Overview: Cost and Design

- Review from Last Lecture (2 minutes)
- Cost and Price (18)
- Administrative Matters (3 minutes)
- Design process (27 minutes)
- Break (5 minutes)
- More Design process (15 minutes)
- Online notebook (10 minutes)

Review: Performance and Technology Trends

- Technology Power: $1.2 \times 1.2 \times 1.2 = 1.7 \times / \text{year}$
  - Feature Size: shrinks 10% / yr. $\Rightarrow$ Switching speed improves 1.2 / yr.
  - Density: improves 1.2x / yr.
  - Die Area: 1.2x / yr.

- RISC lesson is to keep the ISA as simple as possible:
  - Shorter design cycle $\Rightarrow$ fully exploit the advancing technology (~3yr)
  - Advanced branch prediction and pipeline techniques
  - Bigger and more sophisticated on-chip caches

Review: General C/L Cell Delay Model

- Combinational Cell (symbol) is fully specified by:
  - functional (input $\rightarrow$ output) behavior
  - truth-table, logic equation, VHDL
  - load factor of each input
  - critical propagation delay from each input to each output for each transition
    - $T_{\text{HL}}(A, o) = \text{Fixed Internal Delay} + \text{Load-dependent-delay} \times \text{load}$

- Linear model composes
Review: Characterize a Gate

- Input capacitance for each input
- For each input-to-output path:
  - For each output transition type (H→L, L→H, H→Z, L→Z ... etc.)
    - Internal delay (ns)
    - Load dependent delay (ns / fF)
- Example: 2-input NAND Gate

![NAND Gate Diagram]

For A and B: Input Load (I.L.) = 61 fF

For either A -> Out or B -> Out:
- \( T_{lh} = 0.5\text{ns} \)
- \( T_{lhf} = 0.0021\text{ns} / \text{fF} \)
- \( T_{hl} = 0.1\text{ns} \)
- \( T_{hlf} = 0.0020\text{ns} / \text{fF} \)

CS152 Logic Elements

- NAND2, NAND3, NAND 4
- NOR2, NOR3, NOR4
- INV1x (normal inverter)
- INV4x (inverter with large output drive)
- XOR
- XNOR2
- PWR: Source of 1's
- GND: Source of 0's
- fast MUXes
- D flip flop with negative edge triggered

Storage Element’s Timing Model

- Setup Time: Input must be stable BEFORE the trigger clock edge
- Hold Time: Input must REMAIN stable after the trigger clock edge
- Clock-to-Q time:
  - Output cannot change instantaneously at the trigger clock edge
  - Similar to delay in logic gates, two components:
    - Internal Clock-to-Q
    - Load dependent Clock-to-Q
- Typical for class: 1ns Setup, 0.5ns Hold

Clocking Methodology

- All storage elements are clocked by the same clock edge
- The combination logic block’s:
  - Inputs are updated at each clock tick
  - All outputs MUST be stable before the next clock tick
Critical Path & Cycle Time

° Critical path: the slowest path between any two storage devices
° Cycle time is a function of the critical path
° Must be greater than:
  • Clock-to-Q + Longest Path through Combination Logic + Setup

Clock Skew’s Effect on Cycle Time

° The worst case scenario for cycle time consideration:
  • The input register sees CLK1
  • The output register sees CLK2
° Cycle Time - Clock Skew ≥ CLK-to-Q + Longest Delay + Setup
  ⇒ Cycle Time ≥ CLK-to-Q + Longest Delay + Setup + Clock Skew

Tricks to Reduce Cycle Time

° Reduce the number of gate levels
° Use esoteric/dynamic timing methods
° Pay attention to loading
  • One gate driving many gates is a bad idea
  • Avoid using a small gate to drive a long wire
° Use multiple stages to drive large load

How to Avoid Hold Time Violation?

° Hold time requirement:
  • Input to register must NOT change immediately after the clock tick
° This is usually easy to meet in the “edge trigger” clocking scheme
° Hold time of most FFs is ≤ 0 ns
° CLK-to-Q + Shortest Delay Path must be greater than Hold Time
**Clock Skew's Effect on Hold Time**

- The worst case scenario for hold time consideration:
  - The input register sees CLK2
  - The output register sees CLK1
  - fast FF2 output must not change input to FF1 for same clock edge

\[ (\text{CLK-to-Q + Shortest Delay Path} - \text{Clock Skew}) > \text{Hold Time} \]

**Integrated Circuit Costs**

\[
\text{Die cost} = \frac{\text{Wafer cost}}{\text{Die yield}} = \frac{\text{Dies per Wafer}}{\text{Die yield}}
\]

\[
\text{Dies per wafer} = \frac{\pi \cdot (\text{Wafer diam} / 2)^2}{\text{Die Area}} - \pi \cdot \frac{\text{Wafer diam}}{2} - \text{Test dies} = \frac{\text{Wafer Area}}{\text{Die Area}}
\]

\[
\text{Die Yield} = \frac{\text{Wafer yield}}{1 + \left( \frac{\text{Defects per unit area}}{\alpha} \right) \cdot \text{Die Area}^\alpha}
\]

**Die Cost** goes roughly with \((\text{die area})^3\) or \((\text{die area})^4\)

**Real World Examples**

<table>
<thead>
<tr>
<th>Chip</th>
<th>Metal layers</th>
<th>Line width</th>
<th>Wafer cost</th>
<th>Defect/cm²</th>
<th>Area/mm²</th>
<th>Dies/ wafer</th>
<th>Yield</th>
<th>Die Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>386DX</td>
<td>2</td>
<td>0.90</td>
<td>$900</td>
<td>1.0</td>
<td>43</td>
<td>360</td>
<td>71%</td>
<td>$4</td>
</tr>
<tr>
<td>486DX2</td>
<td>3</td>
<td>0.80</td>
<td>$1200</td>
<td>1.0</td>
<td>81</td>
<td>181</td>
<td>54%</td>
<td>$12</td>
</tr>
<tr>
<td>PowerPC 601</td>
<td>4</td>
<td>0.80</td>
<td>$1700</td>
<td>1.3</td>
<td>121</td>
<td>115</td>
<td>28%</td>
<td>$53</td>
</tr>
<tr>
<td>HP PA 7100</td>
<td>3</td>
<td>0.80</td>
<td>$1300</td>
<td>1.0</td>
<td>196</td>
<td>66</td>
<td>27%</td>
<td>$73</td>
</tr>
<tr>
<td>DEC Alpha</td>
<td>3</td>
<td>0.70</td>
<td>$1500</td>
<td>1.2</td>
<td>234</td>
<td>53</td>
<td>19%</td>
<td>$149</td>
</tr>
<tr>
<td>SuperSPARC</td>
<td>3</td>
<td>0.70</td>
<td>$1700</td>
<td>1.6</td>
<td>256</td>
<td>48</td>
<td>13%</td>
<td>$272</td>
</tr>
<tr>
<td>Pentium</td>
<td>3</td>
<td>0.80</td>
<td>$1500</td>
<td>1.5</td>
<td>296</td>
<td>40</td>
<td>9%</td>
<td>$417</td>
</tr>
</tbody>
</table>


**Die Yield**

- Raw Dice Per Wafer
  - wafer diameter: die area (mm²)
  - 100 144 196 256 324 400
  - 6"/15cm: 139 90 62 44 32 23
  - 8"/20cm: 265 177 124 90 68 52
  - 10"/25cm: 431 290 206 153 116 90

- Die yield: 23% 19% 16% 12% 11% 10%
  - typical CMOS process: α = 2, wafer yield = 90%, defect density = 2/cm², 4 test sites/wafer

**Good Dice Per Wafer (Before Testing!)

- 6"/15cm: 31 16 9 5 3 2
- 8"/20cm: 59 32 19 11 7 5
- 10"/25cm: 96 53 32 20 13 9

- typical cost of an 8", 4 metal layers, 0.5um CMOS wafer: ~$2000
Other Costs

IC cost = Die cost + Testing cost + Packaging cost

Packaging Cost: depends on pins, heat dissipation

<table>
<thead>
<tr>
<th>Chip</th>
<th>Die cost</th>
<th>Package type</th>
<th>cost</th>
<th>Test &amp; Assembly</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>386DX</td>
<td>$4</td>
<td>QFP</td>
<td>$1</td>
<td>$4</td>
<td>$9</td>
</tr>
<tr>
<td>486DX2</td>
<td>$12</td>
<td>PGA</td>
<td>$11</td>
<td>$12</td>
<td>$35</td>
</tr>
<tr>
<td>PowerPC 601</td>
<td>$53</td>
<td>QFP</td>
<td>$3</td>
<td>$21</td>
<td>$77</td>
</tr>
<tr>
<td>HP PA 7100</td>
<td>$73</td>
<td>PGA</td>
<td>$35</td>
<td>$16</td>
<td>$124</td>
</tr>
<tr>
<td>DEC Alpha</td>
<td>$149</td>
<td>PGA</td>
<td>$30</td>
<td>$23</td>
<td>$202</td>
</tr>
<tr>
<td>SuperSPARC</td>
<td>$272</td>
<td>PGA</td>
<td>$20</td>
<td>$34</td>
<td>$326</td>
</tr>
<tr>
<td>Pentium</td>
<td>$417</td>
<td>PGA</td>
<td>$19</td>
<td>$37</td>
<td>$473</td>
</tr>
</tbody>
</table>

Administrative Matters

- Review complete: did ok on prob 2 & 3 Problems 1 and 4 more challenging Make sure you look at solutions! (out soon)
  - Average: 73, Std: 13, Low: 49, high: 98
- Read Chapter 4: ALU, Multiply, Divide, FP Mult
- Unbroken SPIM is up now. Broken version up soon
  - Get going on your testing methodology!
  - You will be graded on how thorough your testing technique is in addition to whether or not you get the bugs
- Sections a bit unbalanced
  - Can we get more people in the 4-6 section?

The Design Process

"To Design Is To Represent"

Design activity yields description/representation of an object

- Traditional craftsman does not distinguish between the conceptualization and the artifact
- Separation comes about because of complexity
- The concept is captured in one or more representation languages
- This process is design

Design Begins With Requirements

- Functional Capabilities: what it will do
- Performance Characteristics: Speed, Power, Area, Cost, . . .

Design Process (cont.)

Design Finishes As Assembly

- Design understood in terms of components and how they have been assembled
- Top Down decomposition of complex functions (behaviors) into more primitive functions
- bottom-up composition of primitive building blocks into more complex assemblies

Design is a "creative process,” not a simple method
**Design Refinement**

- Informal System Requirement
- Initial Specification
- Intermediate Specification
- Final Architectural Description
- Intermediate Specification of Implementation
- Final Internal Specification
- Physical Implementation

**Design as Search**

- **Problem A**
  - Strategy 1
  - Strategy 2
  - SubProb 1
  - SubProb 2
  - SubProb 3

**Design involves educated guesses and verification**

- Given the goals, how should these be prioritized?
- Given alternative design pieces, which should be selected?
- Given design space of components & assemblies, which part will yield the best solution?

**Feasible (good) choices vs. Optimal choices**

**Problem: Design a “fast” ALU for the MIPS ISA**

- Requirements?
- Must support the Arithmetic / Logic operations
- Tradeoffs of cost and speed based on frequency of occurrence, hardware budget

**MIPS ALU requirements**

- Add, AddU, Sub, SubU, AddI, AddIU
  - => 2's complement adder/sub with overflow detection
- And, Or, AndI, OrI, Xor, Xori, Nor
  - => Logical AND, logical OR, XOR, nor
- SLTI, SLTIU (set less than)
  - => 2's complement adder with inverter, check sign bit of result
- ALU from from CS 150 / P&H book chapter 4 supports these ops
MIPS arithmetic instruction format

| Signed arith generate overflow, no carry |

R-type:

<table>
<thead>
<tr>
<th>Type</th>
<th>op</th>
<th>funct</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADDI</td>
<td>10</td>
<td>xx</td>
</tr>
<tr>
<td>ADDIU</td>
<td>11</td>
<td>xx</td>
</tr>
<tr>
<td>SLTI</td>
<td>12</td>
<td>xx</td>
</tr>
<tr>
<td>SLTIU</td>
<td>13</td>
<td>xx</td>
</tr>
<tr>
<td>ANDI</td>
<td>14</td>
<td>xx</td>
</tr>
<tr>
<td>ORI</td>
<td>15</td>
<td>xx</td>
</tr>
<tr>
<td>XORI</td>
<td>16</td>
<td>xx</td>
</tr>
<tr>
<td>LUI</td>
<td>17</td>
<td>xx</td>
</tr>
</tbody>
</table>

I-Type:

<table>
<thead>
<tr>
<th>Type</th>
<th>op</th>
<th>funct</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD</td>
<td>00</td>
<td>40</td>
</tr>
<tr>
<td>ADDU</td>
<td>00</td>
<td>41</td>
</tr>
<tr>
<td>SUB</td>
<td>00</td>
<td>42</td>
</tr>
<tr>
<td>SUBU</td>
<td>00</td>
<td>43</td>
</tr>
<tr>
<td>AND</td>
<td>00</td>
<td>44</td>
</tr>
<tr>
<td>OR</td>
<td>00</td>
<td>45</td>
</tr>
<tr>
<td>XOR</td>
<td>00</td>
<td>46</td>
</tr>
<tr>
<td>NOR</td>
<td>00</td>
<td>47</td>
</tr>
</tbody>
</table>

Design Trick: divide & conquer

° Break the problem into simpler problems, solve them and glue together the solution
° Example: assume the immediates have been taken care of before the ALU
  - 10 operations (4 bits)

Refined Requirements

(1) Functional Specification
inputs: 2 x 32-bit operands A, B, 4-bit mode
outputs: 32-bit result S, 1-bit carry, 1 bit overflow
operations: add, addu, sub, subu, and, or, xor, nor, slt, sltU

(2) Block Diagram (powerview symbol, VHDL entity)

Behavioral Representation: VHDL

Entity ALU is
  generic (c_delay: integer := 20 ns;
            S_delay: integer := 20 ns);
  port (signal A, B: in vlbit_vector (0 to 31);
        signal m: in vlbit_vector (0 to 3);
        signal S: out vlbit_vector (0 to 31);
        signal c: out vlbit;
        signal ovf: out vlbit)
  end ALU;

  
  S <= A + B;
**Design Decisions**

° Simple bit-slice
  - big combinational problem
  - many little combinational problems
  - partition into 2-step problem

° Bit slice with carry look-ahead
  °...

---

**Refined Diagram: bit-slice ALU**

---

**7-to-2 Combinational Logic**

° start turning the crank . . .

<table>
<thead>
<tr>
<th>Function</th>
<th>Inputs</th>
<th>Outputs</th>
<th>K-Map</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>M0 M1 M2 M3 A B Cin</td>
<td>S Cout</td>
<td></td>
</tr>
<tr>
<td>add</td>
<td>0 0 0 0 0 0 0 0 0</td>
<td>0 0</td>
<td></td>
</tr>
</tbody>
</table>

---

**Seven plus a MUX ?**

° Design trick 2: take pieces you know (or can imagine) and try to put them together
° Design trick 3: solve part of the problem and extend
Additional operations

\[ A - B = A + (\overline{B}) = A + B + 1 \]

- form two complement by invert and add one

Set-less-than? – left as an exercise

Revised Diagram

- LSB and MSB need to do a little extra

Overflow

<table>
<thead>
<tr>
<th>Dec/10</th>
<th>Binary</th>
<th>Dec/10</th>
<th>2’s Complement</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0000</td>
<td>0</td>
<td>0000</td>
</tr>
<tr>
<td>1</td>
<td>0001</td>
<td>-1</td>
<td>1111</td>
</tr>
<tr>
<td>2</td>
<td>0010</td>
<td>-2</td>
<td>1110</td>
</tr>
<tr>
<td>3</td>
<td>0011</td>
<td>-3</td>
<td>1101</td>
</tr>
<tr>
<td>4</td>
<td>0100</td>
<td>-4</td>
<td>1100</td>
</tr>
<tr>
<td>5</td>
<td>0101</td>
<td>-5</td>
<td>1011</td>
</tr>
<tr>
<td>6</td>
<td>0110</td>
<td>-6</td>
<td>1010</td>
</tr>
<tr>
<td>7</td>
<td>0111</td>
<td>-7</td>
<td>1001</td>
</tr>
<tr>
<td>8</td>
<td>1000</td>
<td>-8</td>
<td>1000</td>
</tr>
</tbody>
</table>

- Examples: \( 7 + 3 = 10 \) but...

Overflow Detection

- Overflow: the result is too large (or too small) to represent properly
  - Example: \(-8 \leq 4\)-bit binary number \(\leq 7\)
  - When adding operands with different signs, overflow cannot occur!

- Overflow occurs when adding:
  - 2 positive numbers and the sum is negative
  - 2 negative numbers and the sum is positive

- On your own: Prove you can detect overflow by:
  - Carry into MSB \(\neq\) Carry out of MSB
Overflow Detection Logic

- Carry into MSB ≠ Carry out of MSB
  - For a N-bit ALU: Overflow = CarryIn[N - 1] XOR CarryOut[N - 1]

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>X XOR Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

CarryOut3

But What about Performance?

- Critical Path of n-bit Rippled-carry adder is n*CP

Design Trick: Throw hardware at it

More Revised Diagram

- LSB and MSB need to do a little extra

Signed arith and cin xor co

More Revised Diagram

- Carry Look Ahead (Design trick: peek)

C0 = Cin

G = A and B
P = A xor B
Plumbing as Carry Lookahead Analogy

Cascaded Carry Look-ahead (16-bit): Abstraction

2nd level Carry, Propagate as Plumbing

Design Trick: Guess (or “Precompute”)
Carry Skip Adder: reduce worst case delay

4-bit Ripple Adder

A0B

S

P0P1P2P3

4-bit Ripple Adder

A4B

S

P0P1P2P3

Exercise: optimal design uses variable block sizes

Just speed up the slowest case for each block

Additional MIPS ALU requirements

° Mult, MultU, Div, DivU (next lecture)
  => Need 32-bit multiply and divide, signed and unsigned

° Sll, Srl, Sra (next lecture)
  => Need left shift, right shift, right shift arithmetic by 0 to 31 bits

° Nor (leave as exercise to reader)
  => logical NOR or use 2 steps: (A OR B) XOR 1111....1111

Elements of the Design Process

° Divide and Conquer (e.g., ALU)
  • Formulate a solution in terms of simpler components.
  • Design each of the components (subproblems)

° Generate and Test (e.g., ALU)
  • Given a collection of building blocks, look for ways of putting them together that meets requirement

° Successive Refinement (e.g., carry lookahead)
  • Solve “most” of the problem (i.e., ignore some constraints or special cases), examine and correct shortcomings.

° Formulate High-Level Alternatives (e.g., carry select)
  • Articulate many strategies to “keep in mind” while pursuing any one approach.

° Work on the Things you Know How to Do
  • The unknown will become “obvious” as you make progress.

Summary of the Design Process

Hierarchical Design to manage complexity

Top Down vs. Bottom Up vs. Successive Refinement

Importance of Design Representations:

- Block Diagrams
- Decomposition into Bit Slices
- Truth Tables, K-Maps
- Circuit Diagrams
- Other Descriptions: state diagrams, timing diagrams, reg xfer, . . .

Optimization Criteria:

- Area
- Logic Levels
- Delay
- Power
- Pin Out
- Cost
- Design time
**Why should you keep an design notebook?**

- Keep track of the design decisions **and the reasons behind them**
  - Otherwise, it will be hard to debug and/or refine the design
  - Write it down so that can remember in long project: 2 weeks -> 2 yrs
  - Others can review notebook to see what happened
- Record insights you have on certain aspect of the design as they come up
- Record of the different design & debug experiments
  - Memory can fail when very tired
- Industry practice: learn from others mistakes

**Why do we keep it on-line?**

- You need to force yourself to take notes
  - Open a window and leave an editor running while you work
    1) Acts as reminder to take notes
    2) Makes it easy to take notes
  - 1) + 2) => will actually do it
- Take advantage of the window system’s “cut and paste” features
- It is much easier to read your typing than your writing
- Also, paper log books have problems
  - Limited capacity => end up with many books
  - May not have right book with you at time vs. networked screens
  - Can use computer to search files/index files to find what looking for

**How should you do it?**

- Keep it simple
  - **DON’T** make it so elaborate that you won’t use (fonts, layout, …)
- Separate the entries by dates
  - type “date” command in another window and cut&paste
- Start day with problems going to work on today
- Record output of simulation into log with cut&paste; add date
  - May help sort out which version of simulation did what
- Record key email with cut&paste
- Record of what works & doesn’t helps team decide what went wrong after you left
- Index: write a one-line summary of what you did at end of each day

**On-line Notebook Example**

- Refer to the handout “Example of On-Line Log Book” on cs 152 home page
1st page of On-line notebook (Index + Wed. 9/6/95)

* Index

<table>
<thead>
<tr>
<th>Date</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wed Sep 6 00:47:28 PDT 1995</td>
<td>Created the 32-bit comparator component</td>
</tr>
<tr>
<td>Thu Sep 7 14:02:21 PDT 1995</td>
<td>Tested the comparator</td>
</tr>
<tr>
<td>Mon Sep 11 12:01:45 PDT 1995</td>
<td>Investigated bug found by Bart in comp32 and fixed it</td>
</tr>
</tbody>
</table>

Wed Sep 6 00:47:28 PDT 1995

Goal: Layout the schematic for a 32-bit comparator

I've layed out the schematics and made a symbol for the comparator. I named it comp32. The files are

```
~/wv/proj1/sch/comp32.sch
~/wv/proj1/sch/comp32.sym
```


2nd page of On-line notebook (Thursday 9/7/95)

<table>
<thead>
<tr>
<th>Date</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thu Sep 7 14:02:21 PDT 1995</td>
<td>Goal: Test the comparator component</td>
</tr>
<tr>
<td></td>
<td>I've written a command file to test comp32. I've placed it in ~/wv/proj1/diagnostics/comp32.cmd.</td>
</tr>
<tr>
<td></td>
<td>I ran the command file in viewsim and it looks like the comparator is working fine. I saved the output into a log file called ~/wv/proj1/diagnostics/comp32.log</td>
</tr>
<tr>
<td></td>
<td>Notified the rest of the group that the comparator is done.</td>
</tr>
</tbody>
</table>


3rd page of On-line notebook (Monday 9/11/95)

<table>
<thead>
<tr>
<th>Date</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mon Sep 11 12:01:45 PDT 1995</td>
<td>Goal: Investigate bug discovered in comp32 and hopefully fix it</td>
</tr>
<tr>
<td></td>
<td>Bart found a bug in my comparator component. He left the following e-mail.</td>
</tr>
<tr>
<td></td>
<td>From <a href="mailto:bart@simpsons.residence">bart@simpsons.residence</a> Sun Sep 10 01:47:02 1995</td>
</tr>
<tr>
<td></td>
<td>Received: by wayne.manor (NX5.67e/NX3.08) id AA00334; Sun, 10 Sep 95 01:47:01 –0800</td>
</tr>
<tr>
<td></td>
<td>Date: Wed, 10 Sep 95 01:47:01 –0800</td>
</tr>
<tr>
<td></td>
<td>From: Bart Simpson <a href="mailto:bart@simpsons.residence">bart@simpsons.residence</a></td>
</tr>
<tr>
<td></td>
<td>To: <a href="mailto:bruce@wayne.manor">bruce@wayne.manor</a>, old_man@gokuraku, hojo@sanctuary</td>
</tr>
<tr>
<td></td>
<td>Subject: [cs152] bug in comp32</td>
</tr>
<tr>
<td></td>
<td>Status: R</td>
</tr>
<tr>
<td></td>
<td>Hey Bruce,</td>
</tr>
<tr>
<td></td>
<td>I think there's a bug in your comparator.</td>
</tr>
<tr>
<td></td>
<td>The comparator seems to think that ffffffff and fffffff7 are equal.</td>
</tr>
<tr>
<td></td>
<td>Can you take a look at this?</td>
</tr>
<tr>
<td></td>
<td>Bart</td>
</tr>
</tbody>
</table>

4th page of On-line notebook (9/11/95 contd)

<table>
<thead>
<tr>
<th>Date</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>I verified the bug. here's a viewsim of the bug as it appeared..</td>
</tr>
<tr>
<td></td>
<td>(equal should be 0 instead of 1)</td>
</tr>
<tr>
<td>SIM&gt;stepsize 10ns</td>
<td></td>
</tr>
<tr>
<td>SIM&gt;v a_in A[31:0]</td>
<td></td>
</tr>
<tr>
<td>SIM&gt;v b_in B[31:0]</td>
<td></td>
</tr>
<tr>
<td>SIM&gt;w a_in b_in equal</td>
<td></td>
</tr>
<tr>
<td>SIM&gt;a_a_in ffffffff\h</td>
<td></td>
</tr>
<tr>
<td>SIM&gt;a_b_in fffffff7\h</td>
<td></td>
</tr>
<tr>
<td>SIM&gt;sim time = 10.0ns A_IN=FFFFFFFF\h B_IN=FFFFFFF7\h EQUAL=1</td>
<td></td>
</tr>
</tbody>
</table>

 Ah. I've discovered the bug. I mislabeled the 4th net in the comp32 schematic. I corrected the mistake and re-checked all the other labels, just in case.

 I re-ran the old diagnostic test file and tested it against the bug Bart found. It seems to be working fine. Hopefully there aren't any more bugs:)
On second inspection of the whole layout, I think I can remove one level of gates in the design and make it go faster. But who cares! the comparator is not in the critical path right now. the delay through the ALU is dominating the critical path. so unless the ALU gets a lot faster, we can live with a less than optimal comparator.
I e-mailed the group that the bug has been fixed
Mon Sep 11 14:03:41 PDT 1995

• Perhaps later critical path changes:
  what was idea to make comparator faster? Check log book!

Lecture Summary

° Cost and Price
  • Die size determines chip cost: cost - die size
  • Cost v. Price: business model of company, pay for engineers
  • R&D must return $8 to $14 for every $1 investor
° An Overview of the Design Process
  • Design is an iterative process, multiple approaches to get started
  • Do NOT wait until you know everything before you start
° Example: Instruction Set drives the ALU design
° On-line Design Notebook
  • Open a window and keep an editor running while you work; cut & paste
  • Refer to the handout as an example
  • Former CS 152 students (and TAs) say they use on-line notebook for programming as well as hardware design; one of most valuable skills

Sample graph from the Alewife project:

- For the Communications and Memory Management Unit (CMMU)
- These statistics came from on-line record of bugs