Micro-Research Finland Oy
Välitalontie 83 C, FI-00660 Helsinki, Finland

CompactPCI Event Generator and Event Receiver Linux Kernel 2.6 Drivers

Contents

Introduction

This document describes the Linux Kernel 2.6 device drivers for the CompactPCI Event Generator and CompactPCI/PMC Event Receiver.

INSTALLATION

Unpack the device drivers sources:

```
$ tar zxf pci_mrfev.tar.gz
```

The kernel headers for the target kernel have to be installed. To build kernel modules run make in the top level directory:

```
$ make
```

To install (as root)

```
# make modules_install
# depmod -a
```

To load and unload the kernel drivers two script are provided. The module_load script loads the modules pci_mrfevg and pci_mrfevr for both the event generator (EVG) and event receiver (EVR).

To load the drivers (as root)

```
# sh module_load
Found 2 Event Generators.
Creating device nodes...
Creating nodes /dev/ega[0-3] for major 253
Creating nodes /dev/egb[0-3] for major 254
Found 2 Event Receivers.
Creating device nodes...
Creating nodes /dev/era[0-3] for major 251
Creating nodes /dev/erb[0-3] for major 252
```

To unload the drivers (as root)

```
# sh module_unload
Unloading modules
Removing device nodes
```

The drivers dynamically allocate one major device number and four minor numbers for each board. The module_load script creates device nodes into the /dev directory and sets the permission for the devices. The group for the devices is set to 'mrf' and the script assumes this group does exist. The device nodes created follow the rules:

/dev/ega[0-3]	First EVG
/dev/egb[0-3]	Seconds EVG
/dev/egc[0-3]	Third EVG etc.

/dev/era[0-3]	First EVR
/dev/erb[0-3]	Seconds EVR
/dev/erc[0-3]	Third EVR etc.

The minor numbers have following functions:
minor 0: read/write access to PCI9030 EEPROM
minor 1: read/write access to XCF08P Platform Flash configuration memory
minor 2: read FPGA status/write FPGA configuration data
minor 3: memory map EVG/EVR registers into user space

The module_load script allow group write access to the minor device 3, all other devices have write access by root only.

PCI9030 EEPROM ACCESS

The serial EEPROM that holds the initialization data for the PCI9030 bridge can be accessed through minor device 0. To dump the EEPROM contents issue command:

```
$ dd if=/dev/ega0 of=eeprom.eep
```

This generates an ascii readable file of the EEPROM contents. One can also dump the contents to the screen:

```
$ cat /dev/ega0
PCI9030 EEPROM Contents

0x20dc Device ID
0x1a3e Vendor ID
0x0290 PCI Status
0x0000 PCI Command
0x1180 Class Code
0x0001 Class Code / Revision
0x20dc Subsystem ID
0x1a3e Subsystem Vendor ID
0x0000 MSB New Capability Pointer
0x0040 LSB New Capability Pointer
0x0000 (Maximum Latency and Minimum Grant are not loadable)
0x0100 Interrupt Pin (Interrupt Line Routing is not loadable)
0x4801 MSW of Power Management Capabilities
0x4801 LSW of Power Management Next Capability Pointer
0x0000 MSW of Power Management Data / PMCSR Brdge Support Extension
0x0000 LSW of Power Management Control / Status
0x0000 MSW of Hot Swap Control / Status
0x4c06 LSW of Hot Swap Next Capability Pointer / Hot Swap Control
0x0000 PCI Vital Product Data Address
0x0003 PCI Vital Product Data Next Capability Pointer
0x0fff MSW of Local Address Space 0 Range
0x0000 LSW of Local Address Space 0 Range
0x0000 MSW of Local Address Space 1 Range
0x0000 LSW of Local Address Space 1 Range
0x0000 MSW of Local Address Space 2 Range
0x0000 LSW of Local Address Space 2 Range
```

```
0x0000 MSW of Local Address Space 3 Range
0x0000 LSW of Local Address Space 3 Range
0x0000 MSW of Expansion ROM Range
0x0000 LSW of Expansion ROM Range
0x0000 MSW of Local Address Space 0 Local Base Address (Remap)
0x0001 LSW of Local Address Space 0 Local Base Address (Remap)
0x0000 MSW of Local Address Space 1 Local Base Address (Remap)
0x0000 LSW of Local Address Space 1 Local Base Address (Remap)
0x0000 MSW of Local Address Space 2 Local Base Address (Remap)
0x0000 LSW of Local Address Space 2 Local Base Address (Remap)
0x0000 MSW of Local Address Space 3 Local Base Address (Remap)
0x0000 LSW of Local Address Space 3 Local Base Address (Remap)
0x0010 MSW of Expansion ROM Local Base Address (Remap)
0x0000 LSW of Expansion ROM Local Base Address (Remap)
0x0180 MSW of Local Address Space 0 Bus Region Descriptor
0x0002 LSW of Local Address Space 0 Bus Region Descriptor
0x0080 MSW of Local Address Space 1 Bus Region Descriptor
0x0000 LSW of Local Address Space 1 Bus Region Descriptor
0x0080 MSW of Local Address Space 2 Bus Region Descriptor
0x0000 LSW of Local Address Space 2 Bus Region Descriptor
0x0080 MSW of Local Address Space 3 Bus Region Descriptor
0x0000 LSW of Local Address Space 3 Bus Region Descriptor
0x0080 MSW of Expansion ROM Bus Region Descriptor
0x0000 LSW of Expansion ROM Bus Region Descriptor
0x0000 MSW of Chip Select 0 Base Address
0x1001 LSW of Chip Select 0 Base Address
0x0000 MSW of Chip Select 1 Base Address
0x0000 LSW of Chip Select 1 Base Address
0x0000 MSW of Chip Select 2 Base Address
0x0000 LSW of Chip Select 2 Base Address
0x0000 MSW of Chip Select 3 Base Address
0x0000 LSW of Chip Select 3 Base Address
0x0030 Serial EEPROM Write-Protected Address Boundary
0x0000 LSW of Interrupt Control/Status
0x0078 MSW of PCI Target Response, Serial EEPROM, and Initialization
Control
0x1100 LSW of PCI Target Response, Serial EEPROM, and Initialization
Control
0x0024 MSW of General Purpose I/O Control
0x9900 LSW of General Purpose I/O Control
0x0000 MSW of Hidden 1 Power Management Data Select
0x0000 LSW of Hidden 1 Power Management Data Select
0x0000 MSW of Hidden 2 Power Management Data Select
0x0000 LSW of Hidden 2 Power Management Data Select
```

To program the EEPROM run (as root)

```
# dd if=eeprom.eep of=/dev/ega0
6+1 records in
6+1 records out
```

NOTE! Invalid data in the EEPROM can cause the board to become unaccessible and require factory service.

XCF08P Platform Flash Access

NOTE! Accessing the flash is very slow. Reading 1Mbyte of configuration data takes about 30 s on a NI PXI-8195, erasing and writing ~400 kbytes of configuration data takes over 3 min.

The Platform Flash device holds the configuration data for the Virtex II Pro FPGA which is loaded when the board is powered up. The contents of the flash device can be read with the following command (this reads in first 1Mbyte of config data):

```
$ dd if=/dev/ega1 of=flash.bit bs=1024 count=1024
1024+0 records in
1024+0 records out
```

To erase the flash and write new configuration data issue the command (as root):

```
# dd if=flash.bit of=/dev/ega1
767+1 records in
767+1 records out
```

After the programming is finished the FPGA is configuration is loaded from the flash.

FPGA Configuration

The FPGA may be configured with direct access overriding the configuration data in the platform flash. To load the FPGA issue the command (as root):

```
# dd if=fpga.bit of=/dev/ega2
767+1 records in
767+1 records out
```

The FPGA status may be displayed following:

```
$ cat /dev/ega2
FPGA status
DONE      1
INIT      1
MODE      000
GHIGH_B   deasserted
GWE       1
DCI_MATCH 1
DCM_LOCK  1
CRC_ERROR 0
```

Memory Mapping EVG/EVR registers into user space

Access to the EVG/EVR registers is provided through the minor device 3 which is given group read/write access by to module_load script.

Below is an excerpt from the API sources for the EVG to show how the user space access to the EVG registers is gained:

```
int EvgOpen(struct MrfEgRegs **pEg, char *device_name)
{
  int fd;

  /* Open Event Generator device for read/write */
  fd = open(device_name, O_RDWR);
#ifdef DEBUG
  DEBUG_PRINTF("EvgOpen: open(\"%s\", O_RDWR) returned %d\n",
               device_name, fd);
#endif
  if (fd != -1)
    {
      /* Memory map Event Generator registers */
      *pEg = (struct MrfEgRegs *) mmap(0, EVG_MEM_WINDOW,
                                       PROT_READ | PROT_WRITE,
                                       MAP_SHARED, fd, 0);
#ifdef DEBUG
  DEBUG_PRINTF("EvgOpen: mmap returned %08x, errno %d\n", (int) *pEg,
               errno);
#endif
      if (*pEg == MAP_FAILED)
        {
          close(fd);
          return -1;
        }
    }

  return fd;
}
```

APPENDIX: module_load script

```
#!/bin/sh

# sh /home/jpietari/mrf/event/sw/linux-multi/module_load
# sh /home/jpietari/mrf/event/sw/linux-multi/module_unload
# cd /home/jpietari/mrf/event/sw/linux-multi

#make modules_install
#/sbin/depmod -a
/sbin/modprobe pci_mrfevg || exit 1
/sbin/modprobe pci_mrfevr || exit 1

majors=$(awk "\$2==\"mrfevg\" {print \$1}" /proc/devices)

echo "Found" $(echo $majors | wc -w) "Event Generators."
echo "Creating device nodes..."

device=1
for major in $majors; do
    dev=$(echo $device | awk '{ printf "%c", 96+ $1}')
    device=$((++device))
```

```
    rm -f /dev/eg$dev[0-3]
    echo "Creating nodes /dev/eg"$dev"[0-3] for major" $major
    mknod '/dev/eg'$dev'0' c $major 0
    mknod '/dev/eg'$dev'1' c $major 1
    mknod '/dev/eg'$dev'2' c $major 2
    mknod '/dev/eg'$dev'3' c $major 3
    chgrp mrf '/dev/eg'$dev[0-3]
    chmod g+w '/dev/eg'$dev'3'
done

majors=$(awk "\$2==\"mrfevr\" {print \$1}" /proc/devices)

echo "Found" $(echo $majors | wc -w) "Event Receivers."
echo "Creating device nodes..."

device=1
for major in $majors; do
    dev=$(echo $device | awk '{ printf "%c", 96+ $1}')
    device=$((++device))
    rm -f /dev/er$dev[0-3]
    echo "Creating nodes /dev/er"$dev"[0-3] for major" $major
    mknod '/dev/er'$dev'0' c $major 0
    mknod '/dev/er'$dev'1' c $major 1
    mknod '/dev/er'$dev'2' c $major 2
    mknod '/dev/er'$dev'3' c $major 3
    chgrp mrf '/dev/er'$dev[0-3]
    chmod g+w '/dev/er'$dev'3'
done
```

APPENDIX: module_unload script

```
#!/bin/sh

echo "Unloading modules"

/sbin/rmmod pci_mrfevg
/sbin/rmmod pci_mrfevr

echo "Removing device nodes"
rm -rf /dev/eg[a-h][0-3]
rm -rf /dev/eg[a-h][0-3]
```