PCI DMA For PowerPC^{TM} Reference Platform

Gary Y Tsao
Power Personal Systems Division
IBM Corporation
Austin, Texas 78758
e-mail: tsao@futserv.austin.ibm.com
September 30, 1994

Abstract

The mechanism to support devices' Direct Memory Access (DMA) is not explicitly specified in the Peripheral Component Interconnect (PCI) bus. There are two major types of device DMA, first party as used in bus master and third party such as the 8237A ISA DMA controller. In this paper, we will explore the various options of DMA and make recommendation for the PowerPC^{TM} Reference Platform.

1. Introduction

References [1] and [2] do not explicitly specify the DMA model for the PCI bus. There are two major types of DMA I/O, first party and third party. The main difference in these types of DMA is who generates the addresses and bus control signals. For first party or DMA master operations the device arbitrates for the bus, obtain ownership, and provides bus addresses and control signals during the transfer. For a third party or DMA slave operations, the address generation and bus control for the transfer is controlled by the system DMA controller which usually is located in the PCI host bridge. In this case the device merely generates a DMA request, waits for acknowledgment from DMA controller and gates data at the appropriate times. In this mode, no addresses are sent on the bus by the device since the addresses are set up in the DMA controller by software before the operation is started. The choice of either DMA master or third party operation is built into the device (adapter) by the device developer.

PCI local bus is a processor-independent bus. A host bridge is required to interconnect the processor bus to the PCI bus. A memory controller can also be incorporated into the bridge controller. Most DMA operations will use the "memory read" or "memory write" commands on the PCI bus. For memory read operation, at least three cycles plus one turn-around cycle is required to perform a single data transfer. This transfer rate is only 33.33 Mbytes per second for a 32-bit bus at clock rate of 33 MHz. Clearly, each transaction has to transfer multiple objects (burst) to approach the 132 Mbytes per second data rate. The burst length is constrained by the PCI's latency requirement [1]. The best burst length is in the range of 16 to 32 data phases.

A typical sequence of events for a DMA read between a processor and a I/O device is as follows:

- The data buffers set aside by the processor must be stable (pinned) and accessible by the device. Explicit flushing of processor cache lines may be needed if coherency is not enforced by the hardware.
- Set-up the DMA channel for the transfer. This is needed only for third party DMA.
- The processor puts a command in the queue area of shared memory or in the device's command registers.
- The processor signals the device to execute the command (wakeup). The device either executes the command from the command register directly or fetches the command from the shared memory queue area.
- The DMA engine transfers data between the device and data buffer while the processor is performing other tasks.

- Device puts a completion status in shared memory queue area or in device's status register and signals this event to the processor.

- The processor processes this status, performs any post processing such as flushing the I/O controller buffer and delivers read data and status to the user's application.

As we can see above, a DMA operation or a data transfer command involves more than data transfers. The shared memory queue area can be a complex data structure. The DMA startup usually involves the operating system (OS) and the latency is non-trivial. An extra data copy from the pinned data buffer to another memory is often necessary. Thus, the overhead of the data transfer portion of a DMA operation could be a small portion of the total overhead. The various aspects of OS processing of DMA and the OS-DMA hardware interface probably dominate the performance of DMA operations.

In this paper we assume the DMA is real, i.e. no virtual DMA is considered here. What this means is that the data buffers used by DMA must be stable (no page fault) in system memory.

2. Example of Reference Implementation

The DMA controller (third party) in Intel 82378ZB (SIO) consists of the functionality of two 82C37A DMA controllers. This chip supports DMA operations between ISA or motherboard I/O devices and either ISA memory or system memory. So, it has nothing to do with DMA functions over the PCI bus. It merely arbitrates the PCI bus and uses the PCI bus for data transfer to system memory.

The NCR 53C810 SCSI Controller implements bus master DMA on the PCI bus. It also supports memory-to-memory DMA transfers. For memory-to-memory DMA features, this chip can be used as a third party DMA controller. The built-in SCSI-SCRIPTS processor fetches and executes SCRIPTS instructions from the memory. A typical DMA write sequence is as follows [3]:

- Data buffers are set up properly.

- The processor writes into the DMA SCRIPTS Pointer (DSP) register (through program I/O) the starting address in memory that points to a SCSI SCRIPTS program.

- The NCR 53C810 fetches its first instruction from this address through the PCI bus. It typically fetches two 32-bit words. If the first instruction (word) is a Block Move instruction, the lower three bytes of the first word are interpreted as the number of bytes to be moved. The second word is interpreted as the beginning address in memory to which the move is directed.

- The NCR 53C810 requests use of the PCI bus to fill its DMA FIFO (64 bytes). The DMA burst size is programmable (8/16/32/64 bytes) on the PCI bus. The DMA burst data transfer continues (repeatedly requesting and releasing the PCI bus) until the move byte count has reached zero.

- The NCR 53C810 fetches another SCRIPTS instruction by using the incremented DSP address. Execution of independent Block Move SCRIPTS instructions (for scatter/gather operations on data) continues until an error condition occurs or an interrupt SCRIPTS instruction is fetched. At this point, the NCR53C810 interrupts the processor and waits for further servicing by the host system.

The NCR 53C810 uses the proprietary SCRIPTS interface programming language and is optimized for the SCSI subsystems. It supports little endian byte ordering only. The NCR 53C825 operates in big or little endian mode.

Other high-performance PCI adapters, such as graphics, do not use DMA to transfer data for current Reference Implementation. However, the situation may change in the future especially in the areas of video capturing or playback of video frames.

3. IEEE P1212.1 DMA Models

IEEE 1212.1 [4] document, called a "DMA Framework", is an optional addition to the IEEE 1212-1991 CSR (Control or Status Register) Architecture [5]. The CSR Architecture defines a register set to be used for control, configuration, identification, and diagnostics between nodes on a system bus interconnect. The CSR architecture and DMA Framework are to be cross-referenced by other IEEE bus specifications, such as Futurebus+ (IEEE 896) or SerialBus (IEEE-1394).

The DMA Framework is on the communication between processor units and I/O units using shared memory. It also covers device-to-device DMA or three-party DMA. The DMA model concentrates on the operation of the shared memory queues (see Section 1). The queue items represent "event" messages that pass through the bus, either containing data and commands directly or pointing to buffers in system memory. For example, for a disk interface, an outbound queue item could represent the transaction initiation for a disk seek, and an inbound queue item could represent transaction completion. The transaction initiation and completion events are represented by messages. These event messages point to the shared memory where buffers and parameter block structures involving bulky or "long-term" storage are located.

Four DMA models are specified in this document:

1) Circular Queue: Circular queues are typically implemented as an array, and have two indexes associated with them, a producer index and a consumer index (also called the head and tail

index). They are operated in first-in-first-out (FIFO) fashion. A given queue is used to pass messages in one direction, from a single producer to a specific consumer. There is always one empty position in the queue to remove the ambiguity as to whether the queue is empty or full. For n queue items (numbered from 0 to n-1), the index starts over again at 0 when the index is moved past n-1. If the queue is not empty, then all queue items from the one pointed to by the consumer index to the item one behind the producer index are "owned" by the consumer. All queue items from the one pointed to by the producer index to the item two behind the consumer index are "owned" by the producer ("two" because of the reserved empty item). Based on these assumptions, the algorithms for enqueue and dequeue are fairly straightforward.

When applied to DMA, the processor writes to an I/O unit CSR to send it a signal, when necessary, that it has processed one or more messages in the circular queue. Similarly, the I/O unit sends an interrupt to the processor to signal it, when necessary, that it has affected a queue.

An attractive feature of circular queue is the simple synchronization mechanisms, no semaphores or special bus transactions are needed between the producer and consumer. Another feature is block access to messages. If the consumer falls behind the producer, several messages may be retrieved in one block read.

The synchronization mechanisms do not extend easily to multiple producers, so circular queues are hard to share. The scheme also requires to reserve a fixed, large contiguous block of memory for item storage. It is difficult to expand this storage on-the-fly. Queue items are fixed-size, which can be a problem to support variable-length messages.

2) Dispatch List: In this model, the DMA queues are one-way linked lists of "dispatch items." Each dispatch item contains a pointer to the next dispatch item and a message being passed to the consumer. The data structures in this scheme are operated in FIFO fashion: items are only added to the back and removed from the front. Unlike the circular queue, the consumer pointer (head of the list) and producer pointer (tail) are not visible to the other party. The producer and consumer rendezvous at the dispatch item at the tail of the list. Some means of synchronization between the producer and consumer is required to ensure there is only one writer at a time. For example, there is a race between the consumer taking the last message in the queue and the producer adding another.

When applied to DMA, the processor writes to an I/O unit CSR to send it a signal that it has processed one or more messages in the outbound queue. Similarly, the I/O unit sends an interrupt to the processor to signal it that it has affected the inbound queue.

There are two ways to allocate the dispatch items. One is to preallocate a worst-case number of dispatch items and stored in a free list. The other approach is to delay a processor’s I/O transaction initiation until a dispatch item is available for dynamic allocation.

There needs to be some method to allow for dispatch items to be recirculated. This recollection of dispatch items is conceptually simple but could be time-consuming.

3) Shareable List: This scheme applies the "swap" transactions over a dispatch list to support shared access by multiple producers. This can be useful, for example, in a multi-processor system to allow processors to pass outbound messages independently to an I/O unit. The swap is an atomic bus transaction. There are two swap primitives: always-swap and compare-swap. Always-swap atomically writes new value and return old value. Compare-swap atomically swaps if old value equals to test value. The PCI bus interface does not directly support the bus swap operation. In addition the memory controller must also support the atomic swap transactions.

The producers always apply the always-swap operation when entering the critical region, e.g. adding new entry to the list. The consumer always applies compare-swap operation when a potential race condition existing, e.g. the consumer taking the last message in the queue and the producer adding another.

Like other shared data structures via semaphores or locks, fast producers or consumers may be delayed by a slow producer sharing the queue.

4) Mailbox: A mailbox is a CSR that maps write data to I/O unit internal message storage locations. The mapping between CSRs and message storage locations is transparent to the one or more producers. A message is sent by a producer to a mailbox by performing an atomic block copy to the mailbox CSR. Normal bus arbitration provides synchronization of mailbox access among multiple producers. Following this message write, the mailbox hardware moves the mapping to the next message storage location to prepare for the next message write. The unit that owns the mailbox hides the complexity of managing the message storage area from the producers.

Mailbox is primarily designed to handle a large number of producers. Mailbox can be located in the I/O unit, a memory unit, a bridge to an I/O bus, or a special node on the system bus. Reading messages from the mailbox is implementation-dependent. The messages can be presented in simple FIFO or in order of software priority encoded in a message field.

Mailbox is conceptually simple, but the mailbox hardware may become the performance bottleneck.

Since communication between the processor and I/O units is inherently asymmetrical, and since each of the DMA models have different advantages and disadvantages, a given system design could combine a hybrid of the previous ones discussed. For example, a design can use one circular queue for each I/O unit in the outbound direction, and one mailbox per processor in the inbound direction. This makes sense because circular queue scheme works well for single producer while mailbox is attractive for multiple producers.

The above DMA models involved two active parties, the processor and the I/O unit. For three party DMA, the processor is the director of data flow between two I/O units, a producer and a consumer. This allows the processor to set up an extensive flow between I/O units that then proceeds with little additional intervention. This paper will not cover the detailed device-to-device DMA because it requires complicated peer-to-peer protocols.

4. IEEE P1285 DMA Model

IEEE Standard P1285 for Scalable Storage Interface (SSI) defines an interface for future physically small-sized, memory-mapped, non-volatile storage units. The interface should be suitable for direct attachment to printed circuit boards. This model emphasizes that storage units become part of main memory rather than attachments to an I/O channel, which simplifies the storage unit design.

Two separate levels of storage-device interface are defined. a low-level interface is appropriate for connecting cpu chips and storage devices on the motherboard. This low-level (beta) interface can be optimized for short distances and can assume that significant portions of the typical disk controller functionality can be done by the processor software. A higher level (gamma) interface is defined when storage devices are remotely located, are actively shared by multiple processors, or when the central processor can no longer perform time-sensitive controller operations (e.g. seek re-ordering, memory-to-memory copies). This paper will concentrate the gamma level interface which is more appropriate for PCI bus operation.

An SSI interface is visible through a set of memory-mapped registers. A node provides a configuration ROM, which allows the position and function of the I/O unit to be readily identified. The structure of the SSI resources within an I/O unit is standardized. An SSI interface consists of a range of memory-mapped addresses. These addresses can include externally-visible data buffers, device-dependent control registers, and one or more sets of transfer-unit registers. The I/O unit’s ROM parameters are expected to specify the structure, function, and size of the memory-mapped resources.

One resource is "ownership" registers which allows sharing of SSI interface by multiple processes. Each I/O node supports a number of transfer units (channels or DMA engines). The transfer unit registers are used to initialize and control data transfers. An initiator is the one which generates commands. The transfer unit is directed by initiator-generated command entries. The transfer unit can be stopped or paused, based on the "control" field of a command block. A "wakeup" mechanism is needed to reactivate transfer units after their state has changed. Transfer unit registers are listed as follows:

- Control register: 3 bits are written by initiator and read by the transfer-unit hardware.

- State register: 10 bits are written by transfer-unit hardware and read by the initiator.

- Wakeup register: 1 bit. Set by initiator to reactivate command-list processing.

- Headhandle register: 8-byte pointer to specify where the head-pointer for the command list is located.

- Tailhandle register: 8-bye address of command-list tail.

SSI uses memory-mapped interrupt dispatch. It uses "write4" [5] transaction to send 32 bits of data to a 4-byte aligned 64-bit address. After the status (e.g. command completion) has been written into memory, the interrupting "write" transaction is sent to the pre-specified processor. The PCI bus does not support memory-mapped interrupts. For PCI in SSI environment, when the data transfer is completed and the target's "interrupting" write transaction is sent to memory, the backplane PCI interrupt-request signal is asserted.

Gamma level of SSI uses linked-list command blocks. Each command entry is 64 bytes in size and is located in shared system memory. After the command processing completes, the target returns the 16-byte status in a status-list as specified (as a pointer) in the command entry. Multiple initiators can concurrently append command entries in a non-blocking fashion. The initiator's append operations and the target's extract (called producer and consumer in Section 3) can also be performed concurrently. Command-list and status-list wakeups are write4 transactions, directed to registers within the target's and initiator's address space respectively.

There are two types of commands, data-transfer and control command. Each command block specifies an data-transfer command to be performed. Status information of transfer unit is updated when the command completes. Following commands can check these status values and can halt further command processing, jump to an alternate next-command-entry address, or wait. A wait involves sleeping until a wakeup is received, whereupon the previous data-transfer is repeated. Control command is another type of command. One example of a control command is the TEST command, used to perform conditional wait and jump operations.

The command block structure supports two interesting features:

- Scatter array: If the SC bit in the "control" field of a data transfer command block is one,

then a scatter array is located at the basePtr (or sinkPtr) address. The basePtr or sinkPtr address are 8-byte pointers, in a command block, which point to the first data of this command. Each item of the scatter array includes a block pointer and the block size. This feature enables scatter-gather DMA or data chaining capability.

- Dispatch groups: A command list may contain multiple sub-lists, called dispatch groups. The initial entry within a multi-entry command group have its m bit (in the "control" field of a command block) set to 1. For a single-entry command group or the last entry within a multi-entry command group, the m bit is set to 0. A dispatch group identifies a processing unit; multiple dispatch groups can be processed concurrently. A dispatch group is also the entity that is influenced by a "flush" (abort) command. The flush command provides the address of the first command-entry within the dispatch group and causes all command entries within that group to be refetched from memory.

The SSI is a robust interface. The features of command list provide powerful capability to handle the most demanding storage subsystems. The transport specification defines which of the endian-order options are used. Special features such as "spindle synchronization" between multiple devices and "clock synchronization" between devices and system time are also supported.

5. Conclusion

Today, none of the PCI-bridge chips contain a third-party DMA controller. Third-party DMA takes more software set-up. Third-party DMA only supports one function at a time. For example, a SCSI adapter attaches multiple SCSI devices. It is awkward to concurrently transfer data for multiple devices when using third-party DMA because each device’s transfer requires set-up from software and a third-party controller only allows one transfer at a time. This results in lower performance. With today’s VLSI technology, the design of a DMA master does not consume that much circuitry. So the DMA master design is the way to go for PCI bus.

Virtual DMA may cut down the start-up cost of a DMA operation. It is a complicated system design issue and is beyond the scope of this paper.

Three-party DMA offers the advantage of performance. It cuts down the redundant data copies from the bus to system memory and overhead of OS processing. It is particularly appealing to multimedia applications for storage of video input from the network without the aid of a cpu. However, the peer-to-peer protocols between devices are complicated. In addition OS must support coordinated device driver models which are required to handle more than one device within a device driver.

As we discussed earlier, there is no universal DMA master design. IEEE P1285 is a robust standard but optimized to block-oriented storage devices. The command extract and execution may need an intelligent controller in the I/O unit. For latency-oriented ATM-cell design, the much simpler circular queue for buffer management may be more appropriate. Besides, each operating system may have its own DMA program model. The underlying DMA models may need to map to the OS model. The device designer should choose the DMA model based on the cost, performance and device characteristics.

6. Trademarks

PowerPC is a trademark of International Business Machines Corporation.

7. Reference

[1] PCI Local Bus Specification, Revision 2.0, April 30, 1993.

[2] PCI System Design Guide, Revision 1.0, September 8, 1993.

[3] NCR 53C810 PCI-SCSI I/O Processor, Data Manual, 1993.

[4] IEEE P1212.1 CSR Architecture (DMA Framework) Draft 4.1, April, 1993

[5] IEEE Standard P1212-1991 CSR Architecture Specification

[6] IEEE P1285-199X IEEE Standard for Scalable Storage Interface April,19 1994, Draft 0.201.