Input/Output

Introduction to Computer Architecture
David Black-Schaffer

Contents

• I/O devices and performance
  — Latency and throughput

• Connecting devices
  — Buses
  — Serial

• Talking to I/O devices
  — How: I/O instructions vs. memory mapped vs. DMA
  — When: polling vs. interrupts

• Example: ethernet

I/O devices and performance

I/O today

• Devices
  — Disks
  — Keyboard/Mouse
  — Cameras
  — Network
  — Displays

• Interfaces
  — USB
  — PCIe
  — SATA
  — Ethernet
  — Wireless (802.11n, Bluetooth)
  — Audio, video
  — FireWire
  — Thunderbolt

Interface technologies and speeds

<table>
<thead>
<tr>
<th>Interface</th>
<th>Throughput</th>
<th>Latency</th>
<th>Power</th>
</tr>
</thead>
<tbody>
<tr>
<td>USB 1.1</td>
<td>1.5, 480, 1200 MB/s</td>
<td>150 ns, 480 ns, 1500 ns</td>
<td>$0.06/GB</td>
</tr>
<tr>
<td>PCIe 5.0</td>
<td>512, 1024 MB/s</td>
<td>250 ns, 1024 ns, 2500 ns</td>
<td>$0.10/GB</td>
</tr>
<tr>
<td>SATA 6</td>
<td>187, 375, 750 MB/s</td>
<td>50 ns, 375 ns, 500 ns</td>
<td>$0.06/GB</td>
</tr>
<tr>
<td>Ethernet</td>
<td>1 Gbps, 10 Gbps</td>
<td>50 ns, 1 Gbps, 500 ns</td>
<td>$0.06/GB</td>
</tr>
<tr>
<td>Firewire</td>
<td>50, 100 MB/s</td>
<td>100 ns, 300 ns</td>
<td>$0.06/GB</td>
</tr>
<tr>
<td>Thunderbolt</td>
<td>125 MB/s</td>
<td>250 ns</td>
<td>$0.06/GB</td>
</tr>
<tr>
<td>IDE 2.5</td>
<td>0.75, 150 MB/s</td>
<td>300 ns, 1500 ns</td>
<td>$0.06/GB</td>
</tr>
</tbody>
</table>

Storage technologies, speeds, and costs

<table>
<thead>
<tr>
<th>Technology</th>
<th>Capacity</th>
<th>Lifespan</th>
<th>Latency</th>
<th>Throughput</th>
<th>Power</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tape</td>
<td>800 GB</td>
<td>20 years</td>
<td>10-100 s</td>
<td>140 MB/s</td>
<td>$0.06/GB</td>
<td></td>
</tr>
<tr>
<td>Disk</td>
<td>3 TB</td>
<td>3-5 years</td>
<td>8 ms</td>
<td>200 MB/s</td>
<td>$0.07/GB</td>
<td></td>
</tr>
<tr>
<td>NVRAM</td>
<td>256 GB</td>
<td>10 years</td>
<td>85 µs</td>
<td>500 MB/s</td>
<td>$0.10/GB</td>
<td></td>
</tr>
<tr>
<td>SRAM</td>
<td>16 MB</td>
<td>0</td>
<td>5 ns</td>
<td>32,384 MB/s</td>
<td>$0.12/GB</td>
<td></td>
</tr>
<tr>
<td>SRAM</td>
<td>1 MB</td>
<td>0</td>
<td>5 ns</td>
<td>32,384 MB/s</td>
<td>$0.12/GB</td>
<td></td>
</tr>
</tbody>
</table>

• NVRAM — Non-volatile Random Access Memory (retains data without power)

Q: Why would someone ever use tape?

• Cheaper
• Faster
• Lasts longer

A: Lasts longer
Hard disks are only good for a few years while data on tape can last for 10-20 years. Moreover, tapes are not really slower than it is being replaced by hard disks even for archiving. The idea is that software and extra disks can ensure the data is safe even on disk.
Spinning media: hard disks

Read/write head: Physically moves across the disk to access different tracks.

Rotational latency: How long does it take the disk to rotate around to get the data under the head?

Seek time: How long does it take the head to move to the right track?

Tracks: Data is stored in rings. Within a ring data is divided into sectors of 512 or 4096 bytes.

PlaQer: Spins (5400rpm, 7200rpm, 15000rpm) around to move the data under the head.

Tracks: Data is moved in rings. Within a ring data is divided into sectors of 512 or 4096 bytes.

Tracks: Data is stored in rings. Within a ring data is divided into sectors of 512 or 4096 bytes.

Spinning media: hard disks

Disks: what do you care about?

- HPC and media: throughput (total data per second)
- Database and web: latency (time to first data)
- Desktop: mixture
- Cell phone: ?

Different media for different purposes

<table>
<thead>
<tr>
<th>Type</th>
<th>Capacity</th>
<th>Longevity</th>
<th>Latency</th>
<th>Throughput</th>
<th>Power</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Disk</td>
<td>3TB</td>
<td>3-5 years</td>
<td>8 ms</td>
<td>200 MB/s</td>
<td></td>
<td>$0.07/GB</td>
</tr>
<tr>
<td>NVRAM</td>
<td>256GB</td>
<td>10 years</td>
<td>85 µs</td>
<td>500 MB/s</td>
<td></td>
<td>$1.48/GB</td>
</tr>
<tr>
<td>DRAM</td>
<td>16GB</td>
<td>10 ms</td>
<td>65 ns</td>
<td>10,240 MB/s</td>
<td></td>
<td>$12.50/GB</td>
</tr>
</tbody>
</table>

What is the worst performance on the disk?

- 275x slower for small random pieces of data than for sequential ones.

Best Performance on the Disk

- Rota7onal latency: How long does it take the disk to rotate around to get the data under the head?
- Seek time: How long does it take the head to move to the right track?

Different media for different purposes

- Disk: what do you care about?
  - HPC and media: throughput (total data per second)
  - Database and web: latency (time to first data)
  - Desktop: mixture
  - Cell phone: ?

Storage prices

- NVRAM (Flash)

  • Problems
    - Must write in large chunks
      - 4096 bits at a time
      - Wear to matrix/parity write the whole chunk
    - Weary to read/write when to write
    - Wear-leveling spreads out the write to increase lifetime
    - Wear blocks enable replacement of defective blocks
    - Expensive (getting a bit cheaper…)
  
  • Benefits
    - Far faster (300MB/s vs. 200MB/s)
    - Far lower latency (ns vs. ms)
    - Far lower power and heat (no spinning piece of metal)
    - NVRAM will take over the world
      - When it gets cheap enough
      - But it may not be Flash: PCRAM, FeRAM, etc., may win...
Connecting devices

- Two main ways to connect devices
- Busses
  - Parallel: many bits at once (e.g., 32 bits together in one clock)
  - Used to be used everywhere
  - Still used inside chips
- Serial links
  - Serial: one bit at a time (e.g., 32 clock cycles to send 32 bits)
  - Used to be used only where distances were long (e.g., networks)
  - Now used for most off-chip communications

Busses

- Shared communications
  - Many devices "on" the bus
    - Run all wires to all devices
- Control
  - Can have one "master" who controls access to the bus
    - Can arbitrate among multiple devices for who gets control
- How they work:
  - Master gets an address and command (read/write) on the bus
  - Slave (based on address) prepares to receive/send data
  - Bus cycles (in parallel) master reads from one device
    - Data is on all wires but only one device has the address
  - Or, slave puts data on the bus and master reads it

What is a bus? (lots of wires)

What happens if signals are late?

The problem with busses

- Wires are ugly
  - And more as things get smaller
- They have capacitance, inductance, and resistance
  - Slows down
  - More devices on the bus, the slower
- Wires interfere with their neighbors
  - Noisy!
- It is very hard to keep 32 wires synchronized to the same clock!
Busses vs. point-to-point serial links

- **Off-chip busses are going away**
  - PCI-PCIe, ASI-SCIA, parallel ports → USB
- **Why?**
  - Data is too fast for long wires (wire = capacitor, time to charge voltage is exponential in the length of wire)
  - Too hard to handle multiple devices on the same wire (wire = complex capacitor/inductor/resistor, each devices complicates)
  - Too hard to keep wires synchronized (each wire is different and wires influence each other)
- **Replacement: SerDes**
  - Serial/Deserializer: sends bits one-at-a-time
  - No need to synchronize wires
  - Recover the clock (timing) as part of the signal
  - Need to run much faster to get same throughput
  - More complex, but transistors are “free”

Serial communications

- **Send one bit at a time**: Have to determine when each bit starts
- **Point-to-point**: Not shared
- **Can run much faster**: Don’t need to synchronize with other wires

**Examples**

- **Send**:
  0 1 0 1 1 0 1 0 1 1 0 1 0 1 0
  0 1 0 1 1 0 1 0 1 0 1 0 1 1 0

  Need to figure out the clock to figure out where to look!

Differential signaling

- **Differences**:
  - More complex, but transistors are “free”
  - Recover the clock (<ming) as part of the signal
  - No need to synchronize wires

Differential signaling to reject noise

- **If we use two wires we can make serial links much better**
  - One wire sends T+ the other sends T-
  - At the end we subtract them and get back the original signal

I/O interfaces today

- **Examples**
  - PCI is a point-to-point serial link
  - USB and Ethernet are external serial protocols
  - GPH/HyperTransport between CPUs is a point-to-point serial link
  - On a bus, but typically limited to 2 DIMMs per link

DMA must still be on a bus, but typically limited to 2 DIMMs per link

- **PCI express and SATA are point-to-point serial links**: One device per link.
Talking to I/O devices

How: I/O instructions

- Format: device# and command
  - Device# indicates the device
    - (e.g., keyboard input buffer is device #3)
  - Command
    - Sent to the I/O bus’s data lines
    - From registers or memory
- Examples from x86:
  - IN 5A, 33
  - OUT 33, 5A
- Problem:
  - Slow: one register at a time
  - Limited: few devices, limited commands

How: memory-mapped

- Map portions of the address space to I/O devices
- Read and write to those addresses to access the device
- No special instructions needed
  - Just need to know where your device is “mapped” into memory
  - Regular load and stores
  - Special addresses for commands and status
- What’s the problem?
  - Uses up some of our address space

Implementing memory-mapped I/O

How: direct memory access (DMA)

- Wouldn’t it be better if we had special hardware that can transfer data for us?
  - No need to read and write ourselves
  - No need to check status for every word
  - Direct Memory Access (DMA)
- Setup the hardware to do the transfer:
  - Which device
  - Which command
  - How much data
  - Where to store/get it
- Then let it do all the work:
  - (e.g., “load 4096 bytes of data from the device at address 128 and store it to address 384”)
  - DMA will execute 128 4-byte loads and write the data into addresses 384-1408.
- Everything is done this way these days because the overhead of managing it manually is too high.
When do we access devices?

The operating system needs to know when:
- The device needs attention (e.g., a network packet has arrived)
- The device has completed an operation (e.g., a disk write is done)
- The device has encountered an error (e.g., the wireless network has gone down)

Polling
- The device puts its status somewhere
- The OS repeatedly checks for it to change

Interrupt
- When the device is ready, it gets the processor’s attention by signaling an interrupt
- The OS then jumps to an interrupt handler to handle the event

Why interrupts?
- Polling is not efficient for active devices
  (And we ignored the cost of switching to the OS, which is 10,000+ cycles!)
- Polling: “Are you done yet? Are you done yet? Are you done yet?”
- Interrupt: “Tell me when you’re done. Until then I’m doing something else.”
- I/O interrupt is like an exception except:
  - Asynchronous (caused by an external I/O device)
  - Need more information than the cause: e.g., what does the device want?

Polling
- Example while (read_status(deviceID) == 1) {
  // wait }
- Why is this good?
  - Simple: the processor is completely in control
  - Easy to see what is happening
- Why is this bad?
  - CPU has to poll all the time to detect changes
  - Where do you put your code?

Polling overhead
- Example:
  - 2GHz processor/100 cycles per polling operation
  - 100 cycles to switch between OS and program (!)
- Mouse: polled 30 times per second
  - Clock cycles per second for polling: 30 x 100 = 3,000
  - Fraction of processor cycles: 3k/2B = 0.00015%
- Hard Disk: transfer rate 50MB/s
  - Transferred in 4-byte words
  - Need to poll (50MB/4) / 4 bytes per transfer = 12.5M times per second
  - Clock cycles for polling: 12.5M * 100 = 1.25B
  - Fraction of processor cycles: 1.25B/2B = 62.5%
Interrupt-driven I/O

- Example
  ```c
  while (!done) {
    calculate_next_answer();
  }
  handle_IO_interrupt() {
    ...
  }
  ```

- Why is this good?
  - User program only halted when there is a real need
  - Multiple interrupts can happen at the same time

- Why is this bad?
  - Special hardware needed to cause and detect the interrupts
  - Have to save processor state
  - Multiple interrupts can happen at the same time

Operating system responsibilities

- Three characteristics of the I/O System
  - Shared (by multiple programs)
  - Causes Interrupts (which must be handled by the OS because they access privileged resources)
  - Complex (drivers and state management are thoroughly obnoxious)

- The OS needs to hide this and provide:
  - Protection to shared resources (security)
  - Abstraction for accessing them (hide the dirty details)
  - Handle the interrupts (and respond correctly)
  - Share the resources “fairly” (whatever that means)
  - Schedule accesses for “best” performance
    (whatever that means: latency? throughput?)

Interrupt details

- Need to know which device generated the interrupt
  - Vectored interrupt (different interrupt address for every device)
  - Status register (check it when the interrupt occurs and choose address)

- Priority
  - Network and USB interrupt at the same time
  - Which one goes first? Need to prioritize

- Handling multiple interrupts at once
  - Pressing a key when the disk is reading
  - Two techniques:
    - Masking: disable interrupts while handling others
    - Nesting: take an interrupt within an interrupt

- What happens if you disable an interrupt and it occurs?
  - Nothing — you miss it...but where does the data go?

Example: ethernet

Ethernet

- Differential serial communication
- Shared medium
- Data divided into packets

How do we know the data wasn’t corrupted?

CRC Verifies the Packet
Ethernet: Collisions

- What happens if two devices transmit at the same time?
  - Detect the collision (read the data we transmit to see if it is corrupted by another packet)
  - Continue for 51.2µs (so everyone sees the collision and ignores the data)
  - Wait for a random (and increasing) period of time and start again
- Works really well

Summary

- Lots of I/O devices today
  - Speeds are constantly increasing
- Parallel busses are being replaced with serial links
  - Hard to keep long wires synchronized
  - Serial links are more complex, but we have the transistors
- I/O devices are mapped to memory addresses
  - Access them with regular read/write commands
- To reduce the overhead of I/O we
  - Use Direct Memory Access (DMA) to copy data in the background
  - Use interrupts to alert us when the DMA is done
- Ethernet works so well that everything else has died out