NATIONAL INSTRUMENTS®
The Software is the Instrument®

Application Note 011

DMA Fundamentals on Various PC Platforms

A. F. Harvey and Data Acquisition Division Staff

Overview

In computer-based data acquisition applications, data incoming or outgoing through computer I/O devices must often be managed at high speeds or in large quantities. The three primary data transfer mechanisms for computer-based data acquisition are polling, interrupts (also known as programmed I/O), and direct memory access (DMA). Polling is a form of foreground data acquisition in which the processor is dedicated to acquiring the incoming data, often by waiting in a loop. The main program calls an acquisition subroutine that waits until the processor collects the required data. With interrupts, the processor is periodically interrupted from executing the main program to store incoming data in a buffer for later retrieval and processing. Interrupts are a form of background acquisition because the main program contains no code that reads data from the input device. Instead, the processor is invisibly stolen periodically from the main program to perform this function. With DMA, a dedicated data transfer device reads incoming data from a device and stores that data in a system memory buffer for later retrieval by the processor. This DMA process occurs transparently from the processor's point of view.

DMA has several advantages over polling and interrupts. DMA is fast because a dedicated piece of hardware transfers data from one computer location to another and only one or two bus read/write cycles are required per piece of data transferred. In addition, DMA is usually required to achieve maximum data transfer speed, and thus is useful for high speed data acquisition devices. DMA also minimizes latency in servicing a data acquisition device because the dedicated hardware responds more quickly than interrupts, and transfer time is short. Minimizing latency reduces the amount of temporary storage (memory) required on an I/O device. DMA also off-loads the processor, which means the processor does not have to execute any instructions to transfer data. Therefore, the processor is not used for handling the data transfer activity and is available for other processing activity. Also, in systems where the processor primarily operates out of its cache, data transfer is actually occurring in parallel, thus increasing overall system utilization.

This application note discusses how DMA is implemented in a typical personal computer (PC) architecture and compares several PC DMA implementations. The software issues involved in using DMA for computer-based data acquisition are also discussed. National Instruments has written software for all the DMA architectures discussed in this application note and has used DMA extensively for computer-based data acquisition applications including streaming data to disk, real-time screen data display, and continuous data acquisition applications.

DMA Controller Tutorial–How DMA Works

DMA has been a built-in feature of PC architecture since the introduction of the original IBM PC. PC-based DMA was used for floppy disk I/O in the original PC and for hard disk I/O in later versions. PC-based DMA technology, along with high-speed bus technology, is driven by data storage, communications, and graphics needs–all of which require the highest rates of data transfer between system memory and I/O devices. Data acquisition applications have the same needs and therefore can take advantage of the technology developed for larger markets. This section introduces DMA controller terminology and explains the basic operation of a PC-based DMA controller along with common modes of operation. Key terminology is italicized.

A DMA controller is a device, usually peripheral to a computer's CPU, that is programmed to perform a sequence of data transfers on behalf of the CPU. A DMA controller can directly access memory and is used to transfer data from one memory location to another, or from an I/O device to memory and vice versa. A DMA controller manages several *DMA channels*, each of which can be programmed to perform a sequence of these *DMA transfers*. Devices, usually I/O peripherals, that acquire data that must be read (or devices that must output data and be written to) signal the DMA controller to perform a DMA transfer by asserting a hardware *DMA request* signal. A DMA request signal for each channel is routed to the DMA controller. This signal is monitored and responded to in much the same way that a processor handles interrupts. When the DMA controller sees a DMA request, the DMA controller responds by performing one or many data transfers from that I/O device into system memory or vice versa. Channels must be enabled by the processor for the DMA controller to respond to DMA requests. The number of transfers performed, transfer modes used, and memory locations accessed depends on how the DMA channel is programmed.

A DMA controller typically shares the system memory and I/O bus with the CPU and has both bus master and slave capability. Figure 1 shows the DMA controller architecture and how the DMA controller interacts with the CPU. In bus master mode, the DMA controller acquires the system bus (address, data, and control lines) from the CPU to perform the DMA transfers. Because the CPU releases the system bus for the duration of the transfer, the process is sometimes referred to as *cycle stealing*. However, this term is no longer appropriate in the context of the new high-performance architectures that are appearing in personal computers. These architectures use cache memory for the CPU, which enables the DMA controller to operate in parallel with the CPU to some extent.

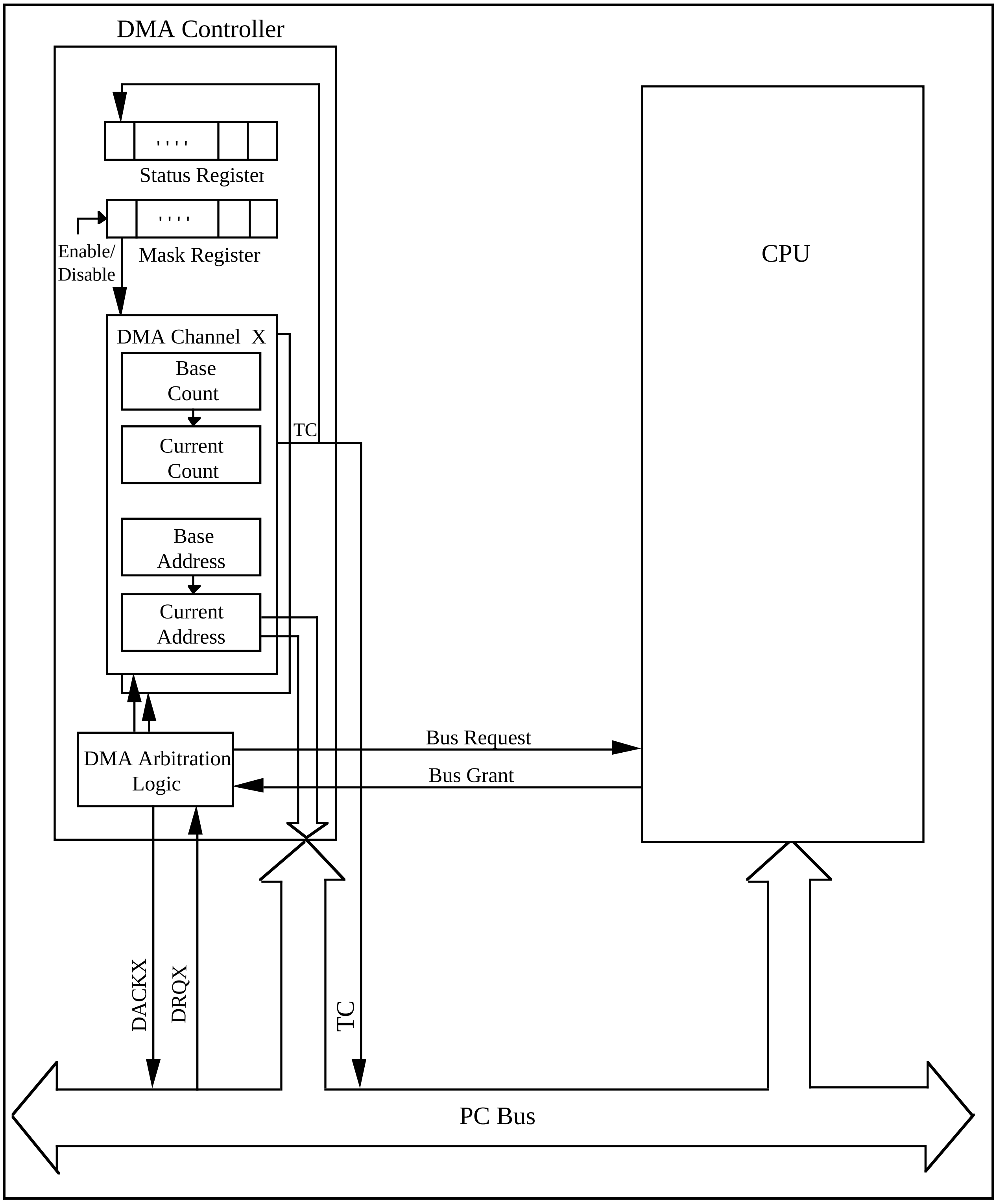


Figure 1. DMA Controller Architecture

In bus slave mode, the DMA controller is accessed by the CPU, which programs the DMA controller's internal registers to set up DMA transfers. The internal registers consist of *source and destination address registers* and *transfer count registers* for each DMA channel, as well as *control and status registers* for initiating, monitoring, and sustaining the operation of the DMA controller.

DMA Transfer Types and Modes

DMA controllers vary as to the type of DMA transfers and the number of DMA channels they support. The two types of DMA transfers are flyby DMA transfers and fetch-and-deposit DMA transfers. The three common transfer modes are single, block, and demand transfer modes. These DMA transfer types and modes are described in the following paragraphs.

The fastest DMA transfer type is referred to as a single-cycle, single-address, or *flyby* transfer. In a flyby DMA transfer, a single bus operation is used to accomplish the transfer, with data read from the source and written to the destination simultaneously. In flyby operation, the device requesting service asserts a DMA request on the appropriate channel request line of the DMA controller. The DMA controller responds by gaining control of the system bus from the CPU and then issuing the pre-programmed memory address. Simultaneously, the DMA controller sends a *DMA acknowledge* signal to the requesting device. This signal alerts the requesting device to drive the data onto the system data bus or to latch the data from the system bus, depending on the direction of the transfer. In other words, a flyby DMA transfer looks like a memory read or write cycle with the DMA controller supplying the address and the I/O device reading or writing the data. Because flyby DMA transfers involve a single memory cycle per data transfer, these transfers are very efficient; however, memory-to-memory transfers are not possible in this mode. Figure 2 shows the flyby DMA transfer signal protocol.

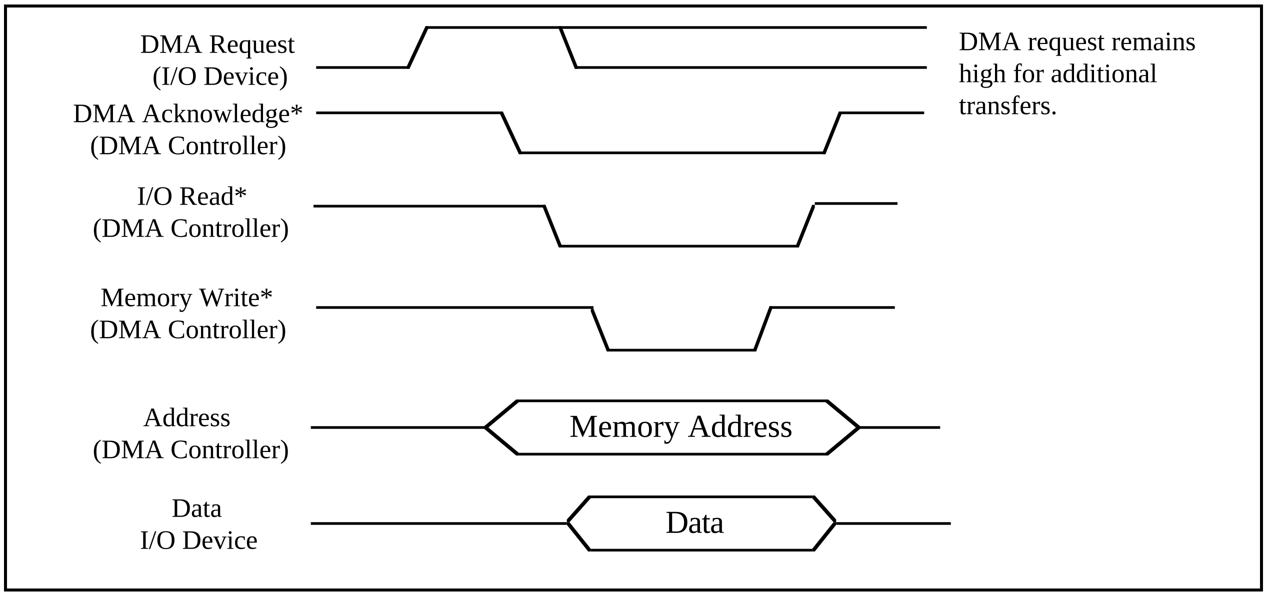

DMA Request (I/O Device)
DMA Acknowledge* (DMA Controller)
I/O Read* (DMA Controller)
Memory Write* (DMA Controller)
Address (DMA Controller)
Data I/O Device
DMA request remains high for additional transfers.
Memory Address
Data

Figure 2. Flyby DMA Transfer

The second type of DMA transfer is referred to as a dual-cycle, dual-address, flow-through, or *fetch-and-deposit* DMA transfer. As these names imply, this type of transfer involves two memory or I/O cycles. The data being transferred is first read from the I/O device or memory into a temporary data register internal to the DMA controller. The data is then written to the memory or I/O device in the next cycle. Figure 3 shows the fetch-and-deposit DMA transfer signal protocol. Although inefficient because the DMA controller performs two cycles and thus retains the system bus longer, this type of transfer is useful for interfacing devices with different data bus sizes. For example, a DMA controller can perform two 16-bit read operations from one location followed by a 32-bit write operation to another location. A DMA controller supporting this type of transfer has two address registers per channel (source address and destination address) and bus-size registers, in addition to the usual transfer count and control registers. Unlike the flyby operation, this type of DMA transfer is suitable for both memory-to-memory and I/O transfers.

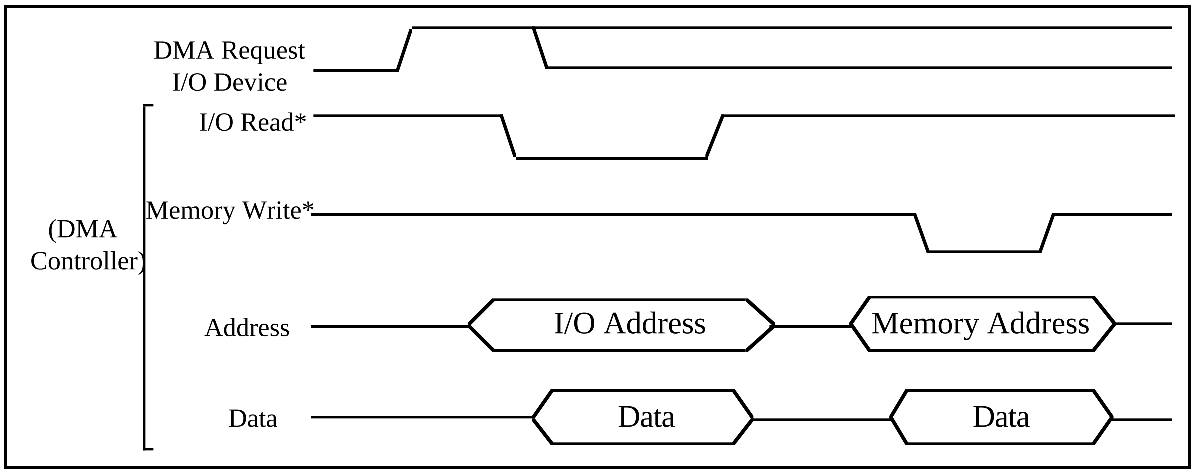


Figure 3. Fetch-and-Deposit DMA Transfer

In addition to DMA transfer types, DMA controllers have one or more DMA transfer modes. *Single*, *block*, and *demand* are the most common transfer modes. Single transfer mode transfers one data value for each DMA request assertion. This mode is the slowest method of transfer because it requires the DMA controller to arbitrate for the system bus with each transfer. This arbitration is not a major problem on a lightly loaded bus, but it can lead to latency problems when multiple devices are using the bus. Block and demand transfer modes increase system throughput by allowing the DMA controller to perform multiple DMA transfers when the DMA controller has gained the bus. For block mode transfers, the DMA controller performs the entire DMA sequence as specified by the transfer count register at the fastest possible rate in response to a single DMA request from the I/O device. For demand mode transfers, the DMA controller performs DMA transfers at the fastest possible rate as long as the I/O device asserts its DMA request. When the I/O device unasserts this DMA request, transfers are held off.

DMA Controller Operation

For each channel, the DMA controller saves the programmed address and count in the base registers and maintains copies of the information in the current address and current count registers, as shown in Figure 1. Each DMA channel is enabled and disabled via a DMA mask register. When DMA is started by writing to the base registers and enabling the DMA channel, the current registers are loaded from the base registers. With each DMA transfer, the value in the current address register is driven onto the address bus, and the current address register is automatically incremented or decremented. The current count register determines the number of transfers remaining and is automatically decremented after each transfer. When the value in the current count register goes from 0 to -1, a *terminal count* (TC) signal is generated, which signifies the completion of the DMA transfer sequence. This termination event is referred to as reaching terminal count.

DMA controllers often generate a hardware TC pulse during the last cycle of a DMA transfer sequence. This signal can be monitored by the I/O devices participating in the DMA transfers.

DMA controllers require reprogramming when a DMA channel reaches TC. Thus, DMA controllers require some CPU time, but far less than is required for the CPU to service device I/O interrupts. When a DMA channel reaches TC, the processor may need to reprogram the controller for additional DMA transfers. Some DMA controllers interrupt the processor whenever a channel terminates. DMA controllers also have mechanisms for automatically reprogramming a DMA channel when the DMA transfer sequence completes. These mechanisms include *autoinitialization* and *buffer chaining*. The autoinitialization feature repeats the DMA transfer sequence by reloading the DMA channel's current registers from the base registers at the end of a DMA sequence and re-enabling the channel. Buffer chaining is useful for transferring blocks of data into noncontiguous buffer areas or for handling double-buffered data acquisition. With buffer chaining, a channel interrupts the CPU and is programmed with the next address and count parameters while DMA transfers are being performed on the current buffer. Some DMA controllers minimize CPU intervention further by having a chain address register that points to a chain control table in memory. The DMA controller then loads its own channel parameters from memory. Generally, the more sophisticated the DMA controller, the less servicing the CPU has to perform.

A DMA controller has one or more status registers that are read by the CPU to determine the state of each DMA channel. The status register typically indicates whether a DMA request is asserted on a channel and whether a channel has reached TC. Reading the status register often clears the terminal count information in the register, which leads to problems when multiple programs are trying to use different DMA channels (see *Managing the DMA Controller* later in this application note for more information).

DMA on the PC

Original IBM PC and PC/XT DMA Implementation

All DMA controllers discussed in this application note are related to the Intel 8237A DMA controller used in the original IBM PC. This DMA controller has four channels with 16-bit memory-address and byte-count registers for each channel. The 16-bit count register limits the length of a preprogrammed DMA transfer sequence to 64 kilobytes. The PC added an 8-bit *page register* external to the 8237A to extend the 16-bit address to the 24-bit PC memory address. With this extension, preprogrammed DMA transfer sequences cannot cross 64 kilobyte *page boundaries*, because the page register does not increment when the address register reaches 0FFFFH. The 8237A implements single, block, and demand DMA transfer modes.

The PC uses two of the DMA channels, one channel to transfer data from floppy disk to memory and vice versa (Channel 2) and the other channel to perform memory refresh cycles (Channel 0). The remaining channels are made available to I/O boards plugged into the PC bus, such as communications peripherals. The PC I/O Channel bus has a TC signal and DMA request signals DRQ*X* and DMA acknowledge signals DACK*X**, where *X* is the DMA channel number. The 8237A DMA controller of the PC performs 8-bit, flyby transfers because the PC has an 8-bit bus. When an I/O device needs a DMA transfer, the device asserts DRQ*X*. Later the DMA controller places the appropriate memory address on the bus and asserts DACK*X** one or more times. The assertion signals the I/O device to read or write data as specified by the read and write control signals. If the transfer is the last of the sequence, TC is asserted along with the final DACK*X**. In addition to the normal I/O-to-memory flyby transfer mode, the 8237A has a special memory-to-memory transfer mode that uses two DMA channels to transfer data from one memory buffer to another.

In the PC, the 8237A operates at 4.77 MHz and can transfer data up to 900 kbytes/sec in demand transfer mode.

ISA (PC AT) Extensions to DMA Implementation

The 16-bit Industry Standard Architecture (ISA) or PC AT computers increased the number of DMA channels to seven. Two Intel 8237A-5 DMA controllers are cascaded together, and each chip has four channels. DMA Controller 1 has Channels 0 through 3 and implements the original 8-bit, flyby DMA transfers of the PC between 8-bit I/O adapters and 8-bit or 16-bit system memory. DMA Controller 2 has Channels 4 through 7.

Channel 4 is used to cascade the operation of the two DMA controllers and is therefore not available for general use. Channels 5, 6, and 7 implement 16-bit, flyby DMA transfers between 16-bit I/O adapters and 16-bit system memory. The count registers in DMA Controller 2 act as 16-bit, word-count registers, and addresses A1 through A16 are written into the address registers. The page registers hold the upper seven bits of the 24-bit address, and address A0 is assumed to be zero. For 16-bit transfers then, the maximum transfer length is extended to 128 kilobytes and DMA transfers cannot be performed across a 128 kilobyte page boundary.

With an 8-MHz ISA bus, the 8237A operates at 4 MHz (it has a 5-MHz clock limit) which translates to an 800 kbyte/sec demand mode transfer rate for 8-bit transfers and 1.6 Mbyte/sec rate for 16-bit transfers.

Both for the original PC and for ISA computers, top DMA transfer rates are limited by memory refresh requirements. One memory refresh cycle must be executed every 15 µsec. A device designed to use demand DMA transfers must deliberately drop DRQ and relinquish the bus every 15 µsec for memory refresh to occur. In an ISA system, block-mode transfers are not very practical because the 8237A does not relinquish the bus until the entire DMA sequence is complete, and only 24 ISA transfers can be performed in 15 µsec. Because sustained DMA transfers are difficult to implement and multiple devices use the DMA controller, I/O devices should have onboard buffering, particularly in critical timing applications such as counter-controlled pattern generation where data must be output at a fixed rate.

Micro Channel DMA Implementation

The Micro Channel DMA controller design evolved from the ISA DMA controller, but several significant changes were made. The Micro Channel has eight DMA channels capable of transferring either 8-bit or 16-bit data. The Micro Channel does not use dedicated DRQ or DACK* signals. Instead, the Micro Channel uses the bus master arbitration signals for DMA requests with the same protocol that bus masters use for requesting the bus. However, instead of the I/O device controlling the bus, the DMA controller recognizes the arbitration level, becomes bus master, and performs a fetch-and-deposit DMA transfer. This transfer is a memory cycle followed by an I/O cycle, or vice versa, and temporarily stores the data between cycles. The I/O device can use the arbitration signals as DMA acknowledge signals during the I/O cycle, which makes programming each channel similar to the ISA programming. The I/O device can also be addressed by the DMA controller during the I/O cycle.

The DMA controller has a set of registers that are compatible with the ISA DMA controller plus extended registers and extended programming modes. In extended programming, the DMA controller has full 24-bit memory-address registers (completely addressing 16 megabytes) and
16-bit I/O-address registers. Therefore, the controller does not have the ISA 64 kilobyte or 128 kilobyte transfer and page-addressing limitations. The DMA controller supports both single and demand transfer modes.

The Micro Channel DMA controller has some limitations not found in the ISA implementation. Autoinitialize mode is not supported[1], and a channel is automatically disabled when reaching TC. As a result, the DMA controller must be reprogrammed whenever a channel terminates, regardless of whether the channel restarts with the same address and count. In addition, the I/O device must recognize the TC occurrence and not assert more DMA requests until the channel is reprogrammed and re-enabled; otherwise, a system exception occurs after a time-out period. In ISA implementation, the DMA controller either autoinitializes and continues data transfer or ignores the DMA request until the controller is reprogrammed. Both limitations make designing DMA software more difficult for a Micro Channel I/O device than for an ISA device.

The Micro Channel DMA controller can achieve a top transfer rate of close to 5 Mbytes/sec. Because Micro Channel DMA arbitration occurs in series with data transfers, arbitration activity slightly reduces overall data transfer bus bandwidth. As in ISA implementation, one memory refresh cycle is executed every 15 µsec, which also slightly reduces the DMA transfer rates. In addition, buffering is required for critically timed DMA operations.

---

[1] Actually, there is a claim that autoinitialize mode *is* supported. However, this claim is not documented, and National Instruments was not able to obtain confirmation from IBM technical support.

EISA Extensions to ISA DMA Implementation

Instead of being a peripheral integrated circuit (IC) like the 8237A, the DMA controller for the Extended Industry Standard Architecture (EISA) bus is integrated into a multifunction IC. As a multifunction IC, the controller is part of the EISA chip set residing on the system board of an EISA computer. The DMA portion of the IC is a superset of the DMA controllers used in PC AT (ISA) computers. The EISA DMA controller is software compatible with and has all the features of the ISA 8237A DMA controllers, except the EISA DMA controller does not support the special memory-to-memory transfer mode. Each DMA channel implements the single, block, and demand DMA transfer modes, and all DMA transfers are flyby DMA transfers. There are some anomalies associated with the programming sequence (for software compatibility), and programs written for EISA computers use a different programming sequence than ISA programs.

The EISA DMA controller is considerably more powerful than the ISA DMA controller. The EISA DMA controller has seven channels, each of which supports full 32-bit addressing (4 gigabytes), 32-bit data transfers, and 24-bit byte/word counters. As in Micro Channel implementation, the full system addressing eliminates the need for a page register and subsequent DMA channel reprogramming to cross page boundaries. The page register is still included, however, for ISA software compatibility. A 24-bit byte-count register means that a channel can be programmed to transfer up to 16 Mbytes of data. Each channel can perform 8-bit, 16-bit, or 32-bit data transfers. The EISA DMA controller also supports dynamic bus sizing, where DMA transfers are possible between devices of different bus data widths. With the EISA DMA controller an I/O device can drive the TC line to terminate the DMA transfer sequence early and software requests can control DMA transfers.

The EISA DMA controller offers a significant increase in transfer speeds when compared to the ISA solution. Many of the new transfer speeds can be used by existing ISA I/O boards. There are four different DMA transfer timing modes: ISA-compatible, Type A, Type B, and Type C (Burst). The various achievable maximum transfer rates are shown in Table 1.

Table 1. DMA Timing Modes and Transfer Rates

DMA Cycle Type	Transfer Rate (Mbytes/sec)	Compatibility
ISA-Compatible:		
8-bit	1.0	All ISA computers
16-bit	2.0	All ISA computers
Type A:		
8-bit	1.3	Most ISA computers
16-bit	2.6	Most ISA computers
32-bit	5.3	EISA computers only
Type B:		
8-bit	2.0	Some ISA computers
16-bit	4.0	Some ISA computers
32-bit	8.0	EISA computers only
Type C (Burst):		
8-bit	8.2	EISA computers only
16-bit	16.5	EISA computers only
32-bit	33.0	EISA computers only

In addition to the standard transfer-the-buffer and terminate-or-autoinitialize scenario, the EISA DMA controller supports some useful termination modes, including buffer chaining mode and ring buffer mode. In buffer chaining mode, the DMA controller interrupts the processor to request a new buffer address and count each time the address registers are reloaded from the base register set. With buffer chaining mode, multiple buffers can be filled/emptied with minimum CPU intervention and no delay between buffers. Ring buffer mode is an enhancement to autoinitialization mode. A stop register can be programmed with an address that is compared against the current address register. If a match is detected, the DMA controller ceases transfers until the stop register is reprogrammed. Ring buffer mode can be used with autoinitialization to manage a circular buffer and prevent data from being overwritten accidentally.

DMA on the Macintosh II Family

Unlike the ISA, Micro Channel, and EISA buses, which are designed to handle slave plug-in boards that do not initiate bus reads or writes, the NuBus (featured in Macintosh II Series computers) is a multiprocessor bus with many provisions for master mode capabilities and limited support for slave boards. The NuBus has only a Non-Master Request (NMRQ) signal, which, upon assertion, indicates that a particular NuBus slave board needs some service. The NMRQ signal is used in the Macintosh II as a CPU interrupt line. There are no NuBus DMA signals.

The lack of DMA support on the NuBus prompted National Instruments to invent the Real-Time System Integration (RTSI®) bus–a secondary bus that runs across the top of the National Instruments plug-in expansion boards and carries DMA requests, interrupt requests, serial communications, and system timing signals that are vital to data acquisition applications. Using the RTSI bus along with a NuBus master DMA server board creates a system whereby multiple slave boards are serviced by a single NuBus master. Simple I/O cards can therefore be used with all the advantages of a typical EISA/ISA/Micro Channel DMA environment.

Figure 4. RTSI Bus and NuBus Boards

The RTSI bus DMA signals are similar to the EISA/ISA implementation of DRQ*X* signals for each channel and a bidirectional TC signal. The DMA acknowledge signals are encoded onto three lines instead of the individual DACK*X** signals available on EISA.

National Instruments has developed two DMA server boards for the NuBus, the NB-DMA-8-G and the NB-DMA2800. These boards are based on the Intel 82380 DMA controller (DMAC) chip, which closely resembles the EISA DMA controller. Both DMA boards support up to seven DMA requests from the RTSI bus and an eighth DMA request used by the onboard IEEE-488 interface. All of the DMA channels can perform 8-bit, 16-bit, or 32-bit NuBus transactions over the full 32-bit, 4 gigabyte address space of the NuBus.

The NB-DMA-8-G and NB-DMA2800 boards implement single, demand, and block DMA transfer modes. The original NB-DMA-8-G implements fetch-and-deposit type transfers using two single NuBus transactions with temporary data storage between the two transactions. The NB-DMA2800 adds flyby transfer capability and supports NuBus single and block transactions. NuBus block transactions are analogous to EISA Type C (Burst) DMA transfers, which transfer up to sixteen 32-bit words in a single bus transaction.

In flyby mode, the NB-DMA2800 board initiates the read or write cycle by arbitrating for the NuBus to supply the memory address and by generating the RTSI bus DMA acknowledge signals as the NuBus transaction is started. The slave board requesting DMA monitors the DMA acknowledge signals on the RTSI bus and supplies or latches the data as required by NuBus transaction timing. The NB-DMA2800 board and the I/O slave board act as a pair and appear to memory as a single NuBus master. In a NuBus block mode flyby transfer, the DMA requestor supplies or reads sixteen 32-bit words, monitoring the NuBus block mode acknowledge lines to determine whether new data is ready or when to supply new data. Because sixteen 32-bit words are transferred in one NuBus transaction, NuBus block-transaction DMA can achieve continual transfer speeds of over 29 Mbytes/sec.

The NB-DMA-8-G and NB-DMA2800 boards have buffer chaining and autoinitialization capability as well as TC interrupt capability, so the DMA board can interrupt the Macintosh II CPU when TC is reached on any DMA channel. These DMA boards also have some DMA request features useful for data acquisition applications that are not found on the more conventional PC DMA controllers. The NB-DMA-8-G and NB-DMA2800 boards have counter/timers that can drive DMA request signals at periodic rates and circuitry for conditioning these pulsed signals or other pulsed DMA requests for proper DMA request timing. The boards can also take DMA requests on Channel 0 and parallel these requests to DMA requests on a programmable number of successive channels. Therefore, all requests are asserted simultaneously, or each request is asserted in round robin fashion. These features are useful for synchronized operation on multiple input and output channels.

Overall Performance Comparison

Tables 2 and 3 compare the overall DMA performance characteristics and features on the various PC platforms.

Table 2. DMA Controller Performance Comparisons

Platform	**Number of DMA Channels**	**Data Size**	**Maximum Buffer Size**	**DMA Bus Operation**	**DMA Transfer Rates (Mbytes/sec)**		
					8-bit	**16-bit**	**32-bit**
PC/XT	four	8-bit	64 kilobytes	flyby	0.9	–	–
PC AT (ISA)	seven (four 8-bit, three 16-bit)	8-bit, 16-bit	64 kilobytes or words	flyby	0.8	1.6	–
EISA	seven	8-bit, 16-bit, 32-bit	16 megabytes	flyby non-burst	1-2	2-4	5-8
EISA	seven	8-bit, 16-bit, 32-bit	16 megabytes	flyby burst	8.2	16.5	33.0
Micro Channel	eight	8-bit, 16-bit	16 megabytes	fetch-and-deposit	2.5	5.0	–
Macintosh II NB-DMA-8-G	eight	8-bit, 16-bit, 32-bit	16 megabytes	fetch-and-deposit to Macintosh II memory	1.4	2.0	2.5
Macintosh II NB-DMA2800	eight	8-bit, 16-bit, 32-bit	16 megabytes	fetch-and-deposit to Macintosh II memory	1.2	1.7	2.2
Macintosh II NB-DMA2800	eight	32-bit	16 megabytes	flyby to Macintosh II memory	–	–	3.1
Macintosh II NB-DMA2800	eight	32-bit	16 megabytes	flyby to fast NuBus memory	–	–	8.0
Macintosh II NB-DMA2800	eight	32-bit	16 megabytes	flyby to NuBus block-mode memory	–	–	29.1

Table 3. DMA Controller Advanced Buffer Management Features Comparison

Platform	Advanced Buffer Features
PC/XT	Autoinitialization
PC AT (ISA)	Autoinitialization
EISA	Autoinitialization, buffer chaining, ring buffers
Micro Channel	Not applicable
Macintosh II (NB-DMA-8-G)	Autoinitialization, buffer chaining
Macintosh II (NB-DMA2800)	Autoinitialization, buffer chaining

Using DMA controllers requires careful software design. The next section discusses the software issues involved in using DMA controllers.

DMA Software Issues

There are two major software-related concerns when using DMA controllers. First, the many channels in a DMA controller are typically used by different and unrelated applications. Because a DMA controller is a system resource that is shared, it must be properly managed or unrelated applications adversely affect each other. The DMA buffers must also be managed as data is collected or generated. Buffer management is highly application dependent. Clever design of the DMA I/O device aids the software in maintaining accurate status information and in controlling the transfer of data.

Managing the DMA Controller

PC, ISA, EISA, and Micro Channel computers, based on the 8237A and derivatives, make it very easy for a program to adversely affect other programs if the proper programming steps are not taken because many of the DMA controller channel registers affect multiple channels. In addition, these computers require programs to directly access the DMA controller to use DMA. In other words, there is no operating system support for DMA. Because of this lack of protection, programs using the DMA controller must observe certain rules to ensure the integrity of DMA programming and status information.

For example, the 8237A DMA controller 16-bit registers are programmed through an 8-bit port, with the DMA controller maintaining a single internal byte pointer that determines which of two internal byte positions receives the data from each 8-bit read or write operation. Clearing the byte pointer before a 16-bit access sequence is necessary to guarantee that the register is properly loaded. Furthermore, to prevent an interrupt routine from changing the internal byte pointer, system interrupts should be disabled when clearing the byte pointer and writing or reading the register.

Another problem involves successfully determining DMA channel completion. The 8237A-type status register can be read to determine whether any of the channels have reached TC. Unfortunately, reading the status register clears the TC status bits for all the DMA channels in the register. A program cannot depend on the contents of the status register and must find other means of determining DMA status on the channel being used. These other means include reading and monitoring the 16-bit address and count registers for the channel. The channel must be temporarily disabled for an accurate reading because the registers are changing and two

8-bit reads per register must be performed. Therefore, the DMA process on a particular channel is stopped when status information is obtained, which at high speeds can introduce sufficient latency to lose incoming data. This penalty must be noted by any program that is using the DMA controller near top system speeds.

A better solution for determining DMA completion status is to have the DMA slave I/O device monitor DMA status, for example, using an onboard counter or by generating TC interrupts. In the EISA and Macintosh II DMA board environments, the DMA controller generates a system interrupt on TC that jumps to the appropriate interrupt handling routine for the channel. In systems without this capability, the DMA slave I/O board can monitor the TC pulse generated by the DMA controller and generate an interrupt when a TC pulse occurs with respect to its DMA channel. With TC interrupts, the program is quickly notified when DMA is complete, which is important for the many DMA maintenance tasks with strict timing requirements–for example, where the DMA channel must be reprogrammed and restarted before the slave device overflows its internal buffer. TC interrupts can be used to overcome some DMA controller limitations such as limited byte-transfer length (PC, ISA) and lack of autoinitialization (Micro Channel) or buffer chaining (PC, ISA, Micro Channel). TC software simply reprograms the channel to continue as if the DMA controller has the built-in capability.

Using a DMA Driver to Manage the DMA Controller

In the Macintosh II DMA server board implementation, National Instruments wrote a special driver for the DMA server board that manages the DMA controller on behalf of all programs using DMA. This driver has system calls for channel allocation and deallocation, programming, starting and stopping, and status and TC interrupt handling. By going through the driver, programs using the DMA controller do not have to handle unfriendly DMA-related critical sections. In addition, these DMA system calls have a software-compatible interface for the NB-DMA-8-G, the NB-DMA2800, and any future DMA boards. If this layer of system software had existed in the other PC implementations, the newer DMA controllers may not have been burdened with backward-compatible registers.

DMA Buffer Management for Data Acquisition

The following sections describe DMA buffer management techniques implemented by the National Instruments LabDriver® data acquisition driver software for DOS and Macintosh environments. These techniques take advantage of the DMA controller's built-in features, when available, and compensate for the controller's limitations. LabDriver has been implemented for all the computer DMA environments discussed in this application note. These techniques result from years of development and experience with demanding data acquisition applications, and, in some cases, have required that special hardware features be added to the I/O DMA device.

Single-Buffered DMA Data Management

The simplest type of data buffer management is single-buffered data management. In single-buffered data management, a fixed amount of data is transferred to or from a sequential block of memory. Single-buffered data acquisition or generation is useful with an externally triggered I/O device. The device waits until the trigger is received before asserting DMA requests for data transfer.

DMA controllers automatically perform this simple function, which is initiated by the CPU and then left to complete on its own. If the program wants to access data as it is transferred into the buffer, the DMA controller must be temporarily disabled to read the address or count registers. The program can then use the register information to access the buffer safely. However, this starting and stopping undermines the high-speed potential of DMA and should only be used when DMA is slow enough that periodically stopping the controller does not cause data loss. Alternatively, the program must wait for a TC indication before accessing the data in the buffer, which means that the program can process other tasks, periodically checking on DMA status. The National Instruments LabDriver software has function calls that return DMA transfer completion status with an option to read the current DMA count. With this option, the user can choose whether or not to stop the DMA controller while it is running.

Although the 8237A is designed to automatically perform single-buffered data acquisition, the 64 kilobyte transfer limit and the 64 kilobyte/128 kilobyte DMA page boundary in PC AT environments forces the data acquisition to be broken up into multiple DMA transfer sequences with the DMA controller reprogrammed between sequences. In these PC AT environments, LabDriver uses TC interrupts generated by the data acquisition boards to automatically reprogram the DMA controller and transfer the next section of the buffer, thus hiding the boundary limits from the user. The Micro Channel, EISA, and Macintosh II DMA board DMA controllers have sufficiently large address and count registers that such piecewise DMA transfer programming is unnecessary.

Acquiring Pretrigger Data

Some applications require pretrigger data (data acquired before the trigger occurs). The National Instruments data acquisition boards are designed to indefinitely acquire samples before a trigger and to stop acquisition after acquiring a programmable number of posttrigger samples (samples acquired after the trigger occurs). With this technique, the driver software can implement pretrigger data acquisition by taking advantage of the autoinitialization feature built into most of the DMA controllers. The DMA controller transfers data into the buffer cyclically so that when the end of the buffer is reached, the DMA controller starts storing data at the beginning of the buffer again. When the I/O device stops, the drivers read the DMA controller to determine where in the buffer the last point was written. The data in the buffer is then rearranged so that the first part of the buffer contains pretrigger data and the second part contains the posttrigger data. Because this cyclic data buffer scheme requires autoinitialization, the driver handles cases where autoinitialization is not built into the DMA controller, as in the Micro Channel case. Additionally, if the buffer crosses a page boundary, or the length exceeds the transfer-count register size, the driver uses the piecewise DMA programming techniques mentioned previously.

Double-Buffered DMA Data Management

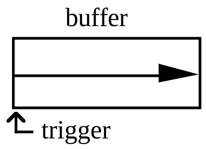
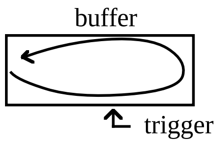
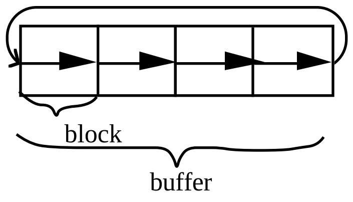
Multiple buffers extend the use of a single buffer, so that while the DMA controller fills one buffer with data, an application program can read data from another buffer without regard to which memory location the DMA controller is accessing. This type of buffer management is known as double buffering because the data must be copied from the DMA buffer into a secondary buffer before processing. A circular double buffer is necessary when the available memory is less than the amount of data to be acquired, or when the most recent data must be retrieved at any time as data is continuously acquired. Double buffering is most advantageous when the CPU and DMA controller are operating at roughly similar rates so that neither is waiting on the other to complete accessing the buffer. Theoretically, only two buffers are needed; in practice, however, multiple buffers are used to help smooth over variations in the response of the CPU and the bus latency experienced by DMA controller arbitration. One of the most popular uses of double buffering is streaming data to disk files during acquisition. Double buffering is also useful when designing an interactive application that updates a real-time display, such as a scaled or transformed picture of the acquired data.

LabDriver implements a double-buffered scheme by allocating a contiguous memory DMA buffer, which is then subdivided into blocks. The buffer chaining feature built into the more sophisticated DMA controllers is used for managing the buffer. With chaining interrupts the driver can seamlessly reprogram the DMA controller for the next block and monitor when each block has completed. In this manner, the driver learns which blocks are safe to access and whether acquired data has been overwritten. An overwrite occurs if the application does not retrieve data often enough to keep up with the acquisition. To avoid overwrites, a larger block size and more buffers can be used, but high acquisition rates can cause the application to fail to keep up. However, an application may only want to monitor the incoming data and not save every point. LabDriver tracks overwrite errors and notifies the application when an overwrite occurs. The application must be notified whenever data is lost.

The EISA implementation of LabDriver uses the ring buffer mode used in the EISA DMA controller to implement double buffering within a single circular buffer. The stop register prevents the DMA controller from overwriting data when the buffer is accessed by the driver to copy acquired data into the application buffer. Programming the stop register with the first address from which the driver copies data causes the DMA controller to wait when reaching that address. When the data is copied, the driver clears the address and the DMA controller continues. As long as the application retrieves data quickly enough, and the I/O device has some onboard buffering, data is not lost by stopping the controller.

Multiple-Frame Buffer Management

In addition to double buffering, LabDriver uses a buffer management technique so that data from multiple triggers can be acquired into a contiguous buffer. Each captured data block is known as a frame. A frame can be thought of as one snapshot of an oscilloscope display. Frames can be used to acquire multiple copies of a repetitive signal, which is useful for signal averaging or other digital signal processing (DSP) operations as well as for continuously displaying the most recent frame. The National Instruments NB-A2000 (for Macintosh II NuBus) and EISA-A2000 1-MHz A/D boards can capture multiple frames in this manner. During posttrigger operation, buffer chaining is used to automatically fill the contiguous buffer. A buffer chaining or TC interrupt notifies the driver when each frame has been acquired. When pretrigger data is also collected, each frame buffer is filled circularly using autoinitialization, as previously described. However, buffer chaining and autoinitialization cannot be used together. The NB-A2000 and EISA-A2000 interrupt the driver when the current frame has been acquired, and the DMA controller is left in autoinitialize mode but is reprogrammed with the base address of the next buffer. The NB-A2000 and EISA-A2000 are designed so that the driver must re-enable the board to respond to the next trigger. The driver can then ensure that the DMA controller is completely programmed before the next frame is acquired. Figure 5 is a summary of the different types of DMA buffer management discussed.

Type	DMA Controller	Driver	Application
Single-Buffered (Posttrigger)	 DMAC fills the buffer once after the trigger is received.	• Starts the DMA Controller (DMAC). • Reports acquisition status.	• Starts the acquisition. • Waits for completion to process the buffer.
Single-Buffered (Pretrigger)	 DMAC uses autoinitialization to circularly fill the buffer until the trigger is received.	• Starts the DMAC. • Reports acquisition status. • Unwraps the buffer when DMA is complete.	• Starts the acquisition. • Waits for completion to process the buffer.
Double-Buffered (Posttrigger Only)	 DMAC uses chaining to fill each block until the total number of blocks is acquired or stopped by the application.	• Starts the DMAC. • Copies requested blocks into the application buffer as they become available. • Reports acquisition status.	• Starts the acquisition. • Fetches data in blocks; can retrieve oldest or most recent data blocks. • Stops the acquisition when sufficient data blocks are collected.

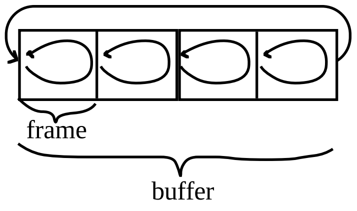
Frame-Oriented (Pretrigger and Posttrigger)	 DMAC handles each trigger and frame as in the single-buffer posttrigger or pretrigger case. Stops when total number of frames is acquired or stopped by application.	• Restarts the DMAC as each frame is acquired. • Copies requested frames into the application buffer as they become available–unwrapped if necessary. • Reports the acquisition status.	• Starts the acquisition. • Fetches *n* frames of data; can retrieve oldest or latest frames. • Stops the acquisition when sufficient frames are collected.

Figure 5. DMA Buffer Management Summary

National Instruments has successfully used buffer chaining and autoinitialization as part of the buffer management strategy in our data acquisition driver software. With the flexibility of the more advanced features of DMA controllers, there is a great deal of flexibility in designing applications.

Other DMA Software Issues

DMA Overhead

A certain amount of overhead is involved in programming a DMA controller. This overhead must be amortized over the total number of data values transferred. If the amount of data to be transferred is small, the CPU may be able to perform the transfer more quickly. This break-even point also depends on the amount of buffering present in the DMA slave device. For example, if the slave buffer has *n* locations, and *n* or fewer data values are to be transferred, it may be best for the slave buffer to fill and then for the slave to interrupt the CPU to read the data at top speed after all data values are acquired. When using DMA, the overhead should be calculated and the program designed so the CPU performs the transfers when the data amount is small, as long as the CPU does not have to wait between data transfers. In applications such as timed data acquisition, an overhead penalty is not usually incurred because the acquisition is often initiated by a trigger. The DMA controller is then set up before the trigger is received. Also, the overhead in setting up the DMA controller is offset if the DMA controller can transfer data faster than the CPU, and the application requires the speed, or if the application must perform some other activity parallel to the data transfer.

Managing DMA Buffers in a Virtual Memory System

DMA buffer management increases in complexity when the CPU uses a virtual memory management system or otherwise dynamically moves information in memory. The Macintosh II system manages the application heap so that data in memory can be moved at any time, which requires that applications request unrelocatable memory when allocating memory from the operating system for use with DMA. The latest Macintosh II computers have also implemented enough pieces of a virtual memory system whereby the addresses used by the CPU (logical addresses) are different than those used by the NuBus DMA controller (physical addresses) because devices on the NuBus are external to the virtual memory system. When programming the DMA board DMA controller, LabDriver translates the CPU logical addresses to physical addresses and locks memory pages so that memory pages cannot be moved or swapped out to disk during the DMA operation. In addition, the buffer chaining can be used to handle physically discontiguous data buffers that appear contiguous in terms of the logical addresses.

Conclusions

National Instruments uses DMA hardware and software technology to achieve high throughput rates as well as to increase system utilization. These achievements are accomplished by using a background mechanism of data transfer that minimizes CPU usage. Data acquisition users are highly aware of the advantages of background data acquisition, and DMA solutions have been very popular. LabDriver double-buffered data acquisition features are popular among users for experiments involving large amounts of data that are reduced as the data is acquired. Users appreciate the flexibility of buffering up the data and processing it in blocks, rather than having to acquire and stop for processing, or having to read individual points.

Although DMA increases system utilization and achieves high throughput, remember that interrupts do offer some advantages over DMA. A DMA controller can only transfer data from one location to another. However, it is often desirable to process data on the fly–for example, searching for a trigger condition in the data before saving it into memory, or implementing a high-speed process control loop. Interrupt-driven acquisition is required in these applications because computation on individual points must be performed. In addition, there are usually fewer DMA channels available than interrupt channels. LabDriver uses DMA or interrupts interchangeably so that channel availability is transparent to the application program. Another advantage is that interrupt routines can transfer data from a set of locations into different data buffers, whereas one DMA channel is required for each location and buffer. The flexibility of interrupts must be weighed against higher CPU utilization, improved service latency, and, in some cases, higher throughput when using DMA.

DMA can still be used when incoming data is slow, although the improvements in CPU utilization and latency are not significant at slow speeds. In fact, at slow speeds waiting for a block of a buffer to fill is not advantageous. In these cases, the DMA controller must be read to determine status, which may interfere with a higher speed channel. In determining whether to use DMA for slower channels, overall system DMA utilization must be understood.

In the last few years there has been a significant increase in the number of bus master boards appearing in the PC market (Micro Channel, EISA, NuBus, and some PC AT), particularly in the communications area. These boards have local processors and directly access host CPU memory when data must be transferred. For performance, the advantages of bus master boards are similar to the advantages of DMA with the added advantage that no DMA channels are used in the later PC architectures. With a local processor, processing on incoming data can be performed on the fly, and complex buffers can be managed. Although a bus master board is the highest performance solution, the hardware cost is higher and functional board area is reduced because the board must carry a processor, memory, and a bus master interface. In addition, the software is more complex and the board software is usually application specific. With built-in PC DMA, an I/O device can achieve high throughput and/or increased system utilization with a relatively low-cost design.