FACULTY OF ENGINEERING

CAIRO UNIVERSITY

LAB 1
INTRODUCTION TO Verilog

CMP3010: COMPUTER ARCHITECTURE COURSE

WHAT IS Verilog?

- Verilog is a Hardware Description Language
- It is a programming language that allows one to model and develop complex digital systems.
- Verilog is C-like

HW DESIGNER GOAL

- Most ‘reliable’ design process, with minimum cost and time.
- Avoid design errors!

*Verilog is **Describing Hardware** so every line counts and cost a lot in real world*

Lexical Convention

The language is case sensitive.
Keywords are lower case letter.
Lexical convention are close to C++.
Comment

// to the end of the line.

/ to */ across several lines.*

Lexical Convention

Numbers are specified in the traditional form or below .

<size><base format><number>

Size: contains *decimal* digitals that specify the size of the constant in the number of bits.

Base format: is the single character ‘ followed by one of the following characters *b(binary),d(decimal),o(octal),h(hex)*.

Number: legal digital.

Lexical Convention

Example :

- 347 // decimal number
- 4’b101 // 4- bit binary number 0101
- ’o12 // octal number
- 12’h7f7 // 12-bit hex number 7f7
- 2’d3 // 2-bit decimal number

Program Structure

```
module  <module name> (< port list>);
   < declares>
   <module items>
endmodule
```

. Module name

an identifier that uniquely names the module.

. Port list

a list of input, inout and output ports which are used to other modules.

Program Structure

Example 1: Behavioral model

```
// Behavioral model of a Nand gate

module NAND(in1, in2, out);
  //  < declares>
      input in1,in2;
      output out;

     assign out=~(in1&in2);
endmodule
```

Data Types

Nets

- Nets are physical connections between structural entities.
- A *net* must be driven by a driver, such as a gate or a continuous assignment.
- Many types of nets, but all we care about is wire

Registers

- Implicit storage – unless variable of this type is modified it retains previously assigned value
- Does not necessarily imply a hardware register
- Register type is denoted by reg

Variable Declaration

Declaring a net

```
wire [<range>] <net_name>;
```

Range is specified as `[MSb:LSb]`. Default is one bit wide

Declaring a register

```
reg [<range>] <reg_name>;
```

Declaring memory

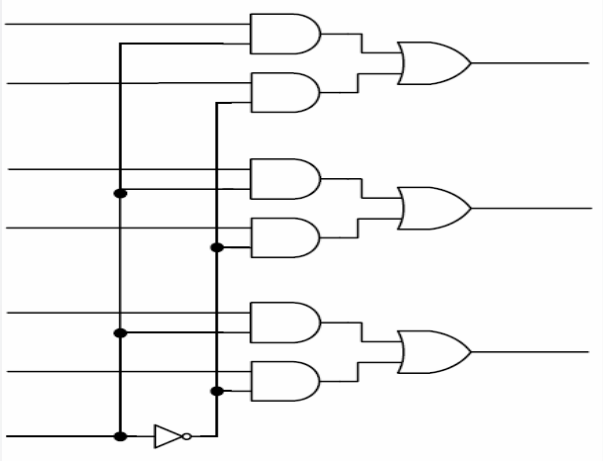
```
reg [<range>] <memory_name> [<start_addr>:<end_addr>];
```

Examples

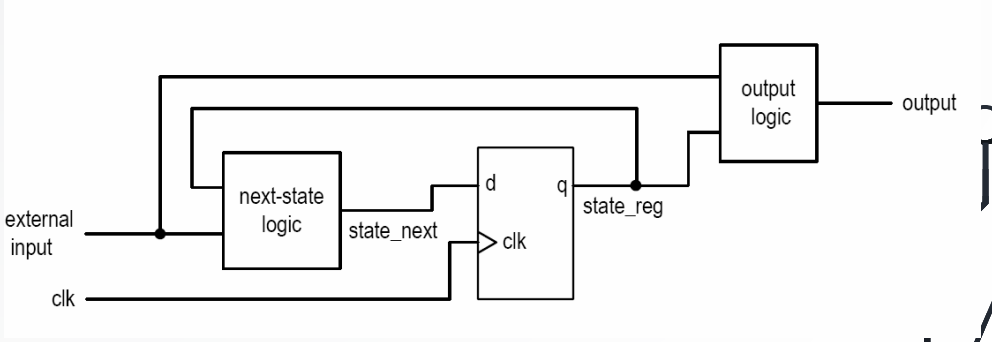
```
reg r; // 1-bit reg variable
wire w1, w2; // 2 1-bit wire variable
reg [7:0] vreg; // 8-bit register
reg [7:0] memory [0:1023]; a 1 KB memory
```

Combinational vs. sequential circuit

- **Combinational circuit:**
  - No latches/FFs or closed feedback loop
  - Output is a function of inputs only



- **Sequential Circuit**
  - With internal state (memory implemented by FF)
  - Output is a function of inputs and internal state



RTL Modeling

Behavioral Modeling.
Structural Modeling.

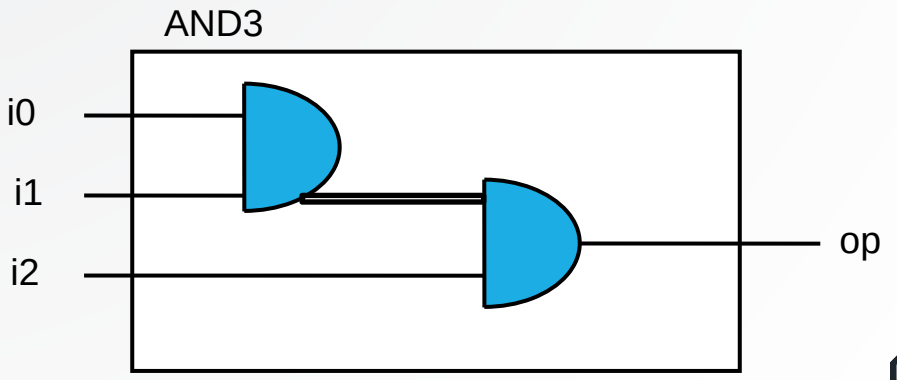
Structural Modeling

Example: //structural model of 3 input and gate

```
module AND(in1, in2, out2);
        input in1,in2;
                output out2;
        and and2(out2,in1,in2);// first port must be output.
endmodule
```

```
module AND3 (i0, i1, i2, op);
         input  i0, i1, i2;
         output op;
         wire temp;

AND a0 (.in1(i0), .in2(i1), .out2(temp));
AND a1 (.in1(i2), .in2(temp), .out2(op));
endmodule
```



Behavioral Modeling

- **Continuous assignment**
  - Starts with reserve word ‘assign’
  - Continuous execution
  - Models combinational circuits
- **Procedural blocks**
  - Initial blocks: executed once, used in tesbenches only
  - Always block: always executed, can model both combinational and sequential ciruicts

Continuous assignment

Continuous assignment statements drive nets (e.g.; wire data type).

- The left-hand side of a continuous assignment must be net data type.
- They are outside the procedural blocks (always and initial blocks).
- They can be used for modeling combinational logic and tri-state buffers.
- The continuous assign overrides any procedural assignments.

Examples:

```
//Explicit continuous assignment
wire [31:0] maxout;
assign maxout= in;

//Implicit continuous assignment
wire [31:0] x1=in;
```

Example 1: Half adder

```
module half_adder(x, y, s, c)
      input x, y;
      output s, c;
      assign s = x ^ y;
      assign c = x & y;
endmodule
```

can we make a fulladder using half_adder ?

Compile and simulate

Compile

```
vlog <filenames separated by space>
```

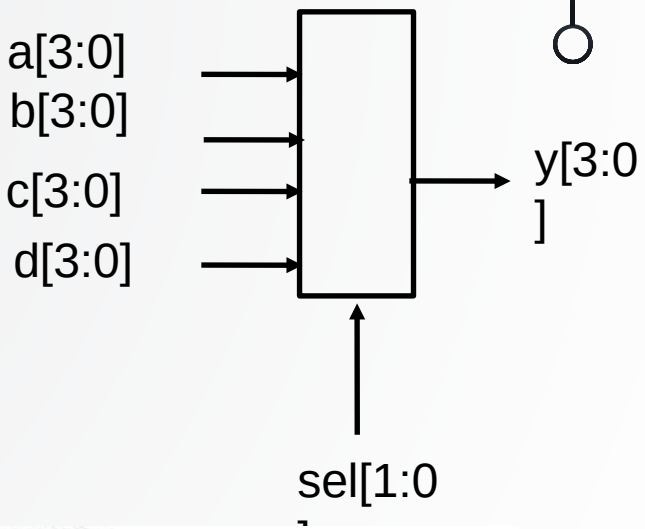
Simulate (do file)

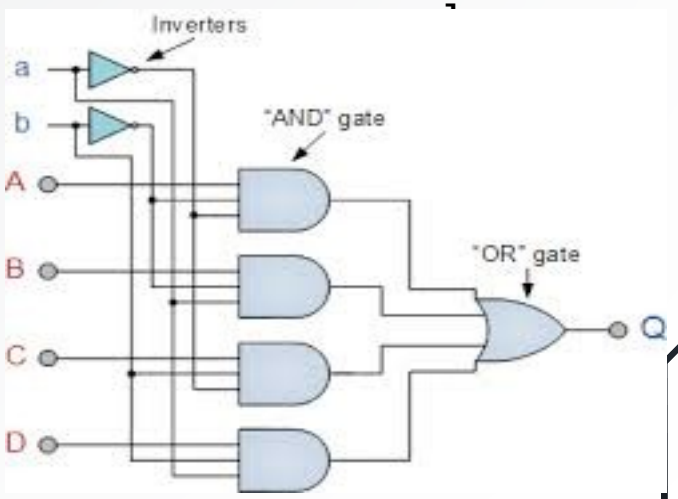
```
vsim <toplevel component>
add wave [componentName]/<signalName>
force [componentName]/<signalName> <value>
run
```

....

Run

```
do <dofilename>
```





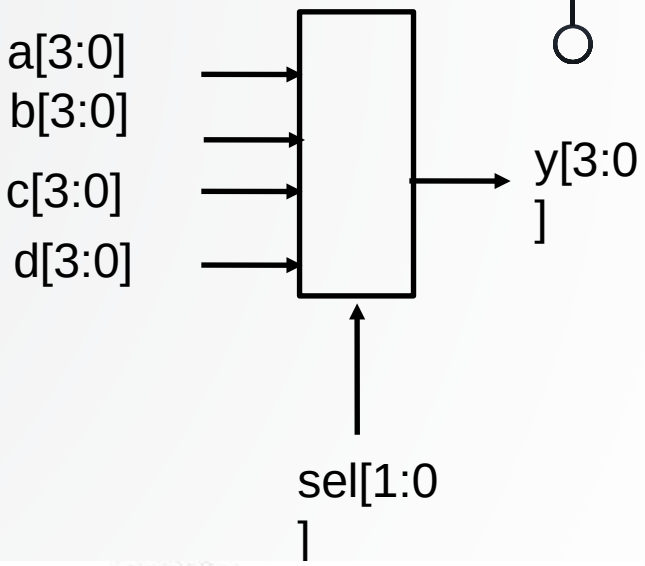
Compile and simulate

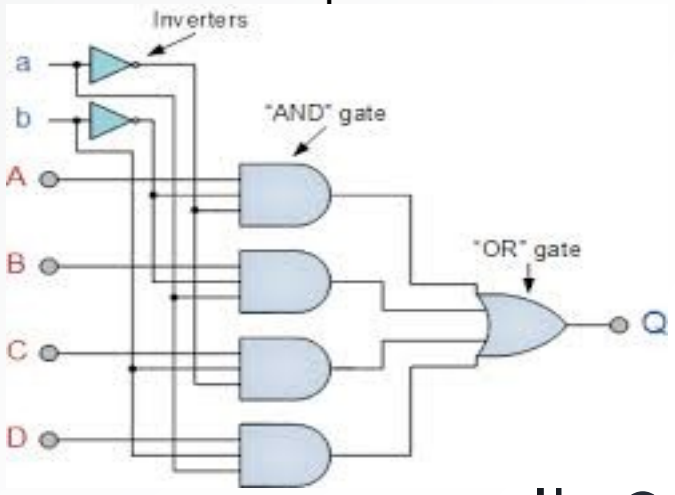
Compile

```
vlog half_adder.v
```

Simulate

```
vsim half_adder
add wave half_adder/x half_adder/y half_adder/s
add wave c
force half_adder/x 1
force half_adder/y 1
run
force half_adder/x 0
force half_adder/y 1
run
```

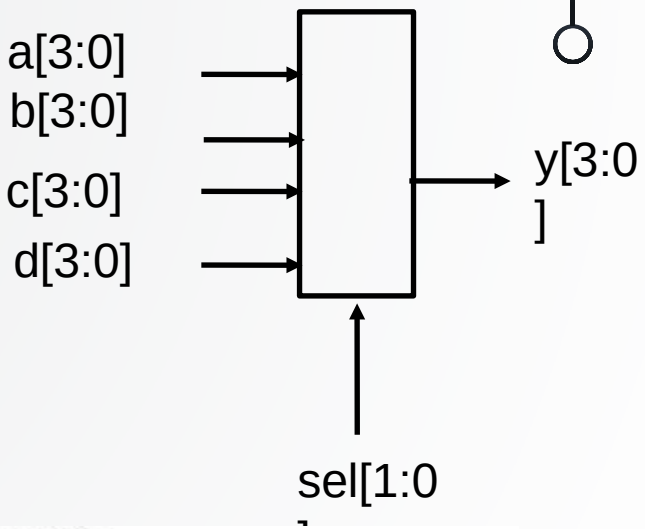


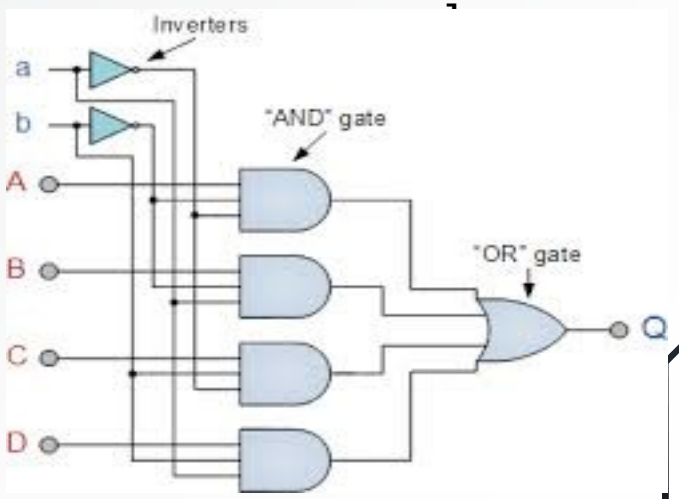


Example2 : 4x1 Multiplexer

Conditional Operator

```
module mux_4bits(y, a, b, c, d, sel);
  input [3:0] a, b, c, d;
  input [1:0] sel;
  output [3:0] y;
  assign y =
    (sel == 0) ? a :
    (sel == 1) ? b :
    (sel == 2) ? c :
    (sel == 3) ? d : 4'bx;
endmodule
```





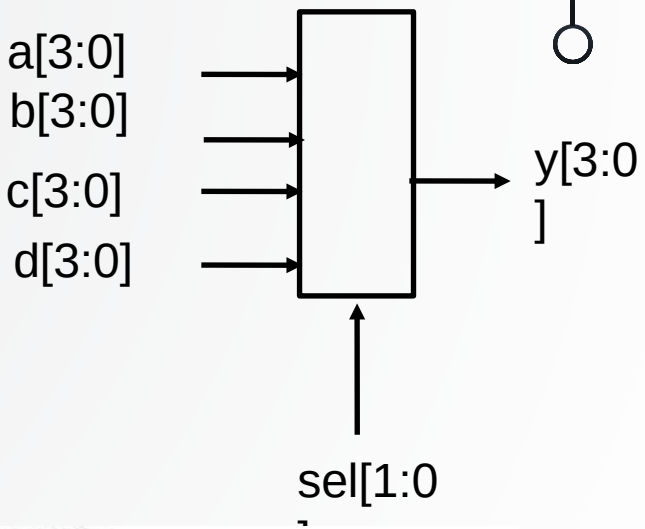
Compile and simulate

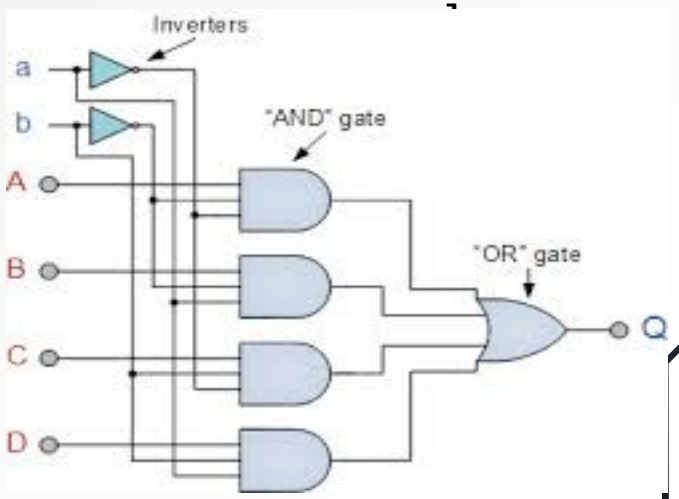
Compile

```
vlog mux_4bits.v
```

Simulate

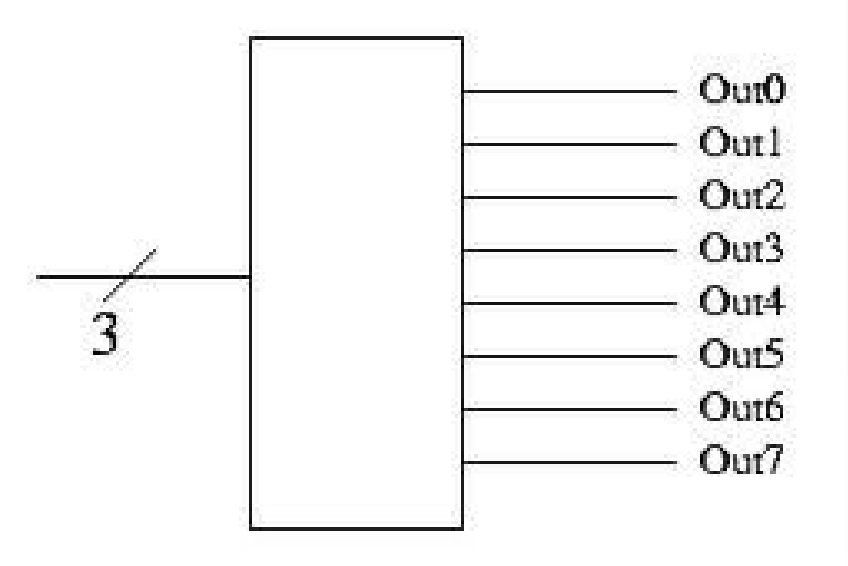
```
vsim mux_4bits
add wave mux_4bits/a mux_4bits/b mux_4bits/c
add wave d sel y
force mux_4bits/a 1
run
…
```





Example3: 3-to-8 decoder

```
module decoder (in,out);
   input [2:0] in;
   output [7:0] out;
   wire [7:0] out;
  assign out  =
   (in == 3'b000 ) ? 8'b0000_0001 :
   (in == 3'b001 ) ? 8'b0000_0010 :
   (in == 3'b010 ) ? 8'b0000_0100 :
   (in == 3'b011 ) ? 8'b0000_1000 :
   (in == 3'b100 ) ? 8'b0001_0000 :
   (in == 3'b101 ) ? 8'b0010_0000 :
  (in == 3'b110 ) ? 8'b0100_0000 :
  (in == 3'b111 ) ? 8'b1000_0000 : 8'h00;
endmodule
```



Exercise

- Implement 8*3 Encoder
  - The Encoder has 8 bits input, 3 bits output, and enable signal
- Write do file to compile & run simulation covering all cases.
- **Bonus: In do files** use loop to create simulation scenario [do files are TCL scripts]

TIPS

1. Always Save & Compile before simulation.
2. Read the Error/Warning messages in “Transcript” tap.
3. Change the “Radix” to make the simulation easier (right click on signal name in simulation).
4. Re-writing your code all over again will **NOT** solve your problems.
5. For any Error, check the few lines before the line with error message.
6. Always use Do files instead of Changing the inputs every time.