Agenda

- Quartus II Tools Introduction
- Combinational Circuit (Cont.)
  - *How to build fast adder?*
  - *Carry-Bypass(Skip) Adder*
  - *16 bit-Carry Bypass Adder*
  - *Critical Pass Analysis*
  - *Carry Lookahead Adder*
- **Test Bench and Test Cases**
- *Timing Hazard and Glitch*
- *Fix Glitch in Logic Design*
Acknowledgement

- Materials in this lecture are courtesy of the following people
- Randy H. Katz (University of California, Berkeley, Department of Electrical Engineering & Computer Science)
- MIT Prof. Anantha Chandrakasan and Donald E. Troxel “Introduction to Digital System Design Lab”
- Professor C.K. Cheng, CSE140L

How to Build the Fast Adder?

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C_i</th>
<th>S</th>
<th>C_o</th>
<th>Carry status</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>delete</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>delete</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>propagate</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>propagate</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>propagate</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>propagate</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>generate</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>generate</td>
</tr>
</tbody>
</table>

Generate ($G$) = $AB$

Propagate ($P$) = $A \oplus B$

\[
C_o(G, P) = G + PC_i
\]

\[
S(G, P) = P \oplus C_i
\]
Carry Bypass (Skip) Adder

We can compute $P$ and $G$ in parallel for all bits.

Key Idea: if $(P_0, P_1, P_2, P_3)$ then $C_{o,3} = C_{i,0}$

What is Carry Lookahead?

Left diagram shows adder with Propagate and generate outputs.

Later stages have increasingly complex logic.
Carry Lookahead Adder

- Re-express the carry logic as follows:

\[ C_1 = G_0 + P_0 \ C_0 \]
\[ C_2 = G_1 + P_1 \ C_1 = G_1 + P_1 G_0 + P_1 P_0 \ C_0 = G_1:0 + P_1:0 \ C_0 \]
\[ C_3 = G_2 + P_2 \ C_2 = G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 \ C_0 = G_2:2 + P_2:2 \ C_0 \]
\[ C_4 = G_3 + P_3 \ C_3 = G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0 + P_3 P_2 P_1 P_0 \ C_0 = (G_3 + P_3 G_2) + (P_3 P_2 )Co_1 = G_3:2 + P_3:2 (G_1:0 + P_1:0 \ C_0) = G_3:0 + P_3:0 \ C_0 \]

... Each of the carry equations can be implemented in a two-level logic network

- Variables are the adder inputs and carry in to stage 0
  - \( G_{ij} \) and \( P_{ij} \) denote the Generate and Propagate functions, respectively, for a group of bits from positions \( i \) to \( j \). We call them Block Generate/Propagate. \( G_{ij} \) equals 1 if the group generates a carry independent of the incoming carry. \( P_{ij} \) equals 1 if an incoming carry propagates through the entire group. For example, \( G_{3:2} \) is equal to 1 if a carry is generated at bit position 3, or if a carry out is generated at bit position 2 and propagates through position 3. \( G_{3:2} = G_3 + P_3 G_2 \). \( P_{3:2} \) is true if an incoming carry propagates through both bit positions 2 and 3. \( P_{3:2} = P_3 P_2 \)

Ripple Carry effect has been eliminated!

16 Bits Carry Lookahead Adder

181 configured for A+B:
\( M = 0, S_{3.0} = 1001 \)

182 computes \( C_{i+1} \) for later stages, using block G & P from earlier stages
**What is Test Bench?**

- Test bench is a method of design verification. Verification is a process of maintaining the consistency of the design at different stages in the design flow.
- Test Bench uses dynamic simulation to verify the design. Usually, this is a VHDL/Verilog program.
- The test bench supplies stimuli to the design, verifies the outputs of the design, and compares the observed outputs with the expected values, see diagram below.

**Diagram:**

- Stimuli flows into the **UUT** (Unit Under Test).
- The **Test Bench** supplies stimuli to the **UUT**.
- **Outputs** are compared with **Expected Outputs**.
- An **error?** indicator is used to verify the correctness of the design.

---

**An Example of a Test Bench (VHDL)**

```vhdl
ARCHITECTURE testbench_arch OF FullAdder_tb IS

FILE RESULTS: TEXT OPEN WRITE_MODE I "results.txt"

COMPONENT FullAdder

PORT (  A : In std_logic;
        B : In std_logic;
        Ci : In std_logic;
        Co : Out std_logic;
        S : Out std_logic;
    );

END COMPONENT;

BEGIN

UUT : FullAdder
PORT MAP (  A => A,
            B => B,
            Ci => Ci,
            Co => Co,
            S => S);

-- Declare FullAdder as a component which can be referenced later
-- Internal signals of the test bench with initial values
-- Unit Under Test. The internal signals of the test bench are mapped onto the UUT
```

---

5
An Example of a Test Bench (Cont.)

Test Bench in Verilog (Full Adder)

```verilog
module test_adder;
    reg [3:0] a, b;
    reg cin;
    wire [3:0] sum;
    wire cout;

    full_adder 4bit du(a, b, cin, sum, cout);

    initial
        begin
            a = 4'b0000; // 0
            b = 4'b0001; // 1
            cin = 1'b1;
            if (sum == 1001, cout = 0)
                if (sum == 1000, cout = 1)
                    if (sum == 1001, cout = 1)
                        if (sum == 1000, cout = 0)
                            end // initial begin
            endmodule // test_adder
```

ModelSim Simulation
Timing Analysis

- Gate Delay

- Worst case delay is depend on the input test pattern, only certain input Pattern may sensitize the worst case delay

- Input test pattern grow exponentially with the number of inputs

- Timing report in QuartusII is static timing report

Worst Case Propagation Delay of 16-Bit Carry Bypass Adder

We assume the following gate delay:

- P, G from A, B: 1 gate delay
- P, G, Ci to Co or Sum for a FA: 1 gate delay
- 2:1 mux delay: 1 gate delay

*** Question: What is the worst case propagation delay for 16 bit Carry Bypass Adder?

- Timing Analysis is Very Tricky –Must Carefully Consider Data Dependencies For False Paths
1. The glitch is the result of timing differences in parallel data paths. It is associated with the function jumping between grouping or Product terms on the K-Maps.
2. Cover another grouping or product Term will fix the Glitch.

\[ F = X \cdot (\overline{Z}) + Y \cdot Z + X \cdot Y \]