Direct Memory Access (DMA)

- So far, we have discussed programmed I/O
  - With busy wait
  - Interrupt-driven
- CPU may have other things to do, or interrupt processing for each word transferred may be too slow to deal with fast I/O device
- Want to delegate control of substantial I/O operation to DMA controller (DMAC)
- CPU (driver driver) initiates by sending the operation, start memory address and transfer size to it
- CPU also issues a command to device controller (DC)
- DMA commences when Ack arrives from DC, indicating its buffer is full (in case of input)
- DMA controller essentially does programmed I/O

Notes

- DMAC can grab the busses at each cycle during instruction execution (cycle stealing) → immediate attention (interrupt can take place only between instructions (multiple cycles))
- Instruction execution is suspended during DMA, which requires the address and data buses for transfer to/from memory.
  - Multiple words can be transferred before the busses are released (burst mode)
  - CPU can still access cache memory
- CPU is interrupted by DMAC just once after the completion of the entire transfer (tremendous saving?)

Transfer via DMAC

1. Device driver is told to transfer a block by
2. Device driver tells controller to transfer to/from memory
3. DMAC initiates CPU to signal transfer completion
4. Controller sends copy of data to buffer at address K
5. DMA controller transfers block to buffer by increasing memory address and decreases K until K = 0
6. When K = 0, DMA interrupts CPU to signal transfer completion

DMAC steps

- After Ack (4) from DC, DMAC repeats the following:
  - Grabs buses from CPU (by a signal on a control wire in bus)
  - Presents address on address bus, sets direction bit (from/to memory), and sends transfer request (2) to DC
  - Waits for Ack (4) from DC, informing completion of transfer of a word to/from memory (3)
  - Interrupts CPU when count words have been transferred.
Printing a string using DMA

```c
#define USER_BUFFER_SIZE 512

void print_string(char *buffer) {
    int length = strlen(buffer);
    copy_from_user(buffer, p, count);
    set_up_dma_controller();
    scheduler();
    acknowledge_interrupt();
    unblock_user();
    return_from_interrupt();
}
```

(a) Kernel code executed when the print system call is made.
(b) Interrupt service procedure: interrupt is sent from DMA to CPU upon completion of entire transfer.

---

Standard buses

- **PCI (Peripheral Component Interconnection)**
  - 8-bytes parallel, 528MB/sec
  - Successor to IBM PC ISA (Industry Standard Architecture, 2-bytes parallel, 16.67MB/sec)
  - Widely used
- **USB (Universal Serial Bus)**
  - For slow serial devices (e.g., keyboard, mouse): 4 wires (2 are for power)
  - Centralized root device polls the attached devices every 1ms. One device driver for all.
  - Aggregate traffic up to 1.5MB/sec.
  - Faster version being developed.

---

Continued ... 

- **IEEE 1394: “Fire Wire”**
  - Serial, up to 50 MB/sec; good for camcorders and other multimedia devices
  - Does not have central controller.
- **SCSI (Small Computer System Interface)**
  - For fast devices, e.g., disks, scanners
  - Handles speeds up to 160MB/sec
- **IDE (Integrated Drive Electronics)**
  - Electrical interface between disk controller and disk; standard on Pentium systems.

---

Recall I/O software layers

```
Layer I/O layer I/O functions
User processes Device-independent software Make I/O call; format I/O; spooling
Device drivers Interrupt handlers Set up device registers; check status
Hardware Wake up driver when I/O completed
```

Perform I/O operation
I/O software goals

- Device independence
  - Device drivers hide differences among devices (floppy, hard drive, or CD-ROM)
- Uniform naming
  - Name of a file or device is simply a string or an integer
- Uniform block size
  - Device-independent block size
- Error handling (e.g., CRC error on a disk block)
  - Handle as close to the hardware as possible for efficiency

Device-independent I/O software

(a) Without a standard device driver interface
(b) With a standard device driver interface

Temporary storage in I/O software

- Buffering (Buffer = memory area that stores data while they are being transferred)
- To accommodate speed mismatch (slow input from modem buffered and then sent to disk)
- Between devices with different data-transfer sizes.
- Copy semantics: to preserve the original form
- Caching (Cache—in fast memory, normally holds copies, cf., buffer)
- For fast access
- Sometimes, the difference is blurred (e.g., block buffer cache for disk blocks)
- Spooling (Spool—buffer that holds output for a device that cannot accept interleaved data streams)

Buffering in network communications

For copy semantics
Buffering schemes

- Unbuffered input (user process interrupted for every byte)
- Buffering in user space
  - N bytes at a time
  - Interrupt on completion
  - Buffer pages (of all processes doing I/O) must be in memory, occupying page frames, leaving few for other processes
- Buffering in the kernel
  - Copied to user space when full
  - User's buffer page need be in memory only when copying takes place
- Double buffering in the kernel
  - Solves the problem of full kernel buffer while user buffer page is being brought in