UNITED STATES DEPARTMENT OF THE INTERIOR

GEOLOGICAL SURVEY

# Modifications to the VAX/VMS DR11-W/DRV11-WA Interface Driver for High Speed, Real-Time Data Acquisition

Lawrence M. Baker[1]

Open-File Report 87-642

[1]Menlo Park, California 94025

1987

# E R R A T A

United States Geological Survey
Open-File Report No. 87-642

Modifications to the VAX/VMS DR11-W/DRV11-WA
Interface Driver for High Speed,
Real-Time Data Acquisition

by

Lawrence M. Baker

1. Page B-6, line 52 (the second machine instruction after the label DO_READ_WRITE:) -- Change BBC to BBS:

```
BBC     #IO$V_WORD,R1,290$     ; Only matters if block mode
```

to

```
BBS     #IO$V_WORD,R1,290$     ; Only matters if block mode
```

Abstract

This report describes changes made to XADRIVER (version X-6), the VAX/VMS device driver for the UNIBUS DR11-W and Q-bus DRV11-WA parallel DMA interfaces. Enhancements include preallocation of bus adapter resources, driver execution at elevated fork IPL, and elimination of faulty driver behavior due to incorrect I/O requests. The result is improved driver performance for high speed, real-time data acquisition, especially on machines with a heavy timesharing load. Care has been taken to preserve the previous driver interface to allow the execution of most existing programs without requiring alterations.

## 1 Introduction

This report describes modifications developed by the USGS to XADRIVER (version X-6), the device driver supplied with the Digital Equipment Corp. (DEC) VAX/VMS and MicroVMS operating systems for the UNIBUS DR11-W and Q-bus DRV11-WA parallel DMA interfaces. These modifications improve the driver's suitability for high speed, real-time data acquisition by reducing the time between a device interrupt request and the start of the next data transfer. The program interface to the previous driver has been preserved for correctly behaved programs (see Section 4.2), enabling them to execute unmodified with the new driver.

The same driver is used with both VMS and MicroVMS. Although no further mention of MicroVMS is made in this report, references to VMS apply equally well to MicroVMS.

This report consists of seven sections and two appendices:

The information in Sections 1 through 6 will be useful to all readers. Application programmers may wish to browse through the appendices for a technical description of the modifications. System programmers and system managers should read the description of the modifications in Appendix A before applying the driver update in Appendix B.

## 2 Requirements for High Speed, Real-Time Data Acquisition

A distinguishing aspect of real-time systems is the requirement to bound the time it takes to service program requests, especially data input and output (I/O) requests. This is a consequence of the requirement in many real-time systems to perform actions based on the data values sampled from one or more external signals.

Sampled data values have finite useful lifetimes. To prevent incorrect actions based on stale information, the program variables storing these values must be refreshed before they have become invalid. (This is particularly critical for the proper detection of process or equipment failure and the initiation of safety-related recovery procedures.) A real-time operating system must provide features that the application may use to synchronize data sampling with the external system being measured (whether sampling is triggerred or periodic), and must complete these I/O requests within time limits acceptable to the application.

Acceptable response times in high speed data acquisition systems are typically very small, since the data values remain valid for a very short period of time. (For example, the system for which these changes were made can tolerate at most a 40 ms delay between the end of one DMA transfer and the start of the next. Other USGS applications restrict this time to less than 4 ms.) On a system dedicated to one application, this may be of little concern. However, if the application requires very small response times, it is important to assess the potential sources of delay and minimize the effect of those which adversely impact the real-time portions of the application.

Any time there is contention for shared resources, the possibility exists that a process will be blocked an indefinite amount of time while waiting for one or more resources to become available. In pathological cases, resource starvation can occur, which can lead to a failure of the entire application.* In addition, the overhead required to manage the allocation and deallocation of shared resources introduces additional delays which are undesireable in high speed, real-time systems.

Application programs can control to a certain extent the time it takes to allocate shared resources on demand (such as memory or processor cycles) with appropriate operating system calls (assuming adequate privileges and quotas). Often, shareable resources can be preallocated in advance, and retained until they are actually needed, thereby eliminating the allocation delay time altogether. However,

* Competition need not come from other users. For example, any process that causes I/O activity is a potential competitor for the resources required to service its I/O requests. Such activities may be an unavoidable part of the real-time application itself, such as the process responsible for storing data on disk or tape in a data acquisition system.

few mechanisms are typically available to control the allocation of resources that are shared within the operating system itself.

The modifications described in this report provide application programs with the ability to control the allocation of these internal operating system resources when using a DR11-W or DRV11-WA parallel DMA interface. With suitable privilege, a program can permanently allocate (or deallocate) the data channel resources required to perform DMA transfers to a particular device, thereby eliminating the allocation delay time when these resources are actually needed. As a byproduct of this preallocation, the device driver is permitted to raise its execution priority relative to the other device drivers in the system to further reduce the delay in servicing subsequent data transfer requests.

## 3 Summary of Modifications

In the discussion which follows, the reader is assumed to be familiar with the material in Chapter 3 of the VAX/VMS I/O User's Reference Manual: Part II, "DR11-W and DRV11-WA Interface Driver", and to be generally familiar with the resources provided by VAX bus adapters, i.e., buffered data paths and I/O mapping registers.

Actions described as being performed by the driver refer implicitly to the particular driver fork process executing on behalf of a single DR11-W or DRV11-WA interface. Actions performed by the driver on behalf of one DR11-W or DRV11-WA have no effect on the others in the system, except to the extent they are affected by changes in the pool of available bus adapter resources (as are all the other devices sharing the same bus adapter).

### 3.1 Performance Enhancements

A new modifier to the IO$_SETCHAR I/O function has been implemented, IO$M_NOWAIT, which invokes the features described below to reduce the latency between successive data transfers (i.e., the time between a device interrupt request and the start of the next data transfer):

1. To minimize delays caused by contention with other devices on the same UNIBUS or Q-bus adapter, the driver

   o optionally allocates a buffered data path and locks it for the exclusive use of the device (direct data paths are always shareable), and
   o allocates and locks a sufficient number of UNIBUS or Q-bus I/O mapping registers for transfer sizes up to the device buffer size.

2. To minimize delays caused by contention with other executive processes executing at the same Interrupt Priority Level (IPL), the driver

   o raises it's fork IPL to the highest fork IPL available.

The bus adapter resources are allocated after any changes to the data path and device buffer size requested in the same IO$_SETCHAR call. They remain locked until explicitly unlocked and deallocated by an IO$_SETCHAR call without the IO$M_NOWAIT modifier. This allows the system manager to assign resources to devices in the system startup command procedure. Users of those devices will benefit from the performance improvements, without requiring any special privileges, and without introducing a risk to the security of the system.

## 3.2 Bug Fixes

The driver has been modified to protect the system from faulty I/O requests and to correct several minor errors:

1. The Unit Control Block (UCB) device type field, UCB$B_DEVTYPE, is used to distinguish between the UNIBUS DR11-W and the Q-bus DRV11-WA. The driver sets the appropriate value in this field (either DT$_DR11W or DT$_XA_DRV11WA) when the device is initialized, and modifies it whenever an IO$_SETCHAR I/O function call is issued.

   The original driver uses the value in UCB$_DEVTYPE at various points to select the proper code path to follow. Since any value for this field is allowed in an IO$_SETCHAR call, the system can become corrupted and behave erratically or crash if an incorrect value is supplied. (Correctly behaved programs first issue an IO$_SENSECHAR to read the previous device characteristics, and preserve any unmodified fields before issuing the IO$_SETCHAR call.) Since the potential for such an error did not exist until support was added to the driver for the DRV11-WA, there may be many programs that do not correctly preserve the device type field which used to work fine, but will eventually cause a system failure.

   To ensure correct operation, the modified driver uses a separate mechanism to select the proper code paths that does not depend on the value in the UCB device type field.

2. Several other minor bug fixes were made to the original driver that do not affect normal device operations. These include:

   o Correctly copy the I/O adapter mapping registers into the error logging packets.

- Correctly transfer byte-aligned buffers.
- Preserve the data path selection when the driver is reloaded.
- Correctly call the device reset routine from the device initialization routine.
- Use instructions with the correct operand context when accessing the I/O page.
- Correctly handle a DRV11-WA in 18-bit mode.
- Restart a DMA transfer if power fails during the device setup sequence.

## 3.3 Optional Enhancements

Several enhancements were made to the original driver that are not necessary to obtain the performance benefits described in this report. These include:

- Cosmetic changes to the driver, such as correcting comments and removing unneeded extended local symbol blocks.
- Minor code optimizations.
- Changes to the driver behavior, such as leaving the device offline instead of BUGCHECKing, and guaranteeing the serialization of all I/O requests.

## 4 Documentation Changes

The following modifications should be made to Chapter 3 of the VAX/VMS I/O User's Reference Manual: Part II, "DR11-W and DRV11-WA Interface Driver":

1. Section 3.3, Table 3-4 -

    - IO$_SETCHAR - Add the IO$M_NOWAIT I/O function modifier.
    - IO$_SENSEMODE (no parameters; logical I/O function) - Returns the device-dependent characteristics (UCB$L_DEVDEPEND) in the second longword of the I/O status block (IOSB).
    - IO$_SENSECHAR (no parameters; physical I/O function) - Same as IO$_SENSEMODE.

2. Section 3.3, "P2" - If I/O mapping registers are locked (see section 3.3.3.1), the maximum block-mode transfer size is the device buffer size in effect (UCB$W_DEVBUFSIZ) when the IO$_SETCHAR!IO$M_NOWAIT call was made (see Figure 3-2). SS$_IVBUFLEN is returned in the I/O status block (IOSB) if the transfer size exceeds the maximum size.

3. Section 3.3.3, Figure 3-2 - Bits 16..31 in the first longword of the P1 device characteristics buffer specify the device buffer size.

4. Section 3.3.3.1 - Add the IO$M_NOWAIT I/O function modifier to preallocate bus adapter resources:

   If IO$M_NOWAIT is set:

   1. If XA$M_DATAPATH is set, a buffered data path is allocated and locked; and
   2. a sufficient number of I/O mapping registers are allocated and locked to accomodate DMA transfers up to the device buffer size.

   Allocation requests are made after first updating the data path selection and device buffer size. If all locking is successful, the device fork IPL is raised to the highest level allowed (11) to minimize the delay while VMS services fork queue requests from higher priority devices.

   If IO$M_NOWAIT is clear:

   1. Any previously locked bus adapter resources are unlocked and released after first lowering the fork IPL back to the default level (8).

## 5 Preliminary Results

The purpose of this report is only to describe the modifications made to XADRIVER; a detailed analysis of the performance gains resulting from these changes has not been made. However, preliminary results from two systems can be presented: a VAX-11/750 using a UNIBUS DR11-W and a MicroVAX II using a Q-bus DRV11-WA.

The application is the same in both cases: a high speed analog-to-digital (A/D) converter is connected to the DR11-W or DRV11-WA for seismic data acquisition and recording. The VAX-11/750 system is used both to digitize analog recordings and for general time-sharing and batch processing. (Even though this is not an "on-line" digitizing system, the application is real-time, since it is the analog tape transport that determines the incoming data rate, not the computer.) The maximum delay that can be tolerated between successive DMA transfers from the A/D converter is 40 ms. If it is exceeded, the digitizing application must be restarted at a point on the analog tape prior to the seismic event that was being digitized.

Using the original driver, such failures often occurred several times during a multi-hour digitizing session, which forced the technicians performing the digitizing to restart the application or, if the failures persisted, to delay working until the load on the VAX-11/750 was reduced. Using the modified driver, successive data transfers are routinely reissued every 10 ms, and there has never been a failure due to a missed data transfer, regardless of the load on the VAX-11/750.

The second system is a MicroVAX II currently being developed to digitize, detect, and record seismic events on-line. Using the original driver, delays between successive DMA transfers from the A/D converter of over 30 ms have been measured with an artificial load on the system. Using the modified driver and the same load, the delays have been reduced to no more than 6 ms. In addition, the minimum time between the completion of one data transfer and the start of the next was reduced from 2 ms to 1 ms with no load on the system.

## 6 References

1. VAX Hardware Handbook, Volume 1-1986, 1985 (Order No. EB-25949-46); Chapter 11, "VAX Input/Output Subsystems".

2. VAXBI Options Handbook, 1986 (Order No. EB-27271-46); Chapter 9, "DWBUA Adapter".

3. VAX Realtime User's Guide, November 1986 (Order No. EK-VAXRT-UG001).

4. VAX/VMS I/O User's Reference Manual: Part II, April 1986 (Order No. AA-Z601B-TE); Section 3, "DR11-W and DRV11-WA Interface Driver".

5. Writing a Device Driver for VAX/VMS, April 1986 (Order No. AA-Y511B-TE).

6. VAX/VMS SUMSLP Utility Reference Manual, September 1984 (Order No. AA-Z432A-TE).

## 7 Acknowledgements

Thanks are due to Bob Dollar and Tom Jackson for testing the new driver on their MicroVAX II and VAX-11/750, respectively, and to the reviewers, Gary Maxwell, Tim MacDonald, Bob Dollar, and Peter Johnson for wading through such dry prose.

This work was funded by U. S. Nuclear Regulatory Commission Interagency Agreement number RES-82-001.

# APPENDIX A

# XADRIVER Modifications

This appendix describes the modifications made to XADRIVER (version X-6). Appendix B contains a command procedure, XADRIVER.slp, with the source code to update the driver using the VAX/VMS SUMSLP editor utility. Each update command in XADRIVER.slp is identified by an audit trail, consisting of the initials "LMB", followed by a three digit sequence number, and corresponds to one of the sections below. The comments at the head of XADRIVER.slp identify the driver changes associated with each audit trail, and are used in shortened form as subsection headings in the sections below.

The reader is assumed to have experience reading Macro-32 assembly language, and to be generally familiar with the VMS I/O subsystem, VMS device drivers, VAX UNIBUS and Q-bus adapters, and DMA device interfaces for the UNIBUS and Q-bus.

## A.1 LMB001

LMB001 contains the bug fixes for the driver outside of the controller/unit initialization routine.

### A.1.1 Correct driver operation for invalid UCB$B_DEVTYPE

The original driver uses the value in the UCB device type field (UCB$B_DEVTYPE) for proper code path selection. Since IO$_SETCHAR calls modify UCB$B_DEVTYPE, unpredictable results can occur if an incorrect or illegal value is supplied. The driver was modified to use a single status bit in the low-order byte of the second device-dependent characteristics longword (UCB$L_DEVDEPND2) for the code selection, both to shorten the instruction sequence and to remove the dependence for correct operation of the driver on a field in the UCB which can be modified by a user program. (This byte is defined symbolically as UCB$B_DRV11WA.) Bit 0 of UCB$B_DRV11WA is used with the BLBC and BLBS instructions (Branch on Low Bit Clear/Set) as a replacement for the original CMPB...BEQL two-instruction sequences (Compare Byte...Branch on Equal).

## A.1.2 Correctly save previous map register contents

The original driver contains a typographical error that results in incorrect values stored in the error logging packet for the final and previous I/O mapping register contents (UCB$L_XA_FMPR and UCB$L_XA_PMPR) when more than one register is allocated. (The previous I/O mapping register contents are stored in the field for the final I/O mapping register, and the field for the previous I/O mapping register contains zero.) The faulty instruction was modified to reference UCB$L_XA_PMPR instead of UCB$L_XA_FMPR.

## A.1.3 Correctly transfer byte-aligned buffers

The original driver will not properly handle all data transfers that are byte-aligned in memory. Byte-aligned buffers are rejected for block-mode transfers when the direct data path has been selected by the Function Decision Table (FDT) routine handling reads and writes. The original driver incorrectly handles the following cases:

1. For block-mode transfers:

   o The data path selection bit (XA$M_DATAPATH) in the device-dependent characteristics (UCB$L_DEVDEPEND) is tested to determine which data path has been selected. However, since the test is performed in an FDT routine, there is no guarantee that the same data path will be selected when the transfer is actually performed. (E.g., an IO$_SETCHAR that is still in the device I/O request queue could change the data path selection.) This test must be properly serialized with other requests that have the potential for modifying the data path selection.

   o The XA$M_DATAPATH bit is not a reliable indicator of the current data path selection. XA$M_DATAPATH is modified by every IO$_SETMODE and IO$_SETCHAR call, whether or not the IO$M_DATAPATH modifier has been specified. Without the IO$M_DATAPATH modifier, XA$M_DATAPATH will have no effect on the selection of the data path, but the driver makes no effort to preserve this bit in UCB$L_DEVDEPEND when the IO$M_DATAPATH modifier is not specified.

   o The bus adapter may have no buffered data paths, as is the case for the Q-bus adapter on the MicroVAX II. The documentation correctly states that requests for the buffered data path are ignored on such processors, but it fails to reject byte-aligned buffers in that case.

2. For word-mode transfers:

   o The FDT routine allows byte-aligned buffers because the driver performs word-mode transfers to and from memory itself. When updating the user buffer pointers in the driver routines MOVEFRUSER and MOVETOUSER, the original driver uses an incorrect test when the word being transferred straddles a page boundary. The result is an incorrect data transfer at best, and corrupted memory at worst.

The simplest fix for these problems is to disallow byte-aligned buffers by moving the test for illegal byte-alignment ahead of the test for word-mode transfers (move line 463 immediately after line 458), and remove the test for buffered data path selection, since it is no longer needed (delete lines 461-462).

However, if the intention is to support byte-aligned buffers, the changes in Appendix B must be used. For block mode transfers, the test was moved from an FDT routine to the point in the driver where the data path is actually allocated. This is required to handle bus adapters that have no buffered data paths. (The test is performed in a way that is independent of the CPU type, so no modifications should be required to support future VAX processors.) For word mode transfers, the buffer pointer update logic in MOVETOUSER was corrected to properly detect page overflow of byte-aligned buffers.

### A.1.4 Restart DMA transfer if power fails during device setup

The original driver does not detect a power fail during the device setup sequence in XA_START. As Section 9.3.7 of Writing a Device Driver for VAX/VMS points out, if one occurs between the time the device registers are loaded and the CSR GO bit is set for a block-mode (DMA) transfer, the result is unpredictable device behavior. In the worst case, the word count and bus address registers will be cleared and a 64KW transfer to address 0 will be initiated (overwriting memory until the VMS power fail recovery routine can call the device initialization routine to stop the transfer). The driver was modified to detect a powerfail just before setting the CSR GO bit and branch back to XA_START in that case to retry the transfer.

## A.2 LMB002

LMB002 contains the bug fixes for the driver within the controller/unit initialization routine.

### A.2.1 Preserve the data path selection when the driver is reloaded

The original driver unconditionally locks the direct data path when the driver is reloaded, even if the device has been configured to share the buffered data paths (VEC$M_PATHLOCK in VEC$B_DATAPATH is clear). One difference between controller and unit initialization routines is that the controller initialization routine is called for both the initial driver load (or system boot), and for driver reloads. The unit initialization routine is only called for the initial load, not for reloads. To avoid disturbing the data path locking context in the CRB/VEC, the controller intialization routine in the original driver was converted to a unit initialization routine, which prevents execution of this code when the driver is reloaded.

### A.2.2 Correctly call device reset routine during initialization

The device reset routine, XA_DEV_RESET, expects R5 to contain the UCB address to access the device type field (UCB$B_DEVTYPE). However, when the original driver calls XA_DEV_RESET from the controller initialization routine, R5 contains the address of the IDB; R0 contains the UCB address. By converting the controller initialization routine to a unit initialization routine, VMS supplies the UCB address in R5 and the CSR address in R4, which are the register contents specified for entry into XA_DEV_RESET.

### A.2.3 Change I/O page access to word context

The original driver uses a BBS instruction (Branch on Bit Set) in the controller initialization routine to test the CSR interrupt enable bit. This instruction has longword context, which is illegal when accessing I/O page registers. This instruction was replaced with an equivalent two-instruction sequence that uses word context to access the I/O page.

### A.2.4 Validate 22-bit mode DRV11-WA on a MicroVAX II

The original driver assumes that a DRV11-WA is in 22-bit mode. The consequences of an incorrect assumption are possible system corruption and either erratic behavior or a crash. The driver was modified to determine whether a DRV11-WA is in 18- or 22-bit mode using the following differences between the two modes:

- o In 18-bit mode, the extended address bits in the CSR (XA_CSR$M_XBA in XA_CSR) are read/write, and there is no bus address extension register (XA_BAE); and
- o in 22-bit mode, the extended address bits in XA_CSR are read-only copies of the two low-order address bits in XA_BAE, and two successive accesses to the bus address register (XA_BAR) with no intervening references to another device register will access XA_BAR and XA_BAE, respectively.

Two successive instructions are used to clear XA_BAR and XA_BAE. An attempt is then made to set the extended address bits in XA_CSR. If the DRV11-WA is in 22-bit mode, clearing XA_BAE will clear the extended address bits in XA_CSR, and they will not be modified by the attempt to set them through XA_CSR. If the DRV11-WA is in 18-bit mode, clearing XA_BAE is superfluous, and the extended address bits in XA_CSR will be set.

The extended address bits in the CSR are tested to determine if they are set or cleared. If they have remained clear, the DRV11-WA is in 22-bit mode and execution continues. Otherwise, the DRV11-WA is in 18-bit mode, and the driver either BUGCHECKs or leaves the device offline, depending on whether the optional enhancements have been included in the driver update.

NOTE

> The optional enhancements are strongly recommended. If the BUGCHECK is left in the driver, the system will never successfully boot VMS as long as the DRV11-WA is incorrectly configured in 18-bit mode. Since there is no way to analyze the system dump file, the failure detected by the driver cannot be differentiated from any other type of failure that leaves the system unusable. With the optional enhancements in place, the driver leaves the unusable device controller off-line (which prevents its use), leaving the remainder of the system completely available.

## A.3 LMB003

The changes in LMB003 implement the performance enhancements to the driver.

### A.3.1 Lock selected data path and I/O mapping registers

If the data path is locked, REQDPR will return immediately.* This feature is used in the driver to select between the direct data path (data path zero) and a buffered data path. If the direct data path has been selected, it is locked to prevent a buffered data path from ever being allocated. If a buffered data path has been selected,

* The method for permanently allocating a buffered data path is documented in section 10.1.2 in Writing a Device Driver for VAX/VMS; the macro issued by a device driver to allocate a data path, REQDPR, is described in Appendix B; and the executive routine that performs the allocation, IOC$REQDATAP(NW), is described in Appendix C.

REQDPR will return one, if the bus adapter has buffered data paths, or the call will fail because the bus adapter has none, leaving zero in the data path field (VEC$B_DATAPATH). Since the direct data path is always shareable, the data path can be locked in either case (ignoring the failure if the bus adapter has no buffered data paths). The data path is locked by setting VEC$M_PATHLOCK in VEC$B_DATAPATH.

The device buffer size (UCB$W_DEVBUFSIZ) is used to calculate the number of I/O mapping registers to permanently allocate to the device.* Since data buffers are not necessarily page aligned, the number requested is UCB$W_DEVBUFSIZ, rounded up to pages, plus one to account for page spillover, plus the extra page required by LOADUBA to trap runaway transfers. The I/O mapping registers are locked by setting VEC$M_MAPLOCK in VEC$W_MAPREG. The current value in UCB$W_DEVBUFSIZ is saved in UCB$L_DEVDEPND2+2 for use later when validating the size of a transfer request. (This word is defined symbolically as UCB$W_MPRBUFSIZ.)

Since an IO$_SETCHAR call may alter the data path selection or the device buffer size, the driver must release any locked resources before modifying the UCB database. The VEC$M_MAPLOCK bit in VEC$W_MAPREG is used by the driver as an indicator that adapter resources have been locked. If it is clear, the code to release the locked resources is bypassed. If it is set, VEC$M_MAPLOCK is cleared and the I/O mapping registers are released.

The VEC$M_PATHLOCK bit in VEC$B_DATAPATH must be cleared to do the same for a locked buffered data path, but not if the direct path has been selected. The allocated data path number is examined to determine which data path is locked. If it is zero, i.e., the direct data path, then the code to release the data path is bypassed. Otherwise, VEC$M_PATHLOCK is cleared and the buffered data path is released.

Any changes to the UCB database are then made, followed by a determination of whether to preallocate the (possibly modified) device configuration.

Code was added to the new driver to restrict the size of a DMA transfer to guarantee the buffer can be mapped when I/O mapping registers are locked. The device buffer size in effect at the time the I/O mapping registers were locked (UCB$W_MPRBUFSIZ) is used as the limit to simplify the calculation, even though it is a conservative estimate of the maximum buffer size that could be mapped.

---

* The method for permanently allocating I/O mapping registers is documented in section 10.2.2 in Writing a Device Driver for VAX/VMS; the macro issued by a device driver to allocate I/O mapping registers, REQMPR, is described in Appendix B; and the executive routine that performs the allocation, IOC$ALOUBAMAP(N), is described in Appendix C.

## A.4 LMB004

The changes in LMB004 implement additional performance enhancements to the driver, especially for heavy timesharing or networking environments.

### A.4.1 Raise fork IPL when bus adapter resources are locked

All devices on the same UNIBUS or Q-bus adapter compete for the adapter's resources, e.g., buffered data paths and I/O mapping registers. Each adapter has a corresponding data structure in the executive, called an Adapter-Control Block (ADB), which describes the current state of these shared resources for allocation and deallocation requests. Access to shared executive databases are serialized in VMS through the use of a pseudo process context, the fork process, which has an associated priority level, called fork IPL (FIPL), which is analagous to the execution priority of a normal VMS process. VMS guarantees that a fork process cannot be preempted by another fork process at the same (or lower) level; the CPU hardware gives preference to fork processes executing at higher IPLs.

VMS device drivers perform all non-time-critical work as fork processes, including interrupt post-processing and initiation of the next I/O request. There is no limit to the amount of time a driver may spend executing as a fork process. Since fork processes executing at the same level are serialized, this can lead to a variable and indeterminate amount of time between succesive transfers.

Since they must share access to a single bus adapter database, all driver processes for devices on the same bus adapter must run at the same fork level. As a result of the changes introduced in LMB003, however, a device that has locked bus adapter resources is no longer competing for those resources, and therefore does not require synchronized access to the the bus adapter database as long as they remain locked.

The driver was modified to take advantage of this situation by raising it's fork IPL (UCB$B_FIPL, normally 8) to the highest level available (XA_TOP_FIPL), thereby raising it's execution priority to the highest level within all fork processes. (The modified driver symbolically defines XA_TOP_FIPL as 11.) The only other driver in a standard VMS system that normally executes at this level is the mailbox driver.

The current value in UCB$B_FIPL is saved in UCB$L_DEVDEPND2+1 before it is raised to XA_TOP_FIPL. (This byte is defined symbolically as UCB$B_DEFFIPL.) A corresponding modification to lower the fork IPL back to the value stored in UCB$B_DEFFIPL was made before unlocking any bus adapter resources to reestablish serialized access to the bus adapter database.

## A.5 LMB005

The changes in LMB005 are optional. They are not necessary to obtain the performance benefits described in this report. In general, they represent alternate choices for driver behavior that are preferable to the choices made in the original driver.

### A.5.1 Correct comments describing IO$_SETCHAR outputs

The original driver mispelled the symbolic name for the device-dependent characteristics bit indicating the choice of data path (XA$M_DATAPATH), and was missing the reference to the bit indicating link mode versus user device mode operation (XA$M_LINK).

### A.5.2 Leave DRV11-WA offline instead of BUGCHECKing

The original driver BUGCHECKs if it is running on a MicroVAX I. It is preferable to make the device unavailable, leaving the user with an otherwise fully functional system. This is done in the unit initialization routine by clearing the device on-line bit (UCB$M_ONLINE) in the device status longword (UCB$L_STS), and returning control to the executive. (The same code is used in the modified driver when an 18-bit mode DRV11-WA is identified.)

### A.5.3 Serialize IO$_SETMODE and IO$_SETCHAR with all requests

The original driver makes some attempts to serialize I/O requests, but it is inconsistent and only partially successful. Several changes were made to the driver to guarantee serialization of all I/O requests, not just requests to change the data path and data transfer requests.

The original driver processes requests to change device characteristics that do not specify data path selection in an FDT routine, instead of queueing them to the driver. Thus, a program that sequences several IO$_SETCHAR calls and waits on the last one could mistakenly assume they all had completed if the last one does not specify the IO$M_DATAPATH modifier.

The driver was modified to queue all IO$_SETMODE and IO$_SETCHAR requests to the driver to provide the normally assumed serial completion of I/O requests to the same device. A routine to handle IO$_SETMODE calls was added in XA_START that modifies the UCB device buffer size (UCB$W_DEVBUFSIZ) and the device-dependent characteristics (UCB$L_DEVDEPEND).

## A.5.4 Serialize IO$_SENSEMODE and IO$_SENSECHAR with all requests

The original driver supports two undocumented I/O functions: IO$_SENSEMODE and IO$_SENSECHAR. These return the device-dependent characteristics (UCB$L_DEVDEPEND) in the second longword of the I/O status block (IOSB). The original driver completes these in the executive FDT routine, EXE$SENSEMODE. By completing the request in an FDT routine, the driver runs the risk of returning incorrect values if a previously issued IO$_SETCHAR request is still in the device's I/O request queue.

The FDT table was modified to pass IO$_SENSEMODE and IO$_SENSECHAR requests to the driver's start I/O routine using EXE$ZEROPARM, and a routine was added in XA_START that returns the device-dependent characteristics (UCB$L_DEVDEPEND) in the second I/O status longword. This guarantees that such requests will be properly serialized with IO$_SETCHAR requests, and stale values will not be returned. (Programs can call the SYS$GETDVI(W) system service to obtain an immediate copy of the device-dependent characteristics, if serialization with previous I/O requests to the device is not required.)

## A.5.5 Remove unnecessary extended local symbol block

In the original driver, no references to local symbols are made outside of, or into, the extended local symbol block surrounding the XA_START and BLOCK_MODE routines. The driver was modified to remove the extended local symbol block to reduce the chance of an unintentional reference to a distant statement label by causing such an error to generate an undefined reference.

## A.5.6 Unroll short loop in XA_REGDUMP

The original driver used a loop to execute three succcessive MOVL instructions (Move Longword), for a total of seven instructions altogether. The driver was modified to use a MOVL...MOVQ two-instruction sequence (Move Longword...Move Quadword) instead. In addition, the three instructions following the loop were optimized to take advantage the adjacency assumed in the definition of the UCB extension.

# APPENDIX B

# XADRIVER Update Procedure

## B.1 Source Code Update

The following DCL command procedure, XADRIVER.slp, includes an inline driver source code update file for the SUMSLP batch editor that generates a new version of XADRIVER.mar in the default directory. Version X-6 of XADRIVER.mar (supplied in Sys$Examples) is required.

Standard Macro-32 syntax is used, i.e., fields are separated by tabs. <FF> represents the ASCII formfeed character.

NOTE

> To fit all the text on the page, the first two tab stops in the listing below are 7 spaces each, instead of the normal 8 spaces. When the text is entered at a terminal with standard DEC tabs stops, there will be some visual differences from what is printed below.

XADRIVER.slp

```
$Edit/SUM /Output=XADRIVER.mar /List=XADRIVER.sum -
          Sys$Examples:XADRIVER.mar/Update=Sys$Input
\
-2,2
       .IDENT 'X-6E'
-57
;      X-6E   LMB005          L. M. Baker     17-Aug-1987
;             Correct comments describing IO$_SETCHAR outputs.
;             If MicroVAX I or 18-bit DRV11-WA, leave unit offline
;                instead of BUGCHECKing.
;             Serialize IO$_SETMODE/IO$_SETCHAR w/ all requests.
;             Serialize IO$_SENSEMODE/IO$_SENSECHAR w/ all requests.
;             Remove unnecessary extended local symbol block.
;             Unroll short loop in XA_REGDUMP.
;
;      X-6D   LMB004          L. M. Baker     10-Aug-1987
;             Raise fork IPL to 11 if UBA resources have been locked
;                 (IO$_SETCHAR!IO$M_NOWAIT); save default FIPL in
;                 UCB$LDEVDEPND2+1; restore to default fork IPL when
;                 UBA resources are unlocked for proper synchronized
;                 fork-level access to the UBA resource database.
;
;       X-6C    LMB003          L. M. Baker     10-Aug-1987
;               Implement IO$_SETCHAR modifier IO$M_NOWAIT to lock data
;                  paths and UBA mapping registers; reject transfers that
;                  exceed UCB$W_DEVBUFSIZ when UBA mapping registers are
;                  locked.
;
;       X-6B    LMB002          L. M. Baker     27-Aug-1987
;               Preserve data path selection when the driver is
;                  reloaded (convert controller initialization routine
;                  to a unit initialization).
;               Correctly call device reset routine from controller/
;                  unit initialization routine.
;               Change instructions in controller/unit initialization
;                  that acces the I/O page to use word context instead
;                  of longword context.
;               Validate 22-bit mode DRV11-WA if MicroVAX II.
;
;       X-6A    LMB001          L. M. Baker     25-Aug-1987
;               Maintain correct driver operation when IO$_SETCHAR
;                  modifies UCB$B_DEVTYPE (define bit 0 of
;                  UCB$L_DEVDEPND2 =0 if DR11-W, =1 if DRV11-WA;
;                  replace CMPB...BEQL instructions with BLBC or
;                  BLBS).
;               Correctly save previous map register contents in
;                  error log packet.
;               Correctly transfer byte-aligned buffers.
;               Restart transfer if power fails during device setup.
;
%
-186,,/; LMB001/

; Fields in second device-dependent status word in UCB (UCB$L_DEVDEPND2)

UCB$B_DRV11WA = UCB$L_DEVDEPND2       ; DRV11-WA flag (=1, DR11-W =0)
-,,/; LMB004/
UCB$B_DEFFIPL = UCB$L_DEVDEPND2+1     ; Saved default FIPL
XA_TOP_FIPL   = 11                    ; Highest fork IPL allowed
-,,/; LMB003/
UCB$W_MPRBUFSIZ       = UCB$L_DEVDEPND2+2     ; Largest mapped buffer size
-279,280,/; LMB002/
       DPT_STORE CRB,CRB$L_INTD+VEC$L_UNITINIT,- ; Address of unit
              D,XA_UNIT_INIT                     ; initialization routine
-308,308,/; LMB005/
       FUNCTAB +EXE$ZEROPARM,<SENSEMODE,SENSECHAR>
-310,310,/; LMB002/
       .SBTTL XA_UNIT_INIT, Unit initialization
-313,314,/; LMB002/
; XA_UNIT_INIT is called when driver is loaded, system is booted, or power
; failure recovery.
```

```
-326,328,/; LMB002/
;       R5 = address of UCB
-338,341,/; LMB002/
XA_UNIT_INIT:

        MOVL    R8,-(SP)                    ; Save R8
        MOVL    UCB$L_CRB(R5),R8            ; Address of CRB
        MOVL    CRB$L_INTD+VEC$L_IDB(R8),R0 ; Address of IDB
        MOVL    R5,IDB$L_OWNER(R0)          ; Make permanent controller owner
        MOVL    R5,R0                       ; Use R0 for address of UCB
-343,,/; LMB001/
        CLRB    UCB$B_DRV11WA(R0)           ; Assume DR11-W (clear bit 0)
-348,348,/; LMB005/

; MicroVAX I or 18-bit mode DRV11-WA (see test below)

3$:     BBSC    #UCB$V_ONLINE,-             ; Set device status "off-line"
                UCB$L_STS(R0),20$           ;    and return to exec
-350,,/; LMB001/
        INCB    UCB$B_DRV11WA(R0)           ; Bit 0 =1 for DRV11-WA
-,,/; LMB002/

; Verify DRV11-WA is in 22-bit mode

        CLRW    XA_BAR(R4)                  ; Clear low order address bits
        CLRW    XA_BAE(R4)                  ; Clear high order address bits
        BISW    #XA_CSR$M_XBA,XA_CSR(R4);   "Set" read-only addr bits in CSR
        BITW    #XA_CSR$M_XBA,XA_CSR(R4);   If clear, DRV11-WA is 22-bit
        BNEQ    3$                          ; If set, DRV11-WA is 18-bit
-359,360,/; LMB002/
        BITW    #XA_CSR$M_IE,XA_CSR(R4)     ; IE already set?
        BNEQ    9$                          ; If NE, yes
-373,,/; LMB005/
20$:                                        ; Reference label
-,,/; LMB002/
        MOVL    (SP)+,R8                    ; Restore R8
-454,457,/; LMB001/
-461,463,/; LMB001/
-514,514,/; LMB002/
; If either I/O function modifier IO$M_DATAPATH or IO$M_NOWAIT is set,
-518,518,/; LMB003/
20$:    BITL    #IO$M_DATAPATH!IO$M_NOWAIT,R0 ; If BDP or NOWAIT modifier,
        BNEQ    30$                           ;     queue packet
-519,520,/; LMB005/
        BRB     40$                         ; Queue packet to start I/O
-525,,/; LMB005/
40$:                                        ; Reference label
-540,540,/; LMB005/
;       This routine has four major functions:
-546,547,/; LMB003/
;       2) Set Characteristics.  If the function is change data path
;          or lock resources, the UCB is updated with the data path
;          and/or UBA resources are locked.
-,,/; LMB005/
```

```
;       3) Set Mode.  Modifies UCB$W_DEVBUFSIZ and UCB$L_DEVDEPEND
;          fields and finishes I/O with success.
;       4) Sense Mode/Characteristics.  Returns UCB$L_DEVDEPEND field
;          in R1 and finishes I/O with success.
-562,562,/; LMB005/
-570,571,/; LMB003/
-580,580,/; LMB003/
; select a data path and/or UBA mapping registers for future use.
-585,585,/; LMB003/
        BEQL    DO_SETCHAR
-,,/; LMB005/
        CMPB    #IO$_SETMODE,R2         ; Set mode?
        BEQL    DO_SETMODE
        CMPB    #IO$_SENSECHAR,R2       ; Sense characteristics?
        BEQL    DO_SENSECHAR
        CMPB    #IO$_SENSEMODE,R2       ; Sense mode?
        BEQL    DO_SENSECHAR
-,,/; LMB003/
        BRW     DO_READ_WRITE
<FF>
-,,/; LMB005/
;++
; SENSE CHARACTERISTICS - Process Sense Mode and Sense Characteristics
;                         QIO functions
;
; Note: IO$_SENSEMODE and IO$_SENSECHAR are processed here (instead of
;       by the executive EXE$SENSEMODE routine) to properly serialize
;       such requests with changes requested by IO$_SETMODE/IO$_SETCHAR.
;--

DO_SENSECHAR:

        MOVL    UCB$L_DEVDEPEND(R5),R1  ; Read device characteristics
        BRB     XA_FINISHIO_R0          ; Finish w/ success

;++
; SET MODE - Process Set Mode QIO function
;
; Note: IO$_SETMODE is processed here (instead of by the executive
;       EXE$SETCHAR routine) to properly serialize such requests with
;       changes requested by IO$_SETCHAR.
;--

DO_SETMODE:

        MOVW    IRP$L_MEDIA+2(R3),-     ; Set device buffer size
                UCB$W_DEVBUFSIZ(R5)
        MOVL    IRP$L_MEDIA+4(R3),-     ; Set device-dependent
                UCB$L_DEVDEPEND(R5)     ;    characteristics
        CLRL    R1                      ; Finish w/success (fall through)

XA_FINISHIO_R0:

        MOVZWL  #SS$_NORMAL,R0          ; Show successful return
```

```
       REQCOM                             ; Complete I/O
-594,,/; LMB003/
;
;      IO$M_NOWAIT I/O function modifier specifies whether to lock
;      (IO$M_NOWAIT=1) or unlock (IO$M_NOWAIT=0) UBA resources.
;
;      IO$_SETCHAR!IO$M_NOWAIT:
;
;             Lock the currently selected data path and enough UBA
;             mapping registers for the current device buffer size.
-,,/; LMB004/
;             Raise fork IPL to the highest fork IPL available (11).
-,,/; LMB003/
;
;      IO$_SETCHAR:
;
-,,/; LMB004/
;             If UBA resources are currently locked, lower fork IPL
;             back to the default fork IPL, and release locked UBA
-,,/; LMB003/
;             resources.
-598,,/; LMB003/
;      UBA resources are locked in the CRB/VEC, if requested.
-,,/; LMB004/
;      UCB$B_FIPL is raised or lowered, if necessary.
-600,601,/; LMB005/
;             XA$M_DATAPATH = 1 -> buffered data path in use
;             XA$M_DATAPATH = 0 -> direct data path in use
;             XA$M_LINK     = 1 -> interprocessor link mode
;             XA$M_LINK     = 0 -> user device mode
-603,,/; LMB003/

DO_SETCHAR:

; Release UBA resources previously locked

       MOVZWL #SS$_NORMAL,R0              ; Assume success
       BBCC   #VEC$V_MAPLOCK,-            ; UBA resources locked?
              CRB$L_INTD+VEC$W_MAPREG(R4),220$
-,,/; LMB004/

       MOVB   UCB$B_DEFFIPL(R5),-         ; Restore IPL to synchronize
              UCB$B_FIPL(R5)              ;    access to UBA resources
       FORK                               ; Synchronize processing

-,,/; LMB003/
       RELMPR                             ; Yes, deallocate private MPRs
       CMPZV  #VEC$V_DATAPATH,-           ; Buffered data path locked?
              #VEC$S_DATAPATH,-
              CRB$L_INTD+VEC$B_DATAPATH(R4),#0
       BEQL   210$                        ; No, direct path is shareable
       BICB   #VEC$M_PATHLOCK,-           ; Yes, unlock BDP
              CRB$L_INTD+VEC$B_DATAPATH(R4)
       RELDPR                             ; Deallocate private BDP
```

```
210$:  MOVZWL IRP$W_FUNC(R3),R1        ; Restore entire function c

; Select UBA data path and set device characteristics

220$:  BBC    #IO$V_DATAPATH,R1,2$     ; If data path modifier set
-610,,/; LMB003/
       MOVZWL #SS$_NORMAL,R0          ; Restore successful status

; Lock UBA resources

       BBC    #IO$V_NOWAIT,R1,-        ; Lock UBA resources?
              XA_FINISHIO_R1
       MOVZWL UCB$W_DEVBUFSIZ(R5),R3  ; Buffer size in bytes
       MOVW   R3,UCB$W_MPRBUFSIZ(R5)  ; To validate transfers lat
       ADDL   #511,R3                  ; Rounded up to pages, allo
       DIVL   #512,R3                  ;    for page spillover and
       ADDL   #2,R3                    ;    invalid trailer page
       JSB    G^IOC$ALOUBAMAPN         ; Allocate private MPRs
       BLBC   R0,XA_FINISHIO_R1        ; Failed
       BISW   #VEC$M_MAPLOCK,-         ; Lock mapping registers
              CRB$L_INTD+VEC$W_MAPREG(R4)

; If the direct data path is selected, it's already locked and RE
; return immediately.  If a buffered data path is selected, we'll
; get one (possibly after waiting for one to become available), o
; fail because the processor has none.  We can lock the data path
; way (ignoring failures), since the direct data path is always s

       REQDPR                          ; Request data path registe
       BISB   #VEC$M_PATHLOCK,-        ; Lock the data path
              CRB$L_INTD+VEC$B_DATAPATH(R4)
-,,/; LMB004/

       MOVB   UCB$B_FIPL(R5),-         ; Save default fork IPL
              UCB$B_DEFFIPL(R5)
       MOVB   #XA_TOP_FIPL,-           ; Raise fork IPL (no need t
              UCB$B_FIPL(R5)           ;    synchronize access to
       FORK                            ; Fork to new IPL
-,,/; LMB003/

XA_FINISHIO_R1:

-612,612,/; LMB003/
-613,,/; LMB003/
<FF>
; Validate transfer size if UBA mapping registers have been locke

DO_READ_WRITE:

       BBC    #VEC$V_MAPLOCK,-         ; UBA resources locked?
              CRB$L_INTD+VEC$W_MAPREG(R4),290$
       BBC    #IO$V_WORD,R1,290$       ; Only matters if block mod
       CMPW   UCB$W_BCNT(R5),-         ; Request too large?
              UCB$W_MPRBUFSIZ(R5)
```

```
        BLEQU   290$                    ; No, process request
        MOVZWL  #SS$_IVBUFLEN,R0        ; Yes, invalid buffer length
        BRB     XA_FINISHIO_R1          ; Finish w/error status

290$:   MOVL    @CRB$L_INTD+VEC$L_IDB(R4),R4 ; Address of CSR
-695,,/; LMB001/
        BLBC    UCB$W_BOFF(R5),251$     ; Word-aligned transfers OK
        MOVL    UCB$L_CRB(R5),R2        ; Address of CRB
        CMPZV   #VEC$V_DATAPATH,-       ; Buffered data path allocated?
                #VEC$S_DATAPATH,-
                CRB$L_INTD+VEC$B_DATAPATH(R2),#0
        BNEQ    251$                    ; Yes, byte-alignment OK
        RELDPR                          ; No, release data path
        MOVZWL  #SS$_BADPARAM,R0        ; Set error status code
        CLRL    R1
        REQCOM                          ; Abort I/O
251$:                                   ; Reference label
-707,709,/; LMB001/
        BLBC    UCB$B_DRV11WA(R5),100$  ; If this is a DR11-W, then branch
-736,738,/; LMB001/
        BLBC    UCB$B_DRV11WA(R5),200$  ; If this is a DR11-W, then branch
-743,,/; LMB001/
        SETIPL  UCB$B_DIPL(R5)          ; Allow powerfail interrupts
        BBCC    #UCB$V_POWER,-          ; Branch if no powerfail and
                UCB$L_STS(R0),260$      ;    clear powerfail indicator
        ENBINT                          ; Enable interrupts
        PURDPR                          ; Purge buffered data path
        RELMPR                          ; Release I/O mapping registers
        RELDPR                          ; Release buffered data path
        BRW     XA_START                ; Restart transfer (R3 & R5 are OK)
260$:
-771,773,/; LMB001/
        BLBC    UCB$B_DRV11WA(R5),300$  ; If this is a DR11-W, then branch
-786,786,/; LMB001/
        MOVL    (R2)[R0],UCB$L_XA_PMPR(R5) ; Save prev map register contents
-802,804,/; LMB001/
        BLBC    UCB$B_DRV11WA(R5),37$   ;   and the operation is incomplete,
                                        ;   in which case it is an expected
                                        ;   error and not worth logging.
-819,820,/; LMB005/
-873,875,/; LMB001/
        BLBC    UCB$B_DRV11WA(R5),17$   ; Branch if this is a DR11-W
-940,942,/; LMB001/
        BLBC    UCB$B_DRV11WA(R5),1037$ ; Branch if this is a DR11-W
-1014,,/; LMB001/
        BITW    #^C<^X01FF>,UCB$W_BOFF(R5) ; Page overflow?
        BEQL    30$                     ; If EQ, not yet
-1016,1016,/; LMB001/
-1175,1177,/; LMB001/
        BLBS    UCB$B_DRV11WA(R5),57$   ; If this is a DRV11-WA, then branch
-1192,1194,/; LMB001/
        BLBC    UCB$B_DRV11WA(R5),70$   ; If this is a DR11-W, then branch
-1362,1362,/; LMB005/
-1364,1367,/; LMB005/
```

```
        MOVQ    (R1)+,(R0)+             ;     (3 longwords total)
        INCL    R1                      ; Point to UCB$W_XA_DPRN+1
        MOVZBL  (R1)+,(R0)+             ; Save Datapath Parity Error Flag
        MOVZWL  (R1)+,(R0)+             ; Save BAE stored prior to xfer
        MOVZWL  (R1)+,(R0)+             ; Save BAE stored following xfer
-1393,1395,/; LMB001/
        BLBC    UCB$B_DRV11WA(R5),20$   ; If this is a DR11-W, then branch
/
```

## B.2 Update Command Procedure

The following DCL command procedure, XADRIVER.com, generates a new version of XADRIVER.exe in the default directory.

XADRIVER.com

```
$ @XADriver.slp
$ Macro /List XADRIVER + Sys$Library:LIB.mlb/Library
$ Link  /Map  XADRIVER,Sys$System:SYS.stb/Selective_search, -
              Sys$Input/Options
Base=0
$
```

To install the new driver, copy it to Sys$System and reboot the system.

NOTE

Readers who wish to obtain machine-readable copies of these files may contact the author at

U. S. Geological Survey
Office of Earthquakes, Volcanoes, and Engineering
345 Middlefield Road MS977
Menlo Park, CA 94025
(415) 329-5608 or FTS 459-5608