ARCHITECTURE AND PERFORMANCE OF THE NEW CESR CONTROL SYSTEM

C. R. Strohman and S. B. Peck
Cornell University, Lab of Nuclear Studies, Ithaca, NY 14853, USA *

ABSTRACT

The new control system for the Cornell Electron Storage Ring (CESR) is based on a multi-port memory which can be accessed by many computers. The computers are either VAXes, which run user programs, or Xbus Processors, which move data to and from the hardware devices which are being monitored or controlled. The control system database is in the multi-port memory, and contains all of the data needed to communicate with various pieces of hardware.

HARDWARE

The hardware is centered around a multi-port memory system (MPM) on a VMEbus (fig. 1). Ten processors are connected to the MPM; up to 16 may be connected. Each processor can access the MPM as if it were a local device. The MPM contains 4 Mbytes of RAM, a semaphore board, a FIFO board, and a system controller.

User programs run on the VAX computers, which have VAX to MPM interfaces. Up to 32 processes on each VAX can independently access the MPM, either reading or writing data in the RAM or sending messages to the XBUS Processors through the FIFO board.

The XBUS Processors (XBPs) are VMEbus systems which move data between the MPM and the hardware which monitors and controls CESR. Each XBP has 1 Mbyte of local memory, a driver which controls the XBUS, and a VME to MPM interface. The XBUS is an in-house developed system of interface cards in crates linked to the computer via an eight-bit parallel differential bus. Crates are daisy-chained with a termination on the last crate of each leg of the XBUS.

MPM System Controller Board

The MPM System Controller resides in slot #1 of the MPM backplane. It contains a crystal oscillator to generate the 16 MHz VMEbus system clock (SYSCLK); a one-shot to generate the VMEbus system reset signal when power is applied or when the manual reset switch is pressed; and a timer which terminates bus cycles which take more than 1 micro-second. The cycle is ended by asserting the VMEbus bus error signal. This relatively short timeout interval prevents the processors connected to the MPM from having to wait so long that they incur local timeouts when another processor addresses a non-existent device.

The system controller also contains a bus arbitration circuit programmed with a round robin select (RRS) algorithm which supports up to sixteen bus requests. The arbiter is the only part of the MPM system which deviates from the VMEbus standard, which only provides for a 4-way RRS. The 16 Bus Request and Bus Grant signals are brought to the system controller through wires added to the VMEbus P2 backplane.

All of the devices which can be bus masters (devices which initiate data transfers) are designed such that they own the bus for only one data operation, either a read or a write. Arbitration occurs in parallel with data transfers, so that a new bus master may take over the bus as soon as the current bus master completes its operation.

MPM Memory Board

The MPM memory board contains 4 Mbytes of dynamic RAM with single bit error correction and multiple bit error detection. It supports 24 or 32 bit addressing and 32 bit data transfers.

MPM Semaphore Board

The semaphore board provides an array of test-and-set registers (semaphores) which can be used to implement mutual exclusion protection for the MPM memory. There is a semaphore for each longword (4 bytes) of memory. The address of the semaphore is determined by adding a constant to the address of the memory location which is being protected.

Semaphores are used when more than one process needs to write data to the same place in memory (a critical section). When a process needs to access a critical section, it first reads the semaphore associated with that memory location. If the process finds that the semaphore is SET, this indicates that some other process has already set the semaphore and that this process should wait. If it finds that the semaphore is CLEAR, this indicates that no other process is using the memory section. The act of reading the semaphore causes it to be set. When a process is finished with the critical section, it clears the semaphore by writing to it. The semaphores do not actually prevent access to a critical section. Critical sections must be identified and the programs using these sections must be written such that they use the semaphores. The need for hardware semaphores arises because a process can only own the VMEbus for one operation. If multiple operations were allowed, programs could use a read-modify-write sequence.

MPM FIFO Board

The FIFO (First In, First Out) board provides a means for implementing a message passing system among the processors connected to the MPM. The FIFO board contains 16 FIFO chips. Each processor in the system is assigned a specific chip, based on which slot in the MPM backplane the processor's interface board is located. Each FIFO is 9 bits wide and 512 words deep.

The FIFO board is designed so that a processor can write data (9 bits) to any number of FIFOs using only a single instruction. This is done by encoding the identity of the desired FIFOs in the upper 16 bits of a 32 bit word and placing the data in the lower 9 bits, and then writing the 32 bit word to the FIFO board. A processor determines if there is any data in its assigned FIFO by reading a status word from the FIFO board. The status word consists of the FIFO-NOT-EMPTY bits from each of the 16 FIFOs. The processor checks the bit corresponding to its assigned FIFO.

VME to MPM Interface

The VME to MPM interface consists of a master and a slave board which transfer data between a VMEbus processor and the MPM. The master board plugs into the XBP backplane; the slave board into the MPM backplane. The boards are linked by two 50 conductor cables which provide a 32 bit, multiplexed, address and data path along with associated control signals. The boards only allow longword (32 bit) data transfers.

The master board maps a range of VMEbus addresses to the slave. Currently, the range is from 300000 hex to FFFFFF hex, an address space of 13 Mbytes. When an address within the selected range appears on the master's bus, the address is passed to the slave. The slave stores the address in a latch and sends an acknowledge signal back to the master.

If the XBP is reading data from the MPM, the slave board requests ownership of the MPM VMEbus as soon as it receives a valid address from the master. When it is granted ownership, it read data from the addressed device, stores it in a latch, and releases ownership of the MPM VMEbus. This makes the MPM bus cycle as short as possible, since the slave gets the data and then holds it until the master is ready. When the master receives the address acknowledge, it removes the address from the cables, reverses the directions of data flow on the cables, and tells the slave to send the data. The slave places the data on the cables and signals the master that the data is valid.

If the XBP is writing data, it sends the data when the address acknowledge signal arrives. The slave board requests ownership of the MPM VMEbus as soon as it receives valid data from the master. When it

* Work supported by the US National Science Foundation

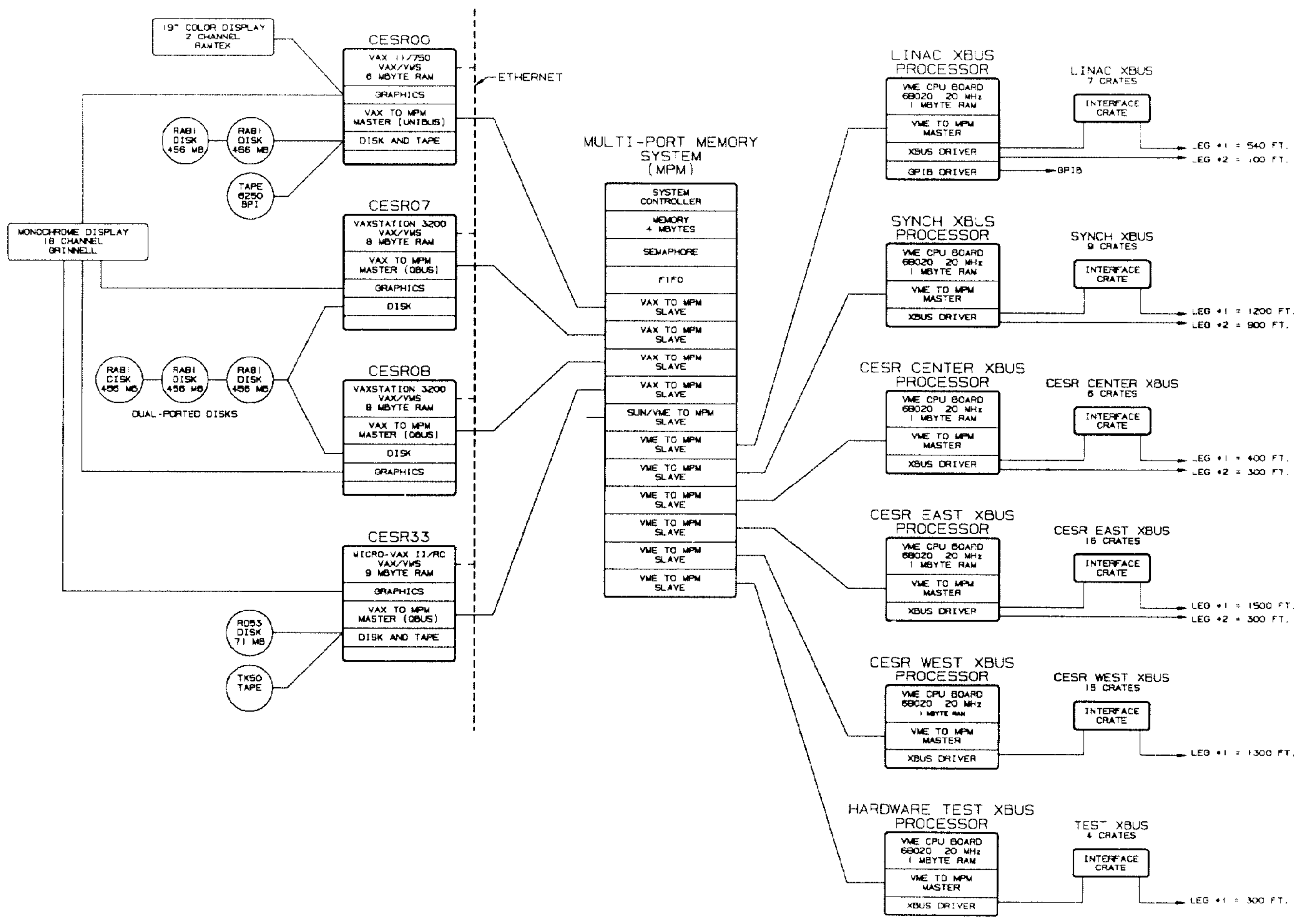

Fig 1. CESR control system hardware configuration.

is granted ownership, it writes the data to the selected device. It then releases ownership of the MPM VMEbus and signals the master that the data transfer is complete.

The RESET signal on the MPM backplane is passes from the slave to the master, where it connects to the XBP backplane. Therefore, a reset on the MPM causes a reset on all of the attached XBPs.

VAX to MPM Interface

The VAX to MPM interface consists of a master and a slave board, which transfer data between a VAX computer (either Unibus or Qbus backplane) and the MPM. The master board plugs into the VAX backplane; the slave board into the MPM backplane. The boards are linked by a 50 conductor cable which provides a 16 bit, multiplexed, address and data path along with associated control signals.

The interface is addressed as an I/O device, and uses 512 consecutive bytes in the Unibus or Qbus address space. The boards provide 32 sets of registers, each of which allow the VAX to access the large address space of the MPM. Each set of registers stores address and data information for an independent MPM access, so 32 different processes running on a VAX can use the MPM concurrently. Each set of registers consists of a status register, an address register, a data register, and a data register with auto-incrementing of the address register. Each register has an upper half (the two most significant bytes) and a lower half (the two least significant bytes). Since the width of the data bus on the VAX is only 16 bits, and the MPM only supports 32 bit operations, two VAX operations are required for each MPM data transfer. One of the two operations actually accesses a device in the MPM; the other operation accesses data stored on the slave board.
When the VAX starts a transfer, it first places the register number on the cable. The register number is determined by address bits 9 through 1 on the VAX backplane. The register number specifies a set of registers in bits 9 through 4 and a specific register within a set in bits 3 through 1. When the slave has stored the number, it acknowledges that fact. The master then either writes data to the slave or reads data from the slave. The data can be either an MPM address or MPM data. MPM addresses are read from or written to address registers on the slave board. MPM data is read from or written to data registers on the slave board. Access to certain data registers causes a VMEbus data transfer in the MPM, with the option of incrementing the contents of the address register.

For example, when the operation initiated by the master is a read from the upper data register, a 32 bit address (previously stored) is retrieved from the address register of the specified register set. The slave board requests ownership of the MPM VMEbus. When it is granted ownership, the address is placed on the bus and 32 bits of data are read from the selected MPM device and stored in latches on the slave board. The slave then releases ownership of the VMEbus. The upper 16 bits are returned to the master. A subsequent read from the lower data register will cause the lower 16 bits to be retrieved from the data registers on the slave and returned to the master.

CESR CONTROL SYSTEM DATABASE

Initialization

At present, four Mbytes of the MPM memory space are allocated for the control system database, and slightly less than half of this is in use. After this memory is initialized by one of the VAX computers, both the VAXes and the XBPs can read or write in it. The source for the database is in ten ASCII files, totaling about 15000 lines and 600000 characters. These describe about 20000 independently selectable elements in about 800 nodes. Each node contains information for 1 to a few hundred control points or memory elements. There are about 10000 hardware control points. A typical piece of this file looks like this:

```
n  'LIN 7BUN PHA' 1 7 40000000 93C7F9CF 8 0 0 5 6 11 16   0 22 !
c  cmd old val read raw-off-sca write raw-off-scale
```

```
c   wr-lowlimm-uplim-mode sts tim dly  cmdinc
c   xberrad rdmap  strptr xbus in-out addr xberrtim
f   5i f 2i f 6i 2h a16 3h
e   5(0) .5 0 0 2. -10 1023  0 1 0 10 0 0 1 ' BUNCH 1 PHASE ' 0 E1250E
    0 !
```

The line starting with 'n' specifies a node name (LIN 7BUN PHA), bit masks for bus systems and properties present, and XBP operation codes. 'c' lines are comments listing the properties. The 'f' line is a format statement which tells how to interpret the element initialization, 'e', lines. The 'e' line for the first element is shown.

The ASCII files is converted into the MPM format in about one minute. Since programs using the database may not be run during this time, a syntax checking program was created to check for errors in database additions or changes. The format types allowed are a variation on Fortran style, allowing floating, binary, octal, decimal and hexadecimal numbers, character strings, and repeated combinations of these. Hexadecimal is the preferred mode for hardware addresses, masks, and shifts, with decimal and float used for quantitative properties and scale factors.

Name Table

The central data structure of the MPM database is the 'NAME TABLE'. This is the sole location for all mnemonic node names, information about how many elements and properties the nodes have, and pointers to the data stored for each element of these nodes. It also contains the operation codes which are most often requested of the XBPs At present, the nametable entry for one node has 112 words of data: 3 of these hold the mnemonic name of the node, and 96 are allocated as pointers to element data. Each element property has a unique location within the name table for its pointer, even though most nodes use only a small fraction of the defined properties. Each data structure in the database has an entry in the name table, including the name table itself.

Lookup of a node mnemonic in the name table is done by hashing (by XOR of the 3 words of a name, modulo HASH_TABLE size) to produce a index in the HASH_TABLE. 90% of the time, this will contain the name table location of the desired name. If not, the HASH_LINK table will, at the same index, point to another HASH_TABLE entry. 99% of the name table entries are found with one such link.

Each of the 96 pointers to element data is either 0 (when the corresponding property is absent), or the address of a vector in MPM memory with one 32-bit word for each element in the node. This means that all data for a single property is contiguous, rather than all properties for a given element, to optimize vector operations. Element data is stored starting from top of available database memory. Additions to the nametable, hash and other system tables occur from bottom up, leaving about 2 Megabytes free space.

OPERATING SOFTWARE - VAX PROCESSORS

The CESR control system attempts to provide rapid and efficient translation of user requests into bus-level actions by minimizing the number of software layers, while enforcing rules in the structure of these layers to provide reliable operation. The flow of data and control from user request to hardware makes use of the global, MPM resident, database.

User-level programs call one of about 25 subroutines to read or write data to the hardware (via the XBPs), or directly in MPM. Naming conventions are used to make it easier to remember the relevant subroutine. Some frequently used routines are:

```
vxgetn('CESR QUAD CUR',1,98,readout_vec) !XBUS read
vxputn('CESR SEXT CUR',1,98,command_vec) !XBUS Write
vmputn('SYN MAILBOX',11,20,DATA) !MPM write
vmgoldn('LIN VERT CUR',1,12,DATA) !MPM get "OLD"
```

The subroutine names identify the system (v), select XBP or memory action (x or m), then direction (put or get) and finally any special properties selected. From the lessons of our previous control system, several limits and standardizations were imposed on such calls. Only a vector form, with both start and stop indices specified, exists for each action; all numeric arguments are 4 bytes; the character string used in the call is the same as seen by users in a directory; and all can be called as subroutines or functions (with status).

In general, these subroutines do not call any further levels of subroutines. To handle and make readable the large amount of repeated code required for MPM access, name conversion, data transfer, XBP wait loops, and error checking; standardized include files, accessing standardized variables, perform these actions. In other words, a set of in-line, fixed argument procedures, rather than function calls, are used to extend the Fortran-77 language. To make this aspect, and the whole system more secure, all variables are explicitly declared: "implicit none" is required.

A constant part for all these routines is an include file that translates a name into a nametable pointer using hash lookup, checks the start and stop element number against limiting values, defines the offset to first datum, and gets a mask that tells which XBPs are used, if any. If a subroutine requires only put/get from the MPM database, (no XBUS operations), then all that remains is shown below. (i# in column 1 indicates contents of one include file)

```
i1    longaddr=ptr_nam+offset_to_property
i1    MPM(base+adrof1)=addr_word1  !load address
i1    MPM(base+adrof2)=addr_word2  !in 2 X16 bits
i1    data_word1=MPM(base+datof1)  !fetch data
i1    data_word2=MPM(base+datof2)  !in 2 X16 bits
i1    longaddr=longdata+ptr_off       !addr of data?
c
      if(longaddr.lt.db_base) then      !test legality
       do i=0,num2-num1                 !clear on err
        datavec(i)=0
       enddo
       status=mpm_ilprop_bot-offset_to_property
       return                      !set error type, then return
      endif
c
i2    MPM(base+adrof1)=addr_word1  !load addr
i2    MPM(base+adrof2)=addr_word2  !in 2X 16 bits
      do i=0,num2-num1                  !load loop
i3     data_word1=MPM(base+autinc1)!read 16 bits
i3     data_word2=MPM(base+autinc2)!and incr adr
       datavec(i)=longdata              !user vector
      enddo
      status=1                          !success
      return
```

If XBP access is required, a immediate request packet (IRP) is assembled in the IRP_PACKET table area. The particular IRP used is allocated at program startup, and is retained throughout execution. An include file loads bit masks for select, start, done, and error status for all XBPs involved. The next packet entry is a code specifying an XBP operation. In some cases it is directly implied by the subroutine, otherwise it is found in the name table. The remainder of the packet contains the nametable pointer, and the first and last elements to use. If the subroutine has user supplied output data, this is now written into the node command data vector (not into the packet). Another include file writes a composite of IRP number and XBP mask to the FIFO board. This allows XBP operation to begin. Before starting to wait for completion, any MPM readout can be done, overlapping xbus processor time. For instance, reading out command values when both command and readback are requested in one call.

After writing to the FIFO, the VAX subroutine goes into a busy loop, reading the packet "DONE" flag, and sleeping when that seems efficient. When all "DONE" bits are clear, error-checking and recovery are done. If required, data written into element vectors by the XBPs is moved to the user's vector, and the subroutine returns. The entire call requires about 500 microseconds for the first datum, and about 25 - 60 microseconds for each following element, exclusive of efficiencies from multiple processors acting in parallel.

OPERATING SOFTWARE - XBUS PROCESSORS

The XBPs spend their time either fulfilling a request or waiting for a new request. When a message appears in an XBP's FIFO, the XBP reads the IRP number from the FIFO and calculates the address of the IRP. In the IRP, it clears its own bit in the START word (using a semaphore if more than one XBP is involved). The operation code is read from the IRP, and a specific action routine is called. The routine checks that there are valid pointers to all required properties. Data is then transferred to or from the XBUS, with manipulations (shifting, masking, etc.) as appropriate. After all of the data is moved, the XBP clears its own bit in the STATUS word of the IRP (if there were no errors) and then clears its bit in the DONE word.