**Laboratory Exercise #1**

**Logic Level Modelling in Verilog**

**Objective**

The first laboratory’s objective is to review how to do logic level modelling by designing Ripple Carry Counter and 4 to 1 Multiplexer. To do so, I made use of the simulator built into the Vivado Design Suite provided by Xilinx and generated waveform results.

**Design**

1. Launch and install vivado and create a new design project.

a) Open a terminal window and create a home user directory to save lab projects. Run commands to install and launch vivado.

b) Create a new project called ‘lab1’ and Leave the device properties as default.

2. Simulate a Ripple Carry Counter

a) Add sources for simulating a Ripple Carry Counter.

b) Implement design source code logic for Ripple Carry Counter top block module, T Flip Flop Module and D flip Flop Module.

c)Implement stimulation/testbench code for testing the simulation results of the ripple carry counter by varying reset and clock signal cycle time that drives the design block for short time intervals.

d) Run synthesis and implementation post which run Behavioural stimulation of the code logic and observe results on waveform generator.

e) Monitor the outputs and capture the results obtained.

3. Source Files:

a) \*\*\*\*RIPPLE CARRY COUNTER BLOCK\*\*\*\*

`timescale 1ns / 1ps  
//////////////////////////////////////////////////////////////////////////////////  
// Company:  
// Engineer:  
//  
// Create Date: 09/04/2019 10:28:45 AM  
// Design Name:  
// Module Name: RCC  
// Project Name:  
// Target Devices:  
// Tool Versions:  
// Description:  
//  
// Dependencies:  
//  
// Revision:  
// Revision 0.01 - File Created  
// Additional Comments:  
//  
//////////////////////////////////////////////////////////////////////////////////  
  
module RCC (q, clk, reset); // Module named RCC having input output parameters  
output [3:0] q; // 4-bit output named q  
input clk, reset; // Single bit input named clk and reset  
T\_FF tff0(q[0], clk, reset); // Calls module T\_FF having T Flip Flop code logic  
T\_FF tff1(q[1], q[0], reset);  
T\_FF tff2(q[2], q[1], reset);  
T\_FF tff3(q[3], q[2], reset);  
endmodule

b) \*\*\*\*FLIP-FLOP T\_FF\*\*\*\*

`timescale 1ns / 1ps  
//////////////////////////////////////////////////////////////////////////////////  
// Company:  
// Engineer:  
//  
// Create Date: 09/04/2019 10:30:52 AM  
// Design Name:  
// Module Name: T\_FF  
// Project Name:  
// Target Devices:  
// Tool Versions:  
// Description:  
//  
// Dependencies:  
//  
// Revision:  
// Revision 0.01 - File Created  
// Additional Comments:  
//  
//////////////////////////////////////////////////////////////////////////////////  
  
module T\_FF(q, clk, reset); // Module named T\_FF having input output parameters  
output q; // 4-bit output named q  
input clk, reset; // Single bit input named clk and reset  
wire d;  
D\_FF dff0(q, d, clk, reset); // Calls module D\_FF having D Flip Flop code logic  
not n1(d, q); // not is a verilog-provided primitive  
endmodule

c) \*\*\*\*FLIP-FLOP D\_FF\*\*\*\*

`timescale 1ns / 1ps  
//////////////////////////////////////////////////////////////////////////////////  
// Company:  
// Engineer:  
//  
// Create Date: 09/04/2019 10:31:19 AM  
// Design Name:  
// Module Name: D\_FF  
// Project Name:  
// Target Devices:  
// Tool Versions:  
// Description:  
//  
// Dependencies:  
//  
// Revision:  
// Revision 0.01 - File Created  
// Additional Comments:  
//  
//////////////////////////////////////////////////////////////////////////////////

module D\_FF(q, d, clk, reset); // Module named D\_FF having input output parameters  
output q; // 1-bit output named q  
input d, clk, reset; // Single bit input named clk and reset  
reg q;  
always @(posedge reset or negedge clk) // Declaration of synchronous behaviour  
if (reset) // If reset is set to 1 set q to 0 else set to input data  
q = 1'b0;  
else  
q = d;

endmodule

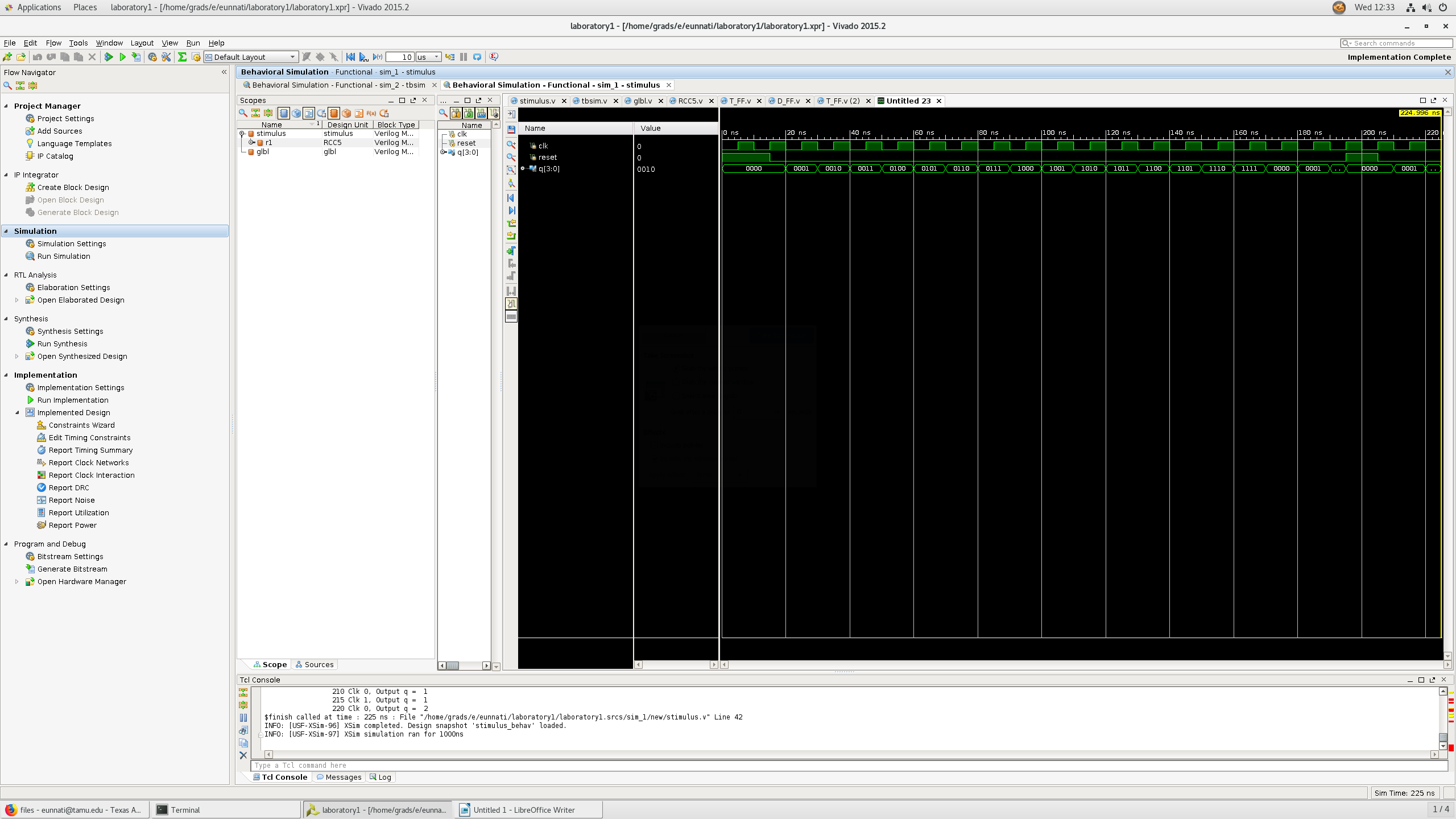
4) Testbench

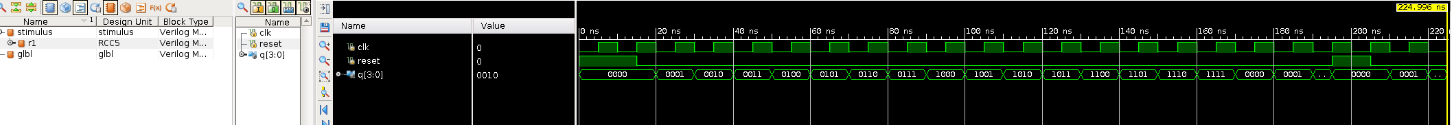
`timescale 1ns / 1ps  
//////////////////////////////////////////////////////////////////////////////////  
// Company:  
// Engineer:  
//  
// Create Date: 09/04/2019 10:35:29 AM  
// Design Name:  
// Module Name: stimulus  
// Project Name:  
// Target Devices:  
// Tool Versions:  
// Description:  
//  
// Dependencies:  
//  
// Revision:  
// Revision 0.01 - File Created  
// Additional Comments:  
//  
//////////////////////////////////////////////////////////////////////////////////

module stimulus;  
reg clk;  
reg reset;  
wire [3:0] q; // instantiate the design block  
RCC r1(q, clk, reset); // control the clock signal that drives the design block. Cycle time = 10  
initial  
clk = 1'b0;  
always  
#5 clk= ~clk; // toggle clk every 5 time units  
 // control the reset signal that drives the design block  
 // reset is asserted from 0 to 20 and from 200 to 220  
initial  
begin  
reset = 1'b1;  
#15 reset = 1'b0;  
#180 reset = 1'b1;  
#10 reset = 1'b0;  
#20 $finish; // terminate the simulation  
end  
// Monitor the outputs  
initial  
$monitor($time, " Clk %b, Output q = %d",clk, q);

endmodule

5) Waveform Generated





6. Simulate a 4 to 1 Multiplexer

a) Add sources for simulating a 4 to 1 Multiplexer.

b) Implement design source code logic 4 to 1 Multiplexer top block module, mux\_4to1\_case module using the case statement.

c)Implement stimulation/testbench code for testing the simulation results of the multiplexer by varying reset, select and clock signal cycle time that drives the design block for short time intervals.

d) Run synthesis and implementation post which run Behavioural stimulation of the code logic and observe results on waveform generator.

e) Monitor the outputs and capture the results obtained.

7. Source Files:

a) \*\*\*\*MULTIPLEXER 4 to1 CASE BLOCK\*\*\*\*

`timescale 1ns / 1ps  
//////////////////////////////////////////////////////////////////////////////////  
// Company:  
// Engineer:  
//  
// Create Date: 09/04/2019 11:08:47 AM  
// Design Name:  
// Module Name: mux\_4to1\_case  
// Project Name:  
// Target Devices:  
// Tool Versions:  
// Description:  
//  
// Dependencies:  
//  
// Revision:  
// Revision 0.01 - File Created  
// Additional Comments:  
//  
//////////////////////////////////////////////////////////////////////////////////

module mux\_4to1\_case ( input  a,                 // 1-bit input called a  
                       input  b,                 // 1-bit input called b  
                       input  c,                 // 1-bit input called c  
                       input  d,                 // 1-bit input called d  
                       input  [1:0] sel,               // input sel used to select between a, b, c, d  
                       output reg out);         // 1-bit output based on input sel

 //The multiplexer selects either of a, b, c, d based on select signal using the case statement and according assigns output to a/ b / c/d,  
  
   always @ (a or b or c or d or sel) begin // This block gets executed when any input value changes  
      case (sel)  
         2'b00 : out <= a;  
         2'b01 : out <= b;  
         2'b10 : out <= c;  
         2'b11 : out <= d;  
      endcase  
   end  
endmodule

8. Testbench

`timescale 1ns / 1ps  
//////////////////////////////////////////////////////////////////////////////////  
// Company:  
// Engineer:  
// Create Date: 09/04/2019 10:48:03 AM  
// Design Name:  
// Module Name: stimulus  
// Project Name:  
// Target Devices:  
// Tool Versions:  
// Description:  
//  
// Dependencies:  
//  
// Revision:  
// Revision 0.01 - File Created  
// Additional Comments:  
//  
//////////////////////////////////////////////////////////////////////////////////

module tbsim;  
   
   // Declare internal reg variables to drive design inputs  
   // Declare wire signals to collect design output  
   // Declare other internal variables used in testbench

   reg  a;  
   reg  b;  
   reg  c;  
   reg  d;  
   wire  out;  
      reg [1:0] sel;  
   integer i;  
   
   // Instantiate one of the designs, in this case, we have used the design with case statement  
   // Connect testbench variables declared above with those in the design

   mux\_4to1\_case  mux0 (   .a (a),  
                           .b (b),  
                           .c (c),  
                           .d (d),  
                           .sel (sel),  
                           .out (out));  
   
   // This initial block is the stimulus

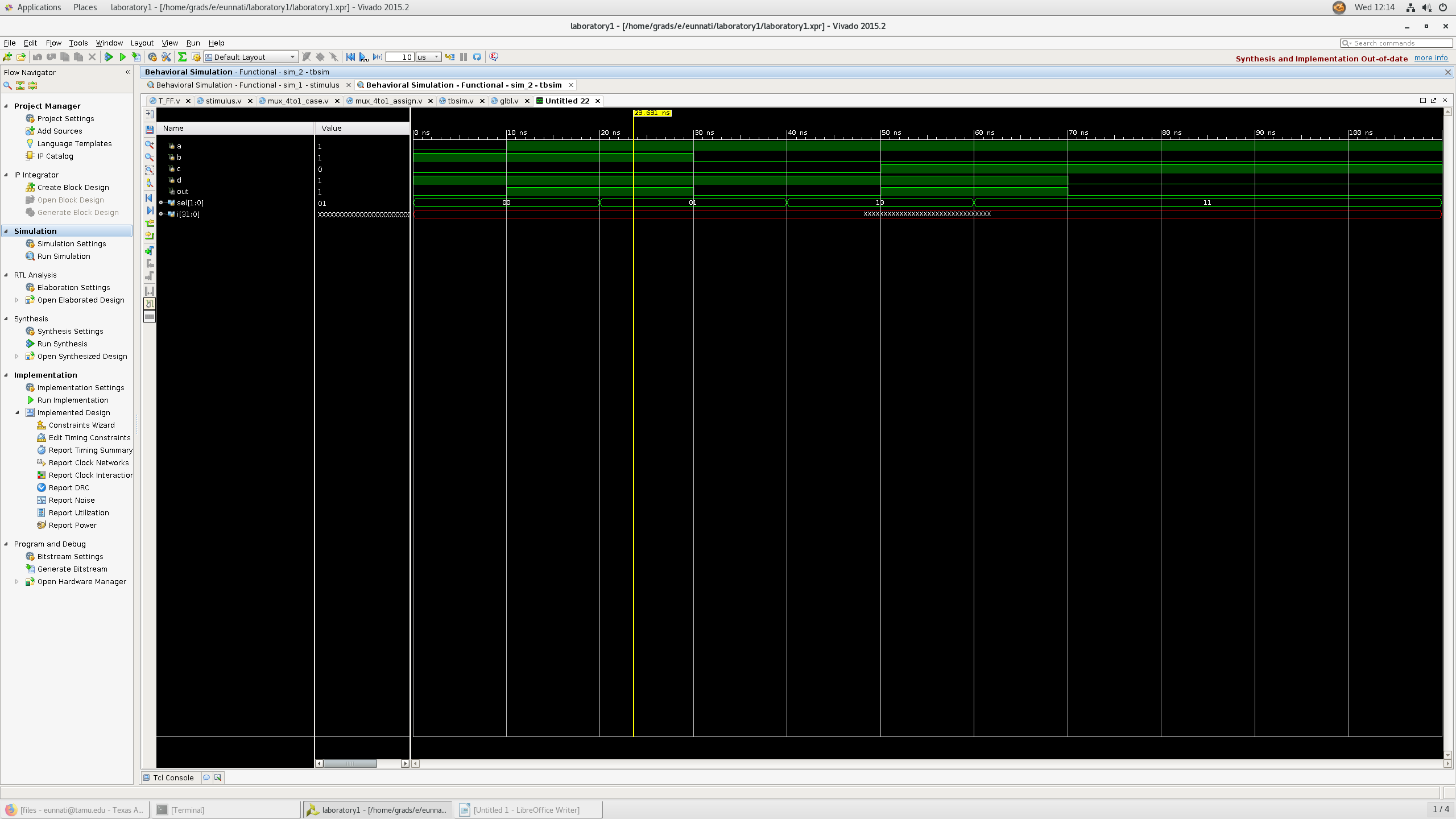
   initial begin

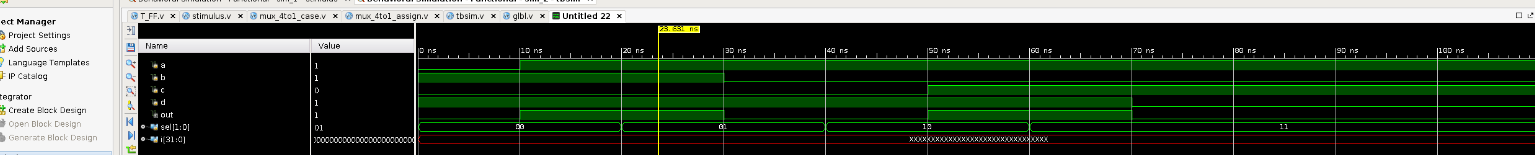
      // Launch a monitor in background to display values to log whenever a/b/c/d/sel/out changes

      $monitor ("[%0t] sel=0x%0h a=0x%0h b=0x%0h c=0x%0h d=0x%0h out=0x%0h", $time, sel, a, b, c, d, out);  
   
       // 1. At time 0, allot random values to a/b/c/d and keep sel = 0  
      sel <= 0;  
      a <= 0;  
      b <= 1;  
      c <= 0;  
      d <= 1;  
       
    
      // 2. Change the value of sel after every 10ns  
       
      #10 a=~a;  
      #10 sel <= 1 ;  
      #10 b=~b;  
      #10 sel <= 2 ;  
      #10 c=~c;  
      #10 sel <= 3;  
      #10 d=~d;  
                       
         
           
             
           
      // 3. After Step2 is over, wait for 40ns and finish simulation

      #40 $finish;  
   end  
endmodule

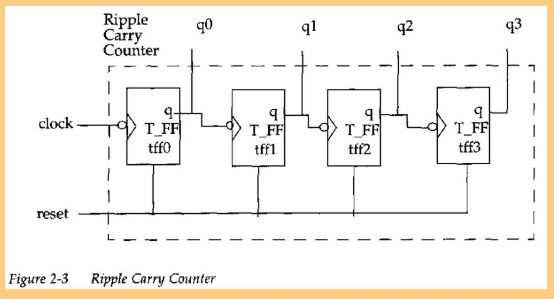
8) WAVEFORM GENERATED



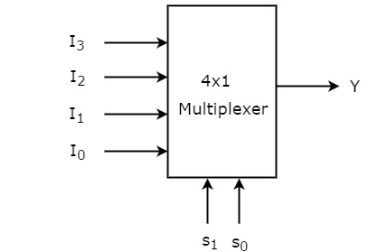


**SCHEMATICS**

1. Ripple Carry Counter



1. 4 to 1 Multiplexer



**QUESTIONS**

1. What does the $monitor statement in the provided test bench do? What about $finish?

* **$monitor** displays the values of its parameters every time any of its parameter changes value. It is used to monitor the changes in the signal list and print them in the format we want. The first argument specifies how the parameters will be displayed and has syntax as:
  + - $monitor ("format\_string", parameter1, parameter2, ...);
* **$finish** finishes a simulation process, exits the simulation process and passes the control back to the host operating system.

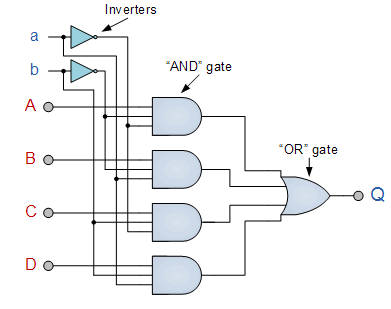
1. Similarly, what do the # symbols signify in the provided test bench? What about $time?

1.The symbol # is used to control time i.e. assign delay to statements. They can be used across statements to define procedural delay, a delayed blocking statement, a delayed non-blocking assignment etc.

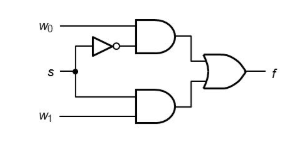
2.When the $time system function is called; it returns the current time as a 64-bit integer value. However, this value is scaled to the `timescale unit.

(c) The 4:1 multiplexer above can be built directly using gates, or it can be constructed using three 2:1 multiplexers, of which are constructed from gates. Provide some intuition as to how the delay through the 4:1 multiplexer would differ for both of the aforementioned cases.

The 4:1 multiplexer designed using gates will have the following schematic:



The 2:1 multiplexer designed using gates will have the following schematic:



A 4:1 multiplexer will have a combination of 7 logic gates.

Since 2:1 Mux will require a combination of 3 Multiplexers to function as a 4:1 Mux, the number of gates implementation will increase. For a single 2:1 mux , it has 6 transistors and 4 logic gates. When designed for a 4:1 mux , the implementation will require 18 transistors , thus increasing the delay for such a combinational logic.