Software Implementation of a High Speed Interface
Between a PDP-10 and Several PDP-11s

Nicanor Peter de Mesa III
Princeton University
Plasma Physics Laboratory
Princeton, New Jersey 08540

Abstract

The DMA10 is a high speed link between a PDP-10 and up to eight PDP-11s; specifically, the PDP-10 shares sections of its memory with the PDP-11s. The two segment concept on the PDP-10 of shared/reentrant code and non-shared code is implemented. The inclusion of read only memory on the PDP-11s allows for the development of "PHNM" software which all the PDP-11s may share. The principal difference between the DMA10 and other communications interfaces is that it is not a block transfer device. Because of the shared memory concept the features of the DMA10 are high data bandwidth and minimal processor intervention between data transfers. Communication programs between the PDP-10 and the PDP-11 may be tested wholly in either processor, independent of the DMA10 interface. In the current mode of operation the PDP-11s simply act as device controllers. Future plans include separate operating systems in various PDP-11s.

Introduction

The DMA10 Interface described in this paper was constructed by DEC's Computer Special Systems Group to CERN specifications. Currently, the DMA10 constitutes the "central nervous system" of the PLT data acquisition system (DAS) or data acquisition computer complex. All data collected is passed through it and in most cases all diagnostic displays are returned through it to various 613 scope displays.

The PDP-10 or DECSYSTEM-10 forms the central processor in which all program preparation, diagnostic analysis, and database support is performed. The PDP-11s are responsible for all real-time data acquisition and data transfer under the direct control of the PDP-10.

*This work was supported by EPDA Contract K111-1-3073.

DISTRIBUTION OF THIS DOCUMENT IS UNLIMITED
Special Concepts and Terminology

The two segment capability of the PDP-10 is implemented so that reentrant system subroutines and/or programs may be shared by concurrently executing jobs. The rationale for the development of this capability is to economize memory usage and allow more executing jobs to be in memory rather than out on the disk swapping device. This capability has been extended by the DMA10 to the PDP-11s. Normally, all PDP-11s map their read only window area to the same PDP-10 physical memory address. Consequently, they may all execute simultaneously reentrant programs in this PDP-10 PROGRAMMABLE PDP-11 READ ONLY (PROM) memory.

The Inter-Process Communications Facility of the DECSYSTEM-10 (IPCF) is used by concurrently executing jobs to communicate and synchronize events.

The terms PI skip chain and CONO represent the DECSYSTEM-10's Interrupt Priority Service Chain and its I/O control output instruction, respectively.

The DECsystem's capability of hibernating and waking is implemented so that jobs which do not I/O block may relinquish processor time in the event they have nothing to do.

Restrictions on Memory Allocation

Due to the mixture of Memory Management Hardware between the KI-10 and the PDP-11/40 certain restrictions are imposed on the PDP-10 programs allocating or using a window. Specifically, the PDP-10-KI-10 is a paging memory management system, whereas, the PDP-11 and the DMA-10 are two register memory management systems. It is not surprising that the DMA-10 is not a paging device. First, the DMA-10 was designed to be used with the KA-10 which was a two register memory management computer. Second, the DMA-10 is implemented as a device interface. To my knowledge to date, DEC makes no peripheral devices with paging address hardware.

Consequently, whereas memory fragmentation is not a problem in paging systems, it is a problem with two register systems. Further, exclusive of fragmentation, contiguous sequential addresses are not required in paging systems and "free core linked lists" implemented for speed preclude contiguous sequential addresses.

Finally, each job on the PDP-10 has its own page map and must, therefore, have each window mapped contiguous and
sequentially. The result is the inability to quickly dynamically allocate window address space.

Usage of Memory Allocations

With the usage of appropriate semaphoring capabilities, the types of buffers or data structures which the DMA-10 can support are phenomenal! Circular buffers, one word buffers, rings of block buffers and queues are the data structures implemented or tried.

Circular buffers of various sizes are used to pass ASCII text lines to and from the PDP-10. This implementation could allow via the pseudo teletype facility of the PDP-10, jobs to log in from the PDP-11's.

One word buffers are ideally suited for digital I/O from switches and thumb wheels.

Rings of block buffers are used to drive the Tektronix 613 scopes. This implementation is such that the PDP-10 views these 613 as 4010's.

Queues have been tested and will be used as the PDP-11's develop with small operating systems so that the PDP-10 will have direct control over data acquisition.

Most dramatically, the UMBUS concept on the PDP-11's allows NPR devices on the PDP-11's to directly transfer data to and from PDP-10 memory. The effect is that the devices are attached to the PDP-10, not the PDP-11!

Analysis of Reentrant Code

Anyone who has ever generated reentrant code is aware of the considerations necessary for reentrant code for non-context switched calls. However, software executed in PROM need only avoid modifying itself to be inter-processor reentrant.

DMACON-DMA10 Software System Overview

Using the facilities of the DMA10 hardware in a timesharing environment on the DECsystem-10, the DMA10 Run Time System allows DAC jobs to conveniently control up to eight PDP-11's. The design goals of the DMA10 Run Time System were:
1. To provide any DECsystem-10 DAC job with an easily used facility to request and communicate with a data acquisition program on the PDP-11,

2. To have the software operate in a standard TOPS-10 environment,

3. To operate during general timesharing on the DECsystem-10 without causing excessive system overhead,

4. To protect the concept of the slave-master relationship of the PDP-10 and PDP-11,

5. To provide advanced diagnostic capabilities so that the DMA10 data collection routines would be reliable or easily serviceable and maintainable.

These design goals dictated the use of a single event-driven control job to funnel interrupt requests to and from the pnp-10 to the PDP-11, and to control memory resource allocation.

All resource allocation on the PDP-11 is controlled via the DAC job which is running. By adhering to a strict format for requesting interrupts and strict window definition the DAC program on the PDP-11 communicates with the DAC job on the PDP-10.

The principal difference between the PDP-10 job and the PDP-11 program are:

1. Only the PDP-10 job can request the PDP-11 be reloaded.

2. Only the PDP-10 job can determine the program running on the PDP-11.

DAC jobs connect to the DMACON high segment via IPCF. Thereafter all requests are handled via window flag definition and wake MMUs between the DAC job and DMACON. Since the system was designed to be used in a FORTRAN environment to simplify the user interface, the concept of a sharable FORTRAN run time system may be extended to include those facilities which are required by FORTRAN DAC jobs utilizing the DMA10 hardware. The standard FORTRAN run time system (FOROTS) provides DAC jobs with the necessary resources to process disk input/output, convert data from external to internal format, print a message on the user's terminal, etc. FORTRAN jobs wishing to use the DMA10 hardware facilities require the ability to address the Communication Window mapped by the DMA10 logic into PDP-11 address space. Since FORTRAN jobs already have FOROTS as their high segment it is elective via parameter entry to DMACON to elect on which high segment it builds its windows.
This overall organization provides the FORTRAN user programs with very powerful, yet simple to use and efficient capabilities.

Generalized System Flow Description

In order to describe the interrelationships between various modules of the DMA10 software, the following narrative presents a description of flow through the system from initialization until the completion of a task on a PDP-11. For the purposes of this description, a simple case of a FORTRAN DAC job calling upon a PDP-11 program to perform a task is used.

Initializing the DMA10 Control Job

The DMA10 control job (DMACON) is started by a system level RUN command after an appropriate user level login on the DECsystem-10.

DMACON then:

1. Allocates low segment core areas for the shared loader. (This is done after the RESET UNO so JOB DATA area parameters will be initialized.)

2. Sets itself in a high priority queues to prevent any race conditions.

3. Sets its disk priority to three to allow quick loading of its PARAMETER FILE and shared loader.

4. Generates IPCF name.

5. Does a lookup for the file DMACON.PAR to see if its default parameters should be changed.

6. Interacts with the operator depending on initialized parameters.

7. Locks itself in core.

8. Connects itself to the PI skip chain.

9. Initializes the DMA10.

10. A core image file is written into the Shared Memory Window of DMACON so that the PDP-11 can begin executing the absolute loader directly from DECsystem-10 memory. The PDP-11 is then manually started to begin the execution of the absolute loader stored in the Shared Memory Window. If the PDP-11 is
not halted or not core cleared, it may not be necessary to manually start it. The PDP-11 enters a load control loop, waiting on data to be written in the segment of the Communication window for that PDP-11 from the DECsystem-10. The data that the absolute loader then receives in the Communication window is a PDP-11 program to perform specific functions known to the PDP-11. The PDP-11 program is stored as a DECsystem file in a blocked, absolute loader format. The initialization process is continued when the job DMA10 reads the PDP-11 parameter file and begins loading the PDP-11.

**Loading the PDP-11**

Program DMA10 is started using the DECsystem-10 RUN command. DMA10 attaches to the DMA10 high segment, sets up its job number with DMA10 and then puts itself to “sleep” until awakened upon wake event from a DAC job or DMA10, when a command to load a particular PDP-11 is typed on DMA10’s terminal, it is awakened by the event and loads a default program or the specified program! DMA10 computes the address of the Communication Area in DMA10 high segment for the specific PDP-11. DMA10 then reads the appropriate absolute loader file (in this example, in generalized PDP-11 control routine called MAINII) from the DECsystem-10 disk and passes it page-by-page to the absolute loader running from the shared Memory Window through the DMA10. When the entire file has been processed by the absolute loader and the program is completely loaded into PDP-11 local memory, the absolute loader transfers control to the program at the start address specified in the program blocked absolute loader file. Since DMA10 has completed its task of passing DECsystem-10 disk files through the Communications Window to the PDP-11 absolute loader, DMA10 goes back to checking other requests, then sleeping. The PDP-11 program is now ready to communicate with the PDP-10 job.

**Servicing DAC Job Request**

Assume that DAC job 1 program has just started running and would like to display data in the LED display on a PDP-11. The DAC Job makes a call to a subroutine in the LIBRARY. That subroutine writes the request in the window in DMA10’s high segment. The subroutine then sends a CONO to the PDP-11 and goes to “sleep” to be awakened only upon a “wake” from the PDP-11 program via the DMA10 interrupt service routine. If the requested interrupt service routine is operating on the PDP-11 it acknowledges by setting a flag in the window.

Assume that the PDP-11 has just completed the DAC Job’s request to write the LED display. At this time, DAC Job’s program is still “sleeping” and continues to “sleep” until the DMA10 interrupt service program awakens him. The PDP-11
writes the completion status of the LF0 service request into
the same Communication Window that was earlier used by the
DECsystem-10 to pass the task parameters to the PDP-11. The
DECsystem-10 Priority Interrupt System passes the interrupt
from the PDP-11 to the DMA10 interrupt service located in
the low segment of DMACON. The DMA10 interrupt service
routine on the DECsystem-10 "awakens" DAC JOB program and
may, due to an error, "awaken" itself (these functions all
happen at interrupt level and do not require DMACON itself
to be running).

If an error has occurred DMACON will take appropriate action.
Having nothing further to do, DMACON will go back to "sleep"
until it is requested again by another interrupt from a
PDP-11 or another request from a DAC job, or terminal
activity upon its own terminal.

During this control cycle, DAC job 1 program has resumed
running as a result of the interrupt level "wake". In this
case, the only data read from the Communication Window is a
completion status code written into the Communication
Window. The FORTRAN LIBRARY subroutine then exits and
returns to its caller in DAC job 1.

Description of the DMA10 Controlling Job

The DMA10 controlling job (DMACON) is a fundamental element
of the DMA10 software organization. As a job with both a
high and a low segment running on the DECsystem-10, DMACON
is a Manager of PDP-10 window. The standard FORTRAN
Run-Time System (FORTS) may be extended to provide the
necessary data areas for DECsystem-10/PDP-11 memory mapping.
It is extremely important to note that in the conventional
DECsystem-10 FORTRAN run-time environment, FOROTS does not
exist as a separate program but rather as a sharable high
segment mapped into one or more FORTRAN users' job address
space. In contrast to this conventional environment, DMACON
exists as a separate job on the DECsystem-10; however,
since DMACON's high segment may contain all the facilities
of the standard FOROTS, DMACON's high segment may be mapped
into all FORTRAN users' address space. For those FORTRAN
users not requiring the additional facilities provided by
DMACON's high segment, DMACON's high segment performs in
exactly the same fashion as would the standard FOROTS.
Thus, the important distinction is made between the job
DMACON and the sharable high segment FOROTS or DMACON.

Initialization complete, Job DMACON goes to "sleep" using
the HIGHER ID0 and becomes a passive, event-driven PDP-11
resource manager. Job DMACON will "wake" and be scheduled
to run in a high priority queue by the DECsystem-10
operating system on the occurrence of one or more of the
following events:
1. An IPCF packet is sent from a DAC job to job DMACON requesting DMACON's high segment.

2. A character is typed upon job DMACON's logical terminal. If a full line of text is present the command is interpreted and executed. Currently implemented commands are:

- **EQ**: exit quickly. don't wait for pty's. don't do a DAEMON dump.
- **EXIT**: exit; logout pty do a full core dump via DAEMON.
- **Ln**: load MAIN1. HIN in PDP11 &n.
- **IN-FILENAME, EXT**: load specified file in cpu &n.
- **Zn**: zero PDP11 &n. zero all but flags of window.
- **ZNF**: zero the flags of the window.
- **Sn**: printout status of PDP11 &n.
- **SlIn**: printout status of interrupt flags of PDP11 &n.
- **SG**: printout global status of counts.
- **R**: reread loader into shared area.
- **G**: get another PID because SYSINF went down.
- **Cn**: connect PDP11 &n.
- **PL-**: enter command into pty. (primary use is debugging tool in conjunction with loader.)
- **Dn**: disconnect PDP11 &n.
- **Kn**: kill NPK in progress in CPU &n.
- **Kn-abbbe**: send a checkpointed interrupt to the PDP11. (This interrupt requires acknowledgement from the PDP11) (used only for entry points which use the 'ackint' macro)
  - a=HR
  - bbe=VECTOR
  - c=function code.
- **In-abbbe**: interrupt the PDP11. (mostly a debugging tool)
  - a=HR
  - bbe=VECTOR
- **R**: print this text.

3. A "wake" is executed upon job DMACON by the DMA10 interrupt service routine. This event would occur when a PDP-11 had an error.

Since DMACON is implemented as an event processor, it only runs when called upon to do so; therefore, the only costs associated with DMACON in its dormant state are those of core occupancy. Furthermore, DMACON high segment may contain the standard FORTRAN. Thus, all currently executing FORTRAN programs have a sharable resident run-time system which reduces to a bare minimum the costs incurred by
locking DMACON in the DECsystem-11 memory.

Conclusions

The pMAIO is a very good communications device when used for different data structures, with very little modification of hardware it could become the prototype for all multiprocessor systems. However, given the current hardware, care should be taken in its use when block transfer devices may be used and its speed is not necessary.

Reference