SLAC–PUB–7201
June 1996

Flexible High Performance VME based Data Acquisition System for the ESA Physics Program[1]

Perry L. Anthony
Stanford Linear Accelerator Center
Stanford, CA 94309

and

Zenon M. Szalata
The American University
Washington, DC 20016

Abstract

We report on the first full scale implementation of a VME based Data Acquisition system at SLAC. This system was designed for use in the End Station A (ESA) fixed target program. It was designed to handle interrupts at rates up to 120 Hz and event sizes up to 10,000 bytes per interrupt. This report summarizes the development, implementation, and performance of the VME based Data Acquisition system.

[1]Work supported by Department of Energy contract DE–AC03–76SF00515 and National Science Foundation Grant Number 9313839.

1 Introduction

With the approval for 50 GeV upgrade to the A-line for precision nucleon polarized structure function measurements it became necessary to also upgrade the data acquisition system used in End Station A (ESA). The approved experiments to measure these structure functions, E154 [1] and E155 [2], would use "sled-ed" electron beams and the electronics would be implemented to be triggerless. This would allow the electronics to capture as complete a snapshot as practical of the event generated by each beam spill passing through the target. This approach would lead to large data rates and consequently to large collected data volumes.

The data acquisition system used in ESA prior to E154 was based on a DEC computer with a "Qbus" interface running the VAXeln real time operating system. The Qbus contained Jorway interfaces to the CAMAC A2 crate controllers in up to three data branches (beam/moller and two spectrometers). Additionally, an Ethernet interface was also located in the Qbus for sending data out for logging. Since the data was read into the processor memory via the Qbus and was also sent for logging via Qbus-Ethernet interface some Qbus cycles would be lost to bus arbitration. Typical CAMAC data transfer speed of $2.5\mu s$ per 16 bit word, was observed for the Jorway-411 interfaces used. This is equivalent to $800 bytes/ms$ data transfer rate. Running with $120Hz$ beam spill rate, one has $8.3ms$ between spills to read out data from CAMAC and send it off for logging. Since the transfer rates for reading data from CAMAC and sending data via Ethernet are approximately the same one would expect to transfer approximately $3.3KB$ of data in $8.3ms$ which corresponds to approximately $400KB/s$ data transfer rate. Bus arbitration would tend to reduce this rate. In fact we observed data transfer rates while taking data for the E142 and E143 experiments of the order of $300KB/s$. The limited data transfer rates in this Qbus implementation was due in part to the CAMAC speed and to the single bus (single data path) architecture. This rate, while sufficient for E142/E143 experiments, would clearly not be adequate for the E154/E155 experiments.

In 1994 we proposed a data acquisition system [3] that would be distributed over a number of VME crates, sharing data between these crates with the use of a reflective memory network. The detector data would be digitized in part in CAMAC and in part in VME. The readout of the CAMAC data would be done via the VSB bus (VME Subsystem Bus extension) which should not interfere with other VME activities. The inter-VME communication, synchronization, and data sharing would be implemented via the reflective memory network. Communication with the non VME components of the data acquisition system would be implemented via network (both Ethernet and FDDI). The decision to use CAMAC for most of the data read out for E154 and E155 was mainly a matter of economics; we had a large investment in both ADC and TDC systems from past experiments which could be reused for these two experiments. The estimated data rates for E154 and E155, and the fact that the data would be brought into the processor memory via a dedicated VSB bus made it possible to use CAMAC. For some future experiments, where even higher data transfer rates might be expected, the use of the CAMAC subsystem would have to be limited to some non-critical functions and the bulk of the data digitization and readout would have to be done either in VME directly or some other "fast" data bus.

This report describes the data acquisition system we designed and implemented for the recently completed E154 experiment. Some discussion of the reasoning behind the choices of individual components is given. All subsystems of the full data acquisition system are described in this note. The performance and planned improvement of this system are also discussed along with recently developed extensions to this data acquisition system to support test-beam experiments and bench test setups.

2 ESA Data Acquisition System Architecture

The ESA VME based data acquisition system (DAQ) is distributed over a number of computers. A block diagram of the DAQ, as it was used for E154, is shown in Figure 1. The VME components are but one subsystem in the ESA DAQ. Experiment control functions and experiment monitoring are done by a number of processes executing on a VAX VMS computer. Data logging to the SCS Silo system is done via the SLAC Staging system controlled by a remote data server job. This job executes on an IBM RISC6000 workstation and gets data from the VME subsystem via a FDDI network. Any number of online analysis jobs can be launched, on a variety of computing platforms, to analyze event data which is obtained from the local data server via a network connection.

2.1 VME Real-Time Subsystem

The VME bus was chosen for the ESA data acquisition (DAQ) since it provides high speed and flexibility. New, relatively inexpensive VME processor boards are continually being developed. Current processor boards have embedded controller options for secondary busses (SCSI, Ethernet, etc) allowing data to be logged without impacting data transfers on the VME bus. Multiple processor boards can be placed in the VME crate, allowing implementation of intelligent data acquisition with data compression and "pre-analysis" to reduce the volume of data written to tape. Relatively inexpensive interfaces to other buses (CAMAC, Qbus, FastBus, MicroChannel, SBus, etc) and networks (notably FDDI) are also available for VME allowing a VME based DAQ to be easily connected to other systems for remote data logging and high statistics online data analysis. We have studied the details of such a system with substantial input from other VME users at SLAC, FermiLab, CERN, and vendor technical representatives, and have designed a system with an expected capacity sufficient to handle data rates from detector to tapes in excess of $1 MByte/s$. In the event that greater performance is demanded by future experiments, this DAQ can be easily enhanced to accommodate those experiments.

It should also be noted that for the E142/3 experiments, the software code was developed in a modular way. This allowed machine independent modules to be directly ported to other systems. Large parts of the DAQ software from E142/3 written for the VAXeln operating system were easily modified to run on the new VME based DAQ system.

2.1.1 Description

A block diagram of the VME based DAQ system is shown in Figure 1. The front end of the DAQ is implemented in three VME crates. The real-time (RT) components of the DAQ system are housed in two VME crates—one, the local VME crate, is located in the ESA Counting House, and the other, the remote VME crate, is located in the south detector hut in ESA. The third VME crate contains

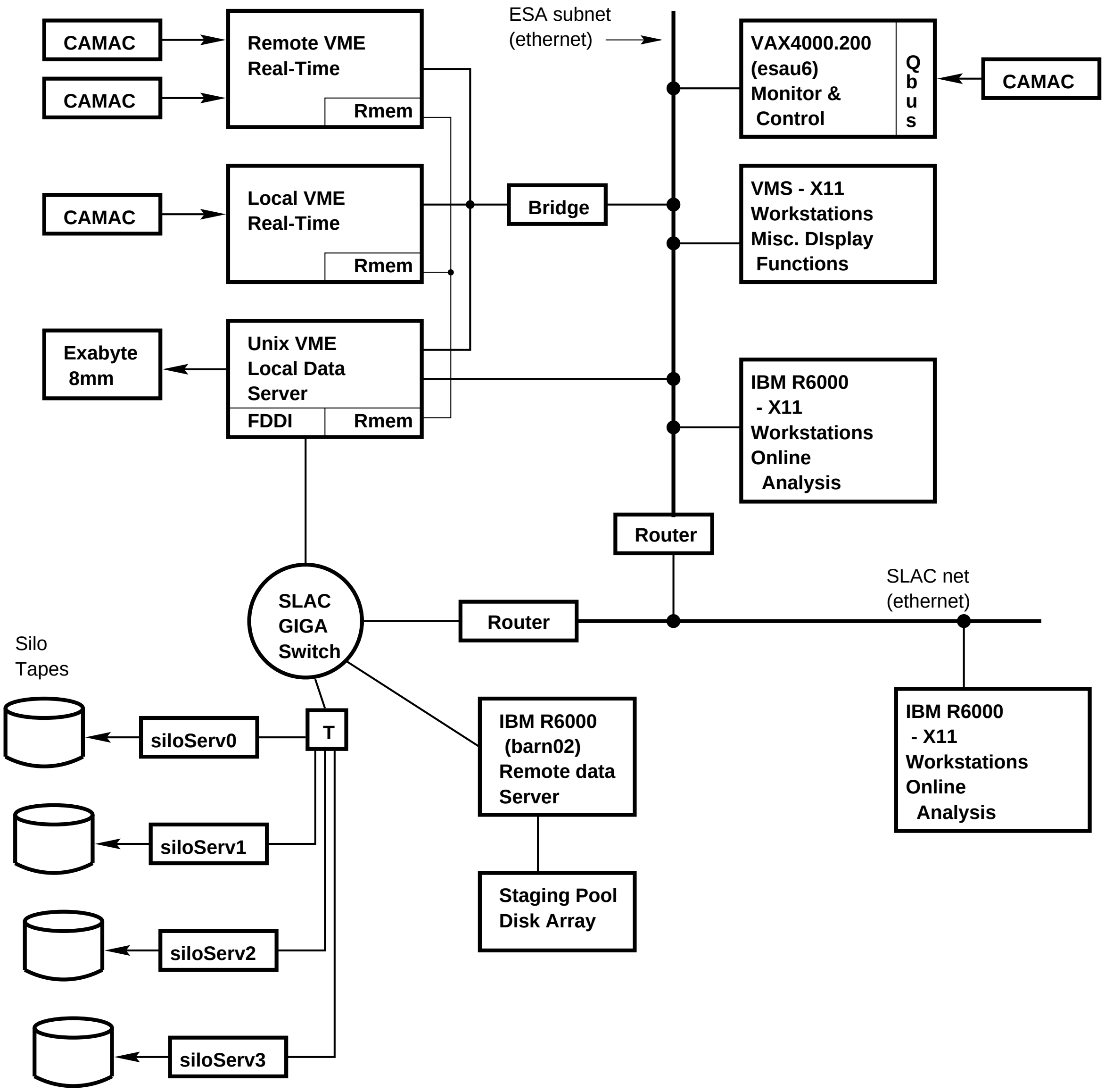

Figure 1: Block Diagram of the ESA VME Based DAQ Hardware Components

Unix processors and interfaces to the data storage systems and is located in the ESA Counting House. This third crate will be referred to as the Unix VME crate. These three VME crates are linked via a reflective memory subsystem (Rmem in Fig. 1. The VME components are shown in detail in Figure 2.

The VAX 4000.200 shown in Figure 1 contains a Qbus-CAMAC interface and is used to monitor and control the overall experiment. The VAX 4000.200 and its Qbus-CAMAC interface is located in the ESA Counting House. This part of the DAQ system controls the target position, High-Voltage, run type (pedestal, Moller, normal, etc), and monitors detector parameters such as NIM and CAMAC power supply voltages and Cherenkov gas pressures. The VAX 4000.200 is loosely coupled to the real-time part of the DAQ system via Ethernet. DAQ control commands (start, stop, run type, non-event data, data logging commands) are issued from the VAX 4000.200 and sent to the real-time components using the TCP/IP networking protocol.

2.1.2 Hardware

The RT local VME crate contains a CAMAC interface (BiRa 1711) to the beam data CAMAC branch. The remote VME crate contains two CAMAC interfaces (BiRa 1711) to each detector CAMAC branch and a Flash ADC system (four Struck DL515 modules) used to digitize signals from the Cherenkov phototubes. These crates and the Unix VME crate are connected via a reflective memory network, allowing data to be shared between all three crates. Each of the RT crates contains a real-time processor which controls the raw data readout and writing of this data to the reflective memory network. The Unix VME crate contains two UNIX processor boards. The first UNIX processor board is used to develop the real-time application software using the Motorola VMEexec software development environment based on the pSOS+ kernel. This UNIX processor board is also used for down loading the real-time application and provides the debugging and monitoring environment for the real-time processors. The second Unix processor is used to run a "Data Server" job, dubbed the local data server. It's main task is to log data by either shipping the data to the SCS Silo, or when the Silo system is not available, write the data to a local 8mm tape subsystem. It also serves a fraction of the data via network to analysis jobs running on a variety of workstations. The local data server is described below in more detail.

Each RT VME crate contains a Motorola MVME166 processor board which runs the real-time application software. The Unix VME crate contains two MVME197 processors. It also contains one Motorola MVME385 FDDI interface which is used for remote logging of data to the SCS Silo system. All three VME crates are connected via reflective memory modules. We chose the VMIC VMIVME5550 modules to implement the reflective memory network. The local RT crate also contains a VMIC VMIVME2532A 32 channel in/32 channel out TTL board for the input of signals (polarization bits, triggers, etc) into the event data, and passing out status bits of the RT system. Finally, each of the RT crates contains a CES VME/VSB bus display module, VMDIS8004, which is used to interrupt the RT processors with external triggers and to show bus activity aiding in setup/debugging/crash analysis.

2.1.3 Determination of the VME Components

The VME system was developed with the intent to interface with the existing CA-

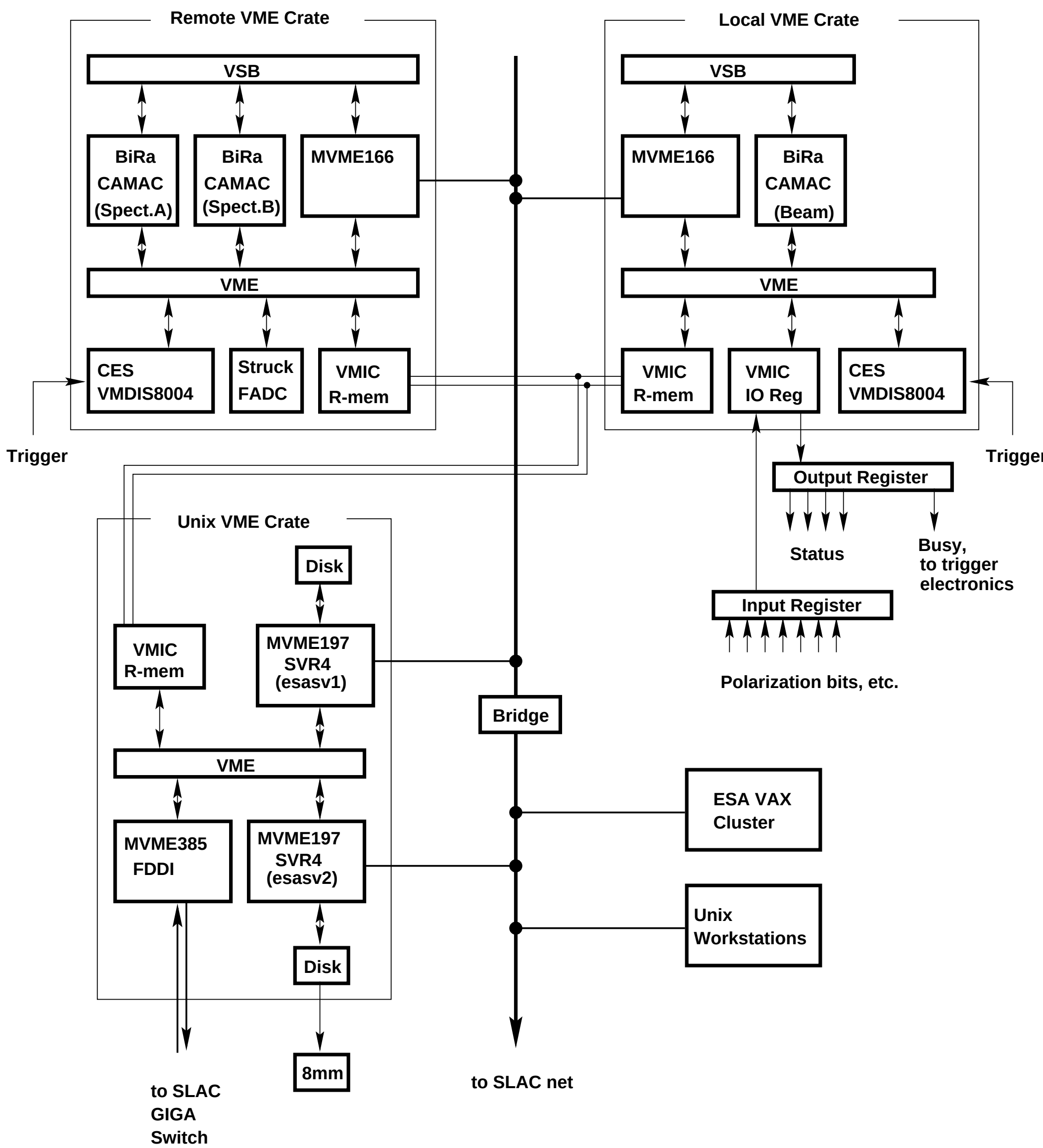

Figure 2: Block Diagram of the Real-Time and Unix VME Components

MAC data readout hardware. The CAMAC readout is controlled by A-2 type CAMAC crate controllers. Only two types of VME-CAMAC interfaces to the A-2 controller exist on the market, a unit made by Creative Electronics Systems (CES) located in Switzerland, and one made by BiRa located in New Mexico. The CES interface was developed at CERN and BiRa is marketing an interface developed at FermiLab. The FermiLab interfaces offer several enhanced features over the CES product for about 2/3 the cost. The most notable feature of the BiRa interfaces that is not available in the CES product is the capability to execute a list of CAMAC commands and to buffer the CAMAC data into local memory. A VME real-time processor then reads out the data from the CAMAC interface via the VSB bus (a VME Subsystem Bus extension). This list processing into local memory feature of the BiRa interfaces allow for simultaneous and non-interfering readout of several CAMAC branches (each branch controlled by a different BiRa interface). Given the amount of CAMAC data expected to be generated in the $50 GeV$ experiments, it is only by running several CAMAC branches in parallel that we can expect to approach the design goals of $1 MBytes/s$. For this reason, the BiRa interface was chosen to implement the CAMAC interfaces in the VME DAQ.

There are many processor boards available, most built around the Motorola 68000 series of processors. We choose the Motorola produced processor board because of the combined features of separate embedded processor control of each I/O bus (SCSI, VME, Ethernet, etc), use of the 33 MHz 68040 processor, and the VSB bus interface. This processor board with the VSB bus interface, the MVME166, is a modification of Motorola's MVME167 board (to add the VSB interface). The MVME167 boards are used at SLAC in the MCC machine protection system and the experience with these boards has been good. In addition, the CAMAC interfaces we are planning to use were developed using Motorola processor boards (the MVME165, and older version of the MVME166). We were thus assured that there would be no differences in how the VSB bus is handled that might cause problems interfacing the processor boards and the CAMAC drivers.

We chose the VMEexec real-time operating system for the DAQ based on several considerations. VMEexec contains built in driver support of a number of peripheral devices and is built around the pSOS+ real-time kernel. In a benchmarking study conducted by SSC [4] in 1991, the pSOS+ kernel compared favorably with other real-time kernels (VxWorks, VRTX32, LynxOS, and PDOS; VxWorks and pSOS+ were comparable and both generally better than the other real-time kernels). A measure of the real-time performance of a kernel is indicated by the length of time it takes the kernel to respond to a hardware interrupt (interrupt latency) and by the length of time it takes the kernel to switch between software processes (context switching). pSOS+ advertises (verified by the SSC study) an interrupt latency of 6 microseconds and a context switching time of 13 microseconds on a 68040 based processor board running at 33 MHz clock frequency. This offered a significant improvement over the VAXeln based Qbus system which had 15 and 200 microsecond times for these measurements. pSOS+ is used in the MCC machine protection system where the experience and opinion of pSOS+ is very good. Also, VMEexec is available for less than half the cost of VxWorks.

The real-time processor boards, UNIX development platform, VMEexec real-time operating system, and the FDDI interface form a tightly coupled integrated package. Motorola provides both technical and maintenance support for these components. This support is based locally (in Sunnyvale) and easily accessible. The technical support includes both software development and implementation of the hardware. Test facilities are also available at Motorola's local office. This support also extends to interfacing the Motorola processor board and VMEexec with other modules. The engineering staff at Motorola's local office have provided us with invaluable assistance during the initial setup of the VME system.

The remaining RT components (32 bit I/O, reflective memory, and bus diagnostic displays) were selected to fill particular needs of the DAQ. The VMIC VMIVME5550 reflective memory modules allow data to be shipped from the remote and local crates to the Unix VME crate for logging without CPU intervention and without interfering with data transport along the VME and VSB back planes. The ESA DAQ system makes use of TTL levels to control aspects of the data acquisition (pedestal, calibration, real, etc) and to reflect the operations of the real-time processor (computer busy, logging, etc). The VMIC VMIVME2532A provides sufficient input and output channels to accommodate these TTL levels. The CERN developed MVDIS8004 modules from CES handle external interrupting of the RT processors, and provide a visual display of operations on the VME and VSB busses to help in understanding system performance and aid in diagnosing system hangs/crashes.

The overall intent of this system design was to ensure that each component be capable of handling the desired data rate, and that interactions between the components do not adversely impact this rate. Careful examination of the VME and component boards timing specifications lead us to believe that, subject to the limitations imposed by the slow CAMAC bus, this system should be able to handle data rates well in excess of the $1MByte/s$ design rate. A discussion of the expected and actual performance of this system follows.

2.1.4 Real-Time DAQ Software

As has been noted, the Real-Time operating system is built using VMEexec. Figure 3 gives a block diagram of the interrupt service routines (ISRs) and process modules that comprise the DAQ RT software. Each box in this diagram is a separate process running under the VMEexec pSOS+ kernel and communicates with the other processes using kernel supplied signaling mechanisms and message queues. All processes have access to all of the physical memory in the VME system, so that shared memory sections are easily implemented by passing the starting address and length of these sections to all processes that need access to these data areas (this is true for both ISRs and process modules). The diagram is arranged so that higher priority processes are closer to the kernel, that is, the ISRs have higher priority than the DAQ RT processes and CAMD and MEMM have higher priority than the other DAQ RT processes. Each of the CPUs in the RT system is running an identical set of these processes. These CPUs coordinate their activities and communicate with each other via flags and data buffers in the reflective memory modules.

In response to an experiment trigger the Real-Time software executes a list of CAMAC and VME operations and transfers detector data from read-out electronics into event buffers in the reflective memory modules. This list of CAMAC and VME operations is configured from a data base setup file.

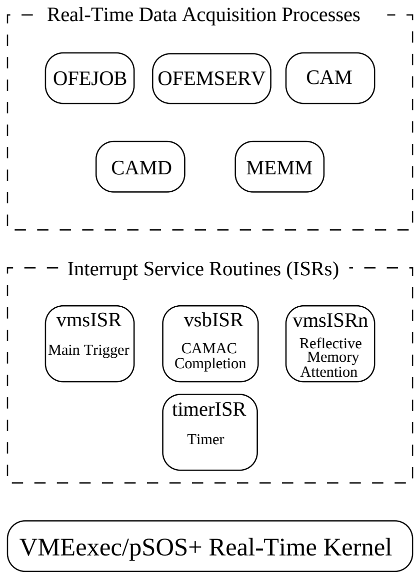

Figure 3: Block Diagram of the Real-Time Software Processes

The type of module and operation to be performed are stored in this setup file. The Real-Time software reads this file (when the Real-Time system is loaded and also in response to a "read data base" command) to set up its internal data base of operations (initialization operations, trigger types, and the list of operations to be carried out for each trigger. Changes in the type and operation of the read-out electronics are then easily accommodated by the Real-Time system by simply changing the setup file and reloading it.

The functions of the individual Real-Time processes and ISRs are as follows (listed in order of decreasing priority):

- **Interrupt Service Routines**

  timerISR: This routine provides elapsed time interrupts with a resolution of $50\mu s$. The VMEexec kernel provides the standard UNIX timing with a $10ms$ time resolution which is insufficient for Real-Time needs. This ISR is connected to a timer on the VMEchip2 which increments at $1\mu s$ intervals. A shared memory data structure is implemented through which the Real-Time processes can request that they be signaled by this ISR after a specified time interval expires. This data structure has room to contain several outstanding signal requests from different processes.

  vmeISR: This routine is connected to the interrupt on the CES VM-DIS8004 module. It signals the appropriate Real-Time process, usually CAMD, when an experiment trigger is received. The Real-Time process to be signaled is specified in its configuration data structure.

  vsbISR: This routine is connected to the VSB bus interrupt. It queries the devices on the VSB bus (the BiRa 1711s) to determine which one has generated the interrupt. This routine then signals the appropriate Real-Time process as set in its configuration data structure (if the system is in data taking mode then CAMD is the process specified in this structure, otherwise CAM is usually the specified process).

  vmeISRn: This is a set of four routines (vmeISR0 though vmeISR3) which are connected to the four interrupts on the reflective memory module. These interrupts are used to signal some kind of change in the reflective memory (e.g. end of data write or new message data) and to synchronize the Real-Time CPUs. Each of these routines signals the appropriate Real-Time process as specified in their configuration data structures.

- **Real-Time Processes**

CAMD: This process waits on a trigger (signal from vmeISR) and starts the BiRa 1711 CAMAC interfaces processing their CAMAC lists. It then reads event data from VME devices (the flash ADCs) and waits for the BiRa interfaces to signal completion of CAMAC processing (through the vsbISR). This process then transfers the CAMAC data from the BiRa 1711s to local memory and signals MEMM that there is data ready to be transferred to the reflective memory. Finally this process resets the BiRa 1711s and waits for the next trigger. This routine keeps diagnostic statistics on both its operation and the performance of the read-out electronics (hardware errors, data overflows, etc).

MEMM: This process waits for a signal from CAMD that there is event data to be moved into the reflective memory. The Local and Remote VME crates synchronize the packaging of the event into buffers in the reflective memory module via flags and pointers also located in the reflective memory module. This routine keeps diagnostic statistics on its operation.

OFEJOB: This process is the main DAQ RT process and is started first by the kernel. It registers all ISRs with the kernel and sets up and initializes all shared data areas. The VME and VSB devices are also initialized in this process. This process then waits for the remaining processes to be started by the kernel and communicates the location of shared data areas to all processes. After all Real-Time processes are running, OFEJOB drops into the background and only handles message passing between the Real-Time crates though the reflective memory (one of the vmeISRn routines is configured to wake OFEJOB when messages are pending in the reflective memory).

OFEMSERV: This process handles TCP/IP messages. It creates a connection to a network message server (netServ, which is described in a later section) and waits on messages directed to the Real-Time processes. It receives these messages and either acts on them or forwards them to CAM for processing. All replies to the network from the Real-Time system pass though OFEMSERV. This routine also gathers the diagnostic statistics from CAMD and MEMM to form an "overview" of the performance of the Real-Time system and read-out electronics. This overview is transmitted to the experiment control process for display.

CAM: This process reads in via the network a list of CAMAC and VME data sources from a data base setup file. From this list, it then sets up the CAMAC operation lists for the BiRa 1711s. CAM also performs single CAMAC operations through the BiRa 1711 interfaces (CAMAC module initialization and CAMAC operation requests received from the network). Prior to beginning a data taking run, this process configures the BiRa 1711s with the appropriate CAMAC list for the operations to be performed for each trigger during that run.

2.1.5 Real-Time DAQ Performance

In general terms, the VME based DAQ system performed quite well. The combination of the VME hardware and VMEexec Real-Time Kernel proved to be a very stable platform. There were no DAQ crashes caused by operating system problems, and there were no VME associated hardware failures during the experiment (CAMAC module failures where expected and provided for). Performance of system as measured by the maximum data throughput rate was, for fixed CAMAC length transfers (test data), approximately $750 KBytes/s$. The system was not able to support data rates at the design goal of $1 MBytes/s$ due to the performance of the CAMAC branch and interface part of the VME DAQ system as detailed below. During normal conditions with varying event sizes, the maximum data rate was approximately $680 KBytes/s$. The reduction in rate with respect to that seen with fixed event size is mainly due to the following: The accelerator produces electron bunches spaced $8.3ms$ apart. If a large event takes $8.3ms$ or more to be transferred from CAMAC, the trigger from the following beam pulse is suppressed. This adds, at a minimum, an additional $8.3ms$ (the time until the next electron bunch) of deadtime into the DAQ system, reducing the overall maximum data rate.

The performance of the various software and hardware components are detailed below:

Software: During E154, the only problems with the Real-Time software portion of the DAQ system occurred when the amount of data read in from each spectrometer branch neared or exceeded 5000 bytes per branch while trying to run at 120 events per second. Under these conditions, the Remote and Local VME CPUs could lose synchronization and either start mis-building events (Local crate data from one event combined with Remote crate data from a previous or following event) or the system busy would not be cleared because of a "race condition" where each VME crate was expecting the other to clear the system busy. Both of these situations could be cleared by issuing a "Pause" command from the controlling program and normal data acquisition would be resumed. However, if the volume of data remains high, the Local and Remote VME crates will continue to lose synchronization. Because most of the data under these conditions were due to noise in the detectors that made physics analysis extremely difficult, the data rate was usually limited by lowering the beam current to levels where detector noise was tolerable and consequently these synchronization problems did not occur.

FDDI Interface for data logging: The FDDI interface is controlled by a process running on one of the CPU boards in the UNIX VME crate. This process takes data from the reflective memory and transmits it via the FDDI interface to the remote logging job. Placing the FDDI interface in a separate VME crate from the Real-Time system avoided possible VME data bus contention problems between the Real-Time processes (moving data to/from the reflective memory) and the process controlling the FDDI interface (moving data from the reflective memory to the FDDI interface). This module and the connected network/workstations had no difficulty in handling data rates on the order of $2 MBytes/s$ and consequently had no impact on the VME Real-Time performance.

Reflective memory link: Under test data simulation, the maximum rate of data transfer via the reflective memory link was $1.4 MBytes/s$. This was lower than the expected rate of $5MBytes/s$. The reflective memory modules transmit data on their private bus at different speeds depending on the length of this bus (length of cable between modules). In the initial DAQ design, the modules were expected to be able to run at their $5MBytes/s$ data rate. When the actual system was assembled, the cable lengths for the reflective memory bus were longer than originally planned due to a change in the VME crate location imposed by electronics needs (to minimize signal cable lengths from sensitive detectors). Consequently, the $5MBytes/s$ rate could not be used and the modules were configured to run in their $2.5MBytes/s$ mode. The observed rate is less than $2.5MBytes/s$ due to CPU overhead to process the data and "handshaking" between the Local and Remote crates on the reflective memory bus. While $1.4MBytes/s$ is above the design goal, this did reduce the safety margins significantly and required that the rest of the system perform optimally to ensure a maximum rate of $1MBytes/s$.

CAMAC interfaces: CAMAC created the major bottleneck in the DAQ system in achieving the designed goal of $1MBytes/s$. This is due to the following reasons: The CAMAC specification implies that it takes $2.0\mu s$ to transmit 16 bits of data between a single crate and the CAMAC branch controller. However, this time significantly increases for successive crates in a multi-crate system, increasing to as much as $3.8\mu s$ per 16 bits data for a fourth or fifth crate on the system. Unfortunately each crate in the E154 system generated approximately the same amount of data so that the order of the connection of the crates to the BiRa 1711 could not be used to minimize data transfer times. If there are, for example, 2000 words of data in one branch, it takes approximately $6ms$ to transfer that data to the VME system in a multicrate setup — 30% longer than for the same amount of data in a single CAMAC crate.

The BiRa VME/CAMAC interfaces that were used to control the E154 CAMAC branches can be operated in two different modes. In the first mode, the 1711 stops processing its list of operations and interrupts the controlling CPU after each CAMAC operation is completed. The controlling CPU can then read a word count register to determine the number of data words transferred into the 1711 by that operation, transfer the data from the 1711 memory to local data buffers, and restart the list processing of the 1711. In the second mode, the 1711 carries out a number of CAMAC operations, storing data is separate predefined regions of its memory. After a predetermined number of operations, the 1711 stops list processing and interrupts its controlling CPU. The number of data words transferred for each CAMAC operation is not stored, instead the controlling CPU must scan the data as it is transferred out of the 1711 to determine where each block of data ends. The controlling CPU then resumes the list processing of the 1711. For both of these modes, the transfer of data out of the 1711 can be done while the 1711 is processing the next item (or items) in its list, if the list is constructed in such a way that different regions of 1711 memory are being used. Obviously, in the first mode of operation the transfer of the data from the 1711 is faster than in the second mode where some logic test must be carried out on each word of data transferred. However, the first mode of operation generates more interrupts to the controlling CPU, possibly hampering other activity that the CPU must carry out (such as transfer of data to reflective memory). If the CPU is controlling more than one BiRa 1711, the interleaving of interrupts and data transfers for each 1711, coupled with the kernel overhead to process each interrupt, can make the first mode of operation less efficient than the second. This was in fact observed for this DAQ system where fiscal constraints required that only one CPU board could be used to control both 1711 interfaces in the Remote VME crate. Because of this, the 1711s were operated in the second mode. This resulted in lower 1711 to VME event buffer transfer rates than originally expected.

These two factors, CAMAC data transfer speed and 1711 operation mode, contributed to a significantly reduced data rate from what was originally designed.

2.1.6 Planned Improvements

Since the major bottleneck in the VME based DAQ system is the performance of the CAMAC branches and interfaces, improvement in performance can only be achieved by increasing the speed at which data can be read out of the CAMAC based electronics. One way to speed up the read out of the CAMAC branches is to place a second VME CPU board in the Remote VME crate so that each BiRa 1711 CAMAC interface would be controlled by a separate CPU. This would allow each CAMAC branch to operate in a parallel fashion. However this is quite expensive, not only in terms of the cost of the additional CPU board, but in terms of the complexity in the software introduced in trying to keep three VME CPU boards synchronized. Fortunately, the bulk of the CAMAC data is contained in modules that support a faster readout bus — an ECL based bus. This bus operates separately from the CAMAC data bus and can transfer data at up to 20 times the rate of the CAMAC bus. This bus can be connected directly to a memory module in the VME crate and data transferred to it without any control from the VME CPU. While the existing VME CPU is reading data from the non-ECLbus CAMAC modules via the 1711s, the bulk of the CAMAC data is transmitted independently to a VME memory module via this ECL bus. After the transmission of the data over the ECL bus is complete, the VME CPU can quickly transfer that data into its event buffer. The implementation of the ECL bus costs only one third that of an additional VME CPU and requires no software changes since the existing software, through its setup data base as described earlier, already handles the addition of new VME data sources such as the VME memory module used by the ECL bus. Because of the low cost and ease of implementation, the ECL bus will be used to boost the DAQ performance for E155. It is anticipated that this change will eliminate the CAMAC bottleneck for E155 and allow the system to operate at rates in excess of $1 MBytes/s$.

2.1.7 Support for Parasitic/Test Experiments

Between fixed target runs, when the Electron Accelerator is being used for running the $e+e-$ colliding beam experiment (SLD), it is possible to run specialized experiments in the End Station A. Photons given off by the electrons in the primary beam are converted into electrons which can be directed down the A-line into End Station A. This provides a very low intensity beam (about 1 electron per beam pulse) in the End Station. E146 used such a beam in a very successful investigation of the LPM effect (see [5]). Because of the success of E146, other experiments have been proposed using this low intensity beam (current users are SLAC's Radiation Physics department, and the GLAST collaboration [6]). Because the Remote VME crate in the DAQ system is fixed in one of the detector huts used for E154 and E155, a second Remote VME crate has been implemented to allow the VME based DAQ system to be used by these low intensity beam experiments. This second Remote VME crate can be placed in a convenient location for the low intensity beam experiment. Both of these Remote VME crates are identical in operation so that the same software can be used on both. The Local VME crate's reflective memory network is connected to the appropriate Remote VME crate to complete the DAQ system. Differences in CAMAC and VME data sources between E154/5 and these low intensity beam experiments are already handled in the DAQ software by way of the database setup file that is loaded when the DAQ system is started.

In addition to the VME based DAQ system, we have also assembled a standalone, portable VMS/Q-Bus based DAQ system. This system can be easily relocated as needed and requires only an EtherNet connection to allow for control and data logging. The details of this system will be discussed in a later section as much of its software is based on the Data Server software elements discussed in the next sections. This system can provide DAQ support for small experiments in ESA and other beam lines. Its maximum data handling capability is about $2000 Bytes/s$ at a trigger rep-rate of $30Hz$. This DAQ system like the VME based system, uses a database setup file to configure its CAMAC operations and provides a flexible DAQ platform.

2.2 Description of Data

This data acquisition (DAQ) handles three types of data: event, non-event, and command/synchronization. Non-event data is a block of data which usually describes the status of a subsystem which is part of the experiment, e.g. spectrometer magnets and power supplies. Some non-event data is a collection of numbers, read from various sources and grouped for convenience. The first two data types are packed into tape record sized blocks and are logged to tape. In this section we describe the data flow, the structure of the packed blocks of data, and the structure of the event data.

2.2.1 Data Flow

A block diagram of the data flow is shown in Figure 4. In this figure the direction of data flow is indicated with the use of arrow heads. Hence, if data can flow in both directions along a particular line, then the arrow heads are drawn on both ends of that line. Note that the data which is logged to tape flows in one direction only. Command and synchronization data flow, typically, in both directions.

As was described earlier, all VME crates are "linked" with a reflective memory network which is used to share data. Data read by the real-time processor in the Remote crate is placed in the reflective memory module in that VME crate. The reflective memory module transmits this data at approximately $2.5 Mbytes/s$ to the Local crate where it is combined with beam data, and the complete event is then available for logging in the Unix VME crate. In Figure 4. the flow of commands and other synchronizing data is shown with dashed lines. All commands originate in the Monitoring & Control subsystem which is described below. The Monitoring & Control subsystem also sends non-event data to the local data server for logging to tape.

The local data server job packs the event and non-event data into tape record sized blocks and sends these blocks to the remote data server job via a FDDI network connection for logging to tape. Figure 4 shows that there are two links between the local and remote data servers. One of these links is used to send data for logging, the other is used to exchange messages/commands. The local and remote data servers were designed with separate data and message links so that if the remote data server can not keep up with the data rate, it can block the data link while still being able to send and receive commands and messages. All commands to be executed by either data server are sent from Monitoring & Control subsystem to the local data server via netServ, a program which facilitates message exchanging between various parts of the DAQ via network. NetServ is described in more details below. Commands executed by the local data server are acknowledged directly by sending a reply via netServ. Commands to be executed by the remote data server are forwarded via the message link. In Figure 4 a one way message link is shown between remote data server and netServ. This link is used to acknowledge commands issued by the Monitoring & Control subsystem which are processed in the remote data server. The local data server also sends a sample of the event data to analysis jobs for online data processing. Data is sent to analysis jobs on request only.

The remote data server receives data blocks and commands from the local data server. It writes the data blocks to a disk file previously opened and cataloged with the SLAC staging system in response to a *begin run* command. When an *end run* command is received this file is closed and a new file is opened and cataloged with the staging system. Finally, when the *close tape* command is received a staging command is issued to write a list of disk files to a specified silo tape. At this point the staging system mounts the tape and starts the transfer of data from disk to tape and the remote data server is ready to open next disk file and continue writing data to disk.

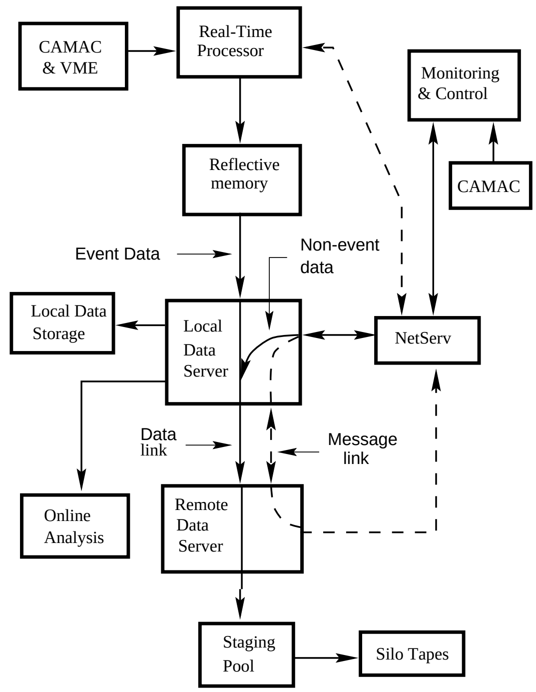

Figure 4: Data Flow Diagram

2.2.2 Data Structure

Tape Blocks

A file on disk or on tape constitutes one experimental run. Each file consists of a number of data blocks which correspond to tape records. The first block of data in each file, called an extended run label, contains an ASCII coded null terminated string. This string consists of information needed to describe the internal representation of the data as it is written to tape. Since an offline analysis job "knows" what processor it is running on, it uses the information in the extended run label to decide if the data needs to be transformed into the "native" data representation of that processor. Online analysis jobs do not need the information in the extended run label since each network message they receive is fully described, this includes information on data representation.

All other data blocks in each file on tape contain packed data, either event or non-event, or both. Non-event data could be either short labels or blocks of monitoring data such as scalers, high voltages, spectrometer magnets, some constants, etc. The structure of these blocks is shown in Figure 5. The first two integers, each four bytes long, are the run number and the record number. The last two integers in the block, each four bytes long, are a check sum and the run number. In the past when the data was written to 9 track magnetic tapes the check sum was calculated for each tape record to assure data integrity. Modern magnetic tape drives perform ECC thus obviating the need for check sum calculation. To preserve compatibility, this check sum, although no longer used, is still part of the data structure. The actual data is preceded by a four byte data descriptor which consists of the data length in bytes and data type — an integer value which identifies this data uniquely. The "Any Data" in Figure 5 can be either event or non-event data. The structure of the non-event data is not described here since it can vary from experiment to experiment.

Note that in Figures 5, 6, and 7 the bytes of data are numbered right to left which is more akin to the VAX/VMS data representation as compared to the Unix data representation.

Event Data

As already stated above, the "Any Data" part in Figure 5 could represent event data. Event data is really an aggregate which consists of at least one of the following: beam data, spectrometer A data, or spectrometer B data, these are called event data components. The event data is composed of up to three of these components, each consisting of a four

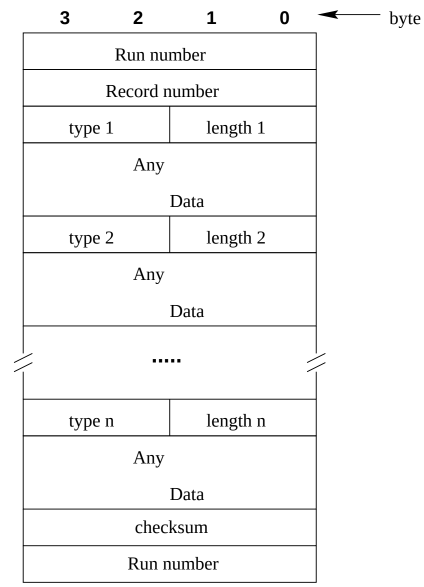

Figure 5: Tape Block Structure

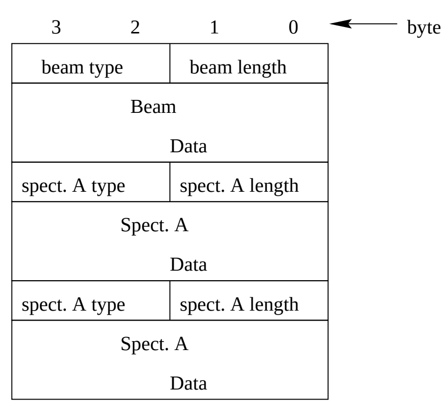

Figure 6: Event Data Structure

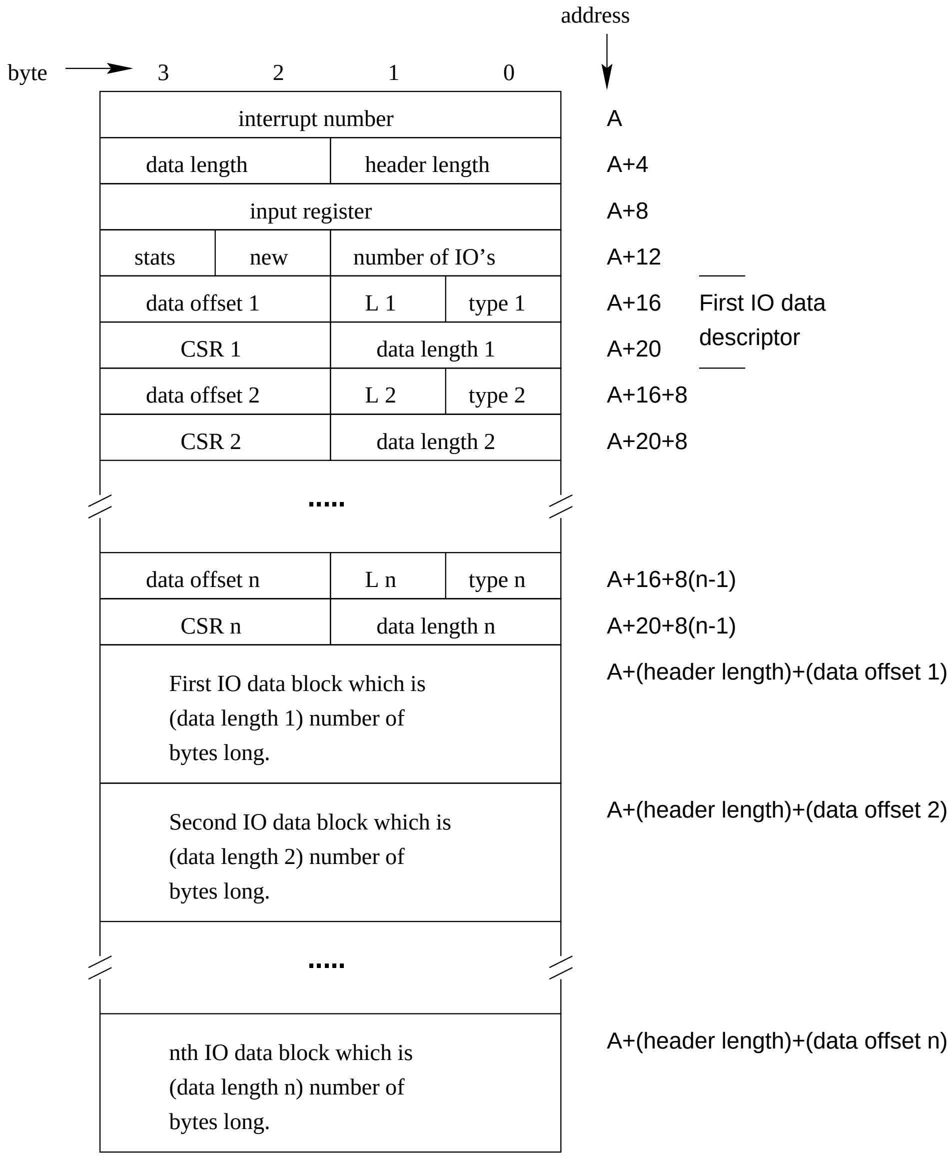

Figure 7: Event Component Data Structure

byte data descriptor followed by the actual component data. This is shown in Figure 6.

The structure of the actual data for any of the three components is shown in Figure 7. It consists of an event data header, a list of IO descriptors, and a list of data blocks. The event header is sixteen bytes long and consists of the following fields:

interrupt number: an unsigned integer value sequence number for this event data. This value is the same in each component of the event data.

header length: the length in bytes of the event header and the IO list.

data length: the number of bytes of the event data. This value is the sum of the data length in each IO descriptor (see below).

input register: contains 32 bit information input via a hardware device, these include the beam polarization information.

number of IO's: specifies how many IO operations were performed.

new: bit 0 in this field should always be 1, a value of 0 indicates stale data. The remaining bits 1 through 7 are used by the data unpacking routines in the analysis job to communicate presence of abnormal conditions in the event data to the data analysis routines.

stats: consists of eight status bits. Bit 0 when set marks an event as truncated because it exceeded the maximum buffer size. The remaining bits 1 through 7 are reserved for future use by the real-time programs.

Each IO operation performed during data acquisition is summarized in the corresponding IO descriptor which consists of the following fields:

type: contains a unique integer value which is used to chose the appropriate data decoder when the data is being analyzed.

L: specifies the data word length for each IO which is used to properly transform the data internal representation if needed.

data offset: used to locate the start of data for this IO within the event data.

data length: specifies number of bytes of data read in this IO. Note that for a nondata-transfer IO this field would be 0 and there would be no corresponding data block.

CSR: contains bits which reflect the status of the IO operation performed; the bit assignment is IO device dependent.

2.3 Networking Protocol

As the computing environment moved from a single computer to a cluster of networked units, the data acquisition used in ESA experiments was periodically redesigned to parallel this trend. In the mid 1980's we moved from a single computer based DAQ to a distributed DAQ. The first distributed version was modest in scope, only the online analysis part was moved from the VAX11/780, which was then our main computer, to two VS2000 workstations. The connection between these analysis workstations and the main part of the DAQ was implemented via DECnet. At that time we formalized the concept of a virtual network connection or virtual link. The virtual link concept was implemented as:

- a standardized way of packaging network messages (datagrams) which includes source and destination "addresses".
- a general purpose network server, called netServ.
- a standardized networking inquiry and handshaking protocol.
- a set of command codes to be used by cooperating clients and servers.

The network server, netServ, establishes itself as a well known network object (TCP port in later versions) and accepts requests for connections from client programs. When a connection or link is established between a client and netServ, the client is given an identifying number. A client sends messages on its link to netServ. The destination address determines whether a message is processed by netServ or is forwarded to the appropriate client. At any time a client can request that netServ broadcast a message to all other clients currently connected to this server. Most clients connect to netServ with the purpose of exchanging information with another client, with netServ serving as an intermediary. In this case a client, after establishing the connection with netServ, asks that an ID query message be broadcast to all other clients. Only one client process responds to the requesting client by sending a reply addressed to the requesting client. This reply is forwarded by netServ to proper destination. When the requesting client receives the reply, both clients then have established a virtual link and can exchange messages with each other via netServ.

One of the major benefits of the virtual link concept is that the code for creating and maintaining a link is standardized and is located in a general purpose library. The programs which wish to exchange data can be easily given the capacity to use a virtual link by calling a few standard routines. This keeps the coding simple. The standard routines used in the ESA environment for this purpose allow creating links, establishing a virtual connection, maintaining the virtual connection, reestablishing the virtual connection should it get broken, waiting for messages with an optional timeout, sending or receiving messages, maintaining the link with netServ, and reestablishing the link should it get broken. However, this scheme is not suitable for two programs that need to exchange large volumes of data since the overhead due to sending the data via netServ is not negligible. In such a case one of the programs is coded as a network server the other as a client. Such server - client pairs can exchange data with maximum efficiency because no intermediary, like netServ, is involved in the process. Some of the processes used in both the local and remote data server jobs are coded as network servers. These are described below.

Another network server, mserv, was coded to control two Metheus Ω300 graphic display devices which are interfaced to the Qbus of our VAX4000.300 computer through DRV11W cards. Mserv accepts TCP connections from a number of clients and paints the data received from a client on a specified unit, which could be one of the two display devices. This display subsystem was integrated into the DAQ. The DAQCntrl control program, described below, establishes itself as the only client of mserv and sends experiment overview data to mserv to be painted on both units for experiment monitoring.

2.4 Data Servers

A data server job consists of a parent and a number of child processes. It performs two main tasks: it moves data from an in-

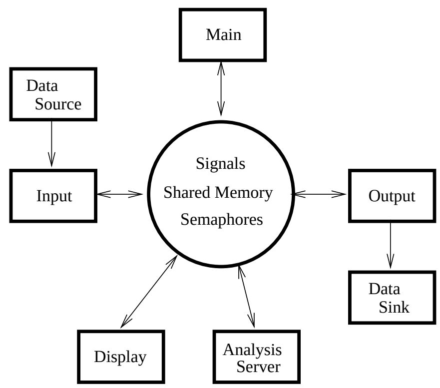

Figure 8: Block Diagram of a Data Server Job

put device to an output device and it serves data to analysis jobs. The parent process creates all interprocess communication structures, creates all child processes and enters an infinite loop. While in the infinite loop it maintains a log file in which entries are made from all processes in the job. It also waits for any of its child processes to terminate in which case it cleans up and terminates the job. Child processes read and write the data and handle communication with other parts of the data acquisition system. A block diagram of a typical data server job is shown in Figure 8, where the round box represents all three interprocess communication mechanisms used. A data server uses a number of shared memory segments to share data, semaphores to synchronize access to some resources, and signals to alert the process being signaled that some action is required.

The server job also makes sure that both the event data and the non-event data are written to tape in a consistent data representation which is in the native data format of the processor on which the event data was collected. In the case of E154 the data representation is big endian and IEEE floating data format. For E154 the non-event data originated on a VAX computer and was transformed to big endian and IEEE in the local data server before it was packed into larger blocks.

2.4.1 Data Server Architecture

A server job consists of a number of processes: main loop, input, auxiliary input, output, display and analysis server. The actual configuration of a server job is declared in a setup file. Several flavors of input and output processes are available, and it is relatively easy to add new ones to the list if needed. Presently the following flavors exist:

- input from an archive device. This could be a disk file or magnetic tape.
- input from network device. This process is a network server. This is the standard input process in the remote data server.
- input from memory and network devices. This is the standard input process in the local data server.
- auxiliary input from network of event data with a network connection to netServ. It gets event data via network from a data source and puts the data in a memory buffer which can then be read in by the input child process from memory and network process. This process is used in the local data server when the event data is generated in the Portable VMS Front End described below. An example of using this child process can be found in section 2.7 and is shown in Figure 10.
- output to an archive device. It writes to a disk file or magnetic tape.

- output to an archive device with a network connection to netServ. It writes to a disk file or magnetic tape. The network connection is for command message acknowledge. This is the standard output process in the remote data server.

- output to a network device in units of tape records. This is the standard output process in the local data server. It sends data blocks to the remote data server.

- analysis server process. Sends a fraction of the available data to a number of connected analysis jobs.

- output to a network device in units of discrete spill data blocks. This process can be used in a data server configuration where data is sent to an analysis type job. The difference between using this process and serving data to analysis jobs with the analysis server process is that this process accepts a single connection only and will send all available data.

- diagnostic display process.

A basic property of the server is that it has two multirecord data buffers, one for event data and some messages and the other for passing control messages. The data buffer can be configured to have a record size as specified in a setup data file. This data buffer record size becomes the maximum record size on tape. The message buffer has records of fixed size of 10240 bytes long each. Data from some input source (or sources) is put into the data buffer and written out from this buffer by an output process. Some flavors of the input process will pack data they read in units of event or non-event data into tape record size blocks.

A setup file is used, which is read at job initialization time, to configure the data server to perform desired functions. Each child process to be created is specified in this file. For example, one can specify that an input child process is to be created and which binary file should be used for that process. A number of other parameters, which are used to tailer the exact functions of the data server, can also be specified to override default values. In fact, the setup file is used to group appropriate flavors of child processes into a job to configure either a local, remote, or a special purpose data server. This is possible because each data server starts with the same main process which creates a well defined set of interprocess communication structures. It is the set of child processes which determines the exact functions that the data server performs.

For E154 two data servers were configured. One executes in a VME Unix processor and is called the local data server. The other executes on an IBM RISC6000 computer which is part of the SLAC data staging system and is called the remote data server. The local data server packs data into tape size blocks (typically 32K bytes long) and sends these blocks via network (FDDI) to the remote data server which writes the data to a staging disk pool.

2.4.2 Local Data Server

One of the VME Unix processors is used to run a local data server job. The local data server can be configured from a setup file to either send data for logging to the remote data server or to log data locally to an Exabyte high density tape drive. In either case it serves a fraction of the available data to analysis jobs for online data analysis. This server is specified in the setup file to consist of the following child processes:

- input - reads event data from (reflective) memory and receives non-event data from network.

- output - sends data packed into tape record size blocks to the remote data server via network.

- analysis server.

- diagnostic display.

2.4.3 Remote Data Server

This data server interacts with the SLAC staging system. It's input process is a network server which accepts two network connections from the output process of the local data server. The data is passed from the local data server to the input process of the remote data server in units of tape record size blocks with full handshaking on one of the two network connections. As explained above this data connection could be blocked by the remote data server if its data buffers become full. The other connection is used to pass "priority" commands and should never get blocked. Some of these commands initiate interaction with the staging system. All commands which originate in the DAQCntrl get acknowledged by the output process sending a "reply" message to DAQCntrl via its connection with netServ. This server is configured from the setup file to consist of the following child processes:

- input - gets data from the network device (a network server).

- output - writes data to a disk file, replies to messages via netServ as needed.

2.4.4 Monitoring Data Servers

The data servers can be monitored in a few ways. An interactive X11 graphic program, scntrl described in section 2.8, allows viewing the status of both the local and remote data servers. Also a diagnostic display child process can be run as part of either data server. It shows the status of all processes in the data server job, and allows displaying the status of various buffers which are part of the data server. Finally, DAQCntrl by periodically sending requests for status information effectively monitors both data servers, although the information it obtains lacks details.

2.5 Experiment Monitoring and Control

Several processes are used to monitor the experimental hardware as well as the performance of the DAQ and to control the state of the DAQ. All of these processes execute on VAX/VMS computers. Each process creates a link with the designated netServ. DAQCntrl is the interactive experiment control process and all remaining monitoring processes exchange messages with DAQCntrl via virtual links. All monitoring processes report error messages to DAQCntrl and DAQCntrl sounds an alarm and displays the error message. The following list describes these processes:

DAQCntrl: an interactive X11 graphic control program allows an operator to change the state of the DAQ. From this program one can start and end a run, mount and dismount a tape, change various setup parameters, etc. It maintains a log file in which it records all control and monitoring activities. It also maintains a record in a disk file of which runs are written to which tapes. It has a number of display windows which are used to display various data from the shared memory segments.

DAQJob: a non-interactive process that periodically reads monitoring data from a CAMAC branch such as High Voltage settings, scaler values, target parameters, etc. It stores these values in a number of shared memory segments (global sections) making the data available to DAQCntrl and other processes executing on the same processor. It performs some data correctness tests.

DAQMagb: a non-interactive program which controls large power supplies used to energize the spectrometer and other magnets used in ESA. It controls the power supplies by performing IO operations on Power Supply Control (PSC) modules in a serial CAMAC branch. This program accepts commands from DAQMagi program, see below.

MCC: runs on a VAX computer in the Main Control Center (MCC) area and snoops the Main Control data base for data of interest to the experiment. It sends the data periodically to DAQCntrl via netServ.

DAQHV: an interactive program used to control the High Voltage settings in the LeCroy 4032 HV system.

DAQSclr: an interactive program used to view scaler data. The data is read by DAQJob and is available to the DAQSclr program via a shared memory segment. This program can also be used to modify the scaler setup data base.

DAQLV: an interactive program used to view low voltage data, which includes both CAMAC and NIM crate voltages and assorted values read in with smart analog monitor (SAM) CAMAC modules.

DAQMagi: an interactive program which is used to control DAQMagb program. It displays the current status of all defined magnet systems. It also allows changing the state of any defined magnet system. A magnet system is a logical grouping of a magnet, a power supply, and a PSC module.

2.6 Online Data Analysis

The architecture of the data analysis job is very similar to that of a data server. It consists of a parent process (main loop) which creates all interprocess communication structures and a number of child processes. The parent process maintains a log file, waits for any of its child processes to terminate, and analyzes the data which it finds in the data buffers. The job is configured from a setup file in which one can specify type and binary file path for each child process to be created. One also specifies data source type and the data source device name and other parameters. Similarly, one can specify data sink type and device name. The simplest configuration of an online analysis job would consist of the parent process, network client input process, a display process, and a Physics Analysis Workstation (PAW) process. Note that the display process allows viewing various "text" displays which typically are tables of numerical values, while, PAW is used to view histograms, etc. Other configurations are possible and are used in offline data analysis. A block diagram of a typical online analysis job is shown in Figure 9.

2.7 Portable VMS Front End

A previously developed VMS-Qbus based data acquisition was redesigned and integrated with the ESA DAQ described in this note. This VMS based DAQ is a two process job which functionally replaces the Remote VME Real-Time and Local VME Real-

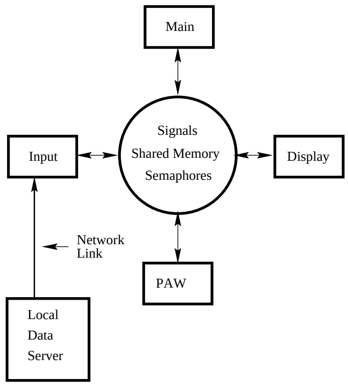

Figure 9: Block Diagram of an Online Analysis Job

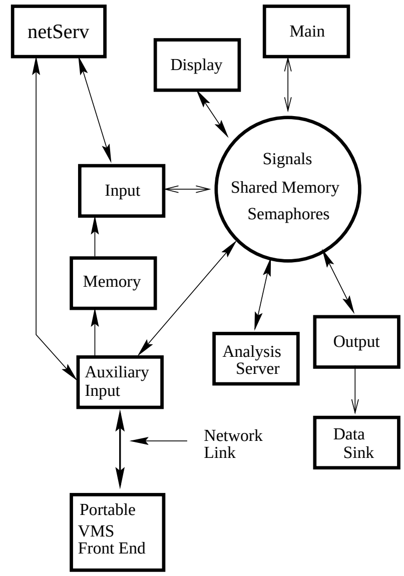

Figure 10: Local Data Server and the Portable VME Front End

Time boxes in Figure 1. It will be referred to here as the VMS Front End. The first of the two processes responds to an external trigger, reads data from the CAMAC subsystem, packages the event data, and makes the data available to the second process via a global section. The second process is a network server which accepts a connection via a TCP socket. When such a connection is established with a client, the client can request data which is sent to it via the net.

The local data server is run in a nonstandard configuration when used with the VMS Front END. This is shown in a block diagram in Figure 10. This nonstandard configuration differs from the standard configuration as follows:

- the local data server instead of using the reflective memory for its data source, uses a memory region in the processor's memory.

- a new child process is added to the local data server job. This process is a network client and is labeled in the figure as "Auxiliary Input". It connects with the VMS Front End server process and obtains event data from it. This client process also receives commands from the monitor/control subsystem via netServ which it passes to the VMS Front End.

The VMS Front End job can run on any VMS computer with a Qbus. We have four Qbus VAX/VMS computers in Counting House A. One of these is VS3200, to which we have interfaced a CAMAC subsystem via a Jorway 411 interface. This computer and the CAMAC setup can easily be moved to locations difficult to reach with the VME subsystem. The only requirements for the VMS Front End are for sufficient AC power and an Ethernet connection. We have measured that the DAQ in this configuration can process event data $2000 bytes$ in size at a rate of $30 Hz$.

The VMS Front End job was developed to provide DAQ services for test experiments to be done either in ESA or other beam lines. It will be a backup system for the test of the

GLAST [6] detector, scheduled for early summer 1996.

The VMS Front End can be configured from a setup data file to be a network CAMAC server. In this configuration a client program could execute on some remote computer, connect with this specially configured CAMAC server, and perform CAMAC IO operations. This makes CAMAC subsystems interfaced to a VAX/VMS computer available to a variety of other computing platforms.

2.8 DAQ Monitoring and Control

A number of software tools have been developed which allow monitoring the DAQ performance as well as controlling it. These are diagnostic tools and are not part of the DAQ proper. The following list includes only those programs which are reasonably general in scope. That is, highly specialized test programs are excluded from this list.

smonx: an interactive X11 graphics network server monitor program. It connects to a specified network server, obtains data from the server, and presents the data for viewing. The information obtained shows a list of clients, the load each client places on the server, etc.

ofecntrl: an interactive X11 graphics program which can be used to monitor and control the real-time part of the DAQ, either the VME version or the portable VMS one. This program can issue all commands that DAQCntrl can, request and display contents of various data buffers, or request a single IO to be performed on a CAMAC module in which case it displays the results.

scntrl : an interactive X11 graphics program to monitor both the local and remote data servers. It connects with the designated netServ server and establishes a virtual link with the local data server. It periodically gathers data from both servers to show the status of each process in each server job and shows data flow rates in various parts of each data server. It can also show the status of the output files where the data is being written to. It has limited control capability. From it one can turn logging to tape on or off, put the servers in beam monitor or regular running state, etc.

References

[1] "A Proposal for a Precision Measurement of the Neutron Spin Structure Functions Using a Polarized HE-3 Target", SLAC-PROPOSAL-E-154, Oct. 1993, 36pp.

[2] "Measurement of Nucleon Spin Structure at SLAC in End Station A", SLAC-PROPOSAL-E-155, Jan. 1994, 56pp.

[3] "E154/5 Nucleon Spin Structure Functions CEP Review", May 16, 1994. Review is available from Experimental Facilities Department, Stanford Linear Accelerator Center. The VME Data Acquisition upgrade was originally proposed in March 1994, and later combined with the E154/5 CEP.

[4] Darren Cathey, "All Things Considered...", Second Quarter Edition of Real-Time Magazine, 1993. This article reports on a study conducted by the Superconducting Super Collider Data Acquisition group, comparing Real-Time operating system performance and features.

[5] "A Proposal for an Experiment to Study the Intereference Between Multiple Scattering and Bremsstrahlung (The LPM Effect)", SLAC-PROPOSAL-E-146, Jun 1992, 40pp.

[6] "Gamma Large Area Silocon Telescope (GLAST): Applying Silicon Strip Detector Technology to the Detection of Gamma Rays in Space", SLAC-PUB-6267, Jun 1993, also Nucl. Instrum. Meth. A342, 302-307, 1994. See also Michelson, P. F., *et. al.*, "Development of GLAST, A Broadband High Energy Gamma-Ray Telescope using Silicon Strip Detectors", NASA Advanced Missions Concept NRA 94-OSS-15.