

CSE140 L

Instructor: *Thomas Y. P. Lee*
January 25, 2006



Agenda

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



Acknowledgement

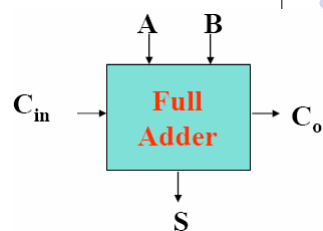


- Materials in this lecture are courtesy of the following people
- Randy H. Katz (University of California, Berkeley, Department of Electrical Engineering & Computer Science)
- Gaetano Borriello (University of Washington, Department of Computer Science & Engineering, <http://www.cs.washington.edu/370>)
- 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?



A	B	C_i	S	C_o	Carry status
0	0	0	0	0	delete
0	0	1	1	0	delete
0	1	0	1	0	propagate
0	1	1	0	1	propagate
1	0	0	1	0	propagate
1	0	1	0	1	propagate
1	1	0	0	1	generate
1	1	1	1	1	generate



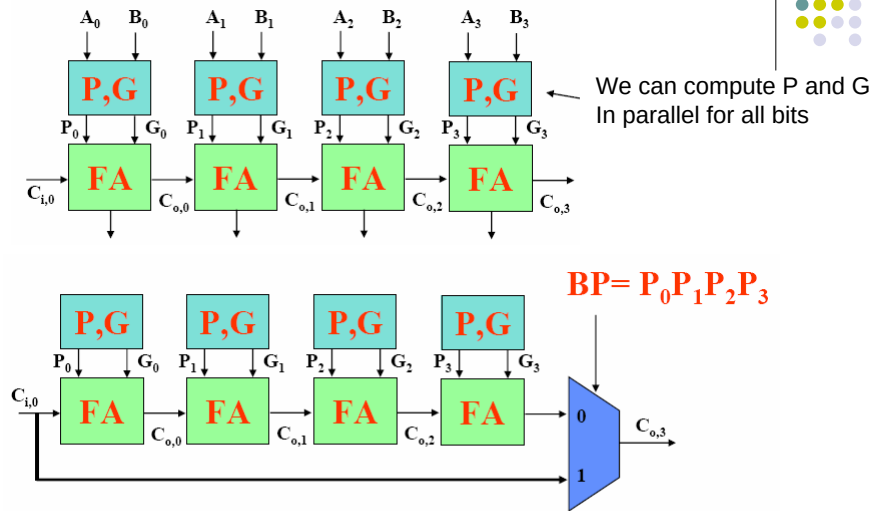
$$\text{Generate (G)} = AB$$

$$\text{Propagate (P)} = A \oplus B$$

$$C_o(G, P) = G + PC_i$$

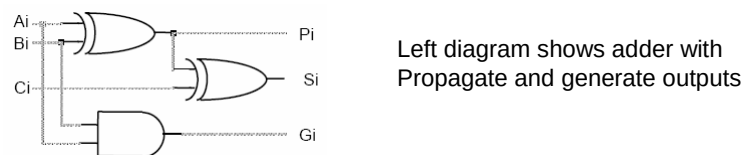
$$S(G, P) = P \oplus C_i$$

Carry Bypass(Skip) Adder

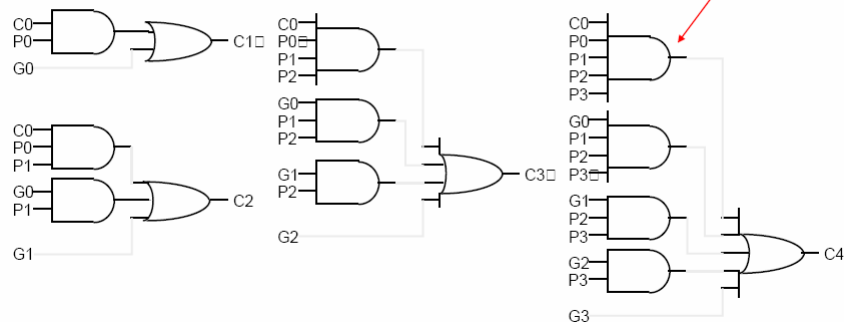


Key Idea: if $(P_0 P_1 P_2 P_3)$ then $C_{0,3} = C_{i,0}$

What is Carry Lookahead?



Later stages have **increasingly complex** logic



Carry Lookahead Adder

- Re-express the carry logic as follows:

$$C_1 = G_0 + P_0 C_0$$

$$C_2 = G_1 + P_1 C_1 = G_1 + P_1 G_0 + P_1 P_0 C_0 = G_{1:0} + P_{1:0} C_0$$

$$C_3 = G_2 + P_2 C_2 = G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_0$$

$$C_4 = G_3 + P_3 C_3 = G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0 +$$

$$P_3 P_2 P_1 P_0 C_0 = (G_3 + P_3 G_2) + (P_3 P_2) C_{0,1} = G_{3:2} +$$

$$P_{3:2} C_2 = G_{3:2} + P_{3:2} (G_{1:0} + P_{1:0} C_0) = G_{3:0} + P_{3:0} C_0$$

...

- 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

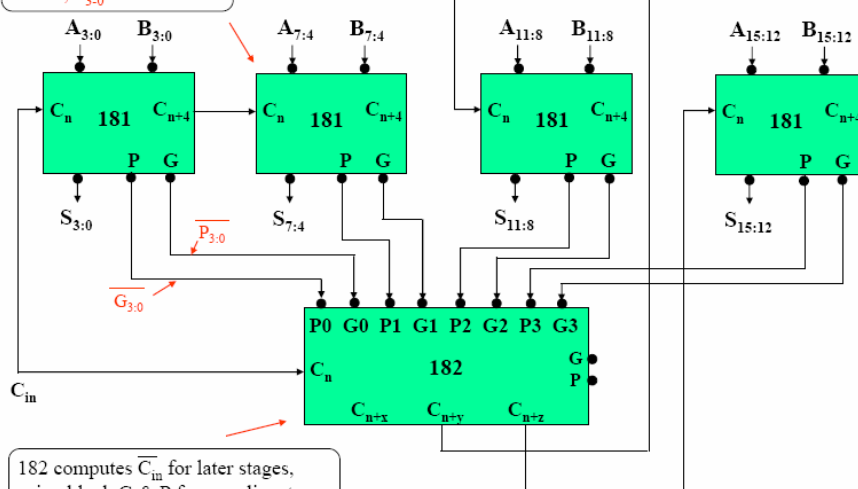
$G_{i:j}$ and $P_{i:j}$ denote the Generate and Propagate functions, respectively, for a group of bits from positions i to j . We call them Block Generate/Propagate. $G_{i:j}$ equals 1 if the group generates a carry independent of the incoming carry. $P_{i:j}$ equals 1 if an incoming carry propagates 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:

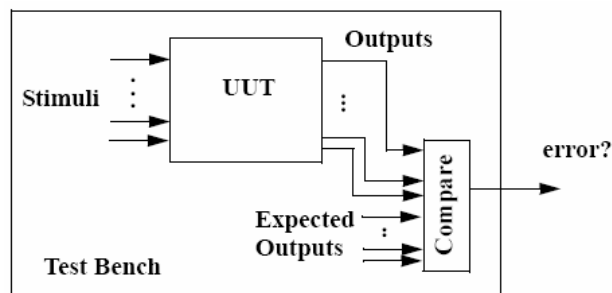
$M = 0, S_{3:0} = 1001$



182 computes $\bar{C}_n$ 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 VHDL/Verilog program
- The test bench supplies stimuli to the design, verify the outputs of the design and compares the observed outputs with the expected values, see diagram below



An Example of a Test Bench (VHDL)

```
ARCHITECTURE testbench_arch OF FullAdder_tb IS
    FILE RESULTS: TEXT OPEN WRITE_MODE IS "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;
```

Declare FullAdder as a component
which can be referenced later

```
    SIGNAL A : std_logic := '0';
    SIGNAL B : std_logic := '0';
    SIGNAL Ci : std_logic := '0';
    SIGNAL Co : std_logic := '0';
    SIGNAL S : std_logic := '0';
```

Internal signals of the test bench
with initial values

```
BEGIN
```

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

Unit Under Test
The internal signals of the test
bench are mapped onto the UUT

An Example of a Test Bench (Cont.)

```

...
...
...
BEGIN
    ----- Current Time: 50ns
    WAIT FOR 50 ns;
    A <= '1';
    -----
    ----- Current Time: 75ns
    WAIT FOR 25 ns;
    CHECK_S('1', 75);
    -----
    ----- Current Time: 100ns
    WAIT FOR 25 ns;
    A <= '0';
    B <= '1';

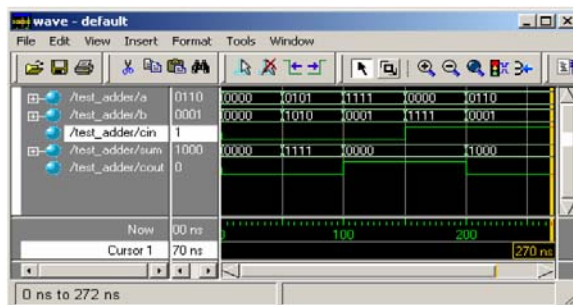
```

assign stimulus

Check output S at 75ns, the expected value is 1.

Test Bench in Verilog (Full Adder)

ModelSim Simulation



```

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

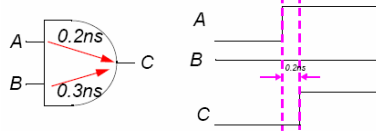
    full_adder_4bit dut(a, b, cin,
                       sum, cout);

    initial
    begin
        a = 4'b0000;
        b = 4'b0000;
        cin = 1'b0;
        #50;
        a = 4'b0101;
        b = 4'b1010;
        // sum = 1111, cout = 0
        #50;
        a = 4'b1111;
        b = 4'b0001;
        // sum = 0000, cout = 1
        #50;
        a = 4'b0000;
        b = 4'b1111;
        cin = 1'b1;
        // sum = 0000, cout = 1
        #50;
        a = 4'b0110;
        b = 4'b0001;
        // sum = 1000, cout = 0
        end // initial begin
    endmodule // test_adder

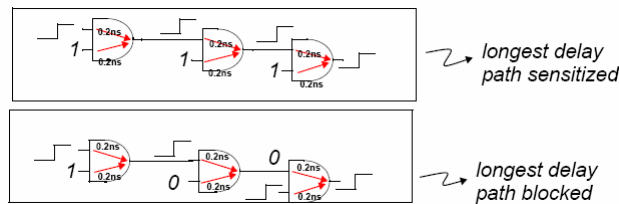
```

Timing Analysis

Gate Delay

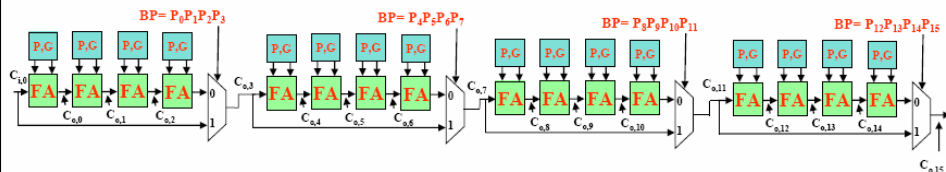


- Worst case delay is depend on the input test pattern, only certain input Pattern may sensize the worst case delay



- 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



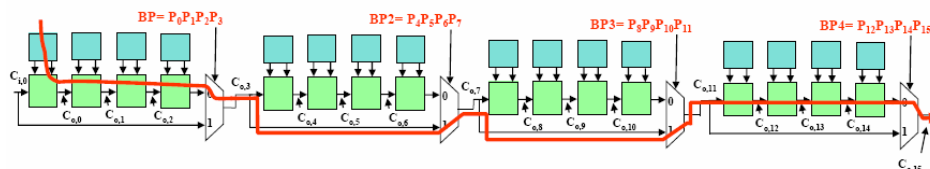
We assume the following gate delay:

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

