EVR(EVG) 설정 메뉴얼

류수

October 5, 2015

1 도입

본 메뉴얼에서 다루는 내용은 EVG, EVR 로 구성되는 타이밍 시스템이며 따라서 EVG, EVR, MVEM6100 모듈을 하드웨어와 소프트웨어로 구성하는 방법과 제어하는 방법에 대해서 이야기한다. 타이밍 시스템에 관련된 일반적인 지식을 정리해놓은 문서는 다음(*)을 참고한다.

본 매뉴얼에서는 사용자가 이미 vxWorks와 EPICS base를 설치하였다고 가정한 상태에서 설명한다. 2장에서는 시스템 구성을 위한 요구사항들에 대해서 정리해놓았다. 여기서 하드웨어 구성과 소프트웨어 구성으로 둘로 나누었는데, 본 메뉴얼에서는 편의상 VxWorks를 이용하여 MVME6100 모듈에 VME-EVR을 인식하는 과정까지를 하드웨어 구성이라고 부르고, mrfIoc2를 이용하여 VME-EVR을 제어하는 과정을 소프트웨어 구성이라고 부르겠다. 3장과 4장에서는 하드웨어 구성과 소프트웨어 구성에 대해서 설명한다.

2 시스템 구성 요소 및 요구사항들

타이밍 시스템은 타이밍 모듈과 네트워크로 구성되어 있다. 타이밍 모듈은 MRF 사의 EVG, EVR 제품을 사용하며 폼펙터는 VME이다. 이 모듈들은 VME crate에 설치되며 VME 모듈을 제어하기 위해서 VME controller인 Emerson 사의 MVME6100을 사용한다. MVME6100에는 VxWorks 6.9를 OS로 사용한다. EPICS의 모듈 중의 하나인 mrfIoc2를 사용하여 EVG와 EVR을 제어한다. 네트워크는 광케이블로 구성한다.

2.1 MVME6100

MVME6100에는 모토로라 사의 MPC 7457 칩이 탑재되어 있으며 DDR2 512MB 외부메모리를 가지고 있고 2개의 64MB NVRAM 을 플래시메모리 영역으로 가지고 있다. 여기서 사용된 MVEM6100 모듈의 정보는 다음과 같다.

2.2 MRF 제품들

Micro Finland Research Oy. 사의 VME-EVG-230, VME-EVR-230 두개의 VME 타입의 모듈을 사용한다. 이 메뉴얼에서는 VME-EVR-230 모듈을 사용했으며 펌웨어 버전은 0x12000006 이다.

2.3 소프트웨어들

VxWorks 6.9 는 Real-Time Operating System 중의 하나로 프로세스간의 determinism 에 초점을 두고 설계된 OS 이다. 사업단에서는 2개의 Node-locked 라이센스를 보유하고 있다. [1] 하나는 ctrlpc0 다른 하나는 ctrlpc3에 설치되어 있으며 각각 32비트, 64비트 환경에 설치되어 있다. 새롭게 설치를 한다면 32비트 환경에서 설치하는 것이 여러모로 유리하다.

EPICS 버전은 3.14.9 이상이 설치되어 있어야 한다. 이는 mrfIoc2의 요구사항이다.

3 하드웨어 구성

들어가기 앞서, 지금 사용하는 MVME6100은 firmware 변경없이 수입업체(에이스트로닉스)에서 들여온 모듈 상태에서 진행하는 것임을 밝힌다. 하드웨어 구성에서 할 일은 먼저 vxWorks Image를 EPICS에 맞도록 생성 및 빌드를 하고 MVME6100을 PC와 네트워크 연결을 시킨다. MVME6100이 vxWorks Image를 로드한 것을 확인한 다음에는 BSP를 수정하여 MVME6100의 플래시메모리에 상주하고 있는 bootrom 이미지를 교체한다.

3.1 VxWorks Image 생성 및 빌드하기

하드웨어를 구성하기 전에 VxWorks Image 부터 생성하도록 한다. VxWork Image를 생성할때 EPICS 모듈을 동작시키는 것을 목표로 하고 있으므로 EPICS에서 요구하는 환경설정 및 라이브러리 패키지들이 들어있어야 한다. 그에 대한 자세한 내용은 EPICS 홈페이지의 EPICS vxWorks6.x 부분을 보고 이미지를 생성하도록한다. 기본적으로 `PROFILE_DEVELOPMENT` 로 이미지를 구성하는 것으로 되어있다. 위 링크에 나온 페키지들은 optional 까지 전부 포함시킨다. 한가지 팁이 있다면 "`PROFILE_DEVELOPMENT` 구성에서 기본적으로 들어있으나 EPICS에서 요구하지 않으니 안전을 위해 제거하라"는 패키지들이 있다. 이 패키지들은 제거하지 않도록 한다. 또한 맨 밑에 6.6 인 경우 하는 패치가 있는데 우리는 6.9 이므로 하지 않도록 한다. 나중에 MVME6100이 본 PC에 접속하여 생성된 이미지를 찾을 수 있도록 이미지의 절대 경로를 기록해두던가 홈디렉토리에 심볼릭 링크를 걸던가 하라. 본 매뉴얼에서는 `$HOME/sysboot` 아래에 심볼릭 링크를 걸어두었다. [2]

[1] 그러나 컴퓨터가 변경이 되면 에이스트로닉스에 연락을 취하여 노드를 변경할 수 있으므로 노드를 변경한 수만큼 라이센스가 늘어나는 셈이 된다 .현 시점(14년 5월 11일)에 VxWorks 는 총 3대에 설치되어 있는데 ctrlpc0, ctrlpc3에 각각 설치되어 있고 지금 Cosylab 용역에 나가있는 델 렙탑에 설치되어 있다. 라이센스 변경에 대해서 문의는 에이스트로닉스(전화:)에 해당 양식을 작성하여 요청하면 쉽게 변경해준다. 너무 자주 변경하지말고 각 피씨에 대한 라이센스를 잘 받아 뒀다가 여차할때 쓰면 된다. 양식 작성법은 부록에서 따로 다룬다.

[2] 패키지 선택시 Kernel Symbol을 이미지에 포함 시킬 것인지 따로 사용할 것인지 선택하는 항목이 있는데 따로 사용한다고 선택한 경우에는 이미지 빌드시 vxWorks.sym 이란 파일도 같이 빌드된다. 이 경우 MVME6100가 부팅할때 이 파일도 같이 찾으므로 vxWorks.sym을 해당 디렉토리에 심볼릭 링크로 걸어두도록 한다.

3.2 PC와 MVME6100간의 연결

하드웨어 구성은 단순하다. VME Crate에 EVG, EVR, MVME6100을 설치하고 MVME6100에 LAN선을 연결한다. MVME6100에 LAN 포트가 3개가 있는데 맨 오른쪽부터 debug, LAN 0, LAN 1 이다. debug포트에는 크로스케이블을 연결한다. 두개의 LAN 포트중 LAN 0 만 활성화가 되어 있는데 나머지 것도 활성화 시키려면 Workbench에서 bootrom을 제작하여 config.h 에 있는 변수중 하나를 `#define` 으로 만들어서 활성화 시킨 후에 플래시메모리에 uploading 해야 한다. bootrom을 만들고 플래시메모리에 저장하는 방법은 이후에 나온다. 정리하면 MVME6100은 PC와 두 채널로 접속한다.

Console PC ↔ ctrlport1 ↔ MVME6100 debug port

Host PC ↔ Switch HUB ↔ MVME6100 LAN port

Console 연결은 MVME6100을 세팅한 후 EVG와 EVR을 세팅하는 터미널 역할을 한다. Host 연결은 MVME6100이 네트워크를 통해 PC에 접근하여 부팅 이미지를 다운로드 받는다. (Host 연결방식)
ctrlport1에 접속은 telnet으로 한다. ctrlport1을 세팅하는 방법은 부록에서 다룬다. 본인의 PC에서 ctrlport1을 통해 MVME6100에 접근하려면 다음과 같이 입력한다.

```
ctrluser@ctrlport0:~$ telnet ctrlport1 2001
Trying 10.1.5.161...
Connected to ctrlport1.risp.net.
Escape character is '^]'.
```

ctrlport1은 접속대상이고 2001은 ctrlport가 MVME6100과 접속 중인 포트를 의미한다. 터미널에서 별다른 반응이 없을 것이다. VME crate의 전원을 넣어야 메세지가 뜰 것이다. 현재 상태에서 빠져나가려면 `^]` 을 입력하면 `telnet>` 이라고 뜬다. 이 상태에서는 telnet 명령어들이 듣게 되니 원하는 telnet 명령을 치면 된다. crate에 전원을 넣고 다음 설명으로 넘어가자.

3.3 MVME6100 설정

crate에 전원을 넣으면 다음과 같은 메시지가 terminal에 나타난다.

```
VxWorks System Boot
```

```
Copyright 1984-2002  Wind River Systems, Inc.



CPU: Motorola MVME6100-0163 - MPC 7457
Version: VxWorks5.5.1
BSP version: 1.2/3
Creation date: Oct 29 2008, 17:13:07


Press any key to stop auto-boot...

[VxWorks Boot]:
```

위의 내용에서 알수 있는 것이 Motorola 사의 MPC 7475 cpu가 들어 있으며 MVME6100의 플래쉬메모리에는 VxWorks 5.5.1 bootrom이 들어있다. BSP 버전이 1.2/3 이고 부트롬이 만들어진 날짜도 알수 있다.
우리가 해야 할 것은 위의 bootrom을 새로 만들고 MVME6100에 심어 넣는 일이다. 그러기 전에 값이 어떻게 세팅이 되어 있는지 알고 넘어가자. 현 상태에서 Command 는 MVME6100 메뉴얼에 보면 자세하게 나와 있다. p를 입력하면 현재 세팅 값들이 나온다.

```
[VxWorks Boot]: p

boot device          : geisc
unit number          : 0
processor number     : 0
host name            : host
file name            : vxWorks
inet on ethernet (e) : 192.168.100.180
host inet (h)        : 192.168.100.7
user (u)             : mv6100
ftp password (pw)    : 1234
flags (f)            : 0x0
```

설명을 하면 boot device는 geisc 이다. 이건 LAN 0 을 의미한다. 원래는 geisc0인데 0이 빠져도 무관한 모양이다. 현재 모듈에서 사용 가능한 디바이스들을 알고 싶다면 `devs` 명령어를 입력하면 된다.
이제 이 세팅 값을 변경하도록 하자. c를 누르고 변경한다.

```
[VxWorks Boot]: c

'.' = clear field;  '-' = go to previous field;  ^D = quit

boot device          : geisc0     [enter]
processor number     : 0          [enter]
host name            : host       ctrlpc0
file name            : vxWorks /home/ctrluser/sysboot/mvtest
inet on ethernet (e) : 192.168.100.180 10.1.5.107:ffffff00
inet on backplane (b):  [enter]
host inet (h)        : 192.168.100.7  10.1.5.40
gateway inet (g)     :  10.1.5.254
user (u)             : mv6100  ctrluser
ftp password (pw) (blank = use rsh): 1234 .
flags (f)            : 0x0   [enter]
target name (tn)     :  ctrlvme7
startup script (s)   :  sysboot/elma1
other (o)            :
```

친절하게도 .을 입력하면 해당 필드는 삭제되고 -를 누르면 이전 필드로 가며 ^D 을 누르면 나간다고 표시되어있다. 위 빨간색으로 된 부분이 입력해야 할 값들이고 특히 [enter] 라고 된 부분은 엔터를 쳐서 넘어가라는 뜻이다.

host name은 ethernet을 통해서 부팅 이미지를 불러오는 pc를 의미한다. file name은 pc안에 있는 부팅 이미지의 경로. 절대 경로를 쓴다. inet on ethernet은 현재 MVME6100의 ip 주소 및 subnet mask (필요하면기입) inet on backplan은 모르겠다. host inet 은 pc의 ip주소이고 gateway inet 은 gateway 주소이다. user는 pc에서 유저 이름이며 ftp password 는 유저의 암호를 입력하라는 것인데 우리는 rsh를 사용하므로 입력하지 않는다. 여기서 주의할 점은 pc에서 rsh 접속하도록 1) 방화벽을 해제하고 2) rsh 서버가 돌아가야 하며 3) rsh에서 인증할 수 있도록 .rhosts 파일에 해당 모듈의 ip주소를 입력 해둬야 한다는 것이다. flags 는 vxWorks 교육에서 배운대로 아니면 Kernel module programming 책에 각 모드들에 대한 설명이 나와있다. target name은 본 모듈의 host name을 지정하는 것이다. startup script는 본 모듈의 시작 스크립트를 지정한다. 지금 설정은 pc의 ctrluser 계정의 `$HOME/sysboot/elma1` 파일을 사용하도록 지정한다. 따라서 pc의 `$HOME/sysboot/elma1` 이란 파일을 만들어 둔다. 그러나 없다고해도 무방하다.

이제 @을 입력해보자. 그러면 모듈을 vxWorks 이미지 파일을 읽어들인다.

```
[VxWorks Boot]: @
```

```
boot device          : geisc
unit number          : 0
processor number     : 0
host name            : ctrlpc0
file name            : /home/ctrluser/sysboot/mvtest
inet on ethernet (e) : 10.1.5.107:ffffff00
host inet (h)        : 10.1.5.40
gateway inet (g)     : 10.1.5.254
user (u)             : ctrluser
flags (f)            : 0x0
target name (tn)     : ctrlvme7
startup script (s)   : sysboot/elma1

Attached TCP/IP interface to geisc0.
Attaching network interface lo0... done.
Loading... 2392272
Starting at 0x100000...

Loading symbol table from ctrlpc0:/home/ctrluser/sysboot/mvtest.sym ...done


                VxWorks

Copyright 1984-2014  Wind River Systems, Inc.

             CPU: Motorola MVME6100-0163 - MPC 7457
    Runtime Name: VxWorks
 Runtime Version: 6.9
     BSP version: 6.9/0
         Created: May  3 2014, 16:58:00
ED&R Policy Mode: Deployed
   WDB Comm Type: WDB_COMM_END
             WDB: Ready.

Executing startup script 'sysboot/elma1'...

Done executing startup script 'sysboot/elma1'.

->
```

여기서 이상한 점을 눈치를 챘을지도 모르겠는데 BSP 버전이 6.9/0라는 점이다. 처음 crate

를 켰을때 BSP버전은 Version: VxWorks5.5.1, BSP version: 1.2/3 이었는데 말이다. 이는 VxWorks의 부팅 단계를 이해를 해야 알 수 있는 부분이다. VxWorks는 두가지 부팅 단계를 거친다: 시스템 부팅과 OS 부팅이라는 것인데, 먼저 시스템 부팅은 BSP 레벨에서 보드에 붙은 디바이스들에 대한 초기화 과정을 거치는 작업을 하는 것이다. 이것이 끝나면 OS에 필요한 라이브러리들과 명령어들을 메모리에 상주시키고 서비스들을 동작시키는 작업을 하는데 이 부분이 OS 부팅이다. VxWorks 가이드에서는 앞에 시스템 부팅부터 시작하는 것을 "Cold Boot" 라고 부르고 OS 부팅을 하는 것을 "Warm Boot"라고 부른다. 한마디로 요약하면, Cold Boot는 BSP에서 진행하고 Warm Boot는 VxWorks 이미지에서 진행하는 것이다. 만약 crate를 통째로 껐다가 켰다면 맨처음 BSP 1.2/3으로 디바이스를 초기화 한 후에 방금 만든 이미지로 Warm Boot를 진행한다. 비록 VxWorks 이미지를 BSP 기반으로 해서 만들었다고 해도 실제로 bootrom 이미지를 플래시메모리에 다시 쓰지 않는 한 VxWorks 이미지에서 변경한 BSP 설정은 보드에서 적용되지 않는다. 또한, 입력을 통해서 진행하는 재부팅은 항상 Warm Boot 이며 앞서 눌렀던 `@`를 통하거나 reboot 또는 `ctrl + x` 를 통한 재부팅은 모두 Warm Boot 이다. 그렇기 때문에 재부팅 한다고 하여 메모리에 상주했던 디바이스에 관한 기본 정보들은 변경되지 않으므로 주의해야 한다.

이제 다음 섹션에서 BSP를 수정하고 bootrom 이미지를 만들어서 플래시메모리에 넣는 작업을 하자.

3.4 BSP 설정 및 bootrom.bin 빌드하기

본 매뉴얼에서는 BSP를 수정하고 bootrom 이미지를 빌드하여 플래시메모리를 다시 쓰는 작업을 해야 한다. 그 이유는 지금 사용 중인 BSP는 CR/CSR (Configuration ROM/Control & Status Registers) 이라는 VME64/VME64-X Standard에서 제공하는 기능을 제공하지 않기 때문이다. 그런데 문제는 현재 VME-EVR/EVG 모듈은 CR/CSR 을 반드시 사용해야만 제어할 수 있다. 따라서 CR/CSR 기능을 사용하도록 BSP를 변경하여 플래시메모리에 넣어줘야한다. BSP 변경하는 방법은 Eric Björklund 가 작성한 "EPICS Support for VME CR/CSR Addressing" 슬라이드를 읽어보기 바란다.

BSP 수정을 위해서는 mv6100 BSP 패키지에서 mv6100A.h 와 sysTempeMap.c 파일을 수정하면 된다. 먼저 MVME6100A.h 의 505번째 줄부터 수정을 한다. 아래의 붉은 부분을 추가하면 된다.

```
#define IS_VME_ADDR_MOD(a) ((a == VME_AM_EXT_SUP_PGM)  || \
                            (a == VME_AM_EXT_SUP_DATA) || \
                            (a == VME_AM_EXT_USR_PGM)  || \
                            (a == VME_AM_EXT_USR_DATA) || \
                            (a == VME_AM_STD_SUP_PGM)  || \
                            (a == VME_AM_STD_SUP_DATA) || \
                            (a == VME_AM_STD_USR_PGM)  || \
                            (a == VME_AM_STD_USR_DATA) || \
                            (a == VME_AM_SUP_SHORT_IO) || \
```

```
                            (a == VME_AM_USR_SHORT_IO) || \
                            (a == VME_AM_CSR))

/*
 * Note that VME_CRG_SLV_SIZE is the size of my CRG register group.
 * This group will be mapped onto the VME bus.  VME_CRG_MSTR_SIZE is
 * the size of 16 consecutive CRG groups.  This size will be used to
 * map an outbound window so we can have access to up to 16 different
 * boards's CRG groups (and hence their mailboxes) - allowing VxWorks
 * shared memory to work with mailbox interrupts.
 */

#define VME_CRG_SLV_SIZE        (0x1000)
#define VME_CRG_MSTR_SIZE       (16 * VME_CRG_SLV_SIZE)
#define VME_CRG_MSTR_LOCAL      (VME_A32_MSTR_LOCAL + \
                 VME_A32_MSTR_SIZE - \
                 VME_CRG_MSTR_SIZE)
#define VME_CRG_MSTR_BUS        (0xfb000000)
#define VME_MBOX0_OFFSET (TEMPE_GCSR_MBOX0 + 3)

#define VME_CRCSR_MSTR_SIZE (0x01000000)
#define VME_CRCSR_MSTR_LOCAL (VME_CRG_MSTR_LOCAL - VME_CRCSR_MSTR_SIZE)
#define VME_CRCSR_MSTR_BUS (0x00000000)
#define VME_CRCSR_MSTR_END (VME_CRCSR_MSTR_LOCAL + VME_CRCSR_MSTR_SIZE)


/*
 * Allocation of VME space to outbound windows, first set up the out0
 * window to point to the CRG space to allow access to (among other
 * things) other board's mailboxes - necessary for VxWorks shared memory.
 */

#define VME_OUT0_START (VME_CRG_MSTR_LOCAL)
#define VME_OUT0_SIZE (VME_CRG_MSTR_SIZE)
#define VME_OUT0_BUS            (VME_CRG_MSTR_BUS)

/* Now out1 will point to some node zero system DRAM */

#define VME_DRAM_ACCESS_SIZE (0x00100000)
#define VME_OUT1_START (VME_A32_MSTR_LOCAL)
#define VME_OUT1_SIZE (VME_DRAM_ACCESS_SIZE)
#define VME_OUT1_BUS            (VME_A32_MSTR_BUS)
```

```
/* Finish up with A24 space (out2) and A16 space (out3) */

#define VME_OUT2_START          (VME_A24_MSTR_LOCAL)
#define VME_OUT2_SIZE           (VME_A24_MSTR_SIZE)
#define VME_OUT2_BUS            (VME_A24_MSTR_BUS)

#define VME_OUT3_START          (VME_A16_MSTR_LOCAL)
#define VME_OUT3_SIZE           (VME_A16_MSTR_SIZE)
#define VME_OUT3_BUS            (VME_A16_MSTR_BUS)

#define VME_OUT4_START          (VME_CRCSR_MSTR_LOCAL)
#define VME_OUT4_SIZE           (VME_CRCSR_MSTR_SIZE)
#define VME_OUT4_BUS            (VME_CRCSR_MSTR_BUS)


#define VME_OUT0_CFG_PARAMS \
             TRUE,                    /* Window enabled */ \
             0, VME_OUT0_START,       /* Local start addrs (upper = 0) */ \
             0, VME_OUT0_SIZE,        /* Size (upper = 0) */ \
             0, VME_OUT0_BUS,         /* VME bus addr (upper = 0) */ \
             0,                       /* 2eSST broadcast select */ \
             0,                       /* Unused */ \
             FALSE,                   /* Read prefetch enable state */ \
             VME_RD_PREFETCH_2_CACHE_LINES, \
             VME_SST160,              /* 2esst xfer rate */ \
             VME_SCT_OUT,             /* transfer mode */ \
             VME_D32,                 /* VME data bus width */ \
             FALSE,                   /* nonsupervisor access */ \
             FALSE,                   /* Not pgm but instead data access */ \
             VME_MODE_A32             /* transfer mode */

                              ...
                              ...

#define VME_OUT3_CFG_PARAMS \
             TRUE,                    /* Window enabled */ \
             0, VME_OUT3_START,       /* Local start addrs (upper = 0) */ \
             0, VME_OUT3_SIZE,        /* Size (upper = 0) */ \
             0, VME_OUT3_BUS,         /* VME bus addr (upper = 0) */ \
             0,                       /* 2eSST broadcast select */ \
             0,                       /* Unused */ \
             TRUE,                    /* Read prefetch enable state */ \
```

```
            VME_RD_PREFETCH_2_CACHE_LINES, \
            VME_SST160,             /* 2esst xfer rate */ \
            VME_MBLT_OUT,           /* transfer mode */ \
            VME_D32,                /* VME data bus width */ \
            FALSE,                  /* nonsupervisor access */ \
            FALSE,                  /* Not pgm but instead data access */ \
            VME_MODE_A16            /* transfer mode */

#define VME_OUT4_CFG_PARAMS \
            TRUE,                   /* Window enabled */ \
            0, VME_OUT4_START,      /* Local start addrs (upper = 0) */ \
            0, VME_OUT4_SIZE,       /* Size (upper = 0) */ \
            0, VME_OUT4_BUS,        /* VME bus addr (upper = 0) */ \
            0,                      /* 2eSST broadcast select */ \
            0,                      /* Unused */ \
            TRUE,                   /* Read prefetch enable state */ \
            VME_RD_PREFETCH_2_CACHE_LINES, \
            VME_SST160,             /* 2esst xfer rate */ \
            VME_MBLT_OUT,           /* transfer mode */ \
            VME_D32,                /* VME data bus width */ \
            TRUE,                   /* nonsupervisor access */ \
            FALSE,                  /* Not pgm but instead data access */ \
            VME_MODE_CRCSR            /* transfer mode */
```

마지막 부분에서 \ 부분이 빨간색이 아닌 이유는 Latex특성상 어쩔 수 없는 일이기 때문이었으며 본래는 \까지 포함해야 한다.
다음은 sysTempeMap.c 파일의 621째 줄로 가서 다음 루틴을 찾는다.

```
STATUS sysVmeToPciAdrs
    (
    int     vmeAdrsSpace, /* VME bus adress space where busAdrs resides */
    char *  vmeBusAdrs,   /* VME bus adress to convert */
    char ** pPciAdrs      /* where to return converted local (PCI) adress */
    )
```

이 루틴 아래에 다음 부분을 찾아서 빨간색 부분을 추가한다.

```
/* It is enabled */

          switch (vmeAdrsSpace)
          {
           case VME_AM_CSR:
           /* See if the window is CR/CSR enabled */
           if((vmeOutWin[i].att & TEMPE_OTATx_AMODEx_MASK) ==
            (TEMPE_OTATx_AMODE_VAL_CSR))
           {
           vmeSpaceMask = 0x00ffffff;
           vmeAdrToConvert = (UINT32)vmeBusAdrs & vmeSpaceMask;
           break;
           }
          else
           continue;

              case VME_AM_EXT_SUP_PGM:
              case VME_AM_EXT_SUP_DATA:
              case VME_AM_EXT_USR_PGM:
              case VME_AM_EXT_USR_DATA:

           /* See if the window is A32 enabled */

           if ( ((vmeOutWin[i].att & TEMPE_OTATx_AMODEx_MASK) ==
                (TEMPE_OTATx_AMODE_VAL_A32)) ||
                ((vmeOutWin[i].att & TEMPE_OTATx_AMODEx_MASK) ==
                (TEMPE_OTATx_AMODE_VAL_CSR)) )
           {
                vmeSpaceMask = 0xffffffff;
                vmeAdrToConvert = (UINT32)vmeBusAdrs;
                break;
           }
```

그리고 새로 빌드를 한 후에 종료하면 된다.

3.5 Bootrom 업데이트 하기

bootrom을 업데이트 하려면 두가지가 준비되어야 한다. 먼저 MVME6100 점퍼를 바꿔야 하고, bootrom을 다운받을 PC에 tftp를 설정하고 수퍼데몬을 이용하여 tftpd를 활성화 시켜야 한다. 본 메뉴얼에서는 tftpd 활성화 시키는 방법에 대해서 다루지 않으니 알아서 활성화 시

키도록한다. 여기서는 tftpd의 루트 디렉토리를 /srv/tftp/ 로 설정해놓고 설명을 한다. 또한, vxWorks의 부트롬 이미지를 bootrom.bin으로 링크 시켜놓았다. ctrlpc0 에서는 vxWorks를 /opt/vxWorks6.9 아래에 설치했다. 그래서 부트이미지는 /opt/vxWorks6.9/vxworks-6.9/target/config/mv6100/bootrom.bin 여기에 위치한다.
Bootrom을 플래시메모리에 업데이트하려면 MVME6100의 점퍼 설정을 바꿔야 한다. S4 점퍼를 찾는다. 초기 값은 off off on off 이다. 여기서 두번째 점퍼를 On 시킨다. 그러면 off, on, on, off 이렇게 될 것이다. (공교롭게도 여기서는 아래로 내려야 On 이다) 모듈에 전원을 넣으면 다음과 같은 화면이 뜬다.

```
Copyright(C)2008-2009,Emerson Network Power-Embedded Computing,Inc.
 All Rights Reserved
 Copyright Motorola Inc. 1999-2007, All Rights Reserved
 MOTLoad RTOS Version 2.0,  PAL Version 2.3 RM01
 Fri Jan 23 14:47:54 MST 2009

 MPU-Type              =MPC74x7
 MPU-Int Clock Speed   =1266MHz
 MPU-Ext Clock Speed   =133MHz
 MPU-Int Cache(L2) Enabled, 512KB, L2CR =C0000000
 MPU-Ext Cache(L3) Enabled, 2MB, 211MHz, L3CR =DC026000

 PCI bus instance 0    =64 bit, 133 Mhz, PCI-X
 PCI bus instance 1    =64 bit, PCI

 Reset/Boot Vector     =Flash1

 Local Memory Found    =20000000 (&536870912)
 User Download Buffer =006B7000:008B6FFF


 MVME6100>
```

MVME6100안에는 플래시메모리가 2개 들어있다. 하나는 사용자 Bootrom을 위해서 만들어 둔 곳이고 다른 하나는 Emerson사가 만든 Bootrom 이미지를 저장하기 위한 장소이다. 점퍼를 자세히 보면 on으로 바꾼 이유를 알 수 있는데 플래시메모리B를 이용하여 부팅하기 위해서 바꾼 것이다. 원래대로 돌려 놓으면 플래시메모리A로 부팅을 하게 된다. 물론 B에는 Emerson이 제공하는 Bootrom이고 A에는 사용자의 Bootrom이 들어간다. [3] Emerson이 제공하는 Bootrom은 MOTLoad라는 이미지인데 쉘 명령어는 MVME6100 사용설명서에 들어있다. 이제 본래 이야기로 돌아가자.
ctrlpc0 에서 /srv/tftp/bootrom.bin 을 다운로드 한다.

[3] 점퍼를 설정을 바꾸면 B 역시 사용자가 쓰고 지울 수 있다. 그러나 누가 그런 짓을 할까?

```
MVME6100> tftpGet -c10.1.5.107 -fbootrom.bin -s10.1.5.40
Network Loading from: /dev/enet0
Loading File: bootrom.bin
Load Address: 006B7000
Download Buffer Size = 00200000

Client IP Address      = 10.1.5.107
Server IP Address      = 10.1.5.40
Gateway IP Address     = 10.1.5.253
Subnet IP Address Mask = 255.255.255.0

Network File Load in Progress...

Bytes Received =&367856, Bytes Loaded =&367856
Bytes/Second   =&367856, Elapsed Time =1 Second(s)

MVME6100> flashProgram -o03f00100 -nfff00 -v
Source Starting/Ending Addresses      =006B7000/007B6EFF
Destination Starting/Ending Addresses =F7F00100/F7FFFFFF
Number of Effective Bytes             =000FFF00 (&1048320)

Program Flash Memory (Y/N)? y
Virtual-Device-Number    =00
Manufacturer-Identifier  =89
Device-Identifier        =1D
Virtual-Device-Number    =01
Manufacturer-Identifier  =89
Device-Identifier        =1D
Address-Mask             =FC000000
```

여기서 입력한 명령어는 2가지 인데 첫번째는

```
MVME6100> tftpGet -c10.1.5.107 -fbootrom.bin -s10.1.5.40
```

이 명령어는 tftp를 통해 bootrom.bin 파일을 플래시메모리A에 다운로드 하는 것이고

```
MVME6100> flashProgram -o03f00100 -nfff00 -v
```

이 명령어는 다운로드된 bootrom을 플래쉬메모리A에 영구적으로(?) 쓰는 작업을 한다.
완료되면 crate를 끈 다음 점퍼를 원래대로 돌려놓고 다시 설치하여 부팅을 시켜본다. 여기까지 완성이 되었다면 다음에는 mrfIoc2를 설치를 해본다.

4 소프트웨어 구성

이 장에서는 VME-EVR을 제어하기 위한 EPICS 소프트웨어인 mrfIoc2 를 빌드하고 VME 인식을 시키고 OPI로 제어하는 방법에 대해서 설명한다.

4.1 mrfIoc2 설치

mrfIoc2를 빌드하기 위해 요구되는 EPICS모듈들과 extensions이 있다.

- devLib2 : EPICS 모듈이다. 각 OS 별로 PCI와 VME에 쉽게 접근 할 수 있도록 저수준 라이브러리를 제공한다.
- IocStat : EPICS 모듈이다.
- msi : EPICS Extensions이다. 코드 내부의 있는 스크립트들을 Parsing 하는 용도로 쓰인다.

mrfIoc2를 VxWorks에서 빌드할때 VxWokrs의 코드를 수정해야 할 경우가 생긴다. (아직 정확한 이유와 조건을 특정하지 못했다.)

4.2 IOC 올리기

이제 mrfIoc2를 구동시키자. $EPICS/modules/mrfioc2/iocBoot/iocevrmrm/ 로 이동한다. 여기서 st.cmd를 실행 시켜서 IOC를 구동하는 것은 알고 있으리라 생각한다. 기본으로 정의된 내용으로는 IOC가 동작하지 않는다. 즉, 수정을 해야 한다. 구동 스크립트에 대해서 설명한다.

```

```

4.3 VME 인식 시키기

VME를 인식 시키기 위해서는 VME 에 대한 기초지식이 필요하다.
EPICS 모듈을 로딩을 하면 에러가 난다.

```
-> sysTempeWinShow

Outbound window 0: ENABLED,  Size = 00000000_00010000

raw: 00000000 8fff0000 00000000 8fff0000 00000000 6b010000 00000000 80040042
PCI Base:  00000000_8fff0000    VME Base:  00000000_fb000000
PCI Limit: 00000000_8fffffff    VME Limit: 00000000_fb00ffff
     2eSST Broadcast Select: 0x00000
```

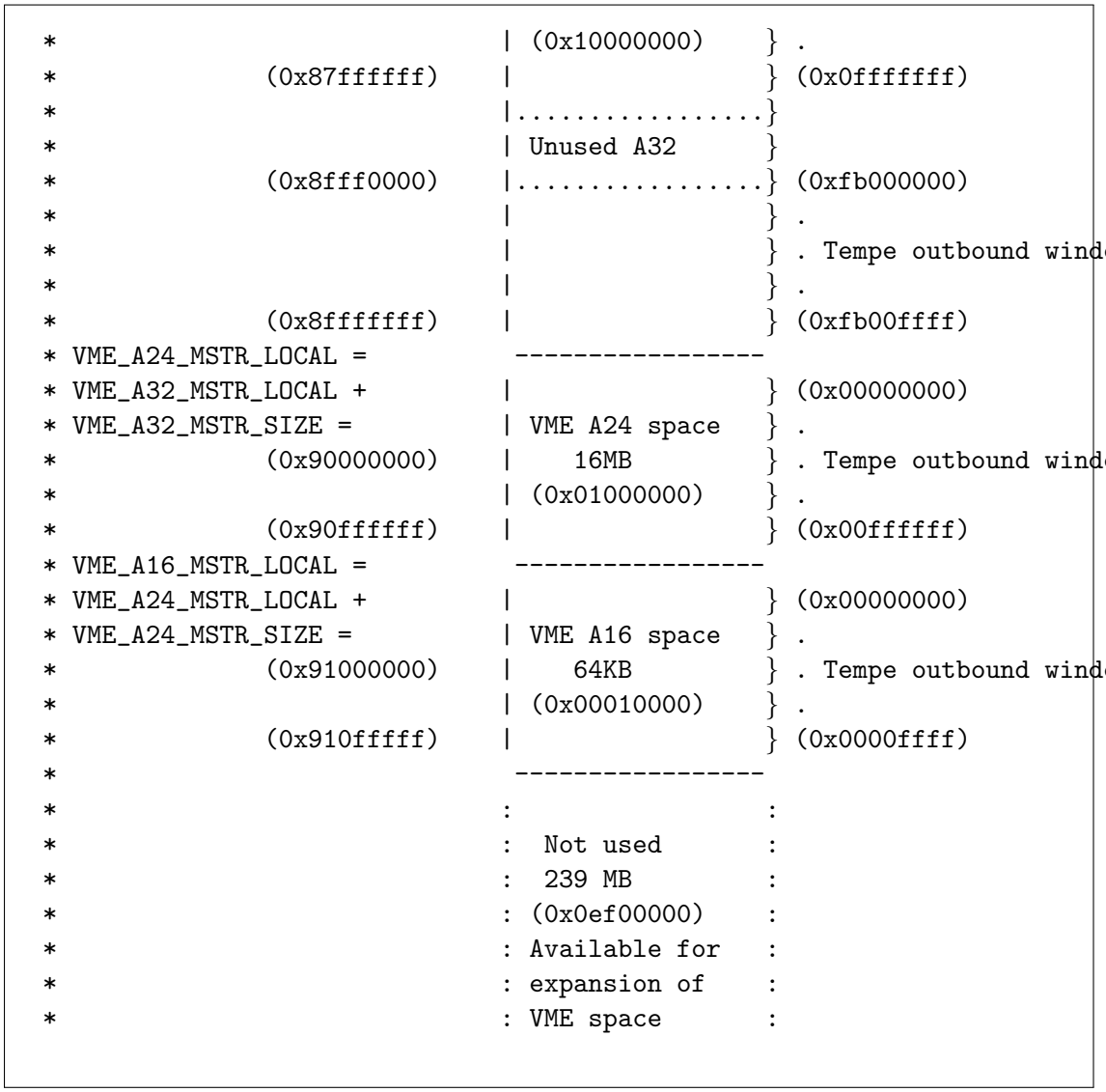
```
     Attributes:
        Memory Prefetch      - disabled
        Prefetch size        - 2 cache lines
        2eSST Mode           - 160 MB/s
        Transfer Mode        - SCT (Single Cycle Transfer)
        VME Data Bus Width   - 32 bit
        VME AM code          - nonSupervisor, nonProgram
        VME Address mode     - A32


Outbound window 1: ENABLED,  Size = 00000000_00100000

raw: 00000000 80000000 00000000 800f0000 ffffffff 88000000 00000000 80000262
PCI Base:  00000000_80000000    VME Base:  00000000_08000000
PCI Limit: 00000000_800fffff    VME Limit: 00000000_080fffff
     2eSST Broadcast Select: 0x00000
     Attributes:
        Memory Prefetch      - enabled
        Prefetch size        - 2 cache lines
        2eSST Mode           - 160 MB/s
        Transfer Mode        - MBLT (Multiplexed Block Transfer)
        VME Data Bus Width   - 32 bit
        VME AM code          - Supervisor, nonProgram
        VME Address mode     - A32


Outbound window 2: ENABLED,  Size = 00000000_01000000

raw: 00000000 90000000 00000000 90ff0000 ffffffff 70000000 00000000 80000241
PCI Base:  00000000_90000000    VME Base:  00000000_00000000
PCI Limit: 00000000_90ffffff    VME Limit: 00000000_00ffffff
     2eSST Broadcast Select: 0x00000
     Attributes:
        Memory Prefetch      - enabled
        Prefetch size        - 2 cache lines
        2eSST Mode           - 160 MB/s
        Transfer Mode        - MBLT (Multiplexed Block Transfer)
        VME Data Bus Width   - 32 bit
        VME AM code          - nonSupervisor, nonProgram
        VME Address mode     - A24
```

```
Outbound window 3: ENABLED,  Size = 00000000_00010000

raw: 00000000 91000000 00000000 91000000 ffffffff 6f000000 00000000 80000240
PCI Base:  00000000_91000000    VME Base:  00000000_00000000
PCI Limit: 00000000_9100ffff    VME Limit: 00000000_0000ffff
     2eSST Broadcast Select: 0x00000
     Attributes:
        Memory Prefetch      - enabled
        Prefetch size        - 2 cache lines
        2eSST Mode           - 160 MB/s
        Transfer Mode        - MBLT (Multiplexed Block Transfer)
        VME Data Bus Width   - 32 bit
        VME AM code          - nonSupervisor, nonProgram
        VME Address mode     - A16



Outbound window 4: NOT ENABLED


Outbound window 5: NOT ENABLED


Outbound window 6: NOT ENABLED


Outbound window 7: NOT ENABLED

Inbound window 0: ENABLED,  Size = 00000000_00040000

raw: 00000000 08000000 00000000 0803fff0 ffffffff f8000000 80000faf
VME Base:  00000000_08000000    PCI Base:  00000000_00000000
VME Limit: 00000000_0803ffff    PCI Limit: 00000000_0003ffff
     Attributes:
        Read-ahead threshold - when FIFO competely empty
        Virtual FIFO size    - 64 bytes
        2eSST Mode           - 160 MB/s
        Cycle response       - 2eSSTB (Two Edge Source Synchronous Broadcast)
                               2eSST (Two Edge Source Synchronous nonBroadcast)
                               2eVME (Two Edge VMEbus)
                               MBLT (Multiplexed Block Transfer)
                               BLT (Block Transfer)
```

```
        Address space        - A32
        VME AM response      - Supervisor, nonSupervisor, Program, Data


Inbound window 1: NOT ENABLED


Inbound window 2: NOT ENABLED


Inbound window 3: NOT ENABLED


Inbound window 4: NOT ENABLED


Inbound window 5: NOT ENABLED


Inbound window 6: NOT ENABLED


Inbound window 7: NOT ENABLED
value = 32 = 0x20 = ' '
->
```

MVME6100의 메모리 공간에 대해서 설명을 해야 한다. 다음 테이블은 WorkBench의 도움말에서 MVME6100에 대해서 검색을 하면 나오는 문서의 일부다. 이것은 config.h 에서 볼 수 있다. 필자는 VME관련된 부분만 가져왔다.

```
 * CPU Space                                PCI (|) or VME (}) Space
 * (default values)
 *
 *                               :                 :
 *                               :                 :
 * VME_A32_MSTR_LOCAL =           ------------------  VME_A32_MSTR_BUS
 *              (0x80000000)     |                 } (0x08000000)
 *                               | VME A32 space   } .
 *                               |    256MB        } . Tempe outbound window 1
```



CPU space 라는 것은 CPU 에서 보는 메모리 주소를 의미한다. 이 말은 d(주소) 해서 나오는 값을 의미한다. PCI or VME Space 라는 것은 PCI/VME 콘트롤 칩에서 바라보는 메모리 주소를 의미한다. 실제로 VME를 제어 하기 위해서 우리가 입력하는 주소 값은 CPU space 에서의 값을 말한다. 또 하나 알아둬야하는 점은 Outbound/Inboudn Window 라는 용어다. Outbound Window 는 CPU에서 타겟 제어칩의 메모리주소공간에 대한 mapping을 의미한다. 이게 무슨 말이냐면, 위의 이름에 Tempe outbound window 0 이라고 적힌 부분이 있을 것이다. MVME6100 에는 VME/PCI용 콘트롤러가 Tempe(TSi148) Chip을 사용한다. Tempe를 DMA로 제어하기 위해서는 CPU주소에서 Tempe로 주소를 매핑(mapping) 해야 한다. 따라서 CPU에서 출발하므로 Outbound 가 되고 매핑하는 영역을 Window라고 부른다. 반대로 CPU의 주소공간을 다른 콘트롤 칩에서 사용하기 위해 매핑하는 공간을 Inbound Window 라고 한다. SystempeWinShow를 실행하면 Tempe칩에서 정의하는 Outbound, Inbound Window를 모두 보여주는데

```
-> mrmEvrSetupVME("EVRX",3,0x08000000,3,0xc0)
Setting up EVR in VME Slot 3
Found vendor: 00000eb2 board: 455246e6 rev.: 00000000
Error: Address does not correspond to an EVR
value = 0 = 0x0
-> FWVersion 0x00000000
```

```
-> d(0x80000000)
NOTE: memory values are displayed in hexadecimal.
0x80000000:  0000 0000 0000 0000 0000 0000 0000 0000  *................*
0x80000010:  0000 0000 0000 0000 0000 0000 0000 0000  *................*
0x80000020:  0000 0000 0000 0000 0000 0000 0000 0000  *................*
0x80000030:  0000 0000 0000 0000 0000 0000 0000 0000  *................*
0x80000040:  0000 0000 0000 0000 0000 0000 0000 0000  *................*
0x80000050:  0000 0000 0000 0000 0000 0000 0000 0000  *................*
0x80000060:  0000 0000 0000 0000 0000 0000 0000 0000  *................*
0x80000070:  0000 0000 0000 0000 0000 0000 0000 0000  *................*
0x80000080:  0000 0000 0000 0000 0000 0000 0000 0000  *................*
0x80000090:  0000 0000 0000 0000 0000 0000 0000 0000  *................*
0x800000a0:  0000 0000 0000 0000 0000 0000 0000 0000  *................*
0x800000b0:  0000 0000 0000 0000 0000 0000 0000 0000  *................*
0x800000c0:  0000 0000 0000 0000 0000 0000 0000 0000  *................*
0x800000d0:  0000 0000 0000 0000 0000 0000 0000 0000  *................*
0x800000e0:  0000 0000 0000 0002 0250 3020 0250 3050  *.........P0 .P0P*
0x800000f0:  0250 3038 0000 0002 0000 0003 0000 0000  *.P08............*
value = 0 = 0x0
```

위에는 VME의 32비트 주소공간을 살펴본 내용이다. 이 공간에 무엇이 쓰여 있는 것을 알 수 있다. 이는 위에서 언급한 `SM_OFF_  = FALSE` 로 맞춰져 있기 때문이다. 여기서 두가지 선택을 할 수 있다.
(1) 부트롬을 바꾸는 방법 (2) 다른 VME address 를 사용하는 방법
(1)로 한다는 말은 공유 메모리 영역을 사용하지 않도록 BSP 설정을 바꾼 후에 bootrom을 컴파일하고 만들어진 bootrom을 플래쉬메모리에 저장하는 방법이다. 이번에는 (2)번으로 해보자. 다른 주소 공간을 사용하려면 32비트 주소공간에서 빈 곳이 어디인지 알아야 할 것이다. 위에서 SM 영역은 `00000000_08000000` 에서 `00000000_0803ffff` 까지 사용했다. 따라서 우리는 VME space에서 0x08040000 부터 사용하면 될 것이다. 우선 비어 있는지 확인해보자

```
-> d(0x80040000)
NOTE: memory values are displayed in hexadecimal.
0x80040000:  ffff ffff ffff ffff ffff ffff ffff ffff  *................*
0x80040010:  ffff ffff ffff ffff ffff ffff ffff ffff  *................*
0x80040020:  ffff ffff ffff ffff ffff ffff ffff ffff  *................*
0x80040030:  ffff ffff ffff ffff ffff ffff ffff ffff  *................*
0x80040040:  ffff ffff ffff ffff ffff ffff ffff ffff  *................*
0x80040050:  ffff ffff ffff ffff ffff ffff ffff ffff  *................*
0x80040060:  ffff ffff ffff ffff ffff ffff ffff ffff  *................*
0x80040070:  ffff ffff ffff ffff ffff ffff ffff ffff  *................*
0x80040080:  ffff ffff ffff ffff ffff ffff ffff ffff  *................*
0x80040090:  ffff ffff ffff ffff ffff ffff ffff ffff  *................*
0x800400a0:  ffff ffff ffff ffff ffff ffff ffff ffff  *................*
0x800400b0:  ffff ffff ffff ffff ffff ffff ffff ffff  *................*
0x800400c0:  ffff ffff ffff ffff ffff ffff ffff ffff  *................*
0x800400d0:  ffff ffff ffff ffff ffff ffff ffff ffff  *................*
0x800400e0:  ffff ffff ffff ffff ffff ffff ffff ffff  *................*
0x800400f0:  ffff ffff ffff ffff ffff ffff ffff ffff  *................*
value = 0 = 0x0
```

비어 있는 것을 확인했으므로 mrmEvrSetupVME 명령어를 넣어보자.

```
-> mrmEvrSetupVME("EVRX",3,0x08040000,3,0xc0)
Setting up EVR in VME Slot 3
Found vendor: 00000eb2 board: 455246e6 rev.: 00000000
VME64 Out FP:7 FPUNIV:4 RB:16 IFP:2
Using IRQ 3:192
value = 0 = 0x0
-> EVR FIFO task start
FWVersion 0x12000006
Found version 6
```

정상적으로 올라 온 것을 확인할 수 있다. 기왕 여기까지 온거 메모리를 한번더 덤프를 하여 펌웨어버전이 제대로 쓰여 있는지 알아보자

```
-> d(0x80040000)
```

```
NOTE: memory values are displayed in hexadecimal.
0x80040000:  0001 0000 0080 0000 0000 0005 0000 0000  *..............*
0x80040010:  0000 0000 0000 0000 0000 0000 0000 0000  *..............*
0x80040020:  0000 0000 0010 0000 0000 0000 1200 0006  *..............*
0x80040030:  0000 0000 0000 0000 0000 0000 0000 0000  *..............*
0x80040040:  0000 0000 0000 0000 0000 0000 0000 007d  *..............}*
0x80040050:  0000 0200 0000 0000 0000 0000 0000 0000  *..............*
0x80040060:  0000 0000 0000 0000 0000 0000 0000 0000  *..............*
0x80040070:  0000 0000 0000 0000 0000 0000 0000 0000  *..............*
0x80040080:  0c92 8166 0000 0000 0000 0003 0000 0000  *...f..........*
0x80040090:  0000 0000 0000 00ff 0000 0000 0000 0000  *..............*
0x800400a0:  0000 0000 0000 0000 0000 0000 0000 0000  *..............*
0x800400b0:  0000 0000 0000 0000 0000 0000 0000 0000  *..............*
0x800400c0:  0000 0000 0000 0000 0000 0000 0000 0000  *..............*
0x800400d0:  0000 0000 0000 0000 0000 0000 0000 0000  *..............*
0x800400e0:  0000 0000 0000 0000 0000 0000 0000 0000  *..............*
0x800400f0:  0000 0000 0000 0000 0000 0000 0000 0000  *..............*
value = 0 = 0x0
```

펌웨어버전이 1200 0006 임을 쉽게 확인할 수 있다.
이후에 EVR 카드를 계속 추가 하고 싶다면 위의 어드레스 스페이스를 피해야 한다. 당연히 개체 이름과 슬롯 넘버도 바꿔야 한다. 어차피 같은 ISR을 사용해야 하니까 IRQ 레벨이나 IRQ 값은 그대로 둔다. 카드 하나당 어드레스 공간 크기는 0x00010000 즉 64KByte 이다. 따라서 만약에 4번 슬롯에 추가를 한다면 0x08050000 로 넣어야 한다.

```
-> mrmEvrSetupVME("EVRX",4,0x08050000,3,0xc0)
```

References

[1] K. Tshoo, *et. al*,"Experimental systems overview of the Rare Isotope Science Project in Korea",