PERSONAL COMPUTER CONTROL SYSTEM FOR THE TAMU K-500 CYCLOTRON

D. R. Haenni, T. Cowden, C. Hargis, J. Reimund, and R. C. Rogers

Cyclotron Institute, Texas A&M University,
College Station, Texas 77843 USA

ABSTRACT

A highly-responsive, parallel-processing control system has been constructed for the TAMU K-500 cyclotron. It is based on networked IBM PC-XT type computers. Hardware features include interfacing via a high-speed, memory-mapped, byte-wide data highway and a simple push button control console which gives immediate access to a large number of controls. Software is written in a special version of FORTH. A program generator developed around a data base helps to produce code for the individual computers. The system does not maintain a centralized on-line data base but rather relies on direct hardware access for necessary values. All network communications use plain text messages. The control system has provided reliable, continuous service since the initial operation of the accelerator. The system is still being expanded as funds, interfacing, and manpower permit.

INTRODUCTION - DESIGN GOALS

The design of the computerized control system for the K-500 superconducting cyclotron at TAMU was arrived at after investigating existing control systems at similar sized accelerator laboratories and commercial systems. Experiences and insight obtained from operating the hard-wired system for the old 88 inch cyclotron at TAMU also played a major role in many design decisions. The following design goals were developed.

The control system must:

1. allow two or more parameters to be adjusted by the operator at the same time.
2. be fast and responsive to operator commands.
3. have a simple operator interface not requiring unusual hand-eye coordination or typing skills.
4. minimize the number of exceptions to general control procedures.
5. monitor all active parameters for slow drift.
6. protect against operator blunders.
7. report system failures.
8. allow for significant system expansion.
9. allow for the future addition of complex control functions.

The control system was implemented with a minimum hardware budget (about the price of a good sized minicomputer when the project began) and manpower (less than two full time hardware persons and one software person).

The first three design goals combined with budget limitations eliminated the options of buying or directly copying an existing system. The control system which has been implemented is an amalgamation of ideas along with some innovation. It meets the design goals while remaining within budget and manpower constraints.

CONTROL SYSTEM HARDWARE

The control system is built around a loosly coupled parallel processor consisting of networked Turbo XT clone computers which are IBM PC-XT compatible but run at 8 to 10 Mhz. These computers were chosen because they have a:

1. relatively large 1 Mbyte memory address space.
2. 16 bit processor (Intel 8088).
3. well documented, open I/O bus which facilitates building interface cards.
4. large general acceptance insuring ample supplies of commercial peripherals, software, and documentation.
5. very low initial cost.
6. no recurring hardware or software maintenance costs.

A local area network provides a high-speed data path for messages. Arcnet (token ring, 2.5 Mbit/sec, 254 nodes) was chosen for this purpose. A commercial interface card provides memory mapped I/O buffers and an LSI controller chip which carries out most of the message passing tasks. Initially this network was also to provide for program downloading. Appropriate network software was not available so an IBM PC-Net (broadband) which runs standard IBM PC-Network software was installed to provide the downloading function. Message passing over Arcnet is managed directly by the control system software.

A simplified schematic diagram of the control system is presented in Fig. 1. The processors are shown along with their network connections and an indication of the function of each is given. Only those systems directly involved in the operation of the accelerator connect to the Arcnet. The processors are logically divided into three groups: operator interface, accelerator interface, and support. The control system software already has provisions for the addition of a second full control console. More interface computers can be added as necessary.

The operator interface consists of the control console (discussed below) and a network terminal. The latter controls the accelerator by typed commands. It was intended as a short term debugging tool but has proven useful during regular operation. The accelerator interface computers connect to the cyclotron

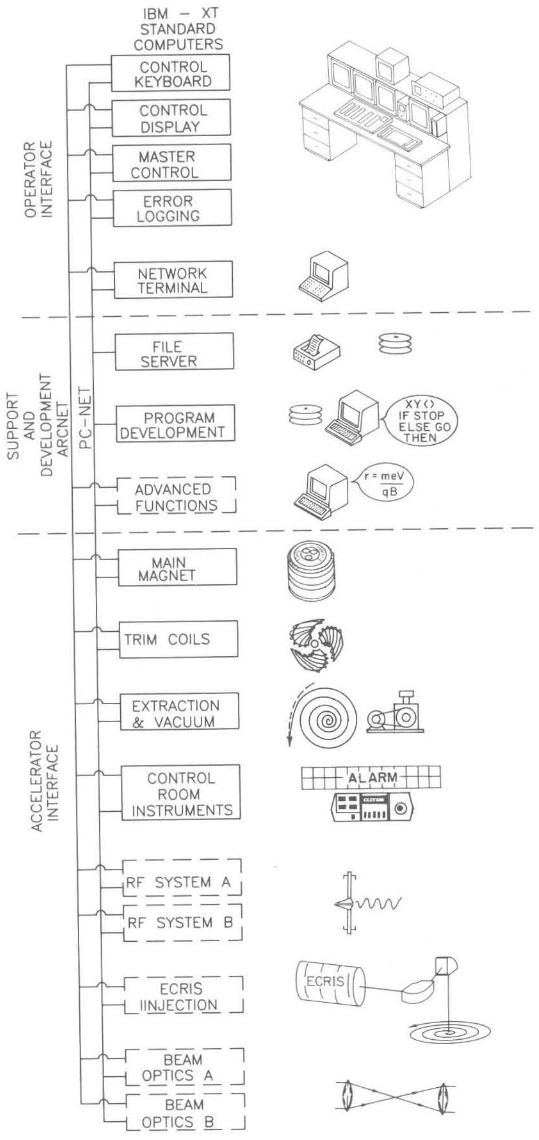

Fig. 1 Schematic diagram of the control system showing computers, network connections, and the major function assigned to each system. Dashed computers are planned but not yet implemented. All computers use Intel 8088 processors except for the file server and programming systems which have 80286 and 80386 processors, respectively.

hardware. The support computers provide file serving and a programming / general processing platform. The advanced function computer will be added when it becomes necessary to implement high-order operations such as emittance measurements.

The various accelerator hardware components are interfaced to the control system through industry standard STD-Bus (IEEE-696) crates and I/O cards. Normally such crates contain local intelligence but here a special driver card accesses the 256 byte I/O space of the STD-Bus. Up to 255 such crate drivers can be connected to a byte-wide data highway driven by a controller which maps to a 64 kbyte segment of the PC memory address space. In this way the various STD-Bus I/O cards can be directly accessed via memory reference instructions in 2 to 4 μsec. The highway and crate driver cards were designed and constructed in-house. Most of the STD-Bus interface cards (TTL and relay I/O)

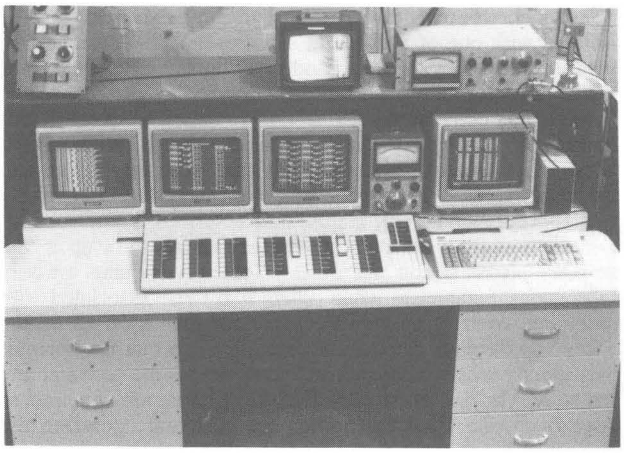

Fig. 2 Photograph of the main operator control console.

are commercial products. An exception to this was the analog output (DAC) cards. A card providing either four 12 bit DACs or 2 pairs of overlapped (coarse / fine, 20 bit resolution) DACs was also designed and fabricated in-house.

Many of the power supplies have both local (front panel) and remote (computer) controls. To eliminate calibration problems, the local parameter displays use digital panel meters, DPMs, with digital outputs that can be connected to TTL I/O cards. Thus both sets of controls share common readout devices. There are many 4.5 and some 5.5 digit DPMs in the system which have much higher accuracy than the usual 12 bit ADCs found on most commercial analog input boards.

OPERATOR CONTROL CONSOLE

In designing the operator console experience gained from the hard-wired control system for the 88 inch cyclotron was most useful. Stability, consistency, speed, direct access to a large number of controls, displaying values in meaningful units, and minimization of keystrokes were the most desirable attributes. Since tuning the accelerator normally involves adjusting parameters while watching the beam on either a current meter or beam line phosphor, emphasis was placed on response to commands not updating changing value displays. The control console does not need graphic displays to accomplish its primary function. Higher-level controls or diagnostics which may require graphic displays will be added as needed through the advanced functions system.

A picture of the operator console is shown in Fig. 2. It consists of four displays, a normal computer keyboard, and a special control keyboard. Four computers are stored in the drawers visible from the front. Two processors run the control keyboard and associated displays. The third provides master controls (currently similar to the network terminal) while the fourth handles error logging. The floppy disk drives, right hand display, and computer keyboard are attached to the master controls computer.

The control keyboard and the two displays in the center of the console are divided into 48 control units. Each control unit can be associated with a controllable hardware object called a device in this system. A control unit consists of a push button pair along with a device name display on the keyboard and a corresponding area on a CRT display containing "key caps" for the push buttons, the device name, the value of the controlled parameter with units, and general device status (fault, local, on, parameter monitored, DAC limit, and device busy). At any time the operator can directly activate one of 96 device commands (2 from each of 48 devices) by selecting the appropriate push button.

The 48 control units displayed at a given time (screen) may be associated with one of 8 different sets (levels) of 96 commands. Levels are selected by the right hand set of lighted switches and the step size selector (the push button with display just above the keypad). For consistency 13 of the 16 possible control unit commands are associated with particular device commands. Eight of these are increase / decrease commands with large, medium, small, and fine step sizes. The others are parameter set, plus / minus (polarity changes), status, and reset. If a control unit provides any of these commands, then they always show up at "the same place" (push button and level). Some control units are only for displaying parameters while others like those associated with Faraday cup controls provide the same commands on all levels. The key cap displays change with each level so that the operator knows exactly what commands and push buttons are active at any given time. In the worst case four keystrokes are required to change from one level to another but in actual operation the average is closer to one. The numeric keypad is only used to enter values in conjunction with the parameter set operation.

The console has a maximum of 96 screens (4608 control units / 73278 commands). The four push buttons with displays at the upper right corner of the keyboard can be randomly assigned to any screen. Any of the four screens can be activated by a single keystroke in slightly over one second. Most of this time is spent downloading a new set of device names to the keyboard. A new screen can be assigned to this set and activated with three keystrokes. The first selects the screen to be replaced and the second activates the assign screen function (bottom left hand lighted button). Each control unit push button then becomes associated with a different screen and the third selects the new screen. The screens are kept as overlays on 6 Mbyte ram disks in the keyboard and display computers. This type of change is accomplished in less than two seconds. Any of the 73278 control console commands can be randomly selected with an average of 5.2 but no more than 7 keystrokes. Since related devices are grouped on the screens, the operator usually has direct access to needed controls and is not continuously changing screens.

A control console appears more friendly to the operator when the controls are always "in the same place." Therefore the assignment of control units to screens is fixed when the display and keyboard computer programs are compiled and cannot be dynamically changed while the programs are running. Some of the control systems investigated while designing this one had the equivalent of dynamically assignable screens. This feature was not heavily used, however, and these systems were run with essentially fixed screen assignments.

The control keyboard is continuously scanned for push button changes. Every 100 ms, commands are issued based on the keyboard status and observed changes. Control push buttons can be set to repeat commands as long as the button is activated (e.g. ramp commands) or to issue a single command per activation (e.g. status commands). The keyboard and display computers coordinate operations by passing messages over the network. The display highlights the key caps corresponding to activated push buttons. The hardware interface computers are informed every time the control console screen is changed. They in turn send value change messages to the display computer only for those parameters which are currently displayed.

The error logging computer displays control system error messages on the logging display (left hand one in Fig. 2) and writes them to a disk file. In addition, replies to status commands requested at the control keyboard are displayed on the logging screen.

GENERAL SYSTEM OPERATION

Control system operation involves interaction between the computers in the operator and accelerator interfaces. The operator interface sends control commands to the accelerator interface and receives value / status replies. The accelerator interface executes the control commands and sends value / status replies. All commands and replies are plain text messages. No messages are allowed between the accelerator interface computers. Except for the control display / keyboard coordination and error messages the operator interface computers are also not allowed to communicate with each other.

Each device in the control system is connected to a single interface computer. The device driver software is responsible for hiding most device hardware details and control procedures from the rest of the system. This allows a fixed list of commands to be used for controlling all devices. The device driver contains code to carry out commands appropriate to the device. The rest of the commands in the list produce error messages. Device drivers are also responsible for monitoring the control console screen and sending value change messages when appropriate. Unlike many other control systems this one does not use an on-line centralized data base. Instead the device drivers rely on the high-speed data highway to obtain information directly from the accelerator hardware. This eliminates a potential I/O and compute bottleneck, minimizes the chances of making decisions based on old information, and greatly reduces the amount of network message traffic. There is one drawback to using intelligent device drivers. If the operation of one device depends on information from a second then both devices must be interfaced to the same computer.

The keyboards and displays attached to the accelerator interface computers provide a means of entering local control commands. Since MS-DOS is a single tasking system, the control system programs provide internal cooperative multitasking with foreground / background operation. All tasks must either quickly terminate or run as regularly scheduled real-time or background tasks. Control commands (local and external) and real-time operations are run in the foreground on a first come, first serve basis. Since there are no extremely time critical functions this allows for excellent response to control commands. While waiting for foreground commands the programs run background tasks. Chief among these is providing value change messages to the control panel.

CONTROL SYSTEM SOFTWARE

The limited programming manpower available for software development dictated that efforts should be made to simplify the programming task. The approach taken to develop the control system software involves use of program generators (i.e. software that helps generate the control system programs) combined with reusable code modules. The program generators were developed around a data base which contains a description of the control system. This approach eliminates problems associated with the sharing of details between the various programs by having only a single source for such information. It is also possible to design the control system programs in such a way that they could be expanded or modified by simply changing the data base. The reason for employing reusable code modules is obvious. This is one of the best ways to minimize the time necessary to code and debug the software. Another important way to simplify the software development task is to have hardware which compliments the software. Many control system design decisions were made with this in mind. For example the accelerator hardware interface design virtually eliminates software overhead by allowing the computer to access the interface as slow memory.

One of the control system design goals is to standardize global operating procedures. In other words control functions at the operator console should always look and act the same. If various devices require different operating procedures for the same control function then the software should hide these details from the operator. Minimizing the number of exceptions to the global operating procedures, which the operator must remember, makes the control system more "friendly." Compensating for a large number of exceptions, however, significantly increases software complexity especially when trying to develop program generators and reusable software modules. Most of the problems encountered during software development have arisen from underestimating the amount of flexibility needed to cope with differences in control procedures. A general rule of thumb might well be "never expect to do the same thing twice."

The control system software is written in FORTH. This language is well suited for control system programming. It allows direct hardware access, provides easy integration with assembly language code, and fosters a programming style which takes every advantage of reusable code modules. Details of this somewhat different computer language can be found elsewhere.[1] FORTH is traditionally a 16 bit language. While this matches the structure of the 16 bit PC processor, it limits programs to 64 kbytes which is too small for the task at hand. Fortunately, FORTH language interpreters are relatively easy to develop in assembly language. mulFORTH was written to support large control system programs by holding multiple FORTH systems in memory and allowing program control to pass between them. mulFORTH was designed with the express purpose of simplifying control system programming. Important features include the ability to have position independent FORTH systems as overlays; draw code from an external library during compilation; and compile programs from ASCII text files. mulFORTH is highly reliable and has been in use for over two years.

The program generation tools were developed with a commercial, programmable data base management system, FoxBase. Programs written in the FoxBase command language extract information from the control system data base and produce FORTH code in the form of mulFORTH libraries. A control system program can be generated by compiling a "skeleton" program which needs code from its corresponding library. The skeleton program contains all parts of the control system program which are not provided through the library. A single skeleton program was developed which generates any of the accelerator interface computer programs when compiled with the appropriate library. The programs for the operator interface computers, however, all require separate skeleton codes.

Logically, the control system data base should store its' information in variable sized records but these are not supported by FoxBase. This is circumvented mapping the logical control system information into multiple fixed size FoxBase records. To simplify working with such a data base, a menu driven editor program has been developed which hides this mapping from the user.

In the areas of general control system information, control unit descriptions, and screen organizations the control system data base contains essentially system parameters. The code generators convert this information primarily into tables which are compiled into the programs. To describe a device driver, however, the data base must contain both parameters and programming information. Initially it was intended that code modules (in the form of FORTH defining words) would be written to carry out complete device driver commands or operations. The data base would specify a single code module and any required parameters for every command and real-time / background operation needed to describe a device driver. Likewise the data base would specify code modules for some support functions (e.g. data highway access, flag / timer management, etc.). This approach would have greatly simplified device driver programming if the the total number of code modules needed to describe the control system stabilized at some reasonably small value. The data base would then have allowed new devices tq be added to the control system without further programming. After considerable effort this approach proved incompatible with the K-500 cyclotron hardware. The number of different support functions did tend to stabilize. This

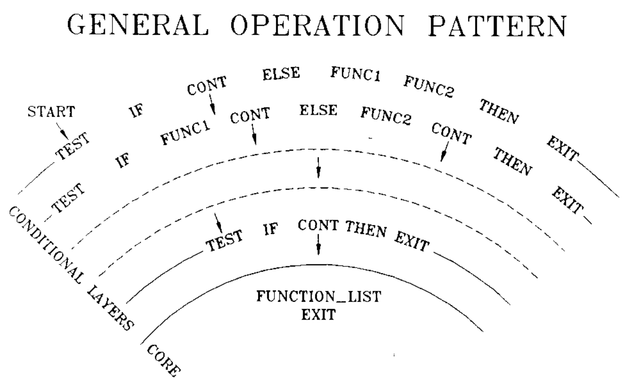

Fig. 3 Program pattern for device driver commands and operations consisting of a core function list surrounded by layers of conditional tests. A few typical layers are shown using IF (true) ELSE (false) THEN constructs. TEST provides a logical value; CONT drops to the next layer; EXIT terminates the operation; while FUNC1 and FUNC2 are functions.

was mostly a consequence of limiting the number of different STD-Bus interface cards allowed in the system. Unfortunately, the number of code modules describing commands and operations increased in proportion to the number of devices in the system. It was found, however, that all of these code modules followed more or less a single simple program pattern. Furthermore the modules consisted of many small but heavily repeated code fragments. This suggested that the number of code modules needed to describe device drivers could be stabilized by increasing the complexity of the data base. The repeated code fragments would be converted into support functions. A command or operation would be specified in terms of these new support functions using the observed program pattern. In this way the data base takes on aspects of a high-level control system programming language.

The program pattern for commands and operations is shown in Fig. 3. It consists of a core action surrounded by layers of conditional tests. The core action is just a list of support functions. Conditional layers are essentially if ... else ... then(endif) constructs. The data base specifies the logic test and the action to take on true and false. One may continue to the next lower conditional layer and/or execute a support function. Code modules have been developed for conditional layers having all combinations of continue actions and up to two support functions. More complex combinations have not yet been needed. The simplicity of the program pattern masks its power. The following sequence is necessary to change a trimcoil polarity; ramp the supply to zero current, turn it off, change the polarity switch, clear the automatic fault, and turn it back on. This sequence is coordinated by a real-time operation having only 12 conditional layers.

When expanded into FORTH code the control system programs are already in excess of 600,000 lines. This grows as more devices and features are added. The software development approach and program generation tools described here are sufficient to manage even such a large system.

REFERENCES

1) Brodie, Leo, Starting FORTH (Prentice-Hall Inc., Englewood Cliffs, NJ, USA, 1981).