EE 421 / CS 425
Digital System Design
Laboratory 2

Fall 2025

Shahid Masud

LUMS
A Not-for-Profit University

Today's Topics

- Learn Questa (or Modelsim) Simulation Tool in the Lab Manual
- Introduction to SystemVerilog HDL
- Structural SystemVerilog Design
- Writing Simple Test Benches
- Examining the Output of Simulation Tool
- Lab Tasks and Lab Deliverables

Step Through EDA Tools

Design Capture → Text Editor

Functional Simulation → Questa or Modelsim

Synthesis → Xilinx Vivado

Post-Synthesis Simulation → Modelsim

Place and Route → Xilinx Vivado

Product FPGA / ASIC

SIMULATION

SYNTHESIS

Timing, Power, Area

Simulation Tool Questa

https://eda.sw.siemens.com/en-US/ic/questa-one/simulation/questa-one-sim/

SIEMENS

Questa One | Overview | Solutions | Products | Ver

SMART VERIFICATION SOLUTION

Questa One

Questa One smart verification solution delivers a transformative shift to AI-powered verification, pioneering intelligent automation to break past traditional limitations and empower teams with unmatched speed, efficiency and scalability.

Read white paper | Read press release

Questa One Sim is the evolution of the functional simulator. It combines various verification flow aspects, boosting performance and productivity by leveraging faster engines. This enables faster engineers who use fewer workloads to finish verification tasks.

Read white paper | View fact sheet

Why Questa One Sim?

Questa One Sim brings cornerstone Questa simulation technologies into a platform that is geared towards solving challenges faced by semiconductor companies designing the next generation of SoC's, ASIC's and FPGAs. Questa One Sim enables faster engines, faster engineers and fewer workloads.

Hardware Description Languages

- Specialized Computer Language for Capturing Hardware Design Idea
- Can be used to describe structure and behavior of digital circuits
- HDL include notion of hardware specific time or delay
- They have support for Concurrency which is peculiar to hardware

Design Capture in SystemVerilog HDL

SystemVerilog Allows Design Capture at Various Hierarchy Levels:

1. Switch Level (NMOS and PMOS transistors)
2. Gate Level (Describing Circuit as Logic Gates)
   a) Data Flow (RTL) Level
3. Dataflow Level
4. Behavior Level

The Design is Captured in a text file or obtained from schematic diagram

Modular Approach in SystemVerilog

- Module is a basic building block
- Module can be a collection of multiple low level logic blocks
- Modules are interconnected through Port Interface
- Internal functionality of any Module can be altered without affecting any other Module

```
module module_name (x,y,z);
    input x,y;
    output z;
    statements ;
    ................... ;
    ................... ;

endmodule
```

```
module top_Level (a, b, c.....)
...
...
  module use_1 (a_wire, b_wire,...);
  module use_2 (c_wire,d_wire,...);

endmodule
```

LUMS
A Not-for-Profit University

SystemVerilog Syntax

```
// comments
module ExampleOne( output f,
                   input  a, b);

   wire Ax, Bx;    // Internal wires

   /* Behavioral description */
   assign Ax = a && b;
   assign Bx = a || b;
   assign f = (Ax && Bx) || a;

endmodule

```

Post-2001 Verilog allows the port name, direction, and type to be declared together.

- Each port needs to have a user-defined name.
- The port directions are declared to be one of the three types: **input**, **output**, or **inout**.
- A port can take on any of the data types, but only **wires**, **registers**, and **integers** are synthesizable.

In new syntax, the input, output is defined as above.
In old syntax, this definition was after the module (…);
input a, b, …..;
output c,d,….;

SystemVerilog Basics – Comments and Number System

- Verilog uses a C-like syntax. It is case sensitive, and all keywords are in lower case letters. Declarations, assignments, and statements end with a semicolon. It also uses C-style comments:

  // comment to end of line

  /* closed comment */

- Numbers are represented as <number_of_bits>'<base><number>, where base can be b, o, d, or h, for binary, octal, decimal, or hex, respectively. Some examples:

```
8'hFF      //8-bit hex number FF
5'b101     //5-bit binary number 00101
1          //decimal number 1  (decimal is the default base)
3'o5       //3-bit octal number 5
4’b1101 //4-bit binary number 1101
16’d255  //16 bit decimal number 255
```

LUMS
A Not-for-Profit University

Primitive Data Types in SystemVerilog

- The data types that you will use are reg (register) and wire.
- Wires cannot hold a value; they are used to connect modules of combinational logic.
- Regs are used to store values. Since regs keep state, a reg cannot be the output of combinational logic.
- Regs are only used in behavioral Verilog.

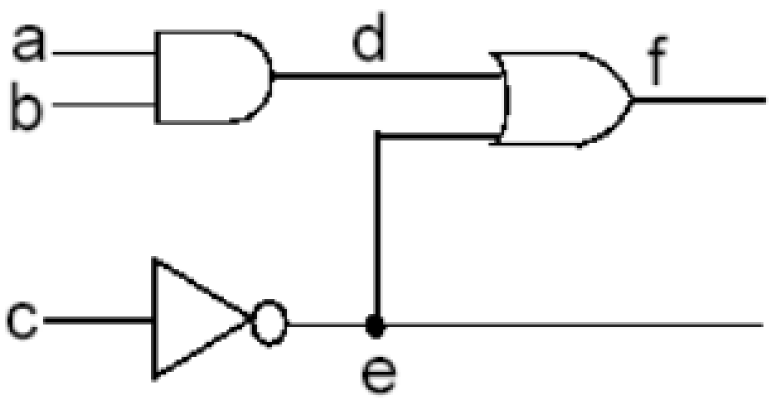
- Declare regs and wires as follows:

```
reg a,b;              //two scalar registers a, b
reg [0:7] byte;       //8-bit vector, bit 0 is MSB
wire u;               //one bit wire called u
wire [31:0] word;     //32-bit vector called word, bit 31 is MSB
```

LUMS
A Not-for-Profit University

Structural SystemVerilog (port mapping)

Components available in library (primitives) are port mapped




```
module example(a, b, c, f, e);
  input a, b, c;
  output f, e;
  wire d;
  and g1(d, a, b);
  not g2(e, c);
  or  g3(f, d, e);
endmodule
```

Youpyo Hong, Dongguk University

LUMS
A Not-for-Profit University

Values of reg and wire in SystemVerilog

- Standard logic values
  - **0** and **1** represent logic 0 (false) and logic 1 (true)
- **x** or **z** values are useful in modeling
- Unknown value is denoted by **x**
- High Impedance value is denoted by **z**

```
6'hx     //6-bit hex number with unknown value
32'bz    //32-bit binary number in high impedance state
```

LUMS
A Not-for-Profit University

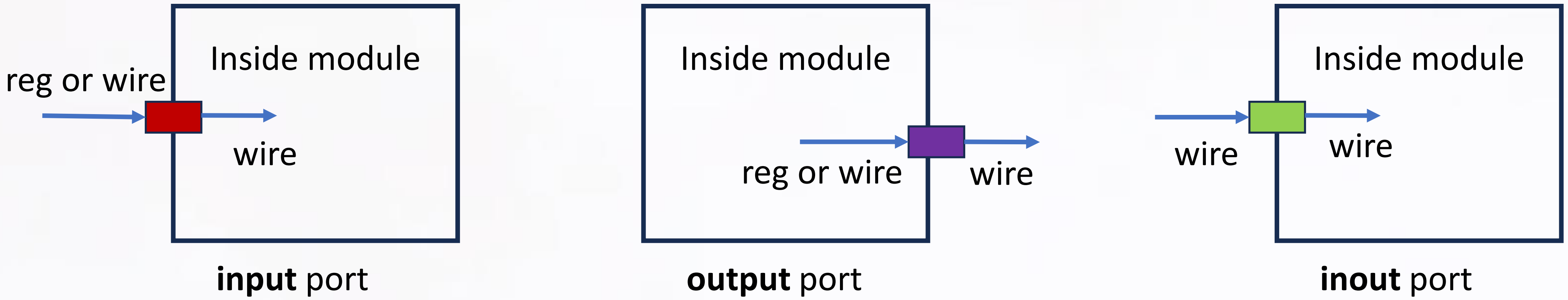
Port Assignment in SystemVerilog

Ports can be:

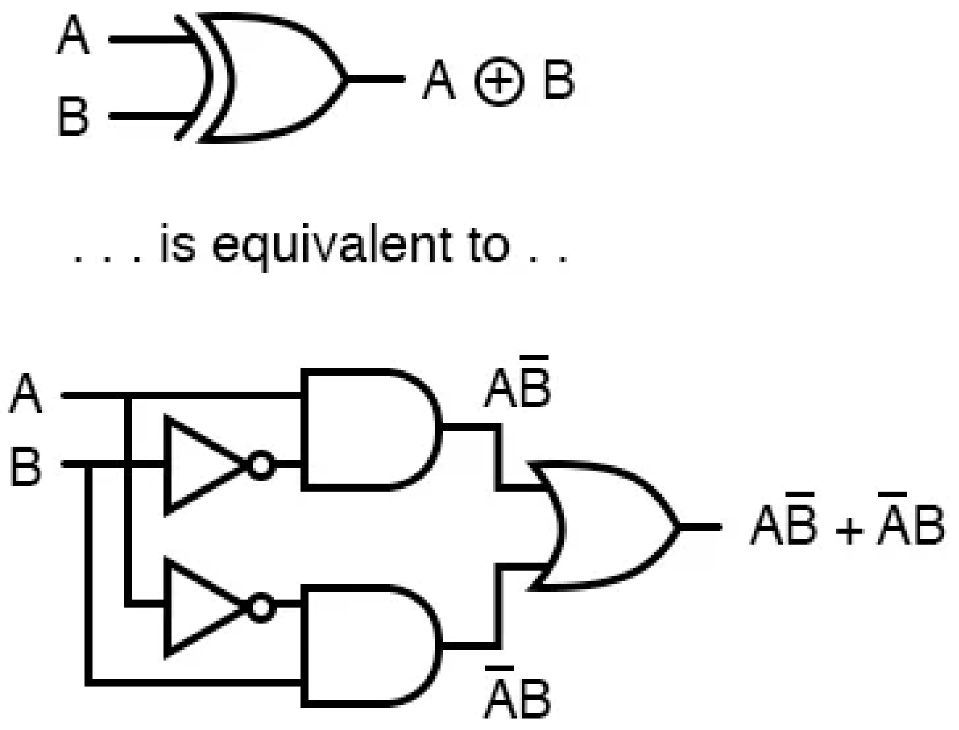
input port – Internally a wire

output port – Internally a reg or a wire

combined input and output port – only a wire



Structural View of an XOR gate



$$A \oplus B = A\bar{B} + \bar{A}B$$

SystemVerilog Instantiation of Primitive Gates

```
/* An example of building a basic
XOR function of two single bit
inputs a and b */
module build_xor (a, b, c);
 input a, b;
 output c;
      wire c, a_not, b_not;
      not a_inv (a_not, a);
      not b_inv (b_not, b);
      and a1 (x, a_not, b);
      and a2 (y, b_not, a);
      or out (c, x, y);
endmodule
```

```
/* Structural description of a half
adder composed of four, 2 input
nand modules */
module halfadd (X, Y, C, S);
input X, Y;
output C, S;
      wire S1, S2, S3;
      nand NANDA (S3, X, Y);
      nand NANDB (S1, X, S3);
      nand NANDC (S2, S3, Y);
      nand NANDD (S, S1, S2);
      assign C = S3;
endmodule
```

LUMS
A Not-for-Profit University

Key Reserved Words for Structural Design

- and(output, *input 1, input 2, …..*)
- or()
- not()
- nand()
- nor()
- xor()
- xnor()
- wire // to interconnect inputs or outputs
- //comment in code

and (y,a,b); //instantiate AND gate with output y and inputs a,b

LUMS
A Not-for-Profit University

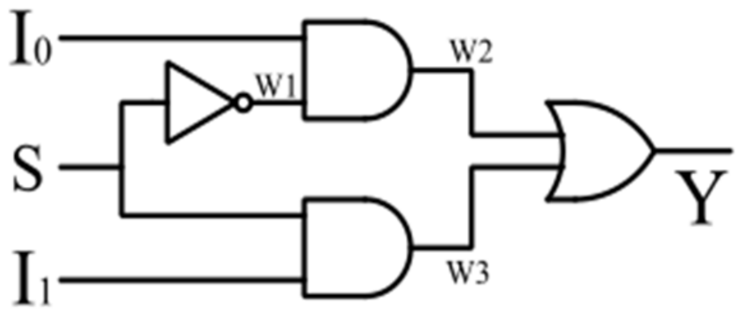
Structural SystemVerilog for 2:1 MUX

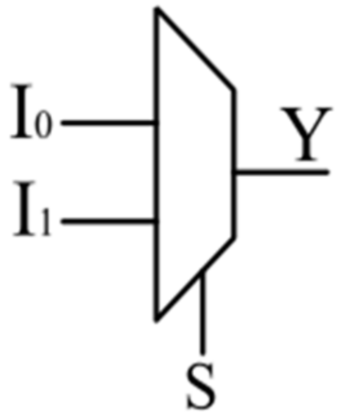
```
module mux2x1(
    input I0,I1,S,
    output Y
    );
      wire w1,w2,w3;
      not my_not(w1,S);
      and my_and1(w2,I0,w1);
      and my_and2(w3,I1,S);
      or my_or(Y,w2,w3);
endmodule
```

Boolean Expression

$$Y = \bar{S}I_0 + SI_1$$

Circuit Diagram

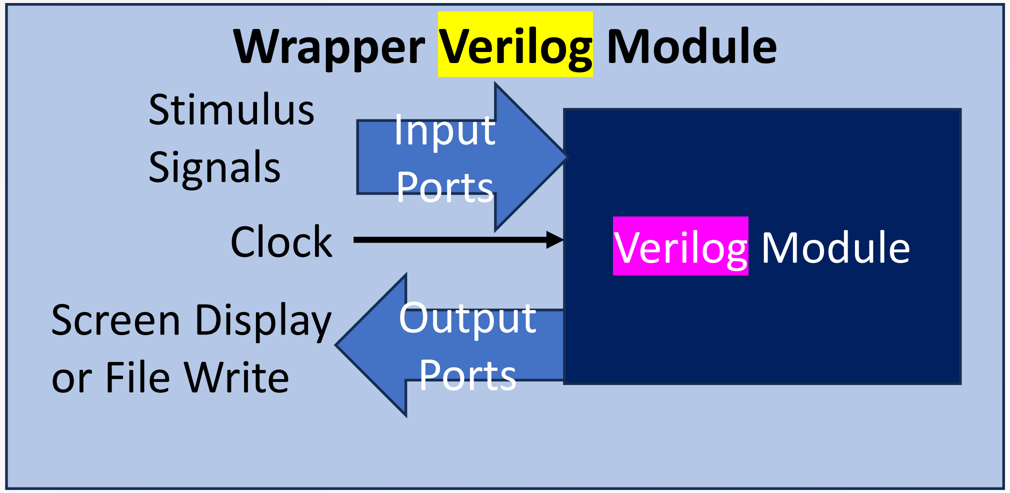




Test Bench

- Used for Functional as well as Post-Synthesis Testing
- Timing Information can be added through Gate Models

Writing Test Benches in SystemVerilog




```
module design (input a, b, c,
               output y);

  assign y = ~b & ~c | a & ~b;
endmodule
```

```
module tb ();
  reg a, b, c;
  wire y;

  design dut (.a(a), .b(b), .c(c), y(y));

  // apply input sequence

  initial
    begin

      a = 0; b = 0; c = 0 ; # 10;
      a = 0; b = 0; c = 1 ; # 10;
      a = 0; b = 1; c = 0 ; # 10;
      a = 0; b = 1; c = 1 ; # 10;
      a = 1; b = 0; c = 0 ; # 10;
      a = 1; b = 0; c = 1 ; # 10;
      a = 1; b = 1; c = 0 ; # 10;
      a = 1; b = 1; c = 1 ; # 10;


    end
endmodule
```

Testbench Examples 1

```
`timescale 1ns/10ns
module testbench1();  // Testbench has no inputs, outputs
reg a, b, c; // Will be assigned in initial block
wire y;
// instantiate device under test
sillyfunction dut (.a(a), .b(b), .c(c), .y(y) );d
// apply inputs one at a time
initial begin         // sequential block
       a = 0; b = 0; c = 0; #10; // apply inputs, wait 10ns
       c = 1; #10; // apply inputs, wait 10ns
       b = 1; c = 0; #10; // etc .. etc..
       c = 1; #10;
       a = 1; b = 0; c = 0; #10;
end
endmodule
```

LUMS
A Not-for-Profit University

Timescale in Questa (or Modelsim) Simulations

- At the top of test bench Verilog code, a compiler directive:

`timescale 2 ns / 1000 ps is included

- This line is vital in a Verilog simulation because it sets up the timescale and operational precision for a design.
- It sets the unit delays to be in nanoseconds (ns) and the accuracy at which the simulator will round the procedures down to 1000 ps.
- This causes a #2 or #4 in a Verilog test bench assignment to be a 2 ns or 4 ns delay, respectively.

More on `timescale

```
`timescale <time_unit>/<time_precision>

// for example

`timescale 1ns/1ps
`timescale 10us/100ns
`timescale 10ns/1ns
```

The time_unit is the measurement of delays and simulation time, while the time_precision specifies how delay values are rounded before being used in the simulation.

LUMS
A Not-for-Profit University

Example of `timescale

```
`timescale 10ns/10ns
module tim( );
reg i;
initial
begin
      i=0;
      #7.7212;
      i=1;
      $display("STATEMENT 1 :: time is ",$stime);
      #7.123;
      $finish;
end
endmodule
```

In the `timescale statement, the first value is the time unit and the second is the precision for the simulation. So with the time unit, when the simulator displays a value, you just have to multiply the value by this time unit to get the real time. With a 10ns time unit, a delay of #7.7212, that means that it is 77.212ns delay.

Testbench of AND gate

```
`timescale 10ns/10ns
module basic_and_tb();
  reg [3:0] a, b;
 wire [3:0] out;
  basic_and #(.WIDTH(4)) DUT (
   .a(a),    .b(b),    .out(out)
 );
  initial begin
   a = 4'b0000;
   b = 4'b0000;
   #20
   a = 4'b1111;
   b = 4'b0101;
   #20
   a = 4'b1100;
   b = 4'b1111;
   #20
   a = 4'b1100;
   b = 4'b0011;
   #20
   a = 4'b1100;
   b = 4'b1010;
   #20
   $finish;
  end

endmodule
```

Lab 2 Task – Design Capture and Questa (or Modelsim) Simulations

- Refer to Lab Manual 2.
- Lab Deliverables Document needs to be completed and submitted for each lab.
- Upload the Lab Deliverables Document on LMS Dropbox.