**CS226- Hardware Lab 8**

**Hardware design using HDL(Verilog)**

The goal of this lab is to familiarize the students with describing computer architectural blocks in Verilog Hardware Description Language (HDL) and to learn how to handle the simulator. The goal is also to interpret outputs from a simulator and to understand how Verilog code is interpreted by the simulator. Further we will simulate various computer harware building blocks.

**Simulator (modelsim).**

**Install Modelsim: http://model.com/content/modelsim-pe-student-edition-hdl-simulation**

**In windows, create a lab directory for the course in your home directory (eg. c:/cs225/Lab8). Download the Lab8 files .**

**Start modelsim**

**Start -> programes->modelsim->**

**Create a project (File->New->Project)**

**Name the project and project location as shown**

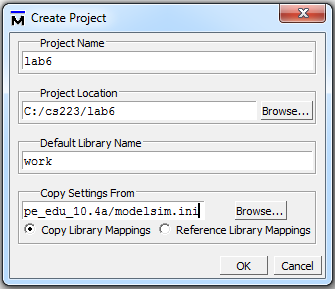
****

Figure 1

**Task1: simulating an AND gate**

**In this exercise you shall simulate an AND gate (figure 2).** Consider the code in listing. This code declares a single module, which you can think of as a class. The module's name is *and\_gate*, which is descriptive, as we want to simulate an AND gate.



Figure 2



Figure 3

// name: and\_gate.v

*module and\_gate(out, a,b);* //you list all inputs and outputs, by convention outputs go first

*output out; // this tells the compile which lines are inputs and outputs*

*input a, b;*

*assign out = a & b; // output function*

*endmodule*

*module tb\_and\_gate(); // Test bench fortb\_ and\_gate.v*

*reg a,b; //* a reg, to allow us to assign the input, and a wire to receive the output

*wire out;*

*and\_gate uut(out,a,b); //* this instantiates a and gate, uut is a label

*initial*

*begin*

*a = 1'b0; // here we apply inputs to the gate*

*b = 1'b0;*

*#10;*

*a = 1'b0;*

*b = 1'b1;*

*#10;*

*a = 1'b1;*

*b = 1'b1;*

*#10;*

*a = 1'b1;*

*b = 1'b0;*

*#10;*

*end*

// set up the monitoring

*initial*

*begin*

*$monitor("a=%b, b=%b, out=%b, time=%t\n", a, b, out, $time);*

*end*

*endmodule*

Now that we have a module (and\_gate), we need to make an instance and simulate it for testing purposes. We only need to simulate and test, so we need to write a test bench. The terminology goes back to the old days of hardwired testing (usually with wire wrap), where you would have a physical testing bench, which typically was set up to rapidly connect and test circuits. The test bench creates virtual environment to verify the correctness design (in this case “and” gate). See figure 3 for a conceptual representation.

The $monitor command continuously monitors the values of the variables or signals specified in the parameter list and displays all parameters in the list whenever the values of any one variable or signal changes. $monitor only needs to be invoked once. Only one monitor list can be active at a time.

**Step by step intructions:**

**From modelsim winow: File-> Add to Project->Existing Files: Select file ‘and\_gate.v’ and tb\_and\_gate.v**

**Select and\_gate.v in the project tab**

**Compile**

**Now compile tb\_and\_gate.v**

**To simulate:**

**Simulate -> Start simulation**

**Click the plus sign (‘+’) near work**

**Choose tb\_and\_gate and click OK**

In the structural window (labeled as ‘sim’) you can see the structure of the design. The source code for the module that is chosen in the structure window is displayed in the source window.

In the object window: Right click and choose Add to Wave -> signals in Region

To simulate select; Simulation -> Run -> Run -all

In the wave window, you can see the waveform for the selected signals

In this simulation outputs are also visible at main console window.

**Study the simulation output and compare them with the code. Examine the output signal of the gate.**

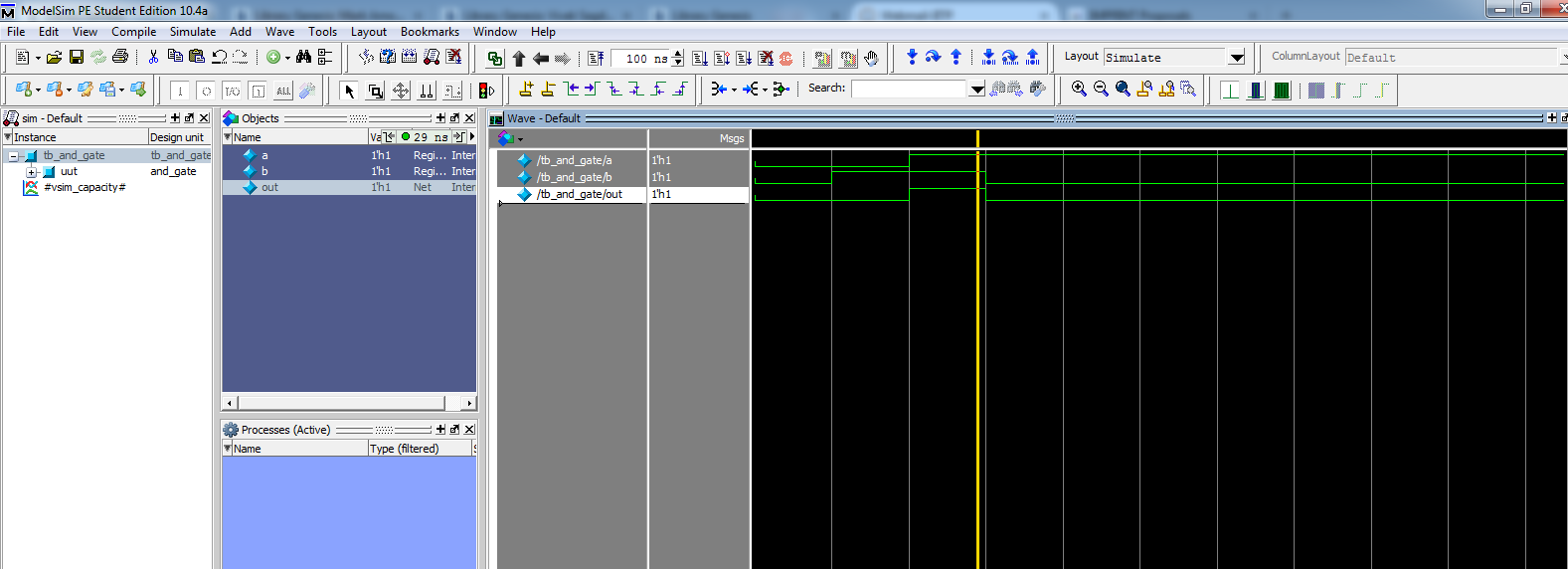
****

Fig4: Simulation output

**Task2: Repat the above step by creating or\_gate.v and tb\_or\_gate.v**

// name: or\_gate

*module or\_gate(out,a,b);* //you list all inputs and outputs, by convention outputs go first

*output out; // this tells the compile which lines are inputs and outputs*

*input a, b;*

*assign out = a | b; // output function*

*endmodule*

*module tb\_or\_gate(); // Test bench for and\_gate.v*

*reg a,b; //* a reg, to allow us to assign the input, and a wire to receive the output

*wire out;*

*or\_gate uut (out,a,b); //* this instantiates a and gate, uut is a label

*initial*

*begin*

*a = 1'b0; // here we apply inputs to the gate*

*b = 1'b0;*

*#10;*

*a = 1'b0;*

*b = 1'b1;*

*#10;*

*a = 1'b1;*

*b = 1'b1;*

*#10;*

*a = 1'b1;*

*b = 1'b0;*

*#10;*

*end*

// set up the monitoring

*initial*

*begin*

*$monitor("a=%b, b=%b, out=%b, time=%t\n", a, b, out, $time);*

*end*

*endmodule.*

***Exercise: Simulate 3input / input AND gate and OR gate. Write appropriate testbench.***

**Task 3: Types Modeling in Verilog**

In this part we will model a logic function in different ways. Consider the following table

|  |  |  |  |
| --- | --- | --- | --- |
| sel | In2 | In1 | out |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 |

Out put can be simplied as

***Out=in1. sel +in2. sel’*** and the implementation is shown in figure 4



Figure 4

**There are various ways of modeling in Verilog.**

1. **Structural**
2. **Behavioural level**

**Let us call this function as mux and now we model the function in the above 3 methods.**

1. **Structural**

**Structural Style:** *The circuit is specified in terms of lower level components ( in this example logic gates, which are Verilog primitives) connected with internal signals. The translation of such a specification into a physical circuit is straight forward.*

// lines start with “//” is a comment

// name: mux\_gate.v // gate level model

***module*** *mux\_struct(out, in1,in2,sel );* //you list all inputs and outputs, by convention outputs go first

***output*** *out; // this tells the compile which lines are inputs and outputs*

***input*** *in1,in2,sel;*

***and*** *a1 (a1\_o,in1,sel); // defines the a1 gate, see figure*

***not*** *n1 (inv\_sel,sel); // defines the inverter gate, see figure*

***and*** *a2 (a2\_o,in2,inv\_sel); // defines the a2 gate, see figure*

***or*** *o1 (out, a1\_o, a2\_o); // defines the 01 gate, see figure*

***endmodule***

1. **Behavioural level**
2. **Behavioural Style:** *It specifies the circuit in terms of its expected behaviour. It is the closest to a natural language description of the circuit functionality.*

// lines start with “//” is a comment

// name: mux\_behav.v // Behavior: event-driven behavior description construct

***module*** *mux\_behav(out, in1,in2,sel );* //you list all inputs and outputs, by convention outputs go first

***output*** *out; // this tells the compile which lines are inputs and outputs*

***input*** *in1,in2,sel;*

***reg*** *out;*

***always@****(in1 or in2 or sel)*

***begin***

***if*** *(sel)*

*out =in1;*

***else***

*out=in2;*

***end***

***endmodule***

1. **Behavioural Style:**

// lines start with “//” is a comment

***module*** *mux\_ behav2(out, in1,in2,sel );* //you list all inputs and outputs, by convention outputs go first

***output*** *out; // this tells the compile which lines are inputs and outputs*

***input*** *in1,in2,sel;*

***assign*** *out=sel ? in1: in2;*

***endmodule***

module tb\_mux\_behav();

reg in1,in2,sel;

wire out;

mux\_behav UUT (out,in1,in2,sel);

initial

begin

in1 = 1'b0; // here we apply inputs to the logic

in2 = 1'b0;

sel = 1'b0;

#10;

in1 = 1'b0;

in2 = 1'b1;

sel = 1'b0;

#10;

in1 = 1'b1;

in2 = 1'b0;

sel = 1'b0;

#10; end

// set up the monitoring

initial begin

$monitor("sel=%b, in1=%b, in2=%b, out=%b, time=%t\n", sel, in1,in2, out, $time); end

endmodule

**Your Lab Assignment 8 (60 points, 10 points each)**

**Submission** Your submission must contain:

* The source code of your design/testbench if any (for each of the problem) - reasonably well documented.
* A *minimal* document detailing the design/ testbench/answers.

Course work submission through [cs225.iitp@gmail.com](mailto:cs225.iitp@gmail.com) with subject: YourrollNo\_Lab8.

**Due at end of the Lab session.**

1. Model the following logic function using Verilog gate level description and design appropriate test bench.



1. A car has a fuel level detector that outputs the current fuel level as a 3-bit binary number, with 000 meaning empty and 111 meaning full. Create a simplified logic circuit (use Boolean algebra for simplification) that illuminates a “low fuel” indicator light (by setting an output L to 1) when the fuel level drops below level 3. Develop a Verilog model for the logic and test bench to verify the model.
2. Use a simulation to demonstrate whether or not the following functions are equal. (Be sure to state which, if any, are equal.) and verify using truth table

f = a b' + b' c' + a c

g = (b' + c) (a + b + c')

h = b' c' + b c + a c

1. Simulate the following function using appropriate test bench.

*module comparators(eq,neq,lt,lte,gt,gte,a,b);*

*parameter N = 8;*

*input [N-1:0] a, b;*

*output eq, neq;*

*output lt, lte;*

*output gt, gte;*

*assign eq = (a == b);*

*assign neq = (a != b);*

*assign lt = (a < b);*

*assign lte = (a <= b);*

*assign gt = (a > b);*

*assign gte = (a >= b);*

*endmodule*

(5) A museum has three rooms, each with a motion sensor (m0, m1, m2) that outputs 1 when motion is detected. At night, the only person in the museum is one security guard who walks from room to room. Create a circuit that sounds an alarm (by setting an output A to 1) if motion is ever detected in more than one room at a time (i.e., in two or three rooms), meaning there must be an intruder or intruders in the museum. Develop a Verilog model for the above and write a test bench to verify the model.

**(6**)Write a structural description of the following schematic. Check the functionality of the circuit by writing appropriate test bench.



**(6**)Write a structural description of the following schematic. Check the functionality of the circuit by writing appropriate test bench.

(7) Design 4 to 1 multiplexer with different coding styles ( at lease 5) and test using appropriate test bench. Include all the designs in a single file( p7.v and tb\_P7.v) and submit.