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)

CONTENTS (Cont)

FIGURES

FIGURES (Cont)

TABLES

PREFACE

MANUAL STRUCTURE AND AUDIENCE

This manual is divided into two parts:

Part I – Installation

Part I includes an introduction to the DWBUA, product specifications, and instructions for installing and testing a DWBUA option. It is intended for DIGITAL personnel or customers who install this 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 engineer, 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 DWBUA is one of a family of processors, memories, and adapters that use the 32-bit VAXBI bus. For a technical summary of the VAXBI bus and a description of VAXBI options, see the *VAXBI Options Handbook* – EB-27271-46.

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

Part I
Installation

Part I
Installation

CHAPTER 1
INTRODUCTION

1.1 PRODUCT DESCRIPTION

The VAXBI to UNIBUS Adapter (DWBUA), enables transfers between the high-speed synchronous VAXBI and the asynchronous UNIBUS. Through the DWBUA, the VAXBI has access to any UNIBUS address space, and the UNIBUS has access to any VAXBI address space.

The DWBUA 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 DWBUA internally buffers as much as one octaword (16 bytes) of data per transfer to maximize the VAXBI bandwidth.

All VAXBI-initiated transactions transfer data through the Direct Data Path. UNIBUS-initiated transactions can transfer data through either the Direct Data Path or a Buffered Data Path.

Other features of the DWBUA are:

- UNIBUS arbitrator
- UNIBUS devices can interrupt on the VAXBI
- Data transfer rate up to 1.0M b/s
- Self-test to verify data paths and control logic, and to report failures to the VAXBI

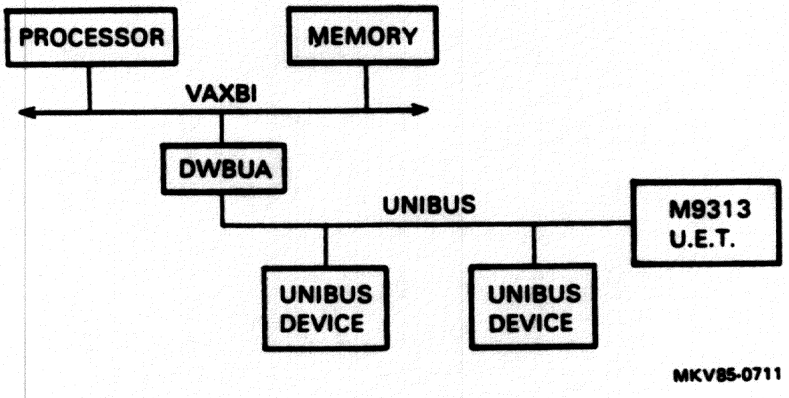


Figure 1-1 Typical DWBUA Configuration

1.2 SPECIFICATIONS

1.2.1 Bus Loading

The DWBUA is 0.5 dc unit load on the UNIBUS.

The DWBUA is 3.5 ac unit load on the UNIBUS.

1.2.2 Power Requirements

Table 1-1 DWBUA Power Requirements

	POWER (Watts)			
VOLTAGE	**Minimum**	**Typical**	**Standard**	**Maximum**
5.00	31.35	36.50	40.50	56.20
−12.00	<0.012	0.036	0.54	0.6

1.2.3 Current Requirements

Table 1-2 DWBUA Current Requirements

	CURRENT (Amps)			
VOLTAGE	**Minimum**	**Typical**	**Standard**	**Maximum**
5.00	6.6	7.3	8.1	10.7
−12.00	<0.001	0.003	0.045	0.048

1.3 SUPPORTED UNIBUS DEVICES

A subset of the available UNIBUS devices is supported in a configuration with a DWBUA. See Appendix A for details.

CHAPTER 2
INSTALLATION AND TEST

2.1 OPTION COMPONENTS

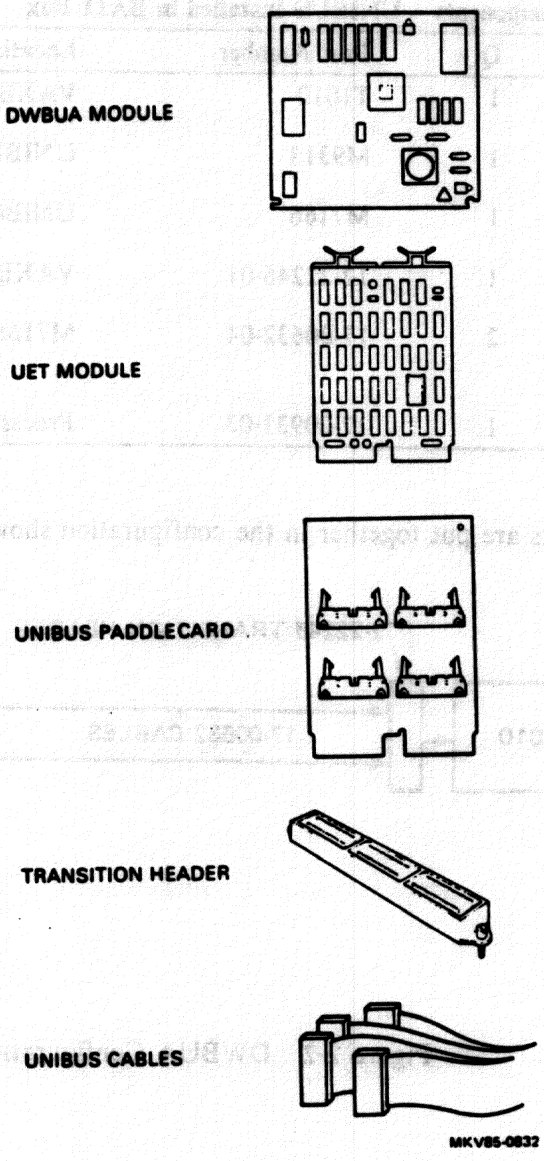


Figure 2-1 DWBUA Components

Table 2-1 DWBUA Components – UNIBUS Installed in BA32-AC/AD Box

Component	Qty	Part Number	Location
DWBUA module	1	T1010	VAXBI cardcage
UET module	1	M9313	UNIBUS cardcage
UNIBUS paddlecard	1	M7166	UNIBUS cardcage
Transition header	1	12-22246-01	VAXBI cardcage
UNIBUS cables	1	17-00631-01	M7166 to transition header

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

Component	Qty	Part Number	Location
DWBUA module	1	T1010	VAXBI cardcage
UET module	1	M9313	UNIBUS cardcage
UNIBUS paddlecard	1	M7166	UNIBUS cardcage
Transition header	1	12-22246-01	VAXBI cardcage
UNIBUS cables	2	17-00632-04	M7166 to transition header
DEC STD 123 power bus cable	1	17-00931-03	Processor cabinet to UNIBUS cabinet

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

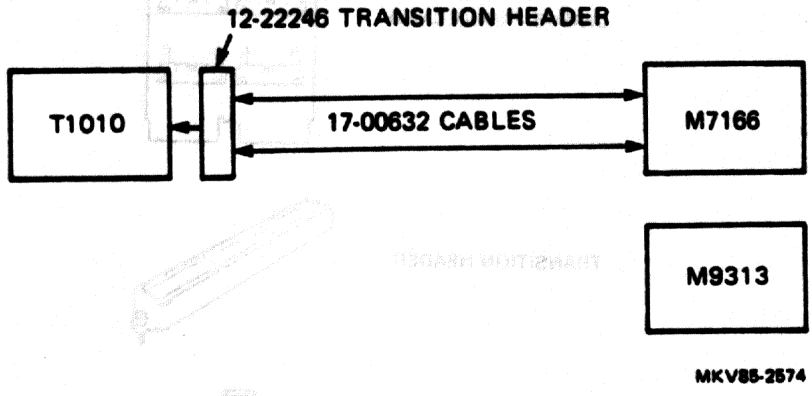


Figure 2-2 DWBUA Configuration

2.2 INSTALLATION

CAUTION

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

WARNING

Shut off system power before proceeding.

1. Attach the four UNIBUS cables to the M7166 paddlecard (Figure 2-3). The connectors are keyed.

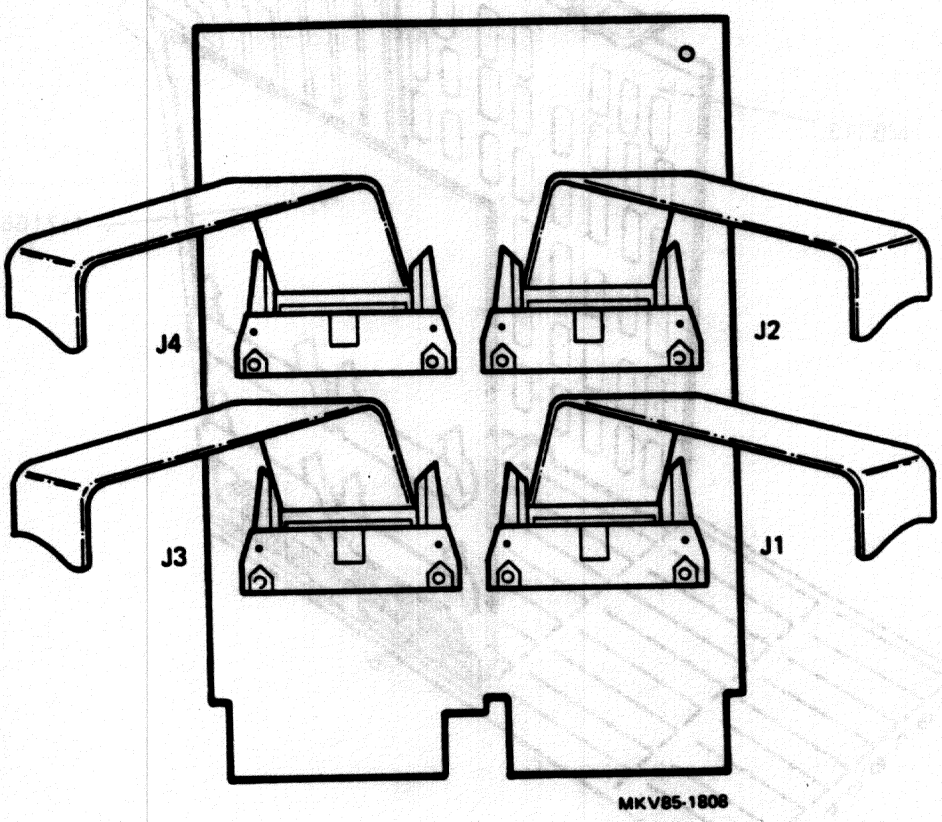


Figure 2-3 M7166 Paddlecard with UNIBUS Cables

2. Insert the M7166 paddlecard into slot 1, segments A and B, of the UNIBUS backplane (Figure 2-4).

3. Insert the M9313 UET module into the last slot, segments A and B, of the UNIBUS backplane (Figure 2-4).

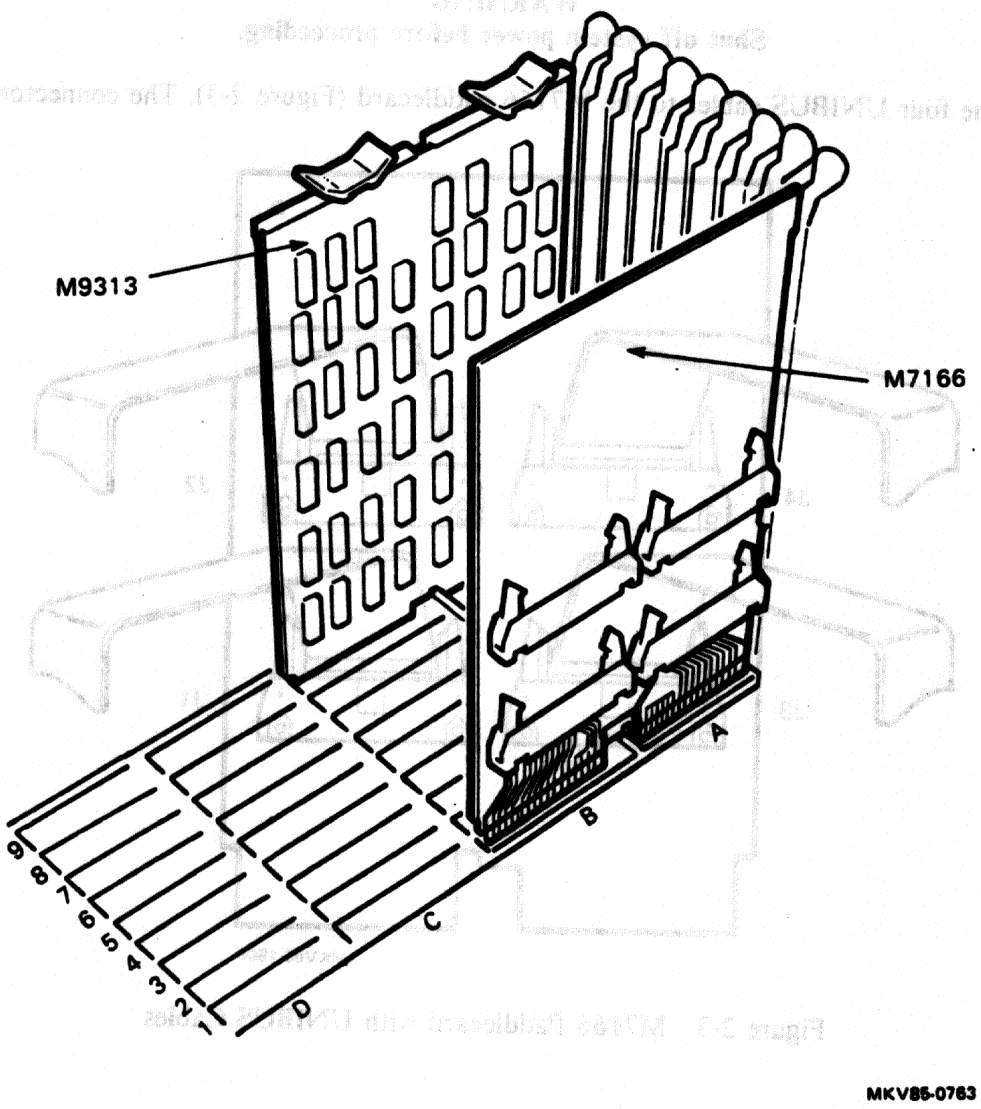


Figure 2-4 UNIBUS Backplane

4. **Insert grant continuity cards in all unused UNIBUS slots.**

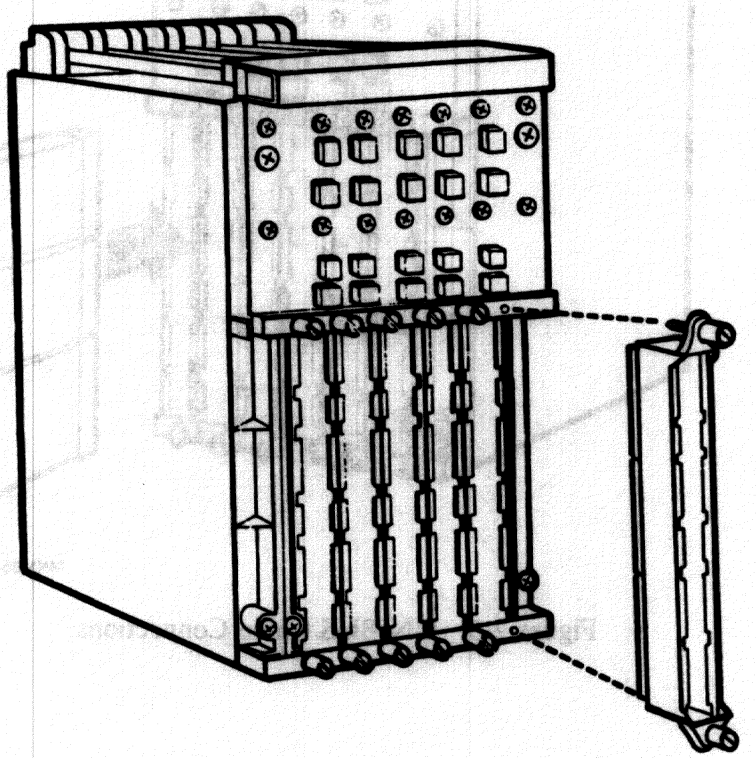
NOTE

For field installations only, the T1010 module may be installed in any empty VAXBI slot (except slot 1). Installation in the next empty slot (after slot 1) is suggested.

5. **Install the transition header on the backplane of the slot that will hold the T1010 module (Figure 2-5).**

NOTE

When installing the transition header, use only the torque screwdriver (P/N 29-17381-00) provided in the Field Service kit.



MKV85-0714

Figure 2-5 VAXBI Transition Header Installation

6. Refer to Figure 2-6 and connect the four UNIBUS cables to the transition header assembly.

J1 – segment E (left)
J2 – segment E (right)
J3 – segment D (left)
J4 – segment D (right)

The connectors are keyed.

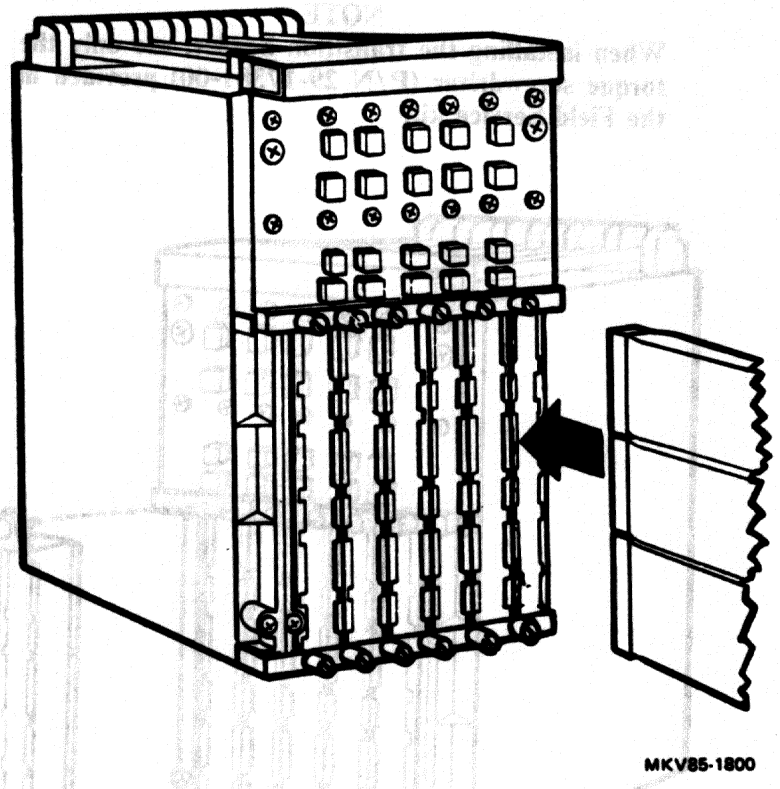


Figure 2-6 UNIBUS Cable Connections

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

8. If the UNIBUS backplane is in an expansion cabinet, the power bus cable (P/N 17-00931-03) may be installed from the processor cabinet to the expansion cabinet.

9. Power up the system. The DWBUA self-test runs upon powerup. 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 two full passes of EVCBB, the DWBUA macrodiagnostic program. See Section 2.3.2.

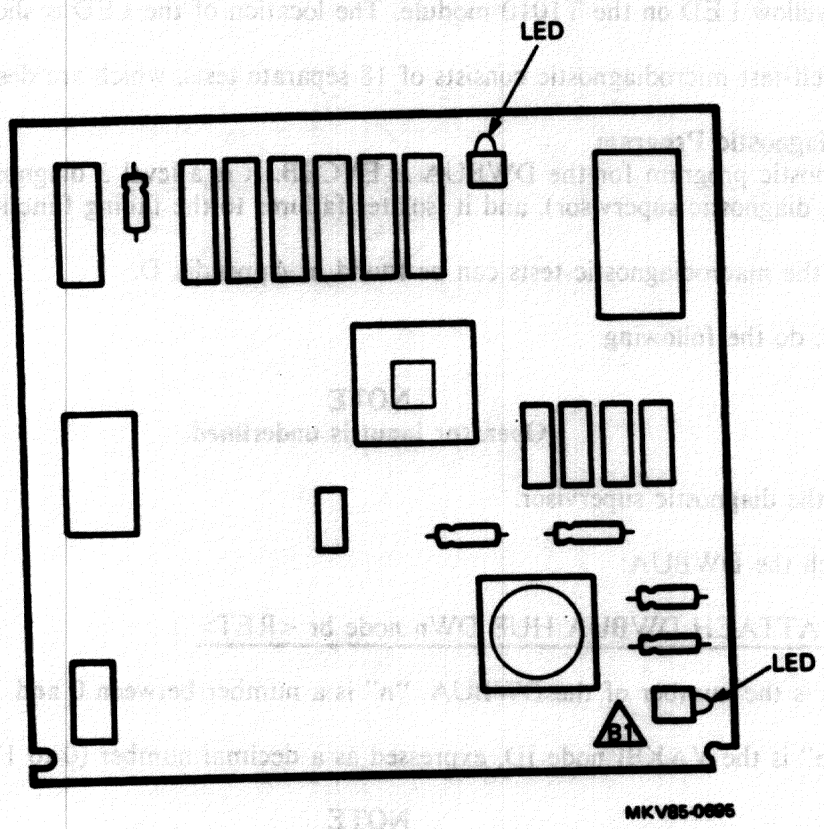


Figure 2-7 T1010 Module

2.3 TEST

2.3.1 Self-Test Microdiagnostic Program

NOTE
A UNIBUS Exerciser Terminator (UET) module (M9313) 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 <10>) 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 18 separate tests, which are described in Appendix C.

2.3.2 Macrodiagnostic Program

The macrodiagnostic program for the DWBUA is EVCBB. It is a level 3 diagnostic (it runs standalone under the VAX diagnostic supervisor), and it isolates failures to the failing function.

Descriptions of the macrodiagnostic 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 HUB DWn node br <RET>

   DWn is the number of the DWBUA. "n" is a number between 0 and 3.

   "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 macrodiagnostic:

   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 Exerciser (UBE) is attached.

Table 2-3 Macrodiagnostic Program Sections

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

2.4 TROUBLESHOOTING

This procedure provides the information needed to isolate a DWBUA failure to one of its assemblies: T1010 module, I/O cable, M7166 paddlecard, UNIBUS, or M9313 UET module. 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 procedures described in this section.

Table 2-4 Tools and Test Equipment for Maintenance Procedures

Equipment	Manufacturer	Designation	DIGITAL Part Number
Gold Wipes	Texwipe	TX809	49-01603-01
Torque Screwdriver	Utica		29-17381-00
Bus Grant Card			G7273

2.4.2 Procedure

This section is a step-by-step procedure for isolating faults to the field replaceable unit (FRU). It uses only the tools and test equipment listed in Table 2-4 and the DWBUA adapter's 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 in the load path for the operating system and diagnostics.)

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

NOTE
Follow the steps in the order listed.

1. **START** – Is the DWBUA malfunctioning?

   The DWBUA may be suspect if:

   a. The system cannot be booted from a UNIBUS device.

   b. No UNIBUS devices can be used.

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

   d. Excessive errors occur when using any UNIBUS 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 DWBUA self-test.

5. **CHECK THE LIGHT ON THE T1010 MODULE** – If the yellow light on the T1010 module is lit, the DWBUA has passed self-test. The problem is most likely not in the T1010 module, the UNIBUS cabling, or the terminator card (UET). 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 DWBUA. Refer to the macrodiagnostic 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 DWBUA self-test passes, the problem is probably somewhere other than in the DWBUA. Refer to Table 2-5 for suggested areas to troubleshoot.

Table 2-5 Symptoms and Possible Causes

Symptom	Possible Cause
Cannot boot from a UNIBUS device	Boot device
Unable to use devices on the UNIBUS	Bad device or software
Excessive errors when using any devices on the UNIBUS	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 those items, look for a UNIBUS device that is causing the UNIBUS 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 (bb+00) for each node space in succession until one with a value of xxxx0102 is found. This is the DWBUA device type. Make a note of the address. (See Figure 3-1 and Appendix G).

   If working on a system that has more than one VAXBI, bus 0 addresses are as described. See Table 2-6 for the base addresses for bus 1 through bus 3.

Table 2-6 Multiple VAXBI Base Addresses

VAXBI Bus #	Base Address
0	2000 0000
1	2200 0000
2	2400 0000
3	2600 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 DWBUA.

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 DWBUA.
>>>E	20002000		This is the base address of node 1.
P	20002000	00010101	Node 1 is not a DWBUA.
.			
.			
.			
>>>E	2000C000		This is the base address of node 6.
P	2000C000	00010102	Node 6 is a DWBUA.

9. **FIND GPR0 ADDRESS** – bb + F0 = GPR0 address. Add F0 (hex) to the address found in the previous step.

Example 2-2: Finding the Address of GPR0

Node 1 GPR0 = 20002000 + F0 = 200020F0
Node 2 GPR0 = 20004000 + F0 = 200040F0
Node 3 GPR0 = 20006000 + F0 = 200060F0
Node 4 GPR0 = 20008000 + F0 = 200080F0
Node 5 GPR0 = 2000A000 + F0 = 2000A0F0
Node 6 GPR0 = 2000C000 + F0 = 2000C0F0

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 microdiagnostic tests.

NOTE

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

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

NOTES

1. **The T1010 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.**

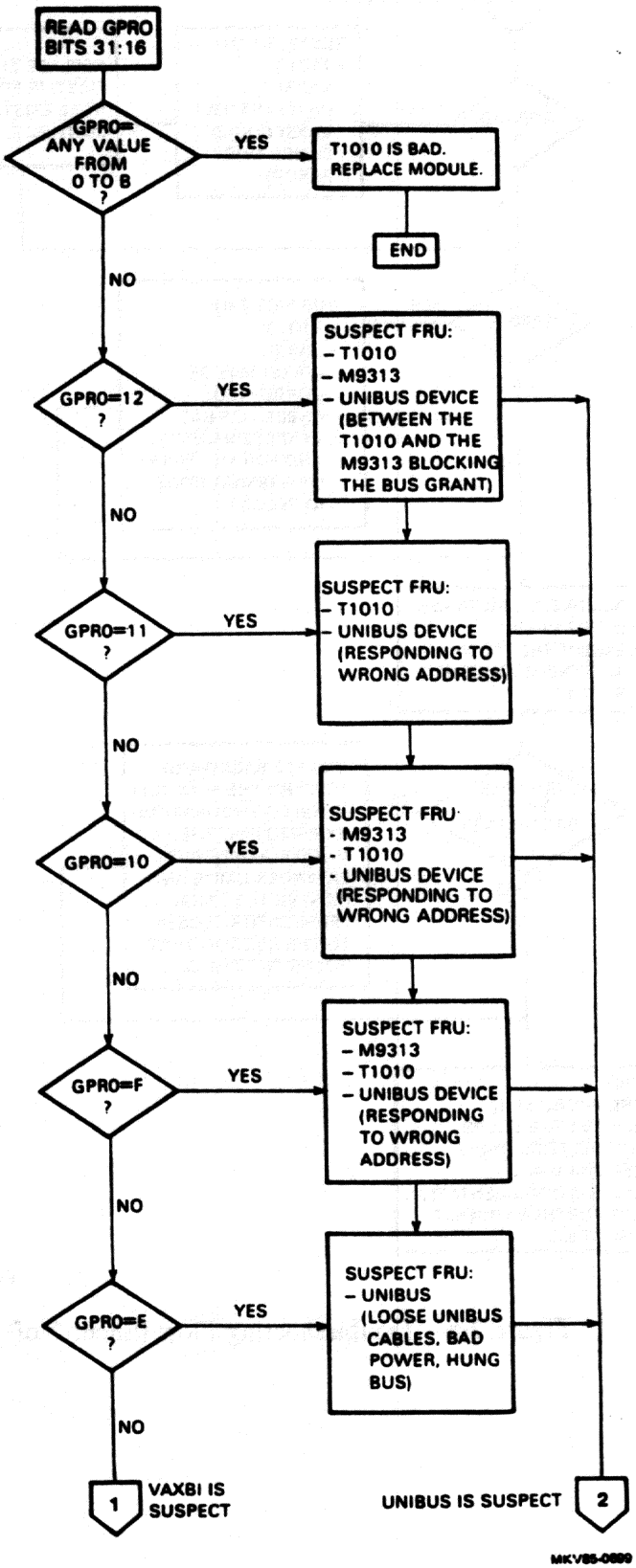


Figure 2-8 Troubleshooting Flow (Sheet 1 of 3)

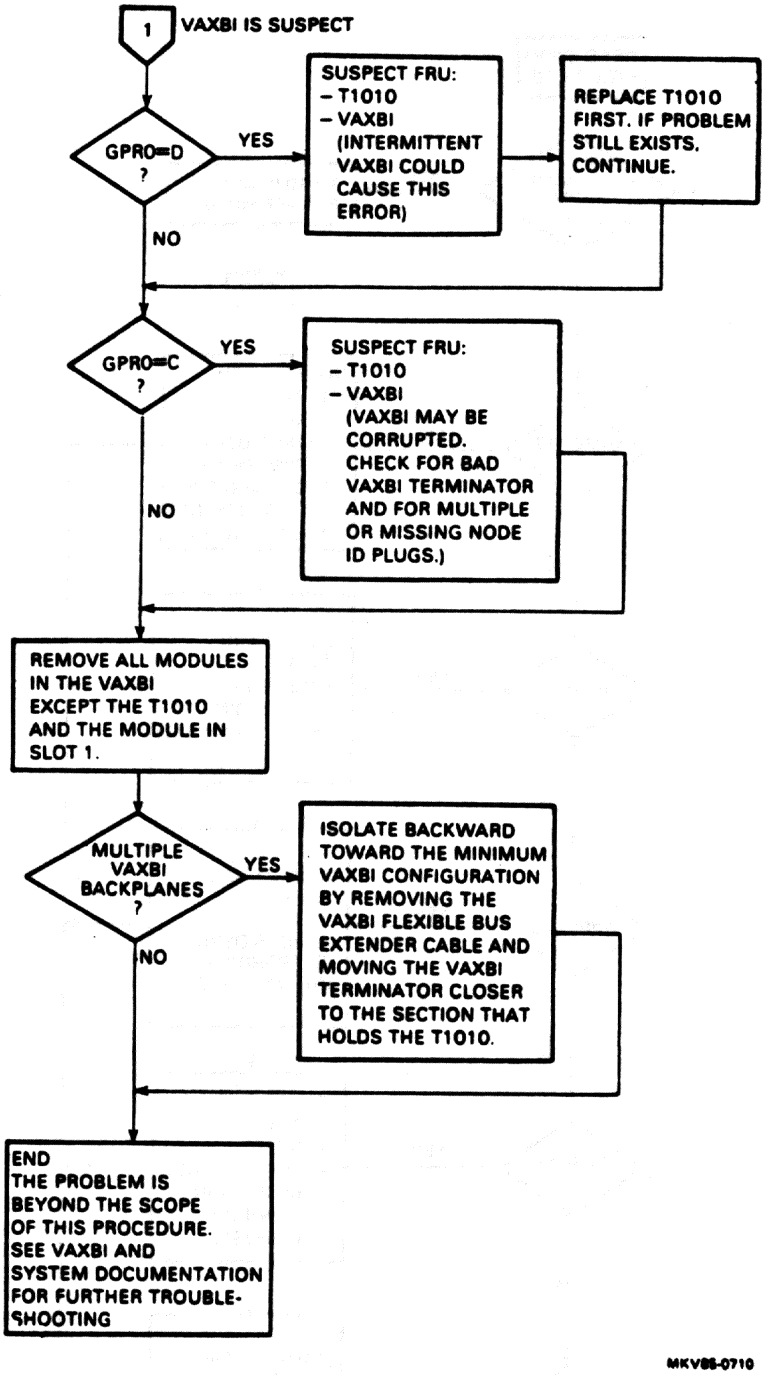


Figure 2-8 Troubleshooting Flow (Sheet 2 of 3)

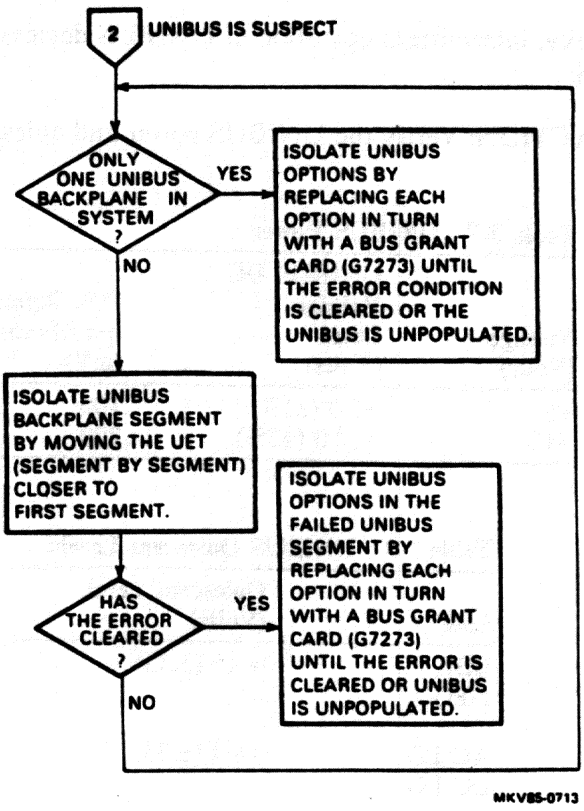


Figure 2-8 Troubleshooting Flow (Sheet 3 of 3)

2.4.3 Helpful Hints

The DWBUA 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 DWBUA self-test may fail if the configuration is large and has extensive VAXBI bus activity.

Try these hints if the self-test has passed but the DWBUA still does not work correctly. Most of the items listed relate to the UNIBUS.

1. **PROBLEM:** SSYN timeout errors on UNIBUS devices.

   SUGGESTED ACTION: Verify VAXBI/DWBUA node ID and arbitration mode.

   The DWBUA must be node 0 (except systems based on BA32-AC/AD boxes), and the software must set the DWBUA to fixed-high priority. Verify this by reading the Device Type Register (bb+00) for node 0 to ensure that the device is a DWBUA (see Appendix H). Also read the VAXBI Control and Status Register (bb+04) for node 0; the contents of <5:0> should be 8 (hex), fixed-high arbitration.

2. **PROBLEM:** Flaky, intermittent operation of UNIBUS devices, involving several or all options on the UNIBUS.

SUGGESTED ACTION: Verify the UNIBUS power and quiescent levels (Tables 2-7 and 2-8).

Table 2-7 UNIBUS Power

Voltage (Volts)	UNIBUS DC Voltage Level (Volts)	P.S. Ripple p-p Maximum (mV)
+5	+5.0 (±5%)	100
+12	+12.0 (±3%)	200

Table 2-8 UNIBUS Quiescent Levels

Line	Quiescent Level (Volts)
BG NPG	+.45 (±.35)
AC LO DC LO	+4.9 (±.35)
BBSY	+3.4 (±.2)
All others	+3.4 (±.2)

SUGGESTED ACTION: Check if the configuration is correct.

a. Verify that the DWBUA is node 0 on the VAXBI (except systems based on BA32-AC/AD boxes).

b. Verify that all NPR grant jumper wires (CA1 to CB1) have been removed from the UNIBUS backplane on every slot that has an NPR option.

c. Check that every empty UNIBUS slot contains a grant continuity card.

Two different grant continuity cards can be used. The first, G727A, goes into the UNIBUS backplane slot D and provides grant continuity for the four interrupts (BR4 – BR7) but not for the NPR. When the G727A grant card is used in the empty slots, a jumper (CA1 to CB1) is needed for NPR grants. The second grant card, G7273, 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 PAULI program to verify the configuration.

SUGGESTED ACTION: Verify that the configuration is supported.

Check that no untested devices or unsupported devices are on the bus.

SUGGESTED ACTION: Look for bus loading problems on large UNIBUS configurations.

Calculate the ac and dc loading of the configuration. Most modules represent 1 dc load and 1 ac load. A UNIBUS (without a repeater) can support 20 dc loads and 20 ac loads. (Refer to the *PDP-11 Bus Handbook* for details.)

3. **PROBLEM:** One option does not work or the entire UNIBUS fails when the option is installed.

   SUGGESTED ACTION: Verify that the option is supported on this UNIBUS.

   a. Check the RM document to ensure that the revisions are correct for the hardware and software.

   b. If the option works but does not work correctly on the full system, run the option on a shortened UNIBUS.

   c. Check that the jumper wire from CA1 to CB1 has been removed from the UNIBUS slot in which this option is being installed.

Part II
Technical Description

CHAPTER 3
PROGRAMMING

3.1 SYSTEM ADDRESS SPACE

3.1.1 Address Space Distribution

The 1024 megabyte system address space on the VAXBI is divided into memory space (from address 0000 0000 through 1FFF FFFF hexadecimal) and I/O space (from address 2000 0000 through 3FFF FFFF 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 distribution.

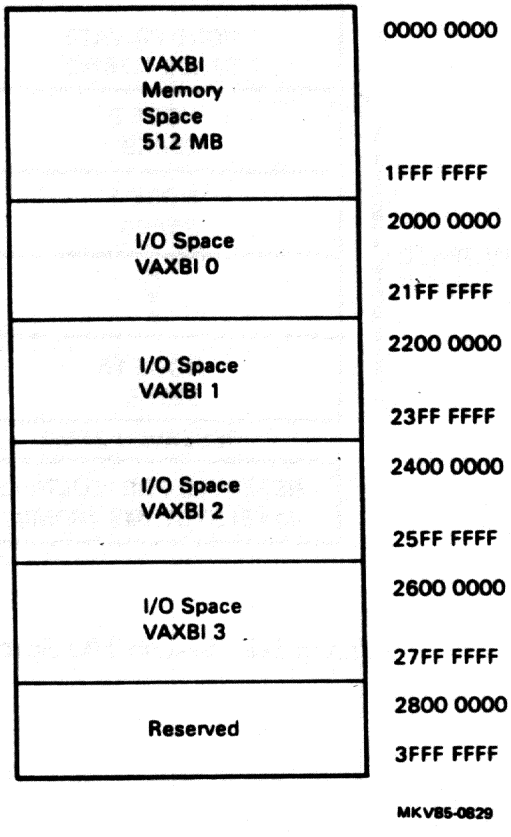


Figure 3-1 System Address Space Distribution

3.1.2 System I/O Space

The 512 megabyte system I/O space is divided into several dedicated sections, as shown in Figure 3-2.

The locations of the DWBUA adapter's node space and window space depend on the DWBUA adapter's assigned node ID. If, for example, the DWBUA is assigned node 1, its node space and window space are located as shown in Figure 3-3. Appendix G lists the starting and ending addresses of the node space and window space for each node ID.

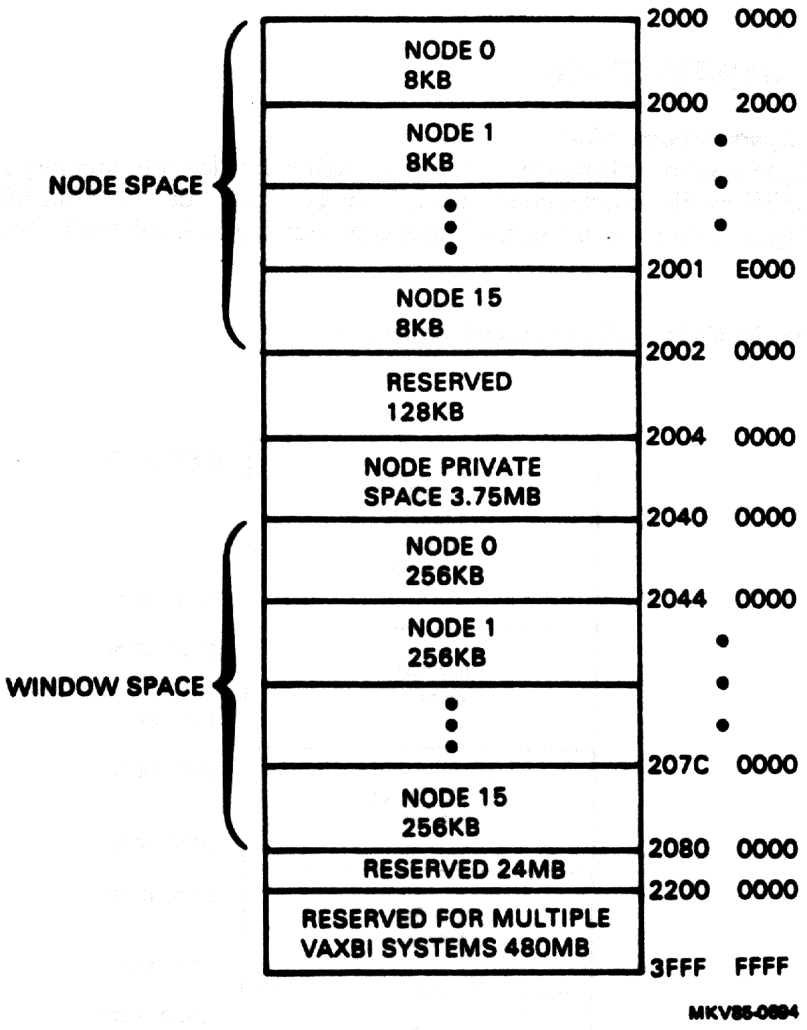


Figure 3-2 System I/O Space

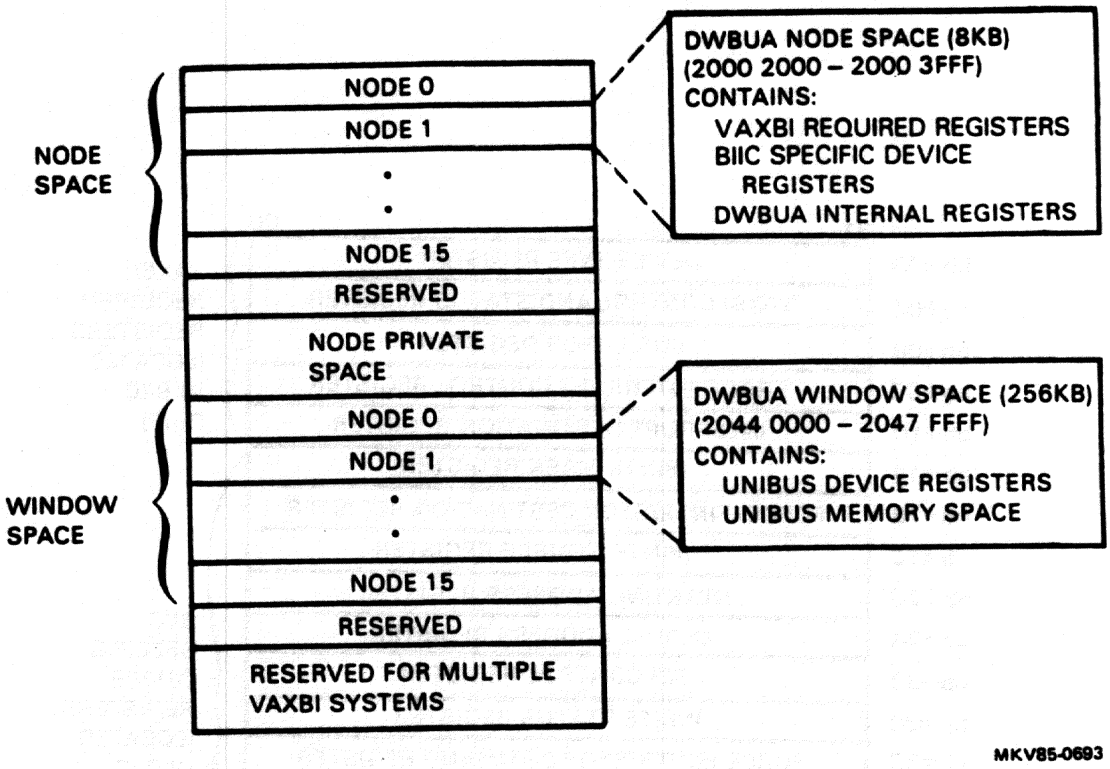


Figure 3-3 DWBUA Node Space and Window Space

3.2 DWBUA ADDRESS SPACE

The DWBUA adapter's node space is divided into three sets of registers: VAXBI required registers, BIIC specific device registers, and DWBUA registers. Figure 3-4 is a map of the DWBUA address space.

NOTE

The address of each register is noted as "bb + xxx." Use the following procedure to calculate the base address, "bb."

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

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

   $$bb = (2000\ 0000_{16}) + ([2000_{16}] \times [\text{node ID}_{16}])$$

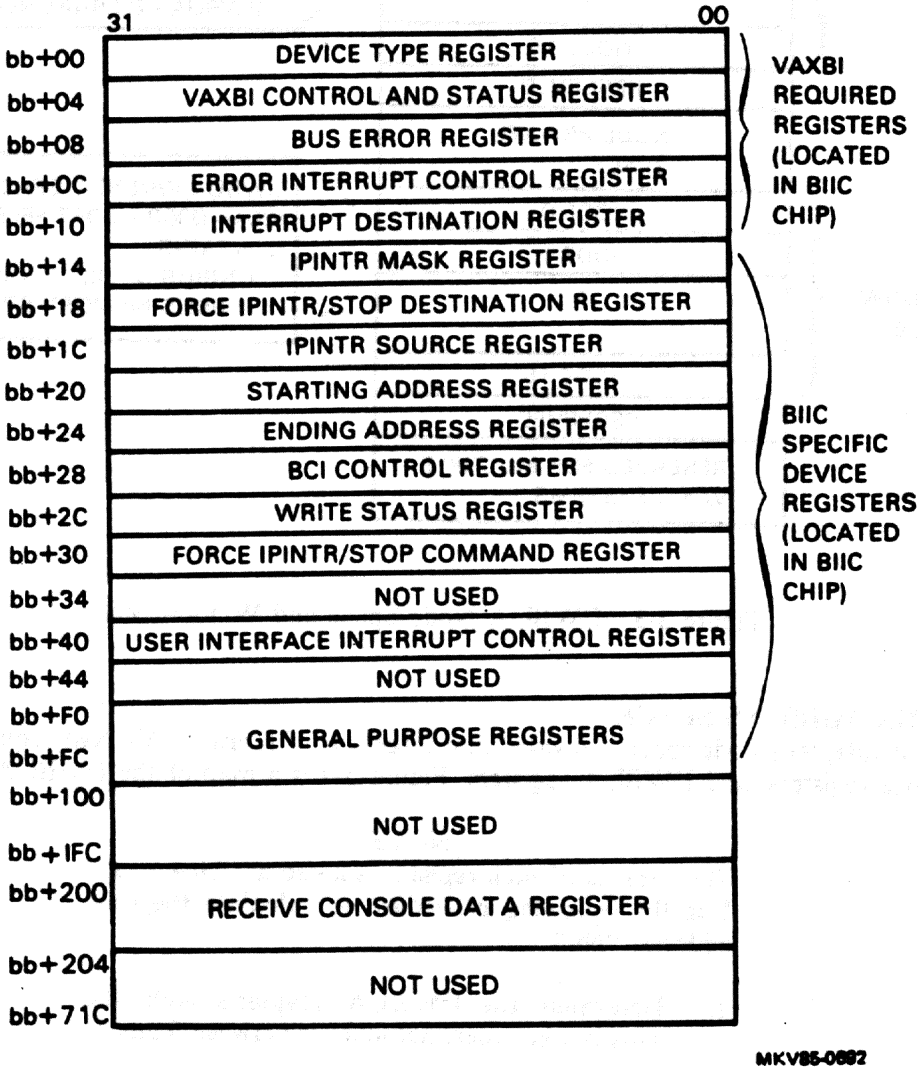


Figure 3-4 DWBUA Address Space (Sheet 1 of 2)

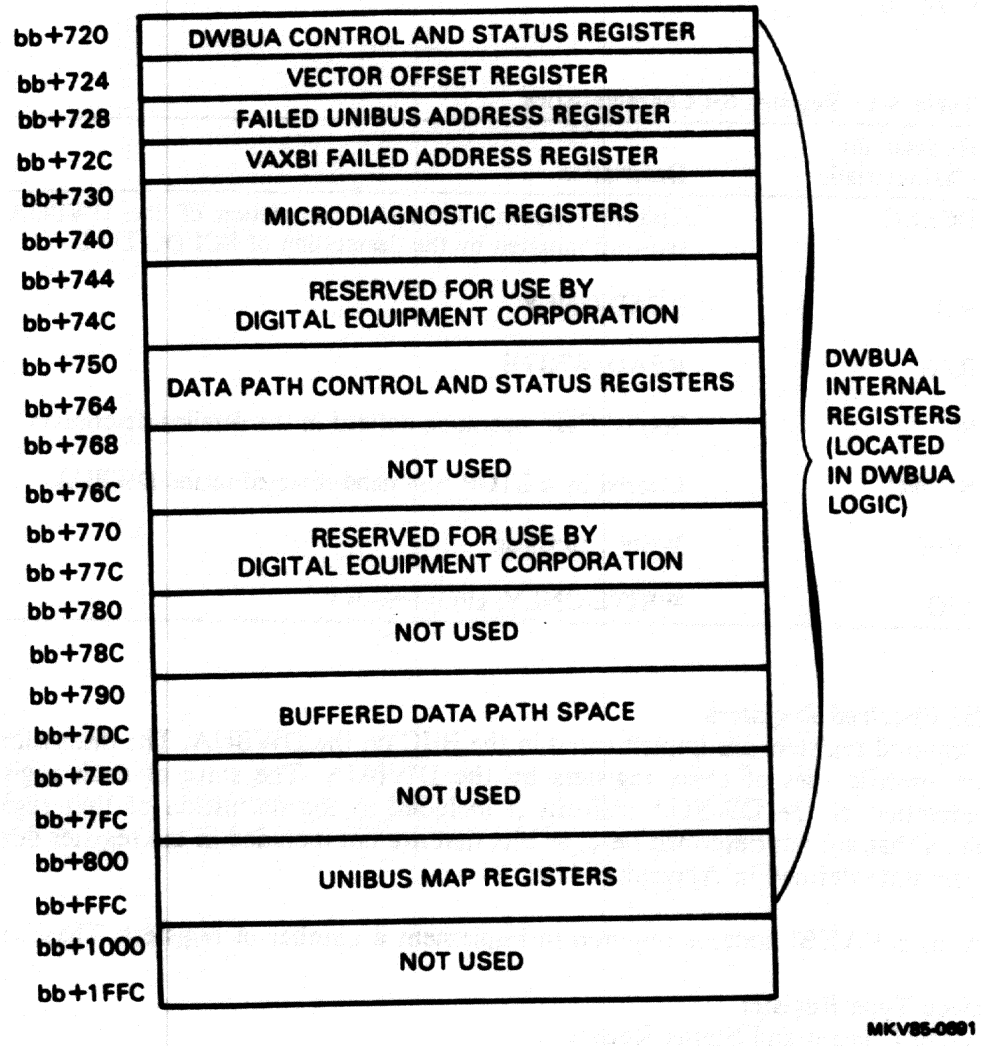


Figure 3-4 DWBUA Address Space (Sheet 2 of 2)

3.2.1 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" in the register diagrams are not implemented. These bits are READ-ONLY locations that always return "0".

Table 3-1 Register Bit Characteristics

Register Bit Characteristic	Description
DCLOC	Cleared following successful completion of the DWBUA self-test; initiated by the deassertion of BCI DC LO L
RO	READ-ONLY
R/W	READ/WRITE
SC	Special Case; operation defined in the detailed description
STOPC	Cleared by a STOP command directed to the DWBUA
W1C	Write 1 to Clear
WO	WRITE-ONLY; always reads 0

3.2.2 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 examined here in detail.

3.2.2.1 Error Interrupt Control Register – The Error Interrupt Control Register (bb+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 3-5 is an illustration of the Error Interrupt Control Register.

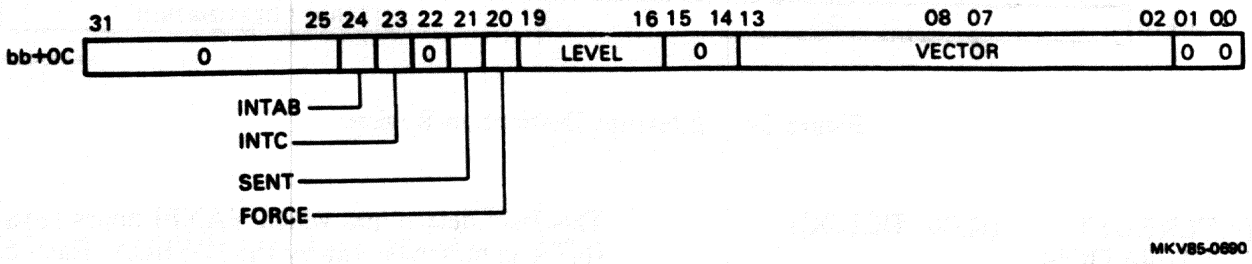


Figure 3-5 Error Interrupt Control Register

INTAB <24>	Interrupt abort (W1C, DCLOC, SC)	This bit is set if a VAXBI INTR command sent by the DWBUA is aborted.
INTC <23>	Interrupt complete (W1C, DCLOC, 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.
SENT <21>	Sent (W1C, DCLOC, STOPC, SC)	The DWBUA has sent the VAXBI INTR command, and it is waiting for IDENT from the VAXBI.
FORCE <20>	Force (R/W, DCLOC)	When this bit is set, the DWBUA forces an interrupt to occur regardless of the state of the Bus Error Register (bb+08). The DWBUA sets the FORCE bit when a DWBUA error has occurred and the DWBUA error interrupt enable (BUAEIE) 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, DCLOC)	The LEVEL field determines the level(s) at which INTR commands are transmitted under the control of this register. Bit <16> corresponds to interrupt level 4, bit <17> to level 5, bit <18> to level 6, and bit <19> to level 7. The operating system must initialize the LEVEL field.
VECTOR <13:02>	(R/W, DCLOC)	The VECTOR field contains the vector used during error interrupt sequences. It is transmitted when the DWBUA wins a VAXBI IDENT ARB 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.

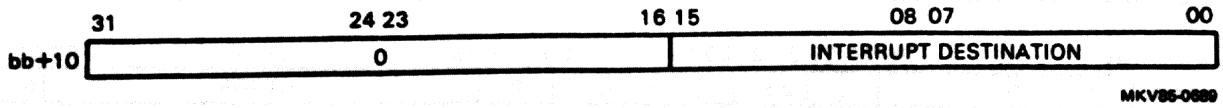


Figure 3-6 Interrupt Destination Register

INTERRUPT DESTINATION <15:00> (R/W, DCLOC)

This field determines which VAXBI nodes receive INTR commands sent by the DWBUA. 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 decoded master's ID (VAXBI node number) is compared to the corresponding bit in the INTERRUPT DESTINATION field. The DWBUA 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 DWBUA self-test sets the bit in the INTERRUPT DESTINATION field which corresponds to the DWBUA 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 INTAB bit in one of two registers is set. The register in which the INTAB bit is set depends on the type of interrupt: interrupt – User Interface Interrupt Control Register (bb+40); error interrupt – Error Interrupt Control Register (bb+0C).

3.2.3 BIIC Specific Device Registers

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

- IPINTR Mask Register
- Force IPINTR/STOP Destination Register
- IPINTR Source Register
- Starting Address Register*
- Ending Address Register*
- BCI Control Register*
- Write Status Register
- Force IPINTR/STOP Command Register
- User 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 (bb+20) defines the lower limit of the DWBUA adapter's window space. Figure 3-7 is the Starting Address Register format.

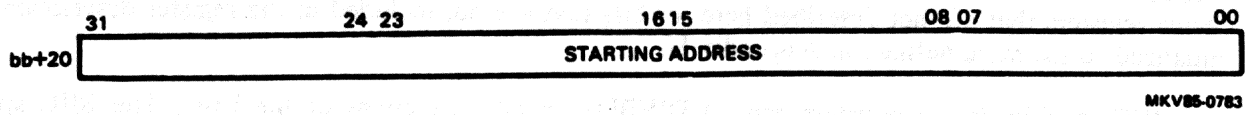


Figure 3-7 Starting Address Register

STARTING ADDRESS <31:00>

This field determines bits <29:18> of the lowest UNIBUS address. The DWBUA self-test leaves in this register the lower limit of the DWBUA adapter's window space, based on the node ID of the DWBUA. The range is 2040 0000 to 207C 0000 (bits <17:0> must be zero).

3.2.3.2 Ending Address Register – The Ending Address Register (bb+24) defines the first location after the upper limit of the DWBUA adapter's window space. Figure 3-8 is the Ending Address Register format.

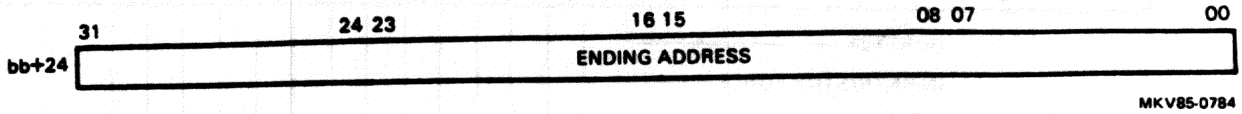


Figure 3-8 Ending Address Register

ENDING ADDRESS <31:00>

This field defines bits <29:18> of the highest UNIBUS address. The DWBUA self-test leaves in this register the (upper limit + 1) of the DWBUA adapter's window space, based on the node ID of the DWBUA. The range is 2044 0000 to 2080 0000 (bits <17:0> must be zero).

3.2.3.3 BCI Control Register – The BCI Control Register (bb+28) format is shown in Figure 3-9.

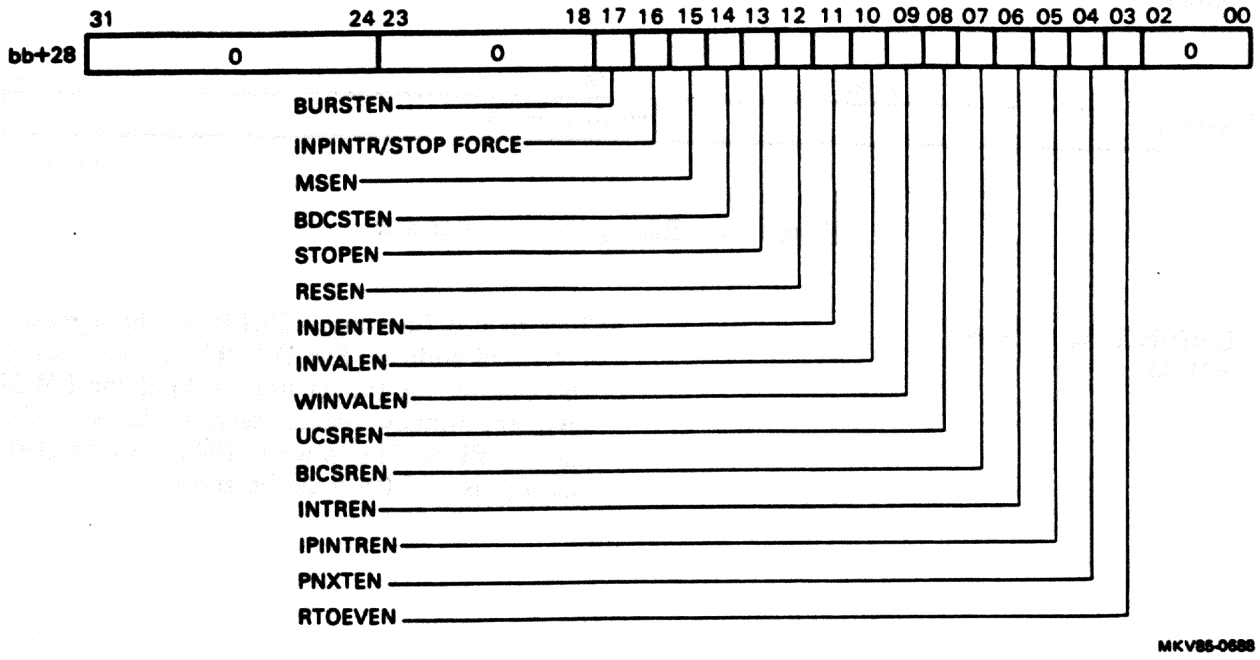


Figure 3-9 BCI Control Register

STOPEN <13>	STOP Enable (R/W, DCLOC)	When set, this bit enables the DWBUA to respond to a VAXBI STOP command directed to it. The DWBUA adapter's BIIC asserts BCI SEL L and the appropriate BCI SC <2:0> code.
IDENTEN <11>	IDENT Enable (R/W, DCLOC)	When set, this bit enables the DWBUA to acquire interrupt vectors from UNIBUS devices when a processor issues an IDENT. The DWBUA adapter's BIIC asserts BCI SEL L and the appropriate BCI SC <2:0> code. This bit affects only the output of SEL and the IDENT SC code.
UCSREN <08>	User CSR Space Enable. (R/W, DCLOC)	When this bit is set, the DWBUA can respond to a VAXBI READ or WRITE command with an address in DWBUA CSR space. The DWBUA adapter's BIIC asserts BCI SEL L and the appropriate BCI SC <2:0> code.

NOTE

The DWBUA sets the above three bits upon successful completion of its self-test. All other bits in this register should be clear.

3.2.3.4 User Interface Interrupt Control Register – Figure 3-10 is the User Interface Interrupt Control Register (bb+40) format.

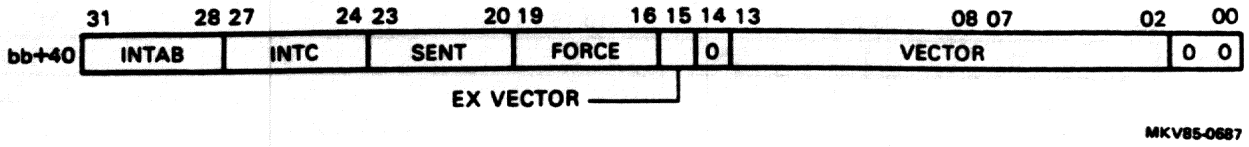


Figure 3-10 User Interface Interrupt Control Register

FORCE <19:16>	**Force**	This field must be zero.
EX VECTOR <15>	**External vector**	This bit is set by the DWBUA self-test, and it must remain set. It enables the DWBUA to use the external vector for transfer of UNIBUS interrupt vectors which have the concatenated vector offset applied.

3.2.3.5 General Purpose Registers – The only General Purpose Register (GPR) used by the DWBUA is GPR0 (bb+F0). Figure 3-11 shows the format of GPR0.

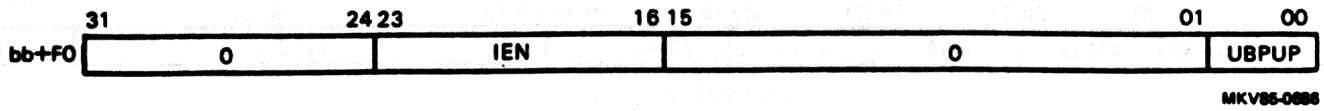


Figure 3-11 General Purpose Register 0

IEN <23:16>	Internal Error Number (RO)	This field is a copy of the IEN field of the BUACSR (bb+720). This field contains the self-test error number if the DWBUA self-test fails anc if the DWBUA functions sufficiently to write to this register. (See Appendix C.) This field is clear if the DWBUA self-test passes.
UBPUP <00>	UNIBUS Power Up (RO)	This bit is set when the UNIBUS power is ON. It is cleared by the DWBUA when UNIBUS power goes down. This bit is set upon successful completion of the DWBUA self-test. (The DWBUA fails self-test if the UNIBUS is not powered up.)

General Purpose Registers 1-3 are not used by the DWBUA. They are cleared by the BIIC self-test.

3.2.4 DWBUA Internal Registers

With the exception of the Data Path Control and Status Registers and the upper 16 UNIBUS Map Registers, all DWBUA internal registers are cleared by the DWBUA self-test.

3.2.4.1 Receive Console Data Register – The Receive Console Data Register (RXCD) is considered an unimplemented register by the DWBUA. The DWBUA responds with NO ACK to any access to this register. Any VAXBI node that might access this register should have a way to detect this timeout. Figure 3-12 shows the format of the Receive Console Data Register.

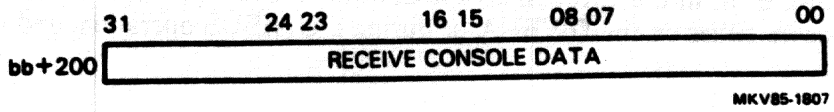


Figure 3-12 Receive Console Data Register

3.2.4.2 DWBUA Control and Status Register – The DWBUA Control and Status Register (BUACSR; bb+720) contains error and other operating information about the DWBUA.. Figure 3-13 shows the format of the DWBUA Control and Status Register.

When an error occurs during DWBUA 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 DWBUA if an error occurs either on the DWBUA or during a UNIBUS operation, and if error interrupts are enabled.

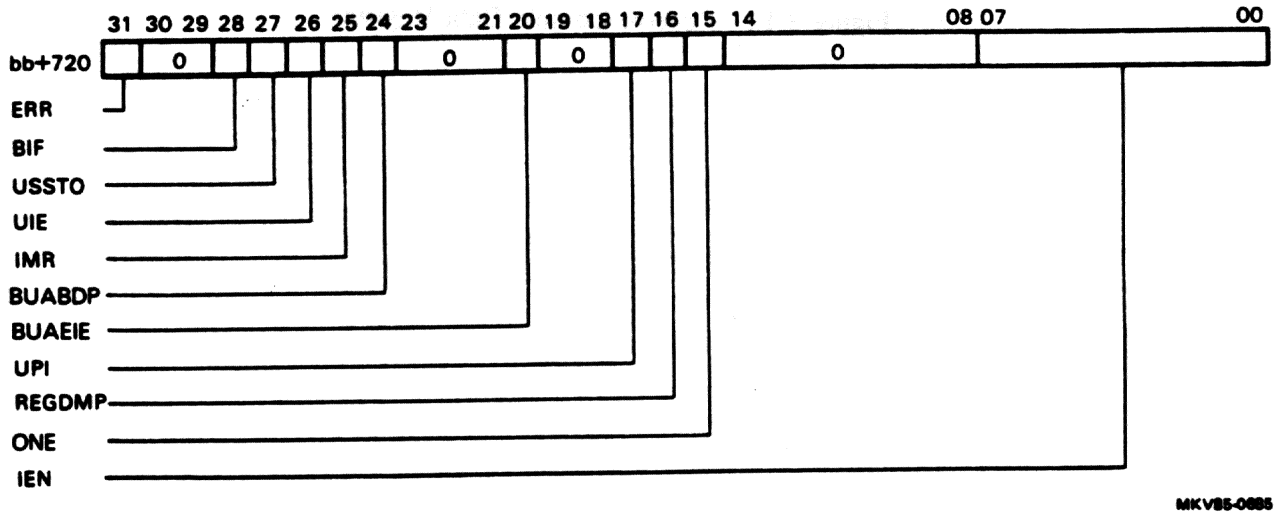


Figure 3-13 DWBUA Control and Status Register

ERR <31>	Error (R0, DCLOC)	This bit is a logical "OR" of all error bits in the BUACSR.
BIF <28>	VAXBI Failure (W1C, DCLOC)	This bit is set if a DWBUA- initiated VAXBI transaction fails. A VAXBI failure has occurred if the DWBUA receives any of the following in response to one of its VAXBI commands: – NO ACK – Illegal confirmation code – Read data substitute status code See Table E-1 for a list of BIIC EVENT codes that cause the BIF bit to set.

USSTO <27>	**UNIBUS SSYN Timeout (W1C, DCLOC)**	This bit is set when a VAXBI-to- UNIBUS command attempts access to a UNIBUS address and does not receive SSYN within 19.2 μs after assertion of MSYN.
UIE <26>	**UNIBUS Interlock Error (W1C, DCLOC)**	This bit is set if a UNIBUS DATIP command is not immediately followed by a DATO(B) command. This happens when BBSY is dropped by a device after the DATIP command.
IMR <25>	**Invalid Map Register (W1C, DCLOC)**	This bit is set if a UNIBUS Map Register (bb+800 - bb+FFC) which has its VALID bit clear is accessed during a UNIBUS-to-VAXBI transaction.
BADBDP <24>	**Bad Buffered Data Path Selected (W1C, DCLOC)**	This bit is set if (nonexistent) Buffered Data Path 6 or 7 is selected.
BUAEIE <20>	**DWBUA Error Interrupt Enable (R/W, DCLOC)**	If an error occurs, the DWBUA initiates an error interrupt on the VAXBI if this bit is set.
UPI <17>	**UNIBUS Power Initialization (WO)**	Writing a one to this bit causes a power-up initialization on the UNIBUS.
REGDMP <16>	**Microdiagnostic Register Dump (WO)**	Writing a one to this bit causes the microcode control to dump its internal registers to the Microdiagnostic Registers (bb+730 - bb+740). A READ 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.
IEN <07:00>	**Internal Error Number (RO)**	This field contains the self-test error number if the DWBUA self- test fails and if the DWBUA functions sufficiently to write to this register. This field is clear if the DWBUA self-test passes.

3.2.4.3 Vector Offset Register – The Vector Offset Register (VOR; bb+724) contains a 5-bit field which is concatenated with the incoming UNIBUS vector to form the new VAXBI vector.

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

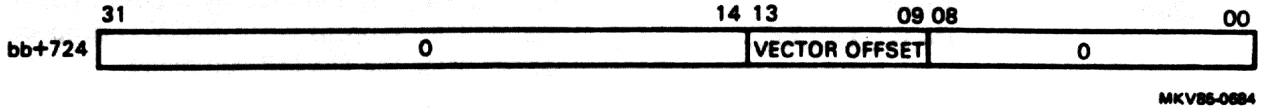


Figure 3-14 Vector Offset Register

VECTOR OFFSET <13:09>	(R/W)	The five bits in this field are concatenated with the incoming UNIBUS vector (UNIBUS bits <08:02>, in the range of 000 to 774) to form the new 14-bit VAXBI vector <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 UNIBUS Address Register – When a VAXBI-to-UNIBUS transaction results in a SSYN timeout, the failed UNIBUS Address Register (FUBAR; bb+728) holds the failed UNIBUS address sent by the VAXBI master. UNIBUS 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 USSTO bit of the BUACSR is cleared.

Figure 3-15 shows the format of the Failed UNIBUS Address Register.

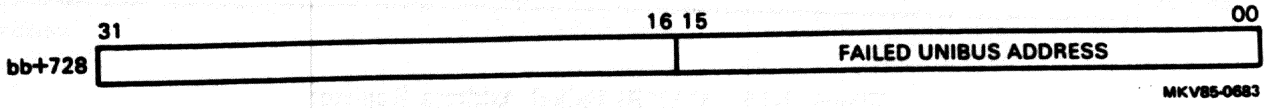


Figure 3-15 Failed UNIBUS Address Register

FAILED UNIBUS ADDRESS <15:00>	(RO)	This field contains bits <17:02> of the first failed UNIBUS address. Subsequent failures are not recorded until the USSTO bit of the BUACSR is cleared.

3.2.4.5 VAXBI Failed Address Register – The VAXBI Failed Address Register (BIFAR; bb+72C) 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 BUACSR 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.

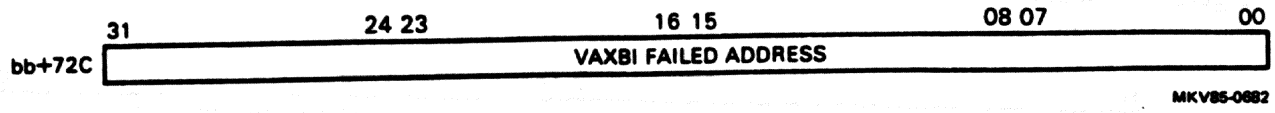


Figure 3-16 VAXBI Failed Address Register

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.

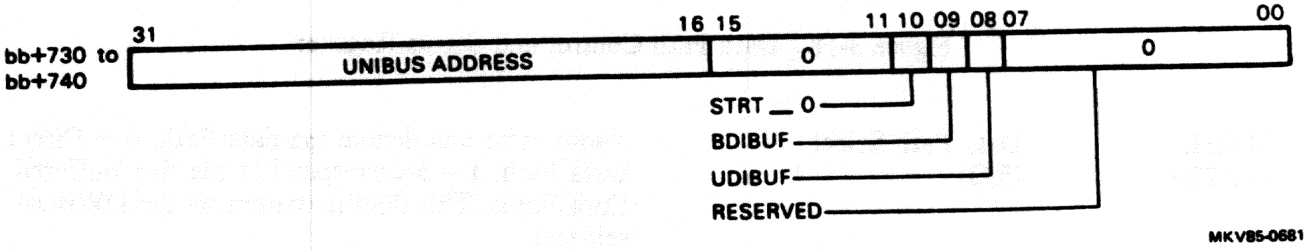


Figure 3-17 Microdiagnostic Register

UNIBUS ADDRESS <31:18> (RO)

This field holds UNIBUS address bits <17:04> of the current octaword transfer through the specific Buffered Data Path.

STRT_0 <10> (RO)

When set, this bit indicates that the first transaction through the Buffered Data Path began at an aligned octaword address.

BDIBUF <09> (RO)

This bit is set to indicate that the BDP buffer contains VAXBI data.

UDIBUF <08> (RO)

This bit is set to indicate that the BDP buffer contains UNIBUS data.

3.2.4.7 Data Path Control and Status Registers – The DWBUA has six Data Path Control and Status Registers (DPCSR; bb+750 – bb+764). 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

DPCSRx	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.

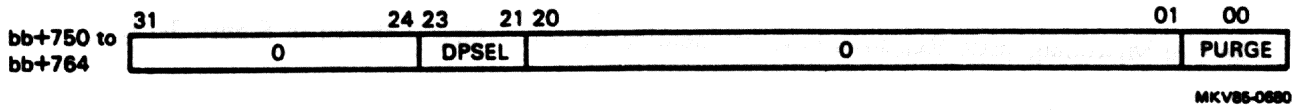


Figure 3-18 Data Path Control and Status Register

DPSEL <23:21>	Data Path Select (RO)	These three bits denote the data Path: 0 = Direct Data Path; 1 – 5 correspond to the five Buffered Data Paths. This field is written by the DWBUA self-test.
PURGE <00>	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:

UNIBUS 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 octaword buffer associated with each Buffered Data Path is addressable in DWBUA I/O space (bb+790 – bb+7D0). Although the BDP buffers are not usually accessed directly through software, they are READ-ONLY and are longword accessible for diagnostic purposes.

3.2.4.9 UNIBUS Map Registers – The 512 UNIBUS Map Registers (bb+800 – bb+FFC) are READ/WRITE accessible to the operating system. These registers are invalidated by writing zeros to their VALID bits or by BCIDCLO.

A UNIBUS Map Register translates an 18-bit UNIBUS address to a 30-bit VAXBI address. This translation is illustrated in Figure 3-19.

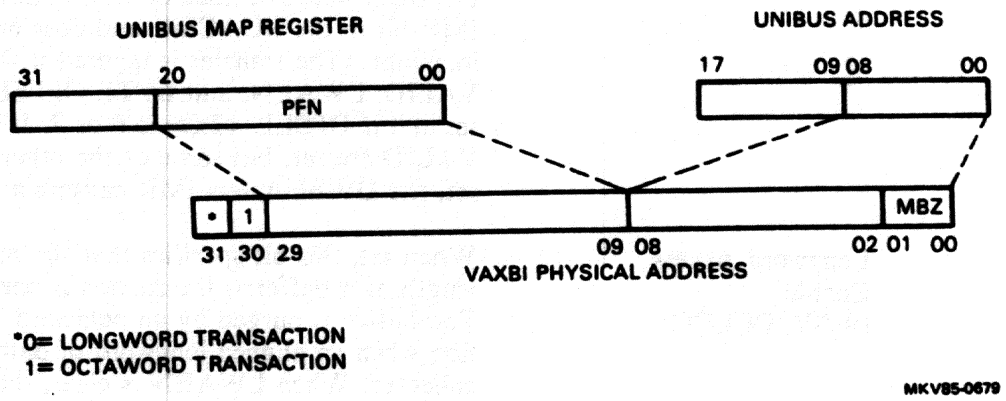


Figure 3-19 UNIBUS-to-VAXBI Address Translation

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

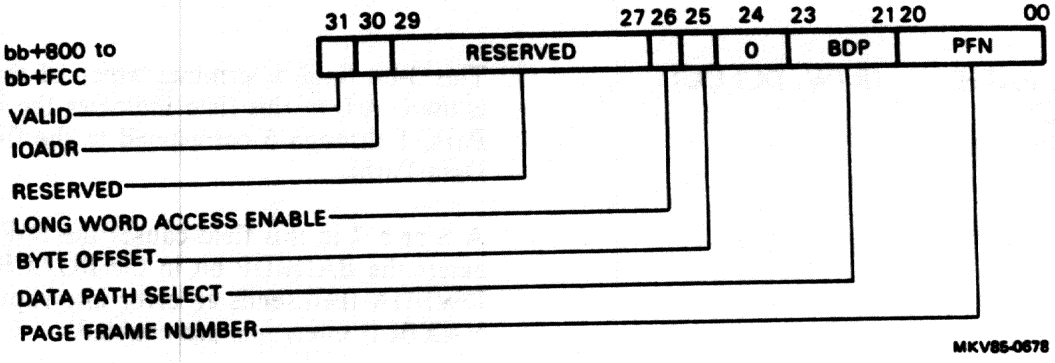


Figure 3-20 UNIBUS Map Register

VALID <31>	(R/W, DCLOC)	Clearing this bit prevents a UNIBUS transfer from mapping to the VAXBI. A transaction that uses a UNIBUS Map Register with a clear VALID bit does not receive SSYN. When this happens, the IMR bit in the BUACSR 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 initiates a transfer to a UNIBUS Map Register with contents FFFFFFFF (hex), the DWBUA ignores the transfer. That is, the DWBUA does not issue SSYN, does not set the IMR bit in the BUACSR, and does not issue an interrupt. (The transfer is ignored if the IOADR, VALID, LWAEN, and BYTE OFFSET bits are set and if DPSEL <2:0> is 6 or 7. If IOADR and VALID are set, but some of the other bits are not set, the DWBUA sets IMR causing an interrupt.)
LWAEN <26>	Longword Access Enable (R/W, DCLOC)	When set, this bit specifies that the maximum length of a buffered transaction is one longword. The buffer is purged by an octaword WMCI operation when an aligned longword of data has been collected. When LWAEN is clear, the buffered transaction depth may be as long as one octaword before the contents are sent to the VAXBI. This bit is ignored when set in a UNIBUS Map Register with the Direct Data Path selected.
BYTE OFFSET <25>	(R/W, DCLOC)	When this bit is set, the UNIBUS address is treated as if it is incremented by one.
DATA PATH SELECT <23:21>	(R/W, DCLOC)	This 3-bit field determines which of the data paths is used. A 0 in this field indicates the Direct Data Path; 1 through 5 correspond to the five Buffered Data Paths. A 6 or a 7 in this field causes the DWBUA to assert the BADBDP bit in the BUACSR. The DWBUA then sends an error interrupt to the VAXBI if interrupts are enabled.
PAGE FRAME NUMBER <20:00>	(R/W, DCLOC)	This 21-bit address field is concatenated with UNIBUS address bits <8:0> to form a 30-bit physical address on the VAXBI.

3.3 INITIALIZATION

3.3.1 DWBUA Hardware Initialization

Upon successful completion of the self-test, all DWBUA internal registers and Buffered Data Path flags 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 DWBUA monitors the UNIBUS AC LO signal. When this signal is asserted, the DWBUA initializes the UNIBUS (but not the DWBUA), clears the UBPUP bit in GPRO, and, if interrupts are enabled, issues an error interrupt. When UNIBUS AC LO is deasserted and UNIBUS initialization is complete, another interrupt is sent if interrupts are enabled.

2. The DWBUA initializes the UNIBUS (but not the DWBUA) if a processor sets the UPI bit in the BUACSR. Two interrupts are issued if interrupts are enabled.

3. The DWBUA asserts UNIBUS AC LO whenever BI AC LO L is asserted; therefore, a brown-out or black-out that affects the VAXBI causes the UNIBUS to be initialized.

4. The DWBUA asserts UNIBUS AC LO when a processor sets the SST bit in the BICSR. This mechanism for initializing the DWBUA also initializes the UNIBUS.

The state diagram for UNIBUS initialization, Figure 3-21, applies to all of the cases above.

During UNIBUS initialization, which takes at least 80 ms, the UBPUP bit in GPRO (bb+F0) is cleared by the DWBUA indicating that UNIBUS power is down. The DWBUA sends an error interrupt to the VAXBI if interrupts are enabled.

During UNIBUS initialization, the DWBUA internal registers are not accessible. The DWBUA sends an ACK response to all WRITE commands from the VAXBI and ignores the data; all 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 on the UNIBUS, UBPUP is set in GPRO and the VAXBI is interrupted if interrupts are enabled.

3.4 PROGRAMMING CONSIDERATIONS

3.4.1 UNIBUS Map Registers

The DWBUA register set includes 512 UNIBUS Map Registers. When its VALID bit is set, 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 CSR addresses of devices on the UNIBUS. To do this, leave the contents of the upper 16 UNIBUS Map Registers unchanged after DWBUA initialization.

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

The lower 496 UNIBUS Map Registers are initialized to zero (also known as invalidated – that is, their VALID bits are cleared) by the DWBUA on receipt of BCIDCLO.

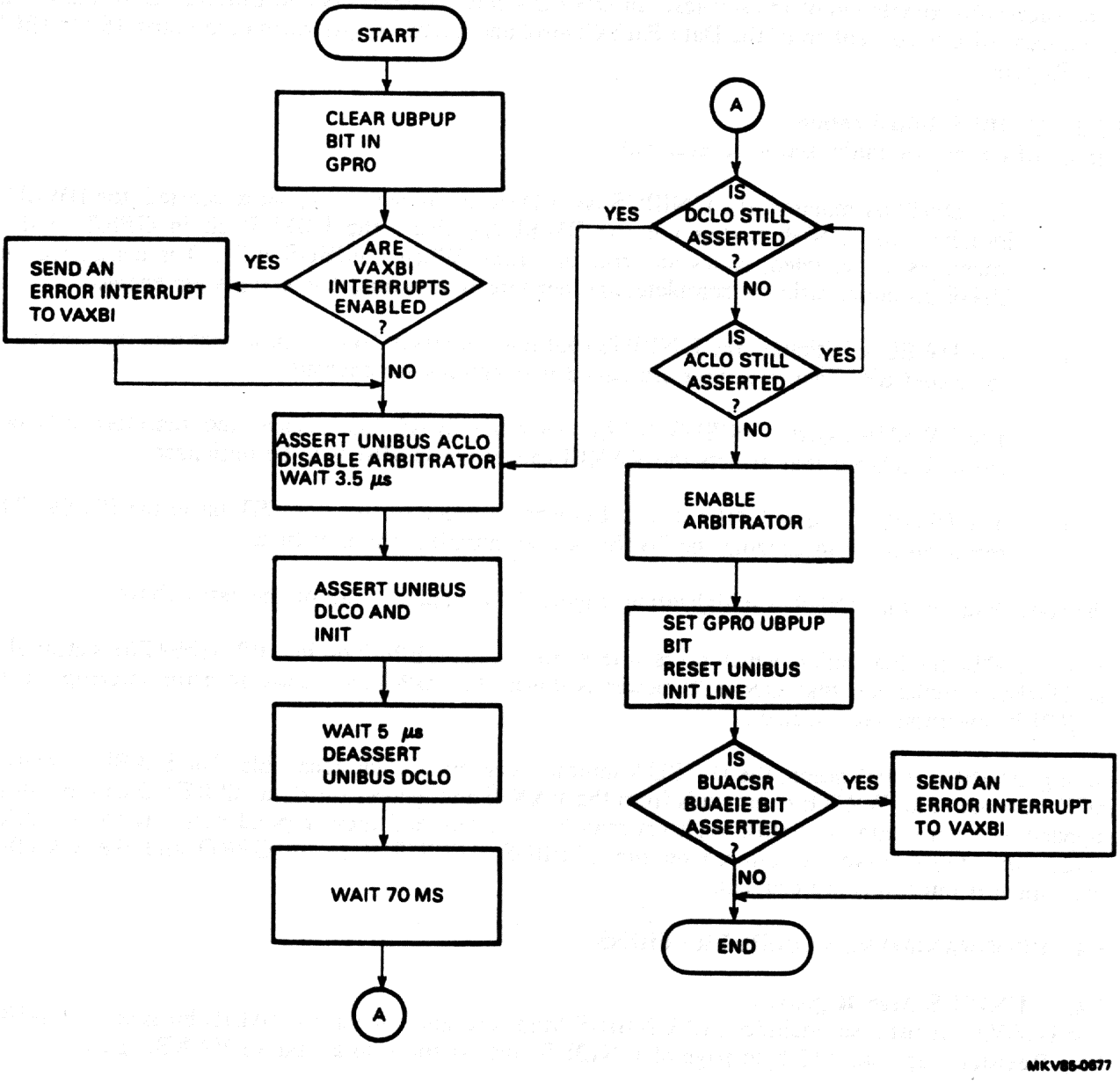


Figure 3-21 UNIBUS Initialization State Diagram

When a UNIBUS device initiates a transfer that corresponds to one of the upper 16 UNIBUS Map Registers, the DWBUA ignores the transfer and expects the UNIBUS device to respond. If a UNIBUS-initiated transfer accesses a UNIBUS Map Register with the VALID, PPIE, LWAEN, and BYTE OFFSET bits asserted and DPSEL<2:0> equal to 6 or 7, the DWBUA ignores the corresponding UNIBUS transfer.

3.4.1.1 Contiguous Allocation – One or more UNIBUS Map Registers must be allocated for each transfer. When more than one is allocated, the UNIBUS Map Registers must be contiguous in UNIBUS space, since sequential transfers are contiguous in UNIBUS space. This means that the set of UNIBUS Map Registers is contiguous in VAXBI node space. The contents of the UNIBUS Map Registers do not normally point to a contiguous area of VAXBI memory space.

3.4.1.2 Mapping to VAXBI I/O Space – UNIBUS Map Registers can be used to map to VAXBI I/O space, although no reason for doing so is known and many restrictions exist.

A UNIBUS device cannot modify the UNIBUS Map Registers of the DWBUA to which it is connected. An attempt to modify a UNIBUS Map Register results in the UNIBUS device never receiving SSYN. In addition, the DWBUA may hang.

3.4.1.3 BYTE OFFSET Bit – If the BYTE OFFSET bit in the UNIBUS Map Register is set and if the transfer uses n UNIBUS Map Registers, then the BYTE OFFSET bit should be set for all n registers and the n+1th register should be allocated and invalidated. If the n+1th register is VALID when a UNIBUS device issues a DATO with a UNIBUS address corresponding to the last word of the nth page, then two VAXBI WRITEs can occur: one includes the last byte of the nth page, and the other includes 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 UNIBUS Map Register since the DWBUA does not prefetch data from VAXBI memory space.

3.4.2 UNIBUS Power Down

When UNIBUS power goes down, the UNIBUS requires a minimum of 80 ms to complete its power-down/power-up sequence. During this sequence, the DWBUA cannot respond normally to VAXBI transactions. Any attempted access to window space or to node space (except the first 256 bytes, which are in BIIC space) receives ACK. For WRITE commands, the DWBUA ignores the data; for READ commands, the DWBUA returns zero data. Further, the IRCI command does not set the DWBUA adapter's internal interlock.

3.4.3 Use of Buffered Data Paths

VAXBI memory may be corrupted if a UNIBUS device issues nonsequential DATO(B) transactions through a BDP. In particular, a DATO with UNIBUS address 8*n followed by a DATO with UNIBUS address 8*n+14 causes the entire octaword in the BDP to be written to VAXBI memory space. A DATOB with UNIBUS address 8*n followed by a DATOB with UNIBUS address 8*n+15 has the same effect. This conforms to the standard restriction on UNIBUS devices which use BDPs (sequential transfers only) and causes the programming restrictions described in the next two paragraphs.

UNIBUS Map Registers associated with BDPs must not be double-allocated. A set of UNIBUS Map Registers may be allocated to only one transfer. Concurrently allocating a set of UNIBUS 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 DPCSR) before the UNIBUS Map Registers allocated to a transfer may be allocated to another transfer, and before the contents of the UNIBUS Map Registers may be changed.

After a UNIBUS power outage occurs and power is restored, all of the BDPs must be purged using the PURGE bit in the DPCSRs.

During a UNIBUS-initiated DATO using a Buffered Data Path transaction, the DWBUA issues SSYN before determining if the corresponding VAXBI transaction is required. (That is, the DWBUA issues SSYN 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 UNIBUS device. If the transaction is a DATI, the DWBUA completes the corresponding VAXBI transfer before it issues SSYN to the UNIBUS device.

3.4.4 VAXBI Access to the DWBUA Internal Registers

All IRCI transactions to the DWBUA internal registers are treated as READ commands and do not set the interlock on the DWBUA. All UWMCI and WMCI 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 UWMCI or WMCI) transaction to the READ-ONLY registers of the DWBUA also results in a NO ACK response.

3.4.5 Data Length

The DWBUA responds only to VAXBI transactions with a data length of longword. Quadword, octaword, and RESERVED data length transactions result in a NO ACK response.

3.4.6 IRCI/UWMCI Commands

When an IRCI transaction is issued to a DWBUA adapter's window space, the DWBUA first performs a DATIP transaction on the UNIBUS using the address supplied with the IRCI command. The DWBUA then sets its interlock. Once interlocked, the DWBUA responds with RETRY to all transactions issued to DWBUA window space or node space (except BIIC space) until a UWMCI transaction is received. The DWBUA ignores the address supplied with the UWMCI command. The DWBUA assumes that the UWMCI is addressed to the same word as the IRCI command and performs the DATO(B) to that word address (taking into account the mask bits supplied with the UWMCI data).

3.4.7 UNIBUS DATIP

When a UNIBUS device issues a DATIP, the DWBUA responds with RETRY to any VAXBI transaction issued to DWBUA window space or node space (except BIIC space) until a DATO(B) is sent by the UNIBUS master device.

3.4.8 Hung UNIBUS

If a UNIBUS device hangs the UNIBUS (for example, by not deasserting MSYN) the DWBUA will RETRY any VAXBI transaction issued to DWBUA window space or node space (except BIIC space).

3.4.9 VAXBI Bus Error

If the DWBUA encounters an error on the VAXBI during a DWBUA-initiated transaction, it sets the BIF bit in the BUACSR. The DWBUA also clears the mask bits and the internal BDP flags, thereby indicating that the buffer is empty for the current BDP. If this error occurs during a W(M)CI transaction, no indication exists of the data path for which the VAXBI transaction failed. The DWBUA may withhold SSYN, resulting in SSYN timeout to the UNIBUS device that initiated the transfer.

3.4.10 UNIBUS Devices

The DWBUA allows those UNIBUS devices that perform data transfers (instead of sending vectors) during the INTR cycle to be attached to the UNIBUS. These devices, however, cause a passive release every time they assert the BR lines to perform a 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 (WCI, WMCI, UWMCI) commands to READ-ONLY registers.

READ transactions to unimplemented BIIC registers read zero data. WRITE (WCI, WMCI, UWMCI) commands to these registers receive an ACK response; but the data is dropped.

CHAPTER 4
FUNCTIONAL DESCRIPTION

4.1 INTRODUCTION

The functional description of the DWBUA 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 DWBUA interfaces between the two buses.

4.2 BLOCK DIAGRAM

Figure 4-1 is the DWBUA block diagram. Table 4-1 contains functional descriptions of the blocks in Figure 4-1.

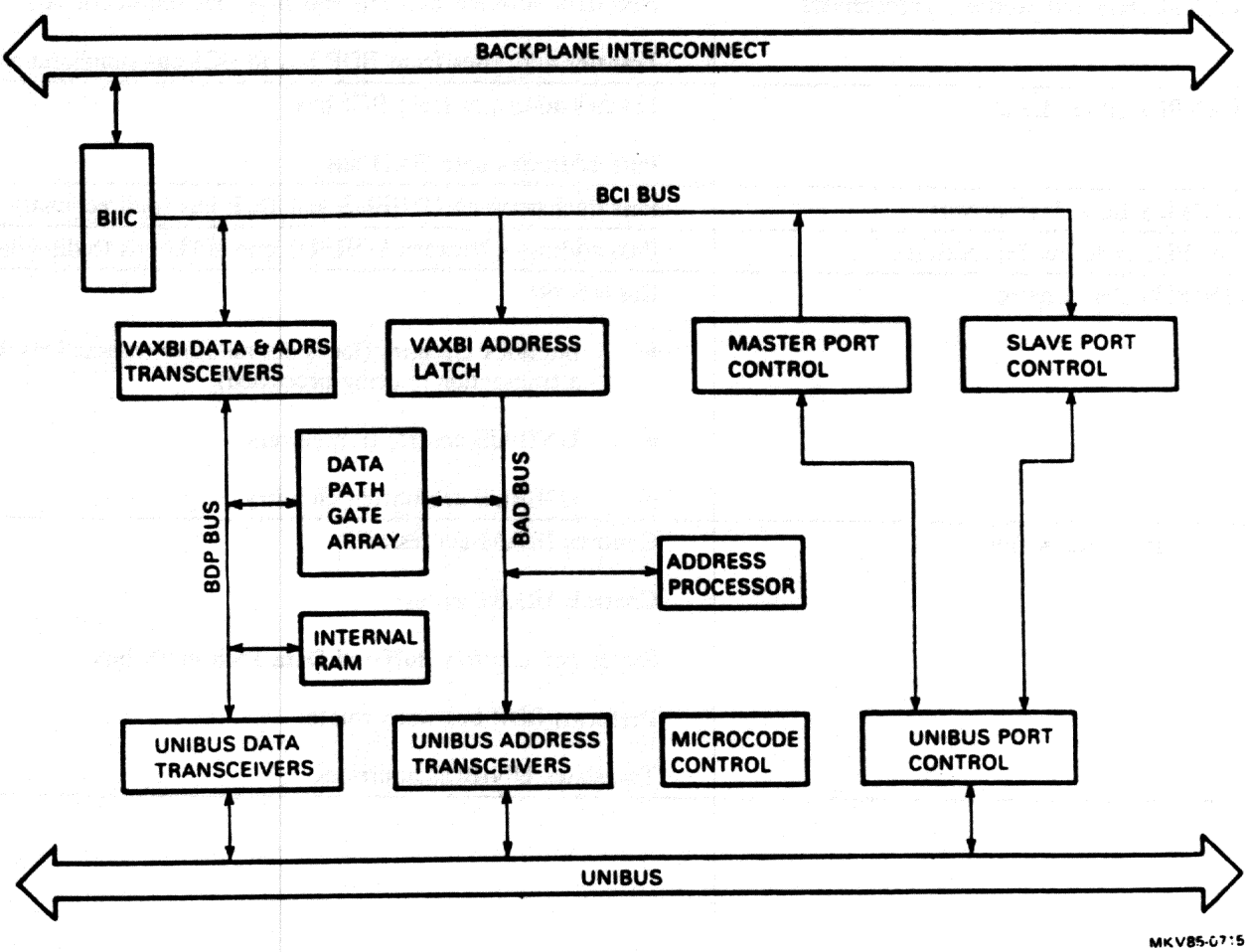


Figure 4-1 DWBUA Block Diagram

Table 4-1 DWBUA Block Diagram Descriptions

Block	Description
BIIC	Transfers data between VAXBI and DWBUA DWBUA adapter's only connection to VAXBI
Master Port Control	Controls: • UNIBUS transactions to VAXBI • DWBUA transactions to VAXBI • DWBUA transfers to BIIC for self-test
Slave Port Control	Receives transactions from VAXBI; verifies that they are intended for DWBUA or its UNIBUS Controls: • Transfers to DWBUA internal registers • Transfers to UNIBUS
VAXBI Data and Address Transceivers	Pass data between BCI bus and BDP bus (bidirectional) Transfer addresses from BDP bus to BCI bus (unidirectional)
VAXBI Address Latch	Latches addresses from BCI bus Puts addresses onto BAD bus
UNIBUS Data Transceivers	Pass data between UNIBUS and BDP bus (bidirectional)
UNIBUS Address Transceivers	Pass addresses between UNIBUS and BAD bus (bidirectional)
UNIBUS Port Control	Consists of: • Interlock circuitry (locks out all other transactions while a transaction is being processed) • UNIBUS control transceivers • UNIBUS arbitration circuitry
Data Path Gate Array	Controls IRAM addresses Controls IRAM writes Stores and controls Buffered Data Path mask bits Performs BDP bus word rotates Translates UNIBUS addresses

Table 4-1 DWBUA Block Diagram Descriptions (Cont)

Block	Description
Internal RAM (IRAM)	2K × 32 RAM Contains: • DWBUA Internal Registers • UNIBUS Map Registers • Temporary storage for self-test • Buffered Data Path buffers (one octaword of storage for each of the five Buffered Data Paths)
Address Processor	Provides address and byte count storage Performs address matching for BDP transactions Generates and stores constants Stores and tests flags for BDP transactions Performs byte rotation for byte offset
Microcode Control	Controls data path gate array Controls address processor Sends instructions to UNIBUS control Sends instructions to master port control Interlocks transactions between UNIBUS and VAXBI
BCI Bus	BI chip interface bus Synchronous interface bus Provides all communication between the BIIC and the DWBUA
BAD Bus	Buffered address bus Internal address bus for all but VAXBI addresses
BDP Bus	Internal data and VAXBI address bus

4.3 TRANSACTIONS

The DWBUA acts as a translator between the VAXBI and the UNIBUS. It interprets commands received from one bus into a format that the other bus can understand, and it provides controls and responses that enable the completion of these commands. These sequences of commands, controls, and responses are called DWBUA transactions. In this section, some typical transactions handled by the DWBUA are examined in detail.

DWBUA transactions are divided into three categories:

- VAXBI-to-DWBUA
- VAXBI-to-UNIBUS
- UNIBUS-to-VAXBI

4.3.1 VAXBI-to-DWBUA Transactions

4.3.1.1 DWBUA Responses to VAXBI-to-DWBUA Transactions – The VAXBI sends READ and WRITE commands to the DWBUA. The purpose of these commands is to read data from or to write data to the DWBUA adapter's internal registers. The VAXBI node that initiates the transaction is the VAXBI master, and the DWBUA is the VAXBI slave in all VAXBI-to-DWBUA transactions.

Table 4-2 DWBUA Responses to VAXBI-to-DWBUA Transactions

VAXBI-to-DWBUA Transaction	DWBUA Response
READ of DWBUA internal register	1. STALL 2. Register data with read data status code 3. ACK
READ of unused DWBUA register space	NO ACK
WRITE to DWBUA internal register	1. STALL 2. ACK (if no parity error on the VAXBI*) 3. Register updated
WRITE to unused DWBUA register space or READ-ONLY register	NO ACK

* If a parity error occurs, the register is not updated.

4.3.1.2 VAXBI-to-DWBUA Commands –

Table 4-3 VAXBI-to-DWBUA Commands

VAXBI Command	DWBUA Response	Possible Errors	See Note
READ	ACK/RETRY	None	1
IRCI	ACK/RETRY	None	1,2
RCI	ACK/RETRY	None	1
WRITE	ACK/RETRY	Parity Error	1,5
WCI	ACK/RETRY	Parity Error	1,5
WMCI	ACK/RETRY	Parity Error	1,4,5
UWMCI	ACK/RETRY	Parity Error	1,3,5

NOTES FOR TABLE 4-3:

(1) Longword length only.

(2) IRCI commands are accepted, but they are treated as READ commands. The DWBUA does not interlock.

(3) UWMCI commands are accepted, but they are treated as WRITE commands. The DWBUA does not interlock. The mask bits are ignored.

(4) WMCI 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 DWBUA ignores the transaction.

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

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

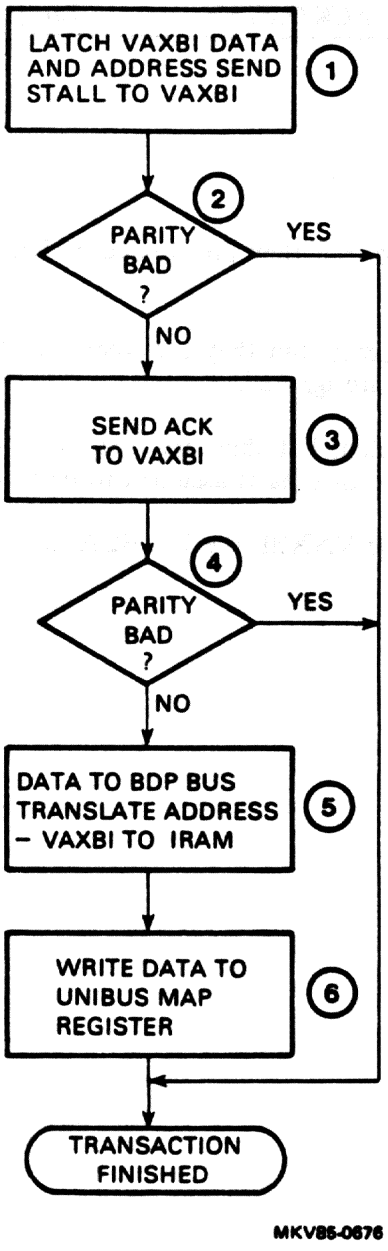


Figure 4-2 VAXBI WRITE to a UNIBUS Map Register Flow Diagram

① The VAXBI command, address, and data are received by the DWBUA. The slave port control determines that the transaction is for the DWBUA. 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 UNIBUS Map Register that will be written. The VAXBI data will be written into this UNIBUS Map Register.

The initial DWBUA response to the VAXBI is STALL.

② The DWBUA 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 DWBUA responds to the VAXBI with ACK. The VAXBI interprets the transaction as complete.

④ The DWBUA checks for a parity error on the VAXBI. The DWBUA terminates the transaction immediately if the data received in the VAXBI cycle in which ACK was sent has a parity error. The DWBUA issues an error interrupt 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 to the address of the UNIBUS Map Register to be written.

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

4.3.2 VAXBI-to-UNIBUS Transactions

4.3.2.1 DWBUA Responses to VAXBI-to-UNIBUS Transactions – In a VAXBI-to-UNIBUS transaction, the VAXBI master sends to the DWBUA a command that requires the DWBUA to read from or to write to a UNIBUS device.

Table 4-4 Bus Masters and Slaves for VAXBI-to-UNIBUS Transactions

Bus	Master	Slave
VAXBI	Node initiating transaction	DWBUA
UNIBUS	DWBUA	UNIBUS device

The DWBUA monitors the UNIBUS BBSY signal before it attempts to perform the DATO(B) transaction on the UNIBUS. (If BBSY is deasserted, the DWBUA asserts BBSY and gains UNIBUS mastership.) If the DWBUA does not gain UNIBUS mastership within 51 μs, UNIBUS 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 (5,9)

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 USSTO bit in the BUACSR

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 BUACSR

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, so it ignores the incoming address.

4.3.2.2 VAXBI-to-UNIBUS Commands –

Table 4-6 VAXBI-to-UNIBUS Commands

VAXBI COMMAND		UNIBUS Command Translation	DWBUA Response to VAXBI	Possible Errors	See Note
Code	Name				
0000	Reserved	None	NO ACK	None	1
0001	READ	DATI	ACK/RETRY	USSTO	2,12
0010	IRCI	DATIP	ACK/RETRY	USSTO	3,12
0011	RCI	DATI	ACK/RETRY	USSTO	4,12
0100	WRITE	DATO	ACK/RETRY	USSTO VAXBI PE	5, 12,13
0101	WCI	DATO	ACK/RETRY	USSTO VAXBI PE	6, 12,13
0110	UWMCI	DATO(B)	ACK/RETRY	USSTO VAXBI PE	3,7, 12,13
0111	WMCI	DATO(B)	ACK/RETRY	USSTO VAXBI PE	7, 12,13
1000	INTR	None	NO ACK	None	8
1001	IDENT	BGn	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	BDCST	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 DWBUA responds to the codes with NO ACK.

(2) All VAXBI READs of UNIBUS space are limited to longword length only. VAXBI address bit A<01> determines which word is read from the UNIBUS.

(3) IRCI/UWMCI commands operate as UNIBUS DATIP/DATO(B) sequences. The DWBUA is interlocked by the IRCI command; the UWMCI command releases the interlock. All other READ and WRITE commands directed to the DWBUA receive RETRY responses while the DWBUA is interlocked. Due to UNIBUS constraints, the address supplied for a UWMCI command must be the same as for the IRCI command. Hence, the DWBUA uses the address of the IRCI command while servicing the UWMCI command, ignoring the address supplied with the latter command.

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

(5) VAXBI-to-UNIBUS WRITEs are 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 length is longword only. VAXBI address bit A<01> determines which word of a longword is written. If either of the two mask bits is not set in the selected word, the DWBUA will respond to the command with ACK. This may corrupt the UNIBUS data or may cause a SSYN timeout. Mask information for the word not selected by VAXBI bit A<01> is ignored by the DWBUA.

(8) Since interrupts are only permitted in the UNIBUS-to-VAXBI direction, the DWBUA responds to all INTR commands with NO ACK.

(9) The DWBUA responds to IDENT commands with a previously failed interrupt vector (if present) at an appropriate level. If there is no failed vector, the DWBUA will fetch an interrupt vector from the UNIBUS device by issuing a BG at the corresponding level of the IDENT command.

(10) The STOP command resets all pending interrupts and DMA requests from the UNIBUS. The DWBUA does not alter the contents of any register implemented in the user CSR space. The DWBUA responds to all subsequent VAXBI commands, but does not attempt to gain mastership of the VAXBI. This effect of the STOP command is reset only by BCI DCLO.

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

(12) USSTO – The corresponding UNIBUS transaction has resulted in a SSYN timeout. (The DWBUA did not receive SSYN within 19.2 μs after asserting MSYN.) The USSTO bit in the BUACSR is set, and an error interrupt is sent to the VAXBI (if interrupts are enabled).

(13) VAXBI PE – Parity error on the VAXBI. The DWBUA ignores the transaction.

(14) SACK – SACK is not asserted by the interrupting UNIBUS device. The DWBUA 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.

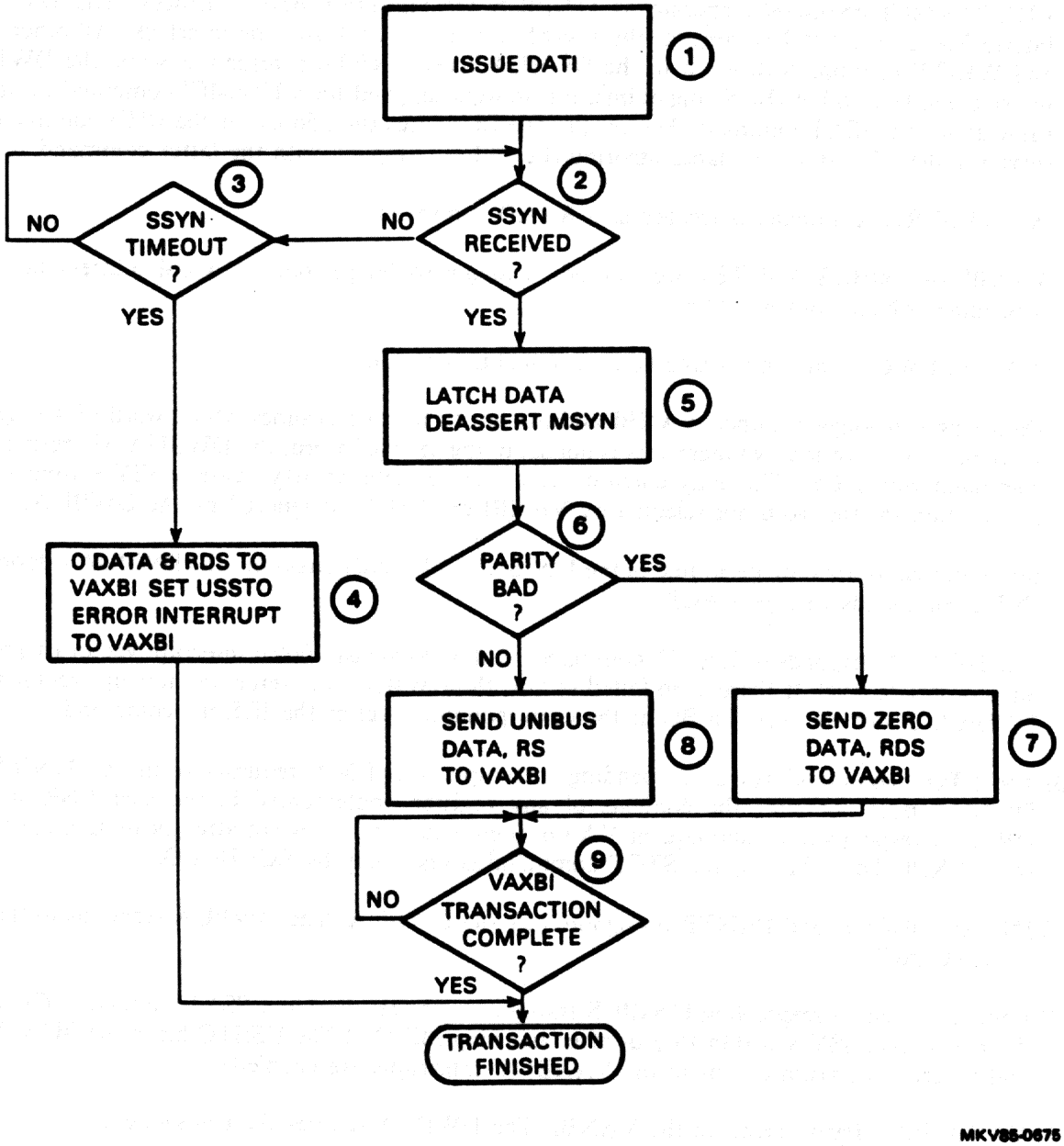


Figure 4-3 VAXBI READ of UNIBUS Data Flow Diagram

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 slave device puts the data onto the UNIBUS D lines.

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

③ If SSYN is not asserted within 19.2 μs from assertion of MSYN, a SSYN timeout occurs.

④ SSYN timeout causes the DWBUA to send zero data and RDS to the VAXBI, set the USSTO bit of the BUACSR, 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 BCI 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.

4.3.3 UNIBUS-to-VAXBI Transactions

4.3.3.1 DWBUA Responses to UNIBUS-to-VAXBI Transactions – In a UNIBUS-to-VAXBI transaction, the UNIBUS master sends a command to the DWBUA that requires the DWBUA to read from or to write to a VAXBI node.

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

Bus	Master	Slave
UNIBUS	Device initiating transaction	DWBUA
VAXBI	DWBUA	VAXBI node

The DWBUA responds to three UNIBUS commands that require DWBUA interaction with other VAXBI nodes: DATI, DATO(B), and DATIP/DATO(B). These responses are listed in Table 4-8. The DWBUA responses to UNIBUS commands are independent of the data path used.

Table 4-8 DWBUA Responses to UNIBUS-to-VAXBI Transactions

UNIBUS-to-VAXBI Transaction	DWBUA Response
DATI (1)	Data (2) SSYN (3)
DATO(B) (4)	SSYN (5) Data to VAXBI (6)
DATIP/DATO(B) (7)	
DATIP (8)	{ Data (2) { SSYN (3)
DATO(B) (9)	{ SSYN (5) { Data to VAXBI (6)

NOTE

If the DWBUA is processing a VAXBI transaction when the UNIBUS request is received, the DWBUA withholds the bus grant until the VAXBI transaction has completed.

NOTES FOR TABLE 4-8:

(1) A DATI command from a UNIBUS device reads data from the DWBUA.

(2) If a VAXBI error occurs while the DWBUA is fetching the data from the VAXBI, SSYN may be withheld. If it is withheld, a SSYN timeout results. The DWBUA sets the BIF bit of the BUACSR and the BIIC issues an error interrupt on the VAXBI if interrupts are enabled.

(3) The DWBUA issues SSYN in response to a DATI command only when the VAXBI slave responds to the VAXBI portion of the transaction with ACK. Any other response from the VAXBI slave results in the DWBUA withholding SSYN.

(4) When the DWBUA processes a DATO(B) command, it accepts data from a UNIBUS device.

(5) The DWBUA issues SSYN before it completes the corresponding VAXBI transaction.

(6) If an error occurs while the DWBUA is writing the data to the VAXBI node, the DWBUA sets the BIF bit of the BUACSR, and the BIIC issues an error interrupt on the VAXBI if errors are enabled. SSYN may be withheld from the UNIBUS device; withholding SSYN results in an SSYN timeout.

(7) The DATIP/DATO(B) command sequence may be performed only through the DWBUA adapter's Direct Data Path. An attempt to perform this command sequence through a Buffered Data Path results in an SSYN timeout. The BYTE OFFSET bit in the UNIBUS Map Register corresponding to the Direct Data Path is ignored and treated as if it is clear; if the bit is set, the command is treated as if the bit is clear.

(8) A SSYN timeout occurs if a DATIP through the Direct Data Path results in a failure on the VAXBI.

(9) UNIBUS protocol requires that a DATIP be followed immediately by a DATO(B) command. BBSY and the address lines must not be deasserted between the two commands. Any deviation from this causes the DWBUA to set the UIE bit of the BUACSR. If a DATO(B) is received, but a VAXBI failure occurs during the UWMCI that is generated, then the BIF bit of the BUACSR is set. Each of these errors causes an error interrupt on the VAXBI if interrupts are enabled.

4.3.3.2 UNIBUS-to-VAXBI Commands Through the Direct Data Path –

NOTE

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

Table 4-9 UNIBUS-to-VAXBI Commands Through the Direct Data Path

UNIBUS Command	UNIBUS Address <3:0>	Command to VAXBI	Transfer Length	Possible Errors	See Note
BYTE OFFSET BIT = 0					
DATI	ANY	READ	LONGWORD	A, B	4
DATIP	ANY	IRCI	LONGWORD	A, B, C	1,4
DATO	ANY	WMCI or UWMCI	LONGWORD	A, B	2,4
DATOB	ANY	WMCI or UWMCI	LONGWORD	A, B	4
BYTE OFFSET BIT = 1					
DATI	ANY	READ	LONGWORD	A, B	3,4
DATIP	ANY	N/A	N/A	N/A	1
DATO	ANY	WMCI	LONGWORD	A, B	3,4
DATOB	ANY	WMCI	LONGWORD	A, B	4

NOTES FOR TABLE 4-9:

(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 OFFSET bit set, the UNIBUS command is ignored and the DWBUA does not issue SSYN. This causes an SSYN timeout. During this time, all VAXBI transactions to the DWBUA receive a RETRY response until the UNIBUS device negates BBSY.

If a DATIP command is not followed by a DATO(B), the DWBUA sets the UIE bit of the BUACSR and forces an error interrupt (if interrupts are enabled).

(2) A UNIBUS DATO(B) through the Direct Data Path translates to a longword WMCI transaction with the mask bits set for each valid data byte.

(3) The DWBUA performs two longword trasactions on the VAXBI for a word length transfer through the Direct Data Path if both the BYTE OFFSET bit and UNIBUS address bit A<1> are set.

(4) Possible Errors:

(A) BIF – The VAXBI transaction has returned an event code that the DWBUA recognizes as an error code: BTO, RDSR, ICRMC, NCRMC, ICRMD, BPM, or MTCE. The BIF bit is set in the BUACSR and an error interrupt is sent to the VAXBI if interrupts are enabled. The DWBUA may withhold SSYN, resulting in an SSYN timeout to the UNIBUS device that initiated the transfer.

(B) IMR – The VALID bit is not set in the UNIBUS Map Register for the incoming UNIBUS address. The IMR bit is set in the BUACSR and an error interrupt is sent to the VAXBI if interrupts are enabled.

(C) UIE – The UNIBUS master deasserted BBSY after the DATIP, before executing the accompanying DATO(B). The UIE bit is set in the BUACSR and an error interrupt is sent to the VAXBI if interrupts are enabled.

4.3.3.3 Example: DATO(B) Using the Direct Data Path – In this transaction the UNIBUS 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 DATO(B) using the Direct Data Path transaction. The numbered paragraphs that follow refer to the corresponding numbers in Figure 4-4.

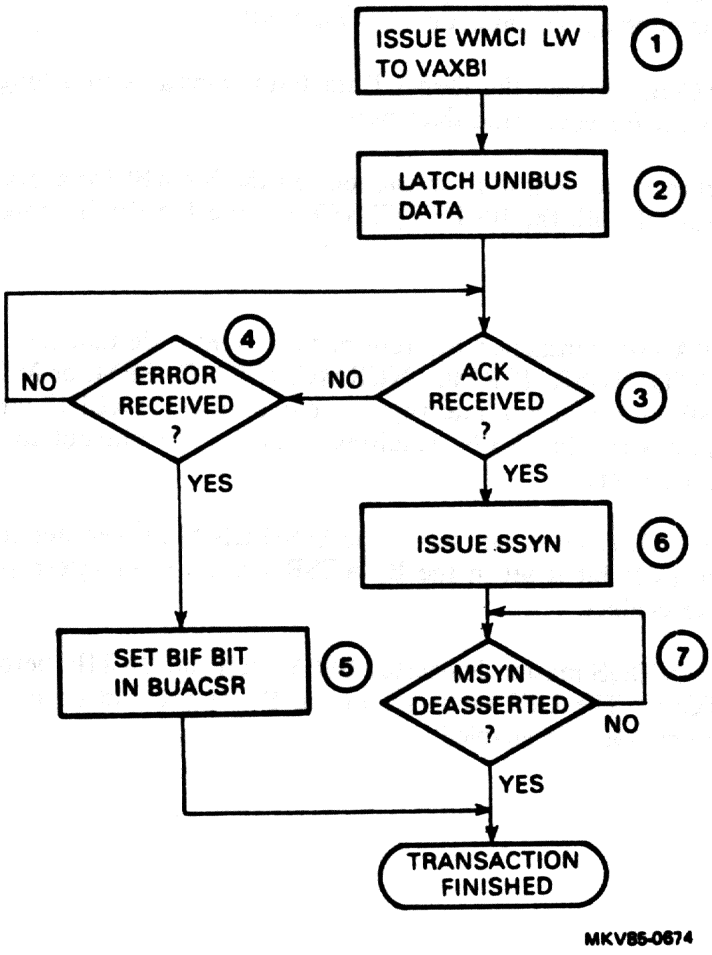


Figure 4-4 DATO(B) Using the Direct Data Path Flow Diagram

① The UNIBUS master sends the address, control bits, and data to the DWBUA, and it then issues MSYN. The DWBUA decodes the control bits and determines that the command is a DATO(B).

The DWBUA requests the VAXBI and starts a WMCI LW (write mask with cache intent – longword) transaction. This is the VAXBI transaction that corresponds to a UNIBUS DATO. The mask bits are determined by the command (DATO or DATOB). The UNIBUS address is longword aligned.

② The UNIBUS data is latched in the UNIBUS data transceivers.

The data goes from the UNIBUS data transceivers to the BDP bus, through the VAXBI data transceivers, and to the VAXBI.

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

④ If ACK is not received, the DWBUA looks for an error.

⑤ If an error is received, the BIF bit in the BUACSR is set and the transaction is terminated. SSYN may not be issued, resulting in an SSYN timeout.

⑥ The DWBUA issues SSYN, completing the UNIBUS transaction.

⑦ The DWBUA monitors MSYN. When it is deasserted, the VAXBI transaction is complete.

4.3.3.4 UNIBUS-to-VAXBI Commands Through a Buffered Data Path –

Table 4-10 UNIBUS-to-VAXBI Commands Through a Buffered Data Path

UNIBUS Command	UNIBUS Address <3:0>	Buffer Status	Command to VAXBI	Transfer Length	Possible Errors	See Note
BYTE OFFSET BIT = 0						
DATI	ANY	EMPTY	READ	OCTAWORD	B, C, D	3,5
DATI	ANY	IN/M	NONE	N/A		6
DATI	ANY	IN/D	READ	OCTAWORD	B, C, D	5,6
DATI	ANY	OUT	WMCI or READ	OCTAWORD	B, C, D	5,6
DATIP	ANY	N/A	N/A	N/A		2
DATO	ANY	IN or EMPTY	WMCI	OCTAWORD	B, C, D	4,5, 6,7
DATO	ANY	OUT/M	WMCI	OCTAWORD	B, C, D	5,6,7
DATO	ANY	OUT/D	WMCI	OCTAWORD	B, C, D	5,6,8
DATOB	ANY	IN or EMPTY	WMCI	OCTAWORD	B, C, D	5,6,7
DATOB	ANY	OUT/M	WMCI	OCTAWORD	B, C, D	5,6,7
DATOB	ANY	OUT/D	WMCI	OCTAWORD	B, C, D	5,6,8
BYTE OFFSET BIT = 1						
DATI	0 to C	EMPTY	READ	OCTAWORD	A, B, D	5
DATI	0 to C	IN/M	NONE	N/A		6
DATI	0 to C	IN/D	READ	OCTAWORD	A, B, D	5,6
DATI	0 to C	OUT	READ	LONGWORD	A, B	5,6
DATI	E	EMPTY	READ	OCTAWORD	A, B, D	5
DATI	E	IN/M	READ	OCTAWORD	A, B, D	5,6
DATI	E	IN/D	READ	OCTAWORD	A, B, D	5,6
DATI	E	OUT	READ	LONGWORD	A, B	1,5,6
DATIP	ANY	N/A	NONE	N/A		2
DATO	0 to C	IN or EMPTY	NONE	N/A		6
DATO	0 to C	OUT/M	NONE	N/A		6
DATO	0 to C	OUT/D	WMCI	OCTAWORD	B, C, D	5,6
DATO	E	IN or EMPTY	WMCI	OCTAWORD	B, C, D	5,6
DATO	E	OUT/M	WMCI	OCTAWORD	B, C, D	5,6
DATO	E	OUT/D	WMCI	OCTAWORD	B, C, D	5,6
DATOB	ANY	IN or EMPTY	WMCI	OCTAWORD	B, C, D	5,6,7
DATOB	ANY	OUT/M	WMCI	OCTAWORD	B, C, D	5,6,7
DATOB	ANY	OUT/D	WMCI	OCTAWORD	B, C, D	5,6,8

NOTES FOR TABLE 4-10:

(1) This special case is treated differently from other Buffered Data Path transfers to avoid delay in issuing SSYN. In this case, the low byte of the requested DATI word is fetched by performing a longword READ through the Direct Data Path. The high byte is fetched from either the current BDP buffer or the VAXBI with a longword READ through the DDP. The current BDP status remains unchanged during this transaction.

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

(3) A UNIBUS DATI command through a Buffered Data Path results in an octaword READ of VAXBI space, if the requested data is not in the BDP buffer. If the UNIBUS data is stored in the BDP buffer, however, the buffer must be purged by performing an octaword WMCI on the VAXBI before reading the data from the VAXBI. The entire octaword is loaded into the buffer; subsequent accesses within the octaword through the same Buffered Data Path cause the DWBUA to fetch the data from the buffer, with no VAXBI transaction requested.

(4) Data for a DATO(B) command through a BDP is stored until the buffer is full. The DWBUA then performs a VAXBI octaword WRITE (nonmasked) if the buffer contains an entire octaword of valid data from the UNIBUS device. A VAXBI octaword WMCI is performed if the buffer contains less than a complete octaword of valid data.

(5) Possible Errors:

(A) BIF - The VAXBI transaction has returned an event code that the DWBUA recognizes as an error code: BTO, RDSR, ICRMC, NCRMC, ICRMD, BPM, or MTCE. The BIF bit is set in the BUACSR and an error interrupt is sent to the VAXBI if interrupts are enabled. The DWBUA may withhold SSYN, resulting in an SSYN timeout to the UNIBUS device that initiated the transfer.

(B) IMR - The VALID bit is not set in the UNIBUS Map Register for the incoming UNIBUS address. The IMR bit is set in the BUACSR and an error interrupt is sent to the VAXBI if interrupts are enabled.

(C) UIE - The UNIBUS master deasserted BBSY after the DATIP, before executing the accompanying DATO(B). The UIE bit is set in the BUACSR and an error interrupt is sent to the VAXBI if interrupts are enabled.

(D) BADBDP - The UNIBUS Map Register that corresponds to the incoming UNIBUS address has a 6 or 7 in the BDP SEL field. (It is attempting to select Buffered Data Path 6 or 7.) The BADBDP bit is set in the BUACSR and an error interrupt is sent to the VAXBI if interrupts are enabled.

(6) Buffer status

IN/M - The BDP buffer contains the UNIBUS DATI data received from the VAXBI, and the addresses match.

IN/D - The BDP buffer contains the UNIBUS DATI data received from the VAXBI, and the addresses do not match.

OUT - The BDP buffer contains the UNIBUS DATO data to be sent to the VAXBI.

EMPTY - The BDP buffer is empty

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

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

4.3.3.5 Example: DATO Using a Buffered Data Path – In this transaction the UNIBUS master writes data to a VAXBI node. Each DATO writes two bytes of data into a BDP buffer. The BDP buffer can hold sixteen bytes, so eight of these transactions are required in order to fill completely the BDP buffer. The buffer is written in one operation to the VAXBI node.

Figure 4-5 is a flow diagram of the DATO using a Buffered Data Path transaction. The numbered paragraphs that follow refer to the corresponding numbers in Figure 4-5.

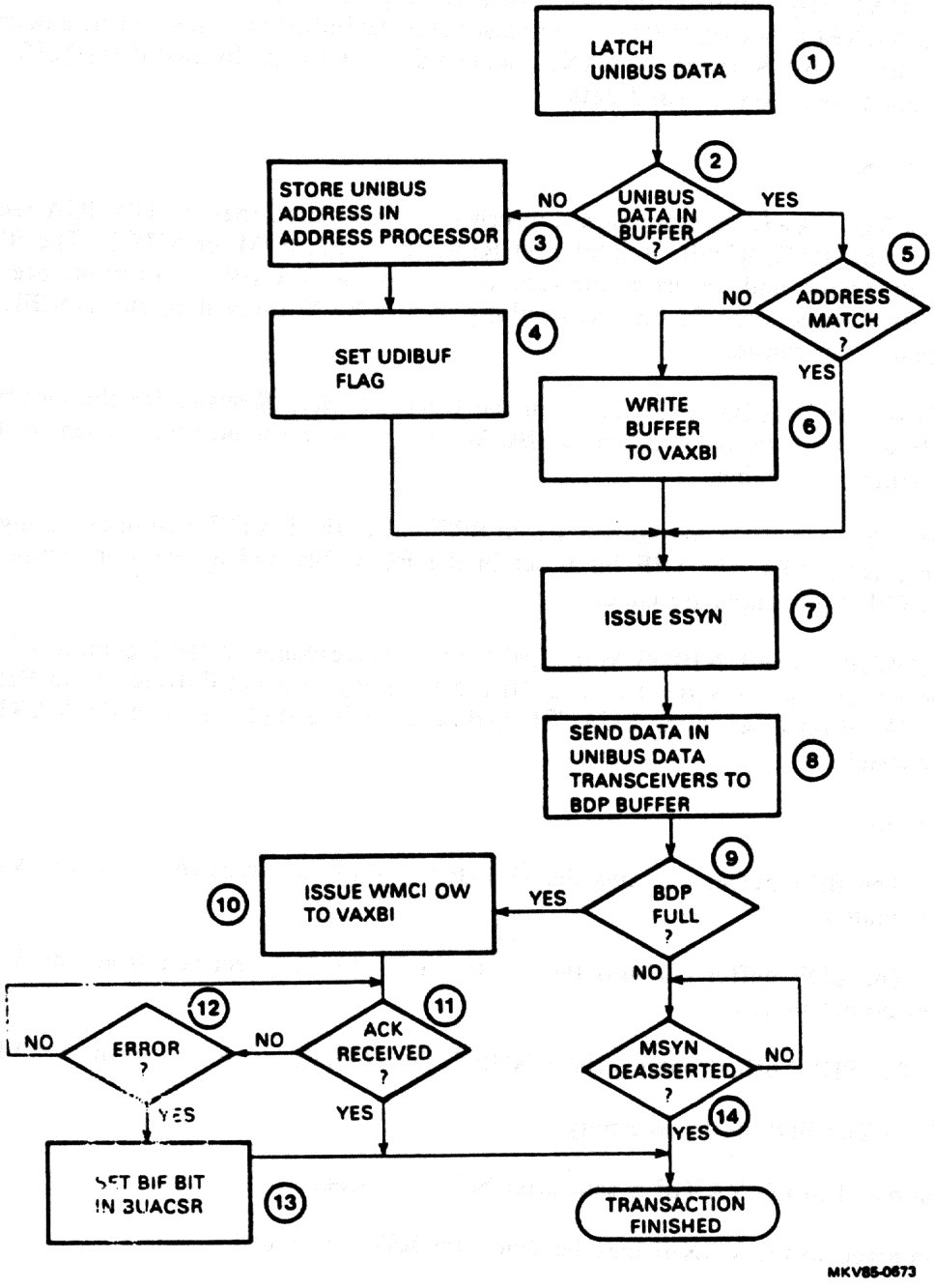


Figure 4-5 DATO Using a Buffered Data Path Flow Diagram

(1) The UNIBUS master sends address, control bits, and data to the DWBUA, and then issues MSYN.

The two bytes of UNIBUS data are latched in the UNIBUS data transceivers.

(2) The UDIBUF bit in the Data Path Control and Status Register is checked.

(3) If the UDIBUF bit is clear, the UNIBUS address goes over the BAD bus to the address processor where it is stored.

(4) The DWBUA sets the UDIBUF bit, indicating that UNIBUS data is stored in the BDP buffer.

(5) If the UDIBUF bit is set, the DWBUA must determine if the present data is part of the same octaword as the data already in the BDP buffer. To determine if the present data is part of the same octaword, bits <17:4> of the address stored in the address processor are compared with the incoming UNIBUS address.

(6) If the compared addresses do not match, the selected BDP buffer is autopurged. That is, the data in the BDP buffer is written to the VAXBI with an octaword WMCI command. In this way, the DWBUA ensures that the existing data is not overwritten and lost.

(7) The DWBUA issues SSYN, completing the UNIBUS portion of the transaction.

(8) The two bytes of data are sent from the UNIBUS data transceivers over the BDP bus to the BDP buffer in the internal RAM, to the appropriate location within the octaword.

(9) The DWBUA checks the BDP buffer to determine if it is full.

(10) If the BDP buffer is full, the DWBUA becomes the VAXBI master and sends the contents of the BDP buffer to the VAXBI. The command used to do this depends on whether or not all of the data in the BDP buffer is valid.

a. All data valid – octaword WRITE
b. Some data not valid – octaword WMCI

The mask bits sent with the WMCI data correspond to all of the valid data words written in the BDP buffer. After the transaction is complete, the DWBUA resets the mask for the entire BDP buffer.

(11) The DWBUA waits for the VAXBI slave node to issue ACK, ending the transaction.

(12) If ACK is not received, the DWBUA looks for an error code.

(13) If an error has occurred on the VAXBI, the DWBUA sets the BIF bit in the BUACSR and the transaction is terminated.

(14) The DWBUA waits until MSYN is deasserted by the UNIBUS master and then it ends the transaction.

4.3.3.6 Example: DATI Using a Buffered Data Path – In this transaction the UNIBUS master reads data from a VAXBI node. The VAXBI sends sixteen bytes of data to a BDP buffer. This buffer is read by the UNIBUS device two bytes at a time, so eight DATI transactions are needed 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.

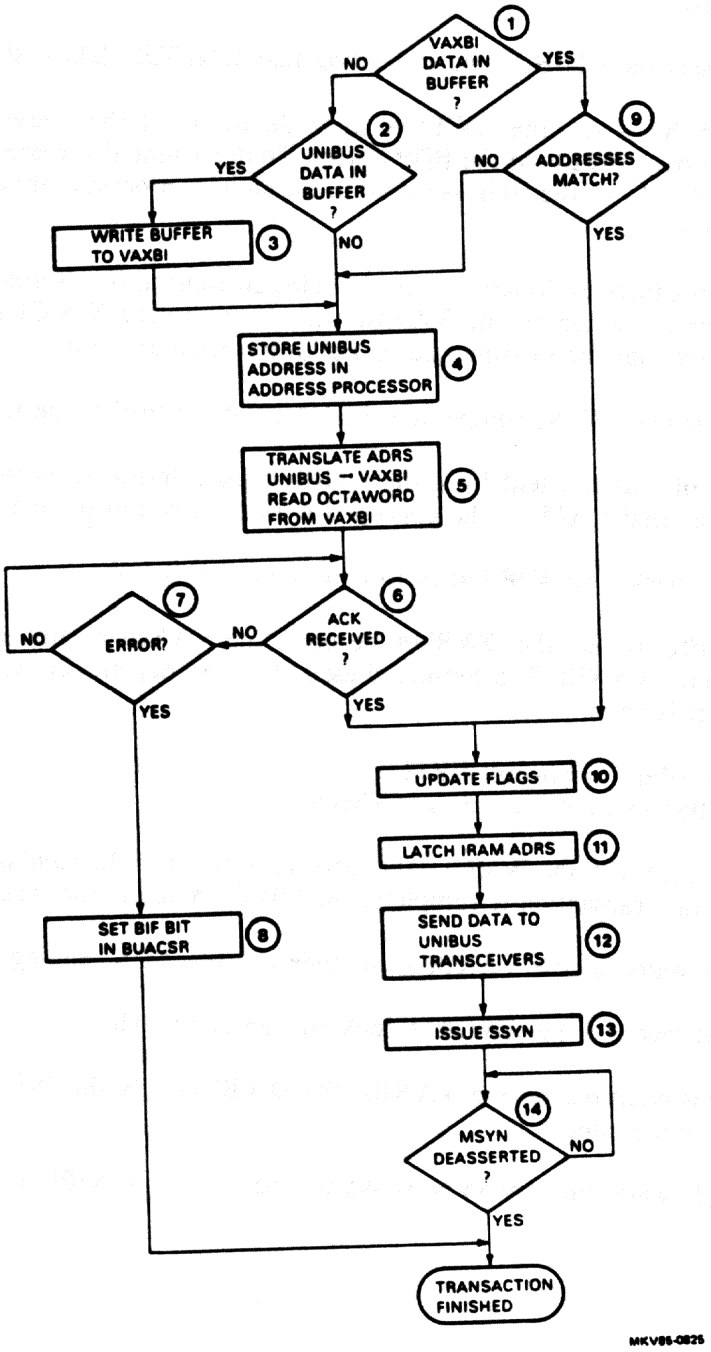


Figure 4-6 DATI Using a Buffered Data Path Flow Diagram

(1) The DWBUA checks the BDIBUF bit in the Data Path Control and Status Register.

(2) If the BDIBUF bit is clear, the BDP buffer does not contain VAXBI data. The UDIBUF bit is tested.

(3) If the UDIBUF bit is set, the BDP buffer contains UNIBUS data. The buffer contents are autopurged.

(4) The UNIBUS address is stored in the address processor.

(5) The UNIBUS address is translated into a VAXBI address. A VAXBI READ is initiated and an octaword of data is returned by the VAXBI slave. The data goes into the BDP buffer.

(6) The DWBUA waits for the VAXBI slave node to issue ACK.

(7) If ACK is not received, the DWBUA looks for an error code.

(8) If an error occurred on the VAXBI, the BIF bit in the BUACSR is set and the transaction is terminated.

(9) If the BDIBUF bit in the DPCSR is set, bits <17:4> of the address stored in the address processor are compared with the UNIBUS address latched in the UNIBUS address transceivers.

(10) BDIBUF and UDIBUF bits are updated.

(11) The UNIBUS address (from the address processor) is latched.

(12) The requested data word goes from the BDP buffer and is latched in the UNIBUS data transceivers.

(13) The DWBUA issues SSYN.

(14) The DWBUA waits for MSYN to be deasserted by the UNIBUS master. When MSYN is deasserted, the transaction is finished.

4.4 REPRESENTATIVE TIMING DIAGRAMS

The timing diagrams in this section represent typical DWBUA transactions. The following assumptions apply that:

1. No errors occur during the transaction.
2. The transaction follows a straight-line path through the flows.
3. No time scale is employed. The diagrams indicate relative timing only.

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

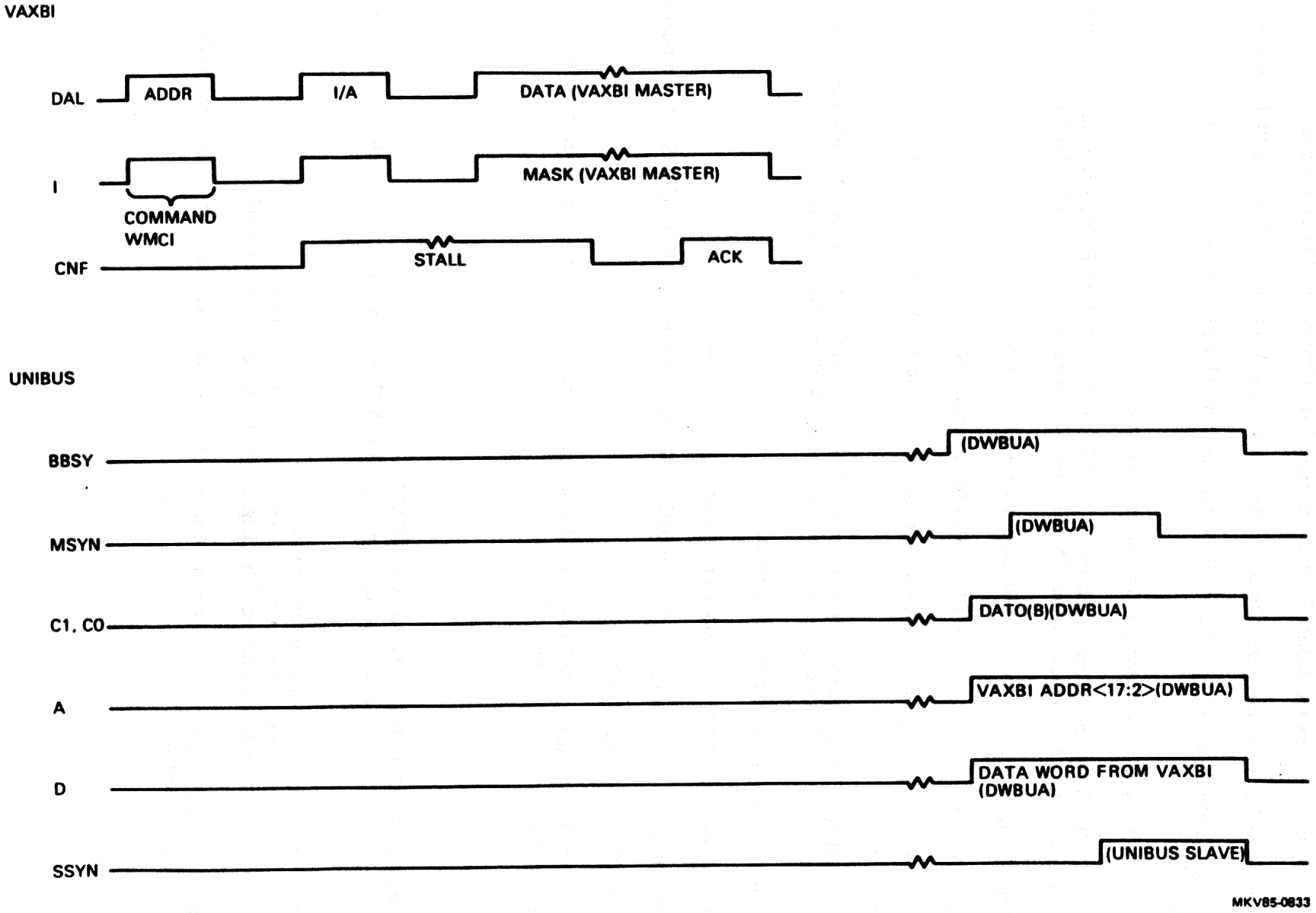


Figure 4-7 VAXBI-to-UNIBUS WRITE Timing Diagram

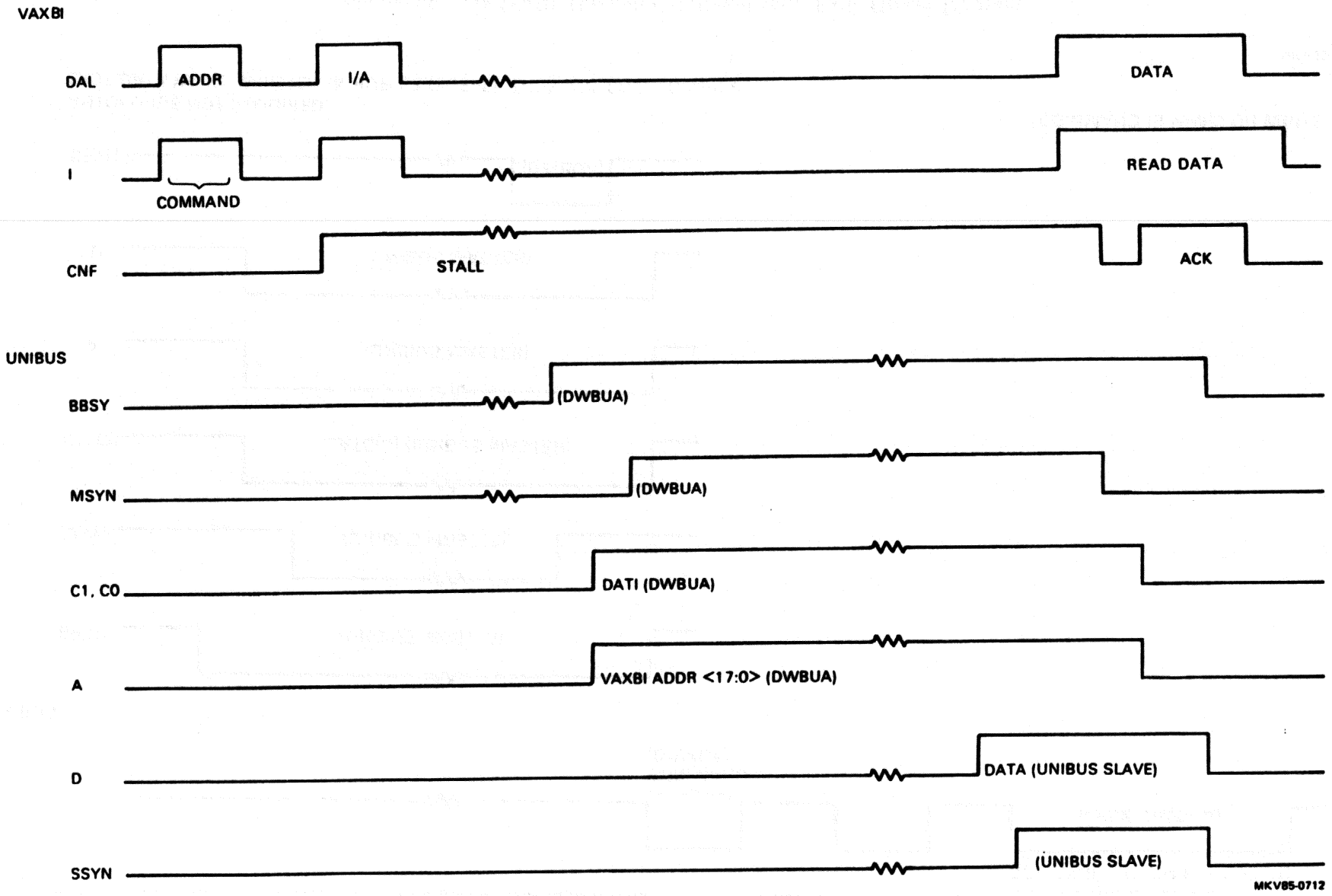


Figure 4-8 VAXBI-to-UNIBUS READ Timing Diagram

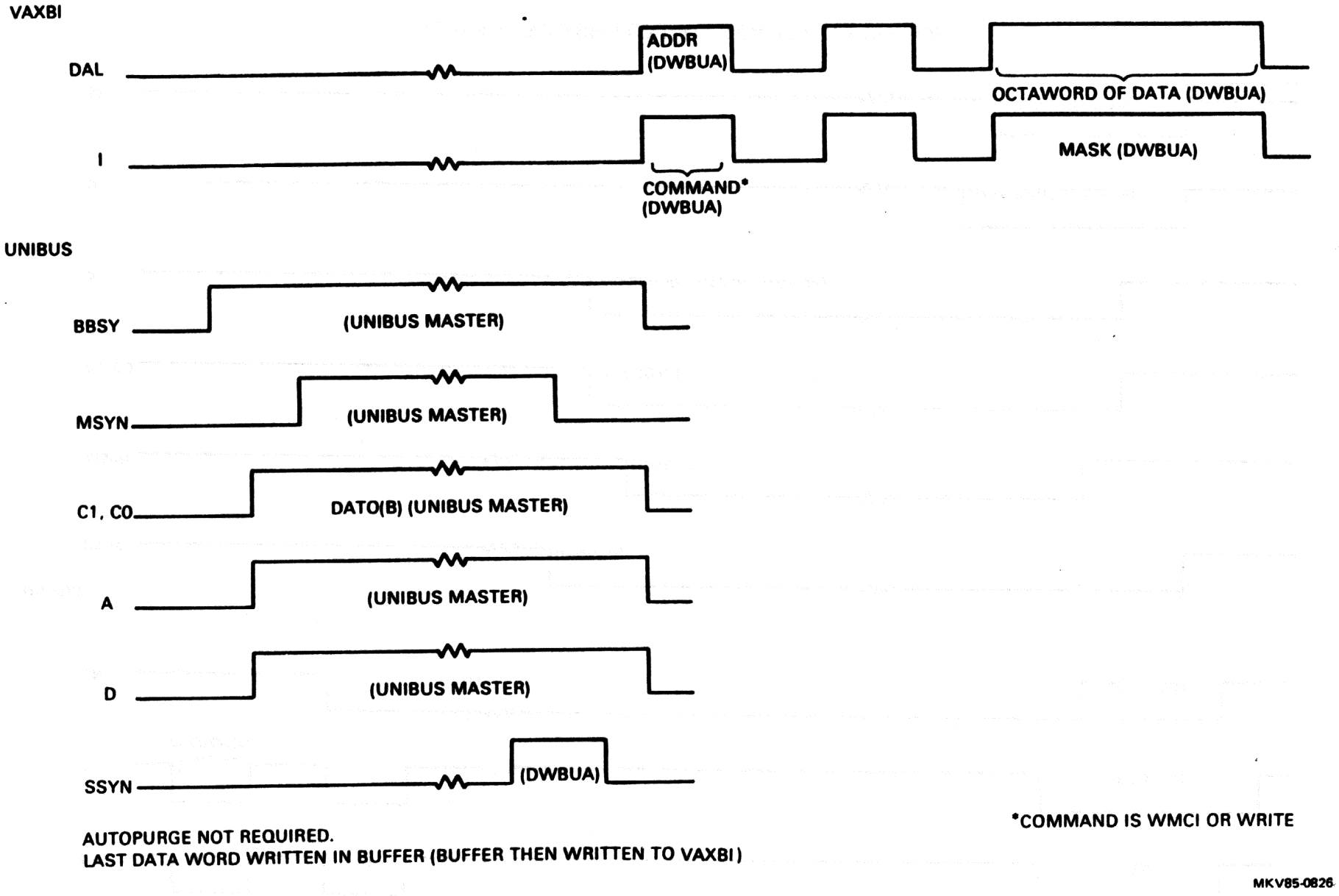


Figure 4-9 DATO(B) Through a Buffered Data Path Timing Diagram

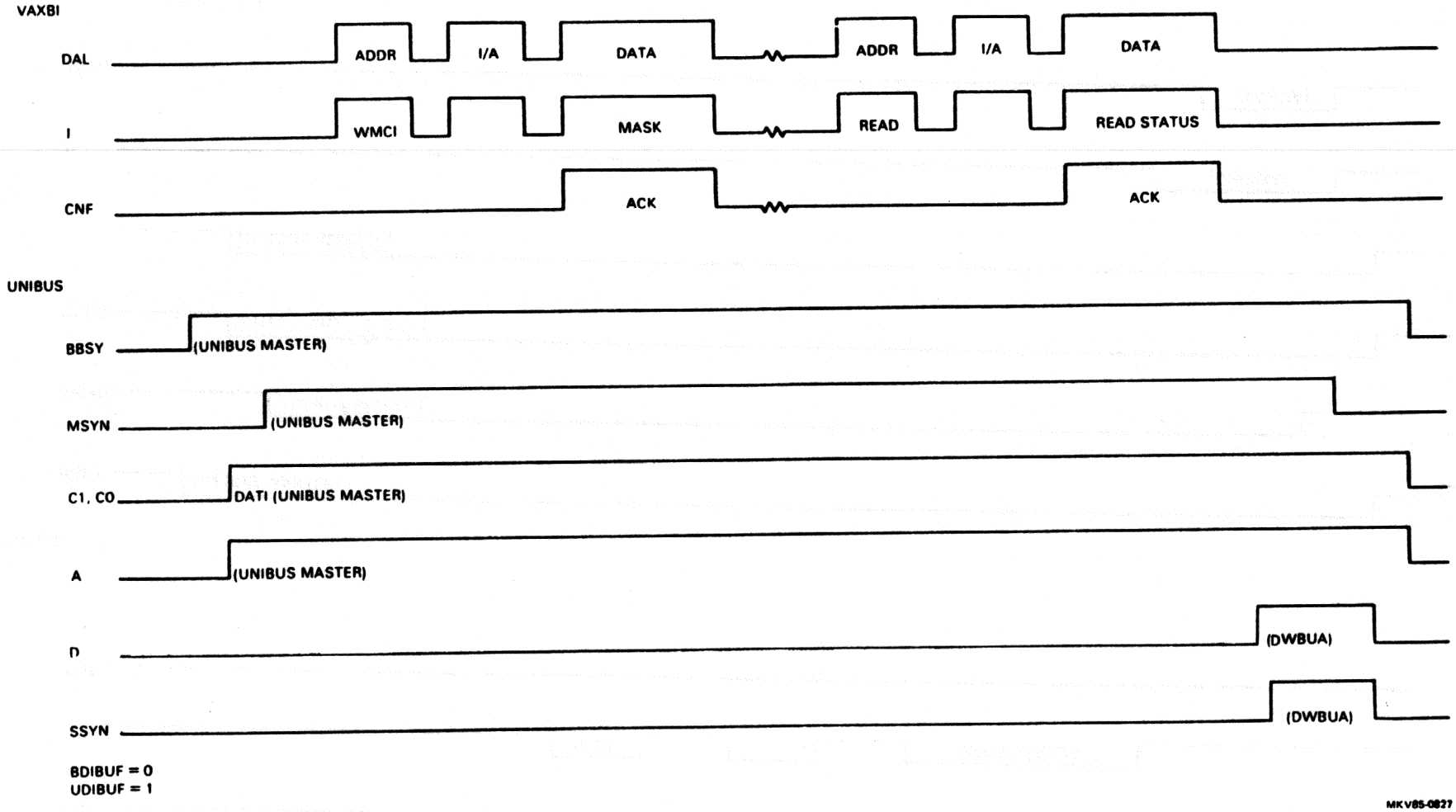


Figure 4-10 DATI Through BDP with Autopurge Timing Diagram

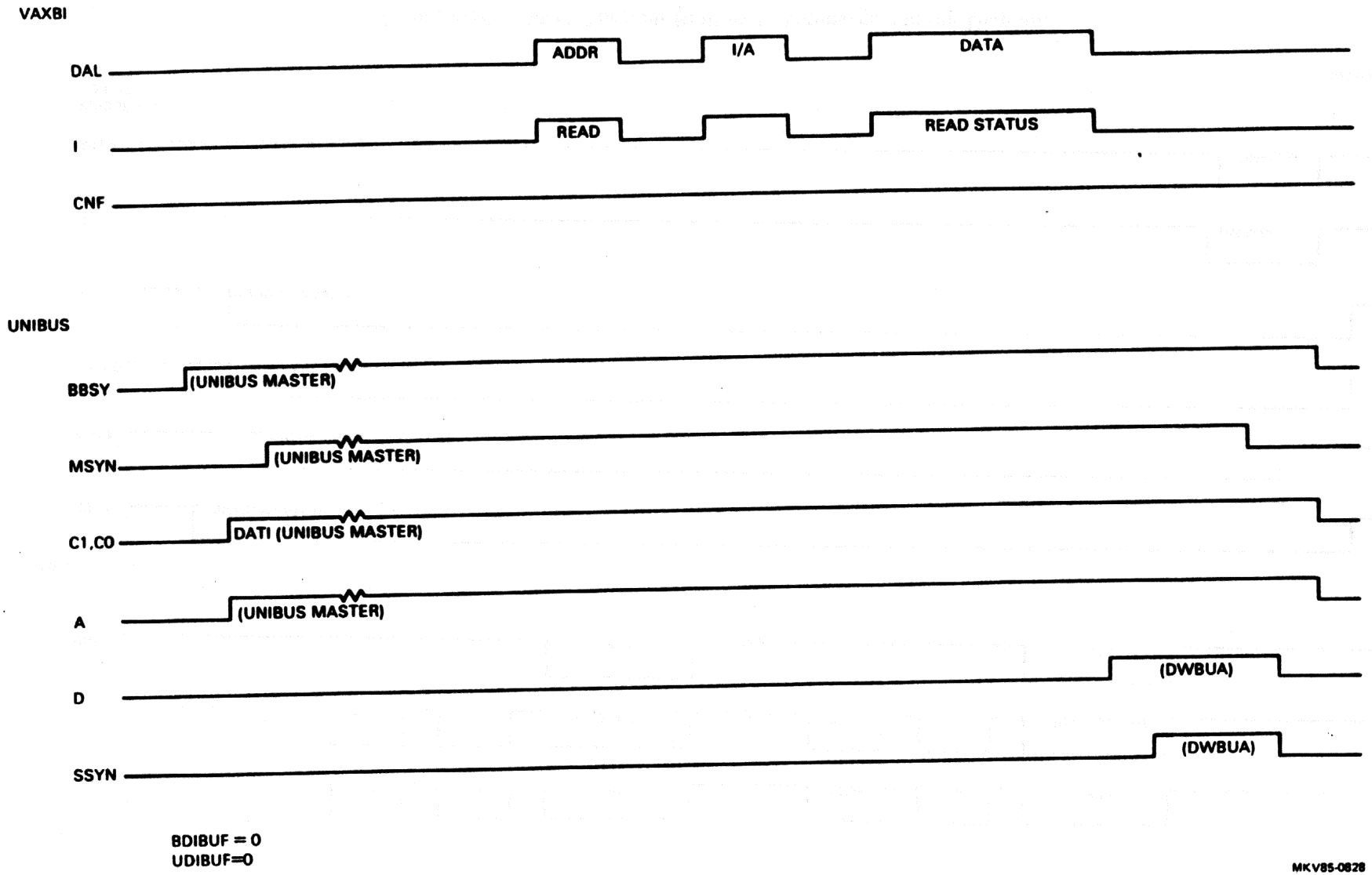


Figure 4-11 DATI Through BDP Timing Diagram

APPENDIX A
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 MYSN after using a BRn to arbitrate for the UNIBUS may not work satisfactorily.

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

APPENDIX B
GLOSSARY

ACK – Acknowledge. As a VAXBI command response, ACK indicates that the VAXBI slave acknowledges that it is capable of executing the command at this time. As a VAXBI data response, ACK indicates that no error has been detected and that the cycle is not to be STALLed.

AUTOPURGE – The act of writing the contents of a partially filled DWBUA BDP buffer to the VAXBI. A buffer is autopurged when it is partially filled with UNIBUS data and one of the following occurs: a DATI is requested through the same BDP; or a DATO(B) is requested and its address is not within the same octaword as the data currently stored in the buffer. Data is written from the buffer using a VAXBI octaword WMCI command with the prestored mask bits set for each valid data byte.

BAD – DWBUA Buffered Address.

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

bb – Base address.

BBSY – Bus busy; a UNIBUS signal. This signal is sent by the bus master to all other bus devices to indicate that the bus is in use.

BCI – VAXBI Chip Interface. This is a synchronous interface bus that provides for all communication between the BIIC and the DWBUA.

BDCST – 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 – DWBUA Buffered Data Path.

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

BUACSR – DWBUA Control and Status Register; a DWBUA internal register.

DATI – Data In; a UNIBUS command. This command requests a transfer of data from the UNIBUS slave to the UNIBUS master. The transfer is always word length.

DATIP – Data In Pause; a UNIBUS command. DATIP is identical to DATI, except DATIP informs the UNIBUS slave that the present transfer is the first part of a read/modify/write cycle. DATIP must be followed by DATO(B) to the same word address.

DATO(B) – Data Out (Byte); a UNIBUS command. This command transfers a word (DATO) or byte (DATOB) of data from the UNIBUS master to the UNIBUS slave.

DDP – DWBUA Direct Data Path.

DMA – Direct Memory Access.

DPCSR – Data Path Control and Status Register; a DWBUA internal register.

DWBUA – VAXBI to UNIBUS Adapter.

FUBAR – Failed UNIBUS Address Register; a DWBUA internal register.

IDENT – Identify; a VAXBI command. This command is used by processors and other intelligent interrupt fielding nodes to solicit vector information.

INTERLOCK – A mechanism in the DWBUA that locks out transactions while waiting for a specific command. It is used when the DWBUA receives an IRCI from the VAXBI or a DATIP from the UNIBUS. It locks out all other transactions (except STOP from the VAXBI) until it receives the UWMCI or the DATO(B) that completes the current transaction.

INTR – Interrupt; a VAXBI command. This command signals interrupts to other nodes on the VAXBI.

INVAL – Invalidate; a VAXBI command. This command from 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 – DWBUA Internal RAM.

IRCI – Interlock READ with Cache Intent; 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; IRCI must be followed by UWMCI.

LWAEN – Longword Access Enable; a UNIBUS Map Register bit.

MBZ – Must Be Zero.

MSYN – Master Sync; a UNIBUS signal issued by the bus master and received by the bus slave. Assertion of MSYN requests the slave, defined by the UNIBUS address lines, to perform the function required by the UNIBUS control lines. Negation of MSYN indicates to the slave that the master considers the data transfer concluded.

NO ACK – No Acknowledge. As 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 in the transaction.

NODE – See VAXBI Node.

NODE 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 DWBUA adapter's node space holds the DWBUA registers.

OCTAWORD – Sixteen contiguous bytes starting on an arbitrary byte boundary.

PORT LOCK – This mechanism locks the VAXBI and UNIBUS ports while the DWBUA is servicing a transaction.

PURGE – The act of emptying a BDP buffer by setting the corresponding DPCSR PURGE bit.

RCI – READ with Cache Intent; 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 sent to it.

SACK – Selection Acknowledged; a UNIBUS signal. A device that has requested the bus, acknowledges that it has been granted the bus, and that it accepts.

SSYN – Slave Sync; a UNIBUS signal issued by the bus slave and received by the bus master. Assertion of SSYN informs the bus master that the slave has concluded its part of the current data transfer. Negation of SSYN informs all bus devices that the slave has concluded the current data transfer.

STALL – As a VAXBI command response, STALL indicates that the slave needs additional time to acknowledge the command, is not ready to return the first data word on a READ command or vector data on an IDENT 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 selectively forces nodes to a state in which they do not issue VAXBI transactions, yet they retain as much error information as possible.

UA – UNIBUS address.

UNIBUS – An asynchronous bus consisting of 56 lines.

UNIBUS ARBITRATOR – A logic circuit that compares priorities from devices requesting the use of the data section of the UNIBUS. The arbitrator determines which device will next be granted control of the UNIBUS. A UNIBUS must have one and only one arbitrator. For the configuration described in this manual, the DWBUA is always the UNIBUS arbitrator.

UWMCI – Unlock WRITE Mask with Cache Intent; a VAXBI command. This command completes a read/modify/write cycle that began with an IRCI command.

VAXBI – VAX Bus Interconnect. It joins a processor to a combination of devices that can include I/O controllers, I/O bus adapters, memories, and other processors. This is a double-clock, synchronously operated interconnect with bus events occurring at fixed intervals. Bus arbitration and address and data transmissions are time multiplexed over 32 data lines. Data transmission is at fixed lengths of 4, 8, or 16 bytes on naturally aligned addressing boundaries.

VAXBI NODE – An interface that occupies one of sixteen logical locations on a VAXBI. A VAXBI node consists of one or more VAXBI modules.

VOR – Vector Offset Register; a DWBUA internal register.

WCI – WRITE with Cache Intent; a VAXBI command. The master node writes data to the slave and alerts other nodes to issue a VAXBI INVAL, if necessary, for the address written.

WINDOW SPACE – A 256K byte block of I/O addresses. Each node, based on its node ID, is allocated a unique window space. The DWBUA adapter's window space holds the UNIBUS device registers and the UNIBUS memory space. The Starting Address Register and Ending Address Register must be set to enable this space.

WMCI – WRITE Mask with Cache Intent; a VAXBI command. This command is similar to WCI, except the master 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 C
SELF-TEST
MICRODIAGNOSTIC TESTS

The self-test microdiagnostic tests run in the order shown in Table C-1. Tests 1 through A check the DWBUA logic, tests B through D check the VAXBI port logic, and tests E through 12 check the VAXBI port logic and the UNIBUS and its port logic.

Table C-1 Self-Test Microdiagnostic Tests

Test Number	Test Name	Description
1	29116 RAM Test	Verifies addressability and data integrity of last 16 locations of address processor RAM space. (Locations are used for storage of constants and DWBUA register addresses.) Other locations are verified in a later test.
2	DWBUA BAD Bus and BAD Register Test	Verifies Buffered Address (BAD) Bus, BAD Register, and BAD Output Mux of Data Path Gate Array.
3	BDP/MAP IRAM March Test	Standard march test (1-0 data pattern) 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 high words of both the rotating and nonrotating 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 DWBUA can properly execute Buffered Data Path transactions by verifying correct storage of buffered 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	2910 Condition Code Test	Performs a branch test on all condition codes that are not tested in other parts of this self-test.
A	29116 Instruction Test	Verifies that address processor can execute all functional microcode instructions that are not otherwise executed during this self-test.

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

Test Number	Test Name	Description
B	Starting/Ending Address Registers Test	Performs VAXBI transactions. Writes to the BIIC Starting Address and Ending Address Registers with the range of UNIBUS window space: reads the node ID from the BIIC CSR, computes the corresponding values for the two registers, and then writes to each of these registers.
C	BDP Write Mask Test	Verifies the write mask flip-flops in the Data Path Gate Array. These flip-flops store the mask for a VAXBI octaword WRITE mask transaction. (This transaction is executed whenever a Buffered Data Path is purged.)
D	VAXBI WMCI/READ Test	Verifies that the DWBUA can perform VAXBI READ and WRITE transactions to the BIIC by performing word-length transactions on the VAXBI. (These operations are used by UNIBUS-to-VAXBI Direct Data Path transactions.) This test uses the BIIC General Purpose Registers, and it verifies that both word and byte length transactions are possible from the UNIBUS to the VAXBI.
E	UNIBUS DATO/DATI Test	Uses the UET module to verify that the DWBUA can write to and read from the UNIBUS. Performs a VAXBI WMCI instruction to set up the VAXBI Address Transceiver with the UET 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 UET module.
F	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/DATO(B) sequence functions properly on the UNIBUS. The test initially writes a known data pattern (AAAA hex) to the UET 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 UET Address Register to verify that the DATOB completed properly.

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

Test Number	Test Name	Description
10	UNIBUS DATI/DATO 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 UET 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 enabled through the UNIBUS Map Register. The test verifies that the correct UNIBUS Map Register was referenced by a microcode 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 unused 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 UET CSR to generate a UNIBUS request. The UET module is written and a UNIBUS BR is asserted. This causes the BIIC to initiate a VAXBI interrupt to the DWBUA. The DWBUA receives a VAXBI IDENT command at the level corresponding to the INTR that was issued. The test generates the IDENT command.

APPENDIX D
MACRODIAGNOSTIC TESTS

NOTE

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		BUA Control and Device Type Registers Test
	1	BUA Self-Test and Register Subtest
	2	BUA Revision and Device Type Subtest
2		BUA Registers Test
	1	VAXBI BER Read/Write Subtest
	2	VAXBI EICR Read Subtest
	3	VAXBI Interrupt Destination Register Read/Write Subtest
	4	BUA VOR 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 INTLK READ/UNLOCK WRITE Test
6		UNIBUS to VAXBI Addressing 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 Autopurge Test
15		Byte Offset DATI Test
16		Byte Offset DDP DATO Test
17		Byte Offset BDP DATO Test
18		Byte Offset DDP 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	BIF 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 Init Test
	1	UNIBUS Init Subtest
	2	VAXBI STOP Command Subtest
30		FUBAR Register Test
31		UBE Multi Transfer Test
32		UBE Block Transfer Test

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 EV <4:0> L	Mnemonic	DWBUA Response
H L H L L	IAL	The current IDENT command is ignored. The bus grant is withheld from the UNIBUS.
L H H H L	BPS	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	ICRSD	
L H H L L	BBE	
H H H L L	BTO	The BUACSR BIF 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	RDSR	
L L H H L	ICRMC	
L L H L H	NCRMC	
L L L H H	ICRMD	
L L L H L	RTO*	
L L L L H	BPM	
L L L L L	MTCE	

* The DWBUA receives this error EVENT code only if the RTOEVEN bit in the DWBUA adapter's BCICSR 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 values.) The following are the only legal mask values:

A1=0	00yy	yy≠[00]
A1=1	yy00	

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 response to the WRITE command is ACK. It then sets the USSTO bit in the BUACSR, issues an error interrupt if interrupts are enabled, and writes the UNIBUS address to the Failed UNIBUS Address Register (bb+728).

- The DWBUA sends zero data and an RDS status code in response to the READ command. It also sets the USSTO bit in the BUACSR, issues an error interrupt if interrupts are enabled, and writes the UNIBUS address to the Failed UNIBUS Address Register (bb+728).

E.1.5 Invalid VAXBI Command

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

- RESERVED (BCI I<3:0> = HHHH)
- INTR
- RESERVED (BCI I<3:0> = LHLH)
- RESERVED (BCI I<3:0> = LHLL)
- 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 internal 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 issue 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 BIF bit and writes the VAXBI address to the VAXBI Failed Address Register (bb+72C).

- DATO(B) through a Buffered Data Path

  The DWBUA issues 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 BIF bit and writes the VAXBI address to the VAXBI Failed Address Register (bb+72C).

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's VALID bit is clear). If this happens, the DWBUA asserts the BUACSR IMR bit, issues 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 6 or 7, the DWBUA asserts the BUACSR BADBDP bit, issues 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

DATO(B) must follow a DATIP, but if BBSY is interrupted during the DATO(B), the DWBUA asserts the BUACSR UIE bit and issues an error interrupt (if interrupts are enabled).

APPENDIX F
UNIBUS
EXERCISER TERMINATOR

F.1 UNIBUS EXERCISER TERMINATOR DESCRIPTION

The UNIBUS Exerciser Terminator (UET) (or M9313 module) is located in sections A and B of the last UNIBUS slot. The UET enables diagnostic testing of the DWBUA adapter's capabilities to handle UNIBUS addressing, data transfers, and interrupts.

F.2 UNIBUS EXERCISER TERMINATOR REGISTERS

Table F-1 UNIBUS Exerciser Terminator Registers

Register Address (octal)	Register Name/Bits	Notes
772140	Address Register A<15:00>	Word load only. Byte loading causes timeout.
772142	Data Register D<15:00>	Both byte and word loading allowed.
772144	Control Register CR<15:00>	Word load only. Byte loading causes timeout.

F.2.1 Control Register Format

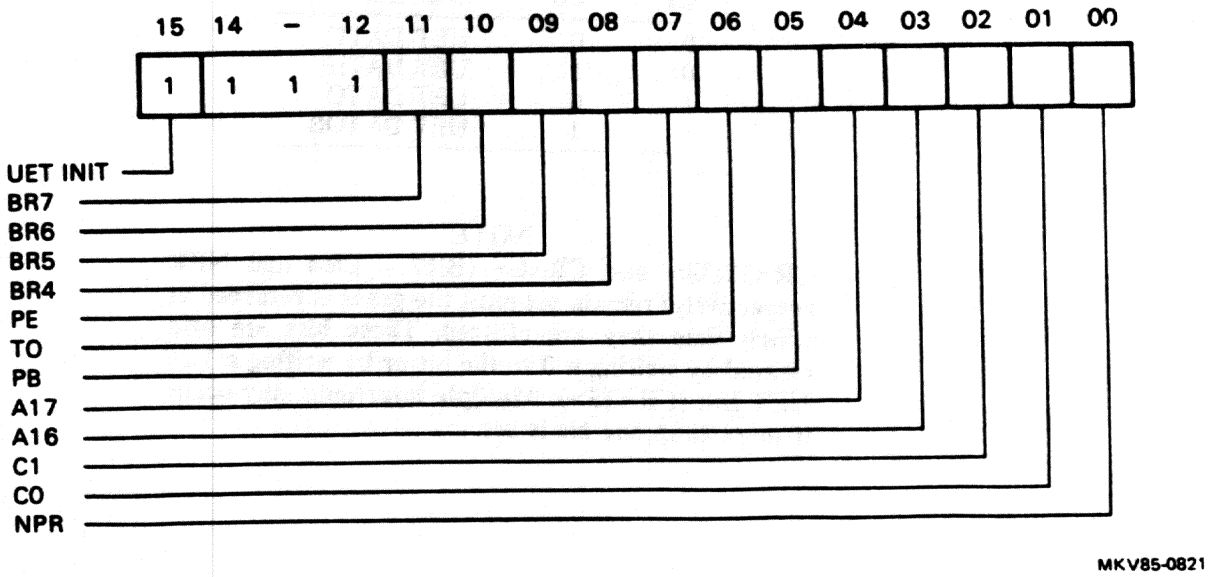


Figure F-1 UET Control Register Format

F.2.2 Control Register Bit Descriptions

UET Init CR<15>	Initialize UET to simulate reset or powerup. This WRITE-ONLY bit always reads 1. It does not clear CR<4,3>.
Unused CR<14:12>	Always read as 1.
BR7–BR4 CR<11:08>	Write 1 to initiate interrupt.
PE CR<7>	Parity Error detected during UET DATI. Clocked on each UET DATI and cleared by UET Init.
TO CR<6>	Timeout (SSYN not returned). Clocked on each transfer; cleared by UET Init.
PB CR<5>	Parity Bit. When set, the PB line will be asserted when the UET Data Register is read. This bit is cleared by UET Init.
A17, A16 CR<4:3>	High-order UNIBUS addressing bits.
C1, C0 CR<2:1>	Transfer command bits (see Table F-2)
NPR CR<0>	Write 1 to initiate transfer.

Table F-2 Transfer Command Bits

C1	C0	Command
0	0	UET DATI
0	1	UET DATIP
1	0	UET DATO
1	1	UET DATOB

NOTE

CR<11:08> and CR<0> (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 UET Init (CR<15>). Multiple interrupts may occur if more than one bit is set.

F.3 NPR DATA TRANSFERS

F.3.1 UET WRITE

A UET WRITE consists of the following sequence of events:

1. Load Address Register A<15:00>
2. Load Data Register D<15:00>
3. Load Control Register to initiate the transfer:

    a. CR<4:3> = A<17:16> of UNIBUS Address
    b. CR<2:1> = 10 for DATO, 11 for DATOB
    c. CR<0> = Generate NPR

F.3.2 UET READ

A UET READ consists of the following sequence of events:

1. Load Address Register A<15:00>
2. Load Data Register D<15:00>
3. Load Control Register to initiate the transfer:

    a. CR<4:3> = A<17:16> of UNIBUS Address
    b. CR<2:1> = 00 for DATI, 01 for DATIP
    c. CR<0> = Generate NPR

NOTE

The UET does not need a DATO(B) following a DATIP. After it has completed the DATIP, the UET drops BBSY and releases the UNIBUS.

F.4 BR INTERRUPTS

The following sequence of events implements a BR interrupt:

1. Load the Data Register D<15:00> with the vector address.

2. Load Control Register bits CR<11:08> with the BR (BR7 – BR4) level.

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	2000 0000	2000 1FFF	2040 0000	2043 FFFF
1	2000 2000	2000 3FFF	2044 0000	2047 FFFF
2	2000 4000	2000 5FFF	2048 0000	204B FFFF
3	2000 6000	2000 7FFF	204C 0000	204F FFFF
4	2000 8000	2000 9FFF	2050 0000	2053 FFFF
5	2000 A000	2000 BFFF	2054 0000	2057 FFFF
6	2000 C000	2000 DFFF	2058 0000	205B FFFF
7	2000 E000	2000 FFFF	205C 0000	205F FFFF
8	2001 0000	2001 1FFF	2060 0000	2063 FFFF
9	2001 2000	2001 3FFF	2064 0000	2067 FFFF
A	2001 4000	2001 5FFF	2068 0000	206B FFFF
B	2001 6000	2001 7FFF	206C 0000	206F FFFF
C	2001 8000	2001 9FFF	2070 0000	2073 FFFF
D	2001 A000	2001 BFFF	2074 0000	2077 FFFF
E	2001 C000	2001 DFFF	2078 0000	207B FFFF
F	2001 E000	2001 FFFF	207C 0000	207F FFFF

APPENDIX H
REGISTER
INITIAL STATES

The initial state of each register is its state after successful completion of the BIIC and DWBUA self-tests.

Table H-1 Register Initial States

Address (bb+)	Register	Initial State	Notes
00	Device Type	xxxx0102	xxxx = DWBUA revision
04	VAXBI Control and Status	xx01280y	xx = VAXBI interface revision y = DWBUA node ID (hex)
08	Bus Error	00000000	
0C	Error Interrupt Control	00000000	
10	Interrupt Destination	0000xxxx	xxxx = decoded DWBUA node ID (one bit set)
14	IPINTR Mask	xxxx0000	xxxx = IPINTR mask
18	Force IPINTR/ STOP Destination	0000xxxx	xxxx = force IPINTR/STOP destination
1C	IPINTR Source	xxxx0000	xxxx = IPINTR source
20	Starting Address	xxxx0000	xxxx = starting address of DWBUA adapter's window space (between 2040 and 207C, last digit 0, 4, 8, or C)
24	Ending Address	xxxx0000	xxxx = starting address of window space after DWBUA (between 2044 and 2080, last digit 0, 4, 8, or C)
28	BCI Control	00002900	STOPEN, IDENTEN, and UCSREN bits set
2C	Write Status	10000000	
30	Force IPINTR/ STOP Command	00001800	
40	User Interface Interrupt Control	00008000	
F0	GPR 0	00000001	UBPUP = 1; self-test passed

Table H-1 Register Initial States (Cont)

Address (bb+)	Register	Initial State	Notes
F4–FC	GPR 1-3	00000000	
720	DWBUA Control and Status	00008000	
724	Vector Offset	00000000	
728	Failed UNIBUS Address	00000000	
72C	VAXBI Failed Address	00000000	
730–740	Microdiagnostic	00000000	
750	DPCSR 0	00000000	DPCSR is Data Path Control and Status Register
754	DPCSR 1	00200000	
758	DPCSR 2	00400000	
75C	DPCSR 3	00600000	
760	DPCSR 4	00800000	
746	DPCSR 5	00A00000	
800–FBC	UNIBUS Map	00000000	Initially invalid
FC0–FFC	UNIBUS Map	FFFFFFFF	I/O space addresses

APPENDIX I
DATA PATH OPERATION

I.1 DIRECT DATA PATH

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

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

CASE 1 – DATO WITH UNIBUS ADDRESS BIT <01> SET

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

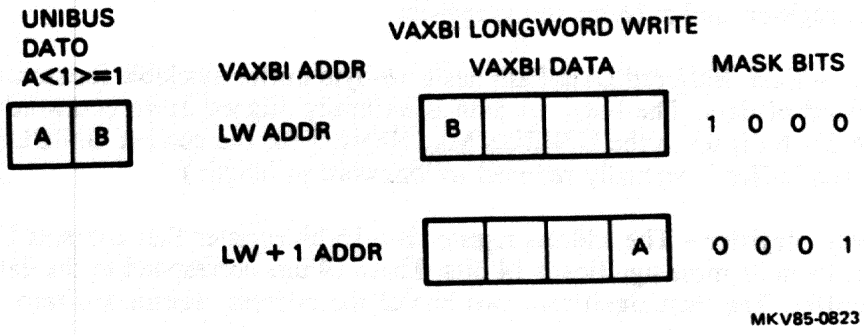


Figure I-1 DATO with UNIBUS Address Bit <01> Set

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 I-2.

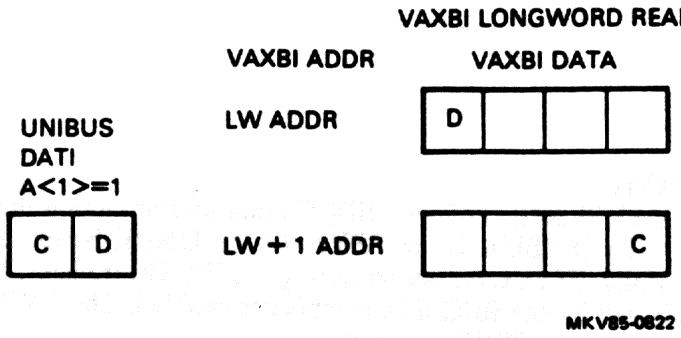


Figure I-2 DATI with UNIBUS Address Bit <01> Set

For improved UNIBUS bandwidth, the DWBUA completes the corresponding UNIBUS DATO or DATOB transaction (by issuing SSYN) prior to performing the VAXBI WMCI transfer. The DWBUA does not issue SSYN 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.

I.2 BUFFERED DATA PATH

The DWBUA has five Buffered Data Paths (BDP). 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 LWAEN 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:

   BDIBUF - VAXBI Data in Buffer
   UDIBUF - UNIBUS Data in Buffer
   STRT_0 - Start Zero

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

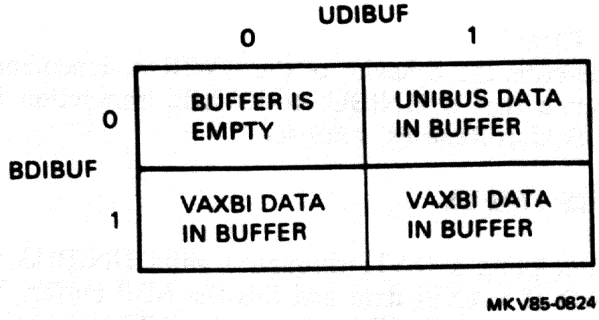


Figure I-3 BDIBUF and UDIBUF Flags

When UDIBUF is set, the DWBUA also updates the STRT__0 flag. The STRT__0 flag indicates that the first transaction through this Buffered Data Path began at an aligned octaword address (UA <3:0> = 0000). When the last byte in the buffer is written, the DWBUA tests 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 (nonmasked) 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.

I.2.1 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 LWAEN 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 LWAEN 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 though the Buffered Data Path.

   b. A DATO(B) 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 prestored 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*.

I.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 octaword READ of VAXBI data and fills the BDP buffer. The DWBUA then places the requested data on the UNIBUS, issues SSYN, updates the BDP flags by setting BDIBUF and clearing UDIBUF, 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 UDIBUF and clearing BDIBUF, stores the incoming UNIBUS address, issues SSYN, 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 autopurged. Once the autopurge 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 autopurged. Once the autopurge is complete, the DWBUA issues SSYN. 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 SSYN, 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 SSYN is issued.

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

I.2.3 BYTE OFFSET Bit Set

When the UNIBUS Map Register BYTE OFFSET bit is set, the DWBUA services requests in 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<3:0> = 1110 (BYTE OFFSET BIT IS SET)

The address is incremented so that A<3:0> = 1111. A word length transaction to 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 last byte of the octaword. If the addresses do not match, the DWBUA requests a VAXBI octaword READ. When this READ transaction is complete, the DWBUA temporarily stores the last 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 at an octaword level, 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 assured. When the READ transaction is complete, the first byte of the second 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 SSYN is issued.

   If the BDP buffer contains UNIBUS data, the DWBUA treats the transaction in a special way. The DWBUA does not autopurge the buffer as it does in a non-byte offset transaction. Instead, the DWBUA READs a longword of VAXBI data through the Direct Data Path. This longword of data contains the low byte of the requested word, which is stored internally. Next, the DWBUA maps the next longword of data, which falls in the next octaword boundary in 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 assure data integrity. The DWBUA then READs the longword and fetches the high byte 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 SSYN. Thus, in this 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 last byte of the buffer, and a Write-to-VAXBI is performed. This generates an octaword WMCI 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 autopurged. When the autopurge 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 UDIBUF and clearing BDIBUF, the incoming address is incremented to the next octaword and stored, and SSYN is issued. The BDP is left in the state it would be if a DATOB had been performed to an octaword-aligned address with no byte offset.

CASE 2 - UNIBUS ADDRESS A<1:0> = 10 (BYTE OFFSET AND LWAEN BITS ARE SET)

This case is handled similarly to CASE 1, the non-LWAEN case. 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 octaword boundaries.

I.2.4 Examples

Figures I-4 through I-8 show the contents of a BDP buffer for various multiple DATO(B) transactions. Each valid byte of data in the buffer has its corresponding mask bits set; all other mask bits for the BDP buffer are clear.

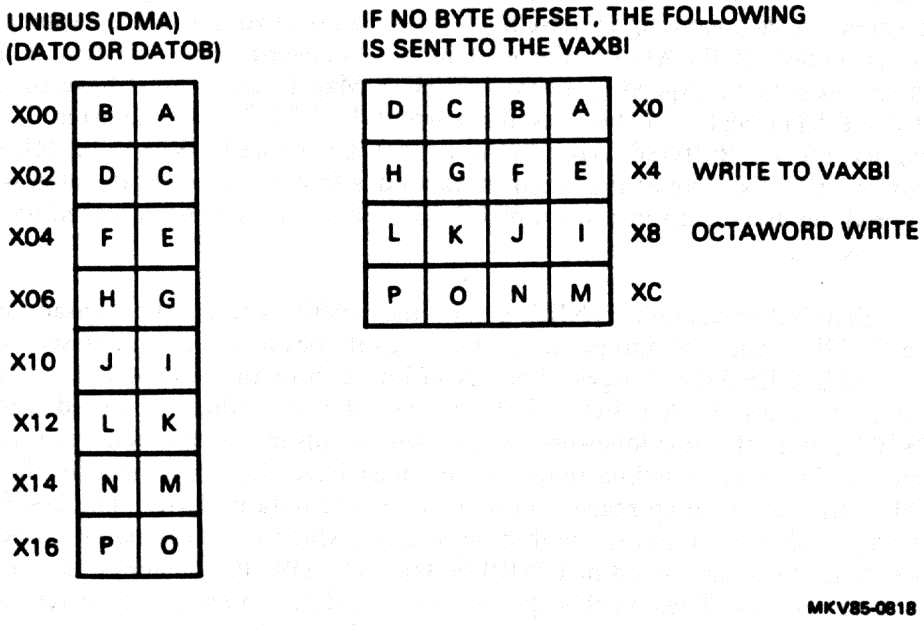


Figure I-4 DATO(B) Through BDP, BYTE OFFSET Clear, Starting at Octaword Boundary

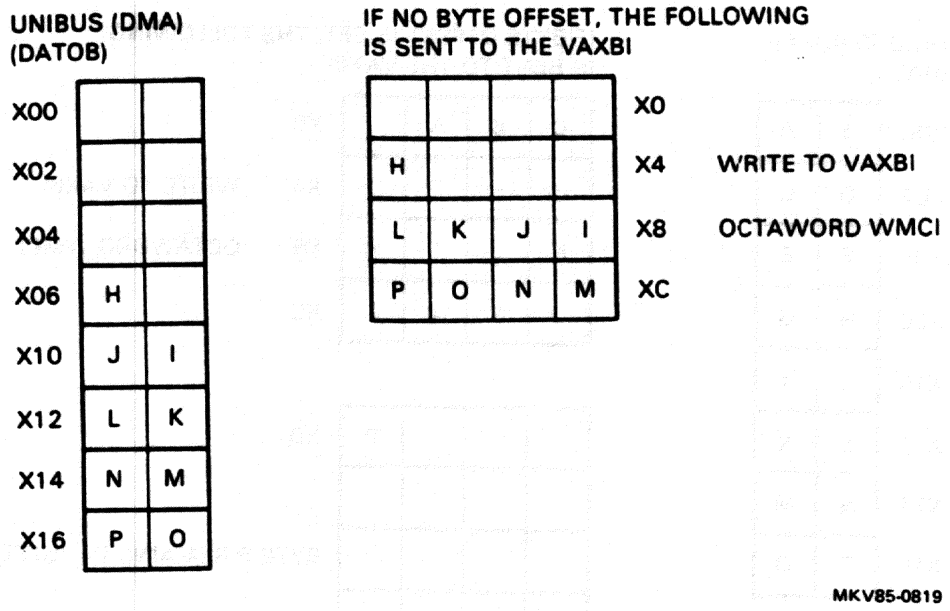


Figure I-5 DATOB Through BDP, BYTE OFFSET Clear, Starting at Byte 8

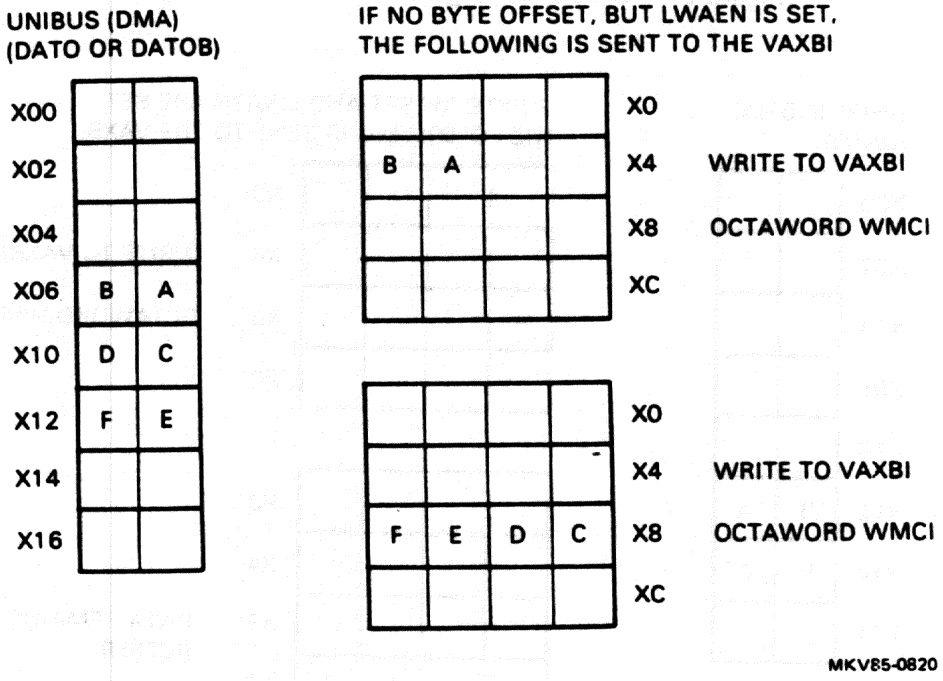


Figure I-6 DATO(B) Through BDP, BYTE OFFSET Clear, LWAEN Set

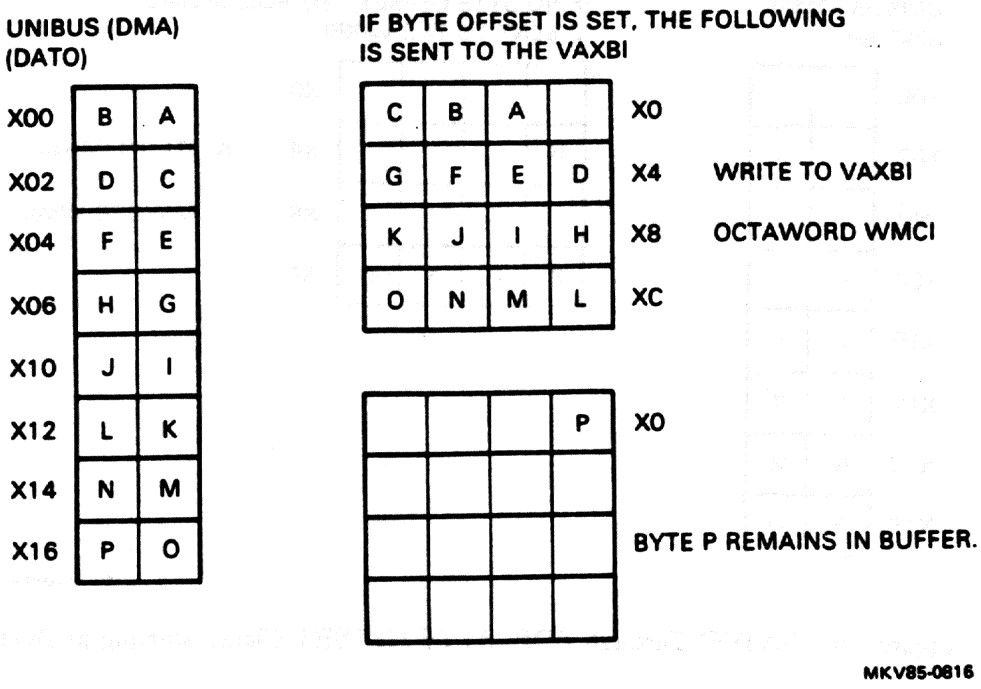


Figure I-7 DATO Through BDP, BYTE OFFSET Set

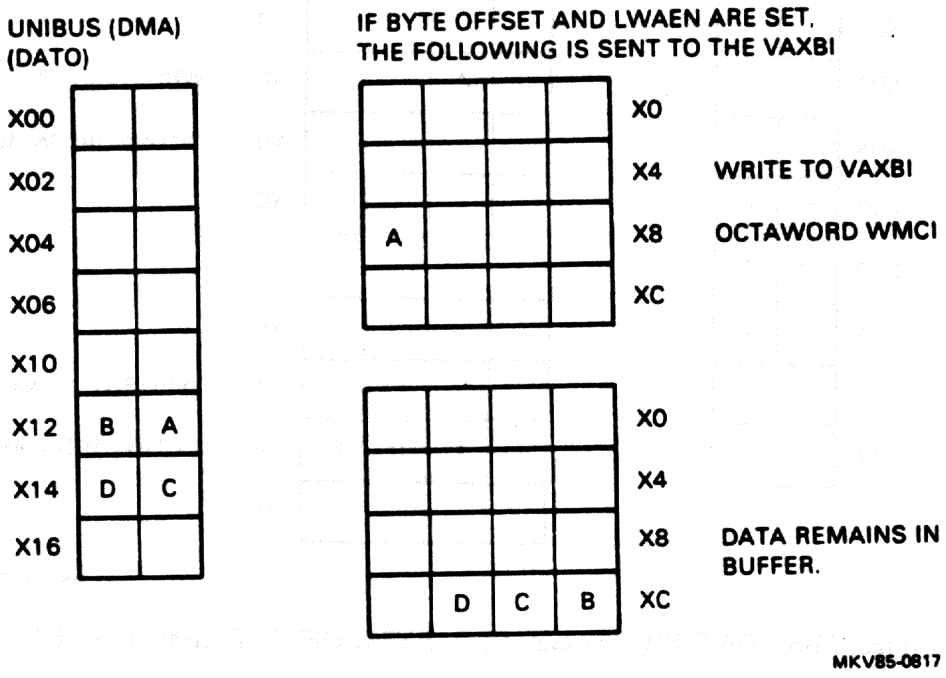


Figure I-8 DATO Through BDP, BYTE OFFSET and LWAEN Set

APPENDIX J
PORT LOCK, RETRY, AND INTERRUPT MECHANISMS

J.1 PORT LOCK MECHANISM

The DWBUA has a port lock mechanism to ensure that it processes only one transaction at a time. This mechanism locks the VAXBI and UNIBUS ports from accepting new transactions until the DWBUA is able to service another request. While the DWBUA is locked, it sends RETRY to all valid incoming VAXBI transactions except the STOP command. (The DWBUA immediately sends an ACK response to the STOP command.) The DWBUA also disables its UNIBUS arbitrator from issuing grants to UNIBUS devices while it is locked.

The DWBUA is locked during the following four occurences and sends a RETRY response to all VAXBI transactions sent to it.

1. The DWBUA has accepted a VAXBI transaction; the lock is released when the DWBUA has completed servicing the transaction. If, however, the VAXBI transaction is an IRCI, the DWBUA sends a RETRY response to all transactions until a UWMCI command is sent to the DWBUA.

2. A UNIBUS DMA request is granted. The lock is released when UNIBUS BBSY is negated by the UNIBUS master.

3. The DWBUA issues a VAXBI transaction (such as autopurge or the forcing of an error interrupt on the VAXBI). The port lock is released when this transaction is completed.

4. The DWBUA is the VAXBI master; the slave VAXBI node responds to its transaction with a RETRY. The DWBUA then sends a RETRY response to all subsequent incoming 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 DWBUA to VAXBI master nodes. It works by disabling and enabling the UNIBUS arbitrator.

When the DWBUA sends a RETRY response to a valid VAXBI command that it would otherwise accept, the DWBUA also disables its UNIBUS arbitrator. The UNIBUS arbitrator is enabled again when the DWBUA, 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 DWBUA should keep requesting the DWBUA until it receives an ACK response. In this way, the VAXBI node ensures that its transaction will be the next one serviced by the DWBUA.

J.3 UNIBUS INTERRUPTS

Interrupts are permitted only from the UNIBUS to the VAXBI.

J.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 (bb+10).

Figure J-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 J-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 straight-line path through the flows.
3. No time scale is employed. The diagram indicates relative timing only.

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

J.3.2 Passive Release

When some UNIBUS devices become bus master under BR-BG transactions, they drop BBSY 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.

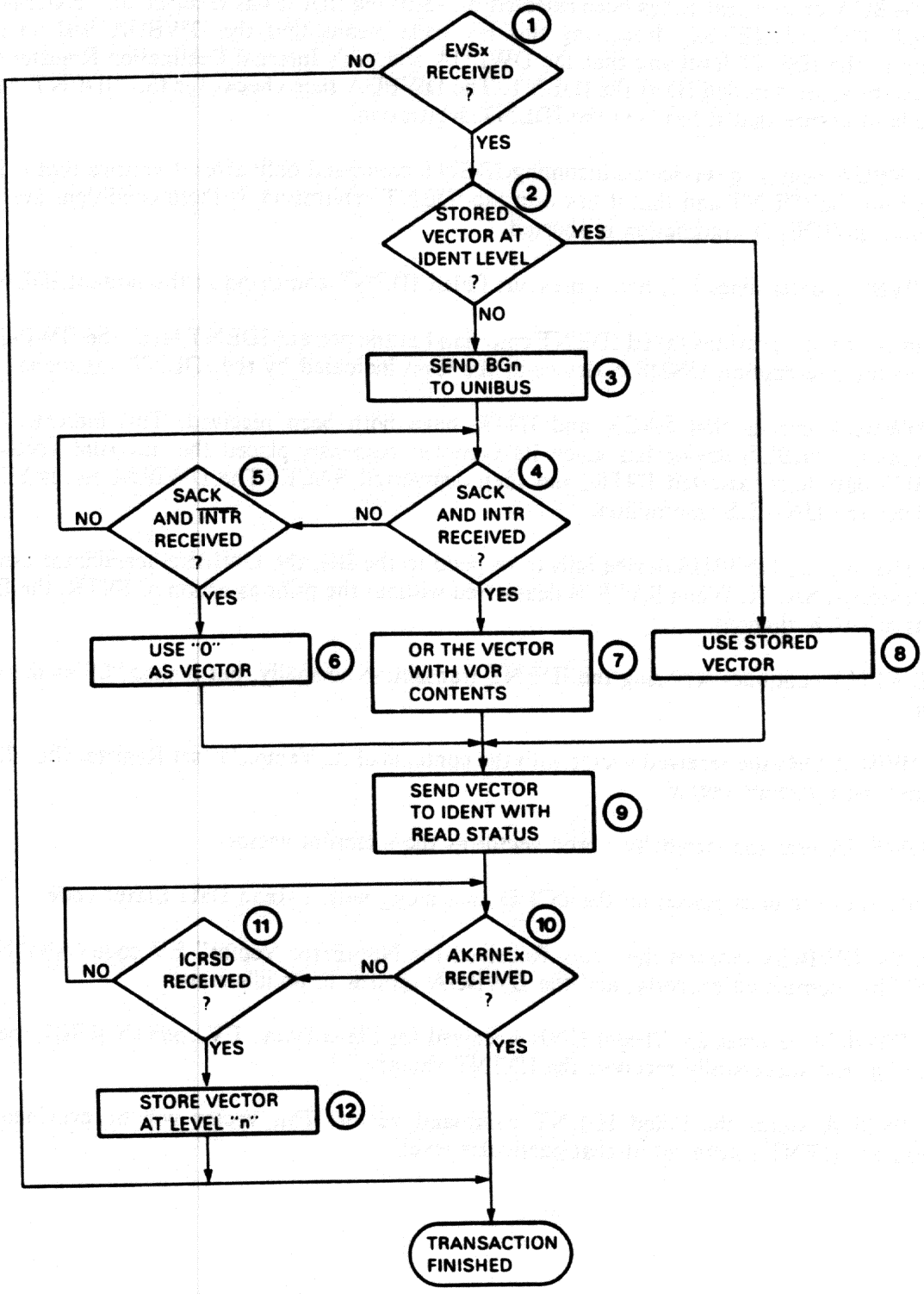


Figure J-1 IDENT Flow Diagram

① The DWBUA 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 DWBUA had an interrupt pending at the IDENT level and that the DWBUA adapter's Internal Destination Register (bb+10) contains the same decoded ID as the IDENT. The DWBUA then checks for the "IDENT Arb Lost" EV code to ensure that it has won the IDENT arbitration.

The DWBUA 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 DWBUA 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 DWBUA issues a BG to the interrupting UNIBUS device at the level indicated by the IDENT command.

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

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

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

⑦ The DWBUA ORs the received vector with the contents of its Vector Offset Register (bb+724). This becomes the interrupt vector.

⑧ The DWBUA uses the internally stored vector as the interrupt vector.

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

⑩ When the DWBUA receives the "Ack Received for Non-Error Vector" EV code (AKRNEx), the IDENT has completed properly, and the DWBUA returns to its idle state.

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

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

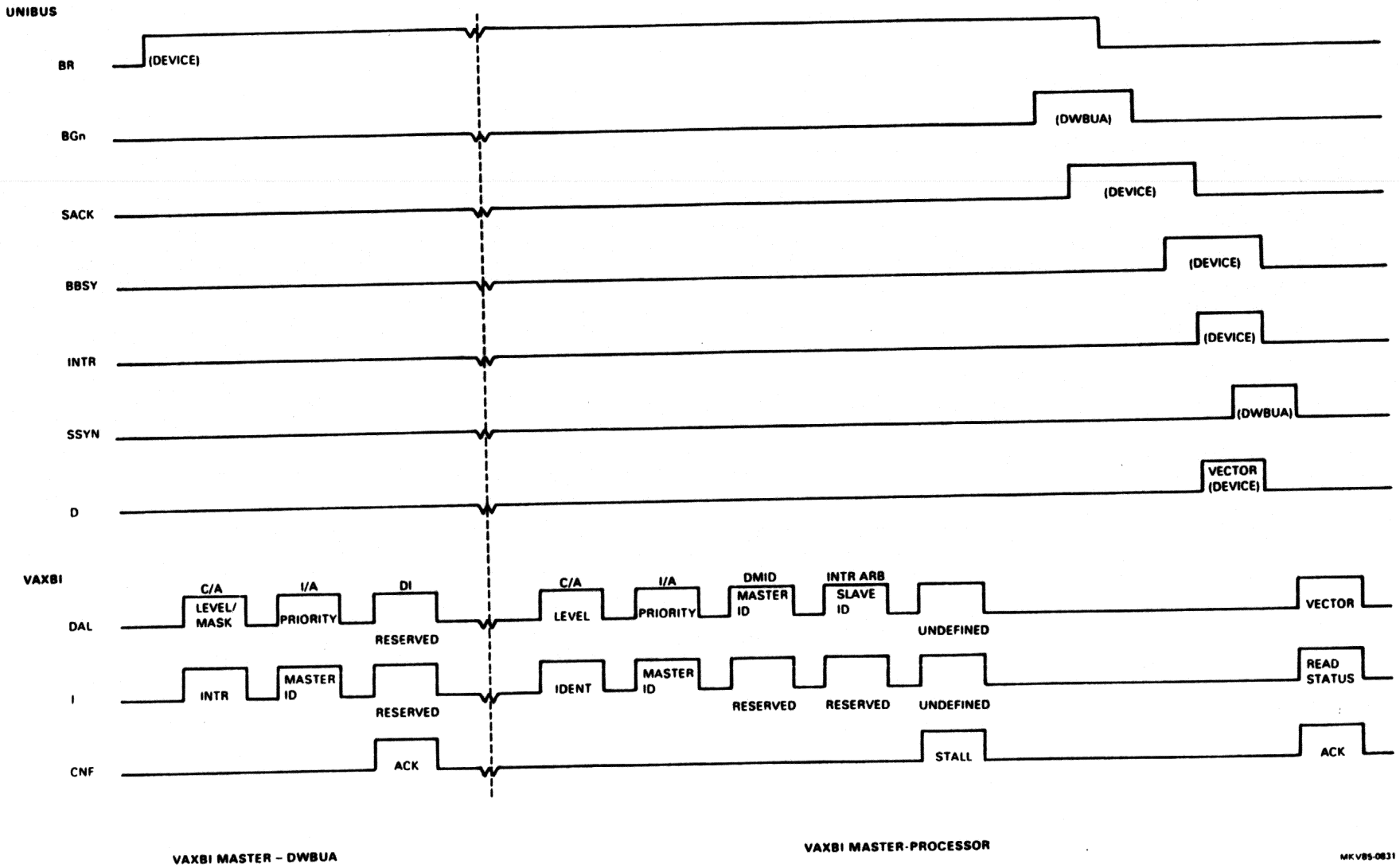


Figure J-2 Interrupt/IDENT Timing Diagram

APPENDIX K
MSYN-SSYN
TIME INTERVALS

NOTE

The MSYN – SSYN time intervals listed in Table K-1 may change with enhancements to the DWBUA option. The times listed were valid at the time of publication of this manual.

The following conventions are used in Table K-1.

- All transactions listed in brackets ([]) are performed after SSYN is issued to the UNIBUS device.
- Out-data means that the BDP buffer contains the UNIBUS DATO data to be sent to the VAXBI.
- In-data means that the BDP buffer contains the UNIBUS DATI data received from the VAXBI.

Table K-1 MSYN – SSYN Time Intervals

Path	Byte Offset Bit	UNIBUS Command	UNIBUS Address <3:0>	Data Path Status	VAXBI Transaction	Max Time MSYN – SSYN (μs)	See Note
DDP	0	DATI	ANY	N/A	LW READ	2.2	
DDP	0	DATIP	ANY	N/A	LW IRCI	2.2	1
DDP	0	DATO	ANY	N/A	LW WMCI or LW UWMCI	2.2	2
DDP	0	DATOB	ANY	N/A	LW WMCI or LW UWMCI	2.2	
DDP	1	DATI	ANY	N/A	LW READ, LW READ if A 1=1	5.5	3
DDP	1	DATIP	ANY	N/A	N/A	SSYN timeout	
DDP	1	DATO	ANY	N/A	LW WMCI, LW WMCI if A 1=1	6.0	3
DDP	1	DATOB	ANY	N/A	LW WMCI	4.0	
BDP	0	DATI	ANY	Empty	OW READ	4.1	4
BDP	0	DATI	ANY	In-data & match	No BI transaction	1.0	
BDP	0	DATI	ANY	In-data, no match	OW READ	4.1	
BDP	0	DATI	ANY	Out-data	OW WMCI, OW READ	7.6	
BDP	1	DATI	0 to C	Empty	OW READ	4.6	
BDP	1	DATI	0 to C	In-data & match	No BI transaction	2.6	
BDP	1	DATI	0 to C	In-data, no match	OW READ	4.7	
BDP	1	DATI	0 to C	Out-data	OW WMCI, OW READ	9.0	
BDP	1	DATI	E	Empty	OW READ, OW READ	9.2	
BDP	1	DATI	E	In-data & match	OW READ	6.1	
BDP	1	DATI	E	In-data, no match	OW READ, OW READ	9.2	
BDP	1	DATI	E	Out-data	LW READ, LW READ	7.2	5
BDP	X	DATIP	ANY	N/A	N/A	SSYN timeout	
BDP	0	DATO	ANY	Empty or in-data	[OW WMCI]	1.2	7
BDP	0	DATO	ANY	Out-data & match	[OW WMCI]	0.9	
BDP	0	DATO	ANY	Out-data, no match	OW WMCI, [OW WMCI]	5.2	
BDP	1	DATO	0 to C	Empty or in-data	No BI transaction	2.3	
BDP	1	DATO	0 to C	Out-data & match	No BI transaction	2.3	
BDP	1	DATO	0 to C	Out-data, no match	OW WMCI	6.2	
BDP	1	DATO	E	Empty or in-data	OW WMCI	6.3	
BDP	1	DATO	E	Out-data & match	OW WMCI	6.3	
BDP	1	DATO	E	Out-data, no match	OW WMCI, OW WMCI	10.1	
BDP	0	DATOB	ANY	Empty or in-data	[OW WMCI]	1.2	
BDP	0	DATOB	ANY	Out-data & match	[OW WMCI]	1.0	
BDP	0	DATOB	ANY	Out-data, no match	OW WMCI, [OW WMCI]	5.0	
BDP	1	DATOB	ANY	Empty or in-data	[OW WMCI]	2.3	
BDP	1	DATOB	ANY	Out-data & match	[OW WMCI]	2.4	
BDP	1	DATOB	ANY	Out-data, no match	OW WMCI, [OW WMCI]	6.1	

NOTES FOR TABLE K-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 OFFSET bit set, the UNIBUS command is ignored and the DWBUA does not issue SSYN (causing an SSYN timeout). During this time, all VAXBI transactions to the DWBUA receive a RETRY response until the UNIBUS device negates BBSY.

If a DATIP command is not followed by a DATO(B), the DWBUA sets the UIE bit of the BUACSR and forces an error interrupt, if interrupts are enabled.

(2) A UNIBUS DATO(B) through the Direct Data Path translates to a longword WMCI transaction with the mask bits set for each valid data byte.

(3) The DWBUA performs two longword transactions on the VAXBI for a word length transfer through the Direct Data Path if both the BYTE OFFSET bit and UNIBUS address bit A<1> are set.

(4) A UNIBUS DATI command through a Buffered Data Path results in an octaword READ of VAXBI space, if the requested data is not in the BDP buffer. If the UNIBUS data is stored in the BDP buffer, however, the buffer must be purged by performing an octaword WMCI on the VAXBI before reading the data from the VAXBI. The entire octaword is loaded into the buffer; subsequent accesses within the octaword through the same Buffered Data Path cause the DWBUA to fetch the data from the buffer, with no VAXBI transaction requested.

(5) This special case is treated differently from other Buffered Data Path transfers to avoid delay in issuing SSYN. In this case, the low byte of the requested DATI word is fetched by performing a longword READ through the Direct Data Path. The high byte is fetched from either the current BDP buffer or the VAXBI with a longword READ through the DDP. The current BDP status remains unchanged during this transaction.

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

(7) Data for a DATO(B) command through a BDP is stored until the buffer is full. The DWBUA then performs a VAXBI octaword WRITE (nonmasked) if the buffer contains an entire octaword of valid data from the UNIBUS device. A VAXBI octaword WMCI is performed if the buffer contains less than a complete octaword of valid data.

APPENDIX L
DWBUA PARITY CHECKING

L.1 PARITY CHECKING

The DWBUA uses its 32-bit internal RAM as an internal storage source. When this RAM is updated, the newer version 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-initiated transfer, which requires 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 newer version of the DWBUA, which is capable of detecting parity errors, the DWBUA Control and Status Register contains these two additional bits:

PARITY ERROR <29>	(W1C, DCLOC)	This bit is set if the DWBUA found a parity error while reading its internal RAM.
PARITY DISABLE <21>	(R/W, DCLOC)	While this bit is set, the DWBUA does not generate any parity when writing to its internal RAM. This causes a parity error when the same IRAM 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 asserts the PARITY ERROR bit in the BUACSR and sends an error interrupt to the VAXBI bus if interrupts are enabled. The following sections describe the other effects of a parity error detection during various operations.

L.3.1 Parity Errors on UNIBUS Map Registers

A DWBUA UNIBUS Map Register enables mapping of an 18-bit UNIBUS address to the corresponding 32-bit 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 actual targeted address. Hence, the DWBUA does not initiate the corresponding VAXBI transfer if it detects a parity error while 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 WRITEs or READs data during a DATO(B) or DATI transfer through a Buffered Data Path. If a parity error occurs while the UNIBUS Map Register is being read, the DWBUA does not start the VAXBI transfer. The DWBUA withholds SSYN to the UNIBUS, if SSYN has not been previously issued, causing an SSYN timeout to the UNIBUS device. The DWBUA then reports a parity error to the VAXBI. The DWBUA also clears the UDIBUF, BDIBUF, and STRT_0 flags for the current Buffered Data Path, indicating that the BDP buffer is empty.

L.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 autopurged, the DWBUA initiates an octaword WRITE (WMCI) 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 read of any 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 clean. The DWBUA may withhold SSYN if it has not been previously issued, 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.

L.3.3 Parity Errors on Vector Registers

The DWBUA may receive an ICRSD event code when it ships vector data from the UNIBUS during an IDENT command. When this happens, the DWBUA stores the failed vector in its internal RAM. Subsequently, the next time the DWBUA receives an IDENT command at the same level, the DWBUA uses the stored data as the vector. If the DWBUA detects a parity error during the reading of the failed 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 L.3.

L.3.4 Parity Errors on DWBUA Internal Registers

When the VAXBI master device READs (RCI, IRCI) 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 while reading the data from the internal RAM, it sends zero data, a READ DATA SUBSTITUTE Read Status code, and an ACK response. The DWBUA then reports a parity error as specified in Section L.3.

L.4 PARITY LOGIC TESTING

The DWBUA self-test tests most of the logic associated with parity generation and detection. It forces bad parity for each location of the internal RAM and ensures that the DWBUA detects a parity error. The DWBUA cannot, however, perform an octaword WRITE (WMCI) transfer on the VAXBI during its self-test. This means that the logic that detects a parity error during one of the four data cycles cannot be verified.

The PARITY DISABLE bit in the BUACSR is a special bit for diagnostic purposes. It disables the parity generation logic. Assertion of PARITY DISABLE allows the DWBUA Level 3 diagnostic, EVCBB, to force a parity error and to verify that the DWBUA detects the error. EVCBB forces a parity error in each of the functions listed under Section L.3 and verifies that the DWBUA responds appropriately. EVCBB also forces a parity error during the octaword WRITE (WMCI) command on one of the longwords of data and verifies that the DWBUA has detected the parity error and reported the error appropriately.

INDEX

L

M

N

O

P

Q

R

V

W

READER'S COMMENTS

1. **How did you use this manual? (Circle your response.)**

   (a) Installation (c) Maintenance (e) Training
   (b) Operation/use (d) Programming (f) Other (Please Specify.) _____

2. **Did the manual meet your needs? Yes ☐ No ☐ Why?** _____
   _____

3. **Please rate the manual on the following categories. (Circle your responses.)**

	Excellent	Good	Fair	Poor	Unacceptable
Accuracy	5	4	3	2	1
Clarity	5	4	3	2	1
Completeness	5	4	3	2	1
Table of Contents, Index	5	4	3	2	1
Illustrations, examples	5	4	3	2	1
Overall ease of use	5	4	3	2	1

4. **What things did you like *most* about this manual?** _____
   _____

5. **What things did you like *least* about this manual?** _____
   _____

6. **Please list and describe any errors you found in the manual.**

Page	Description/Location of Error
_____	_____
_____	_____
_____	_____
_____	_____

7. **Which of the following most clearly describes your job? (Circle your response.)**

   (a) Administrative Support (e) System Manager (i) Educational/Trainer
   (b) Manager/Supervisor (f) Computer Operator (j) Sales/Marketing
   (c) Scientist/Engineer (g) Software Support (k) Other (Please specify)
   (d) Programmer/Analyst (h) Hardware Support _____

OPTIONAL INFORMATION

Name _____ Street _____
Company _____ City/State _____
Department _____ Country _____ Postal (ZIP) code (_____)
Job Title _____ Telephone Number _____

THANK YOU FOR YOUR COMMENTS AND SUGGESTIONS

Please do not use this form to order manuals. Contact your representative at Digital Equipment Corporation or (in the USA) call our DECdirect ™ department at this toll–free number: 800–344–4825.

FOLD HERE AND TAPE. DO NOT STAPLE

digital™

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 33 MAYNARD, MA

POSTAGE WILL BE PAID BY ADDRESSEE

DIGITAL EQUIPMENT CORPORATION
Educational Services/Quality Assurance
12 Crosby Drive BU0/E08
Bedford, MA 01730-1493

FOLD HERE AND TAPE. DO NOT STAPLE