Technotes

Download  Acrobat file (254K)	Download  AppleWorks file (60K)

Understanding PCI Bus Performance

Technote 1008 **OCTOBER 1995**

With the second generation of Power Macintosh computers, Apple has transitioned the Macintosh I/O expansion bus from NuBus(TM) to PCI. Apple's underlying policy is to support the PCI standard, as expressed in the PCI Local Bus Specification, Revision 2.0.

The adoption of the PCI standard brings many advantages to the Macintosh platform. Arguably, one of the most significant is increased I/O bandwidth. Developers frequently ask questions on PCI bus commands with an eye toward bus performance. This Technote examines the PCI bus commands, the operation of the IB chip (the PowerPC processor to PCI interface bridge chip), achievable PCI bandwidth on PCI Power Macintosh computers, and, finally, Mac OS services available to maximize PCI bandwidth.

This Technote is written for PCI hardware designers and driver writers who are developing for the Power Macintosh platform.

Contents

About PCI Performance on the Power Macintosh

A good place to start addressing PCI performance on Power Macintosh CPUs is the PCI standard itself. The Bus Specification, Revision 2.0, features a 32-bit data path -- upgradeable to 64-bits -- with synchronous bus operation up to 33 Mhz, and the ability to transfer a data object on the raising edge of each PCI clock cycle. Assuming that neither the initiator nor the target inserts wait states during each data phase, the maximum theoretical bandwidth over a 32-bit bus is 132 Mbytes/second. This also assumes continuous bursting with a 32-bit data object transferred on each PCI clock cycle. (Apple's implementation incorporates a 32-bit data bus.)

Since the IB chip competes for system memory along with other system devices, continuous PCI bursting is not possible. Therefore, the achievable PCI bandwidth on Power Macintosh computers -- a significant improvement from NuBus -- will be less than the PCI theoretical maximum. Also, the bandwidth will be dependent on the PCI target's hardware design and the architecture of the driver software.

A PCI burst transfer is defined by one PCI bus transaction with a signal address phase followed by two or more data phases. One may ask, how can the bus master transfer a data object on each PCI clock cycle? To initiate a bus transaction, the PCI master only has to arbitrate for ownership of the bus one

time. The master then issues the start address and transaction type during the address phase. It's the responsibility of the target device to latch the start address into an address counter and increment the addressing from data phase to data phase. (A single-beat read or write transaction is defined by a signal address phase followed by only one data phase.)

For data to be transferred between the PowerPC Processor and the PCI Target, or for the PCI Target to transfer data between system memory, one of the following commands is initiated, as shown in Table 1.

Table 1 Commands between PowerPC Processor and PCI Bus

PCI Command	Initiator
I/O Read	Processor generated
I/O Write	Processor generated
Configuration Read	Processor generated
Configuration Write	Processor generated
Memory Read	Processor or PCI Master generated
Memory Read Line	Processor or PCI Master generated
Memory Read Multiple	Processor or PCI Master generated
Memory Write	Processor or PCI Master generated
Memory Write and Invalidate	Processor or PCI Master generated

Table 1. Commands between PowerPC Processor and PCI Bus

Note:

The I/O Read and I/O Write commands are used to transfer data between the Processor and the Target's I/O space.

The Configuration Read and Configuration Write commands are used to transfer data between the Processor and the PCI target's Configuration registers during system initialization.

The Memory Read and Memory Write commands are used to transfer data between the PCI Master and the Target's memory space.

The Memory Read Line command is used by the PCI Master to transfer a cache line of data from the PCI Target's memory space.

The Memory Read Multiple command is used by the PCI Master to transfer more than one cache line of data from the PCI Target's memory space.

The Memory Write and Invalidate command is used by the PCI Master to transfer one or more complete cache lines of data to the PCI Target's memory space.

Note:

A cache line is 32-bytes for Apple Power Macintosh computers.

About the PowerPC Processor and PCI Commands

Now with the basics of the PCI bus under our belt, let's move on to important details regarding PCI Power Macintosh computers. The PowerPC processor has a 64-bit data bus and its system memory space defaults to write back cache mode, while the PCI bus is 32-bits wide and the PPC processor sets PCI address space to cache inhibit mode. For PPC initiated read and write transactions between PCI memory space, the IB chip (the PowerPC Processor to PCI Bridge) will initiate basically one of the three following types of PCI commands:

1. a single-beat Memory Read or Write command;
2. a Memory Read or Write command with two data phases -- defined as a burst transaction; and
3. a Memory Read Line or Memory Write and Invalidate command that bursts a 32-byte cache line.

Important:

The PPC processor will not burst to or from address space marked cache inhibited. Therefore, under default cache settings, the IB chip will not initiate the Memory Read Line or Memory Write and Invalidate commands to a PCI target.

As per the PCI Specification Revision 2.0, PCI Power Macintosh Computers support PCI I/O space. PCI I/O commands and Mac OS services available for them are addressed later in this Technote.

With the basics of the PCI bus described and details of the Power Macintosh PCI implementation outlined, this should be ample background to describe the functionality of the IB chip. In particular, under what circumstance will it perform what type of PCI command?

Bursting from PowerPC to PCI

Provided software is written to utilize floating-point load and store instructions -- as opposed to integer operations -- the IB chip will burst a two-beat Memory Read or Memory Write command (two 4-byte data phases with one PCI transaction). The PowerPC floating-point data is 8-bytes wide and integer data is 4-bytes. Utilizing floating-point instructions in effect nearly doubles the PCI bandwidth over single-beat PCI Memory Read or Write commands. This is worth investigating for solutions where the PCI hardware does not support cache line bursting.

If the PCI target's address space is set to write thru cache mode, the IB chip will perform an eight-beat burst read on PCI with the Memory Read Line command. This translates to a cache line, eight 4-byte long words, i.e. 32-bytes.

If the PCI target's address space is set to write back cache mode, the IB chip will perform an eight-beat burst write on PCI with the Memory Write and Invalidate command.

Important:

Extreme care must be taken for burst writes to PCI address space to perform appropriate cache flushing.

Bursting from PCI to PowerPC

If the address is aligned on an 8-byte boundary, the IB chip will respond to PCI Memory Read and Memory Write commands by a two-beat PCI transaction to align two 32-bit PCI data words to the 64-bit PowerPC bus. On non-8-byte-aligned addresses, single-beat transactions are implemented.

The PCI Memory Write and Invalidate command will perform an 8-beat transaction if the address is aligned on a 32-byte boundary.

The PCI Memory Read Line or Memory Read Multiple commands will perform an eight-beat transaction if the address is aligned to an address less than or equal to 8-bytes less than the next 32-byte boundary. The PCI Memory Read Line and Memory Read Multiple commands are treated the same by the IB chip, in either case the IB chip will disconnect after an eight-beat transaction -- one 32-byte cache line.

Note:

Keep in mind that the main memory space is set to write back cache mode.

As mentioned earlier, 132 Mbytes/sec is the maximum theoretical bandwidth across a 32-bit PCI bus at 33 Mhz. Table 2 and Table 3 show the maximum achievable bandwidth that can be expected, depending on the type of PCI transaction performed. Please note *these values are not guaranteed* but are realistic ranges that have been measured moving large buffers (many thousands of bytes) -- to average out PCI arbitration PCI wait states -- across a Power Macintosh Computer's PCI bus.

The numbers in Tables 2 and Table 3 are based on the following assumptions:

Bus Speeds:

- Processor Bus is running at minimally 40 Mhz
- PCI Bus is running at 33 Mhz

PCI Target responses during PowerPC Processor to PCI transactions:

- PCI Targets are medium DevSel_ timing with NO inserted wait states for reads and writes.
- PCI Target does not assert Stop_ to disconnect bus.

PCI Master requirements during PCI Master with System Memory transactions:

- PCI Master is able to source data within one clock of Frame_ assertion with no inserted wait states for subsequent data phases.
- PCI Master is able to sink data with no inserted wait states for subsequent data phases once the host bridge asserts Trdy_.
- PCI Master is able to start its next transaction within two clocks from the PCI Bus returning to the Idle state from its previous transaction.

Table 2 PowerPC Processor to PCI Maximum Bandwidth Summary

Bus Master	Transaction Description	#bytes/ trans.	PCI BW (MB/s)	PPC setup
Processor	Write To PCI	4	20	Integer Store
Processor	Write To PCI	8	40	FP Store
Processor	Write To PCI	32	85	PCI CopyBack
Processor	Read From PCI	4	11	Integer Load
Processor	Read From PCI	8	20	FP Load
Processor	Read From PCI	32	40	PCI WriteThru

Table 2. PowerPC Processor to PCI Maximum Bandwidth Summary

Table 3 PCI Master to System Memory Maximum Bandwidth Summary

Bus Master	Transaction Description	#bytes/ trans.	PCI BW (MB/s)	PPC command
PCI Master	Write To Memory	4	20	Mem Wr
PCI Master	Write To Memory	8	35	Mem Wr
PCI Master	Write To Memory	32	80	Mem Wr & Inv
PCI Master	Read From Memory	4	10	Mem Rd
PCI Master	Read From Memory	8	15	Mem Rd
PCI Master	Read From Memory	32	30	Mem Rd Ln/Mult

Table 3. PCI Master to System Memory Maximum Bandwidth Summary

Note:

For # of bytes/trans.: 4 indicates single-beat; 8 equals two-beats; and 32 is an 8-beat transaction.

About the Mac OS & Services to Maximize PCI Throughput

Now that the hardware level basics have been examined for PCI Power Macintosh Computers, let's move up to the Mac OS level and review services available to maximize PCI throughput. It's important to mention first that for second generation PCI Power Macintosh Computers, there is a new PCI driver environment -- or I/O architecture -- available in the reference release Mac OS version 7.5.2. Refer to Designing PCI Cards and Drivers for Power Macintosh Computers.

With this reference release OS, Apple starts to separate between APIs (Application Programming

Services) and SPI (System Programming Services). In this present Mac OS release and the future direction, such as Copland, APIs and toolbox services are no longer available to driver SW. The Mac OS version 7.5.2 provides a DSL (Driver Services Library) that implements all SPI services available for drivers; documented in *Designing PCI Cards and Drivers for Power Macintosh Computers* , Chapter 9.

To coordinate I/O operations that transfer buffers between system memory and PCI address space, the Macintosh OS provides two functions with the DSL (Driver Services Library): `PrepareMemoryForIO`, and `CheckpointIO`. The `PrepareMemoryForIO` function allocates resident system memory to buffers, provides logical and physical address information, and in conjunction with `CheckpointIO` manages coherency between system memory and the PowerPC caches. `CheckpointIO` is called after the buffer transfer is complete and either relinquishes the memory back to the OS and adjusts the processor caches for coherency, or prepares for another IO transfer.

Important:

> `PrepareMemoryForIO` should not be confused with PCI I/O space. It is for buffers whether they are located in PCI memory or PCI I/O space.

`PrepareMemoryForIO` is an example of a service in the DSL; PCI cards that have DMA hardware should use `PrepareMemoryForIO` to locate physical addresses in system memory. Older I/O expansion cards would typically use a toolbox call `GetPhysical` to locate physical addresses in system memory. To be fully compatible with the present and future Mac OS releases, drivers should only use SPI services. Again, this is fully documented in *Designing PCI Cards and Drivers for Power Macintosh Computers* .

Remembering that PCI address space defaults to cache inhibit mode, to enable the PowerPC to burst to areas of PCI memory space, that area must be set to cacheable setting. This can be done with the `SetProcessorCacheMode` (see chapter 9 in *Designing PCI Cards and Drivers for Power Macintosh* Computers). Set the desired PCI address space to `kProcessorCacheModeCopyBack` for cache line writes and `kProcessorCacheModeWriteThrough` for cache line reads.

Important:

> Extreme care must be taken for burst writes to PCI address space to perform appropriate cache flushing.

Be advised that the `SetProcessorCacheMode` has an undocumented limitation. The PowerPC address space is divided into sixteen 256-Mbyte segments that are distinguished by the upper 4-bits of the effective address. The `SetProcessorCacheMode` is only capable of changing the cache setting for one contiguous section of memory per 256-Mbyte segment. Therefore, if two PCI cards are configured where they both have PCI address assignments in the same segment only one card can change its address space cache setting.

As an example, if two cards (card x and card y) have addresses mapped into segment 8, one at 0x80800000 and another at 0x80801000, the first call to `SetProcessorCacheMode` from the driver of card x to make a cacheable address space in segment 8 will work. A second call, say from the driver of card y, to modify the cache setting in segment 8 will not work nor will it report an error. This scenario will most likely result in a lower than expected performance for card y, because card y address space is actually cache inhibited which disables PCI transactions of 32-byte cache lines. If the two cards are mapped into different segments, such as 8 and A, then they both can modify the cache settings within their perspective segments. This limitation will be relaxed in the future.

Extensions to the `BlockMove` routine have been incorporated in the DSL that optimizes performance on the PowerPC CPU family. In particular, `BlockMoveData` has been optimized for data that is cacheable and `BlockMoveDataUncached` for data that is cache inhibited. The difference between the cached and uncached versions of these instructions is that, for `BlockMoveData`, the PPC `dcbz` instruction is used to avoid the logically unnecessary read of the destination cache blocks. `BlockMoveDataUncached` does not use the `dcbz` instruction because `dcbz` is extremely slow for address space marked cache inhibited or cache write thru.

Table 4 lists the different BlockMove functions provided in the DSL

Table 4	BlockMove functions provided in the DSL
BlockMove	may be 68K code, cached destination memory
BlockMoveData	not 68K code, cached destination memory (Fastest version)
BlockMoveUncached	may be 68K code, uncached destination memory (Slowest version)
BlockMoveDataUncached	not 68K code, uncached destination memory
BlockZero	to zero cached memory
BlockZeroUncached	to zero uncached memory

Table 4. BlockMove functions provided in the DSL

The difference between `BlockMove` and `BlockMoveData` versions is whether or not the block being moved contains 68K instructions. If the data does contain 68K instructions `BlockMove` must be called which also flushes the DR (Dynamic Recompilation) Emulator's cache. This is costly time-wise, so if the block does not contain 68K instructions, be sure to use `BlockMoveData` or `BlockMoveDataUncached`. Also with performance in mind, when appropriate the `BlockMove` routines will align the source and destination address to utilize floating-point load and store instructions.

To summarize the `BlockMove` routines, for transfers of large buffers between PCI cards the `MoveBlockData` or `BlockMoveDataUncached` functions should be used, depending if the destination address space is marked write back cacheable or not. Furthermore, PCI drivers most likely will not need to consider the non-Data variant of the `BlockMove` routines because destination buffers either in PCI address space or system memory will probably not need to execute 68K code.

A common question from PCI developers is, how to initiate a PCI burst of a cache line? Provided the PCI address space is marked cacheable as explained earlier, the `BockMoveData` function will force the IB chip to burst 32-byte cache lines -- eight-beat data phases per PCI command transaction.

To read or write PCI I/O space, the Expansion Bus Manager provides routines to transfer data -- byte, word, or long word (8, 16, or 32 bits, respectively) -- using PCI I/O Read and I/O Write commands. The Expansion Bus Manager is part of the ROM firmware in PCI Power Macintosh CPUs. These routines also perform appropriate byte swapping. For a further description, refer to *Designing PCI Cards and Drivers for Power Macintosh Computers* , chapter 10. PCI cards that are limited to I/O space, and do not incorporate PCI memory space, are limited to PCI I/O Read and I/O Write commands to transfer data between the PPC and PCI target. If PCI I/O data needs to be processed quickly, note there is a significant performance hit using Expansion Manager Routines. These routines are intended for PCI targets that have I/O registers or low bandwidth I/O buffers. The IB chip does not burst PCI I/O Read nor burst PCI I/O Write commands.

As described in chapter 10 of *Designing PCI Cards and Drivers for Power Macintosh Computers* along with sample code, the PCI property "assigned-addresses" provides vector entries that represent physical addresses on PCI cards. Using the "APPL,address" property a driver can locate a logical address of a physical I/O resource. By accessing the logical I/O address the IB chip will generate the appropriate PCI I/O command. Therefore a driver can generate PCI I/O commands without using the Expansion Bus Manager Routines; the same way it accesses PCI memory space. This provides the fastest way to access I/O space, but note it does not perform byte swapping as the Expansion Bus Manager routines.

Also note, the Expansion Bus Manager provides OS services to generate PCI Configuration Read, Configuration Write, Interrupt Acknowledge, and Special Cycle commands.

Summary

The PCI bus on Power Macintosh computers delivers higher I/O performance along with lower costs and complexity from the previous NuBus architecture. PCI also represents an emerging standard in the desktop PC industry. To maximize bus performance, utilize the services available in the Driver Services

Library, and pay close attention to PCI chip selection -- in particular, chips that can execute cache line burst transactions with Memory Read Line, Memory Read Multiple, and Memory Write and Invalidate commands. And consider *Designing PCI Cards and Drivers For Power Macintosh Computers* as essential documentation for successful PCI development on the Mac platform.

Further References

- *Designing PCI Cards and Drivers For Power Macintosh Computers*
- Creating PCI Device Drivers, *develop* , The Apple Technical Journal, Issue 22
- The New Device Drivers: Memory Matters, *develop* , The Apple Technical Journal, Issue 24
