

# EPICS VME Crate Monitor Software Design Note

7.2.39.19 Rev. 0

Date: 2003-01-09

**Copyright 2003, Canadian Light Source Inc.** This document is the property of Canadian Light Source Inc. (CLS). No exploitation or transfer of any information contained herein is permitted in the absence of an agreement with CLS, and neither the document nor any such information may be released without the written consent of CLS.

***Original on File – Signed by:***

Prepared by: \_\_\_\_\_  
R. Berg

Reviewed by: \_\_\_\_\_  
T. Wilson

Reviewer by: \_\_\_\_\_  
G. Wright

Approved by: \_\_\_\_\_  
E. D. Matias

Canadian Light Source, Inc.  
107 North Road  
University of Saskatchewan  
Saskatoon, Saskatchewan Canada

## REVISION HISTORY

---

<b><i>Revision</i></b>	<b><i>Date</i></b>	<b><i>Description</i></b>	<b><i>Author</i></b>
A	Oct 8 2002	Initial Draft	R. Berg
0	Jan 9 2003	Issued rev 0	R. Berg

---

# Table of Contents

	<u>Page</u>
<b>1.0 Introduction.....</b>	<b>4</b>
Abbreviations .....	4
<b>2.0 Linux, Epics, SIS1100 driver Purpose and Method of Interaction.....</b>	<b>5</b>
SIS1100 Driver .....	5
Linux .....	6
EPICS .....	7
Putting them all together.....	7
<b>3.0 Memory Mapping and register access .....</b>	<b>8</b>
<b>4.0 VME Crate configuration, the .cmd file .....</b>	<b>10</b>
<b>5.0 Software Initialization .....</b>	<b>12</b>
Init_SIS1100 .....	13
Init_110bl.....	13
<b>6.0 Database Files.....</b>	<b>14</b>
<b>7.0 Appendix A .....</b>	<b>15</b>
7.1.1 Supported VME Cards.....	15
7.1.2 Exampleinclude.dbd.....	16
7.1.3 Crate Configuration Files.....	17

## 1.0 Introduction

The purpose of the crate monitor software is to easily interface a PC computer running Linux to one or more VME crates via fiber optic link. The fiber optic link is achieved by using a pair of cards SIS3100(VME) and SIS1100(PCI) installed into their respective back planes and connected with a fiber optic cable. Once connected, a Linux application is able to map the memory locations of any VME board installed in the VME crate into the application memory space of the Linux application. This allows the Linux application to access the registers or FIFO's of a particular board installed in a VME crate simply by reading a pointer to a memory location. The actual transmission and reception of data across the fiber optic link is transparently taken care of by the SIS1100 PCI driver.

This document will discuss the hardware but primarily the software required to achieve the PC to VME link via Fiber. Once implemented, this software/hardware combination will provide a reusable and scalable solution for interconnecting the CLS's feedback and control system. The discussion of software will also include the integration of the EPICS control system and its use of the Crate\_monitor software.

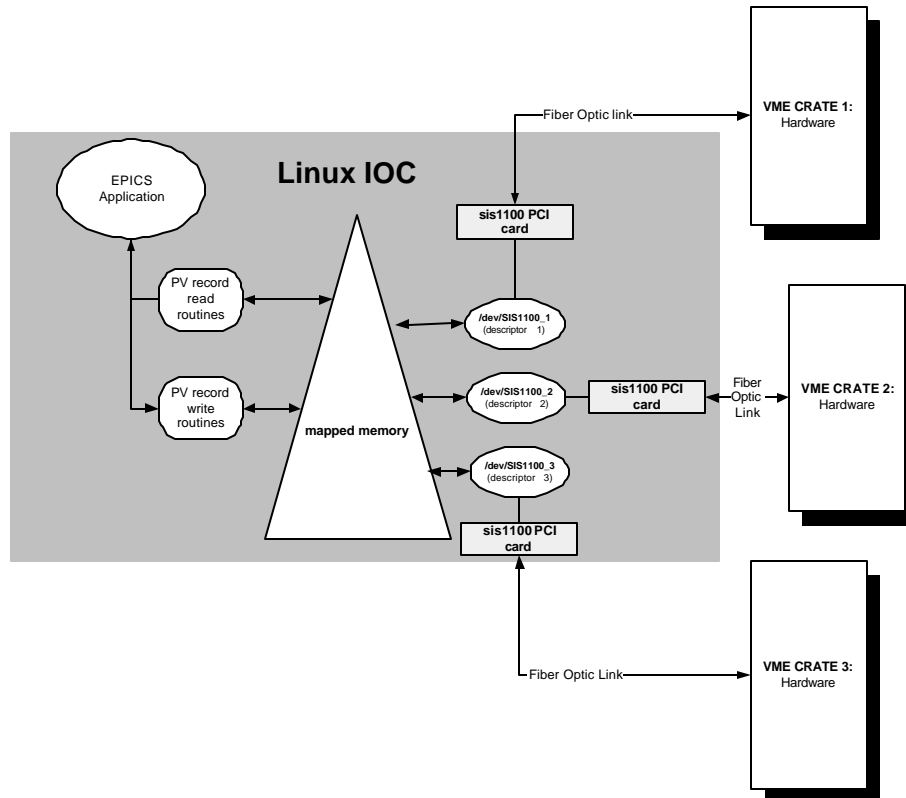
### Abbreviations

- BPM Beam Position Monitor/s
- ADC Analog to Digital Converter
- EPICS the software control system used at the CLS to connect the operator to the diagnostic or control hardware of the Synchrotron
- IOC Input Output Computer

## 2.0 Linux, Epics, SIS1100 driver Purpose and Method of Interaction

### SIS1100 Driver

Provides an abstraction layer between the user software, which needs to access a VME board on a crate and the actual board in that crate. The system is organized in the following manner.

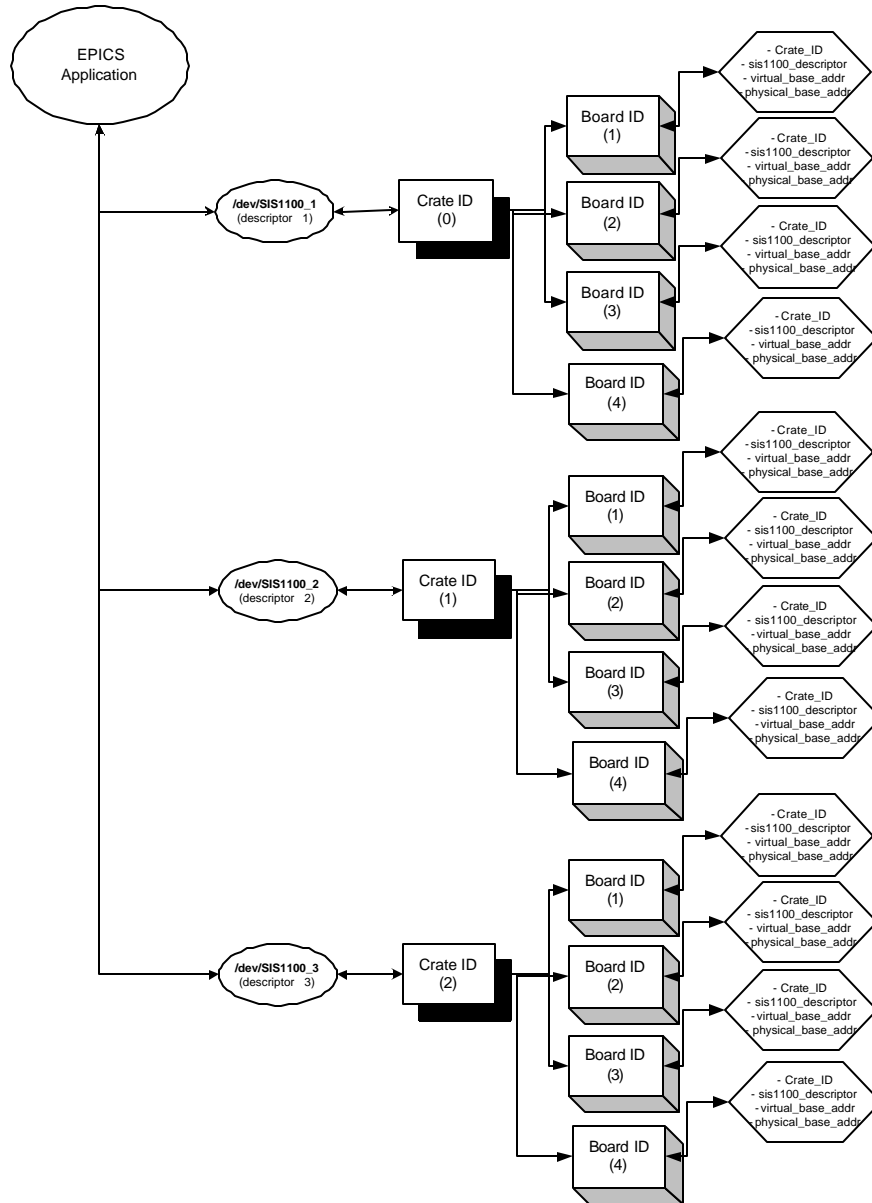


The EPICS application opens the SIS1100 devices as files and requests that memory ranges that define each VME crate be mapped into the user space of the EPICS application. Once the mapping has taken place, each EPICS Process Variable (PV) will make note of its desired memory offset from the mapped address and store this information in its device private structure. In this fashion, the read routine for a process variable can retrieve the correct location in the mapped memory in order to read the data for that PV.

The memory mapping definition for each VME crate is defined by what boards are installed in that crate. Care must be taken not to install VME boards with memory addresses that overlap. Each VME crate will have its board configuration outlined in its .cmd file, there will be one cmd file per VME crate.

Each SIS1100 device can support up to as many VME boards that fit into a single VME crate. The only requirement is that each board be defined in the cmd file for that crate. The individual board data is indexed by the following unique identifiers: 1) SIS1100 device file descriptor 2) crate id 3) board id. The data that is stored for each individual board contains the mapped base address for that board, and from that address a particular register offset can easily be calculated and dereferenced.

Individual crate data is organized in the following manner:



## Linux

Provides an excellent environment for software development as well as an IOC operating system. Take advantage of all of the GNU tools for compiling and debugging. Well supported and documented. The only question is if the generic Linux kernel will provide enough processing

performance given the tasks required. All indications at the time of this authoring are that the generic Linux kernel will suffice given a fast CPU and related hardware.

Should the generic Linux fail to meet the requirements of most of the applications used at the CLS, a real time version of the Linux kernel will be explored.

## **EPICS**

The control systems layer abstraction. The VME hardware is abstracted into process variables accessible anywhere on the network by any supported platform. The VME monitoring software provides a further abstraction such that the user doesn't need to worry about the actual transmission and reception of data across the fiber link and can in fact consider the VME hardware to be local to the IOC, when in fact it may be located 100's of meters away.

EPICS will be used as the bridge that will allow access to VME hardware via process variables. Once available through EPICS channel access calls, the beam control and diagnostic feedback data can be retrieved anywhere on the control network, and in some cases the general office network (feedback).

## **Putting them all together**

The main goal of this approach is to end up with a single binary file that is configured by text files (.cmd files) that requires no re-compilation in order to configure the installation of a new board into an existing VME crate. Each crate will support any combination of VME board/s. Multiple instances of SIS1100 fiber link boards within a single IOC is supported, allowing a single IOC to control multiple VME crates. The fact that a single CPU will control multiple VME crates means that care must be taken when using the system described above in time critical situations.

### 3.0 Memory Mapping and register access

The memory mapping approach provides the following features:

- A uniform access to all software as if it was local
- The configuration for a particular board can be easily setup through the crate configuration file. This configuration file is an ASCII based text file therefore no compilation is required.
- There is a standard way of reading and writing information from any kind of hardware located in a VME crate. All register read/writes and or FIFO reads are executed from standard pointer de-references common to most C code.

The use of the SIS1100/SIS3100 fiber link boards enable an EPICS application to memory map the full register range of the VMIVME-2536 Digital I/O board. To map the memory of a given board, a call is made to the SIS1100 driver with the actual physical base address of the desired board as it exists in the VME crate. The driver then communicates with the SIS3100 VME master board to setup the physical address of the desired board into one of its memory mapping registers.

The driver handles all protocol and communication across the fiber link. Once the call to map the memory completes, the driver returns with a virtual base address for the board, this virtual base address corresponds to the physical base address of the board.

Once the mapping has been accomplished, the 32 bit input and 32 bit output registers of the VMIVME-2536 Digital I/O board are accessed via simple memory reads and writes to the correct offset address of the virtual base address.

A practical example of this relationship is given next. Given the following which is taken from the documentation of the VMIVME-2536 manual:

<u>Relative Address</u>	<u>Register Name</u>	<u>Num Bits</u>	<u>Bit Range</u>
\$00	Board ID	16	Na
\$02	Control and Status Register	16	Na
\$04	Input Data register 0	8	31 – 24
\$05	Input Data register 1	8	23 – 16
\$06	Input Data register 2	8	15 – 08
\$07	Input Data register 3	8	07 – 00
\$08	Output Data register 0	8	31 – 24
\$09	Output Data register 1	8	23 – 16
\$0A	Output Data register 2	8	15 – 08
\$0B	Output Data register 3	8	07 – 00
\$0C	Reserved	NA	

Figure 2. VMIVME-2536 register map

If the physical base address of the VMIVME-2536 board as it exists in the VME crate was **0x00F00000**. The EPICS software would make a call to memory map the address **0X00F0000**, the virtual address returned by the memory map call is (for example) a *Virtual Base Address (VBA)* of **0x40001000**. Now that the memory mapping has completed, we can access the **Control and Status register** with a read or write to:

(VBA + Control and Status Register offset) = 0x40001000 + 0x02 = **0x40001002**

All of the registers outlined above can be accessed in the same manner.

**Note** that if the register is designated *READ ONLY* or *WRITE ONLY* that behavior remains consistent once mapped. A read to a *WRITE ONLY* register returns the value 0xFFFFFFFF (for a 32 bit register), and so on.

## 4.0 VME Crate configuration, the .cmd file

The discussion that follows will deal only with specific additions the cmd file format to achieve the VME crate configuration, to understand the EPICS cmd file in general please refer to EPICS documentation.

In order for the software to keep track of which crate is connected to which SIS1100 interface and further to which VME board within that crate it wants to access, the configuration specific data for each crate is organized in such a way that the index's are used to keep track of the configuration data structure. A VME crate cmd file assumes the following organization.

### **Crate\_ID -> Board\_ID->Configuration data**

Where the configuration data is defined as follows:

```
typedef struct {  
    int                crate_id;  
    int                SIS1100_descr;  
    volatile u_int8_t* virtual_base_addr;  
    u_int32_t          physical_base_addr;  
}base_addrs_st;
```

The data is organized into a two dimensional array of pointers to *base\_addr\_st* structures. These structures contain all of the information required to map the memory of the VME board as well as access any register or FIFO that exists on that board.

#### ***crate\_id;***

This is the Crate ID that was passed in from the cmd file

#### ***SIS1100\_descr;***

This is the file descriptor number that will be was returned from an OPEN() file command when the driver for this VME crate was initialized.

#### ***virtual\_base\_addr;***

This is the address in the memory space of the EPICS application that was returned by the call to map\_memory. The virtual\_base\_addr is to be used when accessing that board from the EPICS software.

An example of a crate configuration file that contains 4 VME boards is as follows:

```
dbLoadDatabase("../db/example.dbd",0,0)
registerRecordDeviceDriver(pdbbase)

#This is the first board, a VME digital I/O board that has a physical base address of 0x0070 0000
dbLoadRecords("../db/VMIVME-2536Rd.db","crate_id=1,board_name=VMi2536-2403:00,board_id=0,board_base=0x00700000")
dbLoadRecords("../db/VMIVME-2536Wr.db","crate_id=1,board_name=VMi2536-2403:00,board_id=0,board_base=0x00700000")

#This is the second board, a VME 24 bit ADC board that has a physical base address of 0x00D0 0000
dbLoadRecords("../db/ICS110bIRd.db","crate_id=1,board_name=ICSADC-2403:00,board_id=1,board_base=0x00D00000")
#This is the second board, a VME 24 bit ADC board that has a physical base address of 0x00D0 0000
dbLoadRecords("../db/ICS110bIFifo.db","crate_id=1,board_name=2403BPM1-14:,board_id=1,board_base=0x00D00000,event_id=110")

dbLoadRecords("../db/ICS110bIRd.db","crate_id=1,board_name=ICSADC-2403:01,board_id=2,board_base=0x00D00000")
dbLoadRecords("../db/ICS110bIFifo.db","crate_id=1,board_name=ICSADC-2403:01,board_id=2,board_base=0x00D00000,oversample=32")

dbLoadRecords("../db/v820Rd.db","crate_id=1,board_name=V820-2403:00,board_id=3,board_base=0xEE000000")
dbLoadRecords("../db/v820Wr.db","crate_id=1,board_name=V820-2403:00,board_id=3,board_base=0xEE000000")

# init_sis1100(crate_id, path to device file)
init_sis1100(1, "/dev/sis1100_3")

# map_memory(crate_id, board_id, physical_base_address)
map_memory(1, 0, 0x00700000)
map_memory(1, 1, 0x00D00000)
map_memory(1, 2, 0x00D00000)
map_memory(1, 3, 0xEE000000)

#init_110b(crate_id, board_id, sampling_rate in Khz, set for internal(0) or external(1) clocking source,0 internal Acquire: 1 External Acquire)
init_110b(1, 1,35.0, 0, 1)
#configure this 110b for external clocking source
#init_110b(crate_id, board_id, sampling_rate in Khz, set for internal(0) or external(1) clocking source,0 internal Acquire: 1 External Acquire)
init_110b(1, 2, 100.0, 1, 0)

ioclnit()
```

The references to "init\_110b" refer to a cpp routine that accessible through the EPICS cmd file, for further information on how this is accomplished please refer to the EPICS documentation.

## 5.0 Software Initialization

As with any EPICS application, the cmd files are read and the statements within them executed.

At the heart of the VME crate monitor software is the ability of the EPICS application to map the memory of a VME crate into the user space of that EPICS application. Without this ability the interaction between EPICS and the hardware would be considerably more proprietary and much less generic and modular.

In order to include memory mapping as part of the initialization process, routines have been written to accomplish the task of connecting the address space of a VME board to the EPICS application memory space. A single memory mapping routine is called directly from the cmd file in the following manner,

**map\_memory(crate\_id, board\_id, physical\_base\_address)**

Arguments to the routine are:

### **Crate\_id**

unique id number of the desired crate, **each cmd file MUST use a different crate\_id number.**

### **Board\_id**

unique VME board id of the desired board, **each board within this cmd file MUST use a different board\_id number.**

### **Physical\_base\_address**

the actual physical base address of that board that it is set to in the VME crate, **each board in this VME crate MUST have a unique physical base address.**

Once the call to map\_memory has been made, the mapped base address of the board can then be found in the **board\_map** array indexed by the crate\_id and board\_id. All of the information required to process the PV is stored in the device private structure for that PV and extracted for use in the read or write routines.

There are other routines that have been created such that they can also be called directly from the cmd file in order to initialize hardware local to the IOC as well as hardware located in a VME crate.

## **Init\_SIS1100**

One example of initialization required local to the IOC is the “**init\_SIS1100**” routine, this routine handles the opening of a particular 1100 device and returning a valid file descriptor that will be used by the rest of the software when interacting with the device. The arguments to `init_SIS1100` are `crate_id` and the path to device file:

Ex: `init_SIS1100(1, "/dev/SIS1100_3")`

## **Init\_110bl**

When a specific initialization is required by a VME board the user can create a cpp routine to accomplish the task. One example is for the 24 bit ADC currently used in the booster ring for beam position measurements. On boot the board requires a calibration and sample rate initialization sequence, this is accomplished with the `init_110bl` routine. The arguments to `init_110bl` are `crate_id`, `board_id`, sampling rate in Khz, set for internal (0) or external(1) clocking source, 0 internal Acquire: 1 External Acquire:

Ex: `init_110bl(1, 2, 100.0, 1, 0)`

Here, the 110bl 24 bit ADC will be configured with the following settings:

- `crate_id` of 1
- `board_id` of 2
- Sampling rate of 100 Khz
- External clocking source to the board
- Control of when to acquire will be set by software (internal)

## 6.0 Database Files

The memory map database files are comprised of three types, **mmapRegRd**, **mmapRegWr** and memory map FIFO database files. The read and write database files are generic 16/32 bit read and write database files that are meant to be used when interacting with PV's that are 16/32 bit registers on a VME board. The input (read: **INP**) and output (write: **OUT**) field parameters for the mem map read/write database files are scanned as follows at record initialization:

### INP:

```
sscanf (pmmmapReg->inp.value.instio.string, "%i%i%x%x%x",
        &crate_id, &board_id, &board_base, &reg_offset, &reg_mask)
```

Here is an example of the mmapRegRd record Db type declaration for a read PV called "Testreg" on a CAEN V820 Scalar VME board.

```
record(mmapRegRd, "$(board_name):Rd_Testreg")
{
    field(DESC, "Scalar Testreg read")
    field(DTYP, "V820READ")
    field(INP, "@$(crate_id)$(board_id)$(board_base) 0x1080 0xFFFFFFFF")
    field(EGU, "Counts")
    ....
}
```

### OUT:

```
sscanf (pmmmapReg->out.value.instio.string, "%i%i%x%x%x",
        &crate_id, &board_id, &board_base, &offset, &reg_mask)
```

Here is an example of the mmapRegWr record Db type declaration for a write PV called "Testreg" on a CAEN V820 Scalar VME board.

```
record(mmapRegWr, "$(board_name):Wr_Testreg")
{
    field(DESC, "V820:Testreg write")
    field(DTYP, "V820WRITE")
    field(OUT, "@$(crate_id)$(board_id)$(board_base) 0x1080 0xFFFFFFFF")
    ....
}
```

## 7.0 Appendix A

### 7.1.1 Supported VME Cards

Currently the list of supported cards includes:

- **ICS** 110bl 24 bit ADC
- **CAEN** V820 32 channel scalar
- **CAEN** V265 sampling ADC
- **VMIC** 2536 digital IO module

Pending support for:

- **CAEN** V792 charge integrating ADC

## 7.1.2 Exampleinclude.dbd

```
include "base.dbd"
include "mmapRegWrRecord.dbd"
include "mmapRegRdRecord.dbd"
include "ICS110blFifoRecord.dbd"
include "ICS110blArrayFifoRecord.dbd"
include "aaiRecord.dbd"
include "v265adcFifoRecord.dbd"

device(mmapRegRd, INST_IO, devMmapRegRead, "V820READ")
device(mmapRegWr, INST_IO, devMmapRegWrite, "V820WRITE")

device(mmapRegRd, INST_IO, devMmapRegRead, "ICS110blREAD")
device(mmapRegWr, INST_IO, devMmapRegWrite, "ICS110blWRITE")

device(mmapRegRd, INST_IO, devMmapRegRead, "VMIVME2536READ")
device(mmapRegWr, INST_IO, devMmapRegWrite, "VMIVME2536WRITE")

device(mmapRegRd, INST_IO, devMmapRegRead, "V792READ")
device(mmapRegWr, INST_IO, devMmapRegWrite, "V792WRITE")

device(ICS110blFifo, INST_IO, devICS110blFifo, "ICS110BLFIFO")

device(aai, INST_IO, devICS110blArrayFifo, "ICS110BLARRAYFIFO")

device(mmapRegRd, INST_IO, devMmapRegRead, "V265READ")
device(mmapRegWr, INST_IO, devMmapRegWrite, "V265WRITE")

device(v265adcFifo, INST_IO, devV265adcFifo, "V265FIFO")
```

## 7.1.3 Crate Configuration Files

### 7.1.3.1 CR006.CMD

```
dLoadDatabase("../dbd/example.dbd",0,0)
registerRecordDeviceDriver(pdbbase)
#the event_id is used as the identifier for a callback event
dbLoadRecords("../db/v792Rd.db","crate_id=2,board_name=0006QCT:00,board_id=0,board_base=0x00100000,event_id=20")
dbLoadRecords("../db/v792Wr.db","crate_id=2,board_name=0006QCT:00,board_id=0,board_base=0x00100000,event_id=20")

dbLoadRecords("../db/v792Rd.db","crate_id=2,board_name=0006QCT:01,board_id=1,board_base=0x00400000,event_id=21")
dbLoadRecords("../db/v792Wr.db","crate_id=2,board_name=0006QCT:01,board_id=1,board_base=0x00400000,event_id=21")

# init_sis1100(crate_id, path to device file)
init_sis1100(2, "/dev/sis1100_1")

# map_memory(crate_id, board_id, physical_base_address)
map_memory(2, 0, 0x00100000)
map_memory(2, 1, 0x00400000)
```

### 7.1.3.2 CR2401.CMD

```
dLoadDatabase("../dbd/example.dbd",0,0)

registerRecordDeviceDriver(pdbbase)

dbLoadRecords("../db/VMIVME-2536Rd.db","crate_id=0,board_name=VMI2536-2401:00,board_id=0,board_base=0x00700000")

dbLoadRecords("../db/VMIVME-2536Wr.db","crate_id=0,board_name=VMI2536-2401:00,board_id=0,board_base=0x00700000")

dbLoadRecords("../db/ICS110blRd.db","crate_id=0,board_name=2401 - ICSADC:00,board_id=1,board_base=0x00D00000")

dbLoadRecords("../db/ICS110blArrayFifo.db","crate_id=0,board_name=2401BPM15-28:,board_id=1,board_base=0x00D00000,event_id=11")

dbLoadRecords("../db/v820Rd.db","crate_id=0,board_name=V820-2401:00,board_id=2,board_base=0xEE000000")
dbLoadRecords("../db/v820Wr.db","crate_id=0,board_name=V820-2401:00,board_id=2,board_base=0xEE000000")

# init_sis1100(crate_id, path to device file)
init_sis1100(0, "/dev/sis1100_2")

# map_memory(crate_id, board_id, physical_base_address)
map_memory(0, 0, 0x00700000)
map_memory(0, 1, 0x00D00000)
map_memory(0, 2, 0xEE000000)

#init_110bl(crate_id, board_id, sampling_rate in Khz, set for internal(0) or external(1) clocking source,0 internal Acquire: 1 External Acquire)
init_110bl(0, 1, 35.0, 0, 1)
```

### 7.1.3.3 CR2403.CMD

```
dbLoadDatabase("../db/example.dbd",0,0)

registerRecordDeviceDriver(pdbbase)

dbLoadRecords("../db/VMIVME-2536Rd.db","crate_id=1,board_name=VMI2536-2403:00,board_id=0,board_base=0x00700000")

dbLoadRecords("../db/VMIVME-2536Wr.db","crate_id=1,board_name=VMI2536-2403:00,board_id=0,board_base=0x00700000")

dbLoadRecords("../db/ICS110blRd.db","crate_id=1,board_name=ICSADC-2403:00,board_id=1,board_base=0x00D00000")

dbLoadRecords("../db/ICS110blArrayFifo.db","crate_id=1,board_name=2403BPM1-14:,board_id=1,board_base=0x00D00000,event_id=110")

dbLoadRecords("../db/ICS110blRd.db","crate_id=1,board_name=ICSADC-2403:01,board_id=2,board_base=0x00D00000")

dbLoadRecords("../db/ICS110blFifo.db","crate_id=1,board_name=ICSADC-2403:01,board_id=2,board_base=0x00D00000,oversample=32")

dbLoadRecords("../db/v820Rd.db","crate_id=1,board_name=V820-2403:00,board_id=3,board_base=0xEE000000")

dbLoadRecords("../db/v820Wr.db","crate_id=1,board_name=V820-2403:00,board_id=3,board_base=0xEE000000")

# init_sis1100(crate_id, path to device file)
init_sis1100(1, "/dev/sis1100_3")

# map_memory(crate_id, board_id, physical_base_address)
map_memory(1, 0, 0x00700000)
map_memory(1, 1, 0x00D00000)
map_memory(1, 2, 0x00D00000)
map_memory(1, 3, 0xEE000000)

#init_110bl(crate_id, board_id, sampling_rate in Khz, set for internal(0) or external(1) clocking source ,0
internal Acquire: 1 External Acquire)
init_110bl(1, 1, 35.0, 0, 1)

#configure this 110bl for external clocking source
#init_110bl(crate_id, board_id, sampling_rate in Khz, set for internal(0) or external(1) clocking source,0
internal Acquire: 1 External Acquire)
init_110bl(1, 2, 100.0, 1, 0)

#make the call to iocinit(), this is only called once by the last cmd file in the command line argument
iocInit()
```