CS152
Computer Architecture and Engineering

Lecture 5: Cost and Design

September 10, 1997
Dave Patterson (http.cs.berkeley.edu/~patterson)

lecture slides: http://www-inst.eecs.berkeley.edu/~cs152/
Review: Performance and Technology Trends

° **Technology Power: 1.2 x 1.2 x 1.2 = 1.7 x / year**
  - Feature Size: shrinks 10% / yr. => 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 => fully exploit the advancing technology (~3yr)
  - Advanced branch prediction and pipeline techniques
  - Bigger and more sophisticated on-chip caches
Review: Technology, Logic Design and Delay

° **CMOOSTechnology Trends**
  - Complementary: PMOS and NMOS transitors
  - CMOS inverter and CMOS logic gates

° **Delay Modeling and Gate Characterization**
  - Delay = Internal Delay + (Load Dependent Delay x Output Load)

° **Clocking Methodology and Timing Considerations**
  - Simplest clocking methodology
    - All storage elements use the SAME clock edge
  - Cycle Time = CLK-to-Q + Longest Delay Path + Setup + Clock Skew
  - (CLK-to-Q + Shortest Delay Path - Clock Skew) > Hold Time
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)
Integrated Circuit Costs

Die cost = \( \frac{\text{Wafer cost}}{\text{Dies per Wafer} \times \text{Die yield}} \)

 Dies per wafer = \( \pi \left( \frac{\text{Wafer diam}}{2} \right)^2 - \pi \frac{\text{Wafer diam}}{\sqrt{2} \times \text{Die Area}} - \text{Test dies} \approx \text{Wafer Area} \)

Die Yield = \( \frac{\text{Wafer yield}}{\{ 1 + \frac{\text{Defects per unit area} \times \text{Die Area}}{\alpha} \}^\alpha} \)

Die Cost is goes roughly with the cube of the area.
# Die Yield

## Raw Dices Per Wafer

<table>
<thead>
<tr>
<th>wafer diameter</th>
<th>die area (mm²)</th>
<th>100</th>
<th>144</th>
<th>196</th>
<th>256</th>
<th>324</th>
<th>400</th>
</tr>
</thead>
<tbody>
<tr>
<td>6”/15cm</td>
<td></td>
<td>139</td>
<td>90</td>
<td>62</td>
<td>44</td>
<td>32</td>
<td>23</td>
</tr>
<tr>
<td>8”/20cm</td>
<td></td>
<td>265</td>
<td>177</td>
<td>124</td>
<td>90</td>
<td>68</td>
<td>52</td>
</tr>
<tr>
<td>10”/25cm</td>
<td></td>
<td>431</td>
<td>290</td>
<td>206</td>
<td>153</td>
<td>116</td>
<td>90</td>
</tr>
</tbody>
</table>

**die yield** 23% 19% 16% 12% 11% 10%

*typical CMOS process: $\alpha = 2$, wafer yield=90%, defect density=2/cm², 4 test sites/wafer*

## Good Dices Per Wafer (Before Testing!)

<table>
<thead>
<tr>
<th>wafer diameter</th>
<th></th>
<th>6”/15cm</th>
<th>8”/20cm</th>
<th>10”/25cm</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>31</td>
<td>59</td>
<td>96</td>
</tr>
<tr>
<td></td>
<td></td>
<td>16</td>
<td>32</td>
<td>53</td>
</tr>
<tr>
<td></td>
<td></td>
<td>9</td>
<td>19</td>
<td>32</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5</td>
<td>11</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>7</td>
<td>13</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>5</td>
<td>9</td>
</tr>
</tbody>
</table>

*typical cost of an 8”, 4 metal layers, 0.5um CMOS wafer: ~$2000*
## 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>

Other Costs

IC cost = Die cost + Testing cost + Packaging cost

Final test yield

Packaging Cost: depends on pins, heat dissipation, ...

<table>
<thead>
<tr>
<th>Chip</th>
<th>Die cost</th>
<th>pins</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>132</td>
<td>QFP</td>
<td>$1</td>
<td>$4</td>
<td>$9</td>
</tr>
<tr>
<td>486DX2</td>
<td>$12</td>
<td>168</td>
<td>PGA</td>
<td>$11</td>
<td>$12</td>
<td>$35</td>
</tr>
<tr>
<td>PowerPC 601</td>
<td>$53</td>
<td>304</td>
<td>QFP</td>
<td>$3</td>
<td>$21</td>
<td>$77</td>
</tr>
<tr>
<td>HP PA 7100</td>
<td>$73</td>
<td>504</td>
<td>PGA</td>
<td>$35</td>
<td>$16</td>
<td>$124</td>
</tr>
<tr>
<td>DEC Alpha</td>
<td>$149</td>
<td>431</td>
<td>PGA</td>
<td>$30</td>
<td>$23</td>
<td>$202</td>
</tr>
<tr>
<td>SuperSPARC</td>
<td>$272</td>
<td>293</td>
<td>PGA</td>
<td>$20</td>
<td>$34</td>
<td>$326</td>
</tr>
<tr>
<td>Pentium</td>
<td>$417</td>
<td>273</td>
<td>PGA</td>
<td>$19</td>
<td>$37</td>
<td>$473</td>
</tr>
</tbody>
</table>
## System Cost: ≈1995-96 Workstation

<table>
<thead>
<tr>
<th>System</th>
<th>Subsystem</th>
<th>% of total cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cabinet</td>
<td>Sheet metal, plastic</td>
<td>1%</td>
</tr>
<tr>
<td></td>
<td>Power supply, fans</td>
<td>2%</td>
</tr>
<tr>
<td></td>
<td>Cables, nuts, bolts</td>
<td>1%</td>
</tr>
<tr>
<td></td>
<td><em>(Subtotal)</em></td>
<td><em>(4%)</em></td>
</tr>
<tr>
<td>Motherboard</td>
<td>Processor</td>
<td>6%</td>
</tr>
<tr>
<td></td>
<td>DRAM (64MB)</td>
<td>36%</td>
</tr>
<tr>
<td></td>
<td>Video system</td>
<td>14%</td>
</tr>
<tr>
<td></td>
<td>I/O system</td>
<td>3%</td>
</tr>
<tr>
<td></td>
<td>Printed Circuit board</td>
<td>1%</td>
</tr>
<tr>
<td></td>
<td><em>(Subtotal)</em></td>
<td><em>(60%)</em></td>
</tr>
<tr>
<td>I/O Devices</td>
<td>Keyboard, mouse</td>
<td>1%</td>
</tr>
<tr>
<td></td>
<td>Monitor</td>
<td>22%</td>
</tr>
<tr>
<td></td>
<td>Hard disk (1 GB)</td>
<td>7%</td>
</tr>
<tr>
<td></td>
<td>Tape drive (DAT)</td>
<td>6%</td>
</tr>
<tr>
<td></td>
<td><em>(Subtotal)</em></td>
<td><em>(36%)</em></td>
</tr>
</tbody>
</table>
Q: What % of company income on Research and Development (R&D)?

<table>
<thead>
<tr>
<th>Component Cost</th>
<th>Direct Costs</th>
<th>Gross Margin</th>
<th>Average Discount</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input: chips, displays, ...</td>
<td>Making it: labor, scrap, returns, ...</td>
<td>Overhead: R&amp;D, rent, marketing, profits, ...</td>
<td>Commission: channel profit, volume discounts,</td>
</tr>
<tr>
<td>+33%</td>
<td>+25–100%</td>
<td>+50–80%</td>
<td>(WS–PC)</td>
</tr>
<tr>
<td>component cost</td>
<td>direct costs</td>
<td>gross margin</td>
<td>(33–45%)</td>
</tr>
<tr>
<td>(8–10%)</td>
<td>(33–14%)</td>
<td>(25–31%)</td>
<td></td>
</tr>
</tbody>
</table>

+33% average selling price

+50–80% list price
Cost Summary

- Integrated circuits driving computer industry
- Die costs goes up with the cube of die area
- Economics ($$) is the ultimate driver for performance!
Administrative Matters

° Review complete: did well on prerequisite quiz! (Midterms will be more challenging)
° Read Chapter 4: ALU, Multiply, Divide, FP Mult
° Load balance of discussion sections
° Dollars for bugs! First to report bug gets $1 check
  • Send 1 bug/ email to mkp@mkp.com
  • Include page number, original text, why bug, fixed text
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

refinement
increasing level of detail
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 CS 150 / P&H book chapter 4 supports these ops
MIPS arithmetic instruction format

R-type: 

<table>
<thead>
<tr>
<th>31</th>
<th>25</th>
<th>20</th>
<th>15</th>
<th>5</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>op</td>
<td>Rs</td>
<td>Rt</td>
<td>Rd</td>
<td>funct</td>
<td></td>
</tr>
</tbody>
</table>

I-Type: 

| op | Rs | Rt | Immed 16 |

<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>
<tr>
<td>Type</td>
<td>op</td>
<td>funct</td>
</tr>
<tr>
<td>------</td>
<td>----</td>
<td>-------</td>
</tr>
<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>

° Signed arith genenerate overflow, no carry
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)

<table>
<thead>
<tr>
<th>Code</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>add</td>
</tr>
<tr>
<td>01</td>
<td>addU</td>
</tr>
<tr>
<td>02</td>
<td>sub</td>
</tr>
<tr>
<td>03</td>
<td>subU</td>
</tr>
<tr>
<td>04</td>
<td>and</td>
</tr>
<tr>
<td>05</td>
<td>or</td>
</tr>
<tr>
<td>06</td>
<td>xor</td>
</tr>
<tr>
<td>07</td>
<td>nor</td>
</tr>
<tr>
<td>12</td>
<td>slt</td>
</tr>
<tr>
<td>13</td>
<td>sltU</td>
</tr>
</tbody>
</table>
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)

![ALU Diagram]
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 . . .**

```markdown
<table>
<thead>
<tr>
<th>Function</th>
<th>Inputs</th>
<th>Outputs</th>
<th>K-Map</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>M0</td>
<td>M1</td>
<td>M2</td>
</tr>
<tr>
<td>add</td>
<td>0 0 0 0</td>
<td>0 0</td>
<td>0 0</td>
</tr>
<tr>
<td></td>
<td>127</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

---

*cs 152  L4  Cost.25*  
*DAP Fa 1997 ©UCB*
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

\[ A - B = A + (\neg B) \]

- form two complement by invert and add one

Set-less-than? – left as an exercise
LSB and MSB need to do a little extra

C/L to produce select, comp, c-in
### Overflow

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

° Examples: $7 + 3 = 10$ but ...
° $-4 - 5 = -9$ but ...

\[
\begin{array}{cccc}
0 & 1 & 1 & 1 \\
0 & 1 & 1 & 1 \\
+ & 0 & 0 & 1 & 1 \\
1 & 0 & 1 & 0 & 1 & 0 & -6
\end{array}
\]

\[
\begin{array}{cccc}
1 & 1 & 1 & 0 & 0 \\
+ & 1 & 0 & 1 & 1 \\
0 & 1 & 1 & 1 & 1 & 7
\end{array}
\]
Overflow Detection

° Overflow: the result is too large (or too small) to represent properly
  • Example: -8 <= 4-bit binary number <= 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 ≠ Carry out of MSB

```
0 1
+ 0 1 1 1
1 0
1 1
7
```

```
1 0
+ 1 1 0 0 0 0
0 1
1 1
3
```

```
1 0
+ 1 0 1 1
0 1
1 1
-6
```

```
0 1 1 1
+ 1 1 1 0 0 0
1 0 1 0
1 1
-4
```

```
0 1 1 1
+ 1 1 1 1 1 1
0 1 1 0
1 1
-5
```

```
1 0 1 0
+ 0 1 1 1
1 0
7
```
Overflow Detection Logic

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

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>Y</td>
<td>X XOR Y</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
More Revised Diagram

- LSB and MSB need to do a little extra

signed-arith and cin xor co

C/L to produce select, comp, c-in
But What about Performance?

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

Design Trick: throw hardware at it
Carry Look Ahead (Design trick: peek)

P = A and B
G = A xor B

C1 = G0 + C0 \cdot P0

C2 = G1 + G0 \cdot P1 + C0 \cdot P0 \cdot P1

C3 = G2 + G1 \cdot P2 + G0 \cdot P1 \cdot P2 + C0 \cdot P0 \cdot P1 \cdot P2

C4 = \ldots
Plumbing as Carry Lookahead Analogy
Cascaded Carry Look-ahead (16-bit): Abstraction

C1 = G0 + C0 \cdot P0

C2 = G1 + G0 \cdot P1 + C0 \cdot P0 \cdot P1

C3 = G2 + G1 \cdot P2 + G0 \cdot P1 \cdot P2 + C0 \cdot P0 \cdot P1 \cdot P2

C4 = \ldots
2nd level Carry, Propagate as Plumbing
Design Trick: Guess

\[ CP(2n) = 2 \cdot CP(n) \]

\[ CP(2n) = CP(n) + CP(mux) \]

Carry-select adder
Carry Skip Adder: reduce worst case delay

Just speed up the slowest case for each block

Exercise: optimal design uses variable block sizes
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:
- Gate Count
- [Package Count]
- Area
- Logic Levels
- Fan-in/Fan-out
- Delay
- Power
- Pin Out
- Cost
- Design time
Break (5 Minutes)
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
* Index ==============================================================

Wed Sep  6 00:47:28 PDT 1995 - Created the 32-bit comparator component
Thu Sep  7 14:02:21 PDT 1995 - Tested the comparator
Mon Sep 11 12:01:45 PDT 1995 - Investigated bug found by Bart in comp32 and fixed it

+ ====================================================================

Wed Sep  6 00:47:28 PDT 1995

Goal: Layout the schematic for a 32-bit comparator

I've laid 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

Wed Sep  6 02:29:22 PDT 1995

- ==============================================================

• Add 1 line index at front of log file at end of each session: date+summary
• Start with date, time of day + goal
• Make comments during day, summary of work
• End with date, time of day (and add 1 line summary at front of file)
Goal: Test the comparator component

I've written a command file to test comp32. I've placed it in ~/wv/proj1/diagnostics/comp32.cmd.

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

Notified the rest of the group that the comparator is done.
Goal: Investigate bug discovered in comp32 and hopefully fix it

Bart found a bug in my comparator component. He left the following e-mail.

-------------------
From bart@simpsons.residence Sun Sep 10 01:47:02 1995
Received: by wayne.manor (NX5.67e/NX3.0S)
          id AA00334; Sun, 10 Sep 95 01:47:01 -0800
Date: Wed, 10 Sep 95 01:47:01 -0800
From: Bart Simpson <bart@simpsons.residence>
To: bruce@wayne.manor, old_man@gokuraku, hojo@sanctuary
Subject: [cs152] bug in comp32
Status: R

Hey Bruce,
I think there's a bug in your comparator.
The comparator seems to think that ffffffffff and ffffffff7 are equal.

Can you take a look at this?
Bart
-------------------
I verified the bug. here's a viewsim of the bug as it appeared..
  (equal should be 0 instead of 1)

SIM> stepsze 10ns
SIM>v a_in A[31:0]
SIM>v b_in B[31:0]
SIM>w a_in b_in equal
SIM>a a_in ffffffff\h
SIM>a b_in ffffffff7\h
SIM>sim
time = 10.0ns  A_IN=FFFFFFFF\h B_IN=FFFFFFF7\h EQUAL=1
Simulation stopped at 10.0ns.

-------------------

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 compactor faster? Check log book!
Lecture Summary

° Cost and Price
  • Die size determines chip cost: cost \( \approx \) die size\(^{\alpha +1}\)
  • 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