DIRECT MEMORY ACCESS

LokeshMadan[1], KislayAnand[2]and Bharat Bhushan[3]

[1]*Department of Computer Science, Dronacharya College of Engineering, Gurgaon, India*

[2]*Department of Computer Science, Dronacharya College of Engineering, Gurgaon, India*

[3]*Department of Computer Science, Dronacharya College of Engineering, Gurgaon, India*

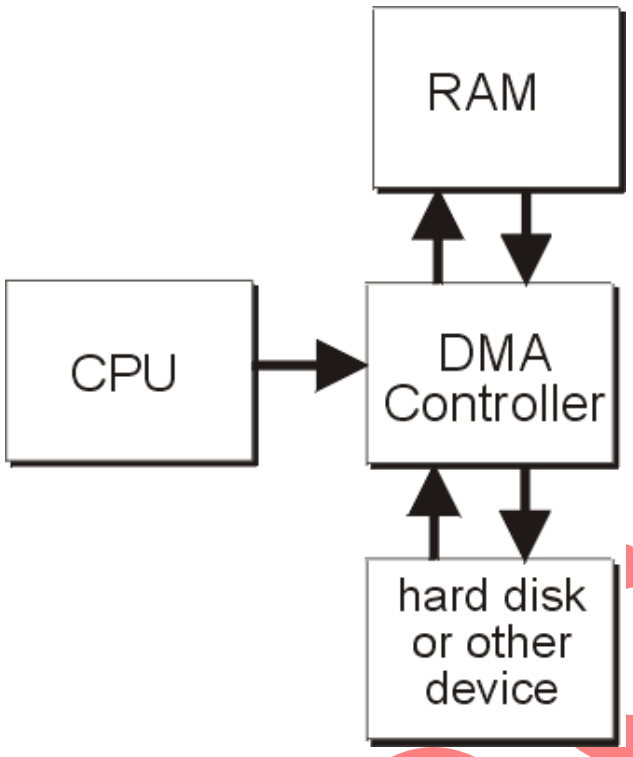
ABSTRACT

Direct Memory Access (DMA) *is a feature in all modern computers that allow devices to be able to move large blocks of data without any interaction with the processor. This can be useful, as you may have already seen from the floppy programming chapter. While the device transfers the block of data, the processor is free to continue running the software without worry about the data being transferred into memory, or to another device. The basic idea is that we can schedule the DMA device to perform the task on its own. Different buses and architecture designs have different methods of performing direct memory access.*

KEYWORDS: Processor register,cyclestealing,busmastering,countregisters,interleaved,c ache invalidation, schematics, gigabit Ethernet.

INTRODUCTION

A DMA controller can generate memory addresses and initiate memory read or write cycles. It contains several processor registers that can be written and read by the CPU. These include a memory address register, a byte count register, and one or more control registers. The control registers specify the I/O port to use, the direction of the transfer (reading from the I/O device or writing to the I/O device), the transfer unit (byte at a time or word at a time), and the number of bytes to transfer in one burst.



To carry out an input, output or memory-to-memory operation, the host processor initializes the DMA controller with a count of the number of words to transfer, and the memory address to use. The CPU then sends commands to a peripheral device to initiate transfer of data. The DMA controller then provides addresses and read/write control lines to the system memory. Each time a byte of data is ready to be transferred between the peripheral device and memory, the DMA controller increments its internal address register until the full block of data is transferred.

DMA transfers can either occur one byte at a time or all at once in burst mode. If they occur a byte at a time, this can allow the CPU to access memory on alternate bus cycles – this is called cycle stealing.Since the DMA controller and CPU contend for memory access. In *burst mode DMA*, the CPU can be put on hold while the DMA transfer occurs and a full block of possibly hundreds or thousands of bytes can be moved. When memory cycles are much faster than processor cycles, an *interleaved* DMA cycle is possible, where the DMA controller uses memory while the CPU cannot.

In a bus mastering system, both the CPU and peripherals can be granted control of the memory bus. Where a peripheral can become bus master, it can directly write to system memory without involvement of the CPU, providing memory address and control signals as required. Some measure must be provided to put the processor into a hold condition so that bus contention does not occur

DMA OPERATION MODES

Burst mode

An entire block of data is transferred in one contiguous sequence. Once the DMA controller is granted access to the system bus by the CPU, it transfers all bytes of data in the data block before releasing control of the system buses back to the CPU, but renders the CPU inactive for relatively long periods of time. The mode is also called "Block Transfer Mode".it is also used to stop unnecessary data.

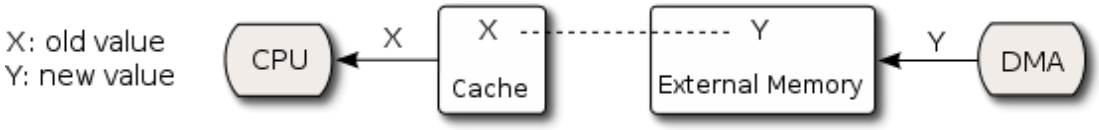
Cycle stealing mode

The *cycle stealing mode* is used in systems in which the CPU should not be disabled for the length of time needed for burst transfer modes. In the cycle stealing mode, the DMA controller obtains access to the system bus the same way as in burst mode, using **BR (Bus Request) and BG (Bus Grant) signals**, which are the two signals controlling the interface between the CPU and the DMA controller. However, in cycle stealing mode, after one byte of data transfer, the control of the system bus is deasserted to the CPU via BG. It is then continually requested again via BR, transferring one byte of data per request, until the entire block of data has been transferred. By continually obtaining and releasing the control of the system bus, the DMA controller essentially interleaves instruction and data transfers. The CPU processes an instruction, then the DMA controller transfers one data value, and so on. On the one hand, the data block is not transferred as quickly in cycle stealing mode as in burst mode, but on the other hand the CPU is not idled for as long as in burst mode. Cycle stealing mode is useful for controllers that monitor data in real time.

Transparent mode

The *transparent mode* takes the most time to transfer a block of data, yet it is also the most efficient mode in terms of overall system performance. The DMA controller only transfers data when the CPU is performing operations that do not use the system buses. It is the primary advantage of the transparent mode that the CPU never stops executing its programs and the DMA transfer is free in terms of time. The disadvantage of the transparent mode is that the hardware needs to determine when the CPU is not using the system buses, which can be complex.

DMA CACHE COHERENCY



Cache incoherence due to DMA

DMA can lead to cache coherency problems. Imagine a CPU equipped with a cache and an external memory that can be accessed directly by devices using DMA. When the CPU accesses location X in the memory, the current value will be stored in the cache. Subsequent operations on X will update the cached copy of X, but not the external memory version of X, assuming a write-back cache. If the cache is not flushed to the memory before the next time a device tries to access X, the device will receive a stale value of X.

Similarly, if the cached copy of X is not invalidated when a device writes a new value to the memory, then the CPU will operate on a stale value of X.

This issue can be addressed in one of two ways in system design: Cache-coherent systems implement a method in hardware whereby external writes are signaled to the cache controller which then performs a cache invalidation for DMA writes or cache flush for DMA reads. Non-coherent systems leave this to software, where the OS must then ensure that the cache lines are flushed before an outgoing DMA transfer is started and invalidated before a memory range affected by an incoming DMA transfer is accessed. The OS must make sure that the memory range is not accessed by any running threads in the meantime. The latter approach introduces some overhead to the DMA operation, as most hardware requires a loop to invalidate each cache line individually.

Hybrids also exist, where the secondary L2 cache is coherent while the L1 cache (typically on-CPU) is managed by software.

EXAMPLES:

ISA

In the original IBM PC, there was only one Intel 8237 DMA controller capable of providing four DMA channels (numbered 0–3), as part of the so-called Industry Standard Architecture, or ISA.

These DMA channels performed 8-bit transfers and could only address the first megabyte of RAM. With the IBM PC/AT, a second 8237 DMA controller was added (channels 5–7; channel 4 is dedicated as a cascade channel for the first 8237 controller), and the page register was rewired to address the full 16 MB memory address space of the 80286 CPU. This second controller performed 16-bit transfers.

Due to their lagging performance (2.5 Mbit/s), these devices have been largely obsolete since the advent of the 80386processor in 1985 and its capacity for 32-bit transfers. They are still supported to the extent they are required to support built-in legacy PC hardware on modern machines. The only pieces of legacy hardware that use ISA DMA and are still fairly common are Super I/O devices on motherboards that often integrate the a built-in floppy disk controller, an IrDA infrared controller when FIR (fast infrared) mode is selected, and a IEEE 1284 parallel port controller when ECP mode is selected.

Each DMA channel has a 16-bit address register and a 16-bit count register associated with it. To initiate a data transfer the device driver sets up the DMA channel's address and count registers together with the direction of the data transfer, read or write. It then instructs the DMA hardware to begin the transfer. When the transfer is complete, the device interrupts the CPU.

Scatter-gather or vectored I/O DMA allows the transfer of data to and from multiple memory areas in a single DMA transaction. It is equivalent to the chaining together of multiple simple DMA requests. The motivation is to off-load multiple input/outputinterrupt and data copy tasks from the CPU.

DRQ stands for *Data request*; DACK for *Data acknowledge*. These symbols, seen on hardware schematics of computer systems with DMA functionality, represent electronic signaling lines between the CPU and DMA controller. Each DMA channel has one Request and one Acknowledge line. A device that uses DMA must be configured to use both lines of the assigned DMA channel.

Standard ISA DMA assignments:

1. DRAM Refresh (obsolete),

2. User hardware,

3. Floppy disk controller,

4. Hard disk(obsoleted by PIO modes, and replaced by UDMA

modes), Parallel Port (ECP capable port),

5. Cascade from XT DMA controller,

6. Hard Disk (PS/2 only), user hardware for all others,

7. User hardware.

8. User hardware.

PCI

A PCI architecture has no central DMA controller, unlike ISA. Instead, any PCI component can request control of the bus ("become the bus master") and request to read from and write to system memory. More precisely, a PCI component requests bus ownership from the PCI bus controller (usually the southbridge in a modern PC design), which will arbitrate if several devices request bus ownership simultaneously, since there can only be one bus master at one time. When the component is granted ownership, it will issue normal read and write commands on the PCI bus, which will be claimed by the bus controller and will be forwarded to the memory controller using a scheme which is specific to every chipset.

As an example, on a modern AMDSocket AM2-based PC, the southbridge will forward the transactions to the northbridge (which is integrated on the CPU die) using HyperTransport, which will in turn convert them to DDR2 operations and send them out on the DDR2 memory bus. As can be seen, there are quite a number of steps involved in a PCI DMA transfer; however, that poses little problem, since the PCI device or PCI bus itself are an order of magnitude slower than the rest of the components (see list of device bandwidths).

A modern x86 CPU may use more than 4 GB of memory, utilizing PAE, a 36-bit addressing mode, or the native 64-bit mode of x86-64 CPUs. In such a case, a device using DMA with a 32-bit address bus is unable to address memory above the 4 GB line. The new Double Address Cycle (DAC) mechanism, if implemented on both the PCI bus and the device itself, enables 64-bit DMA addressing. Otherwise, the operating system would need to work around the problem by either using costly double buffers (DOS/Windows nomenclature) also known as bounce buffers (FreeBSD/Linux), or it could use an IOMMU to provide address translation services if one is present.

I/OAT

As an example of DMA engine incorporated in a general- purpose CPU, newer Intel Xeon chipsets include a DMA engine technology called I/O Acceleration Technology (I/OAT), meant to improve network performance on high-throughput network interfaces, in particular gigabit Ethernet and faster. However, various benchmarks with this approach by Intel's Linux kernel developer Andrew Grover indicate no more than 10% improvement in CPU utilization with receiving workloads, and no improvement when transmitting data.

AHB

Main article: Advanced Microcontroller Bus Architecture

In systems-on-a-chip and embedded systems, typical system bus infrastructure is a complex on-chip bus such as AMBA High- performance Bus. AMBA defines two kinds of AHB components: master and slave. A slave interface is similar to programmed I/O through which the software (running on embedded CPU, e.g. ARM) can write/read I/O registers or (less commonly) local memory blocks inside the device. A master interface can be used by the device to perform DMA transactions to/from system memory without heavily loading the CPU.

Therefore high bandwidth devices such as network controllers that need to transfer huge amounts of data to/from system memory will have two interface adapters to the AHB: a master and a slave interface. This is because on-chip buses like AHB do not support tri-stating the bus or alternating the direction of any line on the bus. Like PCI, no central DMA controller is required since the DMA is bus-mastering, but an arbiter is required in case of multiple masters present on the system.

Internally, a multichannel DMA engine is usually present in the device to perform multiple concurrent scatter-gather operations as programmed by the software.

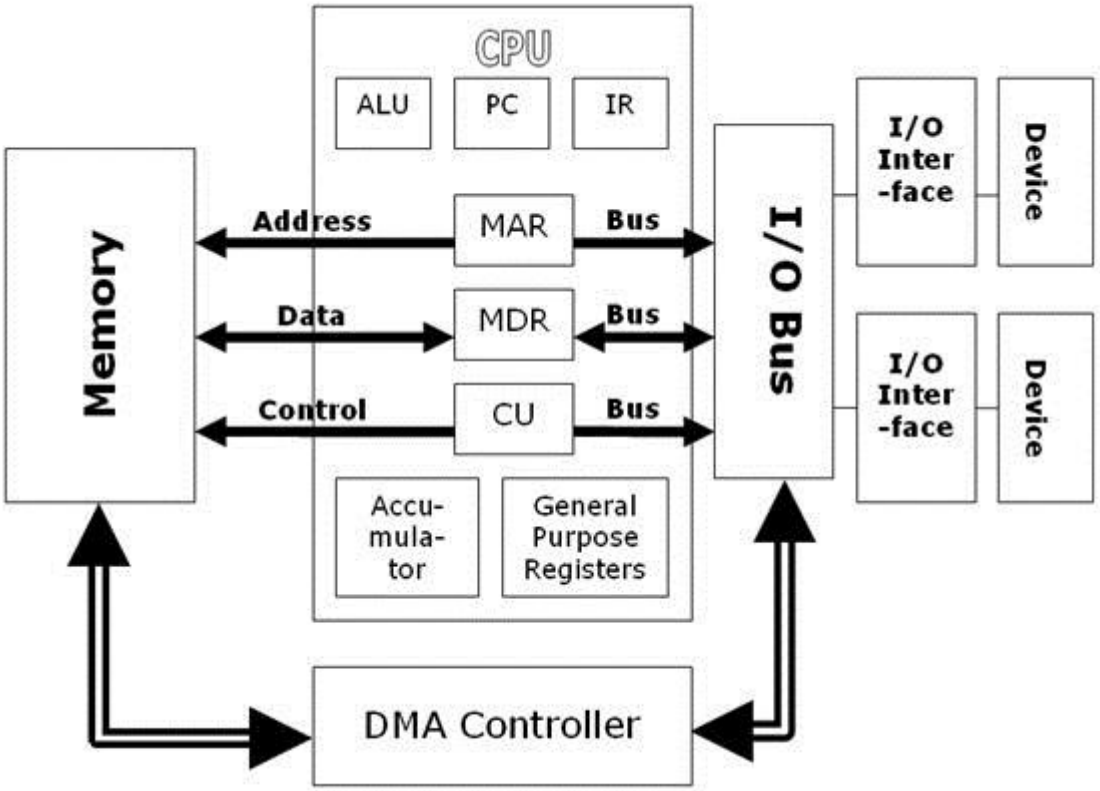
Cell

As an example usage of DMA in a multiprocessor-system-on- chip, IBM/Sony/Toshiba's Cell processor incorporates a DMA engine for each of its 9 processing elements including one Power processor element (PPE) and eight synergistic processor elements (SPEs). Since the SPE's load/store instructions can read/write only its own local memory, an SPE entirely depends on DMAs to transfer data to and from the main memory and local memories of other SPEs. Thus the DMA acts as a primary means of data transfer among cores inside this CPU (in contrast to cache-coherent CMP architectures such as Intel's cancelled general-purpose GPU, Larrabee)

DMA in Cell is fully cache coherent (note however local stores of SPEs operated upon by DMA do not act as globally coherent cache in the standard sense). In both read ("get") and write ("put"), a DMA command can transfer either a single block area of size up to 16 KB, or a list of 2 to 2048 such blocks. The DMA command is issued by specifying a pair of a local address and a remote address: for example when a SPE program issues a put DMA command, it specifies an address of its own local memory as the source and a virtual memory address (pointing to either the main memory or the local memory of another SPE) as the target, together with a block size. According to a recent experiment, an effective peak performance of DMA in Cell (3 GHz, under uniform traffic) reaches 200 GB per second.[7]

IMPLEMENTING DMA

To implement the direct transfer of data between memory and I/O a hardware DMA controller is added to the system. This hardware works like a bypass for a town, allowing large blocks of data to bypass the CPU without snarling it up.



In this scenario large blocks of data are transferred using the DMA hardware. The CPU, on receiving an interrupt, initiates the DMA hardware with

- A memory start address
- The amount of data to be transferred
- The device to be used (hard disk, CD, DVD, etc.)
- The direction of transfer (input or output)

The DMA controller then transfers the data. Upon completion the controller notifies the CPU that the transfer has been completed.

FUTURE SCOPE:

The impact of multiprocessor, multiple DMA channels for DMA cache

A shared cache with an intelligent replacement policy can achieve the effect of DMA cache scheme.

CONCLUSIONS

Memory plays a role of transient place for I/O data transferred between processor and I/O device

DMA cache can significantly improve I/O performance.

REFERENCES

DMA Fundamentals on Various PC Platforms, from A. F. Harvey and Data Acquisition Division Staff NATIONAL INSTRUMENTS

mmap() and DMA, from *Linux Device Drivers, 2nd Edition*, Alessandro Rubini&Jonathan Corbet

Memory Mapping and DMA, from *Linux Device Drivers, 3rd Edition*, Jonathan Corbet, Alessandro Rubini, Greg Kroah- Hartman