The System Bus

- **System bus**: connects system components together
  - Important: insufficient bandwidth can bottleneck entire system
  - Performance factors
    - Physical length
    - Number and type of connected devices (taps)

Three Buses

- **Processor-memory bus**
  - Connects CPU and memory, no direct I/O interface
  - Short, few taps → fast, high-bandwidth
  - System specific

- **I/O bus**
  - Connects I/O devices, no direct P-M interface
  - Longer, more taps → slower, lower-bandwidth
  - Industry standard

- **Backplane bus**
  - CPU, memory, I/O connected to same bus
  - Industry standard, cheap (no adapters needed)
  - Processor-memory performance compromised

Bus Design

- **Goals**
  - **High Performance**: low latency and high bandwidth
  - **Standardization**: flexibility in dealing with many devices
  - **Low Cost**
    - Processor-memory bus emphasizes performance, then cost
    - I/O & backplane emphasize standardization, then performance

- **Design issues**
  1. **Width/multiplexing**: are wires shared or separate?
  2. **Clocking**: is bus clocked or not?
  3. **Switching**: how/when is bus control acquired and released?
  4. **Arbitration**: how do we decide who gets the bus next?

(1) Bus Width and Multiplexing

- [material in class]
(2) Bus Clocking

- **Synchronous**: clocked
  - Fast
  - Bus must be short to minimize clock skew
- **Asynchronous**: un-clocked
  - Can be longer: no clock skew, deals with devices of different speeds
  - Slower: requires "hand-shaking" protocol
    - For example, asynchronous read
      1. Processor drives address onto bus, asserts Request line
      2. Memory asserts Ack line, processor stops driving
      3. Memory drives data on bus, asserts DataReady line
      4. Processor asserts Ack line, memory stops driving
- P-M buses are synchronous
- I/O and backplane buses asynchronous or slow-clock synchronous

(3) Bus Switching

- [material in class]

(4) Bus Arbitration

- **Bus master**: component that can initiate a bus request
  - Bus typically has several masters, including processor
  - I/O devices can also be masters
- **Arbitration**: choosing a master among multiple requests
  - Try to implement priority and fairness (no device "starves")
  - Daisy-chain: devices connect to bus in priority order
    - High-priority devices intercept/deny requests by low-priority ones
  - Centralized: special arbiter chip collects requests, decides
    - Ensures fairness, but arbiter chip may itself become bottleneck
  - Distributed: everyone sees all requests simultaneously
    - Back off and retry if not the highest priority request
  - No bottlenecks and fair, but needs a lot of control lines

Standard Bus Examples

<table>
<thead>
<tr>
<th>Type</th>
<th>PCI</th>
<th>SCSI</th>
<th>USB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multiplexed?</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Clocking</td>
<td>33 (66) MHz</td>
<td>5 (10) MHz</td>
<td>Asynchronous</td>
</tr>
<tr>
<td>Data rate</td>
<td>133 (266) MB/s</td>
<td>10 (20) MB/s</td>
<td>0.2, 1.5, 80 MB/s</td>
</tr>
<tr>
<td>Arbitration</td>
<td>Distributed</td>
<td>Distributed</td>
<td>Daisy-chain</td>
</tr>
<tr>
<td>Maximum masters</td>
<td>1024</td>
<td>7–31</td>
<td>127</td>
</tr>
<tr>
<td>Maximum length</td>
<td>0.5 m</td>
<td>2.5 m</td>
<td>–</td>
</tr>
</tbody>
</table>

- **USB (universal serial bus)**
  - Popular for low/moderate bandwidth external peripherals
  - Packetized interface (like TCP), extremely flexible
  - Also supplies power to the peripheral
This Unit: I/O

- I/O system structure
  - Devices, controllers, and buses
- Device characteristics
  - Disks
- Bus characteristics
- I/O control
  - Polling and interrupts
  - DMA

I/O Control and Interfaces

- Now that we know how I/O devices and buses work...
- How does I/O actually happen?
  - How does CPU give commands to I/O devices?
  - How do I/O devices execute data transfers?
  - How does CPU know when I/O devices are done?

Sending Commands to I/O Devices

- Remember: only OS can do this! Two options ...
- I/O instructions
  - OS only? Instructions must be privileged (only OS can execute)
  - E.g., IA-32
- Memory-mapped I/O
  - Portion of physical address space reserved for I/O
  - OS maps physical addresses to I/O device control registers
  - Stores/loads to these addresses are commands to I/O devices
    - Main memory ignores them, I/O devices recognize and respond
    - Address specifies both I/O device and command
    - These address are not cached – why?
  - OS only? I/O physical addresses only mapped in OS address space
  - E.g., almost every architecture other than IA-32 (see pattern??)

Querying I/O Device Status

- Now that we've sent command to I/O device ...
- How do we query I/O device status?
  - So that we know if data we asked for is ready?
  - So that we know if device is ready to receive next command?
- Polling: Ready now? How about now? How about now??
  - [material in class]
Polling Overhead: Example #1

- Parameters
  - 500 MHz CPU
  - Polling event takes 400 cycles

- Overhead for polling a mouse 30 times per second?
  - Cycles per second for polling = (30 poll/s)*(400 cycles/poll)
  - → 12000 cycles/second for polling
  - (12000 cycles/second)/(500 M cycles/second) = 0.002% overhead
  - Not bad

Polling Overhead: Example #2

- Same parameters
  - 500 MHz CPU, polling event takes 400 cycles

- Overhead for polling a 4 MB/s disk with 16 B interface?
  - Must poll often enough not to miss data from disk
  - Cycles per second for polling = [(4 MB/s)/(16 B/poll)]*(400 cyc/poll)
  - → 100 M cycles/second for polling
  - (100 M cycles/second)/(500 M cycles/second) = 20% overhead
  - Bad
  - This is the overhead of polling, not actual data transfer
    - Really bad if disk is not being used (pure overhead!)

Interrupt-Driven I/O

- **Interrupts**: alternative to polling
  - I/O device generates interrupt when status changes, data ready
  - OS handles interrupts just like exceptions (e.g., page faults)
    - Identity of interrupting I/O device recorded in ECR
    - ECR: exception cause register
  - I/O interrupts are **asynchronous**
    - Not associated with any one instruction
    - Don't need to be handled immediately
  - I/O interrupts are **prioritized**
    - Synchronous interrupts (e.g., page faults) have highest priority
    - High-bandwidth I/O devices have higher priority than low-bandwidth ones

Interrupt Overhead

- Parameters
  - 500 MHz CPU
  - Polling event takes 400 cycles
  - Interrupt handler takes 400 cycles
  - Data transfer takes 100 cycles
  - 4 MB/s, 16 B interface disk, transfers data only 5% of time

- Percent of time processor spends transferring data
  - 0.05 * (4 MB/s)/(16 B/xfer)*[(100 c/xfer)/(500M c/s)] = 0.25%

- Overhead for polling?
  - (4 MB/s)/(16 B/poll) * [(400 c/poll)/(500M c/s)] = 20%

- Overhead for interrupts?
  - 0.05 * (4 MB/s)/(16 B/poll) * [(400 c/poll)/(500M c/s)] = 1%

Note: when disk is transferring data, the interrupt rate is the same as polling rate