

Addition / Subtraction

Andrew Lukefahr

A simple testbench

```
`timescale 1ns/1ps
module BeltAlarm_tb();
logic k, p, s;
wire alarm;
BeltAlarm dut0( .k(k), .p(p), .s(s), .alarm(alarm) );
initial
begin
    k = 'h0; p = 'h0; s= 'h0;
    $monitor ("k:%b p:%b s:%b a:%b", k, p, s, alarm);
    #10
    assert(alarm == 'h0) else $fatal(1, "bad alarm");
    $display("@@Passed");
end
endmodule
```

```
module BeltAlarm(
    input k, p, s,
    output alarm
);
    assign alarm = k & p & ~s;
endmodule
```

Submodule Example

```
'timescale 1 ns/1 ns

module TwoBeltAlarm(
    input k, st_pas, sb_pas,
    input st_drv, sb_drv
    output alarm
);

    wire al_pas, al_drv; //intermediate wires

    //submodules, two different examples
    BeltAlarm ba_drv(k, st_drv, sb_drv, al_drv); //no named arguments
    BeltAlarm ba_pas(.k(k), .p(st_pas),
        .s(sb_pas), .alarm(al_pas)); // with named arguments

    assign alarm = al_pas | al_drv;
endmodule

`timescale 1 ns/1 ns

module BeltAlarm(
    input k, p, s,
    output alarm
);

    assign alarm = k & p & ~s;
endmodule
```

2-BeltAlarm Task

```
task checkAlarm(  
    input kV, stPasV, sbPasV,  
    input stDrvV, sbDrvV,  
    input alarmV  
) ;
```

```
k = kV; stPas=stPasV, sbPas=sbPasV;  
stDrv = stDrvV; sbDrv = sbDrvV;  
#10  
assert(alarm == alarmV) else  
    $fatal (1, "bad alarm, expected:%b got:%b",  
            alarmV, alarm);  
endtask
```

```
module TwoBeltAlarm(  
    input k, st_pas, sb_pas,  
    input st_drv, sb_drv  
);  
    wire al_pas, al_drv;  
  
    BeltAlarm ba_drv(k, st_drv, sb_drv, al_drv);  
    BeltAlarm ba_pas(.k(k), .p(st_pas),  
                    .s(sb_pas), .alarm(al_pas));  
  
    assign alarm = al_pas | al_drv;  
endmodule
```

```

initial begin
    k = 0; st_pas = 'b0; sb_pas = 'b0;
    st_drv = 'b0; sb_drv = 'h0;
#10
    assert(alarm == 'h0) else $fatal(1, "bad alarm");
#10
    checkAlarm(0, 'b0, 'h0, 'h0, 'h0, 'h0);
    for (int i = 0; i < 32; ++i) begin
        $display("i:%d [%b]", i, i[4:0]);
        if ( (i == 18) | (i == 22) | (i == 30)) // driver
            checkAlarm( i[4], i[3], i[2], i[1], i[0], 'h1);
        else if ( (i == 24) | (i == 25) | (i==27)) //passenger
            checkAlarm( i[4], i[3], i[2], i[1], i[0], 'h1);
        else if ( (i==26) ) //both
            checkAlarm( i[4], i[3], i[2], i[1], i[0], 'h1);
        else
            checkAlarm( i[4], i[3], i[2], i[1], i[0], 'h0);
    end
$display("@@Passed");
end

```

```

task checkAlarm(
    input kv, stPasV, sbPasV,
    input stDrvV, sbDrvV,
    input alarmV
);

k = kv; stPas=stPasV, sbPas=sbPasV;
stDrv = stDrvV; sbDrv = sbDrvV;
#10
assert(alarm == alarmV) else
    $fatal (1, "bad alarm, expected:%b got:%b",
            alarmV, alarm);
endtask

```

For Loops in Testbenches

- You can write for-loops in your testbenches

```
module for_loop_simulation ();
    logic [7:0] r_Data; // Create 8 bit value

    initial begin
        for (int ii=0; ii<6; ii=ii+1) begin
            r_Data = ii;
            $display("Time %d: r_Data is %b", $time, r_Data);
            #10;
        end
    end
endmodule
```

- Please *no for-loops in your synthesizable code (yet)!*

Arrays in Verilog

- Bundle multiple wires together to form an array.

```
type [mostSignificantIndex:leastSignificantIndex] name;
```

- Examples

- logic [15:0] x; //declare 16-bit array
- x[2] // access wire 2 within x
- x[5:2] //access wires 5 through 2
- x[5:2]= {1,0,y,z}; //concatenate 4 signals

Arrays in Verilog

- Can also be used in module definitions

```
module multiply (
    input [7:0]      a,      //8-bit signal
    input [7:0]      b,      //8-bit signal
    output [15:0]    c       //16-bit signal
);
//stuff
endmodule
```

Constants in Verilog

- A wire only needs 1 or 0
 - Arrays need more bits, how to specify?
-
- `8'h0 = 0000 0000 //using hex notation`
 - `8'hff = 1111 1111`
 - `8'b1 = 0000 0001 // using binary notation`
 - `8'b10 = 0000 0010`
 - `8'd8 = 0000 1000 //using decimal notation`

Constants in Verilog

```
module mtest;
    logic [7:0] aa = {1'b0, 1'b1, 1'b0, 1'b0,
                      1'b1, 1'b0, 1'b0, 1'b0};
    logic [7:0] bb = 8'b01001000;
    wire [15:0] cc ;
    logic [7:0] yy = {8{1'b1}}; //concat
    logic [7:0] zz = 'hff; //inferred

    multiply m0(.a(aa), .b(8'h1), .c(cc));
endmodule
```

wire vs logic

- ***wire***

- Only used with ‘assign’ and module outputs
- Boolean combination of inputs
- **Can never hold state**

- ***logic***

- Used with ‘always’ and module outputs
- Can be Boolean combination of inputs
- **Can hold state** (but doesn’t have to)

always_comb Blocks

```
wire foo;  
assign foo = x & y | z;
```

... is equivalent to ...

```
wire foo = x & y | z;
```

... is equivalent to ...

```
logic foo;  
always_comb //combinational  
    foo = x & y | z;
```

always_comb blocks with if

```
module decoder (
    input [1:0] sel,
    output logic [3:0] out
);
    always_comb begin
        if (sel == 2'b00) begin
            out = 4'b0001;
        end else if (sel == 2'b01) begin
            out = 4'b0010;
        end else if (sel == 2'b10) begin
            out = 4'b0100;
        end else if (sel == 2'b11) begin
            out = 4'b1000;
        end
    end
endmodule
```

always_comb with case

```
module decoder (
    input [1:0] sel,
    output logic [3:0] out
);

    always_comb begin
        case(sel)
            2'b00: out=4'b0001;
            2'b01: out=4'b0010;
            2'b10: out=4'b0100;
            2'b11: out=4'b1000;
        endcase
    end

endmodule
```

always_comb with case

```
module decoder (
    input [1:0] sel,
    output logic [3:0] out
);

    always_comb begin
        case(sel)
            2'b00: out=4'b0001;
            2'b01: out=4'b0010;
            2'b10: out=4'b0100;
            // what about sel==2'b11?
        endcase
    end

endmodule
```

always_comb with case

```
module decoder (
    input [1:0] sel,
    output logic [3:0] out
);

always_comb begin
    out = 4'b0000; //default
    case(sel)
        2'b00: out=4'b0001;
        2'b01: out=4'b0010;
        2'b10: out=4'b0100;
                                // what about sel==2'b11?
    endcase
end

endmodule
```

Always specify
defaults for
always_comb!

* add logic after class

always_ff

next time

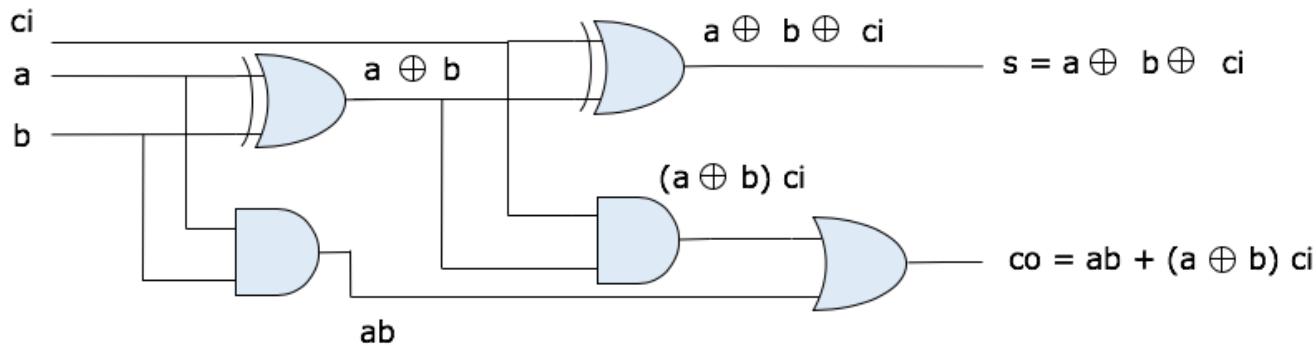
always_comb with case

```
module decoder (
    input [1:0] sel,
    output logic [3:0] out
);
    always_comb begin
        out = 4'b0000; //default
        case(sel)
            2'b00: out=4'b0001;
            2'b01: out=4'b0010;
            2'b10: out=4'b0100;
        ✓      defalt: out=4'b0000; // what about sel==2'b11?
        endcase
    end
endmodule
```

Always specify
defaults for
always_comb!

"Worris : Inferred
Latch"

1-Bit “Full” Adder to ‘always_comb’



```
module FullAddr (
    input a,b,ci,
    output s, co
);

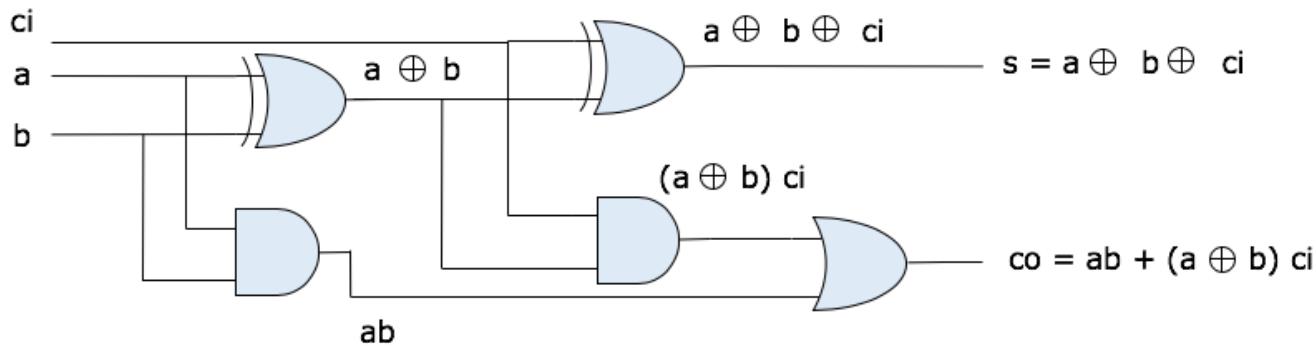
    assign s = a ^ b ^ ci;
    assign co = (a & b) |
        (( a ^ b) & ci);

endmodule
```

```
module FullAddr (
);

endmodule
```

1-Bit “Full” Adder to ‘always_comb’



```
module FullAddr (
    input a,b,ci,
    output s, co
);

    assign s = a ^ b ^ ci;
    assign co = (a & b) |
        (( a ^ b) & ci);

endmodule
```

```
module FullAddr (
    input a,b,ci,
    output logic s, co
);

    always_comb begin
        s = a ^ b ^ ci;
        co = (a & b) |
            (( a ^ b) & ci);
    end

endmodule
```

Addition / Subtraction

Binary Addition

$$\begin{array}{r} x \\ + y \\ \hline z=01 \end{array}$$

- How do we add two binary numbers, x & y?

$$x = \{0, 1\}$$

$$y = \{0, 1\}$$

$$\begin{array}{r} x \\ 0 \\ + \\ 0 \\ \hline 0 \end{array}$$

$$\begin{array}{r} y \\ 0 \\ + \\ 1 \\ \hline 1 \end{array}$$

$$\begin{array}{r} 1 \\ + \\ 0 \\ \hline 1 \end{array}$$

$$\begin{array}{r} 1 \\ + \\ 1 \\ \hline \end{array}$$

math (dec)

$$\begin{array}{l} 0+0=0 \\ 0+1=1 \\ 1+0=1 \\ 1+1=2 \end{array}$$

<u>Carry</u>	<u>Sum</u>
0	0
0	1
0	1
1	0

Binary Addition

Add two binary numbers, $x \oplus y$.

$$x = \{ \begin{matrix} 0 \\ 1 \end{matrix} \} \quad y = \{ \begin{matrix} 0 \\ 1 \end{matrix} \}$$

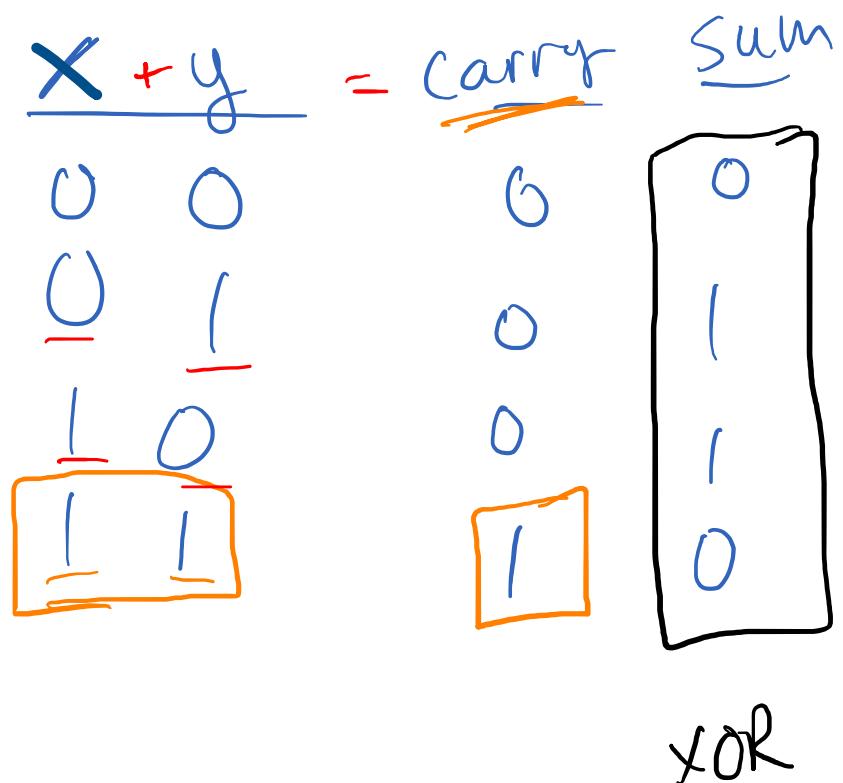
$$\begin{array}{r} x \quad y \\ \hline 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{array} \quad \begin{array}{c} \text{sum} \\ \hline 0 \\ 1 \\ 1 \\ 0 \end{array} \quad \begin{array}{c} \text{carry} \\ \hline 0 \\ 0 \\ 0 \\ 1 \end{array}$$

(10)

$$\begin{array}{r} 0 \\ + 1 \\ \hline 1 \end{array} \quad \begin{array}{r} 1 \\ + 1 \\ \hline 2 \end{array}$$

carry bit
↑
sum bit

Half Adder



$$\begin{aligned} \text{carry} &= x \cdot y \quad // \text{boolean} \\ &x \otimes y \quad // \text{verilog} \end{aligned}$$

$$\begin{aligned} \text{sum} &= \overline{x} \cdot y + x \cdot \overline{y} \quad // \text{boolean} \\ &(\neg x \otimes y) \oplus (x \otimes \neg y) \quad // \text{verilog} \end{aligned}$$

↑ (correct)

simplify

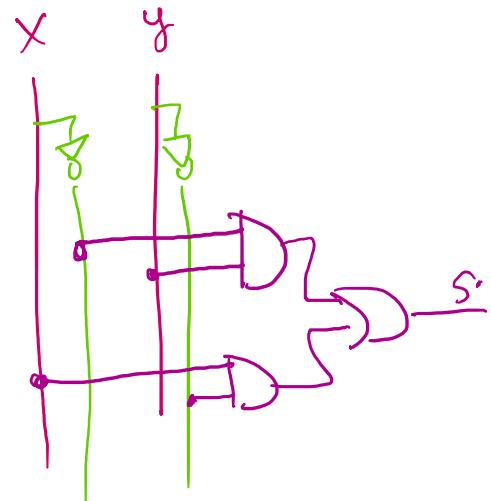
$$\boxed{\text{sum} = x \oplus y} \quad // \text{boolean}$$

$$\text{sum} = x \wedge y_i \quad // \text{verilog}$$

Half Adder

<u>x</u>	<u>y</u>	<u>carry</u>	<u>sum</u>
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

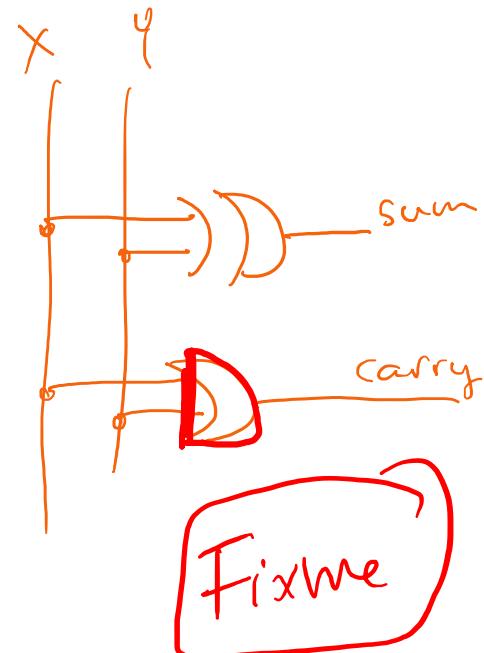
also xor?



OR

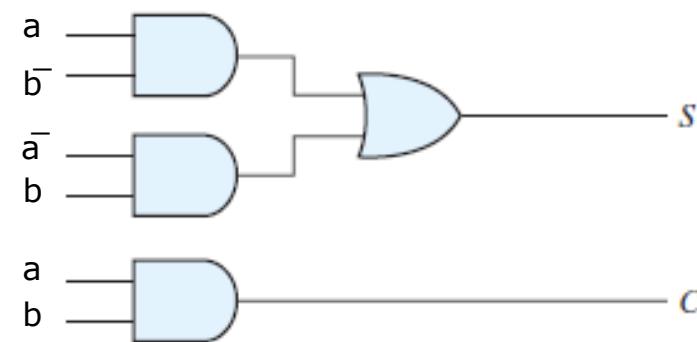
$$\text{sum} = \overline{x}y + \overline{x}\overline{y} = x \oplus y$$

$$\text{carry} = xy = x \wedge y$$



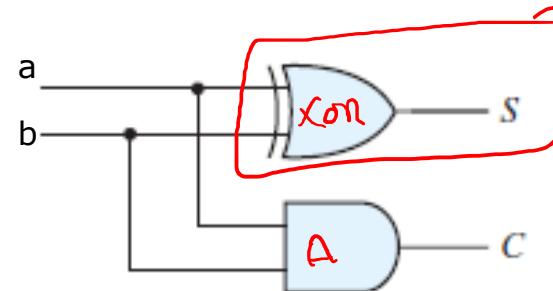
Half Adder

i	a	b	C	S
0	0	0	0	0
1	0	1	0	1
2	1	0	0	1
3	1	1	1	0



$$S = a \bar{b} + \bar{a} b = a \oplus b$$

$$C = a b$$



Binary Addition

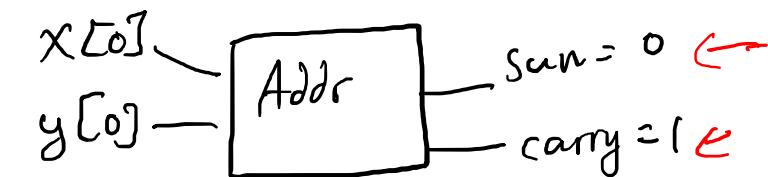
- What if x and y are 2-bits each?

$$x=3, y=3$$

$$\begin{array}{r} 3 \\ + 3 \\ \hline 6 \end{array}$$

$= \begin{array}{r} 0 \textcircled{1} 1 \\ + 0 \textcircled{1} 1 \\ \hline 110 \end{array}$

<u>x</u>	<u>y</u>	<u>carry</u>	<u>sum</u>
1	1	1	0



$$\begin{array}{r} 1 \textcircled{1} \\ 0 \quad 1 \quad 1 \\ + 0 \quad 1 \quad 1 \\ \hline 110 \end{array}$$

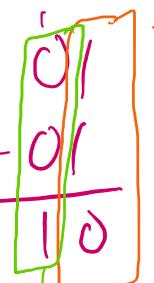
Binary Addition

want to add 2-bit numbers?

$$x = \begin{cases} 00 \\ 01 \\ 10 \\ 11 \end{cases}$$

$$y = \begin{cases} 00 \\ 01 \\ 10 \\ 11 \end{cases}$$

$$\begin{array}{r} x & | \\ + y & + | \\ \hline 2 \end{array}$$



Half Adder

$$\text{sum} = 1 \oplus 1 = 0$$

$$\text{carry} = 1 \mid 1 = 1$$



Half Adder

$$\text{sum} = 0 \oplus 0 = 0$$

$$\text{carry} = 0 \mid 0 = 0$$



Actually need to add 3 things
 x, y , & Carry-in!

Binary Addition

$$\begin{array}{r}
 \text{Cin} \\
 x = \\
 y = \\
 \hline
 \text{Carry} \\
 \text{Sum}
 \end{array}$$

Carry-in

- What if x and y are 2-bits each?

$2^3 = 8$

\downarrow

dec

	x	y	C_{in}	Carry	Sum
0	0	0	0	0	0
1	0	0	0	0	0
2	0	1	0	0	1
3	0	1	1	1	0
4	1	0	0	0	1
5	1	0	1	0	0
6	1	1	0	0	0
7	1	1	1	1	1

$$\begin{aligned}
 \text{Carry} = & \bar{x} \cdot y \cdot C_{in} + \\
 & x \cdot \bar{y} \cdot C_{in} + \\
 & x \cdot y \cdot \bar{C}_{in} + \\
 & x \cdot y \cdot C_{in}
 \end{aligned}$$

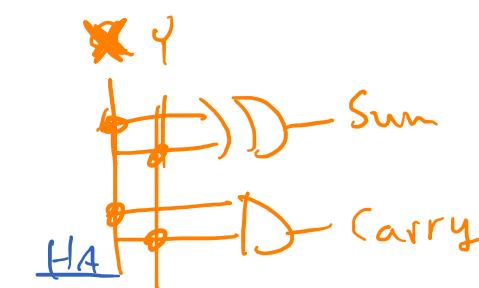
$$\begin{aligned}
 \text{Sum} = & \bar{x} \cdot \bar{y} \cdot C_{in} + \\
 & \bar{x} \cdot y \cdot \bar{C}_{in} + \\
 & x \cdot \bar{y} \cdot \bar{C}_{in} + \\
 & x \cdot y \cdot C_{in}
 \end{aligned}$$

Full Adder

<u>X</u>	<u>y</u>	<u>C_{in}</u>	<u>C_{out}</u>	<u>Sum</u>
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
0	0	0	0	0
1	0	1	0	1
1	1	0	1	0
1	1	1	1	1

Full Adder

<u>X</u>	<u>Y</u>	<u>C_{in}</u>	<u>C_{out}</u>	<u>Z</u>
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	0	1
1	0	0	0	1
1	0	1	0	1
1	1	0	0	0
1	1	1	0	1



$$\begin{array}{r}
 \text{Sum} \\
 0 \\
 1 \\
 1 \\
 0 \\
 1 \\
 0 \\
 1 \\
 1 \\
 \end{array}$$

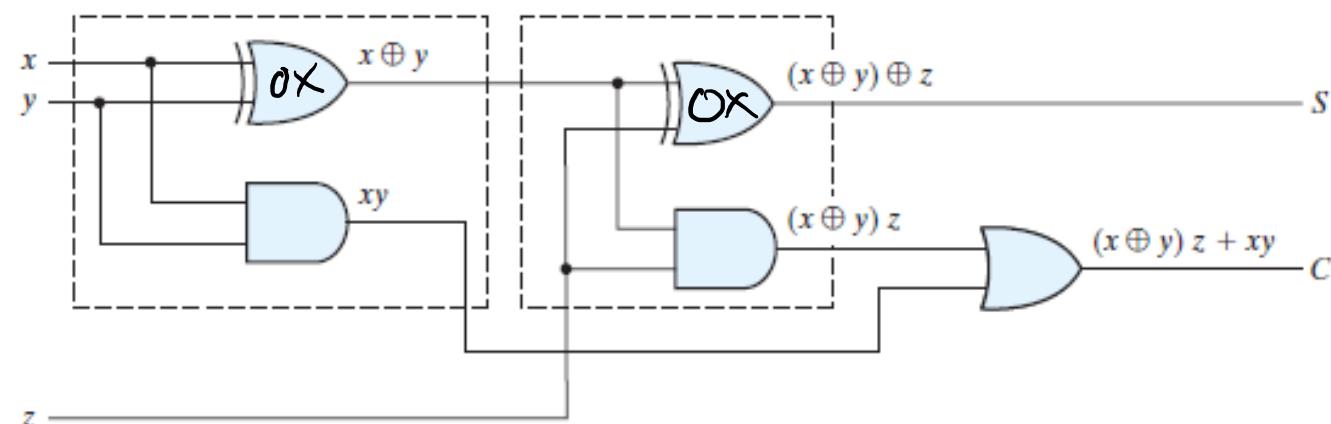
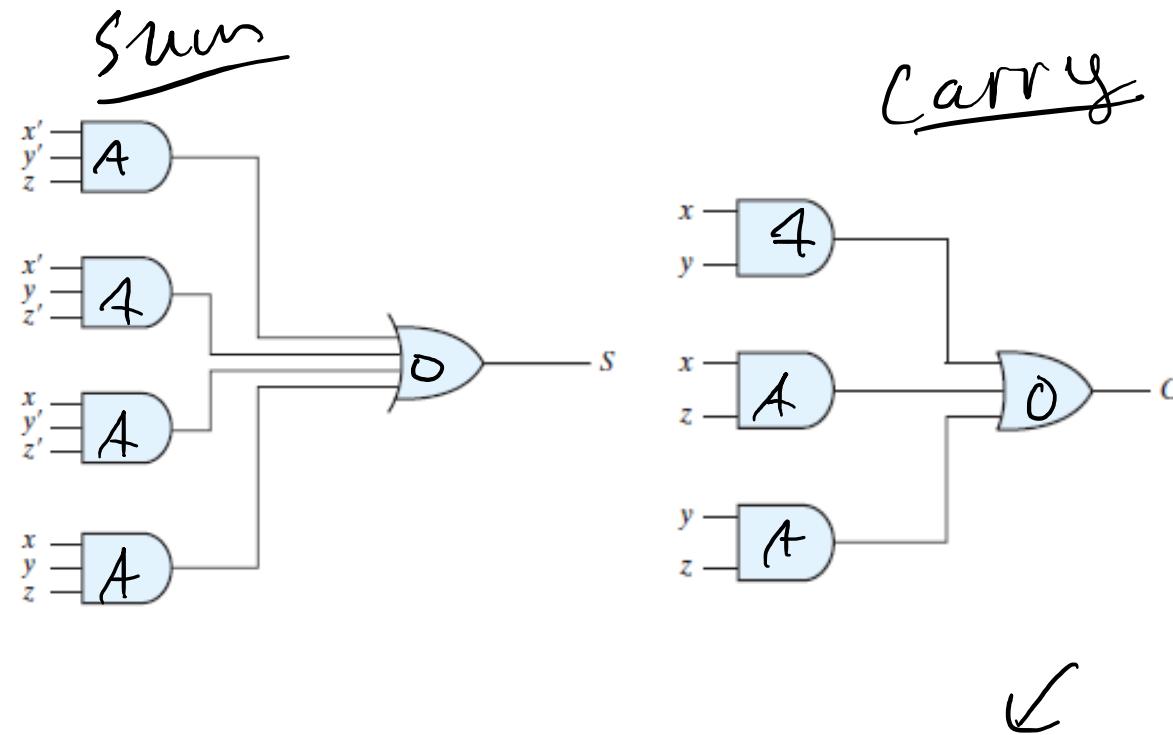
$$\begin{array}{r}
 X \\
 + Y \\
 \hline
 Z \\
 + C_{in} \\
 \hline
 \text{Sum}
 \end{array}$$

$$\text{Sum} = (Z = X \oplus Y) \oplus C_{in} = X \oplus Y \oplus C_{in}$$

P2: use "+" in Verilog

Full Adder

i	x	y	z	c	s
0	0	0	0	0	0
1	0	0	1	0	1
2	0	1	0	0	1
3	0	1	1	1	0
4	1	0	0	0	1
5	1	0	1	1	0
6	1	1	0	1	0
7	1	1	1	1	1

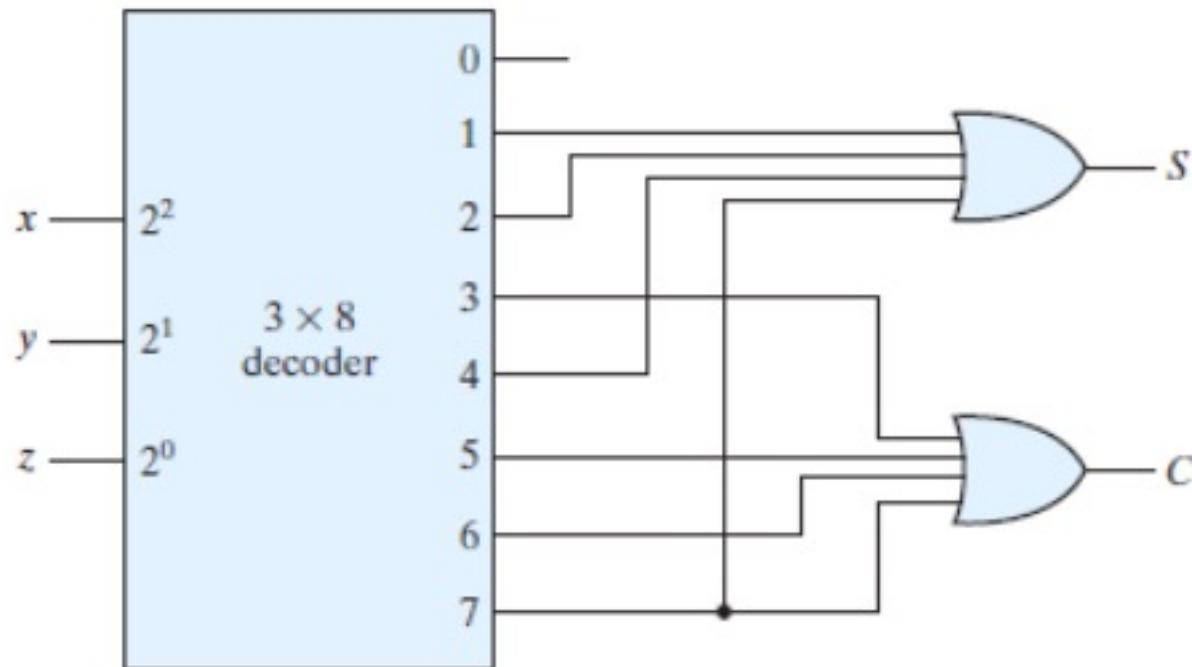


Full Adder (Skipped)

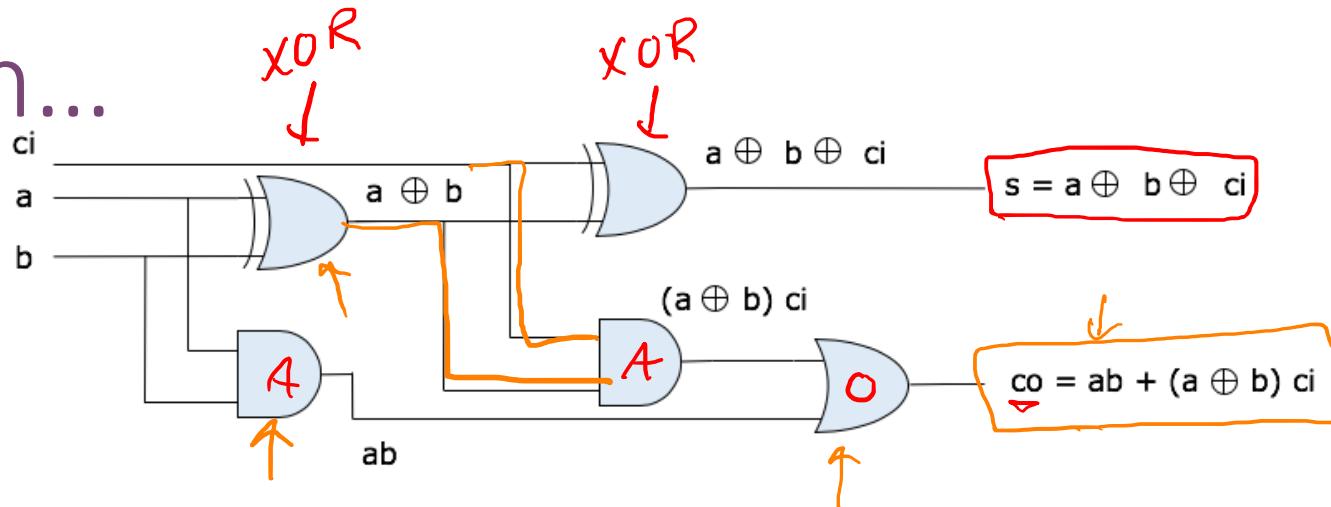
$$S(x, y, z) = \{1, 2, 4, 7\}$$

$$C(x, y, z) = \{3, 5, 6, 7\}$$

i	x	y	z	C	S
0	0	0	0	0	0
1	0	0	1	0	1
2	0	1	0	0	1
3	0	1	1	1	0
4	1	0	0	0	1
5	1	0	1	1	0
6	1	1	0	1	0
7	1	1	1	1	1



Your Turn...



Implement the circuit above in Verilog.

- Hint: XOR in Verilog is '^'

```
module FullAdder (
    input a,b,ci,/* Ci = Carry_in */
    output s, co /* Co = Carry_out */
);
```

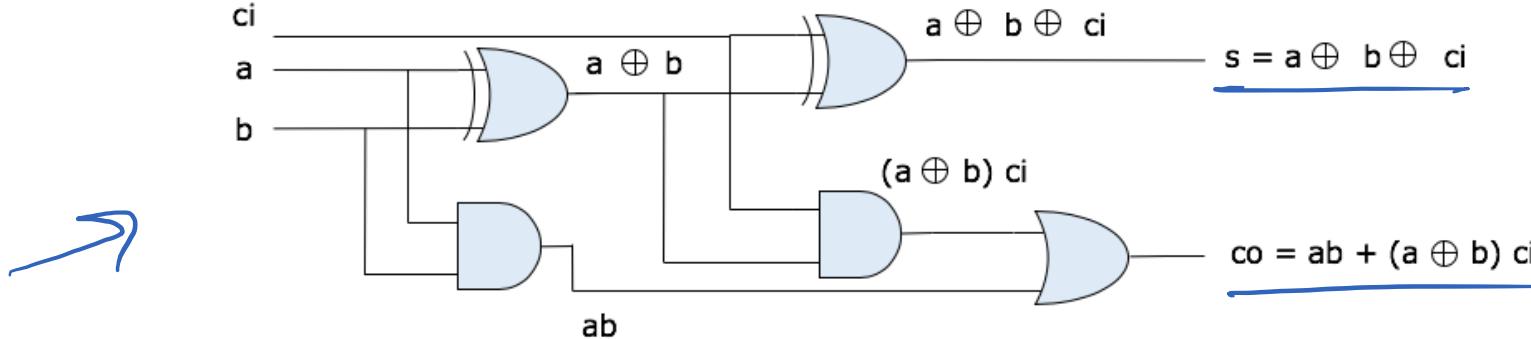
// Your Code Here!

```
assign s = a ^ b ^ ci;
assign co = (a & b) | ((a & b) & ci);
```

endmodule

logic [7:0] x = a + b;

1-Bit “Full” Adder



```
module FullAddr (
    input a,b,ci,
    output s, co
);
```

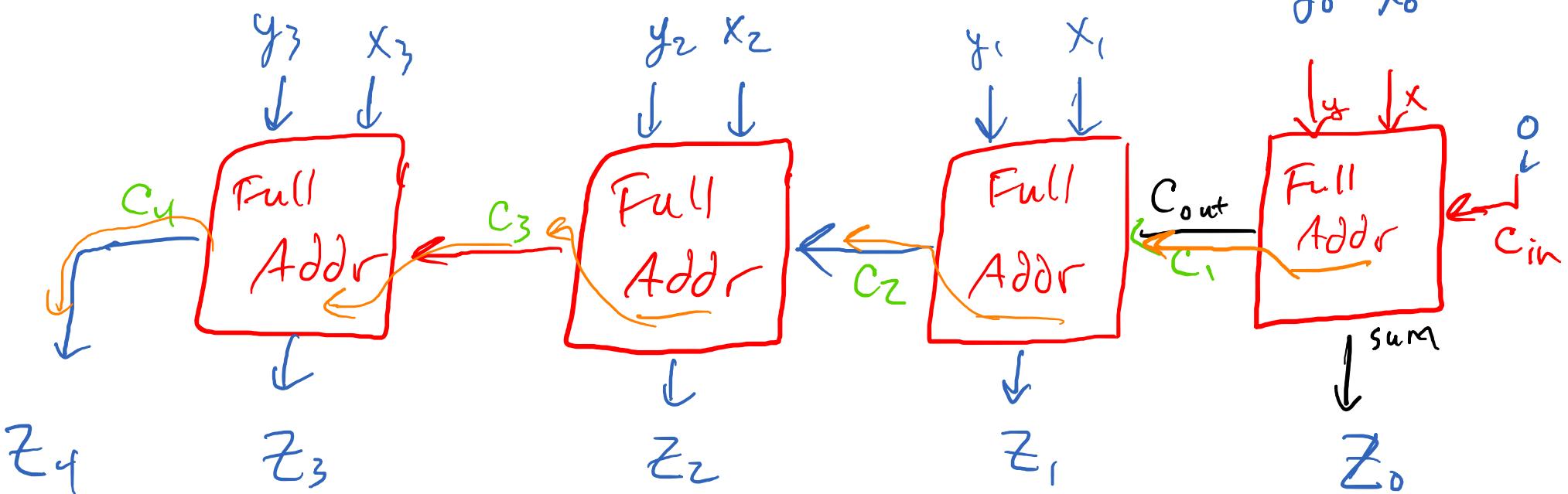
```
assign s = a ^ b ^ ci;
assign co = (a & b) | ((a ^ b) & ci);
```

```
endmodule
```

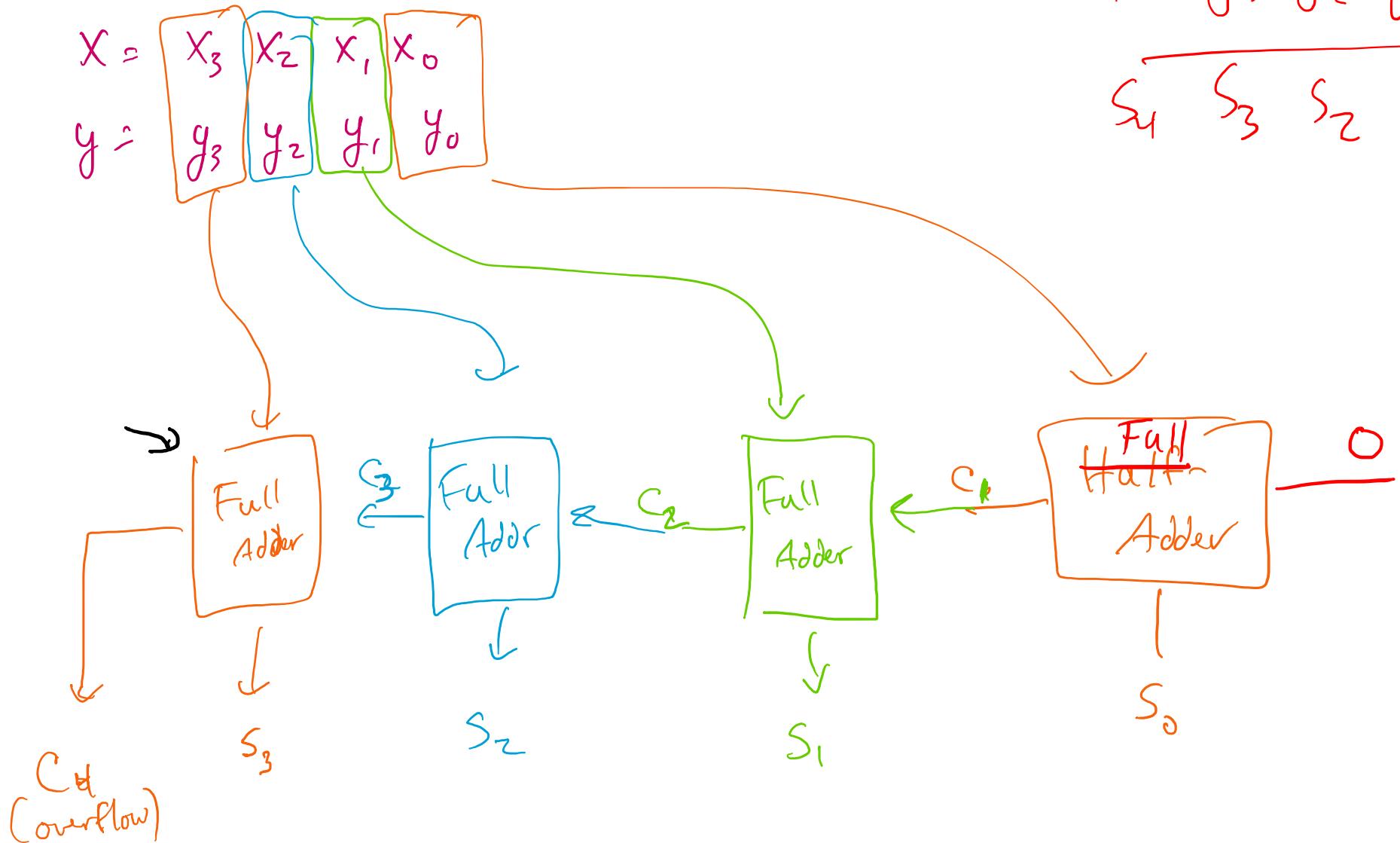
Ripple-Carry Adder

$$\begin{array}{r}
 X = \begin{array}{c} C_4 \\ | \\ X_3 \end{array} \quad \begin{array}{c} C_3 \\ | \\ X_2 \end{array} \quad \begin{array}{c} C_2 \\ | \\ X_1 \end{array} \quad \begin{array}{c} C_1 \\ | \\ X_0 \end{array} \\
 + \quad \begin{array}{c} y_3 \\ | \\ y_3 \end{array} \quad y_2 \quad y_1 \quad y_0 \\
 \hline
 Z_4 \quad Z_3 \quad Z_2 \quad Z_1 \quad Z_0
 \end{array}$$

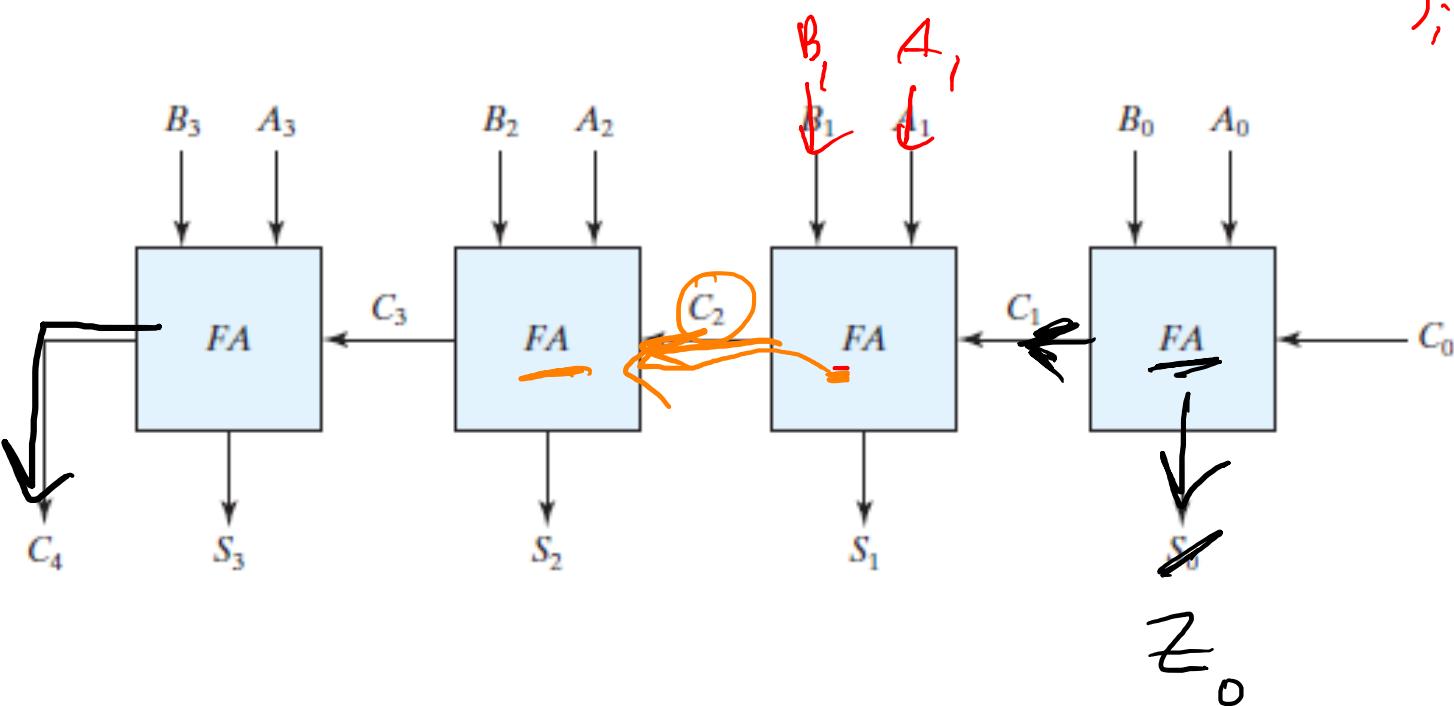
$$\begin{array}{l}
 C_1 \quad | \\
 | \\
 X_1 = | \\
 | \\
 y_1 = \frac{+ \quad 0}{1 \quad 0}
 \end{array}$$



Ripple-Carry Adder



Ripple-Carry Adder



```

module adder (
    input [3:0] a;
    input [3:0] b;
    output [4:0] z
);
    [4:0] c;
    FA fa0(.a(a[3]),
            .b(b[3]),
            .cin('h0),
            .sum(z[3]),
            .cout(c[3]));
    FA fa1(.a(a[2]),
            .b(b[2]),
            .cin(c[3]),
            .sum(z[2]),
            .cout(c[2]));
    assign z[4] = c[4];

```

module adder (

- input [3:0] a;
- input [3:0] b;
- output [4:0] z

);

[4:0] c;

FA fa0(.a(a[3]),

- .b(b[3]),
- .cin('h0),
- .sum(z[3]),
- .cout(c[3]),

FA fa1(.a(a[2]),

- .b(b[2]),
- .cin(c[3]),
- .sum(z[2]),
- .cout(c[2]),

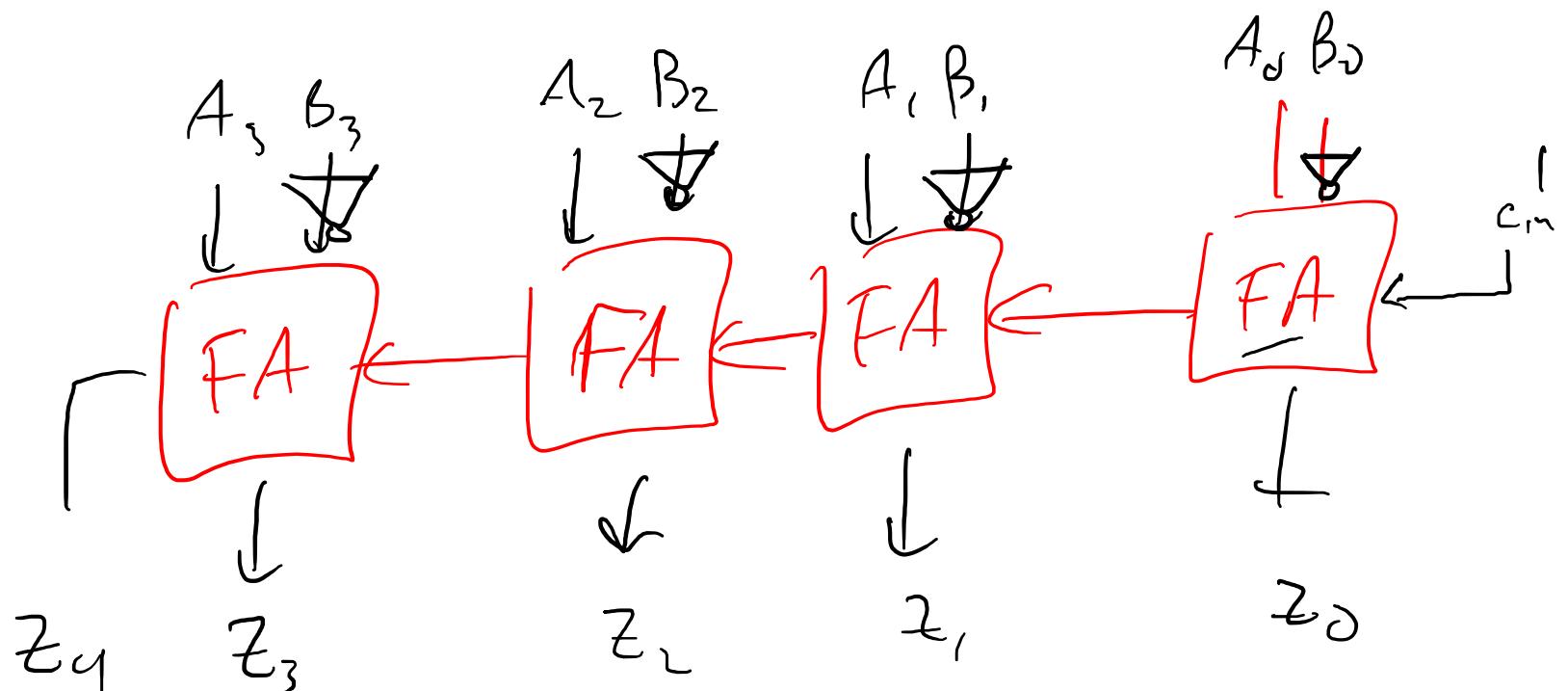
assign z[4] = c[4];

$$A - \bar{B} = A + (\bar{B}) =$$

$$A + \underline{\bar{B}} + \underline{1}$$

Subtraction with Adders?

- We've done $A+B$, what about $A-B$?



Subtraction with Adders?

- We've done $A+B$, what about $A-B$?

$$\begin{array}{r} 5 \\ - 2 \\ \hline 3 \end{array}$$

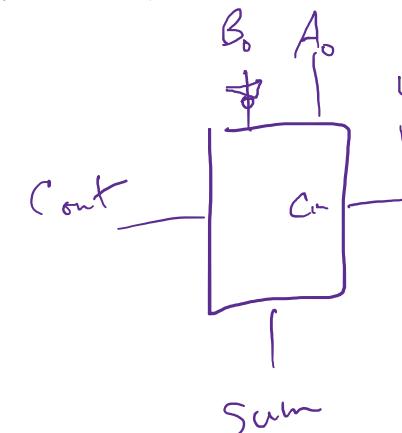
$$\begin{array}{r} 0101 \\ - 0010 \\ \hline \end{array}$$

$$\begin{array}{r} 2^3 \text{ Comp} \\ 1101 \\ + 1 \\ \hline 1110 = -2 \end{array}$$

$$A - B =$$

$$A + (-B) =$$

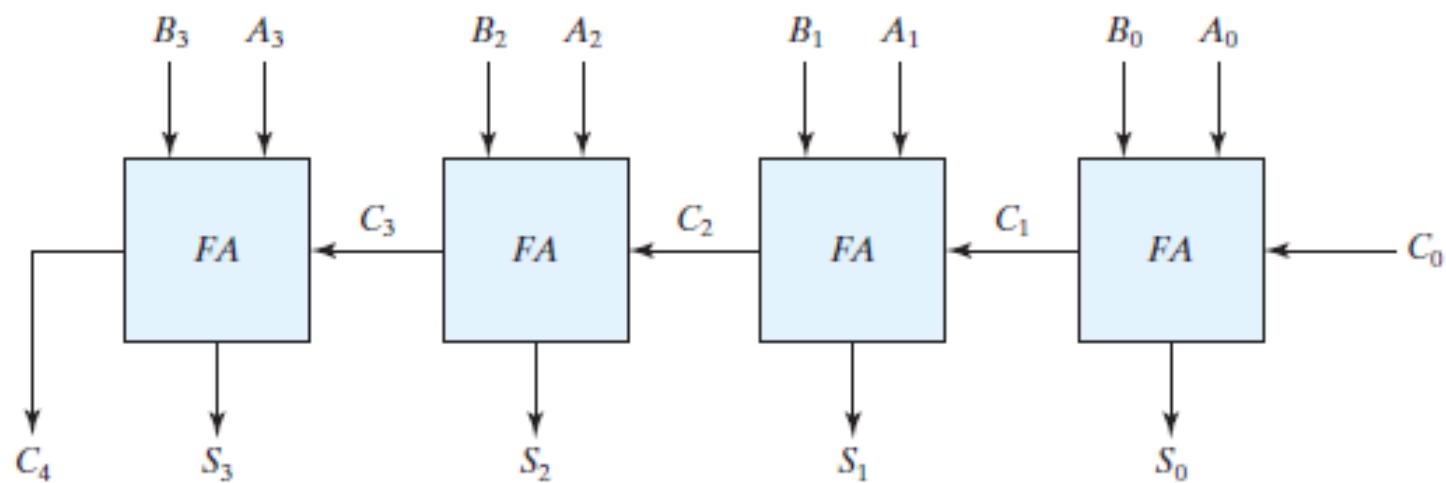
$$A + (\neg B + 1)$$



$$\begin{array}{r} 5 \\ + 2 \\ \hline 3 \end{array}$$

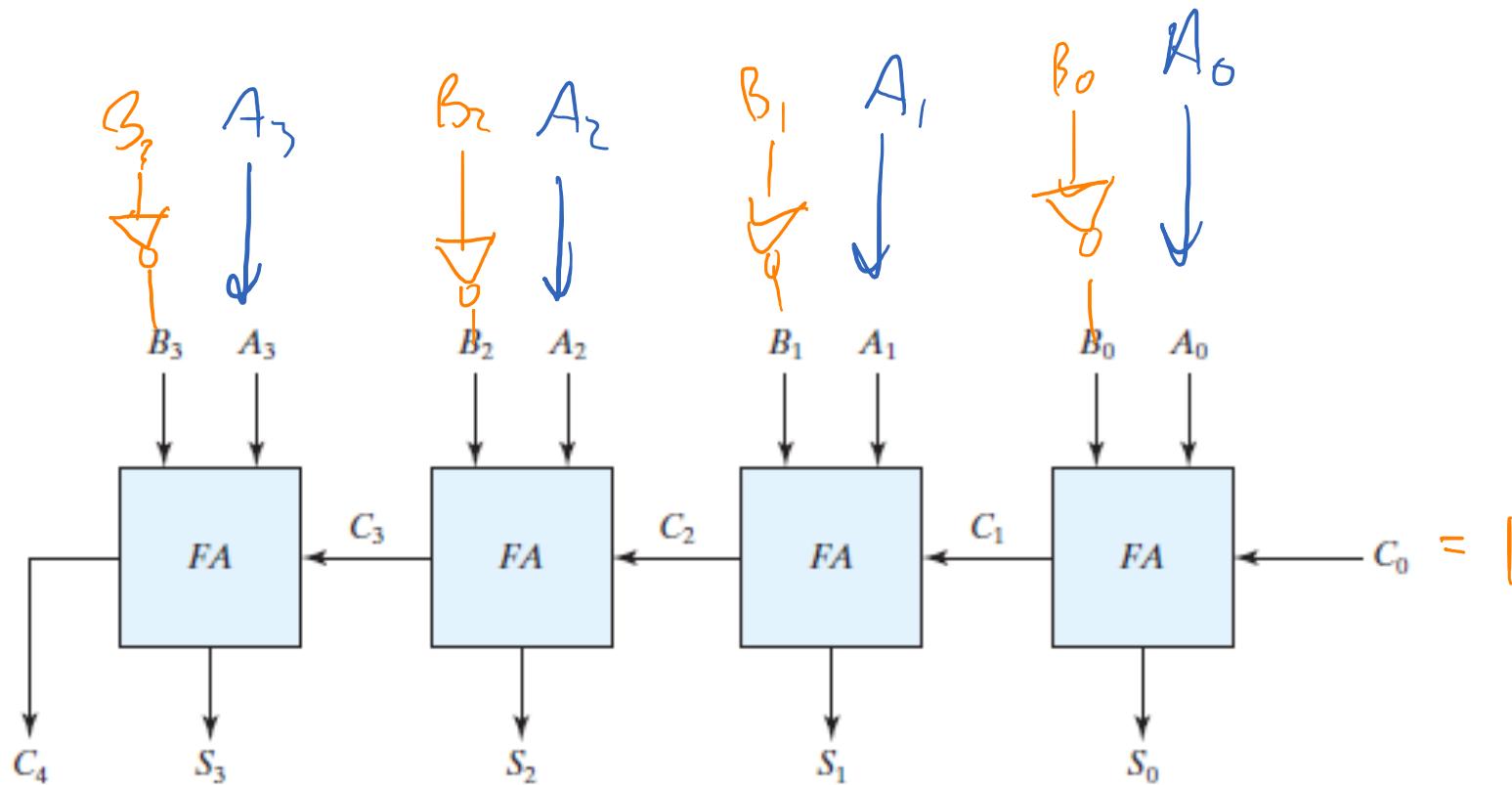
$$\begin{array}{r} 0101 \\ + 1110 \\ \hline 0011 = +3 \quad \checkmark \end{array}$$

Subtraction with Adders?



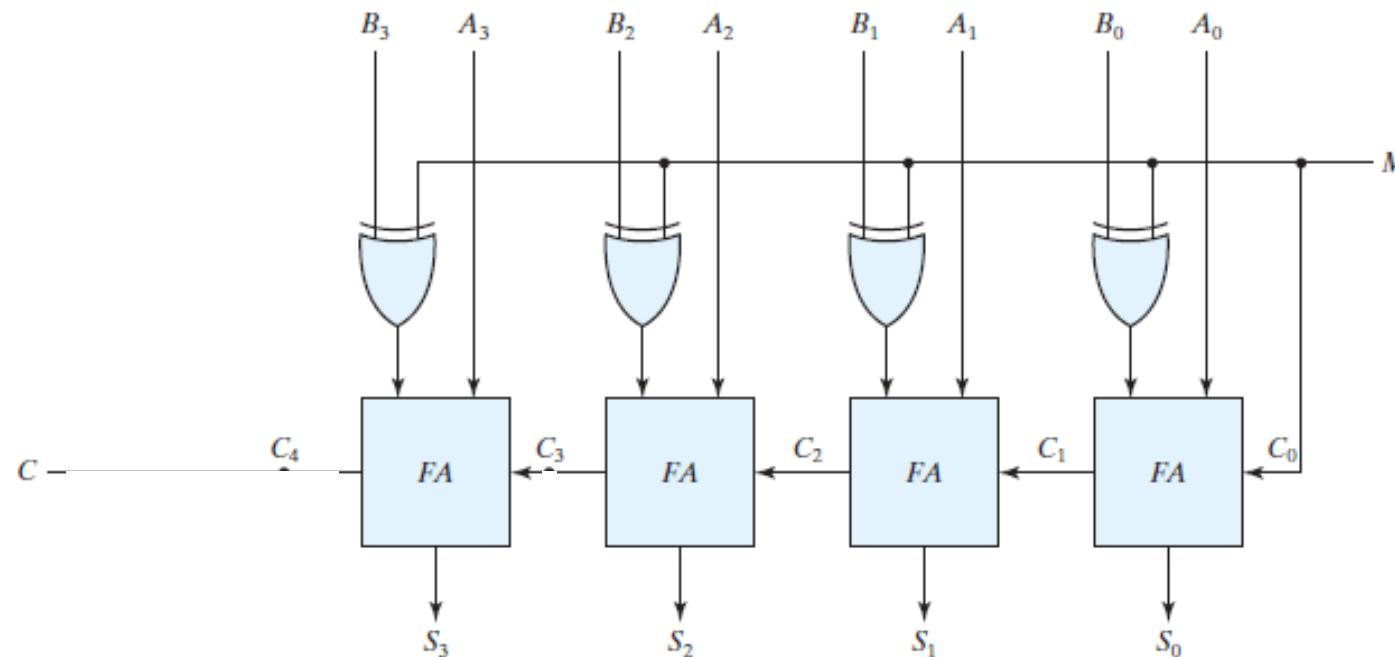
Subtraction with Adders?

$$A - B = A + (-B) = A + (\neg B + 1)$$



Adder/Subtractor

- Mode input:
 - If $M = 0$, then $S = A + B$, the circuit performs addition
 - If $M = 1$, then $S = A + \bar{B} + 1$, the circuit performs subtraction



Overflow for signed numbers?

- Unsigned

$$\begin{array}{r} 10 \\ + 8 \\ \hline 18 \end{array}$$

- Signed

$$\begin{array}{r} 5 \\ + 6 \\ \hline 11 \end{array}$$

Overflow for signed numbers?

$$\begin{array}{r} 10 \\ + 8 \\ \hline 18 \end{array}$$

$$\begin{array}{r} 1010 \\ + 1000 \\ \hline \text{carry } \underline{10010} \end{array} \text{ sum}$$

unsigned = carry out bit
"overflow"

$$\begin{array}{r} \text{Signed} \\ 5 \\ + 6 \\ \hline 11 \end{array}$$

$$\begin{array}{r} 0101 \\ + 0110 \\ \hline \boxed{01011} \end{array}$$

carry = 0 \Rightarrow no overflow?

$$5 = 0101 \quad 1010 \rightarrow 1011$$

$$6 = 0110$$

$$= -(0100+) = -(0101) = -5 \leftarrow \text{overflow!}$$

Overflow for signed numbers?

$$\begin{array}{r} -2 \\ + -1 \\ \hline -3 \end{array}$$

$$\begin{array}{r} +2 \\ + -1 \\ \hline -1 \end{array}$$

Overflow for signed numbers?

$$\begin{array}{r} -2 \\ + -1 \\ \hline -3 \end{array} \quad \begin{aligned} -(0010) &= 1101 + 1 = 1110 \Rightarrow \\ -(0001) &= 1110 + 1 = 1111 \end{aligned}$$
$$\begin{array}{r} 1110 \\ + 1111 \\ \hline \boxed{11101} \end{array} \begin{aligned} \text{sum} &= -(0010+1) \\ &= -(0011) = -3 \end{aligned}$$

No overflow?

$$\begin{array}{r} +2. \quad 0010. \quad 0010 \\ + -1. \quad +- (0001) \quad + 1111 \\ \hline +1 \end{array} \quad 10001 \leftarrow \text{no overflow}$$

stopped
here!

Overflow for signed numbers

$$\begin{array}{r} \text{XXXX} \\ + \text{YYYY} \\ \hline \text{ZZZZ} \end{array}$$

Same = no overflow

different = overflow



XOR (C_4, C_5)

Signed numbers
Only!

unsigned = regular
carry

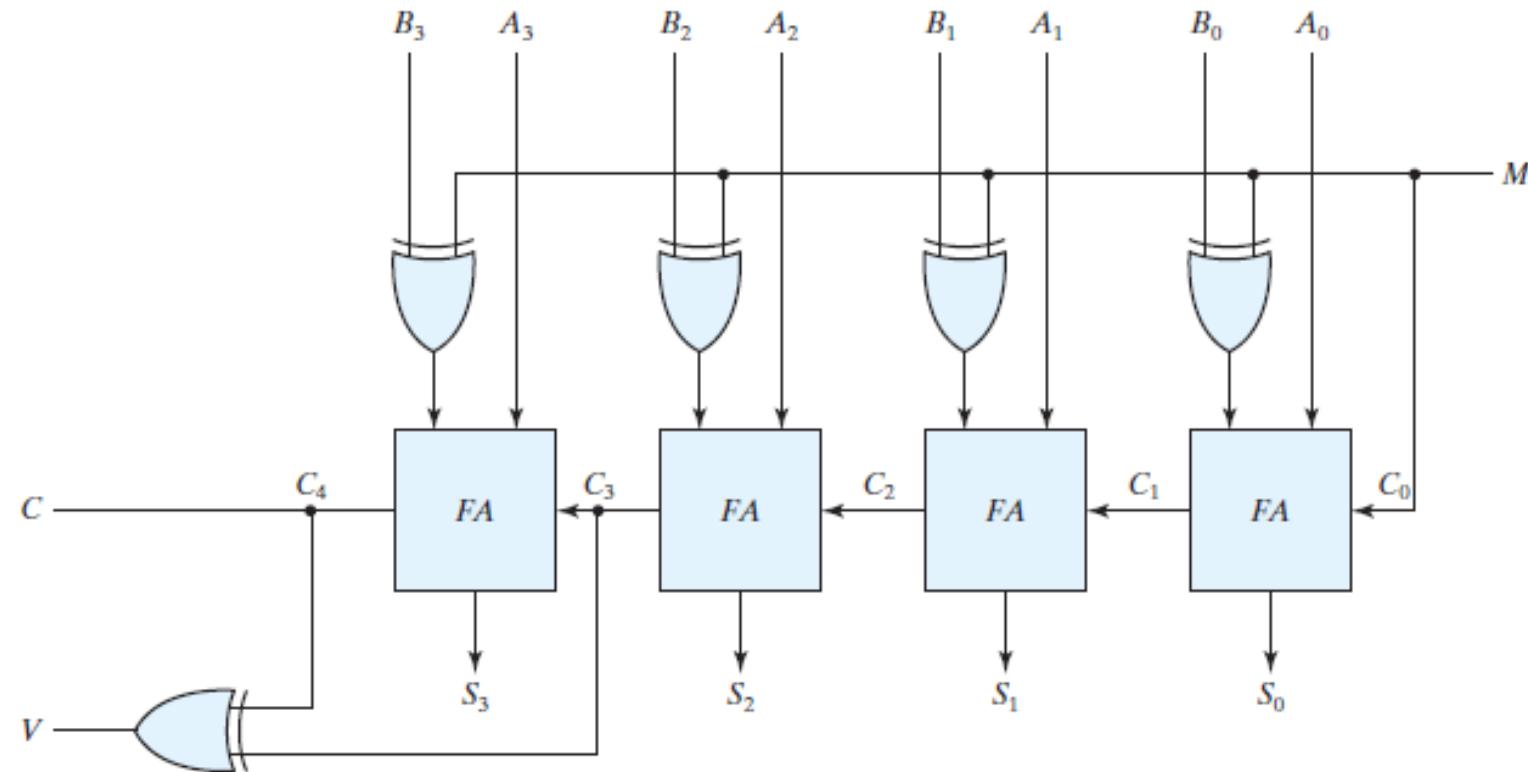
Overflow detection

- When two numbers with n digits each are added and the sum is a number occupying $n + 1$ digits, we say that an overflow occurred.
- The detection of an overflow after the addition of two binary numbers depends on whether the numbers are considered to be signed or unsigned.
- When two unsigned numbers are added, an overflow is detected from the end carry out of the most significant position.
- In case of signed numbers, two details are important:
 - the leftmost bit always represents the sign,
 - negative numbers are in 2's-complement form.
- When two signed numbers are added:
 - the sign bit is treated as part of the number
 - the end carry does not indicate an overflow.

Overflow detection

- An overflow cannot occur after an addition if one number is positive and the other is negative, since adding a positive number to a negative number produces a result whose magnitude is smaller than the larger of the two original numbers.
- An overflow may occur if the two numbers added are both positive or both negative.
- An overflow condition can be detected by observing the carry into the sign bit position and the carry out of the sign bit position.
 - If these two carries are equal, there was no overflow.
 - If these two carries are not equal, an overflow has occurred.
- If the two carries are applied to an exclusive-OR gate, an overflow is detected when the output of the gate is equal to 1.

Adder with overflow detection



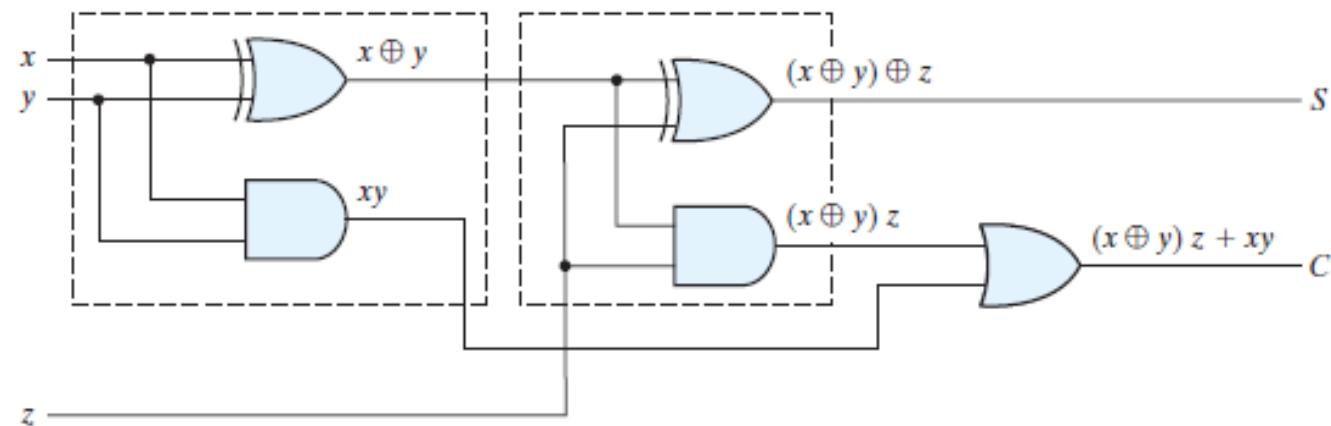
Gate Delay

- Gates are not magic, they are physical
- Takes time for changes flow through
- Assume 5ps (5E-12) / gate

- How fast can we update our adder?

Full Adder Gate Delay

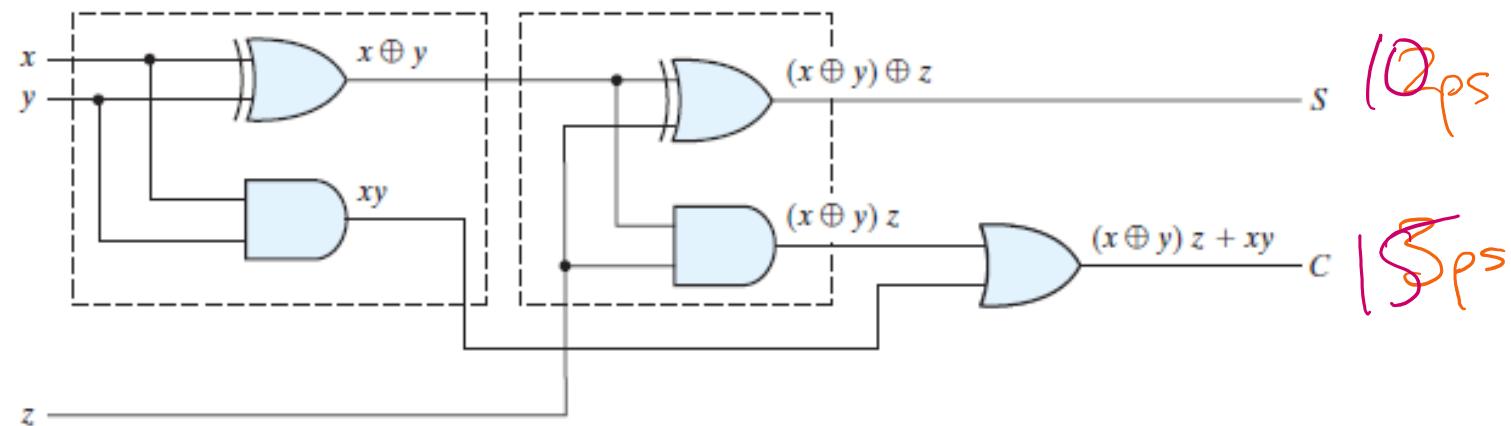
- Assume 5ps/gate



- What is the total delay on s ? on c ?

Full Adder Gate Delay

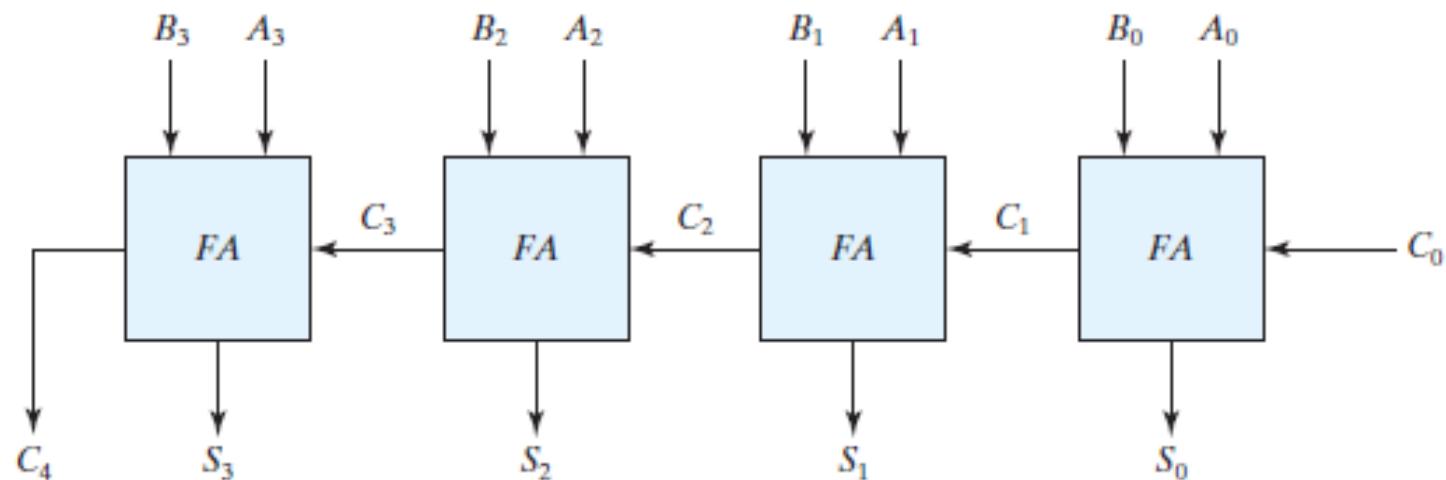
- Assume 1ps/gate \swarrow



- What is the total delay on s ? on c ?

Ripple-Carry Gate Delays

- What is the total delay here?



Adder Gate Delays

- What is the total delay for:
 - 1-bit addition:
 - 4-bit addition:
 - 8-bit addition:
 - 16-bit addition:
 - 32-bit addition:
 - 64-bit addition:

Adder Gate Delays

- What is the total delay for:

- 1-bit addition:

15 ps

- 4-bit addition:

60 ps

- 8-bit addition:

120 ps

- 16-bit addition:

240 ps

- 32-bit addition:

480 ps

- 64-bit addition:

960 ps = ~ 1 GHz

Faster Adder Options?

- What can be done to build a faster 64-bit adder?

Next Time

- Latches / Flip-Flops