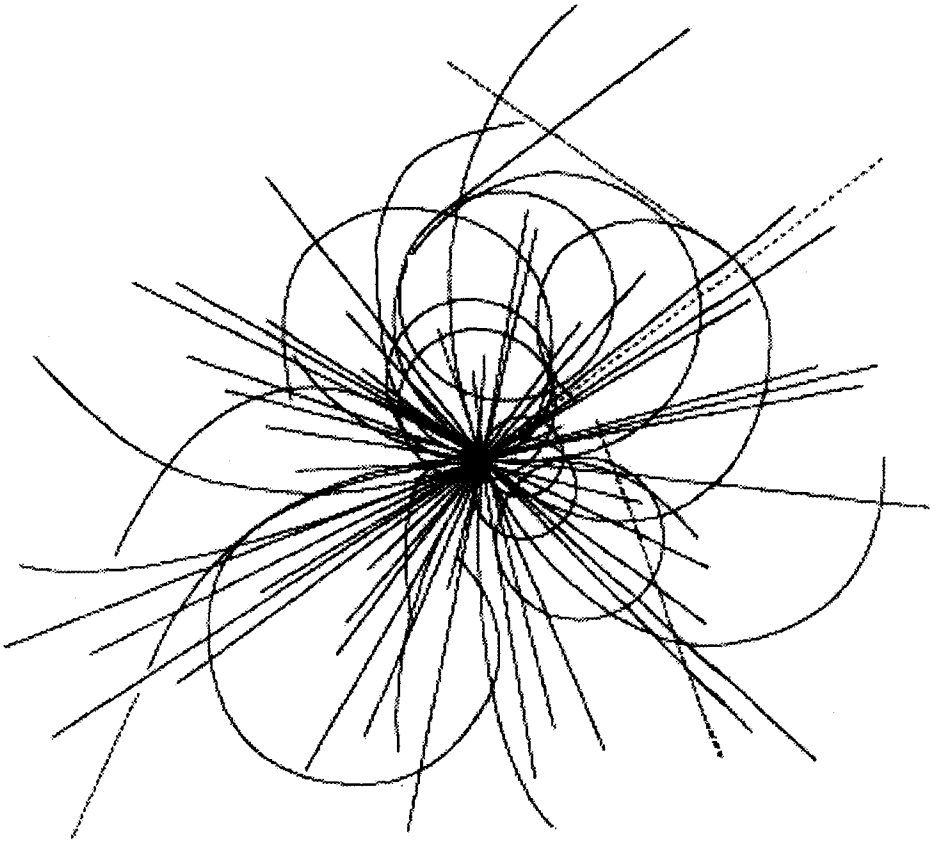
SSCL-Preprint-478

SSCL-Preprint-478
June 1993
Distribution Category: 400

M. Botlo
M. Jagielski
L. Miller
A. Romero

Data Acquisition for Detector Prototypes at the SSCL



Superconducting Super Collider Laboratory

SSCL-Preprint-478

Data Acquisition for Detector Protypes at the SSCL*

M. Botlo, M. Jagielski, L. Miller, and A. Romero

Superconducting Super Collider Laboratory†
2550 Beckleymeade Ave.
Dallas, TX 75237

June 1993

*Presented at the 8th Conference on Real-Time Computer Applications in Nuclear, Particle and Plasma Physics, June 8–11, 1993, Vancouver, Canada.
‡Operated by the Universities Research Association, Inc., for the U.S. Department of Energy under Contract No. DE-AC35-89ER40486.

Data Acquisition for Detector Prototypes at the SSCL

M. Botlo, M. Jagielski, L. Miller, A. Romero
Physics Research Division
Superconducting Super Collider Laboratory[1]
Dallas, TX 75237, USA

Abstract

We report on the design and implementation of a Data Acquisition System for experimental setups within the Physics Research Division of the Superconducting Super Collider. The design of the system is driven by the requirements of flexibility, modularity, portability, and safety. We emphasize off the shelf hard- and software such as Motorola 68040 [1] and RISC based processors, standard data links, and accepted packages such as X-windows and the real-time operating system VxWorks. We discuss both the quantitative and qualitative performance of our system and conclude with a brief review of current research and development in areas such as slow controls and Data-Machine Independence.

I. INTRODUCTION AND REQUIREMENTS

Some setups used to test prototypes for new generations of High Energy Physics detectors have reached a size and complexity comparable to present day experiments. Channel count, data volume, and a wide variety of front-end and data collection electronics set a high standard for the Data Acquisition (DAQ) with which the prototypes are read out. Furthermore, test beam time is becoming an increasingly scarce resource, thus demanding highly efficient use of available time. Therefore, the Data Acquisition System employed has to fulfill several requirements :

Performance: It should be possible to read up to order 10000 channels of non-zero suppressed front-end electronics with a frequency ranging from 100 Hz to several kHz, resulting in an average sustained throughput requirement from the front-end modules to the data logger of up to 2 MByte/s. The total bandwidth in the data collection stage might be an order of magnitude higher. Thus enough on-line processing power for data filtering should be available on demand.

Modularity: The strategic components of the system should work together in a plug and play fashion. This guarantees better configurability and adaptation to particular experimental requirements and allows to develop and commission components of the system in isolation. For example, software to drive front-end modules can be created without having to know about the rest of the acquisition chain.

Partitioning: It is desirable that the data acquisition has the ability to read concurrently several subsystems in isolation. For example, a subdetector can be calibrated while taking data with the other detector components.

Scalability: The DAQ should grow with the requirements of the experiment. For example, for small setups one read-out processor might suffice while bigger experiments require more bandwidth and on-line processing power. The transition from single to multi processing should be as transparent and easy for the DAQ user as possible.

Cost: The system cost of the acquisition system should scale linearly with the overall test setup requirements. This also means that the threshold cost necessary for reading out the first channel of detector should be kept as low as possible.

Standard: Where possible, the DAQ should utilize industry standards such as IEEE, ANSI and POSIX. Hardware should be produced and distributed by established manufacturers rather than in-house whenever possible. Some of the benefits of this approach are reduced cost, availability of components, and a widespread knowledge base throughout the HEP community and beyond.

Safety: Users of a particular experimental setup should not have to touch DAQ software. This requires that the read-out system follows a layered architecture, where most of the embedded code and the data transport layers are well confined. If a component of the read-out fails, the read-out should, depending on the error condition, either stop gracefully or should have enough built-in fire walls to allow data taking to continue, albeit with some reduced performance.

Control and Monitoring: The DAQ should be controlled by a centralized and uniform environment that interacts with the user in an intuitive, e.g. graphical, way. Run configuration, start/stop/pause/resume interaction, event dumps and event displays, alarm conditions, and system performance has to execute from within this central framework. Enough handles to user callable functions for on-line analysis should exist. Monitoring should be available simultaneously to several users with the proper privileges. The monitoring environment should be simply the run control with the control functions disabled. Monitoring of the experiment should be available not only on-site but also from remote locations and there should be no degradation in the real-time performance of the DAQ when monitors are active. Furthermore, an accurate history, recording active

[1] Operated by the Universities Research Association, Inc., for the U. S. Department of Energy under Contract No. DE-AC35-89ER40486.

user interactions with the acquisition system and alarm conditions together with time stamps, should be kept.

Slow Controls: The Data Acquisition has to provide reliable mechanisms for setting and monitoring environmental data such as high voltages, temperatures and pressures. Interactions with the trigger system should also be supported.

Flexibility: Tests for prototyping detectors require frequent changes in configuration. The DAQ has to have features supporting such changes in run configuration by saving and restoring previous or predefined setups. During the initialization phase for a new run, the connectivity and consistency of the run parameters should be checked and errors have to be reported and logged in a comprehensible way.

Portability: We expect to use this acquisition system in several different geographical locations. Thus the DAQ should be small in size and component count so that it can be easily moved, installed, and maintained at remote sites.

We have built a Data Acquisition System for Physics Research which strives to implement the above requirements. In the remaining chapters of this article, we describe the architecture of the system, give a listing of the hardware and software components used today, report on the performance, and conclude by mentioning future enhancements to the system.

II. SYSTEM ARCHITECTURE

The layout of the acquisition system is given in Fig. 1. Its centerpiece is the DAQ-crate in standard 6U VME. This crate holds one or more Front-End Processors (FEP) that communicates with the Front-End Crates (FEC), an Event Builder (EB), a unit for I/O of control and trigger signals (IO), and a State Machine (StM), which steers the read-out from the ensemble of read-out channels to a more complex data structure containing data and data description, i.e. a raw-data event. The Front-end

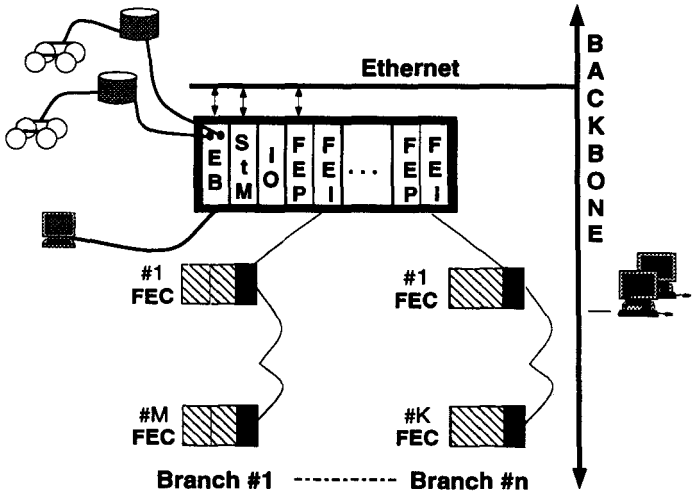


Fig. 1. prDAQ System Layout. EB denotes Event Builder; StM, State Machine; FEP, Front End Processor; IO, I/O module; FEI, Front End Interface; FEC, Front End Crate.

Processor interacts with the Front-end Crates via Front-End Interfaces (FEI) . A group of Front-end Crates that is interfaced to one Front-end Interface forms a read-out Branch and several such branches can be accessed in parallel. Usually each branch is composed of crates of one bus type, such as CAMAC, VME or VXI. All processors (FEP, EB, and StM) are remotely reachable via TCP/IP. The control and monitoring interface , in short User Interface (UI), resides in the Event Builder and can be brought up on any terminal that has X-windows capabilities. Other monitors can execute the same User Interface with its control options disabled. Low level debugging and monitoring can be done directly on the target processor.

The prDAQ provides mechanisms for distributed applications to request services, enabling remote clients to access live data.

The Event Builder has one or more SCSI controllers supporting disk and tape arrays. The formatted events are stored first on disk and later, once the data file has reached a certain size, streamed to 8 mm tape.

At several stages of the read-out there are provisions for callouts to user routines. For instance, users can synchronize their analysis package with the start/stop sequence. In most cases, data are on disk before they are accessed by the user by means of a small library of functions that walk behind the write pointer of the read-out task, although there are provisions for monitoring data fragments as they fly by.

For high bandwidth applications that employ several Front-End Processors, arbitration and self-synchronization can become a bottleneck. The time spent in resolving deadlock situations can become significant and as a result, deadtime increases fast. In such situations a State Machine can be utilized. It can receive control signals from Front-end Processors, the Event Builder, Front-end Interfaces, the User Interface, and the trigger system. It determines whether all required signals for a specific state of the read-out arrived in time, and sends control signals back to specified DAQ components. This cycle of receiving inputs, determining the next state and sending outputs is called a state transition. In fact, every conceivable type of read-out can be described as a sequence of such state transitions, called a state table. If a time-out condition occurs, the system can react in a predetermined way - a time-out triggers just another state transition - and thus indefinite deadlocks can be avoided. In addition, The State Machine records the fraction of time the DAQ was in each state, the frequency with which each state was visited, and which input or time-out caused the transition. This gives an easy and reliable mechanism to debug the system and to improve its performance. It measures system deadtime and allows for easy reconfiguration of the read-out sequence in a multiprocessor environment.

III. SYSTEM COMPONENTS

This section describes the main components of prDAQ that are being used as of June 1993. Currently implemented processors, links and system software will be discussed. Where appropriate, we include performance numbers.

A. Processors

The two main processing elements in the DAQ-crate are the Front-end Processor (FEP) and the Event Builder (EB). The optional State Machine (StM) executes on a FEP architecture.

A.1 Front-end Processor (FEP)

The requirements for the FEP are low-cost, options for SCSI-II and Ethernet, and a VME interface which includes at least DMA BLK. The FEP must be able to execute a preemptable kernel. The MVME162 [2] series of single board computers (SBC) has proven to be cost effective. This VME board contains a MC68LC040 embedded controller running at 25 MHz delivering 27 MIPS. The MVME162-13 includes 512 kB

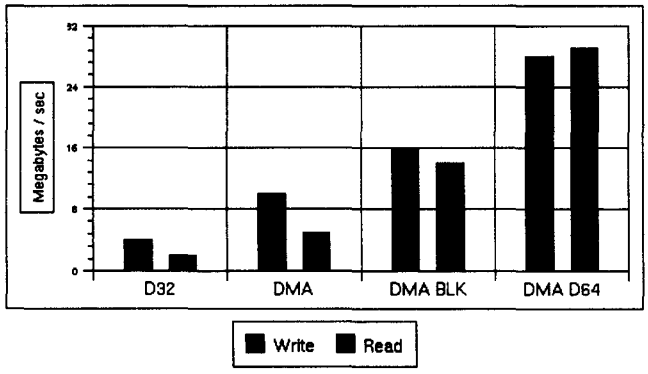


Fig. 2. MVME-162 VME read/write timings for D32 programmed I/O by the CPU and the DMA processor, DMA Block transfers, and DMA VME-D64. The slave system is another MVME 162 .

SRAM, 4 MB DRAM, 1 MB Flash, SCSI-II and Ethernet interfaces, 2 serial I/O, global control status registers for interprocessor communications, and a VME interface with D32, D32/BLK and D64/BLK DMA. By supporting slave DMA transfers, Front-end modules in VME can quickly move data to the FEP and master DMA transfers enable the FEP to transparently push data into the EB. Figure 2 shows various VME timings for the MVME162. For a more complete discussion on VME performance see reference [3].

A.2. Event Builder (EB)

The EB has to run a UNIX kernel that supports standard devices and a windowing environment. There should exist an efficient mechanism to transfer data across the VME backplane. SCSI-II should be fully implemented. Two different SPARC processor based VME boards are currently used in prDAQ.

The first is the Themis Computer SPARC 2LC [4] which is based on the Fujitsu SPARCset. The SPARC 2LC achieves 29 MIPS at 40 MHz. The device includes a DMA Controller which delivers 5.7 MByte/s for reads (7.5 MByte/s for writes) from (to) external memory but unfortunately does not yet support system software for VME slave access. The SCSI interface is only SCSI-I compliant and the SunOS 4.1.2 is a special version of Suns operating system and shows some incompatibilities.

As of today the EB of choice is the SPARC-2CE [5] produced by Force Computers. This SBC contains a 40 MHz SPARC 32-bit RISC processor chip with 16 (soon 32) MBytes of DRAM. The board features two Sbus expansion slots and a VMEbus interface that includes master and slave capabilities and VMEbus interrupt handling. One serious drawback of the board is that it does not support DMA transfers, limiting it to programmed writes and reads to/from VME memory at 4.8 and 4.4 MByte/s, respectively. The board runs standard Solaris, the most widely supported UNIX implementation available today. There is a SCSI-II interface that does not perform at SCSI-II speeds. Data transfer rates to disk are in the range of 2 Mbyte/s. The total throughput from disk on one SCSI controller to an array of four 8mm tape drives (Exabyte 8500) is 1.4 MByte/s.

We expect that new SBC SPARC technology will soon provide better DMA engines, faster processors, and SCSI-II conformance.

B. Links

The following gives a short description of the interconnects currently supported by this DAQ system. Adding new links has little impact on overall implementation.

It should be noted that the Front-end modules can reside in the DAQ-crate. In this configuration no special link is needed.

The Jorway Model 73A SCSI bus CAMAC controller [6] interfaces a CAMAC crate to any host computer supporting the SCSI-II specification, ANSI standard X3.131. The device achieves the maximum rate permitted by the CAMAC specifications of 1 Mtransfers/s. The overhead to execute one CAMAC command, due to SCSI protocol, is about 240 μs [7]. Since the controller uses the SCSI bus, it provides a standard interface to existing device drivers. All 24-bit transfers are automatically longword aligned with the benefit of reducing software overhead. Generally, other controllers need an extra VME module to interface to the crate while the Jorway-73A can be connected to a standard SCSI port of the host computer. This reduces cost and extra VME activity is eliminated. Single-ended SCSI is employed, allowing a maximum bus length of six meters.

For access to instrumentation, the National Instruments GPIB-1014 [8] is supported. This VME module interfaces the VMEbus to at most 15 GPIB based devices with a total cable length of up to 20 meters.

As more and more Front-end modules exist in the VME/VXI bus standards, links between these crates and the DAQ are needed. The short distance solution opted for thus far

has been the National Instruments MXIbus [9]. For long distance applications we use the Systran Corporation Scramnet [10].

The MXIbus from National Instruments provides a common bus to network multiple VME/VXI chassis. It extends VME, including interrupt propagation in a manner that is transparent to the application. Each device (VME, SUN, PC, Macintosh or Instrument) is a frame on the MXIbus network. Each frame has its own local bus such as NuBus, SBus, VMEbus and a common bus that links the frames together. The MXI boards have dual circuity that interacts with both the local and common bus. In our setup we use one and four meter cables. Data throughput (D32) from a MVME162 in one crate to memory in a second crate is 3.03 MByte/s (read) and 3.23 MByte/s (write) respectively [11]. In situations where FEC's are in VME and data filtering at the branch level reduces the event frequency significantly, the FEP can sit in the FEC with the advantage of reduced throughput requirements at the Event Builder. MXI technology makes this change in configuration transparent.

The Shared Common RAM Network (SCRAMNet) from Systran Corporation provides a distributed networking scheme that is fast, reliable and with rigidly predictable timings [12]. SCRAMNet is based on a replicated, shared-memory networking concept. From 128 kByte to 2 MByte of reflective memory is shared amongst the various VME chassis. SCRAMNet adapter boards exist for most popular platforms. A dual link fiber optical cable supports distances of up to 300 meters and theoretically 150 Mbit/s transmission rates. A long link converter can be used for distances up to 4000 meters. The node to node data throughput varies with the number of nodes on the network and the cable length between nodes. For a three node network with 30 meter cables between nodes it is in the order of 2-3 MByte/s. In burst mode data throughput in the order of 6 MByte/s between two nodes are achieved.

C. *Software*

Our architecture features two sets of unique and clearly distinct software environments. On one hand the FEP needs to provide deterministic and predictable timing in all execution paths; the application must be able to control all aspects of task management, such as priority and scheduling. On the other hand, the EB must provide friendly interfaces for tasks like run control, data storage and analysis. For the FEP we currently use the VxWorks 5.1 [13] preemptable kernel running on MVME162 single board computers. The external interrupt latency is 6-8 µs. Dispatching a task from the interrupt service routine requires 4 µs [14]. For the EB we currently use SunOS 4.1.2 running on the SPARC 2 based Force 2CE and Themis SUN2LC cards. Both kernels provide the services one expects from modern technology, including networking (TCP/IP), interprocess communication schemes (pipes, shared memory, queue, semaphores), interrupt handling, and are compatible with the computing environment at the SSC Laboratory. By using these kernels and the clear division between real-time versus non real-time, we have been able to rapidly move from the conceptional design of projects to the delivery of operational systems.

D. *Stand-alone Development Environment*

It is impractical that every user has to acquire expertise in Unix, the C language, and the VxWorks operating system before developing new components such as device drivers for front-end modules. For these users we offer the LabView [15] environment from National Instruments. LabView 3.0 runs interchangeably on Macintosh, MS-Windows, and Solaris. A GPIB, SCSI, VME (for the EB boards) and MXI library interfaces the computer of choice with CAMAC, VME/VXI, and GPIB devices. After prototyping work in LabView is completed it is straight forward to translate the code to C and integrate it with the rest of the DAQ.

LabView provides the capability to develop a complete data acquisition environment at low rate without real-time requirements. This makes it useful for most slow control applications.

IV. IMPLEMENTATION

This section discusses the implementation of the prDAQ using the above mentioned components.

A. *The User Interface (UI)*

The prDAQ currently takes advantage of the OPENLOOK Graphical User Interface (GUI) application environment provided by SunOS. An additional non-graphical interface is available for dumb terminals. Figure 3 shows the three components that comprise the UI. It allows the user to configure, control and monitor the data acquisition run. Using UNIX interprocess communication mechanisms of message queues and shared memory, multiple user's (even from remote machines) may have a GUI hooked to the prDAQ but only one user is allowed to be operator of the run.

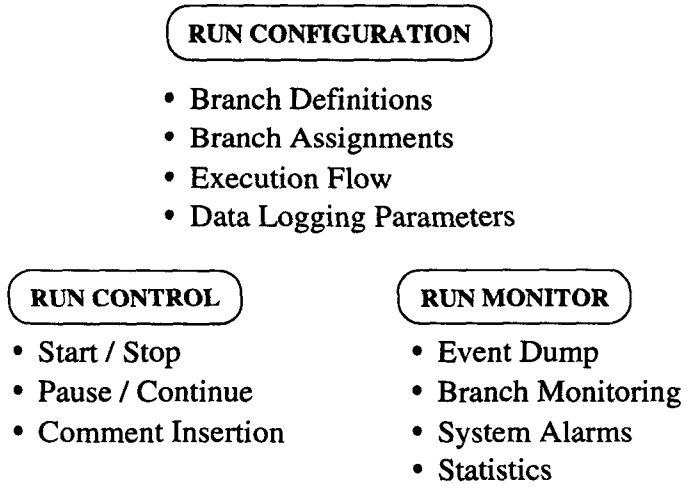


Fig. 3. User Interface/Graphical User Interface Components.

A.1 Run Configuration

The prDAQ configuration follows four steps: Description of the read-out branches, assignment of branches to FEP's, specifying the execution flow and setting the data logging parameters.

The CED provides a hierarchical description of the system that correlates with the data stream at run-time. The data stream consists of a series of events. Each event has a header followed by a set of group data and the end-of-event (EOE) word. The event header is 16 bytes containing event size, event number, DAQ mask, trigger mask, run number, date and time. The EOE word is is 4 bytes long and is used to verify data integrity. A group represents a set of one or more front end modules of the same type inside one FEC connected to the same subdetector type; it is made up of group header and data. The group header is 8 bytes long and includes group data size, mask, subsystem and crate number, module type, starting module and total number of modules in the group. The group data is variable length and 4 byte aligned. The CED output is a branch definition file that describes all the groups in the Front-end crates (FEC) linked to a given Front-end Interface (FEI). The CED provides a method of creating and editing the branch definition files. Crate and module types can be selected interactively. Figure 4 shows a window of the GUI CED displaying a CAMAC crate. The modules may be inserted or removed. The CED allows to configure which modules and

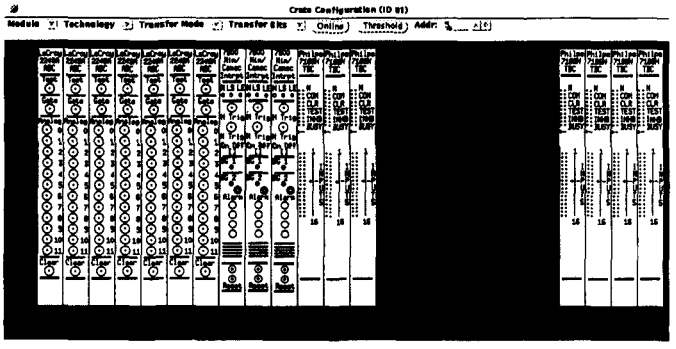


Fig. 4. Crate Editor Window for CAMAC.

crates are active during the run. The event format and a predicted read-out time may be previewed after defining a branch.

The GUI assignment panel provides a mechanism to link the branch definition files, FEP and trigger type. This assignment step associates one FEP with one or more read-out branches and the active trigger types for that branch. The read-out branch is described by the branch definition file, the FEP identification number is stored as part of the the boot parameters, and the trigger type is user defined.

The execution flow is specified through the state machine table. See section D below for more details.

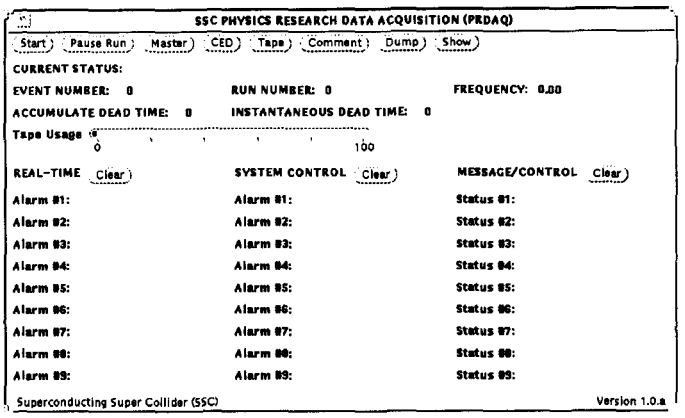


Fig. 5. prDAQ Control Panel Window.

The data logging parameters are configured by specifying whether data should be saved to disk and/or tape. Options allow to set file size, software zero suppression and whether the disk file should be removed after copying it to tape.

A.2 Run Control

From the prDAQ run control panel (Fig. 5), the user can start/stop and pause/continue a run. The start of a run initializes all Front-end Processors and Front-end Crates. Pausing a run simply disables triggers. Terminating a run disables all triggers and flushes all the buffers to disk and tape. At any time ASCII comments may be injected into the data stream thus tying the log book to the data.

A.3 Monitoring

The GUI enables the operator and other users to monitor system alarms, event frequency and dead time from the control panel (Fig. 5). The current event format can be displayed.The event dump mechanism can be invoked at the EB for fully assembled events or at the FEP for event fragments.

A.4 On-line Analysis

The prDAQ provides application entries for user packages. At the level of the FEP events can be rejected or flagged. The full event is available at the EB level. User packages can stay synchronized with the data acquisition through start-run/stop-run scripts, that are executed at the beginning and end of each run. Also a library of routines is provided to trail the current data file remotely over NFS.

B. Branch read-out

Each FEP is responsible for reading at least one branch of Front-end Crates. The branch task executing on each FEP polls on a local register waiting for a signal from the state machine. If it is a trigger signal, the trigger type is determined before jumping to the appropriate read routines. If the trigger type does not require access to FEC in the branch, the FEP generates

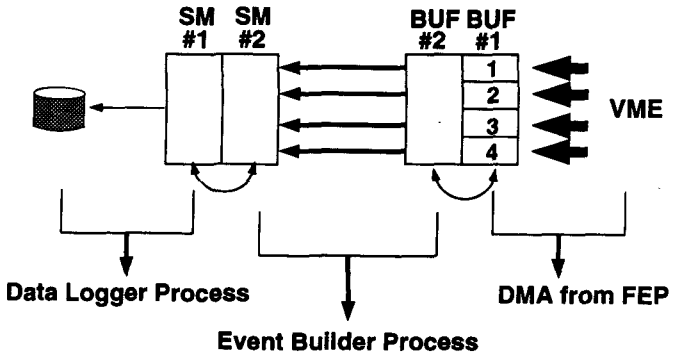


Fig. 6. This graph shows the data flow through the event builder and data logger process. SM = Shared Memory in the EB. 4 FEP's write into Buf(fer) number 1 while the EB reads Buf #2, builds the final event and transports it to SM #2.

a null event, i.e. a data structure containing only a header and EOE marker. A double buffer is employed; while the branch task fills one buffer with event data, the other buffer is pushed to the EB via VME transfers. The FEP acknowledges completion of requests from the state machine.

Currently we work with up to four FEP's in the DAQ-crate. The total throughput from all FEP's to an EB that has no DMA controller, the Force 2CE, is 9.6 MByte/s (6.76 MByte/s for one FEP). We expect the corresponding numbers for an EB with DMA and slave capability to be around 16 MByte/s and 10 MByte/s, respectively [3].

In smaller applications with a single FEP a state machine is not required. The execution flow is synchronized internally.

C. *Event Building and Data Logging*

Three tasks execute in the EB along with the UI/GUI components; the event builder, the data logger, and the tape logger. These three processes run synchronously using the UNIX interprocess communication mechanisms of shared memory and semaphores. Currently the event builder process works in a double buffer scheme. The EB has 1 MByte of VME slave memory. This is non-cached memory used by the FEP to deliver event fragments.This memory is divided into two buffers. The event builder process builds one buffer while the other buffer is being filled from VME (Fig. 6). It inspects the data fragments previously assembled by the FEP. In case more than one FEP is in the read-out, the EB process checks the event fragments for consistency, strips their headers and builds a new, final event.

The data logger process is also double buffered. While one buffer is being filled by merged event fragments, the other buffer is transferred to disk by the data logger process.

The tape logger process unblocks when the data file on disk reaches a configurable maximum size and dumps the file to tape.

D. *State Machine and Carrier*

For applications demanding bandwidth that can not be met with a single FEP, a state machine is used to synchronize trigger signals, multiple FEP's and the EB. The implementation of the state machine was inspired by the successful use of the concept in the UA1 experiment at CERN [16].

There are two types of transitions, input and time-out transitions. The state machine uses a 32 bit register to determine if an input transition is needed. Once a valid input pattern is found, a transition occurs that sets a 32 bit output register as specified by a user supplied program, called the state table. The state machine records the presence of invalid input signals, the frequency with which each transition is visited, and the time the system is in each state with a resolution of one μs.

As many programmable timers as needed are multiplexed out of the real clock. Each software timer has a time-out value and can be started,stopped or reset to zero during transitions. The system updates and checks the active software timers. If one active timer has exceeded its limit and the current state contains a time-out transition on that timer, a state change occurs.

To describe the sequence of action of the system, the programmer uses a state machine definition language to formulate

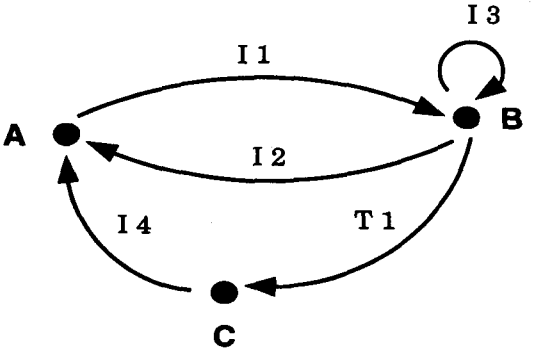


Fig. 7. A system with three states. The code fragments in Fig. 8 and Fig. 9 define and specify the execution of this system for state A and B.

a state table that contains two parts. The definition section defines all state names, input register bit names, output register bit names and software timers.

In Fig. 7, we give an example of a state diagram with three states. State A has one input transition, I1, pointing to state B. State B contains two input transitions, I2 and I3, and one time-out transition, T1.

Figure 8 shows a fragment of the definition section for this example.

The second part of a state program describes the execution

```
DSTATE(A)                    # define state A
DSTATE(B)                    # define state B
DINPUT(start;  0)            # define input bit 0 as 'start'
DINPUT(stop;   1)            # define input bit 1 as 'go'
DINPUT(more;   2)            # define input bit 2 as 'more'
DOUTPUT(flag,0)              # define output bit 0 as 'flag'
DTIMER(too_long; 1000)       # timeout at 1 millisecond
```

Fig. 8. An example of a fragment for the Definition section of a State Table.

flow. Figure 9 shows states A and B in the state machine language. Note that in Fig. 9 some parameters are left out for clarity and state C is not coded. In real life state "A" could be associated with START_RUN, and state "B" with "WAIT_FOR_ TRIGGER ". If B sees "more", it propagates a "flag" to the outside world, updates statistics and returns to

```
STATE(A)

   INPUT(start)              # if the 'start' bit is set,
   TSTART(too_long)          # start the timer
   NEXTSTATE(B)              #  transition to state B

STATE(B)

   INPUT(stop)               # if the 'stop' bit is set
   TRESET(too_long)          # stop and set the soft timer to zerc
   NEXTSTATE(A)              # go to state A

   INPUT(more)               # if the 'more' bit is set
   OUTPUT(flag)              #  input 'more' to somebody else
   NEXTSTATE(B)              # transition to myself

   TIMEOUT(too_long)         # if the 'too_long' timer reached its
   NEXTSTATE(C)              # limit; move to state C
```

Fig. 9. Code fragment for the execution flow of the state diagram in Fig. 7. The corresponding definition section can be found in Fig. 8. State C is not shown.

itself. Signal "stop" returns the system to an infinite wait on "start", and finally, if there was no trigger for 1 ms a transition to state "C" happens.

The state table language has provisions to execute custom code at specific places in the execution flow.

The state machine language is translated to "C" code that is compiled and loaded into the target, currently a MVME167 [17] running VxWorks 5.1.

The performance of the state machine is as follows: The latency between detection of a valid input pattern and sending of the specified outputs is 3 μs, followed by 10 μs before the next state is reached.

In the current implementation a software carrier using the VME protocol sets the input register and sends the output pattern. This adds between 25 μs and 80 μs per transition, resulting in substantial additional system deadtime. We are in the design phase of a hardware carrier that will almost eliminate this overhead. The I/O lines of the hardware carrier will be individually programmable and at least one line will be able to cause a reset or will interrupt the local bus of the processor.

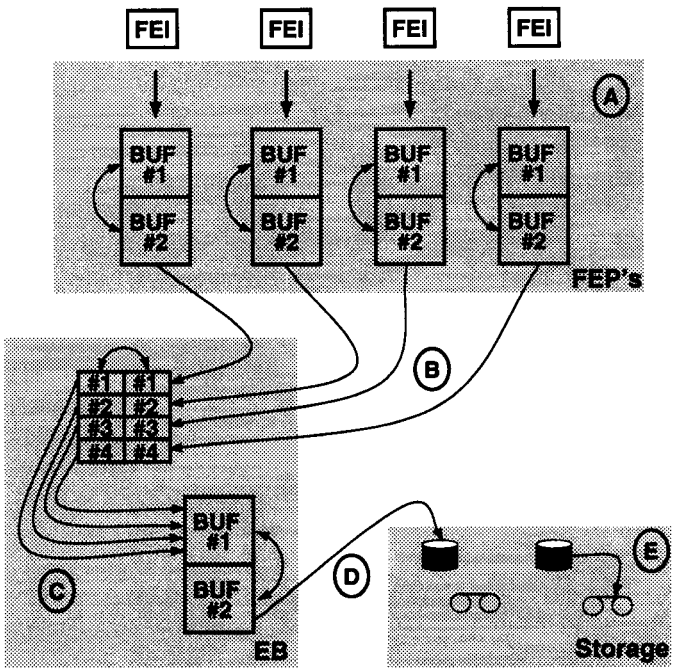


Fig. 10. The five stage (A to E) pipeline of prDAQ.

E. Data Flow

This section summarizes the data flow of the entire system, depicted in Fig. 10. Currently a double buffered scheme can be used throughout prDAQ, resulting in concurrent execution of the five data links (A to E in Fig. 10) in the system.

To double-buffer link E we make use of one of the Sbus expansion slots on the EB to install another SCSI-II controller. The EB now has two disks with at least two drives with half the tape drives and one disk being paired together on each SCSI-II port.

The EB also has a pair of double buffers. One exists in shared memory and the other in the non-cached (slave memory) region of the board. In Fig. 10 we can follow the five stages of the prDAQ data flow. In the first stage of the pipeline (A) data is read from FEC to FEP. Next (B) the event fragments are pushed from FEP to Slave Memory of the EB. Following (C) where the events fragments from slave memory are assembled into complete events in shared memory, events are written from Shared Memory to disk (D). In the last stage of the pipeline (E), event files are copied from disk to tape.

Whenever a stage transfer completes, a buffer, disk or tape swap occurs.

F. *Fault Tolerance/Error Detection*

The prDAQ notifies the UI about problems during initialization of FEC's and of bad events during the run. Alarm conditions are printed in one of three areas of the Control Panel of the GUI (real-time, system control, and message/control lists in Fig.5) and can be acknowledged (cleared) by the user. In addition, the UI is continuously updated about system dead time, event number at the EB, and average read-out rate. Processes running on the Event Builder (EB) check for any I/O bound errors, for example during tape and disk access, and find event format inconsistencies. Fault tolerance is provided through heavy use of multitasking. If a non-mission critical task crashes, i.e., a monitor or the UI, the current run continues. For a read-out with more than one FEI and trigger type a fatal error in a component of a particular read-out branch terminates only the read-out of the trigger tied to that branch and does not stop the run. This mechanism is used as a fire-wall to contain problems inside the smallest possible region of the system. The State Machine keeps track about vital system parameters. For example, the number of events rejected by FEP's or the EB can be determined by the number of times the CLEAR_EVENT state was visited. In a similar fashion, buffer bottlenecks can be detected by the number of visits and time spent at the BUFFER_FULL state.

The FEP provides a mechanism to examine the last event rejected. In the event of critical errors the system shuts down, attempts to flush all buffers and closes files. A log file is provided with run statistics and operator interactions.

The prDAQ system is self sufficient: If the connection to the outside computing world breaks (usually due to some network problem), the systems and application programs are loadable from eprom or local disk. A terminal connected to the EB can be used as operator console.

V. RESEARCH AND DEVELOPMENT

In several areas we have mounted a research and development effort with the goal to improve and augment the present DAQ system. What follows is a summary of some of the activities:

Inter-processor communication: It is evident that even the upcoming revisions of the VME specifications (VME64 and SSBLT) cannot satisfy future increased requirements on the sustained data throughput on the VME backplane. Therefore we explore the possibilities opened up by a variety of high speed front panel communication links (combined with massive compute power) such as the Hydra SBC from Ariel Corp. [18] based on the Texas Instruments TMS320C40 DSP, and the DS-links used by the T9000 Transputer from Inmos [19]. These links can off-load VME traffic to the Event Builder (and their CPU's could be the EB/Trigger) but would keep the DAQ system still in a single crate configuration.

Data Machine Independence [20]: There are still differences in internal data representations implemented in different computer architectures. We have evaluated several widely available packages such as XDR and ASN.1 for their usefulness in real-time applications and plan to devise a scheme that unifies data representations for user level applications.

Slow Controls: Together with the Accelerator Divisions at the SSCL we work on projects [21] that use the Experimental Physics and Industrial Control System [22], in short EPICS, developed originally at LAMPF and selected as the Accelerator Control System for the SSC complex. EPICS became a candidate for slow controls used by SSC detectors and we evaluate its functionality as the control component for prDAQ.

Other activities: We are in the process of recoding the graphics of prDAQ using portable GUI development systems such as XVT from XVT Software Inc. [23]. That should allow us to run the UI of prDAQ on a wide variety of graphics platforms. We experiment with an early version of Microsoft Windows-NT as a possible candidate for a replacement of UNIX.

VII. SUMMARY, CONCLUSIONS, OUTLOOK

We have developed a Data Acquisition system for SSC detector prototypes (prDAQ). The architecture emphasizes performance and scalability, modularity, and flexibility. To keep the cost and complexity of the system inside reasonable bounds and increase its reliability we place all components of prDAQ into a single VME crate and use the backplane for data exchange. That limits the bandwidth to the Event Builder to order ten MByte/s, which is sufficient for the read-out of foreseeable detector prototypes. However, we expect to overcome the bandwidth limitation by employing front panel communication channels for high speed transfers and more powerful processing elements for event building and software triggering. That should make prDAQ useful for high bandwidth physics experiments.

VII. ACKNOWLEDGMENTS

We would like to thank the TTR collaboration at the SSCL, especially L. Villasenor, for their advice and patience during the installation phase of prDAQ into their setup. J. Hayes expertise in low level LabView programming was essential. We thank R. Hubbard for expert help in SCSI matters and M. Lowry and D. Kouzes for the Jorway SCSI driver on the Sun. V. S. Kapoor's help in preparing this paper is appreciated.

VII. REFERENCES

[1] Motorola Inc., "MC68040 32-bit Microprocessor User's Manual", 1989.

[2] Motorola Inc., "MVME162 Embedded Controller Programmer's Reference Guide", January 1993.

[3] Botlo, M., Jagielski, M., Miller, L., Romero, A., "VME Data Throughput", SSCL-629, June 1993.

[4] Themis Computers, "SPARC 2LC User's Manual", Revision 1.1.

[5] Force Computers, Inc. , "SPARC CPU-2CE Manual", Release 1.0, August 18, 1992.

[6] Jorway, "User's Manual Model 73A, 73A-1 SCSI Bus CAMAC Crate Controller".

[7] Botlo, M., Jagielski, M., Romero, A., "Evaluation of Jorway SCSI CAMAC controller", SSCL-630, June 1993.

[8] National Instruments, "GPIB-1014 User Manual", May 1990.

[9] National Instruments, "VME-MXI User Manual", June 1991.

[10] Systran, "Manual for VME Hardware Interface Node", May 29, 1992.

[11] Botlo, M., Dunning, J., Jagielski, M., Miller, L., Romero, A., "MXIbus Data Throughput Tests", SSCL-603, November 1992.

[12] Technology Updates, "Fiber network supports distributed real-time systems", September 1990.

[13] Wind River Systems, "VxWorks Programmer's Guide", Release 5.1, February 1993. Wind River Systems, "VxWorks Reference Manual", Release 5.1 Beta, October 1992.

[14] Botlo, M., Jagielski, M., Miller, L., Romero, A., "VxWorks 5.1 Benchmark Tests", SSCL-627, June 1993.

[15] National Instruments, "Labview Function Reference Manual", January 1993.

[16] Los Alamos National Laboratory, "EPICS User Manual".

[17] Motorola Inc., "MVME166/MVME167/MVME187 Single Board Computers Programmer's Reference Guide", October 1992.

[18] Ariel Corp., "The Monsters of DSP!", Update #1, March 22, 1993.

[19] SGS-Thompson, "Inmos the Transputer Databook", Third Edition, 1992.

[20] Botlo, M., Jagielski, M., Miller, L., Romero, A., "Data Machine Independence", SSCL-628, June 1993.

[21] Bourianoff, G., et al "Interactive Simulation of LEB commissioning procedure on a Hypercube parallel computer", Proceedings of the IEEE Particle Accelerator Conference 1993, Washington D.C., USA (1993) , to be published.

[22] Botlo, M., et al "BATS, the readout control of UA1", NIM A302 (1991) 331-338.

[23] XVT Software Inc., "XVT Users Manual" (1993).