What is a bus?
- Slow vehicle that many people ride together
  - well, true...
- A bunch of wires...

A Bus is:
- a shared communication link
- a 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 reused between computer systems that use the same bus standard
- Low Cost:
  - A single set of wires is shared in multiple ways
- Manage complexity by partitioning the design

Disadvantage of Buses
- 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

The General Organization of a Bus
- 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
  - 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 Buses

- Processor-Memory Bus (design specific)
  - Short and high speed
  - Only need to match the memory system
  - Maximizes memory-to-processor bandwidth
  - Connects directly to the processor
  - Optimized for cache block transfers
- Backplane Bus (standard or proprietary)
  - Backplane: an interconnection structure within the chassis
  - Allows processors, memory, and I/O devices to coexist
  - Cost advantage: one bus for all components
- I/O Bus (industry standard)
  - Usually lengthy and slower
  - Need to match a wide range of I/O devices
  - Connects to the processor-memory bus or backplane bus

Example: Bus Architecture

- Processor/Memory Bus
- Backplane Bus
- I/O Buses

Example: Motherboard Abit BH6

A Computer System with One Bus: Backplane Bus

- A single bus (the backplane bus) is used for:
  - Processor-to-memory communication
  - Communication between I/O devices and memory
- Advantages: Simple and low cost
- Disadvantages: slow and the bus can become a major bottleneck
- Example: IBM PC - AT

A Two-Bus System

- I/O buses tap into the processor-memory bus via bus adaptors:
  - Processor-memory bus: main I/O devices
  - I/O buses: provide expansion slots for I/O devices
- Apple Macintosh II
  - NuBus: Processor, memory, and a few selected I/O devices
  - SCSI Bus: the rest of the I/O devices

A Three-Bus System

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

- This bus connects the CPU to RAM
  - Designed for maximal bandwidth
  - Usually wide, 32 bits or more
  - To further increase bandwidth we use a Cache
  - Burst access between cache and memory, early restart

Memory Bus

- Designed for maximal bandwidth
- Usually wide, 32 bits or more
- To further increase bandwidth we use a Cache
- Burst access between cache and memory, early restart

On Chip Cache (1st level)

- Using separate Instruction and Data caches, we can read a Hit simultaneously for both Instruction and Data
- Pros
  - Simple (one FSM)
  - Fast
- Cons
  - Clock skew limits bus length
  - All devices work on the same speed (clock)

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

Synchronous Bus

- A single clock controls the protocol
  - Pros
    - Simple (one FSM)
    - Fast
  - Cons
    - Clock skew limits bus length
    - All devices work on the same speed (clock)
  - Suitable for Processor-Memory Bus
Asynchronous Bus

- Uses a handshaking to implement a transaction protocol
  - Pros
    - Versatile, generic protocols
    - Dynamic data rate
  - Cons
    - More complex, two communication FSMs
    - Slower, but usually can be made quite fast

Asynchronous Protocol

- Cons
  - More complex, two communication FSMs
  - Slower, but usually can be made quite fast

Busses so far

- 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.

Bus Transaction

- Arbitration
- Request
- Action

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

Bus Arbitration

- Bus Master, (initiator usually the CPU)
- Slave, (usually the Memory)

Arbitration signals
- BusRequest
- BusGrant
- BusPriority
  - Higher priority served first
  - Fairness, no request is locked out
Multiple Potential Bus Masters: the 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 after finish using the bus

- 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

- Daisy chain arbitration
  - Grant line runs through all device, highest priority device first.
  - Single device with all request lines.

- Centralized
  - Many request/grant lines
  - Complex controller, may be a bottleneck

- Self Selection
  - Many request lines
  - The one with highest priority self decides to take bus

- Collision Detection (Ethernet)
  - One request line
  - Try to access bus,
  - If collision device backoff
  - Try again in random + exponential time

The Daisy Chain Bus Arbitration 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

Centralized Parallel Arbitration

- Used in essentially all processor-memory busses and in high-speed I/O busses

Simplest bus paradigm

- All agents operate synchronously
- All can source / sink data at same rate
- => simple protocol
  - Just manage the source and target

Simple Synchronous Protocol

- Even memory busses are more complex than this
  - memory (slave) may take time to respond
  - It need to control data rate
Typical Synchronous Protocol
- Slave indicates when it is prepared for data xfer
- Actual transfer goes at bus rate

Increasing the Bus Bandwidth
- Separate versus multiplexed address and data lines:
  - Address and data can be transmitted in one bus cycle if separate address and data lines are available
  - Cost: (a) more bus lines, (b) increased complexity
- Data bus width:
  - By increasing the width of the data bus, transfers of multiple words require fewer bus cycles
  - Example: SPARCstation 20's memory bus is 128 bit wide
  - Cost: more bus lines
- Block transfers:
  - Allow the bus to transfer multiple words in back-to-back bus cycles
  - Only one address needs to be sent at the beginning
  - The bus is not released until the last word is transferred
  - Cost: (a) increased complexity
    - (b) increased response time (latency) for request

Increasing Transaction Rate on Multimaster Bus
- Overlapped arbitration
  - Perform arbitration for next transaction during current transaction
- Bus parking
  - Master can hold onto bus and perform multiple transactions as long as no other master makes request
- Overlapped address/data phases (prev. slide)
  - Requires one of the above techniques
- Split-phase transaction bus (Command queuing)
  - Completely separate address and data phases
  - Arbitrate separately for each
  - Address phase yields a tag which is matched with data phase
  - "All of the above" in most modern mem busses

Split Transaction Protocol
- Designed to support wide variety of devices
- Allow data rate match between arbitrary speed devices
  - Fast processor - slow I/O
  - Slow processor - fast I/O

The I/O Bus Problem
- Designed to support wide variety of devices
- Allow data rate mismatch between arbitrary speed devices
  - Fast processor - slow I/O
  - Slow processor - fast I/O
**High Speed I/O Bus**

- **Examples**
  - Raid controllers
  - Graphics adapter
- **Limited number of devices**
- **Data transfer bursts at full rate**
- **DMA transfers important**
  - Small controller apologise stream of bytes to or from memory
- **Either side may need to stall transfer**
  - Buffers fill up

**Backplane Bus**

Virtual Address

<table>
<thead>
<tr>
<th>CPU</th>
<th>DB DE EX DM</th>
<th>End Reset Block</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>End Reset Block</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TLB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>HD Interface</th>
</tr>
</thead>
<tbody>
<tr>
<td>Serial Interface</td>
</tr>
<tr>
<td>Graphics Adapter</td>
</tr>
<tr>
<td>Sound Card</td>
</tr>
</tbody>
</table>

**Direct Memory Access (DMA)**

1. We want to move data from HD interface to RAM
2. Why go all the way to the CPU (1) and back to the memory bus (2)

**Direct Memory Access**

- **DMA Processor**
  - 1) Generates Bus Request, waits for Grant
  - 2) Put Address & Data on Bus
  - 3) Increase Address, back to 2 until finished
  - 4) Release Bus
- **Generates Interrupt only**
  - When finished
  - If an error occurred

**DMA and Virtual Memory**

- **If DMA uses Virtual address it needs to pass a TLB**
  - Go through the CPU’s TLB (no good)
  - A TLB in the DMA processor (needs updating)
- **If DMA uses Physical address**
  - Only transfer within one Page
  - We give the DMA a set of Physical addresses
    - (local TLB copy)

**DMA and Caching**

- **INCONSISTENCY problem**
  - We change the RAM contents, but not the cache
  - We write to HD but the RAM holds “old” information
- **Routing All DMA though CPU**
  - No good, spoils the idea
- **Software handling of DMA**
  - Cache: Flush selected cache lines to RAM
  - Cache: invalidate selected cache lines
  - TLB OS: Do not allow access to these pages until DMA finished
- **Hardware Cache Coherence Protocol**
  - Complex, but very useful for multi processor systems
**PCI Read/Write Transactions**

- All signals sampled on rising edge
- Centralized Parallel Arbitration
- overlapped with previous transaction
- All transfers are (unlimited) bursts
- Address phase starts by asserting FRAME#
- Next cycle “initiator” asserts cmd and address
- Data transfers happen on when
  - RD/Y asserted by master/initiator when ready to transfer data
  - TRDY asserted by target when ready to transfer data
- transfer when both asserted on rising edge
- FRAME deasserted when master intends to complete only one more data transfer

---

**PCI Read Transaction**

---

**PCI Write Transaction**

---

**PCI Optimizations**

- Push bus efficiency toward 100% under common simple usage
  - like RISC
- Bus Parking
  - retain bus grant for previous master until another makes request
  - granted master can start next transfer without arbitration
- Arbitrary Burst length
  - initiator and target can assert flow control with xRDY
  - target can disconnect request with STOP (abort or retry)
  - master can disconnect by deasserting FRAME
  - arbiter can disconnect by deasserting GNT
- Delayed (ganged, split-phase) transactions
  - free the bus after request to slow device

---

**Additional PCI Issues**

- Interrupts: support for controlling I/O devices
- Cache coherence:
  - support for I/O and multiprocessors
- Locks:
  - support time-sharing, I/O, and MP
- Configuration Address Space

---

**Summary of Bus Options**

<table>
<thead>
<tr>
<th>Option</th>
<th>High performance</th>
<th>Low cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Address</td>
<td>Multiple address &amp; data lines</td>
<td>Multiple address &amp; data lines</td>
</tr>
<tr>
<td>Data</td>
<td>Wider is faster (e.g., 32 bits)</td>
<td>Narrower is cheaper (e.g., 8 bits)</td>
</tr>
<tr>
<td>Transfer size</td>
<td>Multiple words has less bus overhead</td>
<td>Single-word transfer is simpler</td>
</tr>
<tr>
<td>Bus masters</td>
<td>Multiple (requires arbitration)</td>
<td>Single master (no arbitration)</td>
</tr>
<tr>
<td>Clocking</td>
<td>Synchronous</td>
<td>Asynchronous</td>
</tr>
<tr>
<td>Protocol</td>
<td>pipelined</td>
<td>Serial</td>
</tr>
</tbody>
</table>