Interconnect & Communication

Space, Time, & stuff…

What is the big deal with these things?

I don’t see what is so exciting about the “back-side” either.

Lab & due Tomorrow (Wednesday)!
What's the most important part of this picture?

- Linux
- Mother boards
- Flash Memory
- SDRAM
- Hard Disk Drives
- Windows XP
- LAN technology
- App Servers
- ActiveX Controls
- Graphics Acceleration
Technology comes & goes; interfaces last forever

Interfaces typically deserve more engineering attention than the technologies they interface…

- **Abstraction**: should outlast many technology generations
- **Often “virtualized”**: to extend beyond original function (e.g. memory, I/O, services, machines)
- **Represent more potential value** to their proprietors than the technologies they connect.

**Interface sob stories:**

- **Interface “warts”**: Windows “aux.c” bug, Big/little Endian wars
- **IBM PC debacle**

... and many success stories:

- **IBM 360** Instruction set architecture; Postscript; Compact Flash; ...
- **Backplane buses**
System Interfaces & Modularity

Ancient Times (Ad hoc connections)
- MEM
- MEM
- CPU
- DISK
- I/O

80s (Processor-independent Bus)
- CPU
- DISK
- MEM
- I/O

Late 60s (Processor-dependent Bus)
- CPU
- DISK
- MEM
- MEM
- I/O

Back-side bus
- Graphics
- I/O

“AGP” bus
- CPU
- MEM
- MEM
- L2 $

Front-side bus (PCI and EISA)
- DISK
- I/O

Today
- Buses Galore
- I/O

?
Interface Standard: Backplane Bus

Modular cards that plug into a common backplane:
- CPUs
- Memories
- Bulk storage
- I/O devices
- S/W?

The backplane provides:
- Power
- Common system clock
- Wires for communication

Printed Circuit Cards

BUS LINES

ADDRESS

DATA

OPERATION

START

FINISH

CLOCK

MODULE LOGIC
The Dumb Bus: ISA & EISA

Original primitive approach --

Just take the control signals and data bus from the CPU module, buffer it, and call it a bus.

ISA bus (Original IBM PC bus) -

Pin out and timing is nearly identical to the 8088 spec.

<table>
<thead>
<tr>
<th>Pin</th>
<th>Signal</th>
<th>Pin</th>
<th>Signal</th>
</tr>
</thead>
<tbody>
<tr>
<td>B1</td>
<td>Ground</td>
<td>A1</td>
<td>I/O Channel Check</td>
</tr>
<tr>
<td>B2</td>
<td>Reset Driver</td>
<td>A2</td>
<td>Data 7</td>
</tr>
<tr>
<td>B3</td>
<td>+5VDC</td>
<td>A3</td>
<td>Data 6</td>
</tr>
<tr>
<td>B4</td>
<td>Interrupt Request 9</td>
<td>A4</td>
<td>Data 5</td>
</tr>
<tr>
<td>B5</td>
<td>-VDC</td>
<td>A5</td>
<td>Data 4</td>
</tr>
<tr>
<td>B6</td>
<td>DMA Request 2</td>
<td>A6</td>
<td>Data 3</td>
</tr>
<tr>
<td>B7</td>
<td>-12VDC</td>
<td>A7</td>
<td>Data 2</td>
</tr>
<tr>
<td>B8</td>
<td>Zero Wait State</td>
<td>A8</td>
<td>Data 1</td>
</tr>
<tr>
<td>B9</td>
<td>+12VDC</td>
<td>A9</td>
<td>Data 0</td>
</tr>
<tr>
<td>B10</td>
<td>Ground</td>
<td>A10</td>
<td>I/O Channel Ready</td>
</tr>
<tr>
<td>B11</td>
<td>Real Memory Write</td>
<td>A11</td>
<td>Address Enable</td>
</tr>
<tr>
<td>B12</td>
<td>Real Memory Read</td>
<td>A12</td>
<td>Address 19</td>
</tr>
<tr>
<td>B13</td>
<td>Input/Output Write</td>
<td>A13</td>
<td>Address 18</td>
</tr>
<tr>
<td>B14</td>
<td>Input/Output Read</td>
<td>A14</td>
<td>Address 17</td>
</tr>
<tr>
<td>B15</td>
<td>DMA Acknowledge 3</td>
<td>A15</td>
<td>Address 16</td>
</tr>
<tr>
<td>B16</td>
<td>DMA Request 3</td>
<td>A16</td>
<td>Address 15</td>
</tr>
<tr>
<td>B17</td>
<td>DMA Acknowledge 1</td>
<td>A17</td>
<td>Address 14</td>
</tr>
<tr>
<td>B18</td>
<td>Refresh</td>
<td>A18</td>
<td>Address 13</td>
</tr>
<tr>
<td>B19</td>
<td></td>
<td>A19</td>
<td>Address 12</td>
</tr>
<tr>
<td>B20</td>
<td>Clock</td>
<td>A20</td>
<td>Address 11</td>
</tr>
<tr>
<td>B21</td>
<td>Interrupt Request 7</td>
<td>A21</td>
<td>Address 10</td>
</tr>
<tr>
<td>B22</td>
<td>Interrupt Request 6</td>
<td>A22</td>
<td>Address 9</td>
</tr>
<tr>
<td>B23</td>
<td>Interrupt Request 5</td>
<td>A23</td>
<td>Address 8</td>
</tr>
<tr>
<td>B24</td>
<td>Interrupt Request 4</td>
<td>A24</td>
<td>Address 7</td>
</tr>
<tr>
<td>B25</td>
<td>Interrupt Request 3</td>
<td>A25</td>
<td>Address 6</td>
</tr>
<tr>
<td>B26</td>
<td>DMA Acknowledge 2</td>
<td>A26</td>
<td>Address 5</td>
</tr>
<tr>
<td>B27</td>
<td>Terminal Count</td>
<td>A27</td>
<td>Address 4</td>
</tr>
<tr>
<td>B28</td>
<td>Address Latch Enable</td>
<td>A28</td>
<td>Address 3</td>
</tr>
<tr>
<td>B29</td>
<td>+5VDC</td>
<td>A29</td>
<td>Address 2</td>
</tr>
<tr>
<td>B30</td>
<td>Oscillator</td>
<td>A30</td>
<td>Address 1</td>
</tr>
<tr>
<td>B31</td>
<td>Ground</td>
<td>A31</td>
<td>Address 0</td>
</tr>
</tbody>
</table>
Smarter “Processor Independent” Buses

**NuBus, PCI…**

Isolate basic communication primitives from processor architecture:

- Simple read/write protocols
- Symmetric: any module can become “Master” (smart I/O, multiple processors, etc)
- Support for “plug & play” expansion

**Goal:** vendor-independent interface standard

**TERMINOLOGY –**

**BUS MASTER** – a module that initiates a bus transaction. (CPU, disk controller, etc.)

**BUS SLAVE** – a module that responds to a bus request. (Memory, I/O device, etc.)

**BUS CYCLE** – The period from when a transaction is requested until it is served.

I've been waiting here for hours and I still haven't seen a bus cycle go by yet!
Buses, Interconnect… what’s the big deal?

Aren’t buses simply logic circuits with long wires?

Wires: circuit theorist’s view:
- Equipotential “nodes” of a circuit.
- Instant propagation of v, i over entire node.
- “space” abstracted out of design model.
- Time issues dictated by RLC elements; wires are timeless.

Wires: interconnect engineer’s view:
- Transmission lines.
- Finite signal propagation velocity.
- Space matters.
- Time matters.
- Reality matters.
Bus Lines as Transmission Lines

ANALOG ISSUES:

- **Propagation times**
  - Light travels about 1 ft / ns (about 7”/ns in a wire)

- **Skew**
  - Different points along the bus see the signals at different times

- **Reflections & standing waves**
  - At each interface (places where the propagation medium changes) the signal may reflect if the impedances are not matched.
  - Make a transition on a long line – may have to wait many transition times for echoes to subside.
Coping with Analog Issues...

We’d like our bus to be technology independent...

- **Self-timed** protocols allow bus transactions to accommodate varying response times;
- **Asynchronous** protocols avoid the need to pick a (technology-dependent) clock frequency.

BUT... asynchronous protocols are vulnerable to analog-domain problems, like the infamous

**WIRED-OR GLITCH:** what happens when a switch is opened???

**COMMON COMPROMISE:** Synchronous, Self-Timed protocols

- Broadcast bus clock
- Signals sampled at “safe” times
* DEAL WITH: noise, clock skew (wrt signals)
Synchronous Bus Clock Timing

Allow for several “round-trip” bus delays so that ringing can die down.
A Simple Bus Transaction

**MASTER:**
1) Chooses bus operation
2) Asserts an address
3) Waits for a slave to answer.

**SLAVE:**
1) Monitors start
2) Check address
3) If meant for me
   a) look at bus operation
   b) do operation
   c) signal finish of cycle

**BUS:**
1) Monitors start
2) Start count down
3) If no one answers before counter reaches 0 then “time out”
Multiplexed Bus: Write Transaction

More efficient use of shared wires

We let the address and data buses share the same wires.

Slave sends a status message by driving the operation control signals when it finishes. Possible indications:
- request succeeded
- request failed
- try again

A slave can stall the write by waiting several cycles before asserting the finish signal.
Multiplexed Bus: Read Transaction

On reads, we allot one cycle for the bus to “turn around” (stop driving and begin receiving). It generally takes some time to read data anyway.

A slave can stall the read (for instance if the device is slow compared to the bus clock) by waiting several clocks before asserting the finish signal. These delays are sometimes called “WAIT-STATES”.

Throughput: 3+ Clocks/word
Block transfers are the way to get peak performance from a bus. A throughput of nearly 1 Clock/word is achievable on large blocks. **Slaves must generate sequential addresses.**
Block read transfers still require at least one cycle to turn-around the bus. More WAIT-STATES can be added if initial latency is high. The throughput is nearly 1 Clock/word on large blocks. Great for reading long cache lines!
Split-Transaction Bus Operation

… you knew we’d work pipelining in somehow!

The bus master can post several read requests before the first request is served.

Generally, accesses are served in the same order that they are requested.

Slaves must queue up multiple requests, until master releases bus.

The master must keep track of outstanding requests and their status.

Throughput: 2 Clocks/word, independent of read latency
Bus Arbitration: Multiple Bus Masters

**ISSUES:**

- **Fairness** - Given uniform requests, bus cycles should be divided evenly among modules (to each, according to their needs...)
- **Bounded Wait** - An upper bound on how long a module has to wait between requesting and receiving a grant
- **Utilization** - Arbitration scheme should allow for maximum bus performance
- **Scalability** - Fixed-cost per module (both in terms of arbitration H/W and arbitration time.)

**STATE OF THE ART ARBITRATION:** N masters, log N time, log N wires.
Outside the box…
The Network as an interface standard

ETHERNET: In the mid-70’s Bob Metcalf (at Xerox PARC, an MIT alum) devised a bus for networking computers together.

- Bit-serial (optimized for long wires)
- Asynchronous (no clock distribution)
- Variable-length “packets”

EMERGING IDEA: Protocol “stacks” that isolate application-level interface from low-level physical devices:

```
+---+    +---+     +---+
| Application | Session |
+---+    +---+     +---+
| Transport  | Network |
| TCP       | IP      |
| Physical  | Token Ring |
+---+    +---+     +---+
```

- EMERGING IDEA: Protocol “stacks” that isolate application-level interface from low-level physical devices.
Generalizing Buses...

Communication Topologies

1-dimensional approaches:

“Low cost networks” – constant cost/node

**BUS**

ONE step for random message delivery (but only one message at a time)

**RING**

$\Theta(n)$ steps for random message delivery
Quadratic-cost Topologies

COMPLETE GRAPH:

Dedicated lines connecting each pair of communicating nodes. $\Theta(n)$ simultaneous communications.

CROSSBAR SWITCH:

- Switch dedicated between each pair of nodes
- Each $A_i$ can be connected to one $B_j$ at any time
- Special cases:
  - $A = \text{processors}, B = \text{memories}$
  - $A, B$ same type of node
  - $A, B$ same nodes (complete graph)
Mesh Topologies

Nearest-neighbor connectivity:
Point-to-point interconnect
  - minimizes delays
  - minimizes “analog” effects
Store-and-forward
  (some overhead associated with communication routing)
Logarithmic Latency Networks

HYPERCUBE (n-cube):
Cost = $\Theta(n \log n)$
Worst-case path length = $\Theta(\log n)$

BINARY TREE:
Maximum path length is $\Theta(\log n)$ steps;
Cost/node constant.
Communication Topologies: Latency

Theorist's view:

- Each point-to-point link requires one hardware unit.
- Each point-to-point communication requires one time unit.

<table>
<thead>
<tr>
<th>Topology</th>
<th>$\Theta (n^2)$</th>
<th>Theoretical Latency</th>
<th>Actual Latency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Complete Graph</td>
<td>$\Theta (n^2)$</td>
<td>$\Theta (1)$</td>
<td>$\geq \Theta (\sqrt[3]{n})$</td>
</tr>
<tr>
<td>Crossbar</td>
<td>$\Theta (n^2)$</td>
<td>$\Theta (1)$</td>
<td>$\Theta (n)$</td>
</tr>
<tr>
<td>1D Bus</td>
<td>$\Theta (n)$</td>
<td>$\Theta (1)$</td>
<td>$\Theta (n)$</td>
</tr>
<tr>
<td>2D Mesh</td>
<td>$\Theta (n)$</td>
<td>$\Theta (\sqrt{n})$</td>
<td></td>
</tr>
<tr>
<td>3D Mesh</td>
<td>$\Theta (n)$</td>
<td>$\Theta (\sqrt[3]{n})$</td>
<td></td>
</tr>
<tr>
<td>Tree</td>
<td>$\Theta (n)$</td>
<td>$\Theta (\log n)$</td>
<td>$\geq \Theta (\sqrt[3]{n})$</td>
</tr>
<tr>
<td>N-cube</td>
<td>$\Theta (n \log n)$</td>
<td>$\Theta (\log n)$</td>
<td>$\geq \Theta (\sqrt[3]{n})$</td>
</tr>
</tbody>
</table>

**IS IT REAL?**

- Speed of Light: ~ 1 ns/foot (typical bus propagation: 5 ns/foot)
- Density limits: can a node shrink forever? How about Power, Heat, etc … ?

**OBSERVATION:** Links on Tree, N-cube must grow with n; hence time/link must grow.
Communications Futures

Backplane Buses – *standard for peripherals*
- + easy hardware configurability
- + vendor-independent standards
- - serialized communications
- - bottleneck as systems scale up

Specialized buses for memory, graphics, …

New-generation communications...
- • Log networks (trees, hypercubes, …)
- • 2D Meshes (IWARP, …)
- • 3D Meshes …
  - • 4-neighbor, 3D mesh (NuMesh Diamond lattice)

Space: *the final frontier?*