Test

Please Post

ENGR 210 / CSCI B441

Verilog Basics

Andrew Lukefahr

Announcements

- P5 is due Friday ← Demux / Decoder
- P6 is out - ALU

groups of 2

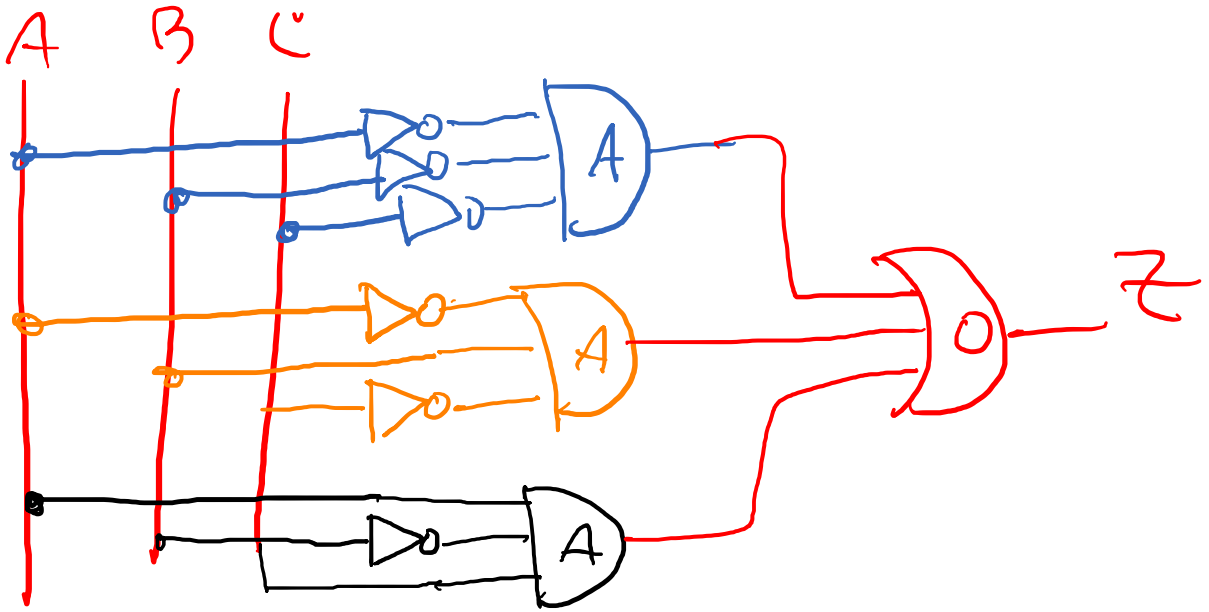
Truth Table to Boolean Equations

Uloy

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

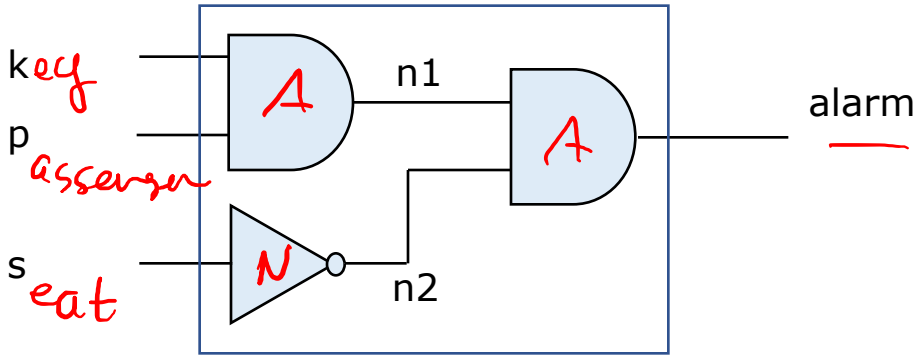
$$Z = \bar{A}\cdot\bar{B}\cdot\bar{C} + \bar{A}\cdot B\cdot\bar{C} + A\cdot\bar{B}\cdot C \Rightarrow$$

```
assign Z =
  (~A & ~B & ~C) |
  (~A & B & ~C) |
  ( A & ~B & C );
```

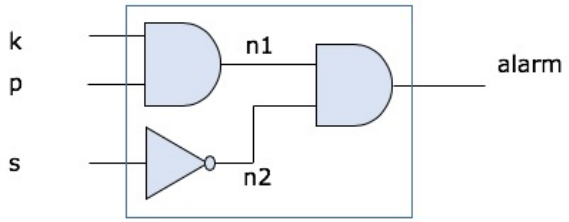


Example: Seat Belt Alarm

- 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 bucked (*s==0*)



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
```

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.

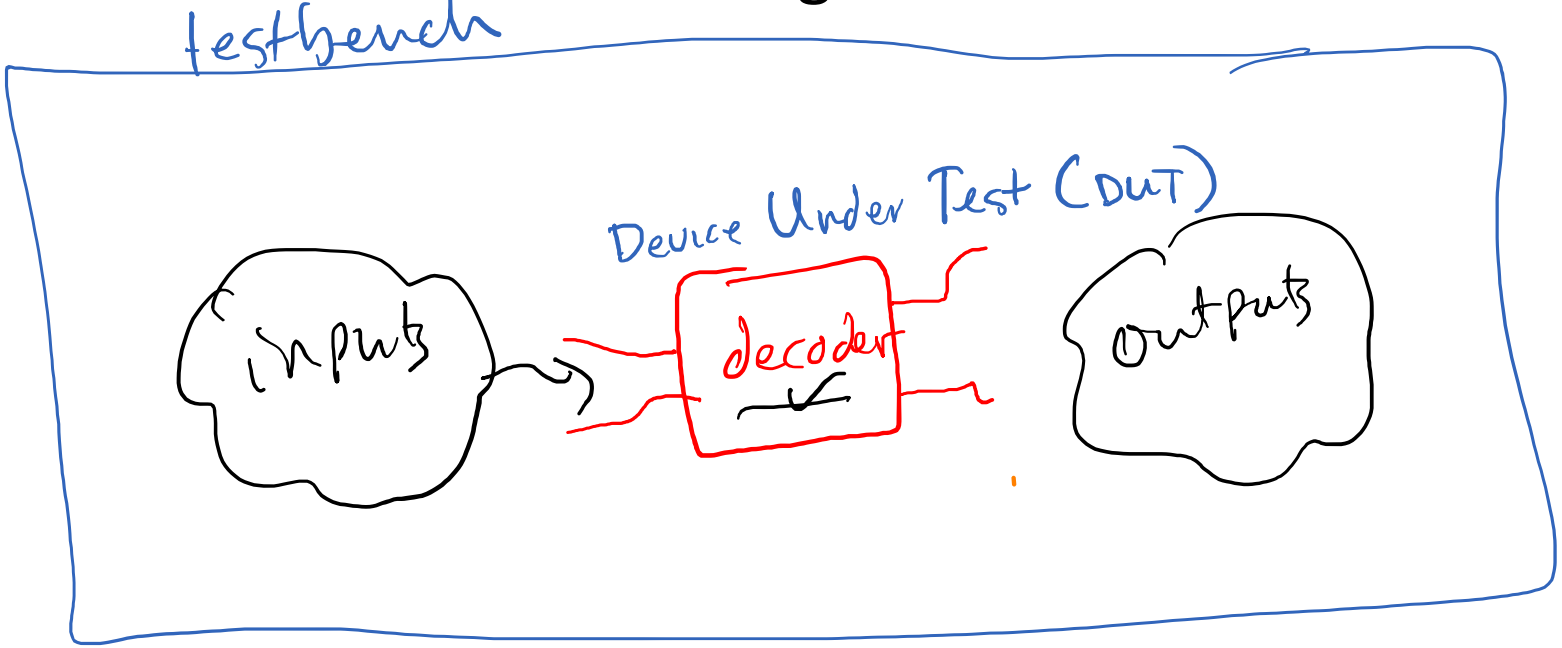
- We're going to test (almost) every module!

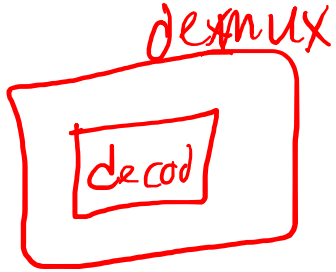
https://softwaretestingfundamentals.com/unit-testing/

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 → decoder / demux } on FPGA
  - Only "synthesizable" Verilog allowed

- Simulation: Test our design with software → decoder_tb, demux_tb } not on FPGA
  - "Non-synthesizable" Verilog allowed
  - $initial
  - $display

"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 → #1 → 1ns

Delayed execution

#10 → 10ns

- If a delay `#<delay>` is seen before a statement, the statement is executed `<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
```

start a=0 x=0

a=1; (x=0)

a=0;

assign x=a

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

a=0;

#1 x=0

a=1; x=0

#1 x=1

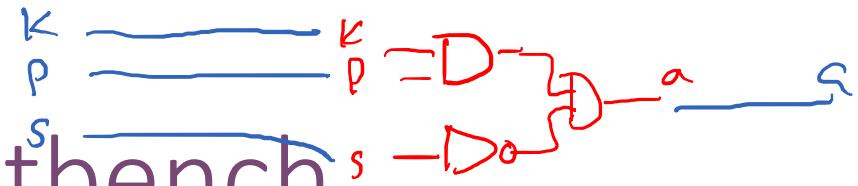
$monitor

- `$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 BeltAlarm_tb();

logic k, p, s;
wire alarm;
BeltAlarm dut0( .k(k), .p(p), .s(s),.alarm(alarm) );
```

.k(0) .p(1) .s(1), .alarm(alarm)

Task c)

```
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
```

'h0 = hexidecimal value 0

x/1 - binary

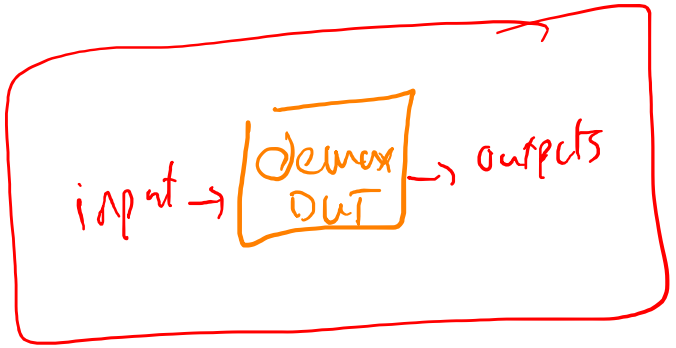
```
module BeltAlarm(
    input k, p, s,
    output alarm
);

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

Tasks in Verilog

functions

- 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
```

~~C lang~~
~~foo (int x)~~
return

C lang
int foo (int x)
return 5

int main()
foo();

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
```

```
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
```

`Tasks` in Testing

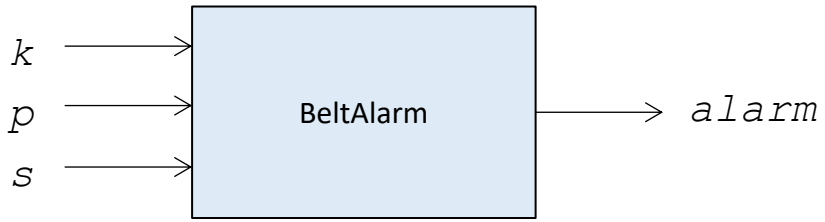
- `tasks` are very useful for quickly testing Verilog code

- 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.

SubModules

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*), <u>and</u>

((*st_drv==1 and sb_drv == 0*) or

(*st_pas==1 and sb_pas == 0*))

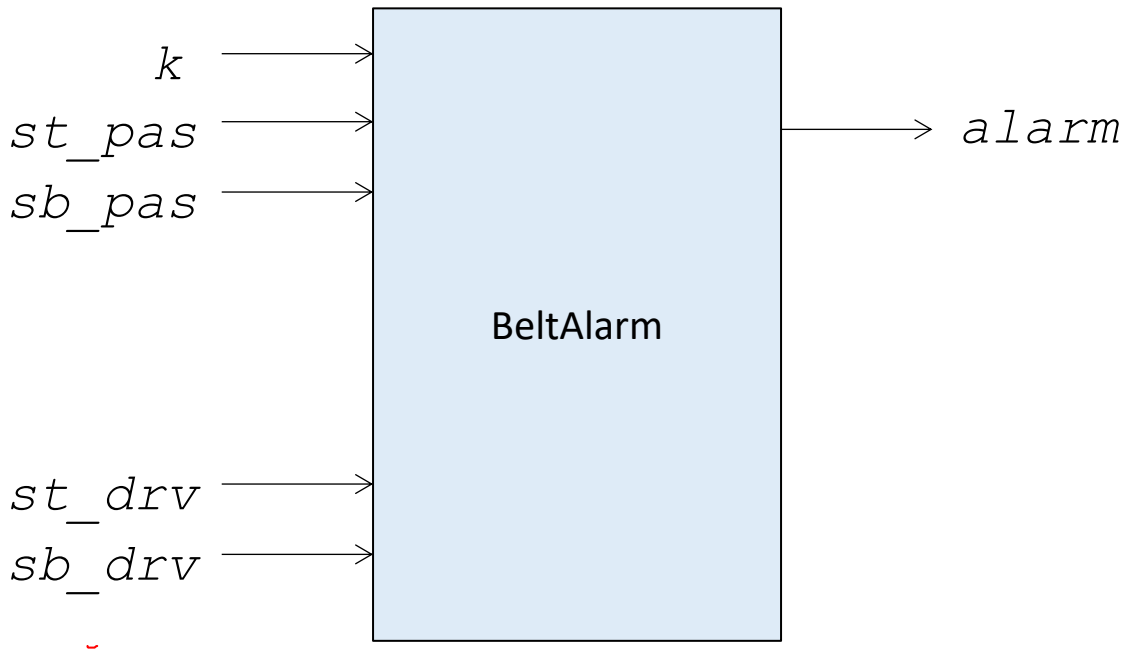
2 seats: Solution 1

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

The key is in the car's ignition slot (*k==1*), <u>and</u>

((*st_drv==1 and sb_drv == 0*) or

(*st_pas==1 and sb_pas == 0*))

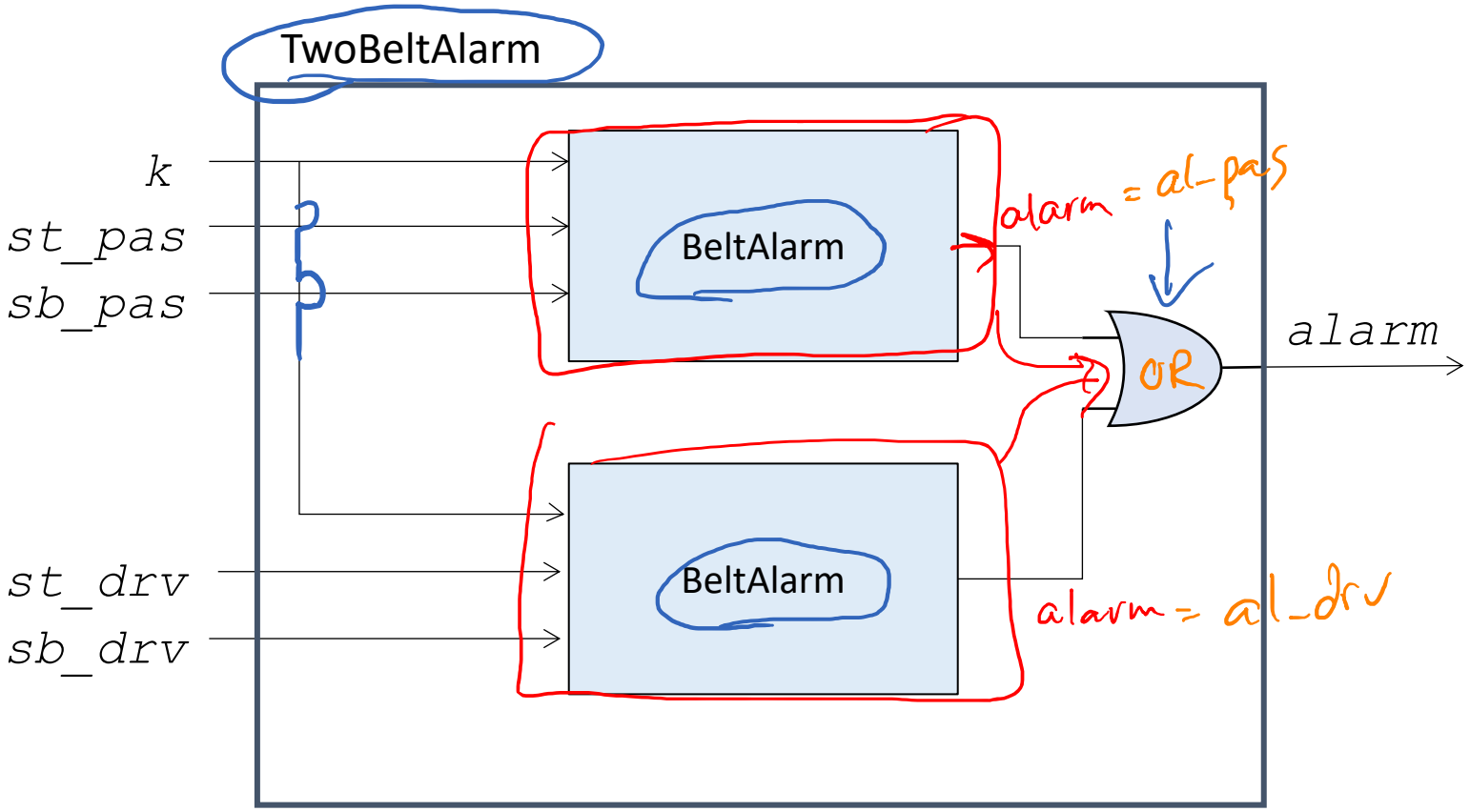



```
module TwoSeatAlarm (
    input k,
    input st_pas, sb_pas,
    input st_drv, sb_drv,
    output alarm
);

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

endmodule
```

(k & st_drv & ~sb_drv) |

(k & st_pas & ~sb_pas)

Solution 2: Use Submodules



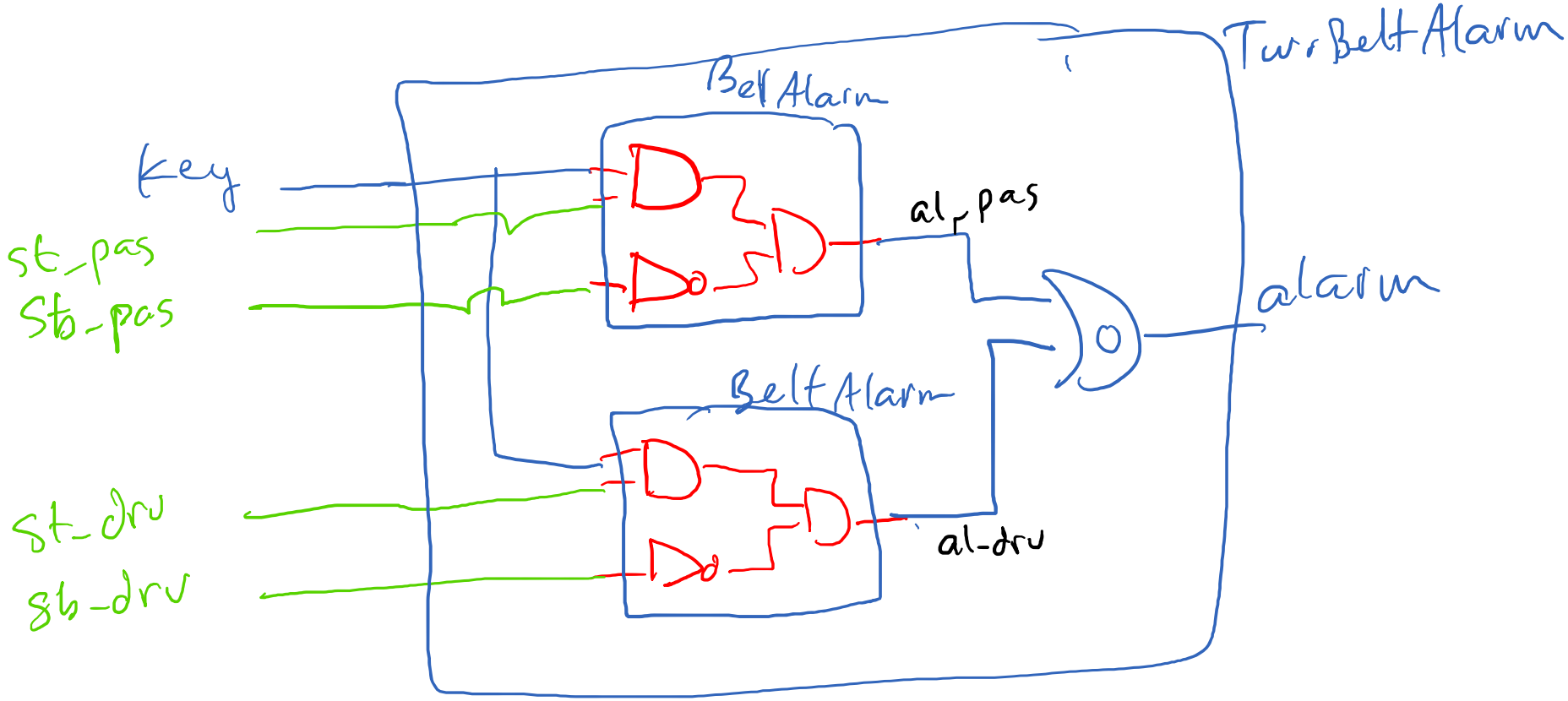
Submodule Example

```
`timescale 1 ns/1 ns

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

    assign alarm = k & p & ~s;

endmodule
```

```
`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
```

Hierarchical Models

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



@TODO: Testbench for 2 SeatBelt!

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

logic
~~reg~~ k, stPas, sbPas, stDrv, sbDrv;
wire alarm;
TwoBeltAlarm dut0( .k(k), .st_pas(stPas), .sb_pas(sbPas),
          .st_drv(stDrv), sb_drv(sbDrv), .alarm(alarm) );

initial
begin
    k = 'h0; stPas='h0; sbPas='h0;
         stDrv='h0; sbDrv='h0;
    $monitor ("k:%b stPas:%b sbPas:%b stDrv:%b sbDrv:%b a:%b", k, stPas, sbPas, stDrv, sbDrv, 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, 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
     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

```
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
```

```
initial
begin
    k = 'h0; stPas='h0; sbPas='h0;
             stDrv='h0; sbDrv='h0;
    $monitor ("k:%b stPas:%b sbPas:%b stDrv:%b sbDrv:%b a:%b", k, stPas,
sbPas, stDrv, sbDrv, alarm);
    #10

    checkAlarm(‘h0,’h0,’h0,’h0,’h0, 'h0);
    //...
    checkAlarm('h1,'h1,'h1,’h1,'h1, 'h0);

$display("@@@Passed");
end
```

For Loops in Testbenches

- You <u>can</u> 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 ***<u>no for-loops in your synthesizable code (yet)!</u>***

- https://www.nandland.com/vhdl/examples/example-for-loop.html

```
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]);
    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
```

```
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
```

i=0 000
i=1 001
=2 010
=3 011

S K

18 = 10010

```
for (i=0; i<4, i++)
        checkAlarm(i[2], i[1], i[0], —)
}
```

$i=0 \Rightarrow$ bin 000 i

$i[2]$ $i[1]$ $i[0]$

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

~a

!a

- 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
```

assign c[0] = 'h0;

c[1] = 'h1;

Constants in Verilog

C: y = 0x6;

h = hex
d = dec
b = binary

```
logic x;
x = '0;  = '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

8'hff ff

8 bits 16 bit

8'hf = 0000 1111

- 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

8 specifies total bits, regardless of 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
```

8'b 1111 1111

replicate

Next Time:

Addition / Subtraction