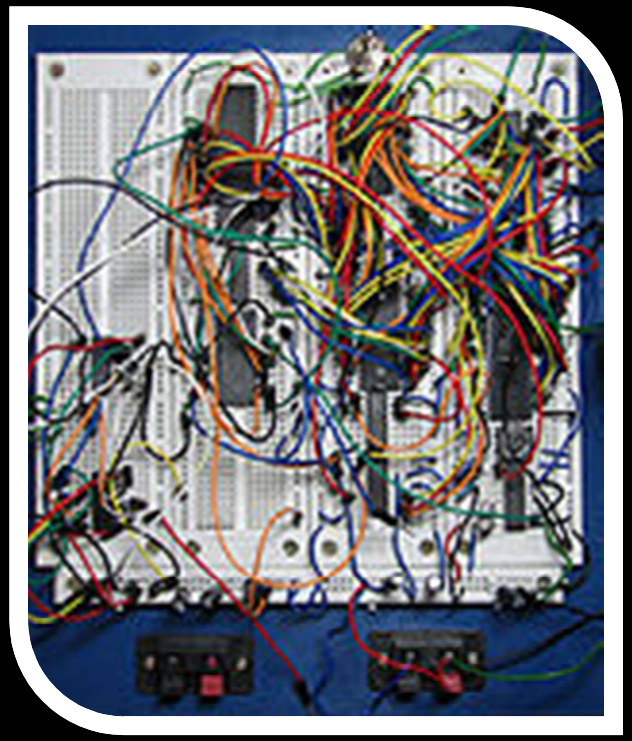
Verilog Tutorial
Combinational Logic

Shutha Pulendran

October 3, 2016

Understanding Verilog

- Verilog is a hardware description language (HDL), **NOT a programming language!!**
- It's used to describe how a circuit must look like

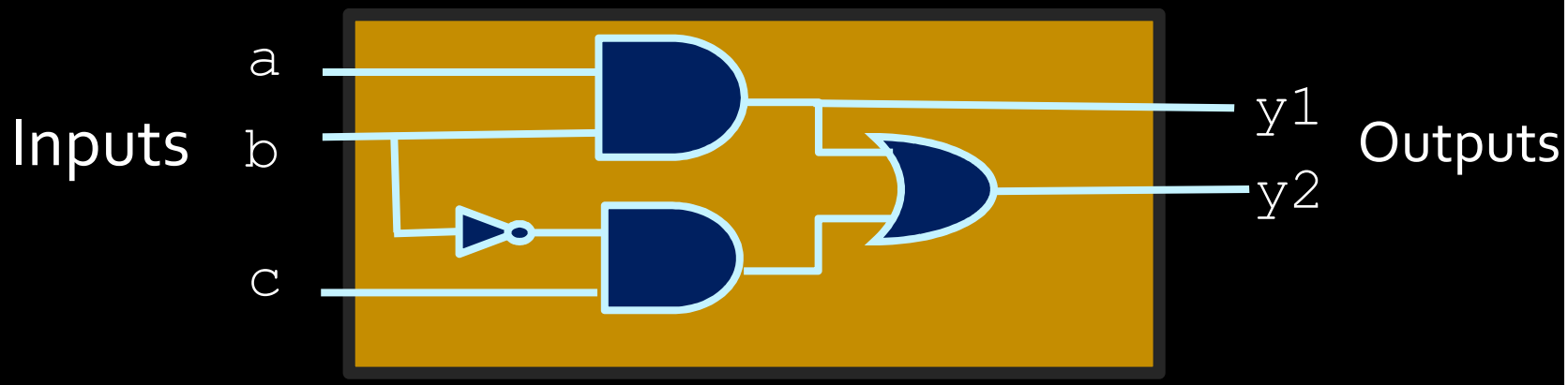


Useful for describing complex circuits

Modules in Verilog

- Each module represents a logic circuit
  - Think of it as a black box with inputs and outputs
  - The relationships between these inputs and outputs are defined inside the module
- Separate modules can be connected to make large, complex circuits

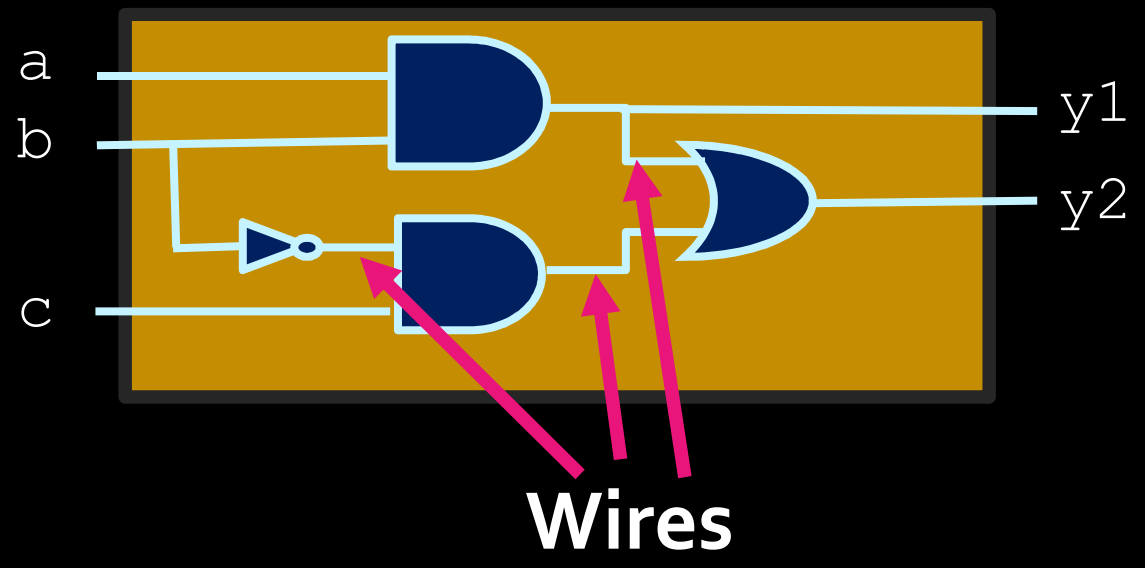
A sample module



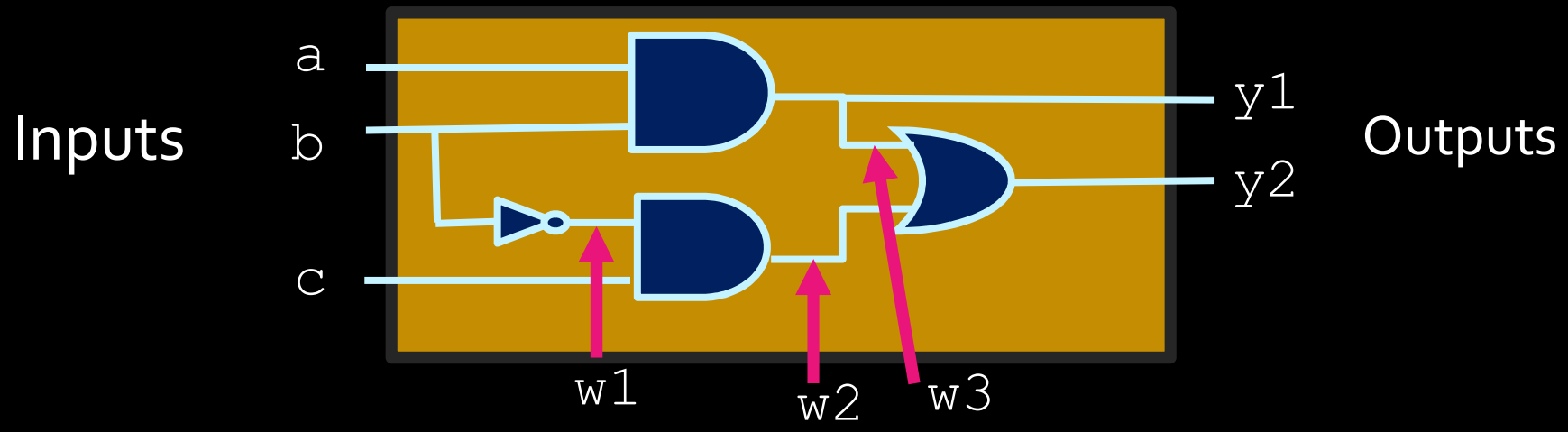

```
module mycircuit (a,b,c,y1,y2); // specify inputs and outputs
       input  a,b,c;   // identify the inputs
       output y1,y2;   // identify the outputs

       // implement the logic functionality below
       assign y1  = a & b;
       assign y2  = (a & b) | (~b & c);
endmodule
```

Wires in Verilog

- They represent physical wires (connections) in a circuit , i.e., connecting two points in a circuit (similar to wires used in Lab 1)
- Commonly used to name internal connections other than inputs and outputs



The same code using wires



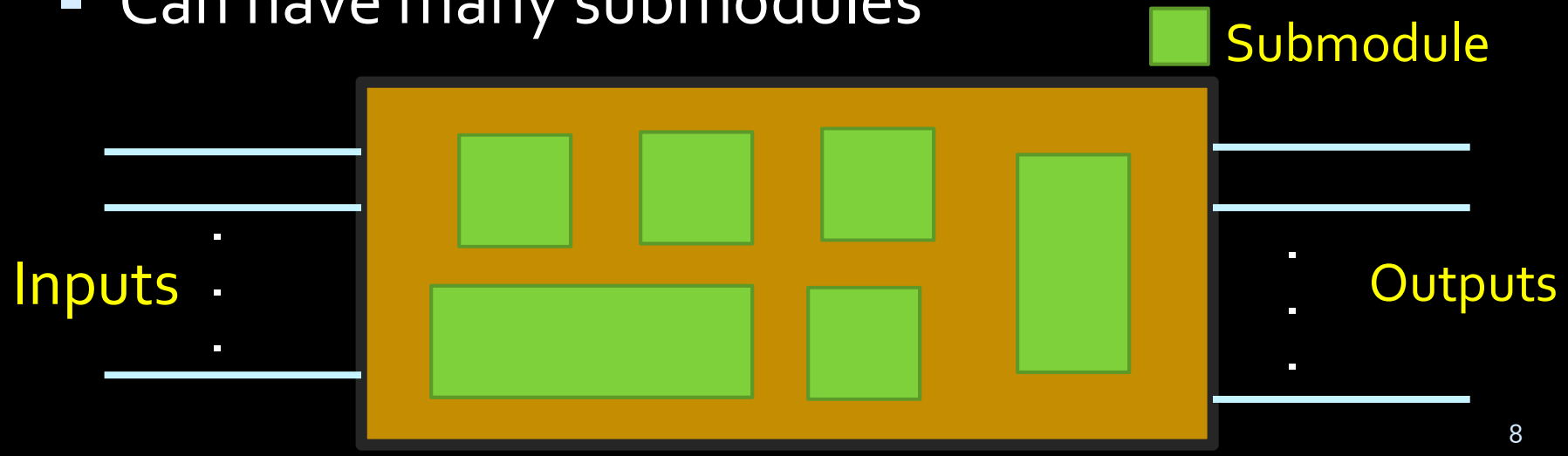

```
module mycircuit (a,b,c,y1,y2); // specify inputs and outputs
       input  a, b, c;   // identify the inputs
       output y1, y2;   // identify the outputs
       wire w1, w2, w3; // identify internal connections

       assign  w1 = ~b;
       assign  w2 =  w1 & c
       assign  w3 =  a & b;

       assign y1 = w3;
       assign y2 = w2 | w3;
endmodule
```

Points to Remember

- **Ordering of Verilog statements doesn't matter** because they represent circuit connections
- As long as the statements describe a complete circuit, the code is good
- Verilog statements don't get executed in a computer similar to other programming languages (e.g., C/C++)
- Always think in terms of building actual hardware connections with wires (e.g., Lab 1)

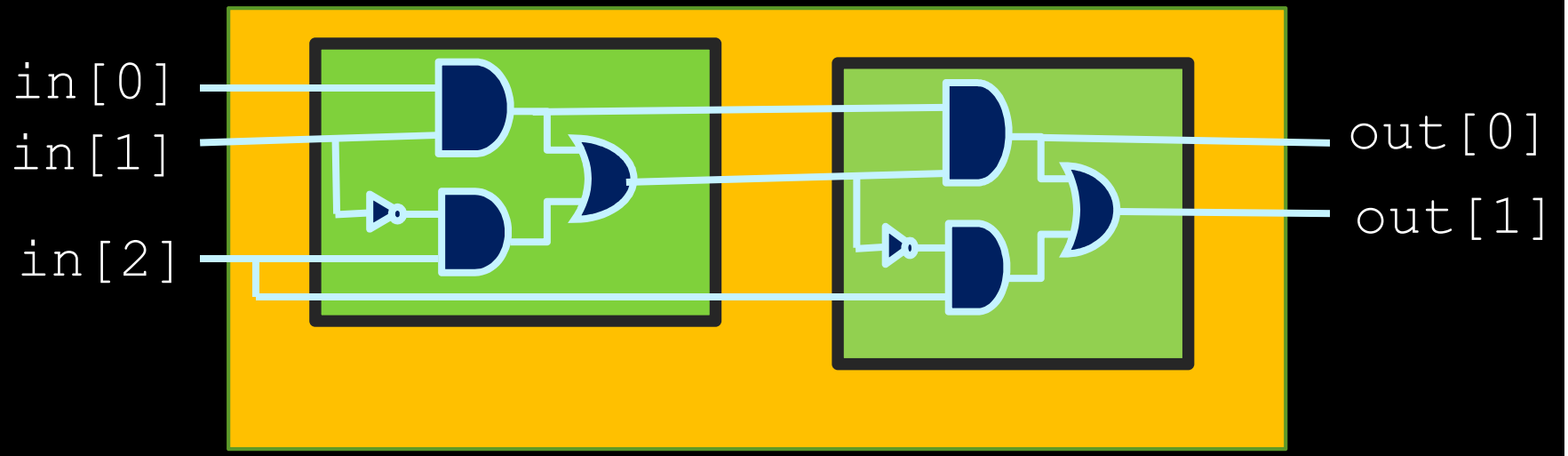
Top-Level Module

- Top-Level module is the largest black box that contains all of your circuits
- Must be named the same as your project in Quartus
- Only **one top-level module** can exist in a project and must be defined
- Can have many submodules



Module Instantiation

- Useful when the same module (same logic circuit) can be used multiple times as submodule
- Example

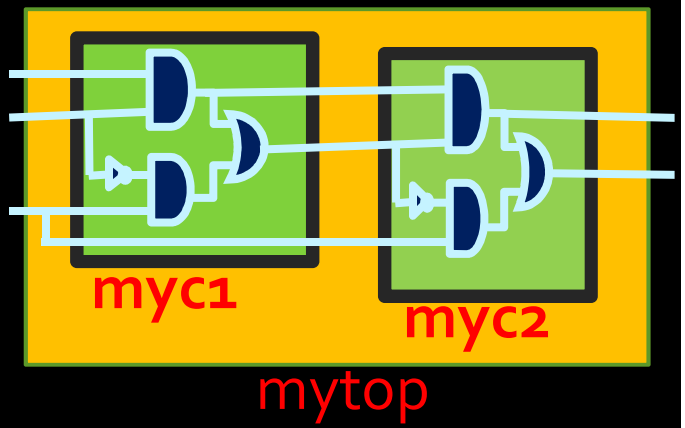


- 3-bit input and 2-bit output

Code For Module Instantiation

- Our previous circuit ("mycircuit") is repeated two times

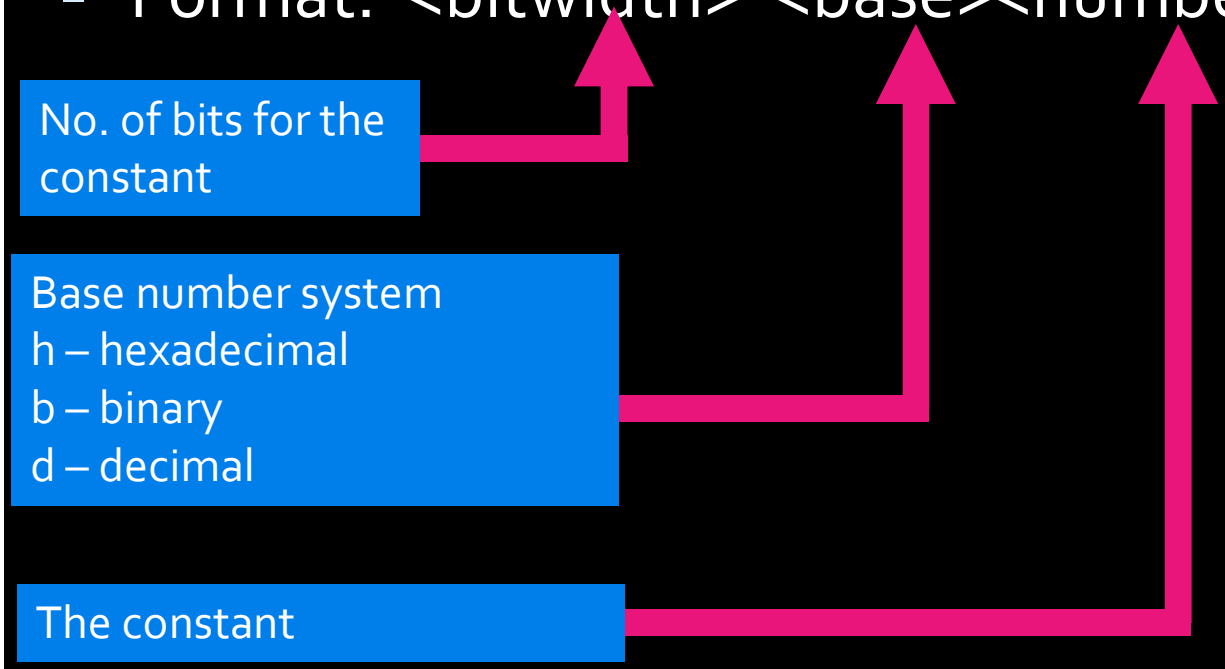
```
module mytop(in, out); // top-level module
        input [2:0] in;
        output [1:0] out;
        wire connector1, connector2;

        mycircuit myc1 ( //submodule 1
                .a(in[0]),
                .b(in[1]),
                .c(in[2]),
                .y1(connector1),
                .y2(connector2),
        );

        mycircuit myc2 ( //submodule 2
                .a(connector1),
                .b(connector2),
                .c(in[2]);
                .y1(out[0]),
                .y2(out[1]),
        );

endmodule
```




```
module mycircuit (a,b,c,y1,y2);
  input  a,b,c;
  output y1,y2;

  assign y1  = a & b;
  assign y2  = (a & b) | (~b & c);
endmodule
```

Constants in Verilog

- Format: <bitwidth>’<base><number>




```
Example:
// All these assignments are equal
assign a = 5’d17;
assign a = 5’b10001;
assign a = 5’h11;
```

Bus Structure Definition & Implication

- The method of defining bus structure is important

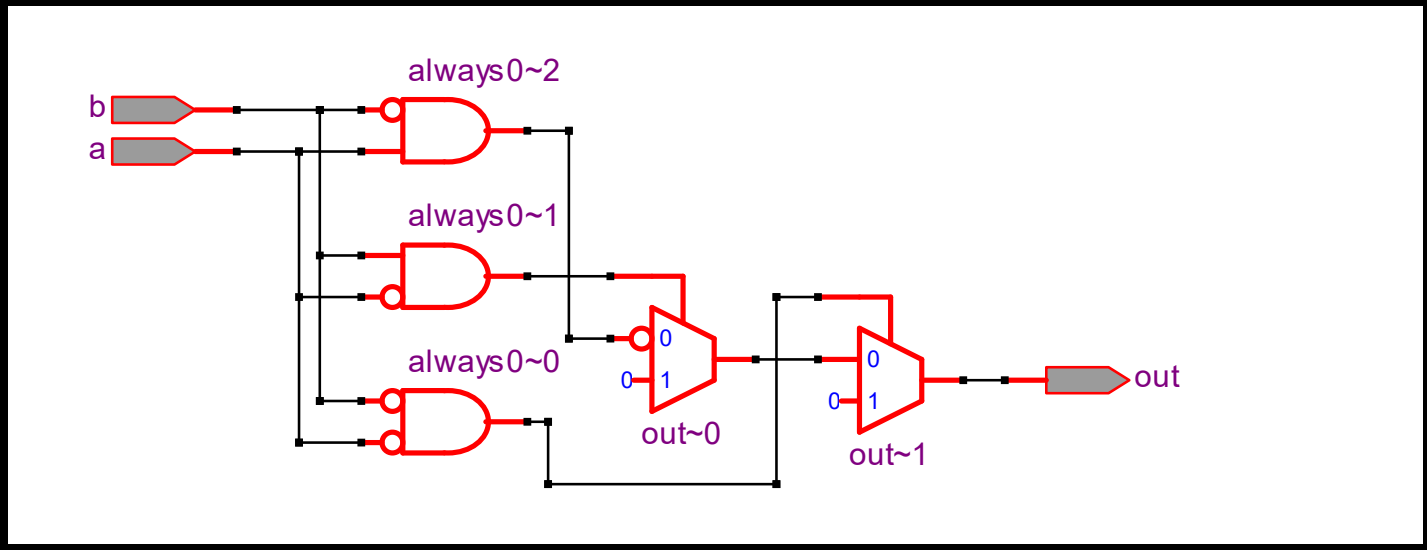
```
wire [2:0] a; // 3-bit
wire [3:1] b; // 3-bit
wire c; // 1-bit

assign a=b; ⇔ a[0] = b[1]; a[1]= b[2]; a[2] = b[3];
assign b=a; ⇔ b[1] = a[0]; b[2] = a[1]; b[3] = a[2];
assign c=a; ⇔ c = a[0];
assign a = 3'd7; ⇔ a = 3'b111; ⇔ a = 3'h7;
```

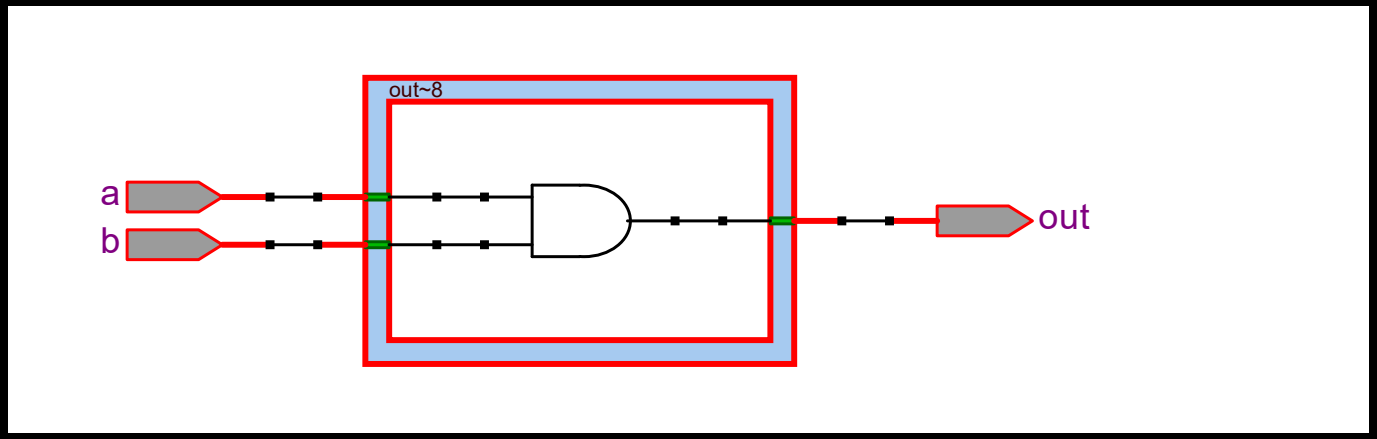
Behavioural Verilog using always block

- No assign statements inside always block
- All if must have corresponding else, i.e., all behaviours must be defined

```
module behavior_example (a, b, out);
input a,b;
output reg out;  //reg is an output wire inside always block
      always @(*) // for changes in any of the signals
            begin
                  if ( a==0 && b==0)
                        out = 1'b0;
                  else if (a==0 && b==1)
                        out = 1'b0;
                  else if (a==1 && b==0)
                        out = 1'b0;
                  else
                        out = 1'b1;
            end
endmodule
```

Results

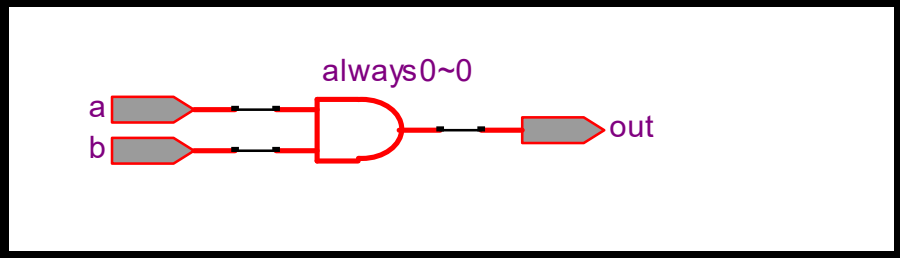

always0~2
b
a
always0~1
always0~0
0
0
1
out~0
0
0
1
out~1
out


out~8
a
b
out

Simplified Description

- Describes the same logic given previously

```
module behavior_example_simplified (input a, input b, output
reg out);
   always @(*)
      begin
        if ( a==1 && b==1)// can write if ({a,b} ==2b'11)
           out = 1'b1;
        else
           out = 1'b0;
      end
endmodule
```



Case Statement

- The same functionality using a `case` statement

```
module case_example (input a, input b, output reg out);
        always @(*)
                begin
                        case ({a,b})
                                2'b00: out=1'b0;
                                2'b01: out=1'b0;
                                2'b10: out=1'b0;
                                2'b11: out=1'b1;
                        endcase
                end
endmodule
```

Case Statement Variation

- An Another variation for the same circuit

```
module case_example_variation (input a, input b, output reg
out);
        always @(*)
                begin
                        case ({a,b})
                                2'b11: out=1'b1;
                              default: out=1'b0;
                        endcase
                end
endmodule
```

Important Points

- The fastest circuit is produced by

```
assign out = a & b;
```

- Cannot instantiate a module within an `always` block
- Don't write multiple `assign` statements or `always` blocks driving the same wire or reg
- Verilog is CASE SENSITIVE!

Summary

- Understanding Verilog for combinational logic
- Verilog Modules
- Defining and using `wire`
- Top-level module and submodules
- Module instantiation for multiple submodules
- Defining constants and bus structures
- Behavioural Verilog using `always` block
  - Using `if … else` statements
  - Using `case` statements
  - Using `reg` for always block output