Input / Output Issues

- I/O organization
- Bus issue -
  - CPU-memory bus,
  - I/O bus
  - Bus width
  - A/D multiplex
  - Split transaction
  - Synchronous vs. asynchronous
- Interface to CPU
  - Polling device
  - Interrupt-driven I/O
  - Memory-mapped I/O
  - Direct memory access (DMA)
  - I/O processors
- Interaction with OS
  - state date
  - DMA vs. virtual memory
  - Disk caches
- Disk array - RAID
What is a bus?

A Bus Is:

- shared communication link
- single set of wires used to connect multiple subsystems

A Bus is also a fundamental tool for composing large, complex systems

- systematic means of abstraction

Advantages of Buses

- Versatility:
  - New devices can be added easily
  - Peripherals can be moved between computer systems that use the same bus standard

- Low Cost:
  - A single set of wires is shared in multiple ways
It creates a communication bottleneck

- The bandwidth of that bus can limit the maximum I/O throughput

The maximum bus speed is largely limited by:

- The length of the bus
- The number of devices on the bus
- The need to support a range of devices with:
  - Widely varying latencies
  - Widely varying data transfer rates

Control lines:

- Signal requests and acknowledgments
- Indicate what type of information is on the data lines

Data lines carry information between the source and the destination:

- Data and Addresses
- Complex commands
Master versus Slave

- A bus transaction includes two parts:
  - Issuing the command (and address) – request
  - Transferring the data – action

- Master is the one who starts the bus transaction by:
  - Issuing the command (and address)

- Slave is the one who responds to the address by:
  - Sending data to the master if the master asks for data
  - Receiving data from the master if the master wants to send data

Types of Busses

- Processor-Memory Bus (design specific)
  - Short and high speed
  - Only need to match the memory system
    - Maximize memory-to-processor bandwidth
  - Connects directly to the processor
  - Optimized for cache block transfers

- I/O Bus (industry standard)
  - Usually is lengthy and slower
  - Need to match a wide range of I/O devices
  - Connects to the processor-memory bus or backplane bus

- Backplane Bus (standard or proprietary)
  - Backplane: an interconnection structure within the chassis
  - Allow processors, memory, and I/O devices to coexist
A Two-Bus System

- **I/O buses tap into the processor-memory bus via bus adaptors:**
  - Processor-memory bus: mainly for processor-memory traffic
  - I/O buses: provide expansion slots for I/O devices

- **Apple Macintosh-II**
  - NuBus: Processor, memory, and a few selected I/O devices
  - SCCI Bus: the rest of the I/O devices
A Three-Bus System

- A small number of backplane buses tap into the processor-memory bus
  - Processor-memory bus is only used for processor-memory traffic
  - I/O buses are connected to the backplane bus
- Advantage: loading on the processor bus is greatly reduced

North/South Bridge architectures: separate busses

- Separate sets of pins for different functions
  - Memory bus
  - Caches
  - Graphics bus (for fast frame buffer)
  - I/O busses are connected to the backplane bus
- Advantage:
  - Busses can run at different speeds
  - Much less overall loading!
Synchronous and Asynchronous Bus

- **Synchronous Bus:**
  - Includes a clock in the control lines
  - A fixed protocol for communication that is relative to the clock
  - Advantage: involves very little logic and can run very fast
  - Disadvantages:
    - Every device on the bus must run at the same clock rate
    - To avoid clock skew, they cannot be long if they are fast

- **Asynchronous Bus:**
  - It is not clocked
  - It can accommodate a wide range of devices
  - It can be lengthened without worrying about clock skew
  - It requires a handshaking protocol

Busses so far

- **Master**
- **Slave**
- **Control Lines**
- **Address Lines**
- **Data Lines**

Bus Master: has ability to control the bus, initiates transaction

Bus Slave: module activated by the transaction

Bus Communication Protocol: specification of sequence of events and timing requirements in transferring information.

- Asynchronous Bus Transfers: control lines (req, ack) serve to orchestrate sequencing.
- Synchronous Bus Transfers: sequence relative to common clock.
Arbitration: Obtaining Access to the Bus

One of the most important issues in bus design:

- How is the bus reserved by a device that wishes to use it?

Chaos is avoided by a master-slave arrangement:

- Only the bus master can control access to the bus:
  - It initiates and controls all bus requests
- A slave responds to read and write requests

The simplest system:

- Processor is the only bus master
- All bus requests must be controlled by the processor
- Major drawback: the processor is involved in every transaction

Multiple Potential Bus Masters: Need for Arbitration

Bus arbitration scheme:

- A bus master wanting to use the bus asserts the bus request
- A bus master cannot use the bus until its request is granted
- A bus master must signal to the arbiter the end of the bus utilization

Bus arbitration schemes usually try to balance two factors:

- Bus priority: the highest priority device should be serviced first
- Fairness: Even the lowest priority device should never be completely locked out from the bus

Bus arbitration schemes can be divided into four broad classes:

- Daisy chain arbitration
- Centralized, parallel arbitration
- Distributed arbitration by self-selection: each device wanting the bus places a code indicating its identity on the bus.
- Distributed arbitration by collision detection: Each device just “goes for it”. Problems found after the fact.
The Daisy Chain Bus Arbitrations Scheme

Advantage: simple

Disadvantages:
- Cannot assure fairness:
  A low-priority device may be locked out indefinitely
- The use of the daisy chain grant signal also limits the bus speed

Used in essentially all processor-memory busses and in high-speed I/O busses
Even memory busses are more complex than this:
- Memory (slave) may take time to respond
- It may need to control data rate

Typical Synchronous Protocol

- Slave indicates when it is prepared for data xfer
- Actual transfer goes at bus rate
1993 Backplane/IO Bus Survey

<table>
<thead>
<tr>
<th>Bus</th>
<th>SBus</th>
<th>TurboChannel</th>
<th>MicroChannel</th>
<th>PCI</th>
</tr>
</thead>
<tbody>
<tr>
<td>Originator</td>
<td>Sun</td>
<td>DEC</td>
<td>IBM</td>
<td>Intel</td>
</tr>
<tr>
<td>Clock Rate (MHz)</td>
<td>16-25</td>
<td>12.5-25</td>
<td>async</td>
<td>33</td>
</tr>
<tr>
<td>Addressing</td>
<td>Virtual</td>
<td>Physical</td>
<td>Physical</td>
<td>Physical</td>
</tr>
<tr>
<td>Data Sizes (bits)</td>
<td>8,16,32</td>
<td>8,16,24,32</td>
<td>8,16,24,32,64</td>
<td>8,16,24,32,64</td>
</tr>
<tr>
<td>Master</td>
<td>Multi</td>
<td>Single</td>
<td>Multi</td>
<td>Multi</td>
</tr>
<tr>
<td>Arbitration</td>
<td>Central</td>
<td>Central</td>
<td>Central</td>
<td>Central</td>
</tr>
<tr>
<td>32 bit read (MB/s)</td>
<td>33</td>
<td>25</td>
<td>20</td>
<td>33</td>
</tr>
<tr>
<td>Peak (MB/s)</td>
<td>89</td>
<td>84</td>
<td>75</td>
<td>111 (222)</td>
</tr>
<tr>
<td>Max Power (W)</td>
<td>16</td>
<td>26</td>
<td>13</td>
<td>25</td>
</tr>
</tbody>
</table>

Processor Interface Issues

- Processor interface
  - Interrupts
  - Memory mapped I/O

- I/O Control Structures
  - Polling
  - Interrupts
  - DMA
  - I/O Controllers
  - I/O Processors

- Capacity, Access Time, Bandwidth

- Interconnections
  - Busses
### I/O Interface

- **Independent I/O Bus**
  - CPU
  - Memory
  - Interface
  - Peripheral

- Separate I/O instructions (in, out)
  - Lines distinguish between I/O and memory transfers
    - VME bus
    - Multibus-II
    - Nubus

- Optimistically 40 Mbytes/sec
- Completely 10 MIP processor completely saturates the bus!

### Memory Mapped I/O

- **Single Memory & I/O Bus**
  - No Separate I/O Instructions
  - CPU
  - Memory
  - Interface
  - Interface
  - Peripheral
  - Peripheral

- ROM
- RAM
- I/O
**Interrupt Driven Data Transfer**

1. **I/O interrupt**
2. Save PC
3. Interrupt service addr
4. Rti

User program progress only halted during actual transfer

1000 transfers at 1 ms each:
- 1000 interrupts @ 2 µsec per interrupt
- 1000 interrupt service @ 98 µsec each = 0.1 CPU seconds

Device xfer rate = 10 MBytes/sec >= 0.1 x 10^{-6} sec/byte => 0.1 µsec/byte
=> 1000 bytes = 100 µsec
1000 transfers x 100 µsecs = 100 ms = 0.1 CPU seconds

Still far from device transfer rate! 1/2 in interrupt overhead

---

**Programmed I/O (Polling)**

Is the data ready? yes read data store data done?
- no

busy wait loop not an efficient way to use the CPU unless the device is very fast!

but checks for I/O completion can be dispersed among computationally intensive code
Delegating I/O Responsibility from the CPU: IOP

1. Issues instruction to IOP
2. CPU
   - Device Address
     - Target device
     - Where commands are
     - What to do
     - Where to put data
     - How much
     - Special requests
3. IOP looks in memory for commands
4. IOP interrupts CPU when done

IOP steals memory cycles.

Delegating I/O Responsibility from the CPU: DMA

- Direct Memory Access (DMA):
  - External to the CPU
  - Act as a master on the bus
  - Transfers blocks of data to or from memory without CPU intervention

CPU sends a starting address, direction, and length count to DMAC. Then issues "start".

DMAC provides handshake signals for Peripheral Controller, and Memory Addresses and handshake signals for Memory.
**Direct Memory Access**

Time to do 1000 xfers at 1 msec each:
- 1 DMA set-up sequence @ 50 µsec
- 1 interrupt @ 2 µsec
- 1 interrupt service sequence @ 48 µsec

.0001 second of CPU time

CPU sends a starting address, direction, and length count to DMAC. Then issues "start".

DMAC provides handshake signals for Peripheral Controller, and Memory Addresses and handshake signals for Memory.

---

**Redundant Arrays of (Inexpensive) Disks**

- Files are "striped" across multiple disks
- Redundancy yields high data availability
  - Availability: service still provided to user, even if some components failed
  - Disks will still fail
- Contents reconstructed from data redundantly stored in the array
  - Capacity penalty to store redundant info
  - Bandwidth penalty to update redundant info

**Techniques:**
- Mirroring/Shadowing (high capacity cost)
- Horizontal Hamming Codes (overkill)
- Parity & Reed-Solomon Codes
- Failure Prediction (no capacity overhead!)
- VaxSimPlus — Technique is controversial
RAID 1: Disk Mirroring/Shadowing

- Each disk is fully duplicated onto its "shadow"
  Very high availability can be achieved
- Bandwidth sacrifice on write:
  Logical write = two physical writes
- Reads may be optimized
- Most expensive solution: 100% capacity overhead

Targeted for high I/O rate, high availability environments

RAID 3: Parity Disk

- Parity computed across recovery group to protect against hard disk failures
  33% capacity cost for parity in this configuration
  wider arrays reduce capacity costs, decrease expected availability, increase reconstruction time
- Arms logically synchronized, spindles rotationally synchronized logically a single high capacity, high transfer rate disk

Targeted for high bandwidth applications: Scientific, Image Processing
Inspiration for RAID 4

- RAID 3 relies on parity disk to discover errors on Read
- But every sector has an error detection field
- Rely on error detection field to catch errors on read, not on the parity disk
- Allows independent reads to different disks simultaneously

Redundant Arrays of Inexpensive Disks RAID 4: High I/O Rate Parity

Example:
- small read D0 & D5
- large write D12-D15

Increasing Logical Disk Address

Stripe

Disk Columns

Insides of 5 disks
**Inspiration for RAID 5**

- RAID 4 works well for small reads

- Small writes (write to one disk):
  - Option 1: read other data disks, create new sum and write to Parity Disk
  - Option 2: since P has old sum, compare old data to new data, add the difference to P

- Small writes are limited by Parity Disk: Write to D0, D5 both also write to P disk

**Redundant Arrays of Inexpensive Disks RAID 5:**

- High I/O Rate Interleaved Parity

- Independent writes possible because of interleaved parity

- Example: write to D0, D5 uses disks 0, 1, 3, 4
**Problems of Disk Arrays: Small Writes**

**RAID-5: Small Write Algorithm**
1 Logical Write = 2 Physical Reads + 2 Physical Writes

![Diagram of RAID-5 small write algorithm]

**Subsystem Organization**

- Host manages interface to host, DMA, control, buffering, parity logic, physical device control.
- Striping software off-loaded from host to array controller, no applications modifications, no reduction of host performance.

- Single board disk controller often piggy-backed in small format devices.

![Diagram of subsystem organization]
System Availability: Orthogonal RAID Arrays

- **Data Recovery Group**: unit of data redundancy
- **Redundant Support Components**: fans, power supplies, controller, cables
- **End to End Data Integrity**: internal parity protected data paths

System-Level Availability

- **Fully dual redundant**: with duplicated paths, higher performance can be obtained when there are no failures
- **Goal**: No Single Points of Failure
Summary: RAID Techniques: Goal was performance, popularity due to reliability of storage

- **Disk Mirroring, Shadowing (RAID 1)**
  Each disk is fully duplicated onto its "shadow"
  Logical write = two physical writes
  100% capacity overhead

- **Parity Data Bandwidth Array (RAID 3)**
  Parity computed horizontally
  Logically a single high data bw disk

- **High I/O Rate Parity Array (RAID 5)**
  Interleaved parity blocks
  Independent reads and writes
  Logical write = 2 reads + 2 writes

Network Attached Storage (NAS)

*Decreasing Disk Diameters*

<table>
<thead>
<tr>
<th>Diameter</th>
<th>Bandwidth</th>
</tr>
</thead>
<tbody>
<tr>
<td>14&quot;</td>
<td>3 Mb/s</td>
</tr>
<tr>
<td>10&quot;</td>
<td>10 Mb/s</td>
</tr>
<tr>
<td>8&quot;</td>
<td>50 Mb/s</td>
</tr>
<tr>
<td>5.25&quot;</td>
<td>100 Mb/s</td>
</tr>
<tr>
<td>3.5&quot;</td>
<td>1 Gb/s</td>
</tr>
<tr>
<td>2.5&quot;</td>
<td>10 Gb/s</td>
</tr>
</tbody>
</table>

*Increasing Network Bandwidth*

Network provides well defined physical and logical interfaces: separate CPU and storage system!

High Performance Storage Service on a High Speed Network

Network File Services

OS structures supporting remote file access

networks capable of sustaining high bandwidth transfers
**Bus Summary**

- **Buses are important for building large-scale systems**
  - Their speed is critically dependent on factors such as length, number of devices, etc.
  - Critically limited by capacitance
  - Tricks: esoteric drive technology such as GTL

- **Important terminology:**
  - Master: The device that can initiate new transactions
  - Slaves: Devices that respond to the master

- **Two types of bus timing:**
  - Synchronous: bus includes clock
  - Asynchronous: no clock, just REQ/ACK strobing

- **Direct Memory Access (DMA) allows fast, burst transfer into processor’s memory:**
  - Processor’s memory acts like a slave
  - Probably requires some form of cache-coherence so that DMA’ed memory can be invalidated from cache.

---

**I/O Summary:**

- **I/O performance limited by weakest link in chain between OS and device**

- **Three Components of Disk Access Time:**
  - Seek Time: advertised to be 8 to 12 ms. May be lower in real life.
  - Rotational Latency: 4.1 ms at 7200 RPM and 8.3 ms at 3600 RPM
  - Transfer Time: 2 to 12 MB per second

- **I/O device notifying the operating system:**
  - Polling: it can waste a lot of processor time
  - I/O interrupt: similar to exception except it is asynchronous

- **Delegating I/O responsibility from the CPU: DMA, or even IOP**