

Testing

ENGR 210 / CSCI B441

Addition / Subtraction

Andrew Lukefahr

Announcements

- P2 is out! Due Friday
- Office Hours: Posted on Syllabus

Addition ✓
Subtraction =



2-BeltAlarm testbench

```
timescale 1ns/1ps
module tb();
    reg k, p, s;
    wire alarm;
    st_pas sb_pas
    st_drv sb_drv
    pas /drv
    l
    Two
    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 TwoBeltAlarm(
    input k, st_pas, sb_pas,
    input st_drv, sb_drv,
    output alarm
);
    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
```

2-BeltAlarm Task

```
task checkAlarm(  
    input kV, pV, sV,  
    input alarmV  
) ;
```

```
k = kV; p=pV; s=sV;  
#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,  
    output alarm  
);  
    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
```

2-BeltAlarm Testing

initial

begin

```
k = 'h0; p = 'h0; s= 'h0;
$monitor ("k:%b p:%b s:%b a:%b", k, p, s, alarm);
checkAlarm('h0,'h0,'h0, 'h0);
checkAlarm('h0,'h0,'h1, 'h0);
checkAlarm('h0,'h1,'h0, 'h0);
checkAlarm('h0,'h1,'h1, 'h0);
checkAlarm('h1,'h0,'h0, 'h0);
checkAlarm('h1,'h0,'h1, 'h0);
checkAlarm('h1,'h1,'h0, 'h1);
checkAlarm('h1,'h1,'h1, 'h0);
$display("@@Passed");
```

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

5 inputs
 $2^5 = 32$ values

2^5 possible inputs

32 possible variations

K st_pas st_drv
↓ ↓ ↓
0 0 0 0 0
0 0 0 0 1
0 0 0 1 0
...
1 1 1 1 0
1 1 1 1 1

end

For Loops in Testbenches

alu-
tb.sv

- 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
```

++i i++ i+=1

alu.sv

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

```
`timescale 1ns/1ps

module tb();

logic k, st_pas, sb_pas, st_drv, sb_drv;
wire alarm;

TwoBeltAlarm dut0 (
    .k(k), .st_pas(st_pas), .sb_pas(sb_pas),
    .st_drv(st_drv), .sb_drv(sb_drv), .alarm(alarm) );

```

```
task checkAlarm(
    input kV, st_pasV, sb_pasV, st_drvV, sb_drvV, alarmV );
#1
    k = kV; st_pas = st_pasV; sb_pas = sb_pasV;
    st_drv = st_drvV; sb_drv = sb_drvV;
#1
    assert(alarm == alarmV) else
        $fatal(1, "bad alarm, expected:%b got:%b", alarmV, alarm);
endtask
```

```

initial begin
    k = 0; st_pas = 'b0; sb_pas = 'b0;
    st_drv = 'b0; sb_drv = 'h0;
    $monitor ("k:%b stPas:%b sbPas:%b stDrv:%b sbDrv:%b, a:%b", k, st_pas, sb_pas, st_drv, sb_drv, alarm);
#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]);
    checkAlarm ( i[4], i[3], i[2], i[1], i[0], 'h0);
end
$display("@@Passed");
end

```

$\overbrace{i = 0}^{\text{dec}}$ $\overbrace{0000}^{\text{binary}}$
 $\overbrace{k}^{\text{st_pas}} \overbrace{0000}^{\text{sb_pas}}$ $\overbrace{st_drv}^{\text{sb_drv}}$

\rightarrow checkAlarm (.k(i[4]), .st_pas(i[3])
 . sb_pas(i[2]), . st_drv(i[1])
 . sb_drv(i[0]), . alarm('h0));

$i = ($ 0000 (←
 $i[4] \uparrow i[3] \uparrow i[2] \uparrow i[1] \uparrow i[0]$

```

initial begin
    k = 0; st_pas = 'b0; sb_pas = 'b0;
    st_drv = 'b0; sb_drv = 'h0;
    $monitor ("k:%b stPas:%b sbPas:%b stDrv:%b sbDrv:%b, a:%b", k, st_pas, sb_pas, st_drv, sb_drv, alarm);
#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

```

Arrays in Verilog

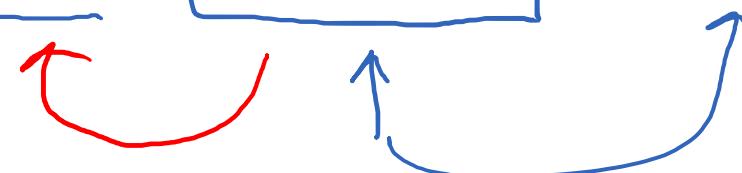
C: int x [s];
type nam El elements

- Bundle multiple wires together to form an array.

→ type [mostSignificantIndex:leastSignificantIndex] name;
type 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[5,4,3,2]
- x[5:2] = {1,0,y,z}; //concatenate 4 signals



Arrays in Verilog

- Can also be used in module definitions

Do use highest: lowest

```
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
```

Do NOT use this

input [0:7] a;

flip endarray

assign $y = \{x[7:0], x[15:8], x[23:16], x[31:24]\}$

$x = \underline{\underline{78}} \underline{\underline{56}} \underline{\underline{34}} \underline{\underline{12}}$ char

Constants in Verilog

- A wire only needs 1 or 0
- Arrays need more bits, how to specify?

C: $y = \underline{\text{O}} \underline{\text{x}} \underline{6};$
' h = hex
d = dec
b = binary

```
logic x;  
x = '50; = 'h0; = 0
```

```
logic [7:0] y;  
assign y[0] = 0;  
y[1] = 1;  
y[2] = 0;
```

assign y = 'h 6 // hexadecimal 6
assign y = 'd 6; // decimal 6
y = 00000110
assign y = 'b 00000110 // binary 6

Constants in Verilog

```
repeat { #times { value}}  
{ 3x 'b01} = 01 01 01  
module mtest;  
    logic [7:0] aa = {1'b0, 1'b1, 1'b0, 1'b0,  
                      1'b1, 1'b0, 1'b0, 1'b0}; // 010101  
    logic [7:0] bb = 8'b01001000; // 8'h48  
    wire [15:0] cc ;  
    logic [7:0] yy = {8{1'b1}}; //concat + repeat  
    logic [7:0] zz = 8'hff; //inferred  
  
    multiply m0(.a(aa), .b(8'h1), .c(cc));  
  
endmodule
```

Addition / Subtraction

Binary Addition

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

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

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

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

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

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

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

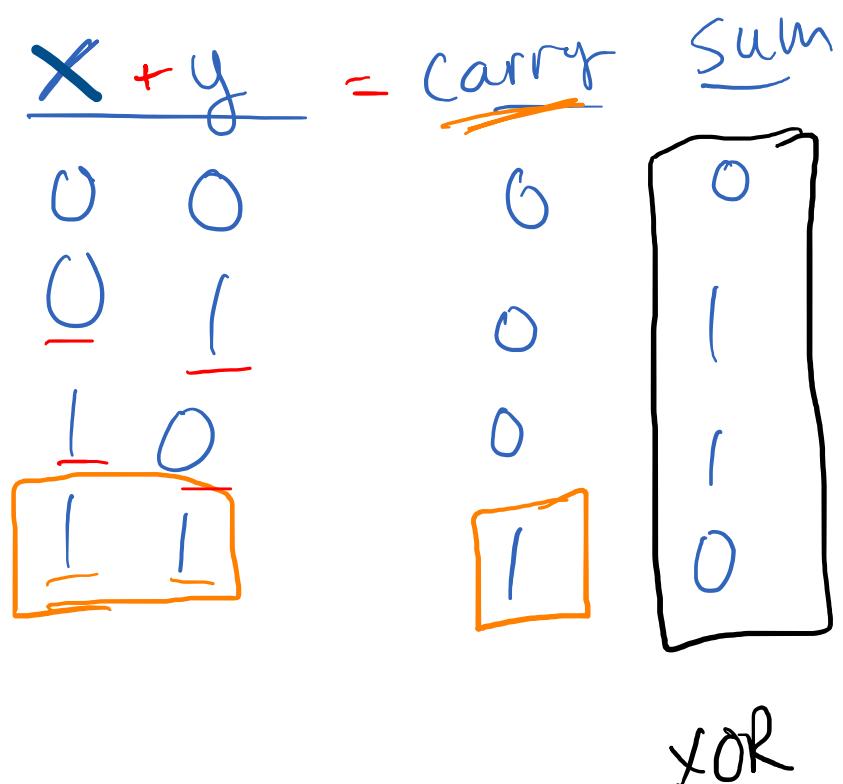
$$\begin{array}{r} 1 \\ + \\ 1 \\ \hline \end{array}$$

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

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

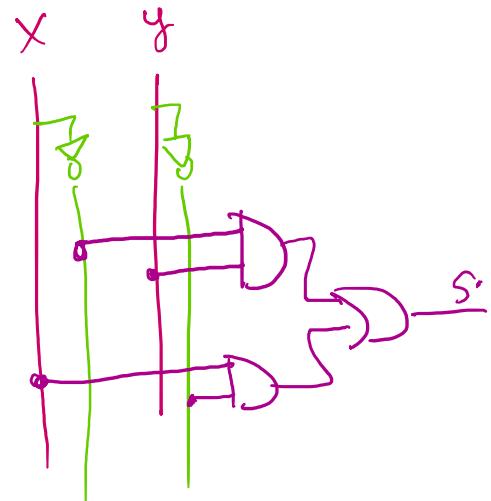
$$\boxed{\text{sum} = x \oplus y} \quad // \text{boolean}$$

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

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

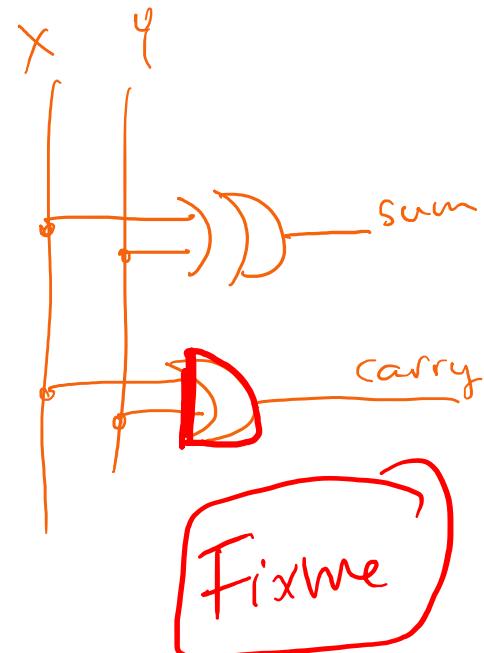
also xor?



OR

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

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



Binary Addition

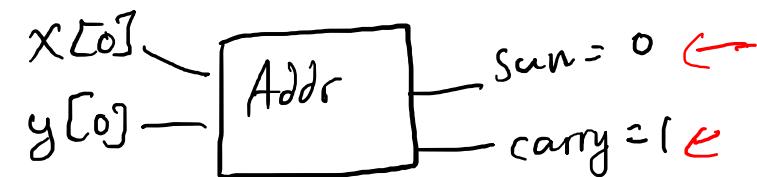
- What if x and y are 2-bits each?

$$x=3, y=3$$

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

$\begin{array}{r} 011 \\ + 011 \\ \hline 110 \end{array}$

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



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

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

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

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

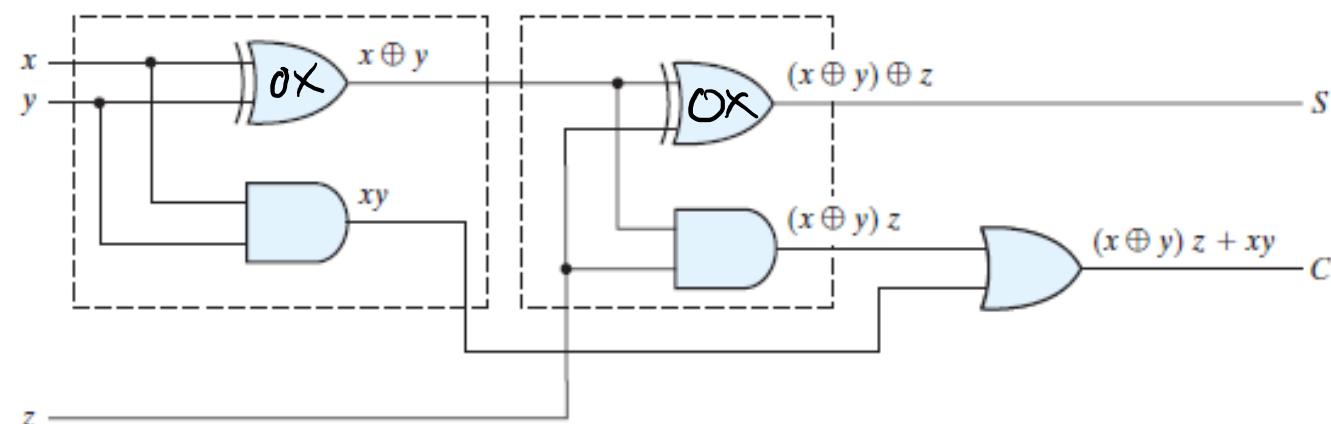
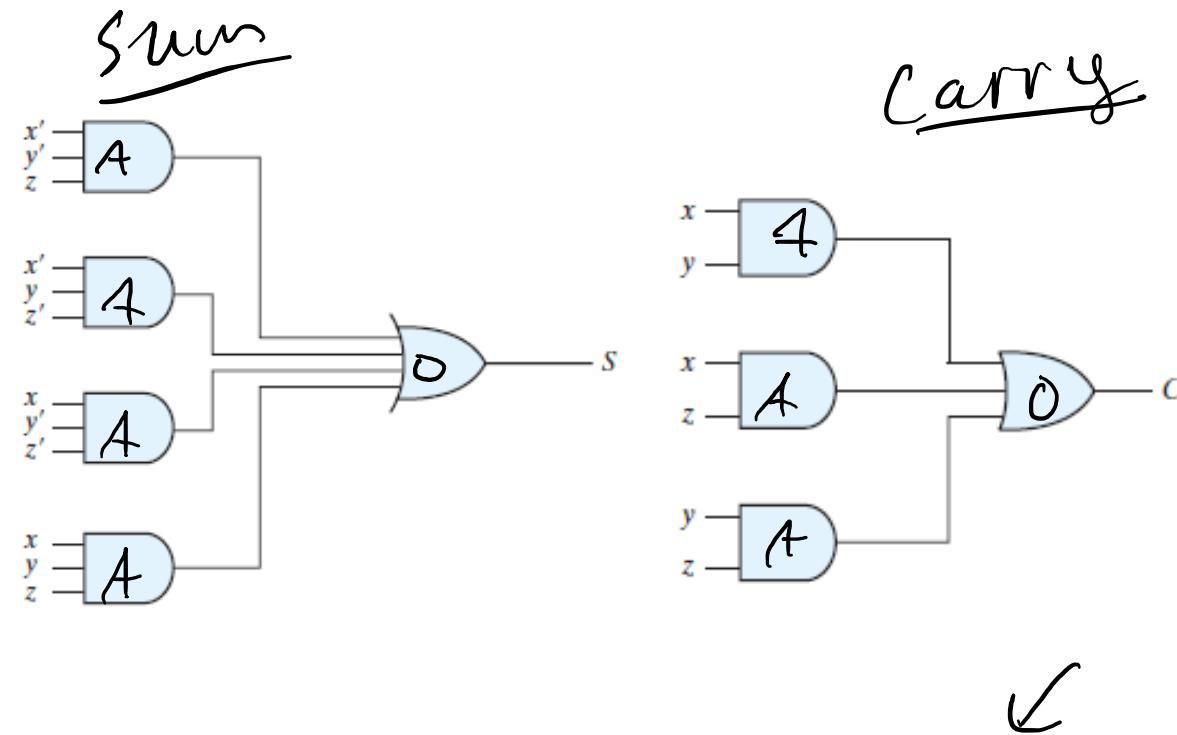
Full Adder

<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

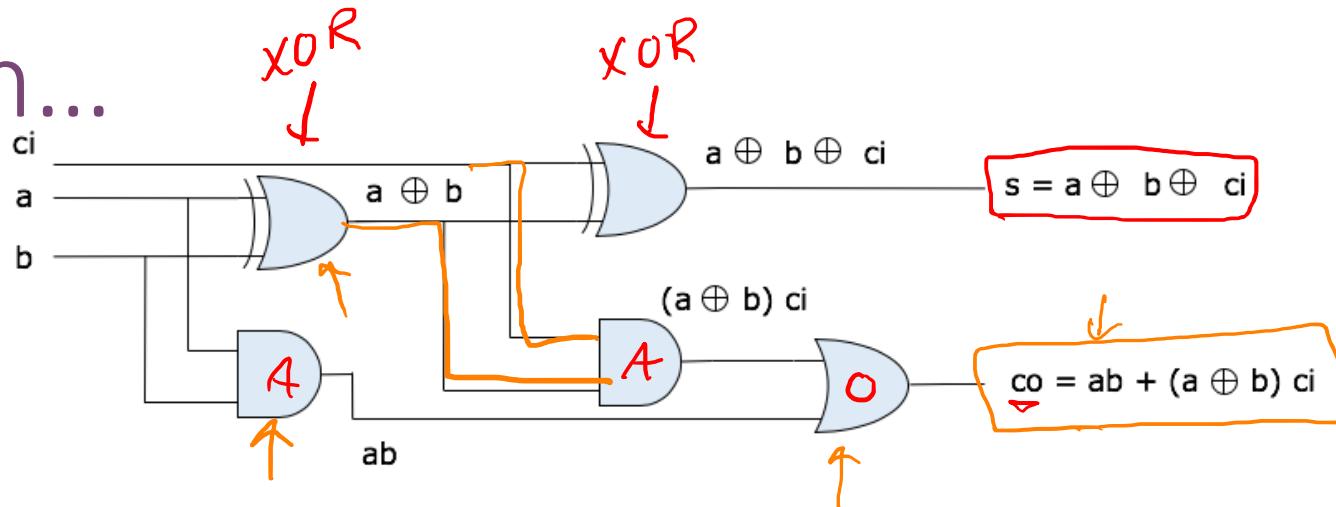
P2: use "+" in Verilog

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



Your Turn...



Implement the circuit above in Verilog.

- Hint: XOR in Verilog is '^'

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

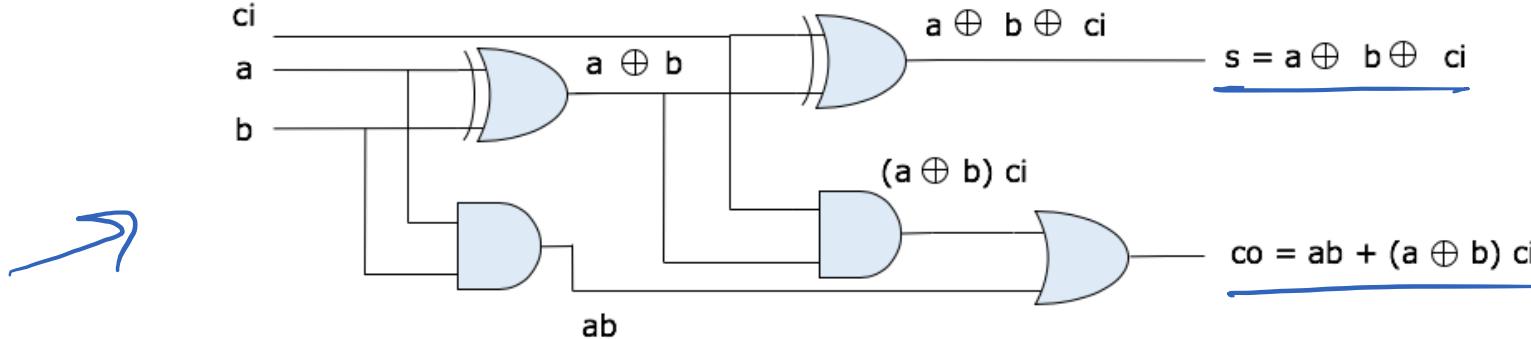
// Your Code Here!

```
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
);
```

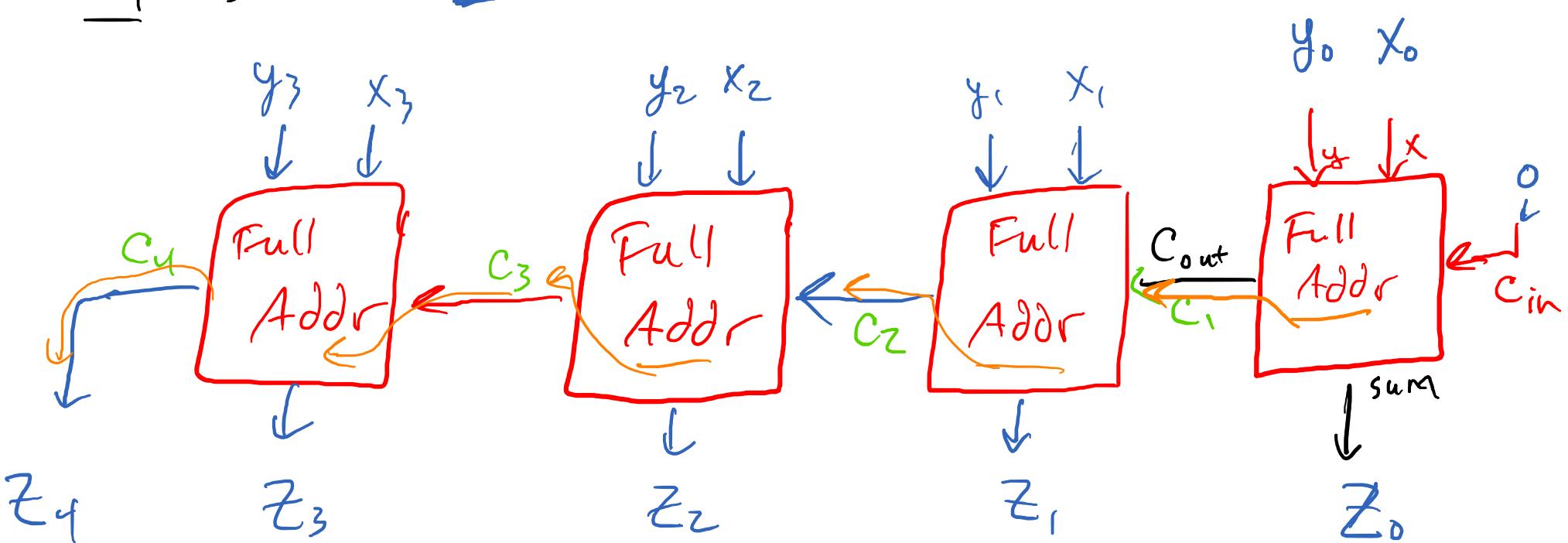
```
assign s = a ^ b ^ ci;
assign co = (a & b) | ((a ^ b) & ci);
```

```
endmodule
```

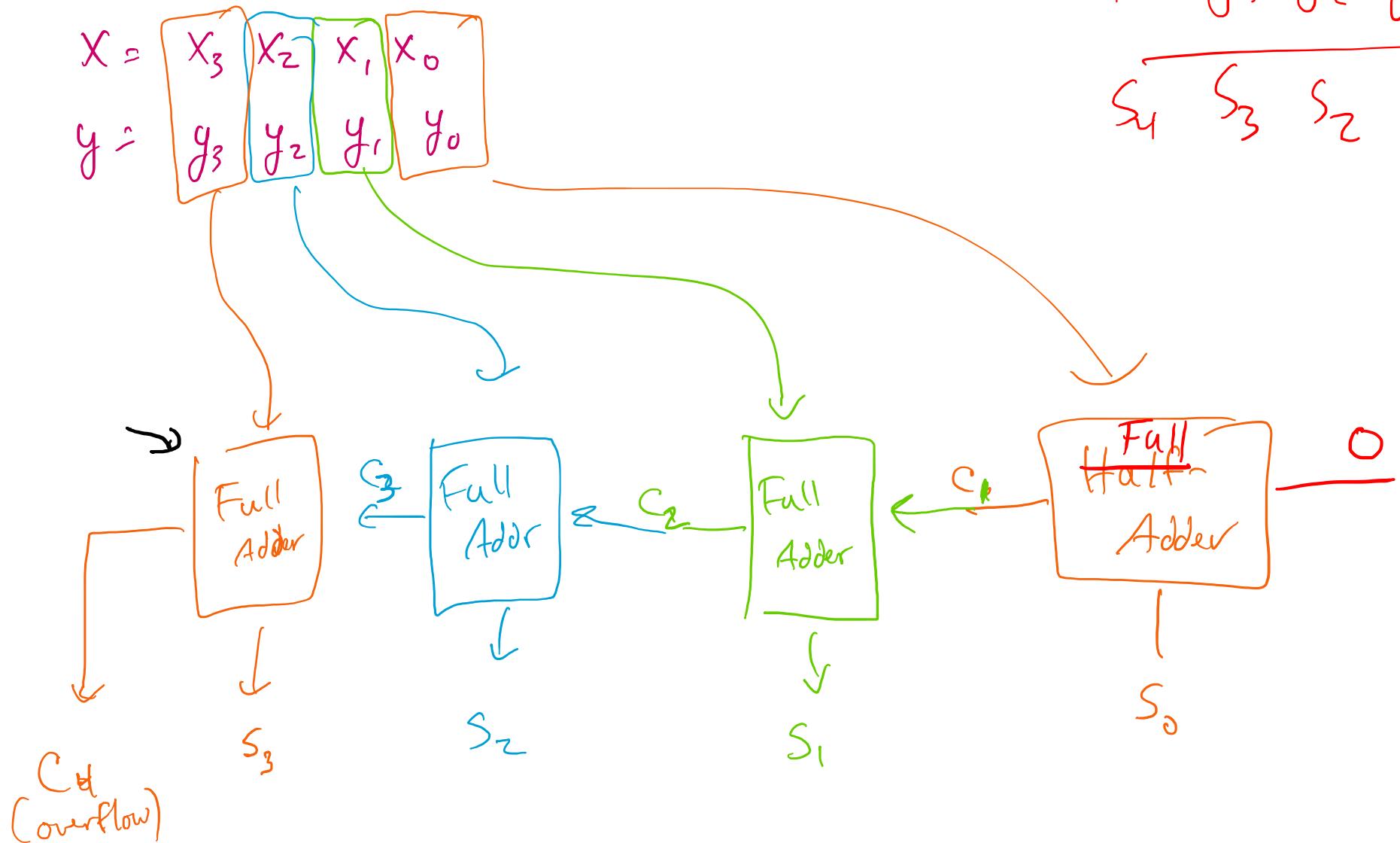
Ripple-Carry Adder

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

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



Ripple-Carry Adder

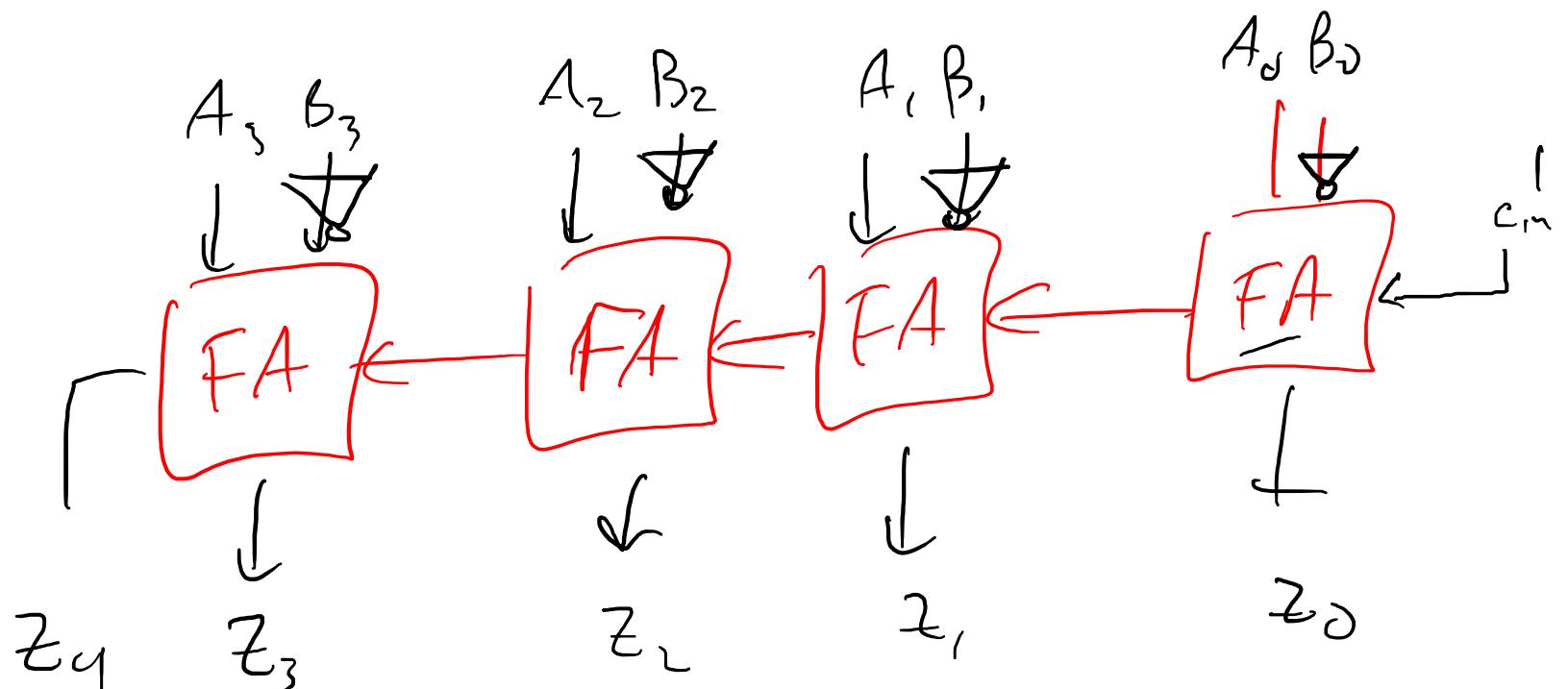


$$A - \bar{B} = A + (\bar{B}) =$$

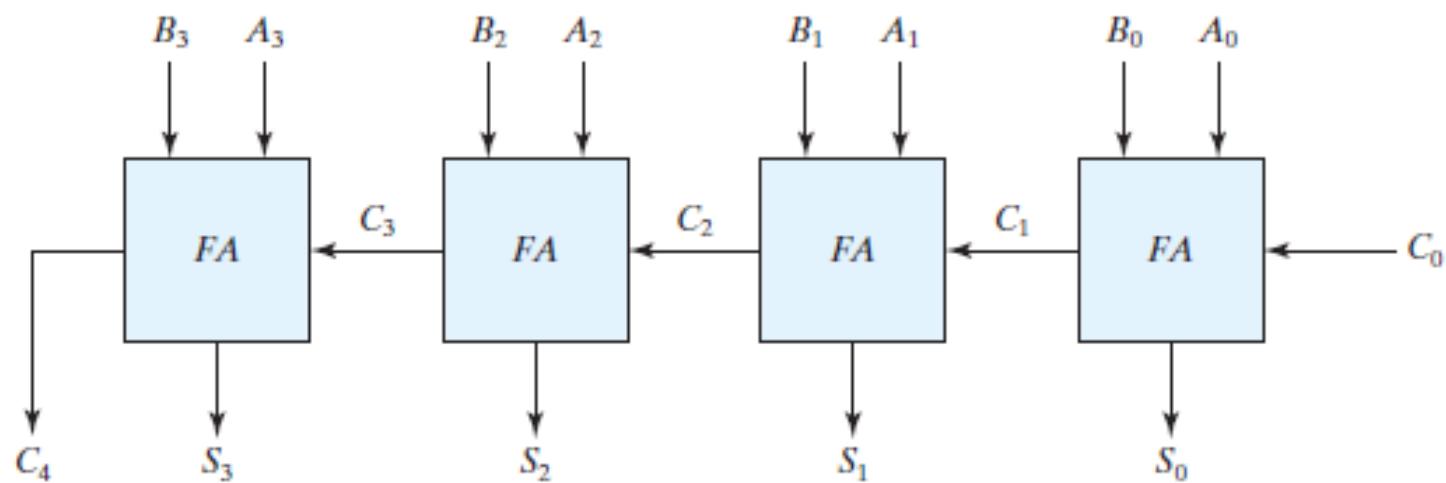
$$A + \underline{\bar{B}} + \underline{1}$$

Subtraction with Adders?

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

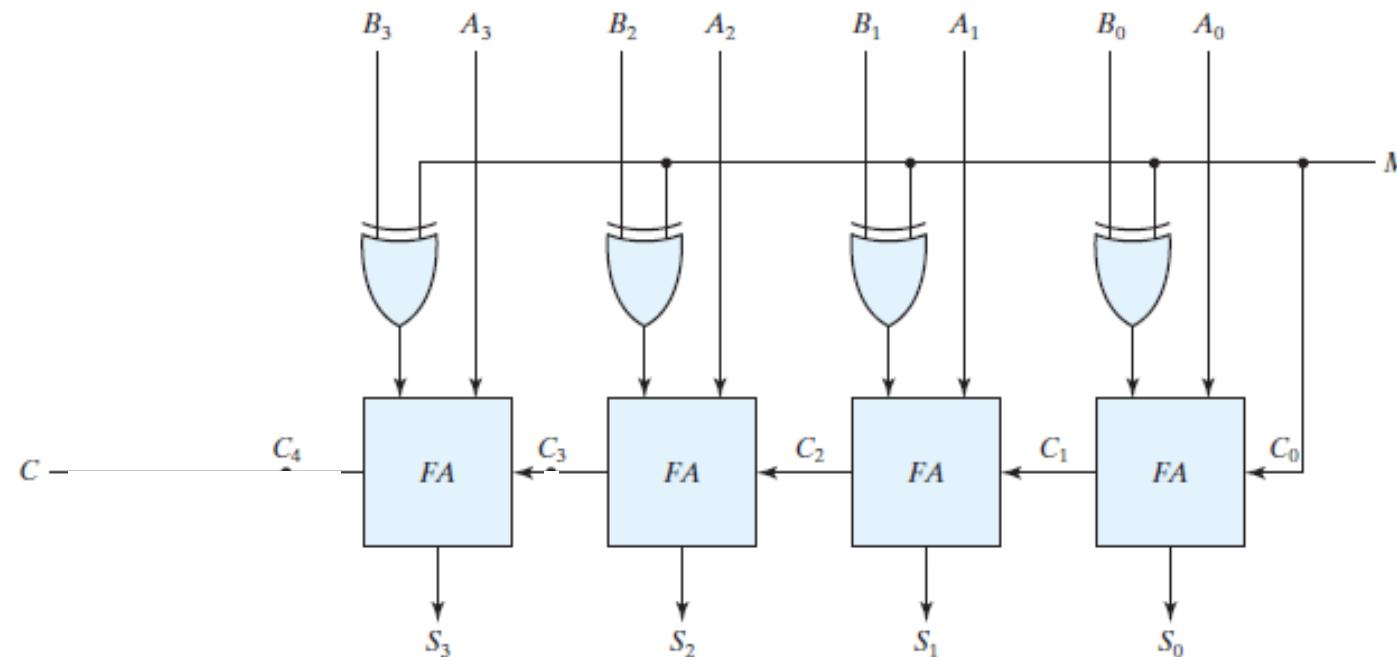


Subtraction with Adders?



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?

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

$$\begin{array}{r} 1010 \\ + 1000 \\ \hline 10010 \end{array}$$

carry sum

unsigned = carry out bit
"overflow"

Signed

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

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

signed

$5 = 0101 \quad 1010 \rightarrow 1011$
 $6 = 0110$

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

carry = 0 \Rightarrow No overflow?

Overflow for signed numbers?

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

Overflow for signed numbers?

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

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

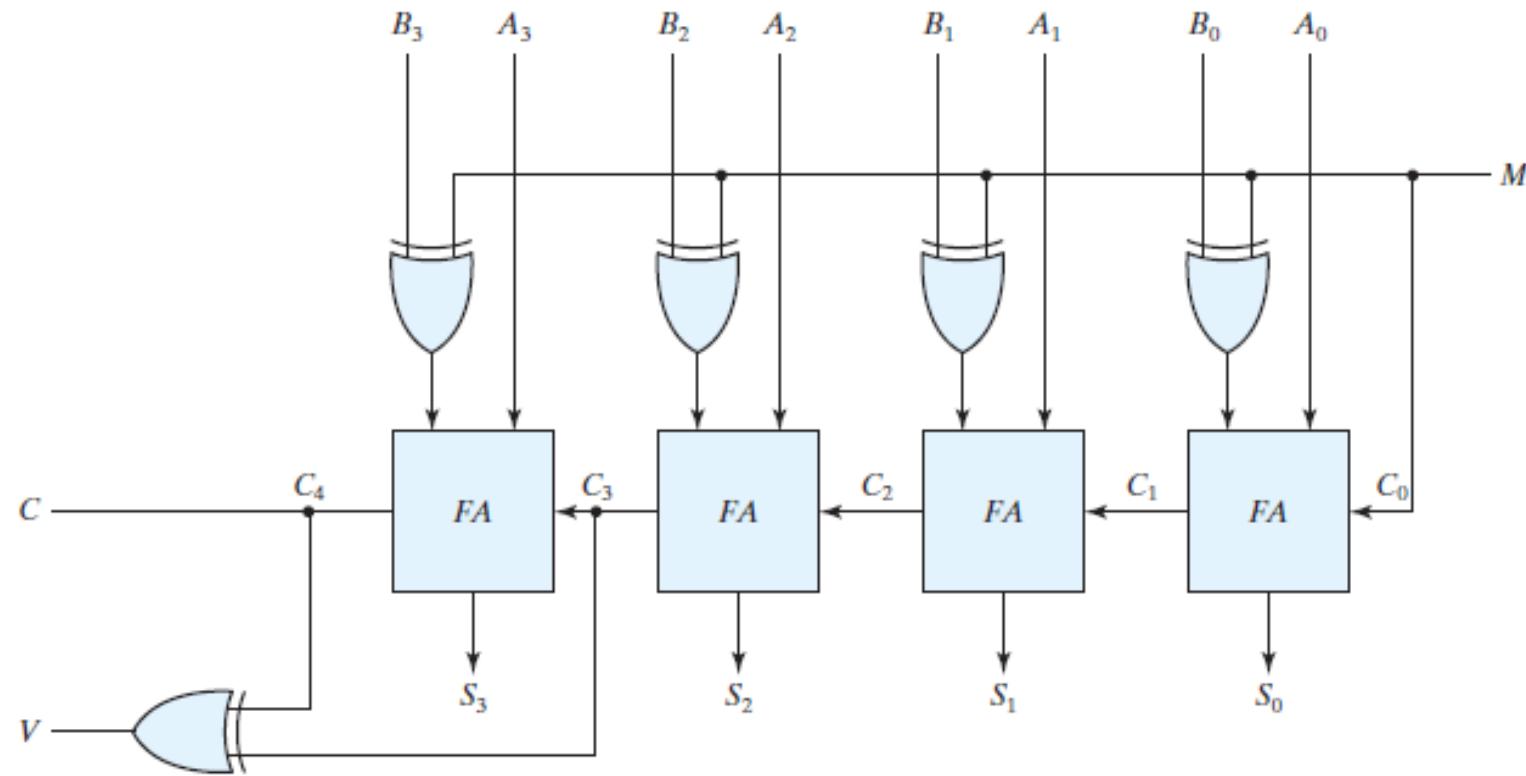
Overflow detection

- 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



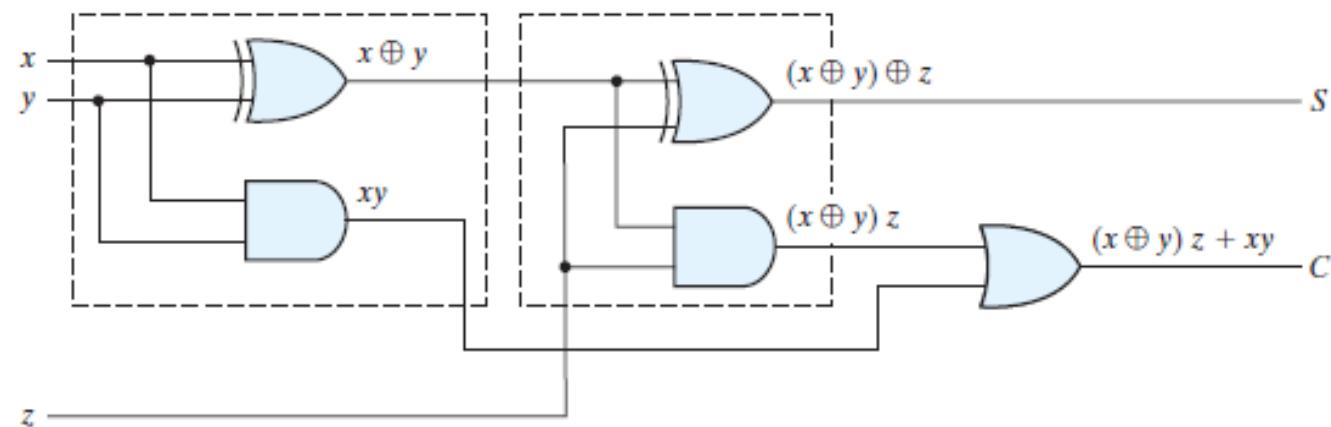
Gate Delay

- 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

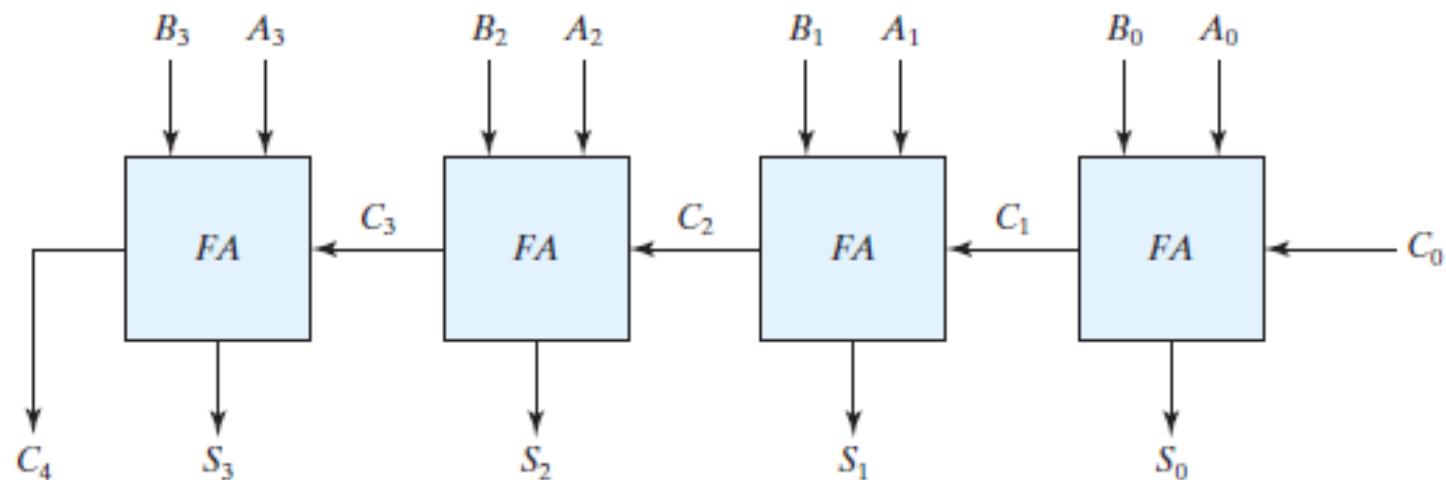
- Assume 5ps/gate



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

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