ISTITUTO NAZIONALE DI FISICA NUCLEARE

Sezione di Torino

INFN/AE-97/14
10 Aprile 1997

A. Raimondo:

LibVME – USER'S GUIDE VER. 1997

PACS N.: 07.05.H

SIS–Pubblicazioni
dei Laboratori Nazionali di Frascati

INFN – Istituto Nazionale di Fisica Nucleare
Sezione di Torino

INFN/AE–97/14
10 Aprile 1997

LibVME

User's Guide ver. 1997

Alessandro RAIMONDO
Dip. Fisica Sperimentale - Università di Torino
INFN Sezione di Torino
via P. Giuria 1, 10125 Torino, Italia
Tel. ++39(11)6707321, Fax. ++39(11)6707325
Email: Raimondo@to.infn.it

This document is a user's guide and a reference manual for the VME library implemented on RTPC8067-LynxOS[1, 2] for the FINUDA experiment.

Any corrections, comments, suggestions, job positions are welcome.

Contents

Chapter 1

Introduction

The intensive use of VME standard in Data Acquisition Systems (DAQ) make me to develop a library for the most useful VME module.

In that feature it becomes the idea to standardise the language of writing these libraries to have an approach as the Standard CAMAC Esone Subroutines[3]. It is not possible to define a fixed number of functions useful for all the module as the CAMAC Esone, because of the different structure of VME. Then libraries become a link of subroutines for a particular VME module and not for a particular CAMAC interface.

The aim of this manual is the description of the implementation of VME libraries for some VME module made by CAEN and CES, but it wants to be a suggestion of a standard language to develop further VME libraries.

Libraries are written in C language and they are, when possible, platform independent. These can be used with different processors or Operative Systems (OS) and can be called by main programs written in different languages taking care of the variables definition and the calling sequence. All that one, but the library for VME error trapping which depends on OS and machine resources.

These characteristics improve the porting of programs among different on-line computers and define a common VME software terminology and programming technique.

For our aim the libraries don't cover all the potentiality of each module, in particular they don't content subroutines for interrupt handling or Block Mover Accelerator (BMA) transfer. These are reasonable choices because interrupts can reduce in some cases the speed of DAQ . For the same reasons the use of BMA is useful only with high transfer rate on VME bus and in this case is better to write the readout of the module without calling any subroutines directly in the calling program in order to increase the speed. In other words the library includes all the instructions to set-up modules and the most used instructions to read them out.

The subroutine naming conventions are based on the following rules:

action_register_module(base,section,data, ...)

first word	action to do on the module
second word	register to address
third word	name of the VME module
first parameter	module base address
second parameter	section to address
other parameters	data to write or read

These standard definitions are chosen for the mnemonic word to identify the function to be performed on the module.

All parameters, but specific cases, are defined unsigned integer and in this manucript their value is indicated in exadecimal format.

Chapter 2

Error handling & pointers

In order to support the whole library the file vme_lib.h has to be included, it defines the useful pointers for the debugging access. All the functions are void and they don't return nothing after calling, but they store in some global variables their status. This choice of global variables is important during the error trap, when the control of the process becomes available by a subroutine called by an error signal.

The defined global variables are:

unsigned short *pointVME	last VME address accessed
short errorVME	status code after a VME access
unsigned operVME	code of the access in progress
char *operVME, char *errorVME	strings associated on each code
buf_struct *jmpVME	pointer of last setjmp

See Appendix C for detailed information about values.

The error trapping routine provides to trap VME bus error coming on each access error on the bus, regardless of using library subroutines. When a bus error occurs, the control of the program flow goes to the trapping routine and a message is printed out. If the error comes during a library access we have also the information concerning the kind of access (read/write/on CBD) and the address where the error occurs. After an error trapping the control is returned through a longjmp(-1) to last fixed setjmp fixed in the main program, which must include <setjmp.h>.

Chapter 3

ScalerLib

A library for the V560 CAEN Scaler[4].

3.1 Set Veto

calling sequence

set_veto_scaler(base)

parameters

variable	*range*	*what?*
base	32bits	module base address

function

Inhibit counters

3.2 Reset Veto

calling sequence

clear_veto_scaler(base)

parameters

variable	*range*	*what?*
base	32bits	module base address

function

Set counters in active state

3.3 Clear Counters

calling sequence

clear_count_scaler(base)

parameters

variable	*range*	*what?*
base	32bits	module base address

function

Reset to zero all counters

3.4 Test Counters

calling sequence

test_count_scaler(base)

parameters

variable	*range*	*what?*
base	32bits	module base address

function

Increase all counters of one

3.5 Read Configuration

calling sequence

read_config_scaler(base,config,count)

parameters

variable	*range*	*what?*
base	32bits	module base address
config[8]	0÷1	configuration of scaler channels 0 = 32bits counter, 1 = 64bits counter

function

Return a vector containing the channel cascade configuration, 0=not cascade (32 bits channel), 1=cascade (64 bit channel). The position i of the vector indicate if channels $2i$ and $2i + 1$ are cascaded.

3.6 Read Counters

calling sequence

read_count_scaler(base,config,count)

parameters

variable	*range*	*what?*
base	32*bits*	*module base address*
config[8]	0÷1	configuration of scaler channel 0 = 32bits counter, 1 = 64bits counter
count[16][2]	32bits	channels counts [0] = low 32bits, [1] = high 32bits

function

Giving the channel configuration, 32 bits or 64 bits channels, reads all scaler counters. For channel i low counts are in [i][0] and high counts in [i][1].

Chapter 4

DelgateLib

A library for the V486 CAEN Delay and Gate Generator[5]

4.1 Clear Channels Output

calling sequence

clear_output_delgate(base)

parameters

variable	*range*	*what?*
base	32bits	module base address

function

Clear all channels output

4.2 Send Software Strobe

calling sequence

send_strobe_delgate(base)

parameters

variable	*range*	*what?*
base	32bits	module base address

function

Send a software strobe to the module for test purpose

4.3 Read Select Register

calling sequence

read_select_delgate(base,&mode,&mux)

parameters

variable	*range*	*what?*
base	32bits	module base address
mode	0÷2	working mode 0=normal, 1=common, 2=couple
mux	0÷7	channel on NIM output

function

Read the content of the select register and answer with the mode and the mux set

4.4 Write Select Register

calling sequence

write_select_delgate(base,mode,mux)

parameters

variable	*range*	*what?*
base	32bits	module base address
mode	0÷2	working mode 0=normal, 1=common, 2=couple
mux	0÷7	channel on NIM output

function

Write in the select register the chosen mode and mux

4.5 Write Delay and Gate

calling sequence

write_delgate_delgate(base,delay,gate)

parameters

variable	*range*	*what?*
base	32bits	module base address
delay[8]	0÷ ff	delays
gate[8]	0÷ ff	gates

function

Write the delays and the gates chosen for each channel

Chapter 5

PatternLib

A library for the V259 CAEN Pattern Unit[6]

5.1 Clear all

calling sequence

clear_all_pattern(base)

parameters

variable	*range*	*what?*
base	32bits	module base address

function

Clear the input pattern and the multiplicity register

5.2 Read Pattern

calling sequence

read_pattern_pattern(base,&pattern)

parameters

variable	*range*	*what?*
base	32bits	module base address
pattern	16bits	input pattern

function

Read the input pattern

5.3 Read and Reset Pattern

calling sequence

readrst_pattern_pattern(base,&pattern)

parameters

variable	*range*	*what?*
base	32bits	module base address
pattern	16bits	input pattern

function

Read the input pattern and reset it

5.4 Read Multiplicity

calling sequence

read_multi_pattern(base,&multi)

parameters

variable	*range*	*what?*
base	32bits	module base address
multi	16bits	input multiplicity

function

Read the multiplicity register

5.5 Read and Reset Multiplicity

calling sequence

readrst_multi_pattern(base,&multi)

parameters

variable	*range*	*what?*
base	32bits	module base address
multi	16bits	input multiplicity

function

Read the multiplicity register and reset it

Chapter 6

IoregLib

A library for the V513 CAEN I/O Register[7]

6.1 Reset

calling sequence

reset_all_ioreg(base)

parameters

variable	*range*	*what?*
base	32bits	module base address

function

Reset the module in the starting configuration

6.2 Clear Strobe

calling sequence

clear_strobe_ioreg(base)

parameters

variable	*range*	*what?*
base	32bits	module base address

function

Clear the strobe

6.3 Clear Input Register

calling sequence

clear_input_ioreg(base)

parameters

variable	*range*	*what?*
base	32bits	module base address

function

Clear the input register. Only channel bits set to input are cleared

6.4 Read Input Register

calling sequence

read_input_ioreg(base,&reg)

parameters

variable	*range*	*what?*
base	32bits	module base address
reg	16bits	input pattern

function

Read the input pattern. Only channel bits set to input are significative

6.5 Write Output Register

calling sequence

write_output_ioreg(base,reg)

parameters

variable	*range*	*what?*
base	32bits	module base address
reg	16bits	output pattern

function

Write the output pattern. Only channel bits set to output are involved

6.6 Set Strobe Polarity

calling sequence

set_strobe_ioreg(base,value)

parameters

variable	*range*	*what?*
base	32bits	module base address
value	0÷1	strobe polarity 0=positive, 1=negative

function

Set strobe polarity: negative or positive

6.7 Read Strobe Register

calling sequence

read_strobe_ioreg(base,&strobe)

parameters

variable	*range*	*what?*
base	32bits	module base address
value	16bits	strobe register

function

Read the strobe register

6.8 Read Channels Status

calling sequence

read_status_ioreg(base,status)

parameters

variable	*range*	*what?*
base	32bits	module base address
status[16]	4bits	channels status

function

Read the status for each channel

6.9 Write Channels Status

calling sequence

write_status_ioreg(base,status)

parameters

variable	*range*	*what?*
base	32bits	module base address
status[16]	4bits	channels status

function

Write the status for each channel

Chapter 7

PluLib

A library for the V495 CAEN I/O Programmable Logic Unit[8]

7.1 Write Configuration Register

calling sequence

write_config_plu(base,config)

parameters

variable	*range*	*what?*
base	32bits	module base address
config	0÷7ff	configuration register 30c=normal, c=strobed 30f=program

function

Write the configuration register

7.2 Read Configuration Register

calling sequence

read_config_plu(base,&config)

parameters

variable	*range*	*what?*
base	32bits	module base address
config	0÷7ff	configuration register

function

Read the configuration register

7.3 Read Input Register

calling sequence

```
read_pattern_plu(base,&patta,&pattb)
```

parameters

variable	*range*	*what?*
base	32bits	module base address
patta	8bits	pattern section A
pattb	8bits	pattern section B

function

Read the input patterns for the two sections

7.4 Software Strobe

calling sequence

```
strobe_section_plu(base,section)
```

parameters

variable	*range*	*what?*
base	32bits	module base address
section	0÷1	section to strobe 0=section A, 1=section B

function

Software strobe a section

7.5 Write Logical Functions

calling sequence

write_function_plu(base,section,array)

parameters

variable	*range*	*what?*
base	32bits	module base address
section	0÷1	section to write 0=section A, 1=section B
array[256]	0÷ff	functions array

function

Write the logical functions in memory for a section

7.6 Read Logical Functions

calling sequence

read_function_plu(base,section,array)

parameters

variable	*range*	*what?*
base	32bits	module base address
section	0÷1	section to read 0=section A, 1=section B
array[256]	0÷ff	functions array

function

Read the logical functions from memory for a section

Chapter 8

Adc9uLib

A library for the VN1465S CAEN 64 Channel Multi QDC[9]

8.1 Reset

calling sequence

reset_all_adc9u(base)

parameters

variable	*range*	*what?*
base	32bits	module base address

function

Reset the module in the starting configuration

8.2 Read Hardware Register

calling sequence

read_hard_adc9u(base,&reg)

parameters

variable	*range*	*what?*
base	32bits	module base address
reg	16bits	hardware register

function

Read the hardware register

8.3 Write General Control Register 1

calling sequence

write_general_adc9u(base,reg)

parameters

variable	*range*	*what?*
base	32bits	module base address
reg	8bits	general control register 1

function

Write the general control register 1

8.4 Write General Control Register 2

calling sequence

write_trigger_adc9u(base,reg)

parameters

variable	*range*	*what?*
base	32bits	module base address
reg	1÷1ff	general control register 2

function

Write the general control register 2

8.5 Write Adjust

calling sequence

write_adjust_adc9u(base,adj)

parameters

variable	*range*	*what?*
base	32bits	module base address
adj	12bits	pedestal adjustment

function

Write pedestal adjustment offset current for all channels

8.6 Write Thresholds

calling sequence

write_threshold_adc9u(base,low,high)

parameters

variable	*range*	*what?*
base	32bits	module base address
low	0÷ 1ff	low threshold
high	0÷ 1ff	high threshold

function

Write low and high thresholds, each bit is 15 or 120 counts of ADC channels depending on range selected

8.7 Write Channels Register

calling sequence

write_chreg_adc9u(base,reg1,reg2)

parameters

variable	*range*	*what?*
base	32bits	module base address
reg1	f ÷ 3ff	channel register 1
reg2	f ÷ 3ff	channel register 2

function

Write channel registers one for each module section

8.8 Write Enable Register

calling sequence

write_enable_adc9u(base,ena1,ena2)

parameters

variable	*range*	*what?*
base	32bits	module base address
ena1	8bits	enable register 1
ena2	8bits	enable register 2

function

Write enable registers one for each module section

8.9 Write Gate Value

calling sequence

write_gate_adc9u(base,gate)

parameters

variable	*range*	*what?*
base	32bits	module base address
gate	8bits	gate value

function

Write an internal gate value for test purpose

8.10 Start Gate Generator

calling sequence

start_gate_adc9u(base)

parameters

variable	*range*	*what?*
base	32bits	module base address

function

Start a generation of a gate set before on the Gate Register

8.11 Read Status Register

calling sequence

read_status_adc9u(base,&reg)

parameters

variable	*range*	*what?*
base	32bits	module base address
reg	16bits	status register

function

Read the status register of the module

8.12 Increase Read Pointer

calling sequence

increase_pointer_adc9u(base)

parameters

variable	*range*	*what?*
base	32bits	module base address

function

Increase the read pointer

8.13 Read Trigger/Event Counter

calling sequence

read_counter_adc9u(base,&reg)

parameters

variable	*range*	*what?*
base	32bits	module base address
reg	16bits	trigger/event counter

function

Read the trigger/event counter

8.14 Read Write and Read Pointers

calling sequence

read_pointer_adc9u(base,&rpoint,&wpoint)

parameters

variable	*range*	*what?*
base	32bits	module base address
rpoint	4bits	read pointer
wpoint	4bits	write pointer

function

Read the read and the write pointers

8.15 Write Pedestal Memory

calling sequence

write_pedestal_adc9u(base,ped)

parameters

variable	*range*	*what?*
base	32bits	module base address
ped[64]	12bits	pedestal array

function

Write the pedestal for each channel into pedestal memory

8.16 Read Pedestal Memory

calling sequence

read_pedestal_adc9u(base,ped)

parameters

variable	*range*	*what?*
base	32bits	module base address
ped[64]	12bits	pedestal array

function

Read the pedestal for each channel from pedestal memory

8.17 Read event

calling sequence

read_event_adc9u(base,&num,data)

parameters

variable	*range*	*what?*
base	32bits	module base address
num	$1 \div 40$	number of conversions
data[64]	32bits	data array

function

Read converted data from a module, num is the number of conversions, the array contains data for num locations

8.18 Read General Control Register 1

calling sequence

read_general_adc9u(base,&reg)

parameters

variable	*range*	*what?*
base	32bits	module base address
reg	8bits	general control register 1

function

Read the general control register 1

8.19 Read General Control Register 2

calling sequence

read_trigger_adc9u(base,&reg)

parameters

variable	*range*	*what?*
base	32bits	module base address
reg	1÷1ff	general control register 2

function

Read the general control register 2

8.20 Read Adjust

calling sequence

read_adjust_adc9u(base,&adj)

parameters

variable	*range*	*what?*
base	32bits	module base address
adj	12bits	pedestal adjustment

function

Read pedestal adjustment offset current for all channels

8.21 Read Thresholds

calling sequence

read_threshold_adc9u(base,&low,&high)

parameters

variable	*range*	*what?*
base	32bits	module base address
low	0÷ 1ff	low threshold
high	0÷ 1ff	high threshold

function

Read low and high thresholds, each bit is 15 or 120 counts of ADC channels depending on range selected

8.22 Read Channels Register

calling sequence

read_chreg_adc9u(base,&reg1,&reg2)

parameters

variable	*range*	*what?*
base	32bits	module base address
reg1	f ÷ 3ff	channel register 1
reg2	f ÷ 3ff	channel register 2

function

Read channel registers one for each module section

8.23 Read Enable Register

calling sequence

```
read_enable_adc9u(base,&ena1,&ena2)
```

parameters

variable	*range*	*what?*
base	32bits	module base address
ena1	8bits	enable register 1
ena2	8bits	enable register 2

function

Read enable registers one for each module section

Chapter 9

Tdc9uLib

A library for the VN1488S CAEN 64 Channel Multi TDC[10]

9.1 Reset

calling sequence

reset_all_tdc9u(base)

parameters

variable	*range*	*what?*
base	32bits	module base address

function

Reset the module in the starting configuration

9.2 Read Hardware Register

calling sequence

read_hard_tdc9u(base,&reg)

parameters

variable	*range*	*what?*
base	32bits	module base address
reg	16bits	hardware register

function

Read the hardware register

9.3 Write General Control Register 1

calling sequence

write_general_tdc9u(base,reg)

parameters

variable	*range*	*what?*
base	32bits	module base address
reg	8bits	general control register 1

function

Write the general control register 1

9.4 Write General Control Register 2

calling sequence

write_trigger_tdc9u(base,reg)

parameters

variable	*range*	*what?*
base	32bits	module base address
reg	1÷1ff	general control register 2

function

Write the general control register 2

9.5 Write Range

calling sequence

write_range_tdc9u(base,sel)

parameters

variable	*range*	*what?*
base	32bits	module base address
sel	8bits	range selected

function

Write pedestal adjustment offset current for all channels

9.6 Write Thresholds

calling sequence

write_threshold_tdc9u(base,low,high)

parameters

variable	*range*	*what?*
base	32bits	module base address
low	0÷ 1ff	low threshold
high	0÷ 1ff	high threshold

function

Write low and high thresholds, each bit is 15 counts of tdc channels

9.7 Write Channels Register

calling sequence

write_chreg_tdc9u(base,reg1,reg2)

parameters

variable	*range*	*what?*
base	32bits	module base address
reg1	f ÷ 3ff	channel register 1
reg2	f ÷ 3ff	channel register 2

function

Write channel registers one for each module section

9.8 Write Enable Register

calling sequence

write_enable_tdc9u(base,ena1,ena2)

parameters

variable	*range*	*what?*
base	32bits	module base address
ena1	8bits	enable register 1
ena2	8bits	enable register 2

function

Write enable registers one for each module section

9.9 Write Delay Value

calling sequence

write_delay_tdc9u(base,del)

parameters

variable	*range*	*what?*
base	32bits	module base address
del	8bits	delay value

function

Write an internal delay value for test purpose

9.10 Start start-stop Test

calling sequence

start_test_tdc9u(base)

parameters

variable	*range*	*what?*
base	32bits	module base address

function

Start a generation of a start-stop pulses with a delay set before on the Delay Register

9.11 Read Status Register

calling sequence

read_status_tdc9u(base,&reg)

parameters

variable	*range*	*what?*
base	32bits	module base address
reg	16bits	status register

function

Read the status register of the module

9.12 Increase Read Pointer

calling sequence

increase_pointer_tdc9u(base)

parameters

variable	*range*	*what?*
base	32bits	module base address

function

Increase the read pointer

9.13 Read Write and Read Pointers

calling sequence

read_pointer_tdc9u(base,&rpoint,&wpoint)

parameters

variable	*range*	*what?*
base	32bits	module base address
rpoint	4bits	read pointer
wpoint	4bits	write pointer

function

Read the read and the write pointers

9.14 Write Pedestal Memory

calling sequence

write_pedestal_tdc9u(base,ped)

parameters

variable	*range*	*what?*
base	32bits	module base address
ped[64]	12bits	pedestal array

function

Write the pedestal for each channel into pedestal memory

9.15 Read Pedestal Memory

calling sequence

read_pedestal_tdc9u(base,ped)

parameters

variable	*range*	*what?*
base	32bits	module base address
ped[64]	12bits	pedestal array

function

Read the pedestal for each channel from pedestal memory

9.16 Read event

calling sequence

read_event_tdc9u(base,&num,data)

parameters

variable	*range*	*what?*
base	32bits	module base address
num	$1 \div 40$	number of conversions
data[64]	32bits	data array

function

Read converted data from a module, num is the number of conversions, the array contains data for num locations

9.17 Read General Control Register 1

calling sequence

read_general_tdc9u(base,&reg)

parameters

variable	*range*	*what?*
base	32bits	module base address
reg	8bits	general control register 1

function

Read the general control register 1

9.18 Read General Control Register 2

calling sequence

read_trigger_tdc9u(base,&reg)

parameters

variable	*range*	*what?*
base	32bits	module base address
reg	1÷1ff	general control register 2

function

Read the general control register 2

9.19 Read Range

calling sequence

read_range_tdc9u(base,&sel)

parameters

variable	*range*	*what?*
base	32bits	module base address
sel	8bits	pedestal adjustment

function

Read range selected

9.20 Read Thresholds

calling sequence

read_threshold_tdc9u(base,&low,&high)

parameters

variable	*range*	*what?*
base	32bits	module base address
low	0÷ 1ff	low threshold
high	0÷ 1ff	high threshold

function

Read low and high thresholds, each bit is 15 or 120 counts of tdc channels depending on range selected

9.21 Read Channels Register

calling sequence

read_chreg_tdc9u(base,&reg1,&reg2)

parameters

variable	*range*	*what?*
base	32bits	module base address
reg1	f ÷ 3ff	channel register 1
reg2	f ÷ 3ff	channel register 2

function

Read channel registers one for each module section

9.22 Read Enable Register

calling sequence

read_enable_tdc9u(base,&ena1,&ena2)

parameters

variable	*range*	*what?*
base	32bits	module base address
ena1	8bits	enable register 1
ena2	8bits	enable register 2

function

Read enable registers one for each module section

Chapter 10

Disc9uLib

VN663 CAEN 32 Ch Discriminator Library[11]

10.1 Read High Thresholds

calling sequence

read_high_disc9u(base,tth)

parameters

variable	*range*	*what?*
base	32bits	module base address
tth[32]	8bits	high thresholds

function

Read the high thresholds

10.2 Write High Thresholds

calling sequence

write_high_disc9u(base,tth)

parameters

variable	*range*	*what?*
base	32bits	module base address
tth[32]	8bits	high thresholds

function

Write the high thresholds

10.3 Read Low Thresholds

calling sequence

read_low_disc9u(base,tth)

parameters

variable	*range*	*what?*
base	32bits	module base address
tth[4]	8bits	low thresholds

function

Read the low thresholds

10.4 Write Low Thresholds

calling sequence

write_low_disc9u(base,tth)

parameters

variable	*range*	*what?*
base	32bits	module base address
tth[4]	8bits	low thresholds

function

Write the low thresholds

10.5 Read Mode

calling sequence

read_mode_disc9u(base,mode)

parameters

variable	*range*	*what?*
base	32bits	module base address
mode	$0 \div 1$	mode 0= remote, 1=local

function

Read mode of module

Chapter 11

CorboLib

A library for the RCB8047 CORBO CES Read-Out Control Board[12]

11.1 Clear Channels Busy

calling sequence

clear_busy_corbo(base)

parameters

variable	*range*	*what?*
base	32bits	module base address

function

Clear all channels busy

11.2 Read Counters

calling sequence

read_count_corbo(base,count)

parameters

variable	*range*	*what?*
base	32bits	module base address
count[4]	32bits	channel counters

function

Read channel counters

11.3 Write Counters

calling sequence

write_count_corbo(base,count)

parameters

variable	*range*	*what?*
base	32bits	module base address
count[4]	32bits	channels count

function

Write channel counters

11.4 Read Dead Times

calling sequence

read_dead_corbo(base,dead)

parameters

variable	*range*	*what?*
base	32bits	module base address
dead[4]	32bits	channels dead time

function

Read channels dead time

11.5 Write Dead Times

calling sequence

write_dead_corbo(base,dead)

parameters

variable	*range*	*what?*
base	32bits	module base address
dead[4]	32bits	channels dead time

function

Write channels dead time

11.6 Read Status Register

calling sequence

read_status_corbo(base,status)

parameters

variable	*range*	*what?*
base	32bits	module base address
status[4]	32bits	channels status register

function

Read channels status register

11.7 Write Status Register

calling sequence

write_status_corbo(base,status)

parameters

variable	*range*	*what?*
base	32bits	module base address
status[4]	32bits	channels status register

function

Write channels status register

11.8 Test Input

calling sequence

test_input_corbo(base,channel)

parameters

variable	*range*	*what?*
base	32bits	module base address
channel	32bits	channel number

function

Simulate an input trigger on the chosen channel

Chapter 12

CbdLib

A library for the CBD8210 CES Branch Driver[13]

12.1 Initialize

calling sequence

copen(base)

parameters

variable	*range*	*what?*
base	32bits	module base address

function

Initialize the base address and CSR address

12.2 Other CBD Esone

All parameters are integer except data for cssa access that is short.

calling sequence

```
cdreg(&ext,b,c,n,a)
cdlam(&lam,b,c,n,a,&inta)
x=cccc(ext)
x=cccz(ext)
x=ccci(ext,l)
x=cccd(ext,l)
x=cclc(lam)
x=cclm(lam,l)
x=ctci(ext,&l)
x=ctcd(ext,&l)
x=ctgl(ext,&l)
x=ctlm(ext,&l)
x=ctlm(ext,&l)
x=cssa(f,ext,&data,&q)
x=cfsa(f,ext,&data,&q)
```

For more information see CAMAC Esone[3].

Chapter 13

BuserrorLib

13.1 Starting Error Trapping

calling sequence

`set_error_trap()`

function

Start the Bus Error Trapping

13.2 End Error Trapping

calling sequence

`reset_error_trap()`

function

End the Bus Error Trapping

Appendix A

Installation and use of LibVME

How to install the VME library

- create in your working area a directory named **LibVME** and add it at your PATH default
- copy **vme_lib.h**, **Makefile** and all the **.c** files in this directory
- run **make** that generate the library **libVME.a**

How to use the library

- include in your main the file "**vme_lib.h**" and the file **<setjmp.h>** whether you want the error trapping
- link your program with the option **-lVME**

Appendix B

Space requirements

Filename	Usage Source	Usage Object	
Makefile		0.94 kb	
vme_lib.h	0.88 kb		
libVME.a		64.1 kb	
corbo_lib	5.29 kb	6.8 kb	
cbd_lib	8.14 kb	12.46 kb	
scaler_lib	3.99 kb	5.36 kb	
plu_lib	4.6 kb	5.5 kb	
pattern_lib	2.75 kb	4.11 kb	
ioreg_lib	4.78 kb	6.89 kb	
disc9u_lib	3.16 kb	4.46 kb	
delgate_lib	3.48 kb	4.53 kb	
adc9u_lib	13.44 kb	17.38 kb	
tdc9u_lib	12.96 kb	10.99 kb	
buserror_lib	2.13 kb	6.04 kb	
TOTAL:	65.6 kb	149.56 kb	215.2 kb

Appendix C

Operation and error codes

Operation code

Error code	Definition
0	Subroutine finished without problems
-1	A Bus Error occured reading or writing module
-2	Illegal addressing parameter
-3	Illegal range parameter

Operation code

Operation code	Definition
0	General access without LibVME
1	Write access on a VME module
2	Read access on a VME module
3	Access on a CAMAC module

Appendix D

Software examples

D.1 Bus Error Trapping

The following program trap a bus error coming from a library call.

```
/*******************************/
/* Bus error trapping example */
/*******************************/

#include <stdio.h>
#include <setjmp.h>
#include "vme_lib.h"

#define PATT_BASE   0xde888800

int main() {

/** Error handling **/
   set_error_trap();

   if (setjmp(jmpVME)<0) exit(-1);

   printf("Resetting Pattern Unit\n");
   reset_all_pattern(PATT_BASE);

   exit(0);
}
```

If access fails the following message is printed out:

Resetting Pattern Unit

FATAL ERROR: Bus$Error occured during Write at location de888810.

D.2 Programming PLU

The following program calculate and set-up a section of a PLU with a set of logical functions chosen.

```
/***************************/
/* Programming PLU example */
/***************************/

#include <stdio.h>
#include "vme_lib.h"

#define BASE_PLU 0xde022000
#define SECTION_A 0x0
#define STROBED_PLU 0xc

unsigned vector[256];
unsigned vector1[256];

int main() {
   void calc_func_plu();
   int k, plu_mode;

   calc_func_plu();

   printf("Programming PLU section A\n");
   write_function_plu(BASE_PLU,SECTION_A,vector);
   read_function_plu(BASE_PLU,SECTION_A,vector1);

   for(k=0;k<=255;k++) {
      if (vector[k] != vector1[k] ) {
         printf ("PROGRAMMATION of PLUV not correctly written\n");
         exit(-1)
      }
   }

   printf("Configuring PLU\n");
   write_config_plu(BASE_PLU,STROBED_PLU);
   read_config_plu(BASE_PLU,&plu_mode);

   if (plu_mode != STROBED_PLU) {
      printf ("OPERATING MODE of PLUV not correctly written\n");
```

```
      exit(-1);
   }

   exit(0);
}

/* Calculate Plu functions */

void calc_func_plu() {
   int m,k,l,ia;
   int i[8],o[8];

   for(m=0;m<=255;m++) {
      l=1;
      for(k=0;k<n_in;k++) {
         i[k]=(m & l) /l;
         l=l*2;
      }

/* functions to write in memory */
      o[0]=i[5]|(i[4]&i[2]);
      o[1]=i[5];
      o[2]=!i[0];
      o[3]=i[3];
      o[4]=(i[4]&i[5])|(!i[2]);
      o[5]=i[0];
      o[6]=i[6]&i[7];
      o[7]=i[5];

      l=1;
      ia=0;
      for(k=0;k<n_out;k++) {
         ia=ia+(o[k] & 1) * l;
         l=l*2;
      }
      vector[m]=ia;
   }
}
```

D.3 Trigger Example

The following program is an example of looping for trigger on CORBO module.

```
/*******************/
/* Trigger example */
/*******************/

#include <stdio.h>
#include "vme_lib.h"

#define BASE_CORBO 0xdebbbb00
#define N_EVENTS 100
#define COUNT_BUSY 0x20
#define ZERO 0x0

unsigned stat[4];
unsigned stat1[4];

int main() {
   int k,j;

   for(k=0;k<4;k++) stat=COUNT_BUSY;

   write_status_corbo(BASE_CORBO,stat);
   read_status_corbo(BASE_CORBO,stat1);

   for(k=0;k<4;k++) {
      if (stat[k] != stat1[k] ) {
         printf ("STATUS of CORBO not correctly written\n");
         exit(-1)
      }
   }

   for(k=0;k<4;k++) stat=ZERO;

   write_count_corbo(BASE_CORBO,stat);
   read_count_corbo(BASE_CORBO,stat1);

   for(k=0;k<4;k++) {
      if (stat[k] != stat1[k] ) {
         printf ("COUNTS of CORBO not correctly written\n");
```

```
            exit(-1)
        }
    }

    for(k=0;k<N_EVENTS;k++) {
        clear_busy_corbo(BASE_CORBO);
        trigger=0;

        while (trigger==0) {
            read_count_corbo(BASE_CORBO,stat1);
            for(j=0;j<4;j++) {
                if (stat1[j] == stat[j]+1)) {
                    stat[j]=stat1[j];
                    trigger=1;
                }
            }
        }
        printf("\n Loop: %d\n",k);
        for(j=0;j<4;j++) printf("Count ch %d = %d\n",j,stat[j]);
    }

    exit(0);
}
```

D.4 CBD Access

The following program is an example of how to access CAMAC module through the CBD.

```
/**********************/
/* CBD access example */
/**********************/

#include <stdio.h>
#include "lib_VME.h"

#define BASE_CBD 0xde800000
#define FALSE -1
#define TRUE 0

int main() {
   int creg,n=0,c,f,data;

   copen(BASE_CBD);

   printf("Resetting Branch Driver\n");
   c=0;
   f=9;
   n=29;
   cdreg(&creg,b,c,n,a);
   cssa(f,creg,&data_s,&q);

   printf("Resetting crate 1\n");
   c=1;
   n=24;
   cdreg(&creg,b,c,n,a);
   cccz(creg);

   printf("Resetting inhibit on crate 1\n");
   ccci(creg,FALSE);

   printf("Enabling demand on crate 1\n");
   cccd(creg,TRUE);

   /* reset a scaler in slot 18 */
   f=9;
```

```
    n=18;
    cdreg(&creg,b,c,n,a);
    data=0;
    cfsa(f,creg,&data,&q);

    exit(0);
}
```

Bibliography

[1] CES, RTPC 8067 Technical Manual (1995)

[2] Lynx Real-Time Systems, LynxOS for PowerPC (1995)

[3] D. Burckart *et al.*, Standard CAMAC Subroutines, CERN DD/OC/80-4

[4] CAEN, V 560 Technical Information Manual (1993)

[5] CAEN, V 486 Technical Information Manual (1993)

[6] CAEN, V 259 Technical Information Manual (1991)

[7] CAEN, V 513 Technical Information Manual (1993)

[8] CAEN, V 495 Technical Information Manual (1993)

[9] CAEN, VN 1465 S Technical Information Manual (1996)

[10] CAEN, VN 1488 S Technical Information Manual (1996)

[11] CAEN, VN 663 Technical Information Manual (1996)

[12] CES, RCB 8047 Corbo User's Manual (1993)

[13] CES, CBD 8210 User's Manual