CDF/DOC/SEC_VTX/GROUP/5101
Version 1
September 9, 1999

Control and monitoring of power to the SVX-II and ISL detectors through a *java* based graphical user interface

David Westbrook (Northwestern University)[1],
Mark Kruse (Rochester)[2]

Abstract

This note documents both the user interface to turn on the SVX-II and ISL detectors, and the underlying code which operates the CAEN power supplies. The java GUI invokes C code resident on a MVME162 processor using the ROBIN protocol, also used for the DAQ (cdfvme). As we get closer to Run II this will serve as a user's manual for the control of the SVX-II and ISL power supplies. Until then it is important we get feedback from the SVX-II and ISL groups on features still in need of implementation, safety concerns, etc., as this is how we will be operating these detectors.

[1]Rochester 1999 summer intern.

[2]We would also like to acknowledge those summer students who over the last 2 summers have helped in developing the CAEN control and monitoring C code, Ben Kilminster, Michelle Ernst, Robert Rudin and Hiwad Ayub.

	Output	V_{set}	V_{max}	I_{max}
Port Card	+5VDOIM	+5.0 V	+8.0 V	2.8 A
	+2VDOIM	+2.0 V	+3.5 V	-1.7 A
	DTERM	+2.5 V	+3.5 V	±0.15 A
Layers 0 to 4	DVDD	+5.0 V	+8.0 V	0.6 A (layer 0) 0.9 A (layer 1) 1.5 A (layer 2) 1.5 A (layer 3) 2.0 A (layer 4)
	AVDD	+7.0 V	+10.0 V	0.6 A (layer 0) 0.9 A (layer 1) 1.5 A (layer 2) 1.5 A (layer 3) 2.1 A (layer 4)
	VBIAS	0 to +250 V	+250 V	5 mA (layer 0...4)

Table 1: Voltage and current values for the SVX-II CAEN power supplies. The ISL supplies are the same except for the requirement of 2 VBIAS lines (with the same specs), and the DVDD and AVDD I_{max} values of 2.7 A and 2.4 A respectively.

1 Introduction

The CDF note documenting the full Interlock, Control and Monitoring System is CDF-4737, and also available from http:........

This note deals specifically with the part of that system designed for the power supply control. Custom SVX-II CAEN power supplies will provide the DVDD, AVDD, and VBIAS voltages for each layer, and the +5VDOIM, +2VDOIM, and DTERM voltages for the Port Card. There is 1 power supply per SVX-II wedge for a total of 72 power supplies. Custom ISL CAEN power supplies are identical except for the requirement of 2 VBIAS lines for every layer of a "wedge". There are a total of 32 ISL wedges (having a different structure for the forward and central sections) each powered by a ISL CAEN supply. The ISL power supplies are 2-slots wide due to the extra VBIAS requirement. Table 1 summarises the SVX-II and ISL power supply values. The VBIAS values are remotely programmable and can be set using the C control code resident on the MVME162 (see later) via the Java GUI. All the low voltage channels can only be adjusted by means of trimmers located on the printed circuit board, and will remain fixed during running. The maximum allowable currents, I_{max}, are all remotely programmable.

The power supplies will only be turned on after receiving a signal from the DAQ that it is ready, and after the cooling has been turned on and the bulkhead temperatures have achieved acceptable values (to be determined). The ability to turn on any power supply will be disabled until these conditions are met.

2 Hardware setup

The power supply control is VME based (6U). The main CPU is a MVME162 residing in a 6U crate in the counting room. A library of C code routines loaded onto the MVME162 are used to perform various operations of CAEN power supply control and monitoring. We have been developing this code in parallel to the development of the entire Interlock and Control system (CDF-4737). Details of the C control code are given in Section 5. The code sends commands to a V288, the CAENet Controller also resident in the 6U VME crate. Each V288 controls 2 CAEN crates (this might change to 4). The components of the system specific to power supply control are shown schematically in Figure 1. The ISL power supply control is analogous, and uses the same C routines on the MVME162. The only difference is in the java GUI which accomodates the different ISL power supplies and wedge structures.

3 The Java Graphical User Interface

PSControlForm is a JAVA GUI class designed for monitoring and control of the SVX power supplies and the contained layers and port cards. It provides up-to-date status, voltage, and current values, displayed numerically for individual channels, and graphically for all components of the SVX. Full control of status and set voltage and current values is provided.

3.1 Java GUI startup and related files

```
***************************************************************************
 Personal note :
 change the cdfbu4 path (from the burnin) to what it will be for the svx.
***************************************************************************
```

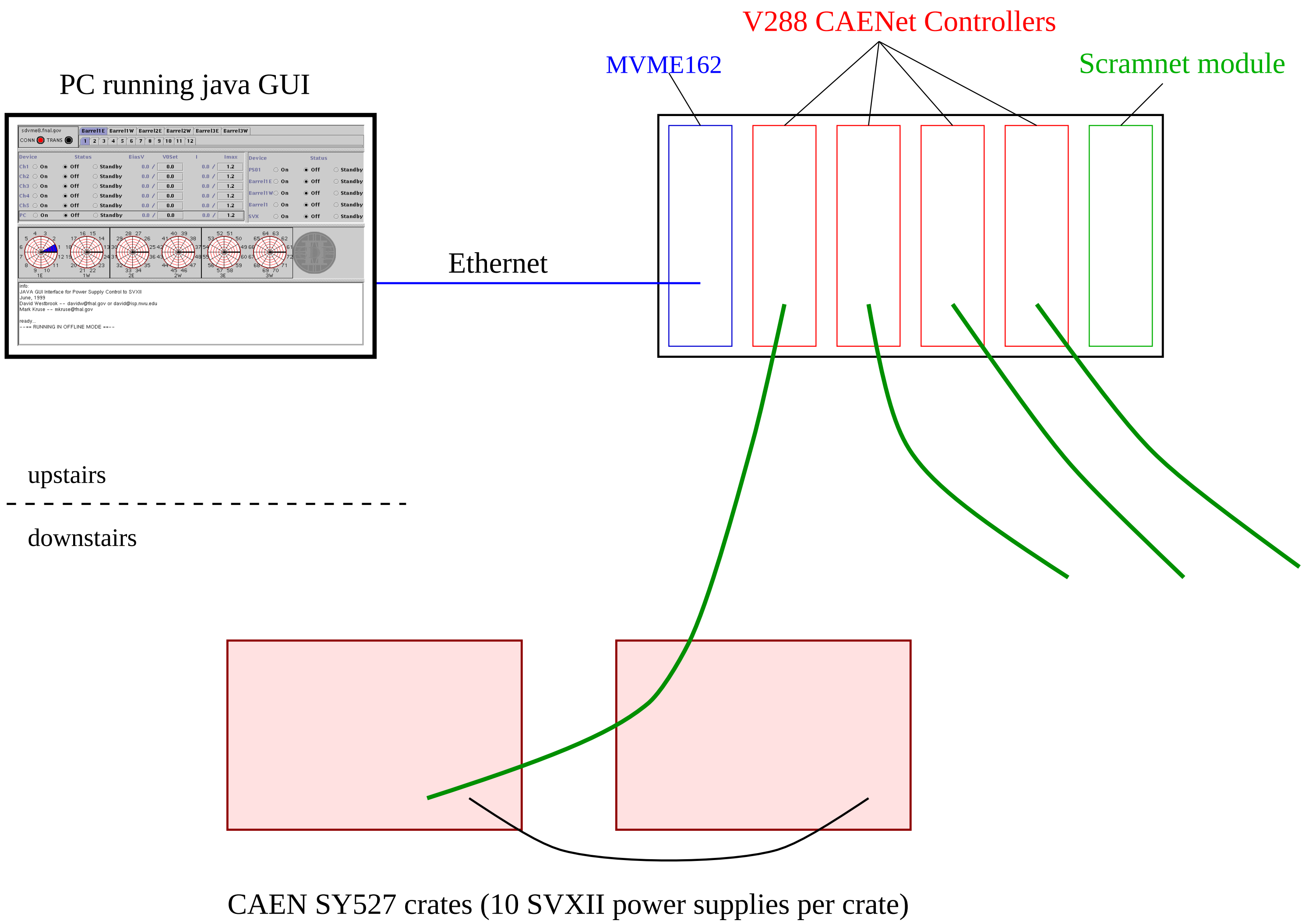

Figure 1: Schematic of the components used for CAEN power supply control. This is a subsystem of the full interlock, control and monitoring system. Shown is a single V288 module controlling 2 CAEN SY527 crates. This may not be the case in the final system.

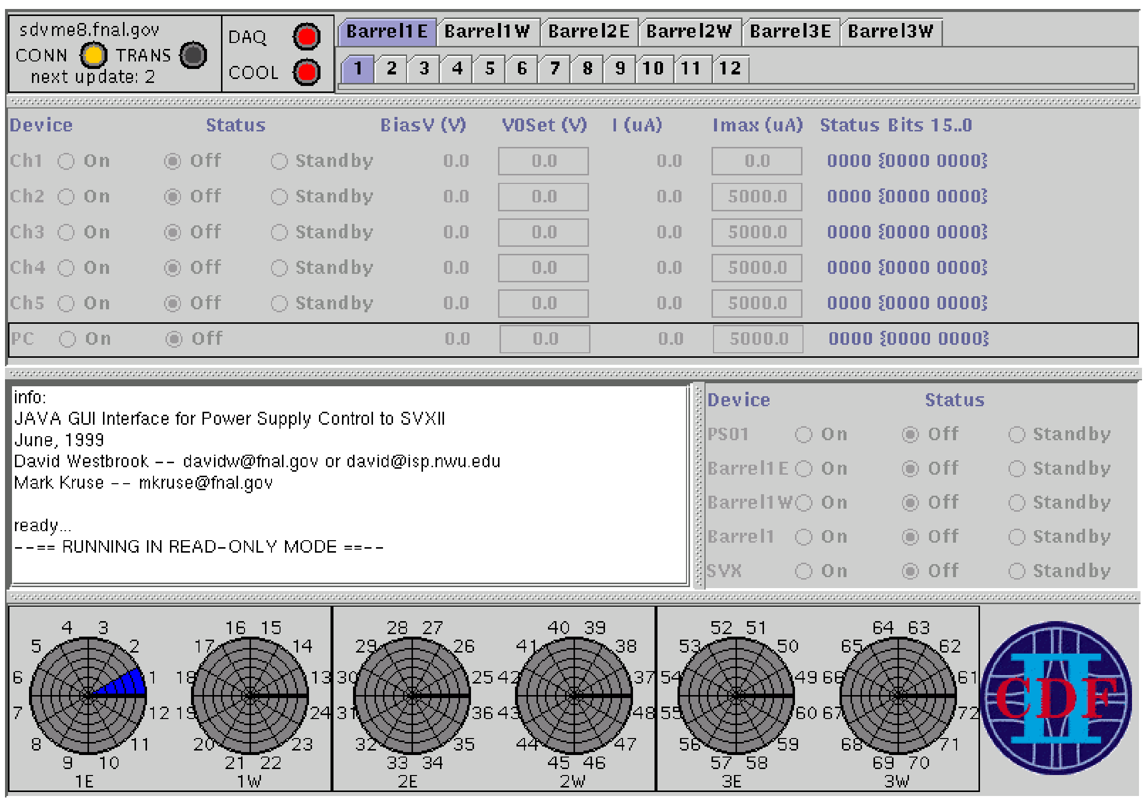

Figure 2: The java GUI used for SVXII power supply control and monitoring.

The java GUI is (presently) run from the Linux machine `cdfbu4` in the area
`cdfbu4:/home/svxii_d/burnin/hv`.
This is an nfs link to an area on `b0dau30` (in the Feynman centre) where all the CAEN power supply code is being developed. To start the GUI go the `gui/` directory and type `SVX` at the prompt. The java GUI will appear and will initialise with values and labels determined from the file `SVX.conf` (see Table 3.1). A screen capture of the current SVXII GUI is given in Figure 2.

The channel labels can be changed by editing this file, as can the default channels to turn ON when the "PS on" button is clicked. (Note: `SVX.conf` is only read when the GUI is first loaded.) From the GUI one can set the bias voltage for each channel (up to 200 V) and the maximum current (up to 1 mA). The monitored voltage and bias current is also displayed, as well as the status word for the channel status (see Table 3.1). Values are updated every 3s from the crate. The default v0set values are also read in from a file by the GUI. This file is `cdfbu4:/home/svxii_d/burnin/hv/volts.dat` (see Table 3.1) and, again, is only read when the GUI is first loaded.

Bits	bit value = 0	bit value = 1
0	Channel not Present	Channel Present
1..4	Don't care	Don't Care
5		Internal Trip
6		Kill
7	Don't care	Don't Care
8		Vmax
9		External Trip
10		Overvoltage
11		Undervoltage
12		Overcurrent
13		Down
14		Up
15	Channel Off	Channel On

Table 2: Channel Status Word Bits–refer to p.90 of the MOD.SY 527 manual.

SVX Directory

Table 3: Format of SVX.conf file which is read upon java GUI startup.

Default Standby Voltage
Default v0set for PS 0, Layer 1
Default v0set for PS 0, Layer 2
Default v0set for PS 0, Layer 3
Default v0set for PS 0, Layer 4
Default v0set for PS 0, Layer 5
Default v0set for PS 1, Layer 1
Default v0set for PS 1, Layer 2
Default v0set for PS 1, Layer 3
Default v0set for PS 1, Layer 4
Default v0set for PS 1, Layer 5
.
.
.

Table 4: Format of volts.dat file.

3.2 Normal Operation

The GUI (Graphical User Interface) has six main sections: info, tabs, channels, masters, output, and display.

Info displays the name of the 162 with which the 162 communicates, as well as two indicator lights. CONN (CONNected) is green if the gui is connected to the 162 and is allowed to write to it; orange if the gui is connected, but Read-Only; and red if it is not connected at all. TRANS (TRANSmission) is yellow while the gui is in communication w/the 162. At this time a lock file is also in place.

The tabs indicated which power supply can be monitored and controled in the channels section. There are two sets of tabs, one for the barrel end, and one for which of the twelve power supplies in the respective barrel end. The power supplies are numbered 1-72.

The channels panel has the readouts and controls for the five layers and the port card of the current power supply. For each channel (1-5, 1 being the innermost layer), the status, V0Set, and IMax (I0Set) are displayed and can be changed. The BiasV (vmon) and I (current–imon) are also display, but cannot be changed. The port card is the same, except the only changeable item is the status, but only to on or off.

The masters panel has the controls for the "larger" detector components (ie "masters" over the smaller parts). These allow control of the entire power supply, barrel end, barrel, and the entire svx. Changing the status of any of these is equivalent to changing all the components "below" it to the same status. Also, "lower" components cannot be turned to on or standby while something higher is off.

The output provides an overall picture of the status of the entire SVX. It displays all six barrel ends, each with twelve wedges (power supplies), each with five layers (1-5) and the outer port card starting at the center and going radially outward. The barrel ends are labeled, and the power supplies are numbered 1-72. Grey wedges indicate an inactive or absent power supply. Red lines indicate an off state; yellow standby; green on; black not functional/available.

The display panel contains the programs output to the user (except in the intial startup where output goes to stdout). Confirmations of changes are here, as well as notices of errors or denials. The display panel is in a scroll bar, so that the entire text can be accessed.

3.3 Details of the Java GUI and code

The info panel and output panel are both custom classes extended from JPanel that basically just have the paintComponent method overridden, so as to be able to manualy draw the graphics and text that is seen. The output panel also acts on mouse clicks, so that clicking on the wedge will change the tabs (visa versa is also true).

The psTabs object is given to whichever of the barrelTabs that is curently selected. There are indices curPS and curBarrel that indicate the current ps and barrel, so that the other panels can update accordingly. This is all called from componentShown of the tab objects.

All of the data representing the values from the 162 is kept in a SVX object–a buffer between the gui and the 162 itself. The SVX object is organized like the physical SVX itself – 3 barrels, 2 ends each, 12 wedges (power supplies) each, each with 5 layers and a portcard. Everything has a status, and the channels (and portcard) also have two voltages and two currents.

The SVX class does the range checking of the voltage and current values. There are the constants CURRENT_MIN, CURRENT_MAX, VOLTAGE_MIN, VOLTAGE_MAX in SVX.java.

The gui does the confirmations of all changes, and the checks that a "higher" component is in an appropriate state.

The SVX class has a member that is a ROBIN object, through which calls to the code running on the 162 can be made. The gui does not directly call the ROBIN code at all – just the SVX object. When calls are made, which is only during the three "set" methods and during `resetValuesFromVME()`, a system of lock files is set in place in the boot directory (arbitrary) of the 162. If online, a file called `.lock.online.162_ADDRESS` is made; if readonly, the file is called `.lock.readonly.162_ADDRESS`. If a lock file is present, it waits for it to go away. If not readonly, and there is a readonly lock file, a file `.lock.request.162_ADDRESS` is made, and the readonly process will recognize this and abort, giving priority to the online process.

On startup of the gui, before the actual gui constructor, several checks are made for cmd line parameters, security, and lock files.

The allowed command line parameters are -readonly or -offline. If there are no parameters, the gui starts up and is connected to the 162 and has control of it. -readonly means that the gui is connected, but cannot control anything – basically a convienent monitoring tool. -offline is basically for gui debugging – it does not connect to the 162 at all.

The gui has a hard-coded, constant list of allowed users. If it is not executed by a valid user, it will promptly exit.

If there are any lock files (for that specific 162) present, the gui will exit. There's not a sure way to know if the lock file is old (most likely; cause by a crash/kill) or "live". This is the safest way.

If online, updates are every 3 seconds from the crate. If readonly, the update interval is 5 times as long – the more immediate knowledge isn't as crucial.

4 The ROBIN interface code

Here we will examine the code executed when the ON/OFF/Standby radio button is changed. This will provide an example of how the java GUI calls the MVME162-resident c-code via ROBIN.

4.1 Channel ON code flow

We first step through the java code: Whenever the radio buttons are pushed, it invokes the RadioListener. svx is a SVX object, which is a class to represent the SVX – it is 3 barrels, each with 2 ends that each have 12 wedges that each have 5 layers and a portcard. The SVX class has methods to get and set the various properties (status, v0set, etc.). The SVX class has a member caen, which is the ROBIN object. The excerpt from SVX.java shows its declaration, initialization, and use in setting the status of a channel.

The setChannelStatus method of caen indirectly runs code from caentalk.c, the library of CAEN control routines, which is compiled and resident on the MVME162 itself. ROBIN is the link that allows the local-machine-java to call the remote-machine-c routines. The setChannelStatus function in caentalk.c is actually called from a ROBIN-generated file CAEN_S.c, which we will illustrate next.

From PSControlForm.java:

```
public class PSControlForm extends JPanel implements ComponentListener{
  SVX svx = null;
  public PSControlForm() {
    svx = new SVX(isOnline,isReadOnly);
  }
  class RadioListener implements ActionListener {
    public void actionPerformed(ActionEvent e) {
      // Figure out which channel/ps and which status is desired
      // Make sure action is valid/permitted.
      // Confirm action.
```

```
      ok = svx.setStatus(barrel, barrelEnd, wedge , layer ,status);
    }
  }
}
```

From SVX.java:

```
class SVX {
  static CAEN.CAENTalk caen;
  public SVX(boolean online, boolean isReadOnly) {
    try {
      caen =  new CAEN.CAENTalkClient(null, CAEN_IP_ADDRESS);
    } catch (java.lang.ExceptionInInitializerError e) { System.out.println("java.lang.ExceptionInInitializerError"); caen = null; }
  }
  static public boolean setStatus(int barrel, int theEnd, int ps, int ch, int newStatus){
     // Check parameters.
     ok = (0==caen.setChannelStatus((short)(ps+NUM_PS_PER_BARREL*(barrel*NUM_ENDS+theEnd)),(short)ch,LAYER_INDEX,(short)newStatus));
  }
}
```

From caentalk.c:

```
ushort setChannelStatus(ushort ps_n, short layer_n, short indiceType, ushort status){
  /*
    Actual code to communication with power supplies goes here.
  */
}
```

4.2 ROBIN setup and compilation

First the idl file must be created, basically containing the prototypes of the desired remote functions.
From this the ROBIN code is generated by using ridl, and is compiled normally (with javac and cc).
The compiled ROBIN code that runs on the MVME162 is what calls the routines in the

`caentalk.c` library. This compiled ROBIN code is loaded automatically by the ROBIN-server, assuming the correct path is in the 162's ROBIN_repository environmental variable and that the library has the same name as the module (in this case CAEN).

Important Files (on b0dau30):

```
====================================================
     -= ROBIN Compilation for Module: CAEN =-
Input:
   IDL FILE:     /cdf/people1/mkruse/david/form/server/CAEN/CAEN.idl
Generated:
   ROBIN FILE:   /cdf/people1/mkruse/david/form/server/interface_rep/CAEN
   JAVA PACKAGE: /cdf/people1/mkruse/david/form/CAEN
   C-CODE:       /cdf/people1/mkruse/david/form/server/CAEN/CAEN_S.h
                 /cdf/people1/mkruse/david/form/server/CAEN/CAEN_S.c
Output:
   C-Library:    /cdf/people1/mkruse/david/form/server/lib/VxWorks-5.3/68k/CAEN
====================================================
```

From CAEN.idl:

```
module CAEN {
  interface CAENTalk {
    typedef unsigned short ushort;
    unsigned short setChannelStatus(in unsigned short ps_n, in unsigned short layer_n, in short indiceType, in unsigned short status);
  }
}
```

From `make_CAEN` (my script to compile the ROBIN code – runs from b0dau30):

```
cd /cdf/people1/mkruse/david/form/server/CAEN
ridl -language=C-service -repository=../interface_rep CAEN.idl
cd /cdf/people1/mkruse/david/form
ridl -language=Java-client -repository=../interface_rep server/CAEN/CAEN.idl
cd /cdf/people1/mkruse/david/form/CAEN
```

```
javac *.java
cd /cdf/people1/mkruse/david/form/server/CAEN
$VXCC -w -I $ROBIN_DIR/include -I caentalk.c_Diretory CAEN\_S.c -o ../lib/VxWorks-5.3/68k/CAEN
cp -p ../lib/VxWorks-5.3/68k/CAEN /cdf/people1/mkruse/mvme162/CAEN.o
```

From the generated CAEN_S.c:

```
/* BEGIN USER DEFINITIONS */
#include "caentalk.h"
#include "psmon.h"
/* END USER DEFINITIONS */

CORBA_unsigned_short *CAEN_CAENTalk_setChannelStatus( this, args, ev )
  {
// generated variables

  /* BEGIN USER VARIABLES for CAEN_CAENTalk_setChannelStatus */
  /* END USER VARIABLES for CAEN_CAENTalk_setChannelStatus */

//generated code

  /* BEGIN USER CODE FOR CAEN_CAENTalk_setChannelStatus */
  *ROBINreturn = setChannelStatus(ps_n,layer_n,indiceType,status);
  /* END USER CODE FOR CAEN_CAENTalk_setChannelStatus */
  return( ROBINreturn );
  }
```

How to start the ROBIN server on the MVME162:

```
telnet sdvme8
<login>
iam "vxworks"
ld < /cdf/code-common/cdfonline/robin/v2_4_10/VxWorks-5.3/68k/ROBINserver
ROBINauthAdd(0xffff0000,0x83e10000)
strcpy(&ROBIN_repository,"/cdf/people1/mkruse/david/form/server/lib/VxWorks-5.3/68k")
```

```
sp ROBINserver
ld < /cdf/people1/mkruse/mvme162/CAEN.o
```

In the final system the `ROBINserver` will be loaded and spawned in the MVME162 startup file, as to will the CAEN code `CAEN.o`, which contains the compiled CAEN library of routines caentalk.c.

5 The CAEN control C code

Here we supply more details on the CAEN control C code routines that are run on the MVME162 and invoked by the java GUI. The routines themselves were listed in the previous section. Firstly, it should be noted that we do NOT use FISION for our VME read and writes to modules, as does the DAQ system. This has also meant that we do not use CDFVME in the usual way documented for the DAQ system (we just take out all the FISION stuff). This because we are talking to modules (the V288 and the Scramnet module) with different VME addressing - the V288 is A16/D16 and the Scramnet module is A24/D16. FISION could not handle this when we were developing this system, but apparently it can now. However, our VME read and writes are much simpler and MUCH faster (by a factor of 10) and so suit our needs better. Specifically they are as follows :

```
int writevme(unsigned int vme_address, unsigned short *data, int size)
{
  if(size == WORD)   /*  WORD = 2  */
    {
    unsigned short *vadd = (unsigned short *)vme_address;
    *vadd = *data;
    return 1;
    }
  else
    {
    return -1;
    }
}
```

```
int readvme(unsigned int vme_address, unsigned short *data, int size)
{
  if(size == WORD)
    {
    unsigned short *vadd = (unsigned short *)vme_address;
    *data = *vadd;
    return 1;
    }
  else
    {
    return -1;
    }
}
```

A desired command is sent to the CAEN power supplies by means of "codes". The full list of codes is extensive, so in Table 5 we only list those that we use for normal operation. Operations such as setting ramp up and down values for each channel are not included as these are hardwired in the SVXII and ISL supplies. The code is one word in a data packet that is sent to the V288, the contents of the data packet (word1, word2, ...) is also given in Table 5. The "Set" commands are achieved by simply sending the data packet shown. The "Read" commands are achieved by sending the data packet shown, upon which the CAEN crate will send back a data packet to the V288 which must be read and decoded by the MVME162 C code. The returned data packet structure depends upon the request, and is detailed for each read operation in the CAEN SY527 manual. We will discuss those of interest to the more general user as appropriate. These operations form the basis of the CAEN control library of C routines in `caentalk.c` and which is loaded onto the MVME162 as part of `CAEN.o`.

word 1 (ID code)	**word 2** (crate #)	**word 3** (code)	**word 4**	**word 5** (settings)	**Meaning**
0x0001	0x00cc	0x00	–	–	Read Module Identifier
		0x03	0x000b	–	Read slot b board characteristics
		0x04	–	–	Read crate occupation
		0x01	0x0bnm	–	Read channel b.nm status
		0x02	0x0bnm	–	Read channel b.nm parameter values
		0x10	0x0bnm	V0set value	Set channel b.nm V0 value
		0x11	0x0bnm	V1set value	Set channel b.nm I0 value
		0x18	0x0bnm	Mask and Flag	Set channel b.nm Flag values (Pon, On/Off, password, Power)
Group operations					
0x0001	0x00cc	0x1B	0x0g	"A","B" followed by words 6 to 10	Set group name
		0x40	0x0g	–	Read channels in a group
		0x41	0x0g	–	Read Vmon/Status/Imon of group channels
		0x50	0x0g	–	Add channel to a group
		0x51	0x0g	–	Remove channel from a group
		0x52	0x0g	–	Set V0set of channels in a group
		0x54	0x0g	–	Set I0set of channels in a group
		0x5A	0x0g	–	Switch ON the channels in a group
		0x5B	0x0g	–	Switch OFF the channels in a group

Table 5: CAEN control data packet description. "cc" is the crate number, "b" the board (slot) number, "nm" the channel number, and "g" is the group number (0 to 15 are the possible values).