must be generated as described in Figure 6-1. This functionality can be included within the RM design. In systems without an RM, the power supply (or supplies) is expected to supply the proper waveforms.

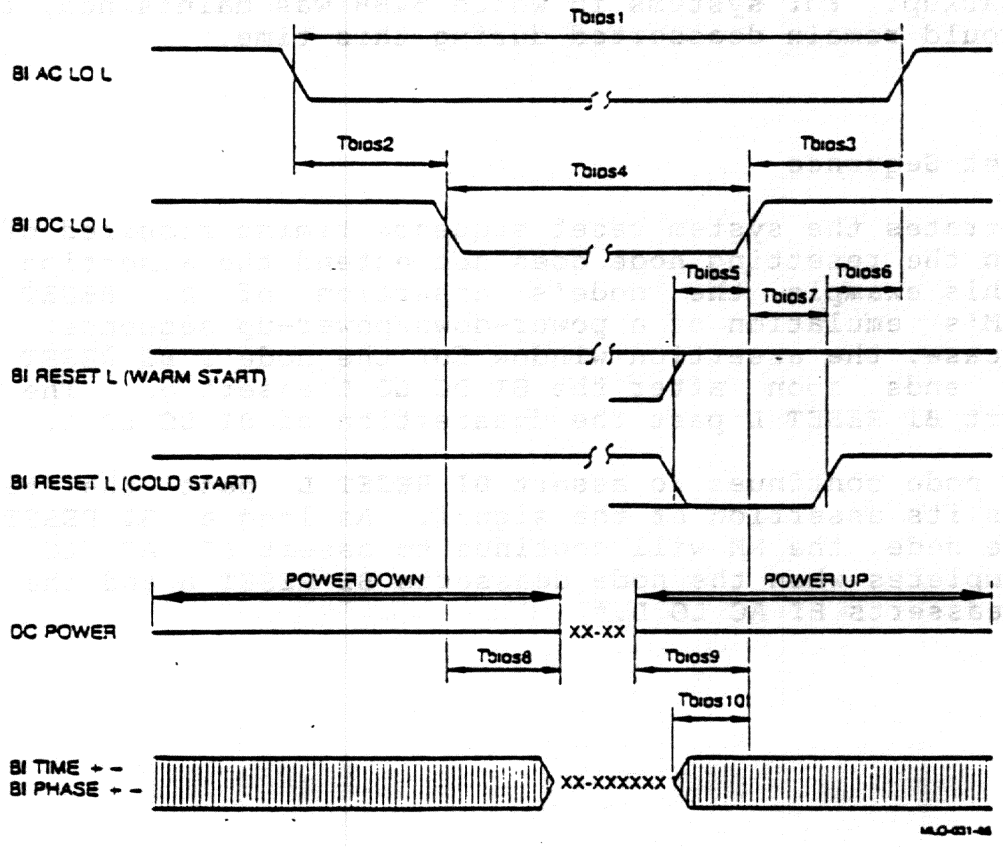


Figure 6-1: System Reset Sequence

6.6 INITIALIZATION TIMING SEQUENCES

6.6.1 Power-Down/Power-Up Sequence

Figure 6-1 shows a power-down/power-up sequence and the definitions of the associated timing parameters. In this example, after power-down, power goes away for some length of time before coming back. BI RESET L in VAXBI systems with battery backed-up memory must reflect the state of the battery backup voltage during the power-up sequence. When DC power is available, the RM asserts BI RESET L if it had sensed that 5VBB (volts battery backed up) had dropped below tolerance. It continues to assert the line until the specified time after BI DC LO L is deasserted. This same BI RESET L assertion would occur in systems without battery backup. For systems in which 5VBB was maintained, the BI RESET L line would remain deasserted during this time.

6.6.2 System Reset Sequence

Figure 6-2 demonstrates the system reset sequence timing required of a VAXBI system when the resetting node does not extend the assertion of BI AC LO L. In this example, the node's assertion of BI RESET L initiates the RM's emulation of a power-down/power-up sequence. In the system reset case, the assertion window for the node's BI RESET L signal assertion ends soon after the BI DC LO L assertion. The RM continues to assert BI RESET L past the deassertion of BI DC LO L.

In Figure 6-3 the node continues to assert BI RESET L past the time that the RM stops its assertion of the signal. As long as BI RESET L is asserted by the node, the RM will continue to assert BI AC LO L. The sequence completes when the node deasserts BI RESET L and the RM correspondingly deasserts BI AC LO L.*

---

*Extended system reset was intended for the purpose explained in section 6.2.2.2, but is not used by any known VAXBI option, and it is not supported by any VAXBI-based system.

digital™

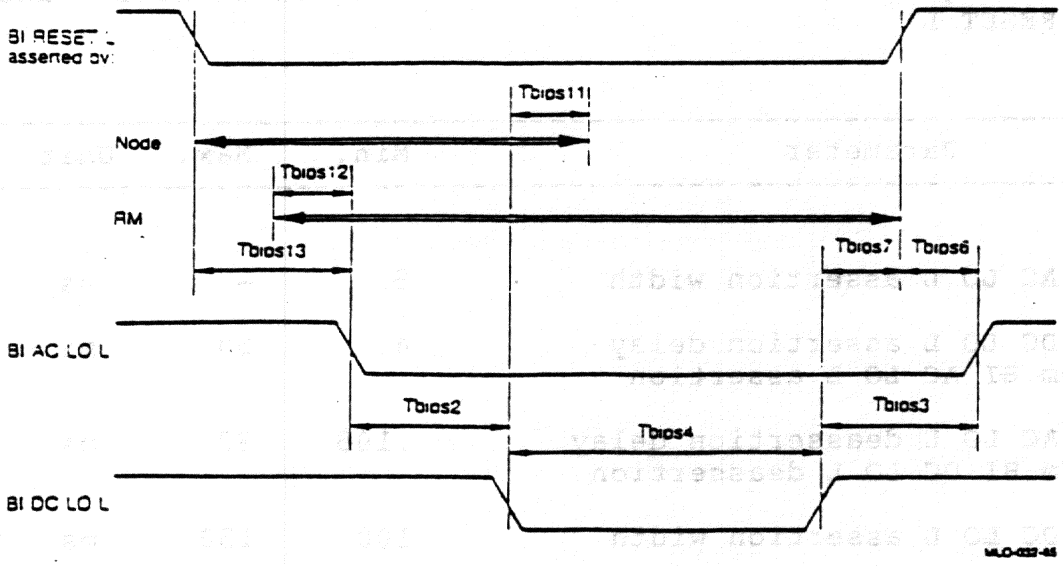


Figure 6-2: System Reset Timing Diagram

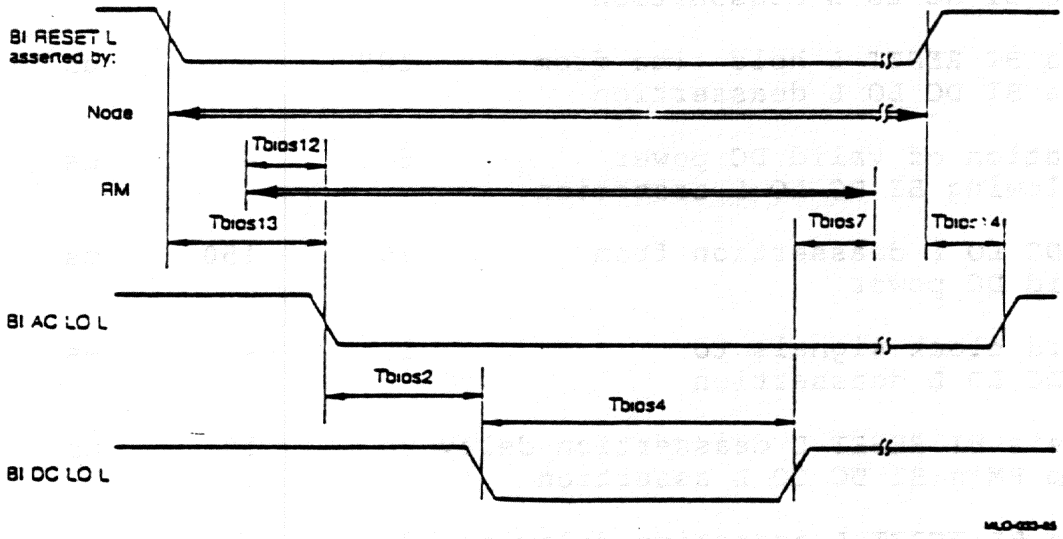


Figure 6-3: "Extended" System Reset Timing Diagram

Table 6-1: Timing Specifications for BI AC LO L, BI DC LO L, and BI RESET L

Symbol	Parameter	Min.	Max.	Unit	Note
Tbips1	BI AC LO L assertion width	5	-	us	-
Tbips2	BI DC LO L assertion delay from BI AC LO L assertion	4	50	ms	1
Tbips3	BI AC LO L deassertion delay from BI DC LO L deassertion	.105	30	ms	2
Tbips4	BI DC LO L assertion width	100	150	ms	2
Tbips5	RM's BI RESET L setup time to BI DC LO L deassertion	5	-	us	-
Tbips6	RM's BI RESET L setup time to RM's BI AC LO L deassertion	5	-	us	-
Tbips7	RM's BI RESET L hold time from RM's BI DC LO L deassertion	100	-	us	-
Tbips8	Duration of valid DC power following BI DC LO L assertion	5	-	us	-
Tbips9	BI DC LO L deassertion from valid DC power	70	150	ms	-
Tbips10	Valid clock signals to BI DC LO L deassertion	1	-	ms	-
Tbips11	Node's BI RESET L deassertion delay from RM's BI DC LO L assertion	0	10	ms	-
Tbips12	RM's BI RESET L assertion delay to RM's BI AC LO L assertion	0	-	-	3
Tbips13	RM's BI AC LO L assertion delay from node's BI RESET L assertion	0	100	ms	-
Tbips14	RM's BI AC LO L deassertion delay from node's BI RESET L deassertion	0	150	ms	-
Tr, Tf	BI RESET L rise time, fall time (10% to 90%)	0	1	us	4

NOTES

1. With certain power supplies, during certain brownout power conditions, BI AC LO L may assert and later deassert without an assertion of BI DC LO L.

2. Maximum specification does not apply for a "real" power sequence case.

3. This specification means that the RM must assert BI RESET L upon the detection of a BI RESET L assertion by a node, at least by the time it asserts BI AC LO L.

4. The maximum time of 1 microsecond corresponds to a maximum capacitance load of 3000 pF. With present VAXBI card cages, terminators, and extension cables, the signal loading exclusive of BI RESET L cables is approximately 500 pF.

Digital Internal Use Only
INITIALIZATION

CHAPTER 7

VAXBI REGISTERS

Each VAXBI node is required to implement a minimum set of registers contained in specific locations within the node's nodespace. Table 7-1 indicates which registers are required by node class. For example, a node of the processor class (as defined in Chapter 8) is required to implement registers indicated by AN (meaning all nodes must implement this register) AND AP (meaning processor nodes must also implement this additional register).

Unless otherwise noted, the VAXBI registers have the following characteristics:

- READ, RCI, and IRCI commands are all treated as READ commands. There is no lock and should be no cacheing.
- WRITE and WCI commands are treated the same.
- WMCI and UWMCI are treated the same. Mask functionality is implemented.
- No STALL or RETRY responses are permitted.
- Only longword data lengths are implemented.

All registers except the Slave-Only Status Register (SOSR) and the Receive Console Data Register (RXCD) are located in the BIIC. Unless otherwise noted, register addresses are reserved for the use specified even if a node does not implement the register.

Transactions to BIIC registers are limited to a maximum length of longword. The BIIC interprets the RESERVED data length code H H as a word data length code. As such, the results of write-type transactions to BIIC CSR space depend on C/A D<1> if the L L (BCI polarity) data length code is received.

digital™

Table 7-1: VAXBI Registers

Name	Abbrev.	Address	Node Requirements
Device Register	DTYPE	bb + 0*	AN**
VAXBI Control and Status Register	VAXBICSR	bb + 4	AN
Bus Error Register	BER	bb + 8	AN
Error Interrupt Control Register	EINTRCSR	bb + C	AN
Interrupt Destination Register	INTRDES	bb + 10	AN
IPINTR Mask Register	IPINTRMSK	bb + 14	AP
Force-Bit IPINTR/STOP Destination Register	FIPSDES	bb + 18	AP
IPINTR Source Register	IPINTRSRC	bb + 1C	AP
Starting Address Register	SADR	bb + 20	AM
Ending Address Register	EADR	bb + 24	AM
BCI Control and Status Register	BCICSR	bb + 28	None
Write Status Register	WSTAT	bb + 2C	None
Force-Bit IPINTR/STOP Command Register	FIPSCMD	bb + 30	None
User Interface Interrupt Control Register	UINTRCSR	bb + 40	None
General Purpose Register 0	GPR0	bb + F0	None
General Purpose Register 1	GPR1	bb + F4	None
General Purpose Register 2	GPR2	bb + F8	None
General Purpose Register 3	GPR3	bb + FC	None
Slave-Only Status Register	SOSR	bb + 100	SO
Receive Console Data Register	RXCD	bb + 200	None

*"bb" refers to the base address of a node (the address of the first location of the nodespace).

**Key to Node Requirements column:

AN	Register must be implemented by all nodes.
AP	Register must be implemented by all processor-class nodes.
AM	Register must be implemented by all memory-class nodes.
SO	Register must be implemented by all slave-only nodes.
None	Register not required for any node class.

digital™

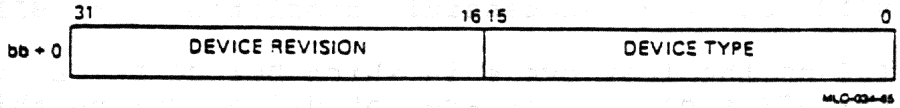
The register descriptions use the codes listed below to describe the type of bits in the VAXBI registers.

DCLOC	Cleared by the BIIC at the successful completion of BIIC self-test.
DCLOL	Loaded by the BIIC on the last cycle in which BCI DC LO L is asserted. If the BCI signal lines are not driven during this cycle, these bits are set.
DCLOS	Set by the BIIC at the successful completion of BIIC self-test.
DMW	BIIC diagnostic mode writable; reserved for use by Digital.
RO	Read-only bit. Write-type transactions do not change the value of this bit.
R/W	Normal read/write bit.
SC	Special case; operation defined in detailed description.
STOPC	Cleared by the BIIC on receipt of a STOP command to the node.
STOPS	Set by the BIIC on receipt of a STOP command to the node.
W1C	Write-1-to-clear bit. Write-type transactions cannot set this bit.

Unless otherwise noted, "set" and "clear" refer to high and low states, respectively.

digital™

7.1 DEVICE REGISTER



The Device Register contains information to identify the node. Both fields are loaded from the BCI D<31:0> H lines during the last cycle in which BCI DC LO L is asserted. Internal pullups on the BCI D<31:0> H lines will load this register with all ones if the BCI data lines are not driven while BCI DC LO L is asserted. Designers should verify that the output current characteristics of these pullup devices are sufficient for their needs (see Section 20.2 for DC characteristics).

Section 11.1.7 discusses the use of this register.

Bits: 31:16 Name: Device Revision (DREV)
Type: R/W, DMW, DCLOL

Identifies the revision level of the device. The use of the Device Revision field is implementation dependent. Refer to Application Note 10 for preferred usage.

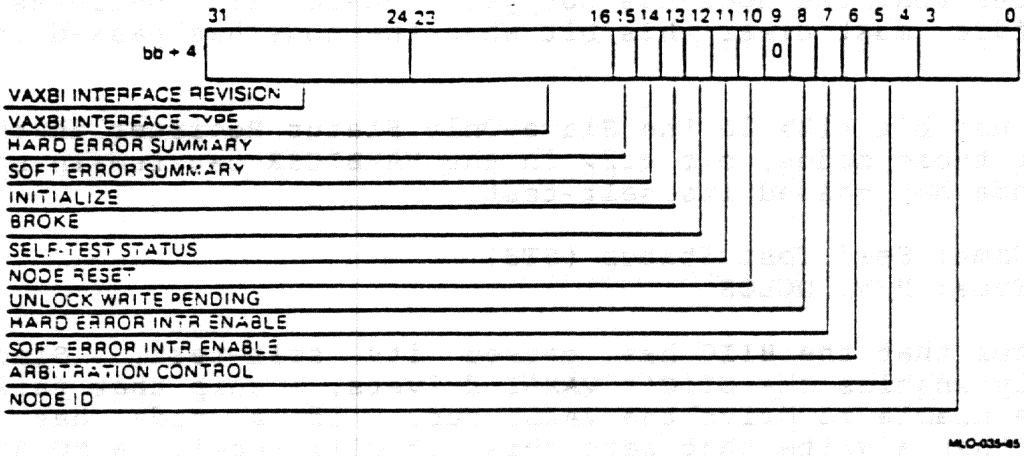
Bits: 15:0 Name: Device Type (DTYPE)
Type: R/W, DMW, DCLOL

Identifies the type of node. The device type must be initialized as specified in Section 11.1.7.

When bit <13> is set, the BIIC does not terminate a transaction even though the node receives more than 127 STALLs (128 STALLs is the required timeout period). When bit <13> is set, the BIIC may continue the assertion of BI BSY L beyond the interval of 127 STALLs. Also, since there is no timeout, the STO bit in the Bus Error Register does not set and the STO EV code does not occur. Operation of the BIIC with bit <13> set is a mode RESERVED to Digital and requires an exception.

digital™

7.2 VAXBI CONTROL AND STATUS REGISTER



Bits: 31:24 Name: VAXBI Interface Revision (IREV)
Type: RO

Indicates the revision of the device that provides the primary interface to the VAXBI bus. The contents of the BIIC's VAXBI Interface Revision field will be incremented for each major revision of the mask set (that is, each "pass").

Bits: 23:16 Name: VAXBI Interface Type (ITYPE)
Type: RO

Indicates the type of device that provides the primary interface to the VAXBI. The BIIC's VAXBI Interface field will always contain 0000 0001.

Bit: 15 Name: Hard Error Summary (HES)
Type: RO

When set, indicates that one or more of the hard error bits in the Bus Error Register is set.

Bit: 14 Name: Soft Error Summary (SES)
Type: RO

When set, indicates that one or more of the soft error bits in the Bus Error Register is set.

Bit: 13 Name: Initialize (INIT)
Type: W1C, DCLOS, STOPS

Usage of this bit is implementation dependent.

Bit: 12 Name: Broke
Type: W1C, DCLOS

When set, indicates that the node has not yet passed its self-test. The user interface must clear this bit when the node has passed its self-test.

Slave-only nodes use bit <12> in the Slave-Only Status Register as the Broke bit. For these nodes, bit <12> in the VAXBICSR can remain set even after the node has passed its self-test.

Bit: 11 Name: Self-Test Status (STS)
Type: R/W, DCLOS

When set, indicates that the BIIC has passed its self-test. Since this bit directly enables the BIIC's VAXBI drivers, a chip that fails self-test will be unable to drive the VAXBI bus. If a node has a reset STS bit, then a write that sets this bit will receive a NO ACK response. Because the node's VAXBI driver is disabled, the write must be either a loopback or a VAXBI internode transaction.

Bit: 10 Name: Node Reset (NRST)
Type: SC

Writing a one to this location initiates a complete node self-test. When this bit is written as a one, the Self-Test Status (STS) bit must also be written as a one to ensure proper operation of the write-type transaction. Reads to this bit location will return a zero. During the self-test sequence the STS bit will automatically be reset by the BIIC to allow proper recording of the new self-test results at the end of self-test. Unlike self-test during power-up operation, the BIIC does not assert the BI NO ARB L line.

Before writing to NRST, the resetting node must disable arbitration on the target node by setting both VAXBICSR ARB bits on the target node. This must be accomplished by a specific sequence of transactions. See section 6.2.3 Node Reset. The BIIC asserts the BCI DC LO L line for Tnrw (see BIIC AC Timing Specifications, Section 20.3) following the setting of the NRST bit. When the BCI DC LO L line is deasserted, the BIIC begins its self-test. The node reset operation simulates a power-down/power-up sequence at the node (except that BI AC LO L is not asserted and power remains valid at all times). The capability of resetting individual VAXBI nodes complements the full "system reset" functionality provided by sequencing the BI AC LO L and BI DC LO L lines. (For more information on use of this bit, see Section 6.2.3, Node Reset, and Section 11.1, Self-Test Operation.)

The Null Bus Parity Error bit in the Bus Error Register may set spuriously at the conclusion of a BIIC self-test triggered by a node reset.

digital™

Bit: 9 Name: RESERVED and zeros
Type: RO

Bit: 8 Name: Unlock Write Pending (UWP)
Type: W1C, DCLOC, SC

When set, indicates that an IRCI transaction has been completed successfully by the master port interface at this node, and this node has not yet issued a subsequent UWMCI command. The bit is cleared by a UWMCI transaction that is completed successfully by the master port interface. If a UWMCI transaction is attempted by the master port interface when the UWP bit is not set, the ISE bit in the Bus Error Register will be set. (The BIIC completes the UWMCI transaction in the normal manner.)

Bit: 7 Name: Hard Error INTR Enable (HEIE)
Type: R/W, DCLOC, STOPC

When set, enables an error interrupt to be generated by the node when the Hard Error Summary (HES) bit is set.

Bit: 6 Name: Soft Error INTR Enable (SEIE)
Type: R/W, DCLOC, STOPC

When set, enables an error interrupt to be generated by the node when the Soft Error Summary (SES) bit is set.

Bits: 5:4 Name: Arbitration Control (ARB)
Type: R/W, DCLOC

Indicates the mode of arbitration to be used by the node (see Table 7-2).

Table 7-2: Arbitration Codes

Bit 5	Bit 4	Meaning
0	0	Dual round-robin arbitration
0	1	Fixed-high priority (RESERVED)
1	0	Fixed-low priority (RESERVED)
1	1	Disable arbitration (RESERVED)

The arbitration following the writing of bits <5:4> is performed based on the old bit setting. The arbitration mode is not updated until the end of the next imbedded ARB cycle. Subsequent arbitrations will reflect the new setting.

digital™

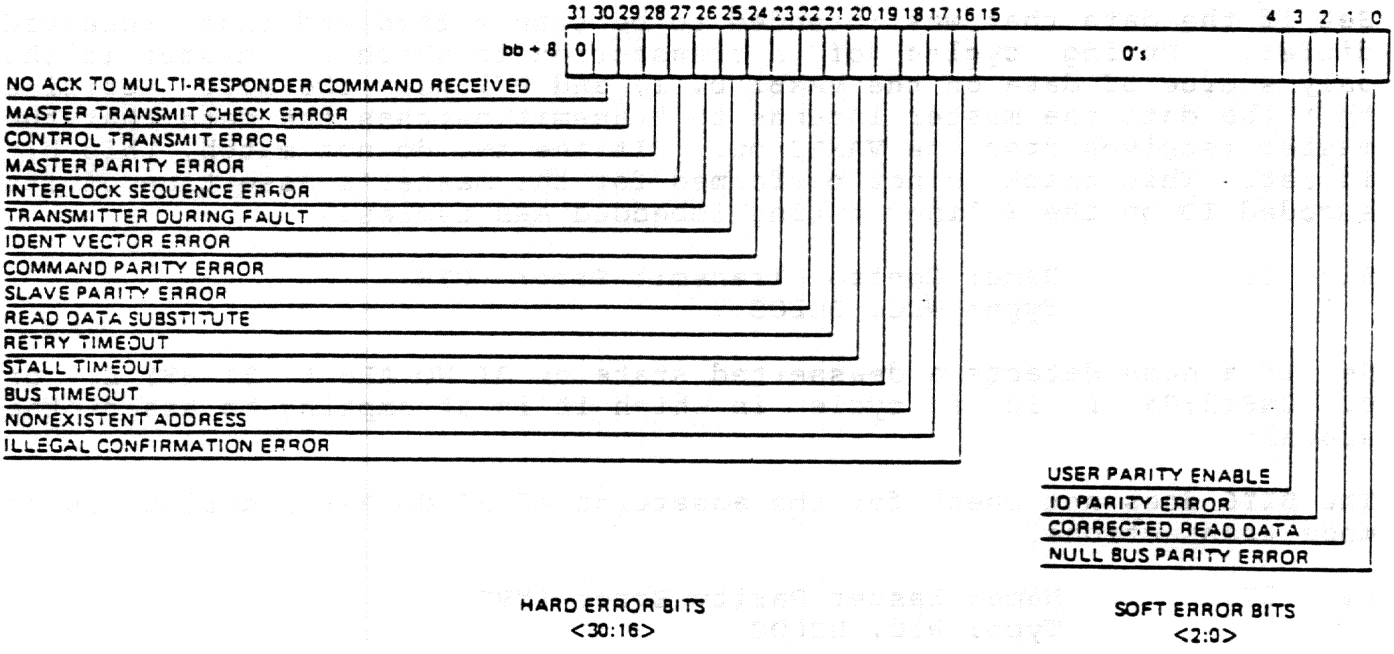
The "disable arbitration" mode can be used to prevent a node from starting a VAXBI transaction. When bits <5:4> are both set, the BIIC can assert the BI NO ARB L line, but it cannot assert its node ID, so it will not win an arbitration.

Bits: 3:0 Name: Node ID
Type: RO, DMW, DCLOL

Indicates the node's ID. This information is loaded from the BCI I<3:0> H lines during the last cycle in which BCI DC LO L is asserted. The user interface must drive the node ID on the BCI I<3:0> H lines while BCI DC LO L is asserted.

digital™

7.3 BUS ERROR REGISTER



Unless otherwise noted, all bits in the Bus Error Register can be set during VAXBI and loopback transactions. Bits <30:16> are hard error bits, and bits <2:0> are soft error bits. Bit <3>, the User Parity Enable (UPEN) bit, is not an error bit but indicates the BIIC parity mode. (All logic required to set Bus Error Register bits is contained in the BIIC.)

Bits: 31 Name: RESERVED and zero
Type: RO

Bit: 30 Name: NO ACK to Multi-Responder Command Received (NMR)
Type: W1C, DCLOC

Set if the master receives a NO ACK command response for an INVAL, STOP, INTR, IPINTR, BDCST, or RESERVED command.

digital™

Bit: 29 Name: Master Transmit Check Error (MTCE)
Type: W1C, DCLOC

Set if the data that was intended to be transmitted and that received differ. During cycles of a transaction in which the master is the only source of data on the VAXBI D, I, and P lines, the BIIC verifies that the data the master intends to transmit matches the data that the master receives from the VAXBI bus. If the two do not match, this bit is set. This check is not performed for the master's assertion of its encoded ID on the I lines during imbedded ARB cycles.

Bit: 28 Name: Control Transmit Error (CTE)
Type: W1C, DCLOC

Set if a node detects a deasserted state on BI NO ARB L, BI BSY L, or BI CNF<2:0> L in a cycle in which it is attempting to assert the signal.

The BIIC does not check for the assertion of BI NO ARB L during burst mode transactions.

Bit: 27 Name: Master Parity Error (MPE)
Type: W1C, DCLOC

Set if the master detects a parity error on the bus during a read-type or vector ACK data cycle.

Bit: 26 Name: Interlock Sequence Error (ISE)
Type: W1C, DCLOC

Set if the node successfully completes a UWMCI transaction when no corresponding IRCI transaction had been issued previously. This sequence error is evident because the Unlock Write Pending (UWP) bit in the VAXBI Control and Status Register was not set.

Bit: 25 Name: Transmitter During Fault (TDF)
Type: W1C, DCLOC

Set if either the master or slave detected a parity error during a cycle in which the master or slave was responsible for transmitting proper parity on the VAXBI bus.

digital™

These cycles include the following types:

- o Command/address cycles (set by the master)
- o Read-type ACK data cycles (set by the slave)
- o Write-type data cycles (set by the master)
- o BDCST data cycles (set by the master)
- o Vector ACK data cycles (set by the slave)
- o Imbedded ARB cycles -- Encoded Master ID Parity Error (set by the master)
- o Master decoded ID cycle of IDENT (set by the master)

This bit is not set for parity errors that occur during loopback transactions.

Bit: 24 Name: IDENT Vector Error (IVE)
Type: W1C, DCLOC

Set if an ACK response is not received from the master. A set bit indicates that the interrupt vector was not correctly received.

Bit: 23 Name: Command Parity Error (CPE)
Type: W1C, DCLOC

Set if a parity error is detected in a command/address cycle. The transaction can be either a VAXBI or a loopback transaction.

Bit: 22 Name: Slave Parity Error (SPE)
Type: W1C, DCLOC

Set if a parity error is detected by the slave during a write-type ACK or write-type STALL data cycle or BDCST ACK data cycle.

Bit: 21 Name: Read Data Substitute (RDS)
Type: W1C, DCLOC

Set if a Read Data Substitute or RESERVED status code is received during a read-type or IDENT (for vector status) transaction. For this bit to be set, the BIIC logic also requires the receipt of good parity for the data cycle that contains the RDS or RESERVED code. This bit is set even if the transaction is aborted some time after the receipt of the RDS or RESERVED code. (See Section 5.3.4 for a description of the read status codes.)

Bit: 20 Name: RETRY Timeout (RTO)
Type: W1C, DCLOC

Set if the master receives 4096 consecutive RETRY responses from the slave for the same master port transaction.

Bit: 19 Name: STALL Timeout (STO)
Type: W1C, DCLOC

Set if the slave port asserts the STALL code on the BCI RS<1:0> L lines for 128 consecutive cycles and bit <13> of the Device Register is not set.

Bit: 18 Name: Bus Timeout (BTO)
Type: W1C, DCLOC

Set if the node is unable to start at least one transaction (out of possibly several that are pending) before 4096 cycles have elapsed.

Bit: 17 Name: Nonexistent Address (NEX)
Type: W1C, DCLOC

Set when the node receives a NO ACK response for a read- or write-type command. This bit is set only if this node's parity check and master transmit check of the command/address data were successful (that is, CPE and MTCE were not set for this C/A cycle). This bit is not set for NO ACK responses to other commands.

Bit: 16 Name: Illegal Confirmation Error (ICE)
Type: W1C, DCLOC

Set if a RESERVED or illegal confirmation code is received by this node. This bit can be set by either the master or slave node. Note that a NO ACK command confirmation is not an illegal response.

Bits: 15:4 Name: RESERVED and zeros
Type: RO

Bit: 3 Name: User Parity Enable (UPEN)
Type: RO, DCLOL

Indicates the BIIC parity mode. A one indicates that the user interface is to generate parity; a zero indicates that the BIIC is to generate parity. These codes are the reverse of those on the BCI P0 line during BI DC LO L which indicate who generates parity. On power-up a high (default) on BCI P0 configures the BIIC for BIIC-generated parity, whereas a low configures the chip for user interface-generated parity.

When UPEN is set, the user interface is required to provide parity on the BCI P0 L line whenever BCI SDE L or BCI MDE L is asserted (that is, whenever data is solicited from the user interface).

Bit: 2 Name: ID Parity Error (IPE)
Type: W1C, DCLOC

Set if a parity error is detected on the BI I lines when the master's encoded ID is asserted during imbedded ARB cycles. All nodes perform this parity check.

This bit is not set during loopback transactions.

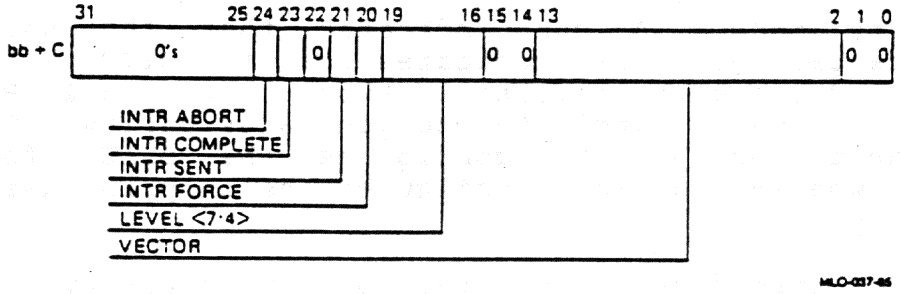
Bit: 1 Name: Corrected Read Data (CRD)
Type: W1C, DCLOC

Set if the master receives a Corrected Read Data status code. For this bit to be set, the BIIC logic also requires the receipt of good parity for the data cycle that contains the CRD code. This bit is set even if the transaction aborts after the CRD status code has been received. (See Section 5.3.4 for a description of the read status codes.)

Bit: 0 Name: Null Bus Parity Error (NPE)
Type: W1C, SC

Set if ODD parity is detected on the bus during the second cycle of a two-cycle sequence during which BI NO ARB L and BI BSY L were not asserted. This bit is cleared on power-up; however the state of NPE that results from undergoing a node reset is UNPREDICTABLE.

7.4 ERROR INTERRUPT CONTROL REGISTER



The Error Interrupt Control Register controls the operation of interrupts initiated by a BIIC-detected bus error (which sets a bit in the Bus Error Register) or by the setting of a force bit in this register. In the descriptions that follow, an error interrupt request can be initiated either by the setting of a force bit or by the setting of a Bus Error Register bit (assuming the appropriate error interrupt enables are set in the VAXBI Control and Status Register).

Bits: 31:25 Name: RESERVED and zeros
Type: RO

Bit: 24 Name: INTR Abort (INTRAB)
Type: W1C, DCLOC, SC

Set if an INTR command sent under the control of this register is aborted (that is, a NO ACK or illegal confirmation code is received). INTRAB is a status bit set by the BIIC and can be reset only by the user interface. The bit has no effect on the ability of the BIIC to send or respond to further INTR or IDENT transactions.

Bit: 23 Name: INTR Complete (INTRC)
Type: W1C, DCLOC, SC

Set when the vector for an error interrupt has been successfully transmitted or if an INTR command sent under the control of this register has aborted. Removal of the error interrupt request resets this bit. No interrupts are generated by this register when the INTRC bit is set. Further, no IDENTS will be responded to by this register when the INTRC bit is set.

Bit: 22 Name: RESERVED and zero
Type: RO

digital™

Bit: 21 Name: INTR Sent
Type: W1C, DCLOC, STOPC, SC

Set when an INTR command has been sent. This bit is cleared during an IDENT transaction following the detection of a level and master ID match. Clearing the bit allows the interrupt to be resent if the node loses the IDENT arbitration or if the node wins but the vector transmission fails. Removal of the error interrupt request resets the INTR Sent bit.

Bit: 20 Name: INTR Force
Type: R/W, DCLOC, STOPC

When set, posts an error interrupt request in the same way as a bit set in the Bus Error Register, except that the request is not qualified by the HEIE and SEIE bits.

Bit: 19:16 Name: Level <7:4>
Type: R/W, DCLOC

Indicates the level(s) at which INTR commands under the control of this register are transmitted.

Also, the Level field is used by the node to determine whether it will respond to an IDENT command. If any level bits of the received IDENT command match the bits set in the Level field of this register, the node will participate in the IDENT arbitration, provided there is also a match in the INTR Destination Register.

A level bit must be set for the BIIC to transmit an error interrupt.

Bits: 15:14 Name: RESERVED and zeros
Type: RO

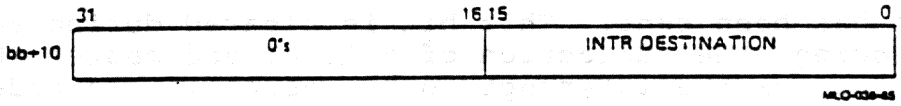
Bits: 13:2 Name: Vector
Type: R/W, DCLOC

Contains the vector to be used during error interrupt sequences.

Bits: 1:0 Name: RESERVED and zeros
Type: RO

digital™

7.5 INTR DESTINATION REGISTER



Bits: 31:16 Name: RESERVED and zeros
Type: RO

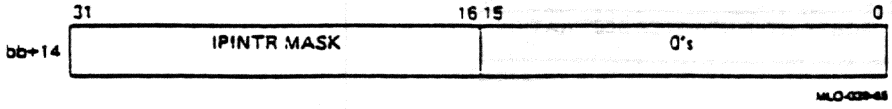
Bits: 15:0 Name: INTR Destination (INTRDES)
Type: R/W, DCLOC

Indicates which nodes are to be selected by INTR commands. The destination is sent out during the INTR command and is monitored by all nodes to determine whether to respond.

During an IDENT command, a node compares the transmitted master's decoded ID to the nodes in the INTR Destination field. If there is no match, this node will not respond to the IDENT. If there is a match, that is, if the bit corresponding to the master's decoded ID is set in the INTR Destination Register, then the node will respond to the IDENT, provided that it has an unserviced INTR request that matches the level transmitted in the IDENT command.

digital™

7.6 IPINTR MASK REGISTER



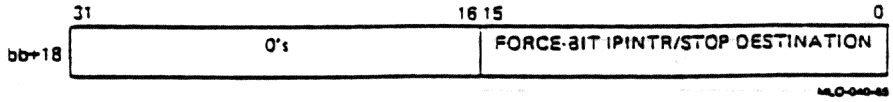
Bits: 31:16 Name: IPINTR Mask
Type: R/W, DCLOC

Indicates which nodes are permitted to send IPINTRs to this node. If a bit in the IPINTR Mask field is a one, IPINTRs directed at this node from the corresponding node will cause selection (assuming the IPINTREN bit in the BCI Control and Status Register is set). If the bit is a zero, IPINTRs from that node will not cause selection.

Bits: 15:0 Name: RESERVED and zeros
Type: RO

digital™

7.7 FORCE-BIT IPINTR/STOP DESTINATION REGISTER



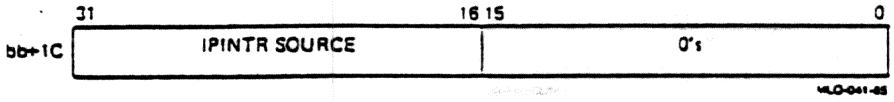
Bits: 31:16 Name: RESERVED and zeros
Type: RO

Bits: 15:0 Name: Force-Bit IPINTR/STOP Destination
Type: R/W, DCLOC

Indicates which nodes are to be targeted by force-bit IPINTR or STOP commands sent by this node. Master port IPINTR transactions use command/address data for this field supplied by the user interface.

digital™

7.8 IPINTR SOURCE REGISTER



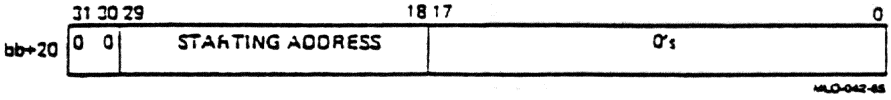
Bits: 31:16 Name: IPINTR Source
Type: W1C, DCLOC, SC

Used by the BIIC to store the decoded ID of a node that sends an IPINTR command to the node. Each bit corresponds to one node on the VAXBI bus. The bit corresponding to the IPINTR master's ID is set when an IPINTR command whose destination matches the ID of this node and whose ID matches a bit in the IPINTR Mask Register is received. The bit in the IPINTR Source Register is set only if the IPINTR command is received with good parity. It is not required that the IPINTREN bit be set in the BCI Control and Status Register for the appropriate IPINTR Source Register bit to be set.

Bits: 15:0 Name: RESERVED and zeros
Type: RO

digital™

7.9 STARTING ADDRESS REGISTER



The Starting and Ending Address Registers define storage blocks in either memory or I/O space.

The Starting and Ending Address Registers must not be configured to include nodespace or multicast space. Software should set up the Starting Address Register before the Ending Address Register to avoid selection problems that may be caused by loading the Ending Address Register with a nonzero value while the Starting Address Register remains cleared.

If the Starting Address Register is set to a value greater than or equal to the contents of the Ending Address Register, no addresses will be recognized.

Bits: 31:30 Name: RESERVED and zeros
Type: RO

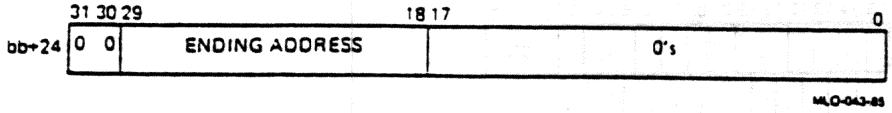
Bits: 29:18 Name: Starting Address
Type: R/W, DCLOC

Determines the address of the first location of a 256-Kbyte block of addresses to be recognized by the BIIC for selection of the slave port.

Bits: 17:0 Name: RESERVED and zeros
Type: RO

digital™

7.10 ENDING ADDRESS REGISTER



The Starting and Ending Address Registers define storage blocks in either memory or I/O space.

The Starting and Ending Address Registers must not be configured to include nodespace or multicast space. Software should set up the Starting Address Register before the Ending Address Register to avoid selection problems that may be caused by loading the Ending Address Register with a nonzero value while the Starting Address Register remains cleared.

If the Starting Address Register is set to a value greater than or equal to the contents of the Ending Address Register, no addresses will be recognized.

Bits: 31:30 Name: RESERVED and zeros
Type: RO

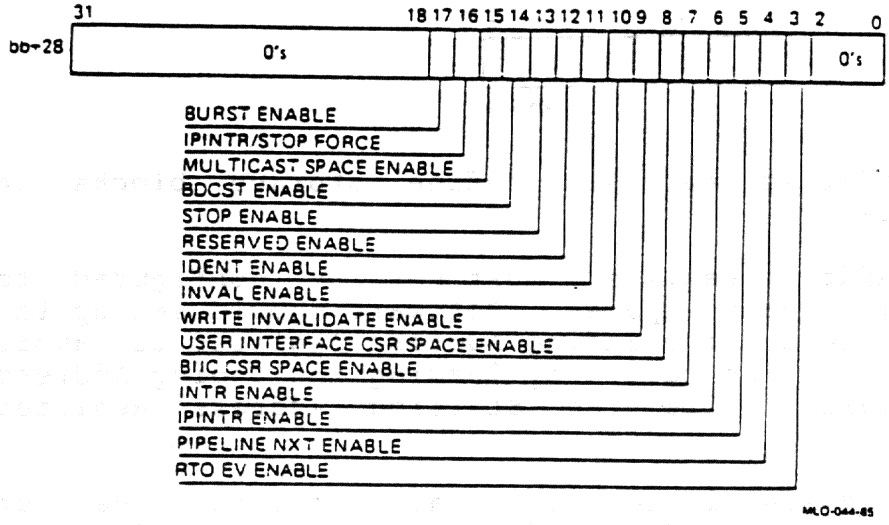
Bits: 29:18 Name: Ending Address
Type: R/W, DCLOC

Indicates the address that is one greater than the highest address recognized by the BIIC for selection of the slave port. The address must be the first location of a 256-Kbyte block of addresses. For example, if the Starting Address Register contains 1C44 0000 and the Ending Address Register contains 1D68 0000, then the BIIC will recognize addresses 1C44 0000 through 1D67 FFFF for selection of the slave port. The register definition prevents VAXBI accesses to the top 256 Kbytes of I/O space. (This block is also in RESERVED space.)

Bits: 17:0 Name: RESERVED and zeros
Type: RO

digital™

7.11 BCI CONTROL AND STATUS REGISTER



This register description makes reference to the BCI SEL L and BCI SC<2:0> L lines, which are described in Sections 15.5.4 and 15.5.5.

The following categories describe the effects of the enable bits in the BCI Control and Status Register on BIIC operation. The category for each bit is given after the other bit characteristics.

- Disables selection. When a bit of this type is reset, the BIIC both suppresses the appropriate SEL/SC assertion and does not respond in any way to transactions corresponding to that enable bit. For example, if the INTREN bit is reset, the node will not be selected for any INTR transactions received from the VAXBI bus. Most of the enable bits are in this class.

- Special case. Some bits do not simply disable participation. Details on how the bit operates are in the bit description.

- Not applicable. These bits have no effect on slave selection.

Bits: 31:18 Name: RESERVED and zeros
Type: RO

Bit: 17 Name: Burst Enable (BURSTEN)
Type: R/W, DCLOC - Not applicable

When set, the BIIC asserts BI NO ARB L after the next successful arbitration by this node until the BURSTEN bit is reset or BCI MAB L is asserted. The assertion of BCI MAB L does not reset the BURSTEN bit. It merely clears the burst mode state in the BIIC, which is holding BI NO ARB L. Unless a subsequent transaction clears this bit,

digital™

then the next successful arbitration by this node will cause the BIIC to once again hold BI NO ARB L continuously.

Burst mode must not be used with loopback transactions, since the loopback transaction will not be able to start, due to the assertion of BI NO ARB L.

Bit: 16 Name: IPINTR/STOP Force (IPINTR/STOP FORCE)
Type: R/W, DCLOC, SC - Not applicable

When set, the BIIC arbitrates for the bus and transmits an IPINTR or STOP command (depending on the command stored in the Force-Bit IPINTR/STOP Command Register), using the Force-Bit IPINTR/STOP Destination Register for the destination field. The IPINTR/STOP Force bit is reset by the BIIC following the transmission of the IPINTR transaction. If the transmission fails, the NICIPS (NO ACK or Illegal CNF Received for Force-Bit IPINTR/STOP Command) EV code is output and the NMR (NO ACK to Multi-Responder Command Received) bit is set.

Bit: 15 Name: Multicast Space Enable (MSEN)
Type: R/W, DCLOC - Disables selection

When set, the BIIC asserts SEL and the appropriate SC<2:0> code following the receipt of a read- or write-type command directed at multicast space.

Bit: 14 Name: BDCST Enable (BDCSTEN)
Type: R/W, DCLOC - Disables selection

When set, the BIIC asserts SEL and the appropriate SC<2:0> code following the receipt of a BDCST command directed at this node. (See Appendix A for the description of the BDCST transaction.)

Bit: 13 Name: STOP Enable (STOPEN)
Type: R/W, DCLOC - Disables selection

When set, the BIIC asserts SEL and the appropriate SC<2:0> code following the receipt of a STOP command directed at this node.

Bit: 12 Name: RESERVED Enable (RESEN)
Type: R/W, DCLOC - Special case

When set, the BIIC asserts SEL and the appropriate SC<2:0> code following the receipt of a RESERVED command code. (See Section 18.3.10 on RESERVED commands.)

Bit: 11 Name: IDENT Enable (IDENTEN)
Type: R/W, DCLOC - Special case

When set, the BIIC asserts SEL and the appropriate SC<2:0> code following the receipt of an IDENT command. This bit affects only the

digital™

output of SEL and the IDENT SC code. Therefore, the BIIC will always participate in IDENT transactions that select this node even if this enable bit is reset.

Bit: 10 Name: INVAL Enable (INVALEN)
Type: R/W, DCLOC - Disables selection

When set, the BIIC asserts SEL and the appropriate SC<2:0> code following the receipt of an INVAL command.

Bit: 9 Name: WRITE Invalidate Enable (WINVALEN)
Type: R/W, DCLOC - Special case

When set, the BIIC asserts SEL and the appropriate SC<2:0> code following the receipt of a write-type command whose address does not fall within the bounds set by the Starting and Ending Address Registers, but which has D<29> equal to zero (that is, not I/O space).

Nodes that monitor VAXBI write-type transactions by using the WINVALEN SC code cannot participate in these transactions.

Bit: 8 Name: User Interface CSR Space Enable (UCSREN)
Type: R/W, DCLOC - Disables selection

When set, the BIIC asserts SEL and the appropriate SC<2:0> code following the receipt of a read- or write-type command directed at this node's user interface CSR space.

Bit: 7 Name: BIIC CSR Space Enable (BICSREN)
Type: R/W, DCLOC - Special case

When set, the BIIC asserts SEL and the appropriate SC<2:0> code following the receipt of a read- or write-type command directed at this node's BIIC CSR space. The BIIC's response to BIIC CSR space accesses cannot be disabled; the BIIC always participates in transactions that access its BIIC CSR space.

Note that this bit makes it easy to keep "shadow copies" of BIIC internal registers, as writes to these registers can be treated the same as writes to user interface CSR space (with the exception that the slave cannot stall).

Bit: 6 Name: INTR Enable (INTREN)
Type: R/W, DCLOC - Disables selection

When set, the BIIC asserts SEL and the appropriate SC<2:0> code following the receipt of an INTR command directed at this node.

Bit: 5 Name: IPINTR Enable (IPINTREN)
Type: R/W, DCLOC - Special case

When set, the BIIC asserts SEL and the appropriate SC<2:0> code following the receipt of an IPINTR command from a node that is included in the IPINTR Mask Register. The state of this enable bit does not affect whether the node receives IPINTR commands. To ensure that a node does not receive IPINTRs, the user interface should clear the IPINTR Mask Register.

Bit: 4 Name: Pipeline NXT Enable (PNXTEN)
Type: R/W, DCLOC - Not applicable

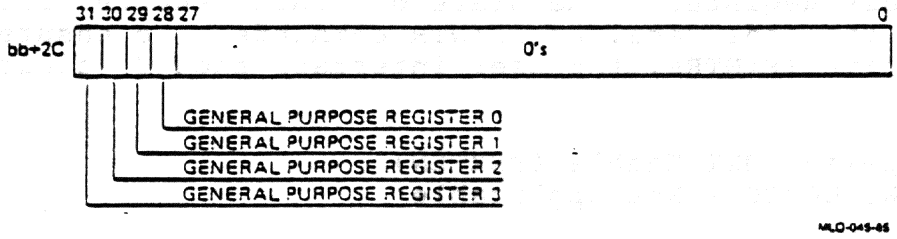
When set, the BIIC provides an extra BCI NXT L cycle (that is, one more than the number of longwords transferred) during write-type and BDCST transactions. This extra BCI NXT L cycle occurs after the last NXT L cycle for write data and makes it easier to implement FIFO pointers for some types of master port interface designs.

Bit: 3 Name: RTO EV Enable (RTOEVEN)
Type: R/W, DCLOC - Not applicable

When set, the BIIC outputs the RETRY Timeout (RTO) EV code in place of the RETRY CNF Received for Master Port Command (RCR) EV code following the occurrence of a retry timeout. If the bit is not set, the BIIC will not output the RTO EV code in place of the RCR EV code following a retry timeout; however, the RTO bit in the BER will be set and an error interrupt will be generated if enabled.

Bits: 2:0 Name: RESERVED and zeros
Type: RO

7.12 WRITE STATUS REGISTER



Bit: 31 Name: General Purpose Register 3 (GPR3)
Type: W1C, DCLOC

Bit: 30 Name: General Purpose Register 2 (GPR2)
Type: W1C, DCLOC

Bit: 29 Name: General Purpose Register 1 (GPR1)
Type: W1C, DCLOC

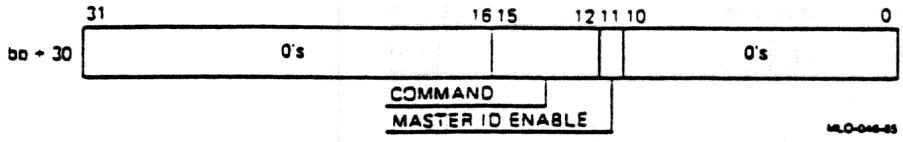
Bit: 28 Name: General Purpose Register 0 (GPR0)
Type: W1C, DCLOC

Bits <31:28> when set indicate which general purpose registers have been written to by a VAXBI transaction. The bit is set only if good parity is received with the write data.

These bits are not set by loopback transactions.

Bits: 27:0 Name: RESERVED and zeros
Type: RO

7.13 FORCE-BIT IPINTR/STOP COMMAND REGISTER



Bits: 31:16 Name: RESERVED and zeros
Type: RO

Bits: 15:12 Name: Command (CMD)
Type: R/W, DCLOS

Indicates the 4-bit command code for either an IPINTR or STOP transaction that is initiated by setting the IPINTR/STOP Force bit. Only the IPINTR (HHHH) and STOP (HHLL) command codes should be loaded into this field.

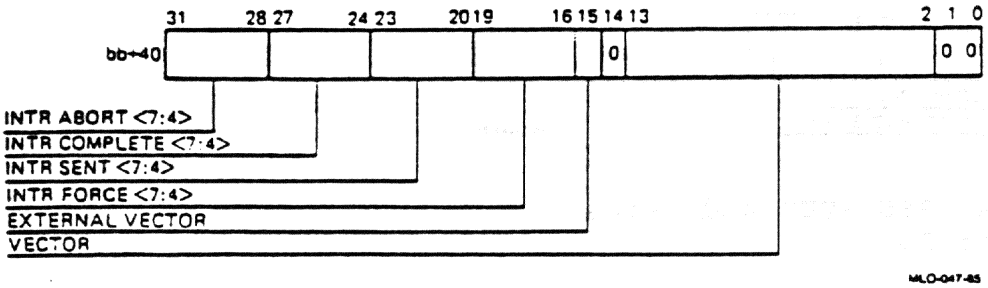
Bit: 11 Name: Master ID Enable (MIDEN)
Type: R/W, DCLOS

Determines whether the master's ID is transmitted on the BI D<31:16> L lines during the C/A cycle of a transaction initiated by setting the IPINTR/STOP Force bit. If the MIDEN bit is cleared, the BI D<31:0> L lines remain deasserted during the C/A cycle. The MIDEN bit should be set to one when the Command field contains the IPINTR command code. (The IPINTR transaction requires that the master's decoded ID be transmitted on BI D<31:16> L.) The MIDEN bit should be cleared when the Command field contains the STOP command code. (The STOP transaction requires that during the C/A cycle the BI D<31:16> L lines be a RESERVED field and should not be driven.)

Bits: 10:0 Name: RESERVED and zeros
Type: RO

digital™

7.14 USER INTERFACE INTERRUPT CONTROL REGISTER



The User Interface Interrupt Control Register controls the operation of interrupts initiated by the user interface. In the following discussion, the phrase "interrupt request" refers to interrupts initiated either by the assertion of any of the BCI INT<7:4> L lines or by the setting of any of the force bits in this register.

Bits: 31:28 Name: INTR Abort <7:4> (INTRAB)
Type: W1C, DCLOC, SC

The four INTR Abort bits correspond to the four interrupt levels. An INTR Abort bit is set if an INTR command sent under the control of this register is aborted (that is, a NO ACK or illegal confirmation code is received). INTRAB is a status bit set by the BIIC and can be reset only by the user interface. The bit has no effect on the ability of the BIIC to send or respond to further INTR or IDENT transactions.

Bits: 27:24 Name: INTR Complete <7:4> (INTRC)
Type: W1C, DCLOC, SC

The four INTR Complete bits correspond to the four interrupt levels. An INTR Complete bit is set when the vector for an interrupt has been successfully transmitted or if an INTR command sent under the control of this register is aborted. Removal of the interrupt request clears the corresponding INTRC bit. While an INTRC bit is set, no further interrupts at that level are generated by this register. Further, no IDENTS will be responded to by this register when the INTRC bit is set at the IDENT level.

Bits: 23:20 Name: INTR Sent <7:4> (SENT)
Type: W1C, DCLOC, STOPC, SC

The four INTR Sent bits correspond to the four interrupt levels. A set INTR Sent bit indicates that an INTR command for the corresponding level has been successfully transmitted. This bit is cleared during an IDENT command following the detection of a level and master ID match. Clearing the bit allows the interrupt to be resent if this

node loses the IDENT arbitration or if the node wins but the vector transmission fails. Deassertion of an interrupt request clears the INTR Sent bit.

It is not necessary for the INTR Sent bit at a given level to be set for the BIIC to respond to an IDENT at that level (that is, the interrupt need not have actually been transmitted on the VAXBI). All that is required is that an interrupt request have been posted that matches the IDENT level.

Bits: 19:16 Name: INTR Force <7:4> (FORCE)
Type: R/W, DCLOC, STOPC

When set, the BIIC generates interrupts at the specified level. The four INTR Force bits correspond to the four interrupt levels. Setting an INTR Force bit is equivalent to asserting the corresponding BCI INT<7:4> L line.

When multiple interrupt requests are asserted simultaneously, the BIIC transmits INTR commands for the highest priority requests first. Similarly, when an IDENT command solicits more than one level, the BIIC responds with the highest pending level. (See Section 18.4 for a discussion of the priority of transactions.)

Bit: 15 Name: External Vector (EX VECTOR)
Type: R/W, DCLOC

When set, the BIIC solicits the interrupt vector from the BCI D<31:0> H lines (rather than transmitting the vector contained in this control register) in response to an IDENT transaction that matches this register. The BIIC's slave port asserts an External Vector Selected at Level n EV code the cycle before the vector can be driven on the BCI D lines. A slave port interface using the BIIC must stall the vector at least one cycle (by asserting the STALL code on the RS lines during the IDENT arbitration cycle) before transmitting an ACK (with vector) or a RETRY response.*

Bit: 14 Name: RESERVED and zero
Type: RO

---

*To comply with VAXBI protocol, BIIC protocol requires a minimum of one STALL cycle before either of these two responses is generated. If no STALL is generated, the BIIC will not properly suppress the transmission of ACK or RETRY CNF codes from nodes that lose the IDENT arbitration during the cycle after the IDENT arbitration. The subsequent collision of CNF codes will then cause bus errors to occur.

digital™

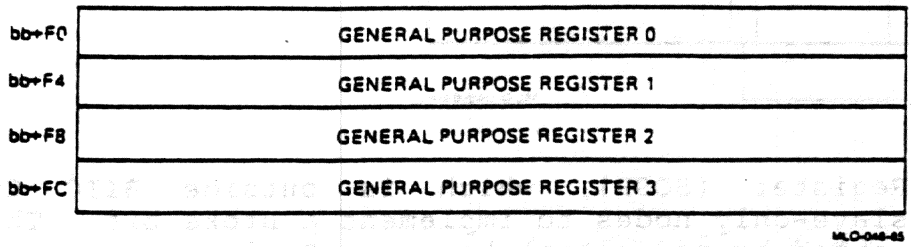
Bits: 13:2 Name: Vector
Type: R/W, DCLOC

Contains the vector used during user interface interrupt sequences (unless the External Vector bit is set). The vector is transmitted when this node wins an IDENT arbitration that matches the conditions given in the User Interface Interrupt Control Register.

Bits: 1:0 Name: RESERVED and zeros
Type: RO

digital™

7.15 GENERAL PURPOSE REGISTERS

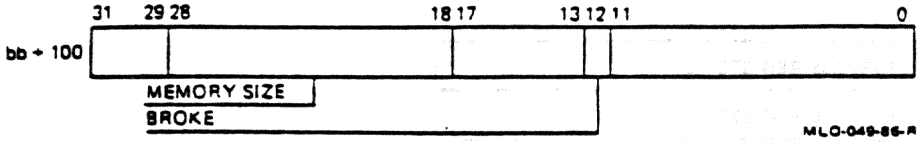


The use of the general purpose registers is implementation specific. The type of the bits in these registers is R/W, DCLOC.

Whenever one of these registers is written, a bit is set in the Write Status Register to indicate which register was written.

digital™

7.16 SLAVE-ONLY STATUS REGISTER



The Slave-Only Status Register (SOSR), which is outside BIIC CSR space, is used by slave-only nodes to implement a Broke bit. This register must be implemented by nodes that have a Device Type code with zeros in bits <14:8>). When implemented, both the Broke bit and the Memory Size field must have valid values. This register must never be written.

Bits: 31:29 Implementation dependent

Bits: 28:18 Name: Memory Size (MSIZE)
Type: RO

Indicates the size of the memory as a multiple of 2**18 bytes (256 Kbytes) expressed as a binary number. Slave-only non memory nodes must load the MSIZE field with 0 before or at each transition of the Broke bit from set to cleared. When DTYPE<14:8> = 0 and SOSR<28:18> = 0, the operating system must treat the node as a slave-only non-memory device and not as a memory.

Bits: 17:13 Implementation dependent

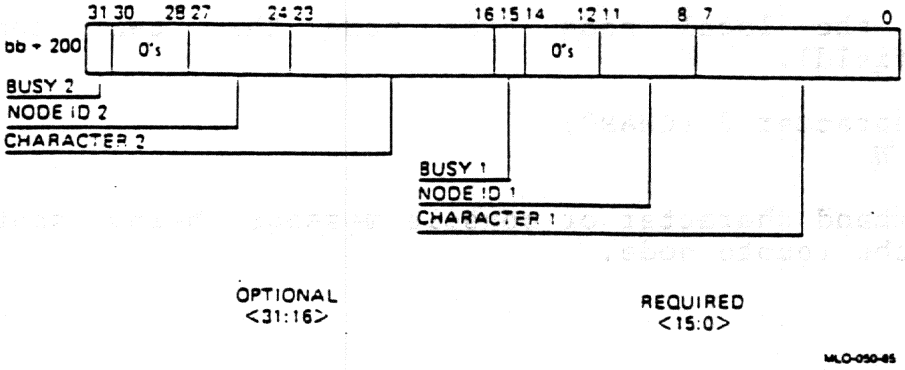
Bit: 12 Name: Broke
Type: RO, SC

When set, indicates that the node has not yet passed its self-test. The user interface must clear this bit when the node has passed its self-test. This bit must be set by the user interface by the time that BCI DC LO L from the BIIC is deasserted.

Bits: 11:0 Implementation dependent

digital™

7.17 RECEIVE CONSOLE DATA REGISTER



The Receive Console Data Register (RXCD), which is implemented by VAXBI nodes that have a console on the VAXBI, is used to receive data from other consoles. Nodes that do not implement a VAXBI console must respond to reads to the RXCD location with either a NO ACK response or return a longword of data in which the RXCD Busy 1 bit is set. In the latter case, the Busy 1 bit must be set before the Broke bit is cleared at the completion of self-test.

The RXCD Register is also used for exercising ROM-based diagnostics (this use is explained in Application Note 9).

A node that implements the RXCD Register responds to longword VAXBI transactions to the RXCD Register. A lock bit must be implemented for the RXCD if it is local to a node that may be a primary console node. (This requirement overrides the statement that implementation of a lock bit is implementation dependent when the operand is in I/O space outside of node window space). Nodes that will never be a primary console need not implement a lock bit for the RXCD. In a UWMCI command to an RXCD where the optional upper word is not implemented, the mask bits may be ignored; that is, the command may act as if the mask bits were all set, regardless of whether they are set.

Optional <31:16>

Bits: 31 Name: Busy 2
Type: R/W

When set, indicates that the CHAR2 field contains a character that has not yet been read by the remote node. The Busy 2 bit must be cleared before the CHAR2 field is available for another character.

Bits: 30:28 Name: RESERVED and zeros
Type: RO

digital™

Bits: 27:24 Name: Node ID 2
Type: R/W

Contains the node ID of the local node (the node that sent the character in the CHAR2 field).

Bits: 23:16 Name: Character 2 (CHAR2)
Type: R/W

Contains the console command character or console message being sent from the local node to the remote node.

REQUIRED <15:0>

Bit: 15 Name: Busy 1
Type: R/W

When set, indicates that the CHAR1 field contains a character that has not yet been read by the local node. The Busy 1 bit must be cleared before the CHAR1 field is available for another character.

Bits: 14:12 Name: RESERVED and zeros
Type: RO

Bits: 11:8 Name: Node ID 1
Type: R/W

Contains the node ID of the remote node (the node that sent the character in the CHAR1 field).

Bits: 7:0 Name: Character 1 (CHAR1)
Type: R/W

Contains the console command character or console message being sent from the remote node to the local node.

CHAPTER 8

CLASSES OF VAXBI NODES

We tend to consider nodes on a bus to be either processors, memories, or adapters. And we expect a certain functionality from each class. Sometimes, however, a node serves multiple functions. This chapter describes the expectations we have for the various classes of nodes and the requirements that a node must meet depending upon its function and the address space that it accesses.

Section 8.1 describes our expectations for each class of nodes, and Section 8.2 defines the requirements that VAXBI nodes of a particular class must meet. Section 8.3 discusses what is required of nodes if the VAXBI bus serves a specific function. In the example presented, the VAXBI bus performs as an I/O bus.

8.1 PROCESSORS, MEMORIES, AND ADAPTERS

Certain informal expectations have evolved of what processors, memories, and adapters do. A single VAXBI node may, in fact, perform a variety of functions.*

Nevertheless, it is useful to classify nodes by function. The names of the classes then are processor, memory, and adapter. The same hardware/firmware can function as a node of one class in a certain configuration and as a node of another class in another configuration.**

---

*For example, the Nautilus-Memory-Interconnect-to-VAXBI adapter (DB88) behaves as both a processor and a memory on the VAXBI, because it responds to both interrupts (a typical processor function) and to reads and writes in memory space (a typical memory function).

**For example, a KA820 processor in a single-processor VAX 8200 system clearly belongs to the processor class. But when a KA820 processor is on the VAXBI bus in a VAX 8800 system, that processor may act as an adapter or as adapter and processor.

digital™

This section describes our informal expectations of node classes in terms of the types of VAXBI transactions that a node of each class will generate or respond to. These "informal expectations" are not hard-and-fast rules. For example, we generally expect that adapters can access memory, but this does not mean that they have to. By describing our informal expectations, we hope to establish some basis for the formal requirements that nodes of each class must observe.

To simplify our discussion, we have excluded VAXBI transactions that are issued or responded to spontaneously by the BIIC (without any specific action by the node to which the BIIC belongs). These transactions include error interrupts generated by the BIIC on detection of bus errors, responses to the corresponding IDENT transactions, and responses to VAXBI transactions that access BIIC registers.

8.1.1 Processors

We expect a processor node to execute machine instructions, to access memory, and to control the action of adapters.

In carrying out these functions, a processor node accesses memory space locations in memory nodes and CSR (control and status register) locations in nodes of all types. (A CSR is an I/O space location, usually in nodespace but sometimes in a node's node window or assignable window, that is used to control or monitor the node.) A processor node issues IDENT transactions to adapters and memories, IPINTR transactions to adapters and to other processors, and STOP transactions to nodes of all types. A processor responds to INTRs and IPINTRs. It also responds to longword accesses to its RXCD Register, if it implements a VAXBI console. (See Section 9.2 for an explanation of the RXCD Register.)

What distinguishes "processors" from nodes of other classes is their ability to respond to INTR transactions and to issue IDENT and IPINTR transactions. Note that, according to this point of view, an array processor would not be a processor but an adapter. This anomaly results from our bus-oriented point of view.

digital™

8.1.2 Memories

A memory node stores instructions and data for processors and adapters.

In general, memories are only slaves on the VAXBI. A memory responds to all read- and write-type VAXBI transactions to memory space. A memory node that can be written to without use of the VAXBI may or may not issue INVAL commands; a memory node that cannot be written to except from a given VAXBI bus need never issue INVAL commands on that VAXBI bus.

What distinguishes a memory node from nodes of other classes is that it responds to memory space accesses.

8.1.3 Adapters

An adapter node transfers data to and from memory and accepts control from a processor.

An adapter generally does not access CSR locations of other adapters. However, because memory can be implemented in I/O space, adapters might perform DMA transfers to and from locations in I/O space that reside either within themselves or on other nodes. Adapters can issue INTRs to processors and respond to IPINTRs, STOPs, IDENTs, and accesses to their own CSRs.

8.2 REQUIREMENTS ON NODES OF EACH CLASS

Section 8.2.1 specifies what transactions nodes of each class must issue or respond to so that compatibility of VAXBI nodes is not compromised.

Section 8.2.2 then discusses those requirements by node class. Section 8.2.3 discusses VAXBI requirements that relate to I/O space.

8.2.1 Required Sets of Transactions

The capabilities required of nodes of each class can be described by examining how nodes participate in transactions. Two distinct sets of transactions are involved. One set consists of transactions that nodes of one class must be able to respond to (MRS). The other set consists of transactions that nodes of a given class must be able to issue (MIS).

Must Respond Set (MRS)

Suppose C1 and C2 are two arbitrary node classes. If a node of class C1 "may" issue transaction type TR to a node of class C2, then for the sake of compatibility all nodes of class C2 "must" respond to TR. (For example, an adapter can issue quadword transactions to memories; therefore, all memories must respond to quadword transactions.)

Consider all the types of transactions that nodes of class C2 must respond to. (In our example, for memories this would be transactions like the quadword transactions.) This set of transactions is the Must Respond Set (MRS) for class C2. Nodes of class C1 MUST NOT depend on nodes of class C2 to respond to any transactions outside of MRS. (In terms of the example, had quadword transactions not been in MRS, adapters could not depend on all memory nodes to respond to quadword transactions. In this case, an adapter that issues quadword transactions to memory nodes might be incompatible with some memory nodes.)

Must Issue Set (MIS)

Suppose a node of class C1 "may" depend on receiving transactions of type TR from nodes of class C2; that is, a function of some nodes of class C1 cannot be exercised without the node receiving TR-type transactions. Then all nodes of class C2 "must" be capable of issuing TR. (For example, adapters depend on processors to issue longword transactions to I/O space; therefore, all processors must be capable of issuing longword transactions to I/O space.)

The set of transactions that nodes of class C2 must be capable of issuing is the Must Issue Set (MIS) for class C2. Nodes of class C1 must not depend on receiving any transactions of any type outside of MIS. (For example, processors must not depend on receiving INTR transactions, since INTR is not in the MIS of any node class.)

The MRS and MIS for each of the three classes of nodes is shown in Figure 8-1.

digital™

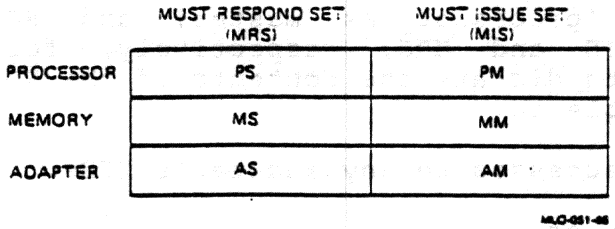


Figure 8-1: Required Sets of Transactions

Note that there is no simple relation between any MIS and any MRS. For instance, quadword transfers are in the MRS of memories, but they are not in the MIS of processors or adapters. On the other hand, IPINTR is in the MIS of processors but not in the MRS of memories or adapters.

8.2.2 Requirements by Node Class

The rationale for why certain transactions are required for processor and memory nodes is given below.

Processor Nodes

The required transactions for processor nodes are mainly the result of adapter design. Adapters communicate with processors and memories by means of memory accesses, processor accesses to adapter CSRs, and interrupts. To be compatible with future processor designs, an adapter must issue to processors only those transactions which all processors can respond to, and must depend on receiving only those transactions which all processors can generate.

Processors must also cooperate to implement "indivisible" actions. These indivisible actions are used to ensure the integrity of data structures that are updated by more than one VAXBI node, or by more than one process running on the same VAXBI node. The protocols that implement these actions must involve transactions that all processors can generate.

As defined in Section 8.2.1, let PM (processor as master) and PS (processor as slave) be the MIS and MRS, respectively, for processors.* The following requirements dictate the contents of the PM and PS subsets. Processors must be able to:

- Generate all the appropriate accesses to any adapter's CSRs.
- Field interrupts from the adapter.
- Generate IPINTR transactions to signal processors and adapters that depend on this capability.
- Generate the IRCI and UWMCI transactions needed to implement indivisible actions.

The PM subset consists of:

- All longword data transfer transactions to I/O space, except:
  (a) Only READ or RCI and only WRITE or WCI need be included. (The data transfer transactions are READ, RCI, IRCI, WRITE, WCI, WMCI, and UWMCI.)
  (b) Data transfer transactions to node private space are excluded from the PM subset.

  Word addressability is required for longword-length node window space data transfer transactions, so that word-accessible adapters will be compatible with the processor.
- IRCI and UWMCI transactions of any one or more lengths, to memory space. The IRCI and UWMCI transactions implemented can be of different lengths.
- The IDENT, IPINTR, and STOP transactions.

---

*For example, both the KA820 processor and the DB88 adapter must be able to issue the transactions in PM and respond to the transactions in PS.

The PS subset consists of:

- o INTR transaction
- o IPINTR transaction
- o STOP transaction*

Only those processors that must communicate with adapters or with other processors need to implement the PM and PS subsets. Processors that do not need to perform this communication are exempt from this requirement. Such processors may be considered adapters for the purposes of this chapter. For example, array processors that do not need to communicate with adapters or other processors are exempt.

Memory Nodes

There are no transaction types that every memory must be capable of issuing. Therefore, MM is an empty set.

All memory nodes must respond to the same set of VAXBI data transfer transactions when these transactions access memory space. This requirement allows software to handle different memory node designs in the same way except when initializing the system. It also allows adapters to access any memory location using any VAXBI transaction in this set. This set of transactions is the MRS for memory class nodes, which we will call MS (for memory as slave). The transactions may originate either from a processor or from an adapter.

The MS set includes:**

- o All data transfer transactions of any length, for the memory space that selects the node. (The data transfer transactions are READ, RCI, IRCI, WRITE, WCI, WMCI, and UWMCI. Longword, quadword, and octaword lengths are all included.)
- o STOP transaction

---

*The KA820, KA800, and KA88 processors have been granted exceptions and are not required to respond to STOP transactions.

**The DB88 adapter, for example, must respond to these transactions that are required for memory nodes. (An exception has been granted so that the DB88 and the KA800 need not respond to the STOP transaction.)

digital™

Adapter Nodes

There are no transaction types that every adapter must be capable of issuing. Therefore, the set AM is empty. All adapters must respond to STOP transactions, and this is the only transaction to which they must all respond. Therefore, the set AS contains just the STOP transaction.

In conclusion, the MS, PM, and PS sets constrain the design of memories and processors but also provide assurances. An adapter (or any other node) may depend on all memories to respond to any of the transactions in MS, and may depend on all processors to respond to those in PS and issue those in PM.

8.2.3 Requirements in I/O Space

8.2.3.1 Read Side Effects - Read-type transactions targeting I/O space locations must not have any side effects. An example of a read side effect is provided in the next paragraph.

Consider a case where on a read-type transaction the bus master obtains the data with bad parity, but the slave node does not detect bad parity. If this transaction has caused side effects, then it cannot be reissued without causing the side effects to recur. In this particular case, the master then cannot reissue the read-type transaction and cannot potentially recover from the original error.

This rule does not mean that the value read cannot change in the interim. If a transaction is reissued, the data obtained by the second read can be different from that obtained by the first read. What is required is that the data obtained by the second read is independent of whether or not the first read took place (that is, the read was not responsible for causing the data to change).

For example, if the data being read is a timer of some sort, the second read might be expected to produce a different value from the first. On the other hand, if the data is being read from a first-in/first-out queue, and the transaction causes the top entry of the queue to be "popped" and discarded, then the transaction has a side effect. Therefore, the queue design violates this rule. To conform to the rule, the top entry can be discarded only on some transaction other than a read-type transaction; for example, the top entry may be discarded on a write-type transaction.

8.2.3.2 Cacheing in I/O Space - Data fetched from I/O space must not be cached by a master.

8.2.3.3 Read-Type Value Equivalence - If an I/O space location J returns a read data value V to any of {IRCI, RCI, READ}, then location J must return value V to all of {IRCI, RCI, READ}. That is, the read value must be independent of which particular read-type transaction was issued.

8.2.3.4 Write-Type Reaction Equivalence - If an I/O space location L reacts to any of {UWMCI, WCI, WMCI, WRITE}, then location L must react in the same way to all of {UWMCI, WCI, WMCI, WRITE}. This means that any actions and state changes triggered at the slave by the write-type transaction must be independent of which write-type transaction was issued. There are two exceptions:

- o UWMCI may clear a lock bit associated with location L at the slave that must not be cleared by any of {WCI, WMCI, WRITE}.
- o The reaction of location L to one of {UWMCI, WMCI} may be a function of the received mask bits during D cycles of the transaction. The reaction of location L to one of {WCI, WRITE} must not be a function of the received mask bits during D cycles of the transaction.

8.2.3.5 Longword Data Length Transactions - Each I/O address location that responds to read-type commands must respond to longword-length read-type commands. Each I/O address location that responds to write-type commands must respond to longword-length write-type commands. In node window space, nodes that respond to longword-length transactions may perform only a byte or word transfer naturally aligned within the longword.

8.2.3.6 Quadword and Octaword Data Length Transactions - Throughout I/O space, response to transactions with data lengths greater than longword is implementation dependent. Nodes must respond with NO ACK for data lengths that are not implemented.

8.2.3.7 Locks in I/O Space - For rules regarding locks in I/O space, see Section 5.2.2.

8.2.3.8 Write Masks in I/O Space - The interpretation of the write mask in I/O space is implementation dependent. However, certain rules apply for specific I/O space locations. These include:

- o Registers in BIIC CSR space must interpret write masks in accordance with the semantics of the write-type transaction.
- o The RXCD Register must interpret write masks in accordance with the semantics of the write-type transaction, except

that, if the optional upper word is not implemented, the mask bits of a UWMCI may be ignored.

8.2.3.9 Translations of VAXBI Transactions to and from the UNIBUS - In UNIBUS window space, an IRCI directed to the adapter from the VAXBI will be interpreted as a DATIP to the UNIBUS. A UNIBUS DATIP must be translated as an IRCI to the VAXBI bus.

8.2.3.10 Rationale for I/O Space Requirements - The I/O space requirements cited above guarantee certain desirable results. For example:

- The response equivalence of RCI to READ and WCI to WRITE ensures that processors have the flexibility to issue cache-intent commands regardless of whether the target address is in memory space or I/O space.

- The response equivalence of READ and RCI to IRCI, and of WRITE, WCI, and WMCI to UWMCI ensures that noninterlocked VAX instructions that generate interlocked VAXBI transactions will not fail, and that registers normally read and written using interlocked transactions can also be read and written using noninterlocked transactions.

8.3 THE VAXBI AS AN I/O BUS

Not all of the VAXBI transactions are required in certain configurations. In a high-performance system, the VAXBI may serve as an I/O bus and occasionally as an interprocessor bus, without serving as a memory bus. It is useful to examine this case as an example of how the principles described above apply in a specific situation.

Figure 8-2 shows a configuration in which a memory bus (MB) connects processor and memory and in turn is connected to two VAXBIs by memory bus adapters (MBAs). Each VAXBI has several adapter nodes that interface to I/O devices.

The MBA acts as a processor node, because it generates I/O space accesses but not memory space accesses on the VAXBI (except for diagnostic purposes). The MBA also acts as a memory node, because VAXBI adapters access memory through the MBA.*

---

*The VAX 8800 system is a case in point.

digital™

Because the MBA acts as both processor and memory, it must implement the MS, PM, and PS subsets of VAXBI transactions:

- It must respond to all lengths of VAXBI data transfer transactions to memory space.
- It must generate these commands of longword length:
  - Either READ or RCI
  - Either WRITE or WCI
  - IRCI, WMCI, and UWMCI
- It must respond to INTR, IPINTR, and STOP transactions and generate IDENT, IPINTR, and STOP transactions.

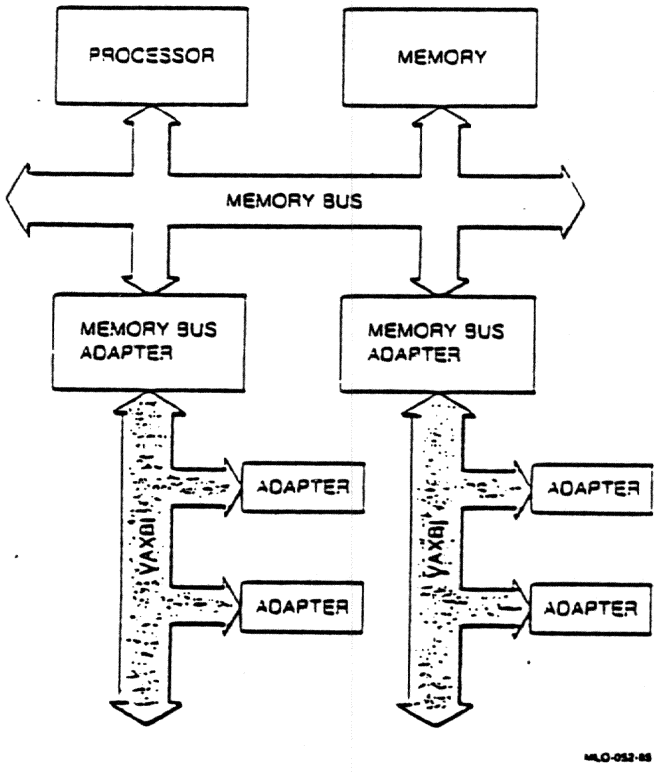


Figure 8-2: The VAXBI as an I/O Bus

digital™

CHAPTER 9

VAXBI CONSOLE PROTOCOL

A VAXBI console is a console at a node (or remotely connected to the VAXBI bus by an adapter) that supports the control of processors on a VAXBI system. In some VAXBI systems, the source of console commands may be a console terminal attached to the node; in others, it might be a program running on a system connected to the node over a local area network. The VAX-11 Architecture Reference Manual specifies the characteristics of VAX consoles. This chapter describes the characteristics of VAXBI consoles -- VAX consoles that are also VAXBI nodes.

A single VAXBI bus can support multiple processor nodes. The VAXBI console protocol allows a single console to control all the processors. The VAXBI console issuing the console commands is considered the master console.

Console communication between VAXBI nodes consists of console commands and console messages. Typically, console commands are typed, and console messages are displayed at a console terminal. These communications are carried out with VAXBI read- and write-type transactions to VAXBI nodespace addresses. This chapter describes the protocol for such communication: the addresses written to, the data written, and the responses to these VAXBI transactions. Also described is the Z console command, which has been designed to meet the needs of VAXBI consoles.

9.1 MASTER CONSOLE

On the VAXBI bus only one node, the master console, is allowed to issue console commands at any one time. Other VAXBI consoles can only send messages to the master console; they cannot communicate with each other. Different nodes can be master console at different times. In the configuration shown in Figure 9-1, processor A can be master console at one time and processor B at other times, but they cannot both be master console at the same time. While processor A is master console, console commands can only be issued from the console terminal

digital™

attached to it. The method of determining which node is master console during the power-up sequence and at other times is implementation dependent.

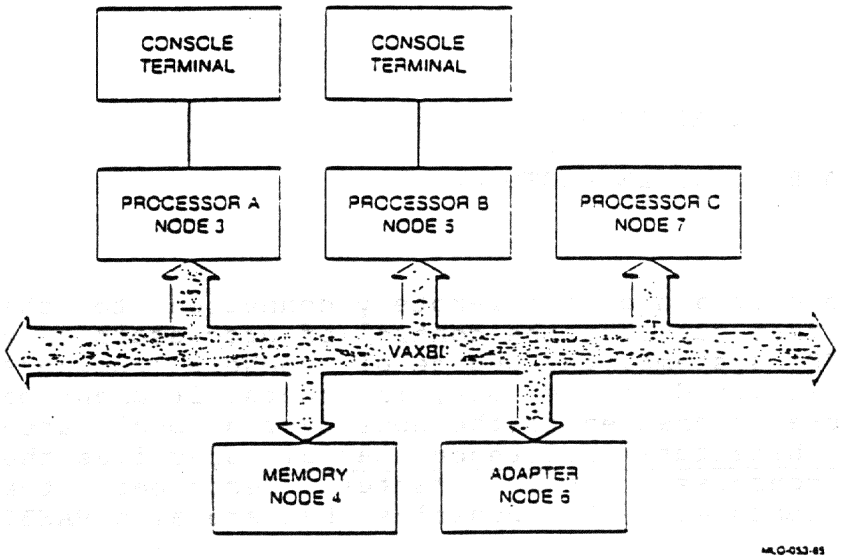


Figure 9-1: Console Configuration

9.2 RECEIVE CONSOLE DATA REGISTER (RXCD)

Each VAXBI console has a nodespace register, the Receive Console Data Register (RXCD), for receiving data from other VAXBI consoles. The RXCD Register occupies the address bb + 200 in the nodespace of the node (bb is the starting address for the node's nodespace). The RXCD Register address is reserved for the RXCD. If a VAXBI node does not implement a VAXBI console, then that node must respond to reads to that location with either a NO ACK confirmation or a longword in which the RXCD Busy 1 bit is set (described below).

The RXCD Register will respond to longword VAXBI transactions. Since VAXBI interlock commands are used in the protocol, locks must be implemented for the RXCD Register. This is true notwithstanding the VAXBI rule that interlock commands are implementation dependent with respect to interlocking when the operand is in I/O space outside of node window space. However, in a UWMCI command to the RXCD the mask bits may be ignored; that is, the command may act as if the mask bits were all set, regardless of whether they are set. VAXBI nodes that do not implement a VAXBI console need not implement locks for the RXCD Register location.

The RXCD Register is described in Section 7.17.

9.3 CONSOLE COMMUNICATIONS

To send a character in a console communication, the sending console carries out the following protocol:

1. Issue a VAXBI IRCI to the receiving console's RXCD Register. Examine the Busy 1 bit (bit 15) of the returned value. If it is one, go to step 2; otherwise go to step 3.

2. The Busy 1 bit is a one, so the receiving node is not ready to receive. Write back what was read with a VAXBI UWMCI transaction to unlock the RXCD. After a short wait (the length of which is implementation dependent), start again at step 1. If the Busy 1 bit remains set for over 1 second, the sending console can WRITE a longword of all zeros into the receiving console's RXCD Register to clear the Busy 1 bit, if the following conditions are met:

   - The sending console is the master console.
   - The sending console is transmitting on behalf of input from the console terminal; that is, the console terminal is in console mode.

   After attempting to clear the RXCD Register, the sending console can repeat step 1 above. Note that, since the sending console is the master console in this case, no other console can be writing to the receiving console, and the RXCD contents that were erased must have been deposited by the sending console itself.

3. The Busy 1 bit is a zero, so the receiving node is ready. Issue a VAXBI UWMCI with the mask bits set to 1111, conveying one character to the receiving node. The issuing node must load the Node ID field with its node ID and set the Busy 1 bit.

Each node monitors its RXCD Register. The Busy 1 bit, which is initially zero, is set to one when the RXCD is written by another console. The local node then issues an IRCI transaction to read the character, followed by a UWMCI to clear the Busy 1 bit and unlock the RXCD.* The protocol for the RXCD Register is described in pseudocode in Figure 9-2.

The receiving node should sort incoming characters according to sending node, using the node ID in the Node ID field of the RXCD. In this way, if two nodes simultaneously send characters to the same node, the two messages will not be interleaved.

---

*In the KA820 case, if the processor is not halted, an interrupt is generated when the RXCD receives the character.

digital™

Send to Remote VAXBI Console

```
! byte_to_send()                    Character of command/data to be sent
! more_to_send                      There are more characters to send
! new                               Longword, set up ready to transmit
! old                               Longword, read from destination RXCD
! my_id                            4-bit encoded node ID of sending node
! dest_RXCD                        Address of RXCD of destination node

while more_to_send begin
    new<15> := 1;
    new<11:8> := my_id;
    new<7:0> := byte_to_send();
    locked := true;
    while (locked = true) begin
        issue (IRCI, dest_RXCD, old);
        if (old<15> = 0) then
            begin
                issue (UWMCI, dest_RXCD, new);
                locked := false;
            end
        else
            issue (UWMCI, dest_RXCD, old);
        end
end
```

Receive from Remote VAXBI Console

```
! newchar()                         RXCD Busy 1 bit became set
! issue (trans, addr, data)        Initiate transaction to address
! my_RXCD                          Address of this node's RXCD Register
! process                          Start processing new character
! newdata                          Longword for holding RXCD contents

while newchar() begin
    issue (IRCI, my_RXCD, newdata);
    issue (UWMCI, my_RXCD, 0);
    process(newdata);
end
```

Figure 9-2: RXCD Protocol

9.4 THE Z CONSOLE COMMAND

All VAXBI consoles must implement the Z console command. The Z command causes console commands received or typed at one console to be forwarded to another console. The format and effect of the command are as follows:

Z <value>

The <value> is a hexadecimal digit indicating the destination node. Subsequent characters input at this console are forwarded to the destination console, except as described below. The destination node echoes back to this console.

The first ASCII escape character indicates that a "literal" follows. That is, any character immediately following the first escape character is to be forwarded, including CTRL/P or another escape character. Since the character immediately after the first escape is forwarded as a literal, an escape can also be forwarded in this manner.

Unless it is a literal, a CTRL/P is not forwarded. Instead, it terminates the Z command (that is, it terminates the forwarding) and causes the local console to enter console mode.

Figure 9-3 shows how the Z console command handles escape characters.

Two Z commands must not be issued from a processor when it is master console, without an intervening CTRL/P. For instance, consider the configuration shown in Figure 9-1. Suppose "Z 5" and "Z 7" are issued from processor A as master console, without an intervening CTRL/P. The first Z command causes subsequent commands to be forwarded to node 5, while the second Z command might be expected to cause subsequent commands to be forwarded first to node 5 (processor B) and then to node 7 (processor C). The effect of subsequent console commands issued from processor A is undefined in this case.

digital™

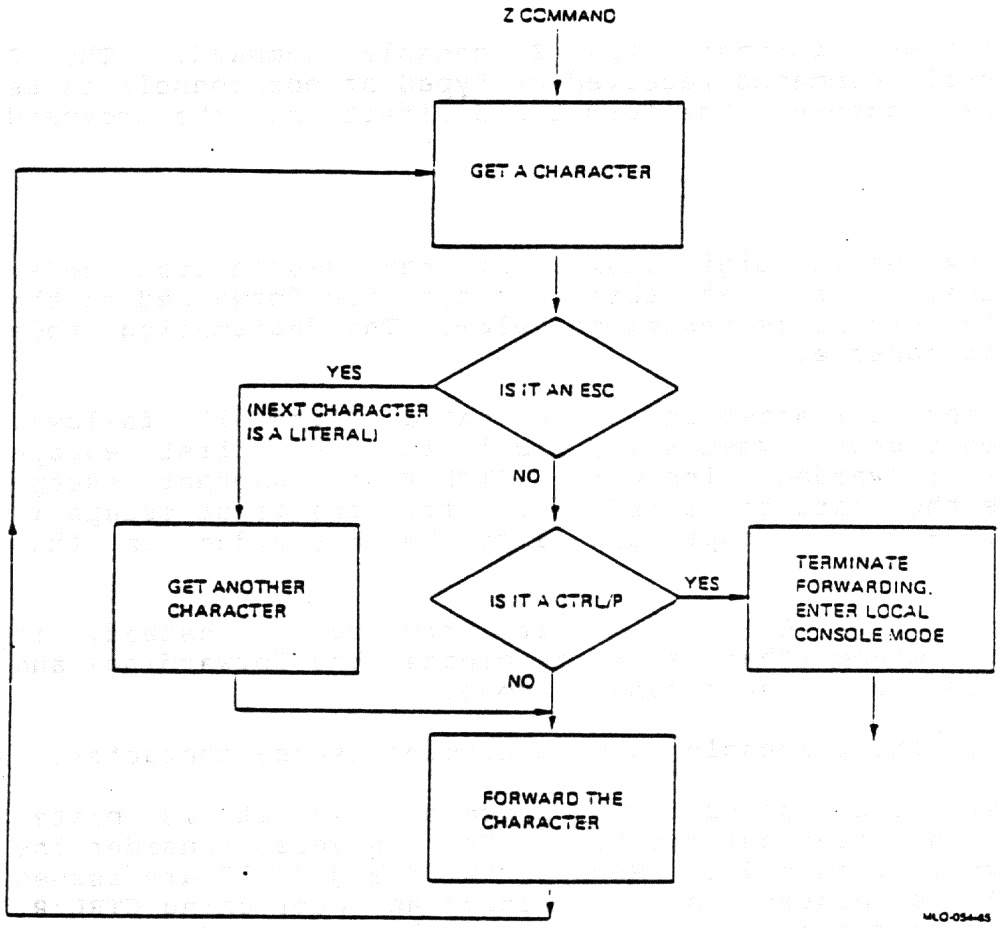


Figure 9-3: Handling of Escape Characters in the Z Console Command

CHAPTER 10

PERFORMANCE

This chapter discusses bus bandwidth and bus access latency and interrupt latency on the VAXBI bus. These measures in general cannot be calculated because they are dependent largely on factors such as memory performance characteristics, which are not characteristics of the bus. However, if the specified restrictions on the number of certain types of cycles are adhered to (or if violations of the restrictions are documented and it is known how many of which violating nodes are in the configuration), an upper bound can be determined from clock frequency and protocol limits.

10.1 VAXBI BANDWIDTH

Bandwidth, the measure of data throughput on the VAXBI bus, is directly related to the length of data transferred. That is, the longer the length of data, the higher the bandwidth. The bandwidth increases as the bus overhead cycles take a smaller proportion of the total transaction time.

Assuming that all transactions are of the same length, the maximum bandwidth that can be achieved on the VAXBI bus for each of the different transaction lengths is as follows:

For octaword transactions (16 bytes): 13.3 megabytes per second
For quadword transactions (8 bytes): 10.0 megabytes per second
For longword transactions (4 bytes): 6.7 megabytes per second

Figure 10-1 shows the VAXBI bandwidths for transactions of each length with no STALL cycles versus one STALL cycle per transaction. The figure also shows the bandwidths for longword transactions that transfer a single word or byte. In each case, all transactions are assumed to be of the same length.

digital™

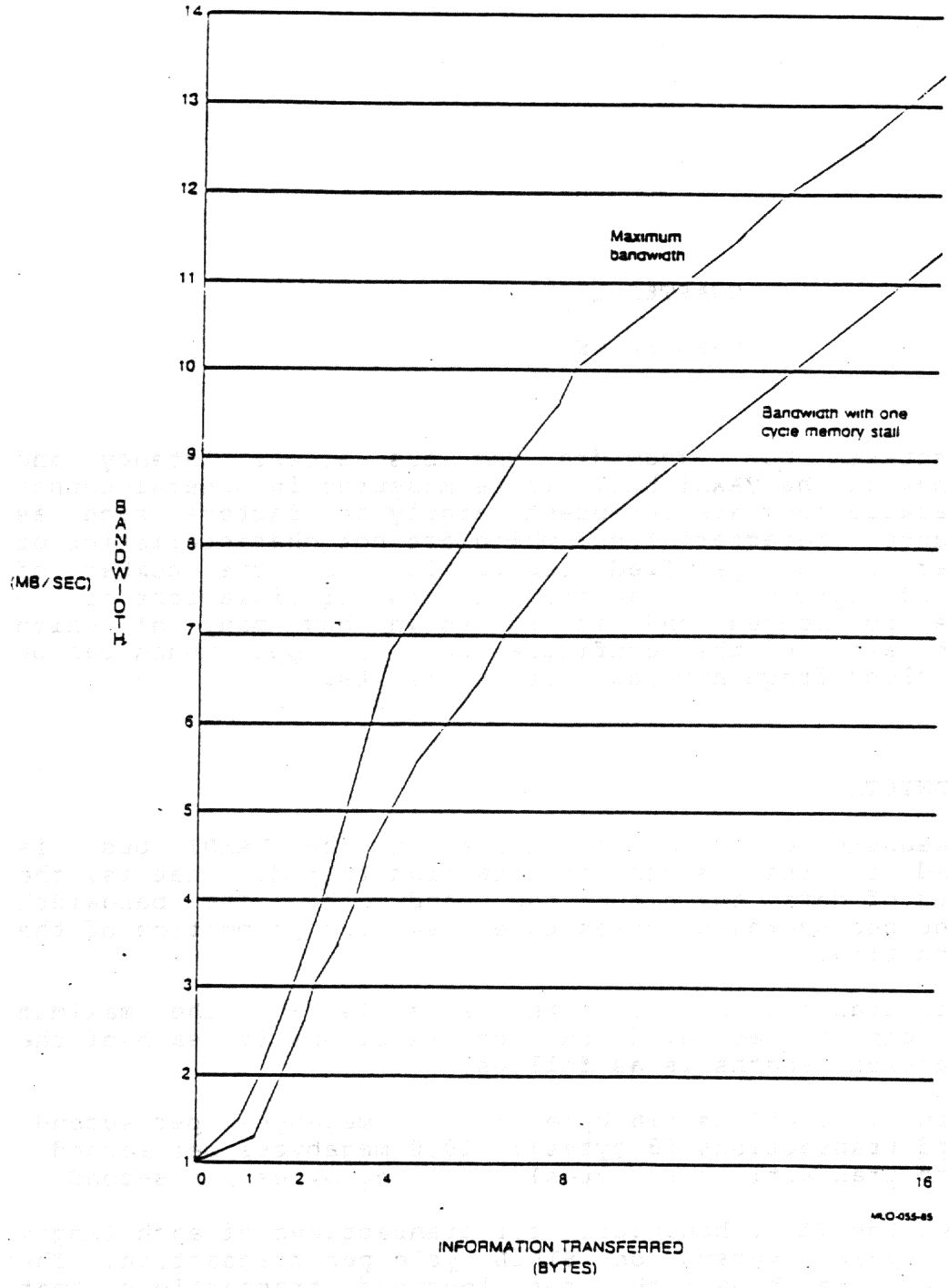


Figure 10-1: Bandwidth Ranges

digital™

10.2 LATENCY

Latency properties of the bus may be largely inferred from bus access latency and interrupt latency. Bus access latency is the delay from the time a node desires to assert its arbitration request on the bus until it becomes bus master. Interrupt latency is the delay from the time a node desires to transmit an interrupt transaction until the start of execution of the node's interrupt service routine.

The minimum bus access latency is one clock cycle or 200 nanoseconds nominal. In many systems this will be the typical latency. To obtain an upper bound on bus access latency, we need to know the longest possible duration of a VAXBI transaction. Bus access latency depends on the arbitration mode of this and other nodes and on the length of VAXBI transactions. The effect of the arbitration mode is discussed in the following sections.

Interrupt latency includes the following components:

- Bus access latency for sending the INTR transaction
- INTR bus transaction time
- Priority level of the interrupt, the number of other interrupts that must be serviced first and the interrupt service time for these interrupts, and other processing that must be performed first
- Bus access latency to send an IDENT transaction
- IDENT bus transaction time
- Context switching time of the processor

All VAXBI nodes should be designed to tolerate long latencies. A worst-case latency time cannot be defined, and any node that makes assumptions about the worst-case latency cannot be guaranteed to work in all configurations.

10.2.1 Extension Cycles

In addition to the command/address, imbedded arbitration, and data cycles, a VAXBI transaction can have additional cycles due to the following:

- STALL data cycles. A slave may extend a transaction by asserting the STALL code on its BCI RS lines. In response, the BIIC asserts the STALL code on the BI CNF lines.

- Busy extension cycles. A node not involved in a VAXBI transaction may assert BI BSY L to extend the transaction.

- Loopback cycles. A node not involved in a VAXBI transaction may access registers in its own nodespace with a loopback request. When such a request is made during a transaction, the BIIC extends the transaction by asserting BI BSY L at the end of the transaction.

Busy extension cycles and loopback cycles increase bus access latency. Suppose, for example, a node extends a VAXBI transaction by asserting a loopback request. During one of the extension cycles, another node could extend the transaction by another loopback or busy extension, and the new extension could again be extended.

The following rules limit extension cycles to control bus access latency:

- Slave nodes must not issue more than eight STALLs in data transfer transactions. If a node could exceed the limit, the extent to which it may exceed the limit must be documented. An exception must be obtained if the node is Digital-supplied.

- Nodes must not use more than 16 consecutive extension cycles without issuing a VAXBI transaction request.

10.2.2 Effect of Arbitration Modes on Bus Latency

The VAXBI protocol specifies three modes for arbitration: dual round-robin, fixed-low priority, and fixed-high priority.

If fixed-low priority and fixed-high priority modes are used, some nodes may encounter long waiting times to use the VAXBI or may even be shut out of the VAXBI bus. So that all nodes can gain access to the VAXBI bus and so that responsiveness is not affected, nodes should not depend on arbitrating using any arbitration mode other than the dual round-robin mode. The fixed-low priority and fixed-high priority modes, intended only as a last resort for special real-time requirements, should be used only after careful analysis of the set of specific node ID assignments and VAXBI utilization patterns for the particular configuration.*

---

*Node designers should assume that in almost all cases nodes will arbitrate in dual round-robin mode. If it is expected that this will not be the case, the designer should reconsider whether the node should interface to the VAXBI bus.

digital™

To arbitrate, a node requests the use of the VAXBI bus by asserting one of the data lines corresponding to its node ID. With 16 node IDs and 32 data bits, two bits correspond to each node ID: one in the high-priority word (BI lines D<15:0>) and one in the low-priority word (BI lines D<31:16>) (see Chapter 3, Figure 3-1). The node's arbitration mode determines which of these two bits is asserted. In fixed-low priority mode, the node asserts the bit in the low-priority word. In fixed-high priority mode, it asserts the bit in the high-priority word. In dual round-robin mode, the node asserts the bit in the high-priority word if and only if its node ID is greater than the node ID of the previous master. Otherwise, it asserts the bit in the low-priority word.

Figure 10-2 shows the arbitration algorithm as implemented at each VAXBI node. When all nodes arbitrate in dual round-robin mode, the nodes may not attain bus mastership in strict round-robin order. However, the important advantages of round-robin are preserved: no node is ever locked out of accessing the bus, bus mastership is awarded "fairly" to all nodes, and the maximum wait for bus mastership is quite low.

The node asserting the lowest numbered data line wins the arbitration. When two nodes arbitrate in the same word (that is, assert bits in the same word of the longword), the one with the numerically smaller node ID wins the arbitration. Because it is confusing to say that the node with the lower number ID has higher priority, we will refer to nodes with "smaller" or "larger" IDs.

digital™

```
(* Signals to be driven are assigned during the "previous" cycle. *)

! my_id:                        This node's encoded ID
! bus_request:                  This node requests/keeps mastership
! i_am_master:                  This node is current master
! i_am_pend_master:             This node is pending master
! arb_mode:                     Arbitration mode
! next_cycle (cycltype):        True:  next cycle is of "cycltype" type
! this_cycle (cycltype):        True:  this cycle is of "cycltype" type
! highest_priority ():          Lowest number D line asserted, encoded
! assert_bit (bitposition):     Next cycle, assert D<bitposition>

i_am_master := false;
i_am_pend_master := false;
while true do
begin
    if this_cycle(imbedded_arb) then prev_master := I<3:0>_H;
    if (bus_request and
        (next_cycle(arb) or next_cycle(imbedded_arb)) and
        (not i_am_master))
        then begin
            if (arb_mode = fixed_high) then assert_bit(my_id);
            if (arb_mode = fixed_low)  then assert_bit(my_id + 16);
            if (arb_mode = dual_round_robin) then
                if (prev_master < my_id)
                    then assert_bit(my_id)
                    else assert_bit(my_id + 16)
        end;
    if (bus_request and (this_cycle(arb) or this_cycle(imbedded_arb))
        and (highest_priority() = my_id)) then
        i_am_pend_master := true;
    if ((next_cycle(cmdaddr) and i_am_pend_master) then
        begin
            i_am_master := true;
            i_am_pend_master := false;
        end
    if (next_cycle(imbedded_arb) and i_am_master) then
        I<3:0>_H := my_id;
    if (not bus_request) then i_am_master := false;
    advance_to_next_cycle();
end.
```

Figure 10-2: Arbitration Algorithm

digital™

10.2.3 Dual Round-Robin Mode Behavior

Even when all nodes arbitrate in dual round-robin mode, bus mastership can be allocated in other than strict round-robin order. Figure 10-3 gives an example of how this can happen. Assume all transactions are, say, longword writes. Notice that all nodes arbitrate in the high-priority word. The decimal numbers indicate node IDs, and each transaction is separated by a line.

Arbitrating Nodes	Cycle Type	Pending Master	Bus Master
1	ARB	None	None
7	C/A	None	1
	Imbedded ARB	None	1
	Data	7	1
5, 12	C/A	None	7
	Imbedded ARB	None	7
	Data	5	7
10, 12	C/A	None	5
	Imbedded ARB	None	5
	Data	10	5
8, 12	C/A	None	10
	Imbedded ARB	None	10
	Data	8	10
12	C/A	None	8
	Imbedded ARB	None	8
	Data	12	8
None	C/A	None	12
	Imbedded ARB	None	12
	Data	None	12

Figure 10-3: Example of Dual Round-Robin Mode Behavior

In this example, node 8 gets to be bus master after node 10 but before node 12, which violates strict round-robin. In effect, two round-robins are operating on alternate transactions. In one of them, the sequence of bus masters was nodes 1, 5, and 8; in the other, the sequence was nodes 7, 10, and 12. Each node arbitrates in both round-robins until it obtains bus mastership. The first round-robin is in effect in imbedded arbitration cycles when the bus master belongs to the second round-robin. Thus, when node 8 started

digital™

arbitrating, node 10 was master, and the first round-robin was in effect. Since in the first round-robin the previous bus master was node 5, node 8 wins the arbitration over node 12.

This dual round-robin behavior is produced because the criterion used in determining whether nodes arbitrate in the high- or low-priority word is the node ID of the last previous bus master, not of the current bus master. If the criterion used was the current bus master, the result would be true round-robin behavior. Since the dual round-robin behavior depends on the use of the imbedded arbitration cycle, this behavior is not apparent unless traffic is heavy enough to make significant use of the imbedded arbitration cycle.

The dual round-robin mode preserves all the desirable properties of the round-robin. The two round-robins operate as true round-robins, and no node ID is favored over any other node ID. In both cases, the worst-case latency for bus mastership (that is, the longest wait to become bus master) is finite, so no node can be locked out of the bus.

In fact, the worst-case latency is the same in both cases: the longest wait arises when all nodes (in turn) arbitrate for the bus, and the node in question has just missed its turn for the bus (that is, in the dual round-robin case, the node starts arbitrating when the node with the next higher ID is previous bus master).

It is interesting to note that if there is quite a lot of bus traffic, then the winning node in an arbitration is most often a node that is arbitrating in the high-priority word. The following scenario illustrates this.

Suppose that several nodes are arbitrating, initially all in the high-priority word. As nodes with smaller IDs win the bus, their next request for the bus causes them to arbitrate in the low-priority word. The next bus master has a larger (lower priority) node ID. As the bus master's node ID gets larger, more nodes arbitrate in the low-priority word, and fewer arbitrate in the high-priority word. All this time, however, the winning node is one that arbitrates in the high-priority word. This pattern continues until finally no node arbitrates in the high-priority word. All nodes now arbitrate in the low-priority word, and the node with the smallest ID wins the arbitration. This is the one time that the winner arbitrates in the low-priority word. At the next arbitration, all nodes that arbitrated but did not win this time arbitrate again in the high-priority word. The winning node is again from the high-priority word, and the pattern repeats.

10.2.4 Dual Round-Robin Mode Latency

For a VAXBI system in which all nodes arbitrate in dual round-robin mode, the maximum waiting time occurs when a node, say node A, requests the bus and must wait until all the other nodes use the bus before it gets its turn.

Suppose the VAXBI bus has k nodes, of which t nodes are transaction-generating nodes. Since node A is waiting for bus mastership, (t - 1) transactions pass before node A gains mastership.

The master and slave nodes of each transaction carry out an octaword transaction for (Si + 6) cycles, where Si is the limit on stalled cycles for node i. This accounts for (St + 6(t - 1)) cycles, where St is the largest sum of the Si for (t - 1) slave nodes (that is, pick the set of (t - 1) slave nodes that yields the largest sum).

During the first transaction, the master node, the slave node, and node A cannot create extension cycles. However, the slave node may be node A. Therefore, during this transaction there may be up to (k - 2) nodes creating extension cycles. These extension cycles will be denoted by Xk.

During the second transaction, again the current master node and node A cannot create any extension cycles. Except for the master of the last transaction, other nodes also cannot create extension cycles because they have reached their limit of extension cycles. Therefore, during this transaction, only the last transaction's bus master can create extension cycles, and it can do so to its limit.

In all succeeding transactions, the previous master is the only node that can create extension cycles. The extension cycles in these transactions are denoted by Xi.

The maximum total contribution of extension cycles is therefore (Xk + Xt), where Xk is the sum of the Xi for all nodes except node A, and Xt is the largest sum of the Xi for (t - 3) nodes. (The (t - 3) factor arises because there are (t - 1) transactions, and the contribution of the first two of these constitutes the Xk term, leaving (t - 3) transactions.) Since node A should be chosen to yield the maximum Xk, Xk is the largest sum of the Xi for (k - 1) nodes.

digital™

In summary, then:

Si is the limit on stalled cycles for node i
Xi is the limit on consecutive extension cycles for node i
k is the number of nodes
t is the number of transaction-generating nodes
St is the maximum sum of the Si for (t - 1) nodes
Xk is the maximum sum of the Xi for (k - 1) nodes
Xt is the maximum sum of the Xi for (t - 3) nodes

The following equation gives the upper bound Tm on the bus mastership latency.

Tm = (St + Xk + Xt + 6(k - 1)) cycles

For example:

For t = k = 6, Si = 8 for all i, and Xi = 32 for all i

St is 40 cycles (8 microseconds)
Xk is 160 cycles (32 microseconds)
Xt is 96 cycles (19.2 microseconds)

Therefore, Tm is 326 cycles (65.2 microseconds).

On the other hand, for t = 3, k = 5, Si = 8, and Xi = 16

St is 16 cycles (3.2 microseconds)
Xk is 64 cycles (12.8 microseconds)
Xt is 32 cycles (6.4 microseconds)

Therefore, Tm is 124 cycles (24.8 microseconds).

Note that t is not necessarily the total number of nodes. For example, a node that is purely memory probably will not generate any transactions, and so it does not count as a transaction-generating node. However, in this analysis we assumed that any node, other than the nodes involved in an ongoing transaction, could create extension cycles.

10.2.5 Fixed-Low Priority and Dual Round-Robin

The following gives an upper bound on the latency time if one node, node A, arbitrates in fixed-low priority mode, and all the other nodes use dual round-robin.

Suppose node A also has the smallest node ID (that is, the highest priority). Then it arbitrates in the same way as dual round-robin, and there is no effect on the maximum waiting time. But then there is

no reason to use fixed-low priority rather than dual round-robin. Suppose then that node A does not have the smallest node ID. If node A arbitrates when no other node is arbitrating, it of course wins the arbitration. Otherwise, it wins the arbitration only if all other nodes arbitrate in the low-priority word and these nodes all have larger node IDs. This situation happens only if the last bus master had a larger node ID and all currently arbitrating nodes have node IDs between the last bus master's and this node's.

Waiting times will therefore be comparable to dual round-robin for all but node A, while node A may have waiting times ranging from just like dual round-robin (in the case where it has a small node ID) to extremely long (in the case where it has a large node ID and the VAXBI bus is heavily used).

If there is a good chance that no other node is arbitrating when node A is arbitrating, then it does not matter what mode node A arbitrates in, and it might as well use dual round-robin. Otherwise, node A stands a reasonable chance of winning an arbitration only if it has a small node ID compared to other nodes.

It is difficult to see the utility of using the fixed-low priority mode, except in one notable case: If a node wants to win an arbitration only when no other node is arbitrating, it should have the largest node ID and use fixed-low priority.

10.2.6 Fixed-High Priority and Dual Round-Robin

The following considers the latency time if a node arbitrates in fixed-high priority mode and all other nodes use dual round-robin.

- If node A has the largest node ID, it behaves as if it were arbitrating in dual round-robin mode. The situation is then no different from the pure dual round-robin mode.
- If node A has the smallest node ID, then whenever it arbitrates it will cause all other nodes to "arbitrate in the high-priority word," so that, if it arbitrates very often, the effect is the same as a fixed priority scheme, and nodes with large node IDs may wait for a long time.
- If node A has neither the largest nor the smallest node ID, an interesting situation arises, which is discussed below.

Suppose node A has neither the largest nor the smallest node ID. Consider the situation where five nodes are using the bus heavily, where the five nodes are B, C, A, D, and E, in order of increasing node ID. Node A is arbitrating in fixed-high priority mode, and all others are arbitrating in dual round-robin mode. Suppose that

initially they all arbitrate in the high-priority word. Nodes B and C will win, followed by nodes A and D. Suppose nodes A, B, and C again request the bus before the imbedded arbitration cycle of D. Node A will win over nodes B and C, which are arbitrating in the low-priority word, because node A always arbitrates in the high-priority word. If before node A's imbedded arbitration cycle, node E should request the VAXBI bus, it will arbitrate in the high-priority word and win over nodes B and C. If now node D arbitrates before node E's imbedded arbitration cycle, node D will arbitrate in the high-priority word because the previous bus master was node A; node D will then win over nodes B and C. If node A next arbitrates before node D's imbedded arbitration cycle, it will again win over nodes B and C. This pattern can repeat in an A-E-D-A-E-D sequence locking out nodes B and C. In short, if node A and nodes with larger node IDs arbitrate often enough, in some order such as the A-E-D sequence, they can shut out nodes such as B and C that have smaller node IDs.

In general, the following statements can be made regarding the case where only a single node (say node A) arbitrates in fixed-high priority mode and other nodes arbitrate in dual round-robin mode:

- Whenever node A wins an arbitration, it starts something like a fixed-priority queue with itself at the highest priority. The effect would be exactly like a fixed-priority queue if each node decides which word to arbitrate in depending on the previous or current bus master's node ID, rather than just the previous bus master's node ID.

- The larger node A's ID, the less often will it win an arbitration, and the more will the effect be like dual round-robin.

- The smaller node A's ID, the more often will it win an arbitration, and the more will the effect be like fixed priority such as on the UNIBUS, with node A as the highest priority node.

Given the fixed-priority effect of arbitrating in fixed-high priority mode, this mode must be used with great care. For if the node using this mode is given a large node ID, the gain for the node will be minimal. On the other hand, giving the node a small node ID may cause nodes with large node IDs to get a much smaller share of the bus than is desirable. In particular, the following situation should be noted:

- Suppose that the transfer rate of an unbuffered disk, transferring through node A, is such that dual round-robin does not ensure fast enough response time to always keep up with the disk. The temptation would be to use fixed-high priority for node A.

- If dual round-robin does not ensure sufficiently fast response time, then node A arbitrates often enough that the total effect will be close to fixed priority for all nodes.

- Some other node having a large node ID (which may, say, have an interrupt to raise) may then be denied access to the VAXBI bus for long periods, even though that node and all nodes but node A arbitrate in the dual round-robin mode.

If more than one node arbitrates in the fixed-high priority mode, the situation is more complicated, but the effect would be similar to the case with just one such node. The fixed-high priority nodes would have a tendency to restart the dual round-robin at their node IDs, producing a fixed-priority effect.

10.2.7 One Fixed-High Priority Node

If only one node is arbitrating in the fixed-high priority mode (the other nodes arbitrating using dual round-robin), and if that node will not arbitrate more often than a certain limiting frequency, it is still possible to calculate a maximum waiting time for all nodes. The rule suggested here relates the maximum waiting time to the frequency with which the fixed-high priority node arbitrates.

Suppose that node A, which is arbitrating in fixed-high priority mode, has just used the bus. When node B arbitrates for the bus, the waiting time is lengthened because node A used the bus. What is the upper bound on this waiting time?

The upper bound can be produced with the following scenario: node B has to wait for all the other nodes, except node A, to use the bus. When its turn finally arrives, node B doesn't get it because node A arbitrates and wins. Due to node A's winning, all the other nodes again arbitrate in the high-priority word, and node B has to wait for all of them again.

Retaining the notation of the previous subsection on dual round-robin mode latency, the first step of the above scenario may take up to

(St + Xk + Xt + 6 (k - 1)) cycles

while the rest may take up to

(St + Xt + 6 (k - 1)) cycles,

for a total of

(2St + Xk + 2Xt + 12 (k - 1)) cycles.

digital™

Let T be this maximum waiting time. Now, if node A does not arbitrate more than once in any span of time T, then our assumption is satisfied: in all this time node A uses the bus at most once.

In summary, then:

> If just one node arbitrates in fixed-high priority mode, and it does not arbitrate more than once in any interval of time T, and no more than R extension cycles occur in the same interval T, where
>
> $$T = (2St + 2Xt + Xk + 12 (k - 1)) \text{ cycles}$$
>
> then the maximum waiting time for any node is also T cycles.

For the two examples in the subsection on dual round-robin mode latency, T would be 556 cycles (111.2 microseconds) and 216 cycles (43.2 microseconds), respectively.

The gain by having node A arbitrate in fixed-high priority mode is a shorter maximum waiting time for node A. The longest waiting time for node A depends on the number of nodes with node IDs smaller than node A's. If among these t of them are transaction-generating nodes, the maximum waiting time would be:

$$St + Xk + Xt + 6 * (t - 1)) \text{ cycles}$$

Although this looks very much like the latency in the dual round-robin mode latency subsection, this t is different since only nodes with smaller node IDs count here. Supposing that in the examples in that subsection all transaction-generating nodes had larger node IDs, then the maximum waiting time for node A would be 110 cycles (22 microseconds) in the first case and 46 cycles (9.2 microseconds) in the second case. These figures compare with 65.2 microseconds and 24.8 microseconds as computed in that subsection.

These results assume that node A does not arbitrate more than once in any interval of length T, and T is greater than the guaranteed response interval if all nodes are arbitrating in dual round-robin mode. The results above are useful, therefore, only for a node which requires faster response than can be guaranteed by dual round-robin but also requires quite a bit less throughput rate than the fast response time requirement might suggest.

In particular, if fast response time is required because of a node's peak transfer rate (for example, that of a fast unbuffered disk device connected to a VAXBI node), the results above are not useful for achieving the required response, because the throughput rate requirement would conflict with the assumption of no second request in any interval T.

CHAPTER 11

ERROR DETECTION AND MAINTAINABILITY

This chapter discusses features that contribute to the efficient functioning of VAXBI systems.

- Self-Test -- Each node automatically performs a self-test or initialization.
- Error Checking -- Data integrity is ensured by parity checking, comparison of transmitted and received data, and protocol checking. The BIIC provides these functions.
- Stopping a Node -- Hardware malfunctions can be diagnosed using the STOP transaction, which causes a node to stop generating VAXBI transactions and allows it to be examined.

11.1 SELF-TEST OPERATION

This section first gives VAXBI requirements for self-test and then specifies self-test operation for nodes that use the BIIC. Other information on self-test appears in Chapter 6 and Application Note 4. Chapter 6 details initialization, which includes self-test, and Application Note 4 gives more information on the operation of self-test.

On initialization every VAXBI node must automatically perform a self-test, which includes a self-test of the VAXBI primary interface (the logic that interfaces directly to the VAXBI signal lines) and self-test of the rest of the node. Node self-test must not depend on other VAXBI nodes to complete.

The mechanism for self-test reporting utilizes the BI BAD L line, the Broke bit, and a pair of yellow LEDs on each VAXBI module. One of the LEDs is on the top of the module, and one is on the front of the module. Both LEDs indicate the same information: a lit LED indicates self-test passed.

digital™

The VAXBI primary interface self-test verifies VAXBI control logic and the accessibility of the VAXBI registers required of all nodes. The remainder of node self-test may be performed after, in parallel with, or partially overlapped with the VAXBI primary interface self-test.

11.1.1 Self-Test Requirements

The specified sequence of self-test is detailed below.

1. On power-up, the Broke bit must be set, the LEDs must be off, and the BI BAD L line must be asserted. The VAXBI primary interface must ensure that all data path and synchronous control signals (except BI NO ARB L) are deasserted.

2. The node begins its self-test following either the deassertion of BI DC LO L or the setting of the Node Reset (NRST) bit in the VAXBICSR. During power-up self-test, and until the VAXBI registers that are required of all nodes are functional, the node must assert BI NO ARB L. However, during node reset self-test, the node must not assert BI NO ARB L. Also, during node reset self-test, and until the VAXBI registers that are required of all nodes are functional, the node must respond with a NO ACK when the VAXBI required registers are accessed. (Table 7-1 lists the registers required of all nodes.) With the exception of the VAXBI registers, other nodes must not access locations within a node undergoing self-test.

   Node designs must make every effort to ensure that the BI NO ARB L line deasserts so that a node that fails self-test does not prevent other nodes from gaining access to the VAXBI bus. The VAXBI primary interface is therefore required to implement a "watchdog timer" or equivalent that will disable the data path and synchronous control signals (including BI NO ARB L).

   The node self-test tests the whole node, and its results include those of the VAXBI primary interface self-test. A node passes node self-test only if its VAXBI primary interface passed its self-test. Two kinds of self-test must be implemented at each node: a fast one and a slower but more thorough one.

digital™

3. If the node's self-test indicates that the node may corrupt the bus, the node should disable all data path and synchronous control signals. This ensures that a failed node will not prevent future bus transactions. If the node does not pass self-test, then the Broke bit must remain set, the BI BAD L line must remain asserted, and the LEDs must remain off.

4. If the node passes self-test, the Broke bit must be cleared, the LEDs must be lit, and the BAD line must be deasserted, with the following timing constraints:

   - The BAD line must be deasserted within 100 ms after the clearing of the Broke bit.
   - The LEDs must be lit within 100 ms of the clearing of the Broke bit.

11.1.2 Self-Test Operation with a BIIC

Self-test operation for nodes that use a BIIC as their VAXBI primary interface is detailed below. This operation complies with the steps in Section 11.1.1.

1. On power-up, for all but slave-only nodes, the Broke bit is set by the BIIC. User interface logic in slave-only nodes must set the Broke bit in the SOSR. User interface logic must ensure that the LEDs are off and the BI BAD L line is asserted. The BIIC as VAXBI primary interface ensures that all data path and synchronous control signals (except BI NO ARB L) are deasserted per the requirement.

2. The BIIC deasserts BCI DC LO L either due to the deassertion of BI DC LO L or the setting of the Node Reset (NRST) bit in the VAXBICSR. The BIIC and user interface logic must therefore begin self-test following the deassertion of BCI DC LO L. (There is no need for user interface logic to monitor the NRST bit to determine when to begin self-test.)

3. During power-up self-test, and until the VAXBI registers that are required of all nodes are functional, the BIIC as the VAXBI primary interface must assert BI NO ARB L. However, during node reset self-test, the BIIC does not assert BI NO ARB L. Also, during node reset self-test, the BIIC as the VAXBI primary interface must respond with a NO ACK when the required registers are accessed.

digital™

If a node is to be operated (that is, after self-test completes) with bit <13> of the Device Register set, then that node's user interface self-test must perform a check to ensure that a Stall Time Out does not occur. If a Stall Time Out condition does occur during such a self-test check, then the user interface must declare a self-test failure. The purpose of these requirements is to ensure that previous revisions of BIICs (which did not have the capability of operating with Stall Time Out disabled) will not be inadvertently installed on nodes that require the disabling of Stall Time Out. If these requirements are met and such an inadvertent installation occurs, the node will not pass self-test.

Bit <13> of the DTYPE register defaults to the set state on power-up if D<13> is not actively driven by the user interface at the deassertion of BCI DC LO L. Because of this, nodes must not rely on the capability of the BIIC to signal a Stall Time Out during node self-test until DTYPE<13> is cleared. See the description of DTYPE<13> in Section 7.1.

The BIIC design includes a "watchdog timer" to ensure that the BI NO ARB L line deasserts at the completion of power-up self-test so that a node that fails self-test does not prevent other nodes from gaining access to the VAXBI bus. If the BIIC's self-test indicates that it may corrupt the bus, the BIIC disables all data path and synchronous control signals. The user interface logic must not reset the Broke bit if the node fails its BIIC self-test. User interface logic must keep the BI BAD L line asserted, and the LEDs must remain off in this case.

4. If self-test completes successfully, the user interface must clear the Broke bit and must ensure that:

   - The user interface logic must deassert the BAD line within 100 ms after the user interface clears the Broke bit.

   - The user interface logic must light the LEDs within 100 ms of the clearing of the Broke bit.

Figure 11-1 summarizes the self-test process. Note that node self-test cannot complete until the BIIC self-test completes.

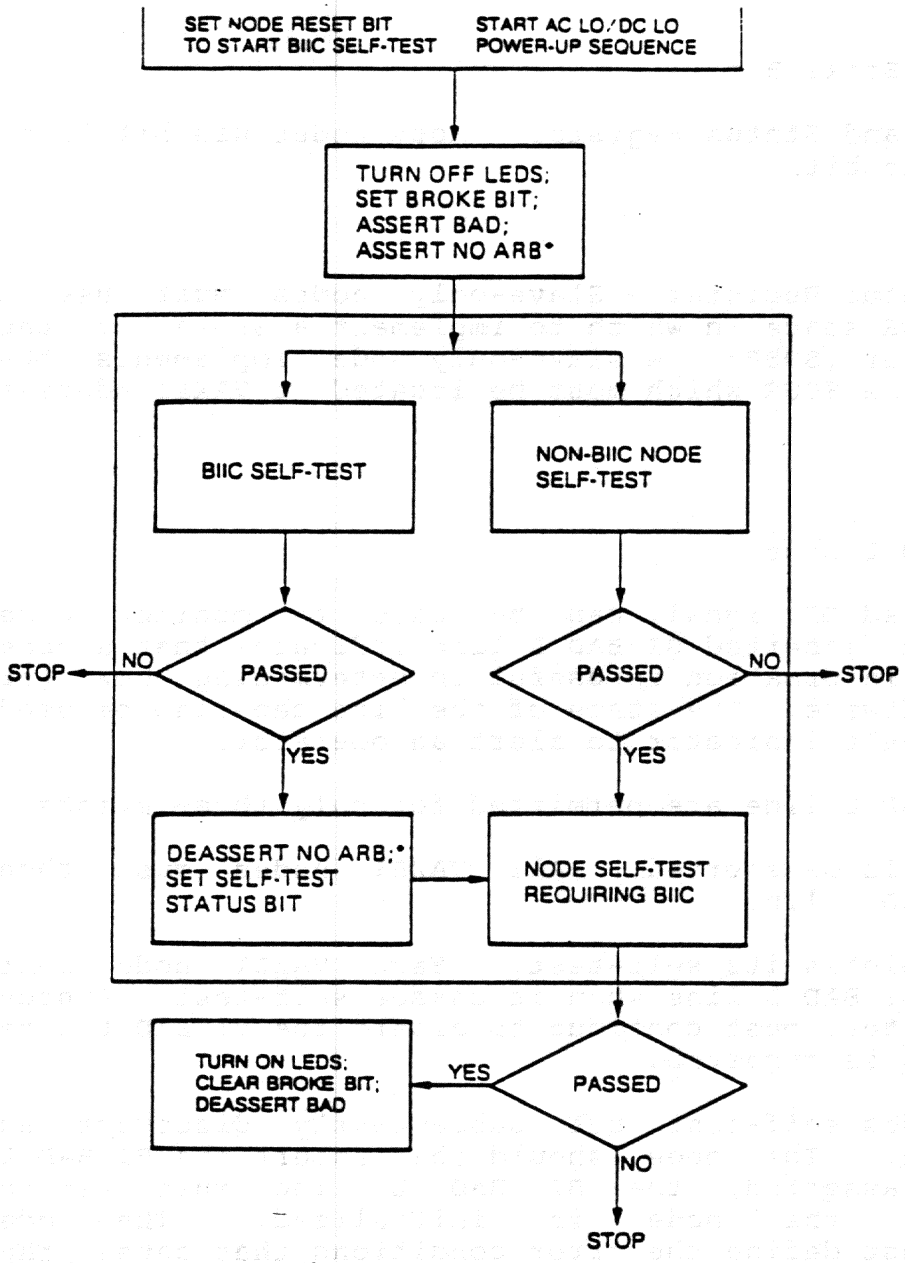


Figure 11-1: Self-Test Flow

If the BIIC fails self-test, the output drivers of the BIIC a disabled so that the BIIC cannot drive the VAXBI lines. Setting the Self-Test Status (STS) bit in the VAXBICSR to one enables these drivers, but this should be done only for diagnostic purposes.

digital™

11.1.3 Location of the Broke Bit

11.1.3.1 VAXBI Control and Status Register - Most nodes use bit 12 of the VAXBICSR as the Broke bit.

11.1.3.2 Slave-Only Status Register - Slave-only nodes must use a register outside BIIC CSR space in which to implement a Broke bit, the Slave-Only Status Register (SOSR). A slave-only node implements the Broke bit in bit 12 of the SOSR which must be located at VAXBI address bb + 100 (hex).

11.1.4 Using the BI BAD L Line

The BI BAD L line, a wired-OR signal, can be used to monitor node self-test results. An asserted BI BAD L line indicates that a node failed self-test. This information is useful in determining how (or if) to start system software. The state of the line can also be used to drive a systemwide fault indicator to alert an operator.

Transitions of the BI BAD L line are permitted for only three events:

- When BI DC LO L is deasserted. All VAXBI nodes must then assert the BI BAD L line.
- When a node completes its self-test. Each VAXBI node must deassert the BI BAD L line when it passes self-test. A node that fails self-test must continue to assert the BI BAD L line until BI DC LO L is asserted.
- When a node passes self-test but subsequently discovers an error condition. The node should then assert the BI BAD L line. Once so asserted, the BI BAD L line must remain asserted until the node is initialized. The node specification must define the error conditions that cause the assertion of BI BAD L.

11.1.5 Self-Test Time Limits

VAXBI Primary Interface Self-Test

If the VAXBI primary interface passes its self-test, its VAXBI registers must be functional and BI NO ARB L must be deasserted within 5 milliseconds (ms) after the deassertion of BI DC LO L. If the VAXBI primary interface self-test fails, BI NO ARB L must be deasserted within 500 ms after the deassertion of BI DC LO L.

Normal Self-Test

When BI STF L is not asserted, all nodes (except the primary processor) must complete self-test within 10 seconds after the deassertion of BI DC LO L, independent of the state of BI AC LO L.

VAXBI nodes are required to complete normal self-test in 10 seconds, regardless of how much VAXBI traffic there is. This is referred to as the "global normal self-test time requirement." There is an analogous global fast self-test time requirement of 250 ms.

These global self-test time requirements make it simple for the primary processor to decide when it can examine nodes to see if they successfully completed self-test. The node designer must somehow allow for worst-case bus latencies for the VAXBI transactions that the node issues during self-test to help ensure that self-test is completed in time.

Suppose each individual node is capable of completing normal self-test on an otherwise-idle* VAXBI bus within 9.9 seconds of the deassertion of BI DC LO L. Then all nodes will be capable of completing normal self-test on a fully populated VAXBI bus within 10 seconds.** The condition in the first sentences is referred to as the "9.9 second criterion."

Note that the 9.9 second criterion is sufficient but not necessary. That is, a node may fail to meet the criterion but still be capable of satisfying the global normal self-test time requirement. However, if the node fails to meet the criterion, then the designer must prove the node satisfies the global normal self-test time requirement.

The primary processor need not conform to the 10 second limit, since this limit is intended to act as a guarantee that all other nodes make to the primary processor. The primary processor is, in this context, the node that determines the action to be taken after self-test. A processor that is not the primary processor must conform to the 10 second limit.

Fast Self-Test

When BI STF L is asserted, all nodes are required to complete self-test within 250 ms after the deassertion of BI DC LO L, independent of the state of BI AC LO L.

Suppose each individual node is capable of completing fast self-test on an otherwise-idle* bus within 220 ms of the deassertion of BI DC LO L. Then all nodes on a fully populated VAXBI bus will be able to

---

*That is, with transactions generated only by the node undergoing self-test.
**Discussed in Application Note 4, Section 4.2.1.

complete fast self-test within 250 ms (discussed in Application Note 4, Section 4.2.2).

Note that the 220 ms criterion (from the example above) is sufficient but not necessary. That is, a node may fail to meet the criterion but still be capable of satisfying the global fast self-test time requirement. However, if a node fails to meet the criterion, then the designer must prove the node satisfies the global fast self-test time requirement.

The 250 ms requirement only applies to nodes that pass self-test. This requirement reverts to the 10 second limit for battery-backed-up memory nodes if the battery was discharged or not installed, or if self-test was initiated in response to the assertion of the BI RESET L line by a node (rather than in response to a power outage).

Extended Self-Test

If a node cannot complete adequate testing within the 10 second limit, it can use an extended self-test (discussed in Application Note 4).

11.1.6 Using the VAXBI Bus During Self-Test

As part of its self-test, a node should perform VAXBI transactions to verify the correct functioning of the node's VAXBI transceivers and data paths. Any VAXBI transactions performed are subject to the following rules:

- Except for reads to the Broke bit, a node must not access another node until the other node completes its self-test.
- All VAXBI transactions must be directed to I/O space allocated to the issuing node. Although the intended segment is nodespace, the node's node window can also be used.
- INVAL, BDCST, and RESERVED code transactions are forbidden. Read- and write-type transactions are limited to longwords.
- IPINTR transactions cannot be sent to other nodes. If INTR or IDENT transactions are issued, or if the HEIE or SEIE bits are set, the Interrupt Destination Register cannot point to any other node.
- Only dual round-robin arbitration can be used.

- The time required by a node for VAXBI transactions is restricted:
  - If BI STF L is asserted, a VAXBI node should perform no more than a combined total of 512 VAXBI and loopback transactions.
  - If BI STF L is not asserted, a VAXBI node should perform no more than a combined total of 2048 VAXBI and loopback transactions.
  - If a node is to operate with Device Register bit <13> cleared, then no transaction should have more than 10 stalls, and the average number of stalls should not exceed 4 per transaction.
  - If a node is to operate with Device Register bit <13> set, then at least one transaction (the one that checks for th occurrence of STO) will require more than 10 stalls Therefore, such a node should not stall more than 250 times for that transaction. Other transactions should not be stalled more than 10 times, and the average number of stalls (computed over the non-STO-check transactions) should not exceed 4 per transaction. See Section 11.1.2 Self-test Operation with a BIIC.
- Upon completion of self-test, the node must be in a well-defined state, with no interrupts pending:
  - The Device Register must be valid. The VAXBI Control and Status Register must be valid; and the UWP, HEIE, and SEIE bits and the ARB field must all be cleared.
  - The Error Interrupt Control Register, the Interrupt Destination Register, the IPINTR Mask Register, the Force-Bit IPINTR/STOP Destination Register, the IPINTR Source Register, and the User Interface Interrupt Control Register must all be cleared.
  - The Starting and Ending Address Registers must either be cleared or set to the node's node window space.

11.1.7 Device Type Requirements

The Device Register contains a Device Type field and a Device Revision field.

Bit <15> of the Device Type field is zero for Digital-supplied devices, while device type codes with bit <15> set to one are reserved

digital™

for devices not supplied by Digital.

Slave-only nodes will have bits <14:8> set to all zeros. These nodes implement the Broke bit in a nodespace register, the Slave-Only Status Register (see Section 7.16). Digital-supplied slave-only nodes will therefore have the high-order byte in the Device Type field set to all zeros, while non-Digital-supplied slave-only nodes will have bit <15> set to one and bits <14:8> set to all zeros.

Bit <13> of the BIIC Device Type field, when set, disables the occurrence of the Stall Timeout on Slave Transaction (STO) EV code, and the subsequent deassertion of BI BSY L by the BIIC slave port. Bit <13>, when set, also disables the setting of the STO bit in the BIIC's Bus Error Register. The use of the BIIC with bit <13> set is RESERVED to Digital and nodes that require bit <13> be set need an exception (see Section 10.2.1 Extension Cycles). Nodes that are allowed to operate with bit <13> set must meet additional requirements beyond those for a normal VAXBI node (see Section 11.1.2).

A device type code of all zeros is reserved for use by Digital. A device type code of all ones indicates that the Device Type field has not yet been loaded.

The Device Type field is either loaded with a device type at power-up and node reset or remains all ones until written with a device type code by the node. Once the device type is loaded, it must not be changed.

The device type code at one node may be examined by another node any time after the VAXBI primary interface passes self-test to determine if the node is a slave-only node (that is, to determine the location of the Broke bit). This may happen before the node completes self-test. Therefore, at the deassertion of BI DC LO L, every node must:

- Allow the VAXBI primary interface to default the Device Type field to all ones. A field of all ones indicates that the device type has not yet been loaded,
  or
- Set the Device Register so that it contains the device type code.

Thus, if the Device Register is examined before the device type is loaded, a device type code of all ones will be found. If this condition persists beyond the self-test time limit, the node did not pass self-test.

For procedures related to the assignment of device type codes to VAXBI licensees, see Appendix G.

11.2 ERROR DETECTION AND RESPONSE

All VAXBI nodes are required to implement several forms of error detection and error logging. These functions must be provided by the VAXBI primary interface without support from the user interface. At each node the VAXBI primary interface (for example, the BIIC) checks parity, compares transmitted and received data, and performs protocol checking. Any errors detected by the VAXBI primary interface are logged in the Bus Error Register (BER) (see Section 7.3).

Section 11.2.1 discusses three types of error detection that are required. Section 11.2.2 discusses conditions that will cause an operation to abort and the sequence of an abort. Section 11.2.3 discusses how nodes should respond to exception conditions.

11.2.1 Error Detection

11.2.1.1 Parity Checking - System integrity is enhanced through the use of parity generation and checking during command/address and data cycles. ODD parity is generated on the BI P0 L signal line for the data path signals. ODD parity is used since VAXBI parity defaults to incorrect when all data lines are deasserted (this allows the immediate notification at the receiving node that the transmitting node has aborted and released the VAXBI bus). Masters generate parity for:

- Command/address data
- Their encoded ID during imbedded arbitration cycles
- Write-type and BDCST data
- Master's decoded ID during IDENT transactions

Slaves generate parity for:

- Read data cycles
- Vector data that is being returned

Masters check parity for:

- Read-type ACK data cycles and vector ACK data cycles

digital™

Slaves check parity for:

- o Write-type STALL and ACK data cycles
- o BDCST ACK data cycles

All nodes check parity for:

- o Command/address cycles
- o Node ID on imbedded ARB cycles
- o Null bus cycles

Upon detection of a parity error in the command/address cycle, nodes should set the Command Parity Error status bit in their Bus Error Register. If error interrupts are enabled, the nodes should also send an error interrupt. Any node detecting a command/address parity error must not allow itself to be selected by the address information.

Detection of a parity error by one of the participants of the read and write transactions specified above causes either a Master or Slave Parity Error status bit to be set in the node's Bus Error Register. If error interrupts are enabled, the node also sends an error interrupt. All nodes check parity on the BI I<3:0> L lines during the imbedded arbitration cycle of a transaction. Since data errors on the received ID do not affect system data transfer integrity, these parity errors must be recorded as a soft error bit (ID Parity Error) set in the Bus Error Register. The detection of a soft error condition must not cause the node to abort the transaction.

In the null cycle state of the bus, the BI I<3:0> L and BI D<31:0> L lines are deasserted. Nodes determine the presence of this condition by monitoring the BI NO ARB L and BI BSY L signals. If both BI NO ARB L and BI BSY L are not asserted for two consecutive bus cycles, then the second cycle of this sequence and all subsequent consecutive bus cycles with BI NO ARB L and BI BSY L deasserted are defined as null bus cycles. Null bus cycles are parity checked. If ODD parity is detected (that is, the BI data path lines were not all deasserted), then a null bus parity error is logged.* Hard Error Interrupts are not generated for this error condition since system integrity has not been compromised and disruption of system operation might be undesirable. A Soft Error Interrupt can be transmitted if failure statistics are to be recorded in an error log.

---

*Spurious null bus parity errors may be logged in a node's BER as a result of that node undergoing a node reset.

11.2.1.2 Transmit Check Error Detection - Each master is required to compare transmitted data with data received at its node during cycles when it is the only source of data on the data path. The transaction must be aborted if the transmitted data does not match the received data. This check prevents data from being corrupted in the event that a transient error causes two masters to take the bus. The detection of a transmit check data error by the transaction master results in the setting of the Master Transmit Check Error (MTCE) bit in the Bus Error Register. This check must not be made during the assertion of the master's encoded ID on the I lines during the imbedded arbitration cycle.

Each VAXBI master and slave is required to verify its assertion of BI BSY L, BI NO ARB L, and BI CNF<2:0> L control lines. If a node detects a deasserted state on one of these lines while it is expected to be driving the signal, this is an error condition and results in the setting of the Control Transmit Error (CTE) bit in the Bus Error Register.

11.2.1.3 Protocol Checking - The Bus Error Register also logs errors that relate to the VAXBI protocol. For example, the Interlock Sequence Error (ISE) bit is set when IRCI and UWMCI transactions are performed out of sequence. Other BER bits are used to indicate that the confirmation responses received are illegal.

11.2.2 VAXBI Primary Interface Abort Conditions

The VAXBI primary interface must determine when a transaction should be aborted. As master, it must abort a bus transaction when it detects one or more of the following:

- A RETRY command response to a single-responder command
- A NO ACK command response
- An illegal or RESERVED command or data response
- A parity error on read data
- A parity error on vector data
- A transmit check error on the D, I, or P lines

digital™

As slave, the VAXBI primary interface must abort a bus transaction when it detects one or more of the following:

- o A stall timeout condition
- o A parity error on write-type or BDCST data

It is important that an orderly transition occurs to end the transaction. Once a transaction has begun, it must be continued for at least three cycles to the command confirmation cycle. In this way the aborting master's ID can be recognized and the reason for the abort can be determined.

A master must abort a bus transaction by concurrently deasserting all VAXBI signals within two bus cycles after the occurrence of the abort condition unless the cycle following the error cycle is stalled by the slave. In this case the master may delay its abort until two cycles after the next non-STALL confirmation from the slave. The master must inhibit parity checking and recognizing CNF responses from the slave for cycles that occur after the master aborts the transaction. This prevents secondary errors from being recorded by the master.

Pending masters may delay their assertion of BI BSY L for up to two cycles after an aborted transaction so that the node has time to complete the abort recovery.

11.2.3 Response to Exception Conditions

This section describes appropriate responses of nodes to unusual conditions on the bus, most of which indicate some sort of error. Generally, a node can repeat a transaction if a slave is temporarily unable to service the transaction or if some error condition occurred during the transaction. However, repeating a transaction is not appropriate in the following situations:

- o If a NO ACK confirmation code is received for an IDENT transaction, the transaction should not be reissued, since NO ACK is an acceptable response. A NO ACK confirmation code is acceptable for an IDENT transaction because the interrupting node may have sent the interrupt to more than one processor, in which case all processors except the first one to respond may receive a NO ACK confirmation.
- o A write-type transaction to I/O space during which a bus error occurred must not be repeated, since the transaction may have caused a side effect. However, READ and RCI transactions may be repeated, since they are required not to have any side effects. (Read-type transactions to a UNIBUS adapter's node window are an exception to this statement.) (Side effects are described in Section 8.2.3.)

digital™

- An unsuccessful UWMCI transaction (that is, one during which a bus error occurred, such as a parity error) must not be repeated. The transaction may have been completed at the slave, so if the transaction is repeated, the second UWMCI may clear a lock that was set after the first UWMCI.

If the bus master repeats an IPINTR transaction on receiving a RESERVED or illegal confirmation code, the slave may receive a redundant IPINTR. (For example, a slave may have sent an ACK response, but the ACK was corrupted and then interpreted as an illegal CNF code). In this case the repetition is permissible.

If exception conditions persist and the issuing node is a processor, the processor should signal the condition to the software if any of the following apply:

- The limit is reached on the number of times a transaction is reissued after receiving a RETRY confirmation.
- Bad parity is received.
- An illegal confirmation code is received.
- The transaction is not an IDENT transaction, and a NO ACK confirmation code is received.
- The transaction is a read-type, write-type, INVAL, or INTR transaction, and the transmitting node detects that the data received on the bus is not the signal that was transmitted.

If a slave detects an error or exception condition involving an IRCI transaction, the slave must not set the lock. The master must not attempt to recover from the situation by unlocking the location with a UWMCI transaction, since this can generate more errors, which are difficult to detect and diagnose. It is permissible for the master to attempt to recover by reissuing the IRCI transaction.

A slave receiving bad parity during a write-type transaction should either suppress the write or, if parity bits are implemented in memory and they can be written, write the data along with bad parity (so that a subsequent read to that location will find bad parity).

See Appendix C for guidelines on how nodes should respond to each event (EV) code for each type of VAXBI transaction.

digital™

11.3 USE OF THE STOP TRANSACTION

The STOP transaction provides for the examination of VAXBI nodes when an error is perceived. The nodes selected by a STOP transaction cease to generate VAXBI transactions but can respond to VAXBI transactions. Posted interrupts are cleared. The goal is to retain as much state as possible for diagnostic purposes. Any locks set by IRCI transactions must not be cleared. In general, after a node receives a STOP command, it must be put through a power-down/power-up sequence to restart properly.

digital™

CHAPTER 12

ELECTRICAL SPECIFICATION

This chapter gives the electrical specification for the signals on the VAXBI bus.

12.1 TEST CONDITIONS

All electrical specifications must be met over the full range of VAXBI operating conditions. These conditions are described in Section 13.7.

12.2 CLOCK SIGNAL TIMING

The relationships between the various VAXBI clock signals are shown in Figure 12-1.

12.2.1 BI TIME +/- Signals

The BI TIME +/- clock signals are generated at the beginning of the bus and are terminated at the opposite end. These signals, in conjunction with BI PHASE +/-, are used by all nodes to generate internal TCLK, RCLK, and other derived synchronous signals. The BI TIME +/- signals constitute a 20 MHz differential square-wave.

Since some VAXBI node designs depend on the specified frequencies of the clock signals for proper operation, single-step operation of the clock signals may not be possible in some VAXBI systems.

digital™

12.2.2 BI PHASE +/- Signals

The BI PHASE +/- clock signals are generated at a single point on the bus (at the same point as BI TIME +/-) and are used by all nodes, in conjunction with BI TIME +/-, to generate internal TCLK, RCLK, and other signals. The BI PHASE +/- signals constitute a 5 MHz differential square-wave.

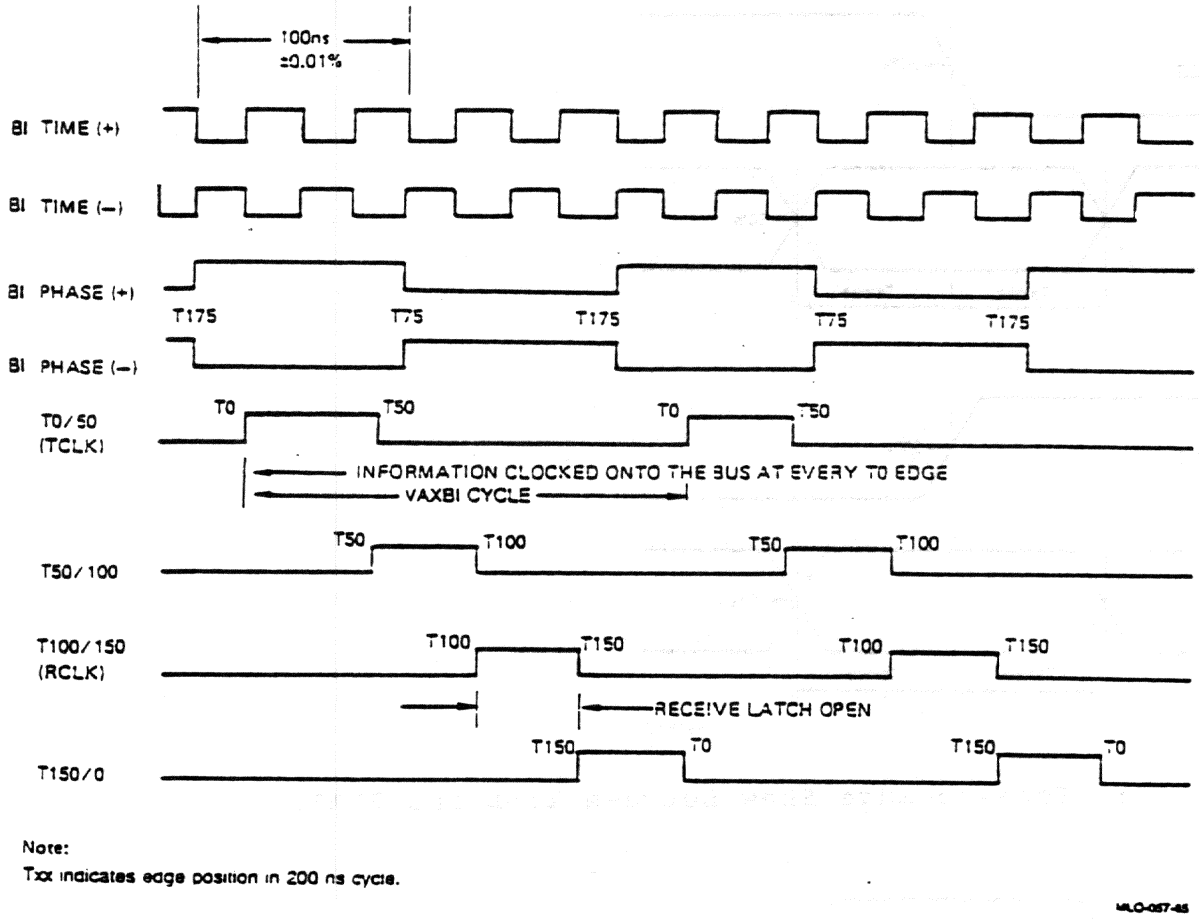
12.2.3 Clock Skew

Skew is defined as the maximum absolute value of the parameters shown in Figures 12-2 and 12-3. Skew can occur in either direction. The skew between the corresponding edges of TIME and PHASE measured at a given node (Tskwe) is +/-3.0 nanoseconds maximum (as measured per Figure 12-2).

Differential clock skew (Tskwd) between true and complement phases of each signal is +/-1.0 nanosecond maximum (as measured per Figure 12-3).

12.2.4 Clock Signal Integrity

The minimum guaranteed difference voltage between the true and complementary differential clock signals in their nontransition region must be 300 mV, measured at any node in any bus configuration.



Note:
Txx indicates edge position in 200 ns cycle.

MLO-057-45

Figure 12-1: Clock Timing

digital™

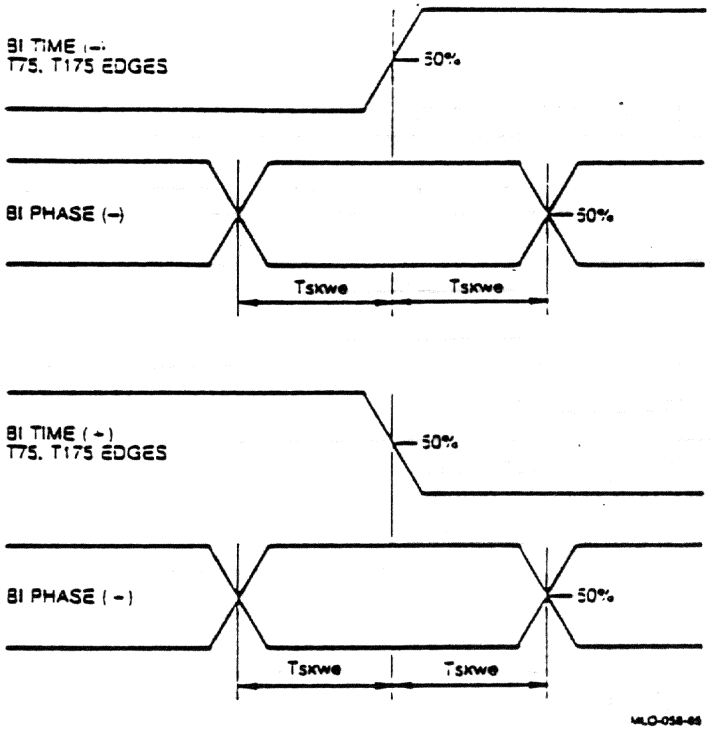


Figure 12-2: Edge-to-Edge Skew Between TIME and PHASE

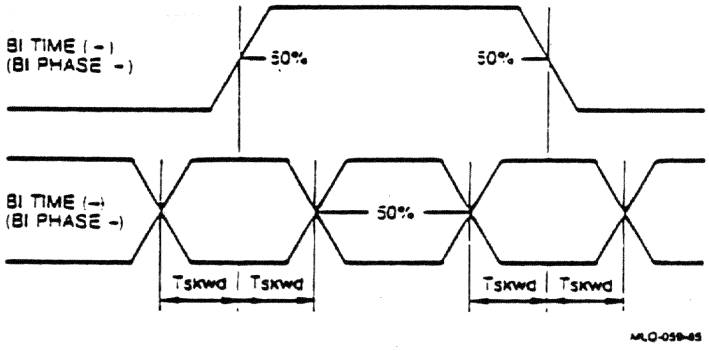


Figure 12-3: Differential Clock Skew

12.2.5 Asynchronous Control Signal Timing

A maximum rise time of 1 microsecond is specified for reducing noise sensitivity along the deasserting edge of these signals. This maximum time corresponds to a maximum capacitance load of 3000 pF. With present VAXBI card cages, terminators, and intercage cables, the signal loading exclusive of power status signal cables is approximately 500 pF.

All rise time and fall time measurements are 10% to 90%.

digital™

12.3 INTERCONNECT ELECTRICAL CHARACTERISTICS

12.3.1 Maximum Capacitance Requirements

All VAXBI signals except clock signals have a maximum capacitance requirement.

12.3.1.1 Data Path and Synchronous Control Lines - All data path and synchronous control lines have a maximum capacitance requirement that must be met in any system regardless of configuration:

> 410 pF maximum total including transceiver silicon and package loading, all mechanical connections including VAXBI backplane etch, connectors, terminator loading, and intercage cables on each signal line.

12.3.1.2 Asynchronous Control Lines - All asynchronous control lines have a maximum capacitance requirement that must be met in any system regardless of configuration:

> 3000 pF maximum total including transceiver silicon and package loading, all mechanical connections including VAXBI backplane etch, connectors, terminator loading, and intercage cables and non-intercage cables on each signal line.

All asynchronous control lines can extend off the backplane without utilizing an intercage cable. If the lines are extended, the extension length must not exceed 2 meters.

12.3.2 VAXBI Backplane Requirements

All VAXBI backplane requirements are satisfied by the VAXBI backplane (Digital internal P/N 50-16148-01). This backplane component is part of the H9400 and H9402 card cage assemblies.

12.3.3 VAXBI Extension Cable Requirements

All VAXBI intercage extension cable requirements are met by the VAXBI intercage cable assembly (Digital internal P/N 17-01038-01). This component is available as part of the H9400 card cage assembly.

12.4 DATA PATH AND SYNCHRONOUS CONTROL LINE TRANSCEIVERS

All required transceiver characteristics are met by the BIIC (Digital P/N 78732).

12.5 ASYNCHRONOUS CONTROL LINE DRIVERS

Circuit configuration: Open collector or open drain
Leakage current: Ioh = 250ua @2.3V
Output low voltage: 0.5 volts max. at Iol = 18.6 mA

12.6 ASYNCHRONOUS CONTROL LINE RECEIVERS

For BI BAD L, BI STF L, and BI RESET L

TTL, LSTTL, FTTL, and STTL inputs allowed

For BI DC LO L and BI AC LO L

All receiver requirements for these lines are met by the BIIC. The BIIC is the only acceptable component for receiving these lines.

12.7 CLOCK LINE DRIVER OUTPUT CHARACTERISTICS

All requirements are met by the VAXBI clock driver (Digital P/N 78701).

12.8 CLOCK LINE RECEIVER INPUT CHARACTERISTICS

All requirements are met by the VAXBI clock receiver (Digital P/N 78702).

12.9 VAXBI TERMINATION SCHEME

12.9.1 Data Path, Synchronous Control, and BI AC LO L and BI DC LO L Signals

Each of these lines must be terminated by a clamped resistive pull up. The clamping network reduces the capacitive discharge current into the bus drivers and thereby reduces inductive ground shifting effects and VAXBI driver power dissipation.

All VAXBI termination requirements are met by the VAXBI terminators (Digital internal P/N 20-24486-01 and 20-24487-01).

12.9.2 Termination of BI TIME +/- and BI PHASE +/- Clocks

The VAXBI clock lines must be terminated by a differential ECL pulldown network.

All VAXBI clock termination requirements are met by the VAXBI terminators (Digital internal P/N 20-24486-01 and 20-24487-01).

12.10 MAXIMUM CONFIGURATION

The maximum VAXBI system configuration consists of the following:

- 16 VAXBI modules with BIICs (refer to Chapter 13, T1999 Module Control Drawings)
- 20 VAXBI expansion modules (refer to Chapter 13, T1996 Module Control Drawings)
- 5 VAXBI intercage cables

Currently, this system can be implemented with six H9400 or H9402 card cages and five intercage cables (P/N 17-01038-01).

12.11 INTERFACING AND CONFIGURATION RULES

Each VAXBI node must use a standard layout macro for interfacing to the bus. The BIIC, the VAXBI clock driver and clock receiver, and associated discrete components are to be located in a specified location on the module.

By standardizing the electrical parameters of the interface, Digital has essentially removed the possibility of varying layout impedance effects from board to board that may be detrimental to proper bus operation. This includes capacitive loading effects as well as but not limited to crosstalk susceptibility, transmission line effects, and inductance effects on noise margin.

See Chapter 13, Mechanical and Power Specification, for more detail on interfacing and configuration rules.

12.12 WORST-CASE VAXBI BUS TIMING

12.12.1 Time Reference Standards

Discussion of worst-case bus conditions can be confusing unless a clear set of time references is defined. The required time references and associated names are defined in this section.

Each node transmits and receives data based on the TIME H/L and PHASE H/L signals output by its clock receiver. These signals are used to define four cycle times or edges:

T0, T50, T100, and T150

T0 occurs at the beginning of a VAXBI cycle (as indicated by the falling edge of TIME L, in conjunction with an asserted PHASE L). T50 occurs 50 ns later, T100 occurs 100 ns later, and T150 occurs 150 ns later. Since the VAXBI clock signals are not received simultaneously by all nodes, there is a set T0/T50/T100/T150 times for each node. The set is indicated by the following notation:

T0_node0, T0_node1, etc.

Since most worst-case analysis involves looking at "near-end" to "far-end" bus transfers, it is useful to introduce the following terms:

T0_ne and T0_fe

T0_ne is the T0 time as seen at the first node on the VAXBI bus (that is, the clock source module), and T0_fe is the T0 time as seen at the last node.

12.12.2 Timing Parameters Used for Worst-Case Calculations

Table 12-1 lists values and sources for all timing parameters used in this manual. Time is in nanoseconds.

Table 12-1: Timing Parameters Used for Worst-Case Calculations

Symbol	Parameter	Min.	Max.	Where Specified
-	VAXBI clock etch max. delay	0.0	14.5	SPICE simulation*
Tplh	VAXBI clock receiver delay (BI TIME =/- to BCI TIME L)	1.5	6.0	78702 Spec.
Tbas	VAXBI signal delay from node's T0 -- H to L	0.0	85.0	78732 Spec.
Tbr	VAXBI signal delay from node's T0 -- L to H	0.0	85.0	78732 Spec.
Tbs	BIIC setup time from node's T0 -- H to L	20.0	-	78732 Spec.
Tbh	BIIC hold time requirement on VAXBI data from T150	20.0	-	78732 Spec.

* Verified with empirical data.

12.12.3 Worst-Case VAXBI Setup Time Calculations

For a 200 ns cycle, the first 150 ns are devoted to satisfying the criteria that data transmitted on the T0 transmit time edge of TIME at a "far-end" node gets reliably received at a "near-end" node.

The analysis is simplified by assuming that VAXBI data path signals can be modeled as a lumped capacitance as opposed to a transmission line. This assumption is substantiated in Section 12.12.4.

The VAXBI clock system must be considered a transmission line in this analysis since clock signal rise and fall times are much less than the round trip times for these signals on the bus. In this case bus propagation time IS significant; however, since the clocks are differentially received, rise and fall times are NOT taken into account.

In the "configuration" shown in Figure 12-4, data is being transmitted from node B to node A. Node B is at the far end of the bus (near the clock terminator); node A is at the near end (near or at the clock source (driver). The bus is fully loaded per Section 12.10. The following analysis examines the setup time provided at node A for data driven by node B.

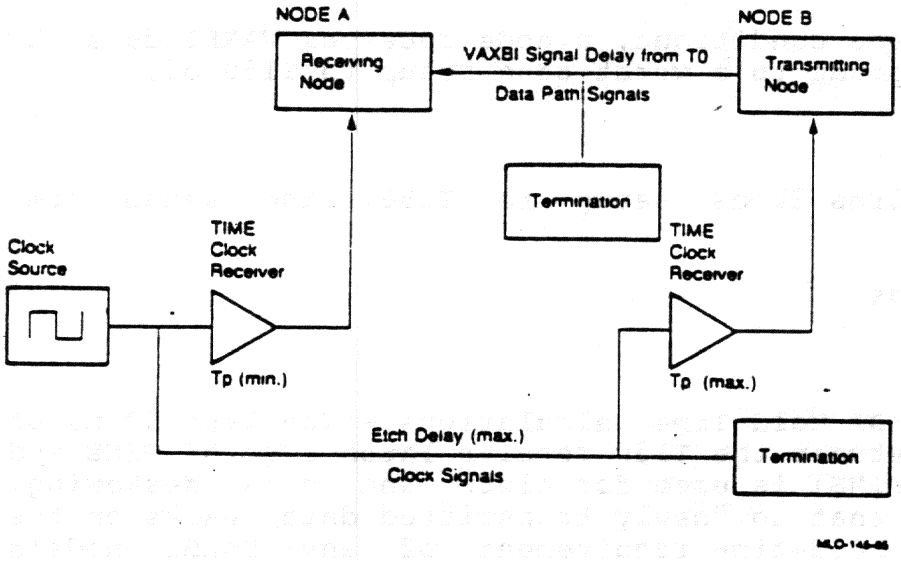


Figure 12-4: Worst-Case Setup Time Configuration

Clock Calculations

T0_nodeA = T0_ne

T0_nodeB = T0_fe = T0_ne + BI clock etch delay +
(Tp_receiver_fe - Tp_receiver_ne)

T0_nodeB(max) = T0_ne + 14.5 ns + (6-1.5) ns
= T0_ne + 19.0 ns

digital™

Worst-Case Setup Time Calculations

Data is transmitted from nodeB at T0_fe and is received by nodeA at T150_ne.

The following equations calculate the time that VAXBI data will be valid at nodeA when transmitted by nodeB:

VAXBI data valid @nodeA = T0_fe + VAXBI signal delay

VAXBI data valid @nodeA (max.) = T0_fe(max.) + VAXBI signal delay(max.)
= T0_ne + 19.0 ns + 85.0 ns
= T0_ne + 104 ns
= T0_nodeA + 104 ns

Therefore, under worst-case conditions, a node receives VAXBI data by "its" T106. This corresponds to a worst-case setup to T150 of,

150 - 104 = 46 ns

Given that the BIIC requires 20 ns setup to T150, the setup time margin is:

46 ns - 20 ns = 26 ns

12.12.3.1 Worst-Case VAXBI Hold Time Calculations - The last 50 ns of bus cycle (the time between the T150 receive latch edge of TIME and the T0 transmit edge of TIME) is used for clock and data deskewing. The 50 ns here assures that no "newly transmitted data" walks on the previous data during the hold-time requirement of any VAXBI node's transceivers.

In the "configuration" shown in Figure 12-5, data is being transmitted from node A to node B. Node B is at the farthest end of the bus (near the clock terminator); node A is at the nearest end (near or at the clock source (driver). The bus is fully loaded per Section 12.10.

In this analysis, we make node A's transmission of VAXBI data occur as early as possible and node B's clock that receives the data from node A is made as late as possible. This scenario provides the minimum hold time at node B.

digital™

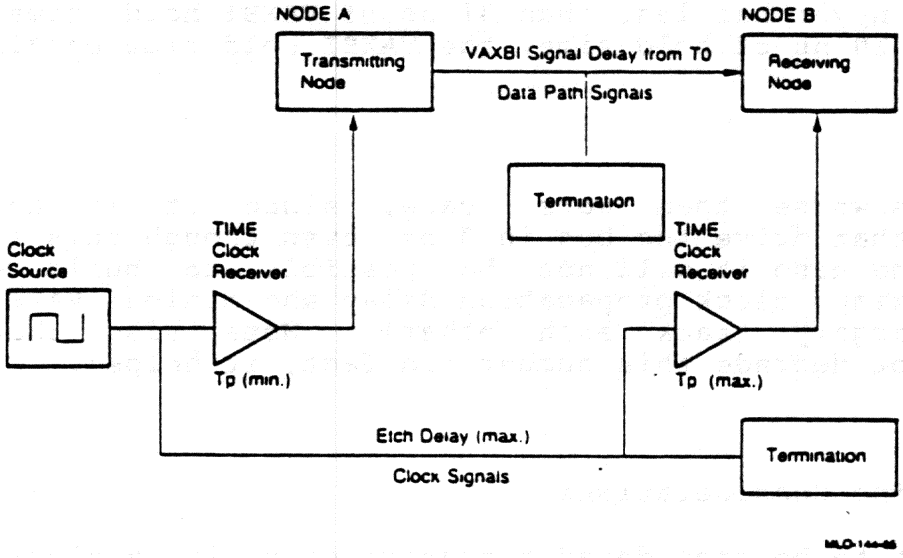


Figure 12-5: Worst-Case Hold Time Configuration

Clock Calculations

T0_nodeA = T0_ne

T0_nodeB = T0_fe
= T0_ne + VAXBI clock etch delay +
(Tp_receiver_fe - Tp_receiver_ne)

T0_nodeB(max.) = T0_ne + 14.5 + (8 - 1.5)
= T0_ne + 21.0 ns

Worst-Case Hold Time Calculations

Given that the minimum VAXBI signal delay is 0 ns, then VAXBI data may change as early as T0_ne.

Calculating the effect on node B's hold time yields:

T150_nodeB(max.) = T150_ne + 19 ns
VAXBI holdtime_nodeB(min.) = T0_ne - (T150_ne + 19.0 ns)
= 200 - (150 + 19) = 31 ns

NOTE: T0_ne is considered to be 200 ns rather than the more traditional 0 ns.

Therefore, a VAXBI node never has less than 31 ns of VAXBI hold time. Since the BIIC requires 20 ns of hold time, the VAXBI hold time margin is:

31 - 20 = 11 ns

Note that this number is worse than worst case, since it is not possible to get parts that drive the bus in 0 ns (even though this is the spec requirement) and also it will not be possible to build a system that has both maximum clock propagation delay and minimum VAXBI data delay (these two largely track each other). Note also that capacitance load does not degrade this number (in fact, it helps).

12.12.4 Transmission Line Considerations

The general rule is that to be considered a transmission line analysis problem, the etch runs must be electrically long compared to a quarter of the wavelength of the maximum frequency component of the signals. A more practical rule of thumb translates this rule to risetimes/falltimes less than the round-trip propagation time to be considered a transmission line problem for analysis. For a six-slot backplane, round-trip time is about 3 ns. For a fully loaded system (see Section 12.10), round-trip propagation time is about 25 ns. The assertions or deassertions of the VAXBI data path and synchronous control signals must be greater than the appropriate number in each case for valid lumped-capacitance analysis.

12.13 WORST-CASE VAXBI DATA PATH RESISTANCE

The calculations in Table 12-2 show the DC noise margin as seen by a receiver at the far end of a maximum configuration VAXBI bus with the driver at the near end. This noise margin is adversely affected by the total resistance of the signal path between the driving BIIC and the other end of the bus. The DC noise margin is calculated using maximum specifications of packaging component resistances.

Table 12-2: Maximum Data Path Resistance (in milliohms)

Item	Each	Total
BIIC package resistance	500	500
Module etch resistance	500	500
VAXBI connector resistance	15	15
Six-slot backplane (X6)	600	3600
Intercage flex cable (X5)	140	700

	Total	5315

Worst-Case Calculations Noise Margin

The voltage rise across the length of a signal path must be added to the driver output voltage. Considering a worst-case Iol of 18.6 mA for the driver, the voltage drop across the signal line is given by:

Vrise = 18.6 mA X 5.315 ohms = 98.6 mV

Therefore, the effective DC noise margin (low state) is:

DC NM(L) = 500 mv - 98.6 mV = 401.4 mV

CHAPTER 13

MECHANICAL AND POWER SPECIFICATION

The VAXBI bus is more than a protocol with associated timing and electrical parameters. Certain physical and electrical requirements must be met to achieve interchangeable units and easily configurable VAXBI-based systems. The interface between the VAXBI module and the VAXBI card cage is critical in meeting these goals.

This chapter defines the physical and electrical (DC power) requirements that VAXBI modules must meet. The chapter also defines the requirements that VAXBI cages must meet to be compatible with VAXBI modules.

- Section 13.1 applies to all VAXBI modules.
- Section 13.2 applies to VAXBI modules that contain BIICs.
- Section 13.3 is a general specification for VAXBI cages.
- Section 13.4 is a reference designator system for the VAXBI bus.
- Section 13.5 defines slot independence and some resulting configuration restrictions.
- Section 13.6 defines VAXBI termination requirements.

Sections 13.1 and 13.2 refer to the Digital T1999 Module Control Drawing Set. The latest revision of the module control drawing printset may contain information that supersedes the information in this chapter.

digital™

13.1 VAXBI MODULE GENERAL SPECIFICATION

A VAXBI module is a board that is 9.18" high, 8.0" deep, and at least 0.083" wide. A VAXBI module occupies one or more slots in a VAXBI cage. Most VAXBI modules occupy one VAXBI slot, but a VAXBI module with very high components may require more than one slot. A module is a VAXBI module only if it is designed to plug into a VAXBI cage.

A VAXBI slot is a place to put a VAXBI module: a slot is 9.18" high, 8.0" deep, and 0.8" wide. VAXBI slots are furnished with power and air.

A VAXBI node is a logical entity within the address space of a VAXBI. A VAXBI node may consist of one or more VAXBI modules or a mixture of VAXBI and non-VAXBI modules.

13.1.1 Physical Requirements

The four edges of a VAXBI module are referred to as the top, bottom, front, and connector edges. The connector edge of a VAXBI module has five groups of pads for contact with the zero insertion force (ZIF) connector. The five groups of pads are referred to as zones A through E. An area in the top-connector corner of a module is referred to as the VAXBI Corner (however, the VAXBI Corner exists only if the module has a VAXBI primary interface). (See Figure 13-1.)

The two sides of a VAXBI module are referred to as the component side and the solder side. It is intended that components be attached only to the component side of VAXBI modules.

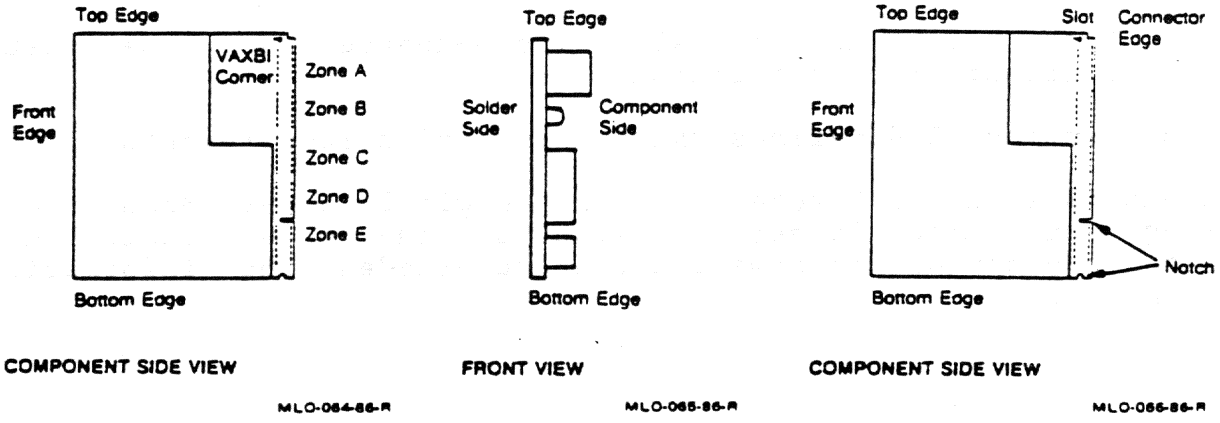


Figure 13-1: VAXBI Module Views

digital™

The overall physical dimensions of a VAXBI module, in inches, are:

Dimension	Minimum	Maximum
Height (top to bottom)	9.175	9.187
Depth (front to connector)	7.995	8.007
Width (within 0.65" of connector)	0.083	0.103
Lead projection (on solder side)	0	0.062
Component projection (on comp. side)	0	0.540

Maximum warpage (within 0.65" of connector) is 0.005 in./in.

The ZIF connector used on the VAXBI is keyed to ensure that VAXBI modules cannot be inserted upside down and to ensure that VAXBI modules are positively seated. The keying scheme used requires that a corner, a slot, and a notch be cut from the VAXBI module.

The corner is cut from the top-connector corner of the VAXBI module. The corner cut measures 0.200" by 0.095" min., 0.150" max.

The slot is cut near the top-connector corner. The slot is parallel to the top of the module, and is located 0.175" from the top edge. Near the front of the slot, the slot width is 0.094" wide, but the slot can be wider nearer to the connector edge. The connector end of the slot is located 0.375" from the connector edge of the module and must be parallel to the connector edge. The front end of the slot is nominally 0.660" from the connector edge. For ease of manufacturing, the slot may be formed as a tee, by drilling three 0.0470" radius holes and then using a router; the drill holes are centered at:

Distance from top edge	Distance from connector edge
0.175"	0.613"
0.128"	0.422"
0.222"	0.422"

One notch is cut near the bottom-connector corner. The sides of the notch, parallel to the connector edge, are located 0.220" and 0.400" from the connector edge. The top of the notch is located 9.03" from the top of the module.

Another notch is cut between zones D and E of the connector edge of the module. The center line of the notch is parallel to the top edge, and located 7.050" from the center line of the T-slot. The notch is 0.094" wide. The top of the notch is located 0.495" from the connector edge of the module.

The connector edge of the module has a 60-degree bevel to a depth of 0.03".

digital™

13.1.2 Border and ESD Requirements

Every VAXBI module has a 0.2" border strip along the top, front, and bottom edges. Component bodies and component leads must not be placed in this border strip to allow clearance for card guides. Furthermore, no printed circuit lines or pads are allowed within this border other than the ESD pads, so that modules are compatible with conductive card guides.

Two electrostatic discharge (ESD) pads are required within the border strip on every module. These ESD pads temporarily contact mating ESD clips on VAXBI card cages during insertion of a VAXBI module to discharge static before any power or signal pin contacts the connector. Two ESD pads are required to allow for two styles of VAXBI cage: one style with insertion from the front edge toward the connector edge, and one style with insertion from the top edge toward the bottom edge. Each ESD pad is (nominally) 0.140" wide by 0.300" long. One pad is on the bottom edge of the module, with its center 0.690" from the connector edge. The other pad is on the front edge of the module, with its center 0.690" from the bottom edge of the module. Both ESD pads are located on the component side of the module. These ESD pads are connected to the ground plane on each VAXBI module through a resistor whose nominal value is 1.0 Kohms.

A border strip that is 0.650" wide is required along the connector edge. Component bodies and component leads must not be placed in this border strip to allow clearance for the ZIF connector. Surface etch on both sides of the module within this area must be gold plated.

13.1.3 LED Requirements

Every VAXBI module must have two yellow LEDs to indicate whether that VAXBI module (or the VAXBI node of which that VAXBI module is a part) passed self-test. (Self-test requirements are defined in Chapter 11; see also Application Note 4.) One of the self-test LEDs is on the top of the module and the other is on the front. No other yellow LEDs are permitted on a VAXBI module.

The edge of the yellow LED on the front edge of the VAXBI module can be between 2.5" and 8.75" from the top edge of the module. The preferred location is 2.5" from the top edge. The lens of the yellow LED should be as close as practical to the 0.2" border strip on the front edge.

13.1.4 Replaceability Requirements

Cables must not connect directly to VAXBI modules. All I/O cables connect to VAXBI modules through the VAXBI connector. All interconnecting cables between the VAXBI modules that comprise a VAXBI node are also connected to those modules through the VAXBI connector.

VAXBI modules should not contain switches or other mechanical adjustments. VAXBI modules should not have pots, trimmers, or other electromechanical adjustments. These requirements eliminate error-prone on-site adjustments; factory-set adjustments and strapping options for the convenience of manufacturing test are permissible.

13.1.5 VAXBI Backplane Pins and Signals

The ZIF connector pins in zone A and zone B of the VAXBI module ar listed on the next page.

All VAXBI modules must have gold pads for all 300 pins on the ZIF connector, whether those pins are used or not. This is to prevent buildup of debris on the ZIF connector, which could occur if the connector made contact with unplated areas of VAXBI modules.

A VAXBI module design that does not have a BIIC must not drive or receive any VAXBI synchronous signals or VAXBI clocks: there can be no connection to any pin in the A or B zone of the ZIF connector other than to pins that supply DC voltages (labeled GND or voltage) and to the VAXBI asynchronous control signals. BI AC LO and BI DC LO may be driven through suitable drivers by VAXBI modules that do not have a BIIC, but they should not be otherwise loaded.

digital™

VAXBI Backplane Pins and Signals

Pin	Signal Name	Pin	Signal Name	Pin	Signal Name	Pin	Signal Name
A01	BI D29 L	A16	BI D31 L	A31	GND	A46	GND
A02	BI D28 L	A17	BI D30 L	A32	PASS THRU	A47	5VBB
A03	GND	A18	GND	A33	5VBB	A48	5VBB
A04	BI D27 L	A19	GND	A34	5VBB	A49	GND
A05	BI D25 L	A20	BI D26 L	A35	5VBB	A50	GND
A06	BI D23 L	A21	BI D22 L	A36	GND	A51	5VBB
A07	BI D21 L	A22	BI D20 L	A37	PASS THRU	A52	PASS THRU
A08	BI D19 L	A23	BI D18 L	A38	GND	A53	GND
A09	BI D15 L	A24	BI D17 L	A39	PASS THRU	A54	PASS THRU
A10	BI D24 L	A25	BI D14 L	A40	GND	A55	GND
A11	BI D13 L	A26	BI D12 L	A41	PASS THRU	A56	BI SPARE L
A12	BI D11 L	A27	BI D10 L	A42	GND	A57	GND
A13	+5.0V	A28	+5.0V	A43	+5.0V	A58	+5.0V
A14	BI D07 L	A29	+5.0V	A44	+5.0V	A59	GND
A15	BI D06 L	A30	BI D16 L	A45	BI D08 L	A60	BI D03 L
B01	BI D00 L	B16	BI D04 L	B31	BI D02 L	B46	BI D05 L
B02	BI P0 L	B17	BI D01 L	B32	GND	B47	BI ID3 H
B03	BI I1 L	B18	BI I2 L	B33	BI ID2 H	B48	GND
B04	BI CNF2 L	B19	BI I0 L	B34	GND	B49	BI ID1 H
B05	BI BSY L	B20	BI CNF1 L	B35	BI ID0 H	B50	+12.0V
B06	BI NO ARB L	B21	BI D09 L	B36	BI STF L	B51	BI BAD L
B07	BI I3 L	B22	BI CNF0 L	B37	-12.0V	B52	GND
B08	-5.2V	B23	-5.2V	B38	-2.0V	B53	-2.0V
B09	BI DC LO L	B24	-5.2V	B39	-2.0V	B54	BI RESET L
B10	GND	B25	BI ECL VCC H	B40	BI AC LO L	B55	GND
B11	GND	B26	BI TIME + H	B41	BI TIME - L	B56	GND
B12	GND	B27	BI PHASE + H	B42	BI PHASE - L	B57	GND
B13	GND	B28	GND	B43	GND	B58	GND
B14	MOD CANNOT USE	B29	GND	B44	GND	B59	MOD CANNOT USE
B15	MOD CANNOT USE	B30	MOD CANNOT USE	B45	MOD CANNOT USE	B60	MOD CANNOT USE

digital™

13.1.6 VAXBI Voltages and Currents

VAXBI-based systems supply power to VAXBI cages, which distribute power to VAXBI modules. Power is supplied through the VAXBI backplane to all VAXBI modules.

VAXBI modules that need voltage levels not supplied by the VAXBI backplane may use power cables attached in the I/O region of the ZIF connector.

13.1.6.1 Voltages Available - Bus rails are allocated through VAXBI cages for six voltages:

- +5.0V is available for general use.

  All VAXBI-based systems are required to supply +5V.

- 5VBB (volts battery backed up) is available only for use by dynamic RAM (DRAM) memories. 5VBB is separated from +5V to permit battery backup of RAM without the need for batteries to carry the full load of VAXBI cages.

  Systems that do not support battery backup for VAXBI DRAM memories are not required to supply a separate 5VBB. If a separate 5VBB supply is not used, then 5VBB should be connected to +5.0V on the VAXBI cage to permit the use of VAXBI modules that load 5VBB.

- +12.0V and -12.0V are intended for communication devices, such as RS232 drivers and receivers and RS423 drivers.

  All VAXBI-based systems are required to supply +12.0V and -12.0V; these voltages may be used by any VAXBI module.

- -5.2V and -2.0V are available for VAXBI modules that contain ECL devices. -5.2V is intended to supply ECL parts, while -2.0V is intended to supply ECL signal terminators. The purpose of -2.0V is to alleviate the heat and space otherwise required for resistor dividers on VAXBI modules that use ECL devices.

  VAXBI-based systems are not required to supply -5.2V or -2.0V, but if -5.2V is supplied then -2.0V must also be supplied. If these voltages are not supplied, then the inputs to the VAXBI cages should be left floating.

Four variations in voltages supplied are allowed in a VAXBI-based system:

- o No BBU voltage and no ECL voltages
- o No BBU voltage, but ECL voltages are available
- o BBU voltage available, but no ECL voltages
- o BBU and ECL voltages are all available

13.1.6.2 Specification of Current and Power for VAXBI Modules - Several methods are used in the computer industry to specify the current and power used by devices: some specifications are based on worst-case calculations, some on typical calculations, and some on measurements. For VAXBI modules, the specifications for current and power used should be based on calculations using the formulas below. The purpose in having a standard specification method is to achieve consistency. The standard method chosen will result in data that is less than worst-case but is more than would be measured with typical modules.

The standard current for a VAXBI module is based on the root-mean-squared sum of the typical and maximum currents specified for IC (integrated circuit) devices, plus nominal currents for passive devices. For an IC:

```
I_std = [ I_typ**2 + (I_max - I_typ)**2 ]**0.5
```

For example, an IC with a typical current of 400 mA and a maximum current of 800 mA would have a standard current of 565 mA.

The standard power for a VAXBI module is the sum of the products of the standard currents times the corresponding nominal voltages. For example, a VAXBI module that uses 4.0 amps of +5 and 100 mA (each) of +12 and -12 would have a standard power of 22.4 watts.

The following numbers should be used to calculate the standard power of a module with a VAXBI Corner with a BIIC. These numbers were calculated by using the typical and maximum currents for components in the Corner and then adding the dynamic power of the VAXBI drivers.

- o Standard power with clock source: 4.32 watts
- o Standard power without clock source: 3.54 watts

The recommended maximum standard power that a VAXBI module should dissipate is 40 watts. The recommended maximum standard currents that a VAXBI module should use are listed below.

Voltage Name	Max. Amps, VAXBI Module
Ground	8.0
+5.0V	8.0
5VBB	5.5
-5.2V	4.0
-2.0V	2.6
+12.0V	0.8
-12.0V	0.8

13.1.7 Worst-Case Current Limits

The absolute DC current limitations for a VAXBI module are given below. These limits apply to all VAXBI modules.

No current limits apply to all VAXBI cages. The appropriate current limits for a VAXBI cage are determined by the number of slots in that VAXBI cage, as well as other factors. Although desirable, VAXBI cages are not required to support all possible combinations of VAXBI modules. An example of the current limits for a typical 6-slot VAXBI cage is shown below.

Voltage Name	Max. Amps, VAXBI Module	Max. Amps, VAXBI Cage (example)
Ground	10.0	60.0
+5.0V	10.0	50.0
5VBB	7.0	30.0
-5.2V	5.0	15.0
-2.0V	3.3	10.0
+12.0V	1.0	3.0
-12.0V	1.0	3.0

digital™

The module limits are based on copper self-heating within printed circuits and the ZIF connector, and are based on the I-R drop budget defined in Section 13.1.8. Accordingly, the limits above represent worst-case currents under the following conditions: worst-case (end of life) components, operated at the worst-case temperature and worst-case voltage extremes, with worst-case configuration and application conditions.

No module may operate at all of the above limits, due to power dissipation limits. The total power used by a VAXBI module that operated at all of the limits above would be 132 watts, which vastly exceeds the cooling capacity intended for VAXBI modules.

VAXBI modules must conform to each of the above absolute current limits and must also conform to the power limits stated in Section 13.1.9.1.

No minimum current for VAXBI modules or VAXBI cages is defined, although some power supplies used for some VAXBI-based systems may impose minimum loads to remain in regulation.

13.1.8 I-R Drop and Voltage Tolerance Budgets

The maximum allowed voltage drop through a VAXBI cage for each voltage and the maximum recommended voltage drop across a VAXBI module are:

Voltage Name	Max. Drop, mV, VAXBI Cage	Max. Drop, mV, VAXBI Module
Ground	30	15
+5.0V	50	20
5VBB	50	20
-5.2V	50	20
-2.0V	35	15
+12.0V	65	15
-12.0V	65	15

These I-R drop limits, like the current limits listed in Section 13.1.7, are worst-case limits: worst-case (end of life) VAXBI cages and VAXBI modules, operated at the worst-case temperature and worst-case voltage extremes, with worst-case configuration and application conditions.

digital™

The reference ground for measuring voltages in a VAXBI cage is the ground connection to the VAXBI cage: the point at which power return cables connect to the VAXBI cage. The minimum and maximum voltages, measured at the points where the power distribution cables connect to the VAXBI cage, and measured with respect to the reference ground, are listed below. In all VAXBI systems with more than one VAXBI cage, a direct connection between ground planes of adjacent backplanes is required with a resistance of less than 10 milliohms.

These minimum and maximum voltages are based on achieving +/-7.5% regulation on -2.0V, and +/-5% on all other voltages, measured across any component on any VAXBI module. These limits, also based on the worst-case currents and I-R drops stated previously, represent the total allowable tolerance: the total static tolerance plus ripple.

Voltage Name	Input Voltage at VAXBI Cage Max.	Min.
+5.0V	5.250	4.865
5VBB	5.250	4.865
-5.2V	-5.46	-5.055
-2.0V	-2.25	-2.045
+12.0V	12.6	11.525
-12.0V	-12.6	-11.525

13.1.9 Power Dissipation and Cooling

The power dissipation limits of VAXBI slots are based on a standard cooling scheme using forced air flow. The limit applies to VAXBI slots, rather than to VAXBI modules: a module that is more than one slot wide is allowed to dissipate power in excess of the stated module power limit.

13.1.9.1 Power Dissipation Limits - A VAXBI module that is one slot wide may not dissipate more than 50 watts worst-case. A VAXBI module that is n slots wide may not dissipate more than 50*n watts worst-case.

Modules that dissipate more than 50 watts may require on-board air deflectors or extraordinary heat sinks to guarantee adequate component cooling, and to conform to the required maximum air temperature of 60 degrees C at the outlet side of the VAXBI cage.

digital™

13.1.9.2 Air Flow - VAXBI cages and modules are designed for forced air cooling with air flow parallel to the backplane. Air flow may be in either direction: from zone A of the VAXBI cage to zone E or from zone E to zone A.

Standard VAXBI cage cooling means at least 12.8 SCFM (standard cubic feet per minute) air flow across each slot, within systems that may be operated with ambient temperatures up to 50 degrees C inlet air temperature. A total supplied air flow of 100 SCFM through each cage against a pressure loss of 0.17 in. of water will ensure the minimum air flow per slot. The system designer must ensure that components receive adequate air flow (the BIIC, for example, requires 200 linear feet per minute [LFM]).

The system designer must ensure that the temperature rise is negligible between external ambient temperature and the temperature at the inlet of the VAXBI cage for systems that are to be operated with ambient temperatures up to 50 degrees C. The module designer must ensure that under these conditions the temperature rise is no more than 10 degrees C across the module. Thus, 50 degrees C ambient conditions for the system translates to a maximum air temperature of 60 degrees C at the outlet side of the VAXBI cage.

It is suggested that part of the node designer's analysis include "thermal profile" simulation modeling of the module and components during the design phase. The simulation should take into account such parameters as component power dissipation, air flow, chip placement, and heatsink geometries.

If a thermal modeling tool is not available, the node designer must take the following more conservative approach to the design. Since some integrated circuits on VAXBI modules can potentially see a 60 degree C ambient temperature condition within the VAXBI cage, the designer must ensure that all integrated circuits on the module will still operate under this condition, at 200 LFM air flow. In addition, during design verification of prototype hardware, the node designer must verify that there is never more than the maximum 10 degree C rise in ambient temperature across the module.

13.2 VAXBI MODULE WITH A BIIC

Most VAXBI nodes include one VAXBI module that has a BIIC in the VAXBI Corner of that module. Section 13.1 applies to all VAXBI modules, while this section describes additional requirements for VAXBI modules that have BIICs.

A VAXBI module that has a BIIC may use all of the VAXBI signals. However, the BIIC is the only direct connection allowed for the majority of VAXBI signals: the user logic, in the user-configurable area outside of the VAXBI Corner, has access to a set of signals on the "virtual connector." This virtual connector serves as the physical and logical boundary between the VAXBI Corner and the user logic.

The BIIC and associated logic in the VAXBI Corner rely on controlled impedance and minimized crosstalk. A standard layout of the components and printed circuits in the VAXBI Corner and a layup of the standard 10-layer printed-circuit board are included in the drawings. All VAXBI modules with a BIIC must have the same R, L, and C characteristics for all signals as the standard design. VAXBI modules must have no more crosstalk between signals than the standard design. None of the active components in the VAXBI Corner may be socketed, unless the approved pin sockets are used.

VAXBI module designers must copy the VAXBI Corner design as referenced in the module control drawings.

Ground layers in the VAXBI Corner should not be changed, since this would affect R-L-C parameters on signal layers. Power layers in the VAXBI Corner may be changed, if necessary, to reallocate the copper from unused voltages to heavily used voltages. The layers on the standard 10-layer VAXBI module are used as follows:

Layer 1 - light copper, cap layer
Layer 2 - 1 oz. copper, signal layer
Layer 3 - 1 oz. copper, signal layer
Layer 4 - 2 oz. copper, ground plane
Layer 5 - 2 oz. copper, power plane
Layer 6 - 2 oz. copper, power plane
Layer 7 - 2 oz. copper, ground plane
Layer 8 - 1 oz. copper, signal layer
Layer 9 - 1 oz. copper, signal layer
Layer 10 - light copper, cap layer

13.2.1 VAXBI Corner Signal Locations

The VAXBI Corner contains the virtual connector, an L-shaped patter of 98 signal connections. Each signal appears on a via connection and is available on all signal layers of the standard VAXBI module as shown in the module control drawings.

digital™

Figure 13-2 shows the approximate location of signals in the VAXBI Corner.

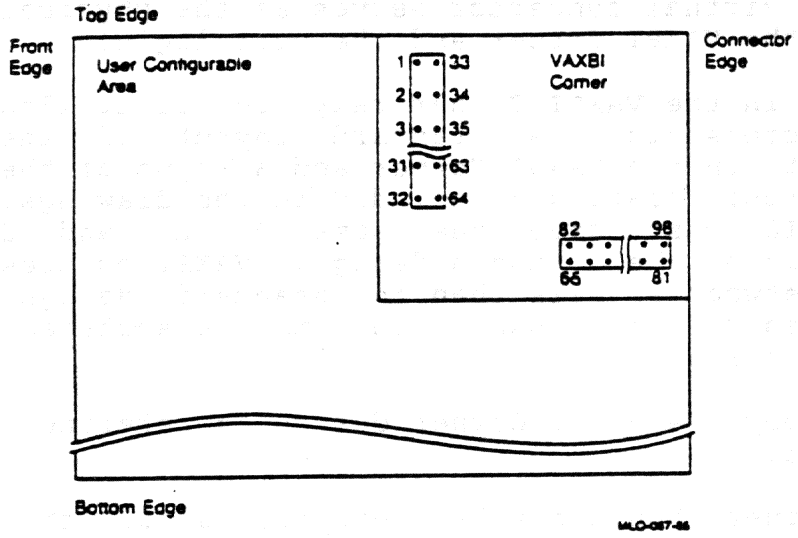


Figure 13-2: VAXBI Corner Signal Locations (component side view)

13.2.2 VAXBI Virtual Connector Signals

The signals available on the virtual connector, at the boundary between the VAXBI Corner and the user-configurable area, are listed on the following pages.

The layer listed is the layer at which a printed circuit connects to the virtual connector from something within the VAXBI Corner. Normally, pads will exist at all layers for each virtual pin in the virtual connector, but these pads may be omitted (to reduce capacitance) for all but the listed layer.

The BI AC LO L and BI DC LO L signals are provided to allow designs to drive these VAXBI signals. In no case may any design monitor these lines directly off the VAXBI bus. The BCI AC LO L and BCI DC LO L signals provided by the BIIC are the only signals that can be received and monitored by designs for power status.

The BCI PASS THRU signals are intrabackplane only. They are not carried in extension cables from one cage to another.

The BCI CK DIS L signal is used by clock source designs that need to know when the VAXBI clock source is enabled (slot K1J1 only) or disabled (all other slots).

The BCI STPASS L signal, when asserted by the user interface, lights the self-test LEDs.

digital™

Pin	Signal Name	Routed From	Layer	Notes
U01	5VBB	ZIF A-35	6	
U02	PASS THRU	ZIF A-37	8	Reserved for Digital
U03	PASS THRU	ZIF A-39	8	Reserved for Digital
U04	PASS THRU	ZIF A-41	8	Reserved for Digital
U05	BCI STPASS L	LED, R997	9	
U06	BCI SC1 L	BIIC, H-2	9	< 100 pF capacitance
U07	BCI D30 H	BIIC, J-2	2	< 100 pF capacitance
U08	BCI D28 H	BIIC, K-2	2	< 100 pF capacitance
U09	BCI D31 H	BIIC, K-1	3	< 100 pF capacitance
U10	BCI D24 H	BIIC, M-2	8	< 100 pF capacitance
U11	BCI D25 H	BIIC, N-1	3	< 100 pF capacitance
U12	BCI D26 H	BIIC, M-1	2	< 100 pF capacitance
U13	BCI D20 H	BIIC, P-2	3	< 100 pF capacitance
U14	N/C (E999-N2)	BIIC, N-2	2	Reserved for Digital
U15	BCI D22 H	BIIC, N-3	2	< 100 pF capacitance
U16	BCI D18 H	BIIC, N-4	2	< 100 pF capacitance
U17	BCI D15 H	BIIC, N-5	2	< 100 pF capacitance
U18	BCI D12 H	BIIC, N-6	2	< 100 pF capacitance
U19	BCI D09 H	BIIC, N-7	2	< 100 pF capacitance
U20	BCI D06 H	BIIC, P-9	3	< 100 pF capacitance
U21	BCI D13 H	BIIC, M-6	9	< 100 pF capacitance
U22	BCI D05 H	BIIC, N-9	2	< 100 pF capacitance
U23	BCI D01 H	BIIC, N-10	2	< 100 pF capacitance
U24	BI AC LO L	BIIC, P-14	3	User interface driver only
U25	BCI D04 H	BIIC, M-9	8	< 100 pF capacitance
U26	BCI I0 H	BIIC, M-12	8	< 100 pF capacitance
U27	BCI CLE H	BIIC, M-13	9	< 100 pF capacitance
U28	BCI RAK L	BIIC, L-13	9	< 100 pF capacitance
U29	BCI I3 H	BIIC, N-11	2	< 100 pF capacitance
U30	BCI EV0 L	BIIC, K-13	9	< 100 pF capacitance
U31	BCI EV3 L	BIIC, K-14	9	< 100 pF capacitance
U32	BCI AC LO L	BIIC, N-13	2	< 100 pF capacitance
U33	PASS THRU	ZIF A-32	8	Reserved for Digital
U34	5VBB	ZIF A-51	6	
U35	PASS THRU	ZIF A-52	8	Reserved for Digital
U36	PASS THRU	ZIF A-54	8	Reserved for Digital
U37	BI SPARE L	ZIF A-56	8	Reserved for Digital
U38	BCI SC2 L	BIIC, H-1	2	< 100 pF capacitance
U39	BCI SEL L	BIIC, H-3	9	< 100 pF capacitance
U40	BCI SC0 L	BIIC, J-1	9	< 100 pF capacitance
U41	BCI D29 H	BIIC, L-1	3	< 100 pF capacitance
U42	BCI D27 H	BIIC, L-2	2	< 100 pF capacitance
U43	BCI D23 H	BIIC, M-3	9	< 100 pF capacitance
U44	BCI D21 H	BIIC, M-4	8	< 100 pF capacitance
U45	BCI D17 H	BIIC, M-5	9	< 100 pF capacitance
U46	BCI D19 H	BIIC, P-3	3	< 100 pF capacitance
U47	BCI D16 H	BIIC, P-4	3	< 100 pF capacitance
U48	BCI D14 H	BIIC, P-5	3	< 100 pF capacitance
U49	BCI D11 H	BIIC, P-6	3	< 100 pF capacitance

Pin	Signal Name	Routed From	Layer	Notes
U50	BCI D10 H	BIIC, P-7	3	< 100 pF capacitanc
U51	BCI D07 H	BIIC, P-8	3	< 100 pF capacitanc
U52	BCI D08 H	BIIC, N-8	2	< 100 pF capacitanc
U53	BI DC LO L	BIIC, H-13	8	User interface driver onl
U54	Open Via		-	Reserved for Digita
U55	BCI D03 H	BIIC, P-10	3	< 100 pF capacitanc
U56	BCI D02 H	BIIC, P-11	3	< 100 pF capacitanc
U57	N/C (E999-B13)	BIIC, B-13	9	Reserved for Digita
U58	BCI EV2 L	BIIC, J-12	9	< 100 pF capacitanc
U59	BCI NXT L	BIIC, M-14	9	< 100 pF capacitanc
U60	BCI EV1 L	BIIC, L-14	9	< 100 pF capacitanc
U61	BCI D00 H	BIIC, P-12	2	< 100 pF capacitanc
U62	BCI I1 H	BIIC, N-12	2	< 100 pF capacitanc
U63	BCI I2 H	BIIC, P-13	2	< 100 pF capacitanc
U64	BCI P0 H	BIIC, N-14	3	< 100 pF capacitanc
U65	BCI DC LO L	BIIC, J-14	9	< 100 pF capacitanc
U66	GND	ZIF (GND)	4/7	Lead hole for C10
U67	BCI INT5 L	BIIC, F-14	3	
U68	BCI INT4 L	BIIC, E-14	3	
U69	BCI RQ1 L	BIIC, D-14	3	
U70	BI BAD L	ZIF B-51	8	
U71	BCI INT6 L	BIIC, G-12	9	
U72	BCI SDE L	BIIC, H-14	9	< 100 pF capacitanc
U73	BCI INT7 L	BIIC, G-13	9	
U74	-2.0V	ZIF (B-38,39,53)	5	Lead hole for C10
U75	BCI PHASE L	Clk Rec Pin 8	3	See Application
U76	BCI TIME H	Clk Rec Pin 1	9	Note 6 for more
U77	BCI TIME L	Clk Rec Pin 15	3	information.
U78	BCI PHASE H	Clk Rec Pin 10	3	
U79	BI ID3 H	ZIF B-47	9	
U80	BI ID1 H	ZIF B-49	9	
U81	BCI CK DIS L	R994,1	2	Disabled driver sensin
U82	BCI EV4 L	BIIC, J-13	9	< 100 pF capacitanc
U83	BCI RS1 L	BIIC, E-13	3	
U84	BCI MAB L	BIIC, F-13	3	
U85	+12.0V	ZIF B-50	8	
U86	BI STF L	ZIF B-36	8	
U87	-12.0V	ZIF B-37	8	
U88	Open Via		-	Reserved for Digital
U89	Open Via		-	Reserved for Digit
U90	Open Via		-	Reserved for Digita
U91	BCI MDE L	BIIC, G-14	9	< 100 pF capacitance
U92	BCI RQ0 L	BIIC, F-12	9	
U93	BCI RS0 L	BIIC, C-14	9	
U94	BI RESET L	BI ZIF B-54	8	
U95	N/C (E998-14)	Clk Rec Pin 14	8	Reserved for Digital
U96	N/C (E998-13)	Clk Rec Pin 13	3	Reserved for Digital
U97	BI ID2 H	ZIF B-33	9	
U98	BI ID0 H	ZIF B-35	9	

digital™

13.2.3 VAXBI Corner Parts List

The components in the VAXBI Corner of VAXBI modules that have a BIIC are listed below, along with their reference designators. Unless otherwise noted, all these components must be installed.

The component nomenclature is shown for reference only. There is no requirement of the VAXBI designer to maintain the reference designators listed on the completed layout of a VAXBI module. Component designations can be specified on an option-specific basis.

Reference Designator	Description	Notes
C987	0.047 uF, 50V, cap.	+5V
C988	0.047 uF, 50V, cap.	+5V
C989	0.047 uF, 50V, cap.	+5V
C990	0.047 uF, 50V, cap.	+5V
C991	10.0 uF, 35V, cap.	+5V; Note 5
C992	10.0 uF, 35V, cap.	+5V; Note 5
C999	0.01 uF, 50V, cap.	BIIC VBB
C996	10.0 uF, 35V, cap.	-12V; Notes 1, 5
C997	10.0 uF, 35V, cap.	-5.2V; Notes 1, 5
C995	10.0 uF, 35V, cap.	-2V; Notes 1, 5
C994	10.0 uF, 35V, cap.	5VBB; Notes 1, 5
C993	10.0 uF, 35V, cap.	+12V; Notes 1, 5
C998	0.047 uF, 50V, cap.	ECL Vcc; Note 2
E999	78732 BIIC	
E998	78702 BI Clock Receiver	
E997	78701 BI Clock Driver	Note 2
Y999	40 MHz crystal oscillator	Note 2
D998	Yellow LED	Self-test passed; Note 3
R998	25.5 Kohm, 1/4W, 1%	BIIC Vcc reference; Note 4
R999	3.83 Kohm, 1/4W, 1%	BIIC ground reference; Note 4
Z999	1.0 Kohm, 1/8W, 5% SIP	ID pullups; may be out of the corner
R995	1.0 Kohm, 1/4W, 5%	BCI DC LO pulldown
R996	1.0 Kohm, 1/4W, 5%	ESD pad discharge resistor
R997	270 ohm, 1/4W, 5%	STPASS LED; Note 3
R994	470 ohm, 1/4W, 5%	Clock disable sense; Note 2

NOTES

1. Capacitors for unused voltages can be omitted.

2. The part can be omitted in VAXBI modules that never drive the clock signals.

3. D998, R997, and D999 (which is not in the VAXBI Corner) are required on all VAXBI modules.

4. The maximum temperature coefficient is 100 PPM/degrees C.

5. 8.0 uF, 25V capacitors can also be used.

The components of the VAXBI Corner use only +5.0V. The maximum power consumption in watts is:

E999	BIIC	4.25
E998	Clock Receiver	.10
Z999	ID pullup	.02 (if used)
		4.37 Total for non-clock source nodes

In addition, for clock source nodes:

Total from above		4.37
E997	Clock Driver	.70
Y999	Oscillator	.26
		5.33 Total as clock source node

digital™

13.3 VAXBI CARD CAGE REQUIREMENTS

A VAXBI card cage generally consists of a backplane, connectors between the backplane and the VAXBI modules, card guides, and some structural members. A cam actuator is required for each ZIF connector to open and close the connector.

13.3.1 General Requirements

Modules can be inserted and removed in two ways, depending on the design style of the VAXBI cage. With the backplane and the modules arranged vertically, these styles use either top entry or front entry. (A third design style, bottom entry, is not possible because the ZIF connector is open at the top but is solid at the bottom.)

The design style of the VAXBI cage also determines the cam actuator design. Some VAXBI cages may use a short cam actuator located adjacent to the ZIF connector, while other VAXBI cages may use a remote cam actuator located approximately 8" away from the ZIF connector in a line perpendicular to the backplane.

All VAXBI modules must be designed to be used in three of the four possible VAXBI cage design styles:

- Front entry, remote cam actuator
- Front entry, short cam actuator
- Top entry, short cam actuator

The fourth design style (top entry, remote cam actuator) is not possible, since the remote cam actuator linkage must be above the VAXBI modules and would, therefore, interfere with top entry. Designing VAXBI modules for use with the first and third styles of VAXBI cage imposes many of the requirements defined in Section 13.1. Designing VAXBI modules for use with the second style of VAXBI cage does not impose any additional requirements.

VAXBI cages can be designed to accommodate different numbers of VAXBI modules. All VAXBI cages must have at least 6 slots, and no VAXBI cage can have more than 36 slots.

All VAXBI cages include a backplane with the 300-pin ZIF connectors for the VAXBI modules. Some VAXBI cages may have an overall backplane, which connects to all 300 pins of each slot. Every VAXBI cage will have a backplane that at least covers zones A and B of the slots: the A/B section includes all the VAXBI (bused) signals and the standard voltage rails.

13.3.2 Orientation

All VAXBI modules are required to operate in VAXBI cages in any of the four allowed orientations. One orientation is preferred, and the other three are allowed.

There are 24 possible orientations of a VAXBI cage: the component side of the VAXBI modules could face up, down, left, right, front, or back; for each of those choices, there are four choices for the orientation of the backplane. These 24 choices reduce to four, due to the following considerations:

- VAXBI cages and VAXBI modules must be designed so that a self-test LED is visible for maintenance purposes (although it may be necessary to open an access door with some styles of cabinet).

- VAXBI modules are not required to operate with the components hanging down from the VAXBI modules. Some VAXBI modules may have socketed components, and it is best not to defy gravity.

- When VAXBI modules are mounted vertically, components will be on the right (as opposed to the left) side of modules. Although there is no technical basis for doing so, this is a Digital convention.

Although one of the four allowed orientations is referred to as "preferred," this preference has no technical basis but has been made with the assumption that it would be the most common orientation of the four allowed orientations. The preferred orientation, now established, is reinforced by the requirement that all labels on VAXBI cages, such as the VAXBI node ID plugs, be readable when the cage is in the preferred orientation.

13.3.2.1 VAXBI Cage Orientation A (Preferred) - In orientation A the backplane and the VAXBI modules are vertical. The backplane is behind the modules, and the components are on the right side of the modules.

The preferred card cage to be used with this orientation is the remote cam actuator version, which allows module insertion and removal from the front of the system.

Figure 13-3 shows a 6-slot card cage as an example.

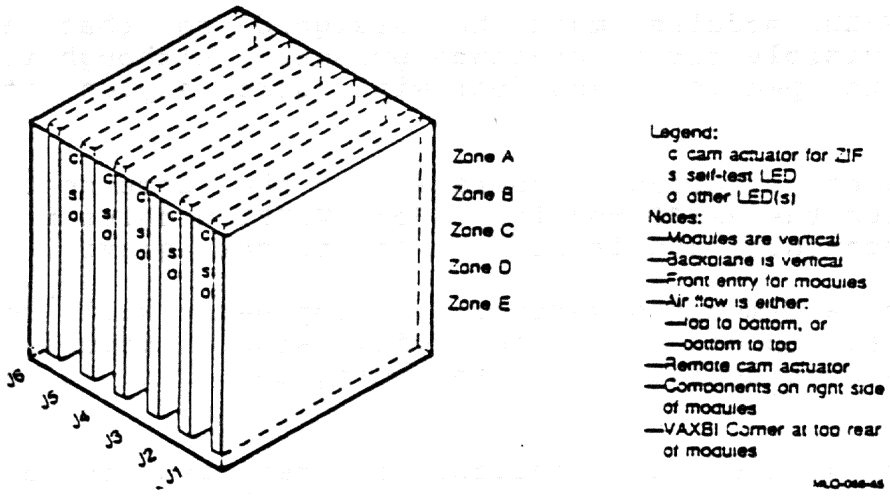


Figure 13-3: Orientation A (front view)

digital™

13.3.2.2 VAXBI Cage Orientation B (Allowed) - In orientation B the backplane is horizontal and the VAXBI modules are vertical. The backplane is below the modules, and the components are on the right side of the modules.

The preferred VAXBI cage to be used with this orientation is the remote cam actuator version, which allows module insertion and removal from the top of the system. The short cam actuator VAXBI cage could also be used with this orientation in some system designs.

Figure 13-4 shows a 6-slot card cage as an example.

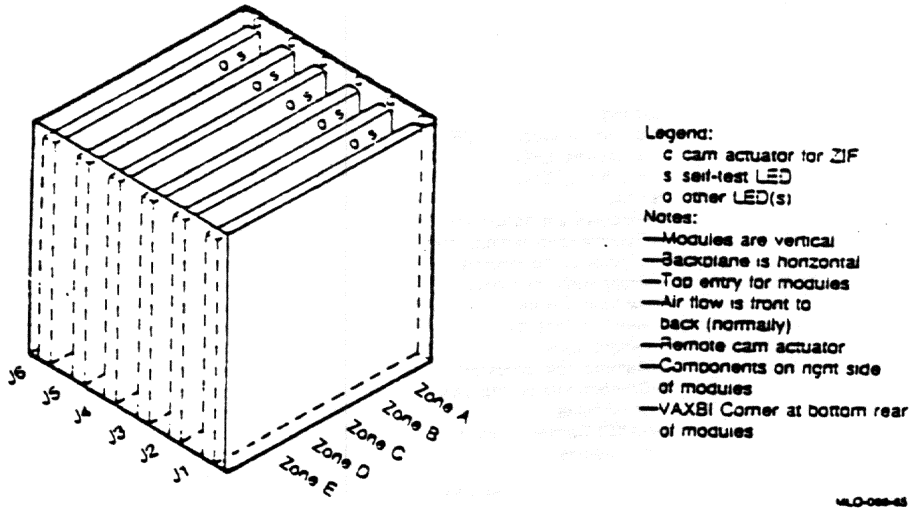


Figure 13-4: Orientation B (front view)

digital™

13.3.2.3 VAXBI Cage Orientation C (Allowed) - In orientation C the backplane is vertical and the VAXBI modules are horizontal. The backplane is behind the modules, and the components are on the top side of the modules.

The preferred VAXBI cage to be used with this orientation is the remote cam actuator version, which allows module insertion and removal from the front of the system.

Figure 13-5 shows a 6-slot card cage as an example.

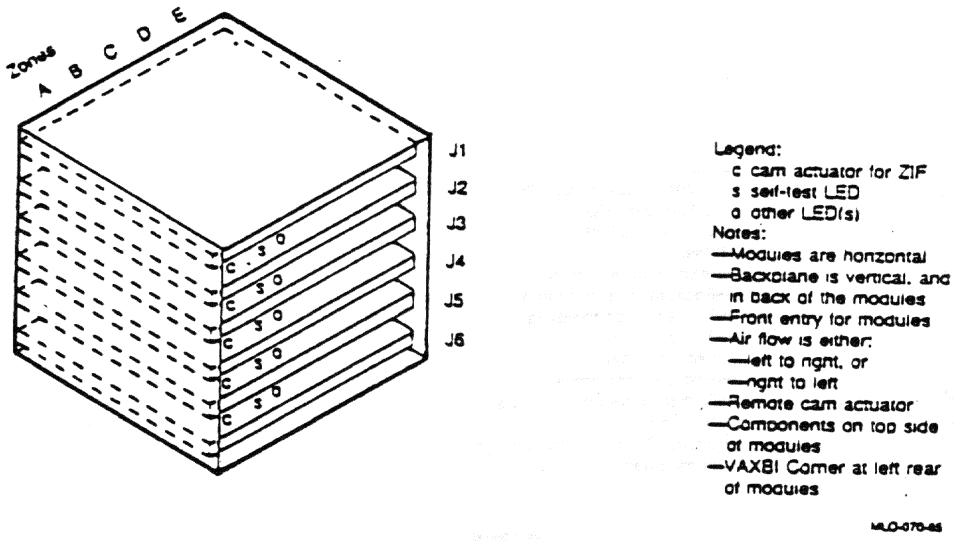
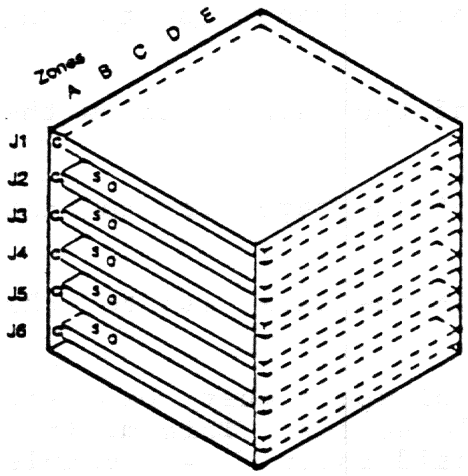


Figure 13-5: Orientation C (front view)

13.3.2.4 VAXBI Cage Orientation D (Allowed) - In orientation D the backplane and the VAXBI modules are horizontal. The backplane is on the left of the modules, and the components are on the top side of the modules.

The preferred VAXBI cage to be used with this orientation is the short cam actuator version, which allows module insertion and removal from the front of the system.

Figure 13-6 shows a 6-slot card cage as an example.



Legend:
c cam actuator for ZIF
s self-test LED
o other LED(s)
Notes:
—Modules are horizontal
—Backplane is vertical, and left of the modules
—Front entry for modules
—Air flow is front to back (normally)
—Short cam actuator
—Components on top side of modules
—VAXBI Corner at left front of modules

MLO-071-46

Figure 13-6: Orientation D (front view)

digital™

13.3.3 Cable Access

All VAXBI cables connect through the backplane side of the VAXBI cage. Installation of a VAXBI node that has I/O signals requires access to the backplane side of the cage to attach I/O cables. Installation of a node that consists of multiple VAXBI modules also requires access to the backplane side of the cage to attach intermodule cables. Installation of another VAXBI cage, to expand a system, also requires access to the backplane side of the VAXBI cage for attachment of the intercage VAXBI cables and the power wiring harness, and to relocate the clock terminator. Access to the backplane side of the cage is also required (rarely) to change node ID plugs.

- For orientation A, the backplane side of the cage is to the rear of the cage.
- For orientation B, the backplane side of the cage is below the cage.
- For orientation C, the backplane side of the cage is to the rear of the cage.
- For orientation D, the backplane side of the cage is to the left of the cage.

For any orientation of the cage, at least 1.5" of space must be left on the backplane side of the cage to allow cable access. This minimum cable access distance requires careful consideration by module designers, particularly when coaxial I/O cables are required. Leaving more than 1.5" for cable access is desirable for ease of access and cable management.

digital™

13.4 VAXBI REFERENCE DESIGNATOR SYSTEM

A reference designator system maps logical locations of entities into physical locations of entities.

The reference designator system shown in the following sections is based on a 6-slot card cage. Cages with more than six slots can use the same system, but the maximum numbers will change.

13.4.1 Vertical Module Mounting

In an assembly of VAXBI cages, the location of a VAXBI module is KkJj, which means the j-th VAXBI slot of the k-th VAXBI cage of the VAXBI. With the mechanical and electrical constraints imposed by 6-slot VAXBI cages:

k1-k6 Indicate the VAXBI cages (maximum of six VAXBI cages).

J1-J6 Identify the six VAXBI module connectors.

With orientation A or B, KkJj translates into the physical location of a VAXBI module, as seen from the front of the system that houses the VAXBI assembly, as shown in Figure 13-7.

If a VAXBI consists of multiple rows of VAXBI cages (whether physically adjacent or not), the cages must be numbered consecutively throughout that VAXBI. K1 is the cage that contains the clock source, and Kn is the cage that is electrically farthest from K1.

Although not required, it is recommended that the slot designators be printed on the module side of the card cages so that the designators can be read when the cage is in orientation A.

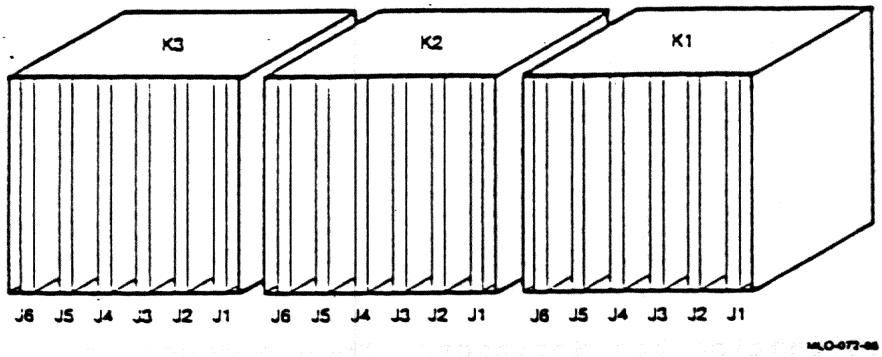


Figure 13-7: Reference Designation for Vertical Module Mounting

digital™

13.4.2 Horizontal Module Mounting

With orientation C or D, KkJj translates into the physical location of a VAXBI module, as seen from the front of the system that houses the VAXBI assembly, as shown in Figure 13-8.

If a VAXBI consists of multiple rows of VAXBI cages (whether physically adjacent or not), the cages must be numbered consecutively throughout that VAXBI. K1 is the cage that contains the clock source, and Kn is the cage that is electrically farthest from K1.

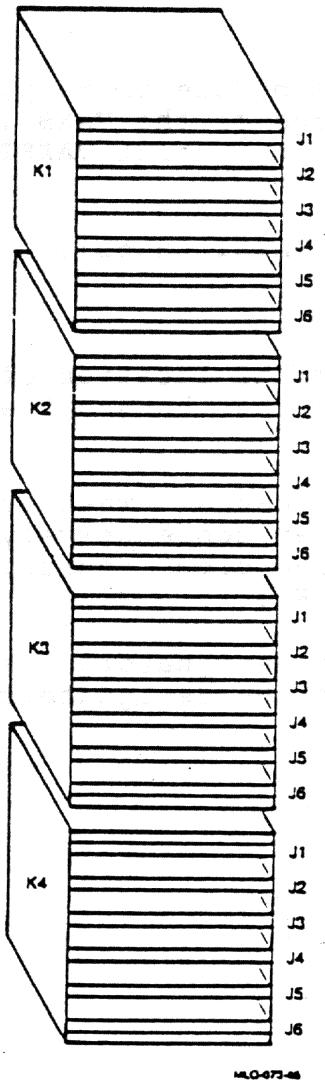


Figure 13-8: Reference Designation for Horizontal Module Mounting

digital™

13.4.3 Connector Zones

In an assembly of VAXBI cages, the location of a backplane pin associated with a module is KkJjzn. KkJj has the meaning described previously, and zn means:

- z — Identifies the pin zone. Pin zones A and B are used for VAXBI signals and power, and zones C, D, and E are used for I/O and intermodule interconnect.
- n — Identifies the 60 pins (1-60) within a zone.

With orientation A of a VAXBI cage, Jjz translates into physical locations of VAXBI zones and pins, as seen from the backplane side of the cage, as shown in Figure 13-9.

Although not required, it is recommended that the slot and zone designators be printed on the backplane side of the card cages so that the designators can be read when the cage is in orientation A.

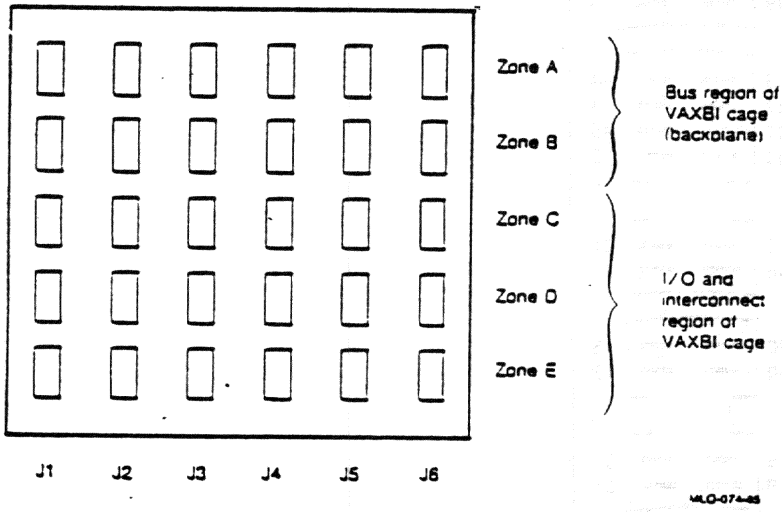


Figure 13-9: Reference Designation for Connector Zones

digital™

13.4.4 Connector Pins

Pins 16-30 and 46-60 are on the component side of the module, while pins 1-15 and 31-45 are on the solder side of the module. Within any group of pins (1-15, 16-30, 31-45, or 46-60), low-numbered pins are above high-numbered pins.

As seen from the backplane side of a VAXBI cage in orientation A, with the normal backplane over the zone A/B area and with no backplane covering the zone C/D/E area, the pin numbering scheme is as shown in Figure 13-10.

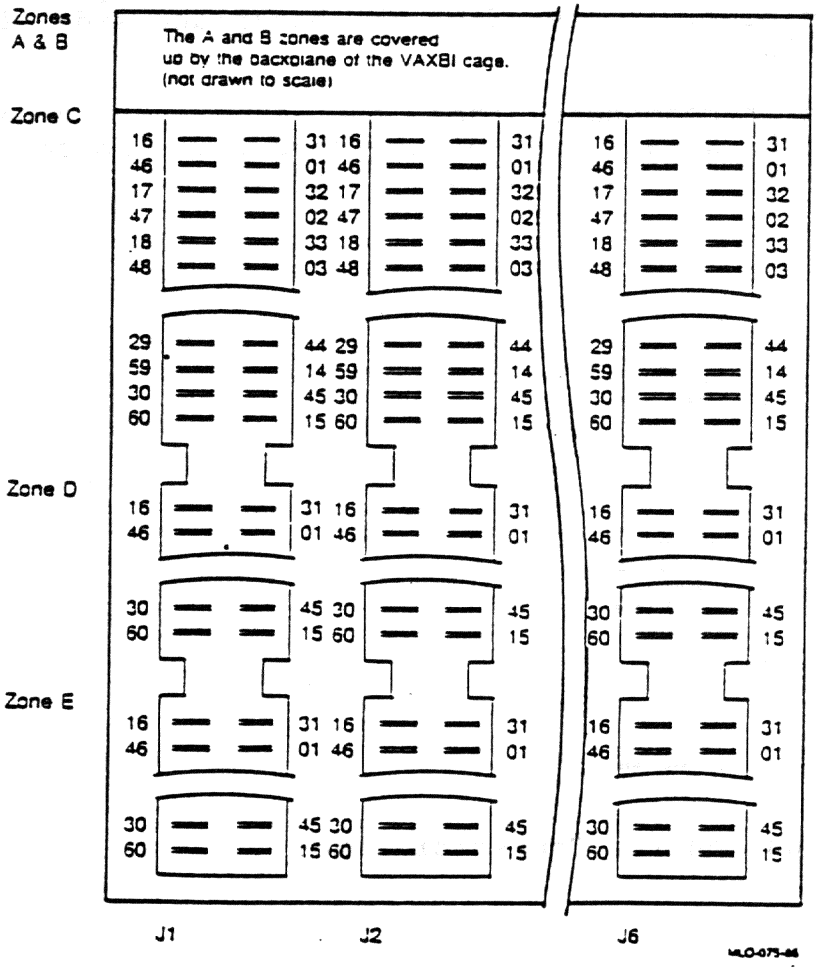


Figure 13-10: Reference Designation for Connector Pins

13.4.5 Header Pins

When transition headers are superimposed on the ZIF connector in the I/O section of the backplane, the pin numbering scheme changes.

Figure 13-11 shows the pin numbering for a commonly used transition header. The view is from the backplane side of a cage in orientation A, with the normal backplane over the zone A/B area and with no backplane covering the zone C/D/E area, and with a transition header connected to the J1 connector.

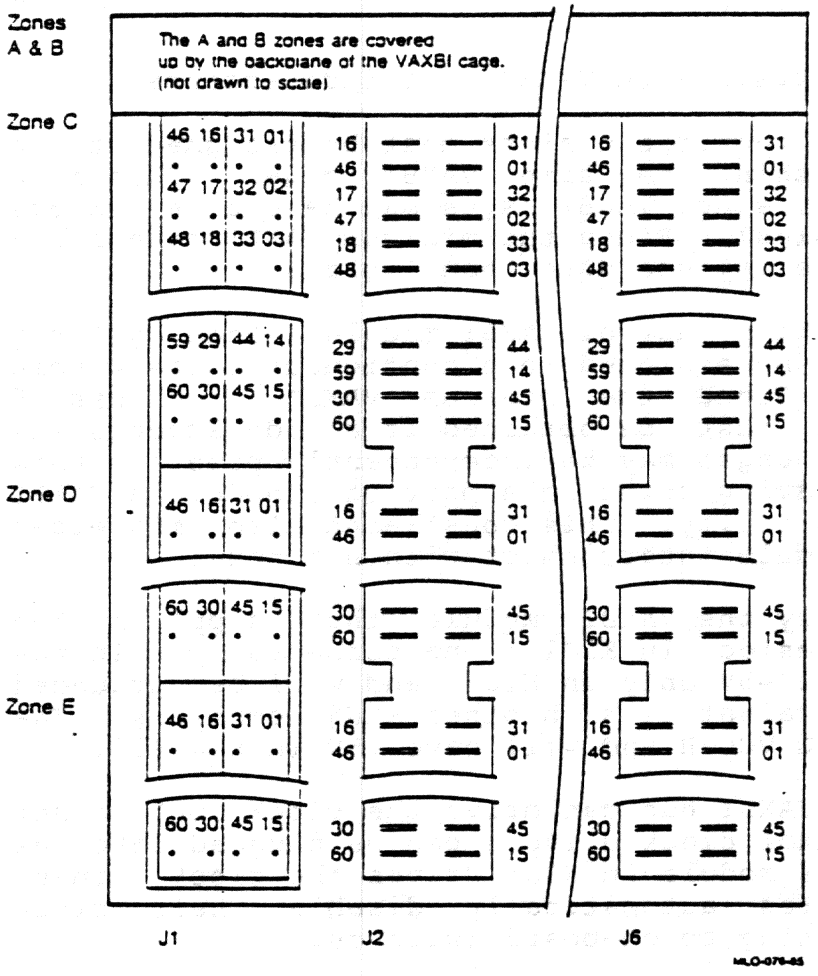


Figure 13-11: Reference Designation for Header Pins

13.5 VAXBI SLOT INDEPENDENCE

The VAXBI bus is designed for slot independence. VAXBI slot independence means that:

- No fixed relationship exists between the performance of a node and the slots used by the node.
- No fixed relationship exists between a node's address and the slots used by the node.
- No difference in power or current limits is imposed by different slots.

There are, however, some constraints on VAXBI slot independence:

- Any VAXBI module that requires n slots must be assigned to n contiguous slots within the same VAXBI cage. This is a mechanical limitation, since VAXBI cages may have side members (parallel to the modules). Since the minimum VAXBI cage has 6 slots, no VAXBI module may use more than 6 slots.
- Any VAXBI node consisting only of VAXBI modules that jointly require n slots must be assigned to n contiguous slots within the same VAXBI cage. Since the minimum VAXBI cage has 6 slots, an exception must be obtained if a node exceeds six modules. Since VAXBI cages may be independently powered, this restriction avoids having power on a part of a node: a VAXBI node is powered as an entity. This constraint also minimizes the length of intermodule interconnect within a node.
- Within a VAXBI assembly the VAXBI module that supplies the VAXBI clock must reside in K1J1. One connector pin in the VAXBI region will have +5V only in K1J1, and will be grounded for all other equivalent pins in other KkJj locations. The signal is named BI ECL VCC H and will have +5V in K1J1B25.

  Only one module on a VAXBI bus can drive the clock, but other VAXBI modules with VAXBI clock drivers can be plugged into the same VAXBI. The VAXBI module in K1J1 drives the clock lines, and the other modules automatically disable their clock drivers without resorting to on-board switches.

digital™

13.6 VAXBI TERMINATORS

A VAXBI bus has two types of terminators: clock terminators and data path and control terminators.

Clock terminators must be located at the far end of the bus as far from the clock source as possible. For the configuration shown in Section 13.4.1, with the clock in K1J1, the clock terminators would be on the backplane at K3J6. For the configuration shown in Section 13.4.2, with the clock in K1J1, the clock terminators would be on the backplane at K4J6.

Data path and control signals are terminated at only one place on a VAXBI bus. The terminators for these lines may be located anywhere on any backplane of the VAXBI assembly. Depending on the implementation, the data terminators could be near the clock source, near the clock terminator, or distributed.

Packaging of the terminators is implementation dependent. Terminations can be in a single package or in many different packages.

13.7 OPERATING ENVIRONMENT

All components and subassemblies of a VAXBI device housed within a VAXBI card cage must operate over the full range of conditions specified below:*

- o +5V Voltage: 4.75 to 5.25V
- o Inlet Temperature: 5 to 50 degrees C
- o Humidity: 10 to 95% (with maximum wet bulb 32 C and minimum dew point 2 C)
- o Altitude: 0 to 2.4 km
- o Air Flow: 200 LFM at any component (min.), 12.8 SCFM per slot (min.)

---

*In a CIBCI subsystem, for example, the above requirements apply to the T1017 and T1018 modules because they reside wholly within the VAXBI card cage. The requirements do not apply to the CIBCI-BC option or to other components at the option level because components such as cables, the external power supply, and the link processor module do not reside within the card cage.

Draft

CHAPTER 14

OVERVIEW OF THE BIIC

Digital Part No. 78732

The BIIC (backplane interconnect interface chip) is a 133-pin ZMOS integrated circuit that serves as the primary interface between th VAXBI[tm]* bus and the user interface logic of a node. The BI implements the VAXBI bus protocol, which includes a distribute arbitration scheme and several error-checking facilities.

The BIIC provides a standard interface that meets the needs of processors, memory devices, and bus adapters. The chip provides capabilities for high-performance VAXBI attachments as well as a number of features to simplify attachment of low-to-medium performance devices.

Figure 14-1 shows a block diagram of a VAXBI node. Signals are input to the BIIC from the user interface on the BCI bus.

---

*VAXBI is a trademark of Digital Equipment Corporation.

digital™

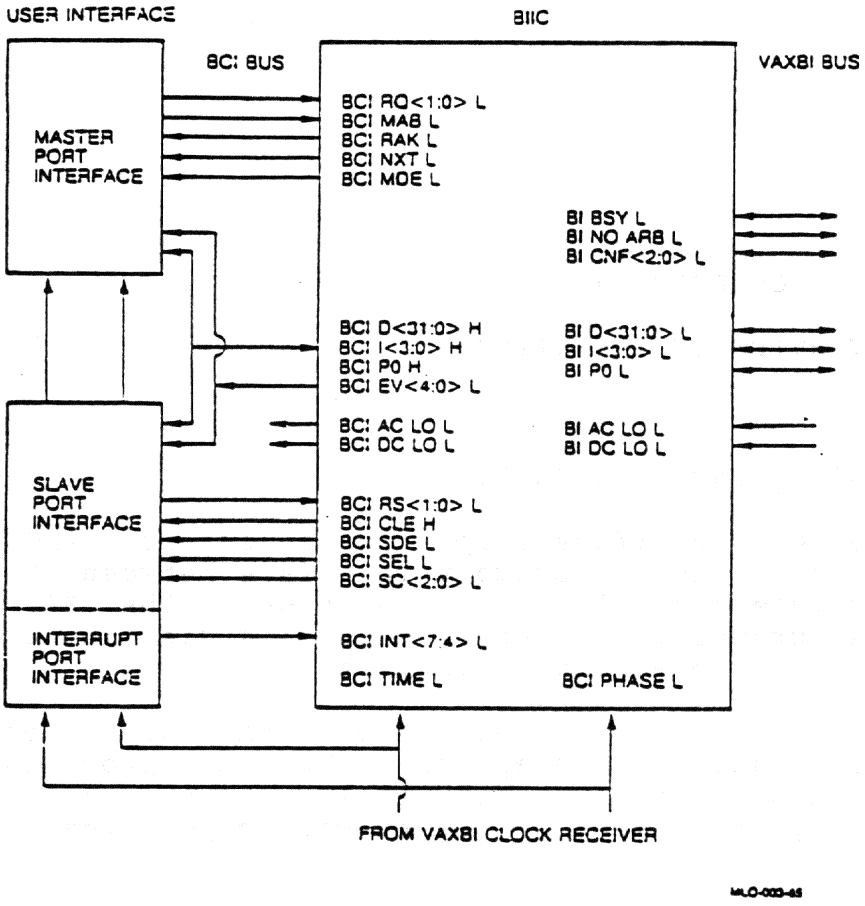


Figure 14-1: Block Diagram of a VAXBI Node

14.1 BIIC FEATURES

The BIIC allows the VAXBI to implement many advantageous features that other buses cannot provide. Overall, the BIIC reduces the overhead that other buses and device interfaces usually require, both in the hardware and in the operation of the bus protocol. Figure 14-2 shows a block diagram of the BIIC. Prominent features of the BIIC are listed below.

- The BIIC handles all aspects of arbitration for the VAXBI bus. The user interface simply makes a transaction request to the BIIC.

- The high level of integration reduces the amount of board area required, providing more room for the user interface logic. The number of connections is reduced, which contributes to maintaining data integrity. With fewer components, less heat is generated.

digital™

- Address decoding and matching is provided on-chip.
- All VAXBI registers are included on-chip. Four general purpose registers are also included for use by the node.
- Full VAXBI bandwidth (13.3 megabytes per second) can be sustained on the slave port. The master port supports 11.4 megabytes per second in pipeline request mode.
- All VAXBI bus transceivers associated with data transfer and interrupt communication are included on-chip.
- All bus receivers except BI TIME +/-, BI PHASE +/-, BI BAD L, BI RESET L, and BI STF L are included.
- A flexible and convenient interrupt system is included on-chip.
- A synchronous user side interface, the BCI, supports the data and control lines required to interface to the VAXBI bus.
- BIIC registers can be read or written by the user interface without using the VAXBI data path (loopback transactions).
- All VAXBI required error checking is provided on-chip.
- On power-up the BIIC begins a self-test. Any BIIC that does not pass its self-test disables its VAXBI drivers and allows the rest of the VAXBI system to continue to operate.
- A powered-down BIIC does not interfere with transactions on the VAXBI bus.
- A single +5 volt supply powers the BIIC and the VAXBI bus.
- All user interface signals are at TTL levels. There is no need for voltage level translation off-chip.
- Because the BIIC is a static design, input clocks can be stopped (except during self-test) without affecting the BIIC's operation. This design permits single-step debugging of static node designs.

digital™

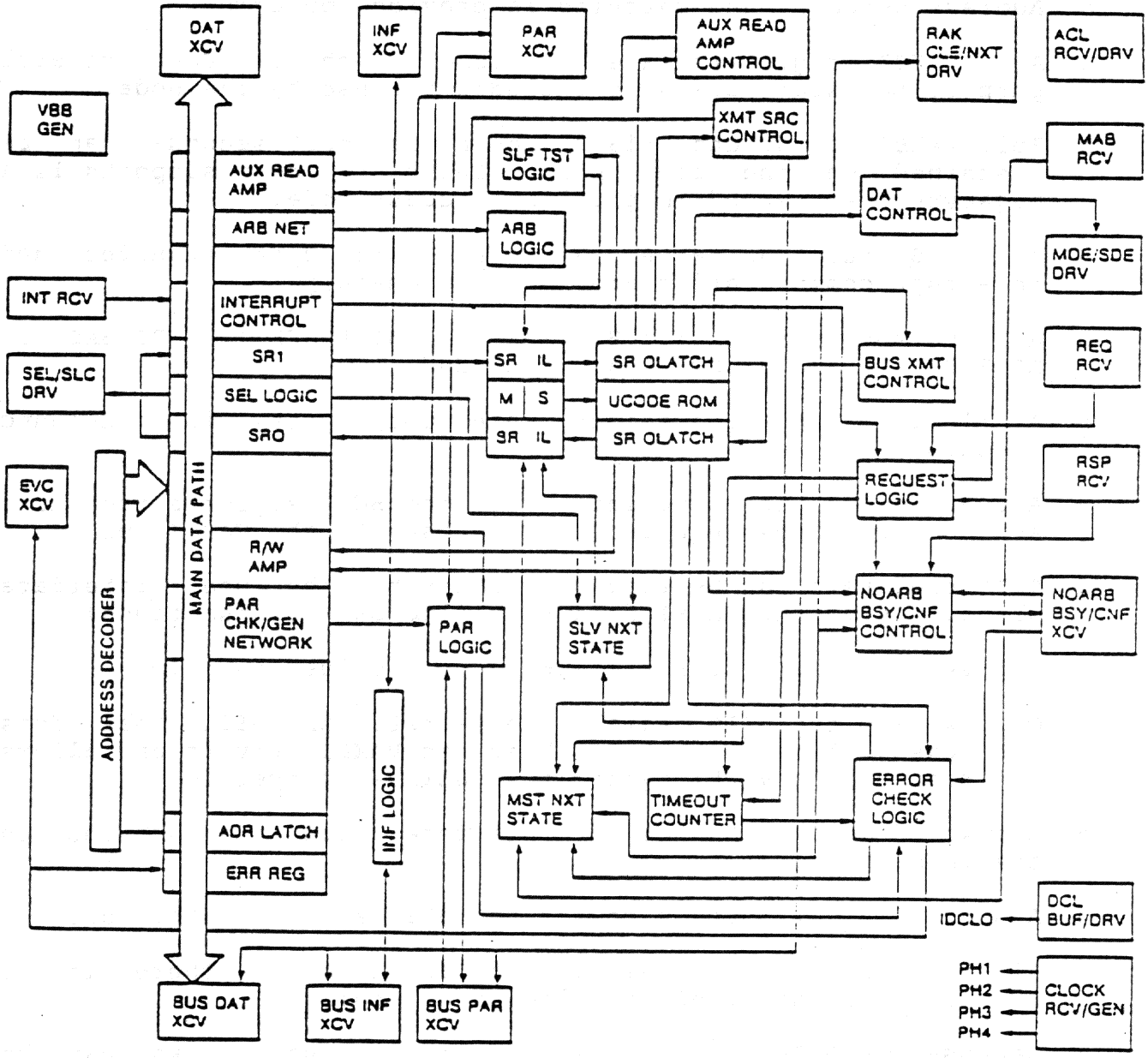


Figure 14-2: Block Diagram of the BIIC

digital™

14.2 BCI AND THE USER INTERFACE

Figure 14-1 shows the architecture of a typical VAXBI node based on the BIIC. User interface logic, represented by the master and slave port interface blocks, communicates with the BIIC over a synchronous interface bus called the BCI (BI chip interface). The BCI is a 64-wire synchronous interface to the BIIC. The user interface can request transfers across the BCI, but they are completed under the control of the BIIC. Because the BIIC includes the primary transceiver and protocol logic required to interface to the VAXBI, the user interface is relieved of this task.

The BIIC has the ability to detect and pass through any VAXBI commands directed to its node. Every address transmitted on the VAXBI bus is available to the user interface for monitoring cache invalidates in a multiprocessor environment. The BIIC supports transfers between a master and slave within a single node.

Data and address information is passed to and from the user interface over a set of 32 time-multiplexed three-state lines called the BCI D<31:0> H lines. Commands are passed over four lines called the BCI I<3:0> H lines. The direction control for these lines is provided by the BIIC. The master port interface initiates command sequences by asserting a code on the request lines (BCI RQ<1:0> L). The BIIC responds by asserting the appropriate enable signal to gate command data onto the BCI. Command and data confirmations are transmitted from the user interface to the BIIC over the response lines (BCI RS<1:0> L). Transaction status (successful completion, parity error, illegal CNF received, and so forth) is provided to the user interface over the event lines (BCI EV<4:0> L).

The BCI has a flexible interrupt system. Several registers in the BIIC control the operation of interrupts. Interrupts can be generated either by asserting a BCI interrupt request line (BCI INT<7:4> L) or by writing data into one of the interrupt control registers. In response to an IDENT command, the BIIC can provide vector information from an internal register, or it can solicit a vector from the user interface (external vector).

14.3 BCI FUNCTIONS

The signal lines of the BCI can be divided by function into three groups: master, slave, and interrupt.

digital™

14.3.1 Master Function

The master port consists of the BCI signals used to generate transactions. The transactions can be targeted to other nodes (internode transactions) or to the node that issues the transaction (intranode transactions). The intranode transactions can be VAXBI transactions (that is, the master port issues a transaction on the VAXBI bus), or they can be loopback transactions* (the VAXBI bus is not used.) The master port also provides the ability to read and write BIIC internal registers. Transactions targeted to other nodes can be of octaword, quadword, or longword lengths. Intranode transactions, however, are limited to longword length.

The master port interface (user interface logic) requests a transaction by asserting a code on the BCI RQ<1:0> L lines. The BIIC asserts BCI MDE L to indicate that the user interface is to place the command and address information on the BCI I<3:0> H and D<31:0> H lines, respectively. In subsequent cycles, along with BCI MDE L, the BIIC asserts BCI NXT L to solicit the user interface data needed to complete the transaction.

14.3.2 Slave Function

The slave port consists of the BCI signals used to respond to transactions targeted to a node. The slave port interface (user interface logic) responds to read and write requests to memory locations in this node. The slave port also receives commands intended for multiple responders, such as INTR commands.

The slave port can respond to all transactions directed to it, even when multiple transactions arrive back-to-back. Thus, the BIIC slave port can operate at the sustained peak bandwidth of the VAXBI bus, if the user interface is capable of sustaining that rate.

The slave port interface is not involved in read- and write-type transactions that access BIIC internal registers. These transactions are handled directly by the BIIC.

---

*Because loopbacks can increase the bus latency for other nodes, it is advisable not to perform large numbers of closely spaced loopbacks.

14.3.3 Interrupt Function

Nodes that generate interrupts can use the interrupt port to request the transmission of an interrupt and to respond with an external vector to IDENT commands. Interrupts are requested by asserting the BCI INT<7:4> L lines or by writing to the force bits in the Error Interrupt Control Register or User Interface Interrupt Control Register.

digital™

CHAPTER 15

BIIC SIGNALS

The BIIC has two sets of signal lines: the VAXBI lines are used for the VAXBI protocol, and the BCI lines are used to communicate with the user interface. The BIIC also has power and ground lines and transceiver and substrate bias reference lines.

15.1 VAXBI SIGNALS

Table 15-1 summarizes the 52 VAXBI signals. Each signal line has a pullup resistor, and all BIIC VAXBI drivers are open drain. (See Chapter 4 for a complete description of the VAXBI signals.)

Table 15-1: VAXBI Signals

Signal Name	Number/ Type	Description
BI D<31:0> L	32/OD	Used for the transfer of addresses and data and for arbitration. (Bidirectional to the BIIC.)
BI I<3:0> L	4/OD	Carry commands, encoded master IDs, read status codes, and write masks. (Bidirectional to the BIIC.)
BI P0 L	1/OD	Carries the parity for the D and I lines; asserted if the number of asserted bits on the D and I lines is an even number (ODD parity). (Bidirectional to the BIIC.)

(continued on next page)

digital™

Table 15-1: VAXBI Signals (Cont.)

Signal Name	Number/ Type	Description
BI NO ARB L	1/OD	Used to inhibit arbitration on the BI D lines; also asserted during BIIC self-test to prevent other nodes from starting transactions until all nodes are ready to participate. (Bidirectional to the BIIC.)
BI BSY L	1/OD	Used to indicate that a transaction is in progress. (Bidirectional to the BIIC.)
BI CNF<2:0> L	3/OD	Used to send responses for command and data cycles. (Bidirectional to the BIIC.)
BI AC LO L	1/OD	Used with BI DC LO L to perform power sequences. (Received by the BIIC.)
BI DC LO L	1/OD	Used with BI AC LO L to perform power sequences. (Received by the BIIC.)
BI TIME + BI TIME -	2/DECL	A 20 MHz clock reference used with BI PHASE +/- to generate all required timing signals.
BI PHASE + BI PHASE -	2/DECL	A 5 MHz clock reference used with BI TIME +/- to generate all required timing signals.
BI STF L	1/OC	A static control line used to enable a faster VAXBI system self-test; does not connect to the BIIC and has no direct effect on BIIC operation.
BI BAD L	1/OC	Used for reporting node failures; does not connect to the BIIC and has no direct effect on BIIC operation.

(continued on next page)

digital™

Table 15-1: VAXBI Signals (Cont.)

Signal Name	Number/ Type	Description
BI RESET L	1/OC	Used for initiating a VAXBI system reset; does not connect to the BIIC and has no direct effect on BIIC operation.
BI SPARE L	1/-	Reserved for use by Digital.

Key to abbreviations:
OD open drain
OC open collector
DECL differential ECL

15.2 BCI SIGNALS

Table 15-2 summarizes the 64 signal lines of the BCI. As shown in Figure 15-1, these lines can be divided by function into seven categories:

- 37 data path signals
- 6 master signals
- 8 slave signals
- 4 interrupt signals
- 5 transaction status signals
- 2 power status signals
- 2 clock signals

digital™

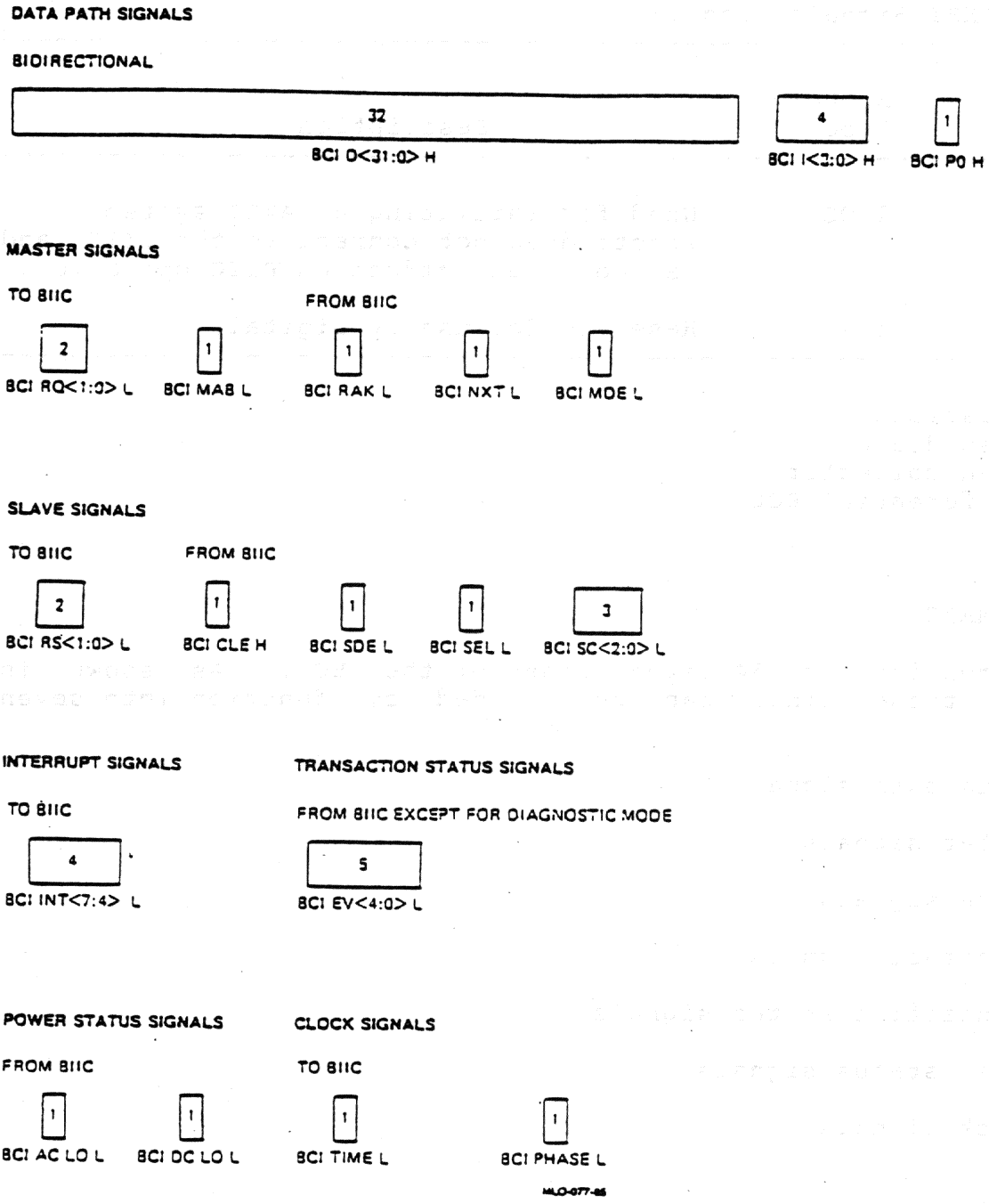


Figure 15-1: BCI Signals

digital™

Table 15-2: BCI Signals

Signal Name	Number	Description
BCI D<31:0> H	32	Used for the transfer of addresses and data to and from the BIIC.
BCI I<3:0> H	4	Carry commands, read status codes, and write masks.
BCI P0 H	1	Carries the parity for the BCI D and I lines; asserted if the number of asserted bits on the D and I lines is an even number (ODD parity).
BCI RQ<1:0> L*	2	Used by the master port interface to tell the BIIC to perform a specified transaction.
BCI MAB L*	1	Used by the master port interface to tell the BIIC to abort an ongoing master port transaction.
BCI RAK L	1	Used by the BIIC to indicate that a transaction requested by the master port interface has been initiated.
BCI NXT L	1	Used by the BIIC to request the next data word from the master port interface (for writes) and to indicate to the master port interface that the data on the BCI is valid (for reads).
BCI MDE L	1	Used by the BIIC to indicate to the master port interface that information to be transmitted on the VAXBI bus should be driven onto the BCI data path.
BCI RS<1:0> L*	2	Used by the slave port interface to indicate to the BIIC what code to send on the BI CNF<2:0> L lines in response to command and data cycles.
BCI CLE H	1	Used by the BIIC to indicate the presence of a command/address cycle.

*When not used in a node design, these signals should be pulled up by any of the following methods: by tying the signal through a resistor to Vcc, by tying the signal to Vcc directly, or by tying the signal to a logic level source that provides the H state.

(continued on next page)

Table 15-2: BCI Signals (Cont.)

Signal Name	Number	Description
BCI SDE L	1	Used by the BIIC to indicate to the slave port interface that information to be transmitted on the VAXBI bus should be driven onto the BCI data path.
BCI SEL L	1	Used by the BIIC to inform the slave port interface that it has been selected by a VAXBI transaction; asserted with BCI SC<2:0> L.
BCI SC<2:0> L	3	Used by the BIIC to give detailed selection information to the slave port interface; asserted with BCI SEL L.
BCI INT<7:4> L*	4	Used by the user interface to request that interrupts be performed; also used with the BCI RS L lines for diagnostic mode function selection.
BCI EV<4:0> L	5	Used to indicate the occurrence of significant events within the BIIC or on the VAXBI bus (except during diagnostic mode).
BCI AC LO L	1	Used by the user interface to monitor power status.
BCI DC LO L	1	Used by the user interface to monitor power status; also used in node reset sequences.
BCI TIME L	1	Used with BCI PHASE L by the BIIC and the user interface to generate all required timing signals; a 20 MHz TTL signal output from the VAXBI clock receiver in the user interface.
BCI PHASE L	1	Used with BCI TIME L by the BIIC and the user interface to generate all required timing signals; a 5 MHz TTL signal output from the VAXBI clock receiver in the user interface.

*When not used in a node design, these signals should be pulled up by any of the following methods: by tying the signal through a resistor to Vcc, by tying the signal to Vcc directly, or by tying the signal to a logic level source that provides the H state.

digital™

15.3 DATA PATH SIGNALS

The data path signals consist of the bidirectional data, information, and parity lines. BCI data path direction control is performed through the use of the BCI MDE L and SDE L lines. When either line is asserted, the user interface should drive the appropriate data onto the BCI. At all other times -- except when BI DC LO L is asserted -- the BCI is driven by the BIIC.

15.3.1 Data Lines (BCI D<31:0> H)

The BCI data lines are bidirectional three-state lines used for data transfers between the user interface and the BIIC. The BCI D<31:0> H lines provide a 32-bit data path for the transfer of address and data to and from the BIIC. During read-type, write-type, and INVAL transactions, bits <31:30> indicate the data length code and bits <29:0> provide the address (see Table 15-3).

Table 15-3: Data Length Codes

BCI D<31:30> H		
31	30	Data Length
L	L	RESERVED
L	H	Longword (LW) 4 bytes
H	L	Quadword (QW) 8 bytes
H	H	Octaword (OW) 16 bytes

15.3.2 Information Lines (BCI I<3:0> H)

The BCI information lines are bidirectional three-state lines used for data transfers between the user interface and the BIIC. The BCI I<3:0> H lines are used to transfer the following to and from the BIIC:

- Command (see Table 15-4 for the BCI command codes)
- Read status (see Table 15-5 for the read status codes)
- Write mask (see Table 15-6 for the write mask codes)

Because the BCI I<3:0> H lines are bidirectional, the BIIC does not guarantee that the master's encoded ID (transferred on the VAXBI bus

digital™

during the imbedded ARB cycle) will always be visible on these lines.

For example, during imbedded ARB cycles of read-type transactions to a slave port, the BCI I<3:0> H lines are used to send read status to the BIIC. Obviously, these lines cannot simultaneously drive out the master's encoded ID.

Table 15-4: BCI Command Codes

BCI I<3:0> H 3 2 1 0	Type	Name	Description
L L L L	--	--	RESERVED
L L L H	SR*	READ	Read
L L H L	SR	IRCI	Interlock Read with Cache Intent
L L H H	SR	RCI	Read with Cache Intent
L H L L	SR	WRITE	Write
L H L H	SR	WCI	Write with Cache Intent
L H H L	SR	UWMCI	Unlock Write Mask with Cache Intent
L H H H	SR	WMCI	Write Mask with Cache Intent
H L L L	MR	INTR	Interrupt
H L L H	SR	IDENT	Identify
H L H L	--	--	RESERVED
H L H H	--	--	RESERVED
H H L L	MR	STOP	Stop
H H L H	MR	INVAL	Invalidate
H H H L	MR	BDCST	Broadcast**
H H H H	MR	IPINTR	Interprocessor Interrupt

*SR = single responder; MR = multi-responder.
**See Appendix A.

digital™

Table 15-5: Read Status Codes

BCI I<3:0> H 3 2 1 0	Description
L * L L	RESERVED
L * L H	Read Data
L * H L	Corrected Read Data
L * H H	Read Data Substitute
H * L L	RESERVED
H * L H	Read Data/Don't Cache
H * H L	Corrected Read Data/Don't Cache
H * H H	Read Data Substitute/Don't Cache

*Bit <2> is RESERVED. Slaves must drive this line to L for all status types, and masters must ignore the state of this line.

Table 15-6: Write Mask Codes

Signal Asserted	Byte to Be Written
BCI I<3> H	BCI D<31:24> H
BCI I<2> H	BCI D<23:16> H
BCI I<1> H	BCI D<15:8> H
BCI I<0> H	BCI D<7:0> H

digital™

15.3.3 Parity Line (BCI P0 H)

The BCI P0 H line is a bidirectional three-state parity line for the BCI I<3:0> H and BCI D<31:0> H signals. The parity sense on the BCI P0 H line is ODD. This means that BCI P0 H should be asserted (assuming user interface-generated parity mode) if the number of asserted BCI D and I lines is an even number.

The operation of the BCI P0 H line is configured by the user interface at power-up. During BCI DC LO L time, the BIIC loads the Bus Error Register with the parity mode. If the user interface holds the BCI P0 H line asserted (or lets it default to H) during BCI DC LO L time, the BIIC will be configured to generate parity when BCI data is to be transmitted on the VAXBI bus. This is the BIIC-generated parity mode. In this mode, a somewhat longer setup time is required on the BCI D and I lines to provide time for the BIIC to generate the parity. An internal pullup defaults the BIIC to BIIC-generated parity, if the user interface does not drive the P0 line during BCI DC LO L time. In BIIC-generated parity mode the BCI P0 L line can be left floating.

Designers should verify that the output current characteristics of these pullup devices are sufficient for their needs. (See Section 20.2 for DC characteristics.)

If the user interface holds the BCI P0 H line deasserted during BCI DC LO L time, the BIIC will be configured for user interface-generated parity. The user interface then must provide proper parity on the BCI P0 H line whenever the BIIC solicits data (that is, whenever BCI MDE L or BCI SDE L is asserted). If the user interface supplies bad parity, the bad parity will be transmitted on the bus and a bus error will result. In this mode, a shorter setup time is permitted on the BCI D and I lines for data to be transmitted on the VAXBI bus.

When data received over the VAXBI bus is passed through to the BCI, the BCI P0 H line reflects the received state of the BI P0 L line. During loopback transaction cycles, the parity generated by the user interface or by the BIIC is passed to the BCI.

15.4 MASTER SIGNALS

The master signals consist of lines used to request, execute, and terminate transactions. The following lines provide input to the BIIC:

- Request lines
- Master abort line

The rest of the master signal lines carry output from the BIIC:

- Request acknowledge line
- Next line
- Master data enable line

15.4.1 Request Lines (BCI RQ<1:0> L)

The BCI RQ<1:0> L lines are used by the master port interface to request that the BIIC perform a transaction. The lines are also used to select the BIIC's diagnostic mode. Table 15-7 lists the BCI request codes.

Table 15-7: BCI Request Codes

BCI RQ<1:0> L	
1	0	Description
H	H	No request
H	L	VAXBI transaction request
L	H	Loopback request
L	L	Diagnostic mode

The BCI request lines are "pseudo-edge" triggered. The BIIC samples the state of the request lines during each T150/0. The BIIC then determines a transition on these lines (that is, an "edge") by comparing the received state from the previous cycle with the state received in this cycle. Only the transition of the request lines from the deasserted state (H H) to an asserted state (H L or L H) is interpreted by the BIIC as a request.

digital™

The request code on the BCI RQ<1:0> L lines can be removed during any cycle after the assertion of BCI RAK L by the BIIC. A new request will not be recognized by the BIIC until the request lines have been deasserted for at least one cycle. Therefore, the earliest that a new request can be presented is during the second cycle after a request has been removed (see Figure 15-2).

When the BIIC receives a VAXBI request, it uses the next opportunity to arbitrate on the bus for the new transaction. If the master presents a new request as soon as possible, the BIIC will arbitrate for the new request in the cycle after the last cycle of the present transaction, if that cycle is available for arbitration. A request is termed a "pipeline request" if it is posted prior to the deassertion of BCI RAK L for the present master port transaction.

Pipeline requests allow the master port interface to achieve an 11.4 megabyte per second throughput (see Application Note 8).

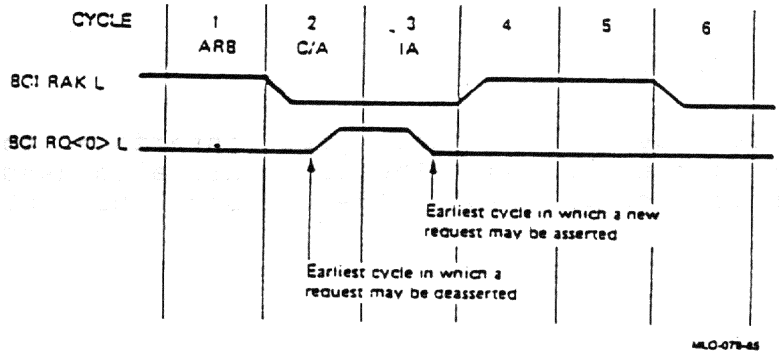


Figure 15-2: Pipeline-Request Signal Timing

15.4.1.1 VAXBI Transaction Request - The VAXBI transaction request code is used by the master port interface to request transactions on the VAXBI bus. The transactions can be directed to other nodes on the bus (internode transactions) as well as to its own slave port (intranode transactions). Intranode transactions, however, are limited to longword length.

All VAXBI commands, except INTR, can be sent through the use of the VAXBI transaction request. INTR commands can be initiated by the user interface only by asserting one of the BCI INT<7:4> L lines or by writing a force bit in the User Interface Interrupt Control Register in the BIIC. IPINTR commands can also be initiated by writing the IPINTR bit in the BCICSR.

15.4.1.2 Loopback Request - The loopback request code is used by the master port interface to request longword intranode read- and write-type transactions to nodespace that do not require the use of the VAXBI data lines. Furthermore, loopback requests allow a node to access its nodespace registers without reference to its node ID. Loopback transactions are performed by the BIIC's internal disabling of the VAXBI drivers (except BI NO ARB L and BI BSY L) and its wrapping the transaction data back to the VAXBI bus receive logic.

The BIIC supports concurrent loopback and VAXBI transactions. For example, a loopback transaction can occur at a given node, while two other nodes are performing a VAXBI transaction. To ensure proper bus operation, however, the BIIC will not initiate a loopback transaction at the same time that another node begins a VAXBI transaction. If the BIIC did not provide this protection, then the node performing the loopback transaction would be "blind" to the VAXBI transaction that might be intended for it.

This mechanism permits rapid access to the BIIC and slave port registers, regardless of activity on the VAXBI bus and in the presence of most types of bus failures (only BI BSY L and BI NO ARB L must be functional). However, since loopback transactions occur outside the control of the arbitration algorithm, there is the potential of degrading overall system access latency, and therefore this mechanism should be used with discretion. (See Section 10.2.1 for restrictions on the number of extension cycles.)

Loopback requests are presented to the BIIC in the same manner as VAXBI transaction requests. The only differences are:

- The high-order bits (D<29:13>) of the address in a loopback transaction are ignored by the BIIC's address selection logic (except for parity qualification). The BIIC will complete the transfer as if D<29:13> were set to 10 0000 0000 000n nnn, where n nnn is the node ID of this node. This corresponds to the nodespace of this node. The address delivered to the slave port interface will not be modified; that is, it will be the same as the address received by the BIIC from the master port interface.

  Due to this "abbreviated" address (node ID information is not required) that the BIIC uses for loopback transactions, loopback read- and write-type transactions have limited addressing capability. Loopback accesses by the user interface must be limited to the node's own nodespace. Addresses defined by the BIIC Starting and Ending Address Registers can be accessed only by VAXBI transaction requests.

- The BIIC does not arbitrate for the bus, and no VAXBI transaction is generated. If the bus is idle, a loopback transaction begins two cycles after the loopback request is

digital™

made. The BIIC asserts BI NO ARB L and BI BSY L just as in a VAXBI transaction, except that both are asserted during the loopback request command/address cycle. This ensures that no other BIIC will interpret this cycle as the C/A of a VAXBI bus transaction. Asserting BI NO ARB and BI BSY extends a current VAXBI transaction to allow completion of the loopback transaction. If a VAXBI transaction is in progress, the node with a pending loopback request waits to verify that it is not selected for the present VAXBI transaction and then begins the loopback transaction.

- The BIIC does not update its dual round-robin arbitration priority during the imbedded arbitration cycle of a loopback transaction.

15.4.1.3 Diagnostic Mode - Diagnostic mode is reserved for use by Digital. This mode is used to construct bus testers/monitors and other diagnostic equipment to facilitate chip testing and to provide more flexible access to the VAXBI bus.

The BIIC supports two transparent mode operations. In both modes the BCI signals are reassigned so that there is a one-to-one correspondence between VAXBI signals and BCI signals (except BI AC LO L and BI DC LO L). Since BI CNF<2:0> L, BI NO ARB L, and BI BSY L have no corresponding BCI signals in normal operation, the BCI EV<4:0> L lines are reassigned in transparent mode to allow these signals to be driven and received. The reassignment is given in Table 15-8. In BCI-to-BI transparent mode, the user interface presents data on the BCI D, I, P, and EV lines with the normal setup time (either Ts or Tsp) (see Section 20.3, AC Timing Specifications), and the data is synchronously asserted on the VAXBI bus by the BIIC.

In BI-to-BCI transparent mode, data received on the VAXBI bus is latched during each cycle and driven onto the BCI pins by the BIIC. The BCI D, I, and P lines are driven with inverted BI D, I, and P data. The BCI EV lines are driven with noninverted versions of the BI

Table 15-8: Diagnostic Mode BCI/BI Signal Assignment

BCI D<31:0> H	<-Inversion->	BI D<31:0> L
BCI I<3:0> H	<-Inversion->	BI I<3:0> L
BCI P0 H	<-Inversion->	BI P0 L
BCI EV<0> L	<-No Inversion->	BI CNF<0> L
BCI EV<1> L	<-No Inversion->	BI CNF<1> L
BCI EV<2> L	<-No Inversion->	BI CNF<2> L
BCI EV<3> L	<-No Inversion->	BI NO ARB L
BCI EV<4> L	<-No Inversion->	BI BSY L

digital™

CNF <2:0> L, BI BSY L, and BI NO ARB L lines. All data is driven on the BCI with the same timing as data for nontransparent mode operation. This means that the signal lines mapped onto the event code lines will be asserted referenced to T0 (not T150 as with the other BCI signals).

Diagnostic mode also supports a command that allows the loading of configuration data (device type, node ID, and parity mode). Outside of diagnostic mode, this loading only occurs at the end of BCI DC LO L time.

When in diagnostic mode, the BIIC examines the BCI response and interrupt lines to determine the desired diagnostic mode operation. The diagnostic mode code must not be asserted on the BCI RQ<1:0> L lines until the BIIC has completed self-test. Table 15-9 lists the codes for the diagnostic mode operations; only those codes listed are permitted during diagnostic mode. All diagnostic mode control signals (BCI RQ<1:0>, RS<1:0>, and INT<7:4> may be asserted concurrently (in the same cycle) when putting the BIIC in transparent mode. Similarly, the RS and INT control lines may be changed without deasserting the RQ lines to change the mode of operation. The new operation will begin within three cycles.

Table 15-9: Diagnostic Mode Operation Codes

BCI RS<1:0> L	BCI INT<7:4> L	
1 0	7 6 5 4	Operation
H H	H H H H	No operation
H H	L L L L	BCI-to-BI transparent mode
H L	H L H L	Load configuration data
L L	L L H H	BI-to-BCI transparent mode

digital™

15.4.2 Master Abort Line (BCI MAB L)

The BCI MAB L line is used by the master port interface to indicate to the BIIC that the present master port transaction from this node is to be aborted. BCI MAB L is also used to clear the retry state of the BIIC that is entered after a RETRY CNF is received.

The assertion of BCI MAB L does not affect BIIC-generated transactions.

In general, it is not possible for the user interface to abort a requested transaction without the risk of generating bus errors that would be detected by other nodes. Therefore, the BCI MAB L line should only be used to abort a transaction request following an error condition. (For example, a pipeline-request master should abort a transaction request for the transaction subsequent to one that fails.)

The actions taken by the BIIC vary, depending on the timing of BCI MAB L assertion.

- If BCI MAB L is asserted before the BIIC arbitrates, no transaction is initiated.
- If the BIIC has won the arbitration cycle and has not transmitted the C/A, it will deassert BI NO ARB L in the next cycle.
- If BCI MAB L is asserted after a RETRY event code is received (either RCR or RTO), the BIIC aborts its retry state so that it can accept a new request. Asserting BCI MAB L is the only way for the user interface to abort the retry state. Following the assertion of BCI MAB L, the user interface should not assert a new request for a minimum of three cycles. In a node that allows pipeline requests, the assertion of BCI MAB L may occur during a transaction from this node. In this case, the description in the next paragraph also applies.
- If BCI MAB L is asserted during a transaction, the BIIC aborts the transaction. The user interface does not necessarily receive an EV code for a transaction that it aborts. The user interface should allow a minimum of three cycles following the assertion of BCI MAB L before asserting a new request. Use of this mode of master abort should be avoided.

In all cases, the user interface should deassert the request lines in the same cycle that BCI MAB L is asserted.

digital™

15.4.3 Request Acknowledge Line (BCI RAK L)

The BCI RAK L line is used by the BIIC to indicate that a transaction requested by the master port interface has been initiated. The line is asserted during the first cycle of a VAXBI or loopback transaction. BCI RAK L remains asserted for the duration of the transaction and is deasserted the cycle when the master's EV code is output. For write-type and BDCST transactions, BCI RAK L stays asserted until the cycle after the slave's final ACK is on the VAXBI bus. This corresponds to three cycles after the last data cycle for write-type and BDCST transactions. For read-type and IDENT transactions, RAK stays asserted until two cycles after the last data cycle to allow time for an internal parity check. BCI RAK L is deasserted during the sixth cycle for STOP, INVAL, and master port IPINTR transactions.

Note that BCI RAK L is asserted only during master port transactions and is not asserted during BIIC-generated INTR and IPINTR transactions.

If the user interface reasserts BCI RQ <1:0> L before the deassertion of BCI RAK L (a pipeline request), the normal BCI RAK L timing is altered. In this case, BCI RAK L is deasserted in the cycle after BCI RQ<1:0> L was reasserted and will be reasserted again at the start of the master port interface's newly requested transaction.

15.4.4 Next Line (BCI NXT L)

BCI NXT L is asserted by the BIIC both to request the next data word from the master port interface (for VAXBI transmit cycles) and to indicate to the master port interface that the data on the BCI is valid (during a VAXBI receive cycle). For cycles in which the master is receiving data (such as during read-type data cycles), BCI NXT L is not asserted when a RETRY, STALL, or NO ACK confirmation code is received with the data, or when the parity on the received data is bad.

During write-type transactions the asserting edge of BCI NXT L serves to request the next write data longword. The new write data must be properly set up on the BCI D<31:0> H lines before the beginning of the next cycle. Because of buffer storage in the BIIC, BCI NXT L is asserted only once for each data word, even if the transaction is retried one or more times.

During read-type transactions the asserting edge of BCI NXT L indicates that the data on the BCI D<31:0> H lines is valid. This may be used to clock the read data into the user interface.

digital™

15.4.5 Master Data Enable Line (BCI MDE L)

The BCI MDE L signal is used by the BIIC to indicate to the master port interface that command/address information or data to be transmitted on the VAXBI bus should be driven onto the BCI D<31:0> H, BCI I<3:0> H, and BCI P0 H lines. BCI MDE L is asserted by the BIIC during each cycle that data from the master port interface is required on the BCI D, I, and P lines.

Because of buffer storage in the BIIC, BCI MDE L is asserted only once to acquire the command/address information and once for the first data longword. This capability may allow some simplification of design for nodes that perform only longword length master port transactions. For lengths longer than longword, the BIIC may assert BCI MDE L multiple times for the same data longword, but BCI NXT L is asserted only once for each data longword.

digital™

15.5 SLAVE SIGNALS

The slave signals consist of lines used to respond to transactions directed to a node. The following lines are output by the BIIC:

- Command Latch Enable line
- Slave Data Enable line
- Select line
- Select Code lines

The other slave signal lines are input to the BIIC:

- Response lines

15.5.1 Response Lines (BCI RS<1:0> L)

The BCI RS<1:0> L lines are used by the slave port interface to indicate to the BIIC what code to send on the BI CNF<2:0> L lines in response to command and data cycles. The slave port interface must respond with the appropriate codes on the RS<1:0> L lines whenever a transaction involving the slave port interface occurs (transactions to BIIC registers do not involve the slave port interface). During slave port interface transactions, a response is required for each cycle in which the slave node is required to drive the BI CNF<2:0> L lines. Table 15-10 lists the slave response codes.

Table 15-10: Slave Response Codes

BCI RS<1:0> L 1	BCI RS<1:0> L 0	Response Code	Resulting BI CNF Code
H	H	NO ACK	NO ACK
H	L	ACK	ACK or NO ACK
L	H	STALL	STALL, ACK, or NO ACK
L	L	RETRY	RETRY or NO ACK

There is not always a direct mapping between the response code and the corresponding CNF code sent on the VAXBI bus. Whenever a node is not involved in a transaction, the BIIC outputs the NO ACK CNF code, regardless of the code asserted on the BCI RS<1:0> L lines.

digital™

Some CNF codes are not legal in some cycle types. For example, during multi-responder transactions RETRY is not a legal CNF response. If the slave port interface sends a response that is not legal for a particular cycle, the BIIC sends a NO ACK on the BI CNF<2:0> L lines (except for the special case of a STALL response to a multi-responder command; see below).

The BCI RS lines are also used for diagnostic mode function selection (see Table 15-9).

15.5.1.1 Use of STALL to Extend VAXBI Transactions - The STALL response code can be used to extend VAXBI transactions. During single-responder commands, the BIIC at the slave node holds BI BSY L and BI NO ARB L asserted in the cycle following the receipt of the STALL code on the BCI RS<1:0> L lines. The BIIC also transmits the STALL code on the BI CNF<2:0> L lines during the same cycle.

Slaves can extend multi-responder commands by asserting the STALL response code for the command confirmation cycle and holding the STALL response for the desired number of extension cycles. During these cycles BI BSY L is held asserted. Because ACK and NO ACK are the only legal confirmations for multi-responder commands, the STALL response for the command confirmation cycle is changed by the BIIC to an ACK on the BI CNF<2:0> L lines. Subsequent STALLs cause NO ACKs to be transmitted on the VAXBI bus while BI BSY L remains asserted.

Nodes that are not slaves can also extend transactions by sending a STALL response code on their BCI RS<1:0> L lines. The BIIC at these nodes holds BI BSY L asserted (assuming BI BSY L was asserted in the previous cycle and that a stall timeout has not occurred) as long as the STALL response code is being sent. The BIIC's stall timeout counter limits the number of extension cycles to 127, after which BI BSY L is deasserted (this deassertion is qualified by the state of Device Register bit <13>, which must be clear for BSY to deassert at this time; otherwise, the assertion of BI BSY L will reflect only the state of the BCI RS lines). No other VAXBI bus signals are affected. If a BCI RS<1:0> L "stuck-at" STALL code failure occurs on a BIIC with Device Register bit <13> not set, the failed node will cause each subsequent VAXBI transaction to be extended by 127 cycles. If a BCI RS<1:0> L "stuck-at" STALL code failure occurs on a BIIC with Device Register bit <13> set, BI BSY L will never deassert and the bus will hang.

digital™

15.5.1.2 Use of STALL and Loopback Transactions - It is possible for a master port interface to initiate a loopback transaction while its slave port interface (as a nonslave) is extending an ongoing VAXBI transaction. If the loopback transaction is to BIIC CSR space, the BIIC handles the loopback transaction independent of the slave port interface's VAXBI transaction extension. The situation is more complicated if the loopback transaction accesses the user interface CSR portion of nodespace. If this occurs, then the STALL response code from the slave effectively stalls the loopback transaction as it extends the VAXBI transaction. Care must be taken to make sure that the slave port interface responds properly to the loopback transaction when the STALL code is removed. Nodes that do not perform loopback transactions to user interface CSR space or that do not use this type of extension will not have a problem.

15.5.2 Command Latch Enable Line (BCI CLE H)

The BCI CLE H line is monitored by the user interface to detect the presence of a command/address cycle. The deasserting edge can be used to latch the received command/address information off the BCI D and I lines. The BIIC asserts BCI CLE H the cycle after the BIIC recognizes BI NO ARB L is asserted and BI BSY L is deasserted. This NO ARB/BSY state corresponds to an arbitration cycle or the last data cycle of a transaction that has a pending master. The BIIC deasserts BCI CLE H in the cycle after the command/address cycle. (The C/A data is on the BCI D and I lines during this cycle.) The C/A cycle is detected at the transition of BI BSY L from the deasserted to the asserted state, with BI NO ARB L asserted in the previous cycle.

In some cases, BCI CLE H may be asserted for more than one cycle. This can occur following power-up when BIICs in different nodes complete self-test at different times, during burst mode operation, and following a pending master abort.

CAUTION

As a result, no node should depend on the deassertion of BCI CLE H for any particular cycle.

During loopback transactions the BIIC sequences BCI CLE H without regard to the BI BSY L and BI NO ARB L signals, but its timing relative to BCI signals is the same as it would be for a transaction on the VAXBI bus.

The BCI CLE H signal does not depend on the receipt of good parity, nor does it depend on whether the node is selected as a slave. A "bus

digital™

watching" silo or cache-resident node that examines all VAXBI transactions, not just those for which it is selected, can use the deassertion of BCI CLE H to latch the information from any C/A cycle on the VAXBI bus.

The assertion of BCI CLE H should be used to force the slave port interface back to a state from which it can respond to an SC code, possibly as early as the following cycle. In addition, the slave port interface should make sure that the STALL response code is removed during the cycle that BCI CLE H asserts. Note that assertions of CLE can be as closely spaced as two cycles apart. A mix of VAXBI and loopback transactions like that shown in Figure 21-3 demonstrates one instance where this can occur.

15.5.3 Slave Data Enable Line (BCI SDE L)

The BCI SDE L line is used by the BIIC to indicate to the slave port interface that data to be transmitted on the VAXBI bus should be driven onto the BCI D<31:0> H, BCI I<3:0> H, and BCI P0 H lines.

15.5.4 Select Line (BCI SEL L)

The BCI SEL L line is asserted by the BIIC to inform the slave port interface that it has been selected by a VAXBI transaction. BCI SEL L is asserted during the imbedded arbitration cycle of a transaction if the slave was selected by the command/address information from the previous cycle. The assertion of BCI SEL L is qualified by the receipt of good parity during the C/A cycle.

The assertion of BCI SEL L is accompanied by the assertion of a select code on the BCI SC<2:0> L lines. The BCI Control and Status Register (BCICSR) allows the user interface to create a customized subset of VAXBI transactions that will select the slave port interface. For instance, nodes that are not required to respond to multicast space read- and write-type commands can clear the MSEN bit in the BCICSR and then do not need to externally decode the multicast space SC code.

digital™

The BCI SEL L line is asserted for the following cases:

- A read- or write-type command whose address falls within the range defined by the BIIC Starting and Ending Address Registers has been received.
- A read- or write-type command whose address falls within multicast space has been received, and the MSEN bit is set in the BCICSR.
- A read- or write-type command matching the user interface CSR space of the node has been received, and the UCSREN bit is set in the BCICSR.
- A read- or write-type command matching the BIIC CSR space of the node has been received, and the BICSREN bit is set in the BCICSR.
- An IDENT command has been received, and the IDENTEN bit is set in the BCICSR.
- A BDCST command directed at the node has been received, and the BDCSTEN bit is set in the BCICSR.
- A STOP command directed at the node has been received, and the STOPEN bit is set in the BCICSR.
- A RESERVED command has been received, and the RESEN bit is set in the BCICSR.
- An IPINTR command directed at the node and matching the IPINTR Mask Register has been received, and the IPINTREN bit is set in the BCICSR.
- An INTR command directed at the node has been received, and the INTREN bit is set in the BCICSR.
- An INVAL command or a write-type command not directed to the range of addresses defined by the Starting and Ending Address Registers has been received, and the INVALEN or the WINVALEN bit, respectively, is set in the BCICSR.

digital™

15.5.5 Select Code Lines (BCI SC<2:0> L)

The BCI SC<2:0> L lines are used by the BIIC to give detailed selection information to the slave port interface. A select code on these lines accompanies an assertion of the BCI SEL L line and vice versa. The assertion of the select code is qualified by the receipt of good parity for the command/address data (just as for BCI SEL L). If the user interface fully decodes the BCI SC<2:0> L lines, there is no need to examine the BCI SEL L line. Table 15-11 gives the select codes.

digital™

Table 15-11: Select Codes

BCI SC<2:0> L 2 1 0	Description
H H H	Default state of the lines indicates that no selection has occurred.
H H L	A read- or write-type command to the user interface CSR space has been received, and the UCSREN bit* is set. Or a read- or write-type command to the BIIC CSR space for the node has been received, and the BICSREN bit is set.
H L H	A read- or write-type command to the range of addresses defined by the Starting and Ending Address Registers has been received.
H L L	A read- or write-type command to the range of addresses defined as multicast space has been received, and the MSEN bit is set.
L H H	An IDENT transaction has been received, and the IDENTEN bit is set.
L H L	An INTR transaction whose destination matches the node has been received, and the INTREN bit is set. Or an IPINTR transaction whose destination matches the node and whose source matches the IPINTR Mask Register has been received, and the IPINTREN bit is set.
L L H	An INVAL or write-type command not directed to the range of addresses defined by the Starting and Ending Address Registers and not having D<29> asserted (that is, not I/O space) has been received, and the INVALEN bit or the WINVALEN bit, respectively, is set.
L L L	A BDCST, STOP, or RESERVED command has been received, with a destination matching the node (except for RESERVED commands), and the BDCSTEN, STOPEN, or RESEN bit, respectively, is set.

* All bits noted are in the BCI Control and Status Register.

digital™

15.6 INTERRUPT SIGNALS

15.6.1 Interrupt Request Lines (BCI INT<7:4> L)

The BCI INT<7:4> L lines are used by the user interface to request that interrupts be performed by the BIIC. The BCI INT<7:4> L signals must be synchronously asserted. Each INT signal is used to cause interrupts at the level corresponding to its bit position (that is, BCI INT<7> L initiates level 7 interrupts, BCI INT<6> L initiates level 6 interrupts, and so forth).

Interrupts can also be generated by writing to the User Interface Interrupt Control Register and setting one or more of the force bits.

The BCI INT lines are "pseudo-edge" triggered. The BIIC samples the state of the lines during each T150/0. The BIIC then determines a transition on these lines (that is, an "edge") by comparing the received state from the previous cycle with the received state in this cycle. The transition of the lines from the deasserted state (H) to the asserted state (L) is interpreted by the BIIC as an interrupt request.

The BCI INT lines are also used with the BCI RS lines for diagnostic mode function selection (see Table 15-9).

digital™

15.7 TRANSACTION STATUS SIGNALS

15.7.1 Event Code Lines (BCI EV<4:0> L)

The BCI EV<4:0> L lines are used to indicate the occurrence of significant events within the BIIC or on the VAXBI (except during diagnostic mode -- see Section 15.4.1.3 for the reassignment of these lines during diagnostic mode). The three classes of EV codes are as follows:

- Summary EV codes -- provide status at the end of a transaction
- Status EV codes -- provide status during a transaction
- Special EV codes -- provide self-test status and bus timeout information

Table 15-12 lists the event codes by class.

15.7.1.1 Summary Event Code Operation

In general, the summary EV codes indicate the result of a transaction involving a particular node. A successful completion or an error code is output at the end of the transaction. The user interface can then take action based on that EV code.

The master port interface receives a summary EV code for each transaction (unless the BCI MAB L line has been used to abort the transaction). The slave port interface receives a summary EV code for transactions in which an error is detected and for successful read-type and IDENT transactions and for VAXBI writes to BIIC internal registers. For all other successful transaction types, the slave controls the transaction, determines when the transaction is complete, and therefore has no need for a "completion" EV code.

Since the EV code lines are shared between the master and slave ports, the lines are time-multiplexed with a specific transaction cycle dedicated to master summary EV codes and a specific cycle dedicated to slave summary EV codes. This ensures there will be no EV line contention during intranode transfers, when both master and slave summary EV codes must be delivered to the user interface. Only one master and one slave summary EV code are output for each master transaction, and each code is output for only one cycle. There are separate internal latches for storing master and slave summary EV codes.

digital™

Table 15-12: Event Codes by Class

Summary	
MCP	Master Port Transaction Complete
AKRSD	ACK Received for Slave Read Data
RCR	RETRY CNF Received for Master Port Command
IRW	Internal Register Written
NICI	NO ACK or Illegal CNF Received for INTR Command
NICIPS	NO ACK or Illegal CNF Received for Force-Bit IPINTR/STOP Command
AKRE	ACK CNF Received for Error Vector
STO	Stall Timeout on Slave Transaction
BPS	Bad Parity Received During Slave Transaction
ICRSD	Illegal CNF Received for Slave Data
BBE	Bus BSY Error
AKRNE4	ACK CNF Received for Non-error Vector -- Level 4
AKRNE5	ACK CNF Received for Non-error Vector -- Level 5
AKRNE6	ACK CNF Received for Non-error Vector -- Level 6
AKRNE7	ACK CNF Received for Non-error Vector -- Level 7
RDSR	Read Data Substitute or RESERVED Status Code Received
ICRMC	Illegal CNF Received for Master Port Command
NCRMC	NO ACK CNF Received for Master Port Command
ICRMD	Illegal CNF Received by Master Port for Data Cycle
RTO	Retry Timeout
BPM	Bad Parity Received During Master Port Transaction
MTCE	Master Transmit Check Error
Status	
ARCR	Advanced RETRY CNF Received
IAL	IDENT ARB Lost
EVS4	External Vector Selected -- Level 4
EVS5	External Vector Selected -- Level 5
EVS6	External Vector Selected -- Level 6
EVS7	External Vector Selected -- Level 7
BPR	Bad Parity Received
Special Case	
BTO	Bus Timeout
STP	Self-Test Passed

digital™

The first occurrence of a summary EV code "condition" causes the corresponding summary EV code to be latched into the appropriate EV code latch and output during the designated cycle. Following are the designated cycles for the output of summary EV codes:

- o From the master --
  - For write-type and BDCST transactions: one cycle after the slave's final ACK response is on the VAXBI bus
  - For read-type and IDENT transactions: the cycle during which the master's final ACK response is on the VAXBI bus
- o From the slave --
  - For write-type and BDCST transactions: the cycle during which the slave's final ACK response is on the VAXBI bus
  - For read-type and IDENT transactions: the cycle after the master's final ACK response is on the VAXBI bus

(See Figures 15-3 and 15-4.) Exceptions occur when a bus error causes the transaction to abort. The summary EV codes may be output sooner than they would be if the transaction completed successfully. In any case, if the transaction is intranode, the master and slave EV codes are output in the same order as if the transaction completed successfully.

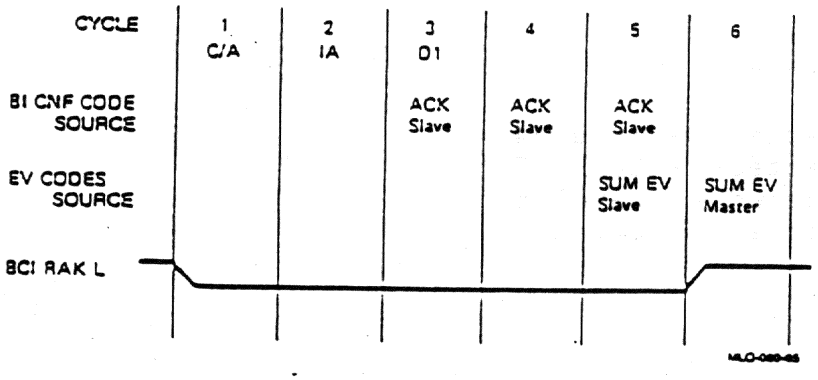


Figure 15-3: Summary EV Code Timing for Successful Write-Type and BDCST Transactions (Longword Write)

digital™

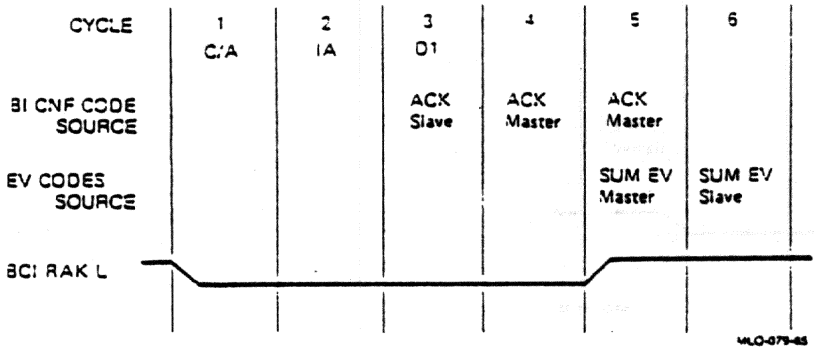


Figure 15-4: Summary EV Code Timing for Successful Read-Type Transactions (Longword Read)

During master port STOP, IPINTR, and INVAL transactions, the master EV code is output during the sixth cycle of the transaction (see Figure 15-5). BIIC-generated IPINTR and INTR transactions receive the appropriate NO ACK or Illegal CNF Received summary EV code during the sixth cycle if the transaction is unsuccessful (no EV code is output for non-master port transactions). The BCI RAK L timing shown in Figure 15-5 applies only to master port transactions. BIIC-generated INTR and IPINTR transactions do not cause the assertion of BCI RAK L.

The slave port interface does not output an EV code for successful STOP, INTR, IPINTR, and INVAL transactions.

Note that since the BI CNF<2:0> L status is typically two cycles behind data activity on the VAXBI bus, the EV codes on the BCI EV<4:0> L lines can be output during a subsequent transaction on the bus. The user interface must associate the event codes with the appropriate master and/or slave transactions. This is a simple process for nonpipelined request master port interfaces (deassertion of BCI RAK L indicates the cycle in which the master summary EV code will be output); however, high-performance master port interfaces that produce pipelined requests must implement a more complicated scheme. A suitable scheme could use the event code window concept discussed in Section 15.7.2.

digital™

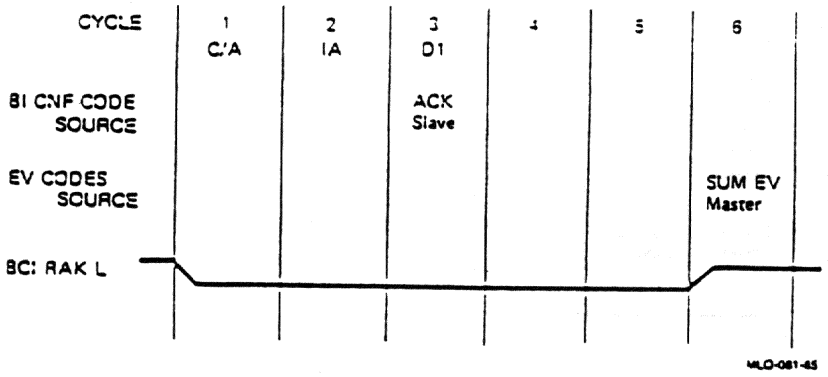


Figure 15-5: Summary EV Code Timing for Successful STOP, INTR, IPINTR, and INVAL Transactions

15.7.1.2 Status Event Code Operation - Status EV codes provide status during a transaction. For example, the Bad Parity Received (BPR) EV code is output the cycle after a data cycle parity error has been detected by either the master or slave BIIC. This EV code gives nodes the earliest possible indication of bad parity on the VAXBI bus and may be used to terminate a write-type operation. The Bad Parity Received During Master Port Transaction (BPM) and Bad Parity Received During Slave Transaction (BPS) summary EV codes are output during the designated cycles at the end of the transaction as well.

The IDENT command has two types of slave status EV codes: the External Vector Selected (EVSn) and IDENT ARB Lost (IAL) EV codes. They can be output during the fourth and sixth cycles of the IDENT transaction, respectively.

15.7.1.3 Special Event Code Operation - The special EV codes provide information not related to a particular transaction.

- The Bus Timeout (BTO) EV code can be output at any time, except during data cycles of a master transaction from this node. Prioritization logic within the BIIC ensures that the BTO code is overridden when any other code must be output. The BTO code will be output continuously until the transaction request(s) are removed or a transaction begins.

- The Self-Test Passed (STP) EV code is output after the BIIC passes its self-test.

digital™

15.7.2 Event Code Windows

The earliest that a summary or status EV code can be output for a transaction is in the fourth cycle of that transaction. The latest that these codes can be output is in the third cycle of the following transaction. The node can use a delayed version of BCI CLE H to create a transaction EV code window to keep track of which transaction owns the present EV code (see Figure 15-6).

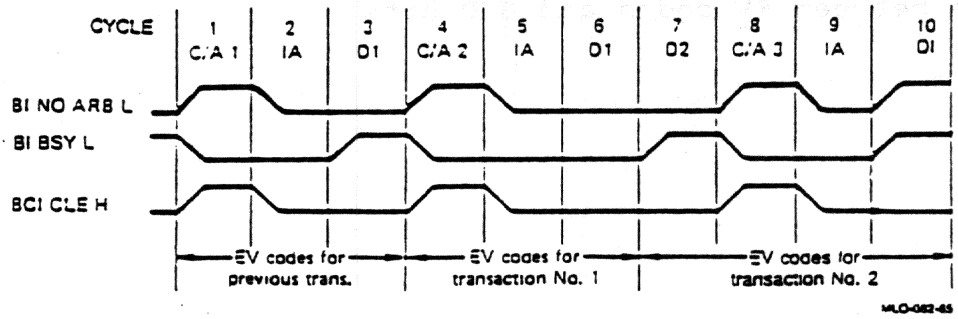


Figure 15-6: Transaction EV Code Windows

15.7.3 Event Codes

Table 15-12 listed the event codes by class, while Table 15-13 lists the event codes with other information. Following each code is the mnemonic for the code, the class, and the type.

The three classes are:

SUM	summary
STA	status
SPC	special case

The EV code type refers to the portion of the node for which the EV code applies:

M	master
S	slave
I	interrupt

digital™

Type refers to the nature of the EV code:

ERROR
INFO information

The remainder of this section describes each EV code, in the order in which the codes are listed in Table 15-13.

In most cases the output of EV codes corresponds to the setting of certain hard-error bits in the Bus Error Register (BER). Table 15-14 shows the relationship between EV codes and BER bits.

digital™

Table 15-13: Event Codes

BCI EV<4:0> L 4 3 2 1 0	Mnemonic	Class	Type
H H H H H	NEV	-	-
H H H H L	MCP	SUM	M:INFO
H H H L H	AKRSD	SUM	S:INFO
H H H L L	BTO	SPC	M:ERROR
H H L H H	STP	SPC	INFO
H H L H L	RCR	SUM	M:INFO
H H L L H	IRW	SUM	S:INFO
H H L L L	ARCR	STA	M:INFO
H L H H H	NICI	SUM	I:ERROR/INFO
H L H H L	NICIPS	SUM	I:ERROR/INFO
H L H L H	AKRE	SUM	I:INFO
H L H L L	IAL	STA	I:INFO
H L L H H	EVS4	STA	I:INFO
H L L H L	EVS5	STA	I:INFO
H L L L H	EVS6	STA	I:INFO
H L L L L	EVS7	STA	I:INFO
L H H H H	STO	SUM	S:ERROR
L H H H L	BPS	SUM	S:ERROR
L H H L H	ICRSD	SUM	S:ERROR
L H H L L	BBE	SUM	S:ERROR
L H L H H	AKRNE4	SUM	I:INFO
L H L H L	AKRNE5	SUM	I:INFO
L H L L H	AKRNE6	SUM	I:INFO
L H L L L	AKRNE7	SUM	I:INFO
L L H H H	RDSR	SUM	M:ERROR
L L H H L	ICRMC	SUM	M:ERROR
L L H L H	NCRMC	SUM	M:ERROR/INFO
L L H L L	BPR	STA	M/S:ERROR
L L L H H	ICRMD	SUM	M:ERROR
L L L H L	RTO	SUM	M:ERROR
L L L L H	BPM	SUM	M:ERROR
L L L L L	MTCE	SUM	M:ERROR

No Event -- NEV -- The default deasserted state. No event is presently reported.

Master Port Transaction Complete -- MCP (SUM M:INFO) -- The last master port transaction on the VAXBI bus completed successfully from the master's viewpoint. This EV code is output during the cycle when the final ACK is transmitted by the master for read-type and IDENT

digital™

transactions and during the cycle after the final ACK has been received from the slave for write-type and BDCST transactions. This EV code is output during the sixth cycle of STOP, INVAL, and master port IPINTR transactions.

For nonpipelined users, the MCP EV code is output in the cycle that BCI RAK L deasserts. If the master performs pipelined requests, this EV code may occur during the C/A or imbedded ARB cycle of the next master transaction from this node. The pipelined master interface logic must be able to associate this code with the proper transaction. Note that it is possible that a read-type or IDENT transaction for which the master receives the MCP EV code might not complete successfully if the final ACK that the master transmits on the bus is corrupted and the slave decodes some other CNF code. In this case the slave node receives an error EV code -- NO ACK or Illegal CNF Received for Slave Data -- and the ICE bit in the slave's Bus Error Register is set.

Upon receipt of a STOP command, the BIIC clears the summary EV code registers. If a STOP transaction selects the master's own slave port interface, the BIIC does not output a summary EV code. However, if the STOPEN bit in the master's BIIC is not set, the slave port interface does not respond to a STOP command (even if the destination matches its node ID) and the master summary EV codes are output.

ACK Received for Slave Read Data -- AKRSD (SUM S:INFO) -- The last read-type transaction completed successfully from the slave's point of view. This EV code is output for read-type transactions one cycle after the slave receives the ACK confirmation for the last read data cycle from the transaction master.

Bus Timeout -- BTO (SPC M:ERROR) -- This EV code is output if the node is unable to start at least one transaction (out of possibly several that are pending) within 4096 cycles after a request (from either the BIIC or the master port interface) has been posted. This condition also results in the setting of the BTO bit in the Bus Error Register. After the bus timeout, the BTO EV code is output until all requests are deasserted or a transaction occurs. Any transaction subsequently initiated by the node clears the BTO condition. The BTO EV code can be superseded by any other EV code.

Note that slave port transactions can take place while there is a BTO condition present at the node.

Self-Test Passed -- STP (SPC INFO) -- This EV code is output the cycle after the BIIC passes its internal self-test. No EV code is output if the BIIC fails its self-test. Nodes can use this code in lieu of accessing the STS (Self-Test Status) bit in the VAXBI Control and Status Register.

RETRY CNF Received for Master Port Command -- RCR (SUM M:INFO) -- This EV code is output if the CNF received from the slave for the master's read- or write-type C/A cycle is RETRY, and a retry timeout has not occurred.

Internal Register Written -- IRW (SUM S:INFO) -- This EV code is output after the completion of a VAXBI write-type transaction targeted to the BIIC CSR space at the node. The IRW EV code is output whether or not a register at this location is implemented. The IRW code is not output for loopback write-type transactions.

Advanced RETRY CNF Received -- ARCR (STA M:INFO) -- This EV code is output by the master's BIIC during the cycle immediately following the receipt of a RETRY CNF. Because this EV code is output one or two cycles earlier (depending on the transaction type) than the RETRY CNF Received EV code, the code is useful for supporting pipeline-request master port designs. Note that a retried transaction causes the output of both the Advanced RETRY CNF Received (ARCR) and RETRY CNF Received for Master Port Command (RCR) (or Retry Timeout (RTO) if a timeout occurs) EV codes.

NO ACK or Illegal CNF Received for INTR Command -- NICI (SUM I: ERROR/INFO) -- This EV code is output if the received CNF code for an INTR command is NO ACK, a RESERVED code, or a code that is an illegal response to an INTR command (RETRY or STALL).

If the user interface needs to differentiate between NO ACK and illegal or RESERVED CNF codes, it can read the BER and examine the ICE bit, which is set only for illegal or RESERVED CNF codes. Conversely, the NMR bit (NO ACK to Multi-Responder Command Received) is only set for NO ACK CNF codes.

NO ACK or Illegal CNF Received for Force-Bit IPINTR/STOP Command -- NICIPS (SUM I: ERROR/INFO) -- This EV code is the same as NO ACK or Illegal CNF Received for INTR Command except that it occurs only for IPINTR or STOP commands initiated by setting the IPINTR/STOP Force bit in the BCI Control and Status Register. IPINTR or STOP transactions that are initiated as VAXBI transaction requests from the master port generate either NO ACK CNF Received for Master Port Command or Illegal CNF Received for Master Port Command EV codes.

ACK CNF Received for Error Vector -- AKRE (SUM I:INFO) -- This EV code is output after the slave BIIC receives an ACK confirmation from the IDENTing master for the slave's transmitted error vector data. The error vector was sent as a result of an error interrupt sent under control of the Error Interrupt Control Register.

digital™

IDENT ARB Lost -- IAL (SUM I:INFO) -- If a slave node loses an IDENT arbitration, the slave's BIIC outputs this EV code two cycles after the IDENT ARB.

External Vector Selected (Level 4, Level 5, Level 6, or Level 7) -- EVS4, EVS5, EVS6, or EVS7 (STA I:INFO) -- These EV codes are used to solicit an external vector from the user interface. The EVSn EV code is output if the following conditions are met:

- The BIIC participates in an IDENT arbitration
- The External Vector bit is set in the User Interface Interrupt Control Register
- No error interrupt is pending at this node at a level selected by the IDENT command (since error interrupts at any level take priority over all normal interrupts)

The BIIC outputs the External Vector Selected EV code that corresponds to the highest priority pending interrupt that had a corresponding bit set in the IDENT Level field. If the highest matching level is 4, then EVS4 is transmitted. Because the external vector is solicited prior to the IDENT ARB decision, the vector may not be transmitted on the VAXBI bus during this IDENT transaction.

The receipt of an AKRNEn EV code indicates that the vector was transmitted by the node and was correctly received by the IDENTing master.

Stall Timeout on Slave Transaction -- STO (SUM S:ERROR) -- This EV code is output if the user interface stalls a data cycle for more than 127 consecutive cycles and if bit <13> of the Device Register is not set. If bit <13> of the Device Register is not set, a node that is not a slave will also receive the STO EV code if it extends a VAXBI transaction for more than 127 cycles. (See Section 15.5.1.1 on the use of STALL to extend VAXBI transactions.) If the user interface asserts the STALL code while BI BSY L is asserted on the VAXBI bus and holds it for more than 127 consecutive cycles, then the BIIC outputs this EV code and the STO bit in the slave's Bus Error Register will also set (unless bit <13> of the Device Register is set). The BIIC slave port terminates its participation in or its effect on the transaction when the timeout occurs and transmits NO ACKs on the BI CNF lines.

Bad Parity Received During Slave Transaction -- BPS (SUM S:ERROR) -- This EV code is output if a slave BIIC detects a parity error during a write-type ACK or STALL data cycle or a BDCST ACK data cycle. This condition also results in the setting of the SPE bit in the Bus Error Register.

digital™

Illegal CNF Received for Slave Data -- ICRSD (SUM S:ERROR) -- The transaction master is required to return two final ACK responses to the slave during read-type and IDENT transactions. If the transaction master sends any CNF responses except two ACKs in the two final confirmation cycles, then an error has occurred and the slave's BIIC outputs this EV code.

Bus BSY Error -- BBE (SUM S:ERROR) -- During the course of a transaction, slave BIICs monitor the BI BSY L line for proper operation. If a slave BIIC detects the absence of BI BSY L when it should have been asserted, the slave BIIC terminates involvement in the transaction and outputs this EV code.

ACK CNF Received for Non-error Vector (Level 4, Level 5, Level 6, or Level 7) -- AKRNE4, AKRNE5, AKRNE6, or AKRNE7 (SUM I:INFO) -- These EV codes are the equivalent of ACK CNF Received for Error Vector except that they apply only to vectors transmitted as a result of a Level n INTR transaction. These four Non-error Vector EV codes allow the user interface to differentiate between interrupts that were the result of a user interface request from those that were originated by the BIIC (that is, error interrupts that result from the BIIC's detection of an error condition that set a BER bit). These EV codes can be used to deassert the user interface's interrupt request at the specified level.

Read Data Substitute or RESERVED Status Code Received -- RDSR (SUM M:ERROR) -- This EV code is output if a Read Data Substitute or a RESERVED status code is received for read status (for read-type transactions) or vector status (in the case of IDENTs). This EV code is output in place of the MCP EV code (for read-type commands), or AKRNEn or AKRE (for IDENT transactions). The BIIC also sets the RDS bit in the Bus Error Register. The RDSR code is the only master error EV code that does not result in aborting the transaction (since the error is not a bus error but is probably a memory node malfunction). Consequently, after the RDSR condition has been discovered and this EV code has been latched in the BIIC's internal EV code latch, a bus error could occur that would cause the transaction to abort. However, the user interface would still receive the RDSR EV code at the end of the transaction.

Illegal CNF Received for Master Port Command -- ICRMC (SUM M:ERROR) -- This EV code is output if the CNF code received for the master's C/A cycle was one of the RESERVED confirmation codes or a RETRY or STALL code to multi-responder command types.

NO ACK CNF Received for Master Port Command -- NCRMC (SUM M:ERROR/INFO) -- This EV code is output if the confirmation code received for the master's C/A cycle was NO ACK and no MTCE error was detected during the C/A cycle.

digital™

Bad Parity Received -- BPR (STA M/S:ERROR) -- This EV code can be output for either the master or the slave. The BPR EV code is output in the cycle after the BIIC detects a parity error during the following transaction data cycle types:

- o Read-type ACK data cycles (output for the master)
- o Vector ACK data cycles (output for the master)
- o Write-type ACK or STALL data cycles (output for the slave)
- o BDCST ACK data cycles (output for the slave)

This EV code is useful to nodes that require an early indication of bad parity during a transaction. Specifically, it can be used to suppress the writing of data received with bad parity.

Illegal CNF Received by Master Port for Data Cycle -- ICRMD (SUM M:ERROR) -- During read-type, write-type, and BDCST transactions, the BIIC outputs this EV code if slaves return an illegal CNF code after the command confirmation has been received. As with other commands, this code can be either a RESERVED CNF or a CNF that is not a legal response for the command (for example, a RETRY or NO ACK data confirmation for a write-type or BDCST transaction).

Retry Timeout -- RTO (SUM M:ERROR) -- This EV code replaces the RETRY CNF Received for Master Command on the 4096th and subsequent consecutive RETRY responses from the slave, if the RTOEVEN bit is set in the BCICSR. After a retry timeout, the BIIC also sets the RTO bit in the Bus Error Register. The user interface can assert BCI MAB L to abort the BIIC's retry state and to clear the timeout counter.

Bad Parity Received During Master Port Transaction -- BPM (SUM M:ERROR) -- This EV code is output if the master detects a parity error on the VAXBI bus during a read-type or vector ACK data cycle.

Master Transmit Check Error -- MTCE (SUM M:ERROR) -- During a transaction the master node verifies that the received data on the BI D, I, and P lines matches the data transmitted from this node during any cycle in which the master should be the only node driving these lines (except for the master's assertion of its encoded ID during the imbedded ARB cycle, which is not "transmit-checked"). This EV code is output for all master port transactions if the transmitted data does not match the received data. The BIIC also sets the MTCE bit in the Bus Error Register. The MTCE EV code is not output for BIIC-generated transactions; however, the BIIC does set the MTCE bit if a mismatch is detected.

Table 15-14: Correlation of Event Codes and Bus Error Register Bits

Event Codes	Bus Error Register Bits														
	NMR	MTCE	CTE	MPE	ISE	TDF	IVE	CPE	SPE	RDS	RTO	STO	BTO	NEX	ICE
BTO													↵		
NICI	↵														↵1
NICIPS	↵														↵1
STO												↵			
BPS									↵						
ICRSD							↵								↵
BBE															
RDSR										↵					
ICRMC															↵
NCRMC	↵													↵	
BPR				↵					↵						
ICRMD															↵
RTO											↵				
BPM				↵											
MTCE		↵1													

MLO-003648

KEY

↵ Direct correlation between the output of the EV code and the setting of the Bus Error Register bit.

↵1 Same as ↵ for master port interface transactions. BIIC-generated transactions will never cause the output of the MTCE EV code if the error condition is detected. The MTCE bit, however, will be set.

↵ This EV code represents one error condition, but not the only condition, that will result in the setting of this BER bit.

↵1 Same as ↵ for illegal CNF responses; however, this error bit will not be set if the response was NO ACK.

↵ This BER bit represents one error condition, but not the only condition, that will result in the output of this EV code.

digital™

15.8 POWER STATUS SIGNALS

The power status signals consist of the BCI AC LO L and BCI DC LO L lines. The use of these signals in a VAXBI system is discussed in Application Note 7.

15.8.1 AC LO Line (BCI AC LO L)

The BCI AC LO L signal, a buffered and synchronized version of the BI AC LO L signal, is used by the user interface to monitor power status.

The BIIC receives BI AC LO L and transmits the received state on the BCI AC LO L line two cycles later. BCI AC LO L is driven synchronously. During the two-cycle delay the BIIC verifies that the received state was stable. It will not change the state of BCI AC LO L unless the new BI AC LO L state was the same for both preceding cycles (effectively filtering out one-cycle glitches of BI AC LO L).

15.8.2 DC LO Line (BCI DC LO L)

The BCI DC LO L signal is used by the user interface to monitor power status. The BIIC asserts BCI DC LO L asynchronously following the assertion of BI DC LO L. The maximum assertion delay is given by the Tdas spec in the AC timing specifications (see Section 20.3). BI DC LO L assertions of less than 50 ns are ignored and do not result in the assertion of BCI DC LO L.

The BIIC deasserts BCI DC LO L synchronously Tdde cycles following a valid deassertion of BI DC LO L. "Valid" deassertion means that BI DC LO L remains stable and deasserted for two consecutive cycles. While BI DC LO L is asserted, the BIIC disables all VAXBI drivers, as well as the BCI D, I, and P drivers. This feature can help facilitate VAXBI module testing by allowing modules to be tested independent of the BIIC.

When the Node Reset (NRST) bit is set, the BIIC drives BCI DC LO L without regard to the state of BI DC LO L. The BIIC asserts BCI DC LO L for Tnrw cycles following the writing of the NRST bit. When BCI DC LO L is deasserted, the BIIC begins its internal self-test (just as it does for a power sequence). As long as BI DC LO L is asserted, the BIIC maintains a valid low-level output voltage on BCI DC LO L even as the Vcc supply voltage to the BIIC is in transition.

A special Iol specification covers the BCI DC LO L output while Vcc = 0v. (See Section 20.2, DC Characteristics.) This specification reflects the operation of special internal BIIC circuitry that allows the BCI DC LO L output to sink a small amount of current even when the

BIIC is powered-down. An external pulldown resistor may be added if the BIIC's power-down current sink capability is insufficent.

The BIIC loads node ID, device type, revision, and parity mode into the BIIC during the cycle before BCI DC LO L is deasserted. Normally, the user interface drives node ID, device type, revision, and parity mode onto the BCI I<3:0> H, BCI D<31:0> H, and BCI P0 H lines respectively, while BCI DC LO L is asserted. The BCI D<31:0> H and P0 H lines all have internal pullups that are "on" during BCI DC LO L, and therefore any undriven lines default to H.

Designers should verify that the output current characteristics of these pullup devices are sufficient for their needs. (See Section 20.2 for DC characteristics.) Some node designs may be able to take advantage of these defaults.

The BCI D and P lines default to H only during BCI DC LO L time.

digital™

15.9 CLOCK SIGNALS

The clock signals BCI TIME L and BCI PHASE L are used by both the BIIC and the user interface.

15.9.1 BCI TIME L

BCI TIME L is a 20 MHz TTL signal that is output from the VAXBI clock receiver in the user interface. BCI TIME L, along with BCI PHASE L, is used by both the BIIC and the user interface to generate all required timing signals.

15.9.2 BCI PHASE L

BCI PHASE L is a 5 MHz TTL signal that is output from the VAXBI clock receiver in the user interface. BCI PHASE L, along with BCI TIME L, is used by both the BIIC and the user interface to generate all required timing signals.

digital™

CHAPTER 16

BIIC REGISTERS

The BIIC registers are located in the first 256 bytes of a node's nodespace. This portion of nodespace is called BIIC CSR space. Table 16-1 lists the BIIC registers. (See Chapter 7 for complete descriptions of VAXBI registers.)

Most locations in BIIC CSR space are unused. When a read-type command accesses unused locations, the BIIC responds by reading zeros. When a write-type command accesses unused locations, the BIIC accepts the command but ignores the data.

The BIIC registers can be accessed in two ways: (1) a node can send out a VAXBI transaction to its own BIIC or to another node's BIIC, and (2) a node's master port interface can do a loopback transaction.

When BIIC registers are accessed using the loopback request command from the master port, the high-order bits of the address (the bits that select the node address space) are ignored by the BIIC. The low-order bits D<12:1> determine whether an internal register is selected (D<12:8> = 0), and if so, which register is selected, just as in the case of a VAXBI transaction.

The BIIC supports mask writes but does not support lock functionality for BIIC internal registers. IRCI and UWMCI commands to internal registers are accepted by the BIIC and treated the same as normal READ and WMCI commands, respectively.

digital™

Table 16-1: BIIC Registers

Name	Abbrev.	Address
Device Register	DTYPE	bb + 0*
VAXBI Control and Status Register	VAXBICSR	bb + 4
Bus Error Register	BER	bb + 8
Error Interrupt Control Register	EINTRCSR	bb + C
Interrupt Destination Register	INTRDES	bb + 10
IPINTR Mask Register	IPINTRMSK	bb + 14
Force-Bit IPINTR/STOP Destination Register	FIPSDES	bb + 18
IPINTR Source Register	IPINTRSRC	bb + 1C
Starting Address Register	SADR	bb + 20
Ending Address Register	EADR	bb + 24
BCI Control and Status Register	BCICSR	bb + 28
Write Status Register	WSTAT	bb + 2C
Force-Bit IPINTR/STOP Command Register	FIPSCMD	bb + 30
User Interface Interrupt Control Register	UINTRCSR	bb + 40
General Purpose Register 0	GPR0	bb + F0
General Purpose Register 1	GPR1	bb + F4
General Purpose Register 2	GPR2	bb + F8
General Purpose Register 3	GPR3	bb + FC

*"bb" refers to the base address of a node (the address of the first location of the nodespace).

digital™

CHAPTER 17

BIIC DIAGNOSTIC FACILITIES

This chapter describes the operation of the diagnostic facilities that the BIIC provides. Section 17.1 describes self-test, Section 17.2 describes error detection, and Section 17.3 describes error recovery.

17.1 SELF-TEST

The BIIC performs self-test on power-up and as part of a node reset sequence. In both cases, BIIC self-test begins following the deassertion of BCI DC LO L. While the BIIC power-up self-test is running, BI NO ARB L is held asserted to prevent bus activity. In the cycle after the completion of self-test, the BIIC outputs the Self-Test Passed EV code if the test completed successfully. The BIIC also sets the Self-Test Status (STS) bit in the VAXBI Control and Status Register upon successful completion of self-test. (The STS bit is cleared on power-up.)

The absence of a Self-Test Passed EV code indicates a self-test failure. If the self-test fails, the BIIC deasserts all VAXBI drivers using a redundant driver disable signal. A successful self-test runs for approximately 4096 cycles (.82 milliseconds). If the self-test does not complete in the normal amount of time, a "watchdog" timer terminates the self-test after approximately 2.5 million cycles (500 milliseconds) and disables the VAXBI drivers.

See Section 18.1 for details on BIIC operation on power-up and Chapter 11 and Application Note 4 for details on self-test.

digital™

17.2 ERROR DETECTION

The BIIC supports extensive error detection logic. Following the detection of an error, the BIIC logs its occurrence, and if enabled, can automatically generate error interrupts to notify the host processor. The BIIC performs the following types of error detection:

- Comparison Checking of Transmit and Receive Data

  The BIIC verifies the data transmitted on the BI D<31:0> L, I<3:0> L, and P0 L lines by looping back the data and comparing it with the desired transmit data. This comparison should detect any driver failure on these lines, as well as collisions between masters that simultaneously take the VAXBI bus, following a transient error during an arbitration cycle.

- Two-Bit Distance Coding

  The BIIC verifies that its assertions of the VAXBI control signals (BI BSY L, BI NO ARB L, and BI CNF<2:0> L) result in assertions of these signals on the VAXBI bus. The two-bit distance between legal CNF codes provides a further level of error detection by ensuring that any single-bit CNF code error will result in the detection of an illegal CNF response.

- Parity Checking

  The BIIC verifies the receipt of correct parity from the VAXBI bus.

- Protocol Checking

  The BIIC both verifies adherence to proper VAXBI protocol and ensures that protocol requirements are met.

17.3 ERROR RECOVERY

The design of the BIIC permits transactions that have produced errors to be reattempted. In most cases the BIIC makes it possible to recover from transient bus failures without corruption of data. Error recovery alternatives and limitations are described in Appendix C, Responses to Exception Conditions.

digital™

CHAPTER 18

BIIC OPERATION

This chapter describes the operation of the BIIC on power-up and during VAXBI transactions.

18.1 POWER-UP OPERATION

On power-up the BIIC loads configuration data from the user interface into its registers and then performs a self-test.

18.1.1 Loading of Configuration Data

On power-up the BIIC asynchronously asserts BCI DC LO L from the assertion of BI DC LO L. All VAXBI drivers are disabled. During the last cycle in which BCI DC LO L is asserted, the BIIC loads the following:

- The Device Register with data from the BCI D<31:0> H lines
- The Node ID field in the VAXBI Control and Status Register (VAXBICSR) with data from the BCI I<3:0> H lines
- The User Parity Enable bit in the Bus Error Register to indicate whether the BIIC or the user interface is to generate parity for outgoing data; taken from the state of the BCI P0 H line

Only BCI DC LO L should be used to enable this configuration data onto the BCI.

digital™

During BCI DC LO L time, pullups (internal to the BIIC) default the BCI D<31:0> H lines and the BCI P0 H line to the high state. This feature can be used to minimize the number of signals that the user interface must drive during power-up. Designers should verify that the output current characteristics of these pullup devices are sufficient for their needs. (See Section 20.2 for DC characteristics.)

As a minimum, the user interface must drive the node ID on the BCI I<3:0> H lines while BCI DC LO L is asserted. If no other data is provided, the parity mode defaults to BIIC-generated parity mode and the BIIC loads all ones into the Device Register. The Device Register can then be loaded with the proper data during node initialization by a normal write-type transaction. Node designers should make sure that their node power-up sequence meets VAXBI architectural requirements.

18.1.2 Self-Test

The BIIC requires that BI DC LO L be asserted for a minimum of 72 cycles (14.4 microseconds) while DC power is valid, so that the BIIC can initialize registers associated with self-test. Following the deassertion of BI DC LO L, the BIIC deasserts BCI DC LO L and begins its self-test.

While the power-up self-test is running, BI NO ARB L is held asserted to hold off bus activity. If the BIIC passes its power-up self-test, the Self-Test Passed EV code is output for one cycle during the cycle after the self-test completes. During the node reset sequence, BI NO ARB L is not asserted.

The user interface can determine the results of the BIIC's self-test in two ways:

- Wait for receipt of the Self-Test Passed EV code.
- Read the Self-Test Status (STS) bit in the VAXBICSR.

The request for this read transaction can be posted anytime after the deassertion of BCI DC LO L. The BIIC will not respond until after the completion of self-test, at which time the STS bit will reflect the result of the self-test. A loopback transaction should be used to read the VAXBICSR, because the STS bit is also the enable for the VAXBI drivers. If self-test fails, thereby leaving the STS bit reset, the VAXBI drivers will remain disabled, and a VAXBI transaction from this node cannot complete successfully. A loopback transaction, on the other hand, since it does not use the VAXBI data path, can still operate (assuming the reason for the self-test failure was not something that affects the ability of the BIIC to perform a loopback read transaction).

digital™

Table 18-1 lists the signals that the BIIC asserts during self-test, the signals that remain deasserted, and those signals that the user interface may assert.

Table 18-1: BIIC Operation During Self-Test

Signals Asserted by the BIIC	Signals That Remain Deasserted	Signals That the User Interface May Assert
BCI D<31:0> H	BCI MDE L	BCI RQ<1:0> L*
BCI I<3:0> H	BCI SDE L	BCI INT<7:4> L
BCI P0 H	BCI NXT L	BCI RS<1:0> L
BI NO ARB L**	BCI CLE H	BCI MAB L
	BCI SEL L	
	BCI SC<2:0> L	
	BCI EV<4:0> L	
	BI D<31:0> L	
	BI I<3:0> L	
	BI P0 L	
	BI BSY L	
	BI NO ARB L**	

*The user interface should not drive the diagnostic mode code on the BCI RQ<1:0> L lines during self-test. Assertion of the diagnostic mode code terminates self-test prematurely and leaves the node in an undefined state.

**BI NO ARB L is asserted by the BIIC during power-up self-test but not during node reset self-test.

18.2 RETRY STATE

RETRY is a legal response code for read-type, write-type, and IDENT transactions. The BIIC, upon receipt of a legal RETRY response code from a slave, enters the retry state. The BIIC outputs the RETRY CNF Received for Master Port Command EV code and stores a copy of the transaction command/address information and the first data longword (only for write-type and BDCST transactions) in its internal buffers.

digital™

The BIIC does not resend the transaction until the user interface deasserts the request lines and then reasserts the request. This permits nodes to implement a variable wait time before retransmission of a retried transaction. Alternatively, nodes can force the resending of the transaction by gating off the VAXBI transaction request with the RETRY CNF Received EV code. Since the EV code is asserted for only one cycle, this has the effect of deasserting and reasserting the VAXBI transaction request. When resending a retried transaction, the BIIC does not re-request the C/A information or the first data longword (if applicable). The BIIC picks up the cycling of BCI NXT L and BCI MDE L where the transaction left off. Therefore, the master port interface can treat the retried transaction like a normal transaction, except for the treatment of the request lines, and the possibility that slave activity may appear on the BCI D, I, and P lines interleaved with the data cycles of the transaction.

After receiving 4096 consecutive RETRY confirmation responses, the BIIC outputs the Retry Timeout EV code in place of the RETRY CNF Received EV code (assuming the RTO EV Enable bit is set in the BCICSR). The user interface can continue to resend the transaction just as for the "non-timed-out case." The BIIC continues to output the Retry Timeout EV code each time it receives a Retry confirmation response for the resent transaction.

If the user interface does not wish to resend the transaction, it must assert BCI MAB L. Assertion of BCI MAB L causes the BIIC to purge its internal data holding registers in preparation for a new request.

18.3 VAXBI TRANSACTIONS AND BIIC OPERATION

This section describes the operation of the BIIC during various types of VAXBI transactions.

18.3.1 Read-Type Transactions

18.3.1.1 Master Operation - The master port interface requests a VAXBI transaction on the BCI RQ<1:0> L lines. The BIIC responds by asserting BCI MDE L (as soon as the following cycle), indicating that the user interface should now assert a VAXBI command, an address, and parity (if in user-interface-generated parity mode) on the BCI I<3:0>, D<31:0>, and P0 lines, respectively.

digital™

After arbitrating and winning the VAXBI bus, the BIIC drives the command/address data onto the bus and asserts BCI RAK L. For each longword, the asserting edge of BCI NXT L indicates when valid read data is on the BCI data lines. The receipt of a STALL from the slave delays the assertion of NXT L until an ACK is received (indicating valid data). Following the last data cycle, the BIIC transmits two ACK CNFs on the VAXBI bus if the transfers were successful. The user interface can inhibit the sending of these final ACKs by asserting BCI MAB L. The Master Transaction Complete EV code is output in the same cycle that the master's final ACK is on the VAXBI bus. Unless the master port interface makes a pipeline request, BCI RAK L is deasserted in the same cycle that the Master Transaction Complete EV code is output.

18.3.1.2 Slave Operation - All BIICs deassert BCI CLE H during the imbedded ARB cycle to allow latching of data from the BCI. Each BIIC determines if the transmitted address is in the range of addresses allocated to its node. The BIIC in the selected node asserts BCI SEL L and the appropriate select code on the BCI SC<2:0> L lines. Before the end of the imbedded ARB cycle, the slave port interface must assert its command response on the BCI RS<1:0> L lines. An ACK response serves as both a positive command confirmation and an indication to the master that the present data cycle contains valid read data. A STALL does not indicate the same positive command confirmation as an ACK, since a node can STALL and then NO ACK if the address is in the range allocated to this node but is not an implemented location. For read-type transactions, a STALL indicates that the present data cycle does not contain valid read data. RETRY indicates that the command cannot be completed at this time. A NO ACK response indicates that the node was not selected by the transmitted address.

The transaction continues, with the slave providing read data, data status, and a response (either ACK or STALL) on the BCI D, I, and RS lines. This pattern repeats until all data is transferred from slave to master. For successful transactions the BIIC in the master node sends two final ACKs on the VAXBI. The BIIC in the slave responds by sending the ACK Received for Slave Read Data EV code in the cycle after the master's final ACK is on the VAXBI.

Internal register read transactions are handled entirely by the BIIC, and the slave port interface does not participate. However, if the BIIC CSR Space Enable (BICSREN) bit in the BCICSR is set, the BIIC asserts SEL and SC for read-type transactions to BIIC internal registers, thereby allowing the user interface to monitor these transactions. A successful internal register read-type transaction causes the output of an ACK Received for Slave Read Data EV code, regardless of the setting of the BICSREN bit.

digital™

18.3.2 Write-Type Transactions

18.3.2.1 Master Operation - The master port interface requests a VAXBI transaction on the BCI RQ<1:0> L lines. The BIIC responds by asserting BCI MDE L (as soon as the following cycle), indicating that the user interface should now assert a VAXBI command, an address, and parity (if in user-interface-generated parity mode) on the BCI I<3:0>, D<31:0>, and P0 lines, respectively.

After arbitrating and winning the VAXBI bus, the BIIC drives the command/address data onto the bus and asserts RAK L on the BCI side. The asserting edge of BCI NXT L occurs during the imbedded ARB cycle to indicate that the first data word should be prepared for presentation on the BCI. BCI MDE L is then asserted later in this cycle to enable the user interface's buffer (containing the first data longword) onto the BCI. This cycling of NXT and MDE continues for each data longword to be transferred. An extra NXT L cycle occurs at the end of the write-type transaction if the Pipeline NXT Enable (PNXTEN) bit is set in the BCICSR. After the last write data cycle, the slave responds with two final ACKs, and the BIIC outputs the Master Transaction Complete EV code to the user interface. Unless the master port interface makes a pipeline request, BCI RAK L is deasserted in the same cycle that the Master Transaction Complete EV code is output.

18.3.2.2 Slave Operation - All BIICs deassert BCI CLE H during the imbedded ARB cycle to allow latching of data from the BCI. Each BIIC determines if the transmitted address is in the range of addresses allocated to its node. The BIIC in the selected node asserts BCI SEL L and the appropriate select code on the BCI SC<2:0> L lines. Before the end of the imbedded ARB cycle, the slave port interface must assert its command response on the BCI RS<1:0> L lines. An ACK response serves as both a positive command confirmation and an indication to the master to send the next data longword (if the transaction is greater than longword) in the next data cycle. A STALL does not indicate the same positive command confirmation as an ACK, since a node can STALL and then NO ACK if the address is in the range allocated to this node but is not an implemented location. The STALL code tells the master that the data from the first data cycle should be repeated in the second data cycle. RETRY indicates that the command cannot be completed at this time. A NO ACK response indicates that the node was not selected by the transmitted address.

The transaction continues, with the slave port interface providing a response (either ACK or STALL) for each data cycle, until all data is transferred from master to slave. If the transfers are successful, the slave port interface provides two final ACK responses which are transmitted on the bus, and the BIIC in the master responds by outputting the Master Transaction Complete EV code. This EV code is output the cycle after the slave's final ACK is on the VAXBI. The slave BIIC does not output an EV code for successful write-type transactions, except for VAXBI writes to internal registers. For internal register writes, the BIIC outputs the Internal Register Written EV code. This EV code is not output for internal register writes that are loopback transactions.

Internal register write transactions are handled entirely by the BIIC, and the slave port interface does not participate. However, if the BICSREN bit in the BCICSR is set, the BIIC asserts SEL and SC for write-type transactions to BIIC internal registers, thereby allowing the user interface to monitor these transactions. An internal register write appears similar to a user nodespace write except that the RS lines do not affect the CNF responses generated by the BIIC. The Internal Register Written EV code is output regardless of the setting of the BICSREN bit.

18.3.3 INTR and IDENT Transactions

18.3.3.1 Master Operation - The BIIC can generate two types of interrupts, and each type can be initiated in two different ways. The two types are user interface interrupts and error interrupts. Both types of interrupts are transmitted on the VAXBI bus with the same command, INTR. One is initiated by the user, whereas the other is generally initiated by the BIIC following the detection of an error on the bus (for example, a command parity error).

To initiate a user interface interrupt, the user interface asserts one of the four BCI INT<7:4> L lines or performs a write to set the appropriate force bit in the User Interface Interrupt Control Register. The error interrupt, on the other hand, is initiated automatically by the BIIC if a bus error is detected (and the appropriate INTR enable bit is set in the VAXBI Control and Status Register). Alternatively, the user interface can force an error interrupt by setting the force bit in the Error Interrupt Control Register.

Another difference between the two types of interrupts is that the User Interface Interrupt Control Register supports up to four pending interrupts, one at each interrupt level, whereas the Error Interrupt

digital™

Control Register is limited to one level (level information for error interrupts is contained in the Error Interrupt Control Register). Also, the BIIC gives higher priority to error interrupts. Error interrupt requests take precedence over user interface interrupts both in terms of the INTR transaction transmission and in the response to IDENT commands.

Following an INTR request, the BIIC arbitrates for the VAXBI bus and upon winning transmits the INTR transaction. Since the required interrupt level and destination information are contained on-chip, the user interface does not provide any data during the INTR transaction. Note that user interface and error interrupts use the same INTR Destination Register. During the command/address cycle of the INTR transaction, the appropriate Sent bit is set in either the User Interface or Error Interrupt Control Register.

If user interface interrupts are pending at more than one level (for example, if the user interface simultaneously asserts all four INT<7:4> L lines), then when the VAXBI bus is available, the BIIC will send an INTR with all pending interrupt levels set. However, since only the Sent bit at the highest pending level will be set, the BIIC will attempt to arbitrate again for the VAXBI bus, this time sending an INTR at the remaining pending levels. ("Pending" in this sense essentially means that there is an INTR request, either force bit or INT<7:4> type, but the INTR Sent bit in the User Interface Interrupt Control Register at the requested level is not set).

If an interrupting condition goes away and the user interface deasserts the BCI INT<7:4> L lines one or two cycles before the arbitration cycle has occurred, the BIIC will transmit the INTR command with no levels set.

18.3.3.2 Slave Operation - Nodes fielding interrupts (that is, the slaves for the INTR transaction) should set the appropriate interrupt pending bit, or its equivalent, to record that an interrupt at a given level was received. When an interrupt-fielding node is ready to respond to an INTR command, this information is used to drive the Level field (D<19:16>) during the IDENT command/address cycle.

If the INTR transaction does not complete successfully (that is, a NO ACK or illegal response is received during the command confirmation cycle), then the BIIC sets INTRAB and INTRC (both in UINTRCSR) at the level(s) sent out with the INTR command. The NO ACK or Illegal CNF Received for INTR Command EV code is output on the BCI EV<4:0> L lines. With the INTRC bit set, no more INTR transactions at that level are sent out. To resend the INTR transaction, a node must deassert and then reassert the interrupt request. Alternatively, the INTRC and INTRAB bits can be reset by a write to the Interrupt Control Register.

digital™

The interrupt-fielding node solicits vector information from the interrupting node through the use of the IDENT command. The IDENT command is initiated by the master port interface as a VAXBI transaction request. IDENT level information is provided by the user interface for transmission during the IDENT command cycle. The IDENT Level field must contain only one asserted bit.

All slaves with interrupt requests pending at the IDENT level participate in the IDENT, verifying that the decoded master ID transmitted during the third cycle matches one of the bits set in their INTR Destination Register. If both requirements are met, the node then participates in the IDENT arbitration. No user interface data is required from the slave port interface unless the External Vector bit is set in the User Interface Interrupt Control Register. If appropriate, the BIIC outputs an External Vector Selected -- Level n EV code during the cycle in which the STALL response code can be used to extend the time available for asserting the external vector on the BCI (this is the IDENT ARB cycle). The receipt of the Externa Vector EV code does not confirm that this node will be transmitting vector for this IDENT, since the outcome of the IDENT arbitration is not yet known.

The IDENT ARB Lost EV code can be used to return the slave port interface logic to an idle state. For example, the node could always stall for at least two cycles and then obtain the vector only if the IDENT ARB Lost EV code was not received. This use of the IDENT ARB Lost EV code is illustrated in Figure 18-1. In this figure, master node 3 (M3) performs an IDENT that selects both slave node 1 (S1) and slave node 2 (S2). Both nodes arbitrate in the IDENT ARB cycle, and S1 wins due to its higher priority (that is, its lower ID number). S1 stalls the required minimum of one cycle,* then provides the interrupt vector two cycles following the IDENT ARB. Meanwhile, S2 is still asserting the STALL response that it first asserted during the IDENT ARB cycle. The STALL code was asserted for one additional cycle to allow time for the IDENT ARB Lost EV code (IAL) to be output so that S2 could avoid fetching a vector that would not be transmitted during this IDENT. Since S2 did in fact lose the IDENT ARB, the S2 STALL responses only served to extend the transaction. If a node loses the IDENT ARB, then the BIIC resends the INTR transaction if the request is still pending.

The winner of the IDENT arbitration transmits the interrupt vecto The slave port interface can stall if the vector is not immediately available. After receipt of the vector has been acknowledged by the master of the IDENT transaction, the appropriate ACK Received for Vector EV code is output by the BIIC on the EV<4:0> L lines, and the INTRC bit in the User Interface Interrupt Control Register is set.

---

*Nodes that use external vectors must stall all vectors at least one cycle.

digital™

The slave port interface can decode this class of EV code and use these signals to deassert the appropriate interrupt request. Alternatively, a software service routine can clear the interrupt condition. It is not necessary to clear the interrupt request immediately because another interrupt will not be generated for that BCI INT<7:4> L line until the line is deasserted and then reasserted. When the INT line is deasserted, the Sent and INTRC bits in the User Interface Interrupt Control Register are cleared.

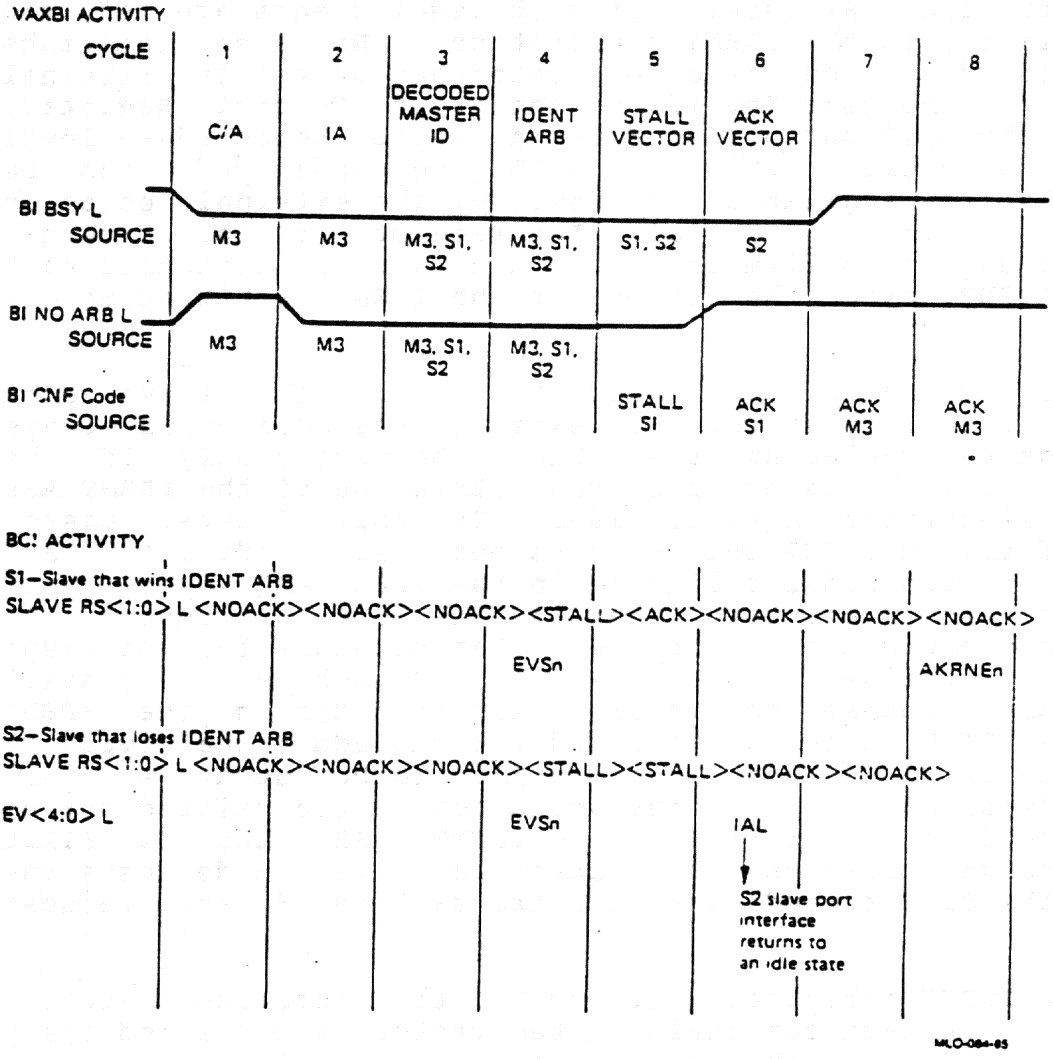


Figure 18-1: IDENT EV Code Timing

digital™

A user interface that OR's multiple interrupt sources onto a single interrupt request line can ensure that interrupts are not lost (because of the deassertion requirement) by requiring the interrupt service routine to clear the INTRC and Sent bits after an interrupt condition is serviced.

If the transmission of the vector fails and the IDENTing master does not send the final ACK response, then the slave port interface outputs the Illegal CNF Received for Slave Data EV code. The BIIC then automatically resends the INTR, if the interrupt request is still pending. In this case, the master may resolicit the same vector in a future IDENT. As long as the node retains the vector, the system can recover from transient bus errors that caused a vector transmission to fail.

The BIIC does not arbitrate for pending INTR transactions during the imbedded ARB cycle of an IDENT transaction. If it did, it would be possible for a node with a posted (but not yet transmitted) interrupt to arbitrate and win the imbedded ARB of an IDENT transaction and then participate in the present IDENT transaction, win the IDENT ARB and be serviced (recall that INTR transactions do not have to be transmitted, only posted, in order for a node to participate in an IDENT transaction). This would leave that node with a pending master state that was no longer required (because the INTR was already serviced). The limitation on the use of IDENT imbedded ARBs avoids this situation.

digital™

18.3.4 INTR Sequence

Figure 18-2 shows the interaction of the BIIC and user interface during an INTR transaction. The following items are keyed to the numbers in Figure 18-2.

1. At this point the INTR Sent, INTRAB, and INTRC bits at the INTR level have been set by the BIIC. (The INTR Force bit is also set if it was used to generate the interrupt request.) Because of the set INTRC bit, this node will not respond to IDENTs at this level until the INTR transaction is reattempted. To reattempt the INTR transaction, the user interface must do one of the following:

   - Deassert the BCI INT<7:4> request for a cycle (this clears the INTRC and INTR Sent bits) and reassert the INT<7:4> request. The equivalent operation could be performed for a force-bit requested interrupt.

   - Clear the INTRC and INTR Sent bits and leave the INT<7:4> request asserted (or INTR Force bit set if applicable).

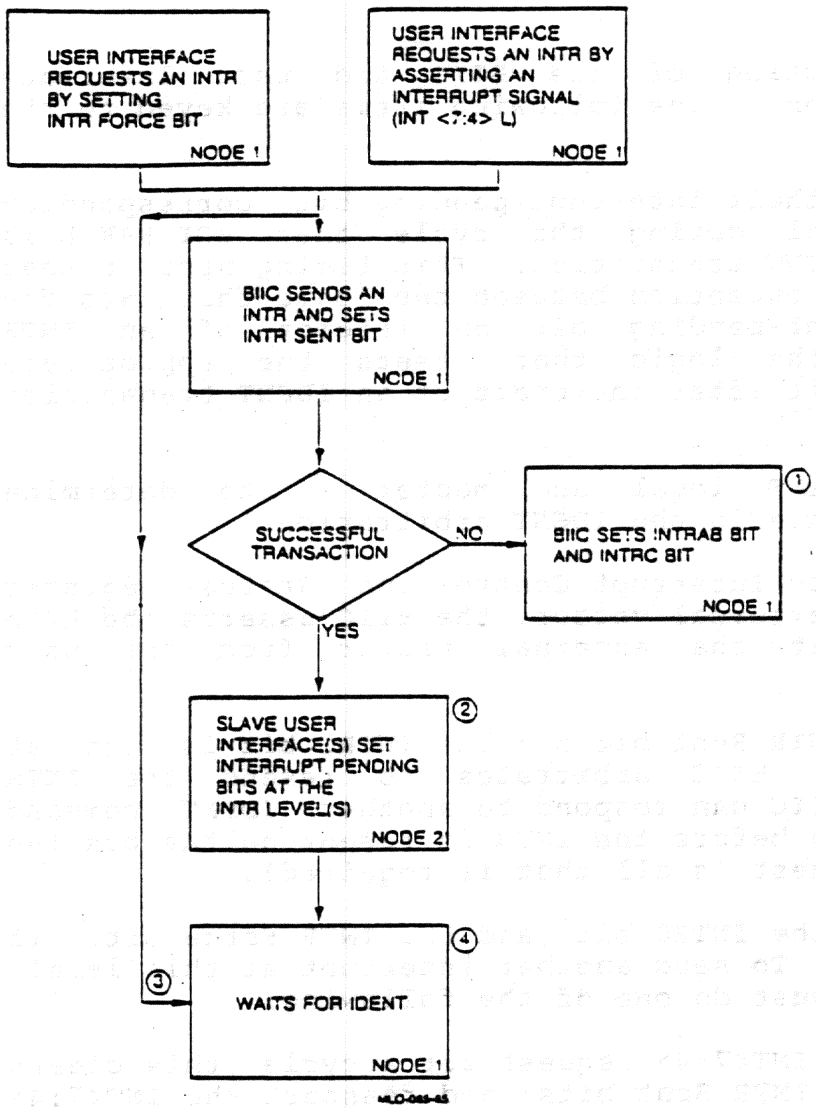
   The BIIC will then resend the INTR transaction.

2. The slave user interface can obtain the level information from command/address data latched on BCI CLE H. The receipt of the INTR SC code (or SEL in conjunction with an INTR command code) must be used to gate the setting of the interrupt pending bits. This assures that the bits are loaded only if good command/address parity was received.

3. A node can respond to IDENTs at interrupt levels whose corresponding INTR transactions have not yet been sent out on the VAXBI bus (that is, only the request has been posted).

4. As long as an INTR Sent bit remains asserted, no more interrupts at the corresponding level will be sent out from this node. The user interface can clear the INTR Sent bit by deasserting the interrupt request.

digital™



NODE 1 Node sending the INTR

NODE 2 Node responding to the INTR

Figure 18-2: INTR Sequence

digital™

18.3.5 IDENT Sequence

Figure 18-3 shows the interaction of the BIIC and user interface during an IDENT transaction. The following items are keyed to the numbers in Figure 18-3.

1. Nodes should reset their interrupt-pending bit corresponding to the IDENT level during the cycle that BCI RAK L is asserted for the IDENT transaction. This timing assures that there will be no contention between the logic that sets the appropriate interrupt-pending bit on receipt of an INTR transaction and the logic that resets the appropriate interrupt-pending bit after the start of an IDENT transaction from this node.

2. The BIIC checks IDENT level and master ID to determine whether to participate in the IDENT arbitration.

3. If the User Interface Interrupt Control and Status Register is configured for external vector, the BIIC asserts the EVSn EV code to solicit the external vector from the user interface.

4. Since neither the INTR Sent bit nor the INTRC bit is set at this point, the BIIC arbitrates to resend the INTR transaction. The BIIC can respond to another IDENT command at this level even before the INTR is resent on the bus (an unserviced INTR request is all that is required).

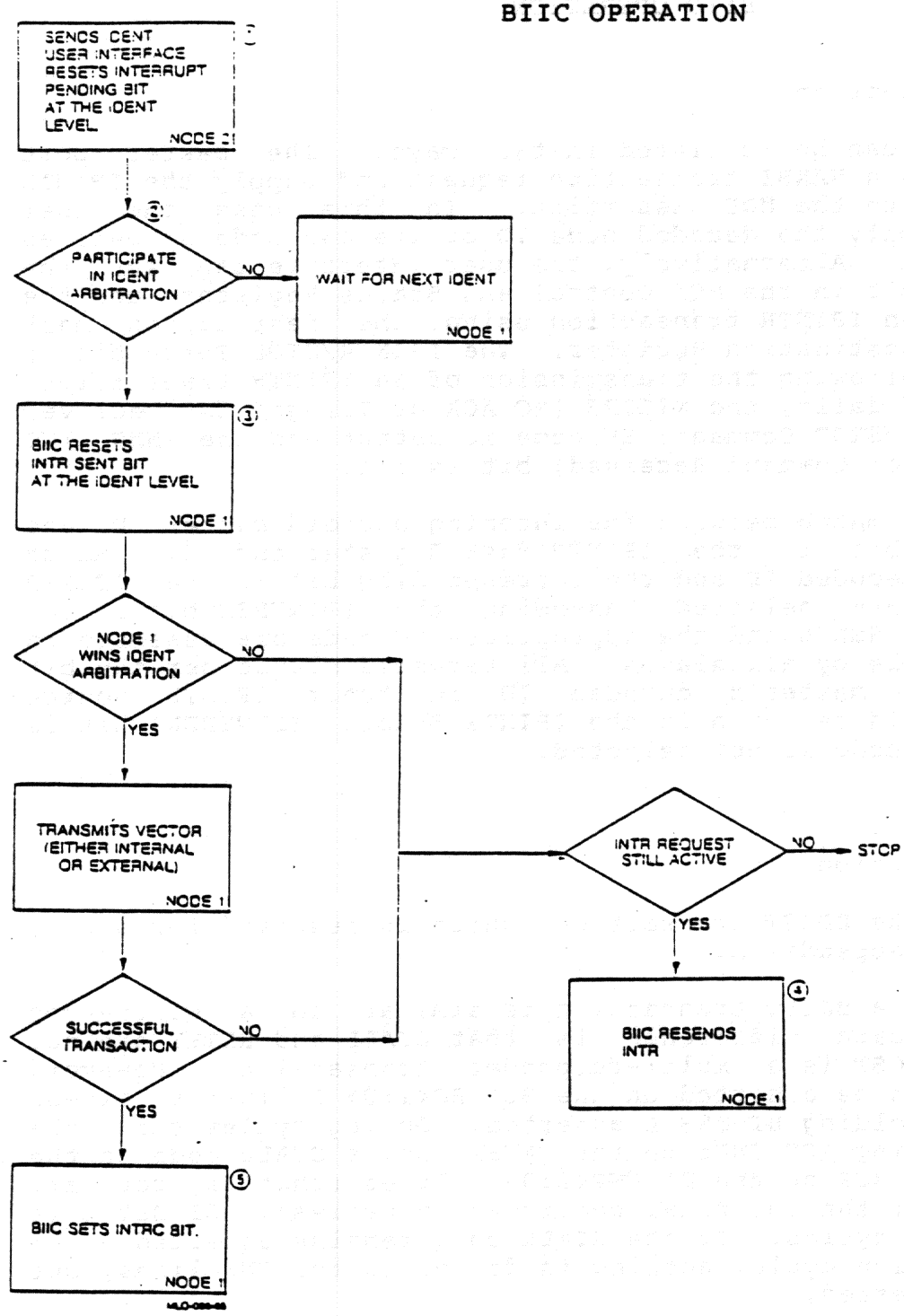
5. At this point only the INTRC bit (and the INTR Force bit, if applicable) is set. To send another interrupt at this level, the user interface must do one of the following:

   - Deassert the BCI INT<7:4> request for a cycle (this clears the INTRC and INTR Sent bits) and reassert the INT<7:4> request. The equivalent operation could be performed for a force-bit requested interrupt.

   - Clear the INTRC and INTR Sent bits and leave the INT<7:4> request asserted (or INTR Force bit set if applicable).

   The BIIC will then send another INTR transaction.

digital™



NODE 1 Node that sent the INTR

NODE 2 Node responding to the INTR

Figure 18-3: IDENT Sequence

digital™

18.3.6 IPINTR Transaction

IPINTR transactions can be initiated in two ways. The master port interface can make a VAXBI transaction request and supply the IPINTR command in response to the MDE assertion. In this case the user interface must supply the decoded node ID of its own node as well as the destination mask. Alternatively, the user interface can set the IPINTR/STOP Force bit in the BCI Control and Status Register, and the BIIC will generate an IPINTR transaction using the destination mask from the IPINTR Destination Register. The IPINTR/STOP Force bit is reset by the BIIC following the transmission of an IPINTR transaction. If the transmission fails, the NICIPS (NO ACK or Illegal CNF Received for Force-Bit IPINTR/STOP Command) EV code is output and the NMR (NO ACK to Multi-Responder Command Received) bit is set.

All nodes with (1) a match between the incoming decoded master ID and the corresponding bit in the IPINTR Mask Register and (2) a match between the node's decoded ID and the corresponding bit in the IPINTR Destination field are selected (assuming the IPINTREN bit in the BCICSR is set). BCI SEL L and the appropriate SC code are asserted in the imbedded ARB cycle by all slaves. All targeted slaves set the bit corresponding to the master's decoded ID in their IPINTR Source Register. This bit is set even if the IPINTR Enable (IPINTREN) bit is cleared so that the node is not selected.

18.3.7 BDCST Transaction

The description of the BDCST transaction, which is reserved for use by Digital, appears in Appendix A.

The BIIC response to a BDCST transaction is similar to a write-type transaction. The main difference is that STALL and RETRY are not legal CNFs, since BDCST is a multi-responder transaction. However, the STALL code can be asserted on the BCI RS<1:0> L lines to extend the transaction by holding BI BSY L asserted. During cycles that the slave node is asserting ACK CNFs on the VAXBI bus, a STALL code on the RS lines produces an ACK on the BI CNF<2:0> L lines (that is, for all data cycles and for the two final confirmation cycles). BI BSY L is held asserted in all cycles. If the STALL code remains asserted after the last confirmation cycle, nothing is driven on the CNF lines, but BI BSY L remains asserted.

18.3.8 STOP Transaction

The STOP command is initiated by a master port interface request, and the user interface provides the destination mask and STOP command code for transmission during the command/address cycle. Slaves are

digital™

selected on the basis of a match between the slave's decoded ID and the corresponding bit in the Destination Mask field. The slave port interface provides the command confirmation on the BCI response lines. Regardless of the response, all slaves perform the following internal initialization steps:

1. A pending master state, if present, is aborted, and the BIIC's assertion of BI NO ARB L is terminated.

2. The BIIC's master and slave sequencers are reset. A consequence of this is that a master port interface that sends a STOP to its own slave port interface does not receive a summary EV code, and BCI RAK L is removed prematurely.

3. All posted interrupt states are cleared. This means that all Sent and Force bits in the User Interface and Error Interrupt Control Registers are cleared.

4. The INIT bit in the VAXBI Control and Status Register is set

5. The RETRY state, if any, is cleared.

6. The RETRY counter is reset.

7. The HEIE and SEIE bits in the VAXBI Control and Status Register are cleared.

Resetting the STOPEN bit in the BCI Control and Status Register suppresses the generation of BCI SEL L and the BCI SC<2:0> L codes on the slave port and the performance of the initialization steps described above.

18.3.9 Invalidate (INVAL) Transaction

INVAL commands are initiated by a master port interface request. The user interface provides an address and a data length code (indicating the number of longwords to be invalidated) when BCI MDE L is asserted. All slaves with the INVAL Enable (INVALEN) bit set in the BCI Control and Status Register are selected by an INVAL transaction.

18.3.10 RESERVED Commands

The BIIC treats all three RESERVED commands as three-cycle VAXBI transactions, each consisting of a command/address cycle (containing user-interface-supplied data), an imbedded ARB cycle, and a data cycle with all data lines deasserted. The master expects an ACK command confirmation from a slave. The MCP (ACK), NCRMC (NO ACK), or ICRMC

(neither ACK nor NO ACK) EV code is output depending on the command confirmation received.

A slave can respond to the HLHL and HLHH RESERVED command codes if the RESERVED Enable (RESEN) bit is set in the BCICSR. ACK, STALL (multi-responder extension case -- STALL converts to ACK for command confirmation cycle), and NO ACK are permissible RS responses. The only EV codes that are output are Bus BSY Error (BBE) and Stall Timeout on Slave Transaction (STO). The BIIC cannot respond as a slave to the LLLL code regardless of the setting of the RESEN bit.

18.4 TRANSACTION PRIORITY

The BIIC supports multiple pending requests. For example, a VAXBI transaction request, an error INTR request, an IPINTR request, and a user interface INTR request can all be pending at the same time. For any combination of pending requests that can occur, the priority for processing the requests is as follows:

1. Loopback requests from the master port interface (highest priority)
2. INTR requests controlled by the Error Interrupt Control Register
3. INTR requests from the User Interface Interrupt Control Register or BCI INT line (Level 7)
4. INTR requests from the User Interface Interrupt Control Register or BCI INT line (Level 6)
5. INTR requests from the User Interface Interrupt Control Register or BCI INT line (Level 5)
6. INTR requests from the User Interface Interrupt Control Register or BCI INT line (Level 4)
7. VAXBI transaction requests from the master port interface
8. IPINTR requests from the BCI Control and Status Register

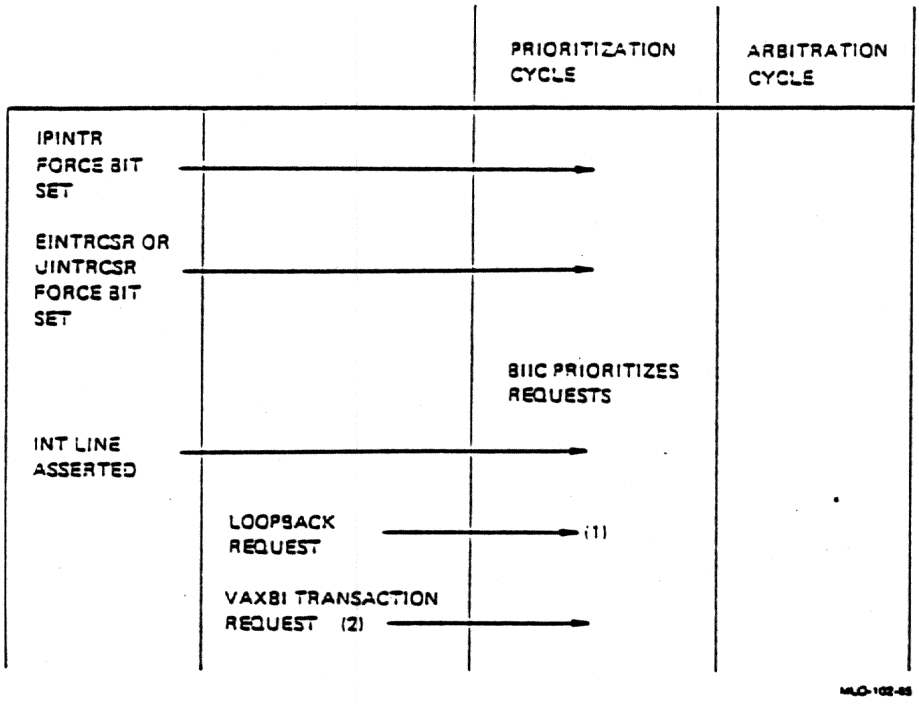
Figure 18-4 shows how the BIIC prioritizes transaction requests. Each column represents a different type of request and the timing relationship between the posting of the request and the priority that the BIIC gives to the request.

Various delays are associated with each type of request between the time the transaction request is posted and the time that the BIIC acts on that request. During the prioritization cycle, the BIIC examines

digital™

the requests and determines which request has the highest priority. If no requests are pending during the prioritization cycle, the BIIC enables immediate recognition of master port VAXBI transaction requests during the following cycle, bypassing the prioritization logic, in order to minimize the typical bus access latency for VAXBI transaction requests.

If the VAXBI transaction request is posted while other types of requests are present, the VAXBI transaction request is prioritized along with the other requests during the prioritization cycle. If the VAXBI transaction request is posted when no other types of request are present in the previous cycle, the BIIC attempts to arbitrate in the next cycle.



NOTES:

1. Loopback transactions have a dummy ARB cycle at this point.
2. If the VAXBI transaction request is posted while other types of requests are present, the BIIC prioritizes the VAXBI transaction request along with the other requests during the prioritization cycle. However, if no other types of requests are present, the BIIC attempts to arbitrate in the next cycle.

Figure 18-4: BIIC Request Prioritization

digital™

CHAPTER 19

PACKAGING INFORMATION

Figure 19-1 shows a packaged BIIC with heat sink. The BIIC package shown here has a die to ambient thermal resistance of 9 degrees C/W at 200 linear feet per minute. Figure 19-2 shows the pin grid array for the BIIC's 133 pins.

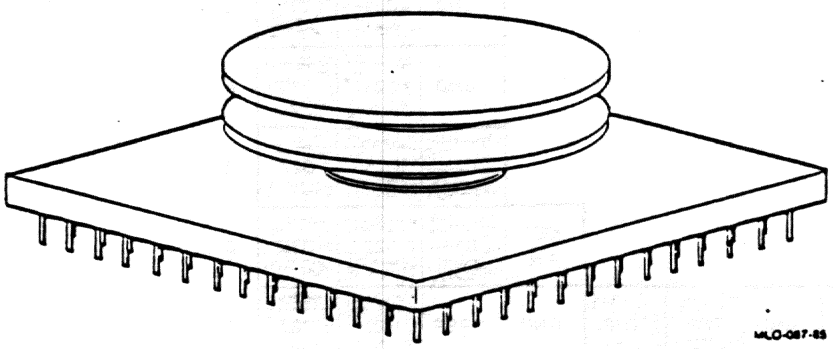


Figure 19-1: Packaged BIIC with Heat Sink

digital™

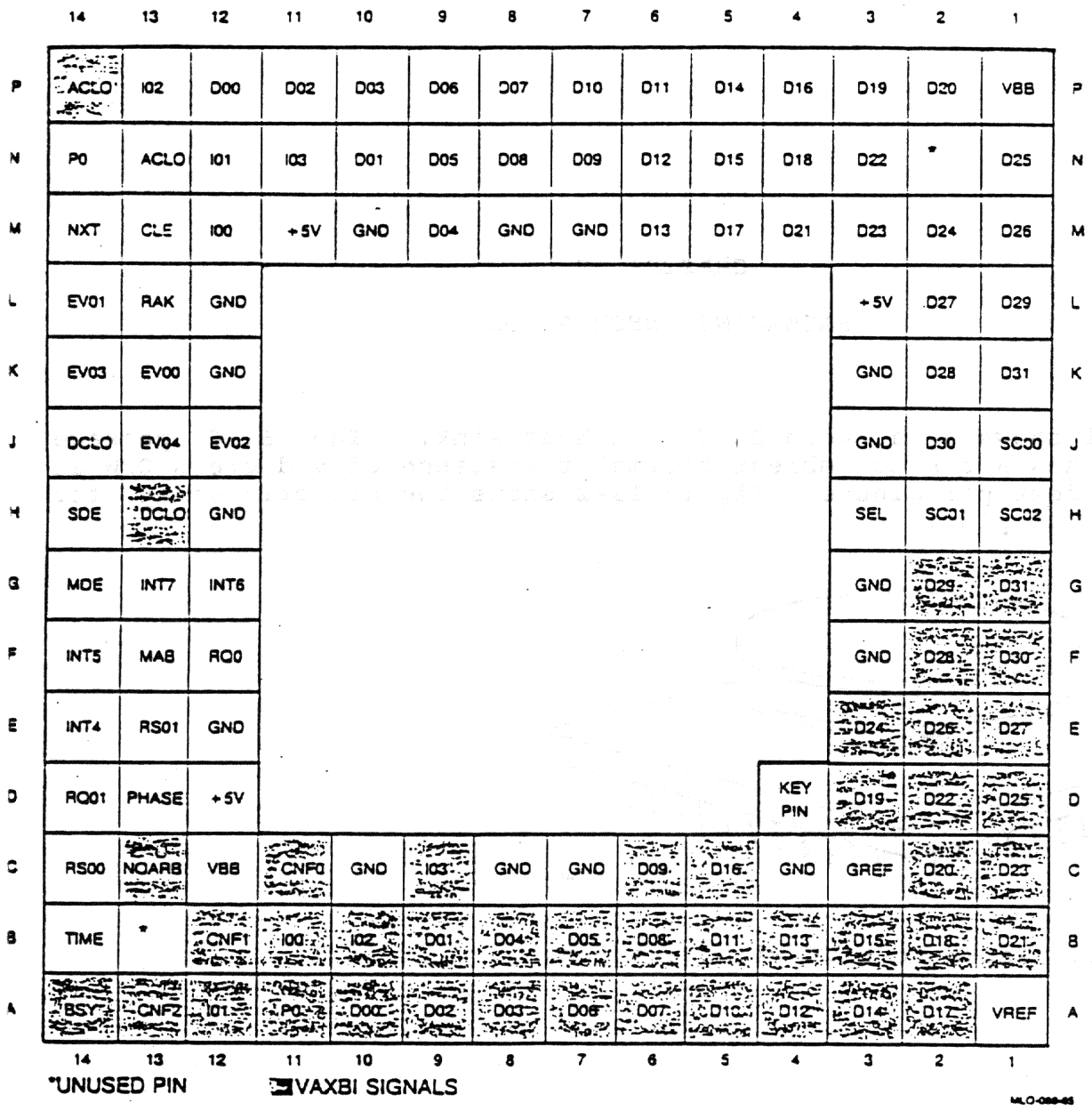


Figure 19-2: BIIC Pin Grid Array (top view)

CHAPTER 20

ELECTRICAL CHARACTERISTICS

This chapter presents the electrical specifications for the BIIC.

20.1 ABSOLUTE MAXIMUM RATINGS

Stresses greater than those listed in the absolute maximum ratings may cause permanent damage to a device. Furthermore, exposure to maximum rating conditions for extended periods may affect device reliability.

Pin voltages	-1.5 to +7.0 V
Operating junction temperature range	0 to 125 degrees C
Storage temperature range	-55 to 125 degrees C
Ambient temperature cperating range	0 to 70 degrees C
Package dissipation	4.25 watts*

*Package dissipation is approximately .575 watts higher than the product of the maximum supply current and supply voltage would indicate. This additional .575 watts is dissipated in the VAXBI drivers, which sink the external VAXBI pullup current.

digital™

20.2 DC CHARACTERISTICS

Unless otherwise specified, all specifications are at Ta = 0 to 70 degrees C and Vcc = 4.75 to 5.25 V.

Symbol	Parameter	Min.	Max.	Unit	Remarks
BCI Ii	Input current	-.33	.10	mA	.2 < Vin < 5.25 V 0 < Vcc < 5.25 V Note 4
BCI Iid	Input current while BCI DC LO L is asserted	-	-.25	mA	Vin = 2.4 V Note 1
		-1.0	-	mA	Vin = 0.5 V Note 1
BCI Ioh	Output high current (except BCI DC LO L)	-400	-	uA	Vout = BCI Voh
BCI Iohd	Output high current (only for BCI DC LO L)	-5.4	-	mA	Vout = BCI Voh
BCI Iol	Output low current	4.0	-	mA	Vout = BCI Vol
BCI Iold	Output low current (only for BCI DC LO L; when Vcc < Vcc min)	100	-	uA	Vout = BCI Vol 0 < Vcc < 4.75
BCI Vil	Input low voltage	-1.0	0.8	V	-
BCI Vih	Input high voltage (except BCI TIME L)	2.0	-	V	-
BCI Viht	Input high voltage (only for BCI TIME L)	2.4	-	V	-
BCI Voh	Output high voltage	2.7	-	V	Iout = BCI Ioh
BCI Vol	Output low voltage	-	0.5	V	Iout = BCI Iol
BCI Cio	Pin capacitance	-	10	pF	0 < Vio < Vcc

digital™

ELECTRICAL CHARACTERISTICS

Symbol	Parameter	Min.	Max.	Unit	Remarks
BI Ii	Input current	-272	+30	uA	0 < Vio < BI Voh Note 4
BI Iol	Output low current	21	-	mA	Vout = BI Vol
BI Vol	Output low voltage	-	0.6	V	Iout = BI Iol
BI Voh	Bus voltage high	2.3	3.5	V	These are bus terminator spec
BI Vih	Input high voltage	1.95	-	V	-
BI Vhhy	Input high voltage (hysteresis voltage)	1.45	-	V	Note 2
BI Vil	Input low voltage	-1.00	1.10	V	-
BI Vlhy	Input low voltage (hysteresis voltage)	-	1.40	V	Note 2
BI Cio	Pin capacitance	-	6.0	pF	Vio = 2.5 V Note 3
Icc	Power supply current	-	720	mA	Iout=0, Tj=27 C
Icc	Power supply current	-	530	mA	Iout=0, Tj=85 C Note 5

NOTES

1. While BCI DC LO L is asserted, the BCI D and P lines are internally pulled up. While pulled up, these lines can source a minimum of 250 uA @ 2.4 V. If the user interface desires to drive any of these lines low during the time BCI DC LO L is asserted, then the user interface logic must sink a minimum of 1.0 mA @ 0.5 V.

2. The hysteresis voltage is defined as follows:

   For BI Vih - The BIIC will not detect a change in input state even if following the application of BI Vih, the input voltage drops to BI Vhhy.

   For BI Vil - The BIIC will not detect a change in input state even if following the application of BI Vil, the input voltage rises to BI Vlhy.

3. The device under test must be powered up during this test and BI DC LO L should be asserted at all times, except when

digital™

measuring Cio for BI DC LO L.

4. These Ii specs apply only when data is not being driven by the output under test.

5. Tj = 27 degrees C is the junction temperature with an ambient temperature of Ta = 0 degrees C.

digital™

20.3 AC TIMING SPECIFICATIONS

Unless otherwise specified, test conditions are at Ta = 0 to 70 degrees C, Vcc = 4.75 to 5.25 V, and BCI signal load = 50 pF.

Figure 20-1 shows the BIIC AC timing. The next two figures show the test load configurations: Figure 20-2, the BCI side and Figure 20-3, the VAXBI side. Figures 20-4 through 20-9 show measurement waveforms.

digital™

BIIC AC Timing Specifications

Symbol	Parameter	Min.	Max.	Unit	Remarks
Tcy	TIME period	49	1000	ns	Note 1
Tcp	TIME pulse width	15	-	ns	
Tps	PHASE setup time to TIME	10	-	ns	
Tph	PHASE hold time from TIME	10	-	ns	
Tr	BCI signal rise time	-	10	ns	Figure 20-9, Note 2
Tf	BCI signal fall time	-	10	ns	Figure 20-9, Note 2
Tp1	BCI signal propagation delay from TIME (except for BCI MDE L and BCI SDE L)	0	30	ns	Figure 20-7, Note 2
Tp2	BCI D, I, and P line signal propagation delay from TIME	0	Tcy+30	ns	Figure 20-7, Note 2
Tp3	BCI MDE L and BCI SDE L signal propagation delay from TIME	0	40	ns	Figure 20-7, Note 2
Tdas	BCI DC LO L assertion delay from BI DC LO L assertion	0	150	ns	
Tdde	BCI DC LO L deassertion delay from BI DC LO L deassertion	45	55	us	
Tnrw	BCI DC LO L assertion width following the setting of the NRST bit	45	55	us	
Tsp	BCI D and I setup time with BIIC configured for BIIC-generated parity	20	-	ns	Figure 20-8
Ts	BCI signal setup time with BIIC configured for user interface to supply parity	0	-	ns	Figure 20-8

BIIC AC TIMING SPECIFICATION NOTES

1. This minimum specification allows for proper BIIC operation in environments with up to +/-1 ns of clock period noise jitter.

2. The Tr, Tf, Tp1, Tp2, and Tp3 parameters specified in this table are for 50 pF loads. The following equations describe the degradation in rise and fall times and propagation delays for a BCI line loaded with between 50 and 100 pF.

   Tr (50 pF < Cl < 100 pF) = Tr (50 pF) + Cl/17.8 - 2.8
   Tf (50 pF < Cl < 100 pF) = Tf (50 pF) + Cl/17.8 - 2.8
   Tp1 (50 pF < Cl < 100 pF) = Tp1 (50 pF) + Cl/5.5 - 9.1
   Tp2 (50 pF < Cl < 100 pF) = Tp2 (50 pF) + Cl/5.5 - 9.1
   Tp3 (50 pF < Cl < 100 pF) = Tp3 (50 pF) + Cl/5.5 - 9.1

BIIC AC Timing Specifications (Cont.)

Symbol	Parameter	Min.	Max.	Unit	Remarks
Th	BCI signal hold time from TIME	15	-	ns	Figure 20-8
Tbcr	BCI D and I release time from TIME	0	40	ns	-
Tbre	BCI D and I release time to MDE or SDE assertion	0	-	ns	-
Tssm	SDE (MDE) deassertion setup time to MDE (SDE) assertion	Tcy-15	-	ns	Note 3
Tdp	BCI NXT, MDE, and SDE deassertion pulse width	Tcy-10	-	ns	-
Tdh	BCI D and I hold time from CLE and NXT	Tcy-25	-	ns	-
Tds	BCI D and I setup time to CLE and NXT	Tcy-25	-	ns	-
Taap	Minimum assertion to assertion period (applies to NXT, MDE, and SDE)	190	-	ns	Note 4
Tbs	BI signal setup time to TIME	20	-	ns	Note 5 Figure 20-5
Tbh	BI signal hold time from TIME	20	-	ns	Figure 20-5
Tbas	BI signal assertion delay from TIME	0	85	ns	Figure 20-4
Tbr	BI signal release time from TIME	0	85	ns	Figure 20-4
Tbif	BI signal fall time (Cl = 410 pF)	20	-	ns	Figure 20-6

BIIC AC TIMING SPECIFICATION NOTES

3. Tssm applies only for equally loaded MDE and SDE signals.
4. Taap applies only if the loading on the BCI output is constant over time.
5. Note for testing this component: only one transition is allowed during T100 to T130 on any BI signal.

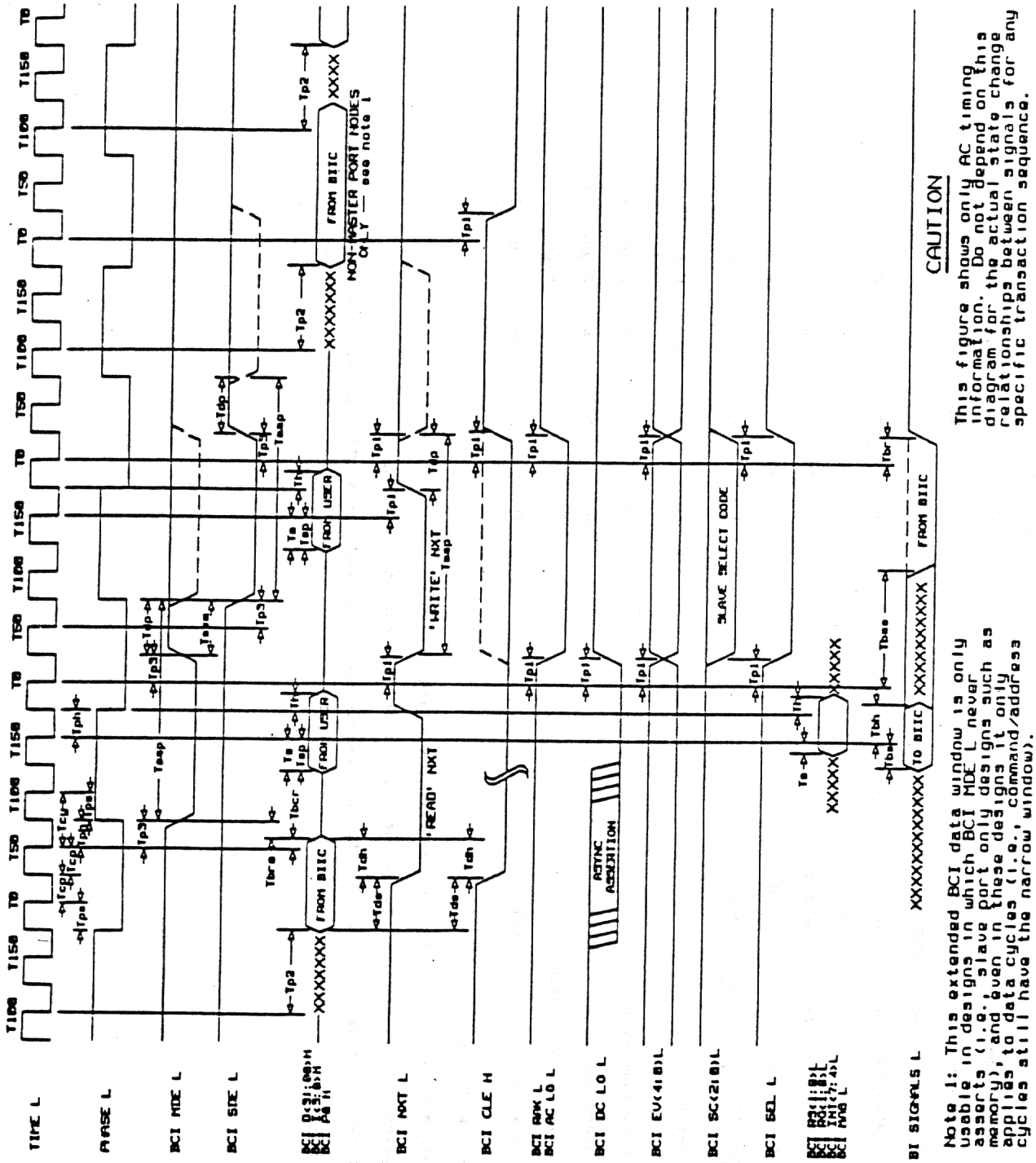


Figure 20-1: BIIC AC Timing Symbol Definitions

digital™

20.4 LOAD CIRCUITS

The circuit of Figure 20-2 must be used as a load for each VAXBI side driver during DC and AC testing.

The circuit of Figure 20-3 must be used as a load for each BCI driver during DC and AC testing.

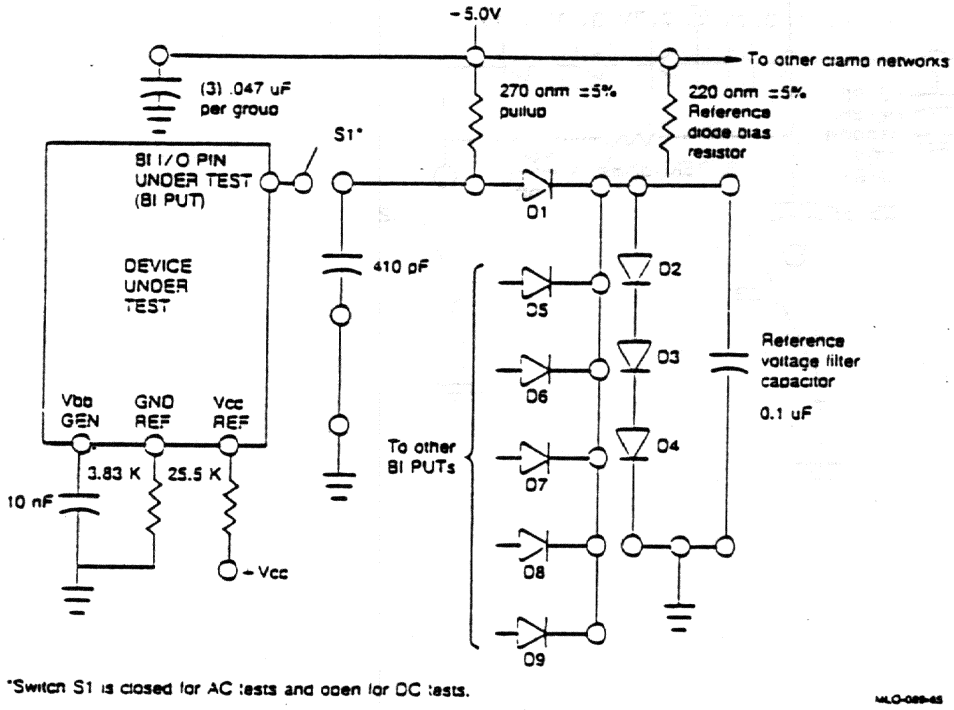
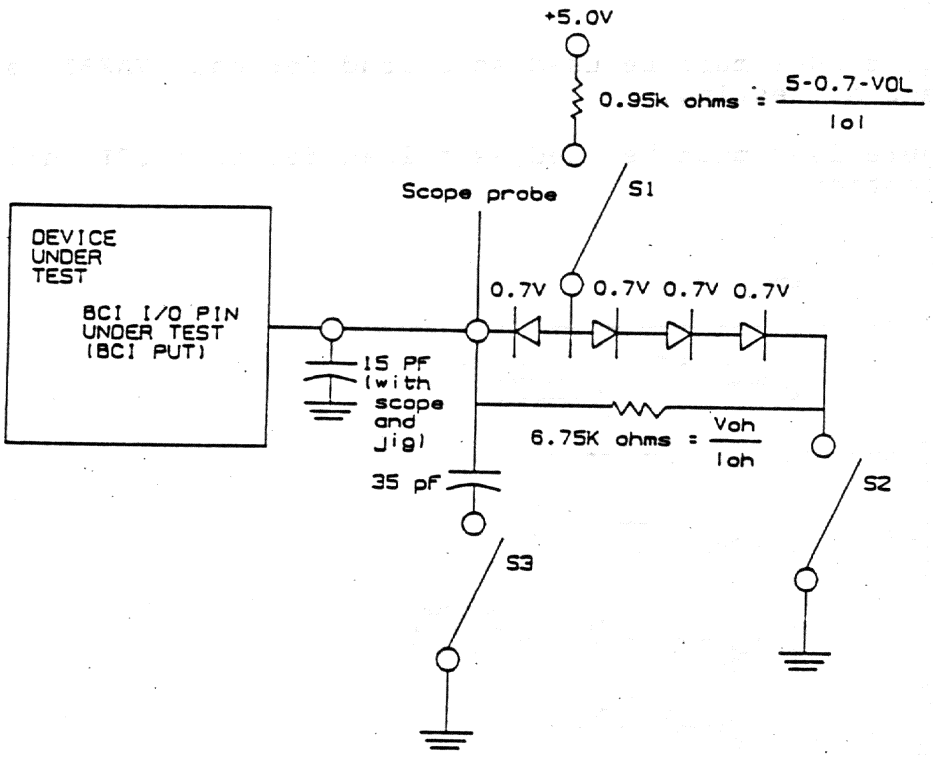


Figure 20-2: Test Fixturing for VAXBI Driver Measurements



BCI Signal Change	S1	S2	S3
0 to 1	OPEN	CLOSED	CLOSED
1 to 0	CLOSED	CLOSED	CLOSED
Z to 0	CLOSED	OPEN	CLOSED
Z to 1	OPEN	CLOSED	CLOSED
0 to Z or 1 to Z	CLOSED	CLOSED	OPEN

NOTE:

The SENTRY test fixture does not match this test configuration.

Figure 20-3: Test Fixturing for BCI Driver Measurements

digital™

20.5 MEASUREMENT WAVEFORMS

Figures 20-4, 20-5, and 20-6 are waveforms for VAXBI signals, and Figures 20-7, 20-8, and 20-9 are waveforms for BCI signals.

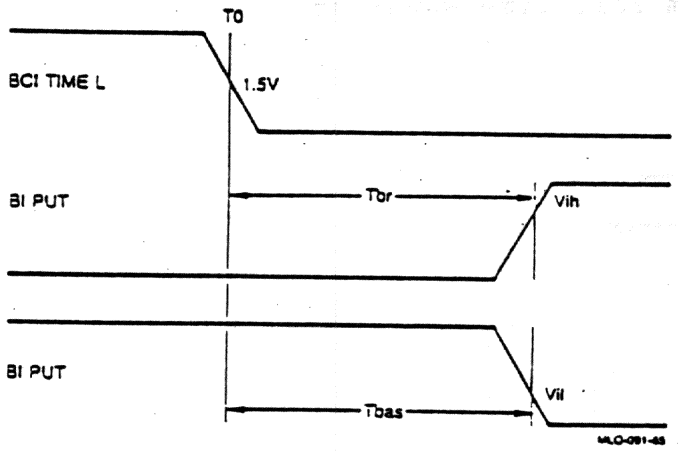


Figure 20-4: VAXBI Driver Output Waveforms

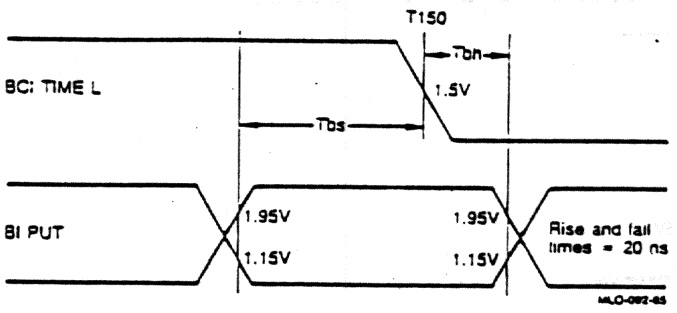


Figure 20-5: VAXBI Receiver Setup and Hold Time Waveforms

digital™

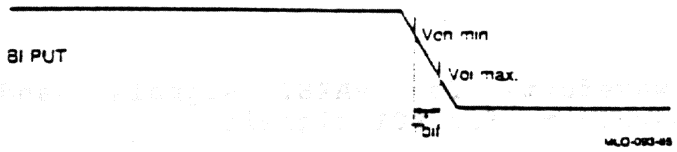


Figure 20-6: VAXBI Driver Minimum Fall Time Waveform

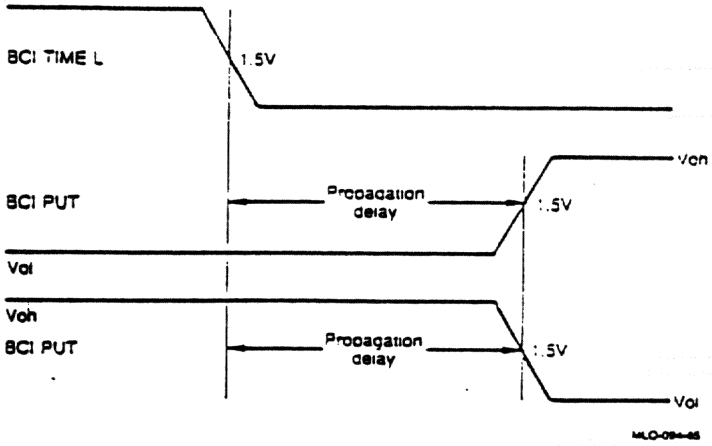
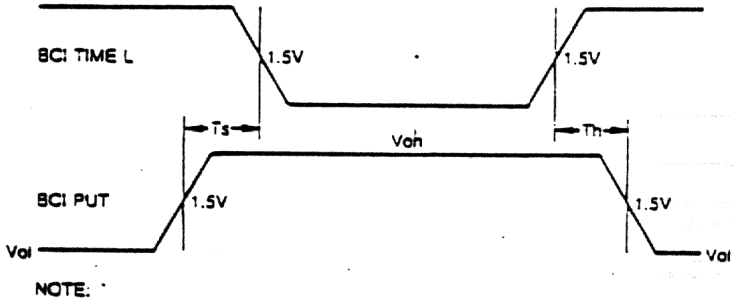


Figure 20-7: BCI Transceiver Propagation Delay Waveforms



NOTE:

BCI input signals should have rise and fail times of < 5 ns (between 0.8 and 2.0 volt points).

Figure 20-8: BCI Receiver Setup and Hold Time Waveforms

digital™

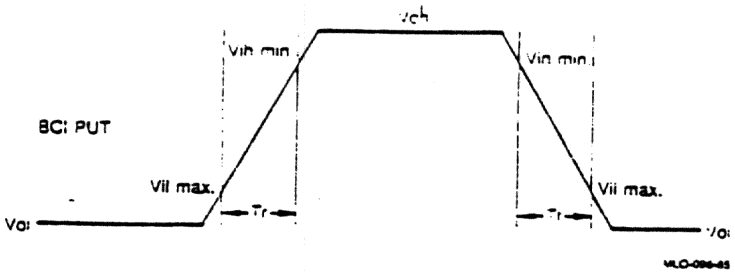


Figure 20-9: BCI Driver Rise and Fall Time Waveforms

digital™

CHAPTER 21

FUNCTIONAL TIMING DIAGRAMS

This chapter contains 28 functional timing diagrams that have been generated for the BIIC. The transactions are listed below along with a short description. Read and write transactions are presented first, followed by other types of transactions.

1. Longword Read-Type with a STALL

2. Loopback Longword Read-Type

   The transaction is to BIIC CSR space with an idle bus. The BICSREN bit is assumed to be reset.

3. Loopback Longword Read-Type with a STALL

   Demonstrates a loopback longword read with a STALL within node 3, which takes place on top of a longword read ("piggy-backed") with a stalled VAXBI transaction between the master port interface at node 1 and the slave port interface at node 2.

4. Retried Longword Read-Type with a STALL

   1. Master 1 transmits a longword read command to slave 2.

   2. Slave 2 sends a RETRY CNF response.

   3. Master 1 reasserts the request to automatically retry the longword read transaction. (Note that the BIIC does nc resolicit the command/address information for the retriec transaction but instead uses the internally stored version).

5. Quadword Read-Type

6. Quadword Read-Type with Pipeline Request

   Master quadword read to same slave node.

digital™

7. Quadword Read-Type with Pending Master

   Quadword read to same slave node.

8. Quadword WRITE

9. Quadword WRITE with Pipeline Request

10. Quadword WRITE with a STALL for Each Longword of Data

11. Octaword WMCI with Variable STALLs for Each Longword of Data

12. Octaword WMCI with Variable STALLs for Each Longword of Data and with Pipeline NXT Enable Bit Set

13. Force-Bit Requested Interrupt

    An INTR transaction initiated by setting one of the force bits in the User Interface Interrupt Control Register.

14. INT<7:4> Requested Interrupt

15. Master Port Interprocessor Interrupt

16. Force-Bit Requested Interprocessor Interrupt

    An IPINTR transaction initiated by setting the IPINTR bit in the BCI Control and Status Register.

17. IDENT with Internal Vector

18. IDENT with External Vector

19. INVAL

20. STOP

    Demonstrates the results of one allowable response to STOP. In this example although BIIC2 wins the imbedded ARB in cycle 5, it does not assert BI NO ARB L in cycle 6 due to the STOP command. Note that if User2 had kept Request asserted, BIIC2 would have arbitrated again in cycle 7 and continued on with the transaction.

21. STOP with Extension

    Demonstrates how the slave port interface can assert a STALL response to hold BI BSY L while node completes STOP operation.

22. Quadword BDCST

23. Burst Mode WRITEs with Pipeline Request

The timing diagram assumes that the Burst Enable bit was set in a previous transaction. The timing diagram begins with the first ARB cycle that this node wins after the bit was set. It shows a quadword write followed by a longword write. The longword write is a WRITE transaction to the BCI Control and Status Register to reset the burst mode bit.

24. Burst Mode WRITEs with Pipeline Request and with Pipeline NXT Enable Bit Set

This timing diagram demonstrates that the BCI CLE H assertion is not always one cycle wide. BCI CLE H is asserted in the cycle following a cycle with BI NO ARB L asserted and BI BSY L deasserted. BCI CLE H is not deasserted until T0 of the command/address cycle.

25. Special Case 1

1. Master 1 wins the arbitration and begins a quadword WMCI transaction to slave 2.

2. Master 2 requests the bus, arbitrates in master 1's imbedded ARB cycle, and becomes the pending master.

3. BIIC2 brings in master 2's command/address data during the imbedded ARB cycle to avoid BCI bus contention.

4. Master 2 performs a longword WMCI to an internal register in slave 2 (an intranode transfer).

26. Special Case 2

Master 1 does a longword read-type transaction to its own slave port interface (slave 1).

digital™

27. Special Case 3

    1. Master 1 and master 2 arbitrate in cycle 3.

    2. Master 1 wins the arbitration and begins a quadword WMCI transaction to slave 2.

    3. Master 2 again arbitrates, this time in master 1's imbedded ARB cycle, and becomes the pending master.

    4. BIIC2 brings in master 2's command/address data during the imbedded ARB cycle to avoid BCI bus contention.

    5. Master 2 performs a longword WMCI to an internal register in slave 2 (an intranode transfer).

28. Special Case 4

    1. Master 2 wins the arbitration and begins a quadword read-type transaction to slave 1.

    2. Master 1 requests the bus, arbitrates during master 1's imbedded ARB cycle, and becomes the pending master.

    3. BIIC1 cannot bring in master 1's command/address data during the imbedded ARB cycle and must wait until SDE is deasserted in cycle 7 before MDE can be asserted.

    4. Master 1 performs a quadword read-type transaction to slave 3.

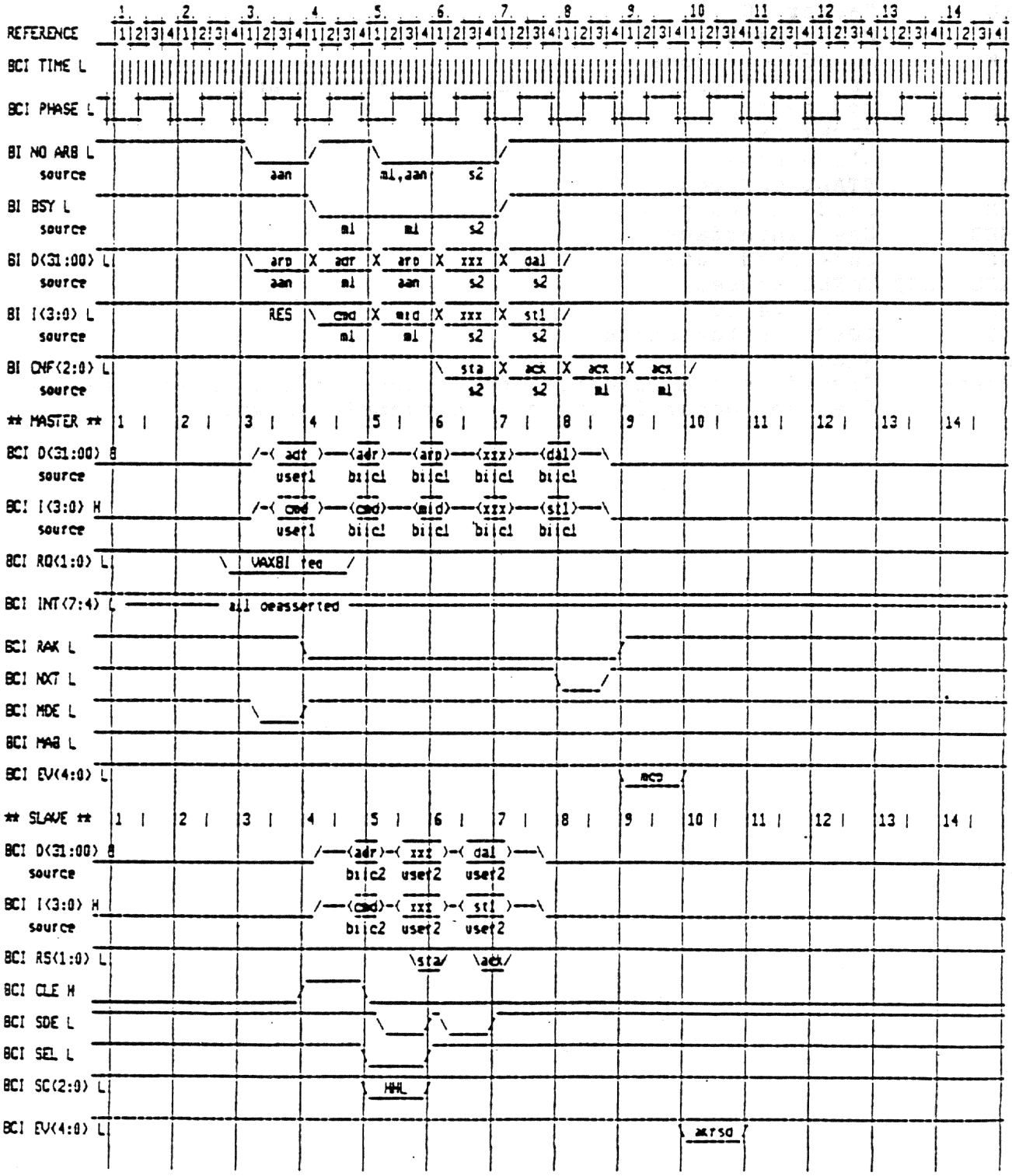
The abbreviations used in the functional timing diagrams are listed below. Event code abbreviations also appear but are not included in the list. Numbers that follow M, S, USER, and BIIC correspond to the node ID.

AAN	All arbitrating nodes
AAS	All arbitrating slaves
ACK	ACK CNF code
ADR	Address, including length field
ARP	ARB pattern (decoded ID from all arbitrating nodes)
CMD	Command
DA1	Data word 1
DMI	Decoded master ID
DSI	Decoded slave ID from all arbitrating nodes (only on D<31:16>)
DSM	Destination mask (including length field for BDCST command)
IDD	Master ID and destination code
ILV	IDENT level(s) on D<19:16>
INTR REQ	INTR request
LB REQ	Loopback request

LDC	Level and destination code
M	Master node
MID	Master ID
MK1	Write mask 1
NAK	NO ACK CNF code
REC	RETRY CNF received for master port command
RES	RESERVED
RET	RETRY CNF code
S	Slave node
STA	STALL CNF code
STS	Read status code
USER	User interface
UDF	Undefined data
VAXBI REQ	VAXBI request
VEC	IDENT vector
VST	IDENT vector status
WS	Winning slave
item	Signifies the potential occurrence of this item; the item does not occur in the timing example shown.

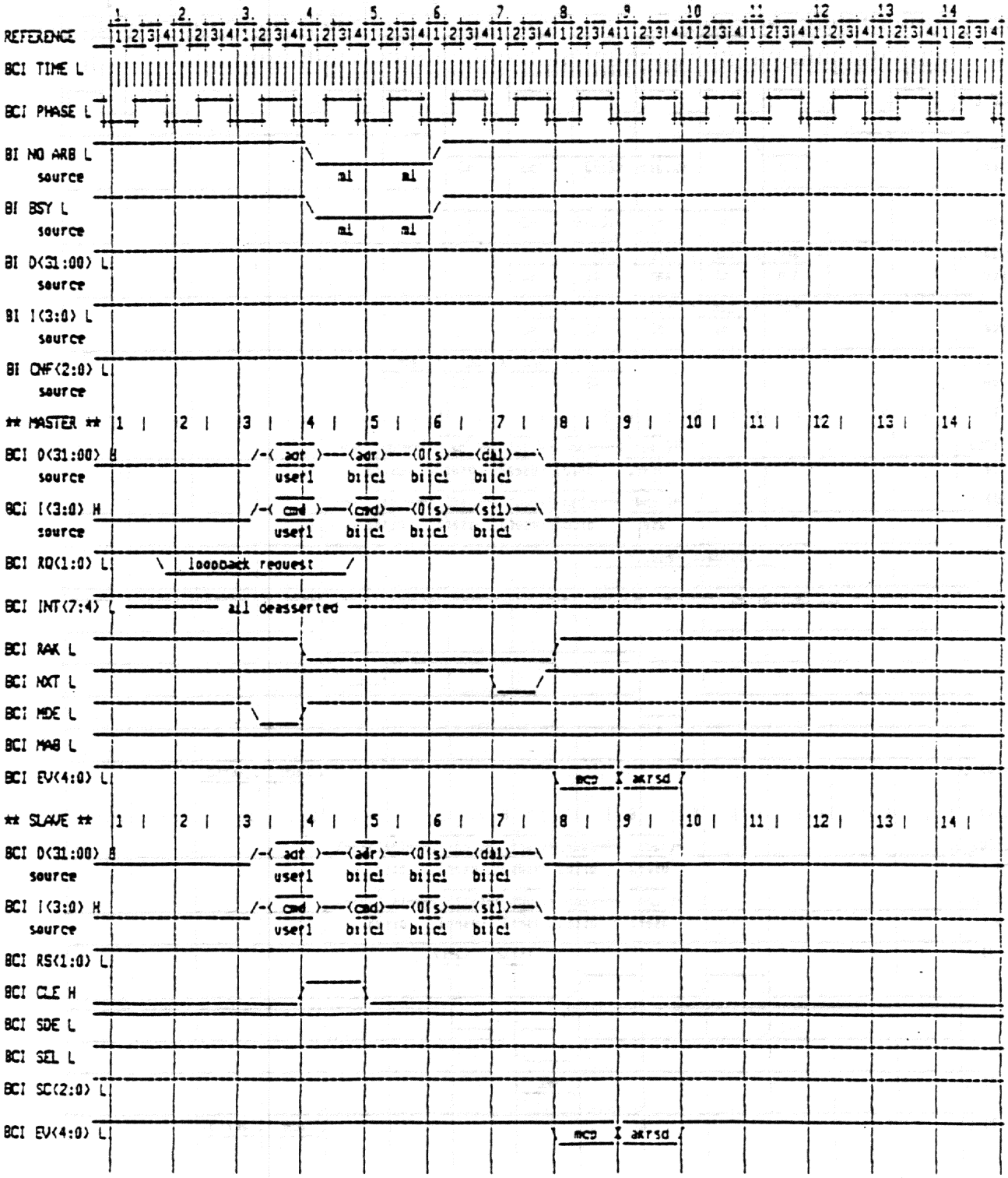
digital™

Figure 21-1: Longword Read-Type with a STALL



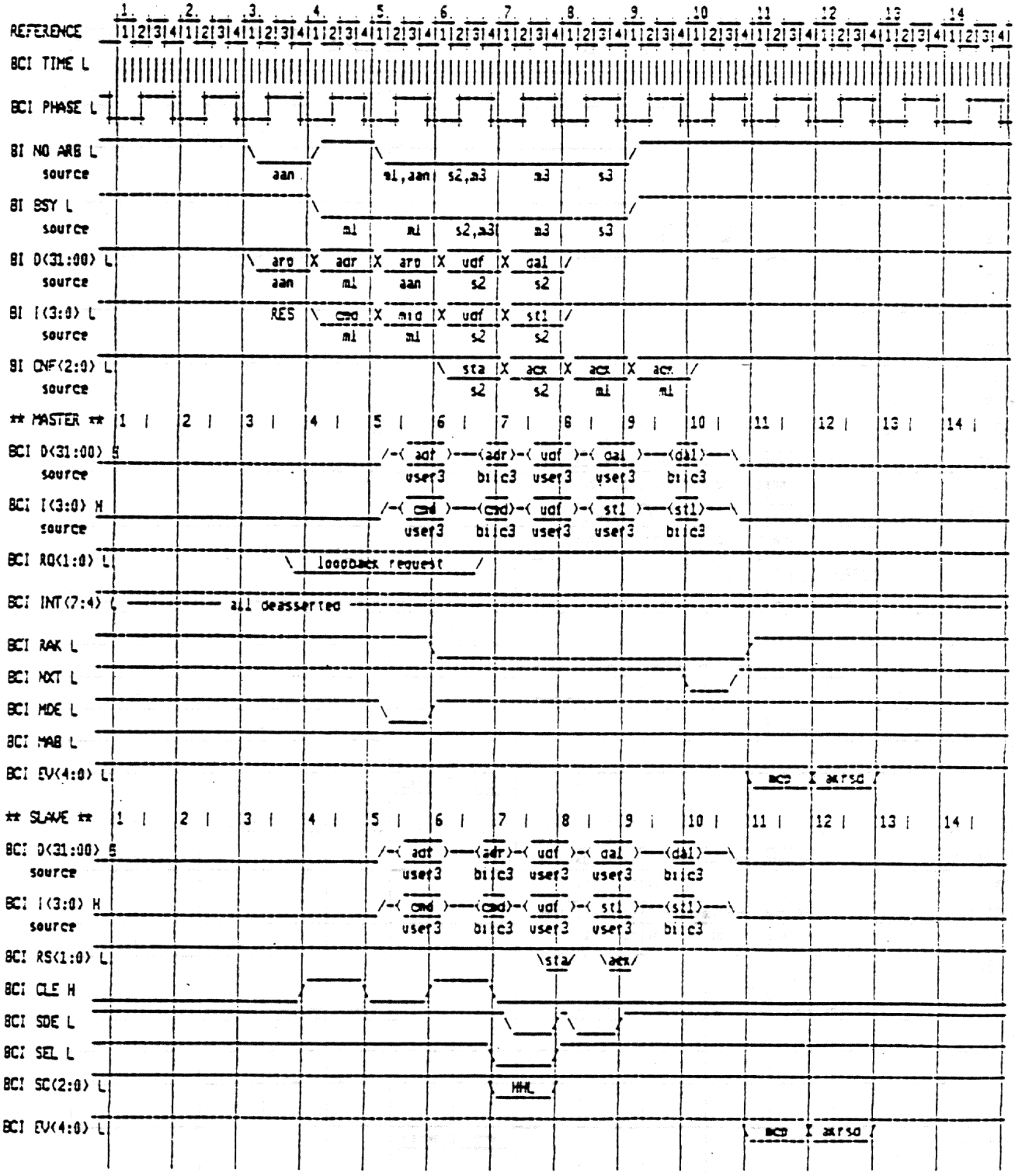
1.TD

Figure 21-2: Loopback Longword Read-Type



2.TD

Figure 21-3: Loopback Longword Read-Type with a STALL



3.TD

digital™

Figure 21-4: Retried Longword Read-Type with a STALL

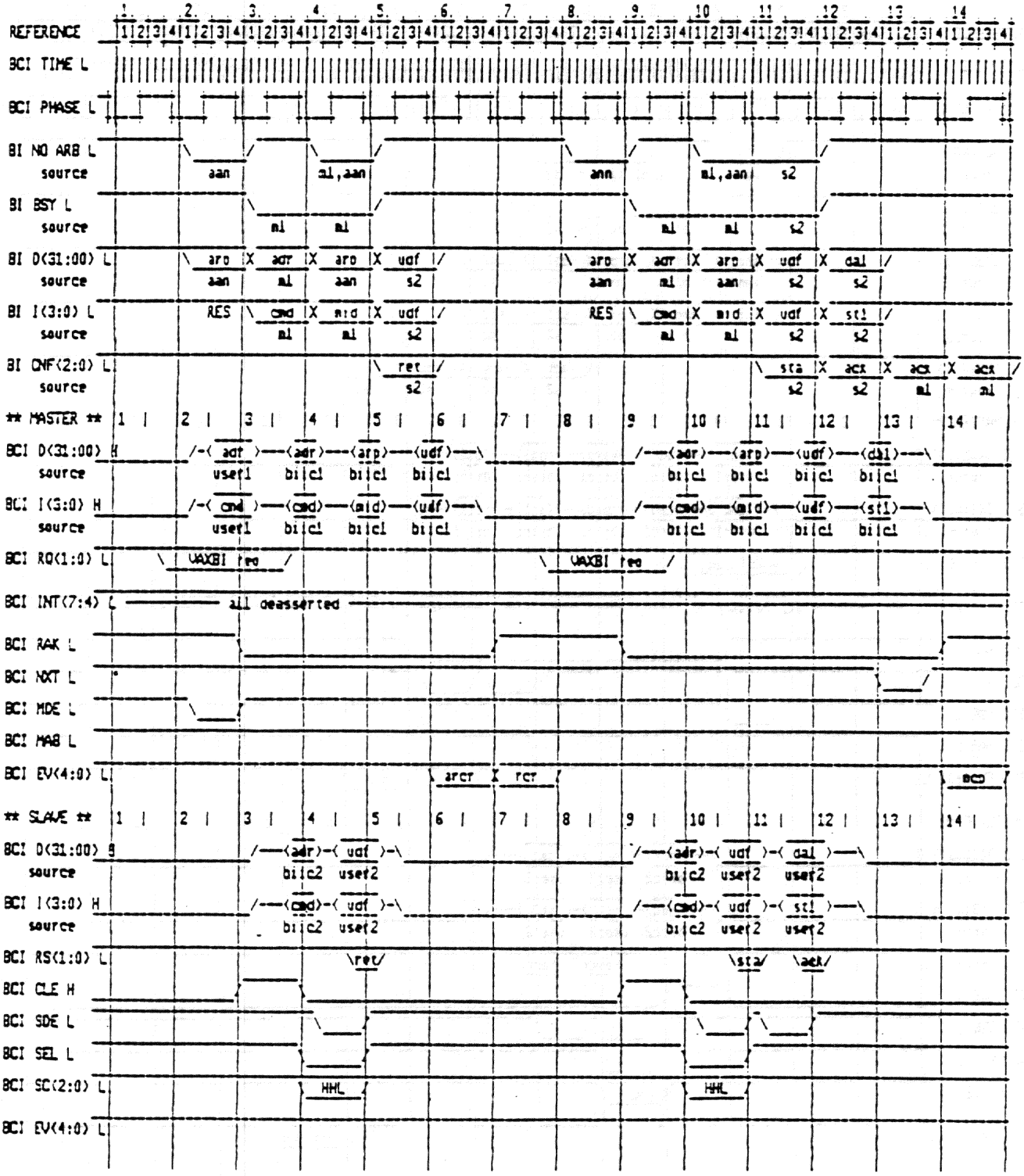
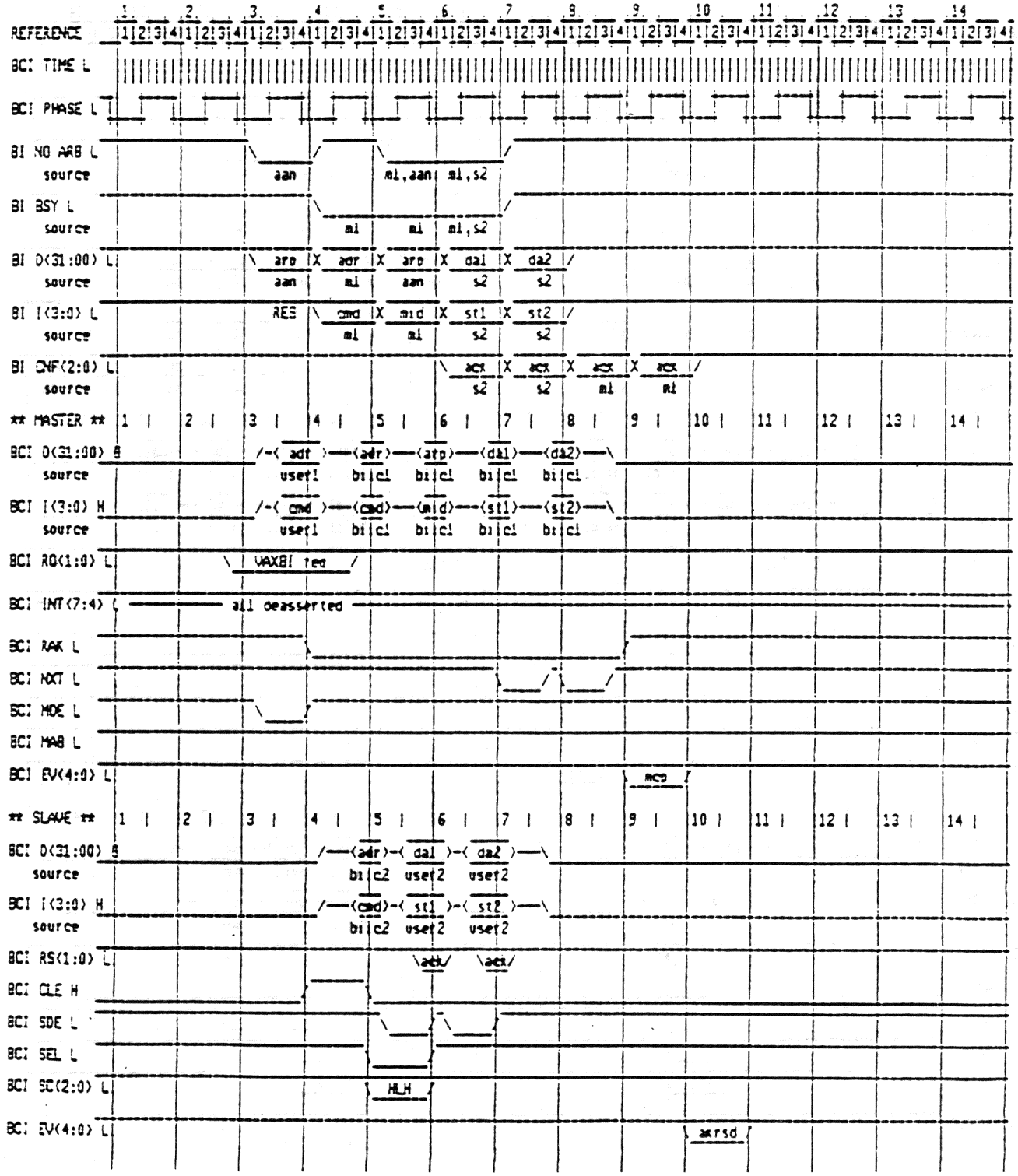


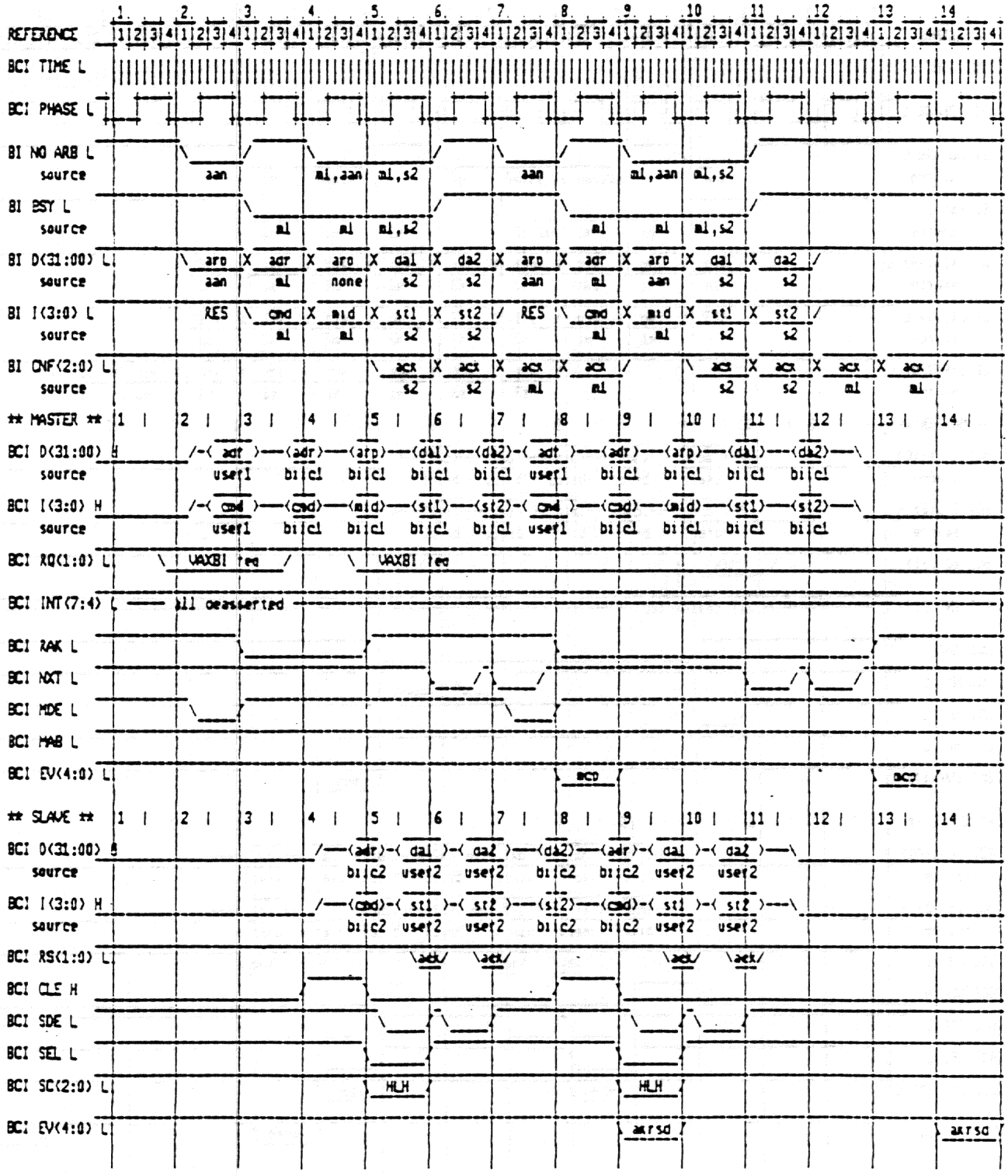
Figure 21-5: Quadword Read-Type



5.70

digital™

Figure 21-6: Quadword Read-Type with Pipeline Request



6.TD

Figure 21-7: Quadword Read-Type with Pending Master

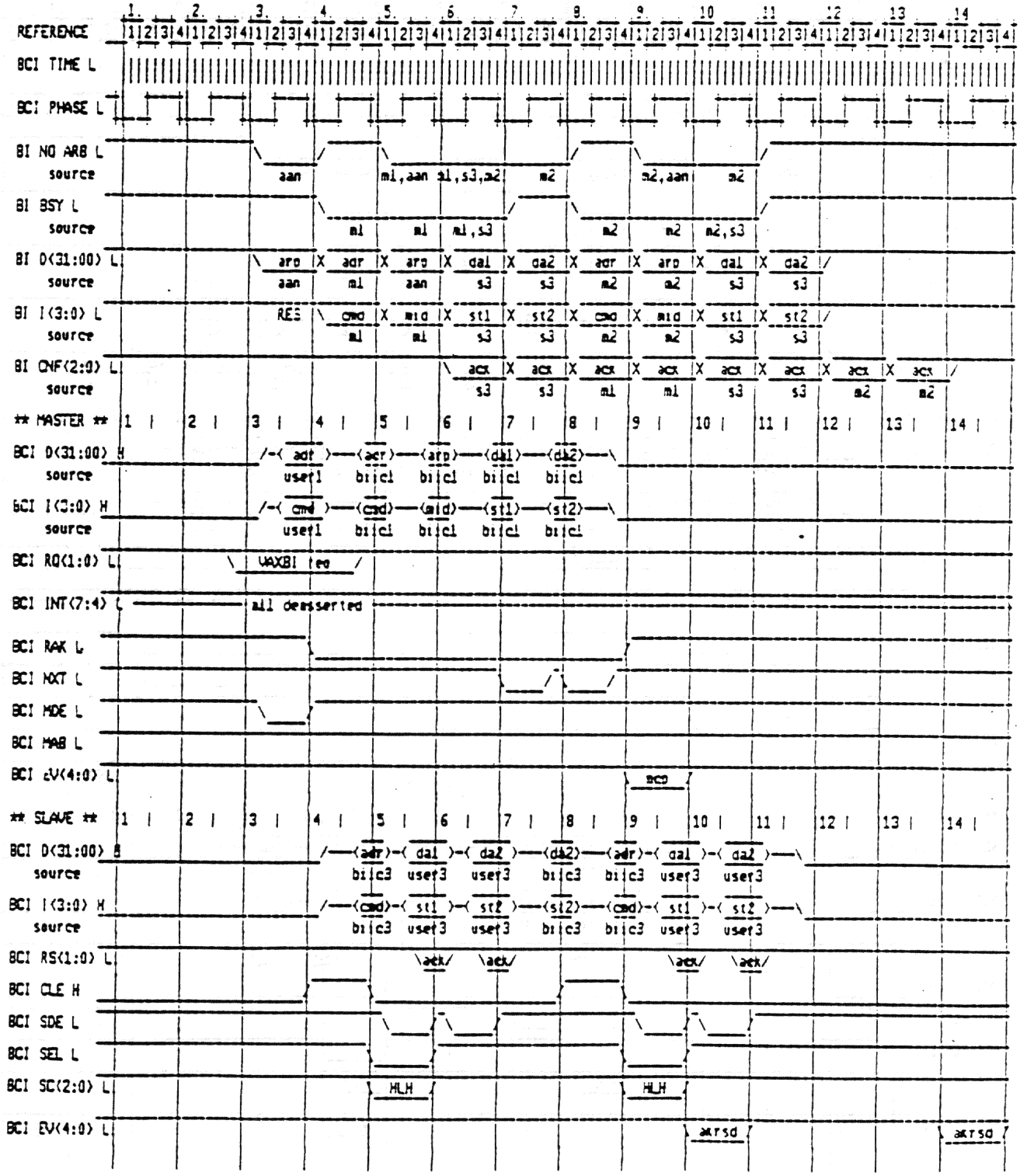
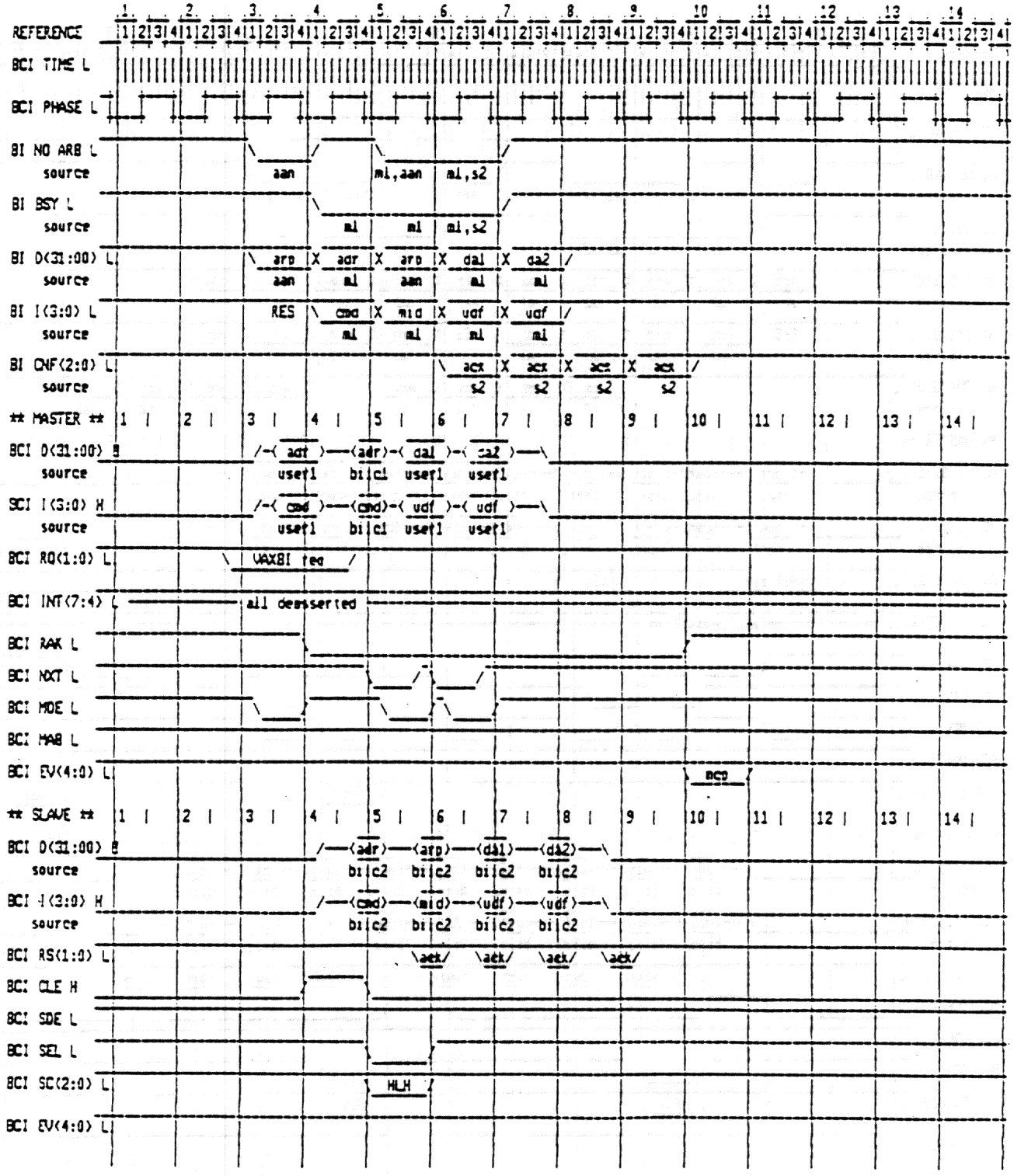
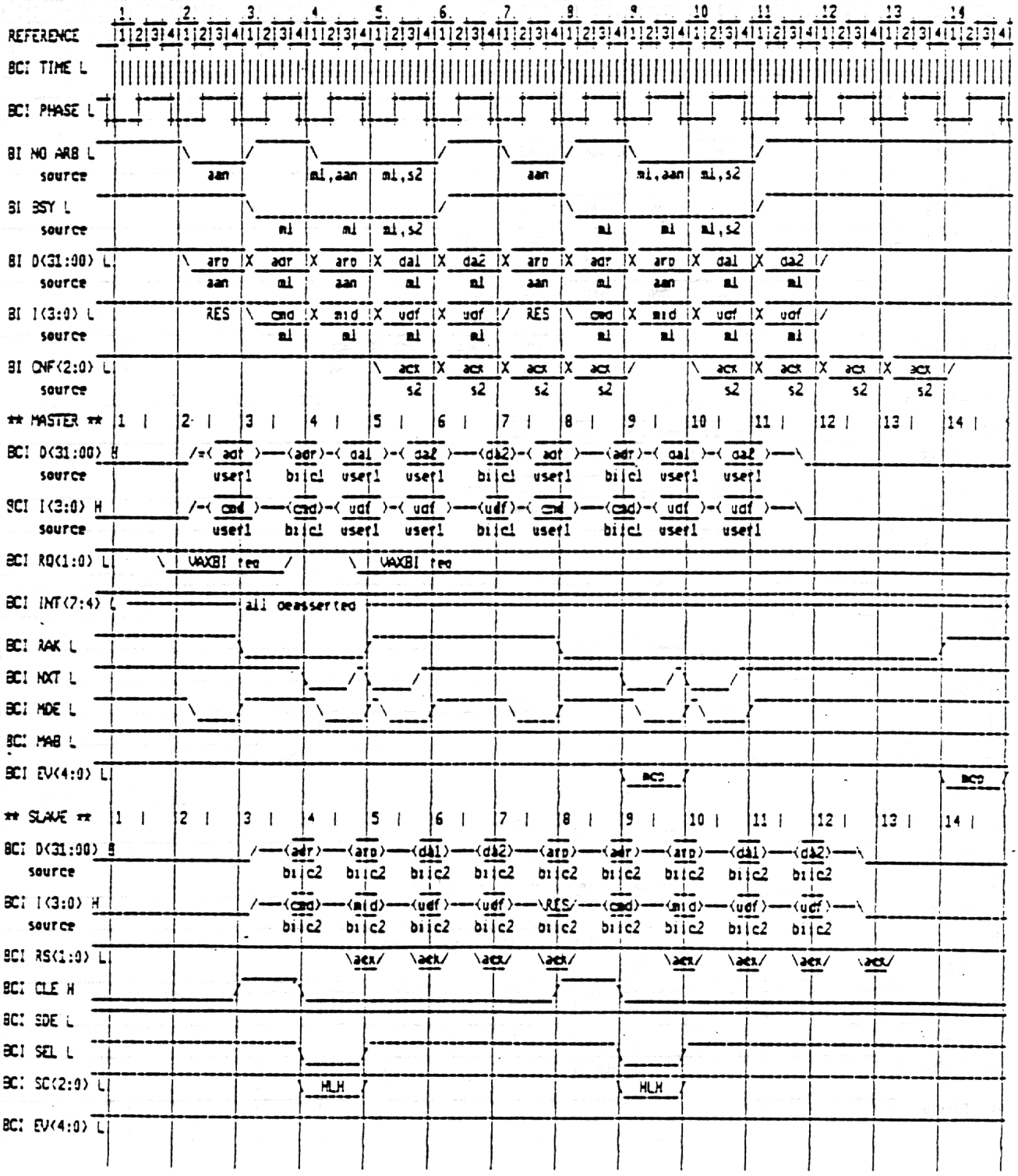


Figure 21-8: Quadword WRITE



8.70

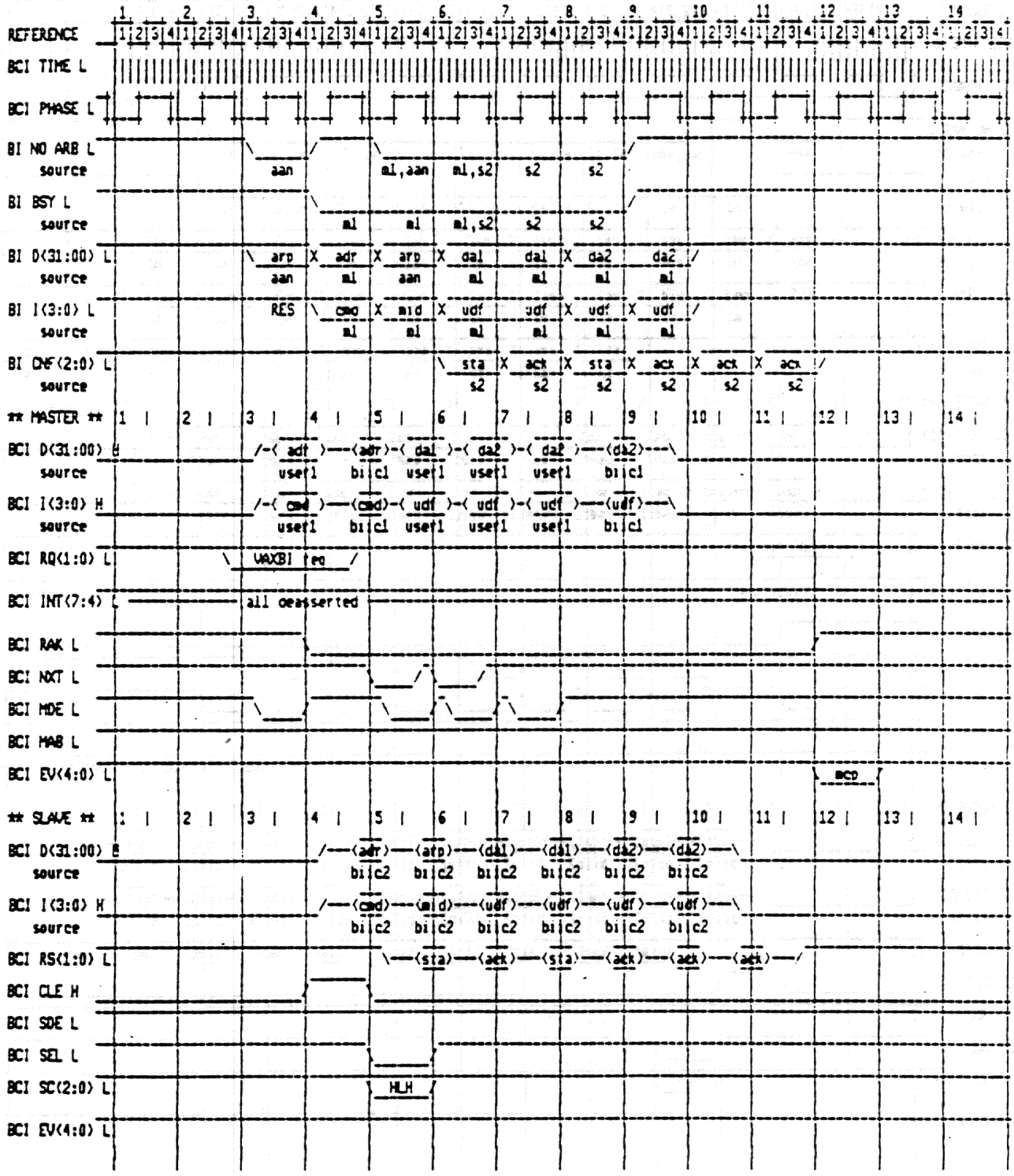
Figure 21-9: Quadword WRITE with Pipeline Request



9.7D

digital™

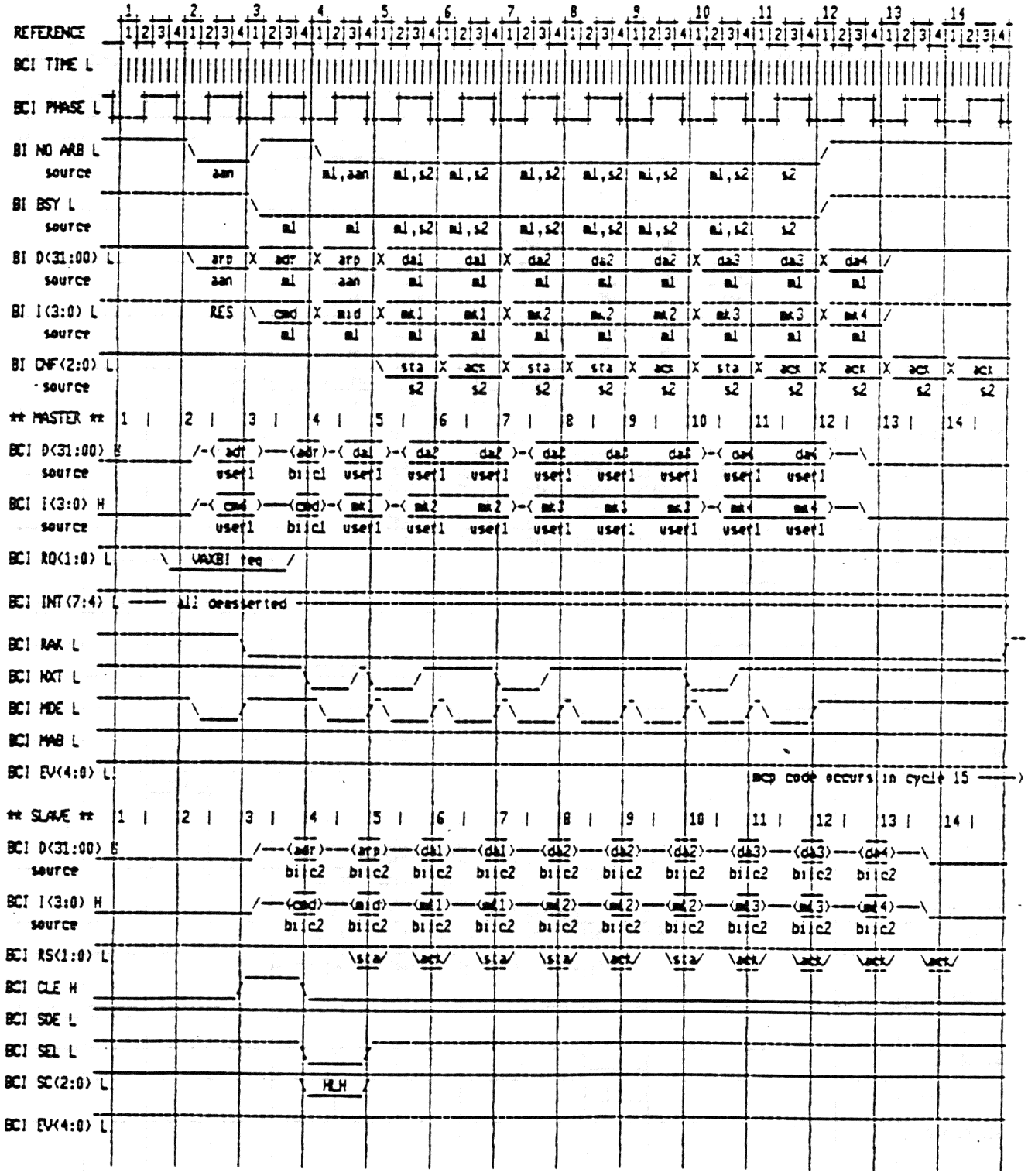
Figure 21-10: Quadword WRITE with a STALL for Each Longword of Data



10.TD

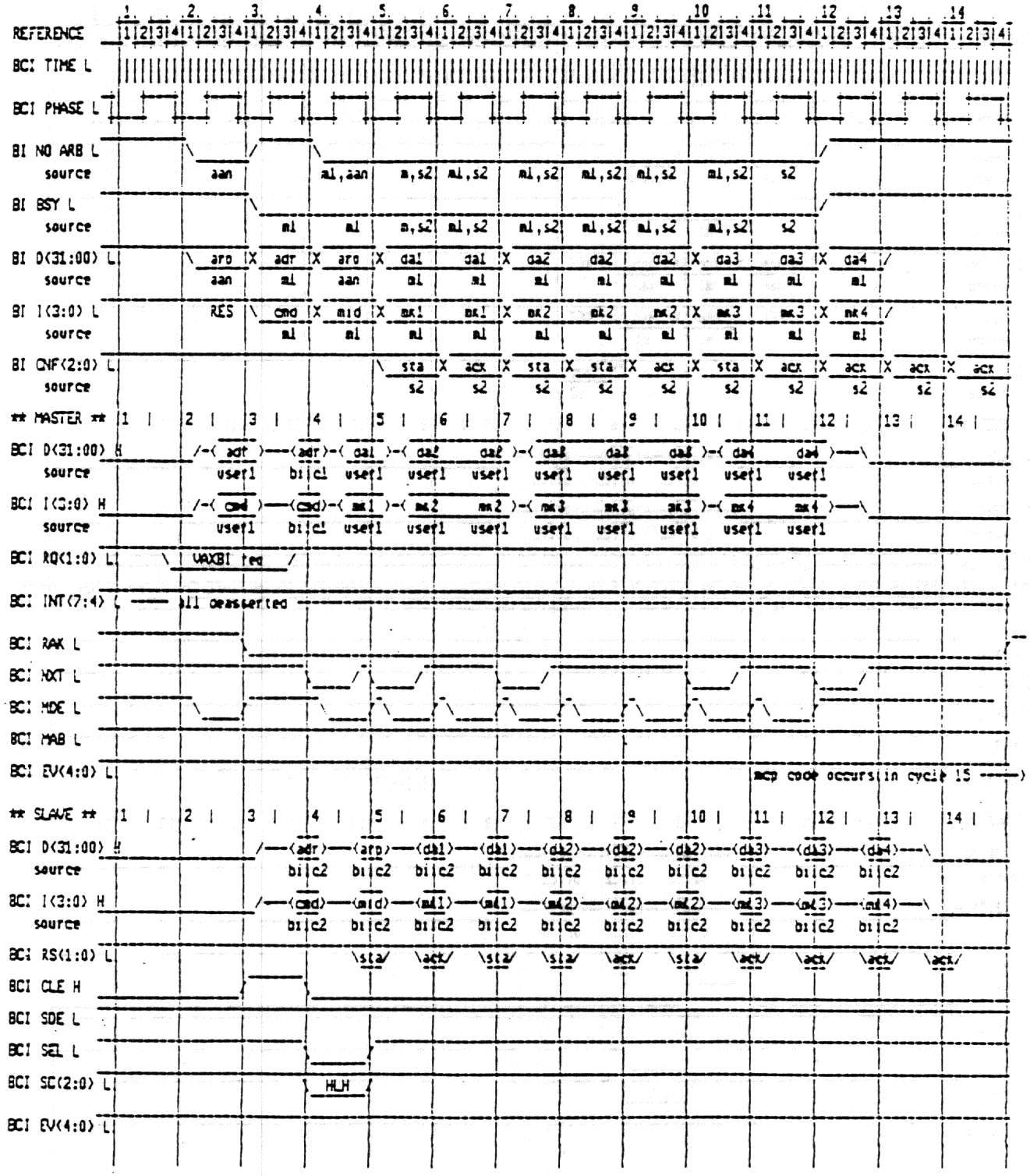
digital™

Figure 21-11: Octaword WMCI with Variable STALLs for Each Longword of Data



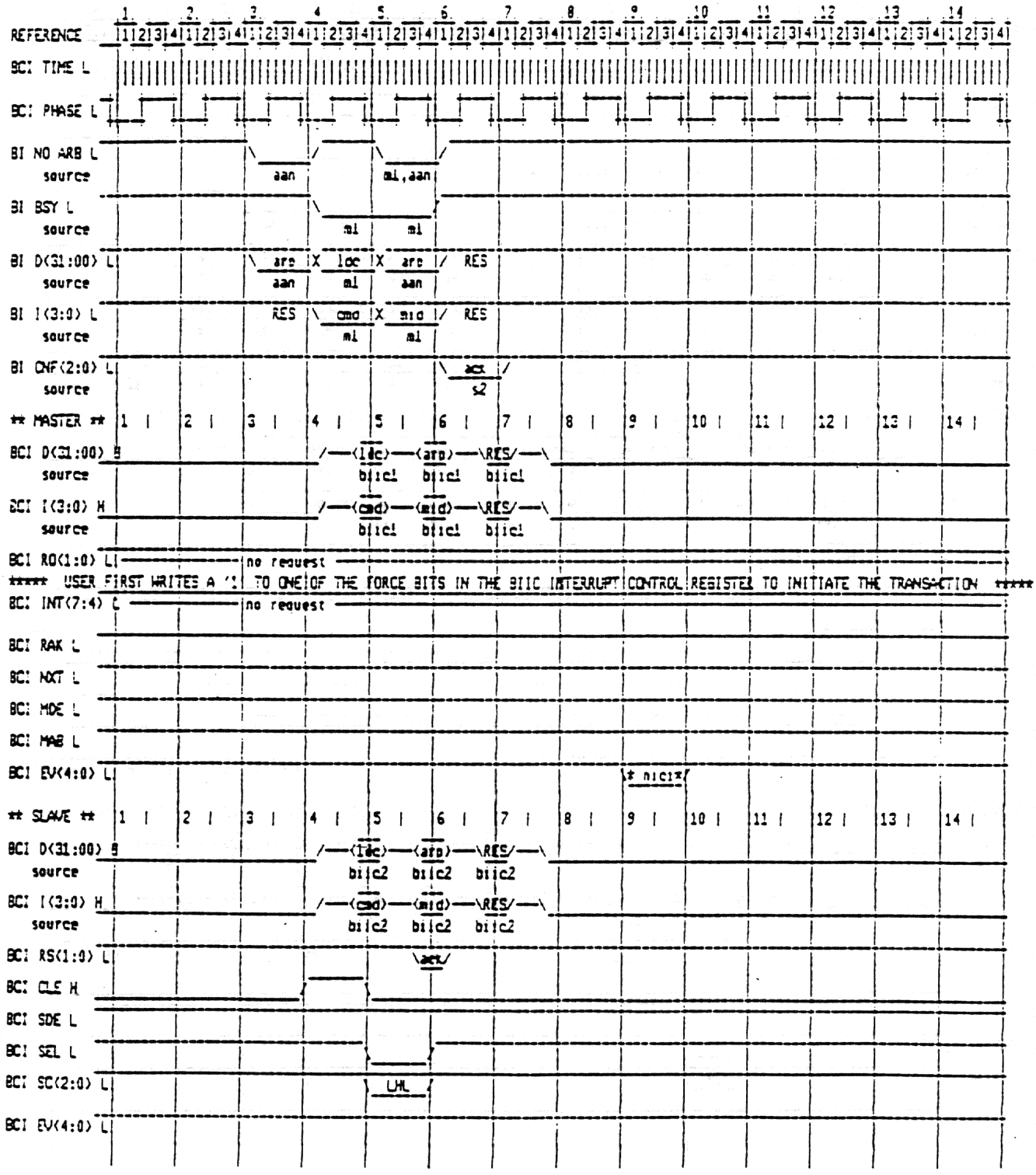
11.TD

Figure 21-12: Octaword WMCI with Variable STALLs for Each Longword of Data and with Pipeline NXT Enable Bit Set



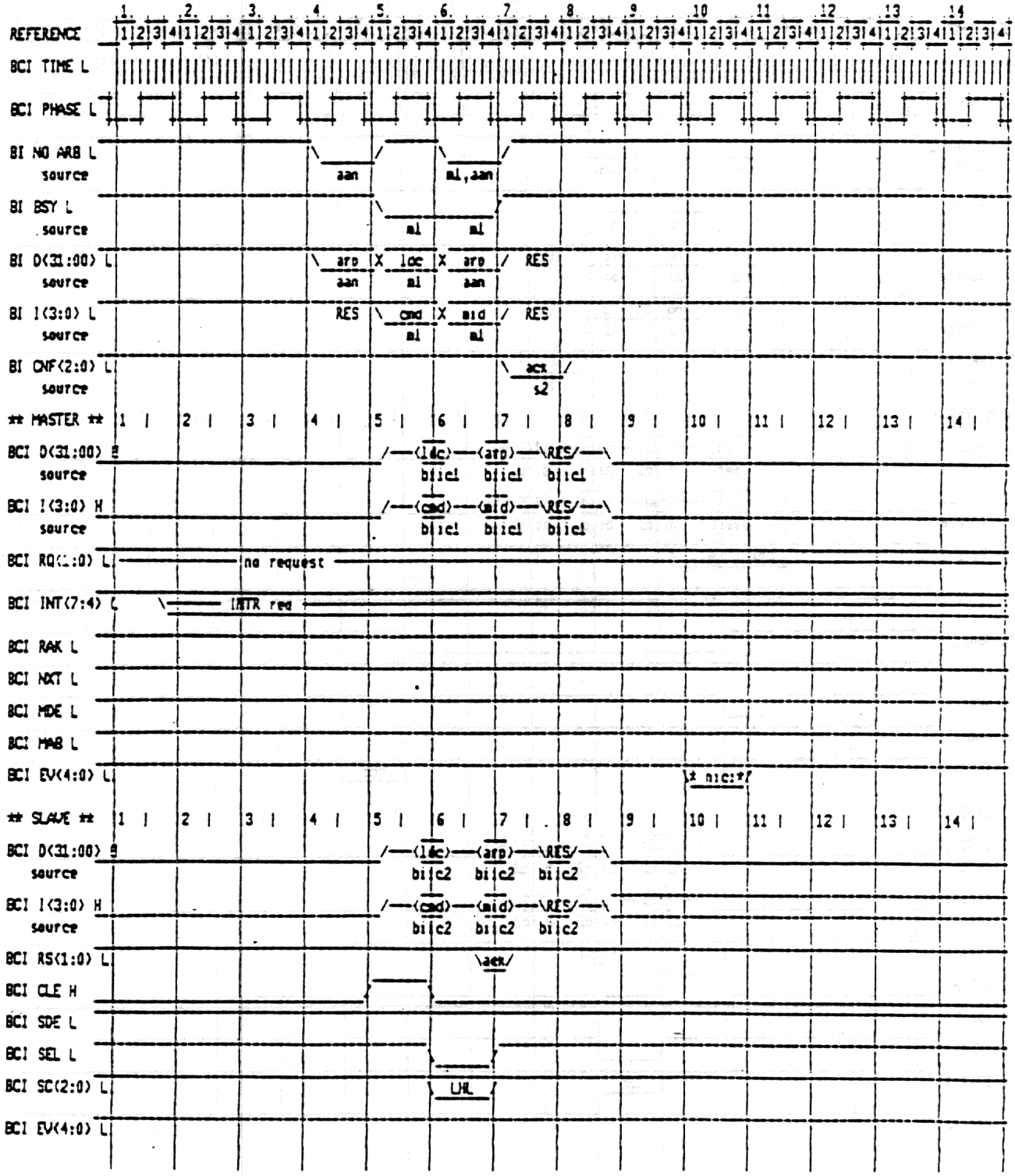
12.TD

Figure 21-13: Force-Bit Requested Interrupt



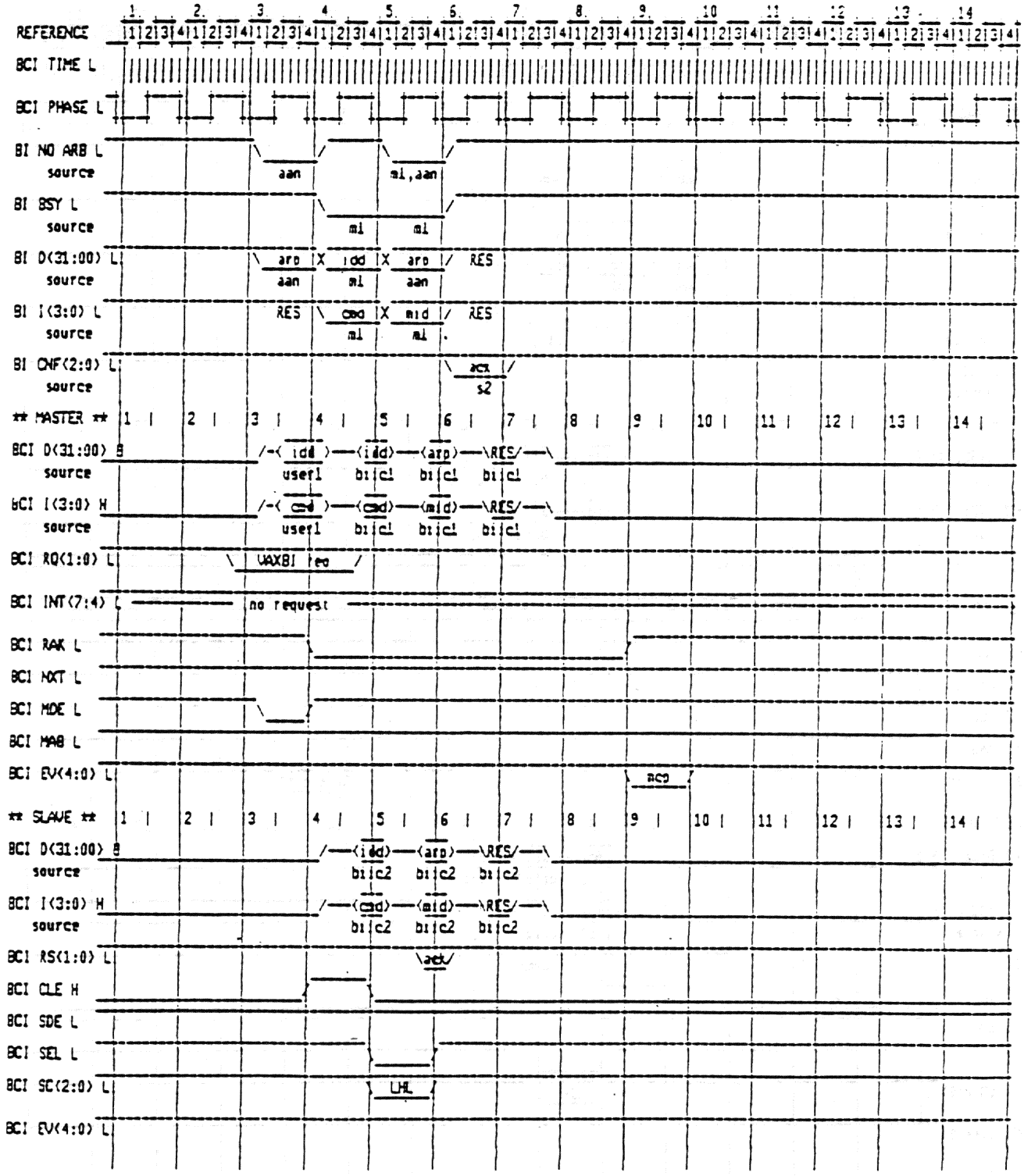
13.TD

Figure 21-14: INT<7:4> Requested Interrupt



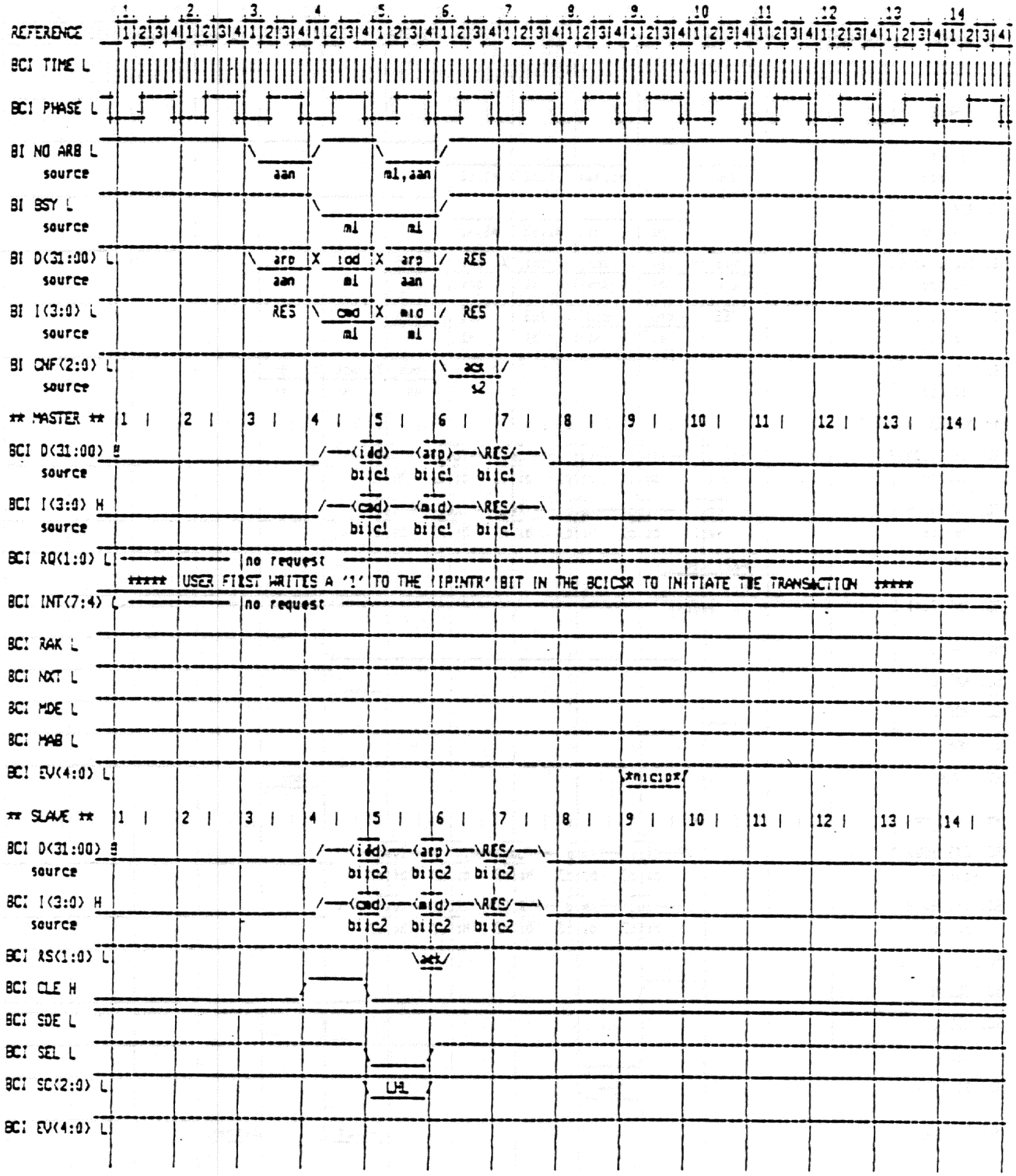
14.TD

Figure 21-15: Master Port Interprocessor Interrupt



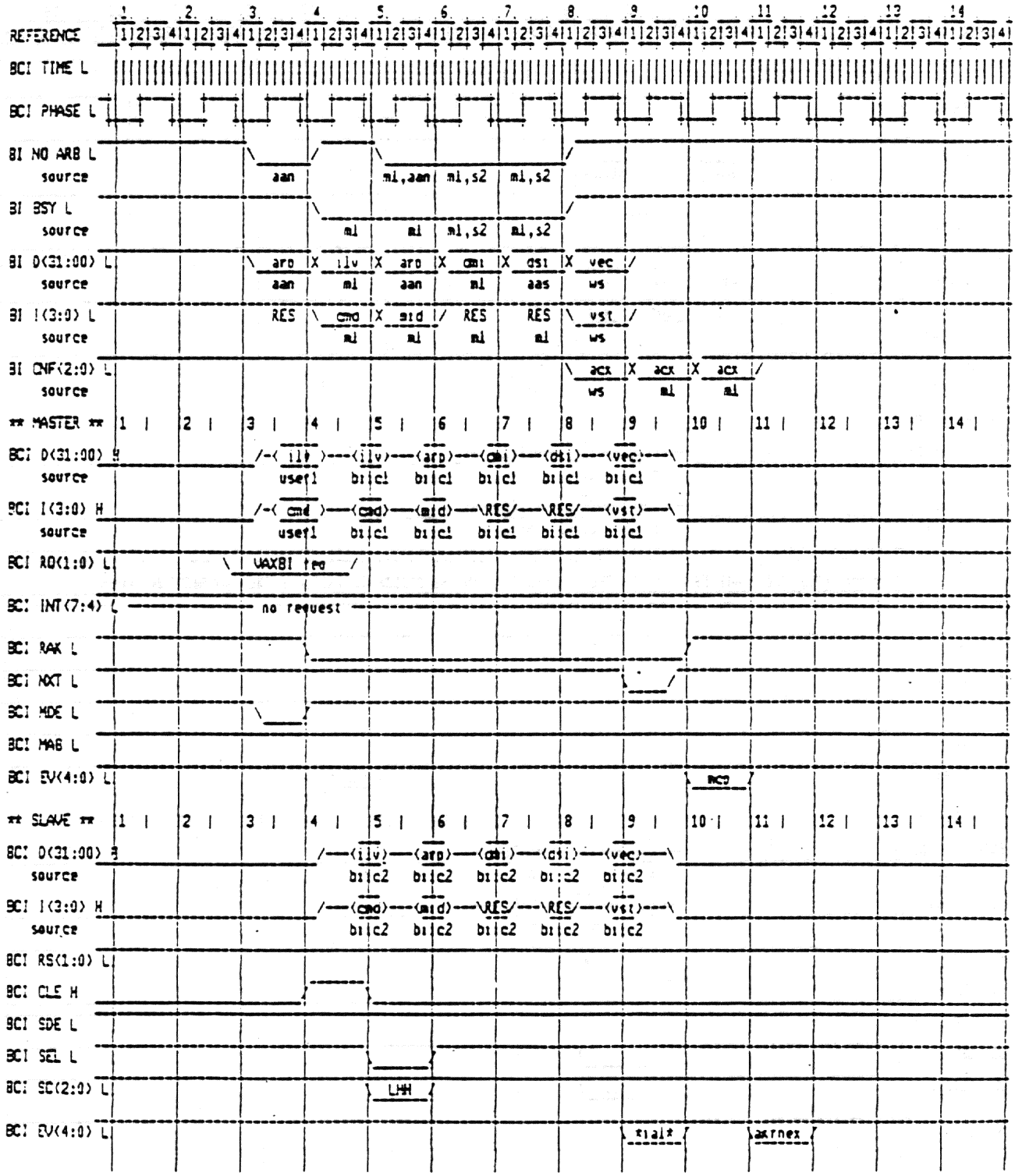
15.TD

Figure 21-16: Force-Bit Requested Interprocessor Interrupt



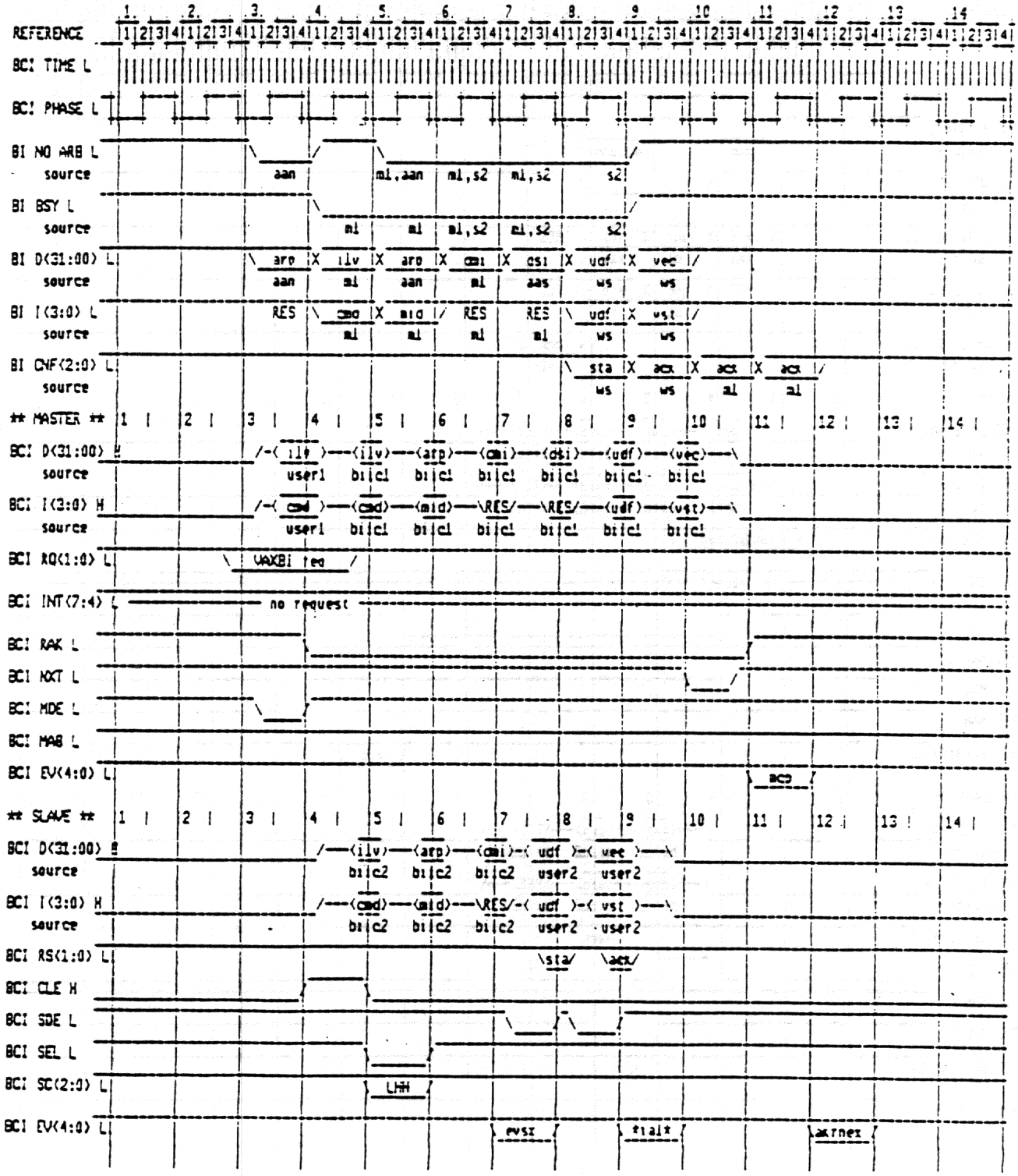
16.TD

Figure 21-17: IDENT with Internal Vector



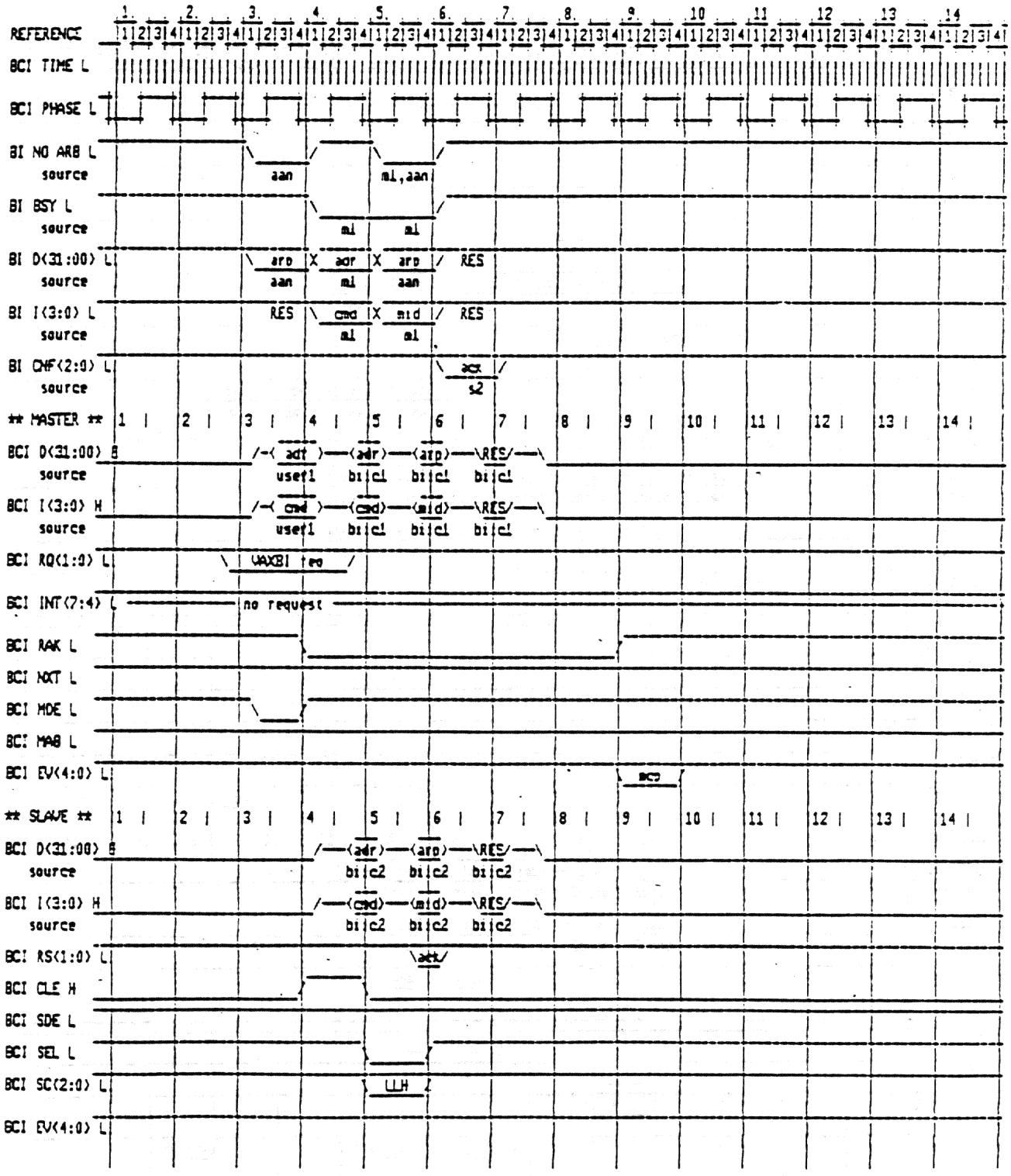
17.TD

Figure 21-18: IDENT with External Vector



18.TD

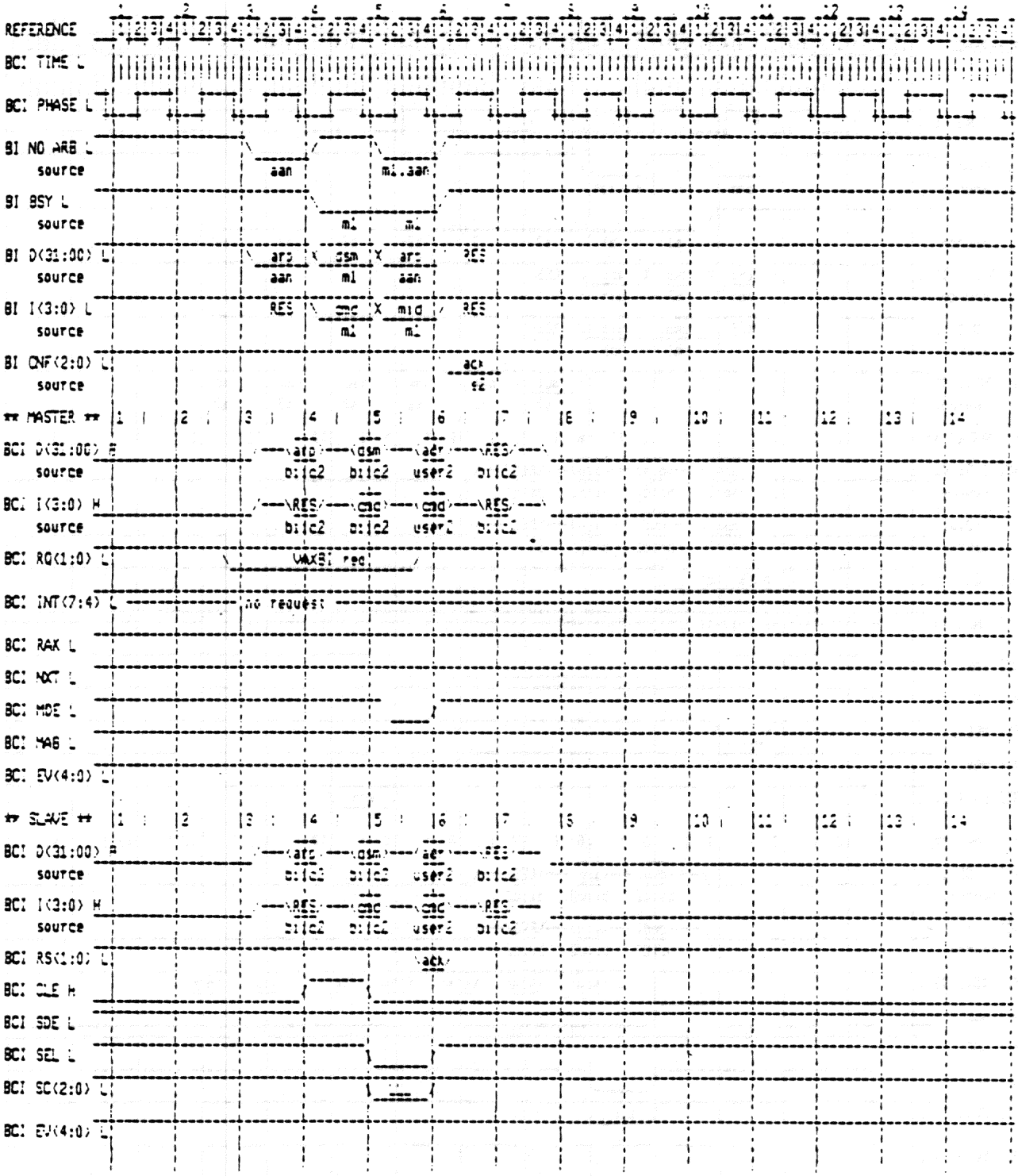
Figure 21-19: INVAL



19.TD

digital™

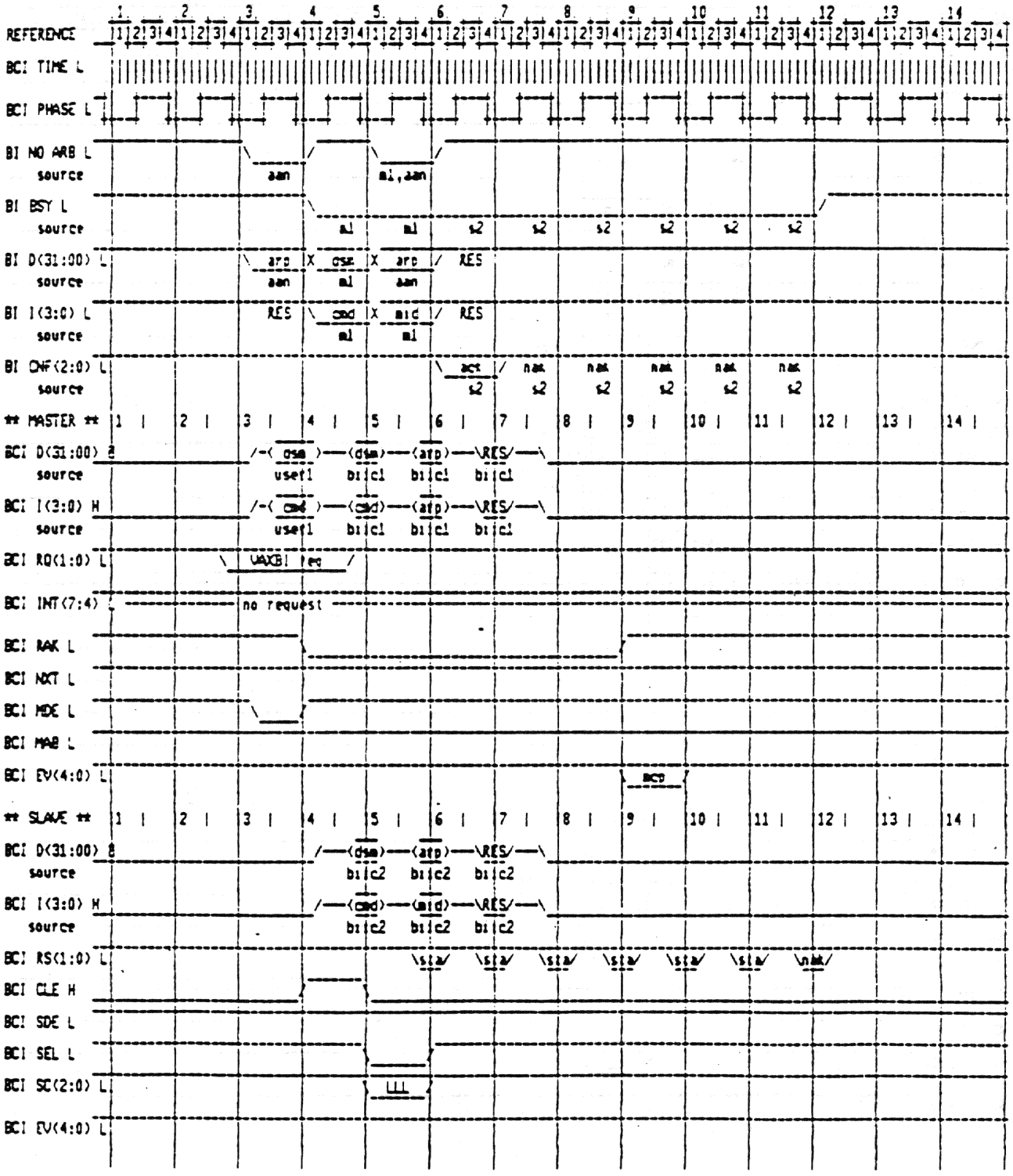
Figure 21-20: STOP



20.TD

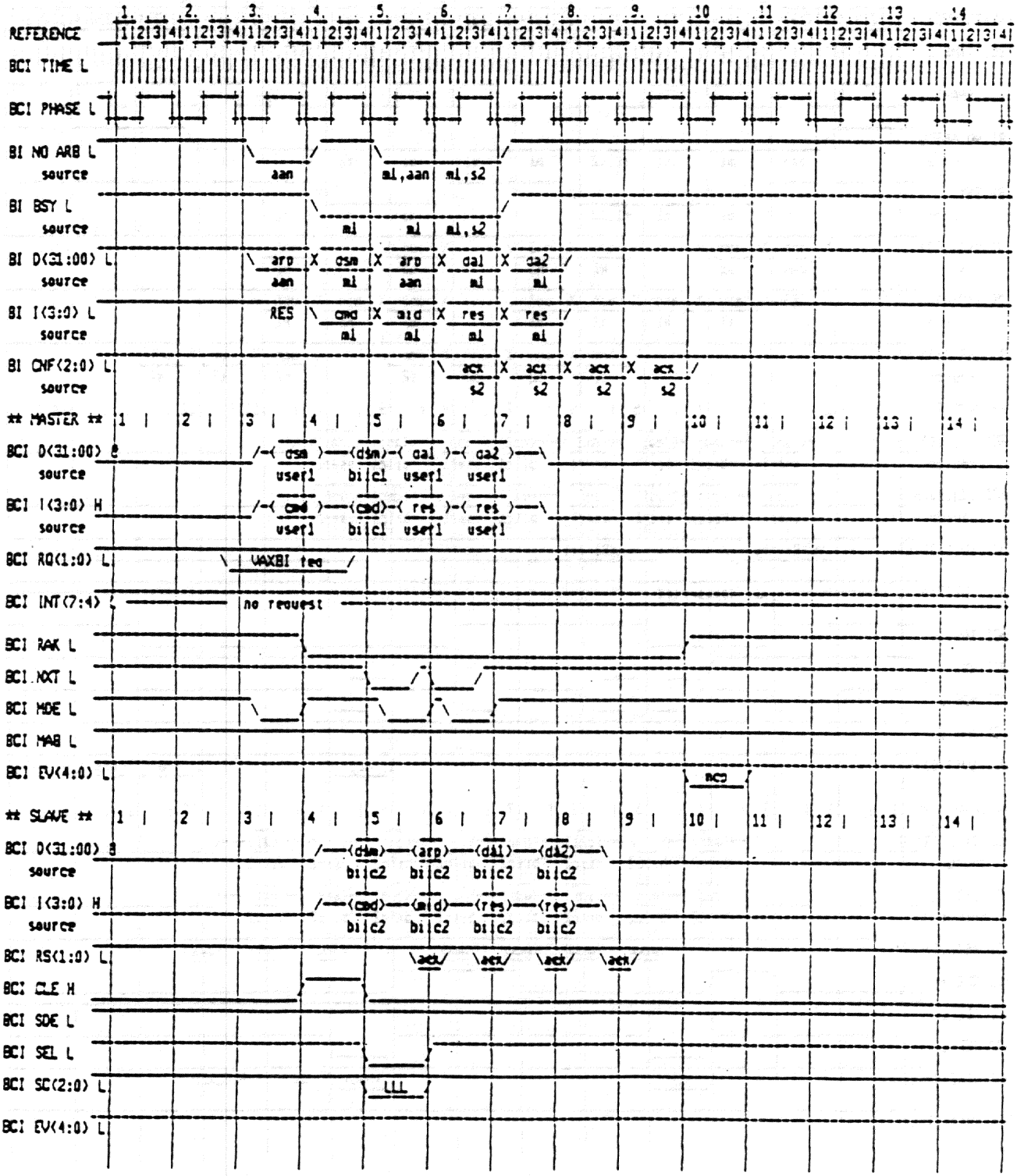
digital™

Figure 21-21: STOP with Extension



21.TD

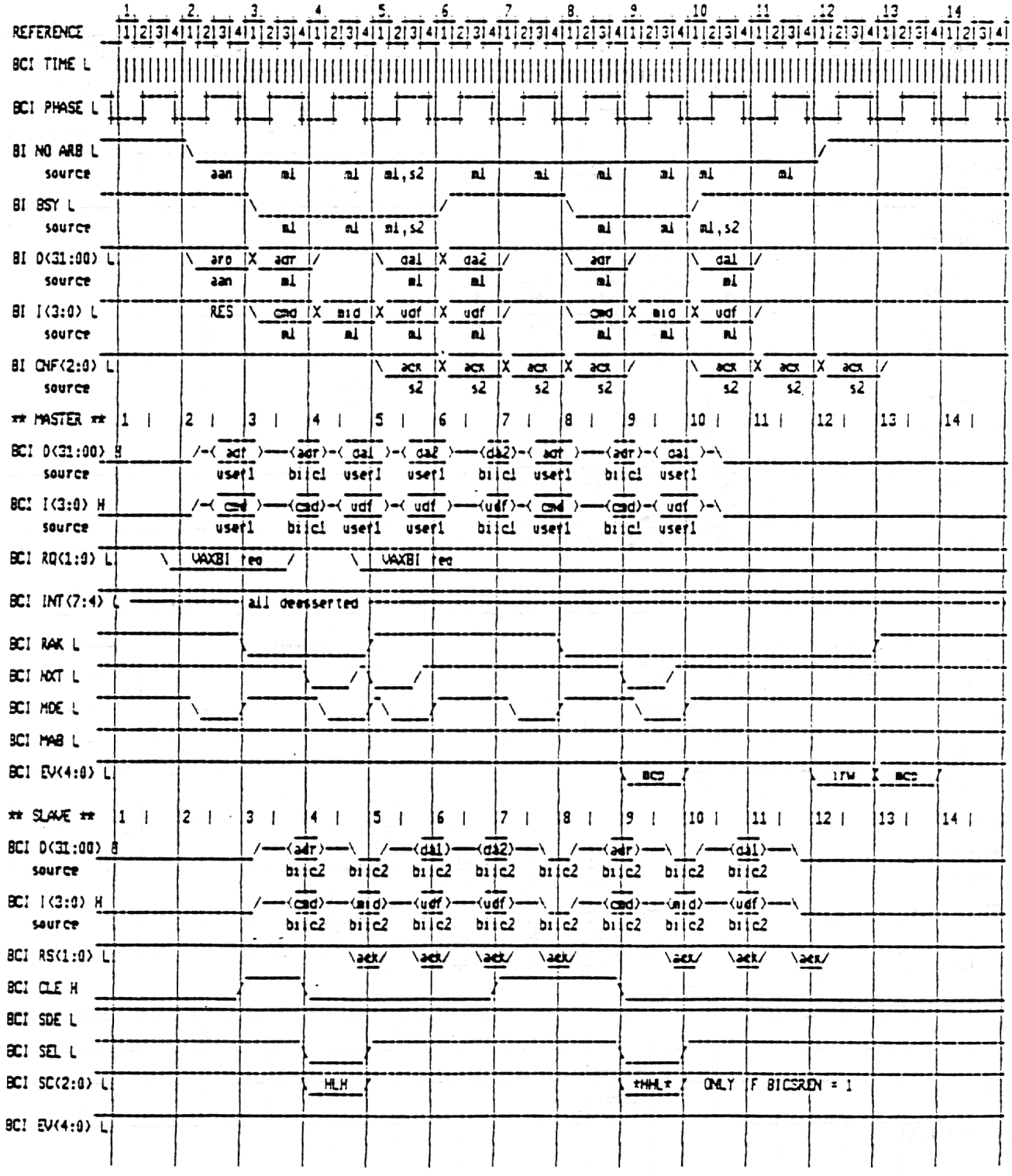
Figure 21-22: Quadword BDCST



22.TD

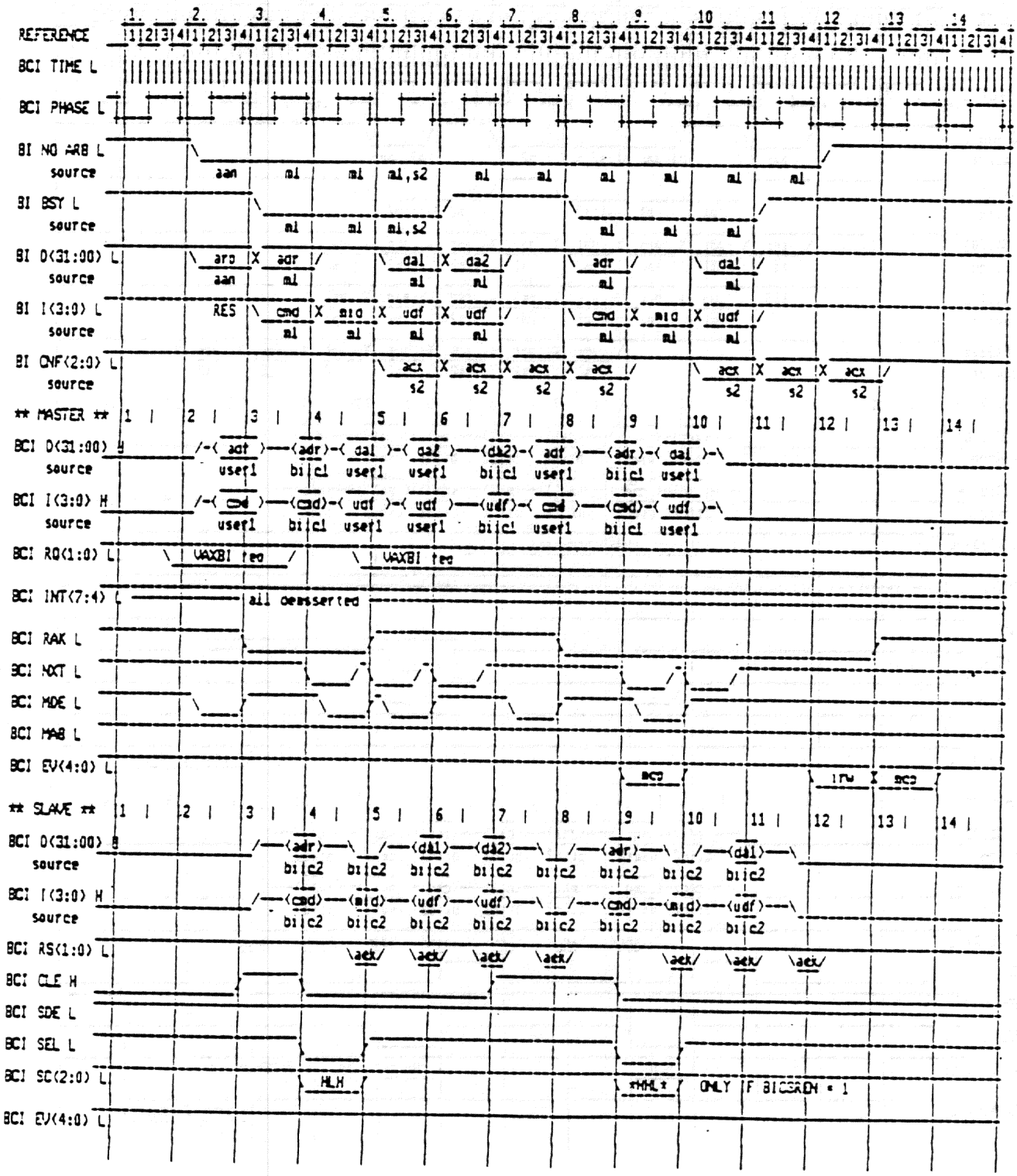
digital™

Figure 21-23: Burst Mode WRITEs with Pipeline Request



23.TD

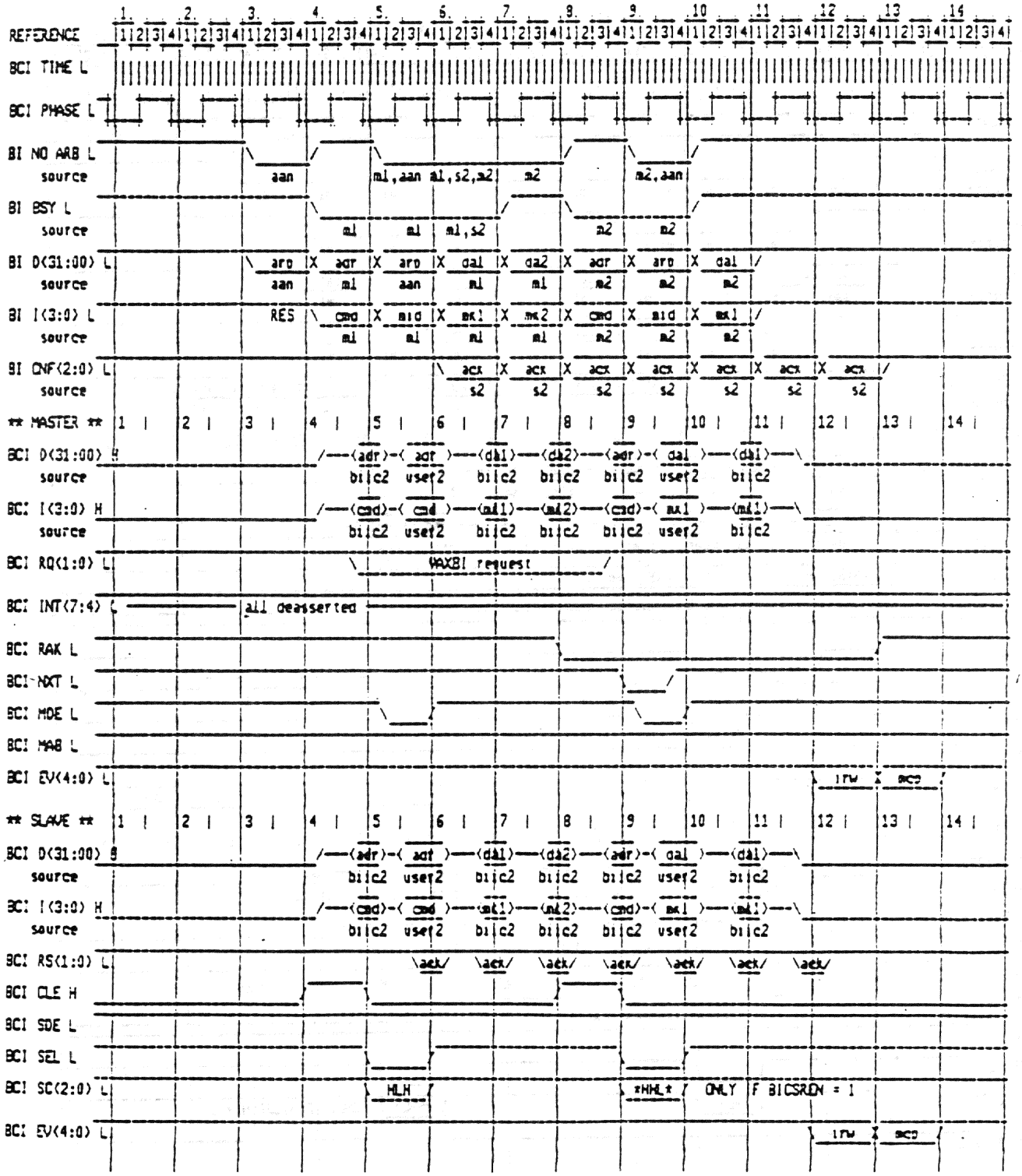
Figure 21-24: Burst Mode WRITEs with Pipeline Request and with Pipeline NXT Enable Bit Set



24.TD

digital™

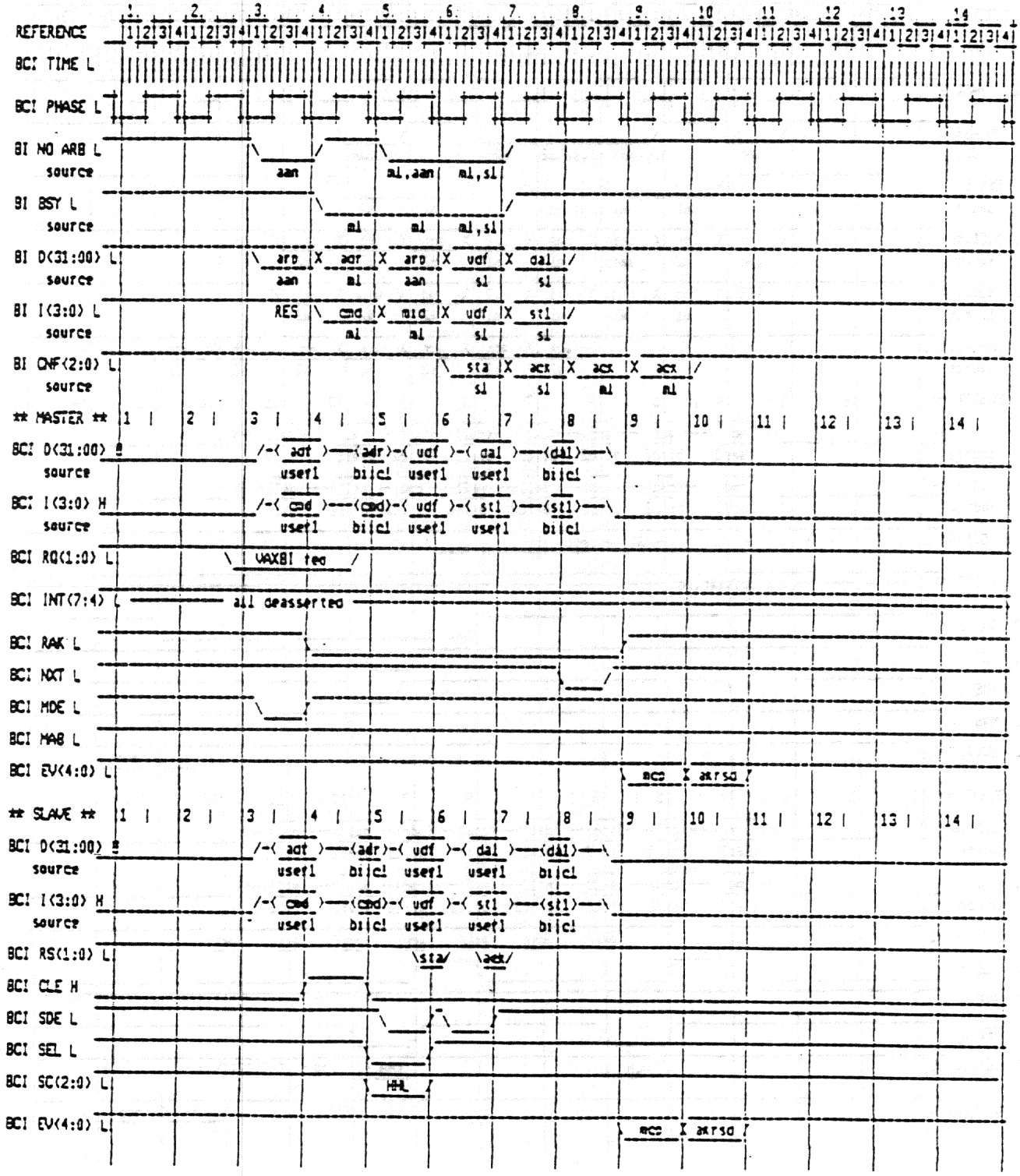
Figure 21-25: Special Case 1



25.TD

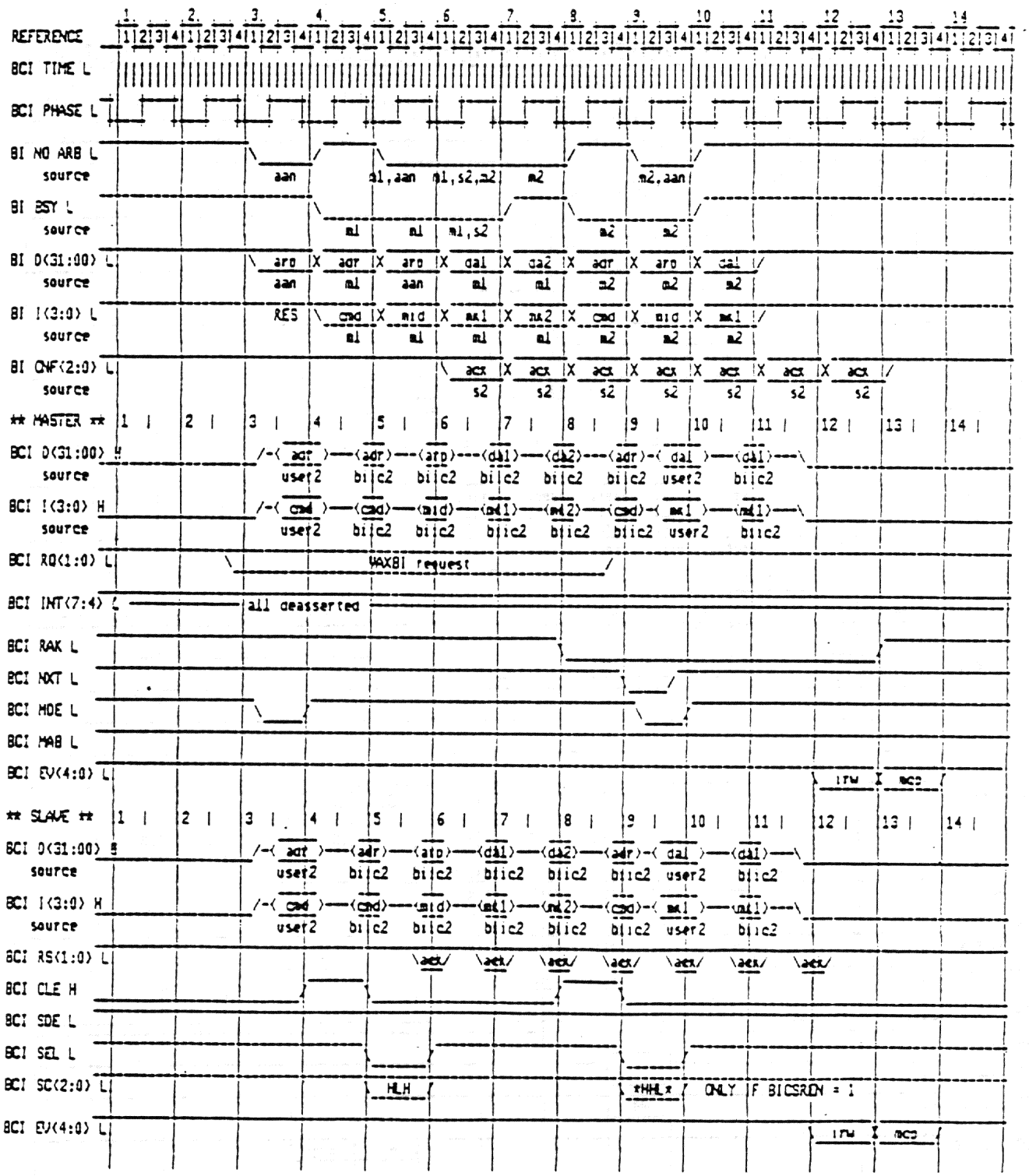
digital™

Figure 21-26: Special Case 2



26.TD

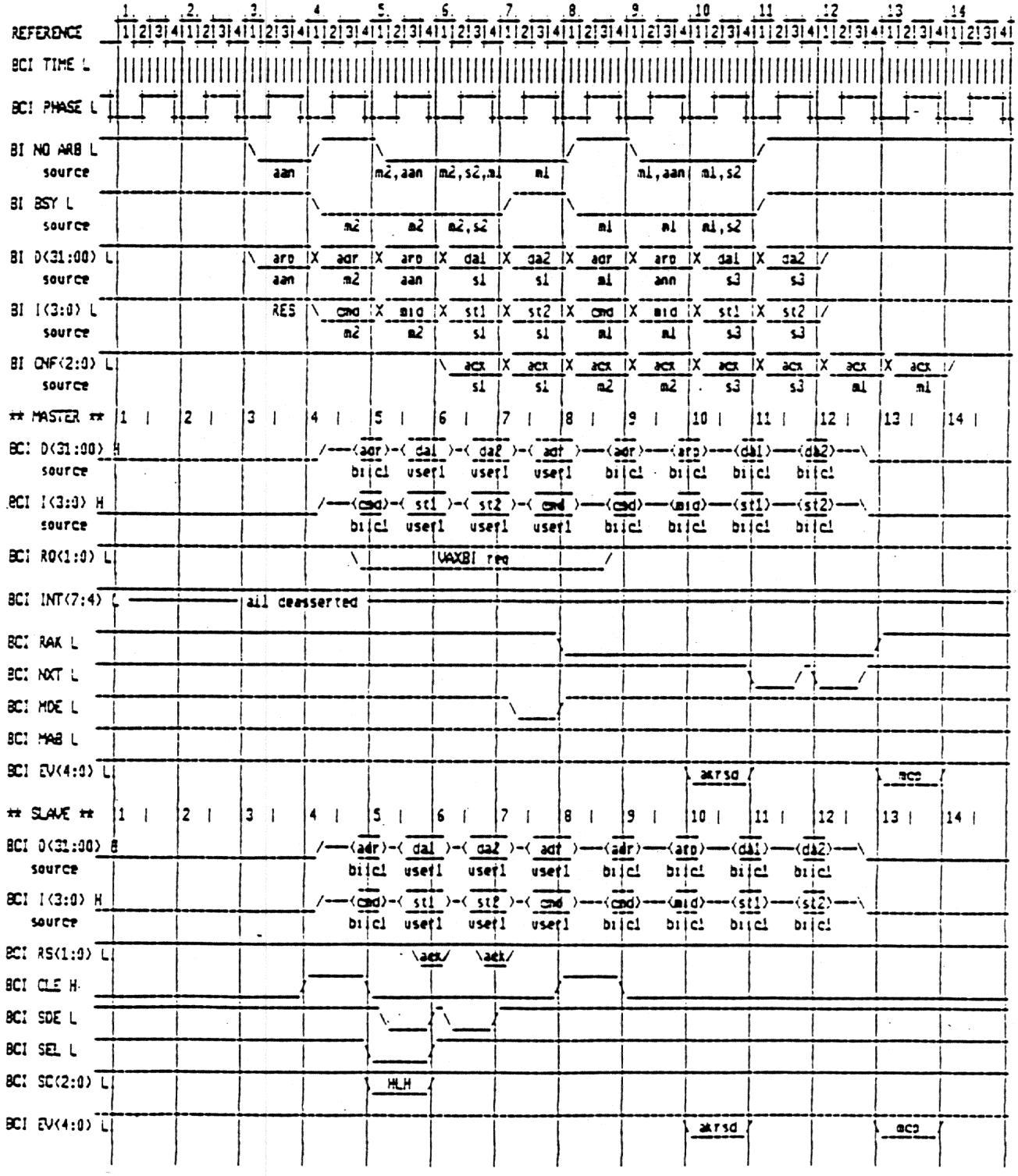
Figure 21-27: Special Case 3



27.TD

digital™

Figure 21-28: Special Case 4



28.TD

digital™

CHAPTER 22

CLOCK DRIVER SPECIFICATION

Digital Part. No. 78701

The VAXBI clock driver is a custom 14-pin DIP bipolar integrated circuit (IC) that serves as the clock source in VAXBI systems. The clock driver is designed to drive 16 clock receivers distributed over a maximum 5 feet of etch. The device requires only a +5V operating voltage.

The circuit provides differential ECL drive outputs to both BI TIME +/- and BI PHASE +/- from an externally applied 40 MHz crystal oscillator reference as well as TTL outputs that may be useful as reference clocks in certain designs.

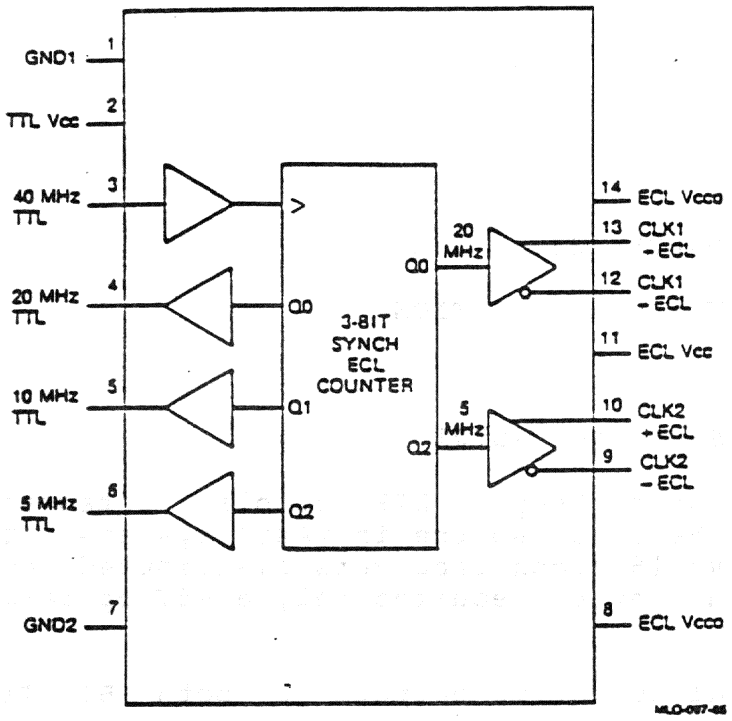
The VAXBI clock receiver IC must be used with each VAXBI clock driver source to provide the proper VAXBI clock receive function if the driving source of the VAXBI clock is also a VAXBI node. The clock driver must be at the electrical beginning of bus line etch of the clock lines.

An output enable included with the device must be used if more than one node can be installed into the drive slot of the VAXBI bus. It is important that there be only one source of clocks in VAXBI systems, and it is assumed that the first slot will be wired to enable this device.

The ECL output load resistor values used to test this device are not the VAXBI clock termination values.

This component is approved for use only in the VAXBI Corner application. See the VAXBI Module Control Drawings for placement and layout information.

digital™



NOTES:

1. Not commoned internally.
2. For pin 3, transition of negative going edge provides clocking input. Duty cycle of clock input is no worse than 60%/40% asymmetry at 1.4 volts.

Figure 22-1: Pin Assignments (top view)

digital™

Absolute Maximum Ratings

Pin voltages --	Input pin 3	-1.5 to +7.0 V
	Differential output voltage	+3 to +7 V
	Supply voltage	7 V
Operating junction temperature		125 degrees C
Storage temperature range		-55 to 150 degrees C
Power dissipation		Approx. 500 mW (no load)
Current applied to TTL output in low state		40 mA

Guaranteed Operating Conditions

Ambient temperature operating range	0 to 70 degrees C
Input frequency range of operation	DC to 50 MHz

Package Type

14-pin ceramic CERDIP package

digital™

Table 22-1 defines the logical operation of the clock driver.

Table 22-1: Clock Driver Truth Table

Input				Outputs			
TTL 40 MHz	TTL 20 MHz	TTL 10 MHz	TTL 5 MHz	ECL CLK2 -	ECL CLK2 +	ECL CLK1 -	ECL CLK1 +
Pin 3	4	5	6	9	10	12	13
L	L	L	L	H	L	H	L
H	L	L	L	H	L	H	L
L	H	L	L	H	L	L	H
H	H	L	L	H	L	L	H
L	L	H	L	H	L	H	L
H	L	H	L	H	L	H	L
L	H	H	L	H	L	L	H
H	H	H	L	H	L	L	H
L	L	L	H	L	H	H	L
H	L	L	H	L	H	H	L
L	H	L	H	L	H	L	H
H	H	L	H	L	H	L	H
L	L	H	H	L	H	H	L
H	L	H	H	L	H	H	L
L	H	H	H	L	H	L	H
H	H	H	H	L	H	L	H

NOTE: Startup state is not defined.

22.1 DC CHARACTERISTICS

Unless otherwise specified, all specifications are at Ta = 0 to 70 degrees C, Vcc = 5.0 volts +/-5%, and at thermal equilibrium.

All specifications (including capacitance values) pertain to packaged parts.

digital™

Clock Driver DC Characteristics -- TTL Signals

Symbol	Parameter	Min.	Max.	Unit	Pin	Figure/Note	Remarks
Vih	Input voltage high	2	-	V	3	Figure 22-1	Vcc = 5.25 V
Vil	Input voltage low	-	0.8	V	3	Figure 22-1	Vcc = 5.25 V
Iih	Input current high	-	20	uA	3	Figure 22-1	Vih = 2.7 V, Vcc = 5.25 V
Iil	Input current low	-	-2	mA	3	Figure 22-1	Vil = 0.5 V, Vcc = 5.25 V
Vic	Input clamp voltage	-	-1.2	V	3	Figure 22-1	Iin = -18 mA, Vcc = 5.25 V
Iinbv	Input high current breakdown	-	100	uA	3	Figure 22-1	Vih = 7 V, Vcc = 5.25 V
Cin	Input capacitance	-	12	pF	3	-	Vih = 4 V
Voh	Output high voltage	2.40	-	V	4, 5, 6	Figure 22-1, Note 1	Io = Ioh max., Vcc = 4.75 V
Vol	Output low voltage	-	0.5	V	4, 5, 6	Figure 22-1, Note 1	Io = 20 mA, Vcc = 4.75 V
Ioh	Output current high	-	-1000	uA	4, 5, 6	Figure 22-1, Note 1	Vo = Voh min., Vcc = 4.75 V
Iol	Output current low	20	-	mA	4, 5, 6	Figure 22-1, Note 1	Vo = 0.5 V, Vcc = 4.75 V
Ios	Output short circuit current	-25	-150	mA	4, 5, 6	Note 2	Vcc = 5.25 V
Icc	Power supply current	-	100	mA	2	-	Vcc = 5.25 V, No load conditions

THIS PAGE DELIBERATELY LEFT BLANK

digital™

Clock Driver DC Characteristics -- ECL Signals

Symbol	Parameter	Min.	Max.	Unit	Pin	Figure/Note	Remarks
Vod	Differential output voltage	700	-	mV	9 and 10, 12 and 13	Note 3	Termination per Figure 22-4
Voh	Output high voltage	3.5	4.3	V	9, 10, 12, 13	-	Termination per Figure 22-2
Vol	Output low voltage	2.8	3.5	V	9, 10, 12, 13	-	Termination per Figure 22-2
Ioz	Output disable current	-	50	uA	9, 10, 12, 13	Figure 22-3, Note 4	Voz = 4.5 V
Vohs	Output stress high	3.5	4.3	V	9, 10, 12, 13	-	Termination per Figure 22-2 except change 110 ohms to 60 ohms
Vols	Output stress low	2.8	3.5	V	9, 10, 12, 13	-	Termination per Figure 22-2 except change 110 ohms to 60 ohms
Coz	Disabled output capacitance	-	6	pF	9, 10, 12, 13	-	-
| Coz + minus Coz - |	Difference in output capacitance	-	2	pF	9, 10, 12, 13	-	-

NOTES

1. Testing either (Voh and Vol) or (Ioh and Iol) is sufficient to satisfy these requirements.
2. TTL outputs shorted to GND, ECL outputs shorted to Vcc, or either outputs open for duration of 5 minutes should not damage the chip. Ios is defined as current into output pin when input conditions force outputs to logic one before short application.
3. This is 318 mV as measured in Figure 22-4 on scope across A and A*.
4. The Ioz test is done with pin 11 grounded (= Vee), pins 1 and 7 = GND, and TTL clock input (pin 3) high. The Ioz test is also done with pins 2, 8, and 14 grounded and at +5.0 V.

22.2 AC TIMING SPECIFICATIONS

Unless otherwise specified, all specifications are at Ta = 0 to 70 degrees C, Vcc = 5.0 volts +/-5%, and at thermal equilibrium.

All specifications (including capacitance values) pertain to packaged parts. All AC specifications apply with all outputs loaded and switching simultaneously.

digital™

Clock Driver AC Timing Specifications -- TTL Signals

Symbol	Parameter	Min.	Max.	Unit	Pin	Figure/Note	Remarks
Tod2	Propagation delay	-	10	ns	3 to 4, 5, or 6	Figure 22-5	500 ohm, 50 pF load
Tr	Output rise time	-	8	ns	4, 5, 6	-	Vol max. to Voh min., 500 ohm, 50 pF load
Tf	Output fall time	-	8	ns	4, 5, 6	-	Voh min. to Vol max., 500 ohm, 50 pF load

Clock Driver AC Timing Specifications -- ECL Signals

Symbol	Parameter	Min.	Max.	Unit	Pin	Figure/Note	Remarks
Tskw1	Single differential output skew	-	1	ns	9 to 10 or 12 to 13	Figure 22-6	-
Tskw2	Differential output to output skew	-	1	ns	9 to 12 or 10 to 13	Figure 22-7	-
Tpd1	Propagation delay	-	12	ns	3 to 9, 10, 12, or 13	Figure 22-5	From Vin = 1.5 V to Vout = 50% point
Tr	Output rise time (30 - 70%)	0.5	2.5	ns	9, 10, 12, 13	Figure 22-4	-
Tf	Output fall time (30 - 70%)	0.5	2.5	ns	9, 10, 12, 13	Figure 22-4	-
Vnse	Vcc noise immunity	-	50	mV	9, 10, 12, 13	Figure 22-8, Note 1	-

NOTE

1. Noise immunity is such that the differential output voltage of each driver cannot change due to additive noise on Vcc. The noise voltage is specified over the +/-5% Vcc operating range with +/-Vnse volts of additive noise.

   The Vcc noise immunity test is done with a pulse generator output of 250 mV amplitude with 1.0 ns rise and fall times. This test guarantees that Vcc noise as described in Figure 22-8 will result in no more than 50 mV of loss in differential output signal.

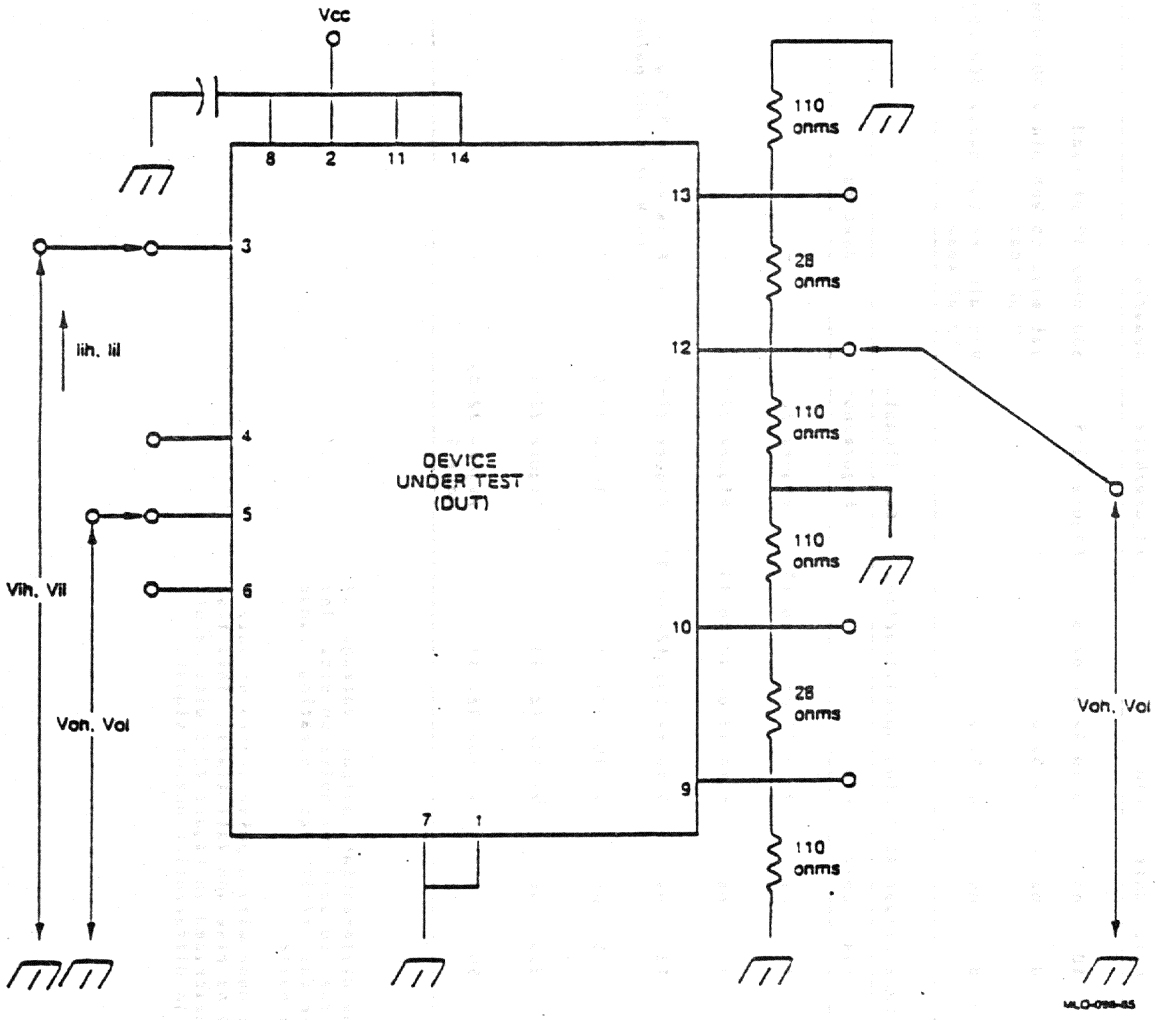


Figure 22-2: DC Tests

digital™

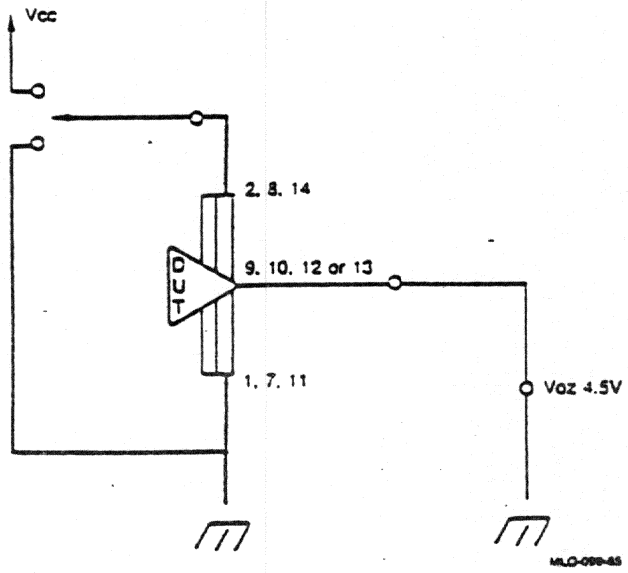


Figure 22-3: Ioz Test

digital™

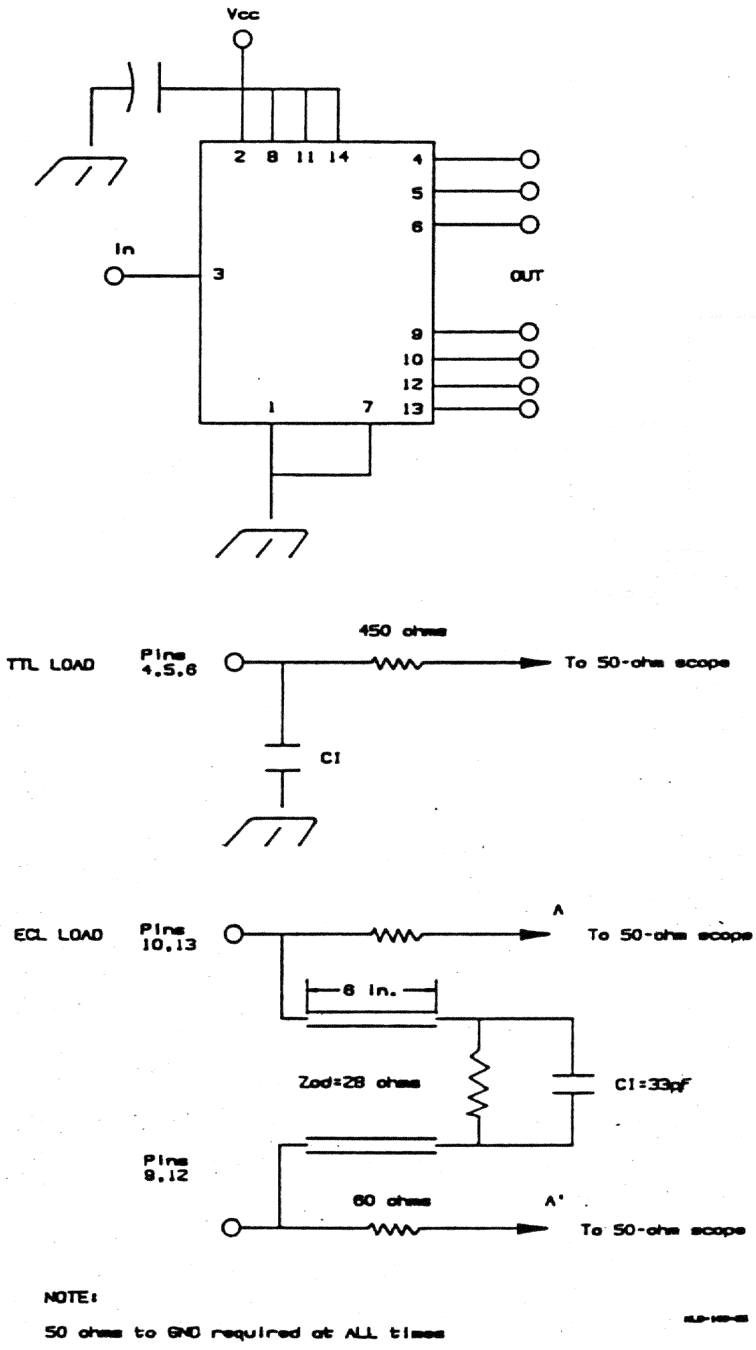
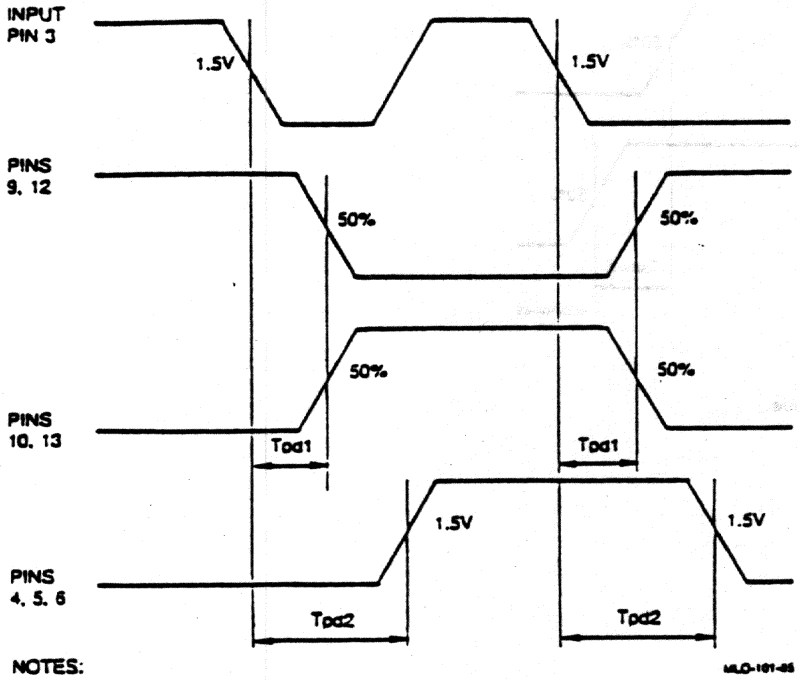


Figure 22-4: AC Test Fixture

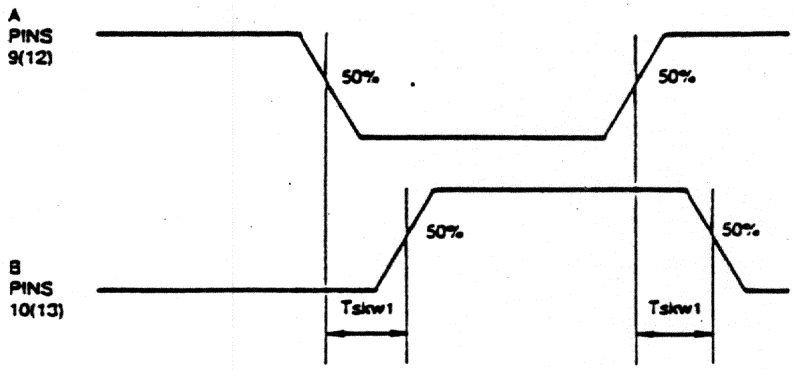
digital™



NOTES:

1. Load outputs per Figure 22-4.
2. Input rise and fall times equal 1.0 ns.

Figure 22-5: Propagation Delay Test

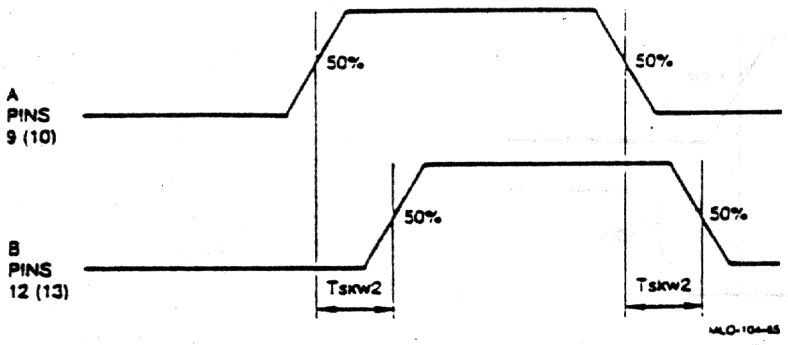


NOTES:

1. Tskw1 = greater of | A – B | or | B – A | absolute value.
2. Load outputs per Figure 22-4.

Figure 22-6: Tskw1 Test

digital™

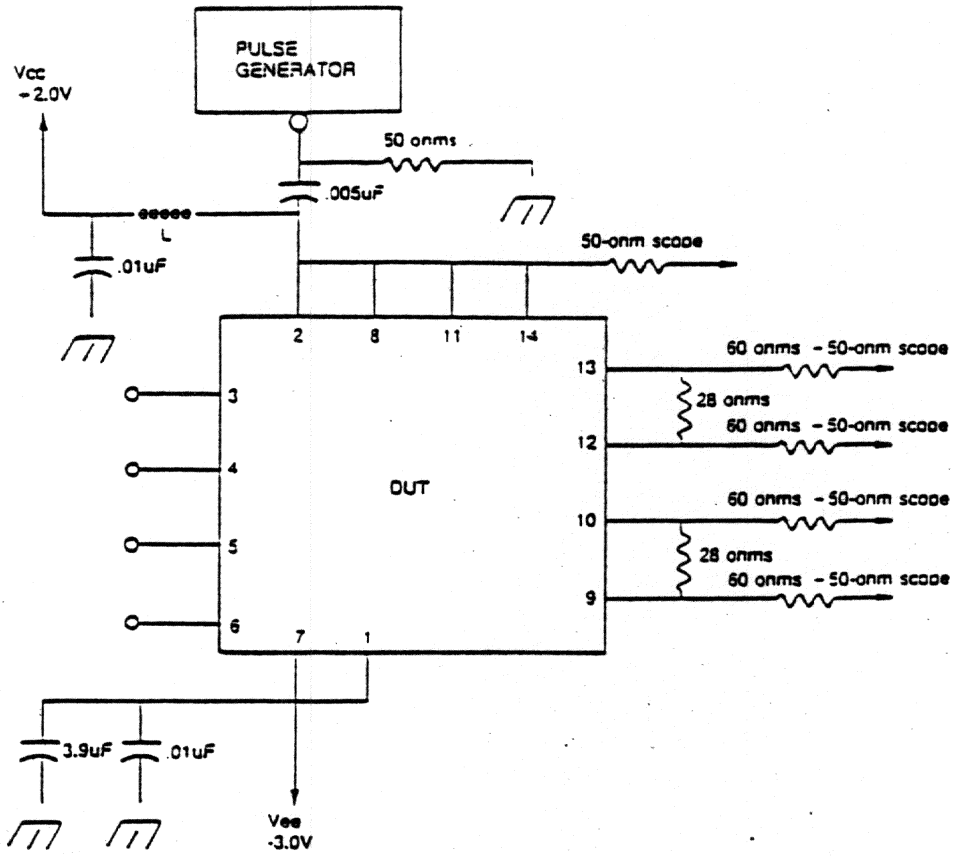


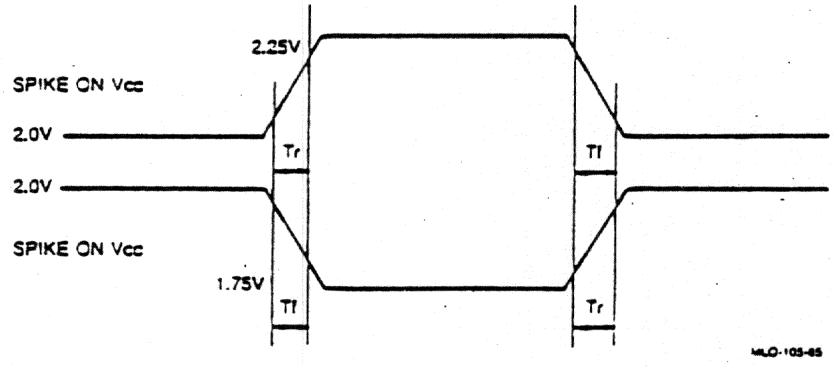
NOTES:

1. Tskw2 = greater of | A – B | or | B – A | absolute value.
2. Load outputs per Figure 22-4.

Figure 22-7: Tskw2 Test

digital™





NOTES:

1. L = 4 turns #30 wire over Ferrite bead (ferroxcube #5659065/38 or equivalent)
   REF: Motorola App. Note AN-592.
2. Input rise and fall times equal 1.0 ns.

Figure 22-8: Vcc Noise Immunity Test

digital™

CHAPTER 23

CLOCK RECEIVER SPECIFICATION

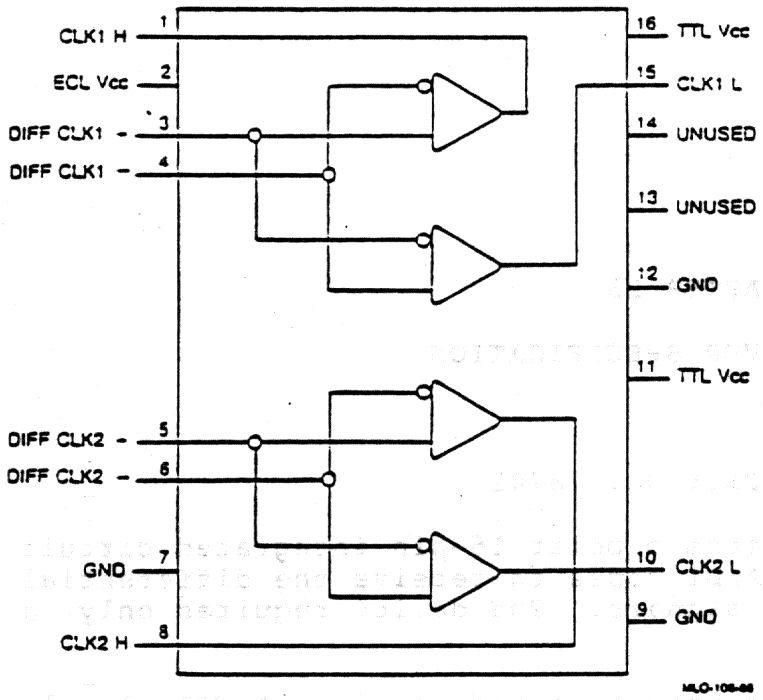
Digital Part No. 78702

The VAXBI clock receiver is a custom bipolar 16-pin integrated circuit (IC) that must be used by all VAXBI nodes to receive the differential ECL BI TIME +/- and BI PHASE +/- signals. The device requires only a +5V operating voltage.

The VAXBI clock receiver provides both true and complement TTL levels of both received differential VAXBI signals to the adapter.

This component is approved for use only in the VAXBI Corner application. See the VAXBI Module Control Drawings for placement and layout information.

digital™



NOTE:

Ground lines are tied internally.

Figure 23-1: Pin Assignments (top view)

digital™

CLOCK RECEIVER SPECIFICATION

Absolute Maximum Ratings

Pin voltages -- Input pins 3, 4, 5, 6	-1.5 to +7.0 V
Supply voltage	7 V
Maximum operating junction temperature	125 degrees C
Storage temperature range	-55 to 150 degrees C
Power dissipation	Approx. 160 mW
Current applied to TTL outputs in low state	40 mA

Guaranteed Operating Conditions

Ambient temperature operating range	0 to 70 degrees C
Frequency range of operation is limited only by Tphl and Tplh as defined.	

Package Type

16-pin plastic DIP package

digital™

23.1 DC CHARACTERISTICS

Unless otherwise specified, all specifications are at Ta = 0 to 70 degrees C, Vcc = 5.0 volts +/-5%, and at thermal equilibrium. All specifications (including capacitance values) pertain to packaged parts.

digital™

Clock Receiver DC Characteristics -- ECL Signals

Symbol	Parameter	Min.	Max.	Unit	Pin	Note/Figure	Remarks
Iih1	Input high current	-	150	uA	3, 4, 5, 6	Figure 23-2, Note 1	Vi = 4.5 V, Vcc = 5.25 V, Complement input @ 4.3 V
Iih2	Input high current	-	3	mA	3, 4, 5, 6	Figure 23-2	Vcc = GND, Pins 2, 11, 16 at GND, Vi = 4.5 V, Vdiff = 1.0 V
Iil	Input low current	-	150	uA	3, 4, 5, 6	Figure 23-2, Note 1	Vi = 2.8 V, Vcc = 5.25 V, Complement input @ 3.0 V
Vdiff	Differential threshold voltage	200	-	mV	3 to 4 or 5 to 6	Figure 23-2, Notes 2, 3	-
Vcm	Common mode voltage range	2.8	4.5	V	3, 4, 5, 6	Notes 2, 4	-
Cmrr	Common mode rejection ratio	40	-	dB	3, 4, 5, 6	Note 5	-
Icc	Power supply current	-	20	mA	-	Note 6	Vcc = 5.25 V
Cin	Single ended input capacitance	-	6	pF	3, 4, 5, or 6 to GND	-	-
| Cin + minus Cin - |	Difference in input capacitance	-	0.7	pF	3 to 4 or 5 to 6	-	-

NOTES

1. Other gate at valid logic state.
2. To be tested at Vi = 2.8, 3.55, and 4.3 V, with the complement input @ 3.0, 3.75, and 4.5 V, respectively.
3. Outputs are guaranteed to be within Vol/Voh limits if both inputs within common mode range and minimum differential input voltage is applied.
4. Common mode range is the range of total input voltage over which the output will respond to the minimum differential input voltage.

Clock Receiver DC Characteristics -- TTL Signals

Symbol	Parameter	Min.	Max.	Unit	Pin	Figure/Note	Remarks
Voh	Output high voltage	2.6	-	V	8, 10, 1, 15	Figure 23-2, Note 7	Vcc = 4.75 V, Vdiff = 200 mV, Io = Ioh max.
Vol	Output low voltage	-	0.5	V	8, 10, 1, 15	Figure 23-2, Note 7	Vcc = 4.75 V, Vdiff = 200 mV, Io = 20 mA
Ioh	Output high current	-	-1000	uA	8, 10, 1, 15	Figure 23-2, Note 7	Vcc = 4.75 V, Vdiff = 200 mV, Vo = Voh min.
Iol	Output low current	20	-	mA	8, 10, 1, 15	Figure 23-2, Note 7	Vcc = 4.75 V, Vdiff = 200 mV, Vo = 0.5 V
Ios	Output short circuit current	-25	-150	mA	8, 10, 1, 15	Note 8	-

NOTES

5. Common mode rejection ratio (Cmrr) is the ability to reject the effect of voltage applied to both inputs simultaneously. A Cmrr of 40 dB means that a 2-volt common mode voltage is processed by the device as though it were an additive differential input signal of 20 mV.

6. Combined power supply current measured with the device in a quiescent state and with no load applied.

7. Testing either (Voh and Vol) or (Ioh and Iol) is sufficient to satify these requirements.

8. TTL outputs shorted to GND or left open do not affect impedance of differential inputs and should not damage the chip. Ios is defined as current into output pin when input conditions force outputs to logic one before short application.

23.2 AC TIMING SPECIFICATIONS

Unless otherwise specified, all specifications are at Ta = 0 to 70 degrees C, Vcc = 5.0 volts +/-5%, and at thermal equilibrium.

All specifications (including capacitance values) pertain to packaged parts. All AC specifications apply with all outputs loaded and switching simultaneously.

CLOCK RECEIVER AC TIMING SPECIFICATIONS NOTES

1. Noise immunity is defined such that the TTL output levels of each receiver are not affected due to additive noise on Vcc. The noise voltage is specified over +/-5% Vcc operating range with +/-Vnse volts of additive noise.

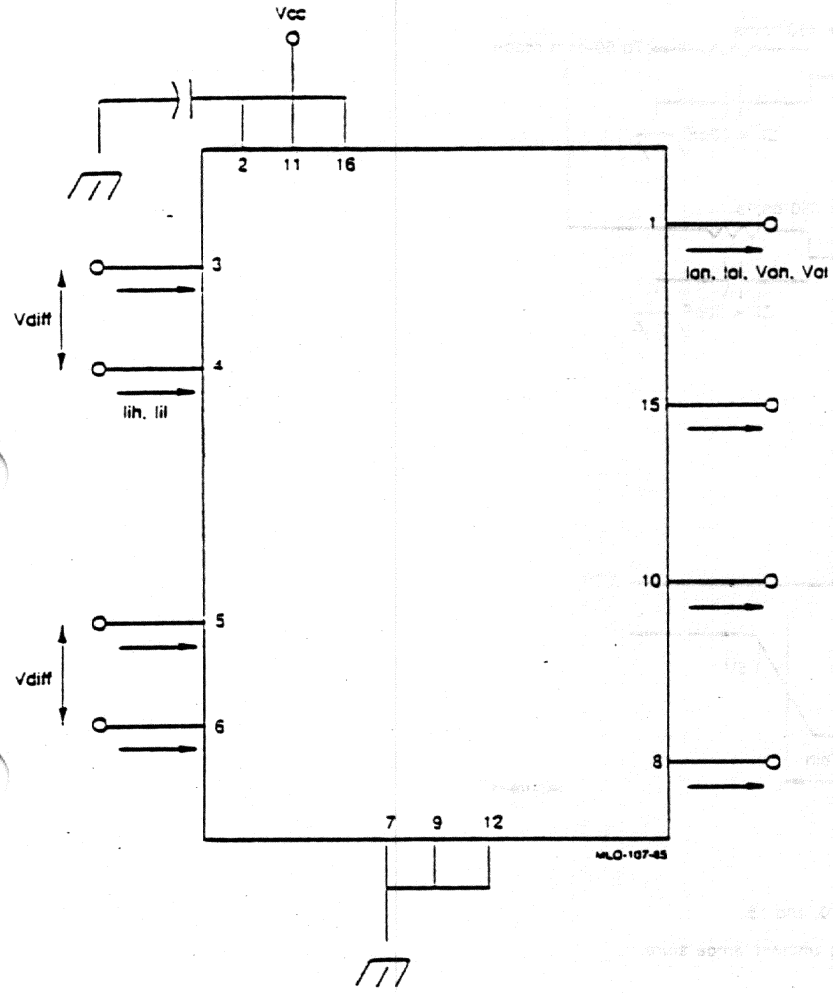
digital™

Clock Receiver AC Timing Specifications -- TTL and ECL Signals

Symbol	Parameter	Min.	Max.	Unit	Pin	Figure/Note	Remarks
Tplh	Prop L to H LVL output	1.5	6	ns	8, 10, 1, 15	Figure 23-3	Cl = 50 pF
Tphl	Prop H to L LVL output	1.5	6	ns	8, 10, 1, 15	Figure 23-3	Cl = 50 pF

Clock Receiver AC Timing Specifications -- TTL Signals

Symbol	Parameter	Min.	Max.	Unit	Pin	Figure/Note	Remarks
Tr	Output rise time	-	5	ns	8, 10, 1, 15	-	Vol max. to Voh min., Cl = 50 pF
Tf	Output fall time	-	5	ns	8, 10, 1, 15	-	Voh min. to Vol max., Cl = 50 pF
Tskw1	Single output skew	-	4	ns	8, 10, 1, 15	Figure 23-3	On individual output pins, Cl = 50 pF
Tskw2	Output to output skew	-	4	ns	8, 10, 1, 15	Figure 23-4	Output to any other pin
Vnse	VCC noise immunity	0.25	-	V	-	Figure 23-5, Note 1	
Ps/pF	Propagation delay vs. capacitance	-	40	ps/pF	8, 10, 1, 15	-	Over the 0 to 100 pF range

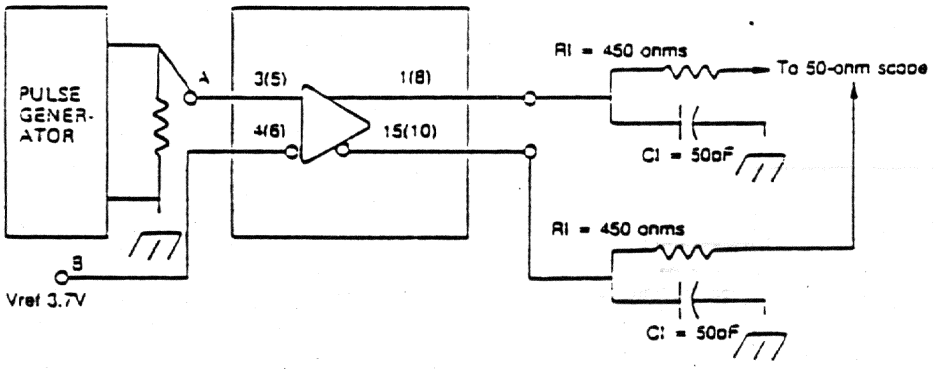


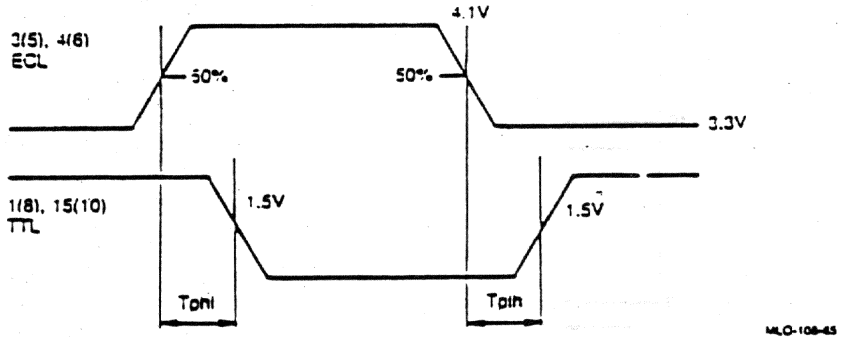
NOTES:

1. Vcc = GND for Iih2 test.
2. Pins 13 and 14 are unused.

Figure 23-2: DC Tests

digital™



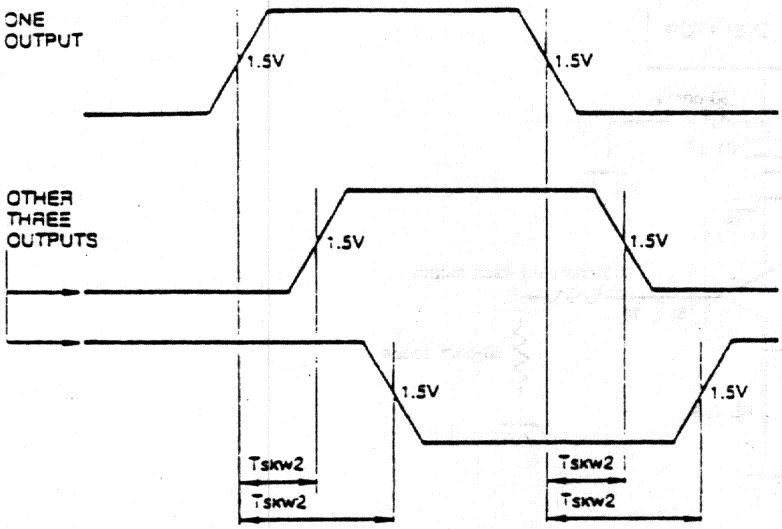


NOTES:

1. Also make measurements reversing A and B.
2. Tskw1 = greater of | Tphl – Tplh | measured for pins 1, 8, 10, and 15.
3. This test guarantees waveform symmetry at a given Vcc and ambient temperature.

Figure 23-3: Tskw1, Tplh, and Tphl Tests

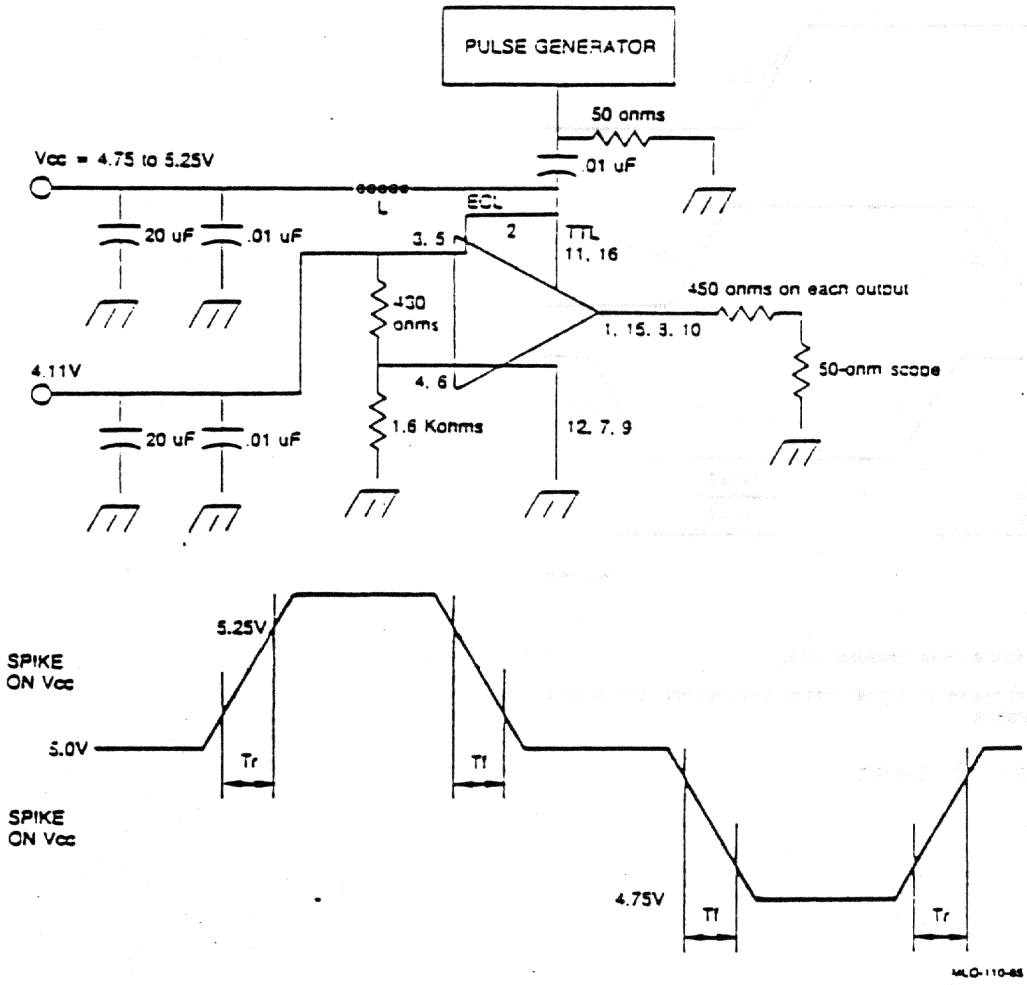
digital™



NOTES:

1. Use test fixture of Figure 23-3.
2. Tskw2 is the greater of six absolute value measurements.
3. This test guarantees maximum skews on a given board (one receiver) and is for a given Vcc and ambient temperature.

Figure 23-4: Tskw2 Test



NOTES:

1. L = 4 turns #30 wire over Ferrite bead (Ferroxcube #5659065/3B or equivalent) REF Motorola App. Note AN-592.
2. Input rise and fall times equal 1.0 ns.
3. This test guarantees that the TTL Von and Vol levels will remain between guaranteed Von and Vol limits.

Figure 23-5: Vcc Noise Immunity Test

digital™

NOTE 1

TYPES OF ADAPTERS

This note describes various types of adapters and the kinds of functions they perform. It describes some of the useful features that can be built into adapters on the VAXBI bus.

An adapter is a VAXBI node that connects peripheral (I/O) buses, communications lines, or peripheral devices to the VAXBI bus and facilitates the transfer of data and control information. Specifically, an adapter can be used to:

- Translate an address from its original form, perhaps to accommodate virtual memory
- Buffer data to smooth traffic rates between the peripheral and the VAXBI bus
- Generate interrupts to a CPU at appropriate times

Many VAXBI adapters will incorporate control and status registers (CSRs) for communication with processors. Many of the CSRs as well as VAXBI registers and other nodespace registers have to be initialized on power-up. Some registers will be initialized by the adapter, others by software (typically the operating system) running on a processor. The documentation for each adapter should clearly indicate for each register whether the register should be initialized, and if so, then to what value and whether by the adapter itself or by software.

AN1.1 PROGRAMMED I/O ADAPTERS

A programmed I/O (PIO) adapter acts as a buffer for data and control information being transmitted between the VAXBI bus and peripheral devices. The adapter can provide an interrupt capability, and it can provide buffering to an adapter-specific depth.

digital™

With a PIO adapter, all transfer of data is done by VAXBI data transfer transactions issued by processors on the VAXBI bus. The PIO adapter does not issue any transactions to carry out data transfers.

If the PIO adapter does not have interrupt capability, the processor must poll to see if the adapter requires service. If the PIO adapter has interrupt capability, the adapter issues a VAXBI INTR transaction when it requires service. The processor responds with an IDENT transaction to solicit the interrupt vector from the adapter. Such interrupts typically can be disabled by writing to a bit in a device register of the adapter.

If a peripheral or a processor transfers data in bursts of high data rates separated by periods of inactivity, it is advantageous to use a queue or silo, which collects data and delivers it as requested. Using a silo allows the data to be processed at a rate that is considerably lower than the peak data transfer rate of the peripheral. The silo can also allow for a difference in the data widths of the peripheral and the VAXBI bus. For example, if the peripheral reads data a word at a time, the silo can allow the processor to deliver an octaword at a time.

AN1.2 DIRECT MEMORY ACCESS ADAPTERS, MAPPED ADAPTERS, AND VAX PORT ADAPTERS

Unlike PIO adapters, direct memory access (DMA) adapters can issue data transfer transactions. Such transfers are done by means of VAXBI transactions without intervention by a processor.

The addresses issued by the DMA adapter must be physical memory addresses. However, a data transfer involving many pages of data is often best described in terms of contiguous virtual memory space, which means that pages in physical address space may not be contiguous. Some DMA adapters perform virtual-to-physical memory space mapping by using internal map registers. These DMA adapters, called "mapped adapters," allow a processor to specify a data transfer that involves multiple, noncontiguous pages of physical memory.*

---

*An example of a mapped adapter is the UNIBUS adapter (DWBUA).

digital™

VAX port adapters do even more than mapped adapters. They access page tables to map adapter virtual addresses into physical addresses, so that a processor can specify a data transfer in terms of virtual addresses without having to set up map registers. VAX port adapters also access main memory queues to fetch command packets and to deposit control and status information. A processor can then issue commands and examine responses independently of the adapters.*

AN1.3 BUS ADAPTERS

An adapter that connects the VAXBI bus to another bus (a "target bus") is a bus adapter. Addresses associated with transactions on the target bus can be specified in a variety of ways, and address mapping issues apply to these addresses.

A bus adapter can map all addresses falling in a "window" in the VAXBI address space into a corresponding address range on a target bus. The bus adapter replaces the most significant bits of the address, thus performing a "translation" of the window from VAXBI space to the target bus space. The address space occupied by the window typically is defined by the Starting and Ending Address Registers in the BIIC.**

---

*The CI780 is an example of a VAX port adapter.

**For example, the UNIBUS adapter is a window adapter.

digital™

NOTE 2

MAIN MEMORY AND CACHE

Data stored in memory locations distributed among one or more VAXBI nodes can be copied into caches at various nodes. The use of caches can greatly improve system performance by reducing the access time for most read-type transactions.

A memory that is located at the same VAXBI node as a processor constitutes that processor's "local" memory. "Local" pertains to an object or an action that involves only one VAXBI node. A processor node may or may not have local memory.

Some configurations of processors and memories are shown in Figure AN2-1. These configurations show only interactions between caches and memories. Configurations in which the VAXBI bus is used as an I/O bus, for example, are not shown here. The meanings of the symbols are as follows:

Pc	CPU
Pio	Input/Output Processor
Mp	Memory
S	Switch
Mwtc	Write-Through Cache
Mwbc	Write-Back Cache

digital™

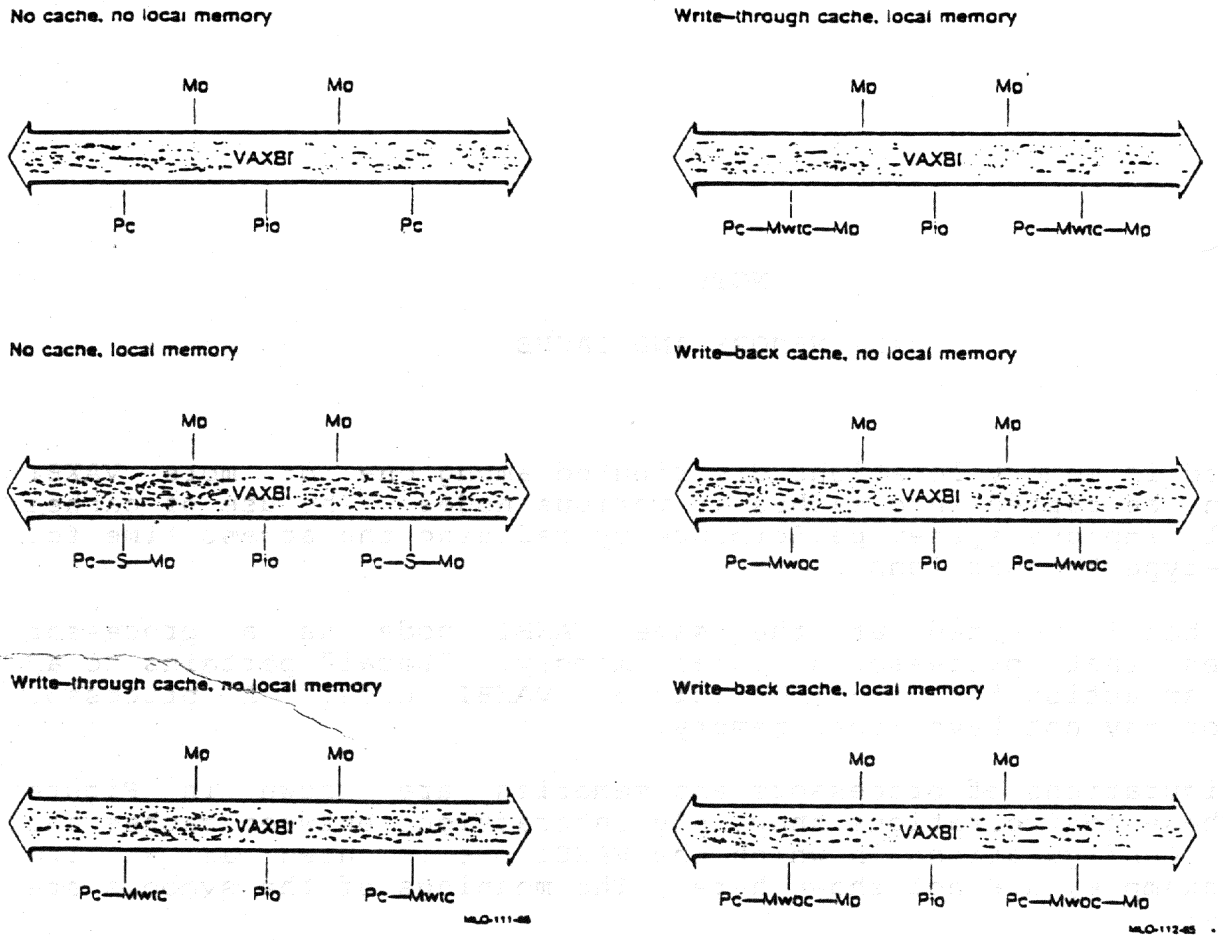


Figure AN2-1: Various Configurations

The existence of copies of data in cache memory presents the possibility that data is not kept up-to-date. (Note that translation buffers are also caches that can have "stale data." However, it is the responsibility of software to ensure the validity of data in translation buffers.) A processor might update its cache without updating the memory copy, or a processor might update memory without notifying the cache.

Various mechanisms have been proposed to counter the possibility that the main memory copy may not be up-to-date. These include write-through caches and various mechanisms associated with write-back caches. With write-through cache, each time a processor writes data to its cache a write is generated to the primary memory. With write-back cache, a processor updates the primary memory only when data in the cache is displaced by new data. In the following discussion the assumption is that caches are write-through, except in the last section, which suggests ways of designing VAXBI systems with write-back caches.

digital™

AN2.1 CACHES AND MULTIPROCESSOR SYSTEMS

The VAXBI bus provides mechanisms for dealing with caches in multiprocessor systems. Suppose two processors share the main memory. Processor A reads location L from main memory and caches the contents. Processor B then writes location L. Processor A's cache now contains an obsolete copy of location L. This cache must notice that L has been written to, and either update or -- more likely -- invalidate its copy.

This method works well if all writes to main memory are visible to processor A's cache. However, in the two configurations shown in Figure AN2-2, processor B can write to main memory without the knowledge of processor A's cache. Since main memory can be written without a transaction occurring on the VAXBI bus, processor A's cache must somehow be notified that its cached copy is no longer valid. This type of write is called a "local write," since from the VAXBI perspective the write is entirely local to the one VAXBI node.

The solutions to this problem are based on the idea that the cache on processor A must be notified when its cache contents become "invalid." The variations depend on the time at which processor A is notified so that the cache copy can be invalidated and on whether unneeded notifications are sometimes sent. If an unneeded notification is sent to the cache on processor A (for example, the corresponding main memory location was not updated; the location was not cached at processor A), system performance may suffer but an error does not occur.

digital™

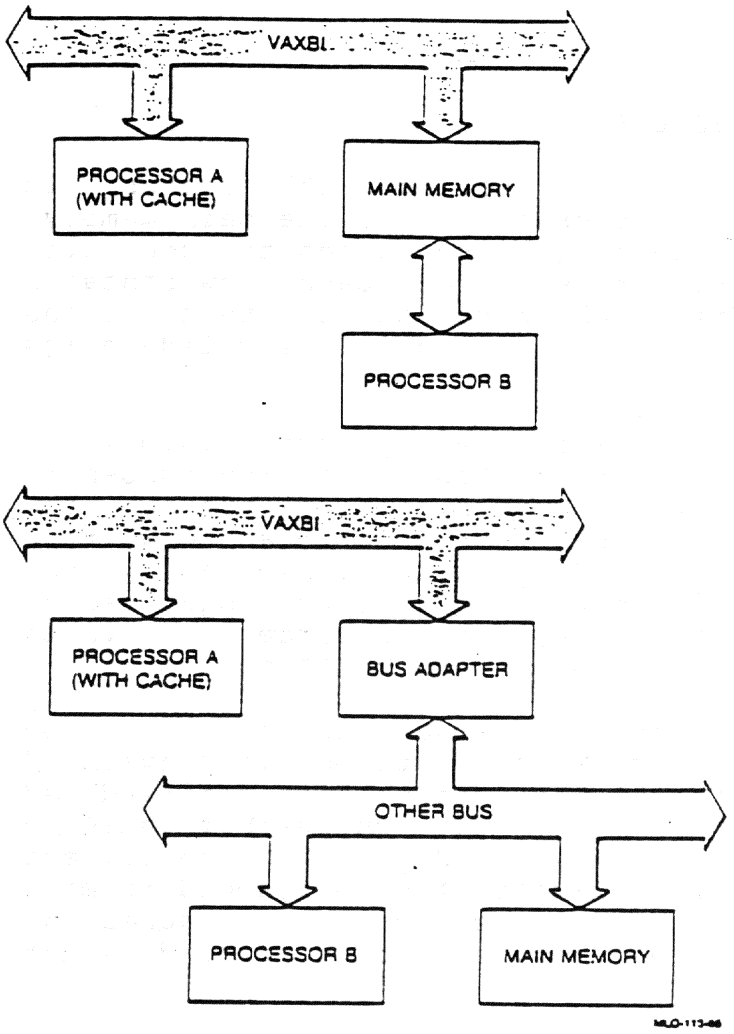


Figure AN2-2: Configurations with Nodes That Do Local Writes

Notification can be sent whenever a local write is made to the memory location. The VAXBI INVAL command provides for such notifications. In the preceding examples, INVAL is sent out on the VAXBI bus to notify processor A of an update of memory contents. If processor A's cache contains a copy, it is invalidated; otherwise, the INVAL command is ignored. Since INVAL transactions are issued even when the location has not been cached, unneeded notifications are sent.

On either a write-type command or an INVAL command, if a node requires more cycles to invalidate a cache, the node can extend the transaction by asserting the STALL code on the BCI RS lines until it has completed the invalidation. On a write-type command, not only can the node being written to extend the transaction, but all other nodes monitoring the VAXBI bus can also extend the transaction.

digital™

AN2.1.1 Notify Only After Cached Access

The number of notifications can be reduced by modifying the design so that notifications are not sent when the data cannot be in the cache.

Suppose that memory is written locally. If the memory location has been accessed since the last time a notification was sent, then a cache copy may exist, and a new notification must be sent. Otherwise, any previous cache copy would have been invalidated by the last notification, and no new notification is needed.

Unfortunately, this method works only if the data is not cached when it is written, and this is not generally the case. This suggests that we establish a method to distinguish between accesses that cache and accesses that do not.

The VAXBI protocol distinguishes between memory accesses that indicate the establishment of cache copies and those that do not. In addition to the usual read and write commands, the protocol has separate transactions that apply to caches.

- RCI -- Read with Cache Intent
- WCI -- Write with Cache Intent
- IRCI -- Interlock Read with Cache Intent
- UWMCI -- Unlock Write Mask with Cache Intent
- WMCI -- Write Mask with Cache Intent

The READ and WRITE transactions imply that no cache copy is being made.

The number of invalidates can be reduced by sending INVAL transactions only when "cache intent" transactions have occurred during or since the last write to a local memory location.

The modified algorithm is therefore as follows. When a write is made to a local memory location, if an RCI or WCI has been issued to the memory location since the last time a notification was sent, then a new notification must be sent. Otherwise, no notification is neede (although, if one is sent, the only undesirable result is the unnecessary use of bus bandwidth).

digital™

One possible implementation of this algorithm is as follows. On an RCI or WCI transaction, the responding node (the one with the memory location) uses a "cached" bit to record that a copy of the data in this memory block was made. (The size of a memory block can also vary.) When data is updated by the local processor, an INVAL transaction is sent only if the corresponding cached bit is set, indicating that a cached copy exists at some other node.

AN2.1.2 Notify Upon Read Access

Figure AN2-3 shows a configuration in which memory and processor B are located on a separate bus, which is connected to the VAXBI bus by a bus adapter. If this other bus is much faster than the VAXBI bus, sending notifications (INVAL commands) on the VAXBI bus for every write that occurs on the other bus may not be feasible; even should it be feasible, the notifications (INVAL commands) on the VAXBI bus may substantially slow down the other bus.

Slowing down the other bus is especially important when processor B accesses the memory much more frequently than processor A. In this case, an alternative may be for processor A to send an INVAL command whenever it issues a "cache intent" transaction to memory, rather than at the time that B performs a local write. In effect, the data is never cached -- at least, not if it comes from this memory. (We assume that other memory may be on the VAXBI bus, which is not shown in the figure.)

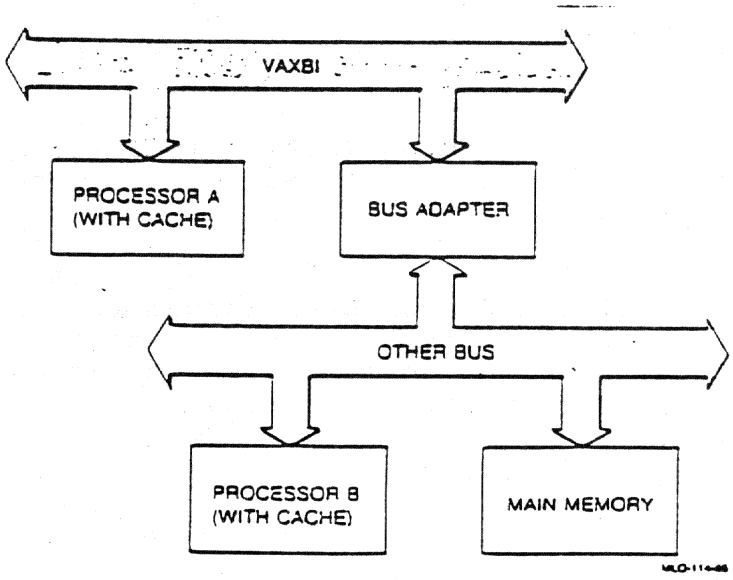


Figure AN2-3: Configuration with Multiple Buses

digital™

With this alternative, INVAL transactions must still be sent, and the problem of slowing down the other bus remains. A variation on this method is to send the notification only when processor A issues an RCI to the memory. In this case, the notification does not have to be a separate VAXBI command. Instead, it can be communicated by means of the "don't cache" return status defined on the VAXBI bus, as part of the RCI command issued by processor A. (See Chapter 5, Section 5.3.4, for the read status codes.)

But what if processor A issues a WCI and caches the data? If processor A then accesses the data in the cache, might it not obtain obsolete data? Suppose that processor A caches the data on a WCI transaction only if the location is already cached to a previous read or write. This supposition is termed "no write allocate," because cache space is not allocated on writes. Since no previous read could have cached the location (because that read must have gotten the "don't cache" notification), and the first write could not have cached the location (because the location had not yet been cached), we can show that the location will never be cached. Note that it is common for caches NOT to do write allocates, since doing write allocates buys little and complicates the cache design.

An alternative to "no write allocate" is as follows. If every write-type transaction is immediately followed by an RCI transaction to the same location with a forced cache miss on the RCI transaction, then the cache entry created by the write-type transaction is invalidated if "don't cache" status is returned on the RCI transaction.

If the "no write allocate" condition is not met and some alternative as described above is not adopted, then this scheme does not work. Therefore, it is essential that this condition be met, especially since it applies to nodes other than the one issuing the "don't cache" status. "Don't cache" status can be returned for all read- and write-type transactions, including the READ and WRITE transactions.*

We can summarize the use of the "don't cache" return status as follows. "Don't cache" status can be used instead of INVAL commands. For this reason, caches on VAXBI nodes must not do write allocates or, if they do, any cache location allocated on a write must be immediately invalidated.

---

*The caches on the VAX-11/780, 11/750, 11/730, and VAX 8200 do NOT perform write allocates.

digital™

AN2.2 REQUIREMENTS FOR VAXBI SYSTEMS WITH WRITE-THROUGH CACHE

All VAXBI nodes must conform to the following architectural rules.

- Nodes while not cacheing data should issue READ and WRITE commands on the VAXBI bus to access locations not in local memory (that is, memory which is not part of the node itself).
- Nodes while cacheing data must use cache-intent read- and write-type commands on the VAXBI bus to access locations not in local memory.
- Nodes that cache data must monitor the VAXBI bus; locations designated in write-type commands and INVAL commands must be invalidated.
- Nodes must not cache any data returned with a "don't cache" status.*
- Nodes must not cache data on a write transaction unless, just before the data is written, the cache contained valid data for the location. This rule is known as "no write allocates."
- Nodes that respond to read- and write-type commands to memory space must either (a) issue INVAL commands on writes to local memory or (b) return "don't cache" status to RCI and IRCI commands if writes to local memory are possible to the specified locations. It is optional for these nodes to return "don't cache" status to other read-type commands.
- Reads to I/O space and all interlock reads must not result in cache hits. Even if the data being read has been cached locally, the cached data must be ignored on these transactions and a VAXBI read-type transaction must be generated.

Note that memory nodes accessible only through the VAXBI bus do not have to return "don't cache" status or issue INVAL commands, because local writes are not possible.

---

*An exception has been granted for the KA820 processor.

digital™

When applied to specific cases, the rules listed above can be simplified as follows:

- A node with no local memory (although, of course, it may have a cache) has no need to issue INVAL commands.
- A node with no cache memory should not issue cache intent commands.
- A node with local memory need not record whether RCI or WCI transactions have been issued to locations in the local memory since the last write-type transaction. If this information is not recorded, either all accesses to local memory receive "don't cache" confirmations or all local writes generate INVAL commands.

AN2.3 HOW TO DESIGN VAXBI SYSTEMS WITH WRITE-BACK CACHE

In systems with write-back cache, memory may not contain up-to-date data for every location. In fact, it may not be possible to determine which node has the up-to-date data unless each node is polled. For this reason, it is difficult to use write-back caches in multiprocessor systems that share memory.

Described below are several methods for designing VAXBI systems with multiple processors and write-back caches. In these methods, all accesses from the VAXBI bus to local memory are routed through the cache first, so that the cache is invisible to other processors that are accessing local memory. The only variable is how this processor accesses nonlocal memory.

- Cache-only local memory. The cache simply becomes part of the local memory.
- Cache all locations, but perform write-through on nonlocal memory locations.
- Cache-only memory locations that are not written locally. This method makes the write-through/write-back question moot, as far as this node is concerned.

These methods are valid in configurations that also include nodes that have write-through caches.

digital™

NOTE 3

RECOMMENDED USAGE FOR BIIC REGISTERS

This note offers some general strategies for using the BIIC registers. The strategies are primarily intended for designers of I/O adapters that use the BIIC, although some suggestions will be of interest to designers of other types of nodes and to software users of the VAXBI bus. The discussion is limited to the normal operation of the BIIC and does not cover self-test or diagnostic operations. Detailed information on the BIIC registers appears in Chapter 7.

The BIIC registers are the means by which an operating system (OS) and an adapter engine (AE) communicate. An OS is a collection of system software executing on a processor; the VAX/VMS operating system is an example. An AE is a collection of logic within a VAXBI node and is typically implemented in microcode. The BIIC referred to in the discussion below is the one on the adapter node. This note suggests how to divide the responsibilities between the OS and the AE and discusses the initialization and control of BIIC registers.

AN3.1 ADDRESS REGISTERS

The Starting Address Register (SADR) and Ending Address Register (EADR) define a window from VAXBI address space to an address space within this VAXBI node. The granularity of the window size is 256 Kbytes. This window has two applications:

1. Memory Applications

   For memory class nodes, the size of the window is defined by the amount of memory on the node.

   For slave-only memory nodes, the SADR and EADR must be initialized by the primary processor during recovery from

digital™

transient power outages and system crashes.* This initialization must occur before the OS is running.

If memory is battery backed up, the processor must restore the SADRs and EADRs to the configuration they were in before the power was lost to ensure that the memory addressing is not scrambled.

2. I/O Applications

For bus adapter nodes that map to another I/O bus (called "window adapters"), the size of a window can be 256 Kbytes (node window space) or some multiple of 1 megabyte (assignable window space). The UNIBUS adapter, for example, uses this node window to map from the VAXBI bus to the UNIBUS. If a boot program that loads the OS uses a device on a mapped I/O bus, the boot program must initialize the SADR and EADR. The OS subsequently can reload the SADR and EADR with the same addresses.

NOTE

ALL subsequent number constants in this section (AN3.1) are in hexadecimal, unless otherwise noted.

For an adapter node with node ID n, that uses node window space, the SADR must be set to 2040 0000 + n * 0004 0000, and the EADR must be set to 2044 0000 + n * 0004 0000. In a system with multiple VAXBI buses, a processor that can access all I/O adapters on all VAXBIs refers to the 0-th byte in the adapter node window of the n-th node of the m-th VAXBI as 2040 0000 + n * 0004 0000 + m * 0200 0000; however, the address transmitted on that m-th VAXBI bus is 2040 0000 + n * 0004 0000 (the m-value is not transmitted on the VAXBI bus).

For an adapter that uses an assignable window, the SADR will be set to A = 2080 0000 + 10 0000*k where k is an integer with 0 <= k <= 17 and the EADR will be set to 2080 0000 + 10 0000*(k + N) where N is the number of megabytes that the adapter needs. The value of k is assigned by the operating system when it sets up the hardware configuration.

The node designer can specify the size of the assignable window and whether or not the window must be aligned to start

---

*This is necessary so that the primary processor can search for a VAX/VMS restart parameter block or for a good block of memory.

digital™

on a natural power of two address boundary. If the node designer requests alignment on a natural power of two boundary, then the operating system will align it on the smallest power of two boundary that is greater than or equal to the requested size of the window, unless the designer requested size is greater than 16 (decimal) megabytes, in which case all 24 (decimal) megabytes of assignable window space will be allocated to the node. The operating system may perform this alignment even when it is not required by the designer, in an attempt to configure all requesting nodes.

Adapter nodes that do not map to other buses can use node windows or assignable windows to map to other resources, such as memories in I/O space. Most adapters will not require any window, and the OS should not initialize the SADR and EADR unless the adapter has a window.

The SADR should be loaded prior to loading the EADR. If the registers must be reloaded with different values, such as for a dynamic memory mapping scheme, the EADR should be cleared before the SADR is reloaded. These procedures prevent double-mapping of physical memory space.

AN3.2 BCI CONTROL AND STATUS REGISTER

The BCI Control and Status Register (BCICSR) is used by the AE to control the BIIC. Generally, a node will initialize its own BCICSR and not subsequently change the value, although some nodes may dynamically change some of the bits in their BCICSR.

BCICSR will normally be ignored by the OS; the OS should not change BCICSR. (The slave-only node is an exception: the OS must initialize BCICSR to 0000 2100 hex to enable STOP recognition and user interface CSR space, since slave-only nodes cannot access their own BIIC registers.)

The OS can manipulate bits in the BCICSR for some types of nodes. This could be used to disable some capability which the node normall_ uses, such as recognition of interprocessor interrupts. Two risks, however, should be considered:

- A capability may be enabled in the BIIC that the node's AE cannot cope with, such as setting the INVALEN bit.
- If both the OS and the AE attempt dynamic control over the BCICSR, a race condition results. A read-modify-write processor instruction, used by the OS to access another node's

digital™

BIIC registers, is not necessarily translated into an atomic sequence of VAXBI transactions, and the BIIC registers do not support locks.

AN3.3 BUS ERROR REGISTER

The Bus Error Register (BER) records errors on the VAXBI bus that are detected by the BIIC and reported to the OS. The BIIC sets the bits, and the OS clears them.

When the OS clears a bit in the BER, by writing a one to that bit, the OS must then reread the BER. If another error has occurred, the OS must handle that error and again clear the bit. This procedure prevents some cases of missing interrupts: the BIIC sends an interrupt when the Hard Error Summary (HES) bit or the Soft Error Summary (SES) bit in the VAXBI Control and Status Register changes from cleared to set, if the appropriate interrupt enable bits are set.

The OS should clear the BER Null Bus Parity Error bit on a node immediately after performing a node reset on that node, but not before the target node's Broke bit has cleared. Note that the Broke bit must not be read within 500 ms of the setting of NRST, and that, even after 500 ms, another node reading the Broke bit might receive a NO ACK response. The clearing of NPE is desirable because a VAXBI primary interface may experience a spurious setting of the NPE bit as a result of undergoing a node reset.

The BER is normally ignored by the AE; it is not written or read.

AN3.4 VAXBI CONTROL AND STATUS REGISTER

Most of the fields in the VAXBI Control and Status Register (VAXBICSR) are read only, both to the OS and to the AE.

The AE should write to the VAXBICSR only once: as the last step in node self-test. If the STS bit is set (meaning that the BIIC passed self-test) and if the rest of the node passed self-test, then the AE should clear the Broke bit. (The slave-only node is an exception: it does not access the VAXBICSR, and the indicator of the result of self-test is not in this register.)

For most nodes, the OS should also write to the VAXBICSR only once. During OS initialization the OS should write ones to the HEIE and SEIE bits to enable interrupts when the BIIC detects hard errors and soft errors. For most nodes, the AE could set HEIE and SEIE when clearing the Broke bit. However, since slave-only nodes cannot set their own HEIE/SEIE bits, the recommended default is that the OS set the HEIE

digital™

and SEIE bits for all nodes.

For all designs, it is the responsibility of the OS to control the value of the Arbitration Control (ARB) field. The standard setting for these bits is 00, for dual round-robin arbitration. The ARB field can be set to fixed-high or fixed-low arbitration to handle special applications if an exception is obtained.

Fixed-high and fixed-low priority arbitration are intended to be invoked only as a last resort for real-time performance extremes, and should not be used on any system without carefully analyzing node ID assignments for a particular system and the performance impact.

The OS can also set the ARB field to 11 to disable arbitration, thus preventing a defective node from generating transactions.*

Since HEIE, SEIE, and ARB are all read-write bits in the same byte of the VAXBICSR, the entity (OS or AE) that writes a new value to the ARB field should be careful not to mistakenly write a new value to the HEIE and SEIE bits. Use of the HEIE and SEIE bits is discussed in Section AN3.7.

AN3.5 DEVICE REGISTER

The Device Register (DTYPE) must be loaded by the AE prior to clearing the Broke bit. It is preferable, but not required, that the DTYPE be loaded even if the node fails self-test. A value of all ones indicates that the register is not yet loaded; a value of all zeros is RESERVED.

---

*The OS must disable arbitration on a target node before performing a node reset on that node.

digital™

The DTYPE should be treated as read-only information by the OS.

The Device Register contains two 16-bit fields: the Device Type field and the Device Revision field. Values for the Device Type field are assigned by Digital for all VAXBI nodes.* The Device Revision field is intended to reflect functional changes, and the meaning is node-specific. Some nodes will subdivide the revision field, and some nodes will have secondary revision information outside of the DTYPE.

AN3.6 GENERAL PURPOSE REGISTERS

Four registers (GPR0, GPR1, GPR2, and GPR3) are available for general communication between the OS and the AE. All four can be read and written by the OS and by the AE.

The Write Status Register (WSTAT) contains status bits for the GPRs. When the OS writes to one of the GPRs, the BIIC sets a bit in the WSTAT; when the AE accepts data from a GPR, it may clear the status bit by writing to the WSTAT. If the AE writes to a GPR using a VAXBI transaction, the corresponding status bit will be set. The bits are not set for loopback transactions.

The purpose and use of the GPRs is determined by the node designer.

AN3.7 INTERRUPT REGISTERS

The BIIC supports several approaches to handling interrupts between the AE and the OS. Four interrupt levels are supported, and it is possible for nodes to dynamically control the level. Interrupts can target multiple nodes and can be masked at both the source and destination nodes.

---

*See Appendix G for details of obtaining a device type code assignment.

digital™

Device interrupts are jointly controlled by the contents of the Error Interrupt Control Register (EINTRCSR), the Interrupt Destination Register (INTRDES), and the User Interface Interrupt Control Register (UINTRCSR). Interrupts can be requested either by the AE or by the BIIC and are fielded by the OS. Device interrupts include two classes of interrupt causes: error interrupts and user interrupts. Error interrupts are initiated by the BIIC when a bus error is detected, while user interrupts are initiated by the AE to signal I/O events. Interprocessor interrupts differ from device interrupts and are discussed below in Section AN3.9.

All nodes can issue interrupts. Before any user interrupts can be posted, the following must be done:

- Write a mask into the INTRDES listing the node(s) to which interrupts should be sent.
- Write a user vector into the UINTRCSR.

For the OS to enable a node to post error interrupts, the following must be done:

- Write an error vector and level information into the EINTRCSR.
- Set the HEIE and SEIE bits in the VAXBICSR.

AN3.7.1 Interrupt Destination Control

The mask in the INTRDES determines where all interrupts from a node will be sent, both errors and normal completion interrupts, and both BIIC-detected events and AE-detected events.

If the INTRDES = 4, then all interrupts will be sent to node 2. (This is the standard node ID for the primary processor in most VAXBI-based systems, according to a Digital internal manufacturing convention.) If the INTRDES = 5, then all interrupts will be sent both to node 2 and node 0.

If the INTRDES = 0, then no interrupts will be sent but will be marked within the BIIC as aborted; since this can potentially cause los interrupts, the INTRDES should be initialized by the OS prior to its initialization of the VAXBICSR, EINTRCSR, and UINTRCSR.

digital™

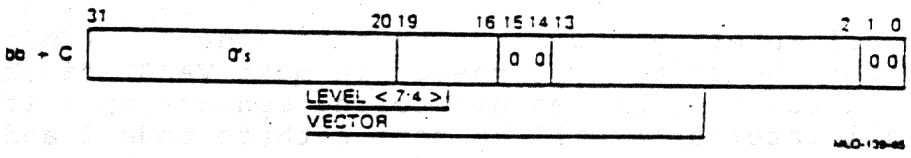
If INTRDES = FFFF, then all interrupts will be sent to all nodes. Since only processor nodes will acknowledge interrupts (by setting the INTREN bit in their BCICSR registers), this value may appear to be a reasonable setting for symmetric multiprocessor systems. However, broadcasting interrupts to all nodes requires the OS to disable recognition of interrupts in some destination nodes, by clearing the INTREN bit in BCICSR registers. It is recommended that OSs exercise interrupt mask control at the source rather than at the destination (INTRDES rather than BCICSR) to avoid any potential hazards in modifying the BCICSR).

The mask in the INTRDES determines the target node(s) for all interrupts from a node. OSs that direct all interrupts to the primary processor should write the same mask to the INTRDES of all nodes (including the primary processor's node). This mask should have one bit set. OSs that direct all interrupts to a set of processors (CPUs, not IOPs) should write the same mask to the INTRDES of all nodes. This mask will have multiple bits set. OSs may use different masks for different nodes to balance the interrupt load.

It is expected, and hoped, that all OS and AE designers will use only static mask settings in the INTRDES registers. Dynamic control of INTRDES masks is possible but difficult to control without a risk of losing or misdirecting interrupts.

AN3.7.2 Error Interrupt Vector Control

The error interrupt vector and control is in the Error Interrupt Control Register (EINTRCSR). This register must be initialized by any OS running on any processor to:



Bits: 31:20 Initialize to zero

Bits <31:20> include some MBZ bits and some bits that should be zero. To be safe, it is recommended that a value of 01An nnnn hex be written when the OS initializes the EINTRCSR, which clears the INTRAB, INTRC, Sent, and Force bits.

digital™

Bits: 19:16 Level <7:4>

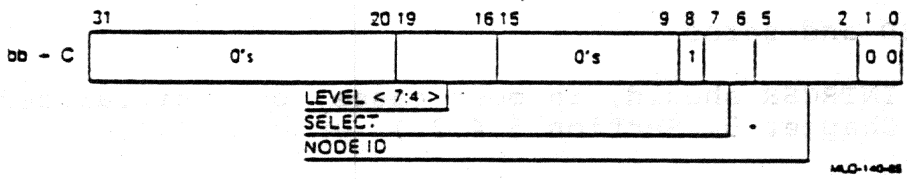
Bits <19:16> is a mask field that identifies the level(s) at which INTRs will be transacted. Although it is possible to concurrently interrupt at more than one level, doing so is not advantageous. Issuing an INTR at more than one level causes multiple IDENTs to occur, which increases both traffic on the VAXBI bus and processor overhead. It is recommended that the Level field contain only a single asserted bit.

Bits: 15:14 RESERVED and zeros

Bits: 13:2 Vector

Bits: 1:0 RESERVED and zeros

For VAX processors the Vector field is broken down to specify the Select code and node ID. The EINTRCSR should be initialized to the following. (See Chapter 5, Section 5.4.2.)



Bits: 7:6 Select

For VAX processors each VAXBI node is allocated four interrupt vectors within the system control block (SCB). The Select field indicates which of the four vectors is to be used for this node.

Bits: 5:2 Node ID

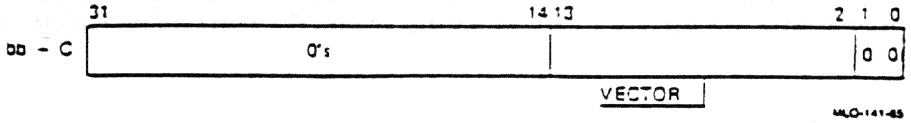
The Node field is the encoded node ID.

Since the BIIC can initiate interrupts independently of the AE and the OS, the EINTRCSR is not intended to support dynamic control of either the level or the vector by either the AE or the OS, due to the risk of losing interrupts. It is recommended that the EINTRCSR be statically controlled by the OS rather than by the AE.

digital™

AN3.7.3 User Interface Interrupt Vector Control

The user interface interrupt vector and control are in the User Interface Interrupt Control Register (UINTRCSR). This register should be initialized by any OS running on any processor to:



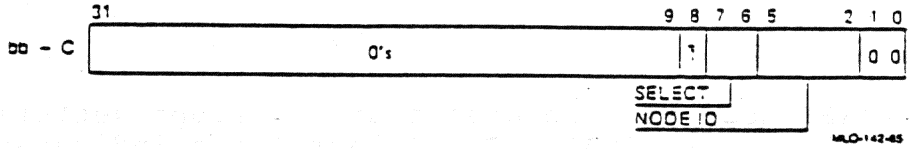
Bits: 31:14 Initialize to zero

Bits <31:14> include some MBZ bits and some bits that should be zero. To be safe, it is recommended that a value of FFF0 nnnn hex be written when the OS initializes the UINTRCSR, which clears the INTRAB, INTRC, Sent, and Force bits.

Bits: 13:2 Vector

Bits: 1:0 RESERVED and zeros

For VAX processors the UINTRCSR should, in most cases, be initialized to the following. (See Chapter 5, Section 5.4.2.)



Bits: 13:9 RESERVED and zeros

Bit: 8 Initialize to one

Bits: 7:6 Select

Bits: 5:2 Node ID

AN3.8 INTERRUPT STRATEGIES

The best strategy for implementing interrupts is the simplest strategy that satisfies the needs of a node. I/O adapters are expected to vary significantly, in terms of their needs for interrupt control, from static and simple to dynamic and complex.

digital™

It is recommended that the level(s) and vector(s) used be controlled by the OS rather than by the AE. (This cannot be done for the UNIBUS adapter, since UNIBUS devices control their own level and vector.) It is anticipated that some nodes will always interrupt at the lowest level (level 4), due to inherently modest requirements for interrupt latency. For most nodes, however, it is recommended that level and vector be controlled by the OS.

Two advantages to OS control of level and vector are as follows:

- The significance of level and vector may vary from one OS to another, as well as from one application to another and from one system manifestation to another, even under the same OS.

- It is generally more cost-effective to set level and vector from RAM-based OS macrocode than from ROM-based AE microcode.

The sections that follow describe several strategies based on the complexity of many possible types of VAXBI nodes. The discussions are ordered from the simplest needs to more complex needs. Some of these strategies may never be implemented, and a complex interrupt strategy does not necessarily imply a complicated or expensive implementation.

These descriptions all assume that interrupt levels and vectors are generally controlled by the OS. The taxonomy used is based on the perspective of the AE. Static level control means that the levels are chosen by the OS and are not altered by the AE, while dynamic level control means that the allowable range of levels is chosen by the OS and a specific level is chosen by the AE when an interrupt is to be posted. The difference between static and dynamic vector control is analogous to the difference for level control, although the reasons for making this choice are different.

AN3.8.1 The "No-Interrupt" Case

For nodes that do not have an AE that invokes interrupts, the content of the UINTRCSR is irrelevant. While few (if any) I/O adapters will not post nonerror interrupts, slave-only nodes and most processors are in this category.

digital™

AN3.8.2 One Interrupt -- With Static Level and Static Vector Control

For a node whose interrupt needs can be met with static control of a single level and vector, there are two design choices:

- Use the level and vector in the EINTRCSR and ignore the contents of the UINTRCSR. When the AE wishes to post an interrupt, it need only set the force bit in the EINTRCSR.

- Use the level and vector in the UINTRCSR and rely on the OS to set the same level and vector as in the EINTRCSR. This scheme has one disadvantage: the AE must know which level to use, in order to set the correct force bit (or assert the correct INT<7:4> signal). Since the preferred strategy is OS control of level, the AE must read the Level field from the EINTRCSR or extract the level from some other OS-controlled register before posting the interrupt.

AN3.8.3 Two Interrupts -- With Static Level and Static Vector Control

For a node whose interrupt needs can be met with static control of two level/vector pairs (one for BIIC-detected events and one for AE-detected events), there is little choice. The AE must extract the level from some OS-controlled register before posting the interrupt, in order to use the level chosen by the OS, and the AE must ignore the EINTRCSR contents.

This strategy does permit error interrupts to be posted at a higher level than nonerror interrupts; however, such posting is not a common need. Whether the BIIC or the AE detects an event, it is likely to be handled by the same OS driver, and there is little advantage in having separate, statically controlled levels based on which piece of the node detected the event, when either interrupt path leads to the same OS driver.

This strategy is more likely to be of value when using the same level, but different vectors, for the two interrupts; the portion of the driver that handles nonerror interrupts can bypass some error-handling code.

digital™

AN3.8.4 2-4 Interrupts -- With Dynamic Level and Static Vector Control

The interrupt capabilities of the BIIC allow a node to exercise dynamic interrupt level control with static interrupt vector control. A node's AE could pick an interrupt level based on the urgency of the event detected; or, a node could post a low-level interrupt initially, and subsequently post a high-level interrupt if the low-level interrupt was not serviced within some time limit.

If this type of control is needed, it is recommended that:

- The OS should set the Level field and the vector in the EINTRCSR, based on the highest interrupt level that the node may use. The OS should also set the same vector in the UINTRCSR.

- The AE should read the Level field in the EINTRCSR to determine the allowable range of levels for posting interrupts, pick a level based on urgency or some other criterion, and post the interrupt by means of the UINTRCSR.

AN3.8.5 2-4 Interrupts -- With Static Level and Dynamic Vector Control

A node may also exercise dynamic vector control of interrupts, with static level control, for up to four vectors. (Since the system control block limits the number of vectors to four, this restriction applies only to nodes controlled by VAX processors.) This case allows a node to support up to four controllers, with a separate OS driver for each controller. This form of interrupt control requires that:

- The OS must inform the AE of the 2-4 vectors, using some unspecified node-specific mechanism. The OS could, for example, pass one vector in the UINTRCSR, with a joint understanding that the other vector(s) vary only in the Select field from the passed vector.

- The OS must inform the AE of the association of vectors to controllers, using another node-specific mechanism.

digital™

- When the AE wishes to post an interrupt for a controller, it must read the UINTRCSR to verify that no interrupt is pending by means of the UINTRCSR. Any pending interrupt takes precedence over this interrupt. Otherwise, the AE may write the vector and the force bit into the UINTRCSR. Since this write serializes all interrupts from this node, this mechanism does not support preemptive interrupt priority and, therefore, is limited only to cases in which all controllers can use the same level.

- The OS must write a vector and level to the EINTRCSR that corresponds to one of the vectors associated with the UINTRCSR. When an interrupt is posted by means of the EINTRCSR, the servicing driver may have to notify the other drivers that support this node by some OS mechanism.

AN3.8.6 2-4 Interrupts -- With Dynamic Level and Dynamic Vector Control

A node may also exercise dynamic vector control of interrupts, with dynamic level control, for up to four vectors. (Since the system control block limits the number of vectors to four, this restriction only applies to nodes controlled by VAX processors.) This case allows a node to support up to four controllers, with a separate OS driver for each controller. This form of interrupt control requires that:

- The OS must inform the AE of the 2-4 vectors and their levels, using some unspecified node-specific mechanism. The OS must also inform the AE of the association of vectors to controllers.

- The OS must initialize the UINTRCSR, setting the External Vector bit (the vector is irrelevant). The recommended value to be written is FFF0 8000 hex.

- To post an interrupt for a controller, the AE asserts the corresponding interrupt level request (either by asserting one of the INT<7:4> lines or by setting a force bit in the UINTRCSR).

When the OS subsequently (implicitly, for VAX computers) issues the IDENT transaction, the AE supplies the corresponding vector.

The OS must write a vector and level to the EINTRCSR that corresponds to one of the highest priority vectors associated with this node. When an interrupt is posted by means of the EINTRCSR, it may be necessary for the servicing driver to notify other drivers that support this node.

digital™

AN3.8.7 2-128 Interrupts -- With Dynamic Level and Dynamic Vector Control

A node may also exercise dynamic vector control of interrupts, with either static or dynamic level control, for up to 128 vectors. (The limit of 128 vectors is due to page alignment and is, strictly speaking, VAX-specific. However, it is assumed that 128 vectors is adequate for any node and any processor, so no more general case will be discussed.) This form of interrupt control requires that:

- The OS must inform the AE of the (up to 128) vectors, and their associated levels, using some unspecified node-specific mechanism. The OS must also inform the AE of the association of vectors to controllers. (Alternatively, specifically for the UNIBUS adapter, the devices may control the vector and level themselves.) These device vectors are only seven bits wide: DEV_VECTOR<8:2>.

- The OS must initialize the UINTRCSR, setting the External Vector bit (the vector is irrelevant). The recommended value to be written is FFF0 8000 hex.

- The OS must initialize some register, external to the BIIC, known as a Vector Offset Register (VOR). For VAX computers, the VOR points to a page of secondary interrupt vectors in the SCB.

- To post an interrupt for a controller, the AE asserts the corresponding interrupt level request (either by asserting one of the INT<7:4> lines or by setting a force bit in the UINTRCSR).

When the OS subsequently (implicitly, for VAX computers) issues the IDENT transaction, the AE supplies the corresponding vector. It is formed by the AE through concatenation:

VAXBI_VECTOR<13:0> = VOR<13:9>'DEV_VECTOR<8:2>'00

The OS must write a vector and level to the EINTRCSR that corresponds to one of the highest priority vectors associated with this node. When an interrupt is posted by means of the EINTRCSR, it may be necessary for the servicing driver to notify other drivers that support this node.

digital™

AN3.9 INTERPROCESSOR INTERRUPT REGISTERS

Interprocessor interrupts are controlled by the IPINTREN bit in the receiving node's BCICSR and the values in three registers in the BIIC:

- The value in the IPINTR Mask Register (IPINTRMSK) is a mask on the receiving node of IPINTR transactions. If node 9 wants to receive IPINTRs from nodes 2 and 1, for example, then the OS should set node 9's IPINTRMSK to 6. This value can generally be set statically.

- The value in the IPINTR Source Register (IPINTRSRC) contains a record for the receiving node of which other nodes have sent IPINTR transactions to this node. An IPINTR sent from node 2 to node 4 results in setting IPINTRSRC bit <2> in node 4's BIIC. It is the responsibility of the portion of the OS (or some equivalent entity) that executes on the receiving node to clear this bit. The IPINTRSRC need not be initialized by the OS, since it is cleared by the BIIC during power-up.

- The value in the Force-Bit IPINTR/STOP Destination Register (FIPSDES) determines the destination node for Force-Bit IPINTRs. The FIPSDES must be dynamically controlled by the transmitting node, since IPINTR transactions are used for two very different functions:

  a. IPINTRs may be sent between peer processors to signal an event (frequently supplemented by a message stored in some portion of shared memory). Typically, an IPINTR will be sent from one processor to all peer processors.

  b. IPINTRs may also be sent between a CPU and an IOP (an I/O, or front-end, processor). A CPU could use this mechanism to signal an IOP of a state change in some shared data structure.

  To permit IPINTR to be used for both these functions on the same system, it is necessary that the FIPSDES be dynamically controlled by the transmitting node when the node is a processor.

Since some operating systems treat the receipt of an IPINTR transaction as indicating some predefined message from a peer processor, I/O adapters should not normally issue IPINTR transactions to processors.

digital™

The preceding strategy of static AE control of the BCICSR, static OS control of the IPINTRMSK, and dynamic OS control of the FIPSDES is not the only viable scheme. This strategy has been used, however, by VAX processors and by Digital I/O adapters, and is therefore the recommended strategy for VAXBI node designs that use IPINTRs.

AN3.10 RESERVED REGISTERS

These registers are RESERVED for use by Digital and are not implemented in the current BIIC. Reads return all zeros for data, and writes discard the data.

digital™

NOTE 4

SELF-TEST

This application note discusses the intended goals of self-test and then comments on the implementation of self-test for various lengths of self-test. Section AN4.4 then gives recommended VAXBI transaction data patterns that a node should perform during self-test. Section AN4.5 discusses how to indicate faults in multimodule nodes. Finally, Section AN4.6 discusses use of the Broke bits and the BI BAD L line to determine the results of self-test for nodes on a VAXBI bus.

AN4.1 RECOMMENDED GOALS OF SELF-TEST

The strategic goals of VAXBI self-test, in their intended order of priority, are:

1. Detection of failed nodes. For simple nodes, self-test can lead directly to system repair by node replacement. For more complex nodes, self-test serves as a guide for selecting the correct repair agent or diagnostic procedure.

2. Systemwide fault coverage. The goal is to detect faults in system hardware before the execution of any system software. System software can then decide not to use broken hardware. Warm starts can (and generally should) be disabled if a node fails to survive a transient power outage.

3. Fault isolation of replaceable units within a node. This goal seeks to eliminate the need for secondary diagnostic procedures and can be used for adaptive system software.

The tactical goals of self-test to support the strategic goals, in their intended order of priority, are:

1. Minimal dependency on the correct operation of other VAXBI nodes. While some dependency is unavoidable, the goal is fault isolation to the VAXBI node. If a system has only one fault and it is in a node, then the broken node should have

digital™

its Broke bit set and all other nodes should have their Broke bits clear. (Some systemwide faults cause many fault-free VAXBI nodes to leave their Broke bits set: faulty power supplies and missing VAXBI terminators are examples.)

2. Maximum coverage for elements that are likely to fail on site. Experience indicates that test emphasis should be placed on connectors, sockets, cables, device leads, and device bond wires.

3. Coverage as far out (from the VAXBI bus) as can be done without configuration dependencies. Self-test is specifically not restricted to testing the VAXBI modules; self-test is also intended to be a test of external buses, controllers, and devices.

   Testing of external devices does require some node-specific decisions about the meaning of "broke." If a node controls a mandatory device and several optional devices, then the node should be declared broken if the mandatory device is missing or if any device has a fault.

4. Maximum coverage for gate failures (solid or stuck-at faults).

5. Fault isolation within a node.

AN4.2 NORMAL AND FAST SELF-TESTS

In node designs that use the BIIC as the VAXBI primary interface, all aspects of VAXBI interface self-test are handled by the BIIC. They include the VAXBI register self-test, VAXBI control logic tests, and the watchdog timer that provides the BI NO ARB L deassertion if the power-up self-test fails.

The limit on the number of transactions allowed during a fast self-test is 512; for a normal self-test the limit is 2048 transactions. This number includes both loopback and VAXBI transactions.

AN4.2.1 Normal Self-Test

The purpose of the 10-second limit is to define a device-independent maximum self-test time interval. If a node fails to complete self-test within 10 s, then the node may be declared broken. It is anticipated that processors will restart the system software as soon as all nodes pass self-test or after 10 s (whichever comes first), and

digital™

it is anticipated that the form of restart employed (warm start versus cold start) will be determined by whether all nodes pass self-test within 10 s.

Cold start means initiating execution of a fresh copy of the system software, while warm start means continuing the (interrupted) execution of a previous copy of the system software. It is recommended that VAXBI-based systems convert warm-start situations (such as transient power outages) to cold start if any node fails self-test.

AN4.2.1.1 Effect of Bus Traffic - Nodes perform VAXBI transactions as part of their self-test, so they must arbitrate with other nodes to gain control of the bus. Thus, the amount of traffic on the bus affects the length of a node's self-test. The node designer needs a way to determine that the node will complete self-test within the 10-second limit regardless of other activity on the bus. The designer can do this by using the maximum number of transactions, and the fact that all transactions by the self-testing node during self-test time must be addressed to itself, so that no STALL cycles are allowed. These facts bound the bus latency suffered by any one node.

A worst-case criterion has been developed to bound self-test time. We find that, if the node can complete self-test within 9.9 s ON AN OTHERWISE IDLE BUS (that is, the node never has to wait for the bus), then it will definitely complete self-test in 10 s regardless of the traffic on the bus. Similarly, if the node can complete self-test in 220 ms ON AN OTHERWISE IDLE BUS, it can complete self-test in 250 ms regardless of the bus configuration.

A node designer need only make sure that the node completes self-test on an idle bus in less than 9.9 s and need not worry how long it MIGHT take on a non-idle bus.

AN4.2.1.2 Calculation of Self-Test Duration - The self-test time criterion for an individual node on a fully populated VAXBI bus (that is, with 16 transaction-generating nodes) will now be derived. It will be shown that: if each individual node completes its normal self-test within 9.9 s on a VAXBI bus on which it is the only transaction-generating node, then all nodes will complete normal self-test within 10.0 s on a fully populated VAXBI bus.

Because intranode transactions are required to be of longword length, the maximum number of cycles that a single transaction can take during self-test is k + 4, where k is the maximum number of stall cycles. These k + 4 cycles consist of an arbitration cycle, a command/address

digital™

cycle, an imbedded arbitration cycle, k STALL cycles, and a data cycle. Because nodes are limited to an average of 4 STALLs per transaction (Section 11.1.6), the maximum duration of a transaction during self-test is 8 VAXBI cycles.

During normal self-test, a node may perform up to 2048 VAXBI transactions consuming a total time of:

(8 cycles/transaction)(2048 transactions/node)(200 ns/cycle)
= 3.2768 ms/node

Let's assume that a node (say node W) completes its normal self-test in Tidl ms on an otherwise idle bus. Tidl includes both the time used to perform VAXBI transactions and the time used for performing internal testing that does not require use of the VAXBI bus.

Now consider the case of node W on a fully populated bus with 16 nodes, all of which may generate VAXBI transactions during self-test. In the worst case, each transaction node W issues may have to wait for each of 15 other nodes to complete one transaction before node W can issue its transaction. The delay of the transactions performed by the 15 other nodes when added to Tidl is the total self-test time of node W. So that all nodes can complete self-test within the 10-second time limit, the following constraint is enforced on Tidl:

Tidl + (15 nodes)(3.2768 ms/node) <= 10000 ms

This can be solved for Tidl:

Tidl = 10 s - (15 nodes)(3.2768 ms/node)(1 s/1000 ms)
= 10 s - 0.049152 s
= 9.950848 s

Thus, if node W completes its normal self-test on an otherwise idle VAXBI bus within 9.90 s (we arbitrarily choose a number a bit smaller than the calculated maximum), then node W will complete normal self-test on a fully populated VAXBI within the 10-second limit. Since node W completes self-test later than any other node in the fully populated system, it follows that all nodes complete within 10 s.

AN4.2.2 Fast Self-Test

The purpose of the 250-millisecond limit is to permit rapid recovery from transient power outages for real-time applications. Because fast self-test gives a very abbreviated self-test, its use is not recommended for typical applications such as time-sharing.

All nodes that conform to the 10-second limit are also required to

digital™

conform to the 250-millisecond limit when BI STF L is asserted. Although the primary processor is not required to conform to either the 250-millisecond or the 10-second time limit, it is recommended that the primary processor conform to the 250-millisecond limit when BI STF L is asserted.

Nodes should test as much as possible within the given time constraints when BI STF L is asserted.

The fast self-test time criterion for an individual node on a fully populated VAXBI bus will now be derived. This derivation proceeds like that used to derive the normal self-test time criterion in Section AN 4.2.1. The only differences are that in this case:

1. Each node is only allowed 512 VAXBI transactions (compare 2048 for normal self-test), and

2. 250 ms (fast self-test time limit) is the number subtracted from, rather than 10 s (normal self-test time limit).

The worst-case additional delay of node W's transaction per node due to waiting for the bus is:

(8 cycles/transaction)(512 transactions/node)(200 ns/cycle)
= 819.2 us/node

To be certain that all 16 nodes complete fast self-test within the 250-millisecond time limit, each individual node must be able to complete fast self-test* on an otherwise idle VAXBI bus within:

250 ms - (15 nodes)(819.2 us/node)(1 ms/1000 us)
= 250 ms - 12.288 ms
= 237.712 ms

Thus, if each individual node completes its normal self-test on an otherwise idle VAXBI bus within 220 ms (an arbitrary number less than the allowed maximum), then all nodes can complete within the 250-millisecond limit.

A VAXBI-based system is expected to complete self-test within 250 ms after a transient power outage only if all the following are true:

- BI STF L is asserted.

- All nodes pass self-test. Some self-test failures cause the self-test time interval to exceed 250 ms.

- 5VBB, the battery-backed-up supply for memory, must have been

---

*The extent of self-test is measured from the deassertion of BI DC LO L.

digital™

maintained during the power outage. If the battery runs down or was not installed, memory self-test will exceed 250 ms.

- The primary processor quickly recognizes that all nodes have completed self-test. The limiting factor in system recovery time may not be VAXBI node self-test time but instead processor recovery time.*

AN4.3 EXTENDED SELF-TEST

Some VAXBI nodes may not achieve the desired level of test coverage within the 10-second limit. For example, a node that controls disks may require disk transfers for an adequate level of self-test coverage. Since disks generally do not spin up within 10 s, a strict interpretation of the 10-second limit precludes adequate test coverage. The recommended solution is to divide self-test: module self-test should verify the operation of the modules that comprise this node and must complete within the 10-second limit; device testing can subsequently complete without regard to the 10-second limit.

If a module passes self-test, the Broke bit must be reset, the BI BAD L line must be deasserted, and the yellow LEDs must be lit before extended self-test begins. If the extended self-test indicates that the node is broken, the BI BAD L line must be asserted and the LEDs turned off. The state of the BAD line and the LEDs must not be changed until the next time that self-test is initiated.

Whether a given device fault is construed as a node fault depends on the node and whether the faulty device is mandatory or optional.

If extended self-test is used, some higher level protocol must be devised to allow software to determine the state of completion and the result of extended self-test. Such a protocol is node-specific. If extended self-test is used to test multiple external devices, it is desirable to report individual device faults through both machine-readable (CSR bit) and visible (LED) indicators.

AN4.4 VAXBI TRANSACTION DATA PATTERNS

As part of its self-test, a node should perform VAXBI transactions to verify the correct functioning of the node's VAXBI transceivers.

While the optimal set of data patterns and commands to be used is node-specific, it is recommended that each node use all of the D and I

---

*The KA820 processor recovers quickly, but the KA88 does not.

digital™

lines. For example, WMCI transactions can be used to access General Purpose Register 0 (bb + F0):

- WMCI of FFFF FFFF hex, with mask = 0001 binary, then READ the GPR. The data read back should be 0000 00FF.
- WMCI of FFFF FF00, with mask = 0010, then read the GPR. The data read back should be 0000 FFFF.
- WMCI of FFFF 0000, with mask = 0100, then read the GPR. The data read back should be 00FF FFFF.
- WMCI of FF00 0000, with mask = 1000, then read the GPR. The data read back should be FFFF FFFF.
- WMCI of 0000 0000, with mask = 1111, then read the GPR. The data read back should be 0000 0000.

AN4.5 MULTIMODULE NODES

All VAXBI modules are required to have two yellow LEDs to indicate the successful completion of self-test. A node that consists of multiple modules has two strategies for controlling these LEDs:

- If self-test does not attempt to isolate a fault to a VAXBI module, then all of the yellow LEDs on all the VAXBI modules that are part of that node should be in the same state. If self-test fails, all LEDs should remain off. If self-test passes, all LEDs should be turned on at the same time.
- If self-test does isolate a fault to a single VAXBI module, the LEDs on the failing module should remain off, and the LEDs on the other modules of that node should be turned on.

  If self-test shows that a VAXBI module is fault-free, the LEDs on that module should be turned on. If self-test finds a fault but cannot isolate the fault to a single VAXBI module, then the LEDs on all the modules that could have the fault should remain off.

Generally, if any module of a node fails self-test, the node's Broke bit should remain set, and the node should continue to assert the BI BAD L line. If some of the node's modules are mandatory and some are optional, the node should be declared broken if any mandatory module is missing or if any module that is present has a fault.

The BI BAD L line and the yellow LEDs on VAXBI modules are intended to allow the use of a simple first-pass maintenance strategy. If the BI BAD L line is asserted when self-test completes, and if the yellow

digital™

LEDs are out on some modules, then those modules should be replaced.

AN4.6 DETERMINING THE RESULTS OF SELF-TEST

In most systems the primary processor determines the results of self-test at other nodes on the VAXBI bus. The following procedures are acceptable methods of determining the results of node self-tests.

AN4.6.1 Using the Broke Bits and the BI BAD L line

When the primary processor has completed its own self-test, it can begin probing the other nodes to examine their Broke bits. This probing session continues until all responding nodes have passed self-test (as indicated by reset Broke bits) or until the maximum self-test interval is over. At this time the processor examines the BI BAD L line to determine if any nodes are present but not responding, due to BIICs that failed to pass their own self-test (recall that a BIIC that fails its internal self-test disables its BI drivers).

With this method, assuming no hardware faults, the primary processor can restart system software as soon as all nodes have passed self-test and it has determined that the BI BAD L line is deasserted.

AN4.6.2 Using Only the BI BAD L Line

In this method the processor does not poll all the nodes to determine the state of their Broke bits but instead waits for the end of the maximum self-test interval, at which time all nodes should have completed their self-tests. The processor then samples the BI BAD L line. The BI BAD L line should not be sampled until the self-test interval is over. This avoids the possibility of errors due to transmission line glitches that occur as nodes independently complete self-test and disable their BI BAD L driver. (See Appendix B for a description of the transmission line glitch phenomenon.) With this method, the processor must examine the state of the BI STF L line to determine the maximum self-test time period.

This restart method is slower than the Broke/BAD method discussed above, since the processor must wait the maximum self-test time, even if all nodes in the system complete their self-tests in a much shorter time.

digital™

AN4.6.3 Using the Broke Bits and Stored System Configuration Information

In this method the processor polls the nodes for the state of their Broke bits but does not examine the BI BAD L line. Instead the processor uses an implementation-specific method of storing the expected system configuration information (most likely in an EEPROM or equivalent). The processor then compares this information against the polled Broke bit data to determine if any nodes that should be present are not responding.

AN4.6.4 Using Only the Broke Bits

Processors may choose to use only the Broke bits, starting the system after the last responding node resets its Broke bit. With this method there is some risk that the system will be restarted while a node is present but not responding, but as mentioned this occurs only if a BIIC fails its own self-test.

digital™

NOTE 5

USE OF RETRY RESPONSE CODE

This application note describes use of the RETRY response code and how to avoid or deal with extraneous retry timeouts.

Dual round-robin arbitration mode ensures that no pattern of traffic can continually prevent a node from gaining access to the VAXBI bus. However, after becoming bus master on the VAXBI and initiating a transaction, a node can receive the RETRY response code, retry the transaction, and again receive a RETRY. Such a pattern can effectively lock out a node. To break this endless loop, the BIIC can issue the Retry Timeout (RTO) EV code indicating a retry timeout. A retry timeout is considered an error condition.

A node can encounter a retry timeout from its BIIC even when all nodes seem to be properly designed and no node is malfunctioning. Because such timeouts can lead to extraneous and perplexing system outages, the RTOEVEN bit should be set only in certain circumstances. This note discusses considerations that govern the setting of this bit.

Section AN5.1 discusses uses of the RETRY response code, Section AN5.2 describes scenarios that lead to a retry timeout, and Section AN5.3 presents strategies to avoid such situations. Section AN5.4 offers recommendations for using the suggested strategies with various kinds of nodes.

AN5.1 WAYS IN WHICH RETRY IS USED

The RETRY response code can be used in three ways:

- As an alternative to a lengthy sequence of STALL cycles
- As a means of preempting the VAXBI bus for another transaction
- In the implementation of interlocks

digital™

AN5.1.1 Using RETRY as an Alternative to STALL

A VAXBI node targeted for a transaction can respond with a STALL or a RETRY when its resources are temporarily unavailable, perhaps because it is servicing a previous transaction. The use of RETRY may be preferable to STALL because RETRY releases the bus. If the node cannot service the transaction for some time, the use of RETRY can increase bus utilization and decrease bus latency.

AN5.1.2 Preempting the VAXBI Bus for Another Transaction

A node may need to issue a VAXBI transaction before it can accept one. If a node in this state is targeted for a transaction, the node must return RETRY so that the master releases the bus and this node can issue its own transaction.

A compelling example of such a situation occurs with bus adapters in which the target bus is a nonpended bus. On a nonpended bus a transaction includes a request and the response, and the bus is not released until the response is made. The VAXBI and UNIBUS are nonpended buses. A pended bus is a bus on which a node accepts a request in one transaction and sends the response to that request in a separate transaction. An example of a pended bus is the SBI of the VAX-11/780.

If an adapter connects some nonpended bus (NPB, for example) to the VAXBI bus, the following situation could happen. Node N on the NPB bus starts a read to the adapter at about the same time that VAXBI node B issues a VAXBI READ transaction to the adapter. Node N cannot complete its read until B completes its READ and releases the VAXBI bus, which B cannot do until N completes its read and releases the NPB bus. To break the deadlock, the adapter must send RETRY on the VAXBI bus, which causes node B to release the VAXBI bus. (The UNIBUS adapter uses this strategy to avoid deadlocks.)

AN5.1.3 Implementing Interlocks

A VAXBI node is required to respond with a RETRY to an IRCI transaction that attempts to access a location that is locked from a previous IRCI transaction. If the node responded with STALL, the bus would be tied up so that the UWMCI transaction required to unlock the location could not be issued.

digital™

In this situation it is unlikely that the RETRYs issued will produce a retry timeout. Since the interval allowed between an IRCI and a UWMCI is quite small (about 100 microseconds), this situation probably would not occur many times in succession.

AN5.2 SCENARIOS LEADING TO A RETRY TIMEOUT

This section presents scenarios in which the use of RETRY results in a retry timeout even though each component of the configuration appears to be properly designed.

AN5.2.1 When RETRY Is Used as an Alternative to STALL

A retry timeout can occur when a node returns RETRY because its resources are temporarily not available. In the configuration shown in Figure AN5-1, when the DMA node writes to the buffered write node, the buffered write node stores the data in a buffer and then releases the VAXBI bus. If the buffered write node is targeted for another VAXBI transaction while it is processing the data, the node responds with a RETRY.*

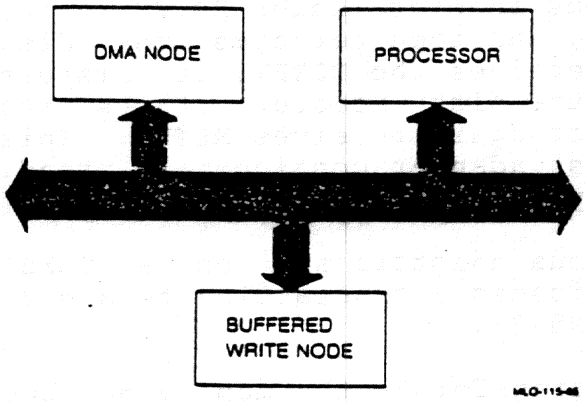


Figure AN5-1: Sample Configuration

---

*A node that uses the VAXBI 78733 (BCI3) chip would behave in this way.

digital™

Suppose the DMA node is performing a block transfer (a sequence of octaword writes) to the buffered write node. While the buffered write node is processing the data, the processor becomes bus master, issues a WRITE transaction to the buffered write node, and gets a RETRY. Meanwhile, the DMA node is ready for another WRITE; it becomes bus master while the buffered write node finishes processing the previous data. The DMA node completes its WRITE transaction, and the processor reattempts its WRITE transaction and again gets a RETRY. The cycle repeats. If the DMA node's block transfer is sufficiently long, the processor will encounter a retry timeout. A timeout occurs even though the processor regularly gets to be bus master. If other DMA devices are on the VAXBI bus, one block transfer can immediately follow another, and the processor can be locked out.

AN5.2.2 When the VAXBI Bus Is Released for Another Transaction

Figure AN5-2 shows a configuration in which a retry timeout occurs when RETRY is used by a nonpended bus adapter.

Suppose the disk adapter is about to accumulate several sectors' worth of data (to write to the disk), and this is done with octaword READs to transfer the data from the memory on the VAXBI bus. Just as the disk adapter issues its read on the nonpended bus, the processor issues a READ to read a CSR on the disk adapter (or any other CSR on the nonpended bus). The nonpended bus adapter issues RETRY to the processor's READ to resolve the deadlock, and then performs the disk adapter's read. When the processor receives the RETRY, it reissues the transaction. But just at this time the disk adapter starts its second octaword READ, so the processor again receives RETRY. This pattern can repeat for as long as the disk adapter continues to read. The processor eventually receives a retry timeout.

A deadlock can occur when two nonpended bus adapters are on a VAXBI bus, and a node on each nonpended bus issues a transaction to a node on the other nonpended bus (see Figure AN5-3).

If the DMA node on bus 1 starts a block transfer to the memory on bus 2 while the DMA node on bus 2 is performing a block transfer to the memory on bus 1, each bus adapter may send a RETRY to the other. The deadlock will not be broken unless some timeout happens or one of the buses implements an operation equivalent to the RETRY.

digital™

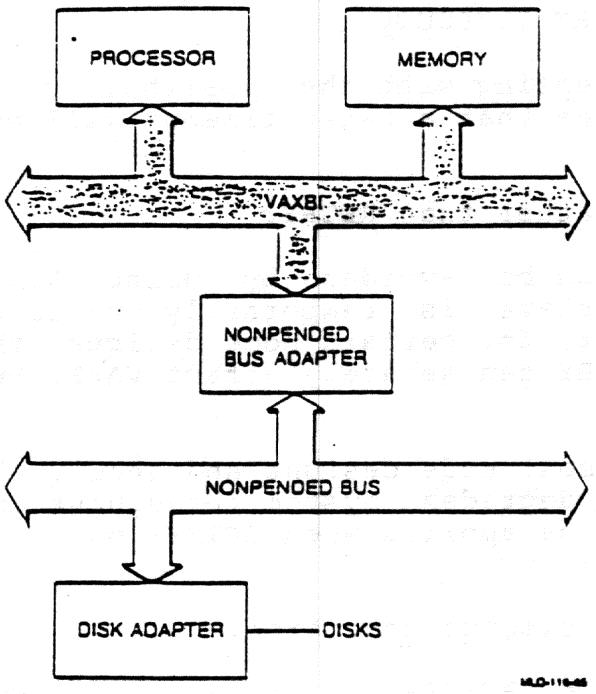


Figure AN5-2: Configuration with a Nonpended Bus Adapter

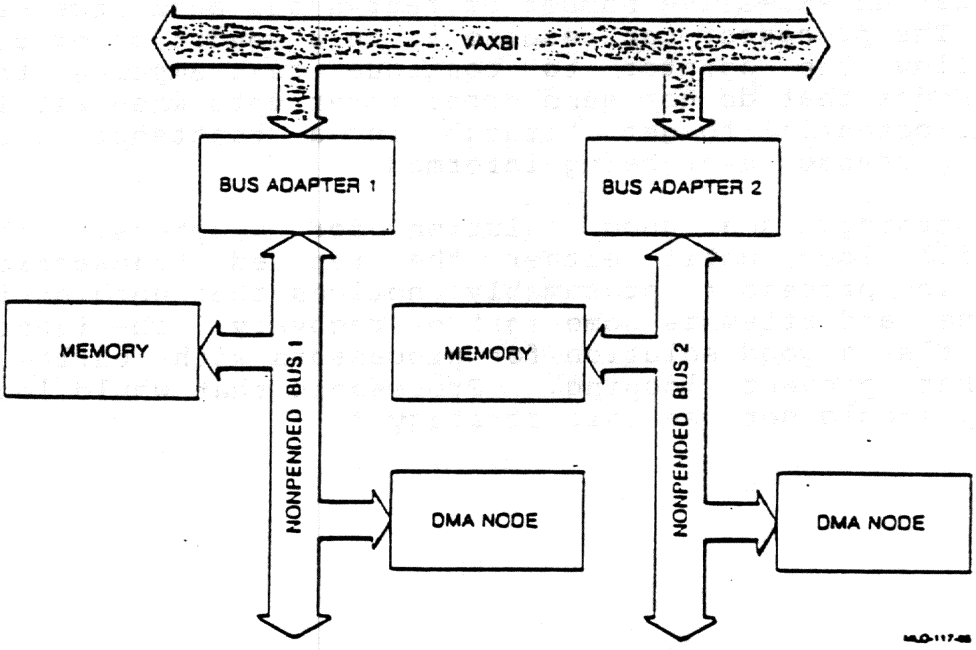


Figure AN5-3: Configuration with Two Nonpended Buses

digital™

AN5.3 STRATEGIES FOR DEALING WITH RETRY TIMEOUTS

This section presents strategies for dealing with the possibility of retry timeouts. Nothing can guarantee that a retry timeout will not occur.

- "STALL" Strategy: Use STALL rather than RETRY.

  One source of retry timeouts can be avoided by using STALL rather than RETRY when a slave is temporarily unable to service a transaction. However, for certain node designs the use of STALL rather than RETRY can severely affect VAXBI bus throughput and latency.

  This strategy is suitable for most node designs and should be adopted unless it severely degrades system throughput and latency. This strategy cannot be applied when RETRY must be used to release the VAXBI bus.

- "Ignore" Strategy: Ignore the timeout condition.

  If the RTOEVEN bit in the BIIC's BCICSR is reset, the RTO event code will not be output and the node will reattempt the transaction until it succeeds. The RTO bit in the BIIC's BER will still be set when the 4096th reattempt is made and if the HEIE bit in the VAXBICSR is set, an error interrupt will be automatically generated by the BIIC. This will notify the processor that an excessive number of retrys has been received by a node. The processor can respond to this condition or can choose to allow the adapter to continue reattempting the transfer. Nodes that do not send error interrupts when RTO is set have the potential to get "stuck" in a reattempt loop without the processor ever being informed.

  The ignore strategy is a good solution for adapters. The adapter will loop until either the retried transaction succeeds or the processor (presumably) notices that nothing is getting done and attempts some sort of recovery. The ignore strategy is also a good solution for processors with built-in timeouts that prevent looping. Processors that would loop indefinitely should not use this strategy.*

---

*The KA88 is an example of a processor with a built-in timeout. The KA820 is an example of a processor that would loop indefinitely.

digital™

- "Window" Strategy: Reserve time windows for retried transactions.

  With this approach, a bus adapter that has output a RETRY response on the VAXBI bus reserves the other bus for a fixed period when that bus becomes available, so that the reissued transaction has a good chance of succeeding. This strategy decreases the likelihood that the node will receive RETRY after the other bus becomes available. For example, the UNIBUS adapter using this strategy could reduce the likelihood of a retry timeout without a severe impact on performance.

  This approach, however, does not eliminate the possibility of extraneous retry timeouts, and it cannot be easily adapted for the other uses of RETRY. The window strategy should be used for processors that cannot ignore a retry timeout (unless the software strategy described below is used). This strategy should be used for nonpended bus adapters.

- "Software" Strategy: Resume software operation after a retry timeout.

  With this approach, a retry timeout generates a machine check condition. The software exception handler in effect extends the retry timeout count by counting in software. The count is incremented if the last retry timeout exception occurred within say 10 ms and is reset to zero otherwise. Unless the software count has reached a (programmable) upper limit, the exception handler would then perform a Return from Interrupt instruction, resuming execution (and retrying the transaction). When the software count reaches the limit, an error condition is signaled.

  The advantage of this approach is that it extends the timeout count to suit the configuration. However, this strategy applies only to processors, and the implementation is complex. In systems in which the VAXBI bus is an I/O bus, the bus adapter that connects the VAXBI bus to a processor typically does not have the capability to cause the processor to machine check, so this approach cannot be implemented.*

---

*The VAX 8800 is a system in which the software strategy cannot be implemented. The VAX 8200 is a system in which this strategy can be implemented.

digital™

AN5.4 RECOMMENDATIONS

Of the strategies outlined above, the ignore strategy should be used by adapters and some processors to ensure that their systems will not crash due to a retry timeout. With the ignore strategy a system will not crash, but the bus bandwidth and latency may be adversely affected. The STALL and window strategies can be used to limit the adverse effects associated with the ignore strategy. Use of the STALL and window strategies reduces the probability of a retry timeout in systems with processors that might loop indefinitely should they ignore a retry timeout. Use of the software strategy is the only solution for these systems.

In summary, all VAXBI nodes can use the STALL strategy. Bus adapters should use the window strategy. All nodes except processor nodes that might loop indefinitely can use the ignore strategy. Only processors can use the software strategy.

digital™

NOTE 6

USE OF THE CLOCK RECEIVER

This application note describes the use of the VAXBI clock receiver. It also presents a suggested method of generating a family of clock waveforms for use by VAXBI node logic.

The VAXBI clock receiver is a 16-pin integrated circuit required of all nodes to provide a node's connection to the differential clock lines BI TIME +/- and BI PHASE +/-. The part combines ECL and FTTL technology circuitry to provide a low skew set of four clock signals that provide the synchronous clock source for the BIIC as well as for the logic external to the VAXBI Corner. The functionality of the part can be thought of as a four-section single rail (+5V only) "backward-ECL" to TTL translator. Here "backward" refers to the reverse power connection nature of both the VAXBI clock driver and VAXBI clock receiver designs which deviates from the typical ECL power supply connections (where Vcc = GND and Vee = -5.2 volts.)

The VAXBI node that is the clock source contains a VAXBI clock driver that provides the differential TIME +/- and PHASE +/- signals received by the VAXBI clock receivers.

The four outputs of the VAXBI clock receiver are available at the boundary of the VAXBI Corner, defined as the following TTL compatible signals:

- o TIME H
- o TIME L
- o PHASE H
- o PHASE L

The performance of these outputs, when receiving the differential bus clocks in any configuration, meets the requirements of the BIIC specification for BIIC clock waveforms. Node designs should depend on no better performance than that indicated in the following table and figures.

digital™

USE OF THE CLOCK RECEIVER

This application note and the BIIC AC timing specifications in Chapter 20, Section 20.3, provide all the information needed for the design of a clock system for a VAXBI node. The DC specifications for the VAXBI clock receiver are in Chapter 23, Section 23.1.

Table AN6-1 specifies the values of the parameters as defined in Figures AN6-1, AN6-2, and AN6-3. The table includes all effects of VAXBI clock driver and bus etch skews and propagation delays, as well as VAXBI clock receiver skews and propagation delays on the receiver's output waveforms. Unless otherwise specified, all specifications are at Ta = 0 to 70 degrees C and Vcc = 4.75 to 5.25 volts. The total capacitance load on any output is < 100 pF. Time is in nanoseconds.

digital™

Table AN6-1: AC Clock System Characteristics

Symbol	Parameter	Min.	Max.	Remarks
Tcy1	TIME H or TIME L period	49.995	50.005	Note 1
Tcp1	TIME H or TIME L pulse width	20	30	-
Tcy2	PHASE H or PHASE L period	199.98	200.02	Note 1
Tcp2	PHASE H or PHASE L pulse width	95	105	-
Tps1	PHASE H setup to TIME H	16.9 - Cd1*SF	-	Notes 2, 3
Tps2	PHASE H setup to TIME L	16.9 - Cd2*SF	-	Notes 2, 3
Tps3	PHASE L setup to TIME H	16.9 - Cd3*SF	-	Notes 2, 3
Tps4	PHASE L setup to TIME L	16.9 - Cd4*SF	-	Notes 2, 3
Tph1	PHASE H hold from TIME H	16.9	- - Cd1*SF	Notes 2, 3
Tph2	PHASE H hold from TIME L	16.9	- - Cd2*SF	Notes 2, 3
Tph3	PHASE L hold from TIME H	16.9	- - Cd3*SF	Notes 2, 3
Tph4	PHASE L hold from TIME L	16.9	- - Cd4*SF	Notes 2, 3
Trt	10% to 90% rise time	-	6	-
Tft	90% to 10% fall time	-	6	-
Tskw1	Skew between TIME H and TIME L	-	5.1	Notes 2, 4 + Cd5*SF
Tskw2	Skew between PHASE H and PHASE L	-	5.1	Notes 2, 4 + Cd6*SF

digital™

NOTES

1. These times take into account the .01 percent frequency tolerance of the 40 MHz crystal oscillator. Most designs can assume "perfect" 20 MHz and 5 MHz frequencies for the TIME and PHASE square waves respectively.

2. The base specification is for equally loaded outputs of the clock receiver. If the outputs are not equally loaded, then the parameter increases by an amount equal to the product of an SF (scale factor which is equal to an additional 40 picoseconds per picofarad) and Cdn (the capacitance loading difference between the two outputs). The differential load variables are defined below:

Parameter	Definition
Cd1	Cphaseh - Ctimeh
Cd2	Cphaseh - Ctimel
Cd3	Cphasel - Ctimeh
Cd4	Cphasel - Ctimel
Cd5	Ctimeh - Ctimel
Cd6	Cphaseh - Cphase

Due to the VAXBI Corner layout, there is an inherent difference in loading on the various receiver outputs. This loading is discussed in Section AN6.1.

3. These setup and hold times refer to the relationship between two outputs (in more common use, setup and hold times refer to inputs). For example, the Tps1 specification guarantees that PHASE H will become valid a minimum of Tps1 nanoseconds prior to the rising edge of TIME H. Similarly, the Tph1 specification guarantees that PHASE H will remain valid for Tph1 nanoseconds following the rising edge of TIME H.

4. This skew is a maximum absolute value and, as shown in Figure AN6-3, can occur in either "direction."

digital™

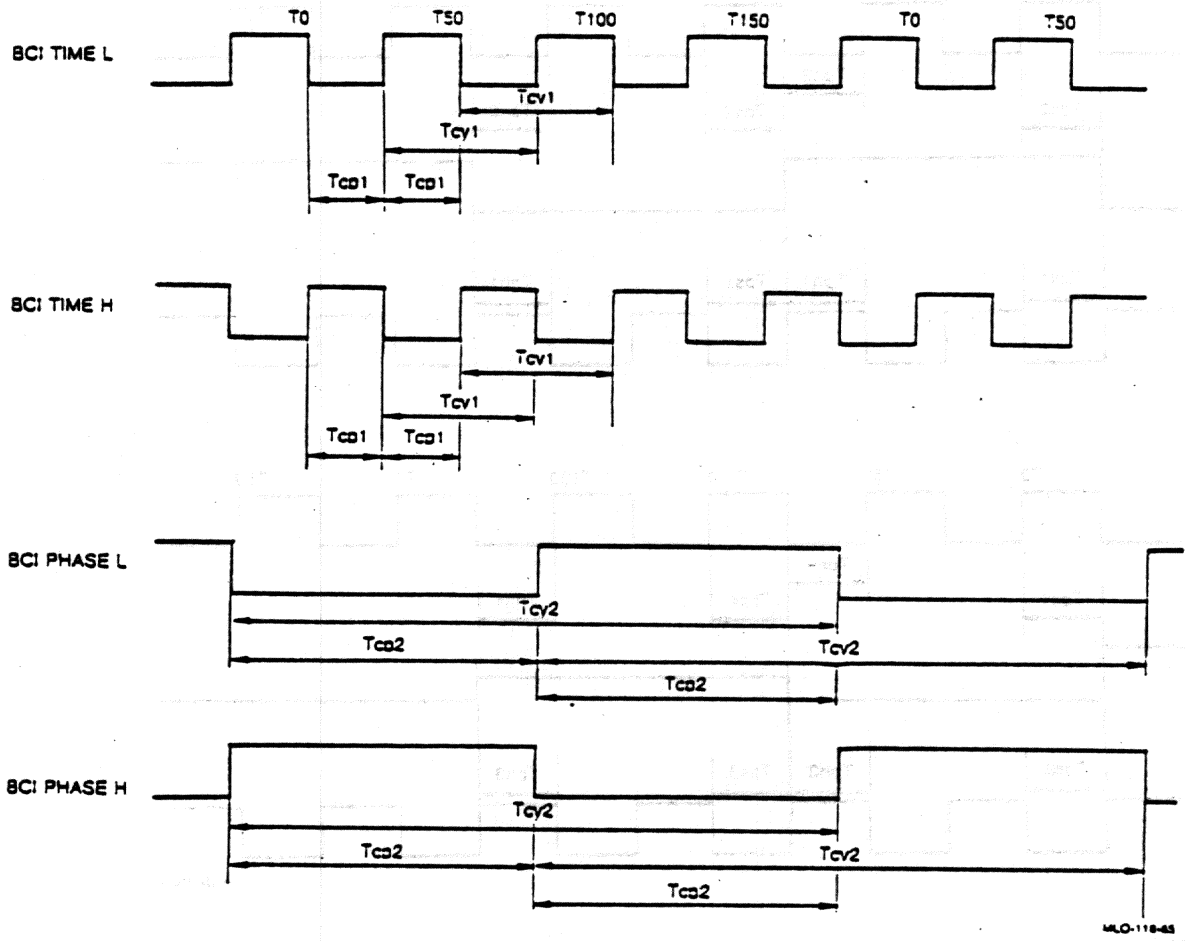


Figure AN6-1: TIME and PHASE Waveforms

digital™

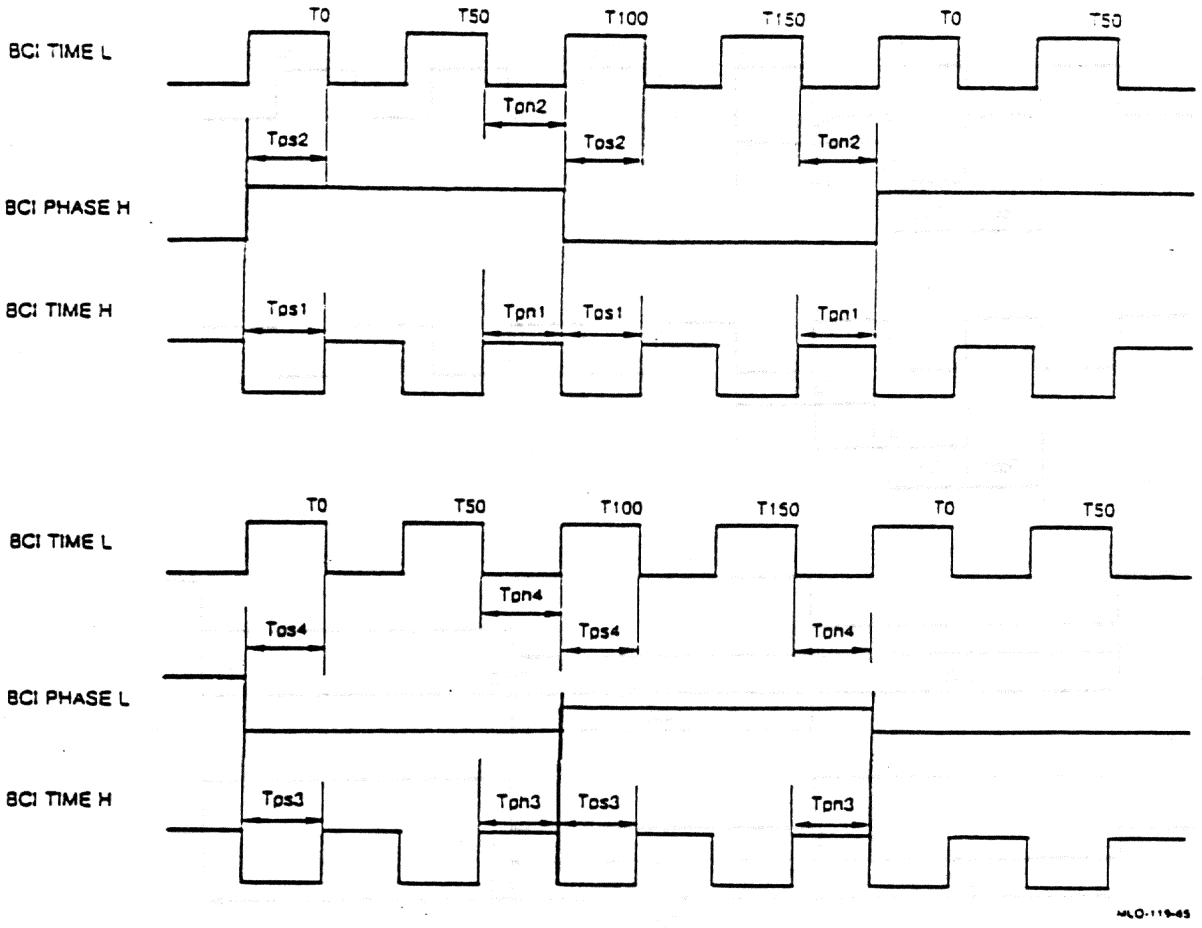


Figure AN6-2: Timing Relationships Between TIME and PHASE Signals

digital™

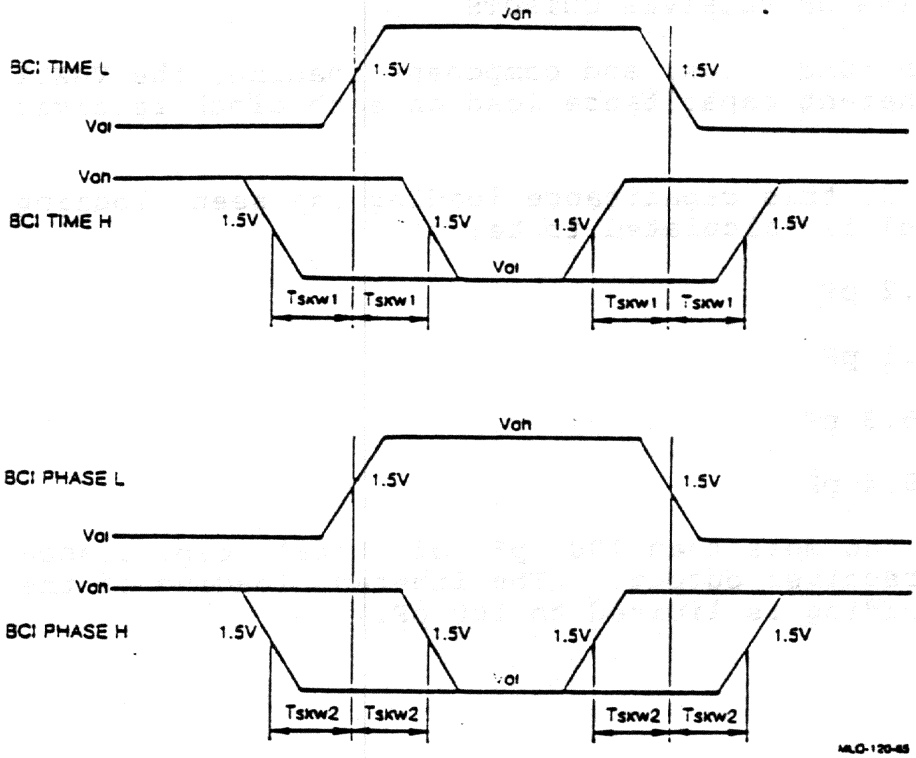


Figure AN6-3: Skew Between True and Complement Outputs of TIME and PHASE

digital™

AN6.1 VAXBI CORNER LOADING OF RECEIVER OUTPUTS

Due to the effect of etch runs, vias, and component loading, the VAXBI Corner presents an inherent capacitance load on each clock receiver output.

For the VAXBI Corner Rev 3, this capacitance loading (as seen looking into the edge coordinates) is calculated to be:

- BCI TIME H -- 17.2 pF
- BCI TIME L -- 18.3 pF
- BCI PHASE H -- 15.8 pF
- BCI PHASE L -- 18.4 pF

For proper bus operation, no more than 100 pF of total capacitance load may appear on any receiver output. (The inherent loading of the VAXBI Corner plus user loading is limited to 100 pF.)

digital™

AN6.2 SAMPLE SKEW CALCULATIONS

As an example, assume a VAXBI node design presents the following loading:

CAPACITANCE LOADING

Signal	VAXBI Corner Loading	User Interface Loading	Total
BCI TIME H	17.2	32.8	50.0 pF
BCI TIME L	18.3	41.7	60.0 pF
BCI PHASE H	15.8	34.2	50.0 pF
BCI PHASE L	18.4	41.6	60.0 pF

Refer to the VAXBI Designer's Notebook, Chapter 3, for the recommended parameters to use for gate, etch, and via loading.

Using the parameters specified above, the setup and hold times between BCI PHASE L and BCI TIME H in this particular design are:

```
Tps3 = 16.9 - (SF*Cd3)
     = 16.9 ns - (.04 ns/pF) * (60.0 pF - 50.0 pF)
     = 16.5 ns

Tph3 = 16.9 - (SF*Cd3)
     = 16.9 ns - (.04 ns/pF) * (60.0 pF - 50.0 pF)
     = 16.5 ns
```

In addition, the skew between TIME H and TIME L is:

```
Tskw1 = 5.1 + (.04 ns/pF) * (cd5)
      = 5.1 + (.04 ns/pF) * (60.0 pF - 50.0 pF)
      = 5.5 ns
```


digital™

AN6.3 SAMPLE VAXBI NODE CLOCK DESIGN

This section presents a VAXBI node clock design that provides 50 nanosecond (nominal) clock pulses in both polarities every 25 ns (nominal). Two quad-D F/F packages in conjunction with a single 2-input NOR gate are used to generate these 16 clock signals. In the design shown in Figures AN6-4 and AN6-5, FTTL parts are used (74F02 and 74F175).

The timing diagram in Figure AN6-6 shows the relationship between the BCI TIME L and BCI TIME H signals that is used to clock the 74F175s. The reference edge for all timing calculations is the falling edge of BCI TIME L, since this is the TIME signal that the BIIC uses for reference. The deasserting edge of BCI TIME L may be skewed by up to (Tcp1 max. - Tcp1 min.)/2 relative to the nominal 25 ns after the asserting edge of BCI TIME L. The maximum skew between opposite edges of BCI TIME L and BCI TIME H is given by the Tskw1 specification.

Figure AN6-7 shows sample output from a spreadsheet program that calculates worst-case clock parameters. Eight parameters are user defined. The user defines the capacitance load on the four receiver outputs that is presented by the node design exclusive of the VAXBI Corner (the program adds in the proper value of VAXBI Corner capacitance for each output). The user can also specify the propagation delay characteristics of the D F/F that provides the user clocks (as released, the program uses the specifications for the 74F175).

The output from the program includes all the variations of setup and hold times for the TIME and PHASE signals. The node designer should verify that sufficent PHASE setup to TIME is provided for the D F/F. In this design the minimum setup time is 16.5 ns (Tps3 specification), which assures the 74F175 of at least 10 ns setup at the D input (the NOR delay is 6.5 ns maximum). The program also calculates the minimum spacing between edges of adjacent clock outputs. For example, the "Tnn-->Tnn+25" specification defines the minimum spacing between the rising edge of a clock output and the rising edge of the adjacent clock output that occurs nominally 25 ns later. These edge spacings can become quite small.

digital™

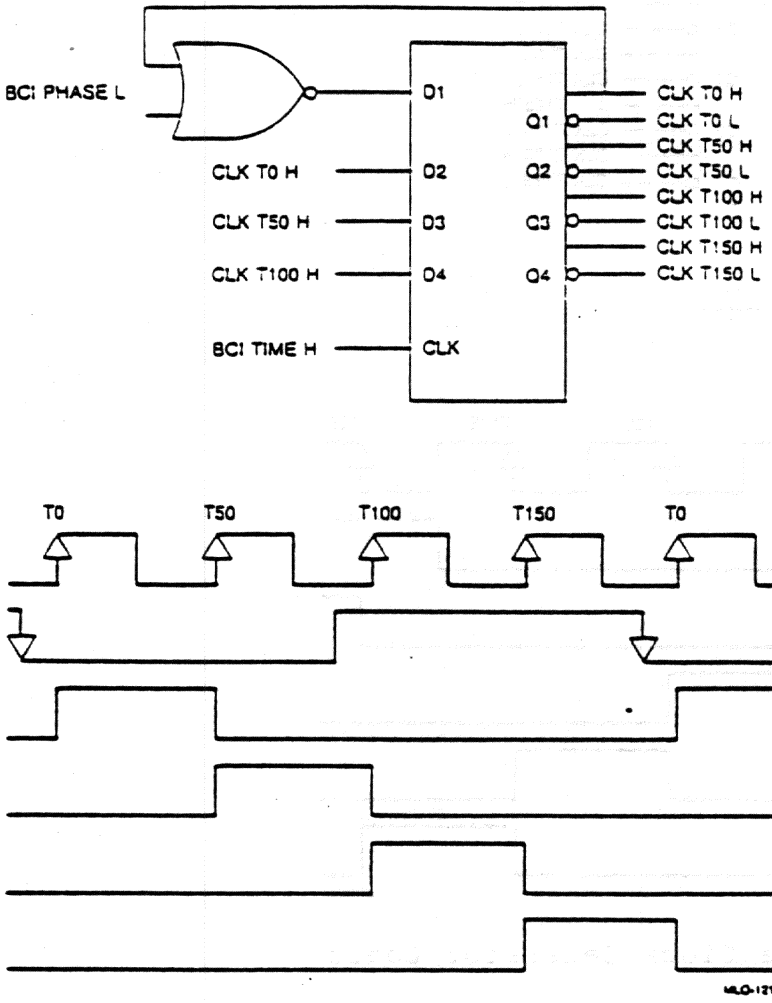


Figure AN6-4: Even Phase Clock Generator Logic

digital™

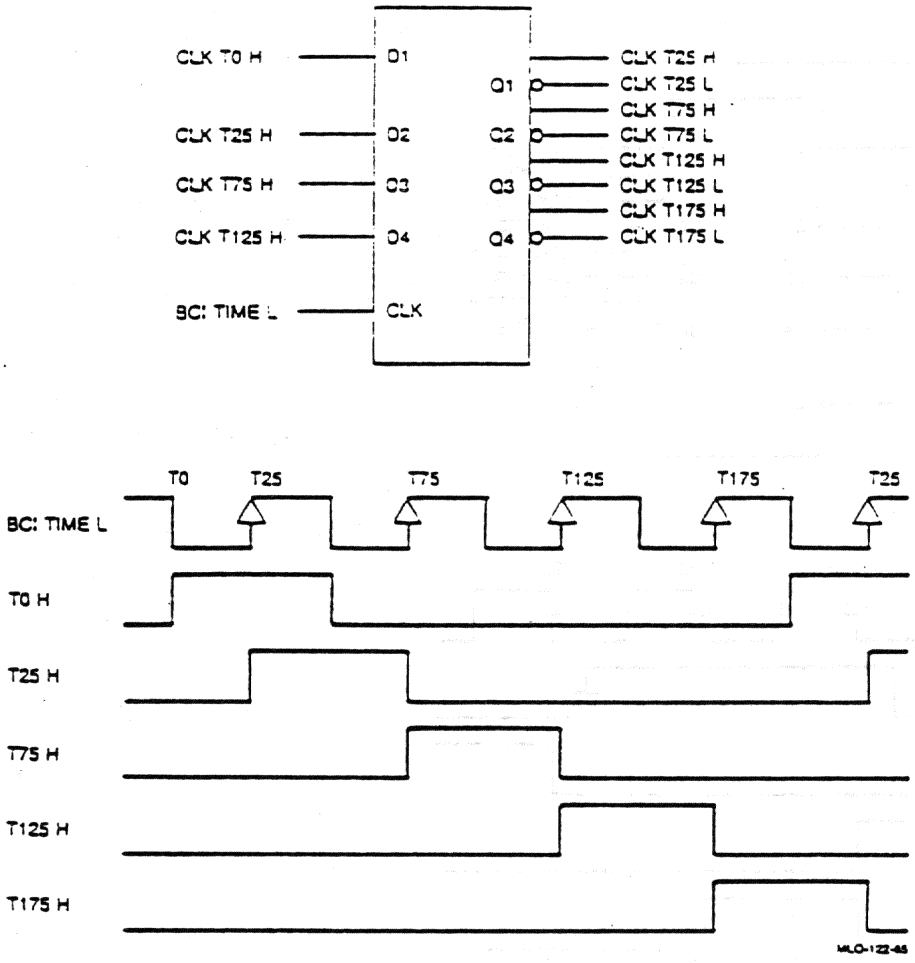


Figure AN6-5: Odd Phase Clock Generator Logic

digital™

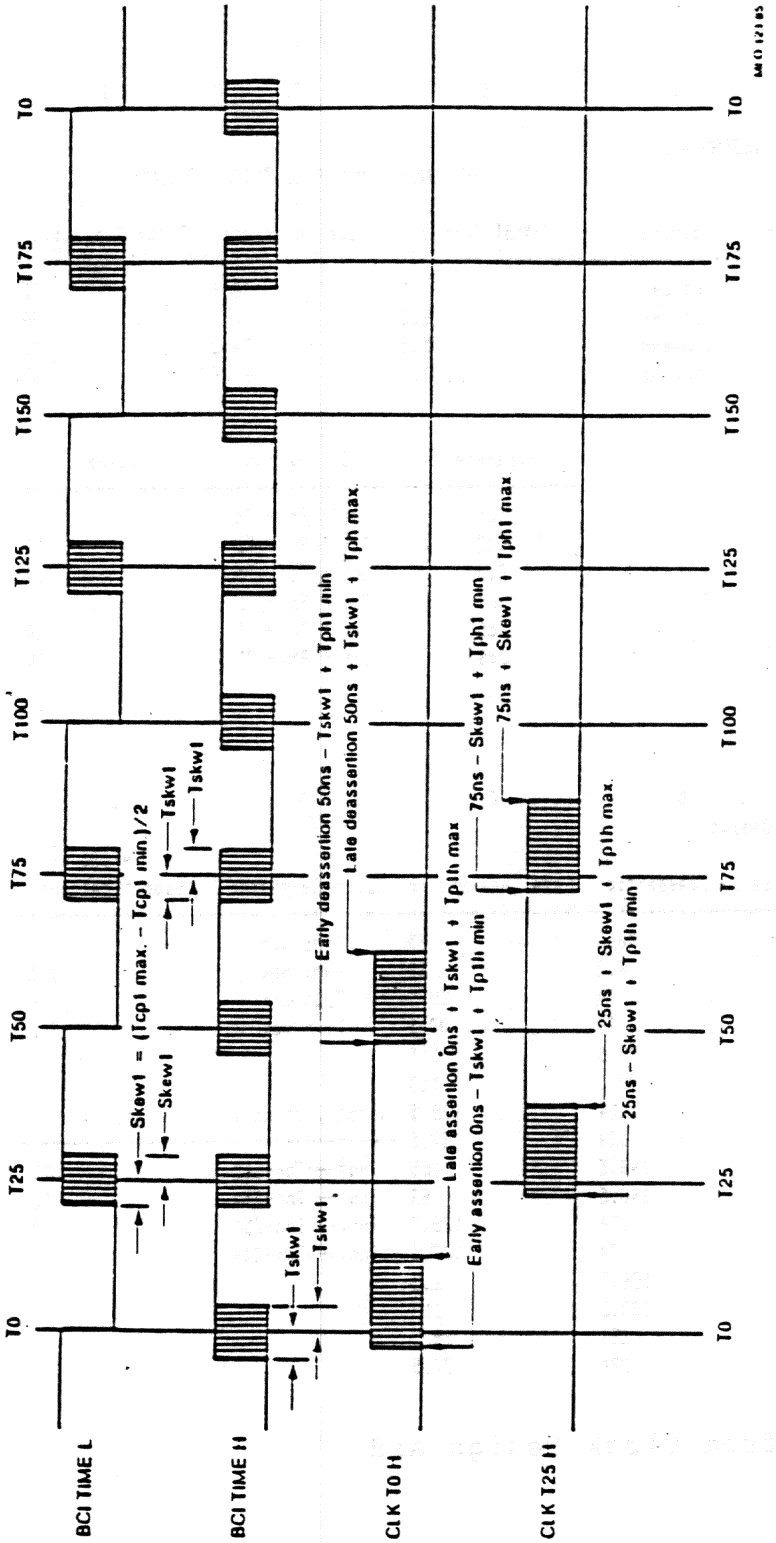


Figure AN6-6: VAXBI Node Clock Worst-Case Waveforms

digital™

PRIMARY WINDOW
Starting Data row: 1

A | B | C | D | E | F | G

VAXBI NODE CLOCK DESIGN AID

Symbol	Minimum Value	Maximum Value
Tcy1	49.995	50.005
Tcp1	20	30
Tcy2	199.98	200.02
Tcp2	95	105
Tps1	16.9	
Tps2	16.5	
Tps3	16.5	
Tps4	16.9	
Tpn1	16.9	
Tpn2	16.5	
Tpn3	16.5	
Tph4	16.9	
Tskw1		5.5
Tskw2		5.5
SF		.04

USER ONLY MODIFIES THIS COLUMN

Symbol	VAXBI Corner	User Interface	Total Cap. Load
Ctimen	17.2	32.8	50
Ctimei	19.3	41.7	60
Canasen	15.3	34.2	50
Canasei	18.4	41.6	60

Parameter	Definition	Value
Cd1	Cdiff Ph(->)Th	0
Cd2	Cdiff Ph(->)Tl	10
Cd3	Cdiff Pl(->)Th	10
cd4	Cdiff Pl(->)Tl	0
Cd5	Cdiff Th(->)Tl	10
Cd6	Cdiff Ph(->)Pl	10

PRIMARY WINDOW
Starting Data row: 26

A | B | C | D | E | F | G |

Sample Clock Design

Clock Phase	Early Assertion	Late Assertion	Early Deassertion	Late Deassertion
CLK T0 H	-1.5	13	48.5	63
CLK T0 L	-1.5	15	48.5	63
CLK T25 H	24	37.5	74	89.5
CLK T25 L	24	39.5	74	87.5
CLK T50 H	48.5	63	99.5	115
CLK T50 L	48.5	65	98.5	113
CLK T75 H	74	87.5	124	139.5
CLK T75 L	74	89.5	124	137.5
CLK T100 H	98.5	113	148.5	165
CLK T100 L	98.5	115	148.5	163
CLK T125 H	124	137.5	174	189.5
CLK T125 L	124	139.5	174	187.5
CLK T150 H	148.5	163	198.5	215
CLK T150 L	148.5	165	198.5	213
CLK T175 H	174	187.5	224	39.5
CLK T175 L	174	189.5	224	37.5

74F175

Clock Generator	Specifications
Tpn1 min	4
Tpn1 max	9.5
Tpth min	4
Tpth max	7.5

Worst Case Edge	Spacing
Tnn^-->Tnn+25^	11
Tnn^-->Tnn+25v	11
Tnnv-->Tnn+25^	9
Tnnv-->Tnn+25v	9

Figure AN6-7: Sample Output from Clock Design Aid

digital™

NOTE 7

BCI POWER SEQUENCE TIMING

This application note discusses the power sequence timing from the BCI viewpoint. The timing of the BCI AC LO L and BCI DC LO L signals differs from their VAXBI counterparts, and this application note provides information on the way they differ. The note also provides timing parameters that define the relationships between BCI AC LO L, BCI DC LO L, and BI RESET L.

Complete timing information on the BI AC LO L, BI DC LO L, and BI RESET L lines appears in Chapter 6, Initialization. The BI AC LO L and BI DC LO L lines are not used directly by the user interface but are buffered and output by the BIIC as BCI AC LO L and BCI DC LO L, respectively. BI RESET L, which is not buffered by the BIIC, is used directly by the user interface.

AN7.1 SIGNAL DESCRIPTIONS

AN7.1.1 BCI AC LO L Signal

The BIIC receives BI AC LO L and transmits the received state on the BCI AC LO L line two cycles later. The BCI AC LO L line is always driven synchronously. During the two-cycle delay the BIIC verifies that the received state was stable, and it will not change the state of BCI AC LO L unless the new BI AC LO L state was the same for both preceding cycles (this effectively filters one-cycle glitches of BI AC LO L).

AN7.1.2 BCI DC LO L Signal

The BIIC asserts BCI DC LO L asynchronously following the assertion of BI DC LO L. The maximum assertion delay is given by the Tdas spec in the BIIC specification.

digital™

The BIIC synchronously deasserts BCI DC LO L Tdde (see Table AN7-2) following a valid deassertion of BI DC LO L. A valid deassertion of BI DC LO L is detected if BI DC LO L remains stable and deasserted for two consecutive cycles.

When the Node Reset (NRST) bit is set, the BIIC drives BCI DC LO L without regard to BI DC LO L. The BIIC asserts BCI DC LO L for Tnrw (see Table AN7-2) following the writing of the NRST bit. When BCI DC LO L is deasserted, the BIIC begins its internal self-test (just as it does for a "real" power-down/power-up sequence).

digital™

AN7.2 VAXBI TIMING SPECIFICATIONS

Table AN7-1: Power Sequence Timing Specifications (From Chapter 6)

Symbol	Parameter	Min.	Max.	Unit	Note
Tbips1	BI AC LO L assertion width	5	-	us	-
Tbips2	BI DC LO L assertion delay from BI AC LO L assertion	4	50	ms	1
Tbips3	BI AC LO L deassertion delay from BI DC LO L deassertion	.105	30	ms	2
Tbips4	BI DC LO L assertion width	100	150	ms	2
Tbips5	RM's BI RESET L setup time to BI DC LO L deassertion	5	-	us	-
Tbips6	RM's BI RESET L setup time to RM's BI AC LO L deassertion	5	-	us	-
Tbips7	RM's BI RESET L hold time from RM's BI DC LO L deassertion	100	-	us	-
Tbips8	Duration of valid DC power following BI DC LO L assertion	5	-	us	-
Tbips9	BI DC LO L deassertion from valid DC power	70	150	ms	-
Tbips10	Valid clock signals to BI DC LO L deassertion	1	-	ms	-
Tbips11	Node's BI RESET L deassertion delay from RM's BI DC LO L assertion	0	10	ms	-
Tbips12	RM's BI RESET L assertion delay to RM's BI AC LO L assertion.	0	-	-	3
Tbips13	RM's BI AC LO L assertion delay from node's BI RESET L assertion	0	100	ms	-
Tbips14	RM's BI AC LO L deassertion delay from node's BI RESET L deassertion	0	150	ms	-
Tr, Tf	BI RESET L rise time, fall time (10% to 90%)	0	1	us	4

NOTES

1. With certain power supplies, during certain brownout power conditions, BI AC LO L may assert and later deassert without an assertion of BI DC LO L.

2. Maximum specification does not apply for a "real" power sequence case.

3. This specification means that the RM must assert BI RESET L upon the detection of a BI RESET L assertion by a node, at least by the time it asserts BI AC LO L.

4. The maximum time of 1 microsecond corresponds to a maximum capacitance load of 3000 pF. With present VAXBI card cages, terminators, and extension cables, the signal loading exclusive of BI RESET L cables is approximately 500 pF.

digital™

Table AN7-2: BIIC-Related Power Sequence Timing Specifications (From Section 20.3)

Symbol	Parameter	Min.	Max.	Unit	Remarks
Tdas	BCI DC LO L assertion delay from BI DC LO L assertion	0	150	ns	-
Tdde	BCI DC LO L deassertion delay from BI DC LO L deassertion	45	55	us	-
Tnrw	BCI DC LO L assertion width following the setting of the NRST bit	45	55	us	-

Table AN7-3: Calculated BCI Power Sequence Timing Specifications

Symbol	Parameter	Min.	Max.	Unit	Remarks
Tbcips1	BI AC LO L deassertion delay from BCI DC LO L deassertion *** CALCULATED PARAMETER *** Tbcips1 = Tbips3 - Tdde(max.)	.050	30	ms	Note 1
Tbcips2	RM's BI RESET L setup time to BCI DC LO L deassertion *** CALCULATED PARAMETER *** Tbcips4 = Tbips5 + Tdde(min.)	50	-	us	-
Tbcips3	RM's BI RESET L hold time from BCI DC LO L deassertion *** CALCULATED PARAMETER *** Tbcips5 = Tbips7 - Tdde(max.)	45	-	us	-

NOTE

1. Maximum specification does not apply for a "real" power sequence case.

digital™

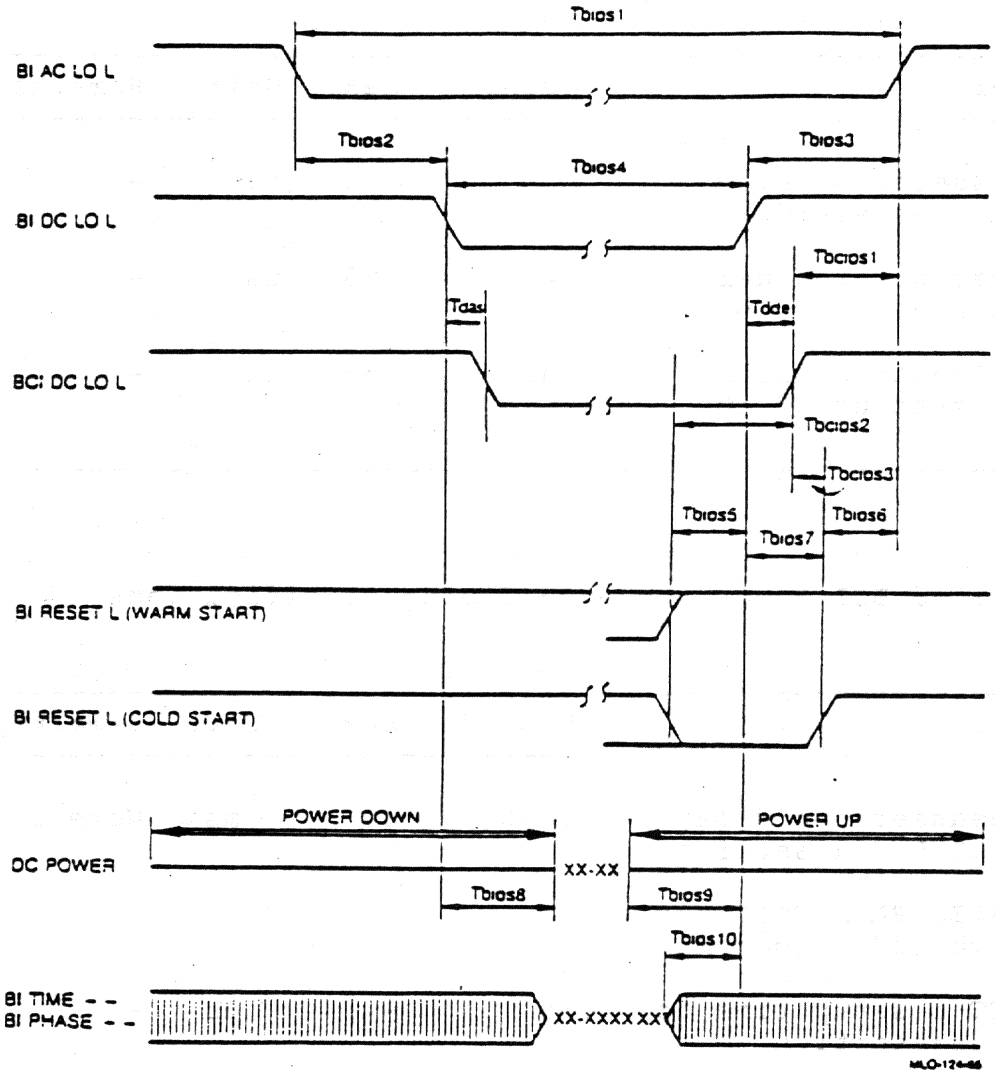
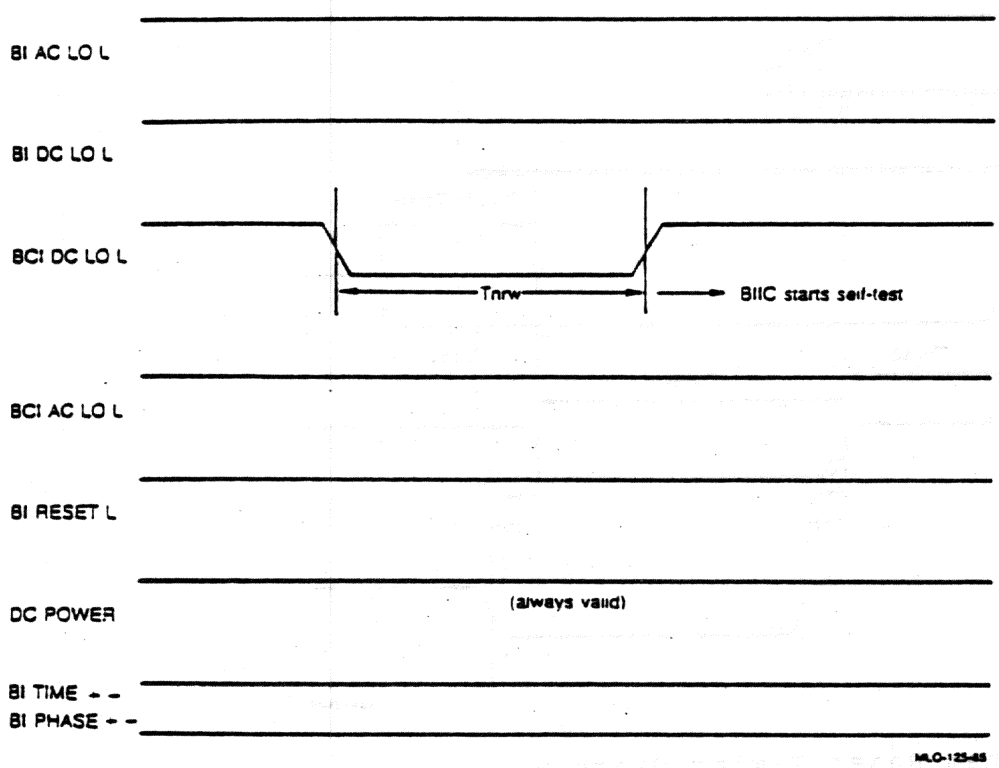


Figure AN7-1: System Reset Sequence

BCI POWER SEQUENCE TIMING



NOTE:

BCI DC LO L asserts two cycles after the setting of the NRST bit.

Figure AN7-2: Node Reset Sequence

digital™

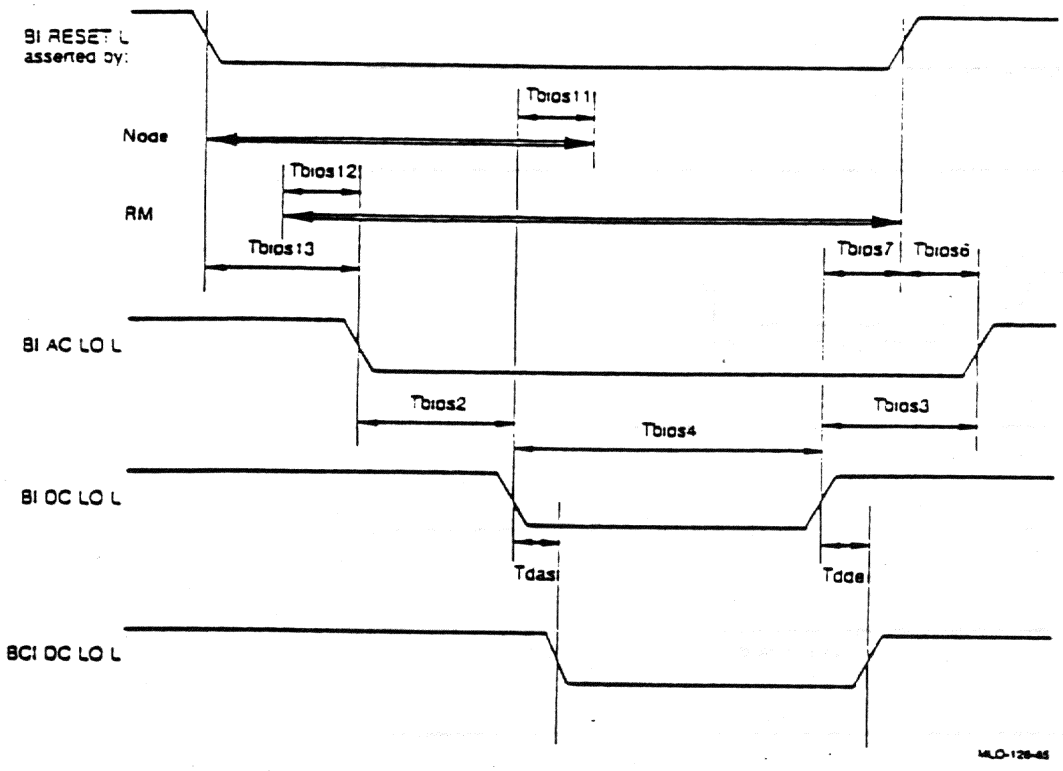


Figure AN7-3: System Reset Timing Diagram

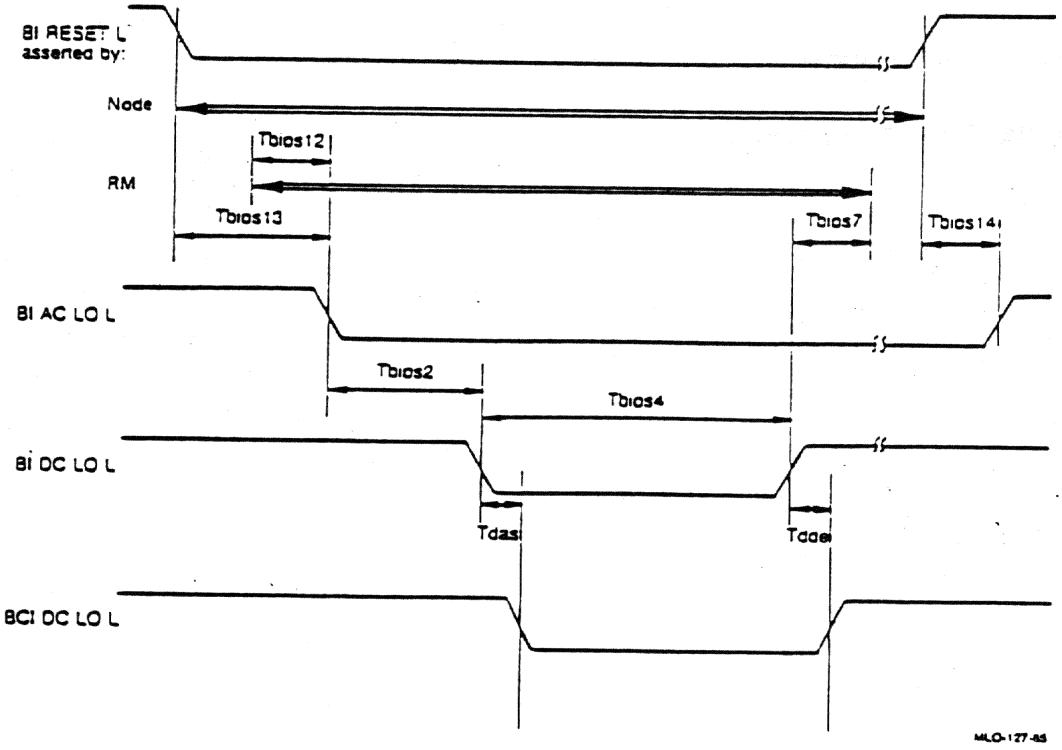


Figure AN7-4: "Extended" System Reset Timing Diagram

digital™

NOTE 8

VAXBI BANDWIDTH AND THE BIIC

The maximum bandwidth of the VAXBI bus, as limited by bus protocol, is described in Chapter 10, Section 10.1. This application note discusses the bandwidth that can be achieved by the master port and slave port resident on a node using the BIIC as the VAXBI primary interface.

The BIIC supports full bandwidth back-to-back transfers to a slave port. Therefore, the maximum slave port bandwidth figures are determined by the VAXBI protocol limits. Figure AN8-1 shows the number of cycles for non-STALLed back-to-back slave port transactions of various lengths.

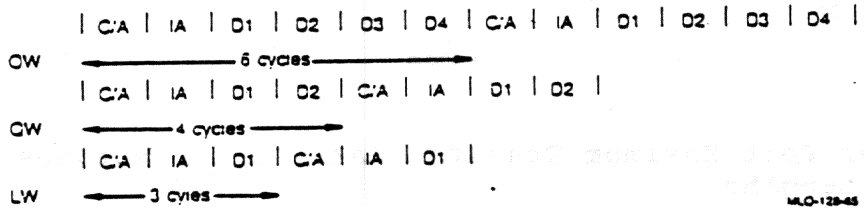


Figure AN8-1: Maximum Bandwidth Obtainable at a BIIC Slave Port for Various Transaction Lengths

The bandwidth for each transaction length can be determined from the following equation:

$$\text{Bandwidth} = \frac{\text{number of bytes transferred}}{\text{number of transaction cycles} * 200\ \text{ns}}$$

digital™

Therefore,

OW (16 bytes) / (6 * 200 ns) = 13.3 Mbytes/s

QW (8 bytes) / (4 * 200 ns) = 10.0 Mbytes/s

LW (4 bytes) / (3 * 200 ns) = 6.67 Mbytes/s

Although the BIIC supports full VAXBI bandwidth transfers to a slave port, a single master port cannot utilize the full VAXBI bandwidth. This is because the BIIC does not permit the master to arbitrate in its own imbedded arbitration cycle, and therefore back-to-back master port transfers from a single node will always be separated by at least one arbitration cycle. This effectively adds one cycle of overhead to each back-to-back transaction from a single node. Figure AN8-2 shows the number of cycles for non-STALLed transactions of various lengths from a single master that wins each arbitration cycle and performs back-to-back transfers.

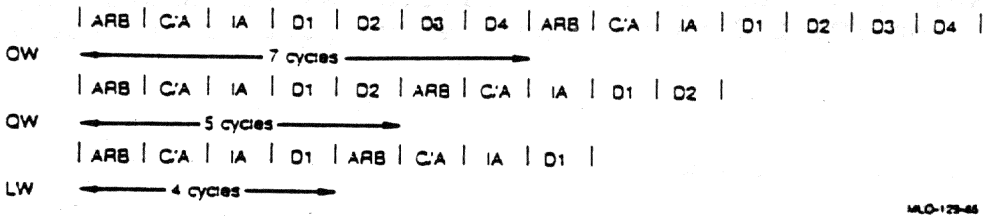


Figure AN8-2: Single Master Port Maximum Transfer Rates for Various Transaction Lengths

These cycle counts correspond to the following maximum transfer rates:

OW = 11.4 Mbytes/s
QW = 8.0 Mbytes/s
LW = 5.0 Mbytes/s

digital™

To obtain this maximum transfer rate, the master port must implement pipeline requests (see Section 15.4.1). Due to the complexity of pipeline-request master port designs, many node designers will choose to implement a simpler nonpipeline-request design (again see Chapter 15, Section 15.4.1). In a nonpipelined design the master port waits for the receipt of the current transaction's summary EV code before posting the request for the next transaction. Since the timing for the summary EV codes differs for read-type and write-type transactions, the maximum transfer rate for nonpipelined designs also depends on the transaction type. Transaction times for longword read-type and write-type transactions are shown in Figures AN8-3 and AN8-4, respectively.

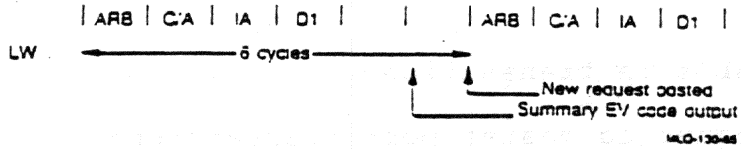


Figure AN8-3: Maximum Transfer Rate for Longword Read-Type Transactions (single nonpipeline-request master port)

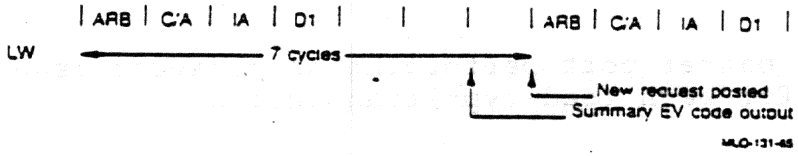


Figure AN8-4: Maximum Transfer Rate for Longword Write-Type Transactions (single nonpipeline-request master port)

digital™

The formula for calculating the maximum obtainable transfer rate from a single master is defined below:

$$\frac{\text{length (bytes)}}{(\text{Arb} + \text{Overhead} + \text{Length} + \text{STALLs} + \text{Latency}) * 200\ \text{ns}} = \text{Transfer rate}$$

Arb = 1 cycle (arbitration)

Overhead = 2 cycles (C/A and IA)

Length = 4 (OW)
2 (QW)
1 (LW)

STALLs = total number of STALLs in transaction

Latency = 0 For pipeline-requested master port transactions
2 For nonpipeline-requested master port read-type transactions
3 For nonpipeline-requested master port write-type transactions

EXAMPLE

Consider a nonpipeline-request master port performing an octaword read of a memory that STALLs twice for each read-type transaction:

$$\frac{16\ \text{bytes}}{(1 + 2 + 4 + 2 + 2) * 200\ \text{ns}} = 7.3\ \text{Mbytes/s}$$

digital™

Table AN8-1 summarizes the master port bandwidth information contained in this application note.

Table AN8-1: Maximum Transfer Rates for BIIC-Based Master Port Nodes

Length	Trans-action	Master Port Type	Non-STALL Case Cycles	Non-STALL Case (Mbytes/s)	2 STALLs/Trans. Cycles	2 STALLs/Trans. (Mbytes/s)
OW	READ	Pipeline	7	11.4	9	8.9
QW	READ	Pipeline	5	8.0	7	5.7
LW	READ	Pipeline	4	5.0	6	3.3
OW	WRITE	Pipeline	7	11.4	9	8.9
QW	WRITE	Pipeline	5	8.0	7	5.7
LW	WRITE	Pipeline	4	5.0	6	3.3
OW	READ	Nonpipeline	9	8.9	11	7.3
QW	READ	Nonpipeline	7	5.7	9	4.4
LW	READ	Nonpipeline	6	3.3	8	2.5
OW	WRITE	Nonpipeline	10	8.0	12	6.7
QW	WRITE	Nonpipeline	8	5.0	10	4.0
LW	WRITE	Nonpipeline	7	2.9	9	2.2

digital™

NOTE 9

USE OF THE RXCD REGISTER IN ROM-BASED DIAGNOSTICS

This application note describes use of the RXCD Register when using diagnostics in read-only memory (ROM) in a VAXBI node. The purpose of these diagnostics is to extend the coverage provided by node self-test. As defined by the VAXBI requirements, node self-test specifies completion time, configuration options, and device setup variables.

Advantages of ROM-based diagnostics over traditional diagnostic approaches that are software-based are as follows:

- Operating system independent
- Reduction or elimination of distribution media
- Faster execution time (diagnostics run on a local processor without use of the VAXBI bus)
- No dependency on diagnostic load path
- Transportability (ROM-based diagnostics are always present and operate on the node regardless of what system the node is in)

Section AN9.1 suggests a way for a VAXBI node with ROM-based diagnostics to report status by writing to the RXCD Register of a processor on a host system. Section AN9.2 describes a way for a node to record diagnostic results when the host system has no RXCD Register.

AN9.1 USE OF THE RXCD REGISTER IN A HOST SYSTEM

Each VAXBI console has a VAXBI nodespace register, the Receive Console Data Register (RXCD) for receiving data from other consoles. The RXCD Register occupies the address bb + 200 in the nodespace of the node (bb is the starting address for the node's nodespace). The RXCD Register can be implemented on any VAXBI node. (Chapter 7, Section

digital™

7.17, describes the RXCD Register.)

If bit <15> (the Busy 1 bit), is clear, a node can write data into this register. The node writing the register sets the Busy 1 bit and places its node ID in the Node ID field. The node with the RXCD Register can then determine which node gave it the data. When the node reads the register, it clears the Busy 1 bit.

The Z console command is implemented by all VAXBI consoles and is used to send a character or message to an RXCD Register. The protocol for console communication is described in Section AN9.3.

The Z console command can be used to forward commands from a VAXBI system console to any node implementing the RXCD Register. This mechanism can be used for executing and reporting status of ROM-based diagnostics. Its use provides a full duplex transparent path, with echoing, which is necessary for efficient diagnostics. By using the console protocol, the invoker of the ROM-based diagnostics does not have to guess when to look for status, as in the case of status written to general purpose registers on the node under test.

Using this protocol, the firmware running on the node to be tested would be interrupted and recognize that the RXCD Register had been written, and examine the ASCII character in the register. If the three characters T/R are consecutively received, the firmware would then pass control to diagnostic code. The diagnostic code would receive and understand further characters in the RXCD Register that mean, for example, "run the disk diagnostic exerciser." By checking the Node ID field in its RXCD, the adapter running the diagnostics will only accept commands from the node that initiated the diagnostics. As the diagnostic runs the test, it reports status back to the console's RXCD Register, which causes the contents to be printed on the console printer when the system is in console I/O mode. The RXCD Registers are written using VAXBI interlocked reads and writes.

In console I/O mode, a user can type sequences of commands to execute ROM-based diagnostics on the node. The console printer will give the results of the tests run. A sequence of commands might be:

1. Z 2 -- Forward all commands to node 2.

2. <ESC><CTRL/P> -- Halt the second node about to be tested. (An escape character is needed so that the first console does not think the user is trying to terminate the conversation with node 2.)

3. T/R -- Invoke the ROM-based diagnostics on node 2.

4. D1 -- Tell node 2 to run diagnostic number 1.

digital™

5. E -- Exit from diagnostic mode.

6. <CTRL/P> -- Terminate the conversation with node 2 and return conversation back to the first console.

When the node under test is the processor to which the console terminal is attached, the Z command is not necessary. A <CTRL/P> halts the processor and a T/R invokes the diagnostics on the node.

When a system is not running in stand-alone mode, as when a diagnostic control program is running in the primary processor, that program can use the same mechanism to invoke the ROM-based diagnostics on the nodes. The program can write the RXCD of the node under test and receive the interrupt when the diagnostic begins reporting results back to its own RXCD.

The console protocol requires the following sequence for writing to the RXCD: (a) IRCI to the RXCD, (b) examine the Busy 1 bit, (c) if it's not set, then issue UWMCI to the RXCD with the Busy 1 bit set and a data character in the prescribed position; otherwise, issue UWMCI to the RXCD with the original contents. (The software on the host may not be able to generate this sequence.) However, when writing to the RXCD of the node under test, it is not the subject of writes from some other source at the same time. Therefore, a READ or RCI transaction can be substituted for IRCI, and a WRITE, WCI, or WMCI transaction can be substituted for UWMCI.

Using the console protocol, a diagnostic control program can receive data and then package the results for the user running the program. In addition, the control program can sort out and package messages according to node ID received, allowing parallel execution. Also, the control program can allow the invocation of diagnostics in the automated manufacturing world and allow simplified customer-runnable diagnostic user interfaces. This mechanism allows the diagnostics to be invoked from another system over the NI.

Use of the RXCD protocol does not prevent the invocation of ROM-based diagnostics while a system is online, but the protocol may be adversely affected if precautions are not taken. When an operating system is running, the assumption made above, that the RXCD is not the subject of writes from some other source at the same time, is not necessarily valid. In this case, the console protocol may not work properly. Care must be taken in higher level software, such as the operating system, to ensure that the protocol is not adversely affected while an operating system is executing.

digital™

AN9.2 USE OF THE RXCD REGISTER IN THE NODE UNDER TEST WHEN THE HOST HAS NO RXCD REGISTER

Some VAXBI nodes that are used as host nodes to invoke ROM-based diagnostics do not have an RXCD Register. To provide for these host nodes without RXCD Registers, an alternative protocol is available.

Bits <31:16> of the RXCD Register of the node under test can be used instead of the host's RXCD Register. The protocol would be implemented similarly to that described previously. The control program must determine which RXCD to use. One way to do this is to determine the processor-type on which it is running. Another method is to attempt a VAXBI transaction to the expected location of the RXCD (on the host node). If the transaction is not acknowledged, or if the expected RXCD location has its Busy 1 bit set over a period of two seconds, then the control program will use the modified scheme, using the RXCD Register of the node under test for all communication. The ROM-based diagnostics must also determine which host RXCD to use. The code running on the node under test can attempt a VAXBI transaction to the expected location of the RXCD (on the host node). If the transaction is not acknowledged, or if the expected RXCD location has its Busy 1 bit set over a period of two seconds, then the ROM-based program will use the modified scheme, using its own RXCD for all communication. Use of the first protocol is preferable over the second protocol, especially in a stand-alone environment. The latter is less efficient in that the host is not interrupted.

digital™

NOTE 10

USE OF DREV FIELD OF DTYPE REGISTER

The revision field is intended to reflect functional changes that occur to VAXBI options. The fundamental purpose of the DREV field is to make functional revisions visible to software. This allows software to verify minimal revision requirements, or to adapt to different revisions. This also allows revision information to be captured as part of crash dumps and other error snapshots for subsequent analysis.

Most VAXBI options include cabling (sometimes called a cabinet kit), in addition to one or more VAXBI modules. Since changes to the cabling do not usually affect the software-visible functionality of VAXBI options, cabling is not normally covered by machine-readable revision information.

If the option consists of a single VAXBI module, then the entire DREV field is normally allocated to revision information for that module. If the option consists of two (or more) VAXBI modules, then the DREV field is normally split into two (or more) revision sub-fields, one for each VAXBI module. Based on current revision control practice in Digital, it is highly inadvisible to allocate less than five writable bits to any module; for programmable modules, eight or more bits are recommended.

To enhance maintainability, the preferred practice is for the DREV field (or the revision sub-fields within the DREV field) to contain a value which may be simply related to the letter revision marked on the option or the module. The machine-readable revision field should not change to reflect numerical (minor) changes to the module.

Some options have used one field for a hardware revision and another field for a firmware revision on the same module. In accordance with the previous paragraph, separate fields are appropriate if the revision marked on the module is tied only to the hardware revision (and is, hence, independent of the firmware revision); in this case, the machine-readable revision should match the revision marked on the

digital™

module. If the hardware revision changes whenever the firmware revision changes (typically true for ROM-based firmware which is not expected to be field-changeable), then only one machine-readable revision field should be used, and this revision should match that marked on the module.

The table on the following pages presents the preferred pairing of revision marking letters and machine-readable VAXBI revision fields. Since certain letters are not used for revision marking by Digital, many machine-readable revision codes should not be used.

digital™

USE OF DREV FIELD OF DTYPE REGISTER

Dec Hex Ltr	Dec Hex Ltr	Dec Hex Ltr	Dec Hex Ltr	Dec Hex Ltr
1= 1 => A	2= 2 => B	3= 3 => C	4= 4 => D	5= 5 => E
6= 6 => F	7=DoNotUse	8= 8 => H	9=DoNotUse	10= A => J
11= B => K	12= C => L	13= D => M	14= E => N	15=DoNotUse
16= 10 => P	17=DoNotUse	18= 12 => R	19= 13 => S	20= 14 => T
21= 15 => U	22= 16 => V	23= 17 => W	24=DoNotUse	25= 19 => Y
26= 1A => Z	27= 1B =>AA	28= 1C =>AB	29= 1D =>AC	30= 1E =>AD
31= 1F =>AE	32= 20 =>AF	33=DoNotUse	34= 22 =>AH	35=DoNotUse
36= 24 =>AJ	37= 25 =>AK	38= 26 =>AL	39= 27 =>AM	40= 28 =>AN
41=DoNotUse	42= 2A =>AP	43=DoNotUse	44= 2C =>AR	45= 2D =>AS
46= 2E =>AT	47= 2F =>AU	48= 30 =>AV	49= 31 =>AW	50=DoNotUse
51= 33 =>AY	52= 34 =>AZ	53= 35 =>BA	54= 36 =>BB	55= 37 =>BC
56= 38 =>BD	57= 39 =>BE	58= 3A =>BF	59=DoNotUse	60= 3C =>BH
61=DoNotUse	62= 3E =>BJ	63= 3F =>BK	64= 40 =>BL	65= 41 =>BM
66= 42 =>BN	67=DoNotUse	68= 44 =>BP	69=DoNotUse	70= 46 =>BR
71= 47 =>BS	72= 48 =>BT	73= 49 =>BU	74= 4A =>BV	75= 4B =>BW
76=DoNotUse	77= 4D =>BY	78= 4E =>BZ	79= 4F =>CA	80= 50 =>CB
81= 51 =>CC	82= 52 =>CD	83= 53 =>CE	84= 54 =>CF	85=DoNotUse
86= 56 =>CH	87=DoNotUse	88= 58 =>CJ	89= 59 =>CK	90= 5A =>CL
91= 5B =>CM	92= 5C =>CN	93=DoNotUse	94= 5E =>CP	95=DoNotUse
96= 60 =>CR	97= 61 =>CS	98= 62 =>CT	99= 63 =>CU	100= 64 =>CV
101= 65 =>CW	102=DoNotUse	103= 67 =>CY	104= 68 =>CZ	105= 69 =>DA
106= 6A =>DB	107= 6B =>DC	108= 6C =>DD	109= 6D =>DE	110= 6E =>DF
111=DoNotUse	112= 70 =>DH	113=DoNotUse	114= 72 =>DJ	115= 73 =>DK
116= 74 =>DL	117= 75 =>DM	118= 76 =>DN	119=DoNotUse	120= 78 =>DP
121=DoNotUse	122= 7A =>DR	123= 7B =>DS	124= 7C =>DT	125= 7D =>DU
126= 7E =>DV	127= 7F =>DW	128=DoNotUse	129= 81 =>DY	130= 82 =>DZ
131= 83 =>EA	132= 84 =>EB	133= 85 =>EC	134= 86 =>ED	135= 87 =>EE
136= 88 =>EF	137=DoNotUse	138= 8A =>EH	139=DoNotUse	140= 8C =>EJ
141= 8D =>EK	142= 8E =>EL	143= 8F =>EM	144= 90 =>EN	145=DoNotUse
146= 92 =>EP	147=DoNotUse	148= 94 =>ER	149= 95 =>ES	150= 96 =>ET
151= 97 =>EU	152= 98 =>EV	153= 99 =>EW	154=DoNotUse	155= 9B =>EY
156= 9C =>EZ	157= 9D =>FA	158= 9E =>FB	159= 9F =>FC	160= A0 =>FD
161= A1 =>FE	162= A2 =>FF	163=DoNotUse	164= A4 =>FH	165=DoNotUse
166= A6 =>FJ	167= A7 =>FK	168= A8 =>FL	169= A9 =>FM	170= AA =>FN
171=DoNotUse	172= AC =>FP	173=DoNotUse	174= AE =>FR	175= AF =>FS
176= B0 =>FT	177= B1 =>FU	178= B2 =>FV	179= B3 =>FW	180=DoNotUse
181= B5 =>FY	182= B6 =>FZ	183=DoNotUse	184=DoNotUse	185=DoNotUse
186=DoNotUse	187=DoNotUse	188=DoNotUse	189=DoNotUse	190=DoNotUse
191=DoNotUse	192=DoNotUse	193=DoNotUse	194=DoNotUse	195=DoNotUse
196=DoNotUse	197=DoNotUse	198=DoNotUse	199=DoNotUse	200=DoNotUse

digital™

USE OF DREV FIELD OF DTYPE REGISTER

Dec Hex Ltr	Dec Hex Ltr	Dec Hex Ltr	Dec Hex Ltr	Dec Hex Ltr
201=DoNotUse	202=DoNotUse	203=DoNotUse	204=DoNotUse	205=DoNotUse
206=DoNotUse	207=DoNotUse	208=DoNotUse	209= D1 =>HA	210= D2 =>HB
211= D3 =>HC	212= D4 =>HD	213= D5 =>HE	214= D6 =>HF	215=DoNotUse
216= D8 =>HH	217=DoNotUse	218= DA =>HJ	219= DB =>HK	220= DC =>HL
221= DD =>HM	222= DE =>HN	223=DoNotUse	224= E0 =>HP	225=DoNotUse
226= E2 =>HR	227= E3 =>HS	228= E4 =>HT	229= E5 =>HU	230= E6 =>HV
231= E7 =>HW	232=DoNotUse	233= E9 =>HY	234= EA =>HZ	235=DoNotUse
236=DoNotUse	237=DoNotUse	238=DoNotUse	239=DoNotUse	240=DoNotUse
241=DoNotUse	242=DoNotUse	243=DoNotUse	244=DoNotUse	245=DoNotUse
246=DoNotUse	247=DoNotUse	248=DoNotUse	249=DoNotUse	250=DoNotUse
251=DoNotUse	252=DoNotUse	253=DoNotUse	254=DoNotUse	255=DoNotUse
256=DoNotUse	257=DoNotUse	258=DoNotUse	259=DoNotUse	260=DoNotUse
261=105 =>JA	262=106 =>JB	263=107 =>JC	264=108 =>JD	265=109 =>JE
266=10A =>JF	267=DoNotUse	268=10C =>JH	269=DoNotUse	270=10E =>JJ
271=10F =>JK	272=110 =>JL	273=111 =>JM	274=112 =>JN	275=DoNotUse
276=114 =>JP	277=DoNotUse	278=116 =>JR	279=117 =>JS	280=118 =>JT
281=119 =>JU	282=11A =>JV	283=11B =>JW	284=DoNotUse	285=11D =>JY
286=11E =>JZ	287=11F =>KA	288=120 =>KB	289=121 =>KC	290=122 =>KD
291=123 =>KE	292=124 =>KF	293=DoNotUse	294=126 =>KH	295=DoNotUse
296=128 =>KJ	297=129 =>KK	298=12A =>KL	299=12B =>KM	300=12C =>KN
301=DoNotUse	302=12E =>KP	303=DoNotUse	304=130 =>KR	305=131 =>KS
306=132 =>KT	307=133 =>KU	308=134 =>KV	309=135 =>KW	310=DoNotUse
311=137 =>KY	312=138 =>KZ	313=139 =>LA	314=13A =>LB	315=13B =>LC
316=13C =>LD	317=13D =>LE	318=13E =>LF	319=DoNotUse	320=140 =>LH
321=DoNotUse	322=142 =>LJ	323=143 =>LK	324=144 =>LL	325=145 =>LM
326=146 =>LN	327=DoNotUse	328=148 =>LP	329=DoNotUse	330=14A =>LR
331=14B =>LS	332=14C =>LT	333=14D =>LU	334=14E =>LV	335=14F =>LW
336=DoNotUse	337=151 =>LY	338=152 =>LZ	339=153 =>MA	340=154 =>MB
341=155 =>MC	342=156 =>MD	343=157 =>ME	344=158 =>MF	345=DoNotUse
346=15A =>MH	347=DoNotUse	348=15C =>MJ	349=15D =>MK	350=15E =>ML
351=15F =>MM	352=160 =>MN	353=DoNotUse	354=162 =>MP	355=DoNotUse
356=164 =>MR	357=165 =>MS	358=166 =>MT	359=167 =>MU	360=168 =>MV
361=169 =>MW	362=DoNotUse	363=16B =>MY	364=16C =>MZ	365=16D =>NA
366=16E =>NB	367=16F =>NC	368=170 =>ND	369=171 =>NE	370=172 =>NF
371=DoNotUse	372=174 =>NH	373=DoNotUse	374=176 =>NJ	375=177 =>NK
376=178 =>NL	377=179 =>NM	378=17A =>NN	379=DoNotUse	380=17C =>NP
381=DoNotUse	382=17E =>NR	383=17F =>NS	384=180 =>NT	385=181 =>NU
386=182 =>NV	387=183 =>NW	388=DoNotUse	389=185 =>NY	390=186 =>NZ
391=DoNotUse	392=DoNotUse	393=DoNotUse	394=DoNotUse	395=DoNotUse
396=DoNotUse	397=DoNotUse	398=DoNotUse	399=DoNotUse	400=DoNotUse

digital™

USE OF DREV FIELD OF DTYPE REGISTER

Dec Hex Ltr	Dec Hex Ltr	Dec Hex Ltr	Dec Hex Ltr	Dec Hex Ltr
401=DoNotUse	402=DoNotUse	403=DoNotUse	404=DoNotUse	405=DoNotUse
406=DoNotUse	407=DoNotUse	408=DoNotUse	409=DoNotUse	410=DoNotUse
411=DoNotUse	412=DoNotUse	413=DoNotUse	414=DoNotUse	415=DoNotUse
416=DoNotUse	417=1A1 =>PA	418=1A2 =>PB	419=1A3 =>PC	420=1A4 =>PD
421=1A5 =>PE	422=1A6 =>PF	423=DoNotUse	424=1A8 =>PH	425=DoNotUse
426=1AA =>PJ	427=1AB =>PK	428=1AC =>PL	429=1AD =>PM	430=1AE =>PN
431=DoNotUse	432=1B0 =>PP	433=DoNotUse	434=1B2 =>PR	435=1B3 =>PS
436=1B4 =>PT	437=1B5 =>PU	438=1B6 =>PV	439=1B7 =>PW	440=DoNotUse
441=1B9 =>PY	442=1BA =>PZ	443=DoNotUse	444=DoNotUse	445=DoNotUse
446=DoNotUse	447=DoNotUse	448=DoNotUse	449=DoNotUse	450=DoNotUse
451=DoNotUse	452=DoNotUse	453=DoNotUse	454=DoNotUse	455=DoNotUse
456=DoNotUse	457=DoNotUse	458=DoNotUse	459=DoNotUse	460=DoNotUse
461=DoNotUse	462=DoNotUse	463=DoNotUse	464=DoNotUse	465=DoNotUse
466=DoNotUse	467=DoNotUse	468=DoNotUse	469=1D5 =>RA	470=1D6 =>RB
471=1D7 =>RC	472=1D8 =>RD	473=1D9 =>RE	474=1DA =>RF	475=DoNotUse
476=1DC =>RH	477=DoNotUse	478=1DE =>RJ	479=1DF =>RK	480=1E0 =>RL
481=1E1 =>RM	482=1E2 =>RN	483=DoNotUse	484=1E4 =>RP	485=DoNotUse
486=1E6 =>RR	487=1E7 =>RS	488=1E8 =>RT	489=1E9 =>RU	490=1EA =>RV
491=1EB =>RW	492=DoNotUse	493=1ED =>RY	494=1EE =>RZ	495=1EF =>SA
496=1F0 =>SB	497=1F1 =>SC	498=1F2 =>SD	499=1F3 =>SE	500=1F4 =>SF
501=DoNotUse	502=1F6 =>SH	503=DoNotUse	504=1F8 =>SJ	505=1F9 =>SK
506=1FA =>SL	507=1FB =>SM	508=1FC =>SN	509=DoNotUse	510=1FE =>SP
511=DoNotUse	512=200 =>SR	513=201 =>SS	514=202 =>ST	515=203 =>SU
516=204 =>SV	517=205 =>SW	518=DoNotUse	519=207 =>SY	520=208 =>SZ
521=209 =>TA	522=20A =>TB	523=20B =>TC	524=20C =>TD	525=20D =>TE
526=20E =>TF	527=DoNotUse	528=210 =>TH	529=DoNotUse	530=212 =>TJ
531=213 =>TK	532=214 =>TL	533=215 =>TM	534=216 =>TN	535=DoNotUse
536=218 =>TP	537=DoNotUse	538=21A =>TR	539=21B =>TS	540=21C =>TT
541=21D =>TU	542=21E =>TV	543=21F =>TW	544=DoNotUse	545=221 =>TY
546=222 =>TZ	547=223 =>UA	548=224 =>UB	549=225 =>UC	550=226 =>UD
551=227 =>UE	552=228 =>UF	553=DoNotUse	554=22A =>UH	555=DoNotUse
556=22C =>UJ	557=22D =>UK	558=22E =>UL	559=22F =>UM	560=230 =>UN
561=DoNotUse	562=232 =>UP	563=DoNotUse	564=234 =>UR	565=235 =>US
566=236 =>UT	567=237 =>UU	568=238 =>UV	569=239 =>UW	570=DoNotUse
571=23B =>UY	572=23C =>UZ	573=23D =>VA	574=23E =>VB	575=23F =>VC
576=240 =>VD	577=241 =>VE	578=242 =>VF	579=DoNotUse	580=244 =>VH
581=DoNotUse	582=246 =>VJ	583=247 =>VK	584=248 =>VL	585=249 =>VM
586=24A =>VN	587=DoNotUse	588=24C =>VP	589=DoNotUse	590=24E =>VP
591=24F =>VS	592=250 =>VT	593=251 =>VU	594=252 =>VV	595=253 =>VW
596=DoNotUse	597=255 =>VY	598=256 =>VZ	599=257 =>WA	600=258 =>WB

digital™

USE OF DREV FIELD OF DTYPE REGISTER

Dec Hex Ltr	Dec Hex Ltr	Dec Hex Ltr	Dec Hex Ltr	Dec Hex Ltr
601=259 =>WC	602=25A =>WD	603=25B =>WE	604=25C =>WF	605=DoNotUse
606=25E =>WH	607=DoNotUse	608=260 =>WJ	609=261 =>WK	610=262 =>WL
611=263 =>WM	612=264 =>WN	613=DoNotUse	614=266 =>WP	615=DoNotUse
616=268 =>WR	617=269 =>WS	618=26A =>WT	619=26B =>WU	620=26C =>WV
621=26D =>WW	622=DoNotUse	623=26F =>WY	624=270 =>WZ	625=DoNotUse
626=DoNotUse	627=DoNotUse	628=DoNotUse	629=DoNotUse	630=DoNotUse
631=DoNotUse	632=DoNotUse	633=DoNotUse	634=DoNotUse	635=DoNotUse
636=DoNotUse	637=DoNotUse	638=DoNotUse	639=DoNotUse	640=DoNotUse
641=DoNotUse	642=DoNotUse	643=DoNotUse	644=DoNotUse	645=DoNotUse
646=DoNotUse	647=DoNotUse	648=DoNotUse	649=DoNotUse	650=DoNotUse
651=28B =>YA	652=28C =>YB	653=28D =>YC	654=28E =>YD	655=28F =>YE
656=290 =>YF	657=DoNotUse	658=292 =>YH	659=DoNotUse	660=294 =>YJ
661=295 =>YK	662=296 =>YL	663=297 =>YM	664=298 =>YN	665=DoNotUse
666=29A =>YP	667=DoNotUse	668=29C =>YR	669=29D =>YS	670=29E =>YT
671=29F =>YU	672=2A0 =>YV	673=2A1 =>YW	674=DoNotUse	675=2A3 =>YY
676=2A4 =>YZ	677=2A5 =>ZA	678=2A6 =>ZB	679=2A7 =>ZC	680=2A8 =>ZD
681=2A9 =>ZE	682=2AA =>ZF	683=DoNotUse	684=2AC =>ZH	685=DoNotUse
686=2AE =>ZJ	687=2AF =>ZK	688=2B0 =>ZL	689=2B1 =>ZM	690=2B2 =>ZN
691=DoNotUse	692=2B4 =>ZP	693=DoNotUse	694=2B6 =>ZR	695=2B7 =>ZS
696=2B8 =>ZT	697=2B9 =>ZU	698=2BA =>ZV	699=2BB =>ZW	700=DoNotUse
701=2BD =>ZY	702=2BE =>ZZ			

digital™

APPENDIX A

BDCST TRANSACTION

The BDCST (broadcast) transaction is reserved for use by Digital. VAXBI nodes must not issue this command during normal operation.*

The BDCST transaction, a multi-responder transaction, allows a node to transfer simultaneously one to four longwords of data to any combination of destination nodes. The BDCST transaction has the same format as write-type transactions; however, the selection information in a BDCST transaction is a destination node ID mask instead of a physical address.

Figure A-1 shows the format of the BDCST transaction.

---

*This restriction ensures that no VAXBI node will lose compatibility because it cannot issue or respond to the BDCST command.

digital™

BDCST TRANSACTION

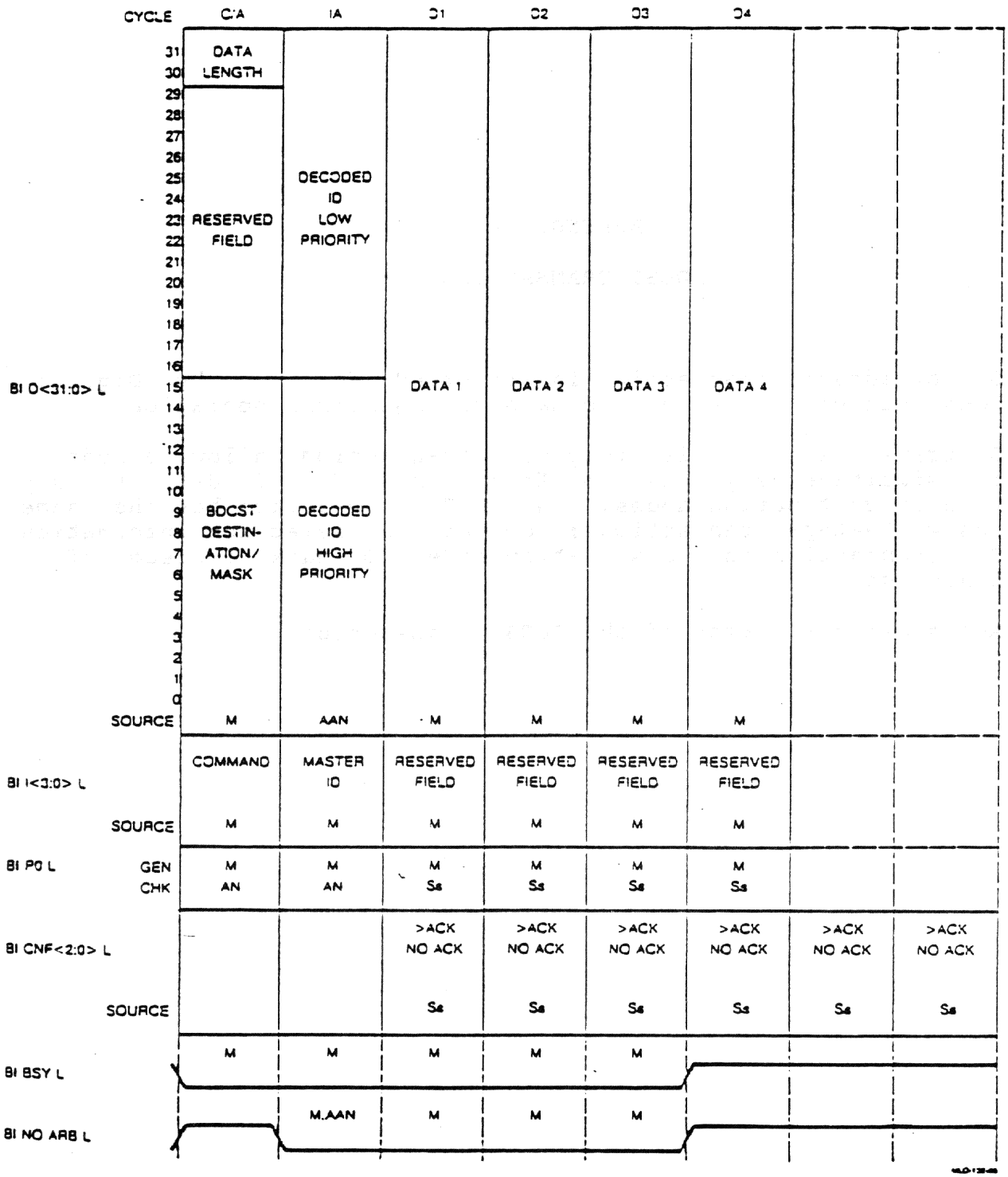


Figure A-1: BDCST Transaction

APPENDIX B

WIRED-OR TRANSMISSION LINE EFFECT

This appendix briefly describes the problem referred to in Chapter 6, Sections 6.3 and 6.4, which mandates that receivers of BI AC LO L and BI DC LO L lines reject short spurious deassertions.

The spurious deassertion problem is best explained as a result of a transmission line effect. Assume only two power supplies are tied into the BI DC LO L line: PS1 at line length 1 and PS2 at line length 2. Each power supply line has characteristic impedance Z0 (see Figure B-1). Also assume that when BI DC LO L is asserted, both drivers Q1 and Q2 are simultaneously conducting for some period and that Q1 of PS1 is "current hogging" 0.9 of the total current (It). If Q1 is the first supply to deassert BI DC LO L and the rise time of the signal is relatively fast, instantaneously the receiver "sees" an input voltage amplitude of approximately V/2. If this voltage is over its threshold, the voltage would be recognized as a valid logic level. Thus, this spurious deassertion is due to the unequal distribution of line lengths and currents. The receiver physically close to PS2 essentially "sees" the length 2 line impedance Z0 and the resistor R (equal in value to Z0) as a voltage divider to the +V power supply. As Q2 begins to sink the unbalanced current and the V/2 wavefront propagates down the line, the receiver will continue to see V/2 until a reflected waveform propagates back from the short circuit stub at power supply PS2. During this propagation time the receiver incorrectly indicates that BI DC LO L is deasserted.

For the VAXBI bus, the BIIC's BI AC LO L and BI DC LO L receivers ignore spurious deassertions of less than 200 ns, thereby assuring that only intended deassertions will be recognized.

digital™

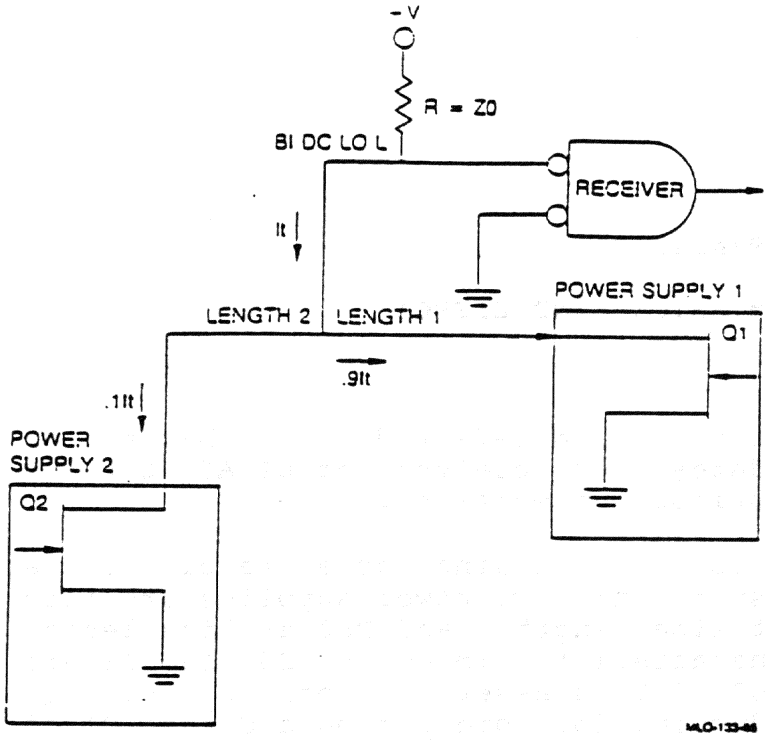


Figure B-1: Circuit to Demonstrate Transmission Line Problem of Wire-ORing

digital™

APPENDIX C

RESPONSES TO EXCEPTION CONDITIONS

This appendix contains guidelines on how to respond to the exception conditions that are expressed in the form of BIIC EV codes.

The following EV codes are cited in the tables below. See Chapter 15 for descriptions of the EV codes.

BBE	Bus BSY Error
BPM	Bad Parity Received During Master Port Transaction
BPR	Bad Parity Received
BTO	Bus Timeout (>4095 cycles)
ICRMC	Illegal CNF Received for Master Port Command
ICRMD	Illegal CNF Received by Master Port for Data Cycle
ICRSD	Illegal CNF Received for Slave Data (last two CNFs)
MTCE	Master Transmit Check Error (received & driven data differ)
NCRMC	NO ACK CNF Received for Master Port Command
NICI	NO ACK or Illegal CNF Received for INTR Command
NICIPS	NO ACK or Illegal CNF Received for Force-Bit IPINTR/STOP Command
RDSR	Read Data Substitute or RESERVED Status Code Received
RTO	Retry Timeout (>4095 RETRYs)
STO	Stall Timeout on Slave Transaction (>127 STALLs)

Upper and lowercase letters in the table denote the following:

H	Hung master transaction.
M	Machine check or similar response appropriate.
N	No action needed.
R	Can reattempt the transaction.
a	If an Interlock Read, don't lock.
b	Slave should reset and prepare for next transaction.
c	If I/O space or Unlock Write, don't reattempt.
d	Suppress write on bad parity.
e	Retry could mean extraneous IPINTR.
f	An NCRMC may mean interrupt was serviced.
g	Don't clear lock with an Unlock Write.
h	Retain interrupt vector in case IDENT is reattempted.

digital™

j Don't use the read data returned.
k Possible only during intranode transfers; EV code is intended for slave port.
m Possible only during intranode transfers; EV code is intended for master port.

Processor Class

	Read-Type	Write-Type	STOP	INVAL	IPINTR	INTR	IDENT
Master Summary							
RTO	Mg	M	-	-	-	-	NR
BPM	MR	-	-	-	-	-	NR
ICRMC	MR	MRc	MR	MR	NRe	-	NR
NCRMC	MR	MR	MR	N	NR	-	NRf
ICRMD	MR	MRc	-	-	-	-	NR
MTCE	MR	MRc	MR	MR	NR	-	NR
RDSR	MRj	-	-	-	-	-	NR
NICIPS	-	-	-	-	NRe	-	-
NICI	-	-	-	-	-	NR	-
Master Status							
BPR	j	Nk	-	-	-	-	N
Master Special							
BTO	H	H	H	H	H	H	H
Slave Summary							
ICRSD	Na	-	-	-	-	-	Nh
BBE	N	N	N	N	N	N	Nh
Slave Status							
STO	b	b	-	-	-	-	b
BPR	Nm	d	-	-	-	-	-

digital™

RESPONSES TO EXCEPTION CONDITIONS

Memory Class

	Read-Type	Write-Type	STOP	INVAL	IPINTR	INTR	IDENT
Master Summary							
ICRMC	-	-	-	NR	-	-	-
NCRMC	-	-	-	N	-	-	-
MTCE	-	-	-	NR	-	-	-
NICI	-	-	-	-	-	NR	-
Master Special							
BTO	H	-	H	-	-	H	-
Slave Summary							
ICRSD	Na	-	-	-	-	-	Nh
BBE	N	N	N	N	N	N	Nh
Slave Status							
STO	b	b	-	-	-	-	b
BPR	-	d	-	-	-	-	-

Adapter Class

	Read-Type	Write-Type	STOP	INVAL	IPINTR	INTR	IDENT
Master Summary							
BPM	NR	-	-	-	-	-	-
ICRMC	NR	NRc	NR	NR	-	-	-
NCRMC	NR	NR	NR	N	-	-	-
ICRMD	NR	NRc	-	-	-	-	-
MTCE	NR	NRc	NR	NR	-	-	-
RDSR	NR	-	-	-	-	-	-
NICI	-	-	-	-	-	NR	-
Master Status							
BPR	j	Nk	-	-	-	-	-
Master Special							
BTO	H	H	H	H	-	H	-
Slave Summary							
ICRSD	Na	-	-	-	-	-	Nh
BBE	N	N	N	N	N	N	Nh
Slave Status							
STO	b	b	-	-	-	-	b
BPR	Nm	d	-	-	-	-	-

digital™

APPENDIX D

DIGITAL EXCEPTIONS TO VAXBI REQUIREMENTS

Each section below first states a VAXBI requirement and then describes exceptions to the requirement. Exceptions to VAXBI requirements are granted by the MSB Systems Architecture Group. References to unannounced products or Digital proprietary information have been replaced with "symbols". "Symbols" are strings of the form '$'integer. The values of these symbols are kept in a table maintained by the MSB systems architecture group. "To be determined" references are denoted by "[]".

D.1 EXTENSION CYCLE LIMIT

The number of STALL cycles allowed per transaction is eight. The BIIC limits the number of STALL cycles to 127 for each data cycle. Upon receipt of the 128th STALL response code, the BIIC causes a timeout. The following nodes exceed the stall limit:

- XMI to VAXBI Adapter (XBI). The XBI may exceed the 127 STALL cycle limit while responding to a VAXBI read-type transaction.
- UNIBUS to VAXBI Adapter (DWBUA). As a VAXBI slave, a UNIBUS adapter may issue up to 127 stalls per data cycle within a VAXBI transaction. If the DWBUA is targeted as slave on the VAXBI bus while it is performing a UNIBUS transaction, the DWBUA issues STALLs until the UNIBUS transaction completes. The UNIBUS timeout for these circumstances is about 50 cycles or 10 microseconds.
- KA88 Memory Interconnect to VAXBI Adapter (DB88). The maximum number of STALL cycles that the DB88 might issue is [].
- Aurora processor module (KA800). The maximum number of STALL cycles that the Aurora processor module might issue is []. The typical number of STALL cycles is 15.

digital™

- VAXBI Memory (MS820). When doing a refresh, VAXBI memory may exceed the limit on STALL cycles. For interleaved memory, the maximum number of STALL cycles per transaction is 9. For noninterleaved memory, the maximum number is 13.

Nodes must not use more than 16 consecutive extension cycles without issuing a VAXBI transaction request.

- UNIBUS to VAXBI Adapter (DWBUA). The DWBUA uses up to 32 consecutive extension cycles.

D.2 INSTRUCTION NOT TO CACHE DATA

Nodes that receive data with a "don't cache" read status must not cache the data. The following node violates this requirement:

- VAX 8200 Processor (KA820)

D.3 UNSUCCESSFUL IRCI TRANSACTIONS

If an IRCI transaction does not complete successfully, the lock must not be set. The following nodes are exceptions to this rule:

- KA88 Memory Interconnect to VAXBI Adapter (DB88). The MS88 memory controller sets the lock before the end of the IRCI transaction, and the DB88 cannot clear the lock if a NO ACK is received. After a timeout the memory controller transmits an interrupt to the KA88 processor to indicate the locked condition.

- Aurora processor module. The memory in the Aurora processor module sets the lock after the successful access of the first longword being accessed, which can happen before the end of the IRCI transaction. Once set, the lock cannot be cleared if a NO ACK is received. Therefore, under certain circumstances the Aurora processor module sets the lock on unsuccessful IRCI transactions. The receipt of a NO ACK generates an interrupt to the processor on the module and sets the Interlock Error bit in the EV Register on the BCI3 chip (at location 2008 2008, hexadecimal, on the local II bus).

digital™

D.4 SELF-TEST

D.4.1 Performance of Self-Test

VAXBI nodes are required to perform a self-test on power-up and assert the BI BAD L line. When nodes pass their self-test, they must clear their Broke bit, deassert the BI BAD L line, and turn on their yellow LEDs. The following nodes violate the self-test requirements:

- KA88 Memory Interconnect to VAXBI Adapter (DB88). On power-up the DB88 does not perform self-test and never asserts the BI BAD L line.

D.4.2 Use of the VAXBI Bus During Self-Test

During self-test no VAXBI transaction is permitted to have more than 10 stalls, and the average number of stalls permitted is 4 per transaction. The following node does not conform to this requirement:

- UNIBUS to VAXBI Adapter (DWBUA). During one VAXBI transaction performed during self-test, the DWBUA issues up to 111 STALL cycles, which exceeds the limit.

During fast self-test, no VAXBI node is allowed to issue a combined total of more than 512 VAXBI and loopback transactions. The following node does not conform to this requirement:

- VAX 8200 Processor (KA820). During fast self-test, the KA820 may issue up to 8192 loopback transactions.

During normal self-test, no VAXBI node is allowed to issue a combined total of more than 2048 VAXBI and loopback transactions. The following node does not conform to this requirement:

- VAX 8200 Processor (KA820). During normal self-test, the KA820 may issue more than 8192 loopback transactions.

D.4.3 Setting of VAXBI Registers at the End of Self-Test

The following nodes do not comply with the requirements regarding the contents of VAXBI registers at the end of self-test:

- UNIBUS to VAXBI Adapter (DWBUA). The DWBUA does not clear these registers:
  - Interrupt Destination Register (INTRDES)

digital™

- User Interface Interrupt Control Register (UINTRCSR)

o VAXBI Communications Option (DMB32). Some DMB32 modules leave bit 15 (External Vector) set in the User Interface Interrupt Control Register at the end of node self-test. This exception applies to module revisions [] through [].

D.4.4 Assertion of BI NO ARB L by VAXBI Primary Interface

During node reset self-test the BIIC as VAXBI primary interface must not assert BI NO ARB L.

o The Pass 4 BIIC asserts BI NO ARB L.

D.4.5 Deassertion and Reassertion of BI BAD L

On successfully passing self-test, a VAXBI node is required to deassert BI BAD L and clear its BROKE bit. If the node then fails an extended self-test, it can reassert the BROKE bit.

o CI Adapter (CIBCI). In early revisions of the CIBCI (revs. A - C) in one mode in which it fails extended self-test, the CIBCI clears the BROKE bit, but instead of deasserting BI BAD L and then reasserting it, it simply keeps BI BAD L asserted continuously. This behavior has been corrected in later revisions. Early models returned from the field will be repaired.

D.4.6 Self-Test Time Limit

VAXBI nodes are required to complete their node self-test within 10 s of the start of self-test.

o VAXBI memory (MS820-CA). This 16 megabyte memory node takes up to 20 s to complete its self-test.

D.5 RESPONSE TO STOP TRANSACTION

Upon the receipt of a VAXBI STOP transaction, nodes must cease issuing transactions as soon as feasible. Exceptions to this requirement may be granted to nodes that are difficult to design so that they enter a special state on receiving a STOP transaction. Furthermore, for some nodes very little state information may be visible on the VAXBI bus.

digital™

The following nodes are not required to respond to STOP transactions:

- VAX 8200 Processor (KA820)
- KA88 Memory Interconnect to VAXBI Adapter (DB88)
- Aurora processor module

D.6 VAXBI/BIIC PROTOCOL

The user interface must deassert the BCI RQ<1:0> L lines in the same cycle that BCI MAB L is asserted. The following node does not meet this requirement:

- VAX 8200 Processor (KA820). This exception applies to a limited number of Revision A1 modules (serial nos. 1-385, 486-503). In response to a BIIC error-type EV code condition, the KA820 processor does not release the request line in the correct cycle. This behavior causes an IPE error to be produced by the node, which is logged in every node in the system.

The VAXBI primary interface (for example, the BIIC) is required to generate proper parity on the BI P0 L line.

- The Pass Four BIIC at times does not generate proper parity for accesses to three of its internal registers. (See Appendix E for details.)

D.7 RESPONSE TO READ-TYPE TRANSACTIONS

Read-type transactions targeting I/O space locations must not have any side effects. The following node violates this requirement:

- UNIBUS to VAXBI Adapter (DWBUA). Read-type transactions to the node window space of a DWBUA may produce side effects, and therefore the UNIBUS adapter has been granted an exception.

D.8 MECHANICAL REQUIREMENTS

On VAXBI modules the solder side lead projection is 0.062" maximum.

- The MS820-BA (4 Mbyte memory option) and the MS820-BA (16 Mbyte memory option) have components on the solder side of the modules and exceed the requirement. The current solder-side component projection for either module is 0.140".

No modules other than the MS820-BA/CA modules may be placed to the left (higher slot numbers - see Figure 13-7) of MS820-BA/CA modules within a single VAXBI cage. This is necessary to avoid physical interference between MS820-BA/CA modules and adjacent modules, and to avoid exceeding the 50 W per slot limit caused by the sum of the components on the solder-side of the MS820-BA/CA and the components on the component-side of the module to its left.

- The VAXstation 8000 violates the above requirement, and is granted an exception.

D.9 OPERATING REQUIREMENTS

All components and subassemblies of a VAXBI device housed within a VAXBI card cage must operate over the full range of conditions specified below:

- +5V Voltage: 4.75 to 5.25V
- Inlet Temperature: 5 to 50 degrees C
- Humidity: 10 to 95% (with maximum wet bulb 32 C and minimum dew point 2 C)
- Altitude: 0 to 2.4 km
- Air Flow: 200 LFM at any component (min.), 12.8 SCFM per slot (min.)

The following devices have been granted exceptions:

- BI to Disk Adapter (KDB50). A number (serial nos. [ ]) of these options are not specified to operate at 50 degrees C.
- VAXBI Connector, Revision X05B. (H9400 cage serial nos. [ ]) An exception has been granted to the connector; however, it is under discussion whether the connector can be specified to operate at a temperature of 50 degrees C.

D.10 ARBITRATION

Nodes are required to arbitrate in dual round-robin mode.

- UNIBUS to VAXBI Adapter (DWBUA). The DWBUA arbitrates in fixed-high priority mode.

D.11 USE OF RETRY RESPONSE

If a node cannot immediately execute the command sent to it, it returns the RETRY response.

- UNIBUS to VAXBI Adapter (DWBUA). A limited number of DWBUA modules (serial nos. []) respond with RETRY to accesses of unimplemented nodespace registers.

D.12 RESPONSE TO RXCD REGISTER

Nodes that do not implement a VAXBI console must respond to reads to the RXCD location with either a NO ACK response or a longword in which the RXCD Busy 1 bit is set.

- UNIBUS to VAXBI Adapter (DWBUA). A limited number of DWBUA modules (serial nos. []) respond with RETRY to accesses of the RXCD Register.

D.13 RESPONSE TO IDENT TRANSACTION

BI D<31:14> L and BI D<1:0> L must be zeros at the time the vector is sent.

- UNIBUS to VAXBI Adapter (DWBUA). During self-test the DWBUA sends vectors with nonzero fields.

D<31:20> and D<15:0> are RESERVED fields during an IDENT C/A cycle.

- BI Communications Adapter (DHB32). The DHB32 interprets bit <31> of IDENT C/A cycles and may NO ACK if bit <31> is asserted.

- BI Communications Adapter (DMB32). The DMB32 interprets bit <31> of IDENT C/A cycles and may NO ACK if bit <31> is asserted.

digital™

D.14 BIIC MUST ALLOW DISABLING OF STO

When bit <13> of the Device Register is set, the BIIC does not terminate a transaction even though the node receives more than 127 STALLs (128 STALLs is the required timout period). When bit <13> is set, the BIIC may continue the assertion of BI BSY L beyond the interval of 127 STALLs. Also, since there is no timeout, the STO bit in the Bus Error Register does not set and the STO EV code does not occur.

- BIICs previous to Pass 5 and Pass 5 BIICs with lot numbers < [] do not allow the disabling of the Stall Timeout on Slave Transaction. The STO behavior of these BIICs is independent of the state of Device Register bit <13>.

D.15 OPERATION WITH BIIC DTYPE BIT <13> SET

Operation of the BIIC with DTYPE bit <13> set (STO disabled) is RESERVED to Digital and requires an exception.

- XMI to VAXBI Adapter (XBI). The XBI operates with DTYPE bit <13> set to solve an inter-bus memory read latency problem.

APPENDIX E

PASS FOUR BIIC

Product: DC324-DA Date: 31-DEC-1985
Author: VAXBI Development Group
Description: This bulletin pertains to all Pass Four BIICs.

The Pass Four 78732 BIIC component deviates from the specification requirements in two respects:

- A design enhancement to the node reset function that is now part of the VAXBI specification is not incorporated in Pass Four BIICs.
- A design error is exhibited at times at the system level in which a parity error occurs during certain bus traffic sequences.

Both items have been corrected in Pass Five of the BIIC. They are discussed below in detail, along with their system implications.

E.1 NODE RESET DURING SELF-TEST

According to VAXBI requirements, during a node reset self-test the VAXBI primary interface (the BIIC) must not assert BI NO ARB L. The Pass Four BIIC does, however, hold BI NO ARB L asserted for the duration of its node reset self-test (.82 milliseconds). (The Pass Four BIIC properly asserts BI NO ARB L during power-up self-test.)

A consequence of this is that all other nodes in the VAXBI system are precluded from arbitrating while a node is being reset. This causes:

- An increase in bus access latency for all nodes
- An increase in interrupt service latency in the system
- Possible bus timeouts on VAXBI nodes as well as on devices supported on other buses through VAXBI adapters

digital™

Nodes that attempt VAXBI transactions during a node reset sequence may get a bus timeout condition. Node designers should ignore the bus timeout error. Operating system designers should ignore, other than for logging purposes, BTO interrupts that occur as a result of a node reset sequence.

The revised (Pass Five) BIIC does not assert BI NO ARB L during its node reset sequence so that the above precautions will not be necessary.

E.2 PARITY AND BIIC REGISTERS

The VAXBI primary interface is required to generate proper parity on the BI P0 L line. At times the Pass Four BIIC does not generate proper parity for accesses to three of its internal registers. In certain bus traffic situations this problem can cause parity errors to occur on read-type transactions that access these BIIC registers. The design error on-chip is the result of a timing error in parity computation on register contents while some of the register bits are changing. The registers involved are:

- o User Interface Interrupt Control Register (UINTRCSR)
- o Error Interrupt Control Register (EINTRCSR)
- o Bus Error Register (BER)

CPU Error Recovery

A general method to prevent CPU crashes due to the parity error (or any type of bus "glitch") is to have the CPU reattempt a transaction that fails. (This is because the type of BIIC problem is similar in nature to transient parity errors: inherently probabilistic events and bus traffic dependent.) The error will still be logged in the master and slave BIICs and an error interrupt generated.

Presented below are scenarios in which the parity error might appear. For each sequence suggestions are made of how to avoid the error. The three interrupt error sequences apply to the User Interface Interrupt Control Register. No examples are given for the Error Interrupt Control Register. However, scenarios similar to sequences 1 and 2 can also occur with the EINTRCSR, since it also contains an INTR Complete bit. Sequence 3 does not apply, since this register is not affected by the BCI INT<7:4> L lines.
In the scenarios it is assumed that, in general, I/O devices do not read the internal registers of other I/O devices.

E.2.1 Interrupt Error Sequences

Interrupt Error Sequence No. 1

In one scenario (see Figure E-1), a node that is slave to the IDENT command (issued by the CPU in the system) has a master port interface that is attempting a read-type transaction to access the User Interface Interrupt Control Register while the IDENT transaction is proceeding.

NOTE

The error can occur with either VAXBI or loopback read-type transactions.

A parity error occurs if the current slave node wins the arbitration to become the pending master for the IDENT transaction. The BIIC produces an error in this case because the INTR Complete bit within the BIIC's UINTRCSR is being set internally while a parity calculation is being done for the data being transmitted on the bus for the read command.

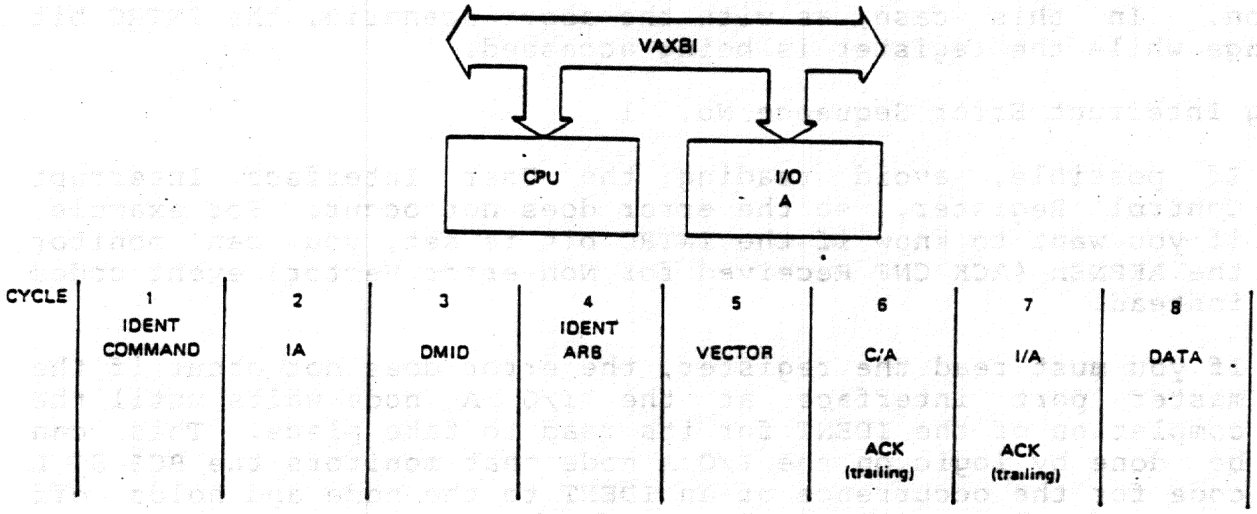


Figure E-1: Interrupt Error Sequence No. 1

Cycle 1 IDENT command from CPU is on the bus.

Cycle 2 During imbedded ARB cycle, I/O A node arbitrates and wins for read command to follow. I/O A then becomes pending master.

Cycle 6 I/O A becomes master for read during the first trailing ACK of IDENT transaction.

digital™

Cycle 7 In I/O A's BIIC, INTR Complete bit is set, since second trailing ACK indicated that the IDENT completed successfully. (See Chapter 7, Section 7.14). Parity computation for new register contents begins this cycle.

Cycle 8 Read data with register contents indicating INTRC bit is set is sent on the bus with bad parity to I/O A since the BIIC's internal parity computation extends into this cycle (when transceiver latch is still open).

As a result of this parity error:

- If the HEIE bit is set in I/O A's VAXBI Control and Status Register, an interrupt is sent out.
- TDF, ICE, and MPE bits are set in I/O A's Bus Error Register.

Note these are errors produced by the BIIC itself.

This type of error can also happen to a node that tries to access the Error Interrupt Control Register. Adapters that use the force bit in this register to send out an interrupt can also have the failure condition. In this case, as with the above scenario, the INTRC bit can change while the register is being accessed.

Avoiding Interrupt Error Sequence No. 1

- If possible, avoid reading the User Interface Interrupt Control Register, so the error does not occur. For example, if you want to know if the INTRC bit is set, you can monitor the AKRNEn (ACK CNF Received for Non-error Vector) event codes instead.
- If you must read the register, the error does not occur if the master port interface at the I/O A node waits until the completion of the IDENT for its read to take place. This can be done by logic on the I/O A node that monitors the BCI SC L code for the occurrence of an IDENT to the node and holds off the master port interface's transaction request.

Interrupt Error Sequence No. 2

This error sequence is similar to sequence number 1, except that the system has three nodes rather than two (see Figure E-2). In this scenario, the third node (CPU B) attempts a read of I/O A's internal UINTRCSR during the IDENT transaction from CPU A. In this case, CPU B gets the parity error while trying to read I/O A's register during CPU A's IDENT transaction.

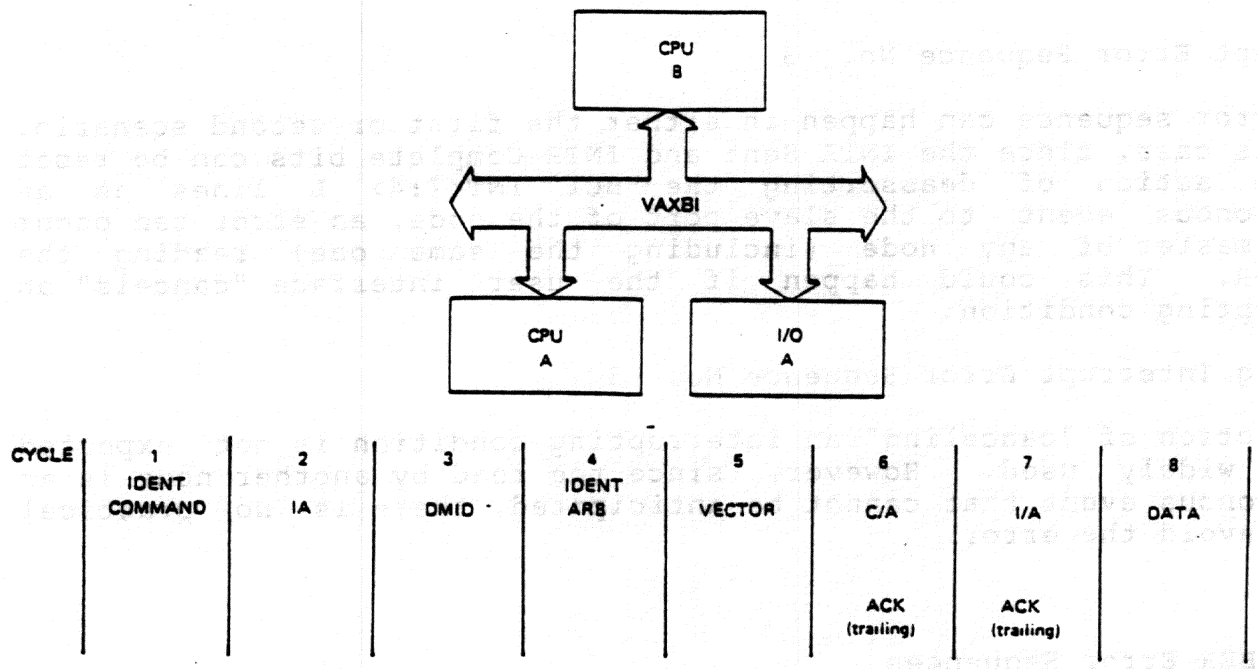


Figure E-2: Interrupt Error Sequence No. 2

This sequence is similar to sequence 1, except in cycles 2 and 6.

Cycle 2 CPU B arbitrates and wins.

Cycle 6 C/A cycle is CPU B's.

As a result of this parity error:

- If the HEIE bit is set in CPU B's VAXBI Control and Status Register, an interrupt is sent out.
- MPE is set in CPU B's Bus Error Register
- TDF and ICE bits are set in I/O A's Bus Error Register.

Avoiding Interrupt Error Sequence No. 2

The third node's (CPU B's) read is essentially an asychronous event to the IDENT of CPU A. For this reason, if a read to I/O A's UINTRCSR is necessary, it may be very hard to avoid the error. In an asymmetric multiprocessing system, this scenario is unlikely to occur. In a symmetric multiprocessing environment, where another processor could likely read the I/O device's register, the parity error may appear. However, the error is still not likely to appear, since the BIIC register that needs to be read to get the error (UINTRCSR) contains information rarely needed by a processor after initial setup.

Interrupt Error Sequence No. 3

This error sequence can happen in either the first or second scenario. In this case, since the INTR Sent and INTR Complete bits can be reset by the action of deasserting the BCI INT<7:4> L lines as an asynchronous event to the slave port of the node, an error can occur to the master of any node (including the same one) reading the UINTRCSR. This could happen if the user interface "cancels" an interrupting condition.

Avoiding Interrupt Error Sequence No. 3

The function of "canceling" an interrupting condition is not expected to be widely used. However, since the read by another node is an asynchronous event that cannot be anticipated, there is no practical way to avoid the error.

E.2.2 BER Error Sequences

Three errors can occur as secondary errors to actual bus errors. The BER error scenarios show that when a bus error occurs during a transaction, as the bit that logs the error changes in the BER, there is a time when there could be a bus parity error being produced by the BIIC. This would happen if a pending master to the failed transaction does a read-type transaction to the BER. The probability of seeing these errors in a working system is small, but not zero in those systems whose nodes read the BER. The chance of getting these errors is the product of the "independent" probabilities of "getting a bus error" X "a node becoming pending master in a transaction that has a bus error" X "the transaction subsequent to the errored transaction being a read-type transaction to the BER." Obviously, the product is much less than 1, but the problem is exacerbated by nodes that do reads of the BER as a result of bus errors. In this case, chances that secondary errors will be flagged are greater (MPE and ICE bits) along with the original error in the BER.

An additional note on BER read failures is that since a master cannot arbitrate in its own imbedded ARB cycle, these failures cannot happen on an individual node basis.

Asymmetric "master/slave" multiprocessing systems on the VAXBI bus are not likely to get this error, since only one processor typically handles I/O interrupt servicing, and it is only that processor that might be reading the BER as a result of the original error interrupt.

However, in a symmetric operating system where processors may vie to service a node's I/O interrupt, the problem is more likely to be seen. This is because the second (or third) processor may have software that asychronously reads the BER of the I/O device while the other

digital™

processor(s) may be getting the failed transaction of the error scenario.

System software cannot tell from the status of the BER whether a bus parity error occurred due to a "glitch" on the VAXBI bus or whether the BIIC Pass Four parity error occurred.

BER Error Sequence No. 1

Node A issues an IDENT to node B. The IDENT fails due to an IDENT Vector Error (IVE). Node C wins the imbedded ARB of node A's IDENT and issues a transaction that reads the BER of node B. This transaction will fail due to the parity error of the BIIC.

In summary:

Sequence	BER Bits Set
Node A issues an IDENT to node B Node C reads node B's BER	Node A - MPE (if vector failed due to bus parity error) Node B - IVE, ICE, TDF Node C - MPE

An IVE condition alone (without this parity failure) would have caused only the TDF, ICE, and IVE bits to be set.

BER Error Sequence No. 2

Node A gets a bus timeout (BTO) while attempting a transaction to some other node (Y). Node B can successfully issue a read-type transaction to node A's BER at the time the BTO condition occurs. Node B's read of node A's BER will fail due to the parity error of the BIIC.

In summary:

Sequence	BER Bits Set
Node A tries to issue a transaction to node Y (BTO) Node B reads node A's BER	Node A - BTO, TDF, ICE Node B - MPE

BER Error Sequence No. 3

Node A issues a transaction to node B. Node B's slave port detects an Illegal Confirmation Error (ICE) during this transaction. Node C wins the imbedded ARB of the transaction and issues a transaction that reads node B's BER. This transaction will fail due to the parity error of the BIIC.

digital™

In summary:

Sequence	BER Bits Set
Node A issues a transaction to node B (ICE)	Node B - ICE, TDF
Node C read's node B's BER	Node C - MPE

digital™

APPENDIX F

ARCHITECTURE MANAGEMENT

This appendix describes the architecture-management process for specifications managed by the Systems Architecture group, and has two purposes. The first is to point out responsibilities for adherence to, interpretation of, and changes to architecture specifications. The second is to define a formal ECO (engineering change order) process that includes a careful review of the impact that a change or exception has on existing and planned products based on the architecture. The goal is to promote hardware and software product compatibility.

The process described here applies to all specifications managed by the Systems Architecture group. The final section lists these specifications, as well as the current maintainer (called "architect" in the rest of this document) for each specification.

F.1 RESPONSIBILITIES

Ensuring product compatibility takes effort on the part of managers, engineers, the architecture group, and architecture-review-group representatives. Their responsibilities in the process are described here.

F.1.1 Project, Product, and Engineering Managers

For each architecture specification that applies to a product, managers of the hardware and software design and support organizations must:

- Ensure that their engineers are aware of the existence of the architecture specification and the long-term adverse consequences to Digital of deviations from it.

digital™

- Designate a representative to vote on architecture issues and to transmit information about the architecture between the organization and the architecture review group (ARG).
- Develop and successfully carry out a plan to demonstrate conformance of their implementation to the architecture specification.
- Provide resources to fix, and a schedule to release fixes for, all nonwaived deviations from the architecture.

F.1.2 Hardware, Microcode, and Software Engineers

Hardware, microcode, and software engineers must be familiar with, and implement designs that conform to, the architecture specification. Hardware and microcode engineers must ensure that ANY software that conforms will execute correctly with their hardware. Similarly, software engineers must ensure that ANY hardware that conforms will work correctly with their software. The software must not depend on any hardware that handles unpredictable or undefined architecture features. All nonconformities must be fixed by product support people or waived by the Systems Architecture group using the ECO process.

F.1.3 Systems Architecture Group and Architects

The Systems Architecture group (and in particular the architect of each of the specifications the group maintains) interprets and develops the specifications, as well as manages the process for review and approval of ECOs to them. More specifically, the group:

- Interprets and clarifies the architecture specifications. This may result in editorial changes that in most cases will not require a formal ECO proposal and vote. The architect publishes and distributes such changes in the same way that approved ECO proposals are made known.
- Decides on the organizations to be represented in architecture review groups, with the goal of including all technical views pertaining to the architecture. The architect asks organization management to choose representatives and considers requests from organizations to participate.
- Processes ECO proposals, including the gathering of all appropriate comment, building consensus, forming the final ECO for inclusion in a specification, collecting votes, and achieving closure on the proposal.

digital™

- Processes exceptions to the architecture specifications. These waiver requests are processed as ECO proposals.

- Archives and disseminates the specifications. This includes keeping the sources, incorporating and distributing changes, maintaining change history data, and updating distribution lists.

- Monitors for compliance and develops architecture verification tools for compliance monitoring.

- Maintains and publishes the lists of known architecture discrepancies.

F.1.4 Architecture-Review-Group Representatives

An ARG representative serves as an organization's primary contact with the architecture group. ARG members are responsible for:

- Circulating ECO proposals and results, and other information sent to them concerning the architecture, to all projects in the organization they represent.

- Collecting and returning all comments and their vote on ECO proposals to the architect by the deadline stated in each proposal, unless an extension is agreed to by the architect.

- Notifying the architect of the latest functional and design specifications and reviews for implementations within the organization they represent.

- Reporting and helping to resolve discrepancies between implementations within their organization and the architecture.

F.2 SPECIFICATION DISTRIBUTION

Due to the confidential nature of architecture specifications, the Systems Architecture group will follow these guidelines when distributing copies:

- Requests from non-Digital employees, whether or not non-disclosure agreements have been signed, will be reviewed on a case-by-case basis by the Systems Architecture manager and possibly others in the management chain. Only hard copy will be sent, and the architect is responsible for keeping a

digital™

list of where and to whom copies were sent.

- Information-only requests from Digital employees will be answered by providing a hard copy of the specification and information on how to find out about updates. Copies will be sent to Digital mailstops only, and the architect is responsible for keeping a list of where and to whom copies were sent.

- Requests from Digital employees who are members of the architecture review group, or who are working on a project with close ties to the specification, will be answered by providing hard or soft copy at their preference.

Requested copies of architecture specifications must not be distributed further.

F.3 CONFORMANCE AND WAIVERS

Any plan to test implementation conformance to an architecture specification must include the following criteria:

- Correct operation with current existing implementations, based on product requirements for coexistence of, and cooperation between, the current and new implementations.

- Passing any established verification procedure that is specified by the Systems Architecture group.

The group responsible for each implementation is required to develop a plan to demonstrate conformance to the architecture. This verification plan must be reviewed and approved by the Systems Architecture group, and then successfully carried out on the version of the product planned for first customer ship. For each version of the product released after first customer ship, this verification plan must be carried out successfully as well. Those assigned to do the testing must notify the Systems Architecture group in writing when this work has been completed, including details on what was tested (including versions) and the results.

When any discrepancy between a design or implementation and the architecture is discovered, the group's ARG member must bring the issue to the attention of the architect before any hardware or software design or implementation decisions are committed. Each implementation must conform to the architecture specification unless it has been granted a waiver by the ECO process. A waiver may allow an immediate shipment followed by a fix. It may require some instances to be fixed and not others, or it may allow a permanent exception. No "system killer" will ever be given more than a temporary waiver with a very short duration. Digital makes fixes for

non-waived deviations available to customers. To minimize the need for ECOs and waivers, the group's ARG member should notify the architect of the existence of, and reviews for, functional and design specifications as early as possible. The architect, or someone working for the architect and not a member of the design or support team, can work to clarify and resolve issues before they require formal ECO resolution.

F.3.1 Publishing Architecture Discrepancies

The Systems Architecture group maintains the lists of known architecture discrepancies. These lists are made available to ARG members for distribution within their organizations. The lists are submitted to publications that are readily accessible to customers if the discrepancies are visible to customers. All discrepancies that have been waived will be published in the architecture specification, either as a note in an appropriate section, or in an appendix for waivers.

The single exception is that we will not publish the list of "system killers" outside of Digital. All questions about "system killers", even ones asking if there are any, will be answered "No Comment". The reason for this is to protect customers (from their users) by providing absolutely no information on the subject. In addition, this "no comment" policy will be published along with the lists of architecture discrepancies.

F.4 ECO PROCESS

The ECO process is biased against change. Architecture changes or exceptions that cause incompatible differences between hardware implementations, or cause software not to run on any hardware that conforms to the architecture, require compelling reasons.

F.4.1 Participants in the ECO Process

The following people and groups are participants in the architecture review process:

- The architect, who manages the review process for the specification. The architect reports to the Systems Architecture manager in this capacity.

- The ARG consisting of the architect, former major contributors to the specification, and representatives of software and hardware engineering groups that are strongly dependent on the architecture specification. Managers of the engineering groups select these representatives.
- The manager of the Systems Architecture group, who participates in ECO approval and appeal.
- Other reviewers, as deemed appropriate by ECO process participants.

F.4.2 ECO Proposals

Proposers of an ECO must submit an ECO proposal in writing to the architect for the specification. Proposing an ECO includes whatever preliminary investigation is necessary to achieve a sound technical proposal. The authors must detail the rationale and known consequences. Among these are:

- Function changes.
- The performance gain (or loss).
- Any effects on existing and planned implementations.
- The plan to change prior implementations, including estimated costs, or the plan to deal with incompatibilities if prior implementations are not changed.
- A hardware and software transition plan.
- A description and cost estimate of any other hardware, software, or economic impact.

An ECO proposal does not always require all of the above information. ARG members must provide information on the impact of the proposed change for the projects they represent as part of the ECO review process.

Once submitted, the architect works with the originators of a proposal to:

- Ensure there is information in the proposal about who to contact for more details on the issue.
- Include alternative solutions, with their pros and cons, in the proposal.

digital™

- Present the proposed alternative in final form, and in the context of the architecture specification. This includes exact wording changes and additions.

F.4.3 ECO Proposal Review

The architect logs, assigns a number, and mails an ECO proposal to the following reviewers for comments and their vote by the stated deadline:

- Members of the ARG, on whom the technical integrity and continuity of the architecture depends.
- Others directly involved in the technical issue.
- Whoever else is deemed appropriate by reviewers.

The architect keeps the list of ARG members and the complete distribution list for any ECO proposal.

The reviewers are responsible for circulating the ECO proposal within the organization they represent, collecting the comments, and returning the comments in writing to the architect by the stated deadline for comments and voting. The architect collects the comments and makes them available to other reviewers. This may result in individual discussions, meetings, and revisions to the ECO proposal. Any non-editoral revision of an ECO proposal must be sent to all reviewers for comments and voting again.

There are ECO proposals that contain sensitive information. As a security precaution, all notes conferences that are used for discussion of ECO proposals are private. Only ARG members, or individuals sponsored by ARG members, are given access to these conferences.

The reviewers must submit their vote (YES, NO, or ABSTAIN) in writing to the architect by the deadline stated in a proposal for comments and voting. ARG members voting for a proposal are asked to explain the impact on their organization and how it would be dealt with. ARG members voting against a proposal are asked to explain their objections and suggest alternatives to the proposal that would be acceptable. The architect tabulates the votes, and attempts to build a consensus based on sound judgment, not just expedient compromise.

F.4.4 ECO Approval And Appeal

To pass, an ECO must have the approval of the Systems Architecture manager and the consensus of the reviewers. The architect determines if a consensus exists using these guidelines:

- A quorum consisting of at least 80% of the reviewers must submit a YES or NO vote. Votes to ABSTAIN or no response from more than 20% of the reviewers requires follow-up before a consensus can be determined.
- At least 75% of the reviewers in the quorum must vote YES.
- Any strong opposition is an indication of no consensus.

If an ECO proposal is about to pass and there were NO votes, those casting NO votes will be given an opportunity to accept or reject the consensus.

An architect's decision about an ECO proposal can be appealed to the Systems Architecture manager and further up the management chain if necessary. In an appeal, management works to find a compromise that is acceptable to the architect, the author(s) of the ECO proposal, those objecting to the architect's decision, and all reviewers.

F.4.5 Publishing And Distribution of ECO Results

The architect notifies the ARG about the decision on an ECO proposal in a timely manner. Background data on this decision, including the text of votes submitted by reviewers, is available to ARG members upon request. When approved, the architect incorporates the ECO into the specification with change bars where appropriate and possible, and makes change pages available to ARG members. ARG members are responsible for making known the results to the projects they represent.

F.4.6 Specification Retirement

The architecture specification will be considered retired when a proposal that it be abandoned meets with the same approval as required for ECO proposals and exceptions.

digital™

F.5 SYSTEMS-ARCHITECTURE-GROUP SPECIFICATIONS

Document	Architect
DEC Standard 032 (VAX Architecture Standard)	Tim Leonard
DEC Standard 057 (VAXBI Standard)	Rich Best
XMI Standard	Bob Chen
BI VAX Port Architecture	Tom Kong
VAX CI Port Architecture	Dave Dagg
System Communication Architecture	Darrell Duffy

Last Update: May, 1987

digital™

Revision History:

Revision 1. C. Wiecek, 13 February 1986.
- o Version for approval

Revision 2. C. Wiecek, 27 March 1986.
- o Added one arch. group responsibility
- o Put ARG responsibilities together in a section
- o Made clarifications to implementation conformance, publishing system killers, ECO proposal review, determining consensus, and publishing ECO results

Revision 3. C. Wiecek, 14 April 1986.
- o Process put into effect
- o Minor edits and clarification to ECO appeal

Revision 4. C. Wiecek, 30 September 1986.
- o Added notes-conference access policy in ECO Proposal Review section
- o Named new CI VAX Port architect
- o Minor edits

Revision 5. C. Wiecek, 26 May 1987.
- o Loosened wording on fixes for non-waived deviations made by Digital to more closely match current policy
- o Updated Systems Architecture Group list of specifications
- o Changed document format to an appendix

APPENDIX G

DEVICE TYPE CODE ASSIGNMENT PROCEDURES

The Systems Architecture Group maintains the list of device type codes. Upon request, the Systems Architecture Group will assign an appropriate device type code to a new Digital-supplied node. When requested to do so by the OEM/Channels Technical Support Group, the Systems Architecture Group also assigns device type codes for nodes designed by VAXBI licensees, as described below.

All licensee communications regarding VAXBI device type code assignments should be routed through the OEM/Channels Technical Support Group.

- Acquiring a Temporary Device Type Code

  Licensees should contact the OEM/Channels Technical Support Group and request a device type code assignment. VAXBI licensees are advised to defer requesting of the device type code until late in the development process. As part of the registration process, licensees will be asked to provide the following information about their node:

  - A node name
  - A one-line description of the node's capability

  The OEM/Channels Technical Support Group will request an assignment from the Systems Architecture Group. The Systems Architecture Group will assign a temporary device type code, which will be valid for one year from the date of issue.

- Making the Device Type Code Permanent

  The temporary assignment becomes permanent if the licensee ships the product within one year of the issue date. The licensee must notify the OEM/Channels Technical Support Group when the node ships to ensure that the device type code assignment is marked as permanent.

- Extension of Temporary Device Type Code Assignment

  If at the end of a year's time the licensee has not yet sent the device to market, the licensee may request an extension of the assignment. Extensions can be granted from year to year as long as the OEM/Channels Technical Support Group determines there is merit in continuing the assignment.

- Lapse of Temporary Device Type Code Assignment

  If an extension is not requested, the temporary device type code assignment lapses and the licensee must subsequently request a new device type code if they still need a device type code.

digital™

GLOSSARY

This glossary defines terms used to describe the VAXBI bus and the BIIC.

ACK data cycle -- A data cycle of a read- or write-type transaction during which the slave asserts the ACK CNF code to acknowledge that no error has been detected and that the cycle is not to be stalled.

adapter -- A node that interfaces other buses, communication lines, or peripheral devices to the VAXBI bus.

arbitration cycle -- A cycle during which nodes arbitrate for control of the VAXBI bus.

assert -- To cause a signal to take the "true" or asserted state.

asserted -- To be in the "true" state.

assertion -- The transition of a signal from deasserted to asserted.

assignable window -- A contiguous range of I/O addresses within assignable window space starting on a naturally aligned 1 megabyte address boundary and having a length that is a multiple of 1 megabyte. An assignable window can be allocated to a particular node ID by the operating system when it configures the hardware.

assignable window space -- A region in I/O space from address 2080 0000 through 21FF FFFF (hex). A range in this region can be allocated to a particular node ID for uses similar to those of node window space. See also assignable window.

atomic -- Pertaining to an indivisible operation.

bandwidth -- The data transfer rate measured in information units transferred per unit of time (for example, megabytes per second). All bandwidth figures quoted in this manual take into account command/address and imbedded ARB cycle overhead.

BCI -- Bus chip interface; a synchronous interface bus that provides for all communication between the BIIC and the user interface.

BIIC -- Bus interconnect interface chip; a chip that serves as a general purpose interface to the VAXBI bus.

BIIC CSR space -- The first 256 bytes of the 8-Kbyte nodespace, which

is allocated to the BIIC's internal registers. See also nodespace.

BIIC-generated request -- A transaction request generated by the BIIC rather than by the user interface. The BIIC can request only error interrupts.

BIIC-generated transaction -- A transaction performed solely by the BIIC with no assistance from the master port interface. Only INTR and IPINTR transactions can be independently generated by the BIIC. The user interface initiates BIIC-generated transactions by using the IPINTR/STOP force bit, the user interface or error interrupt force bits, or by asserting one of the BCI INT<7:4> L lines. A BIIC-generated transaction can also result from a BIIC-generated request, which results from a bus error that sets a bit in the Bus Error Register.

bus access latency -- The delay from the time a node desires to perform a transaction on the VAXBI bus until it becomes master.

bus adapter -- A node that interfaces the VAXBI bus to another bus.

busy extension cycle -- A bus cycle during which a VAXBI node, not necessarily the master or the slave of a transaction, asserts the BI BSY L line to delay the start of the next transaction.

command/address cycle -- The first cycle of a VAXBI transaction. The information transmitted in this cycle is used to determine slave selection. In some cases the data on the BI D<31:0> L lines is not an actual address, but it serves the same purpose: to select the desired slave node(s). For example, during an INTR command a destination mask is used.

command confirmation -- The response sent by the slave(s) to the bus master to confirm participation in the transaction.

command confirmation cycle -- The third cycle of a VAXBI transaction during which slave(s) confirm participation in the transaction.

configuration data -- Data loaded into the BIIC on power-up that includes the device type and revision code, the parity mode, and the node ID.

cycle -- The basic bus cycle of 200 nanoseconds (nominal), which is the time it takes to transfer the smallest piece of information on the VAXBI bus. A cycle begins at the leading edge of T0/50 and continues until the leading edge of the next T0/50.

data cycle -- A cycle in which the VAXBI data path is dedicated to transferring data (such as read or write data, as opposed to command/address or arbitration information) between the master and slave(s). During read STALL data cycles, the BI D<31:0> L and I<3:0>

L lines contain undefined data. See also ACK data cycle, read data cycle, STALL data cycle, vector data cycle, and write data cycle.

data transfer transactions -- VAXBI transactions that involve the transfer of data as well as command/address information: read-type, write-type, IDENT, and BDCST transactions.

deassert -- To cause a signal to be in the "false" or deasserted state.

deasserted -- To be in the "false" state.

deassertion -- The transition of a signal from asserted to deasserted.

decoded ID -- The node ID expressed as a single bit in a 16-bit field.

device -- A VAXBI device includes any VAXBI modules, cabling, and attached peripheral equipment necessary to form a functional subsystem. Compare module, node.

device type -- A 16-bit code that identifies the node type. This code is contained in the BIIC's Device Register.

direct memory access (DMA) adapter -- An adapter that directly performs block transfers of data to and from memory.

encoded ID -- The node ID expressed as a 4-bit binary number. The encoded ID is used for the master's ID transmitted during an imbedded ARB cycle.

even parity -- The parity line is asserted if the number of asserted lines in the data field is an odd number.

expansion module -- A VAXBI module that does not attach directly to the VAXBI bus. A VAXBI node that requires more than one module has one module that attaches directly to the VAXBI (that is, contains a BIIC) and one or more expansion modules that communicate with the first module over the user-defined I/O section.

extension cycle -- A bus cycle during which a VAXBI transaction is "extended." See also STALL data cycle, busy extension cycle, and loopback extension cycle.

H -- Designates a high-voltage logic level (that is, the logic level closest to Vcc). Contrast with L.

IDENT arbitration cycle -- The fourth cycle of an IDENT transaction during which nodes arbitrate to determine which is to send the vector.

illegal confirmation code -- A CNF code that is not permitted in a particular VAXBI cycle (such as a RETRY command confirmation to a

multi-responder command).

imbedded arbitration cycle -- An arbitration cycle that occurs (is imbedded) in a VAXBI transaction.

interlock commands -- The two commands, IRCI (Interlock Read with Cache Intent) and UWMCI (Unlock Write Mask with Cache Intent), that are used to implement atomic read-modify-write operations.

internode transfer -- A VAXBI transaction in which the master and slave(s) are in different VAXBI nodes. Contrast with intranode transfer.

interrupt port -- Those BCI signals that are used in generating INTR transactions.

interrupt port interface -- That portion of user logic used to interface to the interrupt port of the BIIC.

interrupt sublevel priority -- Interrupt priority information used during an IDENT transaction to determine which node with a pending interrupt is to provide the vector. The interrupt sublevel priority corresponds to the node ID.

interrupt vector -- In VAX/VMS systems, an unsigned binary number used as an offset into the system control block. The system control block entry pointed to by the VAXBI interrupt vector contains the starting address of an interrupt handling routine. (The system control block is defined in the VAX-11 Architecture Reference Manual.)

intranode transfer -- A transaction in which the master and slave are in the same node. Loopback transactions are intranode transfers. Contrast with internode transfer.

L -- Designates a low-voltage logic level (that is, the logic level closest to ground). Contrast with H.

latency -- See bus access latency.

local memory -- VAXBI memory that can be accessed without using VAXBI transactions; for example, VAXBI-accessible memory on a single board computer.

loopback extension cycle -- A cycle of a loopback transaction during which a node asserts both BI BSY L and BI NO ARB L to delay the start of the next transaction.

loopback request -- A request from the master port interface asserted on the BCI RQ<1:0> L lines which permits intranode transfers to be performed without using the VAXBI bus.

loopback transaction -- A transaction in which information is transferred within a given node without use of the VAXBI data path. Contrast with VAXBI transaction.

mapped adapter -- A DMA adapter that performs data transfers between a system with a contiguous memory space and VAXBI address space (in which memory need not be contiguous). The mapping is done by using a set of map registers located in the adapter.

master -- The node that gains control of the VAXBI bus and initiates a VAXBI or loopback transaction. See also pending master.

master port -- Those BCI signals used to generate VAXBI or loopback transactions.

master port interface -- That portion of user logic that interfaces to the master port of the BIIC.

master port request -- A request (either VAXBI or loopback) generated by the master port interface through the use of the BCI RQ<1:0> L lines.

master port transaction --Any transaction initiated as a result of a master port request.

module -- A single VAXBI card that attaches to a single VAXBI connector. Contrast with node.

multi-responder commands -- VAXBI commands that allow for more than one responder. These include the INTR, IPINTR, STOP, INVAL, and BDCST commands.

node -- A VAXBI interface that occupies one of sixteen logical locations on a VAXBI bus. A VAXBI node consists of one or more VAXBI modules. Contrast with module.

node ID -- A number that identifies a VAXBI node. The source of the node ID is an ID plug attached to the backplane.

node reset -- A sequence that causes an individual node to be initialized; initiated by the setting of the Node Reset bit in the VAXBI Control and Status Register.

node window -- A unique 256-Kbyte contiguous range of I/O addresses within node window space that is allocated to a particular node ID. The starting address of node window k is 2040 0000 + k*4 0000 (hex). Node windows are typically used to map VAXBI transactions onto a target bus.

node window space -- A region in I/O space ranging from address 2040 0000 through 207F FFFF (hex). One of sixteen aligned (on 256

Kbyte boundary) subranges within node window space (called "node windows") is permanently allocated to each node by node ID. See node window.

nodespace -- An 8-Kbyte block of I/O addresses that is allocated to each node. Each node has a unique nodespace based on its node ID.

null cycle -- A cycle in which all VAXBI lines are deasserted (that is, no transaction or arbitration is taking place).

odd parity -- The parity line is asserted if the number of asserted lines in the data field is an even number.

parity mode -- Specifies whether parity is generated by the BIIC or by the user interface.

pending master -- A node that has won an arbitration but which has not yet begun a transaction.

pending request -- A request of any type, whether from the master port or a BIIC-generated request, that has not yet resulted in a transaction.

pipeline request -- A request from the master port that is asserted prior to the deassertion of BCI RAK L for the present master port transaction; that is, a new request is posted prior to the completion of the previous transaction.

power-down/power-up sequence -- The sequencing of the BI AC LO L and BI DC LO L lines upon the loss and restoration of power to a VAXBI system. See also system reset.

private memory -- Memory that cannot be accessed from the VAXBI bus.

programmed I/O (PIO) adapter -- An adapter that does not access memory on the VAXBI bus but interacts only with a host processor.

RCLK (receive clock) -- The clock phase during which information is received from the VAXBI bus; equivalent to T100/150, as shown in Figure 12-1.

read data cycle -- A data cycle in which data is transmitted from a slave to a master.

read-type commands -- Any of the various VAXBI read commands, including READ, RCI (Read with Cache Intent), and IRCI (Interlock Read with Cache Intent).

RESERVED code -- A code reserved for use by DIGITAL.

RESERVED field -- A field reserved for use by DIGITAL. The node

driving the bus must ensure that all VAXBI lines in the RESERVED field are deasserted. Nodes receiving VAXBI data must ignore RESERVED field information. This requirement provides for adding functionality to future VAXBI node designs without affecting compatibility with present designs. Example: The BI D<31:0> L and BI I<3:0> L lines during the third cycle of an INTR transaction are RESERVED fields.

reset module -- In a VAXBI system, the logic that monitors the BI RESET L line and any battery backup voltages and that initiates the system reset sequence.

resetting node -- The node that asserts the BI RESET L line.

retry state -- A state that the BIIC enters upon receipt of a RETRY confirmation code from a slave. If the master reasserts the transaction request, the BIIC resends the transaction without having the user interface provide the transaction information again. The command/address information and the first data longword, if a write transaction, are stored in buffers in the BIIC.

single-responder commands -- VAXBI commands that allow for only one responder. These include read- and write-type commands and the IDENT command. Although multiple nodes can be selected by an IDENT, only one returns a vector.

slave -- A node that responds to a transaction initiated by a node that has gained control of the VAXBI bus (the master).

slave port -- Those BCI signals used to respond to VAXBI and loopback transactions.

slave port interface -- That portion of user logic that interfaces to the slave port of the BIIC.

STALL data cycle -- A data cycle of a read- or write-type transaction during which the slave asserts the STALL CNF code to delay the transmission of the next data word.

system reset -- An emulation of the power-down/power-up sequence that causes all nodes to initialize themselves; initiated by the assertion of the BI RESET L line.

target bus -- The bus that a VAXBI node interfaces to the VAXBI bus.

TCLK (transmit clock) -- The clock phase during which information is transmitted on the VAXBI bus; equivalent to T0/50, as shown in Figure 12-1.

transaction -- The execution of a VAXBI command. The term "transaction" includes both VAXBI and loopback transactions.

UNDEFINED field -- A field that must be ignored by the receiving node(s). There are no restrictions on the data pattern for the node driving the VAXBI bus. Example: The BI D<31:0> L and BI I<3:0> L lines during read STALL data cycles and vector STALL data cycles are UNDEFINED fields.

UNPREDICTABLE -- Results specified as UNPREDICTABLE may vary from one performance of an operation to the next as viewed by software. In other words, the result is a function of some state or input that is not visible to software. Software must not depend on any result specified as UNPREDICTABLE.

user interface -- All node logic exclusive of the BIIC.

user interface CSR space -- That portion of each nodespace allocated for user interface registers. The user interface CSR space is the 8-Kbyte nodespace minus the lowest 256 bytes, which comprise the BIIC CSR space.

user interface request -- A transaction request from the user interface, which can take the form of a master port request, an assertion of a BCI INT<7:4> L line, or the setting of a force bit.

VAX interrupt priority level (IPL) -- In VAX/VMS systems, a number between 0 and 31 that indicates the priority level of an interrupt with 31 being the highest priority. When a processor is executing at a particular level, it accepts only interrupts at a higher level, and on acceptance starts executing at that higher level.

VAX port adapter -- In a VAXBI system, an adapter that conforms to the VAX port architecture, uses interlock transactions to access command and response queues in VAXBI memory, and performs virtual-to-physical memory translation by using page tables located in memory on the VAXBI bus.

VAXBI primary interface (VPI) -- The portion of a node that provides the electrical connection to the VAXBI signal lines and implements the VAXBI protocol; for example, the BIIC.

VAXBI request -- A request for a VAXBI transaction from the master port interface that is asserted on the BCI RQ<1:0> L lines.

VAXBI system -- All VAXBI cages, VAXBI modules, reset modules, and power supplies that are required to operate a VAXBI bus. A VAXBI system can be a subsystem of a larger computer system.

VAXBI transaction -- A transaction in which information is transmitted on the VAXBI signal lines. Contrast with loopback transaction.

vector data cycle -- A data cycle in which an interrupt vector is transmitted from a slave to a master.

VPI -- Abbreviation for VAXBI primary interface (for example, the BIIC).

window adapter -- A bus adapter that maps addresses that fall within one contiguous region (a "window") of a bus's address space into addresses in a window (possibly in a different region) in another bus's address space.

write-type command -- Any of the various VAXBI write commands, including WRITE, WCI (Write with Cache Intent), WMCI (Write Mask with Cache Intent), and UWMCI (Unlock Write Mask with Cache Intent).

write data cycle -- A data cycle in which data is transmitted from a master to a slave.

INDEX

Abort conditions, 11-13 to 11-14
  BCI MAB L, 15-16
AC timing specifications
  of BIIC, 20-5 to 20-8
  of clock driver, 22-8
  of clock receiver, 23-7 to 23-9
ACK response, 1-7, 4-11, 4-13
Adapter class nodes, 8-3
Adapters, 1-2
  bus, AN1-3
    interrupt vectors, 5-36
    use of RETRY, AN5-2
  direct memory access, AN1-2
  mapped, AN1-2
  node window, 2-1, 2-5, AN3-2
  nonpended bus, AN5-2, AN5-4
  pended bus, AN5-2
  programmed I/O, AN1-1 to AN1-2
  VAX port, AN1-3
  window, AN1-3
Address interpretation, 5-5 to 5-9
Address space, 1-3, 2-1 to 2-7
Allocation of VAXBI address space, 2-1
ARB bit, 7-7
Arbitration
  codes, 3-2, 7-7
  cycle, 1-7, 3-1
  default at power-up, 3-2, 3-3
  distributed, 1-2
  modes, 3-2
    dual round-robin, 3-3
    fixed-high priority, 3-4
    fixed-low priority, 3-3
  priority levels, 3-3
  restriction, 3-4
  setting the mode, 3-2
Arbitration Control bit. See ARB bit
Arbitration modes
  dual round-robin, 10-7 to 10-10
    behavior, 10-7 to 10-8
    latency, 10-9 to 10-10
  fixed-high priority and dual round-robin, 10-11 to 10-13
  fixed-low priority and dual round-robin, 10-10 to 10-11
  one fixed-high priority node, 10-13 to 10-14
Architectural requirements
  arbitration mode, 3-4
  Broke bit location, 11-6
  by node class, 8-3 to 8-11
  cacheing, 5-9 to 5-13
  device type, 11-9 to 11-10
  extension cycle limit, 10-4
  I/O space use, 2-1 to 2-5, 8-8 to 8-10
  interlocks, 5-2 to 5-5
  interrupt level, 5-27
  interrupt vectors, 5-35 to 5-36
  self-test, 11-1 to 11-10
Asserted, 1-5, 4-1
Assignable Window, 2-1
Assignable window, 1-3, 2-6
Assignable Window Space, 2-1
Assignable window space, 1-3
  adapter, 2-6
Atomic operation, 5-24

Backplane pins, 13-5
Bandwidth, 1-4, 10-1
  and the BIIC, AN8-1 to AN8-5
Battery backup, 6-3, 6-12, AN3-2, AN4-5
BCI, 1-10
  and interrupts, 14-7
  and user interface, 14-5
  functions, 14-5 to 14-7
BCI Control and Status Register, 2-7, 7-22 to 7-25, AN3-3 to AN3-4
BCI power sequence timing, AN7-1 to AN7-8
BCI signals, 1-11, 15-3 to 15-44
  BCI AC LO L, 15-42, AN7-1
  BCI CLE H, 15-22
  BCI D<31:0> H, 15-7, 18-1
  BCI DC LO L, 15-42, 18-1, AN7-1
  BCI EV<4:0> L, 15-28 to 15-42
  BCI I<3:0> H, 15-7 to 15-9, 18-1
  BCI INT<7:4> L, 15-27
  BCI MAB L, 15-16
  BCI MDE L, 15-19

BCI NXT L, 15-18
BCI P0 H, 15-10, 18-1
BCI PHASE L, 15-44
BCI RAK L, 15-17
BCI RQ<1:0> L, 15-11 to 15-15
BCI RS <1:0> L, 15-20 to 15-22
BCI SC<2:0> L, 15-25
BCI SDE L, 15-23
BCI SEL L, 15-23
BCI TIME L, 15-44
clock signals, 15-44
data path signals, 15-7 to 15-10
diagnostic mode operation codes, 15-15
interrupt signals, 15-27
master signals, 15-11 to 15-19
power status signals, 15-42 to 15-43
request codes, 15-11
select codes, 15-26
slave signals, 15-20 to 15-26
transaction status signals, 15-28 to 15-42
BDCST Enable bit. See BDCSTEN bit
BDCST transaction, A-1
and the BIIC, 18-16
BDCSTEN bit, 7-23
BI AC LO L, 4-17, 6-8
BI BAD L, 4-18
use of, 11-6, AN4-7
BI BSY L, 4-6
and BI NO ARB L, 4-7 to 4-10
BI CNF<2:0> L, 4-10 to 4-14
BI D<31:0> L, 4-4
BI DC LO L, 4-17, 6-9 to 6-10, 18-2
BI I<3:0> L, 4-4
BI NO ARB L, 4-5
and BI BSY L, 4-7 to 4-10
BI P0 L, 4-4
use of, 11-11 to 11-12
BI PHASE +/-, 4-16
BI RESET L, 4-17, 6-11
BI SPARE L, 4-18
BI STF L, 4-18
use of, AN4-5
BI TIME +/-, 4-16
BICSREN bit, 7-24, 18-5, 18-7
BIIC, 1-9
absolute maximum ratings, 20-1
and error recovery, 17-2
and VAXBI bandwidth, AN8-1 to AN8-5
BDCST transaction, 18-16
block diagram, 14-4
CSR space, 2-4, 16-1
diagnostic facilities, 17-1 to 17-2
electrical characteristics, 20-1 to 20-13
features, 14-2
INTR and IDENT transactions, 18-7 to 18-16
INVAL transaction, 18-17
IPINTR transaction, 18-16
load circuits, 20-9 to 20-10
packaged, with heat sink, 19-1
pin grid array, 19-2
power-up, 18-1 to 18-3
read-type transactions, 18-4 to 18-5
registers, 16-1 to 16-2
and lock commands, 16-1
recommended use, AN3-1 to AN3-17
RESERVED commands, 18-17 to 18-18
restrictions, 2-7
retry state, 18-3
signal characteristics, 20-11 to 20-13
signals, 15-1 to 15-44
STOP transaction, 18-16
timing diagrams, 21-1 to 21-33
write-type transactions, 18-6 to 18-7
BIIC CSR space, 1-3
BIIC CSR Space Enable bit. See BICSREN bit
BIIC functions
BCI functions, 14-5 to 14-7
BIIC-generated transactions, 1-10
Broke bit
location of, 11-6
of Slave-Only Status Register, 2-4, 7-32
of VAXBICSR, 7-6
use of, 6-2, AN4-7, AN4-8
BTO bit, 7-12
Burst Enable bit. See BURSTEN bit
Burst mode, 3-5, 4-5
BURSTEN bit, 7-22

Bus Error Register, 7-9 to 7-13, AN3-4
  and error logging, 11-11 to 11-13
  and initialization, 15-10
Bus Timeout bit. See BTO bit
Busy bits
  of Receive Console Data Register, 2-4, 7-33, 7-34
Busy extension cycles, 4-6
  limits, 10-4

Cables, 13-26
Cache memory, 1-8, 1-9, 2-1, 5-9 to 5-13, AN2-1 to AN2-9
  and local writes, AN2-3
  and multiprocessing, AN2-3 to AN2-7
  no write allocate, AN2-7
  write-back cache and design of VAXBI systems, AN2-9
  write-back cache defined, AN2-2
  write-through cache defined, AN2-2
  write-through cache requirements, 5-12 to 5-13, AN2-8 to AN2-9
Cached bit, 5-11, 5-23, AN2-6
Card cages
  cable access, 13-26
  general specification, 13-20
  module orientation, 13-21 to 13-25
  reference designator system, 13-27 to 13-31
  signal terminators, 13-33
  slot placement, 13-32
CHAR bits, of Receive Console Data Register, 7-34
Clock driver, 22-1 to 22-15
  absolute maximum ratings, 22-3
  AC timing, 22-8
  DC characteristics, 22-4
  module requirement, 13-32
  pin assignments, 22-2
Clock receiver, 23-1 to 23-12
  absolute maximum ratings, 23-3
  AC timing, 23-7 to 23-9
  DC characteristics, 23-4
  pin assignments, 23-2
  use of, AN6-1 to AN6-14
Clock signals
  BCI signals, 15-44
  driver specification, 22-1 to 22-15
  receiver specification, 23-1 to 23-12
  VAXBI signals, 12-1 to 12-4
Clock terminators, 13-33
CMD bits, 7-27
CNF codes. See Response codes
Cold start, 6-3, 6-6, AN4-2
Command bits. See CMD bits
Command codes
  BCI signals, 15-8
Command confirmation, 1-7
Command confirmation cycle, 1-7
Command Parity Error bit. See CPE bit
Command responses, 4-11 to 4-12
Command/address cycle, 1-7, 3-4
Configuration data
  loading of, 11-9 to 11-10, 18-1
Configurations, VAXBI
  and cache memory, AN2-1 to AN2-7
  maximum, 12-8
Console protocol, 9-1 to 9-6
Control Transmit Error bit. See CTE bit
Cooling, 13-12
Corrected Read Data bit. See CRD bit
CPE bit, 7-11, 11-12
CRD bit, 7-13
CTE bit, 7-10, 11-13
Cycles
  length, 1-7
  types, 3-4

Data cycles, 1-7, 3-5
Data length codes
  BCI signals, 15-7
  VAXBI signals, 5-5
Data path signals
  BCI signals, 15-7 to 15-10
  VAXBI signals, 4-4
Data responses, 4-12 to 4-13
Data transfer transactions, 10-4
DC characteristics
  of BIIC, 20-2 to 20-4
  of clock driver, 22-4
  of clock receiver, 23-4
Deasserted, 1-5, 4-1

Decoded ID, 3-5
Device Register, 7-4, AN3-5 to AN3-6
  and initialization, 6-2
  device type codes, 11-9 to 11-10
  on power-up, 18-1
Device Revision bits. See DREV bits
Device Type bits. See DTYPE bits
Device type. See Device Register
Diagnostic mode, 15-14 to 15-15, 15-28
  and self-test, 18-3
Diagnostics
  ROM-based, AN9-1 to AN9-4
Down-line loading of software, 6-5
DREV bits, 7-4
DTYPE bits, 7-4
  device type codes, 11-9 to 11-10
Dual round-robin, 1-2

Electrical specification
  of VAXBI signals, 12-1 to 12-15
Encoded ID, 3-2
Ending Address Register, 2-7, 7-21, AN3-1 to AN3-3
Environmental conditions, 13-33
Error checking, 11-1
Error detection, 11-11 to 11-16
  and logging, 11-11 to 11-13
  by BIIC, 17-2
  parity checking, 11-11
  protocol checking, 11-13
  transmit check error detection, 11-12
Error Interrupt Control Register, 7-14 to 7-15, AN3-7, AN3-8
Error logging. See error detection
Error recovery, and BIIC, 17-2
ESD pads, required on module, 13-4
Event codes, 15-28 to 15-42
  AKRE, 15-37
  AKRNEn, 15-39
  AKRSD, 15-36
  and BER bits, 15-41
  ARCR, 15-37
  BBE, 15-39
  BPM, 15-40
  BPR, 15-40
  BPS, 15-38
  BTO, 15-36
  EVSn, 15-38
  IAL, 15-38
  ICRMC, 15-39
  ICRMD, 15-40
  ICRSD, 15-39
  IRW, 15-37
  MCP, 15-35
  MTCE, 15-40
  NCRMC, 15-39
  NICI, 15-37
  NICIPS, 15-37
  RCR, 15-37
  RDSR, 15-39
  response to, C-1 to C-3
  RTO, 15-40
  STO, 15-38
  STP, 15-36
  windows, 15-33
EX VECTOR bit, 7-29
Exception conditions
  and EV codes, 15-28 to 15-42
  response to, 11-14 to 11-15, C-1 to C-3
Expansion modules, 12-8
Extension cycles, 10-3, 15-21
  limits, 10-3

Fast self-test, 4-18, 11-7, AN4-4
Force-Bit IPINTR/STOP Command Register, 7-27
Force-Bit IPINTR/STOP Destination Register, 7-18, AN3-16

General Purpose Registers, 7-31, AN3-6
GPR bits, in Write Status Register, 7-26

H, 1-5
Hard Error INTR Enable bit. See HEIE bit
Hard Error Summary bit. See HES bit
HEIE bit, 7-7
HES bit, 7-5

I/O address space, 2-1
ICE bit, 7-12

ID Parity Error bit. See IPE bit
IDENT arbitration cycle, 5-27, 5-32 to 5-33, 18-9 to 18-10
IDENT Enable bit. See IDENTEN bit
IDENT transaction
  and the BIIC, 18-7 to 18-16
  defined, 5-32 to 5-33
IDENT Vector Error bit. See IVE bit
IDENTEN bit, 7-23
Illegal Confirmation Error bit. See ICE bit
Imbedded arbitration cycle, 1-7, 3-1, 3-5
INIT bit, 7-5
Initialization
  of individual nodes, 6-6
  of VAXBI systems, 6-2
Initialize bit. See INIT bit
Inter-bus adapters, 8-10
Interlock Sequence Error bit. See ISE bit
Interlock transactions, 5-2 to 5-5, 8-9, 11-15
  and RETRY CNF code, AN5-2
  BIIC registers excluded, 16-1
Internode transactions, 1-10, 14-6
Interprocessor communication, 5-2
Interrupt Destination Register, AN3-7
Interrupt port, 1-10
Interrupt port interface, 14-2, 14-7
Interrupt priority level (IPL), VAX, 5-27
Interrupt sublevel priority, 3-1, 5-30
Interrupts, 1-5, 1-9
  and the BCI, 14-7
  and the BIIC, 18-7 to 18-16
  BCI signals, 15-27
  difference between error interrupts and user interrupts, AN3-7
  interprocessor, 1-9
  level, 5-30
  null, 5-27 to 5-28
  priority, 5-30
  registers, AN3-6 to AN3-10
  strategies, AN3-10 to AN3-15
  sublevel, 5-30
  transactions to support, 5-27 to 5-38
  vectors, 5-35 to 5-36
Interrupts, interprocessor
  registers, AN3-16 to AN3-17
INTR Abort <7:4> bits. See INTRAB bits
INTR Abort bit. See INTRAB bit
INTR Complete <7:4> bits. See INTRC bits
INTR Complete bit. See INTRC bit
INTR Destination bits. See INTRDES bits
INTR Destination Register, 7-16
INTR Enable bit. See INTR Enable bit
INTR Force <7:4> bits, 7-29
INTR Force bit, 7-15
INTR Sent <7:4> bits, 7-28
INTR Sent bit, 7-15
INTR transaction
  and the BIIC, 18-7 to 18-16
  defined, 5-29 to 5-31
INTRAB bit, 7-14
INTRAB bits, 7-28
Intranode transactions, 1-10, 14-6
  length limit, 14-6, 15-12
INTRC bit, 7-14
INTRC bits, 7-28
INTRDES bits, 7-16
INTREN bit, 7-24
INVAL Enable bit. See INVALEN bit
INVAL transaction, 1-9, AN2-4
  and the BIIC, 18-17
  defined, 5-24 to 5-25
INVALEN bit, 7-24
IPE bit, 7-13, 11-12
IPINTR Enable bit. See IPINTREN bit
IPINTR Mask bits, 7-17
IPINTR Mask Register, 7-17, AN3-16
IPINTR Source Register, 7-19, AN3-16
IPINTR transaction
  and the BIIC, 18-16
  defined, 5-37 to 5-38
IPINTR/STOP Force bit, 7-23
IPINTREN bit, 7-25
IRCI transaction, 5-24
IREV bits, 7-5

ISE bit, 7-10, 11-13
ITYPE bits, 7-5
IVE bit, 7-11

L, 1-5
Latency, 10-3 to 10-14
  bus access, 10-3
  effect of arbitration modes, 10-4 to 10-14
  interrupt, 10-3
  use of RETRY, AN5-2
LED requirements
  multimodule nodes, AN4-7
  position on module, 13-4
Level <7:4> bits, 7-15
Load circuits, 20-9 to 20-10
Loopback transactions, 1-10, 4-6, 4-9, 14-6, 15-22, 16-1, 18-2
  and IRW event code, 18-7
  limits on extension cycles, 10-4
  request, 15-13 to 15-14
  use of STALL, 15-21

Master console, 9-1
Master ID Enable bit. See MIDEN bit
Master Parity Error bit. See MPE bit
Master port, 1-10, 14-6
Master port interface, 14-6
Master port request, 14-6
Master port transaction, 14-6
Master signals
  BCI signals, 15-11 to 15-19
Master Transmit Check Error bit. See MTCE bit
Mechanical specification, 13-1 to 13-33
Memories, 1-2, AN2-1 to AN2-9
  local, 5-10, 5-24, AN2-1
  private, 1-9
  with battery backup, 6-3, 6-12, AN3-2, AN4-5
Memory address space, 2-1
Memory class nodes, 8-3
Memory size bits. See MSIZE bits
MIDEN bit, 7-27
Modules, 1-2
  backplane pins and signals, 13-5
  border and ESD requirements, 13-4
  expansion, 12-8
  layers, 13-13
  LED requirements, 13-4, AN4-7
  mechanical and power specification, 13-2 to 13-12
  physical requirements, 13-2 to 13-3
  power dissipation and cooling, 13-11
  replaceability requirements, 13-5
  VAXBI Corner, 13-2, 13-12, 13-13
  VAXBI Corner signals, 13-16
  voltages and currents, 13-7
  with BIICs, 13-12 to 13-19
MPE bit, 7-10, 11-12
MSEN bit, 7-23
MSIZE bits, 7-32
MTCE bit, 7-10, 11-12
Multi-responder transactions, 1-7, 1-8, 3-6, 5-1
Multicast space, 2-5
Multicast Space Enable bit. See MSEN bit
Multiprocessing, 1-2
  and cache memory, AN2-3 to AN2-7
  console protocol, 9-1 to 9-6
  symmetric, 1-13, AN3-8

NEX bit, 7-12
NMR bit, 7-9
NO ACK response, 1-7, 4-11, 4-13, 11-14
NO ACK to Multi-Responder Command Received bit. See NMR bit
Node ID, 1-2, 2-1, 3-1
  and interrupt sublevel priority, 5-30
  arbitration priority, 3-3
Node ID bits
  of VAXBICSR, 7-8
    on power-up, 18-1
Node ID bits, of Receive Console Data Register, 7-34
Node ID plugs, 1-2, 4-1
Node private space, 2-5
Node reset, 6-6

NRST bit, 7-6
Node Reset bit. See NRST bit
Node window space, 1-3, 2-1, 2-5
Nodes, 1-2
  block diagram, 14-1
  initialization mechanisms, 6-2 to 6-8
  initialization requirements, 6-2
  initialization timing sequences, 6-12
  multimodule nodes and self-test, AN4-7
  resetting, 6-5
  self-test, 11-1 to 11-10, 17-1
Nodes, classes of, 8-1 to 8-11
Nodespace, 1-3, 2-1, 2-4, 7-1
Nonexistent Address bit. See NEX bit
NPE bit, 7-13
NRST bit, 6-6, 7-6, 15-42, AN7-2
Null Bus Parity Error bit. See NPE bit
Null cycle, 7-13

Parity, 1-7
  and the BCI, 15-10
  on power-up, 18-1
Parity checking, 11-11 to 11-12
Pending master, 1-3
Performance, 1-4, 10-1 to 10-14
Pin assignments
  clock driver, 22-2
  clock receiver, 23-2
  pin grid array of BIIC, 19-2
Pipeline NXT Enable bit. See PNXTEN bit
Pipeline requests, 15-12, 15-17, 15-31, 15-37
  and assertion of BCI MAB L, 15-16
  and bandwidth, AN8-3 to AN8-4
PNXTEN bit, 7-25
Power dissipation, 13-11
Power failures, 6-2, AN3-2
Power specifications, 13-7 to 13-11
Power status signals
  BCI signals, 15-42 to 15-43
Power-down/power-up sequence, 4-17, 6-2, 6-12
  BCI signals, AN7-1 to AN7-8
Power-up
  and the BIIC, 18-1 to 18-3
Processor class nodes, 8-2
Processors, 1-2
  console protocol, 9-1 to 9-6
  primary, AN3-1, AN3-7, AN3-8, AN4-5, AN4-7 to AN4-8
Protocol
  console, 9-1 to 9-6
  VAXBI, 1-7, 3-1 to 3-6, AN2-5

Queue instructions. See Interlock transactions

RCI transaction
  defined, 5-22 to 5-23
RCLK, 12-1 to 12-2
RDS bit, 7-11
Read Data Substitute bit. See RDS bit
Read side effects, 8-8
Read status codes
  BCI signals, 15-9
  VAXBI signals, 5-14
READ transaction, 1-8
  defined, 5-21 to 5-22
Read-type transactions, 1-8
  and the BIIC, 18-4 to 18-5
Receive Console Data Register (RXCD), 2-4, 7-33, 9-2
  and ROM-based diagnostics, AN9-1 to AN9-4
Registers
  required for VAXBI bus, 7-1 to 7-34
  reserved, AN3-17
Requests
  pending, 18-18
RESEN bit, 7-23
RESERVED commands
  and the BIIC, 18-17 to 18-18
RESERVED Enable bit. See RESEN bit
RESERVED field, 5-25, 5-29
Reset module, 4-17, 6-3 to 6-6, 6-11, 6-12
Reset sequence. See System reset
Resetting nodes, 6-5
Response codes
  BCI codes, 15-20 to 15-22
  meaning and use of, 4-14
  VAXBI codes, 4-10 to 4-14

Restart parameter block, 6-5, 6-6
RETRY response, 1-7, 4-11
RETRY response code, use of, AN5-1 to AN5-8
Retry state of BIIC, 18-3
Retry timeout, 18-4, AN5-1
RETRY Timeout bit. See RTO bit
Revision code, 7-4
RTO bit, 7-12
RTO EV Enable bit. See RTOEVEN bit
RTOEVEN bit, 7-25
RXCD Register. See Receive Console Data Register

SEIE bit, 7-7
Self-test, 6-6, 11-1 to 11-10
  and diagnostic mode, 18-3
  and multimodule nodes, AN4-7
  and the BIIC, 18-2 to 18-3
  determining the results of, AN4-7 to AN4-8
  extended, 11-8, AN4-5
  fast, 4-18, 11-7, AN4-4
  goals, AN4-1 to AN4-2
  normal, AN4-2
  operation, 11-1 to 11-2
  requirements, 11-2 to 11-10
  role of BIIC, 17-1
  time limits, 11-6 to 11-8, AN4-2 to AN4-6
  transactions during self-test, AN4-6
  use of BI BAD L, AN4-7, AN4-8
  use of Broke bits, AN4-7, AN4-8
Self-Test Status bit. See STS bit
SES bit, 7-5
Signals. See VAXBI signals; BCI signals
Single-responder transactions, 1-7, 1-8, 5-1
Slave Parity Error bit. See SPE bit
Slave port, 1-10, 14-6
Slave port interface, 14-6
Slave signals
  BCI signals, 15-20 to 15-26
Slave-Only Status Register (SOSR), 2-4, 7-32
  and Broke bit, 11-6
Slot placement, 1-2, 13-32
Soft Error INTR Enable bit. See SEIE bit
Soft Error Summary bit. See SES bit
Software down-line load, 6-5
SOSR. See Slave-Only Status Register
SPE bit, 7-11, 11-12
STALL cycles, 15-21, 15-22
  limits on extension cycles, 10-4
STALL response, 1-7, 4-12, 4-13, 10-3
STALL Timeout bit. See STO bit
Starting Address Register, 2-7, 7-20, AN3-1 to AN3-3
STO bit, 7-12
STOP Enable bit. See STOPEN bit
STOP transaction, 11-1, 11-16
  and output of summary EV code, 15-36
  and the BIIC, 18-16
  defined, 5-39 to 5-41
STOPEN bit, 7-23
STS bit, 7-6, 11-5
System configurations, 1-11, AN5-3 to AN5-6
System control block, 5-27, 5-36
System reset, 6-4 to 6-6, 6-12

Target bus, AN1-3, AN5-2
TCLK, 12-1 to 12-2
TDF bit, 7-10
Termination
  of clock signals, 12-8
  of VAXBI signals, 12-8
Terminators, VAXBI, 13-33
Timeouts
  bus, 7-12
  retry, 4-12, 18-4, AN5-1
    strategies for dealing with, AN5-6 to AN5-7
  stall, 4-12, 4-13
Timing
  worst-case bus, 12-9
Timing diagrams, 21-1 to 21-33, AN6-10
Transaction status signals
  BCI signals, 15-28 to 15-42
Transactions, 1-6 to 1-10
  during self-test, AN4-6
  format, 3-4

internode, 1-10
intranode, 1-10
local writes, AN2-3
loopback, 1-10, 15-13 to 15-14, 16-1, 18-2
multi-responder, 1-7, 1-8, 3-6
priority of, 18-18 to 18-19
requests, 15-12 to 15-15
response to exception conditions, 11-14 to 11-15
single-responder, 1-7, 1-8
STOP, 11-16
types of cycles, 3-4
VAXBI primary interface abort conditions, 11-13 to 11-14
with extension cycles, 15-21
with STALL cycles, 15-21
Transactions. See also Loopback transactions, VAXBI transactions
Translations of VAXBI transactions, 8-10
Transmission line effect, 6-9, 12-14, B-1
Transmitter During Fault bit. See TDF bit
Transparent mode, 15-14

UCSREN bit, 7-24
UNDEFINED field, 5-13
UNIBUS
and read-type transactions, 11-14
nonpended bus, AN5-2
UNIBUS adapter, 5-4, 5-24, 5-36, 8-10, AN3-2, AN5-2
Unlock Write Pending bit. See UWP bit
UPEN bit, 7-13
User interface, 1-10, 14-5
and parity, 15-10
and self-test, 18-2
on power-up, 18-2
User interface CSR space, 2-4
reserved locations, 2-4
User Interface CSR Space Enable bit. See UCSREN bit
User Interface Interrupt Control Register, 7-28 to 7-30, AN3-7, AN3-10
User Parity Enable bit. See UPEN bit

UWMCI transaction
defined, 5-19 to 5-20
UWP bit, 7-7

VAXBI address space, transaction requirements, 8-8
VAXBI Control and Status Register, 7-5 to 7-8, AN3-4 to AN3-5
and Broke bit, 11-6
VAXBI Corner, 13-2, 13-12
clock receiver capacitance load, AN6-8
parts list, 13-18
signal locations, 13-13
VAXBI Interface Revision bits. See IREV bits
VAXBI Interface Type bits. See ITYPE bits
VAXBI primary interface, 1-9
See also BIIC
VAXBI protocol
and error checking, 11-13
VAXBI signals, 1-4, 15-1
asynchronous control, 4-16 to 4-18, 12-5
BI AC LO L, 6-8
BI BAD L, 11-6, AN4-8
BI DC LO L, 6-9 to 6-10
BI PHASE +/-, 12-2
BI RESET L, 6-4, 6-5, 6-11
BI TIME +/-, 12-1
clock, 4-16, 12-1 to 12-4
data path, 4-4
data path and synchronous control signals, 12-4
synchronous control, 4-4 to 4-14
VAXBI system, 13-1
See also Configurations
VAXBI transactions, 1-10, 14-6
command codes, 5-2
data length codes, 5-5
IDENT, defined, 5-32 to 5-33
INTR, defined, 5-29 to 5-31
INVAL, defined, 5-24 to 5-25
IPINTR, defined, 5-37 to 5-38
IRCI, 5-24
RCI, defined, 5-22 to 5-23
read status codes, 5-14
READ, defined, 5-21 to 5-22
read-type, 5-21 to 5-24

requirements by node class, 8-3 to 8-10
STOP, defined, 5-39 to 5-41
UWMCI, defined, 5-19 to 5-20
VAXBI request, 1-10
WCI, defined, 5-16 to 5-17
WMCI, defined, 5-18 to 5-19
write mask codes, 5-13
WRITE, defined, 5-15 to 5-16
write-type, 5-15 to 5-20
Vector bits, 7-15, 7-30
Voltage drops, 13-10
Voltages
available, 13-7
required, 13-7

Warm restart, 6-3, AN4-2
WCI transaction
defined, 5-16 to 5-17
WINVALEN bit, 7-24
WMCI transaction
defined, 5-18 to 5-19
Worst-case conditions, 13-10
WRITE Invalidate Enable bit. See WINVALEN bit
Write mask
in I/O space, 8-9
Write mask codes
VAXBI signals, 5-13
Write Status Register, 7-26
WRITE transaction, 1-8
defined, 5-15 to 5-16
Write-type transactions, 1-8
and the BIIC, 18-6 to 18-7

Z console command, 9-5, AN9-2