The worst case scenario for cycle time consideration:
- The input register sees CLK1
- The output register sees CLK2

\[ \text{Cycle Time} = \text{CLK-to-Q} + \text{Longest Delay} + \text{Setup} + \text{Clock Skew} \]

For no violation
\( (\text{CLK-to-Q} + \text{Shortest Delay Path}) > \text{Hold Time} + \text{Clock Skew} \)

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

Integrated Circuit Costs
\[
\text{Die cost} = \frac{\text{Wafer cost}}{\text{Dies per Wafer} \cdot \text{Die yield}}
\]

\[ \text{Dies per Wafer} = p \cdot \left( \frac{\text{Wafer diam}}{2} \right)^2 - p \cdot \text{Wafer diam} - \text{Test dies} \]

\[ \text{Die Yield} = 1 + \frac{\text{Die cost}}{\text{Wafer yield}} \]

Die Cost is goes roughly with (die area)^3 or (die area)^4

Also Packaging and Test Cost – can easily exceed die cost
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 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 CS 150 / P&H book chapter 4 supports these ops

MIPS arithmetic instruction format

R-type:

<table>
<thead>
<tr>
<th>op</th>
<th>Rs</th>
<th>Rt</th>
<th>Rd</th>
<th>funct</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADDI</td>
<td>10</td>
<td>xx</td>
<td>xx</td>
<td>ADD 00 40</td>
</tr>
<tr>
<td>ADDIU</td>
<td>11</td>
<td>xx</td>
<td>xx</td>
<td>ADDU 00 41</td>
</tr>
<tr>
<td>SLTI</td>
<td>12</td>
<td>xx</td>
<td>xx</td>
<td>SUB 00 42</td>
</tr>
<tr>
<td>SLTIU</td>
<td>13</td>
<td>xx</td>
<td>xx</td>
<td>SUBU 00 43</td>
</tr>
<tr>
<td>ANDI</td>
<td>14</td>
<td>xx</td>
<td>xx</td>
<td>AND 00 44</td>
</tr>
<tr>
<td>ORI</td>
<td>15</td>
<td>xx</td>
<td>xx</td>
<td>OR 00 45</td>
</tr>
<tr>
<td>XORI</td>
<td>16</td>
<td>xx</td>
<td>xx</td>
<td>XOR 00 46</td>
</tr>
<tr>
<td>lui</td>
<td>17</td>
<td>xx</td>
<td>xx</td>
<td>NOR 00 47</td>
</tr>
</tbody>
</table>

I-Type:

<table>
<thead>
<tr>
<th>op</th>
<th>Rs</th>
<th>Rt</th>
<th>Immed 16</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD</td>
<td>00</td>
<td>40</td>
<td></td>
</tr>
<tr>
<td>ADDU</td>
<td>00</td>
<td>41</td>
<td></td>
</tr>
<tr>
<td>SUB</td>
<td>00</td>
<td>42</td>
<td></td>
</tr>
<tr>
<td>SUBU</td>
<td>00</td>
<td>43</td>
<td></td>
</tr>
<tr>
<td>AND</td>
<td>00</td>
<td>44</td>
<td></td>
</tr>
<tr>
<td>OR</td>
<td>00</td>
<td>45</td>
<td></td>
</tr>
<tr>
<td>XOR</td>
<td>00</td>
<td>46</td>
<td></td>
</tr>
<tr>
<td>NOR</td>
<td>00</td>
<td>47</td>
<td></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, sft, sftU

2. **Block Diagram** (powersview symbol, VHDL entity)

**Behavioral Representation: VHDL**

```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**

- ALU
- bit slice
- PLD Gates
- mux
**7-to-2 Combinational Logic**

° start turning the crank . . .

<table>
<thead>
<tr>
<th>Function Inputs</th>
<th>Outputs</th>
<th>K-Map</th>
</tr>
</thead>
<tbody>
<tr>
<td>add 0 0 0 0 0 0</td>
<td>S Out 0</td>
<td></td>
</tr>
</tbody>
</table>

° Function Inputs Outputs K-Map

<table>
<thead>
<tr>
<th>M0 M1 M2 M3 A B CIN</th>
<th>S Cout</th>
</tr>
</thead>
<tbody>
<tr>
<td>add 0 0 0 0 0 0 0 0</td>
<td>S Out 0</td>
</tr>
</tbody>
</table>

127

**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 + (– B) = A + B + 1
  - form two complement by invert and add one

**Revised Diagram**

° LSB and MSB need to do a little extra

° Set-less-than? – left as an exercise
### Overflow

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

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

### 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

### Overflow Detection Logic

- **Carry into MSB ≠ Carry out of MSB**
  - For a $N$-bit ALU: $\text{Overflow} = \text{CarryIn}[N - 1] \text{ XOR } \text{CarryOut}[N - 1]$

### More Revised Diagram

- **LSB and MSB need to do a little extra**

```markdown
<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

signed with 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)

Design Trick: Guess (or “Precompute”)

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

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, MulU, 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
- Logic Levels
- Fan-in/Fan-out
- Pin Out
- Area
- Delay
- Power
- Cost
- Design time
### Why should you keep a 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 to 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
On-line notebook (Index + Wed. 9/6/95)

* 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 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

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)

On-line notebook (Thurs. 9/7/95)

+ ====================================================================
Thu Sep  7 14:02:21 PDT 1995

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.

On-line notebook (Mon. 9/11/95)

+ ====================================================================
Mon Sep 11 12:01:45 PDT 1995

Goal: Investigate bug discovered in comp32 and hopefully fix it
Bart found a bug in my comparator component. He left the following
email:

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@wanye.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 ffffffff and fffffff7 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>stepsize 10ns
SIM>mov a_in A[31:0]
SIM>mov b_in B[31:0]
SIM>mov a_in equal
SIM>a_in ffffffff\b
SIM>b_in fffffff7\b
SIM>view
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 comparator faster? Check log book!

Sample graph from the Alewife project:

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

Lecture Summary

° Cost and Price
  • Die size determines chip cost: cost - die size\(^3\)
  • 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