SLAC-PUB-2494
April 1980
(M)

A PDP-11 FRONT-END FOR A VAX-11/780*

M. J. Browne, Charles Granieri, D. J. Sherden, Leon J. Weaver
Stanford Linear Accelerator Center
Stanford, California

ABSTRACT

An unpublicized feature of the VAX-11/780 is the provision for attaching a PDP-11 to the VAX UNIBUS Adapter. This can give significantly improved I/O performance for applications which are limited by overhead in the VAX I/O driver rather than by the transfer speed of the UNIBUS itself. We have implemented such a system using a PDP-11/04 as a "front-end" to a CAMAC data acquisition system. Both the PDP and the VAX have full access to the UNIBUS. That portion of the PDP address space which does not have UNIBUS memory can be mapped to buffers in the VAX memory, allowing the PDP to access VAX memory and to initiate DMA transfers directly to the VAX. The VAX also has full access to the PDP memory, providing a convenient means for developing and downloading the PDP software.

INTRODUCTION

As online computers have progressed from simple to more complicated machines such as the VAX-11/780 many of the more difficult tasks have been made easy, but many of the simple tasks have been made difficult. One such area is in the reading of time-critical data. While modern computers offer useful facilities for user protection, DMA capability, and I/O queueing and buffering, the software overhead in the interrupt servicing and I/O systems has in some cases outpaced the increasing speed of the computers. Fortunately, such problems are well suited to micro-computer applications.

In this paper we describe a means of connecting a PDP-11 computer (in our application a PDP-11/04) to the VAX-11/780, which allows the two to interact in a very flexible fashion. We note that this scheme is an unpublicized design feature of the VAX, and hence credit for the system should go to Digital, although we accept responsibility for any misstatements in this paper, and, should it make anyone feel better, blame for any of its failures.

In our particular application a VAX-11/780 is used for the acquisition and analysis of data for elementary particle physics experiments in End Station A of the Stanford Linear Accelerator Center. Communication with the experimental equipment is accomplished through a CAMAC [1] system using three Jorway Model 411 Branch Drivers [2]. Time critical data are acquired at rates of up to 360 "events" per second. While the amount of data read for each event is small (typically several hundred bytes), the high repetition rate, combined with the fact that each event requires several separate PDT (single word transfer) and DMA (block transfer) operations, introduces significant software overhead.

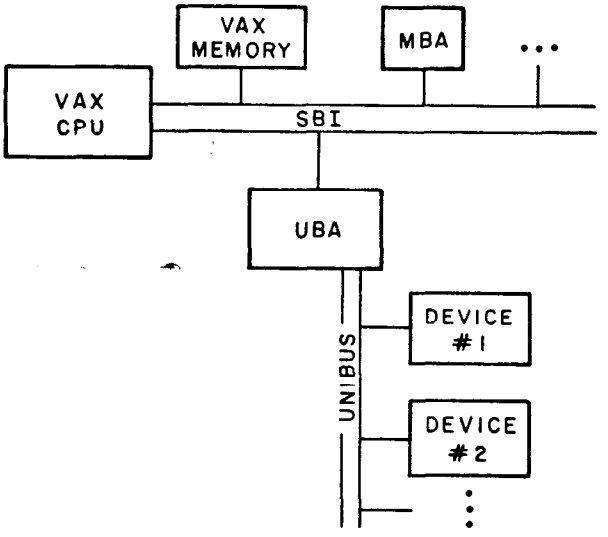
As the system was originally set up, the event data were read by the VAX using a locally written CAMAC I/O software driver [3]. To minimize software overhead this system provides for list-driven multiple CAMAC operations and multiple interrupt servicing within a single QIO system service call (note that the software overhead of a single QIO call is comparable to the time between events). Independent of the overhead involved in the QIO call itself (~2 msec), each CAMAC PDT operation requires ~80 usec, each DMA operation requires ~300 usec, and each event interrupt requires ~300 usec. With this system, the reading of event data required ~25% of the total CPU power of the computer. Particularly when compared to the time available for the analysis of events, this is a quite significant overhead, which, with the present scheme, is almost entirely eliminated through the use of a "front-end" PDP-11/04 to read the data.

THE SECRET PORT

In the most naive picture, one imagines the UNIBUS adapter (UBA) hanging from the SBI with the UNIBUS emanating from the UBA, as shown in Figure 1. The UNIBUS arbitration functions are mentally associated with the UBA. There are two separate UNIBUS arbitration functions: Non-Processor Request (NPR) arbitration and Bus Request

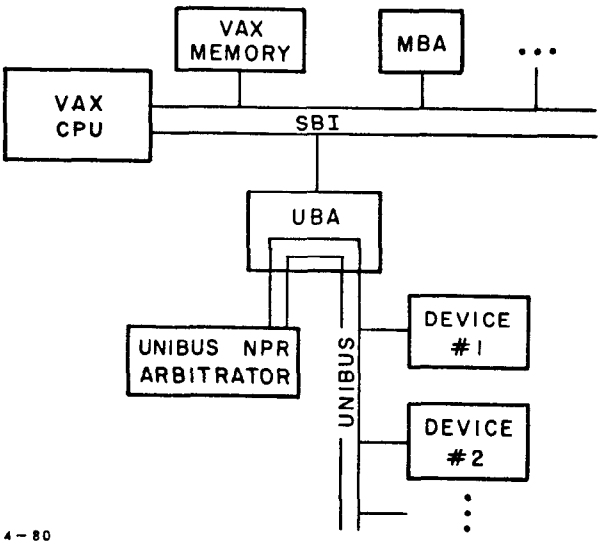
*Work supported by the Department of Energy, contract DE-AC03-76SF00515.

Presented at the 1980 Spring DECUS U. S. Symposium, Chicago, Illinois, April 22 - 25, 1980.



4-80 FIGURE 1 3810A1

A naive view of the UBA.



4-80 3810A2 FIGURE 2

A slightly less naive view showing the separation of UBA and NPR arbitrator.

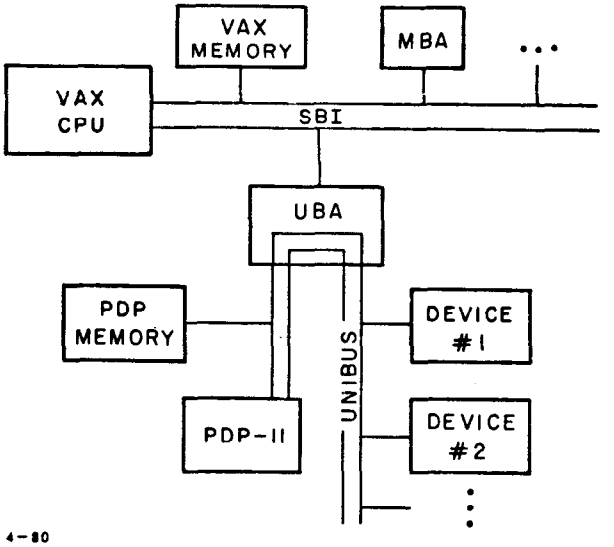
(BR) interrupt processing. While the UBA does contain a BR interrupt processor, the NPR arbitrator is separate from the UBA, as shown in Figure 2. The UBA may, for most purposes, be considered as an NPR device on the UNIBUS. The NPR arbitrator is functionally identical to that in any PDP-11, so one can simply replace the arbitrator card with a standard UNIBUS cable to a PDP-11 without adverse effect, as shown in Figure 3.

THE UNIBUS ADAPTER

Almost all of the information in this section is readily available from the VAX-11/780 Hardware Handbook. We nonetheless present the information here since it is relevant to understanding the VAX/PDP interaction.

One of the most important functions of the UBA is to map the UNIBUS address space 000000-757777(8) to VAX memory. For this purpose, the UBA contains 496 map registers, allowing one to map each page of UNIBUS address space to VAX memory. Because the PDP has its own memory on the UNIBUS, one needs a mechanism for disabling the corresponding map registers in the UBA. For this purpose (or to accommodate external UNIBUS memories, in general) DEC provides the Map Register Disable field (bits 26:30) of the UNIBUS Adapter Control Register (UACR). This field may be loaded with the number of 4K word blocks of external UNIBUS memory, which must begin at UNIBUS address 0. With the corresponding map registers disabled, the VAX has complete access to the UNIBUS memory; the disabling simply prevents the UBA from attempting to associate these pages with VAX (SBI) memory. The fact that the VAX can access the PDP memory directly provides, among other things, an extremely simple method of downloading programs to the PDP.

The PDP can be given access to VAX memory using the UBA map registers. Here one



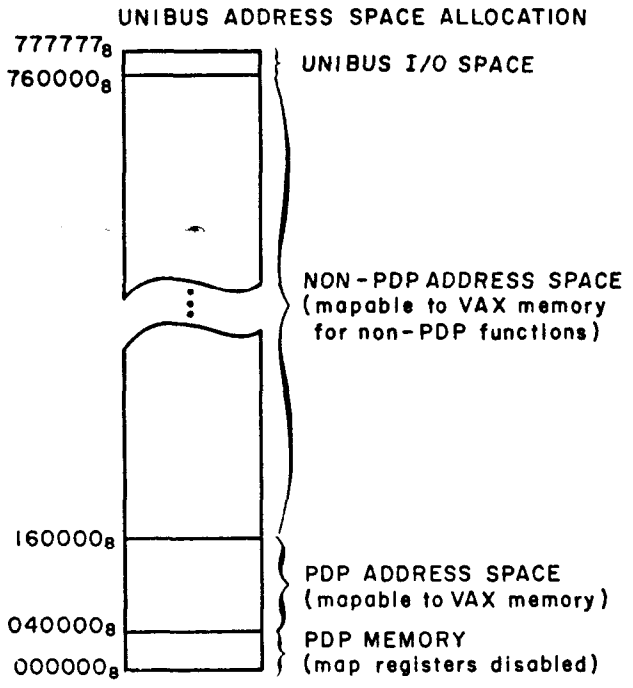
4-80 3810A3 FIGURE 3

UBA with NPR arbitrator replaced by PDP-11

simply maps some portion of PDP address space without UNIBUS memory to the desired pages of VAX memory. Note that for PDP models without memory management, the sum of PDP memory plus PDP-accessable VAX memory is thus limited to 28K words. The UNIBUS address space allocation for our configuration (PDP-11/04 with 8K words of memory) is illustrated in Figure 4.

As long as the map registers remain unchanged, the PDP may treat the associated VAX memory as though it were its own. Since the PDP also has access to the UNIBUS I/O space (760000-777777(8)), it may initiate I/O from a UNIBUS device directly to or from VAX memory.

Because the UBA has its own BR interrupt processor, it intercepts BR interrupts from all UNIBUS devices downstream of the UBA

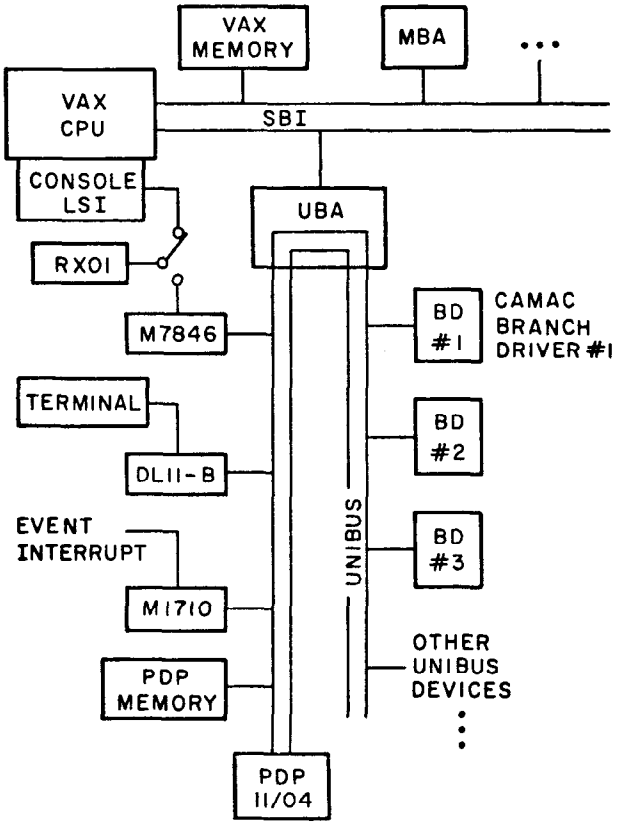


4-80 FIGURE 4 3810A4

(the "VAX side" of the UNIBUS) and interrupts the VAX, not the PDP. BR interrupts from devices upstream of the UBA (the "PDP side" of the UNIBUS) are received by the PDP and not the VAX. This arrangement can be altered by clearing (or, more precisely, by not setting) the Interrupt Field Switch (Bit 6) of the UACR. In this case the UBA will pass all BR interrupts to the PDP.

HARDWARE

Hardware associated with the PDP-11/04 in our system is shown schematically in Figure 5. The PDP is powered up and down with the VAX CPU. When the system is powered up, PDP control is transferred to the ROM of the M9301-YA bootstrap module. Since the UBA is initially unmapped, it is important to prevent the PDP from accessing its own memory until the UACR Map Register Disable Field has been properly initialized. (Note that the M9301 ROM is in the I/O space of the UNIBUS, which is always known to be external by the UBA.) Hence it was desireable to provide a means of communication between the VAX and the PDP which did not require the use of PDP memory. For this purpose a simple interface module was build using an MDB-1710 foundation module [4] plugged into the PDP side of the UNIBUS so that the PDP rather than the VAX receives its interrupts. The interface module consists of a control register (CSR), two data registers, and two interrupts. One register is used by the VAX to initiate program operation, while the second is left free for user applications. The first interrupt is used to signal an event, while the second is used by the VAX to terminate PDP program operation. (As designed, either interrupt can be fired from hardware or from



4-80 FIGURE 5 3810A5
PDP-associated hardware for our system.

software.) To provide for PDP program downloading a simple modification was made to the M9301 ROM program. The program initially loads an odd number into the data register of the interface module. In addition to looking for input from the DL-11B (i.e. the normal console emulator routine) the program also looks at the data register. After the VAX has downloaded the program into the PDP memory, it places the (even) starting address in the data register. This is recognized by the PDP and used to begin program execution. Sufficient unused space exists in the M9301-YA that the downloading feature could be added without elimination of any features of the standard ROM.

In the standard ROM console emulator routine, a START command causes the ROM program to execute a RESET before transfering control to the requested address. Since this instruction resets all devices on the UNIBUS, its execution after the VAX has been bootstrapped would have disastrous consequences. Hence the RESET instruction was eliminated from the ROM program.

In addition to the VAX/PDP interface, the PDP also has a DL-11B serial line interface and an M7846 floppy disk controller, both of which are used solely for diagnostic purposes. The DL-11B can be connected to one of the terminals normally used by the VAX, and the M7846 can be plugged into the

RX01 normally used by the VAX console LSI. As an indication of the reliability of the system we note that the terminal has not been connected to the PDP since the initial debugging of the PDP program, and, apart from verifying that the M7846 module worked, the floppy disk has never been connected to the PDP.

The only unexpected problem which arose in bringing up the system occurred in the bootstrap sequence. With the switches of the M9301 bootstrap module properly set, the power up sequence should cause the PDP to interrupt to the M9301 ROM. This indeed occurred when power was initially applied to the CPUs. However, the power fail and power up sequence can also be generated from VAX software by: (i) SBI UNJAM, (ii) setting the Adapter Init field (Bit 0) in the UACR, or (iii) setting (and resetting) the UNIBUS Power Fail field (Bit 1) in the UACR. At least one of these methods is employed by the VAX bootstrap procedure. It was found that for the software generated power up sequence, the AC LO signal was deasserted simultaneously with DC LO rather than the prescribed >5 usec later. This caused the PDP to trap to location 24 (power fail) rather than to the bootstrap ROM. While a less brutal approach is probably possible, we cured the problem by cutting a trace on the M9301 module.

SOFTWARE

A VAX/VMS I/O driver was written to support the "front-end" PDP-11/04. While not technically necessary, the I/O driver format was chosen because it offered a convenient and well documented means of accessing both VMS system routines and UNIBUS addresses. There is, however, one special feature about this driver. In order to allocate the UBA map registers specifically associated with the PDP, the PDP driver must be loaded before the drivers of any other devices which access the UNIBUS.

When it is loaded, the PDP driver initialization routine performs the following functions:

1. It permanently allocates within VMS the first 28K words of UNIBUS addresses (i.e. the PDP address space).

2. It sets the Map Register Disable field of the UACR for the 8K words of PDP UNIBUS memory which is attached.

3. It sets up VMS system page table entries so that the 8K of UNIBUS memory may be read and written directly from the VAX and saves the generated virtual addresses for later use.

VAX user programs may access the PDP through the PDP driver by using the standard QIO system service. The PDP driver supports the following functions:

1. Write to PDP memory.

2. Read from PDP memory.

3. Read VAX/PDP interface registers.

4. Write VAX/PDP interface registers.

5. Interrupt the PDP by writing into the VAX/PDP interface CSR register.

6. Set up a "never-ending" QIO which maps UNIBUS addresses between 8K and 28K into VAX memory to allow the PDP program to read data directly into or from the VAX memory.

Down-loading of programs from the VAX to the PDP is accomplished by a user level routine using the QIO facility described above. PDP programs are prepared, assembled, and linked using the compatibility-mode RSX-11M facilities of the VAX. The down-loading routine reads the load module to determine the program length, first address, and starting address, as well as the program itself. Using the QIO facility, the program is written to PDP memory, and the starting address is written to the VAX/PDP interface module data register to initiate program operation.

OPERATIONAL EXPERIENCE

In our particular application one of the three CAMAC branch drivers is dedicated exclusively to the reading of event data. This restriction was present in the original VAX-based system and has been carried over to the PDP system. The other two branch drivers are used for I/O which occurs at repetition rates significantly lower than that of the event data, and remain driven by the VAX rather than the PDP. The isolation of the event branch was adopted to avoid handshaking and interlocking problems between the VAX and the PDP.

The event interrupt was moved from the VAX to the PDP. The event branch driver, however, was left on the VAX side of the UNIBUS so that diagnostic programs can be run from the VAX when the PDP is not running its normal event reading program. This prohibits the PDP from receiving DMA completion interrupts from the branch driver. However, since CPU time on the PDP is not at a premium, it is a simple matter for the PDP to monitor the CSR of the branch driver until the DMA operation is completed.

Upon receipt of an event interrupt, the PDP reads event data along with status information from the branch driver directly into a circular buffer in VAX memory. The buffer is large enough to contain roughly 20 events. A VAX program, which is activated roughly 10 times per second, retrieves data from the buffer for analysis and logging on magnetic tape.

Because the PDP could be programmed to read data in a more brute force fashion than the list-driven VAX I/O system, the PDP was able to duplicate the functions of the VAX-based system using less real time. In practice,

the PDP program was expanded to provide error checking with retry capability, and to format the data in a form more convenient to the specific experiment than that of the more general VAX-based system. The final PDP program reads events in approximately the same time as did the original VAX system. While most of the PDP program is specific to the current experiment, it was written with sufficient generality that reprogramming for an auxiliary experiment was accomplished in less than a day.

The PDP system has been in use for six months, and has not encountered problems in that time.

CAVEATS

While we have been extremely satisfied with the system, we must add that it is not all things to all people.

1. The system is effective in eliminating software overhead but does not improve hardware performance. In particular, since the PDP requires UNIBUS cycles to access its own memory, UNIBUS performance will be degraded rather than enhanced. Note, however, that the PDP itself has the lowest UNIBUS priority and the WAIT instruction can be used to inhibit PDP activity during idle periods.

2. For PDP models (including the PDP-11/04) which use the DATIP-DATO (read-modify-write) sequence to write to memory, the UBA direct data path must be used, resulting in heavier traffic on the SBI. Even for models using the DATO sequence, the use of a UBA buffered data path would destroy the feature of 16 bit random access to VAX memory by the PDP, although one could have 32 or 64 bit access. Similarly, for any model, the PDP could initiate DMA transfers from I/O devices to the VAX through a buffered data path if the block size were always in integral units of 32 or 64 bits, or if the VAX were interrupted to purge the data path after DMA completion.

3. VAX buffers which are to be used by the PDP must be page aligned in VAX memory since the UBA map registers can only map a page of VAX memory to a page of UNIBUS address space.

4. A HALT instruction on some PDP models (including the PDP-11/04) hangs the UNIBUS causing the VAX to crash. Don't do that.

5. Access by the PDP to unmapped (and not disabled) UNIBUS addresses also causes the VAX to crash. Don't do that either.

6. One of the VAX micro-diagnostics gives an error condition when the PDP is connected. We have not investigated this further, but simply disconnect the PDP before running the micro-diagnostics.

7. Crash recovery is not an important aspect of our system and we have not paid detailed attention to the relative power up and down sequences of the VAX CPU, PDP CPU, and UNIBUS adapter.

8. As previously mentioned, no general scheme exists for the sharing of interrupts.

9. While it is in principle possible to share a common I/O device between the VAX and the PDP, provision must obviously be made in hardware and/or software to handle the associated handshaking and lock-out problems.

SUMMARY

The system described provides a simple means of interfacing a PDP-11 computer to a VAX-11/780, and offers the following features:

1. Complete access to the UNIBUS by both PDP and VAX computers.

2. Complete access to PDP memory by the VAX.

3. Limited access to VAX memory by the PDP.

4. Initiation by the PDP of I/O directly from UNIBUS devices to VAX memory.

5. Availability of RSX-11M facilities on the VAX, providing a convenient means of PDP program development.

ACKNOWLEDGEMENTS

We would like to thank Mr. Rick Casabona, of the DEC VAX-11/780 Engineering Group, for his advice in this project.

This work was supported by the Department of Energy under contract number DE-AC03-76SF00515.

REFERENCES

[1] Computer Automated Monitoring And Control (CAMAC), IEEE Standard 583-1975.

[2] Jorway Corporation, Westbury, N.Y.

[3] Charles Granieri and Karl Johnson, REAL-TIME DATA COLLECTION USING MULTIPLE VAX/VMS SYSTEMS, Proceedings of the Digital Equipment Users Society, Vol. 5, No. 4, April 1979.

[4] MDB Systems Inc., Orange, California.