LA-UR -87-1825

LA-UR--87-1825

DE87 010140

TITLE: THE LOS ALAMOS NEUTRON SCATTERING CENTER DATA ACQUISITION SYSTEM

AUTHOR(S): R. O. Nelson, G. Cort, A. Gjovig, J. A. Goldstone, D. E. McMillan, J. Ross, J. Seal, LANSCE; and D. R. Machen, Scientific Systems International Inc.

SUBMITTED TO Proceedings of the Fifth Conference on Real-Time Computer Applications in Nuclear and Particle Physics, May 12-14, 1987, San Francisco, CA

MASTER

Los Alamos National Laboratory
Los Alamos, New Mexico 87545

[illegible]

DISTRIBUTION OF THIS DOCUMENT IS UNLIMITED

THE LOS ALAMOS NEUTRON SCATTERING CENTER DATA ACQUISITION SYSTEM

R.O. Nelson, G. Cort, A. Gjovig, J. A. Goldstone, D. E. McMillan,
J. Ross, J. Seal, LANSCE
D. R. Machen, Scientific Systems International

Contributed Paper
Fifth Conference on Real-Time Computer Applications in Nuclear and Particle
Physics, May 12-14, 1987, San Francisco , CA
Submitted May 20, 1987
LA-UR-

The FASTBUS subsystem of the LANSCE data acquisition system consists of a single FASTBUS crate segment with four custom modules and a QPI interface for the VAX. Since experiments at the LANSCE facility always include a time-of-flight parameter for the detected neutron and may optionally include additional position parameters characterizing the event, a time stamp is generated for each event by the Programmable Master Clock (PMC) module. The time and any position information are latched into the Time-Of-Flight buffer (TOF) module. After all events associated with a single neutron burst have been captured in a frame buffer internal to the TOF module, each event is analyzed by the MAPPER module and reduced to a histogram address to increment in the BULKSTORE module. Software access to the histogram is provided through the QPI interface.

Performance of the FASTBUS subsystem far exceeds the LANSCE system requirements. Data can be captured in the TOF module at average rates up to 10 MHz. The frame buffers presently implemented in the TOF modules allow two independent lists of up to 16000 events to be saved concurrently. Designed with a FIFO buffer ahead of the main frame buffers, bursts of data with up to 64 events are acquired at rates up to 20 MHz. A pair of latches in front of the FIFO allows event pairs to be received without data loss for time separation down to 25ns. Failure to store an event can be detected down to pulse pair separation times of 10ns. After capture in the TOF module, data is moved into the MAPPER module with block transfers, separated into time and position components, processed via look up tables, and recombined as a linear combination to form a histogram address. FASTBUS cycles consisting of only

address cycles are used to queue increment requests within the BULKSTORE modules. As currently implemented, the mapping and histogramming processes operate at sustained rates up to 2 MHz.

Software is provided to the experimenter for generalized data acquisition needs as well as to initialize the FASTBUS environment and read the accumulated histograms. This data acquisition command language is thoroughly integrated into the operating system supported command language and therefore can take advantage of all of the features of the rich VMS environment. Because the software is implemented in a highly modular fashion, suites of utility software modules are available to the general user community to support development of specialized monitoring or data analysis software.

The software system utilizes dynamic data structures extensively to support run time configuration of the FASTBUS as well as experimental control functions. This approach allows an experimenter to allocate data structures dynamically for memory control blocks, device descriptors, etc. without regard for artificial limits which exist in preallocated COMMON blocks. Consequently, the software is extremely robust and does not require maintenance revisits to adjust preallocated limits.

The software system also supports a subsystem which permits each user to customize the contents and format of the data files into which the experimental environment and contents of the histograms are saved at the conclusion of a run. This facility permits a user to select, at run time, specific features of the experimental environment (hardware setup and control functions) as well as specific histograms to be included in the data file.

The file is then written in a special format to support extremely fast access to the information stored therein.

Although the graphics software is immature at this time, use of the GKS international standard has substantially accelerated our implementation process. Limited use of special features of the workstation interface have been employed pending DEC's implementation of support for the new industry standard X-Windows under the VMS operating system.

THE LOS ALAMOS NEUTRON SCATTERING CENTER DATA ACQUISITION SYSTEM

R. O. Nelson, G. Cort, A. Gjovig, J. A. Goldstone, D. E. McMillan, J. Ross, J. Seal
Los Alamos National Laboratory
Los Alamos, NM 87545

and

D. R. Machen
Scientific Systems International, Inc.

Abstract

The next generation of data acquisition systems for the Los Alamos Neutron Scattering Center consists of a FASTBUS system controlled by a DEC VAXstation II/GPX. The FASTBUS subsystem features four custom high-performance modules interfaced to the VAX through the commercialized version of the QPI. Control and analysis is supported with the multiwindow graphics capability of the workstation.

Introduction

Roots for the development of the new data acquisition system for the Los Alamos Neutron Scattering Center (LANSCE) began to grow in 1981. Once funding was assured for the the Proton Storage Ring (PSR), it was clear that existing data acquisition facilities at the Weapons Neutron Research (WNR) facility would be overwhelmed with impossible count rates from the increased neutron fluxes. Designs for the PSR and modified target configuration indicated that neutron fluxes for LANSCE experiments would increase by a factor of approximately 250. Anticipating the design of a new system, the facility users defined and published the "Requirements of Data Acquisition and Analysis for Condensed Matter Studies at the WNR/PSR"[1]. The document addressed burst and average count rate issues, defined data compaction algorithms, and estimated storage requirements for both on-line histograms and disk storage. The basic functionality of support software was also described.

In response to the requirements document, the computer support group developed and published a preliminary design, "P-9 Proposed WNR Facility Data Acquisition System for the PSR Era"[2]. Central to the design were application specific FASTBUS modules hosted in a VAX/VMS environment. All real-time data collection was accomplished in the FASTBUS hardware while the VAX provided control and limited analysis capability. As the design moved toward implementation, the advent of new VAX products enabled the use of the miniaturized microVAX family. Continued technological changes have allowed evolution of the design from that first reported in the literature[3] to the final implementation presented here.

In the following section this paper summarizes the data acquisition requirements for the LANSCE system. The next section presents the basic design goals established by the implementers for the development project to assure a useful and flexible data acquisition system. The configuration section discusses the relationship between the FASTBUS hardware and the host VAX system, as well as the network environment which couples the various LANSCE instruments together. The last sections address the principal features of the FASTBUS hardware and associated data acquisition/analysis software.

Data Acquisition Requirements

While the hardware requirements are aligned closely to time-of-flight measurement techniques, the software requirements tend to be far more generic. These requirements are summarized below.

Neutron fluxes from the LANSCE target produce extremely high burst count rates. For instruments consisting of arrays of smaller detector elements, peak instrument burst rates may reach 45 MHz for durations of 10 μsec, with rates falling off from the peak finally reaching time averaged rates of 1 MHz. In other instruments consisting of a few indivisible detectors, *e.g.* a large surface area detector, burst count rates in a single element can approach 2 MHz. In all cases it is required that deadtime corrections be less than 0.5%.

As LANSCE uses time-of-flight techniques to measure the energy of the neutron, all events must be time stamped. Additional parallel information may be included with each event. It must be possible to reduce time and parallel information to form a compact descriptor address for the purpose of generating a histogram. A mapping capability may enable data compaction for the parallel information associated with surface area detectors. The descriptor algorithm must be flexible enough to include periodic frame dependence.

Histogram memory requirements vary from instrument to instrument but the largest requires at least 3 million channels for storage. To reduce the chance of histogram overflow during incrementing, all channels in the histogram memory must be at least 24 bits wide.

Software must provide the interface for run control. This includes run definition and run sequencing. A set of runs must execute without intervention and must process any environmental changes from a list of commands. Should the need arise, the experimenter must be able to interrupt and alter the predefined sequence. Control should be accomplished *via* simple keyboard commands.

The system must provide the user a fast, flexible and physically meaningful method for displaying spectra during acquisition and for runs already completed. Linear and logarithmic plots must be possible for each axis. For multiparameter data, it must be possible to display projections on a single axis or onto two-dimensional contours. Transformations between various physical units must be possible. Control should be accomplished via simple keyboard commands.

Data archiving must be automatic upon completion of the run. As soon as the instrument data file is created on the local disk, the file must be scheduled for transfer via the network to a central data archive system. The local copy of the instrument data file must remain on the local disk for a minimum of 48 hours.

Should the capacity of the local disk approach exhaustion, the user must be warned. The user must be able to catalog the last 5000 runs performed on the instrument.

Implementation Goals

For the implementation of the new LANSCE data acquisition systems, the computer staff places a high value on the development of an extremely reliable and easily maintainable system. To effect these goals, a great deal of attention is focused on the functional decomposition of the requirements and their assignments within modules to be designed both for the hardware and software. Details of the software development methodology are published elsewhere[4-6]. At this point it is sufficient to say that the software is fully engineered and formally tested prior to use by the experimenters. Similarly, the hardware modules are formally tested with a complete suite of custom software-driven diagnostics which test exhaustively all features of the modules. Both unit testing of individual modules as well as integrated system tests are performed.

In addition to reliability and maintainability, we stress development of an interface that is a logical extension of that with which our "typical" user is already familiar. Complete integration with the VMS operating system appears natural since all of our in-house users routinely analyze their data on VAX/VMS systems. External users generally have VAX/VMS experience from other laboratory environments. Not only is command invocation patterned after VMS, but error reporting and interactive "help" as well. For the graphics interface, we prefer to break our reliance on command driven dialog and adopt a more Macintosh-like appearance with emphasis on input *via* a mouse or tablet.

Based on experience with other data acquisition system implementations, we feel that the novice user is often permitted to do irreparable damage to his data inadvertently. From the start we include security features which inhibit illogical operations which could corrupt the data or the data acquisition environment. We recognize that under certain circumstances a user bypass must be provided, but this bypass may only be enabled by the manager of the effected neutron scattering instrument.

Configuration

As shown in Figure 1, the LANSCE data acquisition system configuration is highly distributed with a local area network (Ethernet) binding the various components together loosely. Each neutron scattering instrument has a separate autonomous data acquisition system with computer, FASTBUS, and other interface equipment. Centralized resources provide a data archive, access to line printers and color hard copy devices, magnetic tape drives, data analysis computers, and access to the laboratory-wide network. Distributed throughout the experimental and office areas are numerous terminal servers which allow convenient user access to the various data acquisition and analysis systems. Dial-up modems are interfaced directly to the network to enable equivalent flexibility for the remote user. Personal workstations coupled to the network ultimately will provide great computer power to each experimenter.

The data acquisition system dedicated to a neutron scattering instrument is fundamentally a DEC VAXstation/GPX with special peripheral equipment. This building block computer consists of the microVAX processor, an Ethernet interface, cartridge tape with interface, and the GPX graphics processor and monitor.

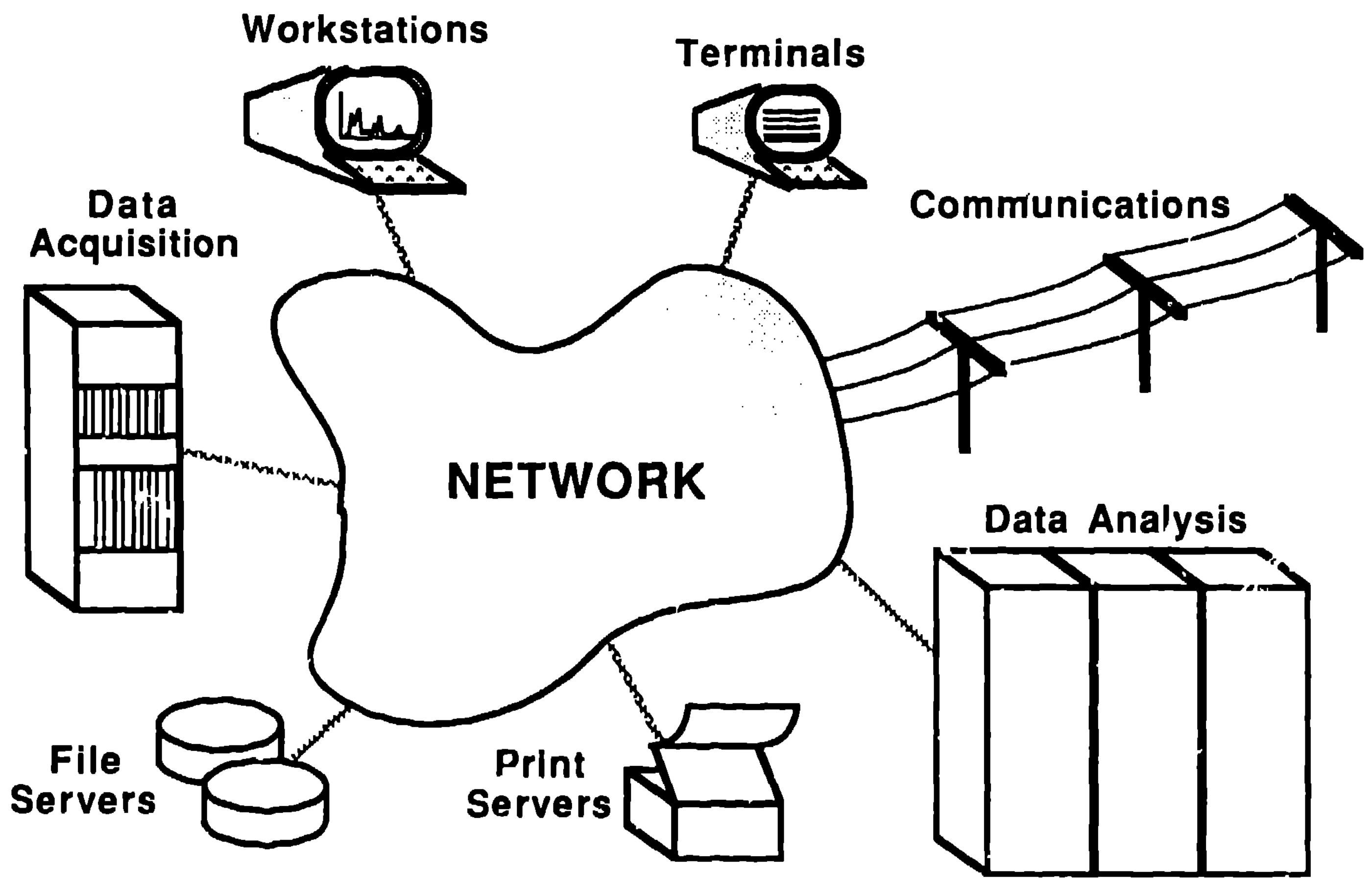

Figure 1. The LANSCE facility configuration.

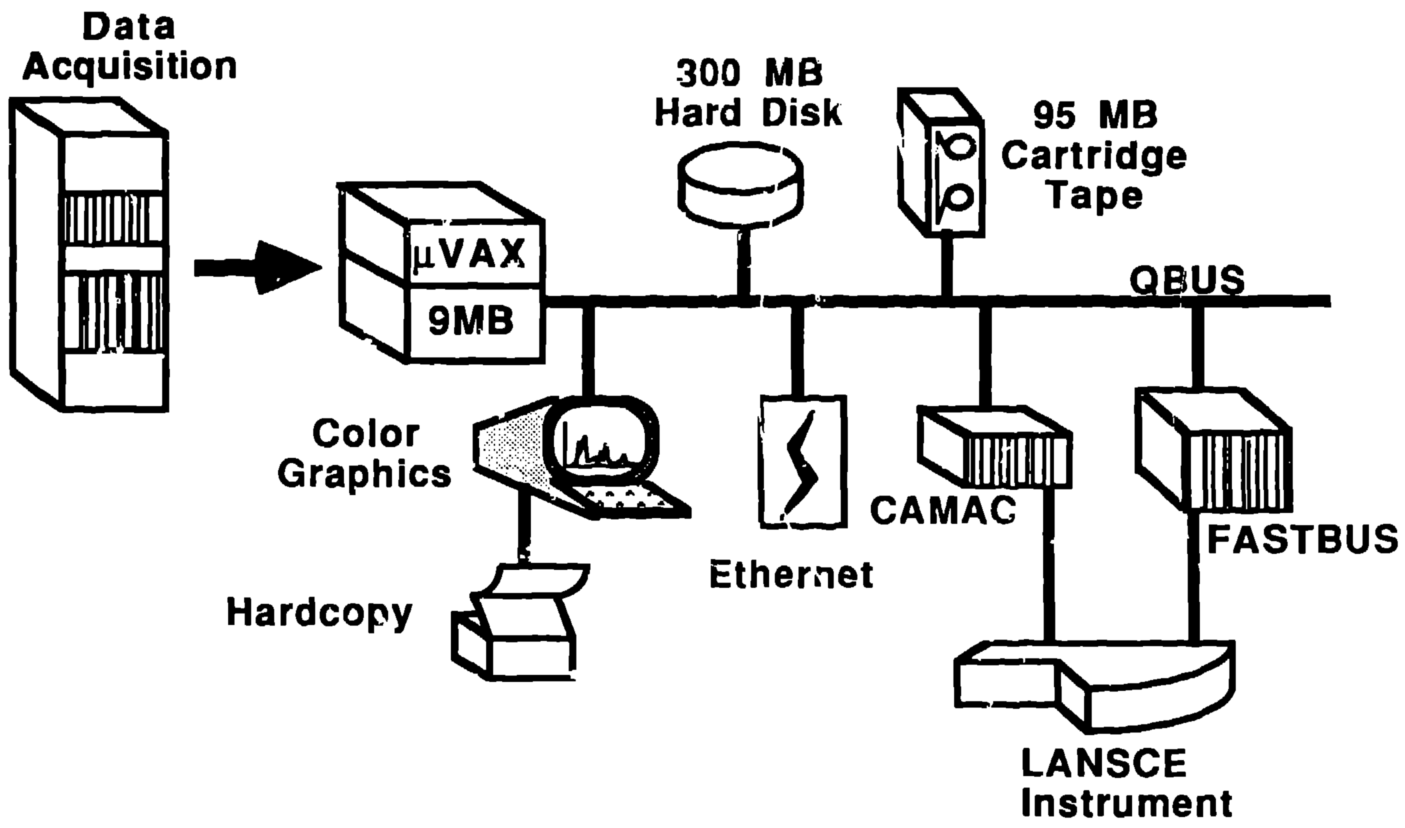

Figure 2. Data Acquisition configuration for typical LANCSE instrument.

The small hard disk included with the system is discarded in favor of a third party disk subsystem which offers 302 Mbytes of formatted capacity. The memory is expanded to 9 Mbytes. For instrument control and data collection, commercial interfaces are installed for CAMAC and FASTBUS, respectively. The FASTBUS interface is a commercialized version of the QPI developed at Fermilab. All the above equipment is installed in an eight slot backplane and mounted in a 5.25" high package.

The 5.25" high computer, FASTBUS, CAMAC and a general purpose NIM bin are all mounted together in an 80" high rack. Figure 2 illustrates the hardware components required for each instrument's data acquisition system.

The CAMAC instrumentation is used primarily for instrument and sample environment control. The most common applications include stepping motor and temperature control. During the transition period to FASTBUS data collection systems, the CAMAC crate remains a good location for simple scalers and preset scalers. Frequently, a CAMAC crate at the computer is connected *via* a serial highway to a remote crate located at the neutron scattering instrument.

The role of FASTBUS within the data acquisition system is described below in detail. Here it is sufficient to point out that all real-time data acquisition tasks are allocated to the realm of FASTBUS while the VAX provides the platform for our user friendly interface and complementary analysis computer power.

The multiwindow color workstation monitor is the primary interface for the experimenter. At his discretion, multiple windows may be opened for command entry or graphics presentation. Although somewhat constrained by memory resources, the experimenter may configure the number and size of windows to fit the requirements for his control, monitoring, and analysis needs. The impact of the workstation on our interface design is profound, as is discussed below.

While normally our use of the network is heavy, it is not an essential component should equipment malfunction or maintenance place it out of service. Under these conditions the experiment may continue but it is necessary for the experimenter to enter all commands at the computer console. Storage of data may be limited to the capacity of the local 302 Mbyte disk. Unlimited capacity is available if the experimenter utilizes the low performance but high capacity cartridge tape.

FASTBUS Hardware

The real-time data acquisition is accomplished with a set of four custom FASTBUS modules plus a commercial interface module for the microVAX. As in most neutron measurements, the flight time over a fixed length path characterizes the neutron energy and is the primary parameter of any measurement. Hence the first of our modules is a time interval generator. The second is simply a buffer serving to derandomize the data and to capture all events associated with a single burst of neutrons from the pulsed source. These two modules are both simple slaves. The third module has both slave and master functions. At the completion of a neutron burst, this module reads and processes the data captured in the buffer module and generates addresses to increment in a histogram storage module. The last custom module is the histogram storage module, and it is purely a slave. The interface module is the Kinetic Systems model F914, a commercialized version of the QPI developed at Fermilab. The flow of data between these modules is shown in Figure 3.

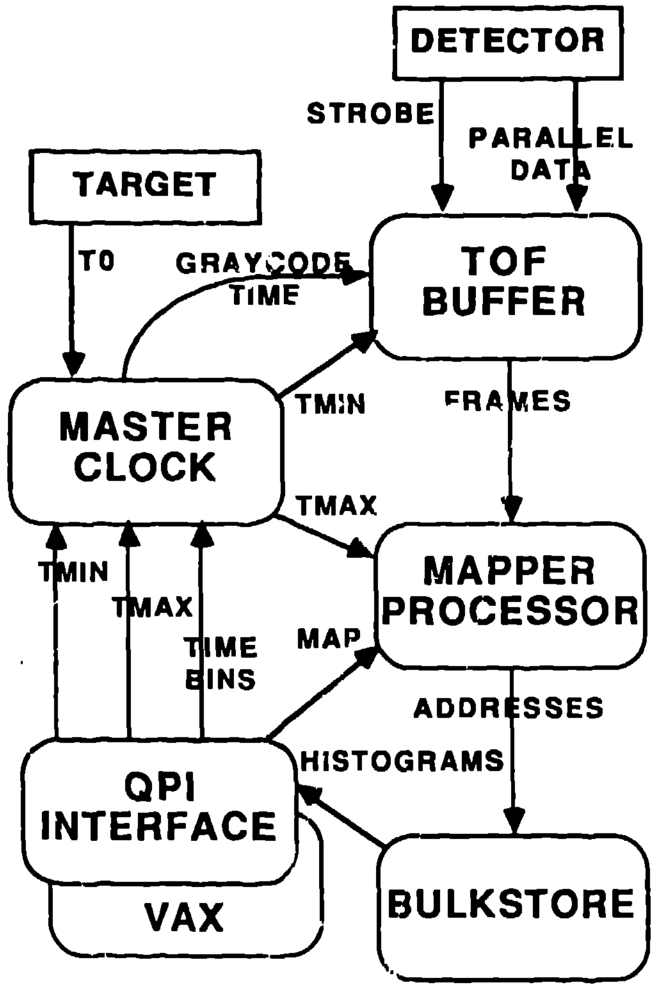

Figure 3. Data flow for FASTBUS hardware.

PMC

Expanding on this brief introduction, we begin a more detailed explanation of system operation starting with the Programmable Master Clock (PMC), the time interval generator. The PMC accepts an external start signal from the pulsed target system and begins incrementing a binary scaler. Uniform time channel widths of 50, 100, and 200 ns may be software selected. A local 1 Mbit memory within the the module allows neighboring uniform time channels to be combined to form logical time bins consisting of multiples of the basic clock frequency. As logical time bin boundaries are encountered, a graycode counter implemented with PALs is incremented. The graycode count is distributed to all TOF buffer modules *via* the auxiliary bus.

The specification of the logical time bins is accomplished through the use of predefined parameterized algorithms. Generally, these algorithms specify the logical time bin width as a linear function of the time interval. Under software control, the appropriate bits in the 1 Mbit memory are set to identify the bin boundaries. Similarly, software loads the PMC hardware register which identifies the time interval during which operation is enabled, TMIN to TMAX. Incrementing the graycode counter is inhibited until TMIN and after TMAX. Front panel outputs corresponding to logic levels for TMIN and TMAX are provided as control inputs for the TOF and MAPPER modules.

TOF

For time-of-flight measurements, system deadtime must be constant in order to apply deadtime corrections. If the data acquisition system is strobed at the arrival of an event, some time interval must elapse before another strobe can be accepted by the system. If this time interval is known and constant, then based on the counts observed in a channel, the effect at later times can be computed. However, if the deadtime is not known, then the shadow cast by any event is of unknown length in time and rigorous corrections are impossible. Thus careful monitoring of deadtime losses in time-of-flight measurements is critical. Normally, deadtime losses are monitored and should a failure to store as a result of deadtime be detected, then all of the data associated with the flawed burst are discarded.

The time-of-flight input buffer (TOF) correctly handles these deadtime issues. Failure to store an event can be detected down to pulse pair separation times of 10 ns. To buffer all events associated with a single burst of neutrons, two independent frames are provided which allow concurrent lists of 16K events in each. Designed with a FIFO buffer ahead of the main frame buffers, bursts of data with up to 64 events are acquired at rates up to 20 MHz. A pair of latches on front of the FIFO receives event pairs without data loss for pulse pair separation down to 25 ns.

Each TOF module supports inputs from four independent detectors. The data path for each input is parallel up to the point that data emerge from the FIFO. At this point data enter the shared frame buffer. Contention for storage is resolved with a round-robin algorithm so that no single channel can lockout other channels' requests for storage.

The TOF module combines time data from the PMC and parallel data from the detector system. The allocation of bits to either time or parallel data is flexible within limits. Generally, a minimum of 12 bits of time, 10 bits of parallel data, and 2 bits of encoded input number are included in the 32 bit event descriptor inserted in the frame buffer. The other 8 shared bits can be allocated to time and parallel data in any combination, *e.g.* 4 bits of time and 4 bits of parallel data or 6 bits of time and 2 bits of parallel data. The allocation is performed with separate jumpers for each of the four input channels. Both time and parallel data inputs are provided through the auxiliary connector.

Timing strobes for the four channels are cabled through front panel LEMO connectors. Each channel also has an external enable/disable LEMO connector which may be used to enable selectively active input channels. Other front panel inputs include the TMIN signal generated by the PMC module. The TOF accepts time strobes only during the period between TMIN and TMAX, *i.e.* the time during which the signal TMIN is at a high logic level.

The TOF includes a front panel connector for a signal provided to reset the frame counter. This counter increments at the completion of each frame so that should experiments require frame dependence for time resolved studies, the frame number in a series could be provided to the MAPPER processor. The reset input allows synchronization of the TOF counter with external periodic conditions applied in the sample environment.

MAPPER

At the completion of data collection for a single frame at time TMAX, the MAPPER processor module processes the frame that we designate the analysis frame. The PMC module provides a TMAX signal which initiates the MAPPER process. First the MAPPER interrogates the TOF buffer to determine if an unflawed frame is available, and if so how many events are present. The frame count, TOF module slot number, and spectrum length are

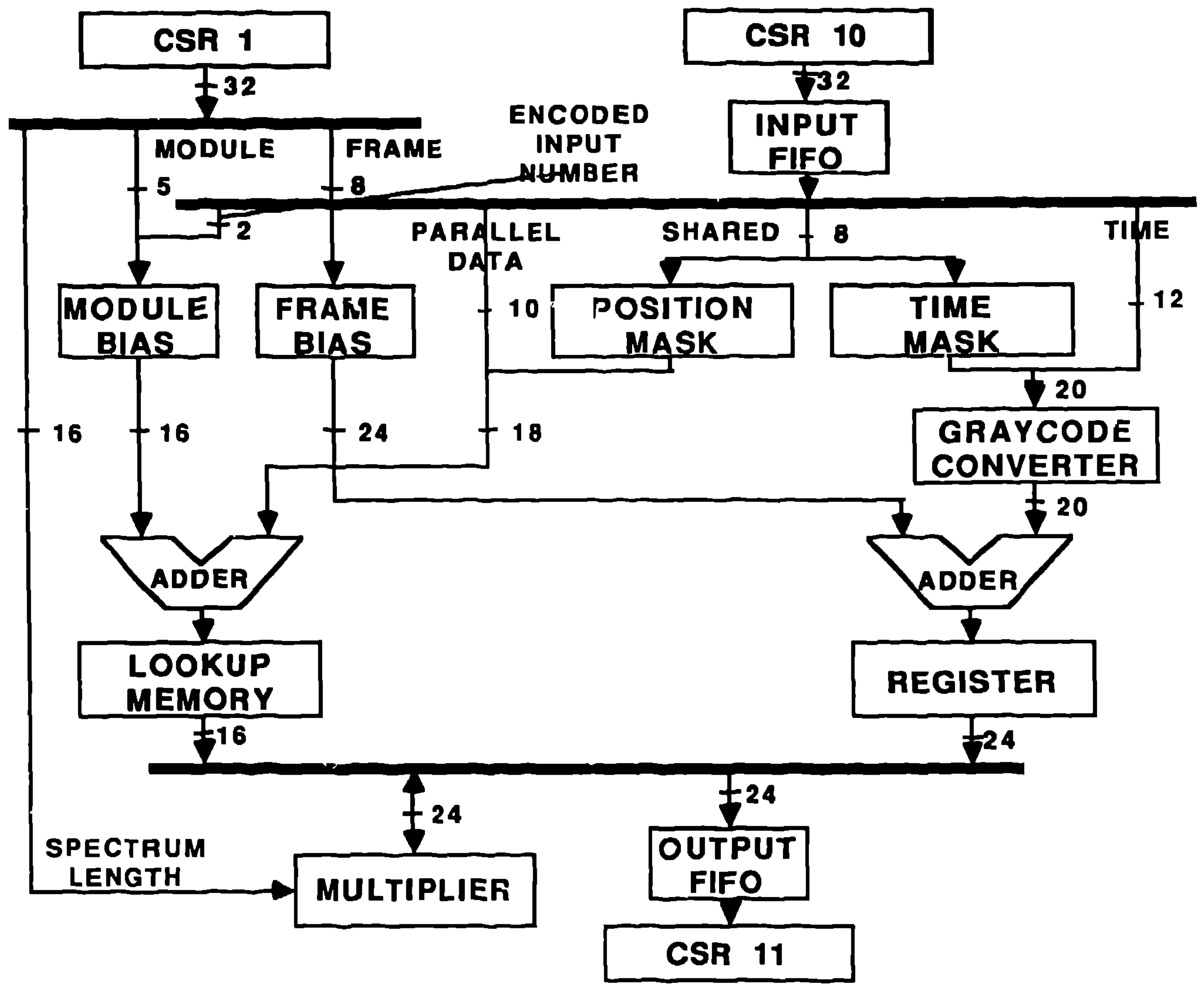

Figure 4. Block diagram for data paths in the MAPPER.

also read from a control register at this time. If valid data are available, then the master processor in the MAPPER moves event data into an internal input buffer. Concurrently, a second sequencer processor removes events from the input buffer, applies a data compaction algorithm, and places the reduced data in a second buffer, the output buffer. After filling the input buffer, the master processor takes data from the output buffer and uses these data as addresses of the channels to increment in the histogramming memory. As in filling the input buffer, the output buffer is processed as blocks of data. The master processor continues to alternate between filling the input buffer and emptying the output buffer until all the events in the frame have been processed.

While the MAPPER does its work, the TOF fills its other frame buffer, the acquisition frame. Should the MAPPER fail to finish processing the analysis frame at the time the TOF wishes to switch frames, then the TOF simply discards the data present in the current acquisition frame, and rewrites new data therein. A signal provided on a front panel LEMO connector on the TOF indicates that a frame has been lost.

At first glance the compaction algorithm for the MAPPER appears hopelessly complex. However, with the help of the block diagram in Figure 4, the process is far more apparent. As noted above, the frame count, TOF slot number and spectrum length are read from the TOF prior to transfer of data from the frame. This information is saved within the MAPPER in control and status register (CSR) 1 and used throughout the processing of the frame. Next the master processor begins filling the input buffer through CSR 10. Under the control of the sequencer, data flows through to the output buffer finally arriving at CSR 11. After emerging from the input buffer, an event is decomposed into time and parallel data. The two bits of encoded input channel number are separated immediately and appended to the other module slot bits to provide a input index number unique for each channel of each TOF. The shared bits are masked and specified bit fields concatenated with those fields simply passed through. The time bits are first converted from graycode to binary, and are then biased by an amount proportional to the frame count. This 24 bit sum then passes directly to the multiplier logic. Similarly, parallel data is biased by an amount determined by the unique input index. Unlike the biased time data, the biased parallel data is used as an address in a 64K deep lookup table which provides an indirect 16 bit result to the multiplier logic. Note that only the low 16 bits of the biased parallel data are effective in addressing the lookup table.

The multiplier logic operates in one of two modes. Either the mapped parallel data is multiplied by the spectrum length and added to the time, or the time is multiplied by the spectrum length and added to the mapped parallel data. The choice is dictated by the application. Generally, for instruments consisting of arrays of individual detector elements, it is desirable to operate in the latter mode so that one has time varying most rapidly, and thus has time-of-flight spectra for each detector element.

For surface area detectors, on the other hand, it is generally more desirable to reverse the order so that the parallel (position) data is most rapidly varying thereby to produce a surface histogram for each time slice. In this case the 24 bit time value is masked to only 16 bits since the multiplier can process only 16 bit operands. The mode for the multiplier is set with a bit in CSR 0.

As with the PMC modules, software algorithms are provided by the data acquisition system to load appropriate values in the table lookup memory and to establish the various biases for most conventional experiments. Currently a single algorithm is supported for the MAPPER which allows definition of an arbitrary number of fields within the parallel data; the user may specify a compression factor for each field. This factor need not be an integral power of 2. For more exotic applications, the user may define bit patterns as required by the experiment and the system software simply moves these values to the specified hardware memories.

BULKSTORE

Compacted data from the MAPPER form the addresses which are incremented within the BULKSTORE modules. The BULKSTORE is designed to treat a data space address connection that is not accompanied by any data cycles as an increment-address command. The address is captured in a FIFO buffer directly from the data space next transfer address (NTA) register. At speeds governed by memory cycle times, the specified internal address is accessed, incremented, and replaced in memory. Normal FASTBUS block transactions are also supported by the BULKSTORE to enable efficient histogram clearing, monitoring, and data read operations at run completion.

The current implementation of the BULKSTORE module offers 2 Mwords of histogramming memory in each module. The internal word length is 24 bits of which one bit is reserved for parity. No error correction is available. Currently under design is a successor module which, based on 1 Mbit chip technology, will offer 8 Mwords per module with error detection and correction for 24 bit words.

General FASTBUS Features

To ensure reasonable access to the large memories present in each custom FASTBUS module type, block mode transactions are implemented throughout. Logical addressing modes are also supported by the module suite. Logical addressing is particularly critical in the BULKSTORE module for use with our MAPPER data compaction algorithms.

Although it is tempting during design to introduce special dedicated buses between modules to simplify the design and to enhance performance, our designs adhere to the standard transactions for intermodule communication defined for FASTBUS systems. As a consequence the MAPPER module is significantly more complicated than it otherwise might have been. This is particularly apparent at the bus interfaces where block input and block output must be buffered. However, since the design is strictly generic FASTBUS, when changing requirements dictate a processor with greater flexibility, albeit with lower performance, it will be possible to substitute a commercial product with an onboard general purpose processor for this custom module.

External input signals for the TOF are brought in through the auxiliary connector. These inputs include four sets of parallel data for each of the four separate channels in the module and a single parallel bus with the time information. We associated the time inputs with the parallel detector inputs and feel that in this case use of FASTBUS protocols for PMC/TOF communication is neither effective nor appropriate.

Data Acquisition Software

Integration with DCL

During the early stages of the software requirements definition, it was decided to integrate our Data Acquisition Command Language (DACL) system into the Digital Equipment Corporation VMS environment to the maximum degree. The advantages of this policy are immediately obvious. VMS constitutes a versatile, mature, stable software development environment with many powerful built-in features which would otherwise require local support to develop and maintain. Particularly interesting among these facilities are the command language interpreter which supports the standard DCL command language, the online help facility, and condition handling and messages.

All DACL commands are implemented as extensions to the standard DCL command language[7]. This is accomplished using the VMS Command Language Definition (CLD) utility and completely eliminates the need for local personnel to develop, modify and maintain a command language interpreter for the data acquisition system software user interface. As a consequence, DACL commands exhibit the same syntax and semantics as standard DCL commands and reside in the same command tables. The CLD provides a modular, high-level mechanism for integrating new commands into the system without creating any impact upon existing data acquisition software. In addition, it provides the means for supporting several versions (*e.g.* production and development versions) of a software product simultaneously, and allows a user to select dynamically at run time the version which is to be executed.

Even more important than the benefits accrued by the development community are the advantages perceived by the users. Most LANSCE users are already familiar with the DCL command language from previous interactions with computers of the VAX family. Consequently, the DACL system appears as merely an extension of an already comfortable environment. Overheads associated with familiarizing the user community with (yet) another command language with its own syntax and semantics are eliminated entirely, and transition into the DACL environment is eased considerably. Because DACL is thoroughly integrated with the DCL system, no special action is required to enable the data acquisition system: a DACL command may be executed from any point at which a DCL command is valid.

Integration with DCL offers numerous additional functional benefits. In addition to the data acquisition functionality which is locally defined for DACL commands, the DACL system has at its disposal all of the features of the DCL system. DACL commands can be included in DCL command files where they may exploit the control structures, symbol substitution mechanisms, arithmetic operations, lexical functions, condition handling and input/output capabilities of the DCL system. In this manner, the DACL system constitutes an extension to a powerful, procedural programming environment, and frees local personnel from the responsibility of maintaining the vast majority of that environment's features. Indeed, some LANSCE scientists have utilized these flexible DCL/DACL features to provide data acquisition "shells" for their instruments.

DCL (and by extension, DACL) also provides a flexible user interface which adapts to the level of expertise of individual users. Commands and qualifiers are meaningfully named and are therefore easily assimilated and retained by the casual user. In their unabbreviated form they provide a self-documenting feature for interactively executed commands or command procedures. Expert users, however, can employ terse, abbreviated forms of all command names and qualifiers in their interactions with the system.

The structure of a typical DACL command is shown schematically below:

```
$ NAME/SYNTAX /qualifiers parameter
```

The NAME of the command specifies the class of hardware module, data acquisition facility or activity which will be operated upon by the command. The /SYNTAX component specifies the generic operation to be performed. For example, the command name MEMORY refers to a class of commands which perform histogram memory management for the DACL system. Particular memory management operations are specified by appending a /SYNTAX qualifier to the command name. Extending this example, MEMORY/ALLOCATE specifies an operation to define and reserve storage for a histogram data area. Similarly, MEMORY/LIST causes the contents and/or characteristics of a data area to be written to an output device (or to a file). MEMORY/DELETE removes a histogram data area definition and releases the memory associated with it, and MEMORY/MODIFY permits a user to change data area characteristics.

A typical DACL command accepts a single parameter on the command line. The parameter is considered to be the object upon which the command operates. In the case of the MEMORY commands specified above, the parameter would specify an individual data area or group of data areas to be allocated, listed, deleted or modified, respectively. DCL rules require positional syntax for parameters. To avoid the confusion which always accompanies positional parameters, the DACL system permits no more than one parameter on a command line.

Most DACL commands accept one or more nonpositional qualifiers on the command line. Qualifiers serve to modify the command action thereby to customize the result. Qualifiers may be required or optional, and may have an associated, user-specified value (or list of values). In the case of the MEMORY example referenced above, qualifiers would be provided to specify histogram characteristics such as dimension, protection, and data type. The combination of qualifiers and a single parameter allows the user to provide position independent information on the command line and results in a far simpler user interface.

The VMS CLD utility provides a high-level language for defining the command name, associated syntaxes, qualifiers and parameters for each DACL command. This facility supports definition of default values, validation criteria and disallowed combinations of command elements, and specifies the image to be executed when the command is invoked. Once a command definition is compiled and integrated into the DCL environment, the DCL command language interpreter handles all parsing, validation and parameter passage between the command and the underlying executable image.

VMS facilities are also employed for condition handling and for issuing messages. All status messages issued by any DACL software emanate from the VMS message facility. This policy allows the development team to maintain central libraries of generic messages which are accessible to all DACL software. The message facility permits each generic message to be customized at execution time with numeric or textual inserts. Through this approach, all DACL messages share a common format which is consistent with the format used by other VMS applications. In addition, reuse of messages is encouraged, thereby reducing the number of ways in which a particular error condition is presented to the user to only one. Finally, use of the VMS message facility allows condition values to be propagated back to the DCL level from any DACL command, thereby allowing the user to test automatically the completion status of a command within a command procedure, and to take meaningful action based on the result of the test.

The VMS online help facility is employed to provide online documentation for all DACL commands and messages. The entire DACL user's guide is available online to any computer user. Because the commands are integrated into DCL, the format of the help screens and organization of the help libraries is identical to those for standard DCL commands, thereby further reducing the overhead incurred by an unfamiliar user in becoming acquainted with DACL. Furthermore, every message that can be issued by the DACL system has a help file entry which elaborates upon the meaning of the message and suggests a course of action for the user to follow to resolve the error condition.

The software which underlies the DACL commands is organized into a single program for each command name/syntax combination. This software is written exclusively in Pascal using sound design and development techniques[8-11] within the context of a rigorous software engineering methodology. Software is highly modular (only one routine per file) and is rigorously specified, designed and documented *before* implementation. Each program is verified and validated by executing a formal suite of validation tests. Because of the thoroughness with which all software is specified, designed, implemented and tested, the DACL system has proved to be an extremely reliable environment, and one which is very easy to modify and maintain.

The DACL software is characterized by an open architecture which permits the ordinary user to access system internals in a high-level fashion. The system supports libraries of standard interface routines which permit a user to interact simply with the data acquisition environment. This provides users with simple building blocks from which they can construct sophisticated custom software for performing specialized monitoring, control or analysis tasks which are unique to their instrument or experiment. The standard routines are black-box solutions which are easily comprehended by a user and which provide controlled and synchronized access to the internal data structures.

The entire DACL system is comprehensively documented at all levels. All design and maintenance documentation is kept current and placed inline within all Pascal sources. This documentation includes three levels of procedural documentation as well as an informal data dictionary. In addition, all commands and library routines are documented at the user level and this documentation is published in the facility software user's guide.

The DACL Heap

Every data acquisition system that relies on more than one program to perform its functions requires a set of global control data structures in which the system configuration is recorded and which can be used to communicate information among the various constituent computer programs. Traditionally, these data structures have been implemented as statically allocated global data structures of predeclared length (usually COMMON blocks).

Within most environments, such statically-defined control structures may create more problems than they solve. The design and implementation difficulties are obvious and quite significant. A designer must foresee every control structure and substructure that will be required by the system, and must be able to predict the amount of storage to preallocate for each. This generally results in each control structure being overallocated to allow for "future expansion". This approach promotes extremely inefficient utilization of resources and provides little additional flexibility.

Most control structures have large amounts of unused storage associated with them and in the event that a control structure overflows its preallocated storage there is no possibility of allocating any of the unused resources of another structure. Consequently, the overhead associated with this data structure organization is continually increasing and the efficiency of the system is always on the decline.

In the LANSCE environment the traditional approach has other serious deficiencies. Data acquisition requirements vary drastically across the spectrum of supported instruments and experiments. What may be adequate control structure storage allocations for one instrument or experiment are likely to be entirely too small for another and excessive for a third. Even more importantly, the mix of required control structures varies widely across the various experiment and instrument configurations. For example, some experiments require large numbers of independent histogram data areas whereas others utilize only a few. Therefore, in order to use the traditional approach, the each control structure must be extensively overallocated thereby to accommodate the demands of the worst case configuration of each experiment. This imposes a huge overhead upon every experiment and reduces all experiments to the least common denominator in terms of efficiency.

Conventional systems that utilize static allocation schemes also exhibit severe functional problems. Regardless of the space available in other (possibly unused) control structures, the user is strictly constrained in the number of control structures of a particular class which can be defined. For example, such a system may preallocate storage for 100 histogram data areas. This places an absolute upper bound on the number of data areas which may be allocated. There is no capability for "borrowing" unused resources from an undersubscribed structure. Extending the allocation for a particular control structure requires extensive modification of the existing system, followed by a global system rebuild. Similar drastic action is required whenever a new control structure is added. As a result, the system becomes functionally rigid. New applications must be postponed until the required system software applications are performed, thereby stifling the creativity and spontaneity of the user community. Compounding this problem, conventional approaches generally implement these structures as arrays in an obsolete programming language. This virtually guarantees a closed architecture which makes access by other applications difficult and extremely prone to error.

A far more appropriate and effective policy for allocating control structures for the DACL system borrows ideas which have been used for many years in computer operating systems and modern programming languages. This approach eliminates the problems of static allocation by implementing a locally developed facility which dynamically (at run-time) allocates control structures from a common pool of preallocated storage[12]. Owing to its functional equivalence (and analogous syntax) to the dynamic variables and pointers which can be found in modern programming languages such as Pascal and Ada[13], we designate this facility the DACL Heap.

The implementation details of the DACL Heap are completely hidden from the majority of the LANSCE data acquisition user community. For those individuals who are developing applications which must interface with it, the only functional difference between the DACL Heap and its Pascal or Ada[13] counterparts is the system-global and nonvolatile nature of the DACL facility. The DACL Heap is accessible by every image in the system and retains its integrity across image executions.

The DACL Heap consists of a data base of singly linked lists of logical descriptors. Each linked list corresponds to a particular class of control structures, *e.g.* memory control blocks (for histogram data areas), scaler descriptors, preset scaler descriptors, descriptors for each of the FASTBUS and CAMAC hardware modules known to the system, *etc.* Each descriptor is 128 bytes in length and is subdivided into fields to contain the information necessary to describe the entity represented thereby. Of these 128 bytes, five are preempted by the system. The DACL Heap currently supports a maximum of 256 descriptor classes of which approximately fifty are predefined by the DACL system. The remainder are available to be customized by the user community.

The DACL Heap is implemented within a VMS global section which has the capacity to hold 10000 descriptors. Each descriptor represents a single entity (a single histogram data area, for example). All control structures are comprised of linked lists of related descriptors within the DACL Heap. As a result, no predefined limit exists (within the 10000 record maximum size) for the size of a particular control structure. In addition, only those control structures which are specifically allocated by a data acquisition application are ever present within the DACL Heap, and these structures always consume the minimum storage necessary. Consequently, control structure allocation becomes efficient and compact.

Each descriptor is comprised of a standard partition (the five bytes of system overhead) and a control-structure-dependent partition. The standard partition is subdivided into two pointers, designated NEXT and MORE, respectively, and a control structure identifier called the TAG. These fields occupy the first five bytes of every descriptor. The remaining 123 bytes of control-structure-dependent information is partitioned into fields as necessary to describe the entity associated with the control structure (as specified by the current value of the TAG). The TAG may assume any of 256 enumeration values.

The NEXT pointer references the next logical entity in the control structure. Although control structures may be multiply linked, they are never linked bidirectionally. The DACL Heap implementation also recognizes that 123 bytes of control-structure-dependent information is not necessarily sufficient to describe completely every DACL entity. It therefore provides the capability to define an extension descriptor which can be uniquely partitioned to accommodate the overflow from the primary descriptor. The extension descriptor is always referenced from the primary descriptor *via* the MORE pointer. An extension descriptor is a distinct DACL Heap construct with a standard partition (including a MORE pointer) of its own; it can be extended as well. Consequently, a single logical entity can be represented within the DACL heap as a linked list consisting of a primary descriptor and an unlimited number of extension descriptors, all linked through the appropriate MORE pointers. In this spirit, the DACL Heap also supports a dynamic string facility which permits strings of any length to be represented and eliminates the necessity of declaring a maximum length for any DACL-Heap-resident character string. Figure 5 is a schematic representation of a typical DACL Heap logical descriptor. Figure 6 illustrates several ways in which logical descriptors can be linked together to form control structures.

The DACL Heap facility provides a complete set of library resident utility functions for interfacing with the control structures which reside therein. Included are utilities to allocate descriptors from the common pool and to release (deallocate) unneeded descriptors back into the pool. A suite of utilities is provided to maintain (insert into and delete from) the control structure linked lists. Utilities for resetting the DACL Heap and for listing its contents are also provided, as well as a set of procedures which support the dynamic string facility. Because the DACL heap is a

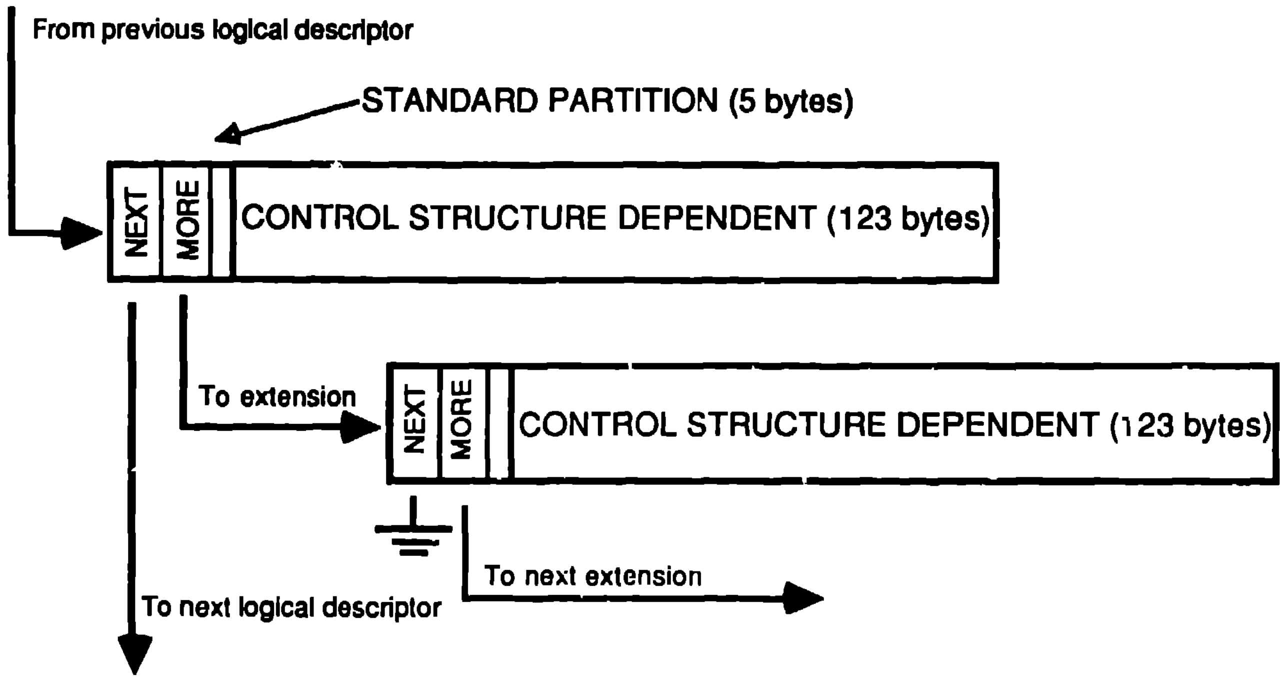
From previous logical descriptor
STANDARD PARTITION (5 bytes)
NEXT
MORE
CONTROL STRUCTURE DEPENDENT (123 bytes)
To extension
NEXT
MORE
CONTROL STRUCTURE DEPENDENT (123 bytes)
To next extension
To next logical descriptor

Figure 5. Schematic representation of a typical DACL Heap logical descriptor.

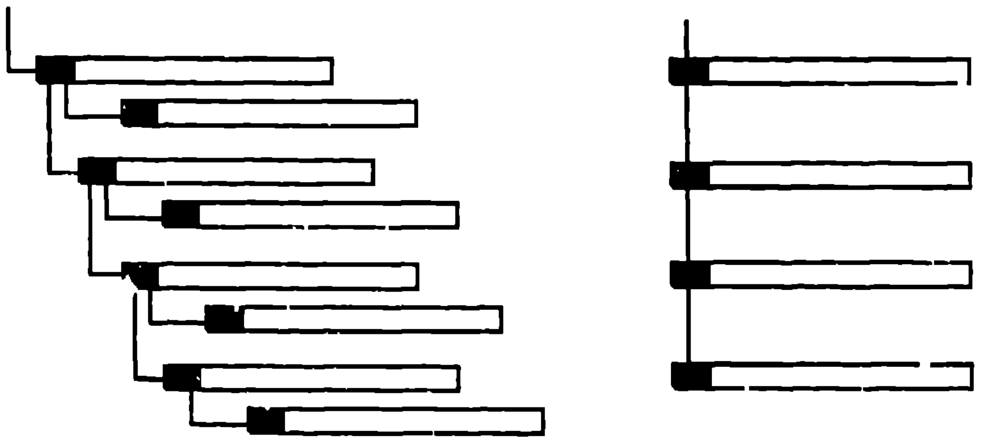

Figure 6. Typical DACL-Heap-resident control structures.

resource which may be shared by many processes and/or users, a pair of utilities is provided which employs the VMS Distributed Lock Manager to mediate access to Heap contents. By making extensive use of recursion, the DACL Heap utilities can process control structures which are linked in very complex ways.

The DACL Heap utilities perform extensive error checking and validation to ensure that the integrity of the Heap is not compromised. They allow the details of the DACL Heap's implementation to be effectively hidden from all programmers, thereby forcing the use of the standard interfaces. These interfaces protect the contents of the DACL Heap from the most error prone operations and significantly reduce the possibility of data corruption due to improper access.

The DACL Heap is designed to provide a simple, consistent interface for non-DACL applications as well. Although its architecture is deliberately kept open to encourage access by a wide range of applications, considerable effort has been expended to keep its underlying structure and implementation details hidden from all programmers and data acquisition users. One method for accomplishing this goal is to reference each DACL Heap descriptor by a unique, user-specified name which is assigned when the descriptor is allocated. This approach eliminates the need for users to specify otherwise meaningless offsets into low level data structures and helps to keep the DACL Heap interface at a very high level. As a bonus, it provides a logically consistent, self-documenting framework in which LANSCE users can operate. The implementation of DACL Heap descriptors as Pascal records further simplifies referencing the information stored therein. Data must be referenced by record and field name so the possibility of data corruption due to incorrect offsets into a data structure is eliminated.

In summary, the DACL Heap provides a practical alternative to traditional, statically-allocated control structures. Its implementation is functionally more powerful and more efficient in its utilization of resources than are static structures. It offers the additional advantage of significantly enhancing system adaptability and robustness and eliminates the need for frequent software modifications by system personnel to implement user requested enhancements.

State Machine Features

One of the major operational deficiencies of many broadly functional systems is the ability to execute syntactically valid commands in a semantically invalid sequence or context. As the functionality and complexity of systems increases, the opportunity for committing such errors also rises dramatically. This is a particular problem at LANSCE owing to the large number of new or unfamiliar users. Of particular concern are errors that compromise data integrity. Owing to the great expense incurred in producing the LANSCE neutron beams, and the high user demand for access to the facility, such errors can have serious economic, political and scientific repercussions.

Individual data acquisition commands and facilities can be designed to enhance their own hardiness and reliability, and certainly this strategy has been exploited to the maximum degree in the DACL system. However, a consistent architectural approach is required to make the system, as a whole, more robust. In order to achieve this goal within DACL, we decided to implement the entire software system as a *state machine*. This implementation defines four valid data acquisition states for the DACL system: INITIALIZE state, RUN state, HALT state and PAUSE state. The prevailing state of the system is maintained in a control structure in the DACL Heap. At any time during data acquisition or setup, the DACL system is required to be in one (and only one) of these states. This approach postulates that there are no dangerous commands, merely dangerous contexts, and that these contexts may be identified and defused prior to command execution.

The DACL state machine implementation defines a set of valid data acquisition states for each DACL command. Upon invocation, the first task of every command is to check the prevailing state of the DACL system against its valid set. If the prevailing state does not match one of the valid states an error message is issued and the command gracefully exits without performing any further processing. As such, command execution is restricted to occur from within a sensible context, and the vast majority of semantic errors can be identified and automatically avoided prior to causing any damage. To further simplify the implementation, the data acquisition state validation process is executed by a single, library-resident, utility function, thereby supporting the easy integration of state machine features into all DACL commands.

The possibility of inadvertently placing the DACL system into an undesired state (as the result of a side effect of a valid command executed in an appropriate context) must also be considered. The DACL state machine implementation prevents the occurrence of this situation by defining a set of state transition commands. These commands represent the only means by which a user may change the data acquisition state of the DACL system. There is one (and only one) state transition command for each valid data acquisition state transition. Furthermore, the state transition commands have no functionality beyond changing the DACL state; all other DACL commands are expressly forbidden from changing the data acquisition state. Hence, the data acquisition state can never be changed implicitly and the problem of side effects is eliminated.

The various DACL data acquisition states are shown schematically in Figure 7 along with the the DACL commands that invoke valid data acquisition state transitions. The INITIALIZE state is provided for data acquisition configuration operations. Operations which define or modify DACL control structures, or which download hardware modules are valid if executed from this data acquisition state. The HALT state provides a well defined state for the data acquisition system to be in prior to the beginning of a run and after the conclusion of a run. The RUN state is

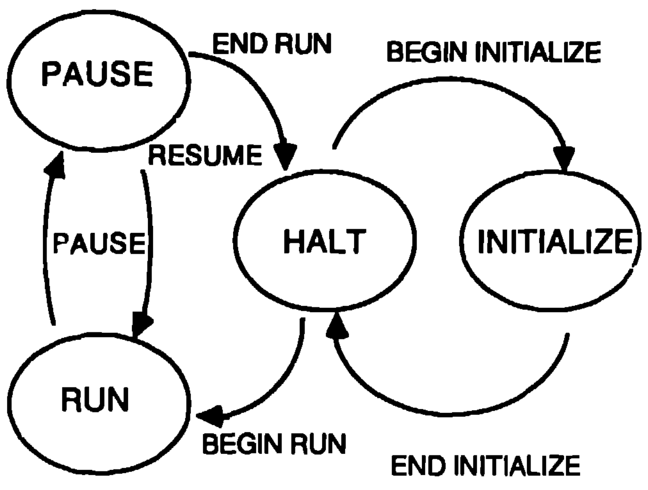

Figure 7. The DACL state machine.

initially reachable only from the HALT state. Upon entering the RUN state, data acquisition hardware modules are automatically started and the time and date of the beginning of the run is fixed and stored in the DACL Heap. The RUN state may be exited only by entering the PAUSE state. Upon transition into PAUSE, data acquisition hardware is automatically inhibited. The PAUSE state is provided to allow the user to interrupt the data acquisition without terminating the data acquisition run. In addition, data can only be saved to a disk file from the PAUSE state. From the PAUSE state the user may re-enter the RUN state to resume data acquisition, or may terminate the run by re-entering the HALT state. In the former case, the data acquisition hardware is re-enabled and restarted. In the latter case, the run is terminated, the run number is incremented and an ending time and date stamp for the run is fixed and stored in the DACL Heap.

Early in the design effort for the state machine facility it was recognized that situations would arise in which enforcement of the state machine features could cause significant inconvenience and provide no real benefits. In diagnostic environments, for example, it is often convenient to bypass state machine features thereby to expedite the debugging or testing function without being forced to issue numerous state transition commands. For this reason a mechanism was devised by which a privileged user (system manager, developer or lead instrument scientist) can enable a mechanism for bypassing data acquisition state checking. If this feature is enabled, state checking can be bypassed by any user for the current execution of a given command by including a standard qualifier on the command line when the command is invoked. It has been our experience that this feature is used only rarely, and has never been used during production data acquisition.

DACL Software Functionality

Introduction: The DACL system is designed to function as a comprehensive acquisition, control and data management system for all experiments mounted at the LANSCE facility. It is thoroughly integrated with both the VAX/VMS and FASTBUS environments, and in conjunction with standard features of each environment, provides all necessary functionality to configure the experiment, and to supervise its execution, to provide status information which documents the progress and health of the experiment, to display data in a variety of formats and to perform a number of other functions in support of data storage, management, manipulation and retrieval, as well as hardware development and diagnostic activities.

The commands that comprise the DACL software system can be logically partitioned into five facilities. The Resource commands address the configuration and management of the various hardware and software resources associated with data acquisition at LANSCE. Included among these are a suite of commands to configure, download and interrogate the locally developed FASTBUS hardware modules which are the backbone of the data acquisition system. Additional commands are provided to configure and control a variety of commonly used CAMAC modules including scalers, preset scalers, temperature controllers and stepping motor controllers. Additional resource commands are provided to manage and examine the contents of the DACL Heap, and to perform memory management for histogram data areas.

DACL Control commands provide the functionality required to control the execution of an experiment. All data acquisition state commands are included in this category, as well as commands to interrogate the system for its current status. In addition, an automated facility for processing data acquisition tasks in a background mode is included among the Control commands.

The DACL Data Management facility is comprised of commands which control the creation of specially formatted files in which experiment data and configuration information can be stored at the conclusion of a run, as well as commands which automatically catalog and archive these files. The Data Management facility supports a library of standard utility subprograms which permit a user to access archived data in various ways. Also supported is a command which permits a user to perform algebraic operations on entire data histograms.

The DACL Graphics facility provides a suite of commands for configuring and executing data plots in near time on a variety of devices. DACL graphics capabilities range from simple, low performance, monochrome histogram plotting to high performance, workstation-based, color plots of multiple histograms and transformed data.

The DACL FASTBUS Diagnostics facility is a comprehensive suite of locally developed commands which are used in conjunction with the debugging and checkout of locally developed FASTBUS hardware modules. Included within this facility is a suite of low level commands which perform basic FASTBUS transactions as well as a complete set of diagnostic commands for each FASTBUS module.

<u>The DACL Resource Commands:</u> This facility provides the functionality to manage all of the various resources required for LANSCE data acquisition. It provides support for all hardware modules used by an experiment as well as DACL Heap and histogram data area memory management functions.

The hardware modules supported by the DACL Resource facility include the four locally developed FASTBUS modules: the Programmable Master Clock (supported by the PMC command), the Time-of-Flight Buffer module (TOF command), the MAPPER module (MAPPER command) and the Bulk Memory module (BULKSTORE command). Supported CAMAC modules include scalers (the SCALER command), preset scalers (PRESET SCALER command), temperature controllers (TEMPERATURE) and stepping motor controllers (MOTOR).

With only minor variations, the hardware Resource commands all employ the same set of generic syntaxes to specify operations upon hardware modules. These syntaxes include /ALLOCATE, /CLEAR, /INITIALIZE, /LIST, /MODIFY, /NEW and /READ. Additional qualifiers and a command parameter may be used in conjunction with the command name and syntax to specify data or control information for the operation specified by the command name/syntax combination.

In order to make the existence of a module known to the DACL system, the user must employ the appropriate Resource facility command with the /ALLOCATE syntax to create a logical descriptor for the particular module and to insert the descriptor into the appropriate control structure linked list in the DACL Heap. The /ALLOCATE operation associates a user-specified name (which is provided as the command parameter) with the module and stores the name in the module descriptor. It also provides module configuration data specified with command qualifiers. The configuration data is also stored in the DACL-Heap-resident logical descriptor. For example the following command defines a logical descriptor for a FASTBUS Bulk Memory module (which the user has chosen to name HISTOGRAMS) located at FASTBUS geographical address 20:

```
$ BULKSTORE/ALLOCATE /SLOT=20 HISTOGRAMS
```

In general, /ALLOCATE operations can be performed only from the INITIALIZE data acquisition state.

The /INITIALIZE syntax causes information resident in a module's logical descriptor to be downloaded into the specified module, thereby readying the module for the data acquisition task. Generally, the module to be initialized is specified by name in the command parameter although the /ALL qualifier may be specified instead of a module name. In the event that /ALL is specified, all allocated modules in the control structure specified by the command name (*e.g.* BULKSTORE) are downloaded. An /INITIALIZE operation can only be performed from the INITIALIZE data acquisition state. In order to initialize the Bulk Memory module which was allocated in the preceding example, either of the following commands may be used:

```
$ BULKSTORE/INITIALIZE HISTOGRAMS
```

or

```
$ BULKSTORE/INITIALIZE /ALL
```

The /MODIFY syntax is similar in function to the /ALLOCATE syntax. It permits a user to change the characteristics of a hardware module which is already allocated. The /MODIFY syntax searches the appropriate control structure for a descriptor which corresponds to the name specified by the user in the command parameter. The descriptor is updated with new information as specified through qualifiers which are included on the command line. The /MODIFY syntax does not download any new information to the affected module--only the DACL-Heap-resident descriptor is changed. In order to communicate the changes to the hardware module, it must be reinitialized. /MODIFY operations are permitted from either the INITIALIZE or the HALT data acquisition state.

The /NEW syntax reinitializes the control structure associated with the command name. This causes all logical descriptors which are currently present in the control structure to be deleted and deallocated. Upon completion of a /NEW operation, no entities of the associated control structure are known to the system. A /NEW operation may be performed only from the INITIALIZE data acquisition state.

The /LIST syntax permits a user to display one or more members of a particular control structure. The /LIST operation searches the associated control structure for the descriptor which corresponds to the name specified in the command parameter. The contents of the logical descriptor are then formatted into a report which is issued to the appropriate output device or file (as specified by the user). Alternatively, the user may specify /ALL instead of providing a name in the command parameter. In this case, the report contains an entry for each currently allocated entity associated with the control structure. A /LIST operation is valid from any data acquisition state.

The /READ and /CLEAR syntaxes are generally associated with modules that contain local memories. The /READ syntax causes the contents of the local memory or storage register to be read and the associated logical descriptor to be updated with the value. The /CLEAR syntax causes the local memory or register to be zeroed. /CLEAR operations are generally permitted from either the INITIALIZE or the HALT data acquisition states. /READ operations are permitted from any data acquisition state.

The Resource facility also supports commands to manage DACL Heap resources as well as histogram data area memory. Two DACL Heap commands are provided. HEAP/NEW provides a mechanism for resetting the DACL Heap to a known state. When this command is issued, all existing DACL Heap records are deallocated (returned to the common pool), all pointers are cleared and a predefined set of descriptors is allocated to configure the DACL system in its default state. HEAP/LIST generates a report which displays the standard partition of each record which is currently allocated in the DACL Heap.

The DACL histogram data area memory management subsystem provides the user with great flexibility to configure histogram memory in a variety of ways to support a data acquisition task. Under the DACL system, histogram data areas may be allocated from either FASTBUS memory or from VAX resources (a global section). Three classes of data area are supported: FASTBUS, SCRATCH and VAX. FASTBUS data areas are allocated from the memory resident in an allocated Bulk Memory module. FASTBUS memory may be used for data acquisition only. VAX and SCRATCH data areas are allocated from a VAX global section. VAX data areas are used exclusively for data acquisition; SCRATCH areas may be used for any purpose at the discretion of the experimenter.

The MEMORY/ALLOCATE command defines a logical descriptor for each histogram data area required by the experiment. The name and substructure of the data area is specified as the command parameter; class and other configuration information is specified *via* qualifiers specified on the command line. Included in the configuration information which may be specified are data type (8-,16- and 32-bit integer, 32- and 64-bit real), data area dimension (1-d and 2-d data areas are supported), protection code and an optional title of unlimited length. FASTBUS and VAX data areas may be allocated only from the INITIALIZE data acquisition state. SCRATCH data areas may be allocated from any data acquisition state.

The DACL system provides a mechanism for allocating groups of related data areas which share a common name. An optional *structure specifier* may be associated with any data area name in the command parameter of the MEMORY/ALLOCATE command. This specifier is of the form [g:s] where *g* specifies the number of groups of histograms to be associated with the data area name, and *s* specifies the number of subareas (histograms) per group. All other configuration information specified on the command line then applies at the subarea level. For example, the data area specification MONITOR[2:5] refers to a data area containing ten histograms organized into two groups of five histograms each. If the MEMORY/ALLOCATE command specifies a dimension of 1000 channels for the data area, *each* component histogram will be 1000 channels long. Individual histograms within the data area may be referenced using an identical specifier notation.

The histogram data area memory management system also provides commands to release data histogram memory (MEMORY/DEALLOCATE), clear histogram memory (MEMORY/CLEAR), modify the characteristics of a histogram data area (MEMORY/MODIFY), list the contents and/or characteristics of a histogram data area (MEMORY/LIST) and clear the memory management data structures (MEMORY/NEW). Execution of each of these commands is subject to data acquisition state validation.

The DACL Control Commands: The Control facility is implemented as two subsections. The first addresses the functionality required for the DACL state machine; the second specifies a subsystem for automatic run submission and control.

The state machine features of the DACL system are supported by the Control facility state transition commands. Four basic transition commands are provided: BEGIN, END, PAUSE and RESUME. The BEGIN and END commands each require a parameter which specifies the data acquisition state to be initiated or terminated. The only function of any state transition command is to change the data acquisition state from the current state to a specified result state. The data acquisition state from which any transition command may be executed is well defined and is validated prior to execution of the state change. See Figure 7 for a summary of the effect of each state transition command.

The Control facility supplements the state change commands with the STATUS command. Invocation of this command generates a report to the user's terminal which specifies, among other things, the current data acquisition state of the system, the last command to be executed, general descriptive information for the experiment and the state of various control and status flags.

The Control facility also supports a suite of commands for automatic submission, execution and control of data acquisition tasks in a background mode. Data acquisition tasks are defined using combinations of DACL and DCL in DCL command procedures. The PRL command may then be used to define and modify the sequence of execution of any number of existing data acquisition command files by modifying the *pending runs list*, a DACL-supported data structure which contains the file specifications of all pending data acquisition command files. The PRL command utilizes the EDT full screen editor for its user interface, thereby to provide a familiar low overhead environment in which the user can operate.

The RUN_SUBMITTER facility interacts with the pending runs list, automatically creates detached processes for each data acquisition command file and executes them serially. This facility performs all necessary synchronization with DACL resources and processes to execute the data acquisition task, and provides status information to the user through screen messages and VMS mail messages. Additional commands permit the user to delete currently executing data acquisition jobs and use preset scalers to control the execution of the detached data acquisition processes.

The DACL Data Management Facility: The DACL Data Management facility provides the functionality to save experiment data and configuration information to a disk file, as well as to automatically archive and catalog the resulting file in the LANSCE data archive. In addition, this facility supports libraries of utility modules to simplify user access to saved and/or archived data as well as a high level data manipulation subsystem.

At the conclusion of a data acquisition sessions, the user usually desires to save the data which has been collected (as well as some parameters which specify the configuration of the

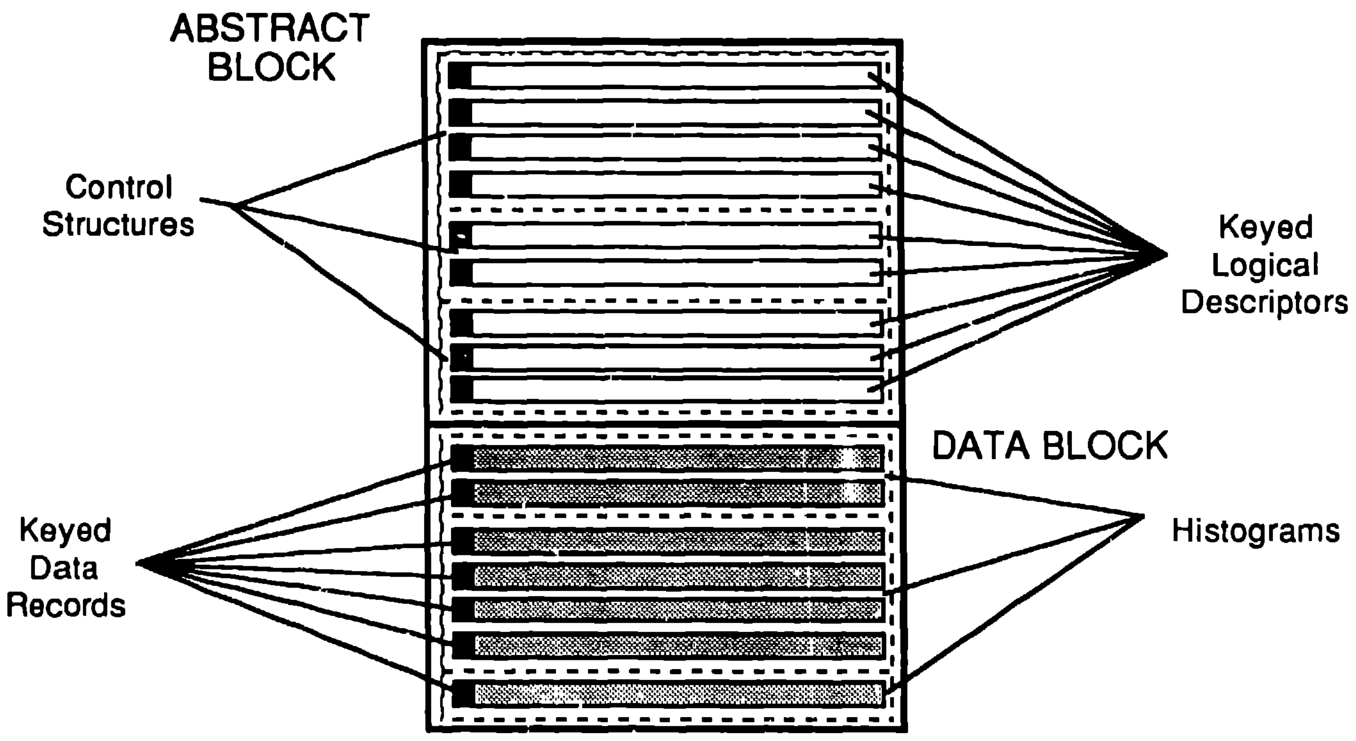

Figure 8. Structure of the DACL instrument data file.

experiment) to a disk file prior to reconfiguring for a subsequent run. The conventional approach to the design of such data files utilizes a sequentially organized file with a rigidly defined internal format.

The disadvantages of the sequential file approach are quite serious. In an environment such as LANSCE where the data file format must service many different users and spectrometers, there are two immediate adverse consequences. First, the file format must be defined in a manner which accommodates all possible information which may be required by any application. This often results in data files which contain large amounts of irrelevant or unnecessary information. The second consequence results from the fact that changing the file format (to accommodate a new application, for example) makes all previously written files obsolete (and unreadable). System personnel are then burdened with the overheads associated with modifying the file format as well as with the additional effort which must be expended to modify utilities which read and write the data files. In most cases it is also necessary to develop utilities which can convert obsolete files into the (current) standard format.

Sequential organization also suffers severe performance overheads. Access to individual entities within the file (a specific data set in a group of 100, for example) can be slow and cumbersome. Updating the file (to include data analysis processing status or other previously unavailable information) is difficult.

The DACL approach abandons sequential files in favor of the indexed sequential access mode (ISAM) organization. ISAM files permit either indexed or sequential access as required by the application. Any record in the file can be very rapidly accessed using the indexed mode. The indexed access may then be followed by a series of sequential accesses to read a large amount of information.

The DACL data file design partitions a data file into two sections: an abstract block and a data block. The resulting Instrument Data File (IDF) then contains all information required to characterize the associated run. A schematic representation of a typical IDF is presented in Figure 8.

The abstract block contains information which describes the experimental state and conditions. Any information that is resident in the DACL Heap may be specified for inclusion in the abstract block. At any time before or during a run, the user may specify which DACL Heap control structures are to be automatically placed in the abstract block when the data is saved. This information is specified in a text file called the Abstract Descriptor File (ADF). An ADF may be created or modified using a text editor, and contains a list of the control structure tags which identify the control structures to be merged into the abstract block when the data is saved.

When the abstract block is built, a unique key is composed for each logical descriptor which is included. Keys are constructed in a manner which permits the corresponding control structures to be exactly recreated in the DACL Heap, exclusively from information contained in the abstract block. The user may construct an ADF which is customized for the particular experiment or may choose to reference the more generic default ADF provided by the system. Often the lead scientist for each spectrometer defines a default ADF which is employed by all users of that instrument. In this manner, the abstract block can be automatically constructed to contain all pertinent configuration information without the inclusion of superfluous configuration records.

The IDF data block contains the data collected during the run, organized by histogram name, group and subarea. Data histograms are decomposed into the data block in a manner similar to abstract records. Data block record lengths are extremely large, approximately 32000 bytes. This enables the system to store or retrieve typical histograms with only a small number of high level file access.

The major advantage of the IDF file format is its flexibility. Because the file format is dynamically defined at run time, IDFs need contain only that information which is directly pertinent to the application. Data files are therefore very compact and can be read or written without incorporating complex protocols. All files can be read and written by the same set of utilities, regardless of the information they contain or when it was written. Consequently, no file ever becomes obsolete. Finally, access to the information stored in an ISAM file is very efficient. Performance is comparable to that of a sparsely populated hash table[14], but does not suffer the large resource overhead necessary to make hash tables efficient.

The SAVE/DATA command performs the necessary processing to create an IDF from the experiment data and the appropriate DACL-Heap-resident logical descriptors. The command parameter specifies the file specification of the ADF which will be referenced in building the abstract block. Command line qualifiers allow the user to specify whether the data should be stored in the data block in compressed format and whether the IDF should be automatically cataloged in the LANSCE data archive. The user may also specify a short identification string to be associated with the IDF.

The SAVE/DATA command may only be executed from the PAUSE data acquisition state. Furthermore, the system intervenes to prevent the user from entering the HALT state (terminating the run) until data has been saved or until the user has expressly informed the system (*via* the NOSAVE command) that the data is to be discarded. Upon creation, the IDF is automatically time stamped with the beginning and ending date and time for the run as well as other information including instrument name, experimenter and run number.

The DACL system automatically moves user data files from the satellite microVax computers associated with each instrument to a central repository on a non-data-acquisition network node. The network process that performs this task periodically searches the data acquisition directories on the satellite computers for IDF's which are marked for cataloging. Whenever such a file is identified it is copied across the Ethernet to the central repository. An ISAM file containing a directory of all cataloged IDF's is then updated and a VMS mail message (stating that the IDF has been archived and cataloged) is sent to the appropriate user on the satellite computer. At the current time, the IDF is automatically deleted from the satellite system upon successful archiving. A future enhancement will retain archived IDF's on the satellite, but will mark these files as archived. A separate command will be provided to allow users to automatically delete archived files from their systems.

In the event that a user does not specify the catalog option when an IDF is built, the IDF will not be archived by the DACL system. At any later time, the user may invoke the CATALOG command to mark such an IDF for archiving. In this event, the automated archiving system will automatically perform the archiving operation.

When a file is archived in the LANSCE data repository, the system assigns it a unique name which is constructed from the name of the source instrument and the current date and time. Responsibility for management of the file then passes from the user to system personnel. Commands are provided to the users which allow them to peruse the contents of the data archive and retrieve a copy of any IDF for subsequent data analysis activities. The ARCHIVE/LIST command is provided to allow the user to perform wildcarded searches of the data archive on combinations various keys including all or a substring of experimenter name, instrument name, run number, date/time, identification string or data location.

The LANSCE data archive currently exists on a large magnetic disk. Provisions are made for moving old IDF's to magnetic tape as the medium begins to fill. A future enhancement is expected to replace these magnetic media with optical technology.

Included in the DACL Data Management facility is a rich set of library-resident utility routines which support user access to IDFs. These routines permit the user to restore data from an IDF either into a local buffer or into a preallocated histogram data area. Other routines allow searches of the data archive to be performed under program control. Additional routines are provided to allow a program to restore selected DACL heap control structures from an IDF. These routines are carefully documented in the facility software user's guide. The existence of these utilities has helped to establish an environment in which users can develop custom software to interact with the data in IDF's with virtually no assistance from system personnel.

The Data Management facility also supports a complementary data manipulation facility through the CALCULATE command. This command provides the functionality to perform channel-by-channel algebraic manipulations upon data areas, scalers and constants. It permits a user to specify a mathematical formula in ordinary infix algebraic notation. Naming conventions are employed to identify each entity in the formula as a data area, scaler value, constant or DCL symbol. The command performs all necessary type conversions and validation to compute the resulting data area contents.

<u>The DACL Graphics Facility:</u> The Graphics facility is implemented on two levels. The most primitive level supports plotting of a single data histogram to a low performance, monochrome terminal such as a VT240. This graphics system is completely command driven. The user is first required to execute the DISPLAY/ALLOCATE command to allocate a plot ***display parameters table*** (DPT) in the DACL Heap. Included on the DISPLAY/ALLOCATE command line are qualifiers which provide plot configuration information such as axis style, format (linear or logarithmic) and ruling, histogram name, marker style, *etc.* This information is inserted into the DPT where it resides until needed to configure a plot.

Additional DISPLAY syntaxes are provided to modify an existing DPT (/MODIFY), to delete a DPT (/DELETE), to list the plot characteristics contained in all or in a specific DPT (/LIST), to reset the DPT control structure (/NEW), and to save the contents of a DPT in a disk file (/SAVE).

The PLOT command accesses the DPT which is specified in the command parameter and creates a plot on the default output device (the user's terminal) according to the characteristics recorded in the DPT. Because the plotting software is built upon the CGS graphics system plotting is very slow and user interaction is limited to simple zooms and cursor locator operations. Hard copy can be obtained through screen dumps to a local LA50 or LN03 printer. There is no metafile capability.

The second level supported by the DACL Graphics facility consists of a workstation-based, intermediate performance, interactive color graphics system. This system is initiated by invoking the VIEW command. This command creates color plots of user-specified histograms on the VR290 monitor of the VAXstation/GPX workstation. Unlike the PLOT command which requires a predefined DPT to specify the name of the histogram and the plot characteristics, the VIEW command permits the user to specify dynamically all plot characteristics as well as the histograms to be plotted.

After the VIEW command is invoked, essentially all interaction with the plot is performed through the workstation mouse (occasional keyboard interaction is required to specify an axis label or a histogram name). The plot is configured with numerous Macintosh-like features to allow the user to adjust how the plot is drawn on the workstation screen. Scroll bars and handles are provided on each axis to allow the user to select axis scales and regions of interest in a very dynamic fashion. A hierarchy of pop-up menus is provided to allow other more complex plot layout, control and selection operations.

Currently, one to six histograms may be plotted on the same axis set. A menu-driven selection process permits a user to easily choose which histograms are to be plotted and what their plotting characteristics will be. Multiple line and marker styles and various colors are user-selectable for each histogram. A user may dynamically modify the plotting characteristics of any histogram, or delete entire histograms from the plot through mouse interactions. The combination of menu-driven and mouse selection operations provides a very natural environment with which our users have rapidly become familiar.

As the user interacts with the VIEW plot, a DACL-Heap-resident control structure called the *window parameter table* (WPT) is continuously updated with the current plot characteristics. Upon exiting the VIEW command (through the QUIT menu choice), and unless otherwise specified by the user, the current WPT automatically becomes the default configuration for the next invocation of the VIEW command. The user may also invoke a menu option which automatically saves the current plot configuration (including the names of the histograms being plotted) to a named WPT which can be explicitly referenced by a subsequent invocation of the VIEW command.

VIEW is built upon the ANSI standard Graphics Kernel System (GKS) and inherits a significant amount of its functionality therefrom. Plotting performance is orders of magnitude better than can be achieved with the CGS based system. Dynamic resizing of the plotting window on the workstation screen is possible. High level routines are provided to support mouse-driven user interaction, and a graphics metafile facility is available. Currently hardcopy is produced in grayscale format on local LN03 laser printers. Acquisition of a color hardcopy device to support these systems is under study.

FASTBUS Diagnostics: The DACL system provides a comprehensive suite of FASTBUS diagnostics commands. At the lowest level, DACL commands are provided to perform basic FASTBUS transactions including read and write cycles in single word or block mode as well as arbitration and address cycles.

In addition to the low level DACL FASTBUS commands, the system supports a comprehensive suite of diagnostic commands for each locally-developed FASTBUS hardware module. These commands provide a suite of tests which allow each module to be thoroughly exercised prior to entering the production data acquisition environment. The diagnostic commands are partitioned in a manner which allows the complete test suite to be executed for a given module, or selective testing to be performed. As such, the commands are extremely useful for validation of new modules as well as for isolating specific problems with hardware which is already in service.

Summary

A single prototype data acquisition system was first installed and used in December, 1985 during the final days of PSR commissioning. Both hardware and software worked well although the software consisted of the minimum required for data collection. Only a few bugs were identified by the users owing to the thoughtful engineering and rigorous testing. Once the prototype system was returned to the developers, additional testing revealed as yet undiscovered errors. In the summer of 1986, four additional systems were placed in service with all known errors removed.

The hardware performance exceeds that specified in the requirements. With the deadtime detection operating below 10 ns, deadtime corrections for worst case conditions are less than 0.02% for single input channel rates of 1.7 Mhz for peak intervals of 10 µsec assuming Poisson statistics, more than an order of magnitude better than the requirements. The system operates with count rates averaging more than 2 MHz, a factor of two better than the specification required.

While the software implementation as yet does not fully meet all the requirements, due to limited staffing resources, it does exceed it in some areas. For example, the impact of workstation systems significantly alters the graphics interface from that specified. We are far more satisfied with the functionality and ease-of-use for our menu driven graphics interface than could have been possible with the conventional command driven approach.

To date the LANSCE data acquisition systems appear reliable and robust. We are satisfied that the features designed into the systems will be invaluable as these systems now enter their maintenance phase.

Acknowledgements

This work was performed under the auspices of the U.S. Department of Energy.

References

1. M. W. Johnson, J. A. Goldstone, and A. D. Taylor, "Requirements of Data Acquisition and Analysis for Condensed Matter Studies at the Weapons Neuton Research/Proton Storage Ring Facility," Los Alamos Report No. LA-9099-MS, November, 1982.

2. D. McMillan, R. Nelson, R. Poore, M. Meier, and R. Woods, "P-9 Proposed Weapons Neutron Research Facility Data Acquisition System for the PSR Era," Los Alamos unpublished report, December, 1982.

3. R. O. Nelson, D. M. Barrus, G. Cort, J. A. Goldstone, D. E. McMillan, L. B. Miller, R. V. Poore and D. R. Machen, "Configuration for the WNR Data Acquisition System for Neuton Measurements," IEEE Trans. Nuclear Science, vol. NS-32, pp. 1422-1425.

4. G. Cort, "The Los Alamos Hybrid Environment: An Integrated Development/Configuration Management System," in Proceedings of the Conference on Software Tools, 1985, pp. 10 17.

5. G. Cort, J. A. Goldstone, R. O. Nelson, R. V. Poore, L Miller, and D. M. Barrus, "A Development Methodology for Scientific Software," IEEE Trans. Nuclear Science, vol. NS 32, pp. 1439-1443, August 1985.

6. G. Cort and R. O. Nelson, "The Los Alamos Tool Oriented Software Development System," in Proceedings of the Digital Equipment Computer Users Society, Fall 1985, pp.395 402.

7. R. V. Poore, D. M. Barrus, G. Cort, J. A. Goldstone, L Miller, and R. O. Nelson, "A Data Acquisition Command

Interface Using VAX/VMS DCL," IEEE Trans. Nuclear Science, vol. NS-32, pp. 1290-1293.

8. P. Bruce and S. M. Pederson, The Software Development Project: Planning and Management, New York: John Wiley and Sons, 1982.

9. R. S. Pressman, Software Engineering: A Practitioner's Approach. New York: McGraw-Hill Book Company, 1982

10. E Yourdan and L. L. Constantine, Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design. Englewood Cliffs, N.J: Prentice-Hall, Inc., 1979.

11. E. H. Bersoff, V. D. Henderson, and S. G. Siegal, Software Configuration Management: An Investment in Product Integrity. Englewood Cliffs, N.J.: Prentice-Hall, Inc., 1980.

12. G. Cort, J. A. Goldstone, R. O. Nelson, R. V. Poore, L. Miller, and D. M. Barrus, "Dynamic Data Structures and Concurrency in a Real-Time Data Acquisition System," IEEE Trans. Nuclear Science, vol. NS-32, pp. 1279-1285.

13. Ada is a registered trademark of the U. S. Government, Ada Joint Programs Office.

14. N. Wirth, Algorithms + Data Structures = Programs. Englewood Cliffs, N.J.: Prentice-Hall, Inc., 1976.