HEDL-SA-1822-FP
CONF-790955--1

MASTER

A GENERAL PURPOSE OPERATOR INTERFACE

S. I. Bennion

July 1979

International Society for
Mini and Microcomputers
September 26-29, 1979
Montreal, Canada

HANFORD ENGINEERING DEVELOPMENT LABORATORY
Operated by Westinghouse Hanford Company, a subsidiary of Westinghouse Electric Corporation, under the Department of Energy Contract No. EY-76-C-14-2170

DISCLAIMER

A GENERAL PURPOSE OPERATOR INTERFACE

Scott I. Bennion

Hanford Engineering Development Laboratory

Richland, Washington 99352

Abstract

The Hanford Engineering Development Laboratory in Richland, Washington is developing a general purpose operator interface for controlling set-point driven processes. The interface concept is being developed around graphics display devices with touch sensitive screens for direct interaction with the displays. Additional devices such as trackballs and keyboards are incorporated for the operator's convenience, but are not necessary for operation. The hardware and software are modular; only those capabilities needed for a particular application need to be used. The software is written in FORTRAN IV with minimal use of operating system calls to increase portability. Several ASCII files generated by the user define displays and correlate the display variables with the process parameters. It is also necessary for the user to build an interface routine which translates the internal graphics commands into device-specific commands. The interface is suited for both continuous flow processes and unit operations. An especially useful feature for controlling unit operations is the ability to generate and execute complex command sequences from ASCII files. This feature relieves operators of many repetitive tasks.

Introduction

Nearly all computer-based data acqusition or process control applications require an interface to an operator who ultimately controls the activity of the system. A considerable portion of the implementation of such systems is directed at the development of a suitable operator interface. It is important that these interfaces provide accurate, timely information in a concise, unambigious manner. Once these criteria have been met, it is then highly desirable for the interface to be both standardized and expandable.

An effort to develop such a system is presently underway at the Hanford Engineering Development Laboratory (HEDL) in Richland, Washington. A general purpose operator interface is being developed which will be suitable for most future needs. The basic system utilizes color graphics displays for information display and touch sensitive screens on the display monitors for operator interaction. A number of hardware and software options provide considerable enhancement and expansion capacity.

Hardware

The user is afforded considerable latitude in configuring an operator's console tailored to his specific requirements. He can choose from a number of options as shown in Fig. 1. The system was designed to allow addition of new hardware devices. That addition should simply be a matter of providing a physical interface and installing an appropriate software interface module.

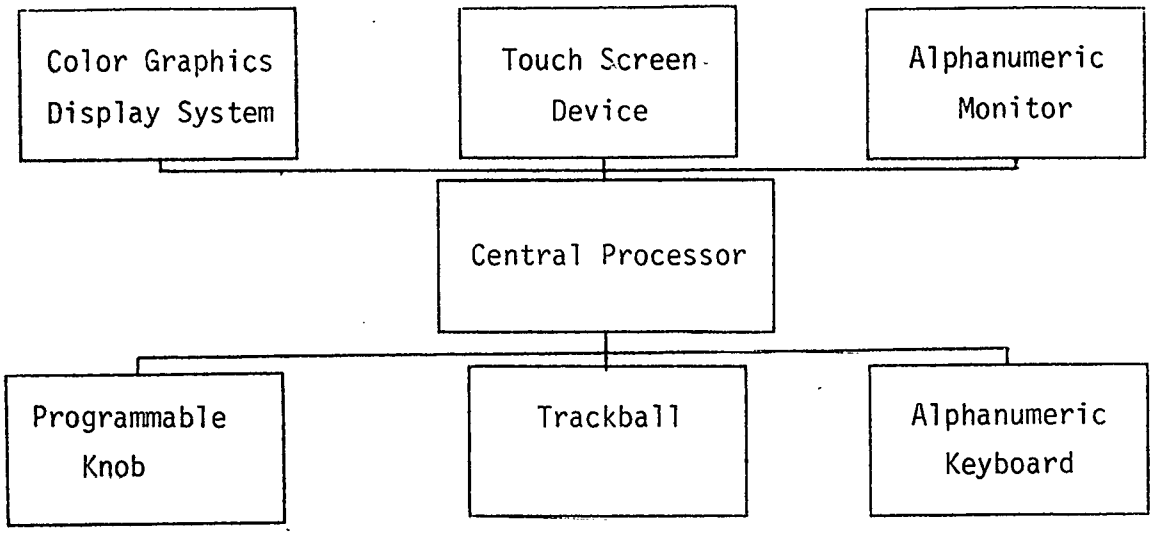

Figure 1. Hardware Options

One of the primary goals was to provide the user with the maximum flexibility in his selection of each hardware device. Thus one user might choose to install an inexpensive color graphics terminal while another might opt for a powerful graphics processor. The two users will realize markedly different operational characteristics due to the widely divergent capabilities they paid for. However, both will be able to make use of the basic software package and hardware configuration with little or no change. Similarly, a user can upgrade to more powerful hardware with minimal modifications in the overall system.

The graphics display device is the principal element in the operator interface. The software was developed in a general manner to allow the use of most available devices, including the case of limiting the "graphics" to black and white alphanumeric displays. The intent at HEDL is to concentrate on the use of microprocessor-based color graphics terminals. The use of color graphics provides maximum flexibility to the system engineer in designing and implementing a data display and process control scheme. Complex data relationships are communicated efficiently and clearly through graphical displays. Adding color to the display greatly increases the density of information which can be displayed and comprehended. The use of color graphics essentially adds another dimension to the display without significantly increasing the probability of operator confusion.

Graphics displays allow the operator to view all required process data at a compact work station. Instead of having a control console consisting of many dedicated signal indicators laid out in a large area, he receives the information he needs in the form of graphic displays. During steady-state operation, he monitors a summary display which shows the key system parameters. During an alarm situation, he observes detailed displays providing data related to the alarm. In all cases, he observes

the information he is currently interested in. A compact console is capable of bringing a great deal of process information to a single work station. The traditional control room requires that the operator travel around the control room gathering the information he needs.

Another consideration to note here is that reconfiguring the signal display in a traditional control room is a complex hardware modification task. In the graphics approach, it consists of simply reformatting the displays.

The graphics display device must be capable of processing a small set of displayable commands. All displayable commands from the software package are sent as a string of characters conforming to the American Standard Code for Information Interchange (ASCII). These ASCII strings are interpreted by the display terminal. The actual method of command interpretation is unknown to the operator interface software and could be any one of several methods, including a separate software interpreter in the host computer, firmware in the display terminal, or a hardware implementation in a graphics processor.

A serious deficiency in the prevalent implementations of graphics-based display and control systems is the practice of requiring the operator to communicate with the control software through a keyboard using standard commands. System operators are highly skilled individuals trained to perform a complex task. To also require them to be a skilled typist is an undesirable situation. During system transients or off-normal operations, where their skills and judgement are most severely taxed, they are also expected to perform the manual command entry functions most frequently. This conflict requires that considerable time and effort be expended in developing their manual skills to a high level, resulting in the loss of otherwise qualified operators who do not happen to possess the manual dexterity required to type quickly and accurately in a stress situation. A decision was made at HEDL to utilize the concept of touch sensitive screens (touchscreens) on the display devices to eliminate the need for keyboard entry of commands.

For our purposes, a touchscreen is defined as a device which reports the position of an operator's finger on or near the screen of the display device. The principal devices presently available are based on one of three general approaches. The first approach consists of imposing a matrix of light beams over the display screen. When the operator touches the screen, the broken beams are detected, generating a position signal to the host computer.

In the second approach, ultrasonic pulses are periodically propagated along the screen in the horizontal and vertical directions. Touching the face of the screen produces an echo. The echo timing provides sufficient data to determine the location of the disturbance. This location is determined by the device electronics and sent to the host computer.

The third approach consists of generating a standing voltage wave on the surface of the display device. When the screen is touched, a conductive overlay makes contact with the screen, picking off a proportional voltage which allows determination of the contact location. Each of these approaches involves a series of design tradeoffs involving resolution, accuracy, reliability, and cost. The operator interface was designed to allow the system engineer his choice of devices based on the system requirements.

The touchscreen allows the operator to direct his attention at the display at all times instead of shifting from the display to the keyboard and back again. All control actions are initiated by touching the appropriate area on the display.

In addition to the signal indicators, a traditional control room contains many manual input devices such as pushbuttons and knobs. A means must be provided to duplicate their functions for the operator interface to be effective. An approach which has been used with some success is to provide a set of general purpose pushbuttons which are used as the input devices. Their functions are modified under program control to provide multiple uses for each button. The current function of each button is indicated through a variety of means, including plastic overlays, projected overlays, backlighted keys, or a displayed menu of functions.

By combining the touchscreen concept with a small alphanumeric monitor, the operator can be provided with a very versatile set of programmable pushbuttons. The buttons are clearly labeled with their functions and operator prompting is available through the use of blinking and reverse video. When the operator changes his current display, the programmable pushbuttons on his alphanumeric monitor are also updated. The system engineer has a great deal of flexibility in the control scheme design by deciding which input functions will be generated through the graphics displays and which will be generated through the programmable pushbuttons.

The operator must be provided with a method of adjusting system parameters quickly and accurately. The capability for two types of input are provided. The first consists of "increase" and "decrease" buttons on the display. By touching a button, the parameter is adjusted in the indicated direction until the desired setpoint is achieved. This option is generally used in situations where adjustments can be anticipated. In this case the adjustment capability is built directly into the display. The second approach utilizes an incremental shaft encoder mounted in the control console. The current assignment of the knob is displayed on one of the display devices. Under operator control, the knob can be assigned as the control device for any system parameter. This assignment is then indicated on the reserved display line.

The parameter behavior under either option is controllable by the operator. He has his choice of linear, logarithmic or incremental adjustments, choice of scale factor and choice of positive and negative

values, positive only, or an interval. In all cases where adjustment buttons are provided as part of the display, he can override those buttons and assign a programmable knob to the parameter.

Two other devices were made available during the initial system design; a trackball and keyboard. The trackball was provided at the request of potential system users involved in the design of a control system for a high-energy particle accelerator. The use of trackballs for operator input in this type of application is widespread and well accepted. Even though the trackball is a redundant feature due to the presence of the touchscreen, it was felt that the addition of the trackball would ease the transition period for trained operators.

Even though the system is completely operable without a keyboard, the capability of connecting a keyboard for use as an input device was considered important. The keyboard is particularly useful during system integration and checkout. All inputs are presented to the software as strings of ASCII data. This allows any input device to be checked out with, or replaced by, a keyboard. It is anticipated that the keyboard will not normally be connected to the interface system; rather it will be used by hardware and software maintenance personnel to diagnose system errors and to make modifications to the system functions.

Application Programming

Once the operator interface hardware is configured and the system engineer has designed the display format and content, he must build three files to implement his system. These configuration files consist of the display file, the background file, and the symbol library. An additional set of files, the command input files, are built once the system is operational if the user wishes to utilize the command input capability for unit operations.

The configuration files are ASCII source files constructed according to a specified format. This simplifies generation and checkout as well as error detection and correction. This structure also improves the initial software implementation effort since the data is easily read and traced.

All display parameters are assigned to a symbol type. Each symbol type has a specified display format and set of valid values. Examples of symbol types would be a normalized bar graph with associated percent of full scale and alarm status, a proportional value with an open/closed indication and percentage open, or a text symbol for display of alphanumeric data.

The user defines his set of valid symbols by building a symbol library. The symbol library is accessed by symbol name. Each symbol name has associated with it a collection of records. Each record corresponds to a valid symbol value or status. The current parameter value is used to access the correct record. Within each record are the ASCII command sequences necessary to display the symbol in the appropriate state.

These commands all plot the symbol relative to the current cursor position rather than at an absolute location.

For new applications, the user has access to all symbol libraries previously created. Since they are source files, he can use these libraries to assemble his own library quickly through the use of a text editor. He then completes his library by defining any new display symbols he needs and adds his new symbols to a master library file for the benefit of future users.

In most cases, the majority of the display contains only static information, the display background. The variable portion is a small percentage of the total display. In addition, several different displays often make use of the same background display. In order to reduce redundancy in the file system, these background displays are stored separately from the definition of the dynamic portions of the display. When a new display is invoked, the appropriate background is retreived and displayed, then the variable data is overlaid on the background.

The background file is accessed by background file name to retreive the executable command sequence needed to draw the static background. This command sequence is one of two types. When the display hardware has the capability for direct memory access(DMA), the command sequence is simply a memory restore command referencing the memory image file of a previously saved screen image. When DMA capability is not provided, the command sequence necessary to draw the background through the display command interpreter is retrieved.

The user generates his background file in a manner appropriate to the hardware capabilities he elects to provide. When DMA is available, he simply draws his background and saves the screen image under an appropriate file name. When DMA is not available, he generates a sequence of executable commands which will draw the background when they are processed by the command interpreter.

The display file is the primary implementation file. It defines the data content and format of the display as well as the response of the touchscreen. The contents of the display file are used to set up the current display and to load two dynamic files which change along with the displays, the parameter correlation file and the screen vectors.

The display file is actually a collection of files. Each display has a unique name which is the access key to the proper file. Each file contains a collection of records. Each record contains a variable name, a parameter name, a symbol name, and x- and y-coordinates, and if the defined screen location has an associated touchscreen response, one or more tokens, (a token is a syntactic entity such as a keyword, identifier or operator.)

The variable name is the internal label assigned to a referenced external parameter. All internal references to data are made through variable names.

Whenever a new display is invoked, the display file is accessed to retrieve the description of the new display. When this takes place, a dynamic file, the parameter correlation file, is updated. The parameter correlation file consists of a series of records keyed on the parameter name. Within each record is a list of all variable names currently associated with that parameter name. Note that all valid parameter names are contained in the file even though they may not all have a variable name currently associated with them. Whenever a command is received which refers to a parameter name, the name is verified using the parameter correlation file. If the name is not found, an error condition exists and the user is notified of an inconsistency in his display definitions.

Whenever data is received from the external process, it is identified by parameter name. The parameter correlation file is used to associate the data with the proper internal variables. Note that several internal variables can refer to the same parameter name but that the converse is not true. In this way, the same external parameter can be tracked on several different displays in several different formats.

Each internal variable has associated with it a symbol name. This symbol name, as previously described, defines the display format of the variable and its allowable values.

As was also previously mentioned, the executable commands in the symbol library are all relative commands for plotting in relation to the initial cursor position. The x- and y-coordinates associated with each variable name specify the screen location for the cursor when plotting the symbol.

For those internal variables with an associated touchscreen response, the display file contains one or more tokens. These tokens constitute all or part of a system command. At the same time that the parameter correlation file is updated, the screen vectors are also updated. The screen vectors consist of a screen position and associated tokens. When the screen is touched, the location is sensed and the tokens are sent to the command interpreter. In this way, the touchscreen replaces the keyboard by composing commands as if they were from the keyboard but without the operator needing to be aware of the specifics of the syntax.

A very powerful capability which is provided for the user is the command input file. These files are sequences of valid commands which are processed without operator involvement. This allows the user to set up files to perform repetitive or well-defined operations, freeing the operator to pay attention to the overall process. This feature is particularly useful in discrete operations where a series of identifiable operations takes place and in directing a continuous flow system through a predetermined process cycle.

The command input file is an ASCII source file containing valid process commands. Once the file begins operation, it continues without operator

interaction. Each time a command is satisfied, the next command is read from the file and executed. Additional features include nesting of files with multiple passes and conditional operations.

An executing command input file can initiate the execution of another command input file. That file can in turn invoke yet others to a nesting depth determined only by the amount of memory the user wishes to allocate to the stack area. When an invoked file completes its operation, commands are accepted from the calling file starting with the next sequential command. This feature allows the user to build his control files in a systematic manner and share common operations with a single command input file, much as the use of subroutines allows sharing of common processes.

The nesting feature also allows the user to specify the number of times the invoked command input file is to be executed. Each time the file has been executed, the number of passes is incremented and compared with a target count. When the target count is reached, control returns to the calling file.

Another feature of the command input file which improves its versatility is the conditional command. The conditional command allows groups of commands to be skipped or repeated depending upon the status of an external parameter. When a conditional command is encountered during the execution of a file, the current value of the specified parameter is requested from the external process. This value is then compared to a specific value using one of the allowed logical operators. If the test is true, then the file pointer is moved forward or backward by the indicated amount. If the test is false, then the next sequential command is executed. The allowable logical operators are:

- less than (<)
- greater than (>)
- equal to (=)
- not equal to (<>)
- less than or equal to (<=)
- greater than or equal to (=>)

To facilitate the testing of command input files, two additional features are provided. The first is a single-step feature. In this mode, the operator interface prompts the user following the completion of each command. The next command is not executed until the operator acknowledges the prompt. This allows the process to be verified step by step. The second feature is a test mode flag. When the interface is in test mode, a flag indicating this status is passed to the external process. If the user has provided a simulation capability in his external process, he can simulate the entire operation of his process, including the control function.

Software

The decision was made early in the project to utilize FORTRAN IV as the programming language for the development of the operator interface. The languages given serious consideration were BASIC, FORTRAN IV, and PASCAL. BASIC was considered because of its simplicity and string

processing capability; FORTRAN IV because of its relative speed and general availability; PASCAL because of its versatility and readability. FORTRAN IV was finally chosen because of its general availability and speed. Since the interface was supposed to be transportable, PASCAL had to be eliminated for the time being even though it was highly rated on its technical merits.

Because the software package should run on any host computer with little or no modification, it was important to restrict the use of extensions to the language. Any use of a non-standard feature was discouraged. This effort toward portability included the use of operating system calls. Even though the original development was done on a DEC PDP-11/40 computer running under the RSX-11M operating system, it was expected that the software would be used on a number of different processors under a variety of operating systems. Therefore, it was important to restrict the usage of operating system calls to the minimum necessary and isolate those that were used. When operating system calls were used for functions such as intertask handshaking and file system calls, they were isolated into subroutines which served that single function. This resulted in additional processing overhead for any single application but improved portability considerably. The specific subroutines must be modified for the host operating system once for each type of call rather than at each occurance of the call throughout the code.

The software for the operator interface was designed to support a wide range of hardware capabilities and operating modes. It is this software which provides a truly general purpose interface. An effort was made to develop the system as a collection of well-structured modules in a transportable language. The upper level software structure is shown in Fig. 2.

Note: PDP-11/40, DEC, and RSX-11M are registered trademarks of Digital Equipment Corporation, Maynard, Massachusetts

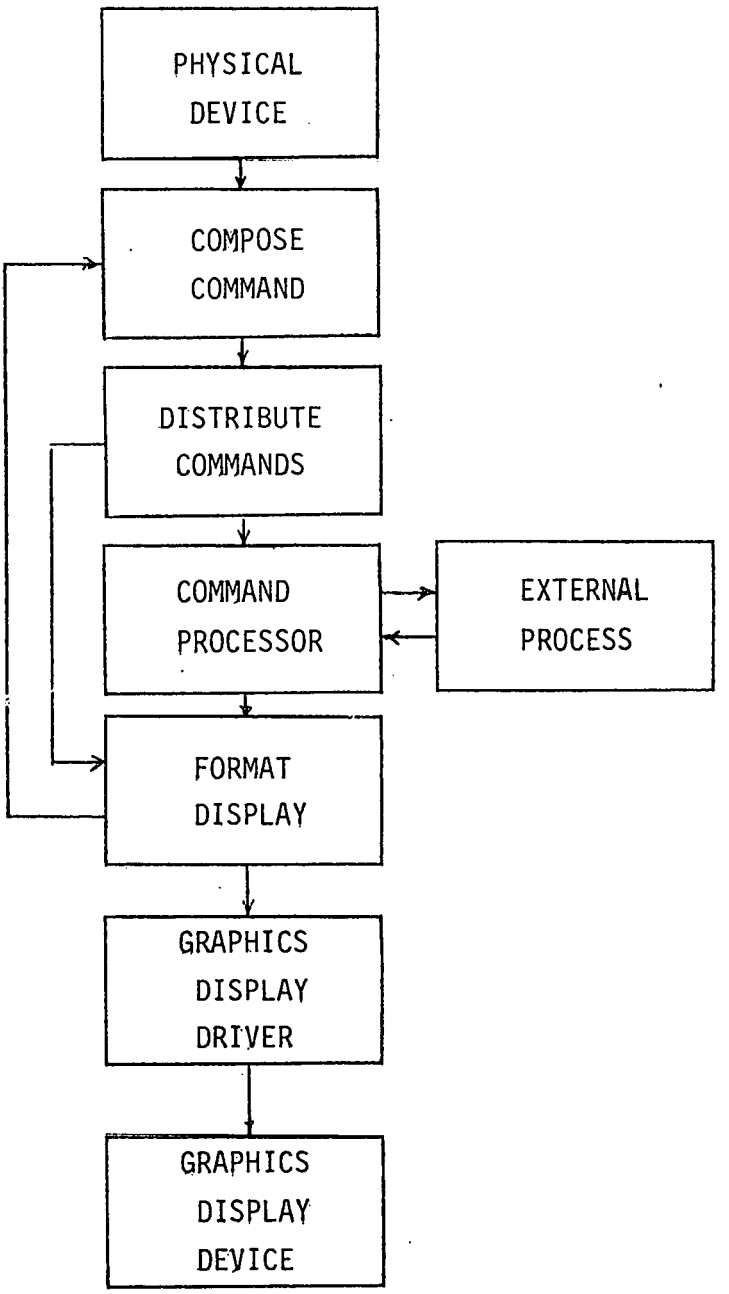

Figure 2. Operator Interface Software Structure

The COMPOSE COMMAND module receives information from the input devices incorporated into the operator interface and generates valid commands to be sent to the DISTRIBUTE COMMANDS module. The three principal input devices are the keyboard, touchscreen, and programmable knob. Even though the keyboard is not intended for use during routine operations, all commands are handled internally as if they came from the keyboard. Since it is likely that commands generated from the keyboard will occassionally contain errors, they must be checked for correct syntax and valid references.

Incoming keyboard commands are parsed into tokens. All tokens which are not keywords or operators are checked against the parameter correlation file to verify that they refer to valid parameter names. If an invalid reference occurs, the command is ignored and the operator is notified of the unmatched parameter name. As the tokens are validated, they are passed to a module which builds an array of tokens. Upon detection of a carriage return, the token array is arranged according to the syntax rules found in a command library. If the command is syntactically correct, it is then sent out for distribution; otherwise, the operator is notified of the error. All information available about the error condition is displayed to the operator. Recall that at this time the user is probably an applications programmer and would be expected to be familiar with the command structure.

An expandable command library is provided as part of the operator interface software. Should a user wish to modify or expand the command structure, he simply makes the necessary modification or addition to the command library. In this case it will be necessary for him to check the affects generated in other sections of the software. The most likely modifications will be the addition of commands for the external process. In this case, there will be no side effects elsewhere in the software.

When the operator input comes from the touchscreen, the information is received as a series of tokens, making parsing unnecessary. Each time the operator presses the touchscreen the screen vectors are accessed using the screen location as the access key. The tokens associated with that location are then added to the token array. In addition, each time an entry is found, the operator is notified through some form of audiovisual feedback through the display device. Likely forms of feedback include generating a tone or changing the color or shape of the corresponding symbol on the display. This action is performed by changing the value of the internal variable associated with the location touched. The feedback is specified by the user as part of the symbol library he generates. If the screen location pressed does not have a token associated with it, no action is taken.

Input from the programmable knob is repetitive in nature. That is, the same command is issued many times with only the value of the operand changing. In order to reduce the processing required in an operation of this type, the command is saved in a buffer. Whenever another

numerical value is received from the programmable knob, that value is substituted into the command and the command issued.

Once a command is issued, it is routed to the appropriate section of the software by the DISTRIBUTE COMMANDS module. The routing depends upon the type of command issued. The fundamental command types are: display command; process command; and command input file command.

Display commands instruct the FORMAT DISPLAY module to modify the current display, usually by calling up a new display. No additional processing of display commands takes place in this module.

Process commands are simply sent to the external process and are not interpreted by the operator interface. In the case of a command to change a parameter value using the programmable knob, however, the command is modified according to one of two options. The first option allows the operator to use the knob to control the absolute parameter value. The parameter is assigned a value determined by applying a transformation algorithm to the value read from the knob.

The second option allows the operator to make the knob a relative adjustment to the parameter value. The knob value is again operated on by the selected transformation algorithm to determine a setpoint value. Then the parameter value is incremented by the new value.

After modifying the input from the programmable knob in one of the two possible ways, the parameter set command is sent to the external process with no further interpretation.

Command input file commands undergo considerable processing in this module. In addition to processing the files, the nested operations are performed and the conditional commands contained in the input file are evaluated.

Command input file commands perform one of the following functions:

- Start a command input file executing
- Place a command input file in single-step mode
- Place a command input file in test mode
- Issue the next command when in single-step mode
- Enter learn mode

Commands to execute a file simply invoke the file by name. Similarly, commands to place the execution of a file in test mode or single-step mode simply name the file. Note that a single console can be executing more than one command input file at the same time, hence the file name must always be specified. Note also that a command input file can be in single-step mode and test mode at the same time.

When a file is in single-step mode, the execution of each command is delayed until the operator acknowledges a prompt. Following the completion of a command, the next command is displayed as a prompt to the operator.

An additional capability provided for the operator's convenience is the learn mode. After placing the console in learn mode, he then issues process commands as if he were performing an operation. The commands are intercepted and used to build a command input file. This feature allows him to build command input files to assist him in performing his tasks. Learn mode does not allow the operator to execute conditional commands nor does it support nesting of files.

The interface with the external process is the COMMAND PROCESSOR module. It not only sends out the process commands, but also receives data for display purposes or as input for an internal decision. The commands are sent and received as ASCII character strings through a pair of circular buffers.

The command types sent to the process include parameter change commands, parameter value requests, and history requests.

Parameter change commands can be either absolute or relative as previously discussed and are used to set a specified parameter such as a process set-point. Parameter value requests are used to request data for display to the operator or to resolve a conditional command in a command input file. In the case of a conditional command, the value is requested once and sent once. In the case of display data, the request can specify an update frequency such that the request is made once, but the data is sent at specified intervals. The reporting frequency of each parameter is a user selected value, allowing the user to optimize the update time of each parameter.

In order to minimize the communications burden between the operator interface and the external process, data is only reported when a change occurs. Therefore, the reporting frequency specified serves to set the maximum reporting rate rather than an absolute rate.

History requests are made to acquire trend data on a specified parameter for presentation to the operator. This feature of the interface assumes the existence of a data storage and retrieval function in the external process.

The external process returns either acknowledge responses, alarm responses, data responses, or history responses.

Parameter change commands are acknowledged when they are written to the circular buffer unless they were issued from a command input file. In that case, they are not acknowledged until the command has been successfully completed. Requests for process or history data are not acknowledged.

until the data is provided. Requests for periodic data reporting are only acknowledged when the first set of data is received.

Warning and alarm conditions are determined by the external process. When an alarm report is received, the COMMAND PROCESSOR module inserts it into an alarm file for processing. All displays have a section reserved for brief alarm messages. Further detail is provided on alarm summary displays generated by the user. Since the alarm message contains the time of the alarm, the alarm file is maintained in chronological order based on alarm time rather than time received. This eases the burden on the user in displaying information on the alarm sequences.

Responses to data requests are handled in one of two ways, depending on the reason the data was requested. Data which was requested to resolve a conditional command is sent directly back to the DISTRIBUTE COMMANDS module and is not stored internally.

All other data is placed into the internal variables table. The parameter correlation file is accessed to determine the names of the internal variables currently assigned to that parameter. The parameter value is then placed in the value field of those internal variables. Since the periodic data reporting from the process is performed on an exception basis, a flag is set in the internal variables indicating that the variable has changed value.

Data provided in response to a history request is placed into a temporary file called the history buffer. This history buffer is then used by the FORMAT DISPLAY module to present the trend data to the operator.

The FORMAT DISPLAY module issues the displayable commands to the display device software driver. It updates display-dependent files and tables, processes alarm and history data, and updates the display.

When a command invoking a new display is received, several functions are performed. The display file is accessed to update the parameter correlation file and the screen vectors previously discussed. In addition, the current internal variables table is updated. The internal variables table, unlike the parameter correlation file, only contains entries for the variables currently being displayed. Thus, when a new display is invoked, the entries must be updated to correspond with the new display. The internal variables table contains all information needed to display the condition of an external parameter, including the type of display symbol, current value, and alarm status.

Once the files and tables are updated, the background file is accessed to retrieve the proper background for the new display. The background display contains all static information presented on the current display.

If a display was invoked to present trend data on a parameter value, then the display command is processed further following completion of the update functions. The history buffer is accessed to retrieve the data relating to the parameter name specified. This data is then normalized for display to the operator and the internal variables table is loaded with the history data. Each reading contains the normalized parameter value and the time the reading was taken.

All other processing in the DISPLAY FORMAT module takes place on a periodic basis rather than in response to a display command. The internal variables table is scanned for a flag indicating that a variable has changed value. Once that flag is found, the value of the variable is read along with the symbol name. The flag is cleared and the information processed to update the display. The symbol library is accessed to determine the commands required to display the condition of the variable. These commands are then sent to the GRAPHICS DISPLAY DRIVER module for processing.

Additional asynchronous processing takes place to process alarm conditions. When a new entry is found in the alarm file, the alarm information is inserted into the internal variables table and the alarm correlation file is accessed. This file contains additional information to be added to the screen vectors in the event of an alarm. These new screen vectors are user generated prompts for additional alarm information. For example, if an alarm is received on an external parameter, new pushbuttons may appear on the display device or the alphanumeric monitor to prompt the operator to call up certain key displays. These key displays might include the control loop parameters for the affected loop, the trend data for the affected parameter, a display summarizing all similar parameters in the process or an alarm summary display.

The graphics display driver implements a standard set of displayable commands for the graphics display device. This standard set of commands is sufficient to generate all figures needed for operator communication. In order to improve the portability of the software, a single set of commands is used in all libraries. When the physical device is selected, the user must provide for the translation of the standard commands into device specific commands. For a sophisticated device, this consitutes a simple table look-up while less capable devices could require extensive processing.

All coordinate references assume a resolution of 1024 horizontal(X) by 1024 vertical(Y) display elements. This was felt to be the maximum resolution which would be required for most applications. Consequently, the driver module must also scale all coordinates to the actual resolution of the hardware.

The following commands constitute the set of displayable commands implemented in the operator interface. Since the software does not interpret the commands, the creation of additional commands is simply a matter of adding them to the user defined files. This action would

result in restricted portability however.

BCOLOR (NAME) - Sets the current display background color to one of the eight colors specified by NAME (BLACK, BLUE, CYAN, GREEN, MAGENTA, RED, WHITE, YELLOW)

BLINK(STATUS) - All further information plotted will blink if STATUS is on and will not blink if STATUS is OFF

CIRCLE (X,Y,RADIUS) - Draws a circle with the specified radius centered at (X,Y)

CURSOR (X,Y) - Moves the graphics cursor to the point (X,Y)

DEVICE (NUMBER) - Directs all further graphics commands to the display device with the logical unit number NUMBER

FCOLOR (NAME) - Sets the current display foreground color in the same manner as BCOLOR sets the background.

FILL (X,Y) - Fills the polygon containing the point (X,Y) with the current foreground color

HBAR (XO,YO,X,WIDTH) - Plots a horizontal bar graph WIDTH units wide, starting at the point (XO,YO) and extending to the point (X,YO) in the current foreground color

POINT (X,Y) - Plots the point (X,Y) in the current foreground color

RESTORE (FILE) - Performs a DMA transfer of a previously saved display image, named FILE, from the mass storage device to the memory of the graphics display device

SAVE (FILE) - Performs a DMA transfer of the current display image from the memory of the graphics display device to the mass storage device under the name FILE

SYMBOL (X,Y,TYPE) - Draws a symbol defined as TYPE referenced to the point (X,Y) in the current foreground color

TEXT (X,Y,STRING) - Displays the character string STRING beginning at the point (X,Y) in the current foreground color

TONE (TIME) - Generates a tone from the display device lasting for TIME milliseconds

VBAR (XO,YO,Y,WIDTH) - Plots a vertical bar graph WIDTH units wide, starting at the point (XO,YO) and extending to the point (XO,Y) in the current foreground color

VECTOR (XO,YO,X1,Y1) - Generates a vector from the point (XO,YO) to the point (X1,Y1) in the current foreground color

VIDEO (STATE) - Special command used for black and white monitors to set the video to the condition of STATE (NORMAL or REVERSE)

Conclusion

A need exists for a general purpose operator interface which is usable in a variety of applications with little or no modification. This paper describes the implementation of one approach to providing that interface through the use of color graphics display devices with touch sensitive screens.

The interface involves a portable software package designed to support a variety of hardware options. The approach taken provides the user with ease of implementation, both in terms of installation in his environment and the application programming. Machine dependent features of the software are well isolated to simplify modifications required by a new environment. Applications programming is performed by constructing simple source files defining the displays and interactions.

The implementation described results in a compact, efficient control console. The anticipated savings, both in hardware requirements and reduced design and implementation effort, over a conventional control scheme are substantial.

The approach taken results in a minimum configuration requirement. This minimum configuration makes the application of this interface to very small systems expensive. However, the ease of maintenance and modification make it an option worth considering even for those small systems.

The speed of response of the system has not been measured since the interface is still being implemented. It was designed for acceptable response time and no major problems are anticipated. A follow-on effort is planned, in any event, to improve the response time.

A set of file preprocessors is planned which will translate the user's source files into a format which can be processed more quickly. The user would not lose the simplicity of constructing source files but would gain an improvement in system operation. A similar preprocessor could be developed to eliminate the need for the GRAPHICS DISPLAY DRIVER module. It would replace the standard displayable commands in the source files with device specific commands. This would eliminate the need for on-line translation.

Another option which will be explored is the possibility of imbedding several microprocessors in the operator's console. Each microprocessor would perform the functions of one of the software modules, achieving an improvement in response time due to the resulting parallel processing.

BIBLIOGRAPHY

Blar, D. E.; "Plant Operator's Computer Interface," Instrumentation Technology, October 1975, pp. 29-34.

Brewer, S. C.; "Interactive Graphics: A Powerful Tool Supplants Large Control Panels," Control Engineering, September 1978, pp. 54-57.

Crook, K.; "CRT Touch Panels Provide Maximum Flexibility in Computer Interaction," Control Engineering, July 1976, pp. 33-34.

Dallimonti, R.; "Future Operator Consoles for Improved Decision-making and Safety," Instrumentation Technology, August 1972, pp. 23-28.

Dallimonti, R.; "Human Factors in Control Center Design," Instrumentation Technology, May 1976, pp. 39-44.

Dallimonti, R.; "New Designs for Process Control Consoles," Instrumentation Technology, November 1973, pp. 48-53.

Dallimonti, R.; "Operator Interfaces: Past, Present, and Future," Advances in Instrumentation, Instrument Society of America, 1977.

Uyetani, A.; Yoshizaki, K.; Nagakawa, K.; "Standardized Operator Interfaces for Distributed Control Systems," Instrumentation Technology, December 1978, pp. 43-47.

Zey, R. B.; "Using Interactive Color CRTs as Operator Interfaces," Instrumentation Technology, December 1978, pp. 49-51.