Technical Note TN1001

On Power Macintosh Interrupt Management

CONTENTS

The Note briefly addresses porting existing 68K interrupt code to the NuBus PowerPC. It then discusses the new interrupt management scheme developed for PCI PowerPC.

This Note is written for driver writers developing drivers of type 'ndrv' for PCI Macintosh computers and for any developer whose code makes calls to the Device Manager as a client of these drivers.

[Oct 01 1995]

About Power Macintosh Interrupt Management

Both the 68K and PowerPC microprocessors can be run in two modes: user and supervisor. The Mac OS runs in supervisor mode on both 68K and PowerPC Macintoshes. On 68K Macintoshes, the Mac OS allows any code to access supervisor mode. All code, therefore, has access to the status register, which can be used to program interrupts. With the introduction of the PowerPC, the Mac OS enforced the lock between the PowerPC user and supervisor modes. As a result, the PowerPC status register is no longer available for working with interrupts. To compound the interrupt problem, the 68K microprocessor has seven levels of available interrupts while the RISC- based PowerPC only has one interrupt level, requiring a different interrupt model.

> **Note:**
>
> Not all interrupt levels are used by the 68K Mac OS.

Porting 68K Interrupt Code to PowerPC

All code dealing with 68K interrupts (seven levels) is emulated on the PowerPC, since there is no transformation from seven levels to one. The mechanism for dealing with 68K code on PowerPC is the 68LC040 emulator. Therefore, developers may choose not to port at all and can be assured that their 68K interrupt code will run on the PowerPC.

For those few who choose to port parts of 68K 'DRVR'driver to a native PowerPC code, 68K non-interrupt code can be ported, but the interrupt handling code must remain emulated because a native interrupt handler would not be able to access the PowerPC status register to block interrupts. If you were to use any PowerPC assembly language instructions dealing with that status register, you would cause a fatal supervisory exception. By leaving the interrupt handler as 68K code, that code could continue to "access" the 68K status register because the emulator provides these services to drivers. The native portions of these drivers will have to use UPPs to interface correctly with the 68K Device Manager. Even though the PowerPC code will have to go through a mixed mode switch for interrupt handling, the developer may choose to

go this way because the bulk of the remaining code is PowerPC and could increase performance. This Note, however, does not address the details of using the Mixed Mode Manager.

See *Inside Macintosh: PowerPC System Software* for using 68K code with PowerPC code via the Mixed Mode Manager.

Note:

For more details, see *Inside Macintosh: PowerPC System Software* , p. 1-6.

Back to top

PCI-Based Power Macintosh Interrupt Management

With the introduction of the 9500 Power Macintosh, Apple has replaced its NuBus I/O model with the PCI I/O model and defined a new interrupt management scheme. This scheme is explained in detail in *Designing PCI Cards and Drivers for Power Macintosh Computers* , Chapter 9, Driver Services Library (DSL).

In this new I/O model, there are multiple levels of executions, including application, secondary, and primary execution levels that replace the old method of using interrupts to protect data. The primary and secondary execution levels are specifically for drivers and cannot use Toolbox services. Note that not a single Toolbox Manager is available. The services supplied at these primary and secondary levels via the Driver Services Library (DSL) include interrupt services. Other services, which replace the need to turn off interrupts that protect your data, include:

- memory management
- timing services
- atomic operations
- queue operations
- string operations
- debugging support

There are limitations placed on the execution context for some of these services. If you're an application developer, this means that some of these services - including interrupts - are off limits to your application. For the PCI driver writer, it means that these services, which exclude the Toolbox, are now your domain.

Again, there is no way to turn off the one and only interrupt on the PowerPC. If you are writing a driver, you may need to deal with a hardware-generated interrupts, assuming your device generates one. For example, a video driver may not generate one while a network driver most likely would. Also, there are new mechanisms and assumptions on how to deal with hardware generated interrupts.

The following example is one type of transactions between an application and a 'ndrv' driver. There are other transactions at present and there will be more types defined with future releases of the Mac OS.

An Example - A Simple Read Transaction

Consider the following scenario. A PCI card in an expansion slot is capable of asynchronously reading data from an external source into the Macintosh main memory. This basic sample will describe this read transaction and what must be done to service the device generated interrupt. But there is an important caveat here. The driver in this example is a generic PCI driver of type 'ndrv', as defined in *Designing PCI Cards and Drivers for Power Macintosh Computers* .

By choosing to write this kind of driver and following all of the guidelines in the above manual, you guarantee compatibility with future releases of the MacOS.

When the driver is loaded into memory by the Mac OS, the driver installs its interrupt routines and data in the Interrupt Source Tree (IST). It does so by using the Driver Services Library (DSL) routine `InstallInterruptFunctions`. The routines are used to disable, enable, and service device interrupts.

At some point, code running at the task level makes a PBRead asynchronous call to the driver via the Device Manager using a data structure called an `I/O Parameter Block (IOPB)`. Of special note here is that in the past, calls such as this one may have had private pointers contained in the parameter block that was passed to the driver. This will *not* work in

future releases of the Mac OS because of the pending implementation of multiple address spaces. Publicly-documented buffer pointers and IOPB structures will continue to be supported.

The Device Manager makes a call to `DoDriverIO` with a read selector. The driver does what is necessary to set up its device for an asynchronous read and returns to the Device Manager with no return error and no indication that the read has been completed. The driver has completed its first task. The Device Manager returns to the caller. When the read has completed, the hardware generates an interrupt which is detected by the Mac OS. Using the interrupt structure which contains the address of the interrupt service routine (ISR) for the device, control is passed to the ISR.

The ISR is allowed to complete its function uninterrupted, which is why an ISR must make every effort to spend as little time as possible servicing its device at the primary interrupt level. The driver should then queue a secondary interrupt handler to do any remaining work to service the device and/or the data using the DSL function `QueueSecondaryInterruptHandler`. This ensures that interrupt latency for the system is kept to a minimum.

Returning Control to the Mac OS

Having completed those tasks, the ISR returns control to the Mac OS. Secondary interrupt handlers are run before control is returned to task level code but can still be preempted by other primary interrupt handlers. By using the secondary interrupt handler to complete non-device-related tasks belonging to the read transaction, the driver writer is not locking out other code and is behaving appropriately in a multitasking environment. The secondary interrupt handler finishes any remaining tasks if they exist and makes a call to `IOCommandISComplete` passing a completion status, in this case, "read was successful." The IOPB is updated by the Device Manager and control is passed back to the caller's I/O completion routine.

And the point of this example? *Interrupts are not available to applications, but there are ways for applications and interrupt handlers to exchange information* .

Note:

See *Designing PCI Cards and Drivers for Power Macintosh Computers* , Chapter 9, Memory Management Services, p.240.

Back to top

Summary

The PowerPC interrupt is not available to an application. Until the next major release of the Mac OS, you can use the emulated 68K interrupt mechanism. Using the interrupt on the PowerPC is not an option and would be catastrophic if attempted. Interrupts from expansion devices, however, are fully supported in PowerPC for both NuBus and PCI slots. For more information about how to manage expansion interrupts on the PowerPC with PCI, refer to *Designing PCI Cards and Drivers for Power Macintosh Computers* .

Back to top

References

Designing PCI Cards and Drivers for Power Macintosh Computers .

PowerPC 601 RISC microprocessor User's Manual MPC601UM/AD.

M68000 8-/16-/32-Bit microprocessors User's Manual, Sixth Edition, M68000UM/AD Rev 5.

Inside Macintosh: PowerPC System Software , Addison-Wesley.

Back to top

Downloadables

Acrobat version of this Note (92K) Download

Back to top

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

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