Talking to CAMAC through VxWorks.

By Robert Millner
Sept. 30, 1993

This is intended to be a quick tutorial and example guide to controlling a standard CAMAC crate from a VxWorks terminal. The examples were written for an MVME 162-12 controller interfaced to a K-Bus 2917 interface talking to a Kinetic Systems 3922 Parallel Bus Crate Controller on a standard CAMAC crate, but should work on any CAMAC, VxWorks system as long as the CAMAC standards are adhered to. The subroutines used to interface to CAMAC versions of the standard CAMAC utilities ported to VxWorks by Chip Watson and Ghram Heyes at CEBAF[C].

Recommended Additional Reading

1. LeCroy 1992 Research Instrumentation Catalogue,
   Appendix D, 'An Introduction To CAMAC',
   pp 335 to 344
2. CODA Manual
3. VxWorks Programmers Guide, Chapter 3, 'Basic OS',
   pp 57 to 108, Chapter 11, 'Shell',
   pp 381 to 405
4. Any basic C language tutorial

I. Brief introduction to VxWorks[A]

VxWorks is a multitasking operating system designed to interface with Posix operating systems such as Unix. The shell operates similarly to the standard Unix csh and is capable of calling procedures and handling variables. It is also capable of interpreting almost any C language expression including standard operators. The shell can be accessed by either using a terminal or via a remote telnet or rsh login. The system is fully accessible from an RS-232 line and a terminal; however, remote shells must log in on a valid account. Contact Dave Barker or Robert Millner for more information on remote login.

Initializing the MVME162 can be done by pressing the reset button on the front end of the controller or from the reboot command. While terminal is rebooting, switch the 3922 CAMAC Controller off line, clear (C), then initialize (Z). It is helpful to clear out everything before first use and occasionally when debugging new programs.

The terminal understands several control characters[B]. For a complete list, see VxWorks, p. 401. A few are listed here:

ctrl-h	backspace-delete
ctrl-u	delete line

ctrl-c	abort and restart shell
ctrl-x	Reboot
ctrl-s	Suspend output
ctrl-q	resume output

Variables may be declared and used at the shell prompt by assigning to them a value. Type redeclaration may be used as in the C language. The shell is case sensitive. For example, the following definitions:

```
-> temp = 0                    /*Type 4 byte integer*/
-> temp = 0x0000               /*4 byte integer, specified in hex*/
-> temp = (float) 3.14 /*4 byte floating point number*/
-> temp = (long) 45    /*4 byte integer*/
-> temp = (short) 33   /*2 byte integer */
-> temp = (char) 'A'   /*1 byte character*/
-> temp = (double) 3.14 /*8 byte double precision float*/
```

In addition, the shell understands referencing (*) and addressing (&). Referencing will not often be needed at the terminal; however, addressing is needed whenever a subroutine requires a variable whose value may change. When a variable is set to any value or expression, it is loaded on to the symbol table. A variable may not be used beforehand as it does not yet exist. Also on the symbol table are the references to subroutines. They are listed by the name that they are to be called in accessing them. The lkup 'string' command will check the symbol table and return any subroutine with a part that matches string. The symbols will be listed with an underscore before the name, this is to be ignored when accessing the symbol. Subroutine calls may be done at the command prompt by entering the symbol name of the routine, then the parameter list. Bear in mind that on the shell, as in C, only a value is passed so if a variable is to be modified by a subroutine then its address must be passed by using the & modifier. Failure to properly specify the address mode can have various effects from simply having an unset variable, writing a random memory location and forcing a reset, or accidently sending a bad bit of info and destroying good data without the user being aware of it. See the tutorial at the end for examples.

The shell understands many commands. A few are listed here.

help	Displays a help message
lkup "string"	Searches the symbol table for all occurrences of string.
i	Summary of tasks
h	Display shell history
version	Print version and boot line
whoami	Prints user info
iam "user"["passwd"]	Sets user (may need a password)
cd "path"	Change directory to path
pwd	Print working directory
ls	Directory listing
ll	Long form dir listing

rename "old", "new"	Change the name of a file
copy ["in"][,"out"]	Copy a file
ld [syms[,noAbort][,"name"]]	Load from console or file into memory syms = add symbol to table
printErrno value	Print the error text assoc with value
reboot	Reboot controller card

Parameters marked by [] are optional.

II. Talking to CAMAC

CAMAC is a standard for attaching several modules to a common backplane which can be controlled by a master module. The modules or "cards" can be referenced along that plane by specifying an address made up of the branch number, crate number, slot number, and sub-address on the card. Since we are only attaching one K-Bus controller in these examples, the branch number is always zero. There can be as many as eight crates attached and the crate number (0..7) can be set on the front panel of the 3922 controller. The slot is the physical location of the card and the address is the register on that card. Control functions of the card are specified by the function code (f) and any optional data. See part III for a list of function codes.

Interfacing to CAMAC can be done through several software routines. Some of these are listed in section III.3. In order to send a function to a card, a user must first encode a register to that address using "cdreg." Then the user can send commands, data, and receive status info through the "cfsa" function. The data is in the form of a four byte integer of which the lower twenty-four bits are significant. In other words, although 0xcebaf000 was passed, 0xbaf000 was sent to the card. The bottom two bits of the status will be the Q and X status. A one in the lowest bit signifies that the Q is missing, the next bit that X is missing. The X status signifies that a command was received and interpreted properly, the Q signifies that data was output properly. Cards request servicing by setting a "Look At Me" or "LAM" bit on the backplane.

Other commands include initializing and clearing the crate and module. This is a good idea to do once before other operations take place. LAMs may be set, enabled, and tested at the terminal with the CAMAC library functions.

III. CAMAC Info

1. Function Codes of particular interest:

0	Read (group 1, standard read)
1	Read (group 2)
2	Read and clear (group 1, standard read & clear)
3	Read compliment (group 1)
4	
5	
6	
7	
8	Test LAM
9	Clear register (group 1, standard)
10	Clear LAM

11	Clear (group 2)
12	
13	
14	
15	
16	Overwrite register (group 1, standard)
17	Overwrite (group 2)
18	Selective Set (group 1)
19	Selective Set (group 2)
20	
21	Selective Clear (group 1)
22	
23	Selective Clear (group 2)
24	Disable
25	Execute
26	Enable
27	Test-Status
28	
29	
30	
31	

Unlisted functions are either non-standard or reserved. Specific cards may have other commands.

2. Additional Function Codes for LeCroy 2228A TDC

9	Clear module
24	Disable LAM
25	Test Module
26	Enable LAM

3. CAMAC C Functions[C]

cdreg(int *ext, int b, int c, int n, int a)
Define a register, ext, to point to a branch b, crate c, slot n, and sub-address a.

cfsa(int f, int ext, int *data, int *q)
Deliver a function code, f to the card defined by register ext, with optional data I/O through data and status through q.

cccz(int ext)
Initialize the crate pointed to by register ext.

cccc(int ext)
Clear the crate pointed to by register ext.

ccci(int ext, int *logic)
Inhibit crate ext. Set logic true to inhibit the crate, false to remove the inhibit.

cccd(int ext, int *logic)
Control Crate demand, status of operation returned in
logic.

ctcd(int ext, int *logic)
Test crate demand, status of operation returned in logic.

ctgl(int ext, int *logic)
Test graded LAM, status of operation returned in logic.

cdlam(int *lam, int b, int c, int n, int a, int *inta)
Define a LAM, pointed to by register lam, referenced from
card b c n a (see cdreg())
int inta (ignored).

cclm(int lam, int logic)
Control LAM by applying logic.

cclc(int lam)
Clear LAM.

ctlm(int lam)
Test LAM.

caopen(char *server, int *success)
Connect to server machine through network.
Server is a null terminated string containing the
server's Internet name. Success returns a status word.

ctstat(int *istat)
Test status of system, returns in istat.

4. Variable Values

q	0 = Normal
	1 = no Q line
	2 = no X line
	3 = neither Q nor X
logic	0 = reset
	1 = set

IV. Examples

These examples have been tested on an MVME162-12 VME interface talking through a Kinetic Systems K-Bus 2917 to a Kinetic Systems 3922 Parallel Bus Interface to a standard CAMAC Crate. These examples must be run from the system prompt "->". Each line entered will return a value of that operation which may or may not be the value of one of the variables returned or status of operation. Value will be printed in both hex and decimal. Text in /* */ are comments and will have no operation. They may be omitted when entering commands. Remember that the shell is case sensitive. On our system, a DD002 Data Display module, BiRa3224 Test Module and a 2228 TDC were present in the CAMAC crate. The Test Module and TDC are necessary for one of the examples. The Data Display is a useful item when testing and debugging code.

1. Clearing The Crate

This is a simple example of delivering the clear and initialize commands to the entire crate. In this example, the commands are addressed to the crate master itself; however, they act on the entire crate. When they effect, you should see the lights on the crate blink for a moment. The lights on the crate master should be blank afterward.

```
/* This example assumes the crate to be 0*/
CRATE = 0
/* The standard is to place the controller in slot 25 */
SLOT = 25

EXT = 0   /* Enter EXT into the symbol table */

cdreg &EXT,0,CRATE,SLOT,0   /*Cause EXT to point to controller card*/
/* &EXT because we must send the address to the variable.*/
/* Can leave out the () in function calls */

cccc EXT        /*Clear the crate*/
/*Any status registers, specifically the DD002 and BiRa 3224 should go blank. The DD002 will
have an active "C" light.*/

cccz EXT        /*Initialize the crate*/
/*The DD002 should now have a "Z" light active.*/
```

2. Writing to the lights on the test board in slot 8

```
CRATE = 0
SLOT = 8

DATA = 0xCEBAF
/* Set data to hex word CEBAF, this will be output to the lights and will show up there in binary.*/

TEST_SLOT = 0
Q = 0
/* Enter into the symbol table a register for the test slot and return status value.*/
```

cdreg &TEST_SLOT,0,CRATE,SLOT,0
/* Cause TEST_SLOT to point to the test card*/

cccc TEST_SLOT
/*Usually a good idea to clear a card before the first intentional access. All lights on it should now be off.*/

cfsa 16,TEST_SLOT,&DATA,&Q
/* This sends the write command along with data to the card. The lights should now have 000011001110101110101111, 0=off, 1=on. &DATA and &Q because address must be passed since some functions write to DATA and Q. */

DATA
/*This will return the line:
DATA = address: value = 846767 = 0xcebaf
where address is a hex longword, 0x3fe734 when I ran this, pointing to the variable in memory. Value reads out the value of the variable in memory*/

Q
/*This will return the line: (see the above comment)
Q = address: value = 1 = 0x1
Significance as follows:
0 = Function executed and data returned ok.
1 = no Q (No or Improper return DATA, only worry if on a read function)
2 = no X (Function not executed properly, worry.)
3 = no X or Q (Function not executed properly and no data returned. Worry!)
*/

cfsa 0,TEST_SLOT,&DATA,&Q
/*This will, read the card. This particular card returns the inverse. It will also return a Q of 1 signifying an erroneous read.
*/

DATA
/*See the above 3 comments. Should return the line:
DATA = address: value = 15930448 = 0xf31450
Note that the data is the inverse of what it should be.*/

Q
/*See the above 4 comments. Should return this line:
Q = address: value = 1 = 0x1
Recall 1 signifies an erroneous read, see the above comments.*/

cdreg &TEST_SLOT,0,1,8,1
/*Set TEST_SLOT to point to address 1 on the card, the other set of lights.*/

cfsa 16,TEST_SLOT,&DATA,&Q
/*The other set of lights should now read:
111100110001010001010000, the inverse of the first set.
*/

cccz TEST_SLOT
/*Note that this does nothing to the card, there are no parameters in the TEST CARD that need be reset.*/

cccc TEST_SLOT
/*The lights are now all clear.*/

3. Preforming a test loop on the 2228A TDC in slot 10
CRATE = 0
SLOT = 10
TDC = 0
/*Define the TDC slot*/
cdreg &TDC,0,CRATE,SLOT,0
/*The TDC reads out its data on channels 0..7*/

cccc TDC
cccz TDC
/*Clear and initialize the crate.*/

DATA= 0
Q = 0
/*I/O variables*/

cfsa 25,TDC,&DATA,&Q
/*Test the module. This places it in a test cycle. Each channel should readout around 80% of its total value, or about 1638. On my test run, I got:

Channel	Number of Counts	Q
0	1739	1
1	1736	1
2	1731	1
3	1727	1
4	1720	1
5	1736	1
6	1729	1
7	1719	1

*/

cfsa 0,TDC,&DATA,&Q
/* Read out channel 0's test data.*/

DATA;Q;

/*Should output the address and value of both DATA and Q. See the above chart.*/

cdreg &TDC,0,CRATE,SLOT,1
/*Redefine to check channel 1.*/

cfsa 0,TDC,&DATA,&Q
/* Read out channel 1's test data.*/

DATA;Q;
/*Should output the address and value of both DATA and Q. See the above chart.*/

/*This may be repeated for the other six channels by re-assigning TDC, reading the data, and outputting the variables.*/

cccc TDC /*Clear the crate*/
cccz TDC /*Initialize the crate*/

Bibliography

A VxWorks Programmers Guide, pp 57 to 108

B VxWorks Programmers Guide, pp 381 to 405

C CODA manual, V. 1.1.3, September 1992, p4-2