RECENT DEVELOPMENTS ON THE CONTROL SYSTEM OF THE SOUTH AFRICAN NAC ACCELERATOR

G.F. Burdzik, I. Cloete, I.H. Kohler, J.N.J. Truter, K. Visser and H.F. Weehuizen
National Accelerator Centre, CSIR, P.O. Box 72, FAURE, REPUBLIC OF SOUTH AFRICA

Summary

The control system of the NAC accelerator is being designed around several mini-computers which will handle the centralized aspects of the control system, a CAMAC Fisher System Crate network and a variety of microprocessors into which are being distributed tasks of a single-task or dedicated nature. In this paper we discuss the recent developments on our control system such as the development of CAMAC driver software, the sharing by several concurrent tasks in the mini-computers of a common memory area, the development of memory hardware which can be shared by up to four mini-computers, and the development and application areas of our microprocessor systems. We also discuss what we feel to be the shortcomings of the CAMAC System Crate network, and are investigating the possibility of using Ethernet as an alternative to the former.

Introduction

In 1979, when the planning of a control system for the NAC accelerator started, we made several design decisions which were based not only on technical considerations but also on prevailing local conditions such as the local availability of expertise and equipment, and our limited manpower situation. These decisions were:

i) to select for centralized control purposes a mini-computer brand for which good technical support was available. Our eventual choice placed us outside the mainstream from the point of view of mini-computer makes commonly used in accelerator control systems;

ii) to make use of commercially-available interfacing networks rather than develop our own. At the time CAMAC was the only system which was sufficiently well-developed to satisfy our requirements. We chose the particular CAMAC system known at that time as the 'GEC-Elliott Executive Suite' system, now known as the 'Fisher System Crate' system. This system allows one to interface more than one computer, by way of a 'system' crate, to a CAMAC network of up to seven parallel branches and/or serial loops in such a way that every computer has access to the entire network;

iii) to distribute tasks of a single-task or dedicated nature amongst a variety of microprocessors; and

iv) to make as much use as possible of vendor-supplied and/or commercially-available systems and high-level languages for the design of the control system software. In the case of the mini-computers we use the vendor-supplied operating system and a vendor-supplied implementation of the Pascal language; in the case of the microprocessors we develop software in Pascal MT+ and run under CP/M.

The control system has evolved from our early design concepts as a result of our better understanding of the problem and the hardware, and the emergence of new technologies and new products. Recent developments on the control system will be discussed under the following headings:

Development of CAMAC driver software
Sharing of a data store in mini-computer memory
A shared hardware memory for the mini-computers
Microprocessor developments
Microprocessor applications
The interfacing network

Development of CAMAC Driver Software

No usable CAMAC driver software (for the combination of mini-computer and CAMAC network we were using) existed at the time we acquired the hardware. A considerable programming effort therefore went into the development of optimized driver routines. Two distinct methods of generating CAMAC commands were implemented, namely the privileged subroutine method and the operating system driver approach.

The privileged subroutine method provides the user with the fastest way of executing a CAMAC command. In this approach the low-level routine with which all CAMAC commands are implemented (the 'privileged' routine), is loaded in the same memory partition as the task needing to access CAMAC. Normally such a subroutine would not be able directly to read from or write to the mini-computer I/O ports without causing a 'memory protect' interrupt. The privileged subroutine avoids this problem by switching off the interrupt system for the duration of the time needed to access the I/O ports to which the CAMAC network is connected. Typical execution times for single CAMAC commands are approximately 320 μs, and CAMAC blockmode transfers under program control take as little as 25 μs per CAMAC command. A large fraction of the single command execution time is due to the use of system subroutines for switching the interrupt system on and off: the implementation of these routines in microcode should reduce the single-command execution time by approximately 50%.

The operating system driver approach is more suited to the handling of demand interrupts which arise from LAM's (Look-At-Me's) in the CAMAC network crates. This is an implementation of a routine which resides in the system driver area of memory and is capable of servicing interrupts from a variety of sources in a CAMAC system. It makes use of a table which contains information for the scheduling of LAM-servicing tasks. The CAMAC driver is also capable of carrying out conventional CNAF commands, but such commands take approximately four to five times as long to execute as do privileged subroutine commands.

When a LAM causes an interrupt in one of the mini-computers the interrupt is routed via the interrupt trap cells of the mini-computer directly to the CAMAC driver, a process which is very fast. The delay between the setting of a LAM and the identification of the LAM source by the driver is only 80 μs. However,

CH1996-3/84/0000-0385 $1.00 © 1984 IEEE

the identification of which LAM service routine is to be used takes rather longer, since its position in the table is calculated from the branch, crate and station information corresponding to the LAM source. Consequently approximately 2 ms elapse before the LAM service routine is identified. If the conventional operating system method of scheduling the LAM-servicing task is followed, there are further delays due to the fact that the operating system Dispatcher (task scheduler) only runs at clock-determined intervals of 10 ms. The resulting delay between the setting of a LAM and the servicing thereof can vary between about 3 and 13 ms with an average of 10 ms. The problem of the delay introduced by the Dispatcher has recently been circumvented by associating the LAM-servicing task with a spare I/O channel and by inducing a software interrupt for that channel from the driver routine: the LAM-servicing task is now scheduled by an interrupt rather than by the system clock; it runs immediately after the setting of the interrupt by the driver, and the mini-computer is able to service up to approximately 300 LAM's per second.

Sharing of a Data Store in Mini-Computer Memory

The cyclotrons and other accelerator equipment are controlled from a number of control consoles linked via CAMAC to the mini-computers. At such a console an operator may call up for display any one of 110 possible 'pages' of accelerator variables, each 'page' containing up to 80 variables. Conceptually, our organization of variables into pages and the manner in which the pages are called up for display is very similar to that of the IUCF control system[1]. When we started planning the software for driving the consoles, we defined the following basic requirements:

1) Response times of console-servicing routines should be reasonably quick, e.g. it should take less than a second to display a page after an operator request to do so;

2) It should be possible to add new accelerator variables and pages to the control system on a routine basis;

3) It should also be possible to add new accelerator variables to an existing page or to move a variable from one page to another on a routine basis.

Requirements (2) and (3) dictate that pages cannot be stored in their display-image form, but that the display for a particular page needs to be built during run-time from some or other page format specification and a list of accelerator variables on that page. After it had been established which data were needed for the console processes, and these data had been reduced to the third normal form (to avoid duplication and update anomalies), a number of implementation options were investigated. All options for which the pages and variable data reside on disk were found to have response times far in excess of what was required: only those options for which the data are held in memory had acceptable response times. These results then led us to formulate some further requirements:

4) all programs running concurrently and requiring access to pages or variables data must have access to a common data store area;

5) The data store in memory must be large (250 - 500k words);

6) Access to the data store should be possible directly from programs written in Pascal;

7) It should be possible to change the sizes of the data structures and the order of records in memory without affecting the way in which the software for sharing memory is implemented;

8) It should in future be possible to share the data store amongst a number of mini-computers.

Because of the limited (15 bit) addressing range of the mini-computer instructions, programs are required to execute in an address space of no more than 32k words. The vendor-supplied operating system provides an additional so-called Extended Memory Area (EMA) facility which allows a program to have a data area as large as the physical memory available. However, the operating system does not allow concurrently-running programs in different partitions to share the same EMA. Methods have been devised by a number of workers to get programs to share the same EMA, but the methods suffer from the following drawbacks:

i) The positions of data in an EMA block are determined at compile time when the EMA variables are declared. This is in direct conflict with requirement (7) above;

ii) Programs sharing EMA must be locked into memory which severely restricts the number of programs which can use shared memory.

The memory-sharing requirements and the limitations mentioned above have led us to utilize a method of memory access in which the shared memory is not declared to the operating system. In our mini-computers physical memory is divided into a section occupied by the operating system itself (starting at address 0), and a number of user partitions in which the user programs execute and the sizes of which (up to a maximum of 32k words) can be specified by the user. The physical memory available to the operating system is declared when the operating system is generated or when partitions are reconfigured. However, not all physical memory need be declared to the operating system, and any memory not declared will not be used by the operating system. This hidden memory, consisting of all memory larger than the maximum declared, is used as the common data store area, and is accessed by the user programs in the different partitions by the 'mapping' of the common memory area into the user partitions. The mapping is accomplished by the use of the 'map registers' of the Dynamic Mapping System (DMS) of our mini-computer and dynamic variable pointers of Pascal. Any user program has available 32 DMS map registers for addressing up to 32 one-kiloword pages of non-contiguous physical memory which constitute the 32k words of logical address space in which it may execute. If a program occupies 31k words or less, one or more unused map registers are available which can be changed to point to a common data area to be shared by programs in different partitions. Large data areas can be accessed in 1k word chunks in this way. Details of the method are given in reference[2]. Response times for the display of pages based on this method vary between 0,5 and 1 s: the improvement in response time over that of the other methods investigated is due to the fact that the data are not moved before they are used.

A Shared Hardware Memory for the Mini-Computers

A memory which can be shared by up to four of our mini-computers has been developed to the proto-type stage and is now being tested. It connects to the memory bus of each of the computers and therefore appears to each of them as part of each one's own memory. It consists of two sections. The first section

CH1996-3/84/0000-0386 $1.00 © 1984 IEEE

2k words in size, is fast memory and allows all four computers simultaneous access within one memory cycle, so that there is no loss of speed. This section will be used to send semaphores and short messages between the computers. The second section, which can be expanded up to 512k words, is about the same speed as the standard mini-computer memory, and will allow only one mini-computer at a time to access it. While a particular computer is accessing the shared memory, the latter generates a signal which inhibits memory cycles in the other computers. Since this signal is one of the standard signals of the memory bus, access to the shared memory is completely transparent to the existing hardware and software.

It is intended that the shared memory be used to hold all the common data which are required by programs running in the several computers of the control system. This shared memory may not be made available to the operating systems of any of the computers, because programs may not execute therein. The shared memory will therefore form part of the memory not declared to the operating systems of the participating computers, as was discussed in the previous section.

Microprocessor Developments

From the outset of the control system design it was apparent that there would be many applications for microprocessors in the control system. However, commercially-available systems tended to be expensive and technical support in South Africa for microprocessors based on standard busses (e.g. the S-100 or Multibus standard) at the time was weak. At the same time there was a proposal within the country to define a local bus standard, viz. the SABUS standard, for the production of cheap microprocessors. Part of the proposed standard was the use of standard-sized p.c. cards which slot in modular fashion into a 3U high cardcage and communicate with each other by connecting to a backplane. We decided to adopt the SABUS standard and to participate in the development thereof.

The early developments of SABUS hardware at the NAC were an 8085 CPU card, a communication and system controller card, an interface card to a commercially-available auxiliary crate controller CAMAC module which meets the ANSI/IEEE 675-1979 standard, and a refinement in the design and layout of the backplane. Other developments have been a variety of input and output cards as well as a card by means of which the signals of the SABUS backplane are brought onto the pins of an 8085 socket, and the SABUS microcomputer can be used to emulate an 8085/Z80 microprocessor. These developments, when taken together with commercially-available SABUS modules such as memory and floppy disk controller cards, form the basis of a fairly powerful, cheap and very reliable microprocessor system. Furthermore, the CP/M 2.2 operating system has been adapted to run on this system, and we use Pascal MT+ as high-level development language. Subroutines for implementing CAMAC CNAF commands have been written and a library of Pascal-implemented programs has been built up. The results is that the SABUS has played the role of a fully-fledged development system in our laboratory.

Recently work has started on the design of a Z80 CPU card with a DMA controller, 48k bytes of EPROM and 16k bytes of memory. With these two cards it is intended to incorporate CP/M 3.0 (with its bank-switching facilities) on the SABUS. Work has also recently started on the design of a bubble memory card which will be used instead of floppy disks in all environments not suited to the using of floppy disks.

Microprocessor Applications

The decision to distribute control tasks by way of local microprocessors proved to be extremely useful, and not only from the point of view of easing the processing burden of the mini-computers. For the development of a control or data-acquisition system for specific equipment, it is far easier to provide a microprocessor linked to a CAMAC crate and the necessary programming tools, than it is to provide equivalent development facilities via the central control system. Furthermore, for the later maintenance of such equipment it is very useful to have a local control facility available into which the necessary maintenance procedures can be built, obviating the need for the central control system having to be available at maintenance time.

The microprocessor applications include control and data-acquisition systems which form part of the control system as well as stand-alone systems. In the case of applications which form part of the control system we have generally tried to follow the policy that

i) The SABUS processor should interface to CAMAC by means of a commercially-available auxiliary crate controller and, where necessary, a mailbox memory;

ii) CAMAC modules should be used to interface the system which needs to be controlled or monitored so that access from the SABUS microprocessor to the system is via CAMAC; and

iii) Any control of the equipment should reside completely in the local processor; control by the central control system should be in the form of commands to the local processor.

The systems mentioned below are representative of the applications which have been or are at present being implemented:

Rf Control System

The Rf Division was given one of the first SABUS microcomputers for the development of the control of the rf system for the injector cyclotron, details of this system being given in another contribution to this conference[3]. The SABUS controls the rf system entirely which includes conditioning of the rf system, tuning of the system at low and high power, the automatic closing of control loops and the monitoring of the necessary rf variables. The central processor communicates with the rf control system in accordance with the policy mentioned above, i.e. instructions about which procedures are to be carried out are sent to the SABUS system together with data such as the rf frequency and voltage amplitude, and the SABUS system then executes the procedures. Similar systems will be built for the rf systems of the separated sector cyclotron and the buncher in the injection beamline.

Data-Acquisition Microcomputers Linked to the Central Control System

Work has recently begun on an SABUS-based data-acquisition system for the emittance measuring equipment. The SABUS will contain the program for the control of the slits and harps of the emittance measurement equipment and will acquire the data, format and block it and store it in a CAMAC mailbox memory. The central control system issues the instructions to the SABUS processor, as to when a measurement should begin and provides it with the parameters of the measurement. When data are

CH1996-3/84/0000-0387 $1.00 © 1984 IEEE

ready to be fetched the SABUS sets a LAM in the mailbox memory signalling the central processor. A similar system is being planned for the differential probes of the injector cyclotron and for an 80-channel current measuring system being built for use in the injection beamline. Logarithmic amplifiers are employed in the latter equipment, and in addition to having to calculate the current in every channel with an expression involving the anti-log, the SABUS microcomputer must also periodically execute a calibration procedure in order to extract calibration constants used in the expression for the current.

Ion Source Position Interlock System

In the central region of the injector cyclotron the allowed positions of the ion source, puller and one of the differential probes are heavily interdependent, but cannot be interlocked by conventional limit switches. Instead we use a stand-alone SABUS microcomputer to provide the interlocking function. The geometry of the motion of the central-region devices is coded in EPROM. The SABUS monitors the absolute encoders which give actual device positions. It compares actual positions with the allowed positions as coded in the geometrical equations, and grants or withdraws permission for the stepping motors which control the devices' motions to be actuated.

Vacuum Control Systems

During the commissioning of the vacuum system for the solid-pole cyclotron it became very apparent that we should attempt to standardize on general hardware that can be programmed to provide the specific control and interlock functions which are required for each of the approximately 30 pumping stations of the facility. To our knowledge there are no commercially-available systems with sufficient flexibility to do the job. However, the modularity of the SABUS system and the availability of Pascal MT+ which produces ROM-able machine code, allow us easily to build up vacuum control systems from SABUS components. A vacuum control station consists of a standard 3U high cardcage with SABUS backplane and modules, and a front panel consisting of a station-specific mimic diagram and a generalized keyboard and alphanumeric display for operator communication.

Other systems which will employ SABUS microcomputers include an NMR multi-probe system which will be used for the stabilization of critical magnet power supplies, the Central Safety Interlock System which at present uses a microprocessor not based on the SABUS but which is being upgraded to make use of Pascal MT+, and systems for monitoring power supplies.

The Interfacing Network

In discussing centralized and distributed systems, Dohan and Gurd, in a recent paper[4], observe that what most clearly characterizes a system is not the power of the processors nor their geographic distribution, but the topology of the hardware communication system linking the processors. The CAMAC System Crate system is a network which has successfully been used in accelerator control systems[5]. However, it has a number of drawbacks which stem from the fact that its topology is a star network, with the system crate being the centre node of the star. The most serious drawback is that information cannot flow from one crate to another within the network without involving one of the processors interfaced to the system crate in the data flow. This places an unnecessary and heavy communication overhead on the central-node processors. The passing of information from one station to another within a crate, and without the involvement of the processors serving the system crate, is possible if an auxiliary crate controller, ANSI/IEEE standard 675, is used. However, the single-crate accessing capabilities of the auxiliary crate controller place rather severe restrictions on the type of distributed control system which is attainable.

A further drawback is that the correct working of the system as a whole, or of parallel branches or serial loops of the network, depends unnecessarily heavily on the correct operation of the system crate modules, i.e. the network is not fault-tolerant with respect to these modules. A third drawback is one of maintenance: it is not possible to do hardware fault-finding and repair work on the modules of the system crate, i.e. to remove individual modules from the crate, without disrupting the network as a whole. There are also maintenance problems with the crates of parallel branches.

In view of the fact that we have thus far implemented only a small portion of the control system, and that there has recently taken place a considerable development in local area networks, we are considering alternatives to the System Crate network, and are looking in particular at an Ethernet-based system. The envisaged system would still make use of conventional CAMAC crates and modules (our capital investment in CAMAC would not allow otherwise) but the parallel branches, serial loop and system crate would be replaced by an Ethernet LAN. For the purpose of evaluating to what extent Ethernet measures up to control system requirements, we have recently bought a small Multibus-based Ethernet system, and are setting up a pilot system.

References

1. J.C. Collins and S.A. Lewis, IUCF Internal Report 78-2 (1978).

2. I. Cloete, NAC Internal Report NAC/I 83-04 (1983).

3. A.H. Botha et al., A Wide-Range Radio-Frequency System for an 8 MeV Injector Cyclotron, this conference.

4. D.A. Dohan and D.P. Gurd, Centralization and Decentralization in the TRIUMF Control System, EPS Conf. on Computing in Accelerator Design and Operation, Berlin (1983).

5. D.P. Gurd et al., Developments in the TRIUMF Control System, Proc. 9th Int. Conf. on Cyclotrons and their Applications (Caen, 1981) p 565.

CH1996-3/84/0000-0388 $1.00 © 1984 IEEE