Technical Note TN2000

PCI Expansion ROMs and You

CONTENTS

During development, a developer may want to confirm that the expansion ROM on a plug in PCI card is present and accessible. Dumping out the expansion ROM using the Open Firmware User Interface or Display Name Registry and PCI Peek under Traditional Mac OS is an easy way to confirm the ROMs existence and contents.

This Technote describes two techniques for reading the contents of a PCI expansion ROM. The first technique uses various Open Firmware words in the Open Firmware User Interface. The second technique uses Display Name Registry, PCI Peek, and MacsBug to read an expansion ROM from the Traditional Mac OS. PCI Peek and Display Name Registry are available as part of the PCI DDK.

Reading an expansion ROM from Mac OS X or from the Classic environment under Mac OS X is not covered in this Technote.

This Note is directed mainly at developers of PCI plug in cards for Traditional Mac OS.

Updated: [Jan 12 2001]

PCI Basics

Header Types

PCI cards must support a standard Configuration Space header. As of this writing there are three different types of headers supported. The header type is indicated by the value in the Header Type register in Configuration Space. In certain headers there is an Expansion ROM Base Address Register (BAR) indicating the presence (or absence) of an expansion ROM:

Header Type	Header Type register	Expansion ROM BAR offset
Zero	0x00	0x30
One	0x01	0x38
Two	0x02	Not supported

Header Type Zero is used for most PCI devices, like SCSI cards, Ethernet adapters, etc. Header Type One is used for PCI-to-PCI bridges, such as connecting your Macintosh to an expansion chassis to provide more PCI slots. Header Type Two is used for PCI-to-Cardbus bridges. Cardbus bridges do not currently support an expansion ROM BAR.

Back to top

Reading an expansion ROM

You can read the expansion ROM on a PCI card in two ways: using Open Firmware or using PCI Peek and MacsBug under traditional Mac OS. Here're the basic steps you'll need to accomplish to read an expansion ROM:

1. Find out if the PCI card of interest has an expansion ROM
2. Enable the ROM
3. Map the ROM into memory (Open Firmware only)
4. Enable Memory Space access to the card
5. Dump the ROM

Back to top

Reading an expansion ROM from Open Firmware

1) Find out if the PCI card has an expansion ROM

To find out if the card has an expansion ROM we need to look at the Configuration Space header. Move down the device-tree to the card's node and look at the properties there using the .properties word. You should see something like this:

```
0 > .properties
vendor-id               00001000
device-id               00000004
revision-id             00000003
class-code              00010000
interrupts              00000001
min-grant               00000008
max-latency             00000040
devsel-speed            00000001
name                    pci1000,4
compatible              pci1000,4
                        pciclass,010000
reg                     00011800 00000000 00000000  00000000 00000000
                        01011810 00000000 00000000  00000000 00000100
                        02011814 00000000 00000000  00000000 00000100
                        02011830 00000000 00000000  00000000 00008000
assigned-addresses      81011810 00000000 00001000  00000000 00000100
                        82011830 00000000 80888000  00000000 00008000
                        82011814 00000000 80880000  00000000 00000100
```

(For an in-depth description of these properties see PCI Bus Binding to Open Firmware Rev. 2.1 (PDF file))

Notice the first long word in the "reg" property. This is the Configuration Space address of the device. We can use this address to look at the Header Type register in the Configuration header of the device. The parent of our PCI plug in card is, in general, a PCI-PCI bridge. We need to open our parent so we can read the Header Type register in Configuration Space:

```
 ok
0 > " .." find-device open  ok
1 > . ffffffff
 ok
0 >
```

Here the string " .." refers to our parent device. Calling find-device makes our parent the active package, and calling open prepares it for further use. After we call open we need to check the return value on the stack to see if the open was successful. We display the top stack item using the . word. A return value of 0xffffffff indicates success (true), and a return value of 0 indicates failure (false). Once we have our parent node open we can read the Header Type register:

```
0 > 1180e config-b@  ok
1 > . 0
 ok
0 >
```

The address 1180e is the Configuration Space address we want to read. The word config-b@ reads a byte from this address. The . word displays the top item on the stack, which is the Header Type register. Since this is a Header Type

Zero, we know the expansion ROM BAR is at offset 0x30.

Looking at the assigned-addresses property we see an entry for a register at offset 0x30:

```
"assigned-addresses" property
82011830 00000000 80888000  00000000 00008000

(The low-byte of the first long word indicates the offset in Configuration Space)
```

So we know there is an expansion ROM that has been assigned a base address of 0x80888000 and is 0x8000 bytes long.

2) Enable the ROM

Bit 0 of the expansion ROM BAR enables the address decoder for the ROM. We can set bit 0 by reading in the BAR, OR-ing in 0x01 and writing this value to the BAR:

```
0 > 11830 dup config-l@ 1 or swap config-l!  ok
```

3) Map the ROM into memory (Open Firmware only)

Since we're in Open Firmware, we need to manually map in the expansion ROM so we can read it. We can do this by using the do-map word. The base address assigned by Open Firmware can be found in the assigned-addresses property for the BAR:

```
82011830 00000000 80888000  00000000 00008000
```

The third long word, 0x80888000, is the address we want. The last long word, 0x00008000, is the length of the ROM. We can map in the ROM like this:

```
0 > 80888000 dup 8000 28 do-map  ok
0 >
```

The do-map word creates an address translation. We pass it a physical address (80888000), a virtual address (the same address, put on the stack by the dup word), a length (8000), and a parameter (28). The do-map word then calls the map method of the MMU to create the translation.

4) Enable Memory Space access to the card

By default, Open Firmware turns off Memory and I/O space accesses to plug-in cards. We can enable Memory space accesses by setting bit 1 of the command register (offset 0x04) in Configuration Space:

```
0 > 11804 dup config-w@ 2 or swap config-w!  ok
0 >
```

5) Dump the ROM

Now assuming all has gone well we can us the base address and length of the ROM with the dump word to see the contents of the ROM:

```
0 > 80888000 100 dump
80888000: ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff :................:
80888010: ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff :................:
80888020: ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff :................:
80888030: ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff :................:
80888040: ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff :................:
80888050: ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff :................:
80888060: ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff :................:
80888070: ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff :................:
80888080: ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff :................:
80888090: ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff :................:
808880a0: ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff :................:
808880b0: ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff :................:
808880c0: ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff :................:
808880d0: ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff :................:
808880e0: ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff :................:
808880f0: ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff :................: ok
0 >
```

Back to top

Reading an expansion ROM from Traditional Mac OS

To dump an expansion ROM using the Traditional Mac OS, you'll need 3 tools: Display Name Registry, PCIPeek, and MacsBug.

First you'll need to know the name of your device. Use Display Name Registry to find the name of your device. PCI plug-in cards will appear under the Devices:device-tree:pci:pci-bridge node. You'll need the name of your device (or its unit address) to access the device using PCIPeek.

Once you find the name of your device, launch PCIPeek. You can use PCIPeek to spy on and manipulate PCI devices. When you launch PCIPeek, a bunch of text will scroll by. This text includes descriptions of the commands that PCIPeek understands and the devices that it found. (Hopefully, one of the devices is yours.)

You should see this prompt:

```
Enter new node name =>
```

Type in the name of your device. You should then get an output something like this:

```
Enter new node name => pci1000,4
Node info:
    Name:           pci1000,4
    Slot:           J10
    Unit-Address:   11800
        Bus Number:          1
        Device Number:       3
        Function Number:     0
>
```

This shows the name of your device, the slot it's in, and the unit-address. (The Bus Number, Device Number, and Function Number are encoded as part of the unit-address. For an in-depth description of the unit-address see PCI Bus Binding to Open Firmware Rev. 2.1 section 2.2 "Address Formats and Representations" (PDF file).)

Now that we've selected the device, let's look at the Configuration Space header. Enter c and press return. You should get a dump of the entire Configuration Space header like this:

```
> c
Vendor ID         = 1000     | 0x00
Device ID         = 0004     | 0x02
Command           = 0004     | 0x04
Status            = 0200     | 0x06
Revision ID       = 03       | 0x08
Class Code        = 010000   | 0x09
Cache line size   = 00       | 0x0C
Latency           = 10       | 0x0D
Header type       = 00       | 0x0E
BIST              = 00       | 0x0F
Base addr 0       = 00001401 | 0x10
Base addr 1       = 80880000 | 0x14
Base addr 2       = 00000000 | 0x18
Base addr 3       = 00000000 | 0x1C
Base addr 4       = 00000000 | 0x20
Base addr 5       = 00000000 | 0x24
Cardbus CIS Ptr   = 00000000 | 0x28
Subsys Vendor ID = 0000     | 0x2C
Subsys ID         = 0000     | 0x2E
ROM base          = 80888000 | 0x30
Reserved          = 00000000 | 0x34
Reserved          = 00000000 | 0x38
Interrupt line    = 00       | 0x3C
Interrupt pin     = 01       | 0x3D
Min_Gnt           = 08       | 0x3E
Max_Lat           = 40       | 0x3F
>
```

1) Find out if the PCI card has an expansion ROM

PCIPeek knows something about Configuration Space. As you can see in this example, it has correctly identified this as a Header Type Zero. Note that it shows us a ROM base field at offset 0x30: this is the base address of the ROM. Notice also that what it doesn't show us is the *length* of the ROM. We can find the length by looking at the reg or assigned-addresses property for the device using Display Name Registry.

2) Enable the ROM

We enable the ROM by setting bit 0 in the ROM base register. You can do this in PCIPeek by using the slc command (set long configuration):

```
>slc 11830 80888001
>
```

Now if you use the c command you'll see that bit 0 of the ROM base register is set.

4) Enable Memory Space access to the card

You can enable Memory space access by setting bit 1 of the Command register (the Command field shown using the c command). You can do this using the swc command (set word configuration):

```
>swc 11804 6
>
```

5) Dump the ROM

Here MacsBug is your friend. Note the base address of the ROM (the ROM base field). Break into MacsBug. Type dm <ROM address> 100 and you'll see the first 0x100 bytes of the ROM:

```
NMI
dm 80888000 100
 Displaying memory from 80888000

  80888000  FFFF FFFF FFFF FFFF  FFFF FFFF FFFF FFFF  ................
  80888010  FFFF FFFF FFFF FFFF  FFFF FFFF FFFF FFFF  ................
  80888020  FFFF FFFF FFFF FFFF  FFFF FFFF FFFF FFFF  ................
  80888030  FFFF FFFF FFFF FFFF  FFFF FFFF FFFF FFFF  ................
  80888040  FFFF FFFF FFFF FFFF  FFFF FFFF FFFF FFFF  ................
  80888050  FFFF FFFF FFFF FFFF  FFFF FFFF FFFF FFFF  ................
  80888060  FFFF FFFF FFFF FFFF  FFFF FFFF FFFF FFFF  ................
  80888070  FFFF FFFF FFFF FFFF  FFFF FFFF FFFF FFFF  ................
  80888080  FFFF FFFF FFFF FFFF  FFFF FFFF FFFF FFFF  ................
  80888090  FFFF FFFF FFFF FFFF  FFFF FFFF FFFF FFFF  ................
  808880A0  FFFF FFFF FFFF FFFF  FFFF FFFF FFFF FFFF  ................
  808880B0  FFFF FFFF FFFF FFFF  FFFF FFFF FFFF FFFF  ................
  808880C0  FFFF FFFF FFFF FFFF  FFFF FFFF FFFF FFFF  ................
  808880D0  FFFF FFFF FFFF FFFF  FFFF FFFF FFFF FFFF  ................
  808880E0  FFFF FFFF FFFF FFFF  FFFF FFFF FFFF FFFF  ................
  808880F0  FFFF FFFF FFFF FFFF  FFFF FFFF FFFF FFFF  ................
```

Back to top

Summary

An expansion ROM is an optional ROM that PCI card developers can build into their devices. An expansion ROM can contain various drivers: FCode drivers are required for any devices that wish to participate in the boot process, for example display cards, SCSI, or ATA cards. A native driver (`'ndrv'`) is used to control a PCI card under Mac OS. By having an FCode and/or native driver (`'ndrv'`) in an expansion ROM a PCI card can achieve true "Plug 'n' Play" compatibility.

References

PCI Bus Binding to Open Firmware Rev. 2.1 (PDF file)

PCI System Architecture, Fourth Edition, published by MindShare, Inc.
ISBN: 0-201-30974-2

Back to top

Downloadables

	Acrobat version of this Note (96K)	Download
	PCI DDK 3.0 (3000K)	Download

Back to top

Technical Notes by API | Date | Number | Technology | Title

Developer Documentation | Technical Q&As | Development Kits | Sample Code