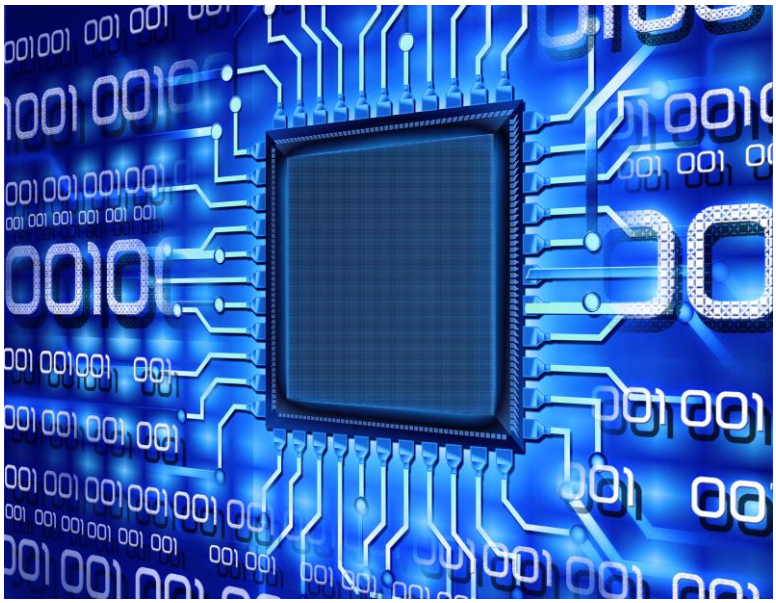
Basic Principles of Direct Memory Access (DMA)



Red Rapids

797 North Grove Rd, Suite 101
Richardson, TX 75081
Phone: (972) 671-9570
www.redrapids.com

Background

It would be impossible to sustain high data rate transfers through computer peripherals if the central processing unit (CPU) had to manage these transactions. It is far more efficient to allow direct memory access (DMA) so that data transfers are orchestrated by the attached device, independent of the CPU or any software intervention. This can be a difficult concept to grasp. Software applications typically have no exposure to physical memory addresses, those hardware details are abstracted to a virtual address space. Fortunately, the driver supplied with the device acts as a bridge between the physical and virtual domains.

It helps to understand some basic computer architecture concepts to appreciate the power and limitations of DMA transfers. Figure 1 illustrates a very basic computer architecture consisting of a CPU, main memory, and bus connected devices. The challenge is to move data between the devices and main memory without passing through the CPU. The memory-mapped address space of the system is flat, eliminating any potential ambiguity about source or destination locations.

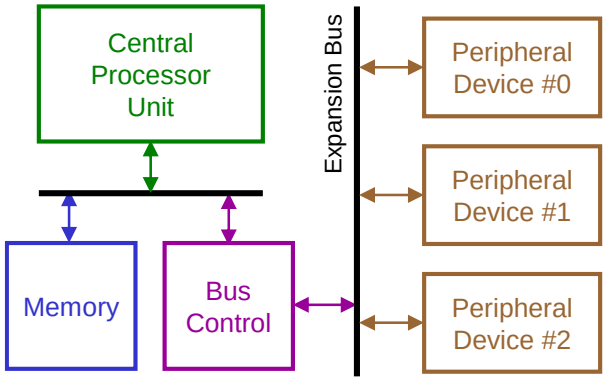


Figure 1 Basic Computer Architecture

To initiate a DMA transfer, the device must become a bus master. In other words, the device must make a hardware request to assume control of the bus. When control of the bus is granted, the device needs only a source or destination address to complete a read or write transfer. It is important to note that the source or destination does not have to be in main memory. It can be any address in the system memory map, making it possible to perform DMA transfers between devices.

DMA transfers typically involve more than just a single word of data. It is much more efficient to transfer a large block of data in a single burst to mitigate the overhead incurred each time the bus is requested. A burst transfer always begins with a single source or destination address. The length of the transfer is accommodated by simply accessing subsequent addresses in sequence from the original source or destination address. This is a very important point since it implies that the device must have access to a continuous segment of addresses to complete a burst.

Device Driver

The device driver plays a key role in the DMA process. Before a DMA transaction to main memory can begin, a protected segment of address space must be allocated by the operating system. Imagine what would happen if a device began writing to unprotected memory space. The operating system would assume that those addresses are available for normal operation and the presence of foreign data would be interpreted as a hardware fault.

Application software must call the driver to allocate the desired DMA page size in main memory. The driver responds with two addresses for each page of memory allocated. The first is a physical address required by the device to perform transfers over the bus. The second is a virtual address that can be used by software to access the contents of the buffer from the application.

Keep in mind that the entire block must occupy contiguous memory. It is possible that the operating system may not even have a large enough block of contiguous memory available to accommodate the request. Some operating systems even place a limit on the maximum contiguous address space that can be allocated.

Scatter Gather

Many devices employ a scatter gather technique to link a series of DMA pages in host memory. The complete DMA buffer is scattered across several pages and then gathered by the application code through indexing. This technique usually employs a DMA descriptor that is also stored in host memory to convey the size of each DMA page and the pointers necessary to index the pages as a single circular buffer as shown in Figure 2. The advantage of this approach is that there is no limit to the number of DMA pages that can be created so the size of each page can be kept small. This reduces the chance that the operating system will fail to find the requested contiguous segment.

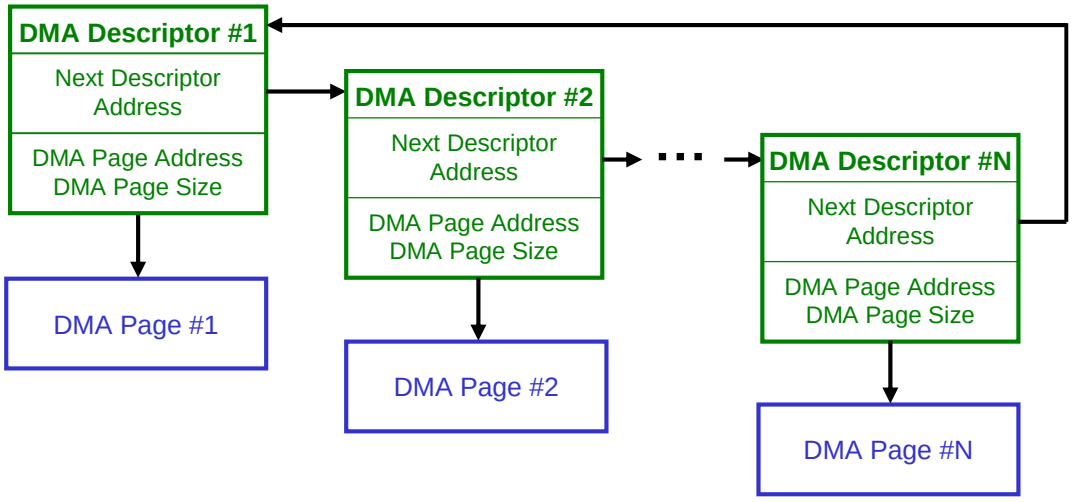


Figure 2 Scatter Gather DMA Indexing

The device must access each page descriptor to obtain the source or destination of the transaction. The device must be told where to find the first descriptor, it can then operate autonomously since each descriptor points to the next.

Status Reporting

There needs to be some mechanism for a device to inform the software application that a DMA buffer requires service. This notification must occur in a timely matter to avoid losing the information as the device continuously cycles through all available blocks. As bus master, the device is the sole source of status regarding the DMA progress. There are two generally accepted techniques to convey these updates, interrupts and polling.

Interrupts can be generated by a device to inform the CPU of status updates or errors that require attention. In the case of DMA transfers, an interrupt can be issued when some number of blocks require service. Of course, the interrupt must be identified by the CPU

and routed to the appropriate service routine. Unfortunately, that process takes time and can be unpredictable without real-time extensions to the operating system. High rate data transfers may require very large buffers to help overcome this latency.

The alternative to interrupts is polling by the application software. In other words, the application simply queries the device for status on some regular interval. This is a less elegant approach since the polling operation consumes CPU and bus resources when there may be no change in status. However, the impact can be minimized by smart scheduling of the polling interval based on the expected DMA buffer refresh rate.

Write Transaction

A bus master DMA write transaction occurs when data is transferred from the device to main memory. As bus master, the device is essentially writing data into the designated buffer. Once a write transaction is initiated, it must be able to complete uninterrupted through the length of the burst. As shown in Figure 3, a hardware FIFO in the device is usually employed to ensure that there are no gaps in the data. The FIFO depth needs to be greater than the length of a single burst so that the device can continue to accept new input data while waiting for access to the bus.

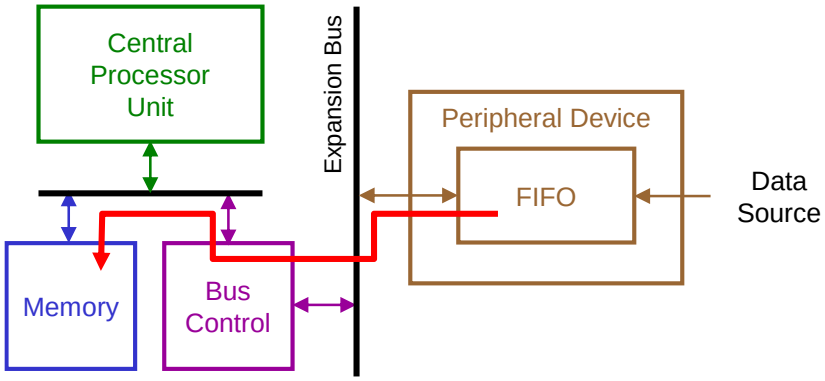


Figure 3 Bus Master DMA Write Transaction

It is important to note that increasing the size of the FIFO can overcome the latency incurred negotiating access to the bus, but it will not compensate for a lack of sufficient bus bandwidth. If new data is arriving at the FIFO at a rate faster than the bus can sustain, adding FIFO capacity simply delays the inevitable overflow that will occur. If the bus is just slow to grant access between transfers, a large FIFO provides storage for the new arriving data.

Read Transaction

A bus master DMA read transaction occurs when data is transferred from main memory to the device. As bus master, the device is essentially reading data from the designated buffer. Once a read transaction is initiated, it must be able to complete uninterrupted through the length of the burst. Some type of memory in the device is usually used to store the incoming data stream. As shown in Figure 4, a FIFO is frequently selected to fulfill this role. The FIFO depth needs to be greater than the length of a single burst so that the device does not run out of data samples to process while waiting for the next DMA transaction.

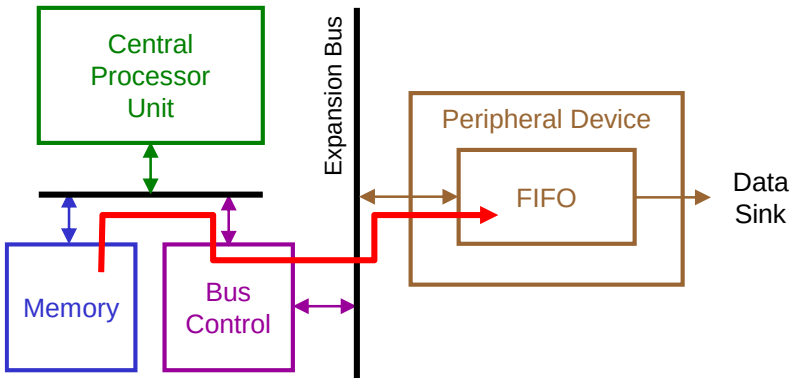


Figure 4 Bus Master DMA Read Transaction

It is important to note that increasing the size of the FIFO can overcome the latency incurred negotiating access to the bus, but it will not compensate for a lack of sufficient bus bandwidth. If new data is removed from the FIFO at a rate faster than the bus can sustain, adding FIFO capacity simply delays the inevitable underflow that will occur. If the bus is simply slow to grant access between transfers, a large FIFO acts as a buffer to supply samples to the device.

Multi-Channel

It is possible for a device to operate more than one DMA channel to memory. This is a convenient approach to keep unique data streams separated in memory. Each DMA channel operates independently with a unique set of pages allocated by the host. There is really no limit to the number of channels that can be operated by a single device, but the complexity of tracking the progress of multiple channels introduces practical considerations.