T E C H N O T E :
Understanding PCI Expansion ROM Choices for Copland

by Wayne Flansburg
W.Flansburg@applelink.apple.com
Apple Developer Technical Support (DTS)

This Technote addresses PCI expansion ROM contents for Copland. The information here also applies to all Macintosh computers with PCI expansion capability, such as the Power Macintosh 9500, 8500, and 7500. The Note looks at the basic device types as defined in the IEEE 1275-1994 Open Firmware standard. It defines the recommended contents common for all ROMs, as well as the specific recommended contents for the various standard device types, excluding bridge devices.

This Note is intended for PCI expansion device driver writers. It can also be of value to board designers because it discusses the various ROM contents that can supersede the values that are hard-coded into PCI Configuration Space. It also addresses why the board designer might want to constrain the design based on the various ROM contents, such as specifying maximum memory requirements in Configuration Space and particular requirements in the “reg”

property — that is, using the Configuration Space register to define memory requirements for one OS and using properties for another OS.

This Technote assumes you understand the Open Firmware process integral to the newer Macintosh computers that use the PCI-I/O expansion model. It also assumes you understand "properties" and know how to create and use them.

Providing Open Firmware Support

Since the ROM contents is ultimately decided by third-party developers, the first issue to address is what level of Open Firmware support the device will provide. This can be divided into the following categories:

- no support
- minimum support
- full support

For each level of support, this Note addresses what can and cannot be accomplished by the device for Copland or other operating systems the device may target.

Defining "Boot" and "Runtime" Drivers

At this point, before explaining the different levels of Open Firmware support provided by the device, we need to distinguish between "boot" and "runtime" drivers.

IMPORTANT

Boot drivers are programmed in FCode, a byte-encoded OS (Operating System) and ISA (Instruction Set Architecture) independent form of software. They are used before control has been passed to the main operating system by Open Firmware and its client. Note that the operating system can be other than the Mac OS.

Runtime drivers are OS-dependent and are, in the context of this Note, Macintosh drivers. ◆

No Support of Open Firmware

The first type of Open Firmware support is that of no support. This lack of support does not require an expansion ROM at all, although an expansion ROM with no FCode is also contained in this category. However, placing an expansion ROM on a device costs money, and it makes little sense to provide an empty ROM.

Problems With This Approach

There are problems that you should consider with this type of support: 1) not being plug-and-play, 2) having an ambiguous device driver name, 3) wasting memory space, since using base registers only allow memory sizes to be multiples of powers of two, and 4) not being able to define sub-apertures. Sub-apertures are defined using the Configuration Space base registers. A typical example is a display device with a frame buffer, video controller, and RAMDAC. By defining three base registers, one for each function on the device, families can write memory differently than they do registers. This is useful for caching.

No Plug-and-Play

Without an expansion ROM, the device won't be plug and play. Nor will it be available to the user until the file system is initialized because the driver container must be placed on a hard disk and loaded from there. Note that the driver container does not necessarily have to be place on a hard disk: it could, for example, be located on a network. However, the device and driver will be matched and loaded.

Ambiguous Device Driver Name

Another issue to consider is that a distinct device "name" property cannot be provided when there is no expansion ROM. This makes unique device/driver matching ambiguous. For example, two manufacturers may supply a similar driver for a particular device, or a chip vendor may rev a PCI interface chip used on your device. Open Firmware during its probing process first looks for a name property in the Expansion ROM. It then looks for the Subsystem ID and Subsystem Vendor ID. And finally, it will choose the Device ID and Vendor ID if the name property is not defined and the Subsystem ID and Subsystem Vendor ID are zero.

Runtime drivers may fall into the no-support category but boot drivers cannot exist without an expansion ROM and therefore do not belong in this category.

The no-support category is populated mostly by applications controlling a device via a private driver. The device is not available to other applications. Since the name may not be unique, the driver should provide additional software to make sure that the device it has been matched with by the operating system is indeed its device. This could be some sort of a signature diagnostic. For instance, all drivers have a validateHardware interface that is called by the family that instantiates the driver to validate that the match is correct and the device is functional.

Minimum Support of Open Firmware

To achieve minimum support requires an expansion ROM. Minimum support or Run Time Support requires a driver,AAPL,MacOS,PowerPC property. It should also include a “name” property. This will eliminate ambiguous driver/devices matching and provide plug-and-play.

Full Support of Open Firmware

The last category — full support — includes but is not limited to unambiguous driver/device matching, plug-and-play support, and boot support. The presence of a “name” property in the ROM guarantees matching, the presence of a runtime driver in the ROM guarantees plug-and-play, and the presence of a boot driver guarantees Open Firmware driver support such as display and network devices.

Properties Common To All Device Types

Given that your device has an expansion ROM, there are properties common to all devices as well as device-specific properties. What follows is an introduction to those properties that are defined by the IEEE 1275-1994 Specification, the PCI Binding Specification, and Apple Computer.

The "name" Property

The “name” property is uniquely used to define the name of your node in the Open Firmware device tree. This is also the name of your device. This is the only required property. If you do not supply this property, the Open Firmware is obliged to construct one on your behalf because it is required. This is the algorithm. The Open Firmware probing process first looks for FCode to define this property in the device tree. If it is not present, it then uses the Subsystem ID and SystemVendor ID that is hard-wired in Configuration Space on your device. If those fields are zero, it then uses the Vendor ID and Device ID also in the Configuration Space. It will construct a “name” property of the form pcixxxx,yyyy. The yyyy field can be the System ID or Device ID while the xxxx field can be a SubsystemVendor ID or Vendor ID.

The "reg" Property

The "reg" property is used to define your device’s PCI memory-mapped areas including I/O space. If a "reg" property is contained in your expansion ROM, it will be used. If not, the Open Firmware probing process constructs one for you. It does so by writing all 1’s to the Base Address Registers. (0x10 through 0x24) in Configuration Space. It then reads back the values. These registers are hardwired in powers of two to define your memory needs. Bits returning a zero effectively define the memory requirements. See the PCI Local Bus Specification Revision 2.1 Section 6.2.5 Base Addresses for details. A word about the "assigned-addresses" property. This property is also generated during the probing process from the "reg" property. It is not, however, contained in the expansion ROM but is placed in the device tree. A Technote on these memory related properties and the device tree is scheduled for release mid 1996.

The "device-type" Property

The "device-type" property defines the device by function such as display, network, block, and so on. A word of caution is required. All device types except type “pci” use a PCI Configuration Register Map. Device type “pci” uses a PCI-to-PCI Bridge Register Map. See *Designing PCI Cards and Drivers for Power Macintosh Computers,* Chapter 4, “Startup & System Configuration,” Figures 4-1 and 4-2 for details.

The "interrupts" Property

Use this property to define whether your device can generate interrupts. Its absence defines no interrupt needs. Since all interrupt pins on the PCI bridge chip are “ORed” together, there can be only one interrupt source on your device. Your device will be required to supply, in the case of multiple sources, a method for the driver to determine which source is the one requiring service.

The "model" Property

The model number or name of your device.

The "address" Property

Used to define large virtual address regions. Currently unused by Open Firmware.

The "compatible" Property

Used to define alternate "name" property values. You would typically use this property to tell the Open Firmware that there are other device choices that can be matched with a driver, given the original "name" property could not be matched. The compatible property allows other drivers to match against your device. For example, consider an XYZ networking card that is register-compatible with an Apple networking card. In this case, the name property would be "XYZ,xxx,yyy" and, correspondingly, the compatible property would be AAPL,xxx,yyy".

The "status" Property

The status of your device. Use this when you run diagnostics on your device during the Open Firmware boot process and you wish to report the results to Open Firmware. Currently unused by Open Firmware.

Standard Device Nodes and Their Properties

That concludes our discussion of the properties that are common to all device nodes. Now let's look at some standard device nodes and their properties. The most common five are: "block", "byte", "display", "network", and "serial".

Table1 Standard devices, properties and methods

Device	Property	Function
block	open	Prepares device for use
	close	Closes previously opened device
	read	Moves device memory contents to main memory
	write	Moves main memory contents to device memory
	seek	Sets device position
	load	Loads a client program from device to main memory
		Note: To be bootable, block devices must support disk-label and deblocker packages and implement associated methods, such as read-blocks.
byte	open	Prepares device for use
	close	Closes previously opened device
	read	Moves device memory contents to main memory
	write	Moves main memory contents to device memory
	seek	Sets device position
	load	Loads a client program from device to main memory
display	open	Prepares device for use
	close	Closes previously opened device
	write	Puts characters on boot screen
	draw-logo	Calls the FCode function to draw caller's bit-mapped logo
	restore	Resets your device
	character-set	Defines which standard set of characters your device supports, such as ISO8859-1
network	open	Prepares device for use
	close	Closes previously opened device
	read	Moves device memory contents to main memory
	write	Moves main memory contents to device memory
	load	Loads a client program from device to main memory
	local-mac-address	The preassigned network address. The mac stands for Media Access Control and should not be confused with the Macintosh.
	mac-address	Contains the last used network address
	address-bits	Contains the length of the network address
	max-frame-size	Contains the maximum packet size

Device	Property	Function
serial	open	Prepares device for use
	close	Closes previously opened device
	read	Moves device memory contents to main memory, i.e., character
	write	Moves main memory contents to device memory, i.e., character
	restore	Resets your device
	install-abort	When used, polls for a console abort sequence
	remove-abort	Reverses the effect of install-abort
	ring-bell	Need no further comment

Note
Using the Open Firmware User Interface to look at the mace device tree node, which is the motherboard ASIC for controlling the Ethernet, gives the following information about that device. It is of type "network". It uses the "local-mac-address" but does not have need for the last used network address. It has 48 address bits and can transfer a packet no bigger that 2048 bytes in size. ◆

Summary

This Technote introduces you to the possible expansion ROM contents for all PCI Macintosh computers. Copland will run on both NuBus and PCI Macintoshes, but NuBus machines have no expansion ROM and therefore this Note does not apply to them. The least risky choice between no ROM, minimum ROM, and full ROM is, of course, full support, but the actual decision is constrained by how the board will be used. This Note is intended to make you aware of your ROM-content choices.

But it must be stated that NuBus devices that want to be boot devices will be able to supply Open Firmware just like a PCI device. Additionally, we expect/hope to see Open Firmware code for ALL devices, so that we can create address spaces correctly and install the right set of device nodes and interrupt tree nodes for multi-function cards.

A closing caveat is that device type 'ndrv' was not addressed in this Note. This type is an Apple-derived type to allow developers to move to PCI before certain families are introduced into the OS. With time, all device types should use families. However, 'ndrv' is a valid type and can be used. Having an expansion ROM on a 'ndrv' device can allow the driver writer to at least generate a "name" property with its unambiguous value. This alone will prevent a third-party manufacturer from having to recall its drivers to revise its DriverDescription data structure.

Further Reference

- Starting FORTH; Brody
- Writing FCode Program for PCI
- PCI Local Bus Specification Revision 2.1

Acknowledgments

Thanks to Monte Benaresh, Ron Hochsprung, Pradeep Kathail, Holly Knight, and Samuel Yan.