I/O Hardware

a typical PCI bus structure
How do the processor and controller communicate?

- Use the controller: a controller usually has a few registers (e.g., status, control, data-in and data-out).
- Use memory-mapped I/O.
- Or, a combination of both.
# Memory-Mapped I/O

Each controller has a few registers that are used for communicating with the CPU.

If these registers are part of the regular memory address space, it is called memory-mapped I/O.

<table>
<thead>
<tr>
<th>I/O address</th>
<th>Device</th>
</tr>
</thead>
<tbody>
<tr>
<td>000-00F</td>
<td>DMA controller</td>
</tr>
<tr>
<td>020-021</td>
<td>Interrupt controller</td>
</tr>
<tr>
<td>040-043</td>
<td>timer</td>
</tr>
<tr>
<td>200-20F</td>
<td>Game controller</td>
</tr>
<tr>
<td>2F8-2FF</td>
<td>Serial port (secondary)</td>
</tr>
<tr>
<td>320-32F</td>
<td>Hard-disk controller</td>
</tr>
<tr>
<td>378-37F</td>
<td>Parallel port</td>
</tr>
<tr>
<td>3D0-3DF</td>
<td>Graphics controller</td>
</tr>
<tr>
<td>3F0-3F7</td>
<td>Floppy-disk controller</td>
</tr>
<tr>
<td>3F8-3FF</td>
<td>Serial port (primary)</td>
</tr>
</tbody>
</table>
Three Commonly Seen Protocols

- Pooling
- Interrupts
- Direct Memory Access (DMA)
## Polling

- The status register has two bits, *busy* and *command-ready*. 

<table>
<thead>
<tr>
<th>Processor</th>
<th>Controller</th>
</tr>
</thead>
<tbody>
<tr>
<td>wait until the <strong>busy</strong> bit is not set</td>
<td>if <strong>command-ready</strong> is set, set <strong>busy</strong></td>
</tr>
<tr>
<td>set the <strong>write</strong> bit in command</td>
<td>do input/output transfer</td>
</tr>
<tr>
<td>set <strong>command-ready</strong> bit</td>
<td>clear the <strong>command-ready</strong> and <strong>busy</strong></td>
</tr>
</tbody>
</table>
Interrupt device driver initiates I/O
receives interrupt calls interrupt handler
processes data and returns
resumes the Interrupt task

CPU

I/O Controller

initiates I/O

done raises interrupt
Direct Memory Access: 1/2

- For large volume data transfer, most systems use direct memory access to avoid burdening the CPU.
- The CPU gives the controller (1) disk address, (2) memory address for storing the block, and (3) a byte count. Then, the CPU goes back to work.
Direct Memory Access: 2/2

- DMA requests data transfer to memory
- The disk controller copies the information into the address provided by the CPU, byte-by-byte, until the counter becomes 0, at which time an interrupt is generated.
Application I/O Interface
I/O Devices

- **Character stream**: a character stream device transfers byte one by one (e.g., modem)
- **Block**: a block device transfers a block of bytes as a unit (e.g., disk)
- **Others**: clocks, memory-mapped screens and so on.
- **Not all devices may be recognized by an OS. Thus, device drivers are needed.**
Kernel I/O System

- Build on top of hardware and device drivers, the kernel usually provide many I/O services:
  - I/O scheduling (e.g., disk head scheduling)
  - I/O Buffering (see below)
  - Caching (see below)
  - Spooling
  - Error handling
Buffering: 1/2

- A buffer is a memory area that stores data while they are transferred between two devices or between a device and an application.

- Major reasons of using buffers
  - Efficiency (see below)
  - Copy semantics. What if there is no buffer and a process runs so fast that overwrites its previous write? The content on the disk becomes incorrect. The use of buffers overcomes this problem.
Buffering: 2/2

No buffer. The user process must wait until data transfer completes.

One buffer: While the user process is running, next data transfer may begin

Double buffer: while the user process is processing the first buffer, data transfer can be performed on the second.

Multiple buffers: very efficient

(figures taken from W. Stallings’ OS text)
Caching

- Just like a cache memory between the faster CPU and slower physical memory, a cache (i.e., disk cache) may be used between the faster physical memory and slower I/O devices.

- Note that buffering and caching are different things.