

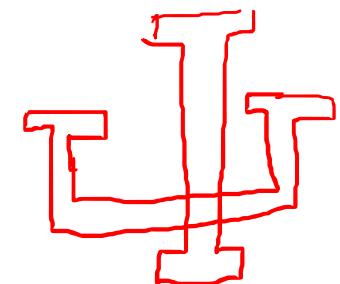
Testing?

ENGR 210 / CSCI B441

Verilog Basics

Andrew Lukefahr

An attempt



Course Website

engr210.github.io

Write that down!

Announcements

Autograder!

Slack

- P1 is out! Due Friday
- Can everyone get onto slack?
- Logic Review on Friday? 2pm?

2-2:30pm?
works.

Offered hours: → right after class
Office hours: → right after class
Oft/Lab zoom → { Merve: 5-7 pm Tues
Malinchai: 5-7 TH } { posted on Syllabus }

Last Time: Truth Table to Boolean Equations

A	B	C	Z
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0

1. Find each '1' output
'AND' the inputs
2. ~~Write the equation for that output's row~~
3. 'OR' the above equations together

$$Z = \overline{A} \cdot \overline{B} \cdot \overline{C} +$$
$$\overline{A} \cdot B \cdot \overline{C} +$$
$$A \cdot \overline{B} \cdot C$$

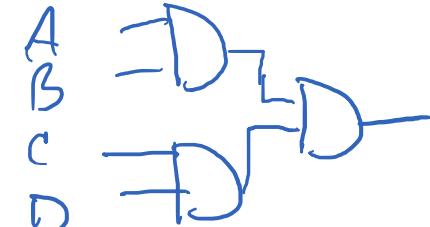
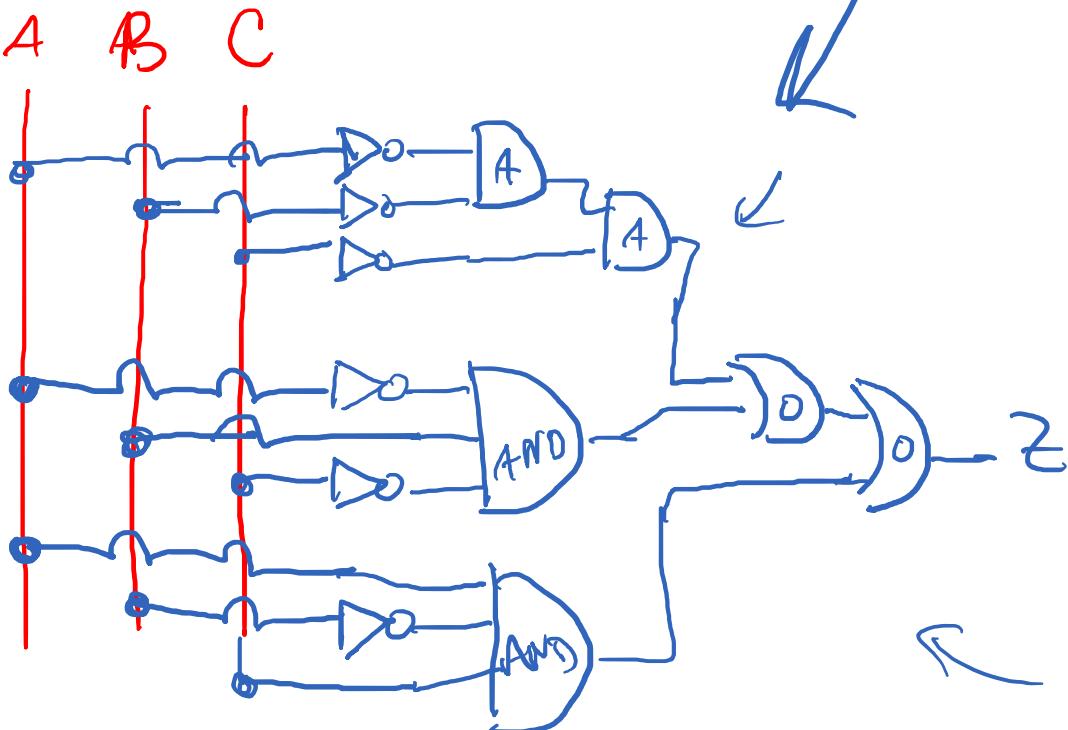
↓
• - AND
+ - OR

$$\overline{A} \Rightarrow A \cdot B \Rightarrow A \oplus B$$

Truth Table to Boolean Equations

A	B	C	Z
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0

$$Z = \overline{A} \cdot \overline{B} \cdot \overline{C} + \overline{A} \cdot B \cdot \overline{C} + A \cdot \overline{B} \cdot C$$



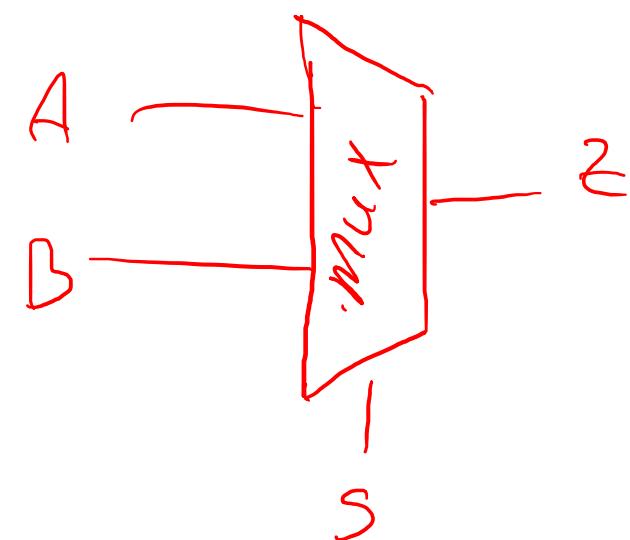
$$Z = (A \oplus B) \cdot C \cdot D$$

• → 0
+ → 1

More Truth Tables!

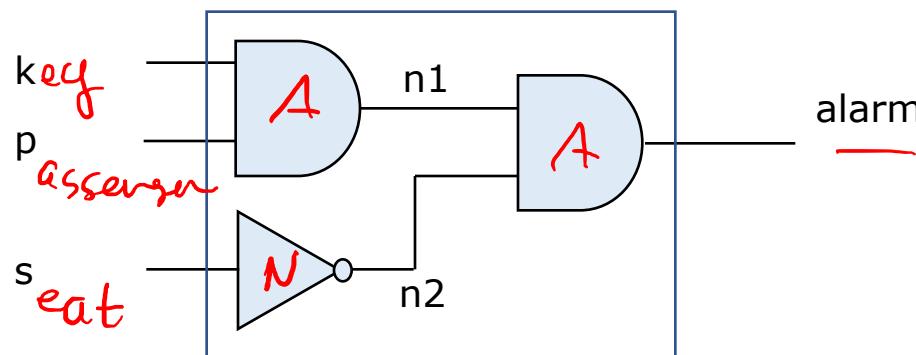
A	B	S	Z
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

$$Z = \overline{A} \cdot \overline{B} \cdot \overline{C} + \\ A \cdot \overline{B} \cdot \overline{C} + \\ \overline{A} \cdot B \cdot \overline{C} + \\ \overline{A} \cdot B \cdot C$$

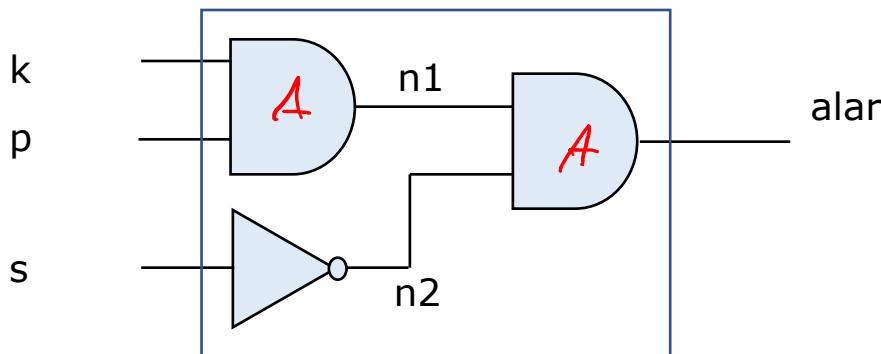


Example: Seat Belt Alarm

- Goal: Set an output alarm to logic 1 if:
 - The key is in the car's ignition slot ($k==1$), and
 - A passenger is seated ($p==1$), and
 - The seat belt is not buckled ($s==0$)



Boolean Logic in Verilog



C-Asside

$a \otimes b \rightarrow \text{bitwise}$

$a \otimes\otimes b \rightarrow \text{logical}$

$$a = 5 = 0101$$

$$b = 2 = 0010$$

$$a \otimes b =$$

$$\begin{array}{r} 0101 \\ \otimes 0010 \\ \hline 0000 = F \end{array}$$

- We can use Boolean logic models in Verilog:

→ assign alarm = $(k \& p) \& \sim s;$

- Evaluated when **any** of the right-hand-side operands changes

- Assigns a new value to the left-hand-side operand

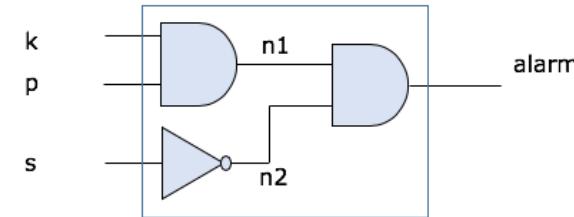
$$\begin{array}{ccc} S \stackrel{T}{=} 0 & \otimes\otimes & Z \stackrel{T}{=} 0 \\ a \otimes\otimes b = T & & \end{array}$$

Verilog Example

```
'timescale 1 ns/1 ns

//-----
// Example: Belt alarm
// Model: Boolean level
//-----

module BeltAlarm(
    input k, p, s,          // definition of input ports
    output alarm            // definition of output ports
);
    assign alarm = k & p & ~s; //Boolean equation
endmodule
```



Aside: Synthesis

- In Synthesis, Vivado auto-magically:

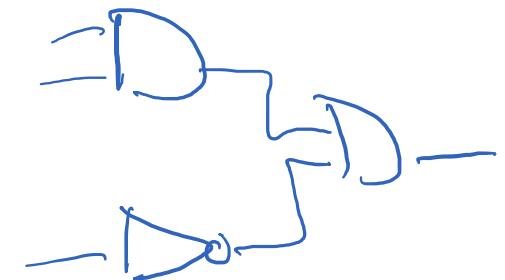
- Translates Boolean models into gate-level models

- Simplifies and minimizes the gate-level models

- All you have to do is ... wait ...

Fridays
afternoon

alarm = k & p & ~s;



alarm = k & p & ~s;

2 seats?



- What if I have a car with 2 seats?

→ • *k*: a car's key in the ignition slot (logic 1)

{ • *st_pas*: the passenger is seated (logic 1)
• *sb_pas*: the passenger's seat belt is buckled (logic 1)

{ • *st_drv*: the driver is seated (logic 1)
• *sb_drv*: the driver's seat belt is buckled (logic 1)

Goal: Set an output *alarm* to logic 1 if:

The key is in the car's ignition slot ($k == 1$), and

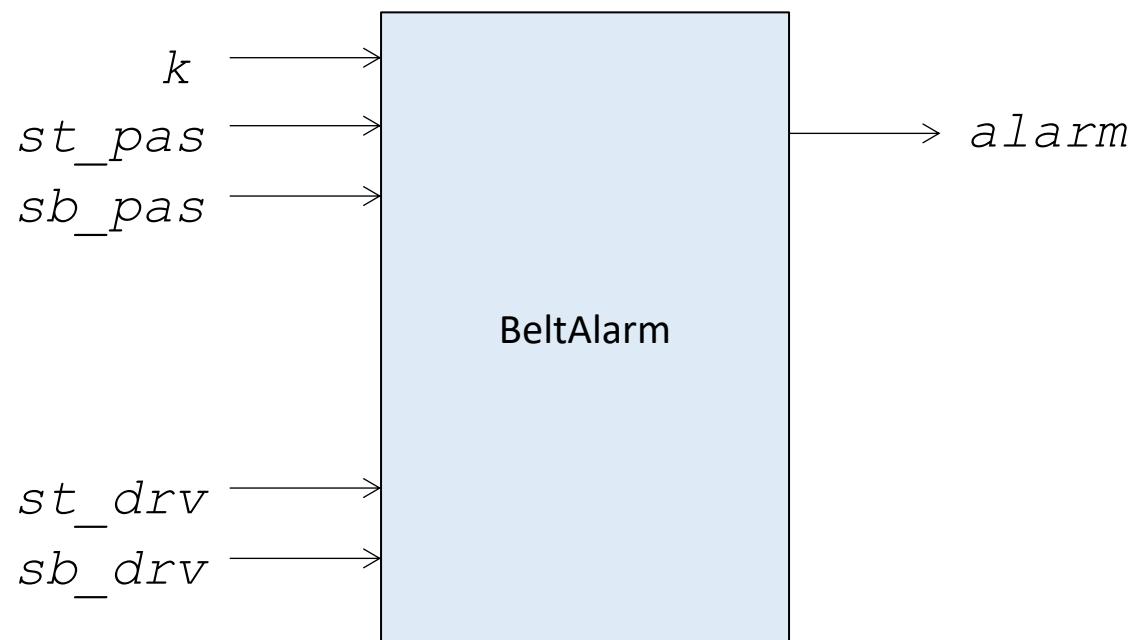
(($st_drv == 1$ and $sb_drv == 0$) or
($st_pas == 1$ and $sb_pas == 0$))

2 seats: Solution 1



Goal: Set an output alarm to logic 1 if:

The key is in the car's ignition slot ($k == 1$), and
($\neg st_drv == 1$ and $\neg sb_drv == 0$) or
($st_pas == 1$ and $\neg sb_pas == 0$)



$$(k \& st_drv \& \neg sb_drv) | \\ (k \& st_pas \& \neg sb_pas)$$

module Two Seat Alarm (

input k,
input st_pas, sb_pas,
input st_drv, sb_drv,
); output alarm

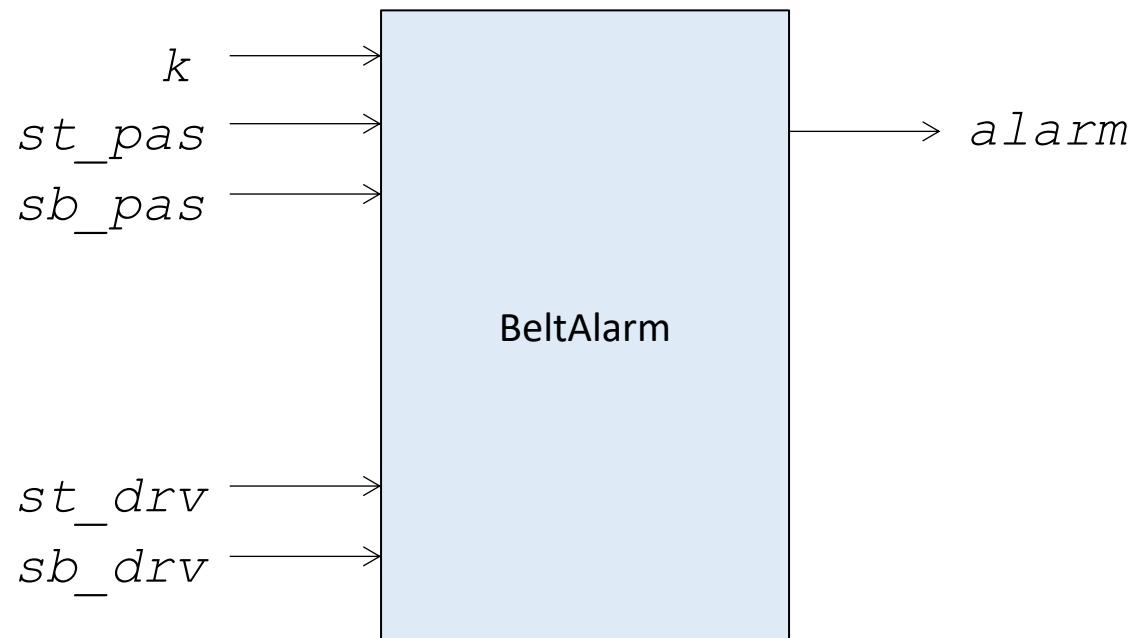
[assign alarm = K & ((st_drv & ~sb_drv) | (st_pas & ~sb_pas));

end module

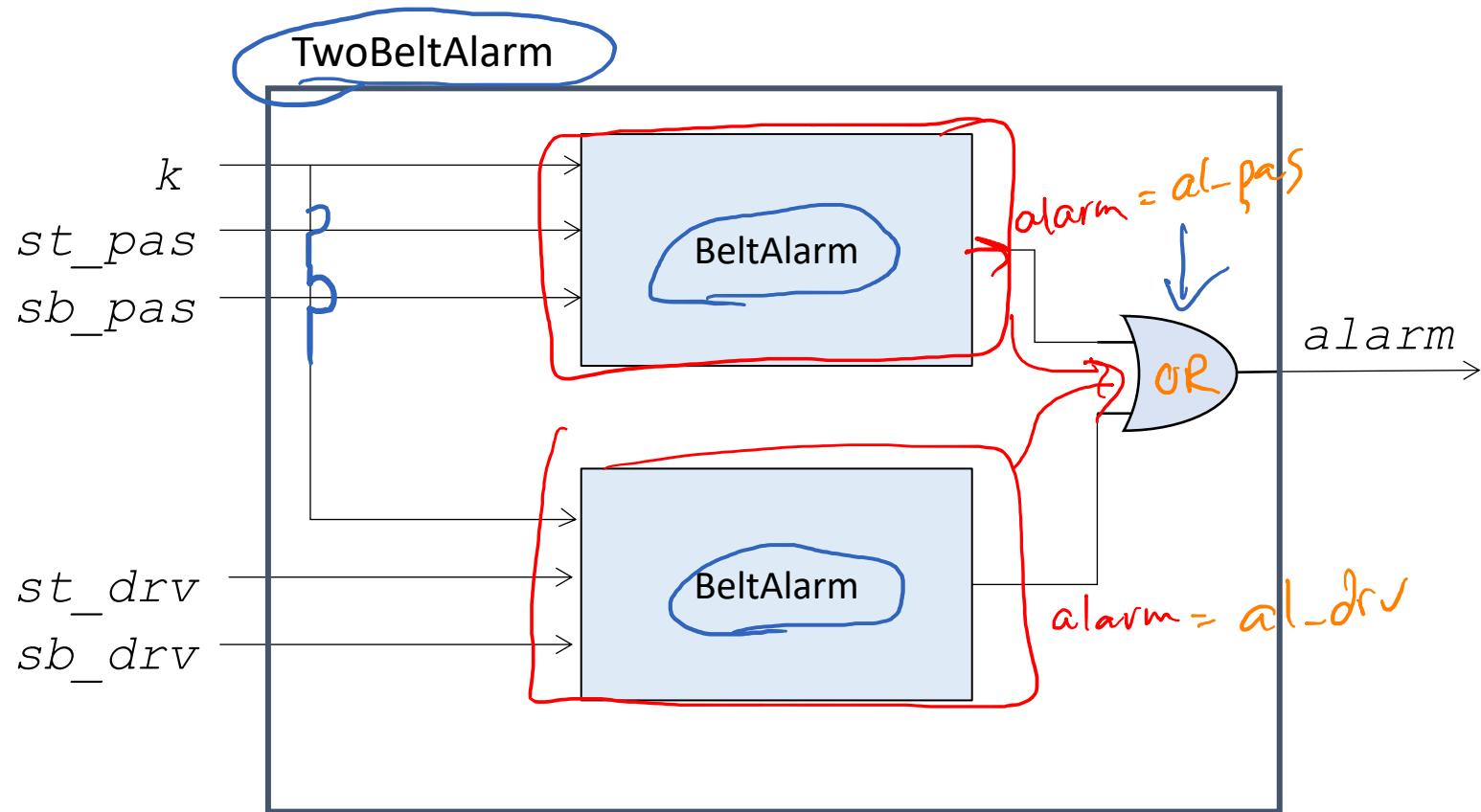
2 seats: Solution 1

Goal: Set an output alarm to logic 1 if:

The key is in the car's ignition slot ($k==1$), and
 $((st_drv==1 \text{ and } sb_drv == 0) \text{ or }$
 $(st_pas==1 \text{ and } sb_pas == 0))$



Solution 2: Use Submodules



C = int X

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

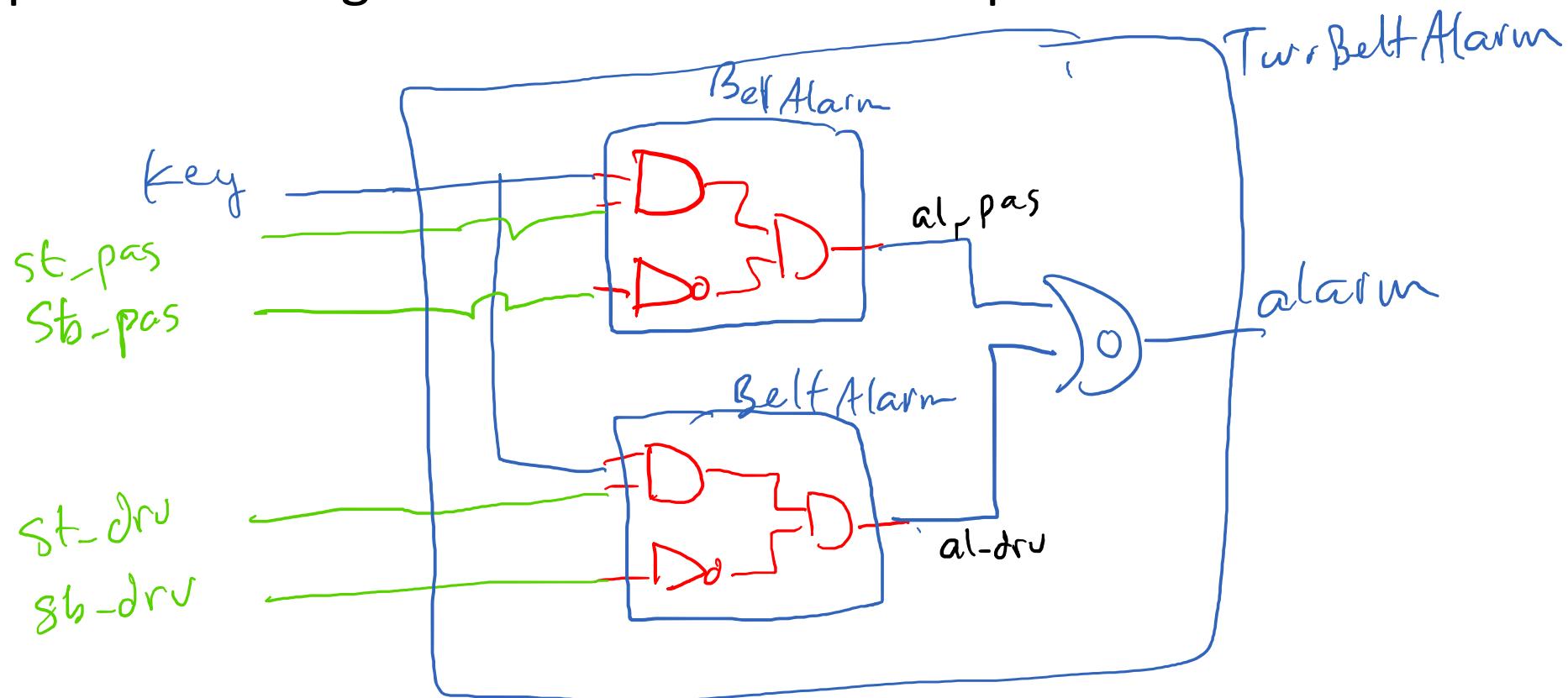
→ **BeltAlarm** ba_pas(.k(k), .p(st_pas),
.s(sb_pas), .alarm(al_pas)); // with named arguments

```
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
```

Hierarchical Models

- Modules are basic building block in Verilog
- Group modules together to form more complex structure

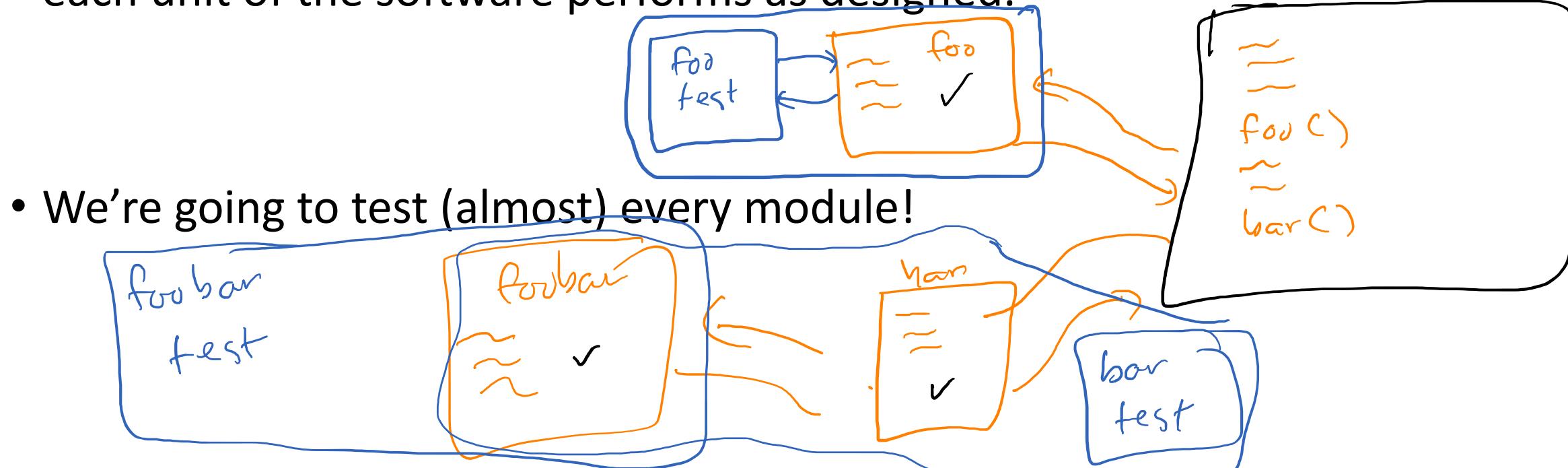


Testing

Unit Testing

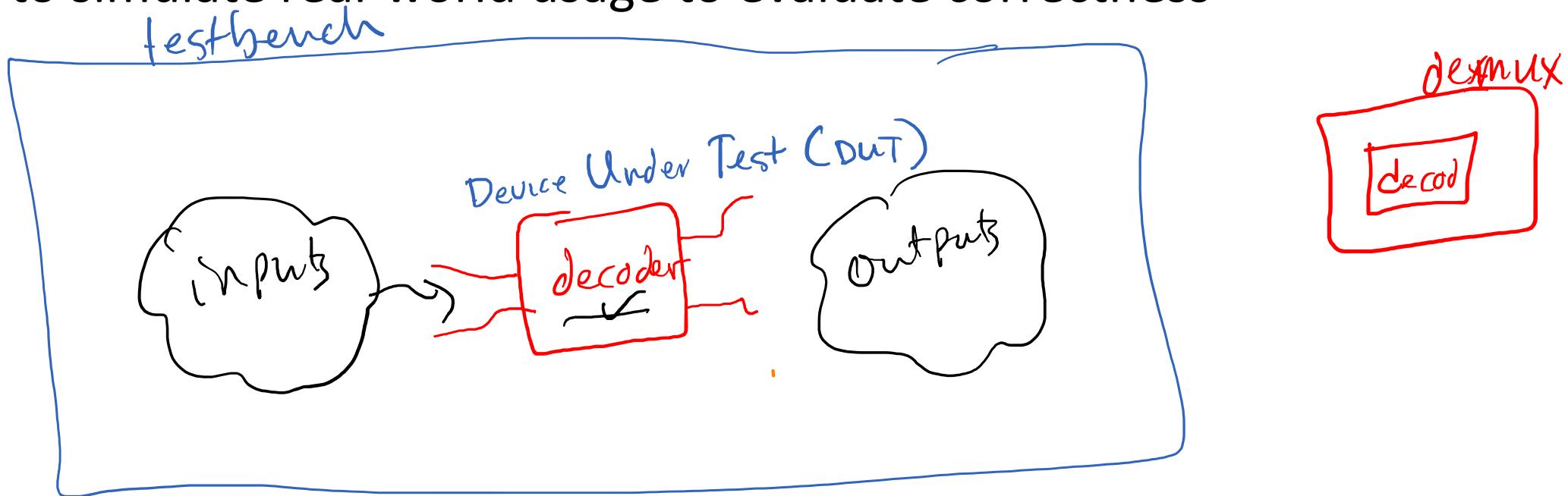


- **UNIT TESTING** is a level of software testing where **individual components** of a software **are tested**. The purpose is to validate that each unit of the software performs as designed.



TestBench

- Another Verilog module to drive and monitor our Verilog module
- Goal is to simulate real-world usage to evaluate correctness



Simulation vs Synthesis

- Synthesis: Real gates on real hardware
 - Only “synthesizable” Verilog allowed

→ decoder (demux) { on FPGA }

- Simulation: Test our design with software

- “Non-synthesizable” Verilog allowed
- \$initial
- \$display

→ decoder_tb
demux_tb { not on FPGA }

“initial” statement

- **Simulation only!**
- An initial block starts at simulation time 0, executes exactly once, and then does nothing.
- Group multiple statements with begin and end.

→ • begin/end are the ‘{’ and ‘}’ of Verilog.

```
initial  
begin
```

```
  a = 1;  
  b = 0;  
end
```

→ a = 1; b = 0;
delay
a = 0; b = 1;

`timescale 1ns / 1ps
Delayed execution → #() → 1ns
 $\#10 \rightarrow 10\text{ns}$

- If a delay $\#<\text{delay}>$ is seen before a statement, the statement is executed $<\text{delay}>$ time units after the current simulation time.

```

initial
begin
    #10 a = 1; // executes at 10 time units
    #25 b = 0; // executes at 35 time units
end
    
```

$a=1;$
 $a=0;$

assign $x=a$

- We can use this to test different inputs of our circuits

$a=0;$
 $\#()$ $x=0$
 $a=1;$ $x=0$
 $\#1$ $x=1$

\$monitor

- \$monitor prints a new line every time it's output changes
- C-like format

```
• $monitor ($time,  
          "K= %b, P= %b, S= %b, A= %b\n",  
          K, P, S, A);
```

Example Output:

```
0  K= 0, P= 0, S= 0, A= 0  
5  K= 1, P= 0, S= 0, A= 0  
10 K= 1, P= 1, S= 0, A= 1
```

A simple testbench

```
✓`timescale 1ns/1ps
module 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 // de.
    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
```

printf ("%d", x)

%b => binary

%h => hex

%d => decimal

Tasks in Verilog



- A task in a Verilog simulation behaves similarly to a C function call.

```
task taskName(  
    ↗ input localVariable1,  
    ↗ input localVariable2,  
    );  
  
    #1 //1 ns delay  
    ↗ globalVariable1 = localVariable1;  
    ↗ #1 // 1ns delay  
    ↗ assert( globalVariable2 == localVariable2)  
        ↗ else $fatal(1, "failed!");  
  
endtask
```

SeatBelt 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
```

SeatBelt 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");

end

Tasks in Testing

- tasks are very useful for quickly testing Verilog code
- Call a task to quickly change + check things
- A task can call another task
- There is a function in Verilog.
- We don't use it.

@TODO: Testbench for 2 SeatBelt!

2-BeltAlarm testbench

```
`timescale 1ns/1ps
module tb();
    reg 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 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
```

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  
);  
    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");
end
```

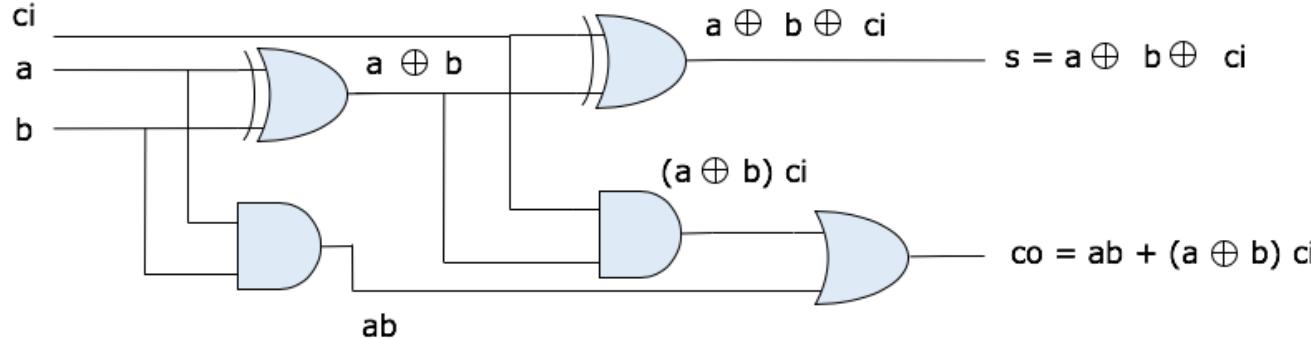
```
module TwoBeltAlarm(
    input k, st_pas, sb_pas,
    input st_drv, sb_drv
);
    wire al_pas, al_drv;

    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
```

Addition / Subtraction

Your Turn...



Implement the circuit above in Verilog.

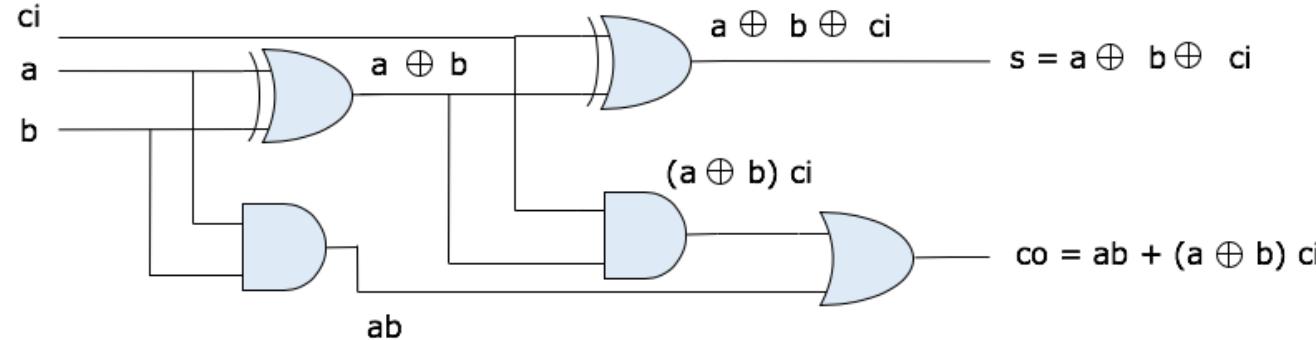
- Hint: XOR in Verilog is '^'

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

    // Your Code Here!
```

```
endmodule
```

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
```

2-Bit Binary Addition

2-Bit Adder

```
module TwoBitAddr (
    input a0, a1, b0, b1, ci,
    output s0, s1, co
);
```

//code me!

endmodule

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

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

endmodule
```

TwoBit Adder

. . .

Stopped here!

```
module TwoBitAddr (
    input a0, a1, b0, b1, ci,
    output s0, s1, co
) ;
```



```
wire r; //caRry ?
```



```
FullAddr fa0 (a0, b0, ci, s0, r);
FullAddr fa1 (a1, b1, r, s1, co);
```



```
endmodule
```

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



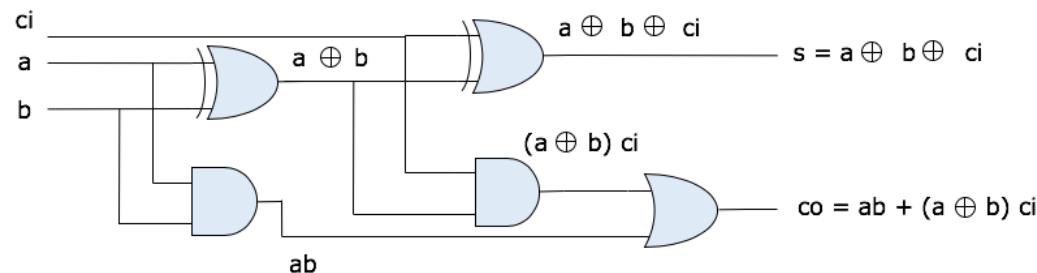
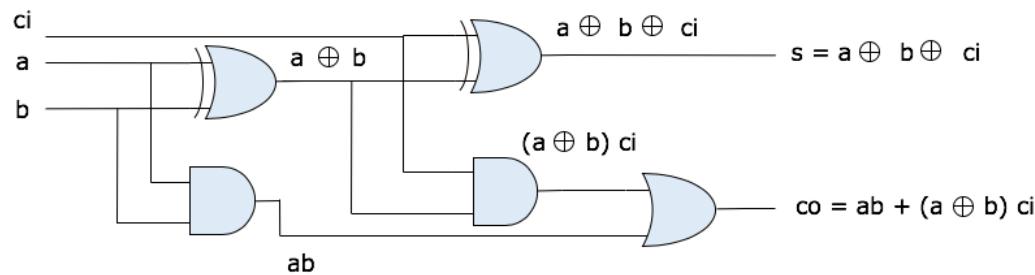
```
s = a ^ b ^ ci;
co = a & b | ( a ^ b) & ci;
```



```
endmodule
```

TwoBit Adder

```
module TwoBitAddr(
    input a0, a1, b0, b1, ci,
    output s0, s1, co
);
    wire r; //caRry ?
    FullAddr fa0 (a0, b0, ci, s0, r);
    FullAddr fa1 (a1, b1, r, s1, co);
endmodule
```



Next Time

- Continue with Addition/Subtraction