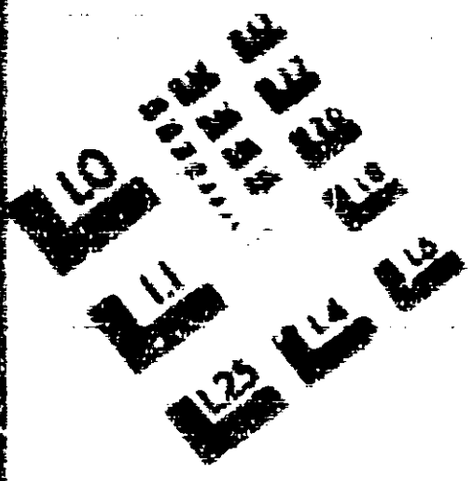
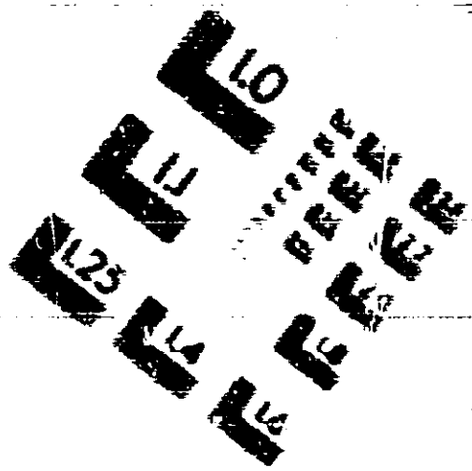
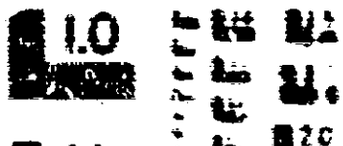
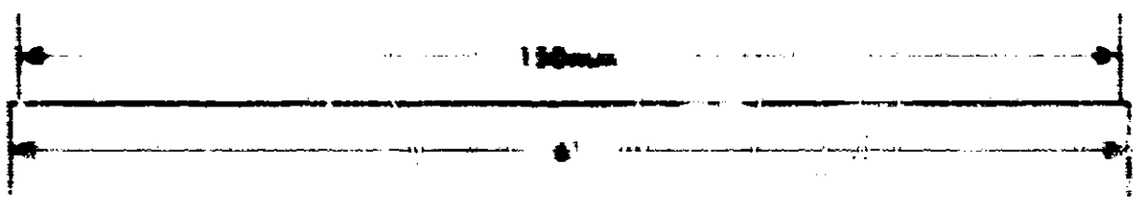
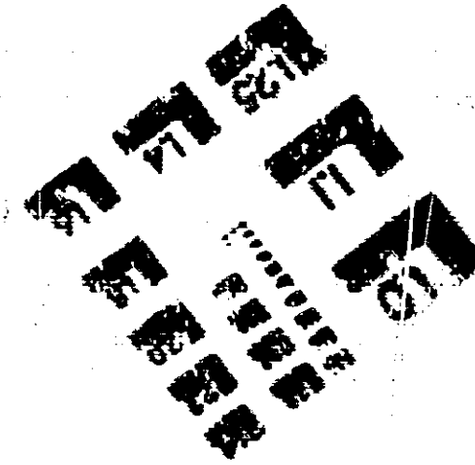




AIIM
TEST TARGET (MT-2)







BELL HOWELL
Photocircuit Systems Division

PHOTOGRAPHIC SCIENCES CORPORATION
770 BASKET ROAD
P.O. BOX 338
WEBSTER, NEW YORK 14580

DWBUA

EK-DWBUA-TM-001

DWBUA UNIBUS Adapter Technical Manual

Prepared by Educational Services
of
Digital Equipment Corporation

EK-DWBUA-TM-001

DWBUA
UNIBUS Adapter
Technical Manual

Prepared by Educational Services
of
Digital Equipment Corporation

1st Edition, January 1986

The information in this document is subject to change without notice and should not be construed as a commitment by Digital Equipment Corporation. Digital Equipment Corporation assumes no responsibility for any errors that may appear in this document.

Printed in U.S.A.

This document was set on a DIGITAL DECset Integrated Publishing System.

CONTENTS

CONTENTS (Cont)

CONTENTS (Cont)

FIGURES

FIGURES (Cont)

TABLES

PREFACE

MANUAL STRUCTURE AND ALIGNMENT

The manual is divided into two parts:

Part I - Installation

Part I includes an introduction to the DWBI A, product specifications and instructions for installing and testing a DWBI A system. It is intended for DIGITAL personnel or customers who install the adapter. A knowledge of VAX hardware is assumed.

Part II - Technical Description

This part of the manual provides the technical information needed by the system programmer and the support group, as well as by customer engineers and programmers who incorporate this adapter into their own product or system. A knowledge of VAX architecture, the VAX Bus Interconnect (VAXBI), and the UNIBUS is assumed.

RELATED DOCUMENTATION

The DWBI A is one of a family of processors, memories, and adapters that use the 32-bit VAXBI bus. For a functional summary of the VAXBI bus and a description of VAXBI options, see the *VAXBI Options Handbook*, EB-22971-46.

NOTE: *For ease of use and for a later comprehension, the DWBI A adapter (VAXBI to UNIBUS adapter) will be referred to as DWBI A throughout this document. The VAXBI bus will be referred to as VAXBI, and UNIBUS bus will be referred to as UNIBUS.*

PART 1
INSTALLATION

CHAPTER 1

CHAPTER 1
INTRODUCTION

1.1 [illegible]

The [illegible] Adapter (DWBI) enables transfers between the high-speed synchronous [illegible] and the asynchronous [illegible]. Through the DWBI, the [illegible] has access to any [illegible] address space, and the [illegible] has access to any [illegible] address space.

The DWBI transfers data between the buses in two ways:

1. Through the Direct Data Path (DDP), the data is transferred immediately.
2. Through a Buffered Data Path (BDP), the DWBI temporarily buffers data in its memory. The burst of data per transfer is maximized, the [illegible] bandwidth.

[illegible] initiated transactions transfer data through the Direct Data Path. [illegible] transactions can transfer data through either the Direct Data Path or the Buffered Data Path.

Other features of the DWBI are:

- [illegible]
- [illegible] devices can interrupt the [illegible]
- Data transfer rate up to [illegible]
- Software to verify data paths and control registers [illegible]

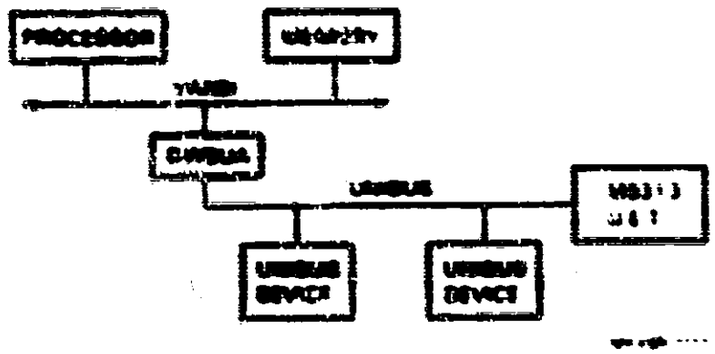


Figure 1-1 Typical DWBI Configuration

1.2 SPECIFICATIONS

1.2.1 Slot Loading

The DWBI-A is 0.5 dc unit load on the UNIBUS

The DWBI-A is 1.5 ac unit load on the UNIBUS

1.2.2 Power Requirements

Table 1-1 DWBI-A Power Requirements

	POWER (Watts)			
VOLTAGE	Minimum	Typical	Standard	Maximum
+5V	11.25	10.50	10.50	10.50
-12V	-0.01	0.036	0.54	0.6

1.2.3 Current Requirements

Table 1-2 DWBI-A Current Requirements

	CURRENT (Amps)			
VOLTAGE	Minimum	Typical	Standard	Maximum
+5V	2.0	2.1	2.1	2.1
-12V	-0.001	0.003	0.011	0.012

1.3 SUPPORTED UNIBUS DEVICES

A subset of the available UNIBUS devices is supported at a configuration with a DWBI-A. See Appendix A for details.

CHAPTER 2

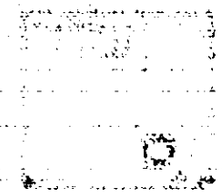


Table 2-1 DWBLA Components – UNIBUS Installed in BA11-MC/AD Box

Component	Qty	Part Number	Location
DWBLA module	1	71010	VAXBI cardcage
UET module	1	50313	UNIBUS cardcage
UNIBUS pathboard	1	50716	UNIBUS cardcage
Transition header	1	12-22240-01	VAXBI cardcage
UNIBUS cables	1	17-00837-01	M7168 to transition header

Table 2-2 DWBLA Components – UNIBUS Installed in BA11 Box

Component	Qty	Part Number	Location
DWBLA module	1	71010	VAXBI cardcage
UET module	1	50313	UNIBUS cardcage
UNIBUS pathboard	1	M7168	UNIBUS cardcage
Transition header	1	12-22240-01	VAXBI cardcage
UNIBUS cables	2	17-00837-01	M7168 to transition header
DR11-W(J) power bus cable	1	17-00931-01	Processor cabinet to UNIBUS cabinet

The DWBLA components are put together in the configuration shown in Figure 2-2.

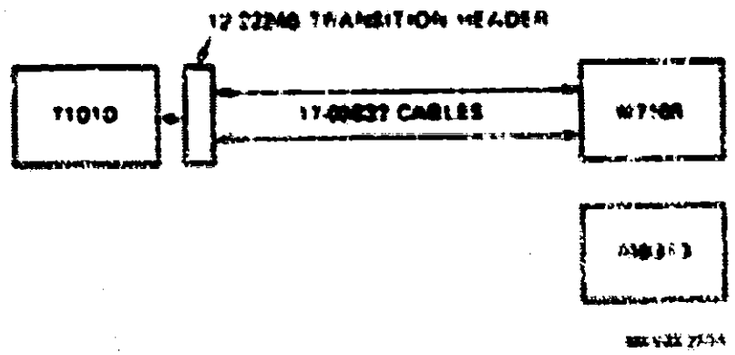


Figure 2-2 DWBLA Configuration

2.2 INSTALLATION

CAUTION
An antistatic wrist strap connected to an active ground must be worn when installing the DWBLA.

WARNING
Shut off system power before proceeding.

1. Attach the four UNIBUS cables to the M7168 pathboard (Figure 2-4). The connectors are keyed

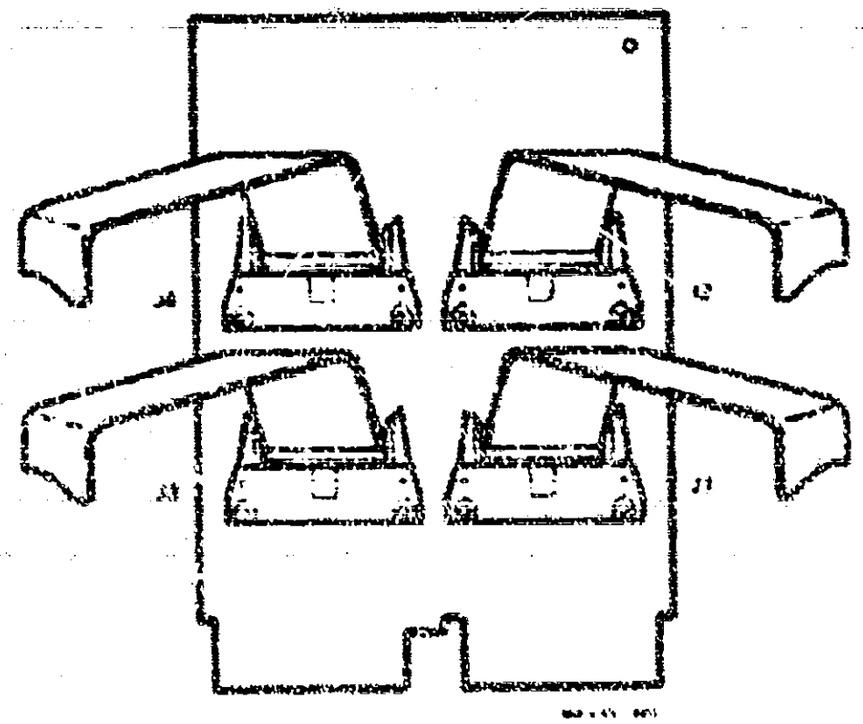


Figure 2-4 M7168 Pathboard with UNIBUS Cables

2. Insert the M7166 per Record into slot 1, segments A and B, of the I/O BUS backplane (Figure 2-4).

3. Insert the M9313 (-H) module into the last slot, segments A and B, of the I/O BUS backplane (Figure 2-4).

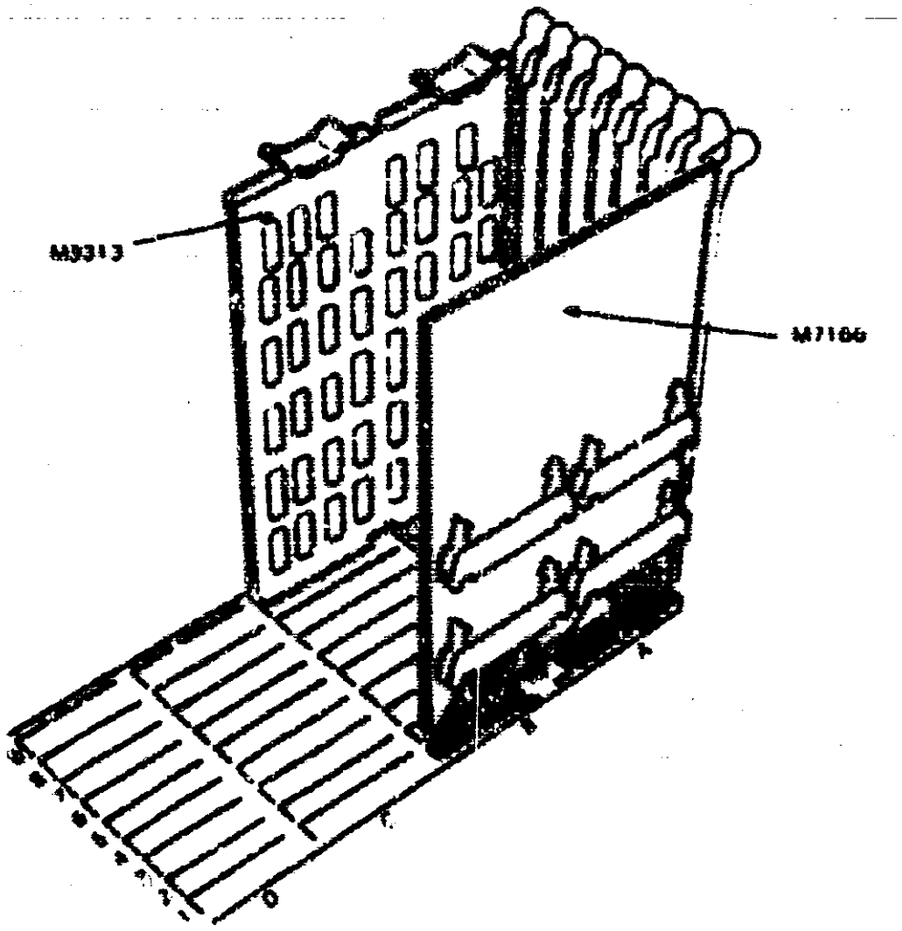


Figure 2-4 UNIBUS Backplane

4. Insert the M7166 per Record into slot 1, segments A and B, of the I/O BUS backplane (Figure 2-4).

5. Insert the M9313 (-H) module into the last slot, segments A and B, of the I/O BUS backplane (Figure 2-4).

6. Install the transition header on the backplane of the slot that will hold the M7166 module (Figure 2-5).

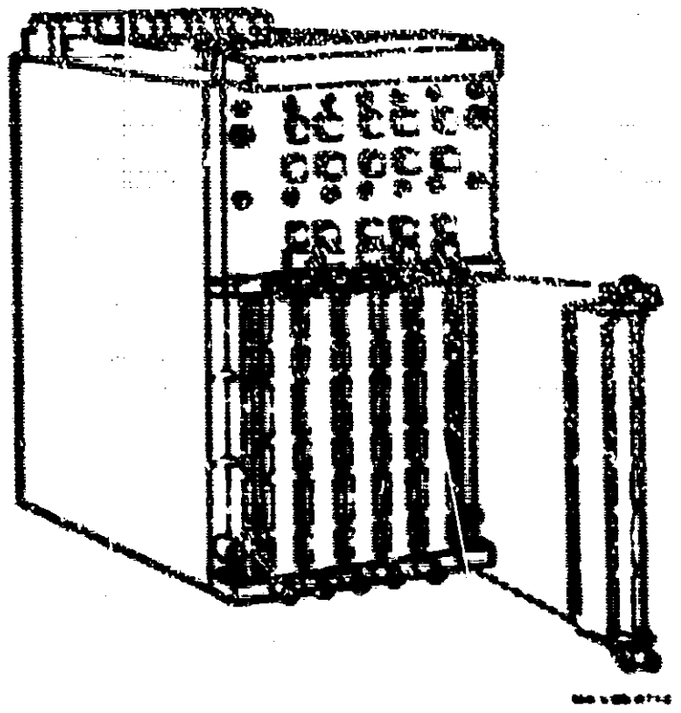


Figure 2-5 VAXBI Transition Header Installation

6. Refer to Figure 2-6 and connect the four I/NIBUS cables to the Ethernet header assembly.

J1 segment 1 (left)
J2 segment 1 (right)
J3 segment 0 (left)
J4 segment 0 (right)

The connectors are keyed.

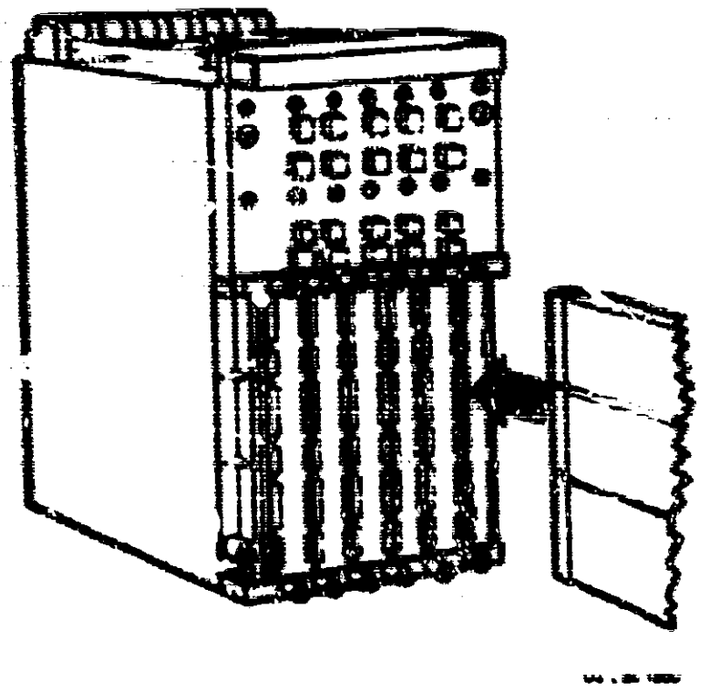


Figure 2-6 I/NIBUS Cable Connections

7. Insert the T1010 module into the appropriate slot of the VAXBI backplane.

8. If the I/NIBUS backplane is in an expansion cabinet, the power bus cable (P/N 17-01961-01) may be installed from the processor cabinet to the expansion cabinet.

9. Power up the system. The DWBI's self-test runs upon power-up. Check that the yellow LED on the T1010 module lights. See Figure 2-7.

If the yellow LED does not light, see Section 2.4.

10. Run one full pass of EVRTB, the DWBI's maintenance program. See Section 2.4.2.

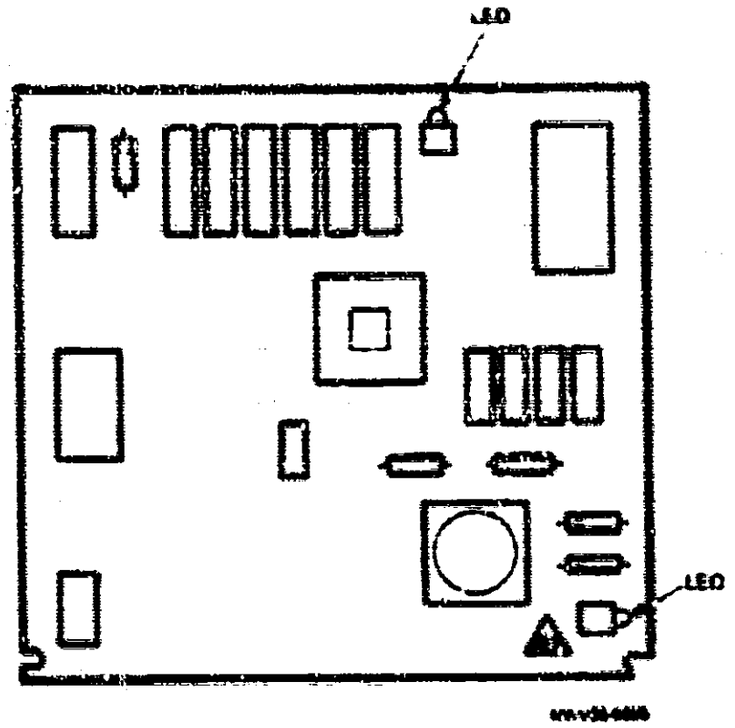


Figure 2-7 T1010 Module

2.3 TEST

2.3.1 Self-Test Microdiagnostic Program

NOTE
A UNIBUS Exerciser Terminator (UET) module (29-23130) must be installed in the last slot, segments A and B, of the UNIBUS backplane. The DWBUA self-test runs only if the UET module is installed.

The DWBUA self-test microdiagnostic program runs at powerup or when the VAXBI Control and Status Register RESTART bit (BICSR<15>) is set. Successful completion of the self-test is indicated by the lighting of the yellow LED on the T1010 module. The location of the LED is shown in Figure 2-7.

The DWBUA self-test microdiagnostic consists of 10 separate tests, which are described in Appendix C.

2.3.2 Microdiagnostic Program

The microdiagnostic program for the DWBUA is EVCBB.B. It is a level 3 diagnostic that runs standalone under the VAX diagnostic supervisor, and it isolates failures to the ratio of functions.

Descriptions of the microdiagnostic tests can be found in Appendix D.

To run EVCBB, do the following:

NOTE
Operator input is underlined.

1. Run the diagnostic supervisor.

2. Attach the DWBUA:

   DS> ATTACH DWBUA HLB DWn node br RET

   DWn is the number of the DWBUA. "n" is a number between 0 and 7.

   "node" is the VAXBI node ID, expressed as a decimal number (0 to 15).

   NOTE
   Refer to the appropriate system user guide to determine the node ID number.

   "br" is the UNIBUS BR interrupt level, a number between 4 and 7. The recommended value is 7.

3. Run the microdiagnostic:

   DS> RUN EVCBB/SECTION=xxx RET

   Inclusion of the SECTION name ("xxx" in the above command) is optional. If no SECTION name is included, the DEFAULT section is run. The SECTION names and the tests they include are listed in Table 2-3.

Tests 31 and 32 can be run only if a UNIBUS Extender (UBE) is attached.

Table 2-3 Microdiagnostic Program Sections

Section	Tests
DEFAULT	1 - 30
ALL	1 - 32
UBE	31, 32

2.4 ERROR RECOVERY

This procedure provides the information needed to isolate a DWBUA failure to one of its assemblies (T1010 module, BC19 cable, M7504 pushbutton, UNIBUS or VAXBI backplane). Corrective maintenance of the DWBUA consists of faulty subassembly replacement.

This procedure does not attempt to isolate problems caused by devices attached to the UNIBUS. It does, however, identify the UNIBUS node that is causing a DWBUA malfunction.

The assumption is made in this procedure that system troubleshooting procedures have indicated a problem in the DWBUA. No system-specific troubleshooting procedures are included here.

2.4.1 Tools and Test Equipment

The tools and test equipment listed in Table 2-4 are needed to perform the maintenance procedure described in this section.

Table 2-4 Tools and Test Equipment for Maintenance Procedures

Equipment	Manufacturer	Designation	DIGITAL Part Number
Cable Wraps	Tyco	TX309	29-01691-01
Torque Screwdriver	Utica		29-17181-00
Bus Grant Card			54-17273

2.4.2 Procedure

This section is a step-by-step procedure for isolating faults to the field replaceable units (FRUs). It uses the tools and test equipment listed in Table 2-4 and the DWBUA adapter self-test. By using this procedure, faults in the DWBUA can be isolated when the system is not capable of running diagnostics. (Such a situation can occur if the DWBUA is on the boot path for the operating system and diagnostics.)

See Section 2.3.1 and Appendix C for information on the DWBUA adapter self-test.

NOTE
Follow the steps in the order listed.

1. **START** - Is the DWBI A malfunctioning?

   The DWBI A may be suspect if:

   a. The system cannot be booted from a CNBI S device.

   b. No CNBI S devices can be used.

   c. The system console indicates that the node number corresponding to DWBI A adapter's node ID is malfunctioning.

   d. Excessive errors occur when using any CNBI S device.

   e. The system crashes.

2. **POWER DOWN THE SYSTEM** - Wait 30 seconds for the stored power to drain off.

3. **OPEN THE CABINET** - Open the system cabinet so that the yellow lights on the modules can be seen.

4. **POWER UP THE SYSTEM** - This starts the DWBI A self-test.

5. **CHECK THE LIGHT ON THE T1010 MODULE** - If the yellow light on the T1010 module is lit, the DWBI A has passed self-test. The problem is most likely not in the T1010 module, the CNBI S cabling, or the termination card (LED). If the light is OFF, go to Step 7.

6. **RUN EVCBB** - If the system is operational, run the system-level diagnostic, EVCBB, to further verify that the problem is not in the DWBI A. Refer to the microdiagnostic printout and documentation to isolate the failing FRU if this diagnostic should fail.

   If one of the symptoms listed in Step 1 exists, but the DWBI A self-test, says the problem is probably somewhere other than in the DWBI A. Refer to Table 2-5 for suggested areas to troubleshoot.

Table 2-5 Symptoms and Possible Causes

Symptom	Possible Cause
Cannot boot from a CNBI S device	Boot device
Unable to use devices on the CNBI S	Bad device or software
Excessive errors when using any devices on the CNBI S	Devices on the bus or system-wide problems
System crashes	System software

7. **THE YELLOW LIGHT IS OFF** - If the yellow light on the T1010 module is OFF, the self-test has failed. Look for a fault in one of the items in Figure 2-1. If no fault exists in these items, look for a CNBI S device that is causing the CNBI S to malfunction.

8. **DETERMINE NODE NUMBER AND STARTING ADDRESS**

   a. Halt the system from the console by typing `CTRL P`.

   b. Type `E address` to examine the contents of the Device Type Register (bbbb 0B) for each node space in succession until one with a value of xxxx0101 is found. This is the DWBI A device type. Make a note of the address. (See Figure 2-1 and Appendix A.)

   If working on a system that has more than one VAXBI, bus 0 addresses are as described. See Table 2-6 for the bus addresses for bus 1 through bus 15.

Table 2-6 Multiple VAXBI Base Addresses

VAXBI Bus #	Base Address
0	2400 0000
1	2500 0000
2	2600 0000
3	2700 0000

NOTE
If nothing is returned from any of the examines, a system problem exists. See the troubleshooting procedures for the system being used. THE PROBLEM IS NOT IN THE DWBI A.

Example 2-1: Determining Node Number and Starting Address

```
>>>E 20000000 CR                    This is the base address of the first node
                                    in I/O space, node 0.
P   20000000    00010001            Address and contents returned. Node 0
                                    is not a DWBI A.
>>>E 20080000                       This is the base address of node 1.
P   20080000    00010101            Node 1 is not a DWBI A.

>>>E 20300000                       This is the base address of node 6.
P   20300400    00010101            Node 6 is a DWBI A.
```

9 **FIND GPR0 ADDRESS** – Subtract 10 (GPR0 address). Add 10 (hex) to the address found in the previous step.

Example 2-2 Finding the Address of GPR0

```
Node 1 GPR0 = 30002000 + 10 = 30002010
Node 2 GPR0 = 30004000 + 10 = 30004010
Node 3 GPR0 = 30006000 + 10 = 30006010
Node 4 GPR0 = 30008000 + 10 = 30008010
Node 5 GPR0 = 3000A000 + 10 = 3000A010
Node 6 GPR0 = 3000C000 + 10 = 3000C010
```

10 **EXAMINE GPR0** – Use the console to examine GPR0 at the address calculated in the last step. GPR0 bits <31:16> contain the test number that failed in the self-test. Refer to Appendix C for a description of the self-test execution tests.

NOTE

Failure of the DWBI A self-test can prevent accessing of the DWBI A internal registers. To access these registers to explore the cause of the self-test failure, set the BCICSR (bb+28) bit <08> (LCSREN).

11 **ISOLATE FRU** – Use the flowchart in Figure 2-8 to isolate the FRU at fault.

NOTES

1. **The T1810 module is suspect throughout this troubleshooting procedure since it is the engine running the test.**
2. **Power-down the system to replace a component. When the system is powered back up, the self-test will run. Return to Step 1 of this procedure to verify the fix.**
3. **When replacing a component, follow the removal and replacement procedures in the installation manual for the system being used.**

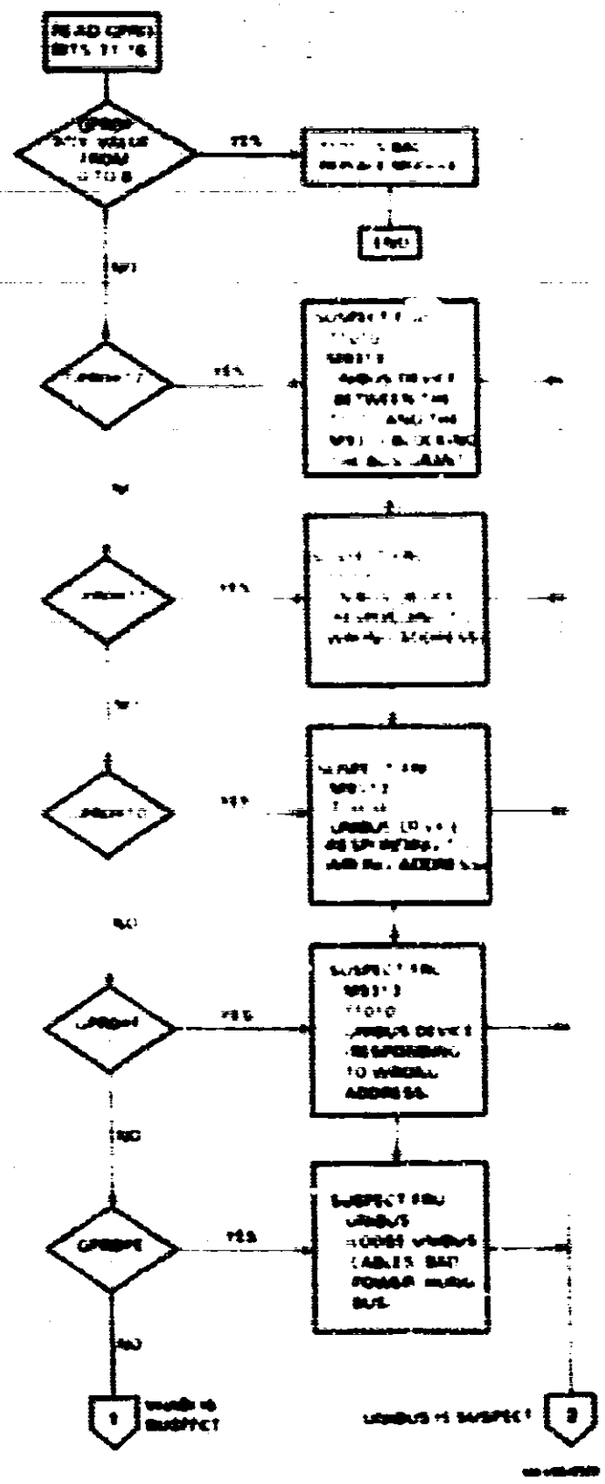


Figure 2-8 Troubleshooting Flow (Sheet 1 of 3)

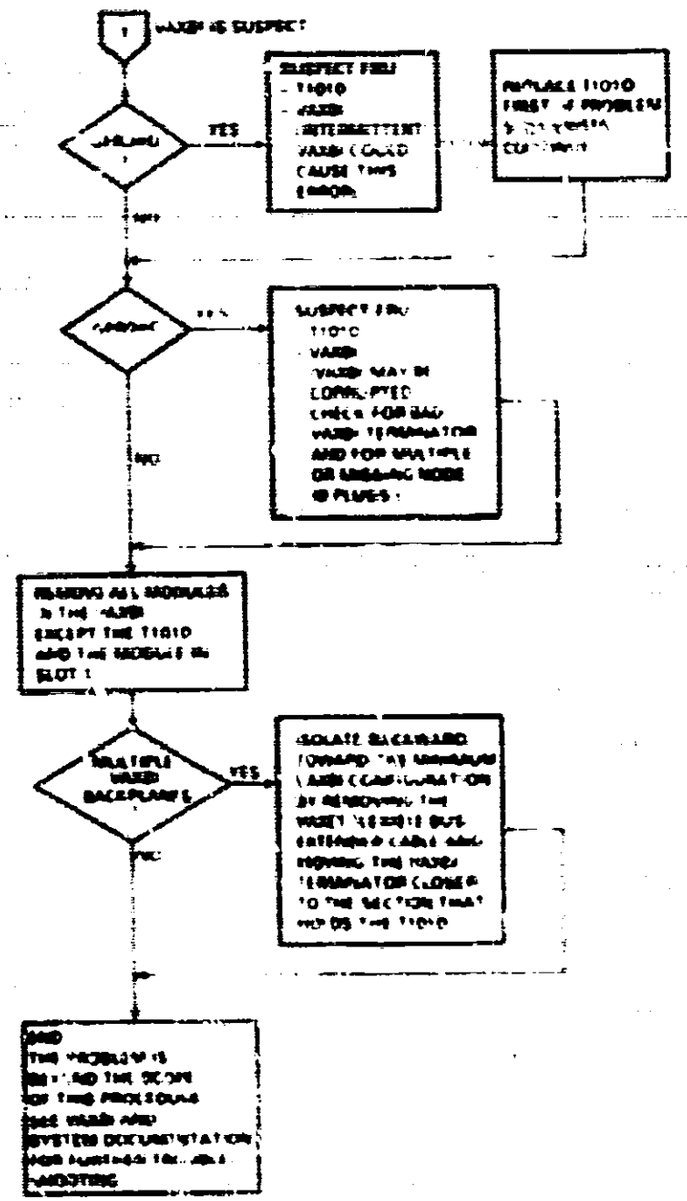


Figure 2-8 Troubleshooting Flow (Sheet 2 of 3)

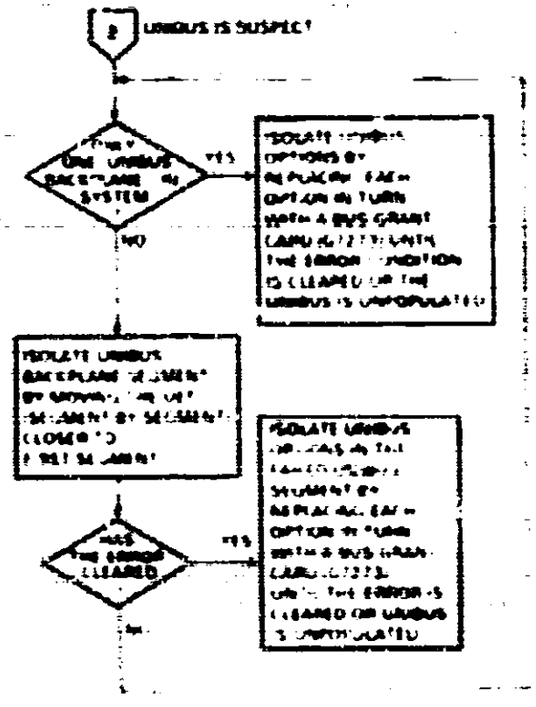


Figure 2-8 Troubleshooting Flow (Sheet 3 of 3)

2.4.3 Helpful Hints

The DWBI A self-test may not run to successful completion if the system includes VAXBI nodes that use burst mode or that excessively stall the VAXBI bus. The DWBI A self-test may fail if the configuration is large and has extensive VAXBI bus activity.

For these hosts if the self-test has passed but the DWBI A still does not work correctly. Most of the items listed relate to the VAXBI S

1 PROBLEM: SSYS timeout errors on VAXBI S device

SUGGESTED ACTION: Verify VAXBI DWBI A node ID and arbitration mode

The DWBI A must be node 0 (receive systems based on BA42 AC AD boxes) and the software must set the DWBI A to fixed-high priority. Verify this by reading the device Type Register (bb+0D) for node 0 to ensure that the device is DWBI A (see Appendix H). Also read the VAXBI Control and Status Register (bb+[illegible]) to verify the contents of [illegible] should be 8 (hex), fixed-high arbitration

PROBLEM: [illegible] operation of I NIB S devices involving several or all slots on the I NIB S

SUGGESTED ACTION: Verify the I NIB S power and quiescent levels (Tables 2-7 and 2-8)

Table 2-7 I NIB S Power

Voltage (Volts)	I NIB S DC Voltage Level (Volts)	P S Ripple pp Maximum (mV)
+5	[illegible]	100
+12	[illegible]	500

Table 2-8 I NIB S Quiescent Levels

Line	Quiescent Level (Volts)
BR NPR	+5 (+/- 5%)
BR 10	+5 (+/- 5%)
BR 10	
(Rb)	[illegible]
All others	[illegible]

SUGGESTED ACTION: Check [illegible]

a Verify that the DWBLA is made to use the I NIB S except systems based on BA11 A (AD based)

b Verify that all NPR grant jumpers were removed from the I NIB S backplane on every slot that has an NPR option

c Check that every complete I NIB S slot contains a grant continuity card

Two different grant continuity cards can be used. The first (7007) goes into the I NIB S backplane slot D and provides grant continuity for the four interrupt (BR4 BR7) but not for the NPR. When the (7007) grant card is used in the empty slots a jumper (C1 to C2 Rb) is needed for NPR grants. The second grant card (7009) provides grant continuity for both BR and NPR grants and is much easier to install

d Verify that the vector and address jumpers are correct for each option and that no two options are selected for the same address or vector

e Use the PAUL11 program to verify the configuration

SUGGESTED ACTION: Verify that the card jumper is removed

Check that no unneeded devices or unsupported devices are on the bus

SUGGESTED ACTION: Look [illegible] I NIB S configuration

Calculate the total loading of the configuration. Make sure that [illegible] and A I NIB S maximum [illegible] Refer to the *PDP 11 Bus Handbook* for details

PROBLEM: One option does not work in the I NIB S slot when the option is installed

SUGGESTED ACTION: Verify that the option is supported in the I NIB S

a Check the KM documents to ensure that the option is supported in the backplane and cabinet

b If the option works but does not work correctly in the system, check the option in a standard I NIB S

c Check that the jumper wire (Jumper W) has been removed from the I NIB S slot in which the option is being installed

PART 2
TECHNICAL
DESCRIPTION

CHAPTER 3

CHAPTER 3
PROGRAMMING

3.1 SYSTEM ADDRESS SPACE

3.1.1 Address Space Divisions

The 1024 megabyte system address space on the VAX 8600 is divided into memory space (from address 0000 0000 through 1111 1111 hexadecimal) and I/O space (from address 2000 0000 through 3111 1111 hexadecimal). Physical memory is assigned addresses starting at 0000 0000. Most I/O space is reserved for special uses.

Figure 3-1 shows the system address space divisions.

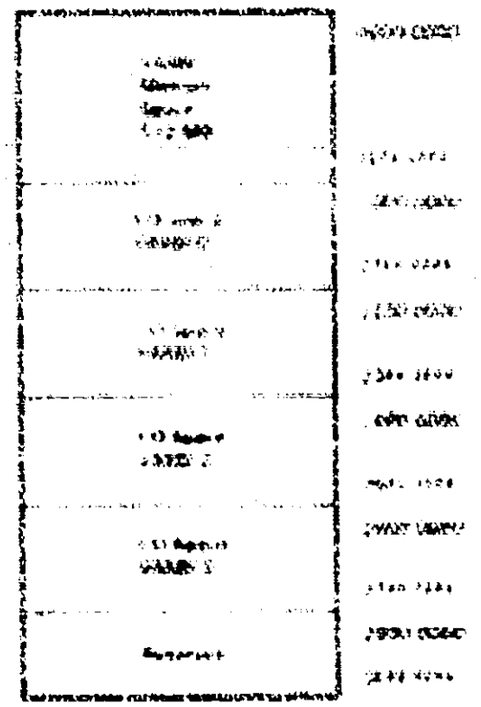


Figure 3-1 System Address Space
(Illustration)

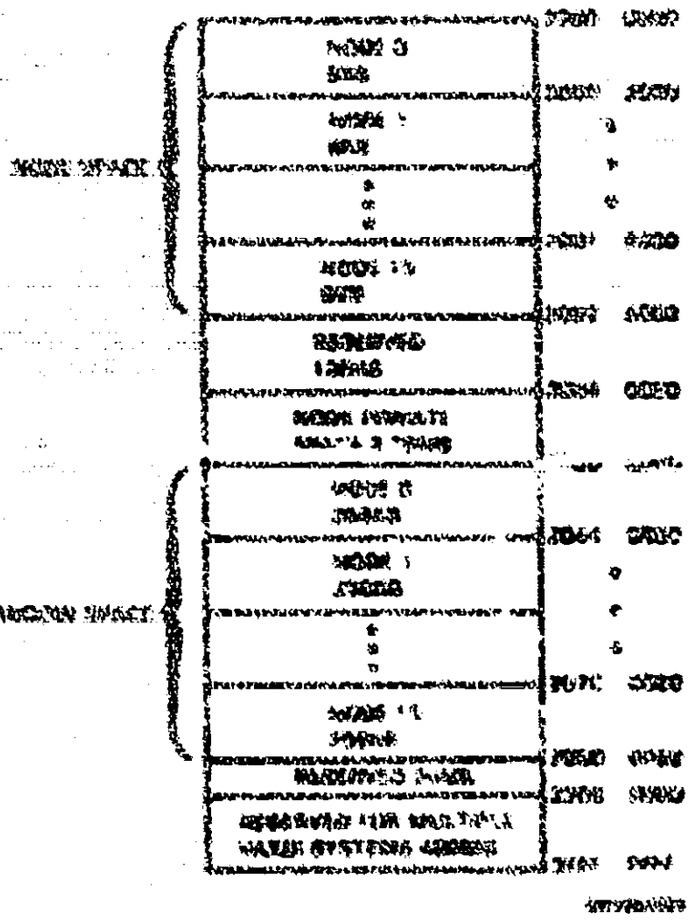


Figure 3-2 Maximum I/O Space

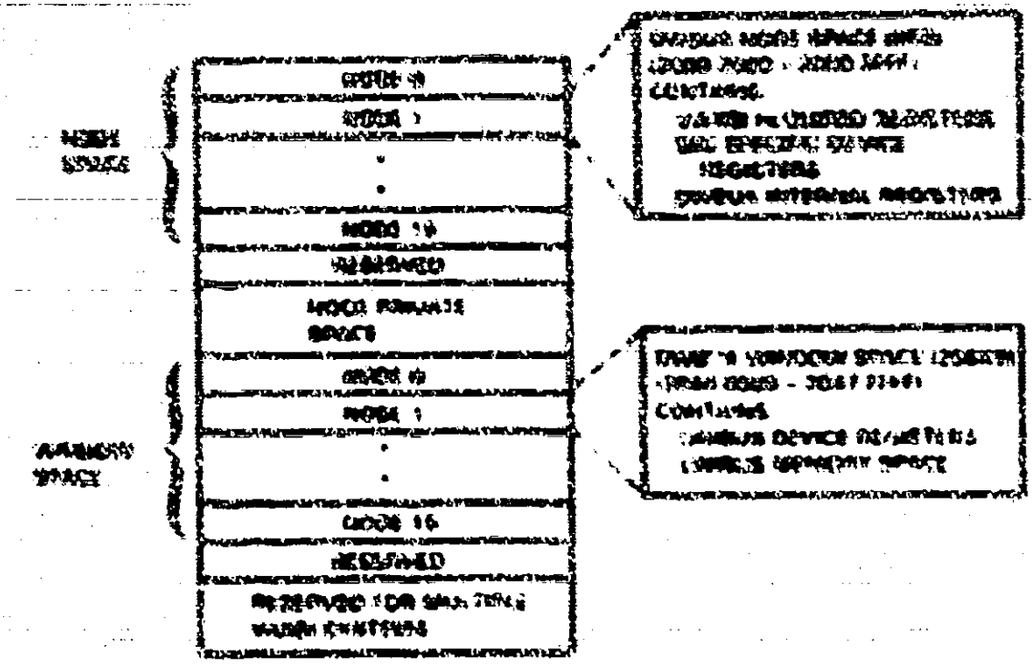


Figure 3-1 IVME134 Bus Address Space

3.3 IVME134 Address Space

The IVME134 adapter's each space is divided into three sections as shown in Figure 3-1 required registers, IOSK space, device registers, and IVME134 registers. Figure 3-1 shows the IVME134 address space.

NOTE

The address of each register is stated as "nh + xxxx." Use the following procedure to calculate the base address, "nh."

1. **Determine the IVME134 adapter's node ID. This is a hexadecimal number between 0 and F.**

2. **Solve for "nh" in the following equation:**

   nh = (2000 0000$_{16}$) + ([node ID]$_{16}$ × [node ID]$_{16}$)

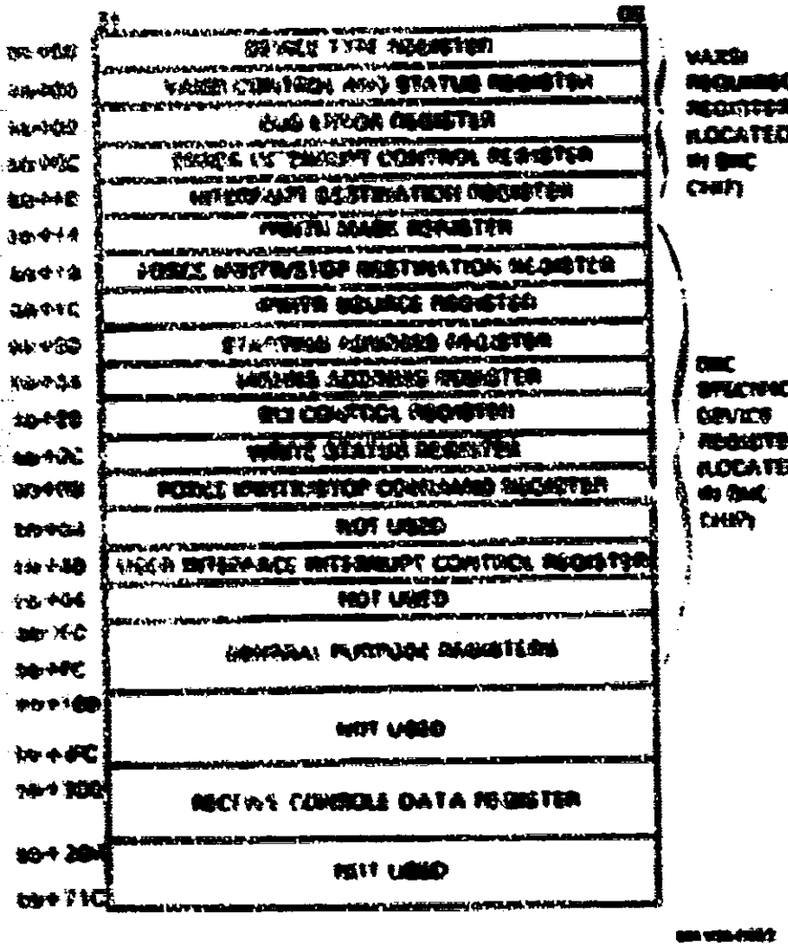


Figure 3-4 DWBR A Address Space (Sheet 1 of 2)

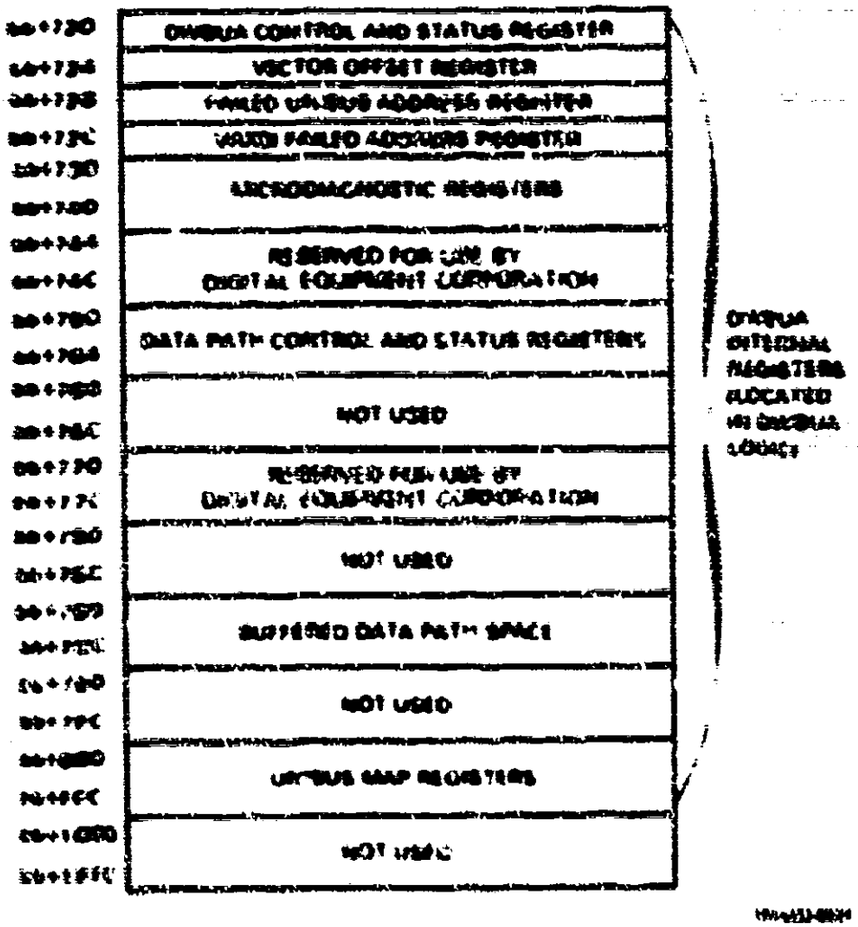


Figure 3-4 DWBR A Address Space (Sheet 2 of 2)

3.2.2 Register Bit Characteristics

The characteristics listed in Table 3-1 can apply to individual bits, to fields, or to entire registers. In the register descriptions in the following sections, the bit characteristics are identified after the name of each bit or field.

Bits indicated as "0" on the register diagrams are not implemented. These bits are READ/ZERO locations that always return "0".

Table 3-1 Register Bit Characteristics

Register Bit Characteristic	Description
DEVICE	Cleared following successful completion of the DWBUA self-test executed by the execution of BOOT 10H
RO	READ/ONLY
R/W	READ/WRITE
SC	Special Case; operation defined in the detailed description
SETUP	Cleared by a SETUP command directed to the DWBUA
W1C	Write 1 to Clear
WO	WRITE/ONLY; always reads 0

3.2.3 VAXBI Required Registers

The VAXBI required registers are implemented in the BIIC on the DWBUA. The discussion that follows focuses on the specific uses of these registers by the DWBUA. The state of each register following successful completion of the DWBUA self-test is included in the discussion of that register. VAXBI required registers that are not described here, or bits that are not included in the register descriptions, are initialized to the state defined in Appendix H.

The DWBUA, as a VAXBI node, is required to implement a number of registers. These registers are:

- Device Type Register
- VAXBI Control and Status Register
- Bus Error Register
- Error Interrupt Control Register*
- Interrupt Destination Register*

NOTE

Registers marked with * are described here in detail.

3.2.3.1 Error Interrupt Control Register – The Error Interrupt Control Register (BIIC+0C) controls the operation of interrupts initiated by a BIIC detected bus error. The LEVEL and VECTOR fields of this register must be initialized by the operating system. These fields are zero after successful completion of the DWBUA self-test. Figure 3-4 is an illustration of the Error Interrupt Control Register.

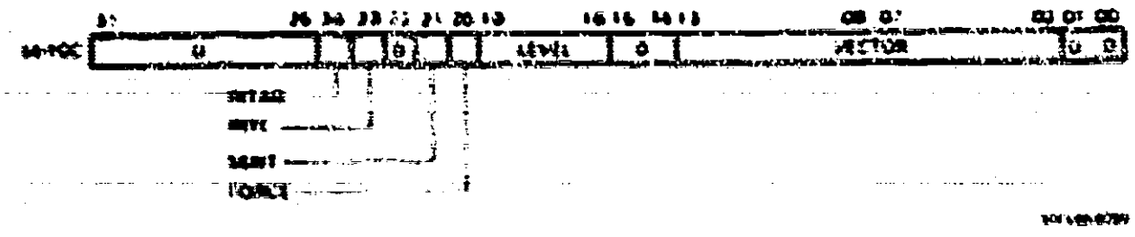


Figure 3-4 Error Interrupt Control Register

INTAH 31	Interrupt abort (W1C, DCLEAR, SC)	This bit is set if a VAXBI INTR command sent by the DWBUA is aborted.
INTC 23	Interrupt complete (W1C, DCLEAR, SC)	This bit is set when the vector for an error interrupt has been successfully transmitted or if a VAXBI INTR command sent by the DWBUA has aborted.
SFAIL 21	Send (W1C, DCLEAR, SETUP, SC)	The DWBUA has sent the VAXBI INTR command, and is awaiting the RSP XT from the VAXBI.
FORCE 20	Force (R/W, DCLEAR)	When this bit is set, the DWBUA forces an interrupt to occur regardless of the state of the Bus Error Register (BIIC+04). The DWBUA sets the FORCE bit when a DWBUA error has occurred and the DWBUA error interrupt enable (EIE) bit is set. The FORCE bit is cleared upon initialization. The operating system must clear this bit after servicing the error interrupt.
LEVEL 19:16	Level (R/W, DCLEAR)	The LEVEL field determines the level at which INTR commands are transmitted under the control of this register. The 0 corresponds to interrupt level A, the 1 to level B, the 2 to level C, and the 3 to level D. The operating system must initialize the LEVEL field.
VECTOR 15:02	(R/W, DCLEAR)	The VECTOR field contains the vector used during error interrupt sequences. It is transmitted when the DWBUA sends a VAXBI INTR cycle on an IDENT transaction that matches the conditions in the Error Interrupt Control Register. The operating system must initialize the VECTOR field.

3.2.2.2 **Interrupt Destination Register** - The format of the Interrupt Destination Register (bb+10) is shown in Figure 3-6.

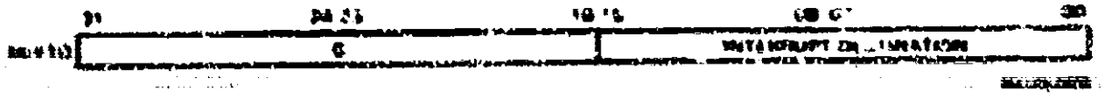


Figure 3-6. Interrupt Destination Register

INTERRUPT DESTINATION (R/W) (INDEX)

The field determines which VAXBI nodes receive INTR commands sent by the IPR08 A. Each bit in the INTERRUPT DESTINATION field corresponds to one VAXBI node. Bit 0 corresponds to node 0, bit 1 to node 1, and so on. During an IDENT command, the destination node ID (VAXBI node number) is compared to the corresponding bit in the INTERRUPT DESTINATION field. The IPR08 A adapter's BIIC responds to the IDENT if that corresponding bit is set and if the level transmitted in the IDENT command matches the level of an interrupt pending in the BIIC.

The IPR08 A will not write bit 0 in the INTERRUPT DESTINATION field, which corresponds to the IPR08 A adapter's VAXBI node ID. The operating system must change this field to reflect the node ID of the interrupt-handler node. If an interrupt occurs before the INTERRUPT DESTINATION field is set by the operating system, the INTAR bit in one of the registers is set. The register in which the INTAR bit is set depends on the type of interrupt (error interrupt, Inter Interface Interrupt Control Register (bb+40), error interrupt Error Interrupt Control Register (bb+48)).

3.2.3 **BIIC Specific Device Registers**

The BIIC specific device registers are implemented in the BIIC on the IPR08 A. The discussion that follows focuses on the specific uses of these registers in the IPR08 A. The state of each register following successful completion of the IPR08 A self-test is included in the discussion of that register. BIIC specific device registers that are not described here, or bits that are not included in the register descriptions, are unrelated to the state defined in Appendix H.

The BIIC specific device registers control IPR08 A-specific functions of the BIIC. The BIIC specific device registers are:

- IPINTR Work Register
- Inter IPINTR VAXBI Destination Register
- IPINTR Source Register
- Scratch Address Register*
- Fetching Address Register*
- BIIC Control Register*
- Write Status Register
- Inter IPINTR VAXBI Command Register
- Inter Interface Interrupt Control Register*
- General Purpose Registers*

NOTE

Registers marked with * are examined here in detail.

3.2.3.1 **Starting Address Register** - The Starting Address Register (bits 31) defines the lower limit of the IFWIB/A adapter's window space. Figure 3-7 is the Starting Address Register format.

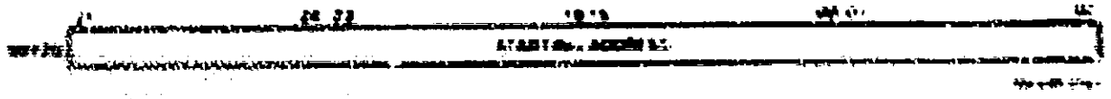


Figure 3-7 Starting Address Register

STARTING ADDRESS
- 31 (RW) -

This field determines bits 20 (MSB) of the lowest 2 MBit S address. The IFWIB/A will not respond to this register the lower limit of the IFWIB/A adapter's window space, based on the node ID of the IFWIB/A. The range is 20000 (HEX) to 207F (HEX) (bits - 17:0 - must be zero)

3.2.3.2 **Ending Address Register** - The Ending Address Register (bits 31) defines the next location after the upper limit of the IFWIB/A adapter's window space. Figure 3-8 is the Ending Address Register format.

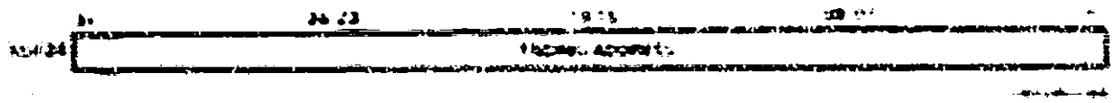


Figure 3-8 Ending Address Register

ENDING ADDRESS
- 31 (RW) -

This field defines bits 20 (MSB) of the highest 2 MBit S address. The IFWIB/A will not respond to this register the upper limit of the IFWIB/A adapter's window space, based on the node ID of the IFWIB/A. The range is 20000 (HEX) to 207F (HEX) (bits - 17:0 - must be zero)

3.3.3.5 [illegible] Control Register - The [illegible] Control Register (bit-24) format is shown in Figure 3-9

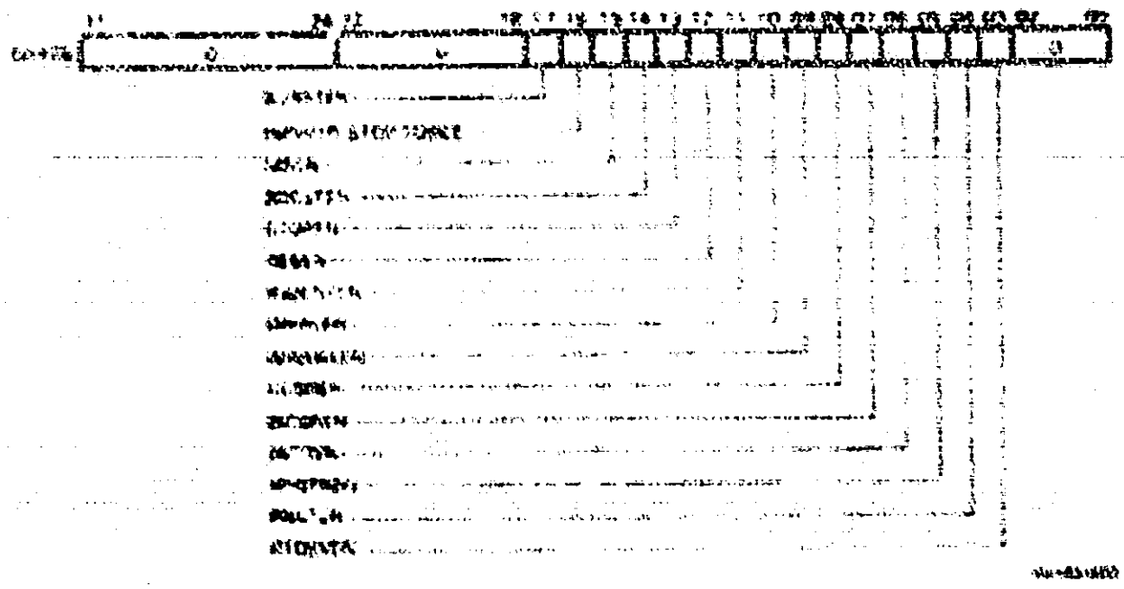


Figure 3-9 [illegible] Control Register

[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]

NOTE:

The [illegible] waits for status from upon execution of the software. All other bits in this register should be clear.

3.3.3.6 User Interface Interrupt Control Register - Figure 3-10 is the User Interface Interrupt Control Register (bit-40) format

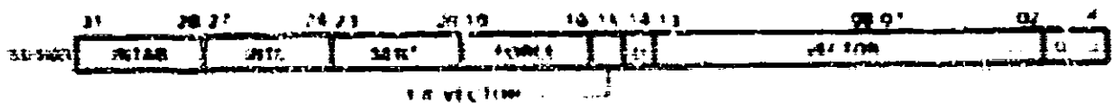


Figure 3-10 User Interface Interrupt Control Register

[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	This bit must be set.
[illegible]	[illegible]	This bit is set by the [illegible] software and it must remain set. It enables the [illegible] to use external control for transfer of [illegible] interrupt requests which have the associated vector valid applied.

3.2.3.5 General Purpose Registers - The sixteen General Purpose Registers (GPRs) used by the 1750A ISA are GPR0 (bits 16-31). Figure 3-11 shows the format of GPR0.

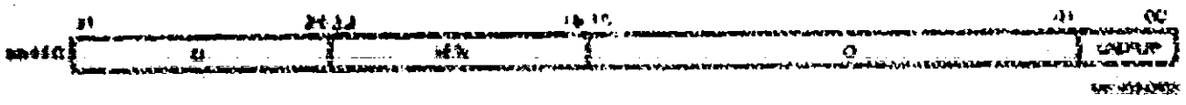


Figure 3-11 General Purpose Registers

[illegible]	Internal Error Number (IEN)	This field is a copy of the IEN field of the [illegible]. This field contains the self-test error number of the 1750A ISA self-test if the 1750A ISA executes software to write to this register. (See Appendix C.) This field is clear if the 1750A ISA self-test passes.
[illegible]	[illegible] Power-Up (PUP)	This bit is set when the [illegible] power is first applied to the 1750A ISA. After [illegible] power goes down. This bit is not reset when external or [illegible] of the 1750A ISA self-test. The 1750A ISA fails self-test if the [illegible] is not powered up.)

General Purpose Registers 1-3 are not used by the 1750A ISA. They are cleared by the BIT self-test.

3.2.4 1750A ISA Internal Registers

With the exception of the Data Path Control and Status Registers and the upper 16 [illegible] Map Registers, all 1750A ISA internal registers are cleared by the 1750A ISA self-test.

3.2.5 Bus Receive Console Data Register - The Bus Receive Console Data Register (BRCD) is considered an unimplemented register by the 1750A ISA. The 1750A ISA responds with NO ACK to any access to this register. An [illegible] that might access the register would have been detected. Figure 3-12 shows the format of the Bus Receive Console Data Register.

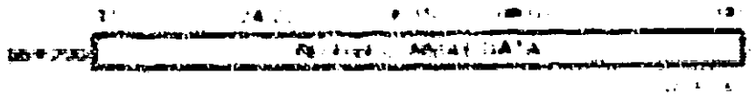


Figure 3-12 Bus Receive Console Data Register

3.2.4.2 DWBI A Control and Status Register - The DWBI A Control and Status Register (BI_CSR, bb+7A0) contains error and other operating information about the DWBI A. Figure 3-11 shows the format of the DWBI A Control and Status Register.

When an error occurs during DWBI A operation, the VAXBI is interrupted if interrupts are enabled. Error interrupts are sent to the VAXBI in two ways:

- The BIIC sends an error interrupt to the VAXBI if an error occurs during a VAXBI transaction.
- The FORCE bit in the Error Interrupt Control Register (bb+0C) is set by the DWBI A if an error occurs either on the DWBI A or during a I_NIBI S operation, and if error interrupts are enabled.

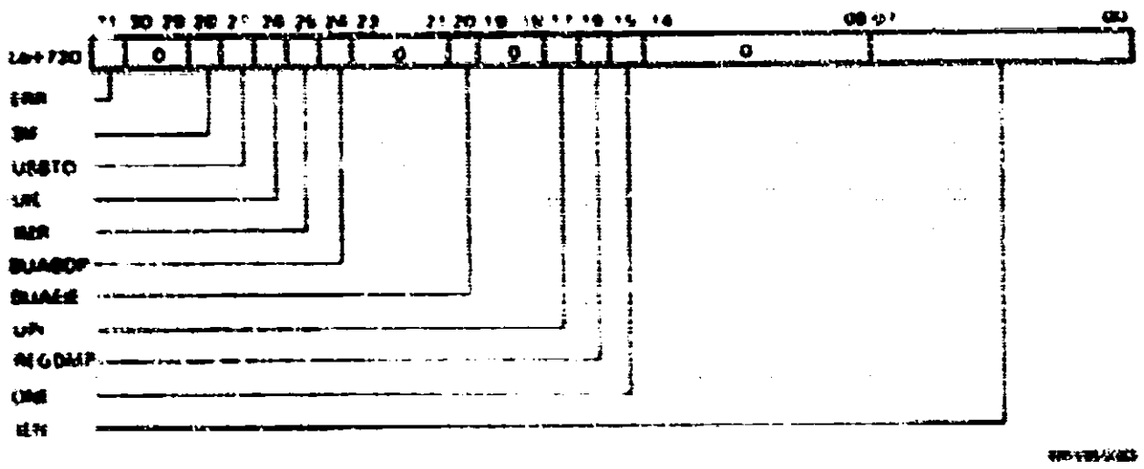


Figure 3-11 DWBI A Control and Status Register

LRR - 31 -	Error (RO, BCI/CI)	This bit is a logical "OR" of all error bits in the BI_CSR.
BII - 28 -	VAXBI Failure (WIC, BCI/CI)	This bit is set if a DWBI A initiated VAXBI transaction fails. A VAXBI failure has occurred if the DWBI A receives any of the following in response to one of its VAXBI commands: – NO ACK – Illegal confirmation code – Read data with data state code See Table 3-1 for a list of BIIC EVENT codes that cause the BII bit to set.
LSSTO - 27 -	I_NIBI S SSYN Timeout (WIC, BCI/CI)	This bit is set when a VAXBI-to-I_NIBI S command attempts access to a I_NIBI S address and does not receive SSYN within 19.2 µs after receiving MSYN.
LII - 26 -	I_NIBI S Interlock Error (WIC, BCI/CI)	This bit is set if a I_NIBI S DAUP command is not immediately followed by a DAUCMD command. This happens when MSYN is dropped by a device after the DAUP command.
IMR - 25 -	Invalid Map Register (WIC, BCI/CI)	This bit is set if a I_NIBI S Map Register (bb+8000 – bb+11FC) which has no VALID bit clear is accessed during a I_NIBI S-to-VAXBI transaction.
DUABDP - 24 -	Dual Buffered Data Path Selected (WIC, BCI/CI)	This bit is set if (maintenance) Buffered Data Path A or B is selected.
DUAIE - 23 -	DWBI A Error Interrupt Enable (R/W, BCI/CI)	If an error occurs, the DWBI A initiates an error interrupt on the VAXBI if this bit is set.
UPI - 19 -	I_NIBI S Power Initialization (WO)	Writing a one to this bit causes a power-up initialization of the I_NIBI S.
MGDMP - 18 -	Microdiagnostic Register Dump (WO)	Writing a one to this bit causes the microcode control to dump its internal registers to the Microdiagnostic Registers (bb+740 – bb+7B0). A RESET of the Microdiagnostic Registers area can then be performed to read the values.
ONE - 15 -	(RO)	The ONE bit is a READ ONLY bit that should always read one. This bit is used for error handling by the operating system, if it reads zero, an error has occurred.
RES - 07:00 -	Internal Error Number (RO)	This field contains the self-test error number of the DWBI A self-test fails and if the DWBI A fails badly enough to write to this register. This field is clear if the DWBI A self-test passes.

3.2.4.3 **Vector Offset Register** - The Vector Offset Register (VOR, bb+0224) contains a 5-bit field which is concatenated with the incoming I/NIBUS vector to form the new VAXBI vector.

The Vector Offset Register format is shown in Figure 3-14.

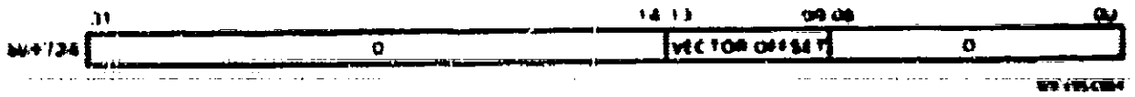


Figure 3-14 Vector Offset Register

VECTOR OFFSET <13:09> (R/W)

The five bits in this field are concatenated with the incoming I/NIBUS vector (I/NIBUS bits <08:02> in the range of 0FC to 1F8) to form the new 14-bit VAXBI vector (1<13:00>). The VECTOR OFFSET field bits are READ/WRITE; they must be set by the operating system.

NOTE

Bits <31:14> and <08:00> must be zero.

3.2.4.4 **Failed I/NIBUS Address Register** - When a VAXBI-to-I/NIBUS transaction results in a SSYN timeout, the failed I/NIBUS Address Register (FUBAR, bb+0228) holds the failed I/NIBUS address sent by the VAXBI master. I/NIBUS address bits <17:02> are stored in FUBAR <15:00>.

The FUBAR is written only on the first occurrence of an address failure. Subsequent failures do not modify the contents of the FUBAR until the LOSTTO bit of the BI_ACSR is cleared.

Figure 3-15 shows the format of the Failed I/NIBUS Address Register.

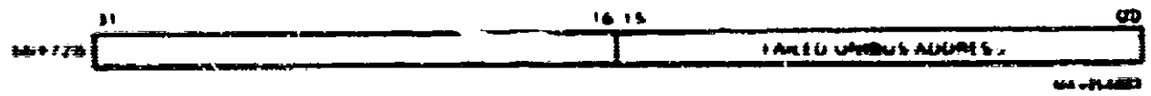


Figure 3-15 Failed I/NIBUS Address Register

FAILED I/NIBUS ADDRESS <15:00> (RO)

This field contains bits <17:02> of the first failed I/NIBUS address. Subsequent failures are not recorded until the LOSTTO bit of the BI_ACSR is cleared.

3.2.4.5 VAXBI Failed Address Register - The VAXBI Failed Address Register (BIFAR, bb+720) holds the address of a failed DWBUA-initiated VAXBI transaction. The BIFAR is written on the first occurrence of a VAXBI address failure only. The BIF bit of the BICSR is set when the BIFAR is written; subsequent failures do not modify the contents of the BIFAR until the BIF bit has been cleared.

Figure 3-16 shows the format of the VAXBI Failed Address Register.

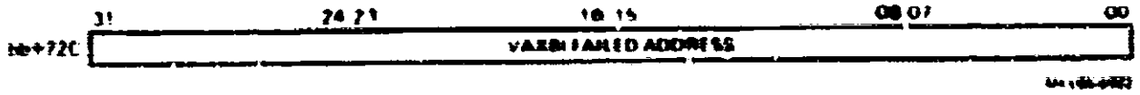


Figure 3-16 VAXBI Failed Address Register

VAXBI FAILED ADDRESS <31:00>	(RO)	This register contains the VAXBI address of the first DWBUA-initiated failure on the VAXBI. Subsequent failures are not recorded until the operating system clears the BIF bit in the BUACSR.

3.2.4.6 Microdiagnostic Registers - The five Microdiagnostic Registers (bb+730 - bb+740) receive the address and status information for the five Buffered Data Paths. This information is received from the microcode control when a one is written to the REGDMP bit in the BUACSR.

The Microdiagnostic Registers are READ-ONLY. A VAXBI WRITE transaction to any of these registers results in a NO ACK response from the DWBUA.

The Microdiagnostic Registers are listed in Table 3-2.

Table 3-2 Microdiagnostic Register Addresses

Microdiagnostic Register for BDP	Address
1	bb+730
2	bb+734
3	bb+738
4	bb+73C
5	bb+740

The five Microdiagnostic Registers have an identical format which is shown in Figure 3-17.

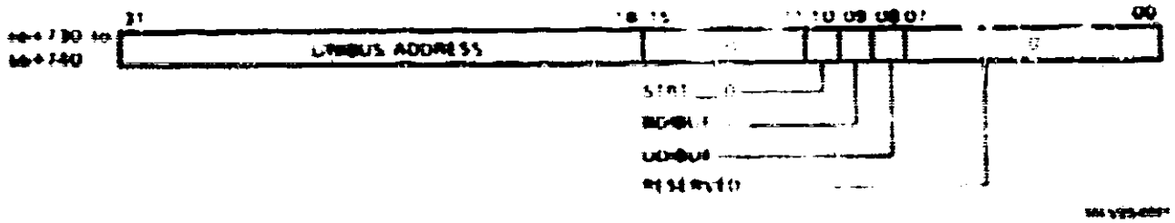


Figure 3-17 Microdiagnostic Register

LVBUS ADDRESS <31:16>	(RO)	This field holds LVBUS address bits <31:16> of the current outbound transfer through the specific Buffered Data Path.
START_0 <10>	(RO)	When set, this bit indicates that the first transaction through the Buffered Data Path began at an aligned or lowword address.
ENABLE <09>	(RO)	This bit is set to indicate that the BDP buffer contains VAXBI data.
LOCBUS <08>	(RO)	This bit is set to indicate that the BDP buffer contains LVBUS data.

3.2.4.7 **Data Path Control and Status Registers** - The DWBI A has six Data Path Control and Status Registers (DPCSR, bb+750 - bb+768). DPCSR0 is for the Direct Data Path, and the remaining five correspond to the five Buffered Data Paths. The addresses of the Data Path Control and Status Registers are shown in Table 3-3.

Table 3-3 Data Path Control and Status Register Addresses

DPCSRn	Address
DPCSR0	bb+750
DPCSR1	bb+754
DPCSR2	bb+758
DPCSR3	bb+75C
DPCSR4	bb+760
DPCSR5	bb+764

The six Data Path Control and Status Registers have an identical format, which is shown in Figure 3-18.

Figure 3-18 Data Path Control and Status Register

DPSEL 15:12	Data Path Select (R/W)	These three bits define the data Path (0 - Direct Data Path, 1 - 5 corresponds to the five Buffered Data Paths. This field is written by the DWBI A software.
PURGE 10	Purge (WO)	When set by the operating system, the PURGE bit causes the specific BDP buffer to be purged. This is a WRITE ONLY bit.

Purging a BDP buffer has different effects depending on the buffer's status. For DPCSR0 (the Direct Data Path Control and Status Register), the BDP buffer is not purged and no further action occurs. For the other five Data Path Control and Status Registers, writing a one to the PURGE bit has the following results:

VAXBI to data in buffer	The data is written to the VAXBI and the flags are cleared, indicating that the buffer is empty.
VAXBI data in buffer	The flags are cleared to indicate that the buffer is empty.
Empty buffer	No action occurs.

3.2.4.8 **Buffered Data Path Space** - The memory buffers associated with each Buffered Data Path are addressable in DWBI A I/O space (bb+780 - bb+7BC). Although the BDP buffers are not usually accessed directly through software, they are READABLE and are supposed accessible for diagnostic purposes.

3.2.4.9 **VMRS Map Registers** - The 512 VMRS Map Registers (bb+800 - bb+FFC) are READ/WRITE accessible to the operating system. These registers are initialized by writing zeros to them. VAXBI bus reset does not affect them.

A VMRS Map Register translates an 18-bit VMRS address into a 30-bit VAXBI address. The translation is illustrated in Figure 3-19.

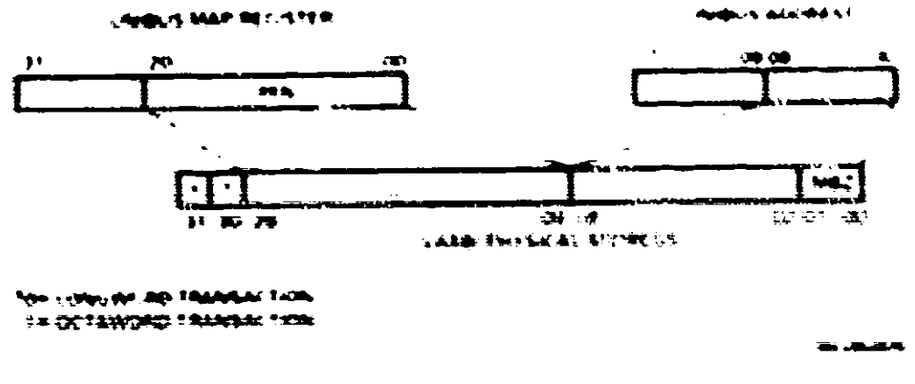


Figure 3-19 VMRS to VAXBI Address Translation

The VMRS Map Register format is shown in Figure 3-20.

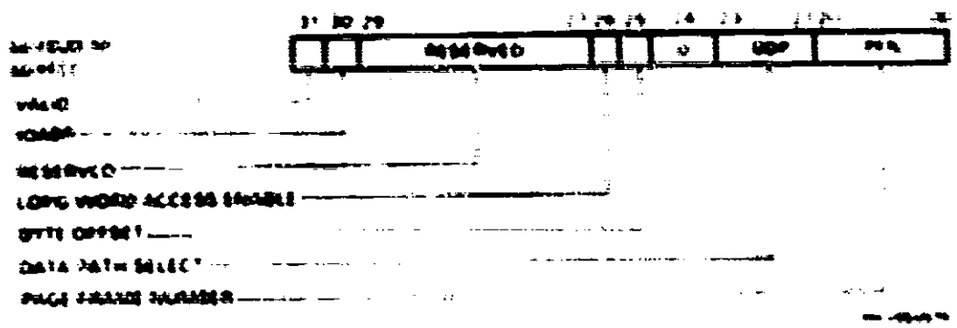


Figure 3-20 VMRS Map Register

VALID - 31 -	(R/W, DCLOC)	Clearing this bit prevents a UNIBUS transfer from mapping to the VAXBI. A malfunction that uses a UNIBUS Map Register with a clear VALID bit does not receive DIVEs. When this happens, the SWR bit in the BI ACSR is set and an error interrupt is sent to the VAXBI (if interrupts are enabled).
IOADR - 30 -	(R/W, DCLOC)	This bit designates I/O address space. When a UNIBUS device requests a transfer to a UNIBUS Map Register with contents 11111111 (hex), the DWBI A ignores the transfer. That is, the DWBI A does not assert SSYN, does not set the SWR bit in the BI ACSR, and does not cause an interrupt. (The transfer is ignored if the IOADR, VALID, LWAIT, and BYTE OFFSET bits are set and if DPSEL - 20 - is 0 or 7.) If IOADR and VALID are set, but neither of the other bits are set, the DWBI A sets SWR, causing an interrupt.
LWAIT - 29 -	Longword Access Enable (R/W, DCLOC)	When set, this bit specifies that the maximum length of a backplane transaction is one longword. The buffer is prepared for an extended WRITE operation when an aligned longword or three bytes have been received. When cleared, the backplane transaction length must be as long as one octaword before the transfer are sent to the VAXBI. This bit is ignored when set in a UNIBUS Map Register with the Direct Data Path selected.
BYTE OFFSET - 28 -	(R/W, DCLOC)	When clear, this bit specifies that UNIBUS addresses in transfers to or from memory are octaword aligned.
DATA PATH SELECT - 27:26 -	(R/W, DCLOC)	This 2-bit field determines which of the data paths is used. A 0 in this field selects the Direct Data Path; 1 through 3 correspond to the four Buffered Data Paths. A 1 in bit 2 in the field causes the DWBI A to assert the BAPAR(H) bit in the BI ACSR. The DWBI A then sends an error interrupt to the VAXBI if interrupts are enabled.
PAGE FRAME NUMBER - 25:00 -	(R/W, DCLOC)	This 26-bit address field is concatenated with UNIBUS address bits <8:0> to create a 35-bit physical address on the VAXBI.

3.3 INITIALIZATION

3.3.1 DWBI A Hardware Initialization

Upon successful completion of the self-test, all DWBI A internal registers and Buffered Data Path buffers are cleared with the exception of the Data Path Control and Status Registers and the upper 16 UNIBUS Map Registers.

3.3.2 UNIBUS Initialization

The UNIBUS can be initialized in several ways:

1. The DWBI A monitors the UNIBUS RESET signal. When the signal is asserted the DWBI A initializes the UNIBUS (but not the DWBI A) clears the UBIC P bit in CSPR0, and, if interrupts are enabled, issues an error interrupt. When UNIBUS RESET is deasserted and UNIBUS initialization is complete, another interrupt is sent if interrupts are enabled.

2. The DWBI A initializes the UNIBUS (but not the DWBI A) if a processor sets the UBIC bit in the BI ACSR. Two interrupts are issued if interrupts are enabled.

3. The DWBI A asserts UNIBUS RESET whenever BI RESET is asserted (unless a hardware error or bus hang that affects the VAXBI causes the UNIBUS to be initialized).

4. The DWBI A asserts UNIBUS RESET when a processor sets the SINI bit in the BI ACSR. This mechanism for initializing the DWBI A also initializes the UNIBUS.

The state diagram for UNIBUS initialization is shown in Figure 3-21 and is explained in the following paragraphs.

During UNIBUS initialization, which takes at least 40 ms, the UBIC P bit in CSPR0 is cleared by the DWBI A, indicating that UNIBUS power is down. The DWBI A sends an error interrupt to the VAXBI if interrupts are enabled.

During UNIBUS initialization the DWBI A internal registers are not accessible. The DWBI A sends an RETRY response to all WRITE commands from the VAXBI and ignores the data and READ commands are supplied with zero data. The BIIC registers may be accessed, and they respond normally to all VAXBI transactions. Once power is restored to the UNIBUS, UBIC P is set in CSPR0 and the VAXBI is interrupted if interrupts are enabled.

3.4 PROGRAMMING CONSIDERATIONS

3.4.1 UNIBUS Map Registers

The DWBI A supports 512 UNIBUS Map Registers. When the VALID bit in a UNIBUS Map Register maps one 512-byte page of UNIBUS address space to a page of VAXBI space.

The user must ensure that VALID pages do not correspond to UNIBUS addresses of devices on the UNIBUS. To do this, leave the contents of the upper 16 UNIBUS Map Registers unchanged after DWBI A initialization.

The upper 16 UNIBUS Map Registers are initialized to 11111111 (hex). If memory is placed on the UNIBUS for special applications, the UNIBUS Map Registers which correspond to that memory should be initialized to 11111111. This is the responsibility of the application software. The DWBI A ignores any transaction involving a UNIBUS Map Register that contains 11111111.

The lower 496 UNIBUS Map Registers are initialized to zero (also known as uninitialized - that is, those VALID bits are cleared) by the DWBI A on receipt of SINI (10).

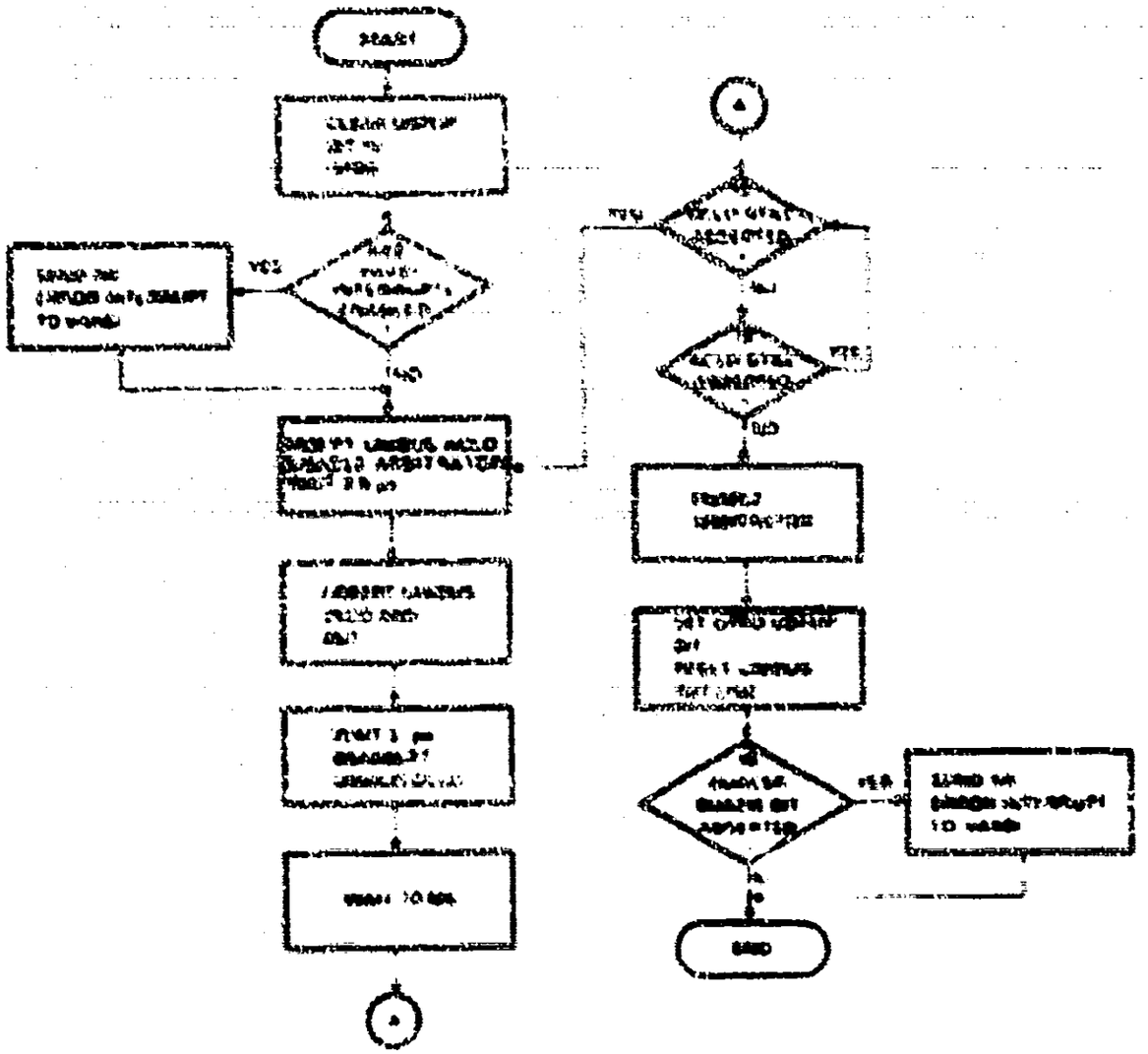


Figure 3-21 [illegible] Initialization State Diagram

[illegible] a [illegible] driver initiates a transfer that corresponds to one of the upper 16 [illegible] Map Registers, the DWBR [illegible] ignores the transfer and captures the [illegible] address in the [illegible] initiated transfer accesses a [illegible] Map Register with the [illegible] PPH [illegible] and BYTE OFFSET bits inverted and DPN[1:0] equal to 0, the DWBR [illegible] the corresponding [illegible] transfer.

3.4.1.1 Contiguous Allocation - One or more [illegible] Map Registers must be allocated for each transfer. When more than one is allocated, the [illegible] Map Registers must be contiguous in [illegible] space. Since sequential transfers are contiguous in [illegible] space, this means that the set of [illegible] Map Registers is contiguous in [illegible] address space. The contents of the [illegible] Map Registers do not normally point to a contiguous area of [illegible] memory space.

3.4.1.2 Mapping to [illegible] I/O Space - [illegible] Map Registers can be used to map to [illegible] I/O space, although the reasons for doing so are known and under [illegible]

A [illegible] driver cannot modify the [illegible] Map Registers of the DWBR. [illegible] An attempt to modify a [illegible] Map Register results in the [illegible] driver never receiving [illegible] address the DWBR [illegible]

3.4.1.3 BYTE OFFSET Bit - If the BYTE OFFSET bit in the [illegible] Map Register is set and if the transfer uses [illegible] Map Registers, then the BYTE OFFSET bit should be set for all registers and the n+1th register should be allocated and unallocated. If the n+1th register is [illegible] where [illegible] driver issues a DATO with [illegible] address corresponding to the last word of the nth page, then two [illegible] WRITE cycles occur: one updates the last byte of the nth page and the other updates the first byte of the n+1th page.

If the BYTE OFFSET bit is not set, it is not necessary to allocate the n+1th [illegible] Map Register when the DWBR does not perform data from [illegible] memory space.

3.4.2 [illegible] Power Down

When [illegible] power goes down, the [illegible] requires a minimum of 200 ms to complete its power down/power up sequence. During the sequence, the DWBR cannot respond normally to [illegible] transfers. Any attempted access to nonexistent memory or to node space (except the first 764 bytes which are RAM space) receives NAK. For WRITE commands, the DWBR ignores the data. For READ commands, the DWBR returns zero data. Further, the IRQ command does not set the DWBR adapter [illegible] interrupt.

3.4.3 Use of Buffered Data Paths

[illegible] memory may be corrupted if a [illegible] driver issues unsequential DATO(B) transactions through a BDP. In particular, a DATO with [illegible] address n+a followed by a DATO with [illegible] address n+a+16 causes the entire data stored in the BDP to be written to [illegible] memory space. A DATO(B) with [illegible] address n+a followed by a DATO with [illegible] address n+a+16 are the same effect. This conforms to the standard restriction on [illegible] which use BDPs (sequential transfers only) and causes the programming restrictions described in the next two paragraphs.

[illegible] Map Registers associated with BDPs must not be double-allocated. A set of [illegible] Map Registers may be allocated to only one transfer. Concurrently allocating a set of [illegible] Map Registers to two transfers may cause VAXBI memory space to be corrupted.

A BDP must be purged (by writing one to the PURGE bit in the corresponding DPR[N]) before the [illegible] Map Registers allocated to a transfer may be allocated to another transfer, and before the contents of the [illegible] Map Registers may be changed.

After a [illegible] power outage occurs and power is restored, all of the BDPs must be purged using the PURGE bit in the DWSR.

During a [illegible] Buffered Data Path transaction, the DWBUA times MSYN before determining if the corresponding VAXBI transaction is required. (That is, the DWBUA times MSYN before determining if the buffer is full.) This means that if an error occurs during the VAXBI transfer the DWBUA cannot report that error to the [illegible] driver. If the transaction is a DATI, the DWBUA completes the corresponding VAXBI transfer before it times MSYN to the [illegible] driver.

3.4.6 VAXBI Access to the DWBUA Internal Registers.
All IDC transactions to the DWBUA internal registers are treated as READ commands and do not set the interlock on the DWBUA. All LOCK and WRITE transactions to DWBUA registers are treated as WRITE transactions, and the mask bits are ignored by the DWBUA.

The DWBUA responds with NO ACK to all accesses to the unused register locations in the DWBUA internal register space. A WRITE or [illegible] WRITE transaction to the READ/ONLY registers of the DWBUA also results in a NO ACK response.

3.4.5 Data Integrity
The DWBUA responds only to VAXBI transactions with a data length of longword, Quadword, octaword, and READ/WRITE data length transactions result in a NO ACK response.

3.4.6 [illegible] WRITE commands
When an IDC transaction is routed to a DWBUA adapter's window space, the DWBUA first performs a DATIP transaction on the [illegible]'s master to obtain the address supplied with the IDC command. The DWBUA then sets its interlock. Once interlocked, the DWBUA responds with RETRY to all transactions routed to DWBUA's window space or node space (except IDC space) until a [illegible] WRITE transaction is received. The DWBUA ignores the address supplied with the [illegible] WRITE command. The DWBUA assumes that the [illegible] WRITE is addressed to the same board as the IDC command and performs the DATOB(E) to that board. Address parity may accompany the mask bits supplied with the [illegible] WRITE data.

3.4.7 [illegible] DATIP
When a [illegible] driver issues a DATIP, the DWBUA responds with RETRY to any VAXBI transaction routed to DWBUA's window space or node space (except IDC space) until a DATIP(E) is sent to the [illegible] master driver.

3.4.8 Hung [illegible]
If a [illegible] driver hangs on a [illegible] interrupt, by not dismissing MSYN, the DWBUA will RETRY any VAXBI transactions routed to DWBUA's window space or node space (except IDC space).

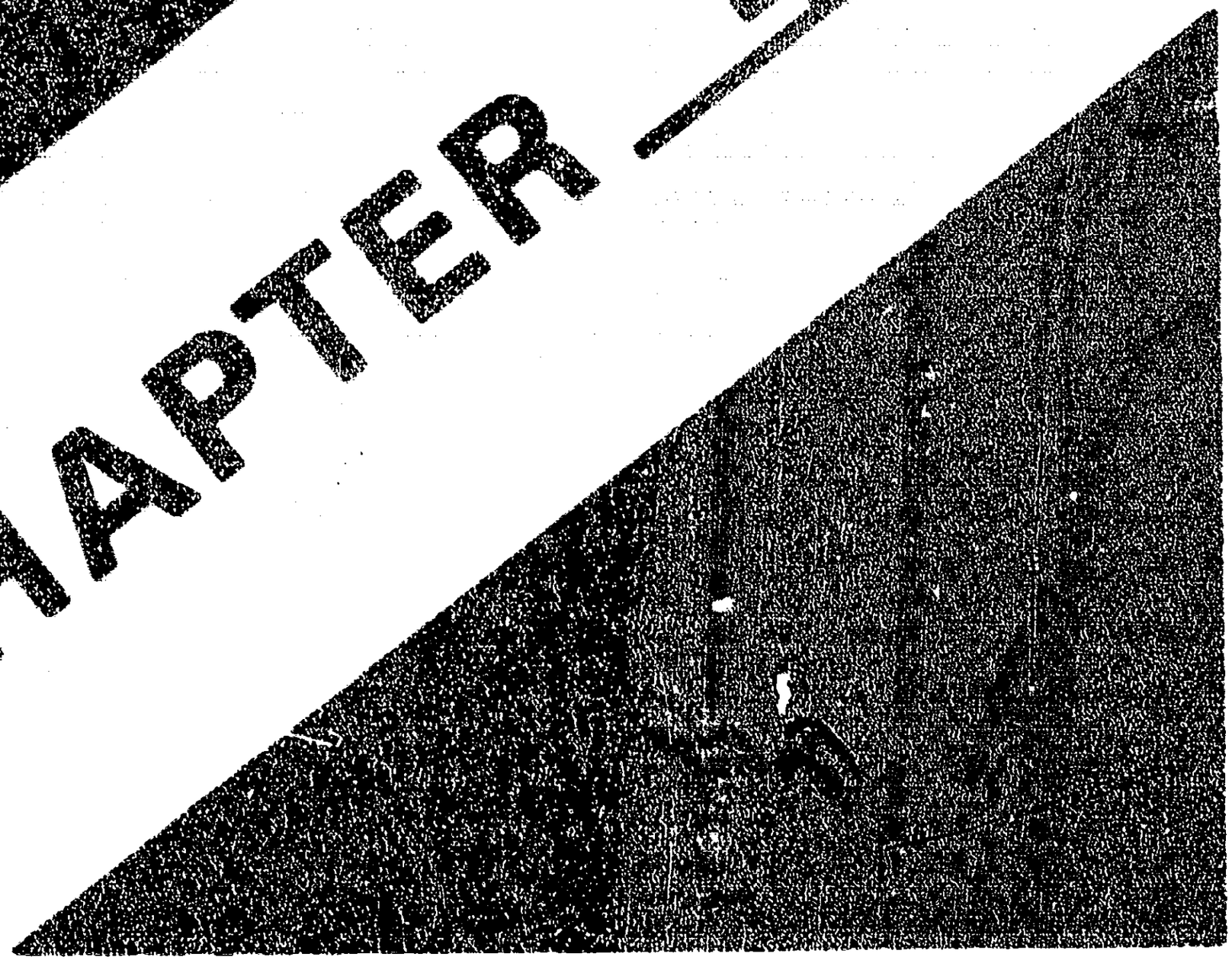
3.4.9 VAXBI Bus Errors
If the DWBUA encounters an error on the VAXBI during a DWBUA-initiated transaction, it sets the BUS ERR bit and the ACCLO. The DWBUA also clears the mask bits and the associated BDP flags, thereby effectively emptying the buffer in readiness for the next BDP. If the error occurs during a [illegible] transaction, the interlock resets at the data path for which the VAXBI transaction failed. The DWBUA then withholds MSYN, resulting in this transaction to the [illegible] driver that initiated the transfer.

3.4.10 [illegible] Devices
The DWBUA allows those [illegible] devices that perform data transfers (instead of sending vectors) during the INTR cycle to be attached to the [illegible]. These devices, however, cause a passive release every time that causes the BR lines to participate in DMA transfer.

3.4.11 Access to Nonexistent Registers
The DWBUA responds with NO ACK to any VAXBI command with an address in unused DWBUA register space. It also responds with NO ACK to WRITE (WRITE, WRITE, WRITE) commands to READ-ONLY registers.

READ transactions to unimplemented IDC registers read zero data. WRITE (WRITE, WRITE, WRITE) commands to these registers receive an ACK response, but the data is dropped.

CHAPTER – 4



CHAPTER 4
FUNCTIONAL DESCRIPTION

4.1 [illegible]

The functional description of the [illegible] is presented in two parts. In the first part, the components on the block diagram are described. The second part explains the way in which the [illegible] interfaces between the two buses.

4.2 [illegible]

Figure 4-1 is the [illegible]'s block diagram. Table 4-1 contains functional descriptions of the blocks in Figure 4-1.

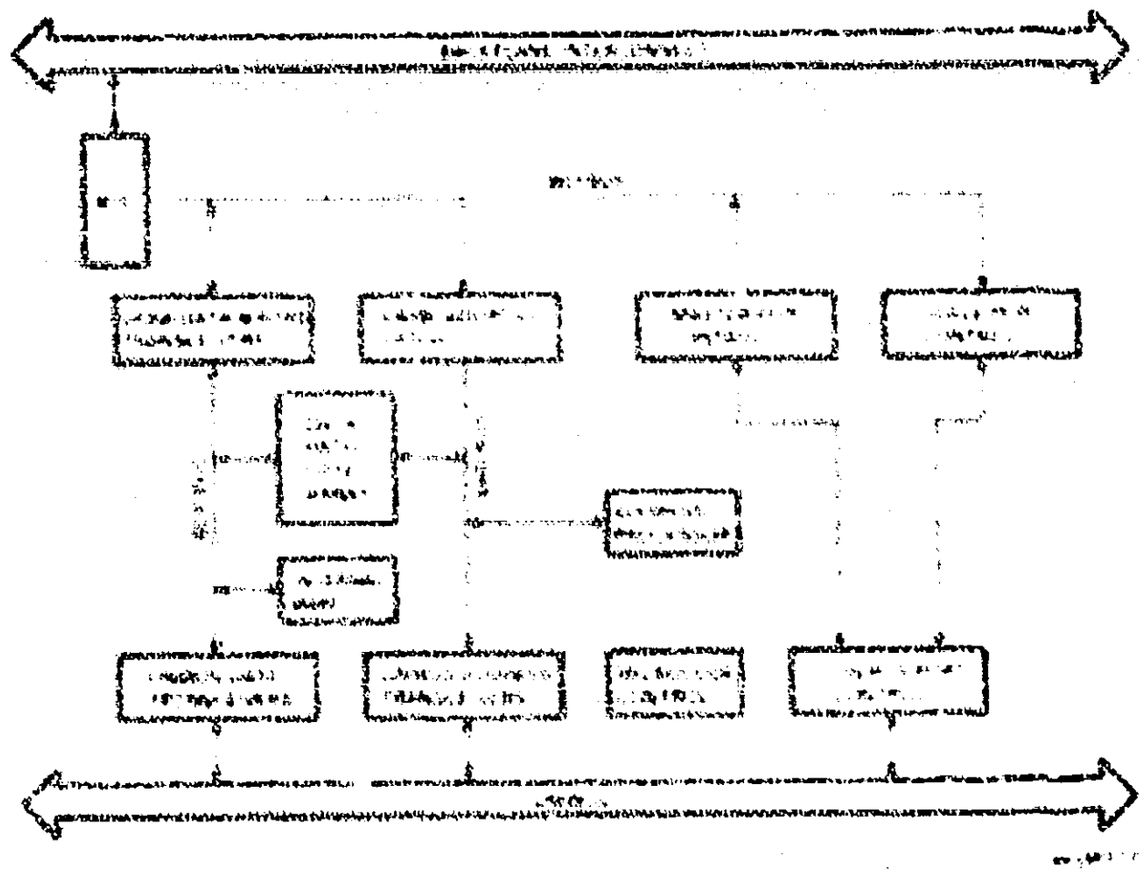


Figure 4-1 [illegible]'s Block Diagram

Table 4-1 [illegible]

Block	Description
[illegible]	[illegible]
Master Port Control	[illegible]
Slave Port Control	[illegible]
VAX/VME Data and Address Transceivers	[illegible]
VAX/VME Address Latch	[illegible]
[illegible] Data Transceivers	[illegible]
[illegible] Address Transceivers	[illegible]
[illegible] Port Control	[illegible]
Data Path Gate Array	[illegible]

Table 4-1 [illegible] (Continued)

Block	Description
[illegible]	[illegible]
Address Decoder	[illegible]
Miscellaneous Control	[illegible]
SCB Chip	[illegible]
NVRAM Chip	[illegible]

4.3 [illegible]

The [illegible] serves a translation between the [illegible] and the [illegible] It interprets commands received from each bus and in turn causes that other bus to understand and to perform commands and responses that emulate the properties of those commands. These sequences of commands, commands and responses are called [illegible] transactions. In the section below, typical transactions handled by the [illegible] are summarized in detail.

[illegible] transactions are divided into three categories:

- [illegible]
- [illegible]
- [illegible]

4.3.1 [illegible] Transactions

4.3.1.1 [illegible] Response to [illegible] Transactions - The [illegible] and READ commands in the [illegible] The purpose of these commands is to read data from or to write data to the [illegible] internal registers. The [illegible] mode that initiates the transaction is the [illegible] master, and the [illegible] is the [illegible] slave in all [illegible] transactions.

Table 4-2 [illegible] Response to [illegible] Transactions

[illegible] Transaction	[illegible] Response
READ of [illegible] internal register	1 STALL 2 Requests data each read data strobe cycle 3 ACK
READ of unused [illegible] register space	NO ACK
WRITE to [illegible] internal register	1 STALL 2 ACK if no parity error on the VAXBI[1] 3 Register updated
WRITE to unused [illegible] register space or READ [illegible] register	NO ACK

[1] If a parity error occurs, the register is not updated.

4.3.1.2 [illegible] Commands - [illegible]

Table 4-3 [illegible] Commands

[illegible] Command	[illegible] Response	Possible Errors	See Note
READ	ACK, RETRY	None	1
IREAD	ACK, RETRY	None	1,2
RCI	ACK, RETRY	None	1
WRITE	ACK, RETRY	Parity Error	1,4
WCI	ACK, RETRY	Parity Error	1,4
MASK	ACK, RETRY	Parity Error	1,3,4
IMASK	ACK, RETRY	Parity Error	1,3,4

NOTES FOR TABLE 4-3:

(1) Unsupported length code

(2) IRCI commands are accepted, but they are treated as RCI commands. The [illegible] does not interlock.

(3) IMASK commands are accepted, but they are treated as MASK commands. The [illegible] does not interlock. The mask bits are ignored.

(4) MASK commands are accepted, but they are treated as WRITE commands. The mask bits are ignored, and the full longword of data is assumed to be valid.

(5) If a parity error occurs on the VAXBI, the [illegible] ignores the transaction.

4.3.1.3 Example: VAXBI WRITE to a I NIBI S Map Register - The VAXBI-to-DWBI A transaction used as an example in this section is a VAXBI WRITE to a I NIBI S Map Register. The purpose of this transaction is for the operating system to set up a I NIBI S Map Register for a future I NIBI S-to-VAXBI transaction. A I NIBI S Map Register corresponds to a block of addresses on the I NIBI S. In a future direct memory access (DMA) transaction (not necessarily the one following this transaction), data will be transferred between this block of I NIBI S addresses and a VAXBI address.

Figure 4-2 is a flow diagram of the VAXBI WRITE to a I NIBI S Map Register transaction. The numbered paragraphs that follow refer to the corresponding numbers on Figure 4-2.

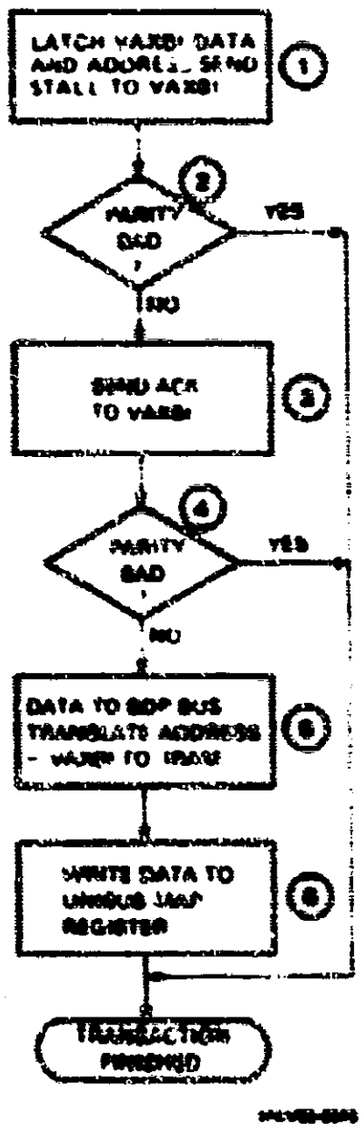


Figure 4-2 VAXBI WRITE to a I NIBI S Map Register Flow Diagram

1. The VAXBI command, address, and data are received by the DWBI A. The state sequence control determines that the transaction is for the DWBI A. The VAXBI address is latched in the VAXBI address latch, and the VAXBI data is latched in the VAXBI data and address transceivers. The VAXBI address is the address of the I NIBI S Map Register that will be written. The VAXBI data will be written into the I NIBI S Map Register.

   The initial DWBI A response to the VAXBI is STALL.

2. The DWBI A checks for a parity error on the VAXBI. If either the data or the command/address has a parity error, the transaction is immediately terminated.

3. The DWBI A responds to the VAXBI with ACK. The VAXBI interprets the transaction as complete.

4. The DWBI A checks for a parity error on the VAXBI. The DWBI A terminates the transaction immediately if the data received in the VAXBI cycle in which ACK was sent has a parity error. The DWBI A issues an error or corrupt to the VAXBI if errors are enabled.

5. The data in the VAXBI data and address transceivers is put onto the BDP bus.

   The VAXBI address is translated into an internal RAM address specifically for the address of the I NIBI S Map Register to be written.

6. The data on the BDP bus is written to the I NIBI S Map Register in the internal RAM. The transaction is complete.

4.3.2 VAXBI-to-I NIBI S Transactions

4.3.2.1 DWBI A Response to VAXBI-to-I NIBI S Transactions - In a VAXBI-to-I NIBI S transaction, the VAXBI master sends to the DWBI A a command that requires the DWBI A to read from or write to a I NIBI S device.

Table 4-4 Bus Masters and Slaves for VAXBI-to-I NIBI S Transactions

Bus	Master	Slave
VAXBI	Node initiating transaction	DWBI A
I NIBI S	DWBI A	I NIBI S device

The DWBI A monitors the I NIBI S BUSY signal before it attempts to perform the DATO(B) transaction on the I NIBI S. If BUSY is deasserted, the DWBI A asserts BUSY and gains I NIBI S mastership. If the DWBI A does not gain I NIBI S mastership within 51 µs, I NIBI S timeout occurs.

Table 4-5 details the DWBUA responses to three types of VAXBI commands that require DWBUA interaction with devices on the UNIBUS: READ, WRITE (WMCI, WCI), and IRCI/UWMCI.

Table 4-5 DWBUA Responses to VAXBI-to-UNIBUS Transactions

VAXBI-to-UNIBUS Transaction	DWBUA Response
READ	Initial response — STALL Initiates UNIBUS DATI command Continues STALL responses to VAXBI until • SSYN received from UNIBUS slave, or • SSYN timeout occurs (1) Sends to VAXBI • Data from UNIBUS • Read data status code (2) • ACK
WRITE (WMCI, WCI)	Initial response — STALL Checks for parity error on the VAXBI (3) Sends ACK to VAXBI Initiates UNIBUS DATO(B) command (4) Waits for SSYN from UNIBUS device or for SSYN timeout (5)
IRCI/UWMCI	
IRCI	Initial response — STALL Initiates UNIBUS DATIP command Continues STALL responses to VAXBI until • SSYN received from UNIBUS slave, or • SSYN timeout occurs (1) Sends to VAXBI • Data from UNIBUS • Read data status code (2) • ACK Sets interlock (6)
UWMCI (7)	Initial response — STALL Checks for parity error on the VAXBI (8) Sends ACK to VAXBI Releases interlock Initiates UNIBUS DATO(B) command (4,9)

NOTE

When the DWBUA is busy, it sends RETRY to the VAXBI. It does this when:

- **VAXBI attempts access of UNIBUS address space while the DWBUA is processing a UNIBUS transaction, or**
- **Current transaction requires DWBUA mastership of the VAXBI.**

NOTES FOR TABLE 4-5:

(1) The DWBUA does the following in response to a SSYN timeout during a READ transaction:

a. Sends zero data with read data substitute status code to the VAXBI.

b. Sends an ACK response to the VAXBI.

c. Sets the UNSTO bit in the BIACSR.

d. Issues an error interrupt to the VAXBI if interrupts are enabled.

(2) If the UNIBUS PB (parity bad) line is asserted, the DWBUA sends zero data with a read data substitute status code to the VAXBI.

(3) If a parity error has occurred on the VAXBI, the DWBUA terminates the transaction.

(4) The DWBUA issues a DATO or a DATOB depending on the mask bits.

(5) The DWBUA does the following in response to a SSYN timeout during the UNIBUS portion of a WRITE transaction:

a. Sets the USSTO bit in the BIACSR.

b. Issues an error interrupt to the VAXBI if interrupts are enabled.

(6) After the DWBUA sets its interlock, it sends a RETRY response to all VAXBI commands except UWMCI.

(7) The DWBUA may receive a UWMCI command without having received a preceding IRCI command. When this happens, the DWBUA processes the UWMCI command as a WMCI command.

(8) If the VAXBI command, address or data has a parity error, the corresponding UNIBUS DATO(B) command is not issued, and the DWBUA adapter's interlock is not released, hanging the DWBUA.

(9) The DWBUA assumes that the UWMCI is targeted for the same address as the IRCI and ignores the incoming address.

4.3.2.2 VAXBI-to-I NIBUS Commands -

Table 4-6 VAXBI-to-UNIBUS Commands

VAXBI COMMAND		I NIBUS Command Translation	DWBI A Response to VAXBI	Possible Errors	Size Note
Code	Name				
0000	Reserved	None	NO ACK	None	1
0001	READ	DATI	ACK/RETRY	I SSTO	2,12
0010	IRCI	DATIP	ACK/RETRY	I SSTO	3,12
0011	RCI	DATI	ACK/RETRY	I SSTO	4,12
0100	WRITE	DATO	ACK/RETRY	I SSTO	5
				VAXBI PE	12,13
0101	WCI	DATO	ACK/RETRY	I SSTO	6
				VAXBI PE	12,13
0110	LWMI	DATOBI	ACK/RETRY	I SSTO	7,9
				VAXBI PE	12,13
0111	WMCI	DATOBI	ACK/RETRY	I SSTO	7
				VAXBI PE	12,13
1000	INTR	None	NO ACK	None	8
1001	IDENT	Illegal	ACK/RETRY	SACK	9,14
1010	Reserved	None	NO ACK	None	1
1011	Reserved	None	NO ACK	None	1
1100	STOP	SEE NOTES	ACK	None	10
1101	INVAL	None	NO ACK	None	11
1110	BURST	None	NO ACK	None	11
1111	IPINTR	None	NO ACK	None	11

NOTES FOR TABLE 4-6:

(1) These codes are reserved by Digital Equipment Corporation for future expansion. The DWBI A responds to these codes with NO ACK.

(2) All VAXBI READ and I NIBUS space are limited to longword length. The VAXBI address bit A<01> determines which word is read from the I NIBUS.

(3) IRCI and WMCI commands operate as I NIBUS DATIP DATOB sequences. The DWBI A is interlocked by the IRCI command; the I WMCI command responds to interlocked READ and WRITE commands directed to the DWBI A register. RETRY responses with the DWBI A not interlocked (but not I NIBUS constraints that require suppression of I WMCI commands) must be the same as for the IRCI command. Hence, the DWBI A uses the address of the IRCI command while servicing the I WMCI command, ignoring the address supplied with the latter command.

(4) A VAXBI RCI command is treated as a RDATI command.

(5) A VAXBI or I NIBUS WRITE is limited to longword length only. VAXBI address bit A<01> determines which word is written.

(6) A VAXBI WCI command is treated as a WRITE command.

(7) Data is written or read only if VAXBI address bit A<01> determines that word is longword aligned; otherwise the byte mask bits must be set to the selected word. The DWBI A will respond to the command with ACK. The error bit in the I NIBUS data word is set. A VAXBI timeout status is returned for the word not selected by VAXBI bit A<01> as generated by the DWBI A.

(8) Some interrupts are only permitted in the I NIBUS to VAXBI direction; the DWBI A responds to all INTR commands with NO ACK.

(9) The DWBI A responds to IDENT commands with a pending vector interrupt request at the highest priority level. If there is no linked vector, the DWBI A will fetch an interrupt vector from the I NIBUS device by issuing a IIV at the corresponding level of the IDENT command.

(10) The STOP command resets all pending interrupts and DMA requests from the I NIBUS. The DWBI A does not alter the contents of any register implemented in the user CSR space. The DWBI A responds to all subsequent VAXBI commands but does not attempt to gain access to/control the VAXBI. The effect of the STOP command is reset only by BUTR TO.

(11) INVAL, BURST, and IPINTR are ignored by the DWBI A. The DWBI A responds to these codes with NO ACK.

(12) I SSTO. The corresponding I NIBUS transaction has resulted in a SSYN timeout. The DWBI A did not receive SDIN within 10 µs after asserting MSYN. The I SSTO bit in the BI CSR is set, and an error interrupt is sent to the VAXBI if interrupts are enabled.

(13) VAXBI PE. Parity error on the VAXBI. The DWBI A ignores the transaction.

(14) SACK. SACK is not asserted by the interrupting I NIBUS device. The DWBI A sends zero data for the vector.

4.3.2.3 Example: VAXBI READ of UNIBUS Data - The VAXBI-to-UNIBUS transaction used as an example in this section is a VAXBI READ of UNIBUS data. In this transaction, the VAXBI master reads data from a device on the UNIBUS.

Figure 4-3 is a flow diagram of the VAXBI READ of a UNIBUS data transaction. The numbered paragraphs that follow refer to the corresponding numbers in Figure 4-3.

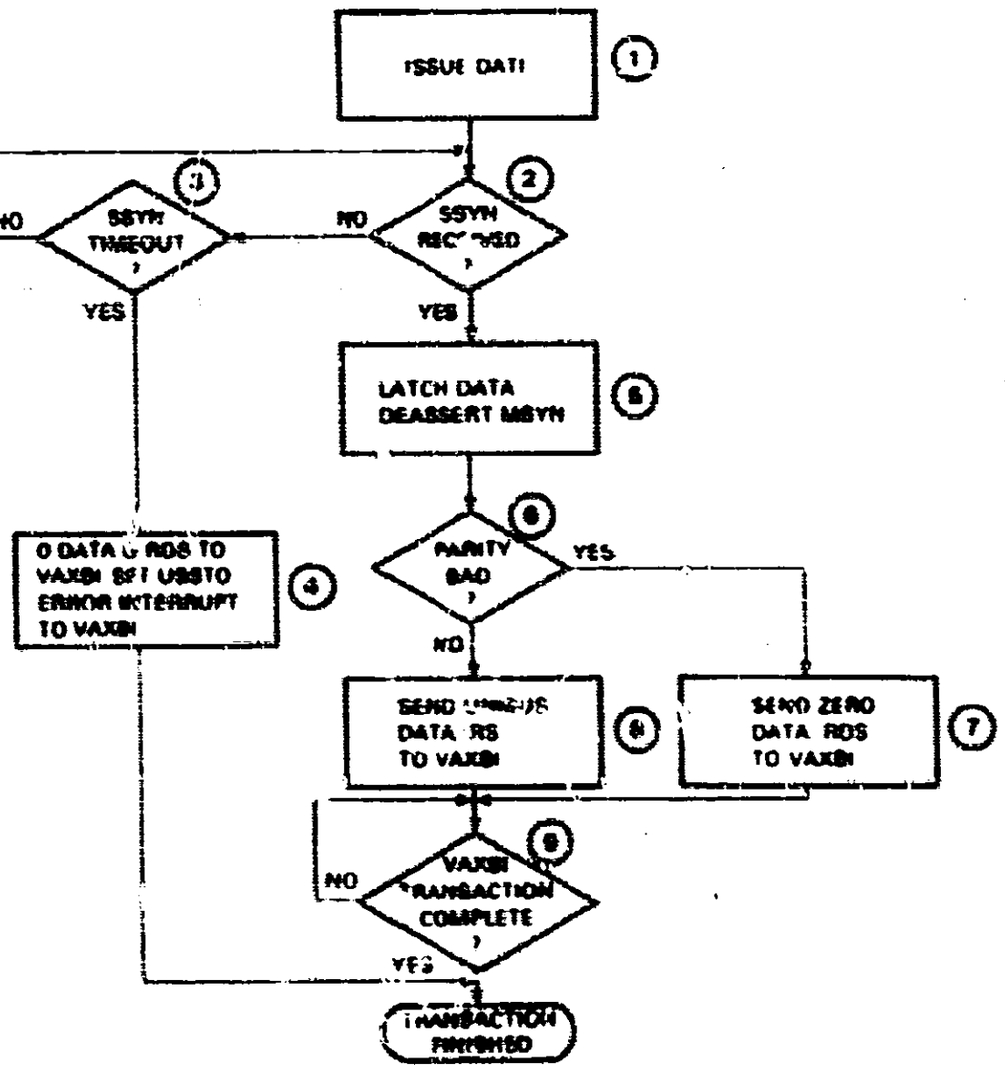


Figure 4-3 VAXBI READ of UNIBUS Data Flow Diagram

(1) The VAXBI command and address are received by the DWBUA. The slave port control determines that the transaction is for the DWBUA. The DWBUA response to the VAXBI is STALL.

The VAXBI address is latched in the VAXBI address latch.

The DWBUA monitors the BBSY signal on the UNIBUS. If it is asserted, the DWBUA waits until it is deasserted. Since the DWBUA is the UNIBUS arbitrator, it has the highest priority on the UNIBUS. When BBSY is deasserted by the present UNIBUS master, the DWBUA asserts BBSY and gains bus mastership.

The DWBUA issues a DATI command. The address in the VAXBI address latch is sent over the BAD bus to the UNIBUS address transceivers. Address and control bits are sent out on the UNIBUS.

The DWBUA asserts MSYN. The data drivers [illegible] the UNIBUS D lines.

(2) The DWBUA monitors SSYN and waits for it to be asserted.

(3) If SSYN is not asserted within [illegible] µs from assertion of MSYN, a SSYN timeout occurs.

(4) SSYN timeout causes the DWBUA to send zero data and RIDS to the VAXBI, set the USSTO bit of the BICSR, and issue an error interrupt to the VAXBI if interrupts are enabled. The transaction is terminated.

(5) The UNIBUS slave sends the data and SSYN to the DWBUA. Data is received at the UNIBUS data transceivers.

(6) Parity for the data is checked.

(7) If the data has a parity error, zero data and a read data substitute (RDS) status code are sent to the VAXBI. The RDS status code warns the VAXBI that the data contains an uncorrectable error. The transaction is terminated.

(8) If parity is good, the UNIBUS data is sent over the BDP bus to the VAXBI data transceivers. From there it is sent over the BI bus to the BIIC and out to the VAXBI. A read data status code is sent to the VAXBI, indicating that the data is error free.

(9) When all of the data has been sent to the VAXBI, the DWBUA sends three ACKs to the VAXBI, indicating that the transaction is complete.

4.3.1 I²NIM S-to-VAXBI Transactions

4.3.1.1 DWBI A Responses to I²NIM S-to-VAXBI Transactions

In an I²NIM S-to-VAXBI transaction, the I²NIM S master sends a command to the DWBI A that requires the DWBI A to read from or write to a VAXBI node.

Table 4-7 Bus Masters and Slaves for I²NIM S-to-VAXBI Transactions

Bus	Master	Slave
I²NIM S	Driver initiating transaction	DWBI A
VAXBI	DWBI A	VAXBI node

The DWBI A responds to these I²NIM S commands that require DWBI A interaction with other VAXBI nodes: DATI, DATIBI, and DATIP. These responses are listed in Table 4-8. The DWBI A responds to I²NIM S commands are independent of the data path used.

Table 4-8 DWBI A Responses to I²NIM S-to-VAXBI Transactions

I²NIM S-to-VAXBI Transaction	DWBI A Response
DATI (1)	Data (2) SSYNC (3)
DATIBI (4)	SSYNC (5) Data to VAXBI (6)
DATIP DATIBI (7)	
DATIP (8)	Data (9) SSYNC (10)
DATIBI (9)	SSYNC (11) Data to VAXBI (12)

NOTE

If the DWBI A is processing a VAXBI transaction when the I²NIM S request is received, the DWBI A withholds the bus grant until the VAXBI transaction has completed.

NOTES FOR TABLE 4-8

(1) A DATI command from an I²NIM S device to read data from the DWBI A.

(2) The VAXBI slave responds when the DWBI A is returning the data from the VAXBI SSYNC response. The withheld IRQ is withheld. SSYNC terminal results. The DWBI A sets the BBR bit in the BB WSR and the BBR causes an error interrupt on the VAXBI if interrupts are enabled.

(3) The DWBI A issues SSYNC in response to a DATI command only when the VAXBI slave responds to the VAXBI portion of the transaction with WAIT. Any other response from the VAXBI slave results in the DWBI A withholding SSYNC.

(4) When the DWBI A receives a DATIBI command, it requests data from the I²NIM S device.

(5) The DWBI A issues SSYNC before it completes the corresponding VAXBI transaction.

(6) If an error occurs while the DWBI A is writing the data to the VAXBI node, the DWBI A sets the BBR bit in the BB WSR, and the BBR causes an error interrupt on the VAXBI if interrupts are enabled. SSYNC must be withheld from the I²NIM S device until the SSYNC results of an SSYNC command.

(7) The DATIP DATIBI command sequence must be performed over the path through the DWBI A called the Direct Data Path. An attempt to perform the command sequence over the Buffered Data Path results in an SSYNC timeout. The BYTE QUINT bit in the I²NIM S Maintenance Register corresponding to the Direct Data Path is cleared and treated as if it is zero. If the bit is set, the command is treated as if the bit is clear.

(8) VAXBI timeout occurs if DATIP through the Direct Data Path is not followed by the VAXBI

(9) I²NIM S protocol requires that a DATIP be followed immediately by a DATIBI command. The address must not be deasserted between the two commands. The sequence from the time the DWBI A receives the LIR bit of the BB WSR for a DATIBI command to the VAXBI transaction occurs during the I²NIM S that is generated when the BB bit in the BB WSR is set. Under these circumstances an error interrupt on the VAXBI is unchanged.

4.3.3.2 [illegible] Commands Through the Direct Data Path

NOTE:
A complete description of data path operation can be found in Appendix I.

Table 4-9 [illegible] Commands Through the Direct Data Path

[illegible] Command	[illegible] Address <31:0>	Command to VAXBI	Transfer Length	Possible Errors	See Note
BYTE OFFSET BIT = 0					
DATI	ANY	READ	LONGWORD	A B	4
DATIP	ANY	RDL	LONGWORD	A B C	1,4
DATO	ANY	WMSK or LWMSK	LONGWORD	A B	3,4
DATOB	ANY	WMSK or LWMSK	LONGWORD	A B	4
BYTE OFFSET BIT = 1					
DATI	ANY	READ	LONGWORD	A B	1,4
DATIP	ANY	N/A	N/A	N/A	2
DATO	ANY	WMSK	LONGWORD	A B	3,4
DATOB	ANY	WMSK	LONGWORD	A B	4

NOTES FOR TABLE 4-9:

(1) A DATIP command is valid only through the Direct Data Path. A DATIP is attempted through a Buffered Data Path or through the Direct Data Path [illegible] BYTE OFFSET bit set, the [illegible] command is ignored and the DWBUA does not issue [illegible] commands. During this time, all VAXBI transactions to the DWBUA receive RETRY responses until the [illegible] device register RDL).

If a DATIP command is not followed by a DATIBE, the DWBUA [illegible] and issues an error interrupt if interrupts are enabled.

(2) All [illegible] DATIBE through the Direct Data Path [illegible] with the word byte set for each valid data byte.

(3) The DWBUA performs read-longword operations on the VAXBI for [illegible] the Direct Data Path. Both the BYTE OFFSET bit and [illegible] address bit A<1> are set.

(4) Possible Errors:

(A) PE — The VAXBI transaction has returned an error code that the DWBUA recognizes as an error code. RTO, RBER, RSE, NORSP, HARD, RPE, or SITE. The PE bit is set in the BI_ISR and an error interrupt is sent to the VAXBI if interrupts are enabled. The DWBUA then terminates the VAXBI transaction, returning an RETRY response to the [illegible] device that initiated the transfer.

(B) IVAR — The VAXBI bit is not set in the [illegible] Map Register for the incoming [illegible] address. The IVAR bit is set in the BI_ISR and an error interrupt is sent to the VAXBI if interrupts are enabled.

(C) [illegible] — The [illegible] memory data received from the DATIP sequence is not the same as the [illegible] DATIP(B). The [illegible] bit is set in the BI_ISR and an error interrupt is sent to the VAXBI if interrupts are enabled.

4.3.3.3 A sample DATOB(B) Using the Direct Data Path - In this transaction the I/NIBUS master writes data to a VAXBI node. The data is not temporarily stored in a BDP buffer, as it is during a Buffered Data Path transaction, instead, it goes directly to the VAXBI.

Figure 4-4 is a flow diagram of the DATOB(B) using the Direct Data Path transaction. The numbered paragraphs that follow refer to the corresponding numbers in Figure 4-4.

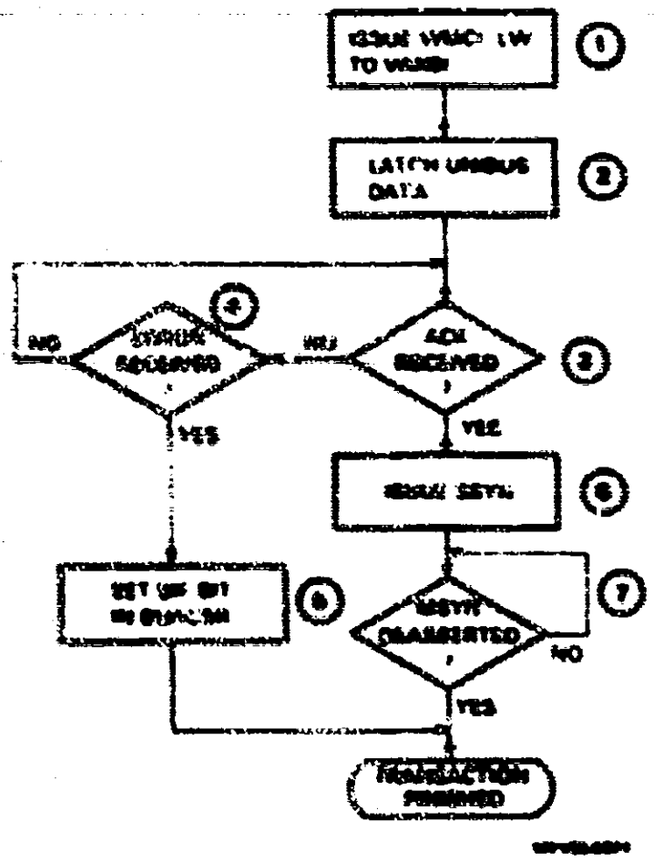


Figure 4-4 DATOB(B) Using the Direct Data Path Flow Diagram

(1) The I/NIBUS master sends the address, control lines, and data to the DWBI A and it then asserts MSYN. The DWBI A decodes the control lines and determines that the command is a DATOB(B).

The DWBI A requests the VAXBI and starts a WRITE LW (write mask with cache invalidate longword) transaction. This is the VAXBI transaction that corresponds to a I/NIBUS DATO. The mask bits are determined by the command (DATO or DATOB). The I/NIBUS address is longword aligned.

(2) The I/NIBUS data is latched in the I/NIBUS data transceivers.

The data goes from the I/NIBUS data transceivers to the BDP bus through the VAXBI data transceivers and to the VAXBI.

(3) The VAXBI slave sends ACK to the DWBI A indicating that it has received the data. After ACK is received, the BIIC sends the DWBI A a master transaction complete signal.

(4) If ACK is not received, the DWBI A looks for an error.

(5) If an error is received, the BIIC bit in the BI_WSR is set and the transaction is terminated. SSYN may not be issued, resulting in an MSYN timeout.

(6) The DWBI A issues SSYN completing the I/NIBUS transaction.

(7) The DWBI A deasserts MSYN. When it is deasserted, the VAXBI transaction is complete.

4.3.3.4 I/OBUS-to-VAXBI Commands Through a Buffered Data Path –

Table 4-20 I/OBUS-to-VAXBI Commands Through a Buffered Data Path

I/OBUS Command	I/OBUS Address <3:0>	BDPBR Status	Command to VAXBI	Transfer Length	Possible Errors	Size Value
BYTE OFFSET BIT = 0						
DATI	ANY	EMPTY	RD-RD	QUADWORD	R (1)	2,3
DATI	ANY	FULL	WRITE	N/A		4
DATI	ANY	FULL	READ	QUADWORD	R (1)	4,6
DATI	ANY	QUAD	WRITE or READ	QUADWORD	R (1)	4,6
DATIP	ANY	N/A	N/A	N/A		5
DATO	ANY	FULL or EMPTY	WRITE	QUADWORD	R (1)	4,6
DATO	ANY	QUAD/M	WRITE	QUADWORD	R (1)	4,6
DATO	ANY	QUAD/D	WRITE	QUADWORD	R (1)	4,6,8
DATOB	ANY	FULL or EMPTY	WRITE	QUADWORD	R (1)	4,6,7
DATOB	ANY	QUAD/M	WRITE	QUADWORD	R (1)	4,6,7
DATOB	ANY	QUAD/D	WRITE	QUADWORD	R (1)	4,6,8
BYTE OFFSET BIT = 1						
DATI	Byte	EMPTY	READ	QUADWORD	R (1), (2)	4
DATI	Byte	FULL	WRITE	N/A		4
DATI	Byte	FULL	READ	QUADWORD	R (1), (3)	5,6
DATI	Byte	QUAD	READ	LONGWORD	R (4)	5,6
DATI	1	EMPTY	READ	QUADWORD	R (1), (2)	5
DATI	1	FULL	READ	QUADWORD	R (1), (2)	5,6
DATI	1	FULL	READ	QUADWORD	R (1), (3)	6,8
DATI	1	QUAD	READ	LONGWORD	R (4)	5,6,7,8
DATIP	ANY	N/A	WRITE	N/A		5
DATO	Byte	FULL	WRITE	N/A		6
DATO	Byte	EMPTY	WRITE	N/A		6
DATO	Byte	QUAD/M	WRITE	LONGWORD	R (1), (5)	5,6
DATO	Byte	QUAD/D	WRITE	LONGWORD	R (1), (5)	5,6
DATO	1	FULL or EMPTY	WRITE	QUADWORD	R (1), (5)	5,6
DATO	1	QUAD/M	WRITE	QUADWORD	R (1), (5)	5,6,7
DATO	1	QUAD/D	WRITE	QUADWORD	R (1), (5)	5,6,8
DATOB	ANY	FULL or EMPTY	WRITE	QUADWORD	R (1), (5)	5,6,7
DATOB	ANY	QUAD/M	WRITE	QUADWORD	R (1), (5)	5,6,7
DATOB	ANY	QUAD/D	WRITE	QUADWORD	R (1), (5)	5,6,8

NOTES FOR TABLE 4-20

(1) The second case is caused automatically when a Buffered Data Path transfer is aborted after an empty BDP is the write. The low byte of the requested DATI word is supplied by performing a requested DATI through the Direct Data Path. The high byte is supplied from either the current BDP buffer or the VAXBI with a subsequent DATI through the BDP. The second BDP word remains unchanged during this transaction.

(2) A DATIP transaction is valid only through the DDP.

(3) A I/OBUS DATI command through a Buffered Data Path results in an automatic READ of VAXBI space if the requested data is not in the BDP buffer. If the I/OBUS data is stored in the BDP buffer, however, the buffer must be purged by performing an automatic WRITE to the VAXBI before reading the data from the VAXBI. The entire automatic is handled with the buffer; subsequent accesses within the automatic through the same Buffered Data Path cause the BDP to fetch the data from the buffer with no VAXBI transactions required.

(4) Data for a DATI(B) command through a BDP is stored onto the buffer as well. The BDP then performs a VAXBI automatic WRITE transaction of the buffer contents to ensure automatic of valid data from the I/OBUS device. A VAXBI automatic WRITE is performed if the buffer contains less than a complete automatic of valid data.

(5) Possible Errors:

(5.1) RBT – The VAXBI transaction has returned an error code that the BDP/BI recognizes as an error code. RBTO (TIMEOUT, NXM, NORSP, RRESP) or WRITE. The BDP does not write into the BDP buffer and an error interrupt is sent to the VAXBI if interrupts are enabled. The BDP/BI error code is set to RBT, resulting in an SSYNC issued on the I/OBUS device that initiated the transfer.

(5.2) BMR – The VAXBI has written into one of the I/OBUS's Map Registers that corresponds to the addressed I/OBUS address. The BMR bit is set in the BDP/BI and an error interrupt is sent to the VAXBI if interrupts are enabled.

(5.3) BAUWERR – The I/OBUS's Map Register that corresponds to the addressed I/OBUS address has a parity error in the BDP/BI field (it is accompanied by either Buffered Data Path 0 or 1). The BAUWERR bit is set in the BDP/BI and an error interrupt is sent to the VAXBI if interrupts are enabled.

(6) Buffer states:

(6.1) FULL – The BDP buffer contains the I/OBUS's DATI data received from the VAXBI and the addresses match.

(6.2) EMPTY – The BDP buffer contains the I/OBUS's DATI data received from the VAXBI and the addresses do not match.

(6.3) QUAD – The BDP buffer contains the I/OBUS's DATO data to be sent to the VAXBI.

(6.4) EMPTY – The BDP buffer is empty.

(7) The command to the VAXBI is sent after SSYNC is issued.

(8) The command to the VAXBI may be sent after SSYNC is issued.

[illegible] Example: 8237 Using a Buffered Data Path - In this example the [illegible] master sends data to a [illegible] node. Each [illegible] node has two types of data sets a DMA buffer. The 8237 system can build [illegible] of these [illegible] are assigned at initialization the 8237 buffer. The buffer [illegible] the [illegible] node.

Figure 4-7 is a flow diagram of the 8237 using a Buffered Data Path transaction. The numbered paragraphs that follow refer to the corresponding numbers in Figure 4-7.

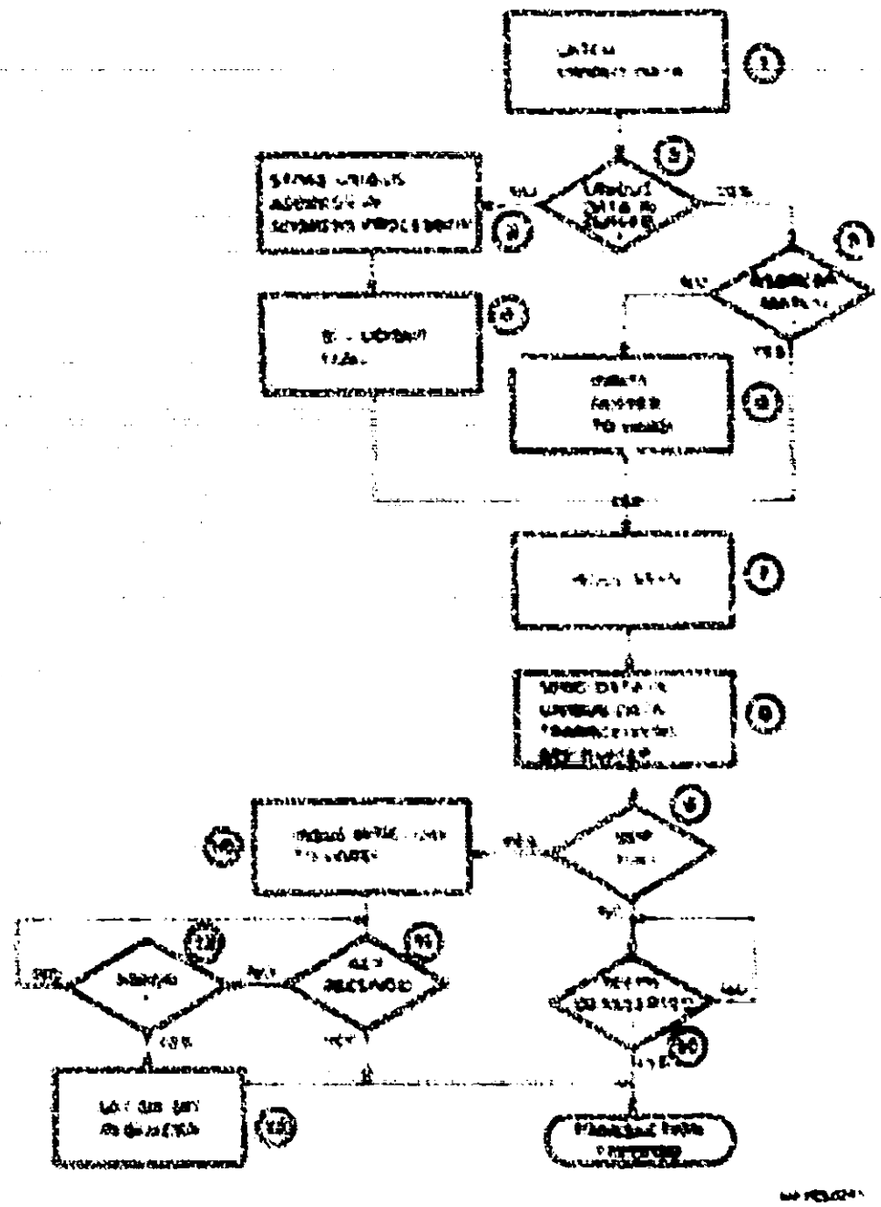


Figure 4-7. 8237 Using a Buffered Data Path Flow Diagram

(1) The [illegible] master sends address, control bits, and data to the 8237. [illegible] and then requests [illegible]

The next three of [illegible] data are latched in the [illegible] data transceivers.

(2) The [illegible] bit in the Data Path control and status Register is checked.

(3) If the [illegible] bit is clear, the [illegible] address goes over the DAD bus to the address translation where it is stored.

(4) The 8237 [illegible] the [illegible] that [illegible] is stored in the DMA buffer.

(5) If the [illegible] bit is set, the 8237 reads destination of the present data or part of the word contained in the data already in the DMA buffer. The interpretation of the present data is part of the word contained but [illegible] and the address stored in the address translation are combined with the destination [illegible] address.

(6) If the complete word is not yet in the DMA buffer, [illegible] is repeated. [illegible] the data in the DMA buffer is written to the [illegible] destination. If [illegible] is required by this step, the 8237 [illegible] data and increments the [illegible]

(7) The 8237 [illegible] the [illegible] that [illegible] is ready.

(8) The data in the DMA buffer is written over the [illegible] bus to the [illegible] buffer in the selected [illegible] the translation address stored in [illegible]

(9) The 8237 [illegible] the DMA buffer [illegible]

(10) If the DMA buffer is not full, the 8237 [illegible] the [illegible] master that it is ready for the next transaction.

(11) The 8237 [illegible] the [illegible] master that [illegible] is ready for the next transaction.

(12) If the DMA buffer is full, the 8237 [illegible] the [illegible] master that it has the contents of the DMA buffer. [illegible] the destination is [illegible] the [illegible] bit is set, the 8237 [illegible] the data in the DMA buffer is [illegible]

a. [illegible]
b. [illegible]

The second time that [illegible] is executed, [illegible] the [illegible] master to write the contents of the DMA buffer. [illegible] the destination is [illegible] the 8237 [illegible] the word. For the second DMA buffer

(13) The 8237 [illegible] the [illegible] that [illegible] is ready for the next transaction.

(14) If [illegible] has occurred, the 8237 [illegible]

(15) If an error has occurred on the [illegible] the 8237 [illegible] the DMA bus to the [illegible] and the [illegible] is terminated.

(16) The 8237 [illegible] that [illegible] is disconnected by the [illegible] master and there is [illegible] transaction.

4.3.1.b Example DATI Using a Buffered Data Path - In this example, the I/OBM is master reads data from a VAXBI node. The VAXBI node writes blocks of data to a BDP buffer. The buffer is used by the I/OBM's disk controller to store, or write, data to disk. DATI instructions are executed to read the entire buffer.

Figure 4-6 is a flow diagram of the DATI using a Buffered Data Path transaction. The numbered paragraphs that follow refer to the corresponding numbers in Figure 4-6.

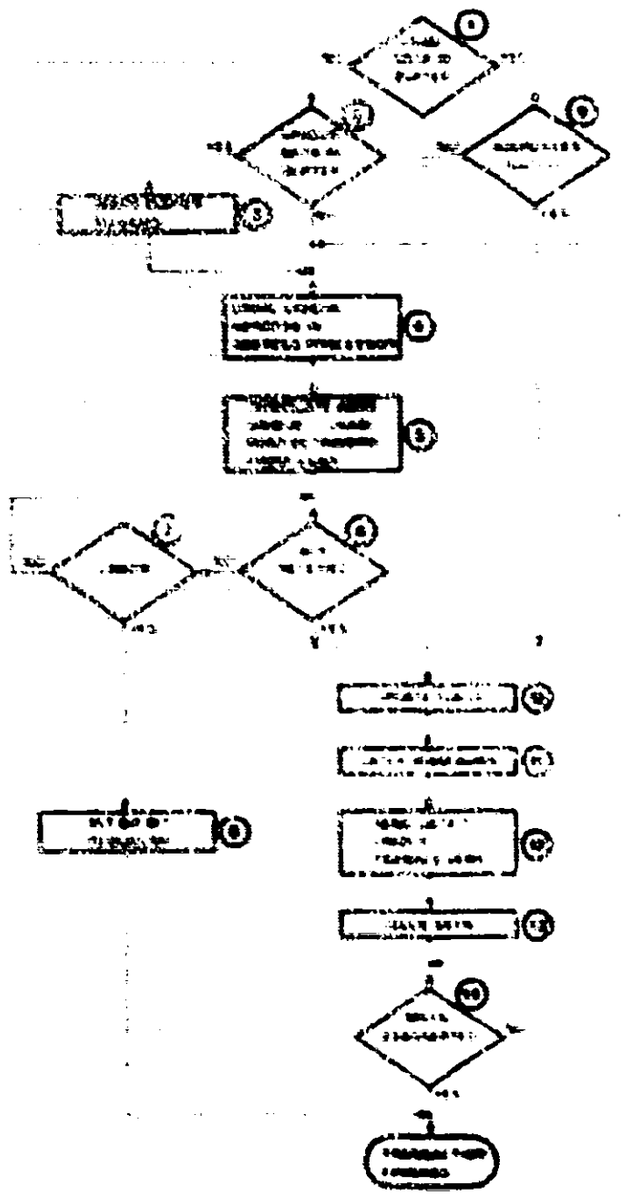


Figure 4-6 DATI Using a Buffered Data Path Flow Diagram

(1) The DWBI A checks the BDBR 1 bit in the Data Path Control and Status Register.

(2) If the BDBR 1 bit is clear, the BDP buffer does not contain VAXBI data. The DATI transaction

(3) If the BDBR 1 bit is set, the BDP buffer contains I/OBM's data. The buffer contents are unsupported.

(4) The I/OBM's address is stored in the address processor.

(5) The I/OBM's address is translated into a VAXBI address. VAXBI BI NID is inserted and an unlocked read data is initiated to the VAXBI slave. The data goes into the BDP buffer.

(6) The DWBI A waits for the VAXBI slave mode transaction to end.

(7) If NXM is not received, the DWBI A waits for an error code.

(8) If an error is received on the VAXBI, the BDI bit in the IB_MASK is set and the transaction is terminated.

(9) If the BDBR 0 bit in the DPR_BI is set, the address stored in the address processor is compared with the I/OBM's address latched in the I/OBM's address comparator.

(10) BDBSM 1 and 0 bits are updated.

(11) The I/OBM's address from the address processor is latched.

(12) The requested data word goes from the BDP buffer and is latched in the I/OBM's Data Multiplexer.

(13) The DWBI A waits for MSYN.

(14) The DWBI A waits for MSYN to be deasserted by the I/OBM's master. When MSYN is deasserted, the transaction is finished.

4.4 [illegible]

The following diagrams in this section represent typical DWBI A transactions. The following assumptions apply to all:

- No errors occur during the transaction.
- The transaction follows a straight line path through the flows.
- No time waste is experienced. The diagrams emphasize relative timing only.

In the following diagrams the clock name that appears in parentheses under the assertion is the clock that asserts that signal.

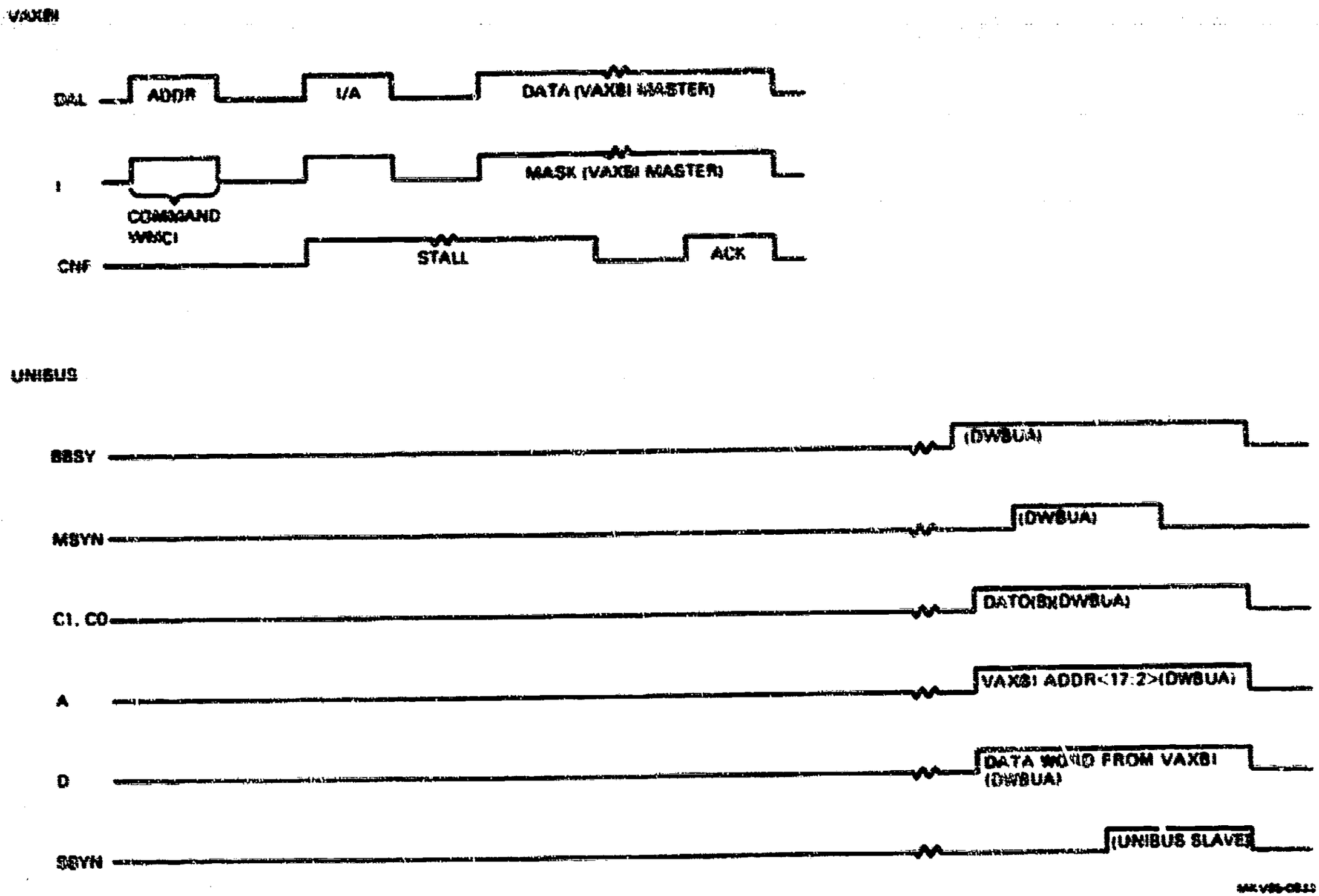


Figure 4-7 VAXBI-to-UNIBUS WRITE Timing Diagram

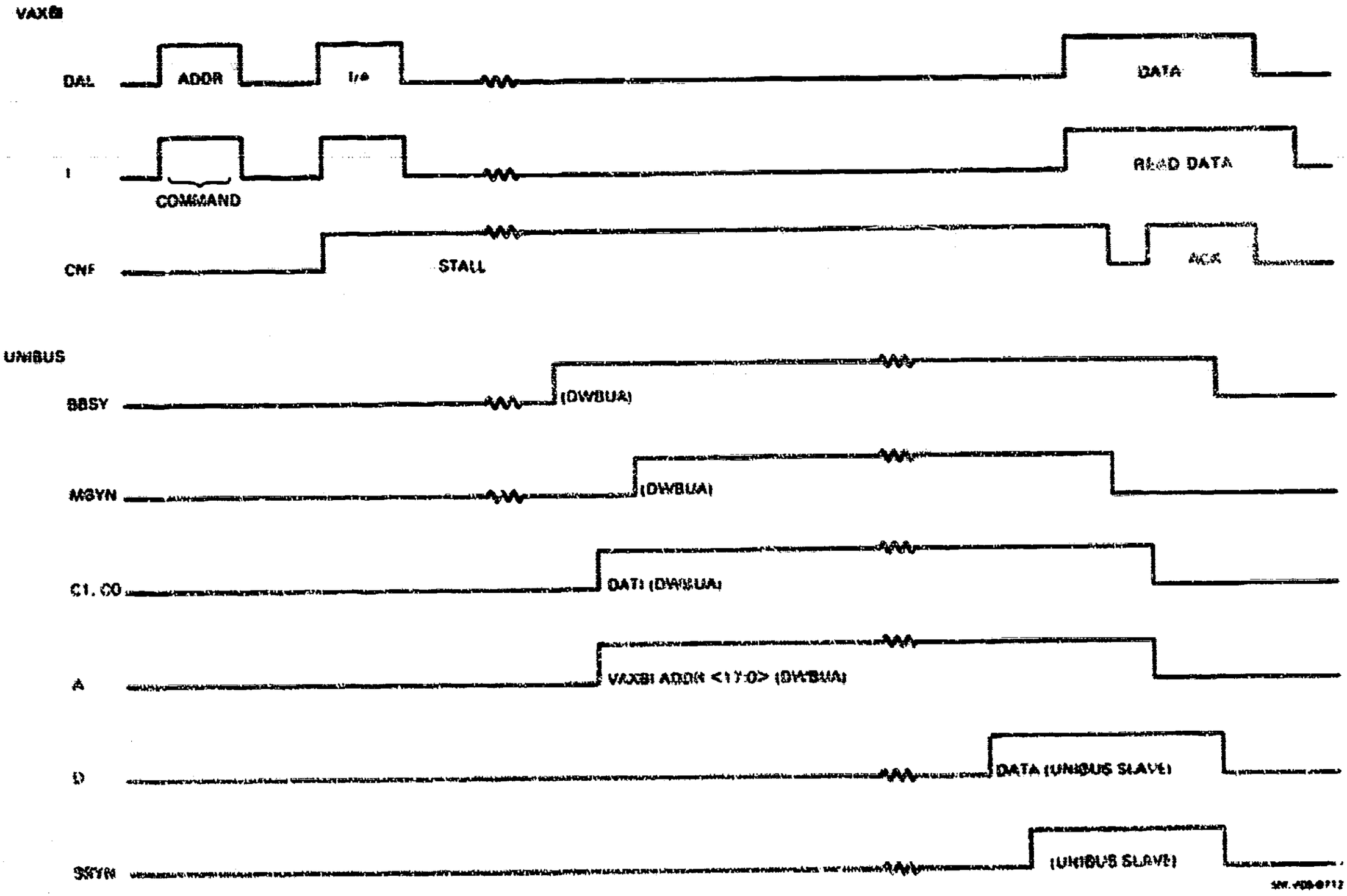


Figure 4-8 VAXBI-to-UNIBUS READ Timing Diagram

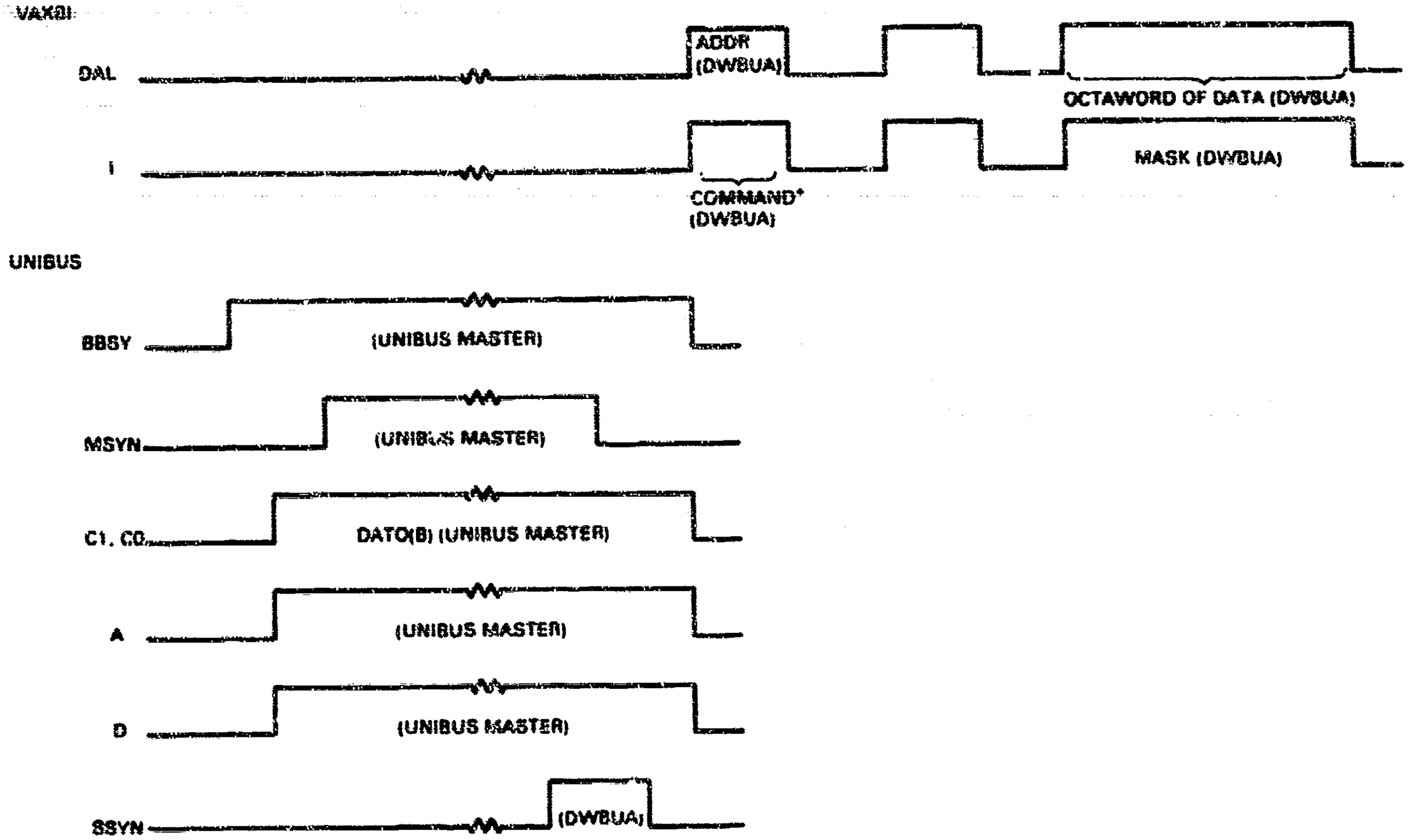


Figure 4-9 DATO(B) Through a Buffered Data Path Timing Diagram

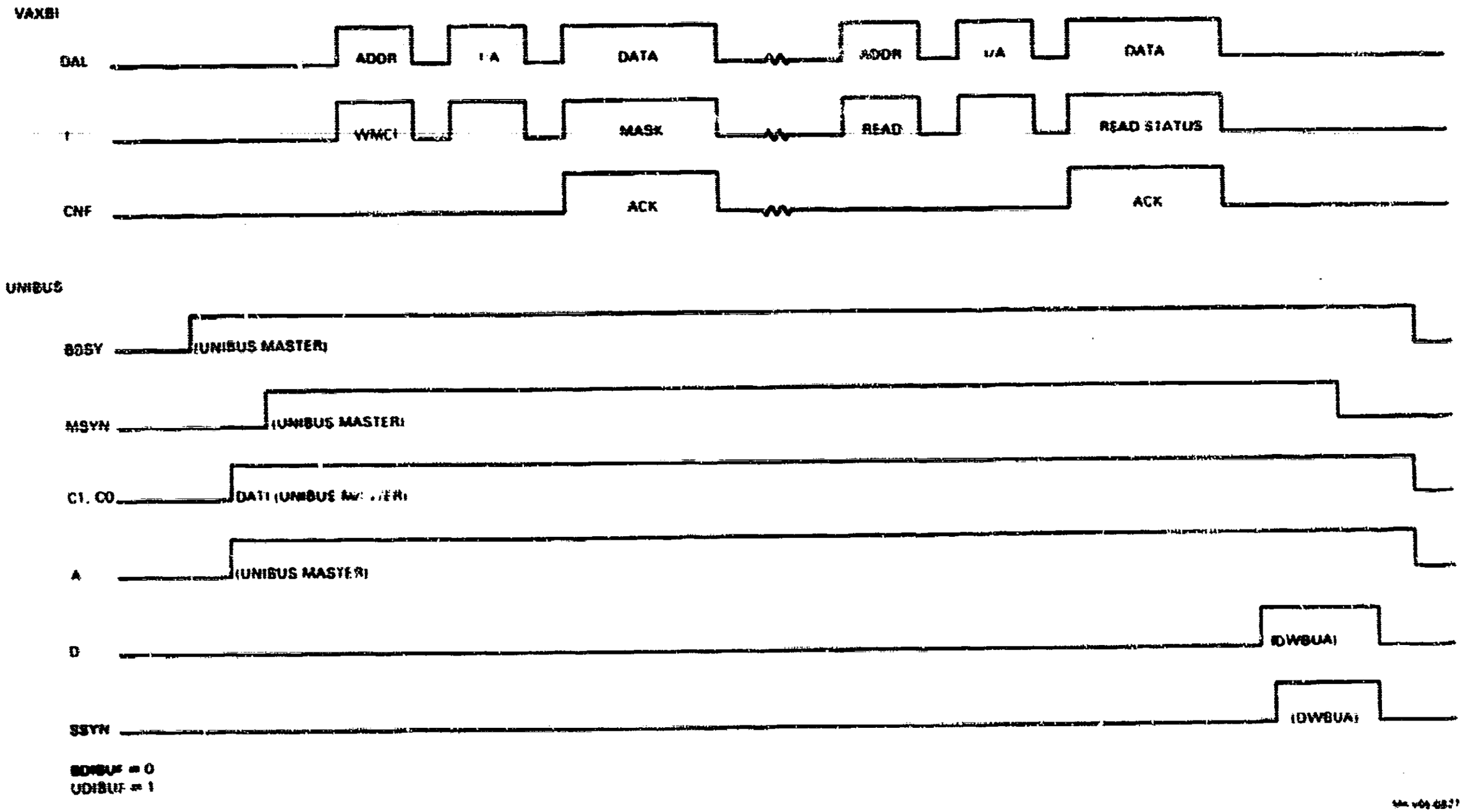


Figure 4-10 DATI Through BDP with Autopurge Timing Diagram

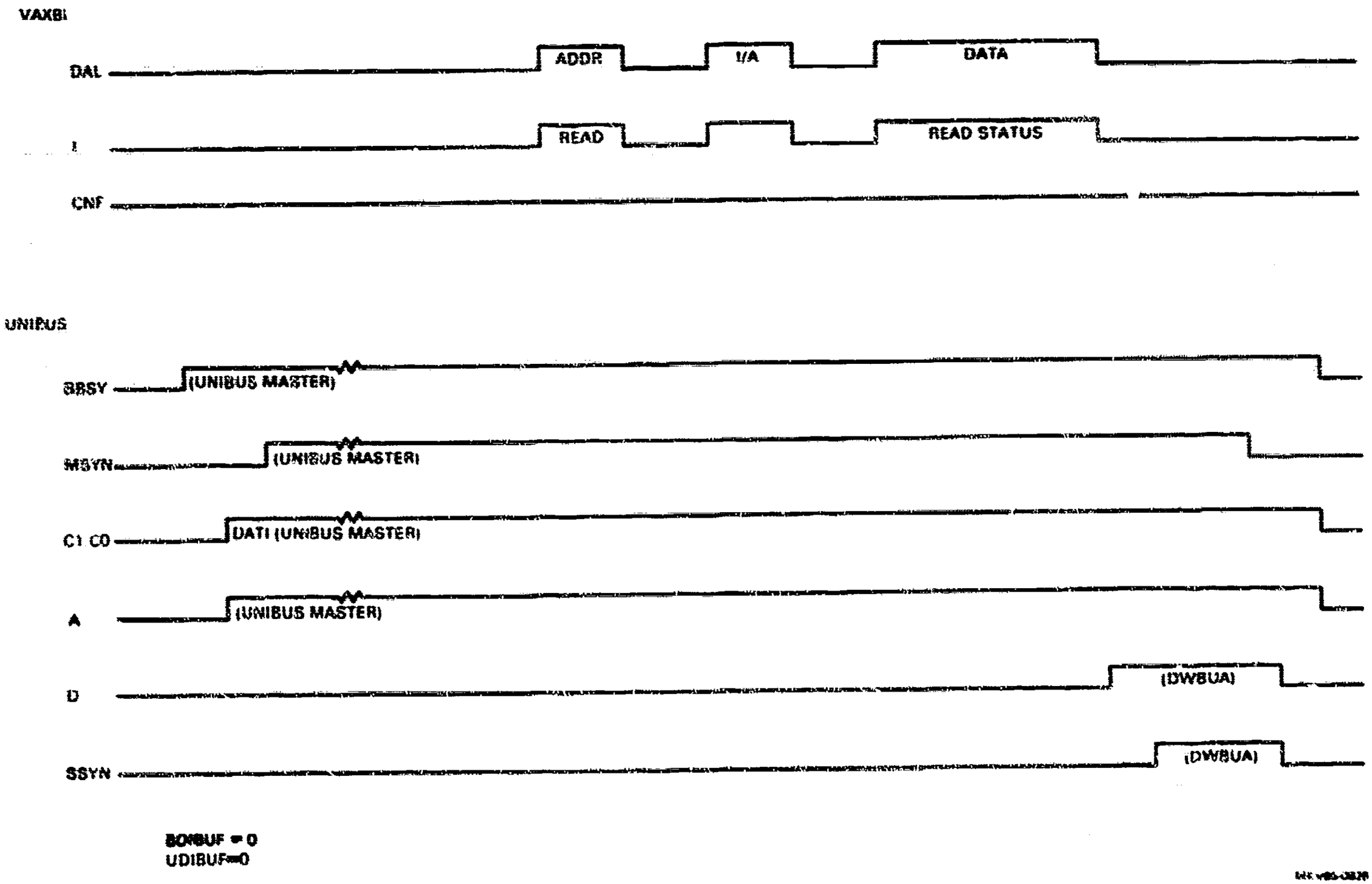


Figure 4-11 DATI Through BDP Timing Diagram

APPENDIX A

APPENDIX A
DWBUA-SUPPORTED UNIBUS DEVICES

A DWBUA UNIBUS configuration supports a subset of the available UNIBUS devices. The following devices cannot be put on a DWBUA-controlled UNIBUS:

- Any PDP-11 processor
- Any device that attempts to perform UNIBUS arbitration
- Any device that has an SSYN timeout period of less than 20 µs

Any device that issues MSYN after using a BR to arbitrate for the UNIBUS may not work satisfactorily.

Contact the local DIGITAL software office for a list of currently supported devices.

APPENDIX B

APPENDIX B
GLOSSARY

[illegible] Acknowledge. A VAXBI command response. ACK indicates that the VAXBI slave acknowledges that it is capable of executing the command at the time. A VAXBI data response. ACK indicates that no error has been detected and that the cycle is not to be retried.

ADP Adapter. The portion containing the components of a particular VAXBI. A BDP buffer in the VAXBI adapter is unsupported when a particular VAXBI's data and one of the following occurs: a DATI is requested through the same BDP or a DATIB is requested and its address is not within the same quadword as the data currently stored in the buffer. Data is unstaged from the buffer unless a VAXBI write, WAIT command with the preselected mask has set for each valid data byte.

BAD BDP's Backward Address.

BASE ADDRESS. The starting address of a VAXBI node's node space.

BB Base address.

BBSY Bus busy. A VAXBI signal. This signal is used on the bus memory to all other bus devices to indicate that the bus is in use.

BCI VAXBI Cap Interface. The [illegible] interface bus that provides for all communications between the BDP and the DMA's.

BDACK Broadcast a VAXBI command. This command announces a significant event without incurring the overhead of an interrupt. The use of this command is reserved to Digital Equipment Corporation.

BDP DMA's Backward Data Path.

BIIC The Interconnect Interface Chip. This chip is a general purpose interface to the VAXBI.

BI of VAX BI's Control and Status Registers. A DMA's internal register.

DATI Data In. A VAXBI command. This command requests a transfer of data from the VAXBI slave to the VAXBI's master. The transfer is always word length.

DATIP Data In Plenty. A VAXBI command. DATIP is identical to DATI except DATIP informs the VAXBI slave that the present transfer is the first part of a read modify write cycle. DATIP must be followed by DATOB to the same word address.

DATOB Data Out (Byte). A VAXBI command. This command transfers a word (DATO) or byte (DATOB) of data from the VAXBI's master to the VAXBI's slave.

DFP DMA's Forward Data Path.

DWBA - DWBI Window Address

DPCSR - Data Path Control and Status Register. a DWBI A internal register

DWBI A - VAXBI to DWBI A Adapter

FI WAD - Failed DWBI A Address Register. a DWBI A internal register

IDENT - Identify a VAXBI command. This command is used by processors and other intelligent modules (including DWBI A) to obtain vector information

INTERLOCK - A mechanism on the DWBI A that locks out transactions while waiting for a specific command. It is used when the DWBI A receives an IRQ 1 from the VAXBI or a DATIP from the DWBI A. It locks out all other transactions except STOP from the VAXBI until it receives the DWBI A or the DATXB that completes the current transaction

INTR - Interrupt a VAXBI command. This command sends interrupts to other nodes on the VAXBI

INVDAT - Invalidate. a VAXBI cmd word. This command forces a processor or another intelligent node signals to other nodes that they may have in their caches data that is no longer valid

IPINTR - Interprocessor interrupt. a VAXBI command. This command is used by a processor to interrupt another processor or an intelligent adapter

IRAM - DWBI A Internal RAM

IRD 1 - Interlock READ with Cache Invalidate. a VAXBI command. The data READ from the slave is placed in the master's cache. This is the first part of a read-modify-write cycle. IRD 1 must be followed by LWMSK

LMAP n - Longword Access Enable. a DWBI A Map Register for

MBZ - Must Be Zero

MSYN - Master Sync. a DWBI A signal issued by the bus master and received by the bus slave. A system of MSYN requests the slave, defined by the DWBI A address lines, to perform the function requested by the DWBI A control lines. Negation of MSYN indicates to the slave that the master completes the data transfer considered

NO ACK - No Acknowledge. A a VAXBI command response. NO ACK indicates that no slave has been selected or that an error occurred during transmission of the command/address cycle. As a VAXBI data response, NO ACK indicates that an error has been detected on the transaction

NOOP - See VAXBI Node

NPBW ID - A hexadecimal number between 0 and F (or a decimal number between 0 and 15) that indicates which of the sixteen logical locations a particular VAXBI node occupies

NODE SPACE - An 8k byte block of I/O addresses. Each node, based on its node ID, is allocated a unique node space. The DWBI A adapter's node space holds the DWBI A registers

OCTAWORD - Sixteen contiguous bytes starting on an arbitrary byte boundary

PORT LOCK - This mechanism locks the VAXBI and DWBI A ports while the DWBI A is servicing a transaction

PROBE - The act of copying a GDP buffer to setting the corresponding DPCSR PI REQ bit

RD - READ with Cache Invalidate. a VAXBI command. The data read from the slave is placed in the master's cache

READ - A VAXBI command. The master node reads data from the slave

RETRY - A VAXBI command response. This response indicates that the slave cannot immediately execute the command and to

SACK - Selection Acknowledge. a DWBI A signal. A device that has requested the bus acknowledges that it has been granted the bus and that it accepts

SSYN - Slave Sync. a DWBI A signal issued by the bus slave and received by the bus master. Assertion of SSYN informs the bus master that the slave has completed its part of the current data transfer. Negation of SSYN informs all bus devices that the slave has completed the current data transfer

STALL - As a VAXBI command response, STALL indicates that the slave needs additional time to acknowledge the command or not ready to receive the first data word on a READ command or to put data on an IRD 1 command, or is not ready to accept a data word on a WRITE command. As a VAXBI data response, STALL is sent by the slave to delay the transmission of data

STOP - A VAXBI command. This command also causes nodes to enter a state in which they do not issue VAXBI commands, nor take actions or initiate error interrupts as previously

TA - DWBI A address

TIMEOUT - An error detection that computes the time between the use of the data resource at the DWBI A. The interpulse separation which device will need be granted control of the DWBI A. A DWBI A must have one and only one arbitrator. For the configuration described in this manual, the DWBI A is always the DWBI A arbitrator

TRANSACT - Logical VAXBI block read/write transaction between a VAXBI command. This command comprises a read modify write cycle that begins with an IRD 1 command

VAXBI - VAX Bus Interconnect. It joins a processor to a combination of devices that can include I/O controllers, MA 780 memories, and other processors. The is a multiplexed, synchronous interconnect with bus control occurring at fixed intervals. Bus addresses and address and data transmissions are done multiplexed over 32 data lines. Data transmission is at fixed lengths of 4 bytes or 16 bytes. Provides for several additional functions

VAXBI NODE - An interface that encapsulates one of various logical locations on a VAXBI. A VAX 8600 node consists of one or more VAXBI modules

VBR - Vector Base Register. a DWBI A internal register

WRITE - WRITE with Cache Invalidate. a VAXBI command. The master node writes data to the slave and alerts other nodes to mark a VAXBI INVALID if necessary for the address written

[illegible] — A VAXBI write access to this address. Each node, based on its node ID, is allocated a unique memory space. The DWLA adapter's memory space holds the I/OMAP, device registers and the I/OBUS memory space. The Starting Address Register and Ending Address Register must be set to enable this space.

WRITE, WRITE — Most nodes ignore a VAXBI command. This command is similar to WRITE except the register selects the bytes of the addressed location that it wants to modify.

WRITE — A VAXBI command. The master node writes data to the slave.

APPENDIX - ع

APPENDIX C
SELF-TEST
MICRODIAGNOSTIC TESTS

The self-test microdiagnostic tests run in the order shown in Table C-1. Tests 1 through 3 check the DWBI 3 logic, tests 4 through 6 check the 3 WBI port logic, and tests 7 through 9 check the 3 WBI port logic and the 3 WBI S and its port logic.

Table C-1 Self-Test Microdiagnostic Tests

Test Number	Test Name	Description
1	29116 RAM Test	Verifies address/data and data integrity of last 16 locations of address processor RAM space. (Locations are used for storage of constants and DWBI 3 register addresses.) Other locations are verified in a later test.
2	DWBI 3 BAD Bit and BAD Register Test	Verifies Buffered Address (BADi) Bus, BAD Register, and BAD Output Mux of Data Path Gate Array.
3	BDP MAP IRAM March Test	Standard march test (14 data patterns) of Internal RAM (IRAM). Verifies that each RAM location is uniquely addressable, checks each location for data integrity. This test cannot differentiate between data and address failures.
4	BDP Bus Latch Test	Verifies logic levels of both the receiving and maintaining BDP bus latches.
5	IRAM Mask Chip Select Test	Verifies the chip select logic used when accessing the IRAM in mask mode.
6	BDP Stored Address Test	Checks that the DWBI 3 can properly execute Buffered Data Path instructions in verifying correct storage of buffers' addresses.
7	IRAM Address Increment Test	Verifies that the Data Path Gate Array can properly increment an IRAM address.
8	Translation Buffer Test	Verifies the integrity of the Translation Buffer.
9	29116 Condition Code Test	Performs a branch test on all condition codes that are not [illegible] other parts of this self-test.
A	29116 Instruction Test	Verifies that address processor can execute all functional macrocode instructions that are not otherwise executed during this self-test.

Table C-1 Self-Test Microdiagnostic Tests (Cont)

Test Number	Test Name	Description
8	Starting/Ending Address Registers Test	Performs VAXBI transactions. Writes to the DMR Starting Address and Ending Address Registers with the range of UNIBUS window space, reads the node ID from the DMR CSR, compares the corresponding values for the two registers, and then writes to each of these registers.
9	BDP Write Mask Test	Verifies the write mask flip-flops on the Data Path Gate Array. These flip-flops store the mask for a VAXBI extended WRITE mask transaction. (This transaction is executed whenever a Buffered Data Path is purged.)
10	VAXBI WAIT/READ Test	Verifies that the DWBUA can perform VAXBI READ and WRITE transactions to the DMR by performing word-length transactions on the VAXBI. (These operations are used by UNIBUS-to-VAXBI Direct Data Path transactions.) This test uses the DMR General Purpose Registers, and it verifies that both word and byte length transactions are possible from the UNIBUS to the VAXBI.
11	UNIBUS DATO/DATI Test	Uses the LFT module to verify that the DWBUA can write to and read from the UNIBUS. Performs a VAXBI WAIT instruction to set up the VAXBI Address Transceiver with the LFT module's Address Register address and to set a data pattern on the BDP bus. The test then operates similarly to the DWBUA functional microcode. Failure of this test indicates a problem in the UNIBUS cabling, power, or LFT module.
12	VAXBI-to-UNIBUS IRCI/UWMCI Test	This test ensures that VAXBI IRCI/UWMCI commands can be processed by the DWBUA. The DWBUA verifies that the corresponding DATIP/DATOB sequence functions properly on the UNIBUS. The test initially writes a known data pattern (AAAA hex) to the LFT Data Register. A DATIP is then issued to read this register. The DATIP is immediately followed by a DATOB. The address is driven on the UNIBUS through the duration of the DATIP/DATOB sequence. The data for the DATOB (5555 hex) is loaded into the Data Path Gate Array prior to initiation of the DATIP. After completion of the DATOB, the data from the DATIP is read from the Data Path Gate Array and verified in the address processor. A DATI is then issued to the LFT Address Register to verify that the DATOB completed properly.

Table C-1 Self-Test Microdiagnostic Tests (Cont)

Test Number	Test Name	Description
10	UNIBUS DATO/DATI Test	Verifies that a UNIBUS DATI command can execute through the the Direct Data Path. UNIBUS Map Registers are set up and the corresponding UNIBUS address is written into the LFT Address Register. A DATI is issued, and the test then waits for the UNA port request to come into the DWBUA. If it does, the incoming address is resolved through the UNIBUS Map Register. The test verifies that the correct UNIBUS Map Register was referenced by a macrocode jump with values from that register.
11	DWBUA Error Test	Attempts special-case transactions between the VAXBI and UNIBUS and verifies proper execution of these transactions. These special cases are: VAXBI READ of an unmated UNIBUS address, and a DWBUA RETRY response to the VAXBI due to the servicing of a concurrent UNIBUS request.
12	VAXBI INTR/IDENT Test	Verifies that a UNIBUS device can successfully interrupt the VAXBI and pass along its vector information. Writes the LFT CSR to generate a UNIBUS request. The LFT module is written and a UNIBUS BR is asserted. This causes the DMR to initiate a VAXBI interrupt to the DWBUA. The DWBUA processes a VAXBI IDENT command at the level corresponding to the INTR that was issued. The test generates the IDENT command.

APPENDIX D

APPENDIX D
MACRODIAGNOSTIC TESTS

\MDP

The macrodiagnostic error messages indicate the failing test number, the expected data, and the received data.

Table D-1 Macrodiagnostic Tests

Test Number	Subtest Number	Name
1		IR A Common and Device Logic Registers Test
	1	IR A Self Test and Register Subtest
	2	IR A Receive and Status Logic Subtest
2		IR A Registers Test
	1	VAXBI IR R Read Write Subtest
	2	VAXBI IMR R Read Subtest
	3	VAXBI Interrupt Destination Register Read/Write Subtest
	4	IR A VUR Read/Write Subtest
	5	VAXBI GPR Read/Write Subtest
3		Map RAM March Test
4		UNIBUS Read/Write Test
	1	Word Read Subtest
	2	Word Read/Write Subtest
	3	Word Read Byte Write Subtest
5		UNIBUS INTR READ/INTACK & WRITE Test
6		UNIBUS to VAXBI Wideramp Test
7		Data Path Select Test
8		Direct Data Path DATI Test
9		Direct Data Path DATOB Test
10		Buffered Address Register Test

Table D-1 Macrodiagnostic Tests (Cont)

Test Number	Subtest Number	Name
11		Buffered Data Path DATI Test
12		Buffered Data Path DATO Test
13		Buffered Data Path DATOB Test
14		Buffered Data Path Autosize Test
15		Byte Mask DATI Test
16		Byte Offset Bus DATO Test
17		Byte Offset BDP DATO Test
18		Byte Offset BDP DATOB Test
19		Byte Offset BDP DATOB Test
20		Page Boundary Transfer Test
	1	UET DATI Subtest
	2	UET DATO Subtest
	3	UET DATI/DATO Subtest
	4	UET DATO/DATI Subtest
21		BDP Byte to Octaword Transfer Test
	1	Address Match Octaword DATOB Subtest
	2	Address Match Octaword DATI Subtest
22		BDP Longword Access Enable Test
23		Bus Transceiver Test
	1	VAXBI to UNIBUS Bus Transceiver Subtest
	2	UNIBUS to VAXBI Bus Transceiver Subtest
24		Map Invalid Test
25		Map Entry Functional Test
26		CSR Status Bit Test
	1	BII and NEX Error Subtest
	2	REGDUMB Subtest
	3	USSTO Error subtest
	4	BADBDP Error Subtest
	5	IMR Error Subtest

Table D-1 Macrodiagnostic Tests (Cont)

Test Number	Subtest Number	Name
27		Interrupt Test
	1	UET BR7 Interrupt Subtest
	2	UET BR6 Interrupt Subtest
	3	UET BR5 Interrupt Subtest
	4	UET BR4 Interrupt Subtest
28		VAXBI Error Test
	1	UNIBUS Parity Bit Subtest
	2	UET Invalid BDP DATIP Subtest
29		Bus Int Test
	1	UNIBUS Int Subtest
	2	VAXBI STOP Command Subtest
30		IVBAR Register Test
31		UBI Multi Transfer Test
32		UBI Block Transfer Test

APPENDIX E

APPENDIX E
ERROR CONDITIONS

E.1 VAXBI-TO-UNIBUS TRANSACTIONS

E.1.1 Quadword and Octaword Transfers
The DWBUA accepts only valid longword transfers. The DWBUA responds to all quadword and octaword transfers with NO ACK.

E.1.2 BIIC Error EVENT Codes
Table E-1 lists the DWBUA responses to BIIC error EVENT codes. In the responses listed, the DWBUA sends error interrupts to the VAXBI only if interrupts are enabled.

Table E-1 DWBUA Responses to BIIC EVENT Codes

EVENT CODE					Mnemonic	DWBUA Response
H	L	H	L	L	IAI	The current BIIC command is ignored. No bus grant is withheld from the UNIBUS.
L	H	H	H	L	BPE	The current slave WRITE type transaction is ignored, and the data is not updated. If the transaction is a READ type, it is ignored. The BIIC sends an error interrupt.
L	H	H	H	H	STO	
L	H	H	L	H	XRSD	
L	H	H	L	L	RBE	
H	H	H	L	L	BSO	The BI ACKR bit is asserted. The BIIC sends an error interrupt. SSYN may be withheld from the UNIBUS device which would result in an SSYN timeout.
L	L	H	H	H	RDMR	
L	L	H	H	L	XCRA	
L	L	H	L	H	NCRA	
L	L	L	H	H	RXRMD	
L	L	L	H	L	RTO*	
L	L	L	L	H	BPM	
L	L	L	L	L	MTO	

* The DWBUA receives this error EVENT code only if the RTOEN bit in the DWBUA adapter's BICSR is asserted.

E.1.3 Mask Values
The mask value in a WRITE mask command is legal only if at least one mask bit is set in the word pointed to by address bit A1, and no mask bits are set in the other word of the longword. (The DWBUA responds with ACK regardless of the mask value.) The following are the only legal mask values:

A1=0 0000yyyy yy≠00
A1=1 yyyy0000

Any mask values that do not conform to this format are illegal. These illegal values either corrupt UNIBUS data or cause an SSYN timeout to the DWBUA.

E.1.4 Nonexistent UNIBUS Address

A valid WRITE or READ command is sent to a nonexistent UNIBUS address.

- The DWBUA responds to the WRITE command as ACK. It then sets the UNIBUS bit in the BUACSR, makes an error interrupt if interrupts are enabled, and writes the UNIBUS address to the Failed UNIBUS Address Register (bb+772561).

- The DWBUA sends zero data and an RIDO status code in response to the READ command. It also sets the UNIBUS bit in the BUACSR, makes an error interrupt if interrupts are enabled, and writes the UNIBUS address to the Failed UNIBUS Address Register (bb+772561).

E.1.5 Invalid VAXBI Command

A VAXBI command that the DWBUA considers an invalid results in a NO ACK response from the DWBUA. The VAXBI commands that the DWBUA considers invalid are:

- RESERVED (BCI 1-bus - 100000)
- INTR
- RESERVED (BCI 1-bus - 100010)
- RESERVED (BCI 1-bus - 100011)
- INVALIDATE
- BROADCAST
- IPINTR

E.1.6 Improper Use of a DWBUA Register

An attempt to improperly use a DWBUA register results in a RETRY response from the DWBUA. Improper use of a DWBUA register is:

- Attempted WRITE to a READ-ONLY bit in a DWBUA control register
- Attempted access of an unused address in the DWBUA register space

E.2 UNIBUS-TO-VAXBI TRANSACTIONS

E.2.1 VAXBI Error in UNIBUS-Initiated Transfer

- Direct Data Path and DATI through a Buffered Data Path

  The DWBUA does not assert SSYN to the UNIBUS device when a VAXBI error is encountered during a DDP transaction or during a DATI through a BDP. The DWBUA asserts the BUACSR BEI bit and writes the VAXBI address to the VAXBI Failed Address Register (bb+772560).

- DATO(B) through a Buffered Data Path

  The DWBUA asserts SSYN to the UNIBUS device before it checks for VAXBI errors during a DATO(B) through a BDP. The DWBUA causes an SSYN timeout during the next transfer within the present UNIBUS arbitration cycle. If the current transfer, however, is the last transfer within the present UNIBUS arbitration cycle, the UNIBUS device cannot be notified of the VAXBI error. The DWBUA asserts the BUACSR BEI bit and writes the VAXBI address to the VAXBI Failed Address Register (bb+772560).

E.2.2 Illegal Map Entries

- DMA access through an invalid map page

  A UNIBUS device might attempt a DMA access through an invalid map page (that is, the UNIBUS Map Register is INVALID bit is clear). If this happens, the DWBUA asserts the BUACSR IMR bit, makes an error interrupt if interrupts are enabled, and withholds SSYN, causing an SSYN timeout for the UNIBUS device.

- DMA access through an illegal BDP

  If a UNIBUS device attempts a DMA access through BDP 4 or 5, the DWBUA asserts the BUACSR BADBDP bit, makes an error interrupt if interrupts are enabled, and withholds SSYN, causing an SSYN timeout for the UNIBUS device.

- DATIP through a BDP

  If a UNIBUS device attempts a DATIP through any Buffered Data Path, the DWBUA withholds SSYN, causing an SSYN timeout for the UNIBUS device.

E.2.3 Illegal UNIBUS Transaction

DATI(B) must follow a DATIP but a BSYNC is encountered during the DATI(B), the DWBUA asserts the BUACSR UBE bit and makes an error interrupt if interrupts are enabled.

APPENDIX E

APPENDIX F
[illegible]
EXERCISER TERMINATOR

F.1 [illegible] EXERCISER TERMINATOR DESCRIPTION

The [illegible] Exerciser Terminator (F) [illegible] module is mounted in [illegible] and B of the test [illegible] slot. The [illegible] enables diagnostic testing of the DWBB [illegible] adapter's capabilities to handle [illegible] addressing, data transfers, and interrupts.

F.2 [illegible] EXERCISER REGISTERS

Table F-1 [illegible] Exerciser Terminator Registers

Register Address Switch	Register Name/Mnemonic	Notes
[illegible]	Address Register A [illegible]	Word load only. Byte loading causes timeout
[illegible]	Data Register D [illegible]	Both byte and word loading accepted
[illegible]	Control Register CR [illegible]	Word load only. Byte loading causes timeout

F.2.1 Control Register Format

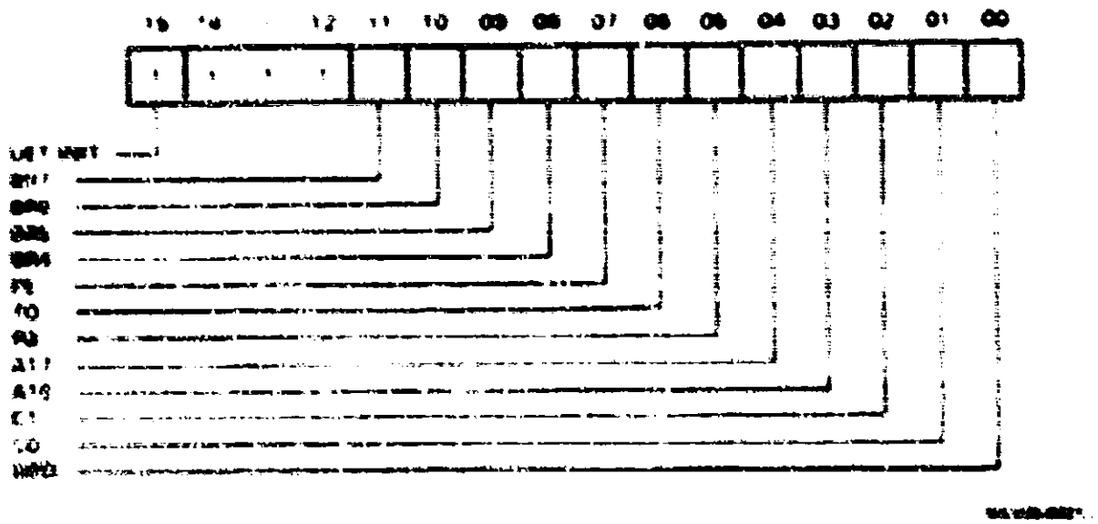


Figure F-1 [illegible] Control Register Format

5.2.2 Control Register Bit Descriptions

LET bit <R<15>>	Indicates LET to terminate receive operation. Use WRITE/MOVE bit always reads 1. It does not clear <R<4>>.
Unused <R<14:12>>	Always read as 0
BUS ERR <R<11>>	Write 1 to initiate interrupt
PE	Parity Error detected during LET DATI. Cleared on each LET DATI and cleared by LET bit
TO <R<9>>	Timeout when SSYN not returned. Cleared on each transfer. Cleared by LET bit
FB <R<8>>	Force Bit. When set the FB bit will be asserted when the LET Data Register is read. This bit is cleared by LET bit
A17, A16 <R<5:4>>	High-order UNIBUS addressing bits
Command <R<3:1>>	Transfer command bits (see Table 5-2)
GO <R<0>>	Write 1 to initiate transfer

Table 5-2 Transfer Command Bits

<3>	<2>	Command
0	0	LET DATI
0	1	LET DATIP
1	0	LET DATO
1	1	LET DATOB

NOTE

<R<13:6>> and <R<4>> (BR7 - BR4 and NPR respectively) remain set until the grant is returned at which time they are cleared. These bits are also cleared by writing a 0 to the bit or by writing a 1 to LET bit (<R<15>>). Multiple interrupts may occur if more than one bit is set.

5.3 NPR DATA TRANSFERS

5.3.1 LET WRITE

A LET WRITE consists of the following sequence of events:

1. Load Address Register (R<17:00>).
2. Load Data Register (R<15:00>).
3. Load Control Register to initiate the transfer.
   a. <R<4:1>> = A<17:14> of UNIBUS Address
   b. <R<7:1>> = 10 for DATO, 11 for DATOB
   c. <R<0>> = 1 (execute NPR)

5.3.2 LET READ

A LET READ consists of the following sequence of events:

1. Load Address Register (R<17:00>).
2. Load Data Register (R<15:00>).
3. Load Control Register to initiate the transfer.
   a. <R<4:1>> = A<17:14> of UNIBUS Address
   b. <R<7:1>> = 00 for DATI or 01 for DATIP
   c. <R<0>> = 1 (execute NPR)

NOTE

The LET does not send a DATIP following a DATIP. After it has completed the DATIP, the LET drops SSYN and releases the UNIBUS.

5.4 BR INTERRUPTS

The following sequence of events implements a BR interrupt:

1. Load the Data Register (R<15:00>) with the vector address.
2. Load Control Register (R<7:1>) with the BR (BR7 - BR4) level.

APPENDIX G

APPENDIX G
NODE SPACE AND WINDOW SPACE ADDRESSES

Table G-1 Node Space and Window Space Addresses

NODE NUMBER	NODE SPACE ADDRESSES		WINDOW SPACE ADDRESSES	
	Starting	Ending	Starting	Ending
0	[illegible]	[illegible]	[illegible]	[illegible]
1	[illegible]	[illegible]	[illegible]	[illegible]
2	[illegible]	[illegible]	[illegible]	[illegible]
3	[illegible]	[illegible]	[illegible]	[illegible]
4	[illegible]	[illegible]	[illegible]	[illegible]
5	[illegible]	[illegible]	[illegible]	[illegible]
6	[illegible]	[illegible]	[illegible]	[illegible]
7	[illegible]	[illegible]	[illegible]	[illegible]
8	[illegible]	[illegible]	[illegible]	[illegible]
9	[illegible]	[illegible]	[illegible]	[illegible]
A	[illegible]	[illegible]	[illegible]	[illegible]
B	[illegible]	[illegible]	[illegible]	[illegible]
C	[illegible]	[illegible]	[illegible]	[illegible]
D	[illegible]	[illegible]	[illegible]	[illegible]
E	[illegible]	[illegible]	[illegible]	[illegible]
F	[illegible]	[illegible]	[illegible]	[illegible]

APPENDIX H

APPENDIX H
REGISTER
INITIAL STATES

The initial state of each register is the state after successful completion of the IOR and DWBI A self-tests.

Table H-1 Register Initial States

Address (Hex)	Register	Initial State	Notes
00	Device Type	xxxx0101	xxxx = DWBI A version
04	VAXBI Control and Status	x0012x0x	x = VAXBI receive timeout x = DWBI A node ID (hex)
08	Bus Error	00000000	
0C	Error Interrupt Control	00000000	
10	Interrupt Destination	000Bxxxx	xxxx = decoded DWBI A node ID
14	IPINTR Mask	xxxx0000	xxxx = IPINTR mask
18	Force IPINTR STOP Destination	000Bxxxx	xxxx = force IPINTR STOP destination
1C	IPINTR Source	xxxx0000	xxxx = IPINTR source
20	Starting Address	xxx00000	xxxx = starting address of DWBI A adapter's window space (between 2000 and 207C, last digit 0, 4, 8, or C)
24	Ending Address	xxx00000	xxxx = starting address of window space after DWBI A (between 2004 and 2080, last digit 0, 4, 8, or C)
28	BI Control	0000200D	STOPEN, IDENTEN, and LCSRIEN bits set
2C	Write Status	10000000	
30	Force IPINTR STOP Command	00001600	
40	User Interface Interrupt Control	00000000	
70	GPR 0	00000001	LBIP = 1, self-test passed

Table H-1 Register Initial States (Cont)

Address (hex)	Register	Initial State	Notes
[illegible]	GPR [illegible]	[illegible]	
720	DWDR A Control and Status	[illegible]	
724	Vector Offset	[illegible]	
728	Failed UNIBUS Address	[illegible]	
72C	VAXBI Failed Address	[illegible]	
730-740	Microdiagnostic	[illegible]	
740	DPCSR 0	[illegible]	DPCSR = Data Path Control and Status Register
744	DPCSR 1	[illegible]	
748	DPCSR 2	[illegible]	
74C	DPCSR 3	[illegible]	
7xx	DPCSR 4	[illegible]	
7xx	DPCSR 5	[illegible]	
8200-83FC	UNIBUS Map	[illegible]	Locations invalid
8400-87FC	UNIBUS Map	[illegible]	I/O space addresses

APPENDIX

APPENDIX I
DATA PATH
OPERATION

I.1 DIRECT DATA PATH

The DWBUA starts the VAXBI section of a LNIBUS-initiated transaction immediately after it receives the LNIBUS command. The DWBUA issues SSYNC to the LNIBUS transaction only if the VAXBI transfer completes successfully. If an error occurs during the VAXBI transfer, the BI ACNO BIT bit is set and an error interrupt is issued by the BIIC if interrupts are enabled. The DWBUA may not issue SSYNC to the LNIBUS device, causing an SSYNC timeout.

The following two special cases must be noted for LNIBUS-initiated transactions through the Direct Data Path. In both cases, the BYTE CONTROL bit in the corresponding LNIBUS Map Register is set, causing the LNIBUS address to be incremented by one before the corresponding VAXBI transaction is completed.

CASE 1 - DATO WITH LNIBUS ADDRESS BIT <01> SET

Two VAXBI longword WRITE transactions, with the data and mask bits shown in Figure I-1, are performed.

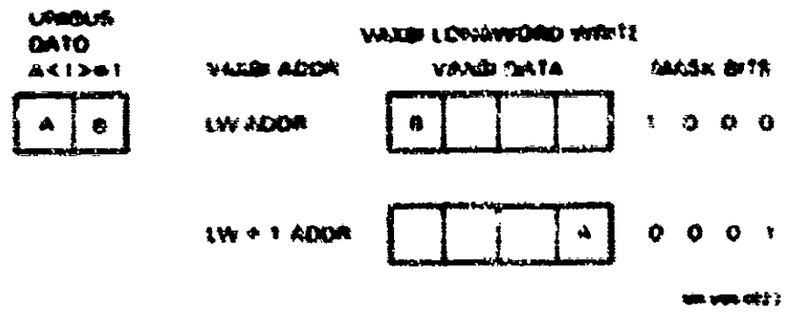


Figure I-1 DATO with LNIBUS Address Bit <01> Set

CASE 2 - DATI WITH UNIBUS ADDRESS BIT <01> SET

Two VAXBI longword READ transactions are performed. They obtain data for the UNIBUS transaction as shown in Figure 1-2.

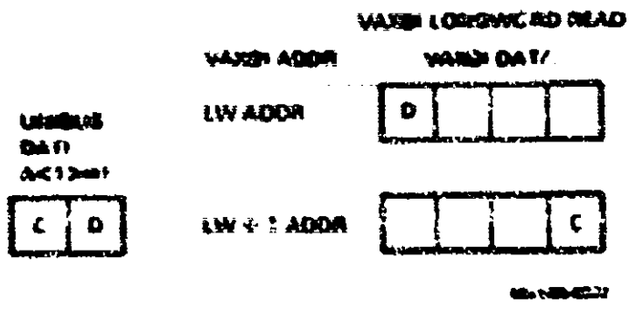


Figure 1-2 DATI with UNIBUS Address Bit <01> Set

For improved UNIBUS bandwidth, the DWBUA completes the corresponding UNIBUS DATO or DATOB transaction (by issuing SSYNI) prior to performing the VAXBI WRITE transfer. The DWBUA does not issue SSYNI as early for Direct Data Path DATI and DATIP transactions as it does for Buffered Data Path transactions, since the VAXBI transfer must first be completed in order to obtain the requested data.

1.3 BUFFERED DATA PATH

The DWBUA has five Buffered Data Paths (BDPs). Each BDP consists of three sections: a 16-byte buffer, a 16-bit address register, and a 16-bit status register.

1. Buffer - Each Buffered Data Path has a 16-byte buffer available for storage of as much as one octaword of data. The buffered data is naturally aligned at an octaword address. When the LWAIN bit is set in the UNIBUS Map Register for the current UNIBUS-to-VAXBI transaction, the buffer is virtually reduced to longword in length.

2. Address Register - The address register is a 16-bit register that contains UNIBUS address bits <17:04> in its most significant 14 bits. These 14 bits correspond to the data currently stored in the buffer. The least significant two bits of the address register are zero.

3. Status Register - Internal flags monitor the status of the data in the buffer. These flags are:

   BDBUF - VAXBI Data in Buffer
   UDBUF - UNIBUS Data in Buffer
   STRT_0 - Start Zero

   UDBUF and BDBUF are updated only during the first transaction through a Buffered Data Path. They indicate that either UNIBUS Data (UDBUF) or VAXBI Data (BDBUF) is being held in the buffer, as shown in Figure 1-3.

   UNIBUS-to-VAXBI buffered transactions do not necessarily cause the DWBUA to generate a VAXBI transfer. Rather, the DWBUA stores as much as one octaword of data locally.

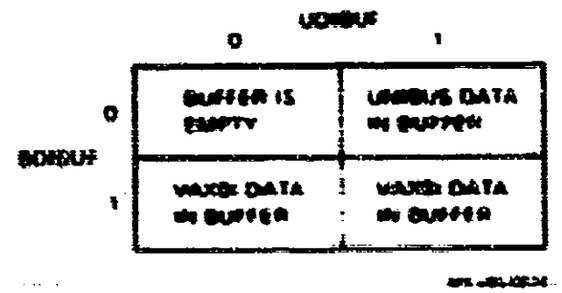


Figure 1-3 BDBUF and UDBUF Flags

When UDBUF is set, the DWBUA also updates the STRT_0 flag. The STRT_0 flag indicates that the first transaction through the Buffered Data Path began at an aligned octaword address (i.e., 10 - - 0000). When the last byte in the buffer is written, the DWBUA sets the STRT_0 flag. If STRT_0 is set, the DWBUA assumes that the buffer contains a full octaword of valid data. The DWBUA then purges the data by performing an octaword WRITE (unmasked) transaction. If STRT_0 is not set, the DWBUA performs an octaword WMCI operation when writing the buffer to the VAXBI.

Each Buffered Data Path has its own status register and address register. These registers can be read by using the REGDMP feature, as explained in Section 3.2.4.1.

1.3.1 Definitions

Three common terms used in discussing Buffered Data Path behavior are Address Match, Autopurge, and Write-to-VAXBI. These terms are defined as follows:

1. Address Match - The BDP address register holds the UNIBUS address of the current octaword of data stored in the buffer. When another UNIBUS-to-VAXBI transaction is received, bits <17:04> of the incoming UNIBUS address are compared to the stored address. If the addresses match, the DWBUA manipulates the data in the buffer. If the addresses do not match, however, and the buffer contains UNIBUS data, then the DWBUA performs an autopurge.

2. Write-to-VAXBI - This term describes the process of writing UNIBUS data in a buffer to the VAXBI when the buffer is full. When LWAIN is not set and the buffer is full, the DWBUA checks the buffer's STRT_0 flag. If the flag is set, the DWBUA assumes that a full octaword of data is being held in the buffer. The Write-to-VAXBI will be performed using a VAXBI octaword WRITE. If the STRT_0 flag is not set or if the LWAIN bit is set, then the DWBUA assumes that the buffered transaction began with a nonaligned octaword address. Only part of the buffer contains valid data and the Write-to-VAXBI is performed using a VAXBI octaword WMCI.

3. Autopurge - If the buffer is not full and UNIBUS data is in the buffer, two occurrences will cause the data in the buffer to be written to the VAXBI. They are:

   a. A DATI is requested through the Buffered Data Path.

   b. A DATUBx is requested, but the addresses of the transaction and the data in the buffer do not match.

   Data is written from the buffer using a VAXBI octaword WMCI command with the possible mask bits set for each valid data byte. This act of writing the partially filled buffer to the VAXBI due to an address mismatch or mixed transaction types is known as autopurge.

3.2.2 BYTE OFFSET Bit Clear

The following three cases describe the behavior of the DWBUA depending on the contents of the BDP buffer. For each case, assume that a UNIBUS-to-VAXBI transaction is requested, and the BYTE OFFSET bit in the UNIBUS Map Register is not set.

CASE 1 - THE BUFFER IS EMPTY

The UNIBUS master is attempting a DATI through a valid UNIBUS Map Register. The DWBUA performs an outward READ of VAXBI data and fills the BDP buffer. The DWBUA then places the requested data on the UNIBUS, issues SSYNC, updates the BDP flags by setting BDPBUF and clearing UDBIT, and stores the address value for the Buffered Data Path.

The UNIBUS master is attempting a DATO(B) through a valid UNIBUS Map Register. The DWBUA updates the BDP flags by setting UDBIT and clearing BDPBUF, stores the incoming UNIBUS address, issues SSYNC, and stores the data in the appropriate bytes of the BDP buffer with the correct mask bits set.

CASE 2 - THE BUFFER CONTAINS UNIBUS DATA

1 The UNIBUS master requests a DATI.

The BDP buffer contains UNIBUS data, the current data in the buffer is unmapped. Once the unmapper is complete, the DWBUA treats the DATI request as it did in CASE 1 (since the buffer is empty).

2 The UNIBUS master requests a DATO(B).

The BDP buffer contains UNIBUS data. The DWBUA checks for an address match. If the addresses do not match, the incoming UNIBUS address and data are temporarily stored within the DWBUA and the data currently in the BDP buffer is unmapped. Once the unmapper is complete, the DWBUA issues SSYNC. The DWBUA then loads the BDP address register with the address of the temporarily stored data. The DWBUA stores the data in the appropriate bytes of the BDP buffer, with the correct mask bits set.

If the addresses do match, the DWBUA first issues SSYNC, then stores the data in the BDP buffer. If the DATO(B) writes the last byte in the buffer, the DWBUA performs a Write-to-VAXBI and marks the buffer as empty.

CASE 3 - THE BUFFER CONTAINS VAXBI DATA

The UNIBUS master requests a DATI; the buffer contains VAXBI data. The DWBUA checks for an address match. If the addresses do not match, the buffer is treated as if it were empty. The buffer is overwritten with the new octaword of VAXBI data (see CASE 1). If the addresses do match, the requested data is taken from the buffer, placed on the UNIBUS, and SSYNC is issued.

The UNIBUS master requests a DATO(B); the buffer contains VAXBI data. The DWBUA treats the buffer as if it were simply full (CASE 1).

3.2.3 BYTE OFFSET Bit Set

When the UNIBUS Map Register BYTE OFFSET bit is set, the DWBUA-A services requests as much the same way as when that bit is clear. The only exception is that the incoming UNIBUS address is incremented prior to address matching and storage. The following special cases, however, can occur when the BYTE OFFSET bit is set.

CASE 1 - UNIBUS ADDRESS A<1:0> = 11 (BYTE OFFSET BIT IS SET)

The address is incremented so that A<1:0> = 11 (3). A word-length transaction at this address crosses an octaword boundary.

1 The UNIBUS master requests a DATI.

If the BDP buffer contains VAXBI data, the DWBUA checks for an address match. If the addresses match, the DWBUA temporarily stores the low byte of the octaword. If the addresses do not match, the DWBUA requests a VAXBI octaword READ. When the READ transaction is complete, the DWBUA temporarily stores the low byte of the octaword.

Once the low byte of data is stored within the DWBUA, the high byte of data is fetched by incrementing the incoming UNIBUS address and remapping, and requesting a VAXBI octaword READ for the next higher octaword address. Because the next octaword address must be remapped, the next UNIBUS Map Register must have the same value in the DATA PATH SELECT field as the current UNIBUS Map Register. If it does not, data integrity for all Buffered Data Paths cannot be ensured. When the READ transaction is complete, the first byte of the octaword is fetched and concatenated with the temporarily stored low byte to form a word of UNIBUS data. This word is placed on the UNIBUS and SSYNC is issued.

If the BDP buffer contains UNIBUS data, the DWBUA ignores the transaction as a special case. The DWBUA does not unmap the buffer as it does in other Direct Data Path transactions. Instead, the DWBUA READs a longword of VAXBI data through the Direct Data Path. This longword of data contains the low bytes of the requested word, which is stored internally. Next, the DWBUA swaps the next longword of data which falls in the next octaword boundary on VAXBI memory. The corresponding map register must have the same value in the Data Path Select field as the current map register in order to ensure data integrity. The DWBUA then READs the longword and fetches the high bytes of data, which it concatenates to the previously stored lower byte, forming a word of UNIBUS data. The DWBUA places this word on the UNIBUS and issues SSYNC. Then, as in the special case, the data stored in the buffer remains unchanged and the transaction is carried out through the Direct Data Path.

2 The UNIBUS master requests a DATO.

If the buffer is empty or if it contains VAXBI data, the low byte of the incoming data is written into the low byte of the buffer, and a Write-to-VAXBI is performed. This generates an octaword WRITE transaction with only one byte of valid data. If the buffer contains UNIBUS data, the DWBUA checks for an address match. If the addresses match, the low byte of the incoming UNIBUS data is written to the last byte of the octaword, and a Write-to-VAXBI is performed. If the addresses do not match, the buffer is unmapped. Once the unmapper is complete, the low byte of the incoming UNIBUS data is written to the last byte of the octaword buffer. A Write-to-VAXBI is performed, where only the last byte of the octaword contains valid data.

Once the low byte has been written to the VAXBI, the high byte is written into the buffer with the appropriate mask. The flags are updated by setting UDBIT and clearing BDPBUF. The incoming address is incremented to the next octaword, and stored, and SSYNC is issued. The BDP is left in the state it would be in a DATO(B) had been performed to an octaword-aligned address with an byte offset.

(... 2 – ... ADDRESS ... = ... OFFSET ... ALIGN BITS ARE SET)

... is handled similarly to ... the ... ALIGN ... The only difference is that the multiple transactions that are generated (as explained above) occur each time a longword boundary is crossed rather than at ... boundaries.

1.3.4 Example

Figures 1-4 through 1-6 show the ... of a BDP buffer for various multiple DATOB(I) transactions. Each valid byte of data in the buffer has a corresponding mask bit set. All other mask bits for the BDP buffer are clear.

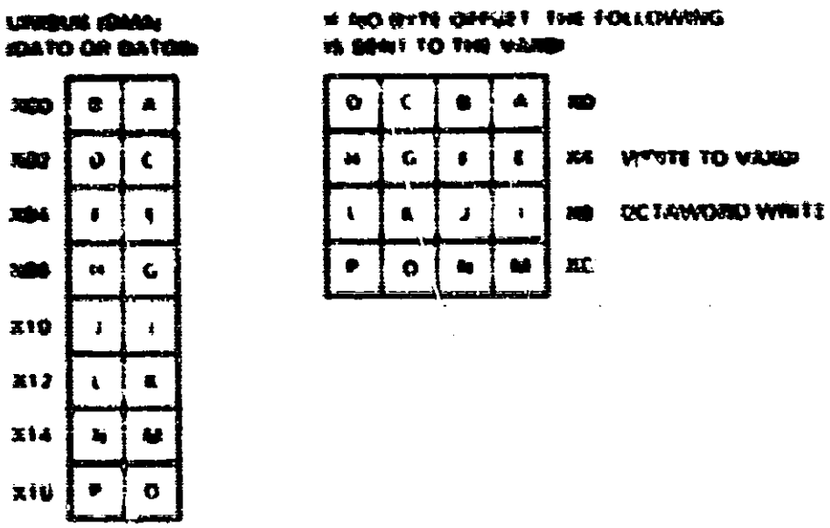


Figure 1-4 DATOB(I) Through BDP, BYTE ... (... Stabilizing at Octaword Boundaries

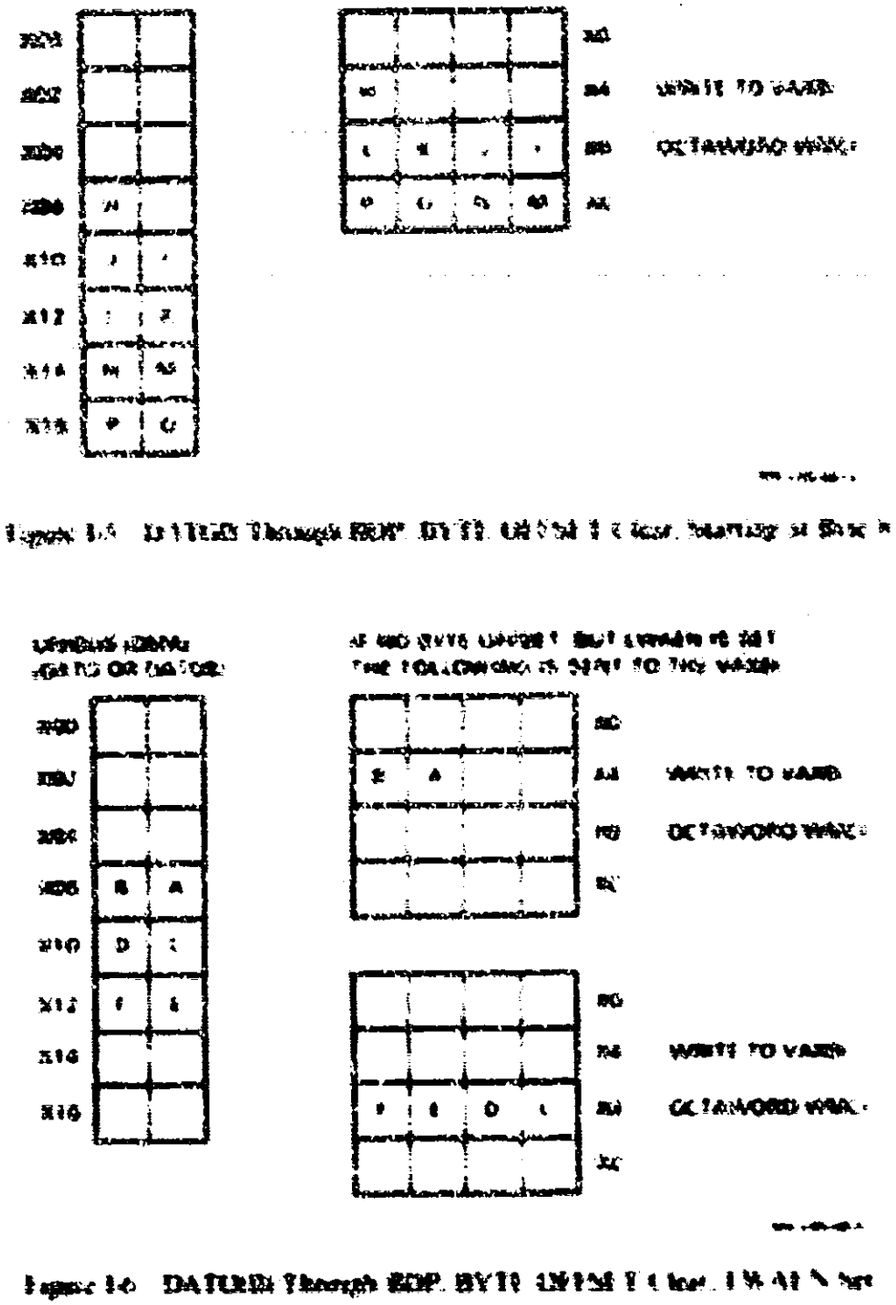


Figure 1-5 DATOB(I) Through BDP, BYTE ... (... Stabilizing at Octaword Boundaries

Figure 1-6 DATOB(I) Through BDP, BYTE ... (... Stabilizing at Octaword Boundaries

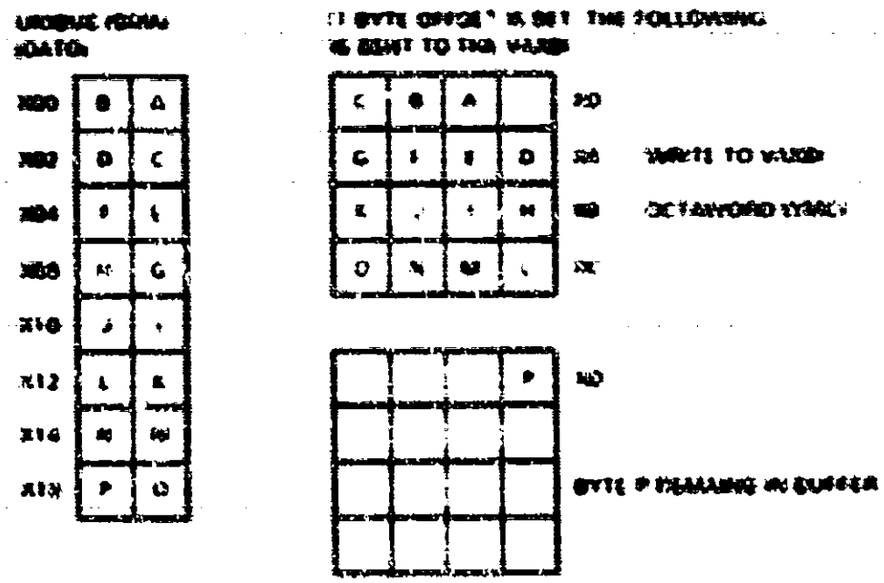


Figure 1-7 DATO Through BDP, BYTE OFFSET Set

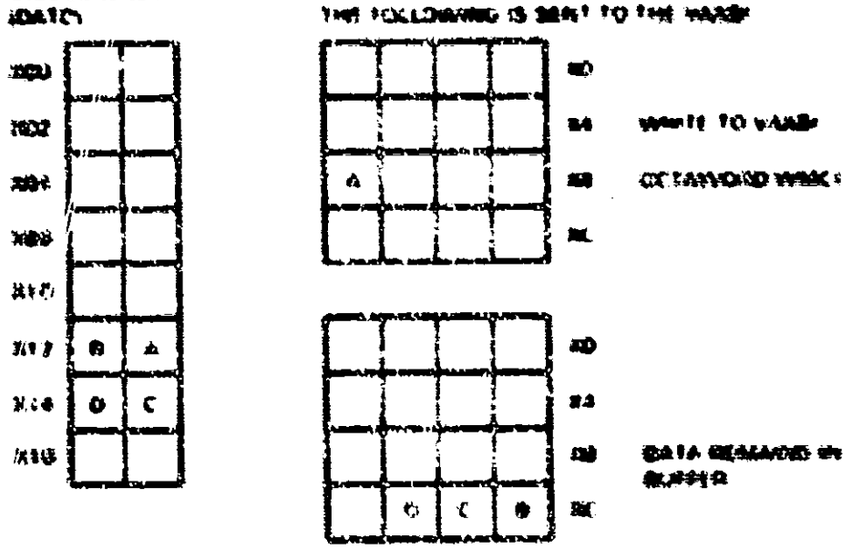


Figure 1-8 DATO Through BDP, BYTE OFFSET and [illegible] Set

APPENDIX J

APPENDIX J
PORT LOCK, RETRY, AND INTERRUPT MECHANISMS

J.1 PORT LOCK MECHANISM

The DWBR A has a port lock mechanism to ensure that a processor only runs transactions at a time. The mechanism locks the VAXBI and LAMB ports from accepting new transactions until the DWBR A is able to service another request. While the DWBR A is locked it sends RETRY to all odd numbered VAXBI transactions except the STOP command. The DWBR A unconditionally sends an ACK response to the STOP command. The DWBR A also disables the LAMB's arbitrator from issuing grants to LAMB devices while it is locked.

The DWBR A is locked during the following four circumstances and sends a RETRY response to all VAXBI transactions until it is:

1. The DWBR A has accepted a VAXBI transaction. The lock is released when the DWBR A has completed servicing the transaction. However, the VAXBI transaction is not RETRY the DWBR A sends a RETRY response to all transactions until a CLEAR command is sent to the DWBR A.

2. A LAMB to DWBR A request is granted. The lock is released when LAMB BDOUT is negated by the LAMB master.

3. The DWBR A issues a VAXBI transaction (such as interrupt or the issuing of an error interrupt on the VAXBI). The port lock is released when the transaction is completed.

4. The DWBR A is the VAXBI master the slave VAXBI node responds to its transaction with a RETRY. The DWBR A then sends a RETRY response to all subsequent outstanding VAXBI transactions until its RETRYed master transaction has successfully completed.

J.2 RETRY MECHANISM

The RETRY mechanism reduces the number of RETRY responses sent by the DWBR A to VAXBI master nodes. It works by disabling and enabling the LAMB's arbitrator.

When the DWBR A sends a RETRY response to a valid VAXBI command that it would otherwise accept, the DWBR A also disables the LAMB's arbitrator. The LAMB's arbitrator is enabled again when the DWBR A, as a slave node on the VAXBI, sends an ACK response to any VAXBI command.

A VAXBI node that receives a RETRY response from the DWBR A should keep requesting the DWBR A until it receives an ACK response. In this way, the VAXBI node ensures that its transaction will be the next one serviced by the DWBR A.

3.3 UNIBUS INTERRUPTS

Interrupts are permitted only from the UNIBUS to the VAXBI

3.3.1 Interrupt/IDENT Sequence

A BR of any level generates a VAXBI INTR transaction at the same level. The DWBUA does this by asserting the corresponding BCI INT line. The BIIC then performs the VAXBI INTR transaction.

The DWBUA responds to any VAXBI IDENT that meets two conditions:

1. The DWBUA must have a pending interrupt at the same level, and
2. The VAXBI master's decoded ID must match the ID in the Interrupt Destination Register (bits<10>).

Figure 3-1 is a flow diagram of the IDENT transaction. In the explanation that follows the figure, the numbered paragraphs refer to the numbers in the figure.

Figure 3-2 is a timing diagram of the interrupt/IDENT sequence. The following assumptions apply:

1. No errors occur during the transaction.
2. The transaction follows a straightforward path through the flows.
3. No time scale is employed. The diagram indicates relative timing only.

In the diagram, the device name that appears in parentheses under any waveform is the device that asserts that signal.

3.3.2 Passive Release

When some UNIBUS devices become bus master under BR-BG transactions, they drop BUSY and SACK and never issue an interrupt vector or assert INTR. This is known as a passive release. Passive release causes the DWBUA to send a zero vector back to the VAXBI, but the DWBUA does not flag an error interrupt.

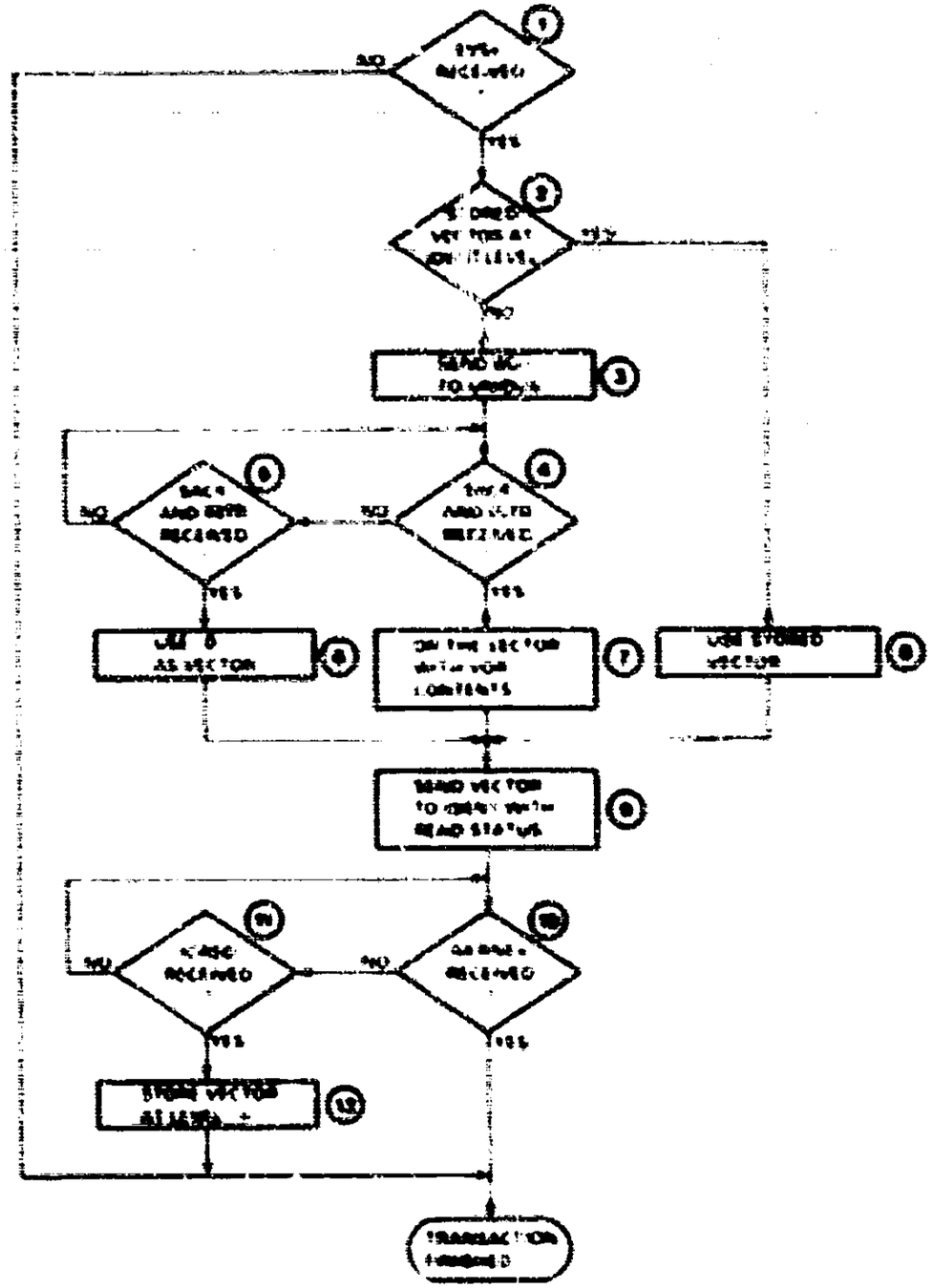


Figure 3-1 IDENT Flow Diagram

(1) The DWBLA checks that it has been selected by verifying that it has received an "External Vector Selected" EV code (EVSx). Receiving this EV code means that the DWBLA had an interrupt pending at the IDENT level and that the DWBLA adapter's Internal Destination Register (bits 7-0) contains the same decoded ID as the IDENT. The DWBLA then checks for the "IDENT Arb Lost" EV code to ensure that it has won the IDENT arbitration.

The DWBLA begins to service an incoming IDENT command only after it verifies that it has been selected for the IDENT and that it has won the IDENT arbitration. If these conditions are not met, the Interrupt/IDENT transaction is aborted.

(2) The DWBLA determines if it had a previous failed IDENT command at the present IDENT level.

(3) If it did not have a previous failed IDENT command at the present IDENT level, the DWBLA issues a BG to the interrupting UNIBUS device at the level indicated by the IDENT command.

(4) The DWBLA checks that SACK and INTR have both been received. This indicates that the interrupting UNIBUS device has given an expected response, placed the interrupt vector on the UNIBUS data lines, asserted INTR, and then deasserted SACK. The DWBLA issues SSYN and completes the UNIBUS transaction.

(5) If the interrupting UNIBUS device fails to respond to the BG, the UNIBUS terminator asserts SSYN and then deasserts SACK. When SACK is deasserted without the prior assertion of INTR, the DWBLA detects a SACK timeout.

(6) The DWBLA continues servicing the IDENT transaction normally, but it uses "0" as the interrupt vector.

(7) The DWBLA ORs the received vector with the contents of its Vector Offset Register (bits 7-3). This becomes the interrupt vector.

(8) The DWBLA uses the internally stored vector as the interrupt vector.

(9) The interrupt vector is placed on the BERR D lines along with a Read Data Status code.

(10) When the DWBLA receives the "Data Received for Slave Error" EV code (DATRSE), the IDENT has completed properly, and the DWBLA returns to its idle state.

(11) If the DWBLA receives an "Illegal CNF Received for Slave Data" EV code (ILRSD), the VAXBI master has not successfully received the IDENT vector.

(12) The DWBLA stores the failed IDENT command vector. This vector will be provided for any subsequent IDENT command at that particular level.

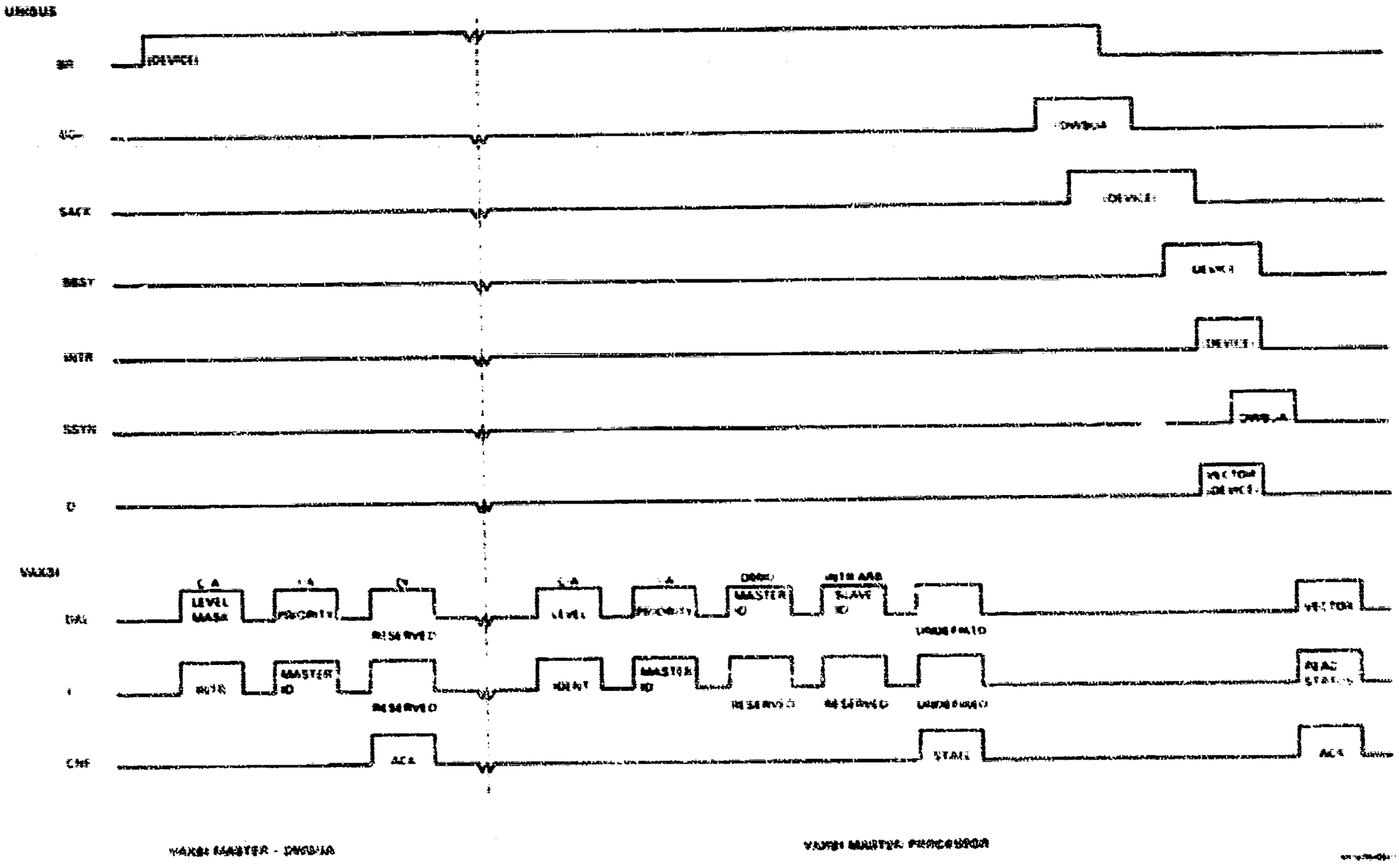


Figure J-2 Interrupt/IDENT Timing Diagram

APPENDIX K

APPENDIX K
MSTN-6STN
TIME INTERVALS

NOTE

The MSTN - 6STN time intervals listed in Table K-1 may change with enhancements to the DSN. A update. The times listed were valid at the time of publication of this manual.

The following conventions are used in Table K-1:

- All measurements listed in brackets (||) are performed after MSTN is tuned to the LNBR S signal.
- The data means that the BBP buffer contains the LNBR S DSN data to be sent to the VAXBI.
- The data means that the BBP buffer contains the LNBR S DSN data received from the VAXBI.

Table A-1 [illegible] Timer Intervals

Path	Byte Offset (hex)	VMEbus Command	VMEbus Address Width	Data Path Action	VMEbus Transaction	Max Data Rate (MB/s)	See Note
DDP	0	DATI	A32	32	1 BLT READ	27.9	
DDP	4	DATIP	A32	32	1 RMC	22.2	1
DDP	6	DATO	A32	32	1 BLT WRITE or 1 BLT WRITE	31.2	2
DDP	8	DATOB	A32	32	1 BLT WRITE or 1 BLT WRITE	31.2	
DDP	A	DATI	A32	32	1 BLT READ (1 BLT READ)	27.9	3
DDP	C	DATIP	A32	32	N/A	50.0/s command	
DDP	E	DATO	A32	32	1 BLT WRITE (1 BLT WRITE)	31.2	3
DDP	F	DATOB	A32	32	1 BLT WRITE	31.2	
DDP	0	DATI	A32	Longword	1 BLT READ	27.9	4
DDP	2	DATI	A32	Longword & match	1 BLT READ	27.9	
DDP	4	DATI	A32	Longword no match	1 BLT READ	27.9	
DDP	6	DATI	A32	Quadword	1 BLT READ (1 BLT READ)	31.2	
DDP	8	DATI	8 or 0	Longword	1 BLT READ	27.9	
DDP	A	DATI	8 or 0	Longword & match	1 BLT READ	27.9	
DDP	C	DATI	8 or 0	Longword no match	1 BLT READ	27.9	
DDP	E	DATI	8 or 0	Quadword	1 BLT READ (1 BLT READ)	31.2	
DDP	0	DATI	0	Longword	1 BLT READ (1 BLT READ)	27.9	
DDP	2	DATI	0	Longword & match	1 BLT READ (1 BLT READ)	27.9	
DDP	4	DATI	0	Longword no match	1 BLT READ (1 BLT READ)	27.9	
DDP	6	DATI	0	Quadword	1 BLT READ (1 BLT READ)	31.2	4
DDP	8	DATIP	A32	Longword or octaword	N/A	50.0/s command	
DDP	A	DATO	A32	Longword or octaword	1 BLT WRITE	31.2	5
DDP	C	DATO	A32	Quadword & match	1 BLT WRITE	31.2	
DDP	E	DATO	A32	Quadword no match	1 BLT WRITE (1 BLT WRITE)	31.2	
DDP	0	DATO	8 or 0	Longword or octaword	1 MBLT transaction	31.2	
DDP	2	DATO	8 or 0	Quadword & match	1 MBLT transaction	31.2	
DDP	4	DATO	8 or 0	Quadword no match	1 BLT WRITE	27.9	
DDP	6	DATO	0	Longword or octaword	1 BLT WRITE	27.9	
DDP	8	DATO	0	Quadword & match	1 BLT WRITE	27.9	
DDP	A	DATO	0	Quadword no match	1 BLT WRITE (1 BLT WRITE)	31.2	
DDP	C	DATOB	A32	Longword or octaword	1 BLT WRITE	31.2	
DDP	E	DATOB	A32	Quadword & match	1 BLT WRITE	31.2	
DDP	F	DATOB	A32	Quadword no match	1 BLT WRITE (1 BLT WRITE)	31.2	
DDP	1	DATOB	A32	Longword or octaword	1 BLT WRITE	31.2	
DDP	3	DATOB	A32	Quadword & match	1 BLT WRITE	31.2	
DDP	5	DATOB	A32	Quadword no match	1 BLT WRITE (1 BLT WRITE)	31.2	

NOTES FOR TABLE A-1:

(1) A DATIP command is valid only through the Direct Data Path. If a DATIP is attempted through a Buffered Data Path or through the Direct Data Path with the BYTE control line set, the VMEbus command is ignored and the DWB A does not issue AS* to command an MBLT transaction. During the DATIP transaction, the DWB A receives RETRY responses until the VMEbus device releases BBSY*.

If a DATIP command is not followed by a DATIPB, the DWB A sets the LIP bit in the BLMSR and issues an error interrupt if interrupts are enabled.

(2) A VMEbus DATOB through the Direct Data Path translates into a longword WRITE transaction with the mask bits set for each valid data byte.

(3) The DWB A performs two longword transactions on the VMEbus for each octaword transfer through the Direct Data Path if both the BYTE control line and VMEbus address bit A [illegible] are set.

(4) A VMEbus DATI command through a Buffered Data Path results in an initial READ VMEbus cycle if the requested data is not in the DDP buffer. If the VMEbus cycle is successful, the DDP buffer is updated; the buffer must be purged by performing an unlatched WRITE to the VMEbus before reading the data from the VMEbus. The entire octaword is loaded into the buffer; subsequent accesses within the octaword through the same Buffered Data Path cause the DWB A to fetch the data from the buffer until the VMEbus transaction is requested.

(5) This special case is treated differently from other Buffered Data Path transfers to avoid data corruption. In this case, the low byte of the requested DATI word is latched by performing a longword READ through the Direct Data Path. The high byte is latched from either the current DDP buffer or the VMEbus with a longword READ through the DDP. The current DDP buffer remains unchanged during this transaction.

(6) A DATIP transaction is valid only through the DDP.

(7) Using a DATOB command through a DDP is valid only if the buffer is full. The DWB A then performs a VMEbus octaword WRITE (unmodified) if the buffer contains an entire octaword of valid data from the VMEbus. A VMEbus longword WRITE is performed if the buffer contains less than a complete octaword of valid data.

APPENDIX

APPENDIX L
DWBUA PARITY CHECKING

L.1 PARITY CHECKING

The DWBUA uses its 32-bit internal RAM as an internal storage source. When the RAM is updated, the [illegible] section of the DWBUA stores odd parity for the updated data in a separate RAM. The DWBUA checks for a parity error every time the internal RAM is read, thus verifying data integrity. This appendix describes the DWBUA adapter's parity checking and parity error reporting scheme.

L.2 INTERNAL RAM

The DWBUA adapter's internal RAM contains the UNIBUS Map Registers, BDP buffers, vector registers, and other DWBUA internal registers. The internal RAM is read during such operations as a VAXBI-requested READ of a DWBUA internal register, a UNIBUS [illegible], reading of a UNIBUS Map Register, a data transfer involving reading a BDP buffer, and any other operation that requires the DWBUA to read the contents of its internal RAM. A parity error can occur during any of these operations. The DWBUA handles a parity error slightly differently in each case.

L.3 PARITY ERRORS

In the register section of the DWBUA, which is capable of detecting parity errors, the DWBUA Control and Status Register contains these two parity-related bits:

PARITY ERROR <29>	(W1C, DEFAULT 0)	This bit is set if the DWBUA detected a parity error while reading its internal RAM.
PARITY DISABLE <21>	(R/W, DEFAULT 0)	When this bit is set, the DWBUA does not generate odd parity when writing to its internal RAM. This causes a parity error when the same RAM location is read later. This bit is for use in Digital Equipment Corporation diagnostics only and should not be used for any other purpose.

When the DWBUA detects a parity error while reading its internal RAM, it sets the PARITY ERROR bit in the BUACSR and sends an error interrupt to the VAXBI bus, if interrupts are enabled. The remaining sections describe the other effects of a parity error detected during various operations.

L.3.1 Parity Errors on UNIBUS Map Registers

A UNIBUS Map Register enables mapping of an 18-bit UNIBUS address to a [illegible] VAXBI address. If a parity error occurs during a UNIBUS Map Register READ, it can cause the DWBUA to transfer the data to a different location from the correct target address. Hence, the DWBUA does not initiate the corresponding VAXBI transaction if it detects a parity error when reading a UNIBUS Map Register.

When a UNIBUS device initiates a transfer, the DWBUA reads the corresponding UNIBUS Map Register before servicing the UNIBUS data transfer. If a parity error occurs during this time, the DWBUA withholds an SSYN, causing the UNIBUS device that initiated the transfer to detect an SSYN timeout. The DWBUA then proceeds to report a parity error to the VAXBI. The data transfer associated with the error is not completed.

The UNIBUS Map Register may also be read when the DWBUA WRITE [illegible] READ data during a DATO(B) or DATI transfer through a Buffered Data Path. If a parity error occurs when the UNIBUS Map Register is being read, the DWBUA does not send the VAXBI [illegible]. The DWBUA withholds SSYN [illegible] the UNIBUS [illegible] SSYN has not been previously [illegible] causing an SSYN timeout to the UNIBUS device. The DWBUA then reports a parity error to the VAXBI. The DWBUA [illegible] the [illegible] and [illegible] flags for the current Buffered Data Path indicating that the BDP buffer is empty.

1.3.2 Parity Errors on BDP Buffers

The DWBUA BDP buffers are in the internal RAM. These buffers are read under the following conditions:

1. The BDP buffer contains the VAXBI data that is requested by the UNIBUS device initiated DATI transfer. If a parity error occurs while this data is being read from the BDP buffer, the DWBUA does not send the data to the UNIBUS device. The DWBUA withholds SSYN, causing an SSYN timeout. The DWBUA then reports a parity error to the VAXBI.

2. When a BDP buffer is full or has been [illegible], the DWBUA initiates an octaword WRITE (MASK) transfer on the VAXBI. The DWBUA reads its internal RAM for each longword of data to be shipped. If a parity error occurs during the reading of one of the four longwords of data, the DWBUA completes the VAXBI transfer with the data that has the parity error. The DWBUA then reports the error to the VAXBI. The DWBUA clears its BDP flags, indicating that the Buffered Data Path is clear. The DWBUA may withhold SSYN if it has not been previously [illegible] causing an SSYN timeout. Hence, it may not be possible for the VAXBI processor to detect which data in the VAXBI memory has a parity error.

1.3.3 Parity Errors on Vector Registers

The DWBUA may receive an IDENT [illegible] from the UNIBUS during an INTR command. When this happens, the DWBUA stores the [illegible] in its internal RAM. Subsequently, the next time the DWBUA receives an IDENT command at the [illegible], the DWBUA uses the stored data as the vector. If the DWBUA detects a parity error during the reading of the [illegible] vector, it sends zero data, a READ DATA status code, and an ACK response to the VAXBI master which initiated the IDENT command. This should be treated as a passive release by the IDENTing master. The DWBUA then reports a parity error as specified in Section 1.3.

1.3.4 Parity Errors on DWBUA Internal Registers

When the VAXBI master device READs [illegible] from a DWBUA internal register, the DWBUA reads the data from the internal RAM and sends the data to the VAXBI master device with a READ DATA status code and an ACK response. If the DWBUA detects a parity error when reading the data from the internal RAM, it sends zero data, a READ DATA [illegible] Read Status code, and an ACK response. The DWBUA then reports a parity error as specified in Section 1.3.

1.4 PARITY ERROR TESTING

The DWBUA [illegible] most of the logic associated with parity [illegible] parity for each longword of the internal RAM and [illegible] the DWBUA [illegible] The DWBUA cannot, however, [illegible] WRITE (MASK) [illegible] VAXBI [illegible] This means that the logic that detects a parity error during one of the [illegible] verified.

The PARITY DISABLE [illegible] generation logic. Setting PARITY DISABLE allows the DWBUA [illegible] a parity error and [illegible] the DWBUA detects the error [illegible] of the functions listed under Section 1.3 and verifies that the DWBUA [illegible] a parity error during the octaword WRITE (MASK) [illegible] and verifies that the DWBUA has detected the parity error and reported [illegible].

INDEX

INDEX