EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH

CERN - SL DIVISION

EVOLUTION IN CONTROLS METHODS FOR THE SPS POWER CONVERTERS

A.H. Dinius, J.G. Pett - CERN, Geneva, Switzerland

J.C.L. Brazier - Brazier Systems and Consultants

Abstract

In common with much accelerator specific material, there is a constant need to improve both hardware and software for power converter control. Maintenance and performance improvements of older systems have become extremely tedious and in some areas impossible. By using modern real-time software and the latest high-performance processors, such problems should be substantially reduced.

This paper describes the software concepts and the hardware chosen for the upgrade of the existing facilities. Using the UNIX compatible LynxOS real time kernel, running on a PowerPC 603 in a VME environment, this new approach provides excellent performance while retaining the desired flexibility for future enhancements.

The 64 channel system is implemented as a set of cooperating processes, several of which are multi-threaded. Processes include analogue function generation, analogue measurement and digital I/O, all of which are accurately scheduled by the accelerator timing system. This generalised structure, which performs complex sequences of operations is described in detail, as well as how it can be adapted to a wide variety of accelerator tasks.

1. INTRODUCTION

The controls methods for the SPS power converters have continuously evolved since 1976, due to the changing requirements of the SPS machine and also due to the constant progress in controls hardware and software. Many other machine systems have had similar need to be updated. The present controls for the power converters are based on a VME hardware solution which was conceived in the mid-eighties[1]. This design allows up to 64 power converters to be controlled from one VME-based chassis and was designed using the then latest hardware and software tools. Unfortunately, whilst the present system provides adequate service for the moment, an upgrade, of the software in particular, was deemed necessary such that maintenance could be assured for the foreseeable future.

It was concluded therefore that the best overall approach would be to design a general software solution which would be able to emulate the present configuration, whilst allowing plenty of scope for future expansion and easy replacement of hardware modules. To support such a software approach, a powerful VME crate controller would be necessary to alleviate the time critical code problems. At the same time it was clear that some of the existing hardware modules needed to be replaced such that system expansion and improved analogue acquisition capabilities could be implemented.

One of the major constraints to such an upgrade is the need to remain operational and to introduce the new control equipment progressively. The huge base of existing applications software demanded that functionally, the replacement of controls material had to be essentially transparent to the users.

2. DESIGN GOALS

Given that the existing system (*original Mugef*) was both difficult to modify and that certain parts were aging, the upgraded system (*ROCS based Mugef*) was designed with the following goals in mind:

- The system should be written in a high level language and run on top of a modern operating system.
- The system should allow easy additions and modifications.
- Good diagnostic and monitoring should be built-in for both system debugging and usage monitoring.
- The performance of the system should initially be able to emulate the previous Mugef System.

The first two goals were inspired by the situation with the current system which was monolithic and written in Modula2 and assembler. Whilst this produced a very efficient system, it was not *technology proof*. Careful evaluation of the hardware revealed difficulties with replacement parts and a need to improve performance. At the time that this project was conceived, CERN was starting to standardise on LynxOS[i], a real-time version of the popular Unix[ii] operating system and as we had considerable experience of writing Unix system programs, this seemed the logical choice. Moreover, if we restricted our use of the systems to those parts that were POSIX compliant and wrote the code in ANSI C, then it should make the porting of the code to another POSIX based system trivial. At this time, LynxOS (version 2.1) complied with the POSIX.1 standard and implemented draft 9 of the POSIX.4 Real Time Extensions and Draft 4 of the threads extension, POSIX.4a.

It is more natural to design Unix based solutions as a set of co-operating processes rather than a large monolithic program. The processes are connected by Inter Process Communications channels, of which POSIX provides a rich set. This approach has the advantage that the interfaces are clean and that whole processes can be exchanged without

[i] LynxOS is a trademark of Lynx Real Time Systems Inc.

[ii] Unix is a trademark of X/Open Ltd.

affecting the rest of the system. It was foreseen that the type of changes that would be required during the system's life would be:

- Subsystem change to accommodate new hardware such as a new Data Acquisition module.
- Additional functionality provided by an existing subsystem.
- A new user interface or command set.
- New methods of function description to better model the required control of the power converters.

From the outset, the intention was to provide a framework to enable such modifications to be carried out by several members of the group. The system needs to be structured so that, within reason, changes can be made without an intimate knowledge of the entire system or the programming techniques used (eg. multi-threading).

With regard to performance, it is a generally true that for a given system, a special purpose solution will always out perform a general purpose one. With any real time control system, performance is normally the primary criteria and everything is sacrificed for it. Whilst this may seem laudable at the time, some of these performance shortcuts can haunt the system in later life. Our initial development CPU was a Themis TSVME133 board with a 20MHz 68030 processor. It was always known that new, more powerful, processor boards were on their way, the current one being a PowerPC 603 board. This has been estimated to give an improvement of 15 times over the Themis board[2]. It was felt that this would compensate for the heavier, but more flexible design envisaged.

Diagnostic capability is required during the commissioning phase to quickly isolate problems and also during operation to monitor the health and activity of the system. Some novel ideas have been incorporated into the software framework to enable this to be achieved.

The following sections outline the areas where hardware upgrades have been introduced, while concentrating on the overall software approach.

3. THE HARDWARE ENVIRONMENT

The diagram Figure 1 below, shows the original and Rocs type Mugef hardware structures. From the original system, only three boards are retained namely, the Static Ram (TSVME 206), Status & Control (Statophone 1.0) and the Function Generators (Newave 2.0). The crate, wiring and power supplies are also retained.

The crate has a total height of 9U, the top 3U are used for the various power supplies, the lower 6U are used for the VME boards. A 20 position VME bus is used for the P1 while for the P2 three different dedicated buses are installed. This makes the different types of boards "slot dependent". The lower 6U of the back panel consist of one large multi-layer PCB with press-fit connectors for all I/O and internal wiring. Due to extensive use of flat cable techniques, important savings on cabling costs and reductions of errors during assembly could be achieved. Below is a short description of each type of board which will be used in the new system.

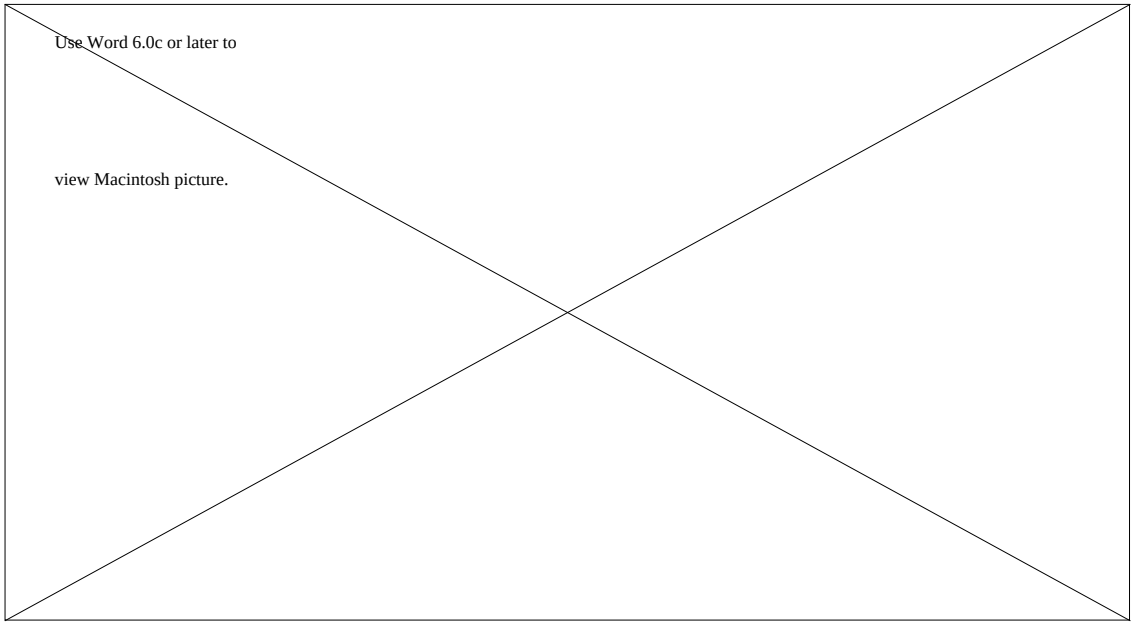

Figure 1 - Hardware Structures

3.1 The General Services

The SAC[3] is a VME System Arbiter Controller board and provides the system with several Controls Group's functions which are useful for creating a standardised approach for a VME crate on the Ethernet. Amongst other functions the board memorises the Ethernet address, this allows a simplified exchange of the VME bus master CPU. The board also drives the VME sysreset bus signal from a front panel input, this allows an external, remotely initiated reboot of the system.

3.2 The CPU

The RIO2[4] is a PowerPC 603 VME based microprocessor board and has an on board Ethernet Interface. This board has recently been standardised by our Controls Group. This CPU replaces the 68000-10 MHz. HAMAC-1 microprocessor board, which in conjunction with a MIL-1553 VME interface board formerly fulfilled these functions. Having an Ethernet Interface on the CPU, the need for the MIL-1553 field bus and the Process Control Assembly (PCA) computer with MIL-1553 Bus Controller (BC), vanishes therefore simplifying the whole control structure for both hardware and software.

3.3 The Static Ram

The TSVME 206[5] is a 2-MB VME based 32-bit static ram board with battery backup. This board stores all the dynamic data for configuring the crate which is, on user request, downloaded on initial start-up of the system. Furthermore, the board stores all the tables for the functions which have to be generated by the function generator boards. The battery backup feature guarantees a return to identical functionality of the crate after a mains power fail.

3.4 The Timing

The TG8[6] timing board, the modernised version of the formerly used TG3 timing board, is a VME based timing receiver board with 1 mS resolution which connects to the SPS Master Timing Network and allows a program to receive notification of timing events. These events are four byte frames which describe all the significant events in the machine, including those which will eventually require new functions to be output. The TG8 holds a list of those events which are of interest and will generate an interrupt when any are received. In addition to software notification of events, the board is also capable of asserting signal lines as well. These are used for direct control of the function generators. In this system, the events of primary interest are those designed to start functions being output to the power converters. These

events are arranged in pairs, viz. a Warning event followed by a Start event. Actually only the Warning event is real and is seen by the TG8 board. The Start Event is generated internally by the TG8 board after a programmed delay and simply activates a line on the function generator's P2 bus.

3.5 The Function Generators

The Newave 2.0[7] is an 8 channel VME based analogue output function generator with 16 bit resolution and an output range of +/-10V. A maximum of 8 Newave 2.0 boards can be used in the crate, giving a total of 64 channels. All analog outputs, synchronisation signals and power supplies for the analog part of the board are connected via a specially designed P2 bus. Though the VME boards in the crate are *slot dependent* any Newave 2.0 board can take any of the 8 slots on the function generator P2 bus by decoding 4 address lines on the P2 bus. Data sets consisting of time (mS) and value (bits) are downloaded and synchronised with a warning event from the timing system to the on-board memory. The analogue outputs are generated by DAC's where the on-board microprocessor interpolates between adjacent data points. Synchronisation of all Function Generators is assured by the accelerator timing system. Due to the fact that these boards are based upon now unobtainable hardware, an alternative is under development.

3.6 The Analog Inputs

The MPV915-21[8] is a VME based, high accuracy analog input board with 16 bit resolution and provides 32 channels of differential input with a range of +/-10V. A maximum of 4 MPV915-21 boards can be used in the crate thus giving a total of 128 channels. All analog inputs, timing signals and power supplies for the galvanically separated analog part of the board come in via a special designed P2 bus. This board is also capable of being used in any of the 4 slots on the analog acquisition P2 bus by means of decoding address lines on the P2 bus. It replaces a system built using a controller board type MPV940 and two 64 channel differential input modules, type MPV941. This complete combination of boards was necessary even to read a single power converter channel and only provided a maximum of 12bits of resolution. Multi-point measurements could only be done for one pair of inputs at a time and took a least a few elementary cycles to produce the data. With the new system, measurements are performed continuously every millisecond, on all channels. This data is stored in the on-board memory and is available immediately on request. Synchronisation of readings is again assured by the accelerator timing system.

3.7 The Status and Control

The Statophone 1.0 is a VME based I/O board capable of controlling 64 power converters. A total of 2x16 status bits and 8 command bits are available for each power converter. Each power converter has its individual interface box arranged in a *daisy chain*. Status and command enable signals are output for selecting the required power converter on the daisy chain. Software tables map any of the 16 pre-defined power converter families into the hardware environment and thus create a very flexible approach for controlling many different types of power converter.

4. THE SOFTWARE DESIGN

The software that makes up the system becomes crucially important and was designed with the aims stated earlier. With these in mind, it was therefore necessary to split the complete software into a number of interconnected modules.

We were also interested in keeping the software as portable as possible. As the final system would consist of a number of processes connected together with Inter Process Communication Links (IPC), all these would be chosen from the POSIX set provided by LynxOS. This would cover 80% of the portability as the individual processes would be very constrained in their nature. Several of these processes would be multi threaded since the technique offers very elegant solutions.

The system layout naturally follows on from the decisions made above. Each of the hardware components, (function generation, acquisition, status & control and timing) would have a separate process to control and manage it. There remains the interface to the *outside world* (which we will term the User Interface) with its command emulation software and the sequencing of all the actions. It should be borne in mind that the final result is to load function data into the appropriate output channel at the correct time. This is to be scheduled concurrently with a number of other tasks on different channels. Since this is a generalised scheduling and sequencing task, it would seem sensible to decouple this from the User Interface by having a dedicated sequencing process. This process, fig.2, will also contain most of the real-time aspects of the system.

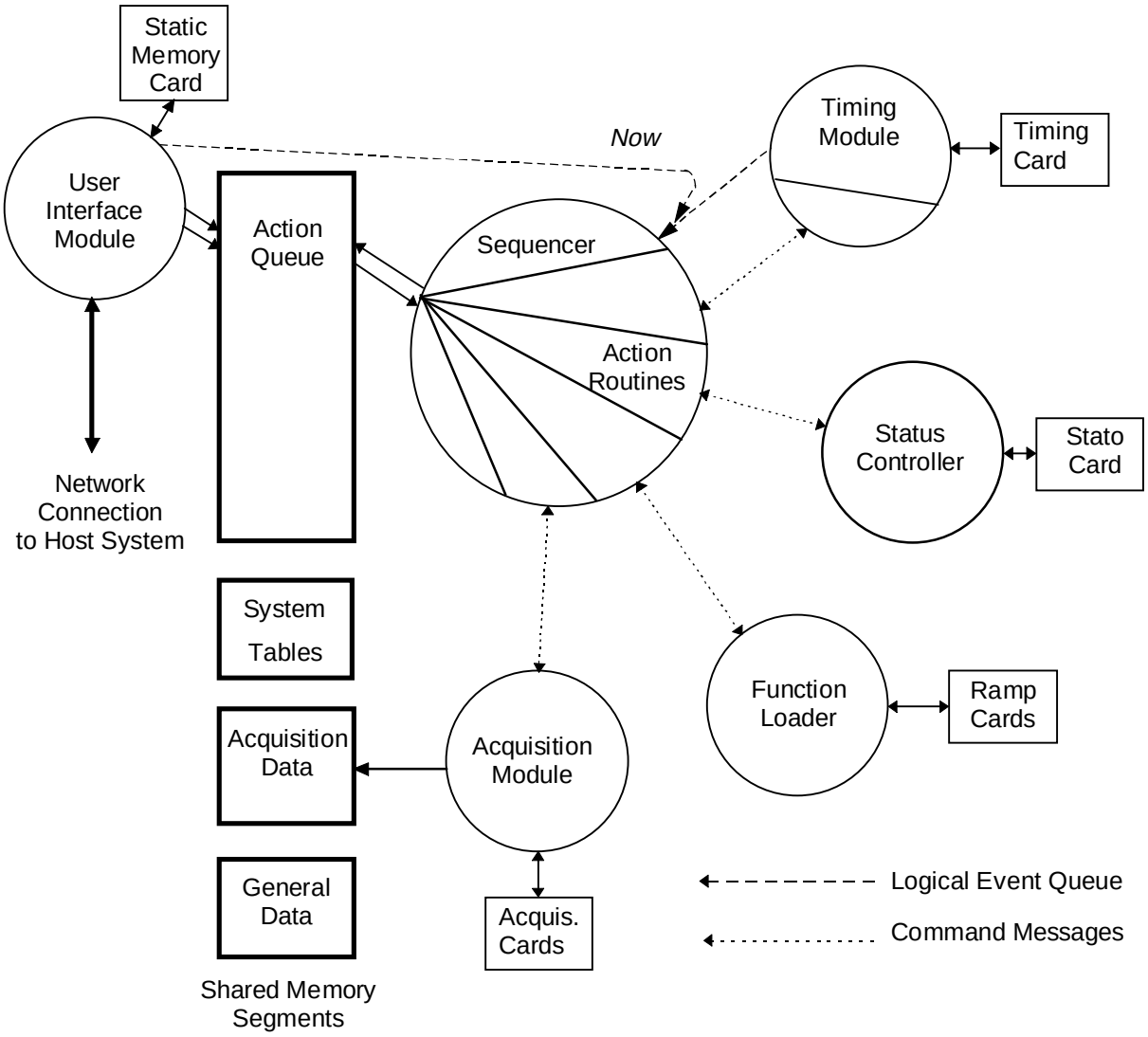

Figure 2 - Block Diagram of the Software

4.1 Device Drivers

Clearly the several attached devices need to be controlled. In a protected mode operating system such as LynxOS this is normally done using device driver code in the kernel. This is the case with the TG3/TG8 which is a standard CERN peripheral. The other boards are both special to this system and have the particular advantage that they are memory mapped in VME address space and do not use interrupts. This means that they can be manipulated in User Space by using a LynxOS system call to map a portion of VME address space (either A16 or A24) into a user program. The device driver code can therefore appear as a library of normal C routines, thus simplifying the debugging greatly. This approach also lends itself to having a process supporting each of the devices, such processes being connected to the rest of the system by Inter Process Communication Links. The same method is used to attach a 2Mb Static memory module to the address space of the User Interface Software.

4.2 The Sequencer and Action Tasks

The heart of the system is the sequencer. This is a multi-threaded program which, on an event, examines a queue of actions and for each of the actions that need to be done now, spawns a thread to carry out the action. Using the thread priorities provided by LynxOS, the master thread which scans the queue runs at a higher priority than any of the action threads that will be spawned. The master thread will therefore make a complete, uninterrupted pass through the queue and will then go back to waiting for the next event. As soon as it goes to sleep, all the action events will start at once and be scheduled according to their given priorities. To a first approximation, (ie. leaving out the effects of varying queue length) this will exhibit deterministic performance.

The action queue is a list of action structures in a shared memory segment and the inbound events emanate from a system message queue. Each action has an associated *action routine* which contains the required functionality. These routines are tied into the rest of the process by a table of (action type, action routine, priority) records.

There are three types of action, namely:

- One shot - where the action is performed once and then discarded
- Repetitive - where the action is left on the queue to be repeated when the next event of this type happens
- Indirect - where this action is simply a *wrapper* for another action (complete with its own event details). The action routine in this case will simply unwrap the *sub-action* and place it on the queue in its own right.

By giving the action routine the responsibility for removing the action structure from the queue, it is a simple matter to implement the above action types. Bearing in mind the extensibility requirements of the system, the structure has been designed such that the action routines need know nothing about the sequencer mechanism. Each routine is a simple subroutine called with a pointer to the action data. After the routine has finished, it simply returns and the thread exits. The action routine contains all the functionality of the user command. This will involve some data processing and probably communication with one or more of the sub-systems. The communication is done with messages sent to already open channels.

One of the actions, dealing with function conversion from a *user format* into the required and fixed, *hardware format*, deserves special mention and is shown in Figure 3. One of the user formats has the functions supplied as data sets of straight line segments describing a curve. The action routine is the place to carry out this conversion, at a low priority since it is essentially a background job. It would be wrong however to embed such an essentially mathematical piece of program in the rest of the system code since changes to it would be likely to be carried out independently. It was therefore concluded that this code should stand alone as a separate program.

This conversion program is run from the action thread *forking* a child process. The child has its standard input and output reconnected to the action data and a message buffer respectively. The conversion program is then run with an *exec()* system call. To minimise the speed penalty, the conversion programs are stored in a part of the file system implemented in RAM disk.

The advantage of this seemingly complex exercise, is that the conversion programs can be developed away from the system, for example on a PC under DOS, without any knowledge of, or contact with, the ROCS infrastructure. The finished conversion routine is incorporated simply by compiling on the target machine and placing the executable in the correct directory.

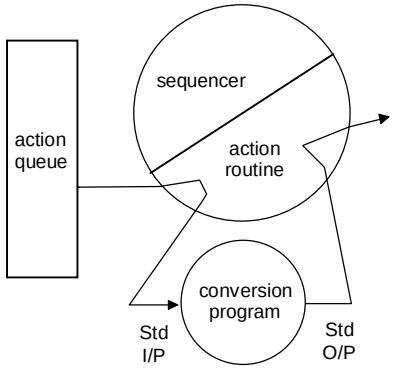

Figure 3 - Conversion Programs

4.3 The User Interface Module

The User Interface module is written to emulate a command set, currently that of the existing Mugef system. It is connected to the control room via an RPC based *Equip* call[9]. The commands are decoded and passed to a command processor. Most often, this processing simply consists of parameter checking and packaging up into an Action Structure, tagged with the special event *Now*. This structure is linked on to a chain of such actions in the action queue. The action is triggered by placing a *Now* event in the Logical Event Queue. The action is synchronised by the User Interface module waiting for an acknowledgment message to be placed in a special message queue by the Action Routine. This message both flags the completion of the action and passes back any error information.

The original Mugef system maintained many hardware/software configuration tables which need to be loaded at system startup and describe the system and the connected power converters. These tables are now combined into one master table which is available in a shared memory segment to any process which requires them.

4.4 The Timing Module

The power converter control system is event driven as are most things on the accelerator. As has been previously described, there is an event timing system on the accelerator and a TG3/8 board to receive these events. Rather than have the sequencer handle the timing, it was decided to decouple the generation of events with the sequencing of actions. Thus there is a separate module which receives events from the hardware and generates a stream of fast messages which are logical events.

By doing this, the option is there for incorporating more sophisticated events like Real time, Hardware Status Lines, User Events and indeed, combinations of all of these. To enable these extra facilities, the Timing module is multi- threaded having a master thread that responds to a command stream with extra threads for each of the sources. Currently there is one extra thread which constantly waits for timing events from the TG3/8 and generates messages. The command thread is responding to requests from the action routines for current Super Cycle numbers, times of events etc.

4.5 The Status Controller

The attached power converters need additional control functions (eg. on, off, reset, etc.) and checking for status. This functionality is carried out by a *Statophone 1.0* board. The board is quite simple and so is the controlling program. Switching on a series of large power converters can have many consequential effects, therefore the board is given a batch of power converters to control with one instruction. The only complexity comes in determining which bits in an 8 bit command register correspond to ON, OFF and RESET for a given power converter (which come in many different *flavours*). This mapping is carried out by a series of tables in the user interface.

4.6 The Function Loader

The function loader module, in conjunction with the Newave 2.0 function generators, is responsible for outputting the correct data sets to the correct channels at the correct times. The Newave 2.0 board has inbuilt intelligence which will handle the actual generation of the functions, thus leaving the housekeeping to the controlling program. The program initialises the hardware and then waits for messages, (eg. LoadFunction, EnableFunction or StartFunction).

The LoadFunction command sends the already converted data to the Function Loader. This will then wait in the process until it is enabled, some time later. The EnableFunction command is merely a change of state and means “load to hardware on the next start event”. The StartFunction command causes the data to be loaded into the Newave 2.0 board, which will produce the analogue output. With the function on board, subsequent StartFunction commands (which will come once every Super Cycle), simply require manipulation of the pointers.

4.7 The Data Acquisition Module

The acquisition hardware runs autonomously with regard to sampling and since there is no local processing power on the boards, they rely on the controlling program for setup, calibration etc. Every board has two memory banks each of which will hold a Super Cycle’s worth of data. These banks are being filled continuously and swapped on each Start Super Cycle event.

Sampling is carried out relative to an event, the time of which is retrieved from the timing module. The system is fully flexible with respect to extracting analogue measurements from any channel at any time. To save excessive copying of the data, a separate Operating System shared memory segment is used to hold the acquired data. Some processing is done in the Acquisition module and more is done in the action routine that demanded the data. Finally, the data is copied back to the user directly from the shared memory.

4.8 Overall Control

The system is started by a single, table driven control program. Each line of the control file has the form (program, priority, I/O redirect, command line). Every error exit of every module returns a different number which is logged to a system log. If a process exits with an error, all the other processes are shut down for safety. The error numbers are

graded according to fatality, *soft errors* will cause the system to restart whilst hard errors will not. If the system crashes again within a certain time, the restart will be inhibited to prevent thrashing.

5. DIAGNOSTICS TECHNIQUES

Diagnostics techniques and maintainability have been key points throughout this development. Many things are easier with a system such as this than would be the case with a more monolithic approach. This is helped to a huge extent by running the system on top of a general purpose operating system. Just being able to *telnet* on to the running system is of inestimable value. The *test points* in such a system lie on the interfaces of the components which are very visible. For example, the sequencer operation can be studied by looking at the actions present in the action queue. This is extremely easy to do with a small program that just maps the action queue in and prints the list of structures. Similarly the other shared memory segments containing other data can be spied upon. Another program is available for printing the current system tables.

As mentioned in the section on the User Interface, all the system tables plus the current set of functions are stored on a battery backed, static memory board. On the same board, there is a circular buffer containing the details and time stamps of the last 1000 commands that had been executed. This has proved extraordinarily useful for the following reasons:

- The board is a self contained static snapshot of the system. On a serious hardware failure, a new crate can be installed and the system restarted simply by swapping this memory board.
- All the system tables can be examined by a separate program but run on the same crate, which maps in the memory board and prints the data.
- A usage profile can be built up by running another separate program which will examine the command buffer. From this it is possible to see patterns of usage by, for example, an alarm surveillance program.
- All this data can be simply dumped into a file and taken away for off-line analysis

A novel built-in diagnostic mechanism is currently being added to the system for which we have great hopes. Each of the processes has a diagnostic thread running in the background. The thread is mostly asleep at a priority less than the main thread. At regular intervals it wakes up, makes some diagnostic tests (reading status registers etc.) and writes them into a private area of the “General Shared Data Segment”. At the very least, the thread will write an “I am Alive” message.

Other diagnostic monitoring programs can then look at the shared memory without interfering with the running system. As a point of interest there is no semaphore controlling access to this region. It would be disastrous if an errant monitoring program were to leave the semaphore locked and block the diagnostic thread as a consequence. This also lends itself to remote monitoring using standard SNMP techniques.

6. EXPERIENCE

The system has been operating continuously for five months in a beta test site controlling ten orbit corrector dipole magnet supplies in the SPS accelerator. The system is currently running on a Themis TSVME133 which uses a 68030 processor. We are currently in the process of moving to the final PowerPC implementation which we expect to exhibit greater throughput. As a consequence of the way the system has been designed, it is possible to move to the new processor by only changing two address mapping constants. After that the port is a simple re-compile.

The auto restarting control program has proved valuable. We simply monitor the log file on a regular basis. Occasionally, there is a software crash but within a minute the system has recovered automatically and the operators are usually unaware of anything going wrong.

The diagnostics are proving very useful and other members of the group are already exploiting the features of the new system.

7. OVERALL CONCLUSIONS

The above outline of a *upgraded* system for the SPS power converter control system has shown how a modern software approach can be implemented in an existing environment while providing the desired improvements to both performance and long term usage. The test system currently in the accelerator has demonstrated that the initial design is compatible with the existing application programs and has performed adequately. Future improvements can now be integrated into the software with minimal difficulty and fully transparent to the user. The changes should now ensure that the overall system remains viable for the foreseeable future.

The software design could be considered to be of general interest to many similar control problems both for other accelerators as well as commercial applications. In fact any event sequenced application could profit from a similar structure. Much of the hardware used is of a sufficiently general nature to be used elswere as well.

8. ACKNOWLEDGMENTS

We are indebted to the contributions made by our colleagues, in particular Philippe Semanaz and Pierre Martinod whose knowledge of the way the original Mugef system works is incalculable, and Philippe Malacarne who is testing the new system. Also to the SL Controls Group and Alastair Bland in particular for their support of a particularly demanding customer.

9. REFERENCES:

[1] - A VME bus approach for the Control of the Closed Orbit Correction Power Supplies within the SPS Super Cycle
P.Martinod, G.Mugnai, J.Savioz, P.Semanaz, SPS/ABM/Note/85-18

[2] - CERN PowerPC benchmark tests. (private communications)

[3] - SAC Version 2, VME System Arbiter Controller, SL/CO/Note/93-70, CERN

[4] - RIO2 preliminary information, CES, Geneva, Switzerland

[5] - “TSVME 206 User’s Manual, 32 bit static RAM board with backup”, Themis, Gieres, France

[6] - “A new VME Timing Module: TG8, G. Beetham, G. Deams, J. Lewis, B. Puccio, CERN

[7] - Newave 2.0 Technical Reference, P. Semanaz SL/PC/CS, CERN

[8] - Product Manual MPV915-21 High Accuracy Analog Input Board, Pentland Systems Ltd., Livingston, Scotland

[9] - Accessing Equipment in the SPS/LEP Controls Infrastructure: The “SL-EQUIP Package” - P. Charrue
SL/Note 93-86(CO) - September 1993.