WIKIPEDIA

Direct memory access

Direct memory access (**DMA**) is a feature of computer systems that allows certain hardware subsystems to access main system memory (random-access memory), independent of the central processing unit (CPU).

Without DMA, when the CPU is using programmed input/output, it is typically fully occupied for the entire duration of the read or write operation, and is thus unavailable to perform other work. With DMA, the CPU first initiates the transfer, then it does other operations while the transfer is in progress, and it finally receives an interrupt from the DMA controller (DMAC) when the operation is done. This feature is useful at any time that the CPU cannot keep up with the rate of data transfer, or when the CPU needs to perform work while waiting for a relatively slow I/O data transfer. Many hardware systems use DMA, including disk drive controllers, graphics cards, network cards and sound cards. DMA is also used for intra-chip data transfer in multi-core processors. Computers that have DMA channels can transfer data to and from devices with much less CPU overhead than computers without DMA channels. Similarly, a processing element inside a multi-core processor can transfer data to and from its local memory without occupying its processor time, allowing computation and data transfer to proceed in parallel.

DMA can also be used for "memory to memory" copying or moving of data within memory. DMA can offload expensive memory operations, such as large copies or scatter-gather operations, from the CPU to a dedicated DMA engine. An implementation example is the I/O Acceleration Technology. DMA is of interest in network-on-chip and in-memory computing architectures.

Contents

Principles

Third-party

Standard DMA, also called third-party DMA, uses a DMA controller. A DMA controller can generate memory addresses and initiate memory read or write cycles. It contains several hardware 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. Depending on what features the DMA controller provides, these control registers might specify some combination of the source, the destination, the direction of the transfer (reading from the I/O device or writing to the I/O device), the size of the transfer unit, and/or the number of bytes to transfer in one burst.[1]

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 commands peripheral device to initiate data transfer. 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.

Bus mastering

In a bus mastering system, also known as a first-party DMA system, the CPU and peripherals can each 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.

Transfer types

DMA transfers can transfer either one byte at a time or all at once in burst mode. If they transfer a byte at a time, this can allow the CPU to access memory on alternating bus cycles – this is called cycle stealing since the CPU and either the DMA controller or the bus master 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.[2] 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.

Modes of operation

Burst mode

In *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".

Cycle stealing mode

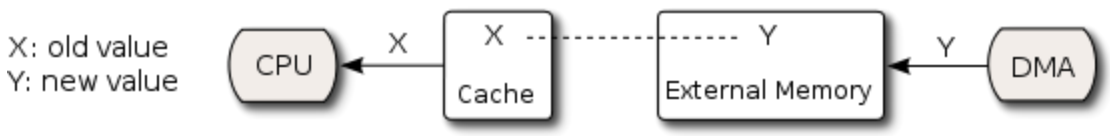
The *cycle stealing mode* is used in systems in which the CPU should not be disabled for the length of time needed for burst transfer modes. In the cycle stealing mode, the DMA controller obtains access to the system bus the same way as in burst mode, using *BR (Bus Request)* and *BG (Bus Grant)* signals, 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

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. In transparent mode, the DMA controller transfers data only when the CPU is performing operations that do not use the system buses. The primary advantage of transparent mode is that the CPU never stops executing its programs and the DMA transfer is free in terms of time, while the disadvantage is that the hardware needs to determine when the CPU is not using the system buses, which can be complex.This is also called as "Hidden DMA data transfer mode".

Cache coherency

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.



Cache incoherence due to DMA

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 (and the follow-up PC/XT), there was only one Intel 8237 DMA controller capable of providing four DMA channels (numbered 0–3). These DMA channels performed 8-bit transfers (as the 8237 was an 8-bit device, ideally matched to the PC's i8088 CPU/bus architecture), could only address the first (i8086/8088-standard) megabyte of RAM, and were limited to addressing single 64 kB segments within that space (although the source and destination channels could address different segments). Additionally, the controller could only be used for transfers to, from or between expansion bus I/O devices, as the 8237 could only perform memory-to-memory transfers using channels 0 & 1, of which channel 0 in the PC (& XT) was dedicated to dynamic memory refresh. This prevented it from being used as a general-purpose "Blitter", and consequently block memory moves in the PC, limited by the general PIO speed of the CPU, were very slow.

With the IBM PC/AT, the enhanced AT Bus (more familiarly retronymed as the ISA, or "Industry Standard Architecture") added a second 8237 DMA controller to provide three additional, and as highlighted by resource clashes with the XT's additional expandability over the original PC, much-needed channels (5–7; channel 4 is used as a cascade to the first 8237). The page register was also rewired to address the full 16 MB memory address space of the 80286 CPU. This second controller was also integrated in a way capable of performing 16-bit transfers when an I/O device is used as the data source and/or destination (as it actually only processes data itself for memory-to-memory transfers, otherwise simply *controlling* the data flow between other parts of the 16-bit system, making its own data bus width relatively immaterial), doubling data throughput when the upper three channels are used. For compatibility, the lower four DMA channels were

still limited to 8-bit transfers only, and whilst memory-to-memory transfers were now technically possible due to the freeing up of channel 0 from having to handle DRAM refresh, from a practical standpoint they were of limited value because of the controller's consequent low throughput compared to what the CPU could now achieve (i.e, a 16-bit, more optimised 80286 running at a minimum of 6 MHz, vs an 8-bit controller locked at 4.77 MHz). In both cases, the 64 kB segment boundary issue remained, with individual transfers unable to cross segments (instead "wrapping around" to the start of the same segment) even in 16-bit mode, although this was in practice more a problem of programming complexity than performance as the continued need for DRAM refresh (however handled) to monopolise the bus approximately every 15 µs prevented use of large (and fast, but uninterruptible) block transfers.

Due to their lagging performance (1.6 MB/s maximum 8-bit transfer capability at 5MHz[3], but no more than 0.9 MB/s in the PC/XT and 1.6 MB/s for 16-bit transfers in the AT due to ISA bus overheads and other interference such as memory refresh interruptions[4]) and unavailability of any speed grades that would allow installation of direct replacements operating at speeds higher than the original PC's standard 4.77 MHz clock, these devices have been effectively obsolete since the late 1980s. Particularly, the advent of the 80386 processor in 1985 and its capacity for 32-bit transfers (although great improvements in the efficiency of address calculation and block memory moves in Intel CPUs after the 80186 meant that PIO transfers even by the 16-bit-bus 286 and 386SX could still easily outstrip the 8237), as well as the development of further evolutions to (EISA) or replacements for (MCA, VLB and PCI) the "ISA" bus with their own much higher-performance DMA subsystems (upto a maximum of 33 MB/s for EISA, 40 MB/s MCA, typically 133 MB/s VLB/PCI) made the original DMA controllers seem more of a performance millstone than a booster. They were supported to the extent they are required to support built-in legacy PC hardware on later machines. The pieces of legacy hardware that continued to use ISA DMA after 32-bit expansion buses became common were Sound Blaster cards that needed to maintain full hardware compatibility with the Sound Blaster standard; and Super I/O devices on motherboards that often integrated a built-in floppy disk controller, an IrDA infrared controller when FIR (fast infrared) mode is selected, and an IEEE 1284 parallel port controller when ECP mode is selected. In cases where a original 8237s or direct compatibles were still used, transfer to or from these devices may still be limited to the first 16 MB of main RAM regardless of the system's actual address space or amount of installed memory.

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/output interrupt 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.

16-bit ISA permitted bus mastering.[5]

Standard ISA DMA assignments:

0. DRAM Refresh (obsolete),
1. User hardware, usually sound card 8-bit DMA
2. Floppy disk controller,
3. Hard disk (obsoleted by PIO modes, and replaced by UDMA modes), Parallel Port (ECP capable port), certain SoundBlaster Clones like the OPTi 928.
4. Cascade to PC/XT DMA controller,
5. Hard Disk (PS/2 only), user hardware for all others, usually sound card 16-bit DMA
6. User hardware.
7. 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 AMD Socket 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 Physical Address Extension (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,[6] 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 called I/O Acceleration Technology (I/OAT), which can offload memory copying from the main CPU, freeing it to do other work.[7] In 2006, Intel's Linux kernel developer Andrew Grover performed benchmarks using I/OAT to offload network traffic copies and found no more than 10% improvement in CPU utilization with receiving workloads.[8]

DDIO

Further performance-oriented enhancements to the DMA mechanism have been introduced in Intel Xeon E5 processors with their **Data Direct I/O** (**DDIO**) feature, allowing the DMA "windows" to reside within CPU caches instead of system RAM. As a result, CPU caches are used as the primary source and destination for I/O, allowing network interface controllers (NICs) to DMA directly to the Last level cache of local CPUs and avoid costly fetching of the I/O data from system RAM. As a result, DDIO reduces the overall I/O processing latency, allows processing of the I/O to be performed entirely in-cache, prevents the available RAM bandwidth/latency from becoming a performance bottleneck, and may lower the power consumption by allowing RAM to remain longer in low-powered state.[9][10][11][12]

AHB

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 an experiment, an effective peak performance of DMA in Cell (3 GHz, under uniform traffic) reaches 200 GB per second.[13]

Pipelining

Processors with scratchpad memory and DMA (such as digital signal processors and the Cell processor) may benefit from software overlapping DMA memory operations with processing, via double buffering or multibuffering. For example, the on-chip memory is split into two buffers; the processor may be operating on data in one, while the DMA engine is loading and storing data in the other. This allows the system to avoid memory latency and exploit burst transfers, at the expense of needing a predictable memory access pattern.

See also

- AT Attachment
- Blitter
- Channel I/O
- DMA attack
- Polling (computer science)
- Remote direct memory access
- UDMA
- Autonomous peripheral operation
- Network on a chip
- In-memory processing
- Hardware acceleration

Notes

1. Osborne, Adam (1980). *An Introduction to Microcomputers: Volume 1: Basic Concepts* (https://archive.org/details/introductiontomi00adam) (2nd ed.). Osborne McGraw Hill. pp. 5–64 through 5–93. ISBN 0931988349.
2. Horowitz, Paul; Hill, Winfield (1989). *The Art of Electronics* (https://archive.org/details/artofelectronics00horo) (Second ed.). Cambridge University Press. p. 702. ISBN 0521370957.
3. "Intel 8237 & 8237-2 Datasheet" (http://www.jbox.dk/rc702/hardware/intel-8237.pdf) (PDF). *JKbox RC702 subsite*. Retrieved 20 April 2019.
4. "DMA Fundamentals on various PC platforms, National Instruments, pages 6 & 7" (https://www.ing.unlp.edu.ar/catedras/E0225/descargar.php?secc=0&id=E0225&id_inc=1196). *Universidad Nacional de la Plata, Argentina*. Retrieved 20 April 2019.
5. Intel Corp. (2003-04-25), "Chapter 12: ISA Bus" (http://faculty.chemeketa.edu/csekafet/elt256/pcarch-full_isa-bus.pdf) (PDF), *PC Architecture for Technicians: Level 1*, retrieved 2015-01-27
6. "Physical Address Extension — PAE Memory and Windows" (http://www.microsoft.com/whdc/system/platform/server/PAE/PAEdrv.mspx#E2D). Microsoft Windows Hardware Development Central. 2005. Retrieved 2008-04-07.
7. Corbet, Jonathan (December 8, 2005). "Memory copies in hardware" (https://lwn.net/Articles/162966/). *LWN.net*.

8. Grover, Andrew (2006-06-01). "I/OAT on LinuxNet wiki" (http://www.linuxfoundation.org/collaborate/workgroups/networking/i/oat). *Overview of I/OAT on Linux, with links to several benchmarks*. Retrieved 2006-12-12.
9. "Intel Data Direct I/O (Intel DDIO): Frequently Asked Questions" (http://www.intel.com/content/dam/www/public/us/en/documents/faqs/data-direct-i-o-faq.pdf) (PDF). Intel. March 2012. Retrieved 2015-10-11.
10. Rashid Khan (2015-09-29). "Pushing the Limits of Kernel Networking" (http://rhelblog.redhat.com/2015/09/29/pushing-the-limits-of-kernel-networking/). *redhat.com*. Retrieved 2015-10-11.
11. "Achieving Lowest Latencies at Highest Message Rates with Intel Xeon Processor E5-2600 and Solarflare SFN6122F 10 GbE Server Adapter" (http://www.solarflare.com/content/userfiles/documents/intel_solarflare_webinar_paper.pdf) (PDF). *solarflare.com*. 2012-06-07. Retrieved 2015-10-11.
12. Alexander Duyck (2015-08-19). "Pushing the Limits of Kernel Networking" (http://events.linuxfoundation.org/sites/events/files/slides/pushing-kernel-networking.pdf#page=5) (PDF). *linuxfoundation.org*. p. 5. Retrieved 2015-10-11.
13. Kistler, Michael (May 2006). "Cell Multiprocessor Communication Network" (http://portal.acm.org/citation.cfm?id=1158825.1159067). *Extensive benchmarks of DMA performance in Cell Broadband Engine*.

References

- DMA Fundamentals on Various PC Platforms (http://cires.colorado.edu/jimenez-group/QAMSResources/Docs/DMAFundamentals.pdf), from A. F. Harvey and Data Acquisition Division Staff NATIONAL INSTRUMENTS
- mmap() and DMA (http://www.xml.com/ldd/chapter/book/ch13.html), from *Linux Device Drivers, 2nd Edition*, Alessandro Rubini & Jonathan Corbet
- Memory Mapping and DMA (http://www.oreilly.com/catalog/linuxdrive3/book/ch15.pdf), from *Linux Device Drivers, 3rd Edition*, Jonathan Corbet, Alessandro Rubini, Greg Kroah-Hartman
- DMA and Interrupt Handling (http://www.eventhelix.com/RealtimeMantra/FaultHandling/dma_interrupt_handling.htm)
- DMA Modes & Bus Mastering (http://www.pcguide.com/ref/hdd/if/ide/modesDMA-c.html)

External links

- Mastering the DMA and IOMMU APIs (http://elinux.org/images/4/49/20140429-dma.pdf), Embedded Linux Conference 2014, San Jose, by Laurent Pinchart