SLAC-PUB-2443
December 1979
(M)

THE SAMAC PROGRAM - THE COMPUTER SUPPORT * FOR A STAND-ALONE MONITORING AND CONTROL SYSTEM

By C.A. Logg
Stanford Linear Accelerator Center
P.O. Box 4349
Stanford, California 94305

ABSTRACT

The high energy physics experiments at SLAC require constant monitoring and control of the numerous components contained in the particle detection apparatus. This paper describes a basic hardware configuration and operating system which have been designed and implemented to satisfy the monitoring and control requirements of the many different setups used in these high energy physics experiments. It is based on the LSI-11 microprocessor with up to one million words of RAM and EPROM which are interchangeably mappable into the normal LSI-11 RAM/EPROM address space of 28K words. The entire system is modular in hardware and software so that it can easily tailored be to an individual experiment. The human interface is such that little training is required for effective use of the system. Since the items monitored include gas pressures in detectors, temperatures, and detector voltages which may have a bearing on the final analysis of the data taken during the experiment, these readings are communicated to the central data acquisition system (usually a VAX 11/780 or a SIGMA 5) so that they can be logged along with the data.

INTRODUCTION

In the high energy physics experiments which are done at SLAC, there are numerous factors such as voltages, pressures, and temperatures which must be continuously monitored. The stability and values of these factors are crucial to the experiments, and in some cases may have an influence on the final analysis since they may determine the characteristics of the various detectors. A monitor chassis (1) and various cards for digitizing the voltages, pressures and temperatures have been designed at SLAC. This paper describes the computer hardware and basic operating system software of an LSI-11 based computer system which has been designed and implemented to facilitate the monitoring and controlling of the various voltages, pressures and temperatures for which the monitor chassis provides an interface. The readings are also made available to the host experiment data acquisition computer (usually a VAX) so that it can record any information which may be relevant in the final analysis.

The design goals have been to provide a system that:

++ has an easy to use human interface.

++ can be easily tailored for individual setups.

++ has an expandable data base.

++ can provide fast response for interrupts that might be generated by fault conditions.

++ could be developed in a about six months with existing software development facilities.

++ is fully functional in a stand alone mode.

++ has multi-terminal support.

++ provides some protective control over the issuing of commands so that contradictory commands can not be issued by different terminals.

*This work was supported by the Department of Energy under contract number DE-AC03-76SF00515.

(Contribution to the 1979 Fall DECUS U.S. Symposium, San Diego, CA, December 10-13, 1979.)

++ is low in cost.

++ includes facilities that experimenters have found useful in previous LSI-11 systems.

In general, during checkout, the experimenter wants to look at displays of data on command, then maybe adjust the hardware, reread the data, and display it again. During the experiment, there may be numerous readings which are to be taken on a timed basis (every N seconds, minutes or hours), checked for tolerance, and displayed on command. All reading addresses, tolerance limits, and settings must be interactively changeable. In many cases, the user may be interested in a function of some reading rather than the actual reading. The following sections describe a package which is intended to satisfy these monitoring needs. It also has been designed so that control subsystems can easily be interfaced.

LSI-11 HARDWARE CONFIGURATION

The LSI-11 hardware we have put together has a minimum of two 32K word Digital Pathways (2) RMA-Ø32 random access memory (RAM) boards, one 16K word Digital Pathways RMS-Ø16 EPROM board, a DLV11-J, a Standard Engineering Corporation CCLSI-11/A CAMAC Crate Controller and LSI-11 interface (3), an LSI 11/2 processor, and a Digital Pathways (BSC-256) Bank Switch Controller for handling the RAM and EPROM memory. By adding additional 32K RAM boards or 16K EPROM boards we can further expand the memory as needed.

The LSI-11 system is connected (Figure 1) to the monitor chassis via two CAMAC modules: a Kinetic Systems 12-bit A/D Converter (model 3553) (4) and a Kinetic Systems 36Ø1 Input/Output register (IGOR) (5). The IGOR module is used to send commands to the monitor chassis and read digital data. The A/D Converter is used to digitize the analog data which is retrieved from the monitor chassis.

The experimental data acquisition computer, usually a VAX, has access to the same CAMAC crate via a CAMAC CCA-2 crate controller. A 4K Kinetic Systems 3821 RAM memory module (6) is also in the CAMAC crate. The LSI-11 can receive commands from the VAX either via this memory module or via a serial line as a virtual terminal to the VAX. Any data which must be sent to the VAX is transferred via the RAM memory.

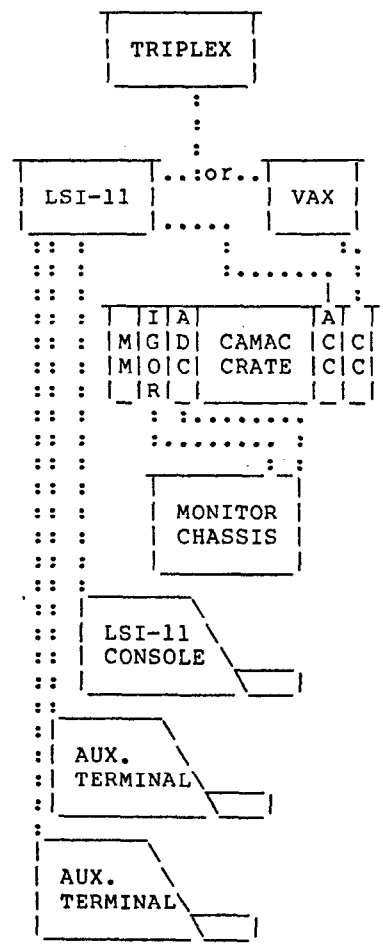

Figure 1

The basic configuration includes an LSI-11 with an auxiliary CAMAC crate controller, a DLV11-J, a CAMAC crate, and the monitor chassis. The DLV11-J is used to connect up to 3 terminals, and for a virtual terminal connection to either the expermental data acquisition computer (usually a VAX), or the central SLAC computer complex (the TRIPLEX).

Minimally the CAMAC crate contains:

MM - the 4K Kinetic Systems RAM module
ADC - A/D Converter
IGOR - Input/Output module
ACC - LSI-11 auxiliary crate controller
CCA-2 - the VAX crate controller

Nominally, up to 3 terminals can be attached, although with the addition of more DLV11-J boards and a slight system generation modification more terminals could certainly be used.

The LSI-11 RAM and EPROM is divided into 4K word banks. Each bank is given a unique number in the range of 0-$377 octal (Note that $NNNN is used to indicate an octal number). For convenience, the EPROM banks are numbered from $377 octal down and the RAM banks are numbered from 0 up. The number is actually strapped by jumpers onto the memory boards. The bank switch controller has 8 16-bit registers which we have located at $170600-$170616 in the I/O page. These registers are used to map into the normal 28K address space of the LSI-11 the various RAM and EPROM banks. The register at $170600 holds the number of the memory bank which responds to the addresses (0-$17776). The memory bank number contained in the register at $170602 will be the one used when an address in the range ($20000-$37776) is accessed. Similarly

$170604 maps addresses ($40000,$57776)

$170606 maps addresses ($60000,$77776)

$170610 maps addresses ($100000,$117776)

$170612 maps addresses ($120000,$137776)

$170614 maps addresses ($140000,$157776)

RAM and EPROM are not mapped into the address space reserved for the I/O page.

These registers are all under program control. The BSC controller has on the board a 82S123 ROM which contains a power-up default configuration. Bit 15 of the register located at $170616 determines whether the default ROM located on the BSC controller, or the registers in the I/O page determine the mapping. On power-up, bit 15 of $170616 is always 1. It can be set under program control to 0 to make the I/O page registers the effective map controller.

For the implementation of the software for this system, the following mapping scheme (Figure 2) has been chosen: continually resident RAM is present in the range 0-$60000 for the program area which must be resident. That is, registers $170600, $170602, $170604 will always contain 0, 1, and 2. Register $170606, which controls the address space ($60000-$77776), is used for various overlayable EPROM banks. These EPROM banks contain pieces of the SAMAC code which can be executed out of EPROM and 'overlayed'. The register $170610, which controls the address space ($100000-$117776), is used for RAM banks which are overlayed. These are RAM banks which a software memory (buffer) manager manages as a buffer pool. The register $170614 contains the bank number of a general 4K EPROM kernel. The contents and significance of this EPROM kernel will be discussed later. It should suffice to say for now that the kernel contains some very general stand-alone system support routines. The register $170612 controls a continuously resident RAM bank which is used for the system stack and various buffers and pointers which the EPROM kernel uses. Although this full 4K is not needed for the EPROM kernel support, partial usage of this block by the stand alone system (other than as stack space) is more trouble than it is worth.

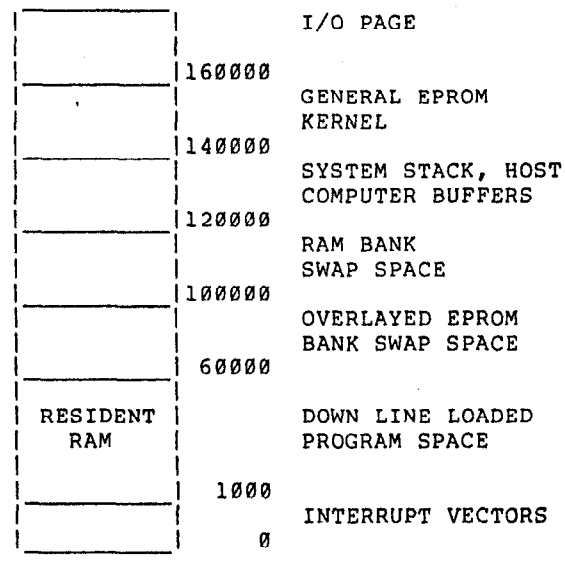

Figure 2

Diagram of LSI-11 address space usage.

SOFTWARE DEVELOPMENT

The software for this system has been developed by using the existing software development facilities (7) available on the large SLAC IBM computer complex (8), called the TRIPLEX. These facilities include a PL-11 cross compiler (9), a cross linker (10), the WYLBUR text editing system (11), IBM OS data sets, and the lineprinters. The programs are edited, compiled, and linked together on the TRIPLEX to form an absolute load module. This load module is then down line loaded into the LSI-11 resident RAM memory by the down line loader contained in the EPROM kernel.

The parts of the stand-alone system which are contained in the overlayed EPROM banks are created in a similar fashion. There

is one absolute load module for each 4K EPROM bank. The only difference is that instead of being down line loaded, the absolute load module is burned into EPROMs which are placed in the 16K EPROM board.

Linkage between subroutines which are contained in the stand alone system EPROMs (henceforth referred to as SAS-EPROMs) and subroutines contained in the resident RAM, subroutines contained in one SAS-EPROM set and another SAS-EPROM set, and subroutines contained in the resident RAM and any of the SAS-EPROM sets, is done by TRAP calls. Each separate load module in the stand-alone system contains a TRAP address table. The addresses of any routines in a load module which may be accessed by a TRAP instruction are loaded into the TRAP address table contained in that module. In the case of an SAS-EPROM set, if a given TRAP routine is not contained in that set, then the EPROM bank number where it is contained is loaded; or, if the routine is in RAM, the TRAP table entry is flagged to indicate that.

Linkage between the stand-alone system code and the EPROM kernel is done by EMT calls.

THE EPROM KERNEL

The EPROM kernel contains facilities used in the software development: a terminal emulator and a down line loader. It also contains various routines which are used by all stand-alone systems such as binary number to ASCII character conversion, ASCII character to binary number conversion, and input and output routines. Other miscellaneous facilities include a FORTH bootstrap, RT-11 bootstrap, floppy disk read/write routines, and a memory test.

SOFTWARE DESIGN PHILOSOPHY

The overall design for the SAMAC software has relied heavily on the fact that the various factors which are to be monitored or controlled can be grouped into small logically related groups. For example, one group may be composed of CAMAC crate voltages; another group may contain Hall Probe readings. Groups are defined by specifying a unique group number, the maximum number of elements (henceforth referred to as variables although these may be settings or readings) which will be defined in that group, and a title for the group. The individual variables in a group each have various subparameters associated with them: high (HI) and low (LO) tolerance limits, a CAMAC address (CA), a monitor chassis address (CH), a scale factor or ADC gain (SC), a setting (SE), and, optionally, an expression set (which will be discussed later).

The basic operating system software has been designed and implemented in a highly structured and modularized fashion. The main routine, SAMAC, calls the system initialization routine, SYSINI, which calls the initialization routines for all of the subsystems which make up the basic operating system. SAMAC then calls a routine called UINIT, which is provided individually for each system. UINIT should initialize any special purpose subsystems which may have been written to expand the basic system and provide additional functionality. It can also do group, display, variable, expression, or histogram definitions. SAMAC then goes into a loop which calls a routine named BKGRND. BKGRND continuously checks a background queue for any commands which may have been posted to it. The BKGRND routine is one basic system routine to which the user may wish to make additions when he tailors a SAMAC system.

Once the system has been initialized, all actions are generated by interrupts. The terminal keyboards are all interrupt driven. When a complete command has been entered (break or <CR> is a terminator), it is passed to the command subsystem for processing. Other actions can be generated by the clock subsystem and CAMAC system.

The entire SAMAC software package is made up of many functionally independent subsystems; although some do use the facilities of the others. The following sections describe in detail the various basic subsystems and the facilities they provide.

GENERAL SUBSYSTEM STRUCTURE

All of the various subsystems in the SAMAC software have the same basic structure.

++ Each subsystem has an initialization routine. The initialization routine posts to the command processor the address of a routine which will process commands relevant to that subsystem. It also posts to the HELP command processor the address of a routine which will print, on a specified unit, a description of the subsystem commands. The initialization routine ends with the characters INI.

++ A subsystem command processor is provided to handle any commands associated with a subsystem. An example is shown in Figure 3. There are two parallel tables in a command processor: one contains the command strings and the other the addresses of the routines to be called when the respective command string is found. This information is passed onto the command search routine (CMDSER) which

```
^TITLE  SYSCMD - DEBUG COMMAND PROCESSOR
GLOBAL PROCEDURE SYSCMD (R5) STACK;
BEGIN
"

 Calling sequence:
 EXTERNAL PROCEDURE SYSCMD STACK;
 INTEGER UNIT=$177560,OK;
 ARRAY BYTE EXCMD=('TRACEON',$15);
  .
  .
  .
 SYSCMD(REF(OK),REF(EXCMD),REF(OK));

This routine contains the tables for  some
of the basic system commands.

"
ARRAY BYTE CMDLIST=('HELP@     '
                   ,'HALT@     '
                   ,'TRACEOFF@ '
                   ,'TRACEON@  '
                   ,'SYSHELP@  '
                   );

EXTERNAL PROCEDURE TRACEO,TRACE,HELPQ,
                   STOP,SYSHLP;

ARRAY INTEGER CMDPRO=(REF(HELPQ)
                     ,REF(STOP)
                     ,REF(TRACEO)
                     ,REF(TRACE)
                     ,REF(SYSHLP)
                     );

INTEGER NCMD=LENGTH(CMDPRO)/2;

EXTERNAL PROCEDURE CMDSER STACK;

"arguments"

INTEGER UNITADD SYN MEMORY(R5+2),
        CMDSTR  SYN MEMORY(R5+4),
        OKADD   SYN MEMORY(R5+6);

CMDSER(REF(NCMD),REF(CMDLIST),REF(CMDPRO),
       CMDSTR,UNITADD,OKADD);

RETURN;
END.
```

Figure 3

Example of a subsystem command processor.

utilizes the information to process commands. Since the command processing is done by the command subsystem, standard calling sequences must be utilized by the routines which are called. The command processor ends with the characters CMD.

++ A subsystem help routine is supplied. It is called in response to the command HELP, and it lists a description of various subsystem commands available. An example is shown in Figure 4. The help processor ends with the characters HLP.

```
^TITLE  SYSHLP - BASIC SYSTEM HELP COMMAND
GLOBAL PROCEDURE SYSHLP(R5) STACK;
BEGIN

"
Calling sequence:
EXTERNAL PROCEDURE SYSHLP;
INTEGER UNIT=$177560;
.
.
.
SYSHLP(REF(UNIT));

SYSHLP prints on the specified UNIT a
description of the system commands
available.

"
EQUATE CR SYN $15, LF SYN $12;

INTEGER UNITADD SYN MEMORY(R5+2);

ARRAY BYTE SYSHELPLIST=(
 'HELP - PRINTS THIS LIST',CR,LF,
 'HALT - ODT HALT',CR,LF,
 'TRACEON - SET A TRACE TRAP',CR,LF,
 'TRACEOFF - CLEAR A TRACE TRAP',CR,LF,
 'SYSHELP - LIST SYSTEM DEBUG COMMANDS',
  0);


EXTERNAL PROCEDURE PRTSTR STACK;

PRTSTR(UNITADD,REF(SYSHELPLIST));

RETURN;
END.
```

Figure 4

Example of a subsystem help processor.

++ The other routines contained in a subsystem are those which are necessary to respond to the subsystem commands. If the subsystem handles any interrupts, then the appropriate interrupt processing routines are also supplied.

With regard to the command processors, any characters following the basic command string are passed on to the routine which is called. For example, when the command

HOUT HID=1, UNIT=LP<CR>

is typed, the routine which is called to process the HOUT command is passed the remainder of the string, that is:

HID=1, UNIT=LP<CR>

It then does its own parsing to pick off the specifications. Various parsing routines are available as part of the basic system.

SYSTEM DEBUG COMMANDS

The system command processor SYSCMD contains facilities to aid in debugging. These commands, with the exception of the HELP command, can only be issued by the console terminal and only if it is in CONTROL mode (discussed in the next section). SYSCMD provides the following commands:

++ HELP - results in a call to all the help routines which have have been posted to the HELP processor.

+C TRACEON - allows the user to turn on a TRACE TRAP when the instruction at a specific absolute memory address is executed.

+C TRACEOFF - clears the last unexecuted TRACE TRAP that was set.

+C HALT - executes a HALT instruction and puts the LSI-11 into ODT. This should be done before turning the CAMAC crate power off.

++ SYSHELP - executes a call to just the SYSCMD help processor.

TERMINAL CONTROL SUBSYSTEM

The terminal control subsystem provides the software facilities for attaching additional terminals to the system. The basic system comes with one terminal attached - the console terminal. Other terminals can either be attached by the UINIT routine, or by keyboard commands. This subsystem also ensures that contradictory commands are not issued simultaneously by two different terminals. Certain commands can only be issued by a terminal which is in CONTROL mode. All active terminals are either in CONTROL mode or MONITOR mode; however, only one is in CONTROL mode at any given time. In the remainder of this paper, in the subsystem command descriptions, CONTROL mode only commands are indicated by '+C' in the left margin. Terminal subsystem commands are:

++ CONTROL - which says to make the terminal issuing the command the CONTROL terminal. If another terminal is in CONTROL mode, then a message is printed on that terminal to indicate that another terminal wishes control. The terminal issuing the command also gets a message stating which terminal port is in CONTROL.

++ MONITOR - causes the issuing terminal to be put in MONITOR mode.

+C DETACH <unitno> - causes terminal port <unitno> to be detached. It can then no longer issue any commands. The LSI-11 console cannot be detached.

++ CONNECT <unitno> - results in the software activation of the specified <unitno>. It is attached in MONITOR mode.

++ TERMHELP - causes the terminal subsystem help processor to print a list of the terminal manipulation commands on the issuing terminal.

THE VARIABLE GROUP MANAGEMENT SUBSYSTEM

A SAMAC system may have a data base containing hundreds of variables. These variables are divided into various groups according to their functions. The group management subsystem handles the manipulation of these variable groups.

GPINIT - The Group Management Initialization Routine

GPINIT is the group management initialization routine. It initializes the group pointer index table so that, initially, no groups are defined. It passes to the command processing subsystem the address of a command processor (GPCMD) which will handle any group management related commands, and to the help subsystem the address of the help processor GPHLP.

GPCMD - Group Management Command Processor

The GPCMD routine processes the group management related commands. Group management commands are:

++ GPDEF - The GPDEF commands results in the allocation of a variable group with space for the definition of up to <# of elements> variables. This command merely allocates the group space. It does not define any of the variables in the group. The syntax is (Note: commands must be typed completely on one line. They are split over several lines only for the formatting of this document):

```
GPDEF GPID=<gpid>
      GPSIZE=<# of elements>
      GPTITLE=<char str>
```

where: GPID is the group identifier. <gpid> is a number in the range 1001 to 1999.
GPSIZE is the maximum number of elements that will be defined in that group.
GPTITLE is a group title.

++ GPLIST - The GPLIST command has 3 (three) different modes. The syntaxes are:

```
GPLIST
```

which lists the specifications (<gpid>, <# of elements>, title) of all defined groups.

GPLIST <gpid>
which lists the specifications of the indicated group, all the defined variables in that group and all of their subparameters.

GPLIST <gpid> <subparameter 1>, ..., <subparameter n>
which lists the specifications of the indicated group, and the specified subparameters of all the variables in that group.

+C GPACT - The GPACT command indicates that a group is to be activated - that is read on a timed basis. The syntax is:

GPACT <gpid> <# of seconds>
which causes all the variables in the group identified by <gpid> to be read every <# of seconds> seconds.

A 60-cycle interrupt-driven clock subsystem is also a part of the system. The GPACT command results in the posting of a read command and time interval to the clock processing subsystem which handles timed events.

+C GPDEACT - The GPDEACT command allows one to deactivate a group so that it is not read. The syntax of the command is:

GPDEACT <gpid>
which causes the read command and time interval to be removed from the clock subsystem queue.

++ GPREAD - The GPREAD command is a one-shot command to the system to read a specified group. The syntax is:

GPREAD <gpid>

+C GPSET - The GPSET command is a one-shot command to the system to do the settings specified for a group. The syntax is:

GPSET <gpid>

+C GPDEL - The GPDEL command results in the deletion of the specified group. After the execution of this command, all of the variables and expressions associated with the group are undefined. The syntax is:

GPDEL <gpid>

++ GPHELP - which results in the printing of a description of the group management commands on the terminal issuing the command.

THE VARIABLE MANAGEMENT SUBSYSTEM

The variable management subsystem provides the actual variable definition and variable parameter setting facilities.

VARINIT - Variable Management Initialization Subsystem

In initializing the variable management subsystem, VARINIT simply posts to the system command processor the variable management command processor VARCMD, and to the system help subsystem processor the variable help command processor VARHLP.

VARCMD - Variable Management Subsystem

VARCMD provides the following variable manipulation commands:

++ VARDEF - The VARDEF command is used to define a variable. The group <gpid> in which the variable is to be defined and the variable name must be specified. A variable name is an up-to-8-character string, containing any combination of numbers (0-9) and letters (A-Z), as long as it contains at least one letter (i.e., for parsing reasons a variable name must not convert to a number). The command syntax is:

VARDEF GPID=<gpid> VNAME=<vname>

+C VARDEL - The VARDEL command causes the specified <vname> to be deleted. The syntax is:

VARDEL <vname>

+C VARSET - The VARSET command allows one to set a subparameter of a variable to a specified value. Subparameters include the high (HI) and low (LO) tolerance limits, a setting (SE), a CAMAC address (CA), and a chassis address (CH). The syntax is:

VARSET <vname> <subparameter>=<num>

++ VARHELP - which results in the printing of a description of the variable management commands on the terminal issuing the command.

THE ALIAS SUBSYSTEM

The alias manipulation subsystem is a facility which was designed to enable users to give names which may make some symbolic sense to the variables which are defined in a SAMAC system. For example the 40 volt monitor card can handle up to 31 inputs. Initially, these are defined with names like 40V0, 40V1, 40V2, 40V3, ..., 40V31. However a user will probably prefer much more descriptive names such as 24CR1, 24CR2 for 24 volts crate 1, 24 volts crate 2. The alias facility enables him to rename these, as desired, without reallocating and/or redefining them.

ALINIT - Alias Initialization Routine

The alias initialization routine posts to the system command processor the address of the alias command processing routine ALICMD. It also posts to the system help routine the address of the alias help command processor ALIHLP. For initialization, it clears the alias table.

ALICMD - Alias Command Processor

The alias command processor handles the following alias commands:

++ ALIAS - which results in <namel> being equated to <name2>. Any time <namel> is referenced, the variable <name2> is actually used. The syntax is:

ALIAS <namel>=<name2>

++ ALIASLIST - which results in the listing of the aliases for <namel>; or, the aliases which lie alphabetically in the range <namel> through <name2>; or, in the listing of the entire alias table.

ALIASLIST <namel>
or ALIASLIST <namel>-<name2>
or ALIASLIST

+C ALIASCLR - which results in the deletion of the specified <namel>; or, in the deletion of all aliases which lie in the alphabetic range <namel> through <name2>.

ALIASCLR <namel>
or ALIASCLR <namel>-<name2>

++ ALIASHELP - which results in the printing of a description of the ALIAS commands on the terminal issuing the command.

THE EXPRESSION SUBSYSTEM

Usually the raw readings do not have immediate significance to an experimenter. When, for example, a temperature probe is read, the degree centigrade to which the ADC reading converts has much more meaning than the raw reading. The user may also be interested in some function of a reading or readings; he may want to accumulate a histogram of the readings; or he may want to keep a running average, or mean and standard deviation of some readings. The expression subsystem is intended to fulfill these needs.

There are three classes of expressions. There is the global expression block, the group-associated expression block, and the variable-associated expression block. Any expression in any block can access any expression result or any variable in the entire system. The three different classes are allowed primarily to make it easier for the user to have some logical association between expressions and what they apply to. For example, the expression which converts an ADC reading for a temperature probe to degrees centigrade belongs in that variable's expression block. An expression which forms the average of the temperature readings in a variable group belongs, logically, in that group expression block. And an expression which operates on variables from several different groups probably belongs in the global expression block.

A histogram identifier can be specified with an expression definition. In that case, the result of the expression evaluation is binned in the specified histogram. It is the user's responsibility to see that the histogram is defined. If it is not defined, no warning is given; the result is simply not histogrammed.

Operands in the expressions can be any of the following:

++ <gpid>:<ind> - This kind of operand is the result of a group associated expression. The result from the last evaluation of expression number <ind> in the group expression block associated with the variable group <gpid> is used as the operand.

++ <vname>:<ind> - This kind of operand indicates that the result from the last evaluation of the expression number <ind> in the expression block associated with the variable <vname> is to be used as the operand.

++ <vname>:<subparameter> - This kind of operand indicates that the specified subparameter of the variable <vname> is to be used as the operand.

++ %:<ind> - This kind of operand indicates that the value resulting from the last evaluation of the global expression number <ind> is to be used as the operand.

++ Decimal integer numbers in the range -32768 through 32767 may be used as operands.

++ Octal numbers, in the range Ø-$177777, may be used as operands.

++ Hexadecimal numbers, in the range Ø-#FFFF, may be used as operands.

Note if the operand is an expression result or a variable subparameter, and if the expression or variable and/or subparameter are not defined, then the evaluation of that expression is aborted and an error message is printed on the console terminal.

Note the following with regard to the evaluation of the expressions: EXECUTION IS STRICTLY LEFT TO RIGHT WITH NO OPERATOR

PRECEDENCE and ALL OPERATORS ARE BINARY OPERATORS.

The following operators are provided:

```
+ addition
- subtraction
* multiplication
/ division
< arithmetic shift left
> arithmetic shift right
^ (carat) maximum function
_ (underscore) minimum function
& logical and
| logical or
```

32-bit intermediate results are propagated if the next operation is an addition, subtraction, or division. For multiplication, if the multiplicand is greater than 16 bits, an overflow is presumed to have occurred and the expression evaluation is aborted.

Conditionals can also be attached to any expression. The operands may be any of the above, and valid conditional operators are:

```
> greater than      < less than
= equal             # not equal
```

The conditionals are also evaluated in strictly a left-to-right fashion. Note that A>B#C is evaluated as A>B and A#C. Multiple conditions may be specified for an expression. If any of them fail, the expression is not evaluated and a zero is saved as its value. If one expression calls for the value of another as an operand, and the operand expression failed one of its conditionals or was aborted during its evaluation, then a zero is used as the operand.

EXPINIT - The Expression Subsystem Initialization Routine

EXPINIT initializes the expression subsystem by clearing the global expression block pointers. It also posts to the command processor the address of the expression command processor, EXPCMD; and the address of the expression help processor, EXPHLP, is posted to the system help processor.

EXPCMD - The Expression Command Processor

Expression management commands are:

++ EXPDEF - which defines an expression. EXPDEF has three diferent forms:

Global expression block expression definition:
EXPDEF IND=<ind>, HID=<hid>,
EXP=<expression>

Group expression block expression definition:
EXPDEF GPID=<gpid>, IND=<ind>,
HID=<hid>, EXP=<expression>

Variable expression block expression defintion:
EXPDEF VNAME=<vname>, IND=<ind>,
HID=<hid>, EXP=<expression>

In the expression definition, the <expression> has the following form:

(<condition>)<arithmetic expression>

For example:

EXPDEF IND=1, HID=5,
EXP=(40V0:RE>10)(40V1:RE>10)40V0+40V1/2

This example indicates that expression number 1 in the global expression block is to be: if the reading associated with the variable 40V0 is greater than 10, and the reading associated with the variable 40V1 is greater than 10, then take the average of the two, and histogram that result in the histogram which has the identifier 5.

Note: if just a variable name is given, the subparameter RE (the raw reading) is assumed.

++ EXPHELP - lists an explanation of the expression commands on the terminal issuing the command EXPHELP.

+C EXPCLR - Stores zero for the result of the specified expression(s). Since the result of an expression evaluation can be used as an operand in another expression, this command is provided so that the user can set a result to zero.

++ EXPLIST - Lists the specifications of the indicated expression(s) and the result from the last evaluation.

+C EXPDEL - Deletes the indicated expression(s). The buffers which were used for holding the expression are returned to the system buffer pool.

++ EXPEXEC - Results in the evaluation of the specified expression(s).

The expression specifications for EXPCLR, EXPLIST, EXPDEL, and EXPEXEC are exactly the same.

For the entire global expression block:
EXPCLR IND=0 | ALL

For a specific expression in the global expression block:
EXPCLR IND=<ind>

For an entire group expression block:
EXPCLR GPID=<gpid>

For a specific expression in a global expression block:
EXPCLR GPID=<gpid> IND=<ind>

For a variable expression block:
EXPCLR VNAME=<vname>

For a specific expression contained in a variable expression block:
EXPCLR VNAME=<vname> IND=<ind>

THE HISTOGRAM SUBSYSTEM

Histograms provide a convenient way of accumulating and displaying data distributions. Up to 999 histograms (ID's range from 1 to 999) can be defined at any given time. For each, an identifier, the number of bins, the low bin, the bin width, and a title must be specified.

HINIT - The Histogram Subsystem Initialization

HINIT clears the histogram pointer table, posts the address of the histogram subsystem command processor, HISCMD to the system command processor, and posts the address of the help processor, HISHLP, to the system help processor.

HISCMD - The Histogram Command Processor

HISCMD provides the following histogram manipulation commands:

++ HDEF - which facilitates run-time histogram definition. The syntax is:

HDEF HID=<hid>,LOW=<low>,WIDTH=<width>,
NBINS=<#bins>,
TITLE=<up-to-80 characters>

where:
<hid> is a number in the range 0-999, which is an identifier for the histogram.
<low> is the lowest bin edge.
<width> is the bin width.
<#bins> is the number of bins.
<up-to-80 characters> is the title of the histogram.

+C HCLR - zeroes the bins of a histogram.

+C HDEL - deletes a histogram definition and returns the buffer which was used for the specifications and bins to the central buffer pool.

HDEL and HCLR have the same calling sequence:

HCLR <hid> or HCLR ALL
where: ALL specifies that all histograms are to be cleared or deleted.

++ HLIST - lists the specifications of the indicated histogram(s). The syntax is:

HLIST <hid> <unit#>

++ HSTAT - calculates various statistics and prints them, along with some additional information. Printed are: HID, LOW, WIDTH, # of calls, # of underflows, # of overflows, sum of the histogram bins, mean and standard deviation. A range can be specified so that the statistics can be calculated on a selected portion of a histogram.

++ HOUT - displays a histogram. A range can be specified so that only a portion of a histogram is displayed.

The syntax for HOUT and HSTAT is the same:

HOUT <hid>

HOUT <hid>, FIRST=<bin#>, LAST=<bin#>

++ HHELP - lists a description of the histogram subsystem commands on the terminal issuing the command.

The basic operatins system's only use of the histogram facility is made by the expression subsystem. However, the full power of the histogram facility is available to any user subsystems.

THE DISPLAY SUBSYSTEM

Although the variables are already grouped, the grouping may not be the one the experimenter wants to display. He may prefer various other combinations, or he may want to look at transformations of the raw readings rather than the raw readings. It is this need that the display subsystem is intended to fulfill. This display subsystem allows the user to define displays which he can later display on command. Each display is identified by a unique number. The elements of a display can be any of the operands which are accepted by the expression subsystem. That is, the display elements can be variable subparameters or the results of expressions. The display is done as a bar histogram.

DSPINI - The Display Subsystem Initialization Routine

DSPINI intializes the display subsystem by clearing the display definition pointer table. It also posts the command procssor (DSPCMD) and the help processor (DSPHLP) to the basic system command processor and help processor respectively.

DSPCMD - The Display Subsystem Command Processor

DSPCMD provides the following commands:

++ DISPDEF - which does the display definition. The syntax is:

DISPDEF DID=<did>,
<element 1>, ..., <element 2>

++ DISPLAY - which causes the indicated display to be made. If a time interval is specified, then the display is updated according to the specified time interval.

DISPLAY DID=<did>, <# of seconds>

++ DISPDEL - deletes the specified display definition. The space used to store the definition is returned to the central buffer pool. The syntax is:

DISPDEL <did>

++ DISPLIST - which causes the defintion of the indicated display to be listed on the issuing terminal.

DISPLIST <did>

++ DISPHELP - prints a list of the display subsystem commands on the terminal issuing the command.

HOST COMPUTER SERIAL LINE COMMUNICATION SUBSYSTEM

During development of the software it became convenient to have a file of configuration commands (GPDEFs, HDEFs, VARDEFs, VARSETs, and EXPDEFs) on the host computer (in that case, the TRIPLEX). After the system is loaded, commands can be read off of the TRIPLEX line and executed just as if they came from the keyboard. This enabled us to create a data set with a wide variety of commands which could be used to test the various command facilities. Also, since the system may be used in different hardware configurations to monitor different sensors, it is convenient to have stored on a host computer files of commands which can be sent to configure the SAMAC system in different ways. In order to load commands from the host computer over the serial line (the LSI-11 is basically a virtual terminal on the host computer system), the LSI-11 console must be the CONTROL terminal. All commands received from the host computer are then printed on the console terminal along with an execution code. If the code is 'OK', then the command was executed properly. If it is anything else, then an error occured.

A specific protocol has been established for the host computer to LSI-11 communication. Normally, the host serial line communication subsystem prints on the console terminal any characters received from the host computer. Any time the host computer goes into a state where it can receive LSI-11 transmissions, it sends a 'DC1'. The LSI-11 then sends whatever is to be sent until a terminating character (a break, carriage return, or '<ctrl>D' is encountered). It then waits until it receives another 'DC1' ('<ctrl>Q') from the host computer before it sends any more.

Any commands issued by the console terminal which are not recognized by any of the SAMAC command processors are sent on to host computer, if the host computer communication is enabled.

HOSINIT - Host Serial Line Communication Initialization

HOSINIT initializes and enables the host serial line communication subsystem. Host computer serial line communication can only come in on the DLV11-J port at $176510 and interrupt vector $310. If host communication over the serial line is not desired, then the user should type HOSTOFF to disable the communication.

HOSCMD - Host Serial Line Communication Command Processor

HOSCMD handles the commands for this subsystem. The commands provided are:

++ HOSTOFF - disable host serial line communication.

++ HOSTON - enable host serial line communication.

++ HOSTABORT - abort the loading of commands from the host computer.

++ HOSTHELP - list the commands associated with the host serial line communication subsystem.

INTERFACING SPECIAL PURPOSE SUBSYSTEMS

The basic SAMAC operating system is a monitoring system. Control systems must be tailor-made to individual specifications. The basic operating system has been carefully designed to facilitate the addition of individually tailored subsystems.

Basically, to add a subsystem, all the user has to supply is:

++ An initialization routine which does any special initialization. The initialization routine must post to the basic system command processor a command routine with the same structure shown in Figure 3, and, to the basic system help processor, the address of a help processor with the structure shown in Figure 4.

++ The command processor.

++ The help processor.

++ Any routines needed to respond to the various subsystem commands or interrupts which the subsystem uses.

Any of the facilities used and/or provided by the basic stand alone system can be used by the user's subsystem. These include:

++ Various parsing routines.

++ The clock subsystem.

++ The background queue. The user would have to provide his own background routine, BKGRND, which, most likely, would be a modified version of the basic system background routine.

++ All of the expression, group management, variable management, alias management, histogram, host computer, and display facilities. The subsystems can be utilized either by calls from the subsystem code or by the terminal or host computer commands.

++ Buffer management subsystem.

++ Monitor chassis access routines.

++ Any of the facilities contained in the EPROM kernel.

USEFUL IMPROVEMENTS

As the first system was delivered, it became clear that it would be convenient to have a floppy disk or TU58 cartridge tape for backup resident RAM loading, and to hold files of commands for generating various configurations of variable, display, expression, and histogram definitions. At some point a subsystem for handling these needs will probably be added.

Another useful addition would be some form of cheap hardcopy lineprinter, although the hardcopy need can now be fulfilled by putting a hard copy terminal on one of the terminal ports.

CONCLUSIONS

The real, extensive field testing of this system is just starting. By the end of January, some real feedback and major control subsystem tailoring requests will be implemented. In the extensive development phase and limited testing which has been done, the system hardware and software appear to be adaquate and reliable. Out of the 64K words of RAM in the system, 48K words are available in the form of buffers for the various histogram, variable, group, expression, and display definitions. If needed, more 32K RAM banks can be added. Exactly how many variables can be monitored and controlled by the system will depend on the frequency with which readings are made and expressions are evaluated. Carrying around extensive definitions of groups, variables, expressions, displays, and histograms costs nothing in CPU time or program space. The practical limit of the maximum number of variables which can be handled by the CPU is not yet known. If a limit is reached in a practical application, the system can be upgraded with an LSI-11/23. The RAM and EPROM boards and the bank switch controller are compatible with an LSI-11/23, as, hopefully, is the code.

ACKNOWLEDGEMENTS

I want to thank Patrick Clancey for his extensive work on the command parsers and documentation; John Kieffer, Ray Larsen, and Leo Paffrath for their explanations of the monitor chassis hardware and suggestions on facilities needed in the software; Michael Stoddard for his research and development of the LSI-11 components and configuration, which have made this implementation possible; and all of the above for their encouragement, many helpful hints, and enlightening discussions.

REFERENCES

1. Monitor Chassis 1 System Description, Preliminary Technical Note, John Kieffer, SLAC, Stanford University, Stanford, California.

2. Digital Pathways Bank-Switchable Memory Family for DEC LSI-11 Based Computers, Digital Pathways Inc., Palo Alto, California.

3. CC-LSI-11/A User's Manual, Standard Engineering Corporation, Fremont, California.

4. Kinetic Systems Model 3553 12-bit A/D Converter, Kinetic Systems, Lockport, Illinois.

5. Kinetic Systems Model 3061H/T Input Gate/Output Register, Kinetic Systems, Lockport, Illinois.

6. Kinetic Systems 3821 4K 16-bit RAM Memory Module, Kinetic Systems, Lockport, Illinois.

7. R. L. A. Cottrell and C. A. Logg, An IBM 370/36 Software Package for Developing Stand-Alone LSI-11 Systems. Proceedings of the Digital Equipment Computer Users' Society, Vol. 4, No. 4, pp. 985/991, April 1978.

8. SLAC Computer Services User Note No. 99, The TRIPLEX Users Guide, June 1978.

9. Robert Russell, PL-11: A Programming Language for the DEC PDP-11 Computer, Edited by T. C. Streater, CERN 74-24 (1974).

10. S. Steppel and H. E. Syrett, XASM11/XLINK11, A PDP-11 Cross Assembler/Cross Linker User's Manual, CGTM No. 160 (1974), SLAC, STANFORD, CA.

11. WYLBUR/370, The Stanford Timesharing System Reference Manual, Third Edition, November 1975.