**Laboratory Exercise #2**

**Behavioural, Dataflow, and Structural Verilog**

**Objective**

The second laboratory’s objective is to review various levels of abstraction used to model digital logic in Verilog, while reviewing a few architectural level digital components. An architecture can be written in one of the three basic coding styles: a) Dataflow b) Behavioural c) Structural. A dataflow architecture uses only concurrent signal assignment statements. A behavioural architecture uses only process statements. A structural architecture uses only component instantiation statements. Typically, a design will consist of a mixture of the aforementioned design description levels. Experience developing hardware in Verilog will reveal which level is more appropriate for a given circumstance.

**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 ‘lab2’ and Leave the device properties as default.

2. Simulate a JK Flip Flop using Structural Verilog

a) Add sources for simulating a JK Flip-flop using structural Verilog.

b) Implement design source code logic for JK Flip-flop top block module using logic diagram.

c) Implement stimulation/testbench code for testing the simulation results of the JK Flip-flop 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:

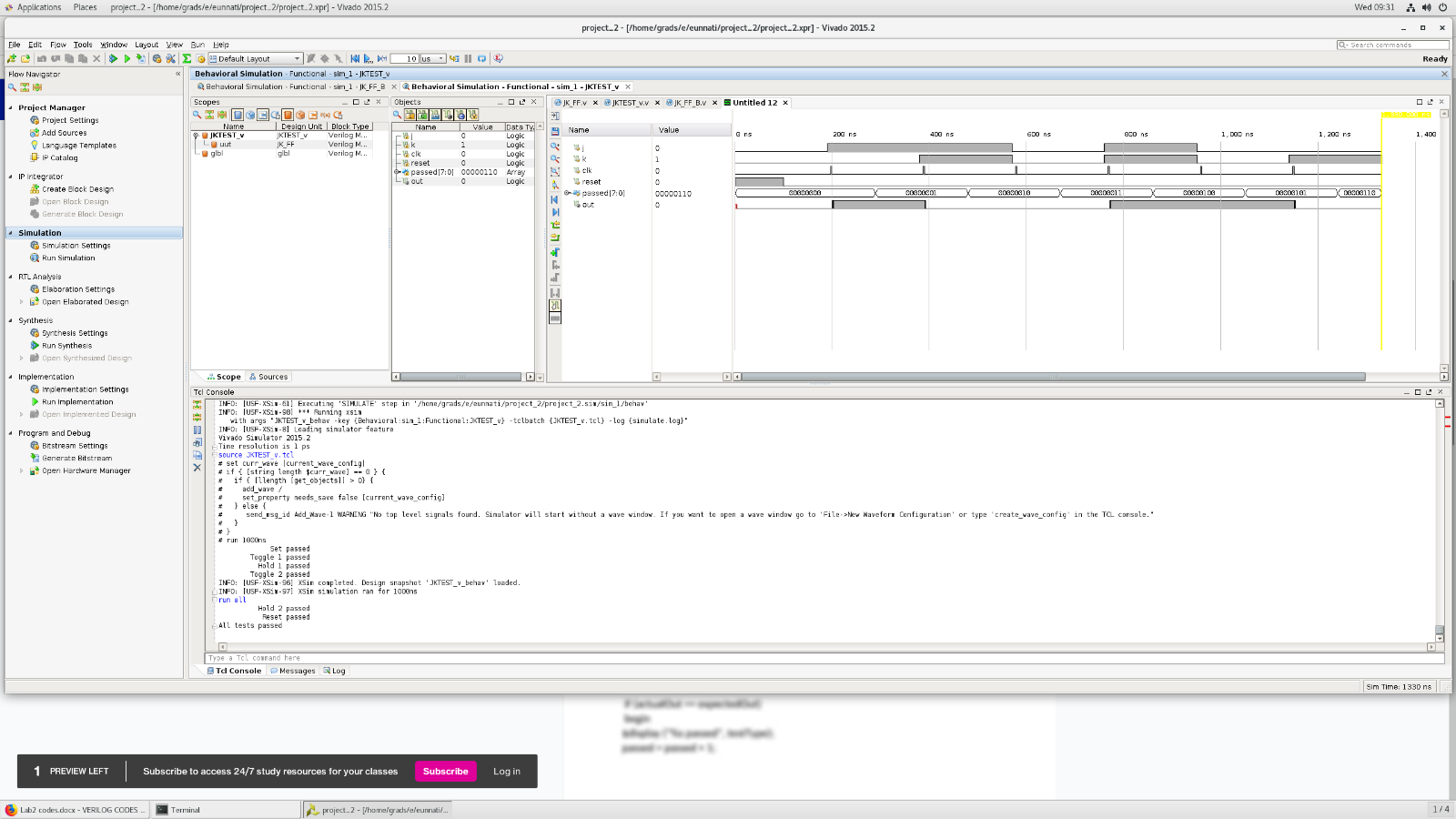
a) \*\*\*\*STRUCTURAL VERILOG FOR JK FLIP FLOP\*\*\*\*

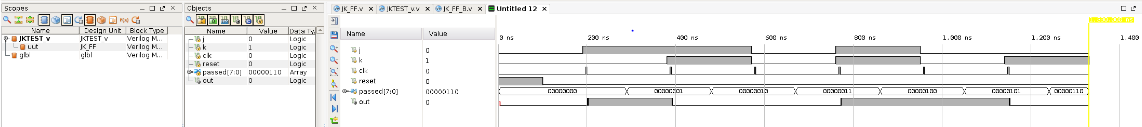
`timescale 1ns / 1ps  
//////////////////////////////////////////////////////////////////////////////////  
// Company:  
// Engineer:  
//  
// Create Date: 09/11/2019 08:07:29 AM  
// Design Name:  
// Module Name: JK\_FF  
// Project Name:  
// Target Devices:  
// Tool Versions:  
// Description:  
//  
// Dependencies:  
//  
// Revision:  
// Revision 0.01 - File Created  
// Additional Comments:  
//  
//////////////////////////////////////////////////////////////////////////////////  
  
module JK\_FF( out, j, k, clk, reset); //Module initialization  
output out; //Defining outputs  
input j, k, clk, reset ; // Defining inputs  
wire a,b,q1,q2,outBar; // Defining internal connections  
//gate level modelling  
//assign q2=~out;  
//assign q1=out;  
nand #2 N1 (a, clk, j, q2); //providing delay of 10ns for nand gate output  
nand #2 N2 (b, clk, k, q1);  
nand #2 N3 (q1, a, q2);  
nand #2 N4 (q2, b, q1, ~reset); //  
assign out=q1;  
// connecting internal connections to output  
assign outBar=q2;  
assign out=~outBar; // Qbar is complement of Q  
endmodule // end of module definition

b) Testbench

`timescale 1ns / 1ps  
//////////////////////////////////////////////////////////////////////////////////  
// Company:  
// Engineer:  
//  
// Create Date: 09/11/2019 08:13:27 AM  
// Design Name:  
// Module Name: JKTEST\_v  
// Project Name:  
// Target Devices:  
// Tool Versions:  
// Description:  
//  
// Dependencies:  
//  
// Revision:  
// Revision 0.01 - File Created  
// Additional Comments:  
//  
//////////////////////////////////////////////////////////////////////////////////  
  
`define STRLEN 15  
module JKTEST\_v;  
  
task passTest;  
input actualOut, expectedOut;  
input [`STRLEN\*8:0] testType;  
  
inout [7:0] passed;  
  
if(actualOut == expectedOut)  
begin  
   $display("%s passed", testType);  
   passed = passed + 1;  
end  
else  
  $display("%s failed: %d should be %d", testType, actualOut, expectedOut);  
endtask  
  
task allPassed;  
input [7:0] passed;  
input [7:0] numTests;  
  
if(passed == numTests)  
  $display("All tests passed");  
else  
  $display("Some tests failed");  
endtask  
  
//Inputs  
reg j;  
reg k;  
reg clk;  
reg reset;  
reg [7:0] passed;  
  
//outputs  
wire out;  
  
//Instantiate the Unit Under Test(UUT)  
JK\_FF uut( .out(out),  
        .j(j),  
        .k(k),  
        .clk(clk),  
        .reset(reset)  
       );  
         
initial begin  
   // Initialize inputs  
   j=0;  
   k=0;  
   clk=0;        
   reset = 1;  
   passed = 0;  
     
   //wait 100ns for global reset to finish  
   #100;  
     
   //Add stimulus here  
   reset = 0;  
     
#90; j=1; k=0; #7; clk = 1;  
#3; clk = 0; #90;  
passTest(out, 1,"Set", passed);  
     
#90; j=1; k=1; #7; clk = 1;  
#3; clk = 0; #90;  
passTest(out, 0,"Toggle 1", passed);  
       
#90; j=0; k=0; #7; clk = 1;  
#3; clk = 0; #90;  
passTest(out, 0,"Hold 1", passed);  
           
#90; j=1; k=1; #7; clk = 1;  
#3; clk = 0; #90;  
passTest(out, 1,"Toggle 2", passed);  
             
#90; j=0; k=0; #7; clk = 1;  
#3; clk = 0; #90;  
passTest(out, 1,"Hold 2", passed);  
   
#90; j=0; k=1; #7; clk = 1;  
#3; clk = 0; #90;  
passTest(out, 0,"Reset", passed);  
               
#90 allPassed(passed,6);  
  
end  
endmodule

5) Waveform Generated





6. Simulate a JK Flip Flop using Behavioural Structure

a) Add sources for simulating a JK Flip-flop using behavioural Verilog.

b) Implement design source code logic for JK Flip-flop top block module by executing statements sequentially.

c) Implement stimulation/testbench code for testing the simulation results of the JK Flip-flop 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.

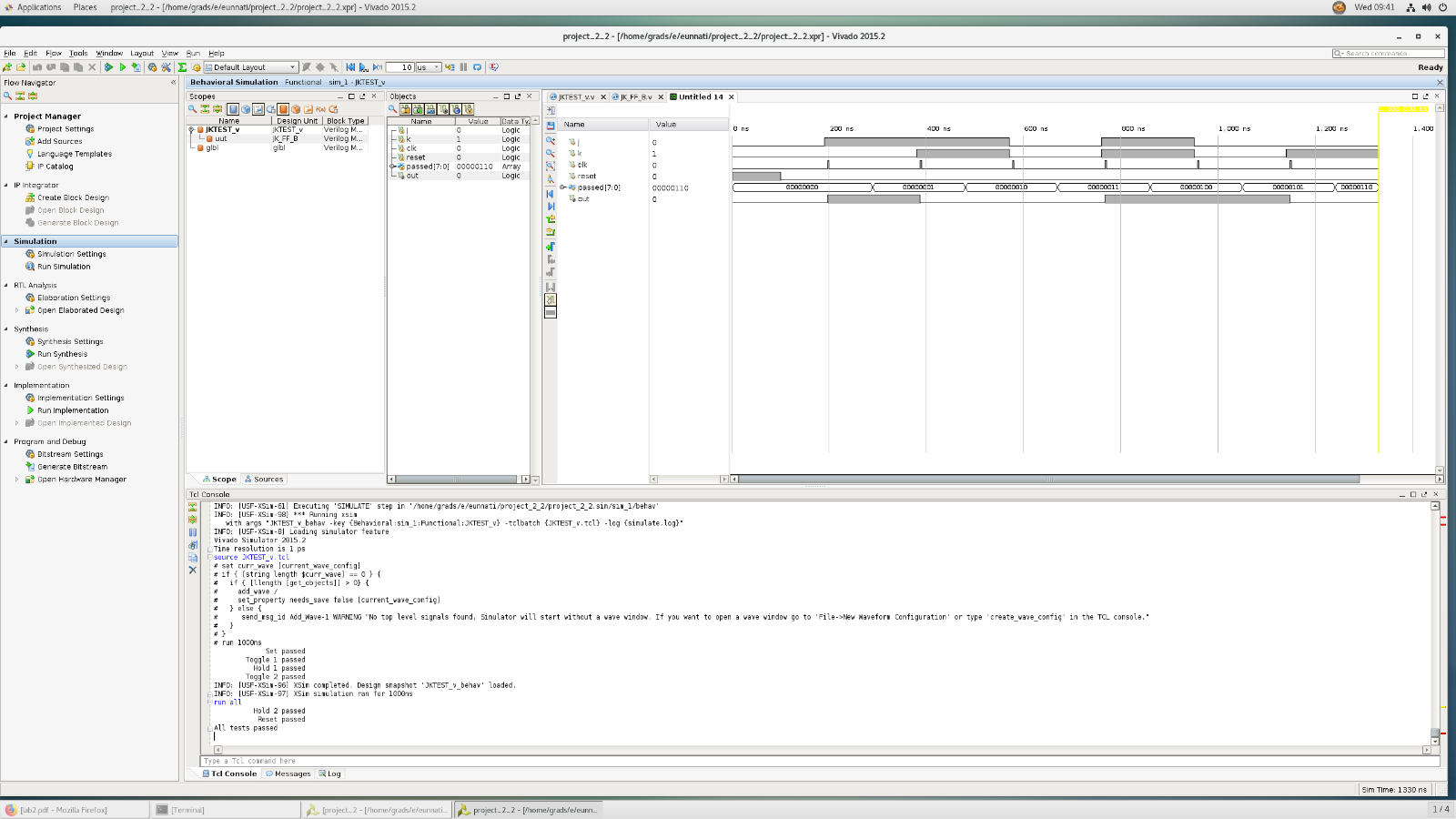
7. Source Files:

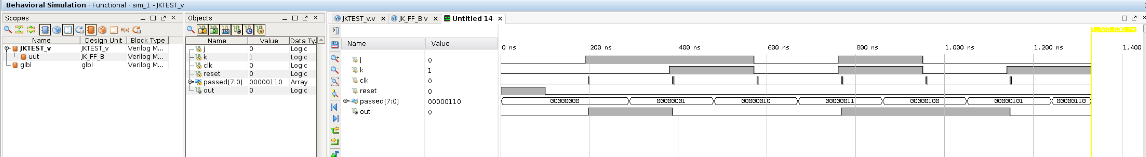
a) \*\*\*\*JK Flip Flop using Behavorial\*\*\*\*

`timescale 1ns / 1ps  
//////////////////////////////////////////////////////////////////////////////////  
// Company:  
// Engineer:  
//  
// Create Date: 09/11/2019 09:18:56 AM  
// Design Name:  
// Module Name: JK\_FF\_B  
// Project Name:  
// Target Devices:  
// Tool Versions:  
// Description:  
//  
// Dependencies:  
//  
// Revision:  
// Revision 0.01 - File Created  
// Additional Comments:  
//  
//////////////////////////////////////////////////////////////////////////////////  
  
module JK\_FF\_B( out,Qbar,j,k,clk,reset); // module definition  
output out, Qbar; // defining outputs  
input j, k, clk, reset ; //defining inputs  
reg out,Qbar; //declaring outputs as reg type  
  
always @ (posedge clk or posedge reset) //output changes if either clock edge is there or reset is high  
begin if (reset == 1) begin // if reset is high then Q+ =0 irrespective of clock  
out <= 1'b0;  
Qbar <= 1'b1;  
end  
  
else if ( j==0 & k==0)  
begin // J=0,K=0 => Q+ =Q  
out <= out;  
Qbar <= Qbar; end  
  
else if ( j==1 & k==0) begin // J=1,K=0 => Q+ =1  
out <= 1'b1;  
Qbar <= 1'b0;  
end  
  
else if ( j==0 & k==1) begin // J=0,K=1 => Q+ =0  
out <= 1'b0;  
Qbar <= 1'b1;  
end  
  
else if ( j==1 & k==1) begin // J=1,K=1 => Q+ =Qbar  
out <= Qbar;  
Qbar <= out;  
end  
end  
endmodule //module definition ends

8. Testbench

`timescale 1ns / 1ps  
//////////////////////////////////////////////////////////////////////////////////  
// Company:  
// Engineer:  
//  
// Create Date: 09/11/2019 08:13:27 AM  
// Design Name:  
// Module Name: JKTEST\_v  
// Project Name:  
// Target Devices:  
// Tool Versions:  
// Description:  
//  
// Dependencies:  
//  
// Revision:  
// Revision 0.01 - File Created  
// Additional Comments:  
//  
//////////////////////////////////////////////////////////////////////////////////  
  
`define STRLEN 15  
module JKTEST\_v;  
  
task passTest;  
input actualOut, expectedOut;  
input [`STRLEN\*8:0] testType;  
  
inout [7:0] passed;  
  
if(actualOut == expectedOut)  
begin  
   $display("%s passed", testType);  
   passed = passed + 1;  
end  
else  
  $display("%s failed: %d should be %d", testType, actualOut, expectedOut);  
endtask  
  
task allPassed;  
input [7:0] passed;  
input [7:0] numTests;  
  
if(passed == numTests)  
  $display("All tests passed");  
else  
  $display("Some tests failed");  
endtask  
  
//Inputs  
reg j;  
reg k;  
reg clk;  
reg reset;  
reg [7:0] passed;  
  
//outputs  
wire out;  
  
//Instantiate the Unit Under Test(UUT)  
JK\_FF\_B uut( .out(out),  
        .j(j),  
        .k(k),  
        .clk(clk),  
        .reset(reset)  
       );  
         
initial begin  
   // Initialize inputs  
   j=0;  
   k=0;  
   clk=0;        
   reset = 1;  
   passed = 0;  
     
   //wait 100ns for global reset to finish  
   #100;  
     
   //Add stimulus here  
   reset = 0;  
     
#90; j=1; k=0; #7; clk = 1;  
#3; clk = 0; #90;  
passTest(out, 1,"Set", passed);  
     
#90; j=1; k=1; #7; clk = 1;  
#3; clk = 0; #90;  
passTest(out, 0,"Toggle 1", passed);  
       
#90; j=0; k=0; #7; clk = 1;  
#3; clk = 0; #90;  
passTest(out, 0,"Hold 1", passed);  
           
#90; j=1; k=1; #7; clk = 1;  
#3; clk = 0; #90;  
passTest(out, 1,"Toggle 2", passed);  
             
#90; j=0; k=0; #7; clk = 1;  
#3; clk = 0; #90;  
passTest(out, 1,"Hold 2", passed);  
   
#90; j=0; k=1; #7; clk = 1;  
#3; clk = 0; #90;  
passTest(out, 0,"Reset", passed);  
               
#90 allPassed(passed,6);  
  
end  
  
endmodule

8) WAVEFORM GENERATED



6. Simulate a D Flip Flop using Behavioural Verilog

a) Add sources for simulating a D Flip-flop using behavioural Verilog.

b) Implement design source code logic for D Flip-flop top block module by executing statements sequentially.

c) Implement stimulation/testbench code for testing the simulation results of the D Flip-flop 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.

7. Source Files:

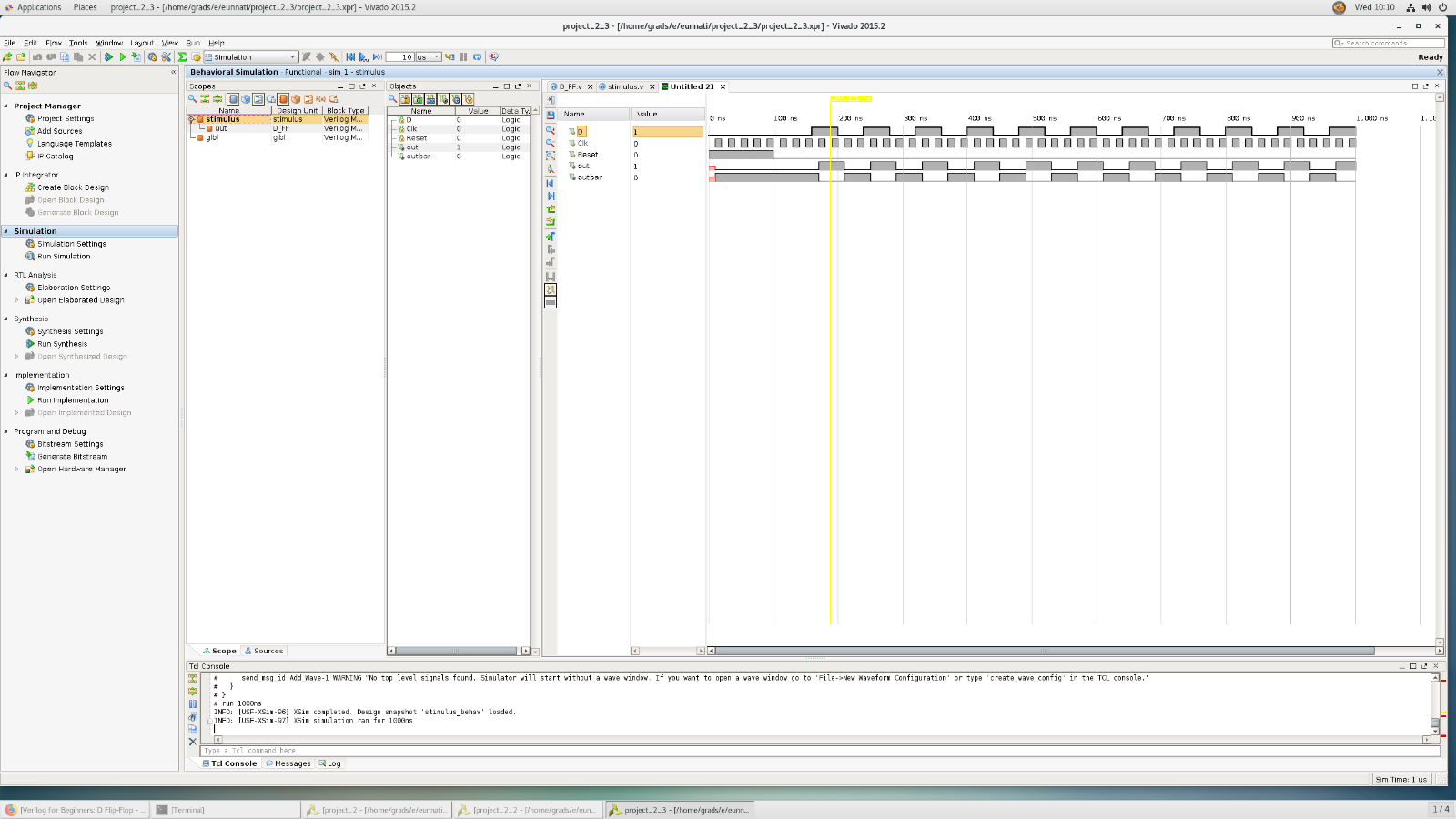
a) \*\*\*\*D Flip Flop \*\*\*\*

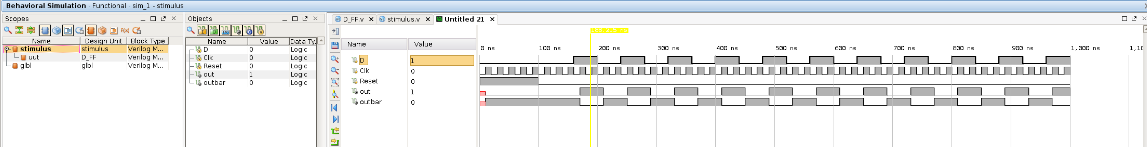
`timescale 1ns / 1ps  
//////////////////////////////////////////////////////////////////////////////////  
// Company:  
// Engineer:  
//  
// Create Date: 09/11/2019 09:44:10 AM  
// Design Name:  
// Module Name: D\_FF  
// Project Name:  
// Target Devices:  
// Tool Versions:  
// Description:  
//  
// Dependencies:  
//  
// Revision:  
// Revision 0.01 - File Created  
// Additional Comments:  
//  
//////////////////////////////////////////////////////////////////////////////////  
    module D\_FF( out,  
  
      outbar,  
  
      D,  
  
      Clk,  
  
      Reset  
  
        );  
  
    output  reg  out;  
  
    output   outbar;  
  
    input   D,  
  
      Clk,  
  
      Reset;  
  
    assign outbar = ~out;  
  
    always @(posedge Clk)  
  
    begin  
  
     if (Reset == 1'b1) //If not at reset  
  
      out = 1'b0;  
  
     else  
  
      out = D;  
  
    end  
  
    endmodule

8. Testbench

`timescale 1ns / 1ps  
//////////////////////////////////////////////////////////////////////////////////  
// Company:  
// Engineer:  
//  
// Create Date: 09/11/2019 09:45:11 AM  
// Design Name:  
// Module Name: stimulus  
// Project Name:  
// Target Devices:  
// Tool Versions:  
// Description:  
//  
// Dependencies:  
//  
// Revision:  
// Revision 0.01 - File Created  
// Additional Comments:  
//  
//////////////////////////////////////////////////////////////////////////////////  
  
module stimulus;  
  
 // Inputs  
  
 reg D;  
  
 reg Clk;  
  
 reg Reset;  
  
 // Outputs  
  
 wire out;  
  
 wire outbar;  
  
 // Instantiate the Unit Under Test (UUT)  
  
 D\_FF uut (  
  
  .out(out),  
  
  .outbar(outbar),  
  
  .D(D),  
  
  .Clk(Clk),  
  
  .Reset(Reset)  
  
  );  
  
 initial begin  
  
  // Initialize Inputs  
  
  D  = 1'b0;  
  
  Clk  = 1'b0;  
  
  Reset   = 1'b1;  
  
  // Wait 100 ns for global reset to finish  
  
  #100;  
  
  // Add stimulus here  
  
  Reset = 1'b0;  
  
  #20;  
  
  forever #40 D = ~ D;  
  
 end  
  
   always #10 Clk = ~Clk;    
  
endmodule

8) WAVEFORM GENERATED





6. Simulate a 2-to-4 Decoder using Dataflow Verilog

a) Add sources for simulating a 2-to-4 Decoder using Dataflow Verilog.

b) Implement design source code logic for 2-to-4 Decoder top block module with an enable signal by assigning the outputs directly having the inputs inside of expressions that give us the output values directly.ie based on Boolean function.

c) Implement stimulation/testbench code for testing the simulation results of the 2-to-4 Decoder 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.

7. Source Files:

a) \*\*\*\*2To4 Decoder \*\*\*\*

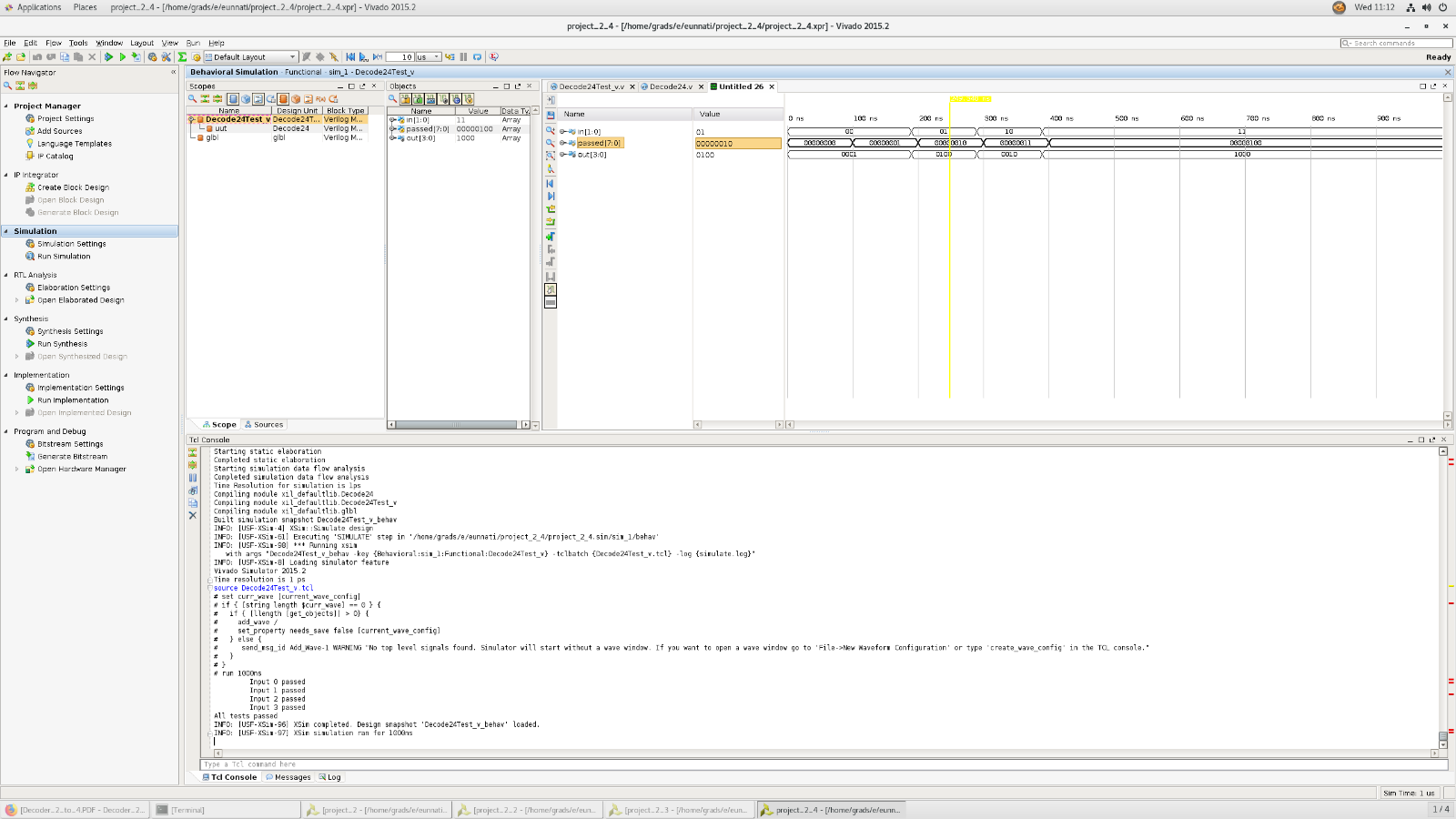
`timescale 1ns / 1ps  
//////////////////////////////////////////////////////////////////////////////////  
// Company:  
// Engineer:  
//  
// Create Date: 09/11/2019 10:38:27 AM  
// Design Name:  
// Module Name: decoder2to4  
// Project Name:  
// Target Devices:  
// Tool Versions:  
// Description:  
//  
// Dependencies:  
//  
// Revision:  
// Revision 0.01 - File Created  
// Additional Comments:  
//  
//////////////////////////////////////////////////////////////////////////////////  
  
  
module Decode24(  
in,  
out  
);  
  
input [1:0] in;  
output [3:0] out;  
  
//reg out;  
wire en;  
  
assign en = 1;  
  assign out[0] = (~in[0] & ~in[1]);  
  assign out[1] = (~in[0] & in[1]);  
  assign out[2] = (in[0] & ~in[1]);  
  assign out[3] = (in[0] & in[1]);

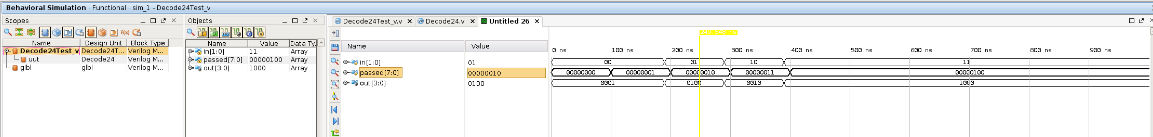
endmodule

8. Testbench

`timescale 1ns / 1ps  
//////////////////////////////////////////////////////////////////////////////////  
// Company:  
// Engineer:  
//  
// Create Date: 09/11/2019 10:18:06 AM  
// Design Name:  
// Module Name: Decode24Test\_v  
// Project Name:  
// Target Devices:  
// Tool Versions:  
// Description:  
//  
// Dependencies:  
//  
// Revision:  
// Revision 0.01 - File Created  
// Additional Comments:  
//  
//////////////////////////////////////////////////////////////////////////////////  
  
`define STRLEN 15  
module Decode24Test\_v;  
  
task passTest;  
input actualOut, expectedOut;  
input [`STRLEN\*8:0] testType;  
  
inout [7:0] passed;  
  
if(actualOut == expectedOut)  
begin  
   $display("%s passed", testType);  
   passed = passed + 1;  
end  
else  
  $display("%s failed: %d should be %d", testType, actualOut, expectedOut);  
endtask  
  
task allPassed;  
input [7:0] passed;  
input [7:0] numTests;  
  
if(passed == numTests)  
  $display("All tests passed");  
else  
  $display("Some tests failed");  
endtask  
  
//Inputs  
reg [1:0] in;  
reg [7:0] passed;  
  
//outputs  
wire [3:0] out;  
  
//Instantiate the Unit Under Test(UUT)  
Decode24 uut( .in(in),  
          .out(out)  
       );  
         
initial begin  
   // Initialize inputs  
   in = 0;  
   passed = 0;  
       
   //Add stimulus here  
     
#90; in = 0; #10  
passTest(out, 1,"Input 0", passed);  
     
#90; in = 1; #10  
passTest(out, 2,"Input 1", passed);  
       
#90; in = 2; #10  
passTest(out, 4,"Input 2", passed);  
           
#90; in = 3; #10  
passTest(out, 8,"Input 3", passed);  
               
allPassed(passed,4);  
  
end  
  
endmodule

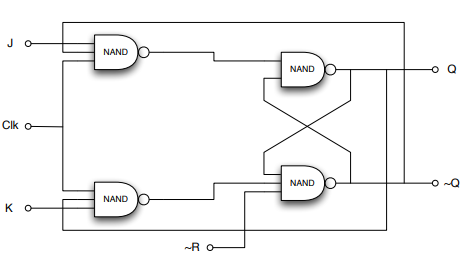
8) WAVEFORM GENERATED



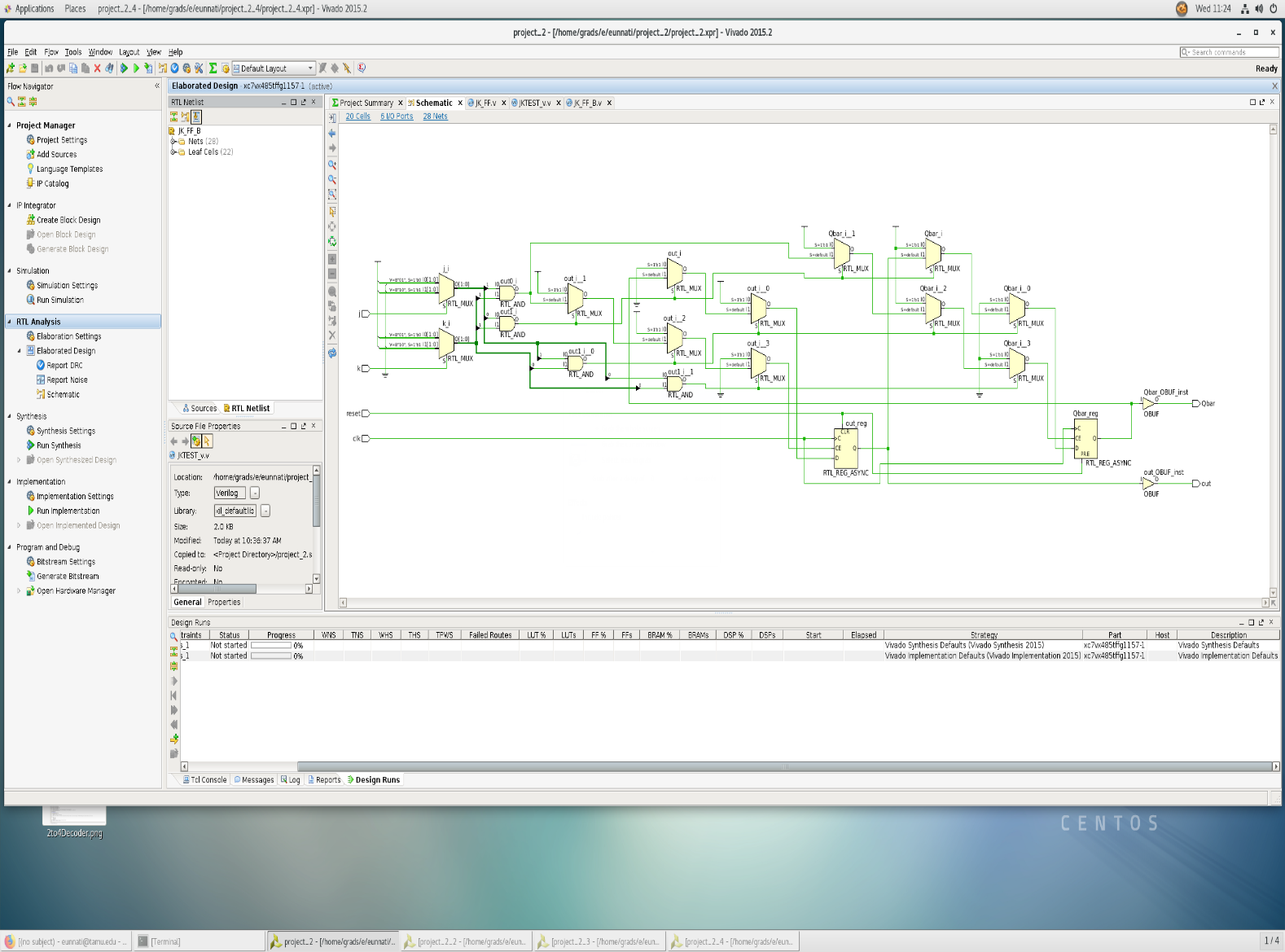


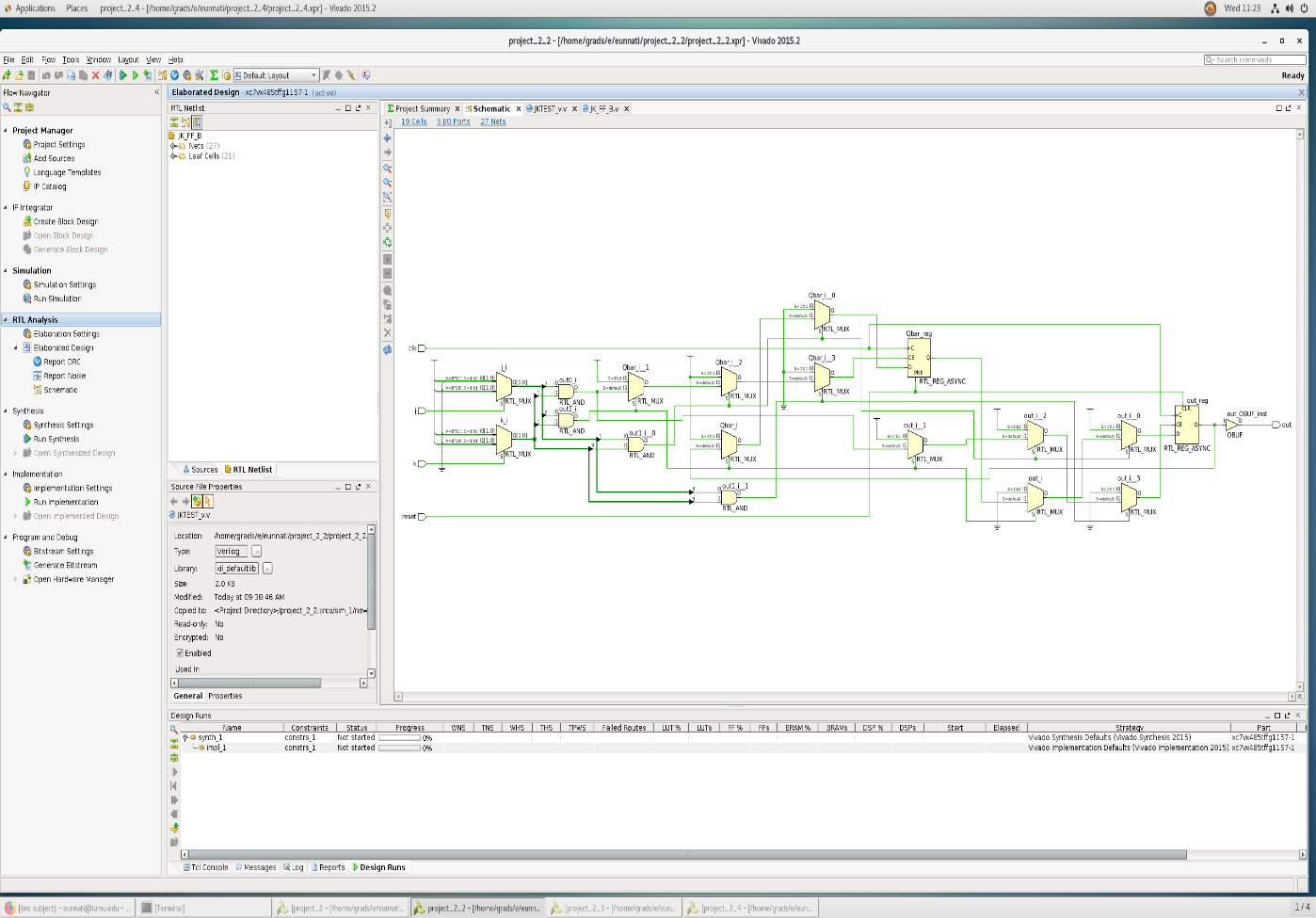
**SCHEMATICS**

1. JK Flip Flop

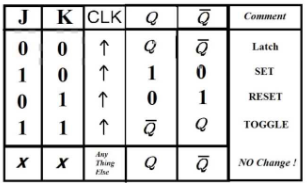


A JK Flip-Flop constructed with NAND gates

Schematic of JK Flip Flop following Structural Verilog

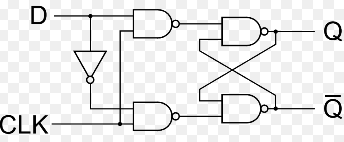


Schematic of JK Flip Flop following Behavioural Verilog

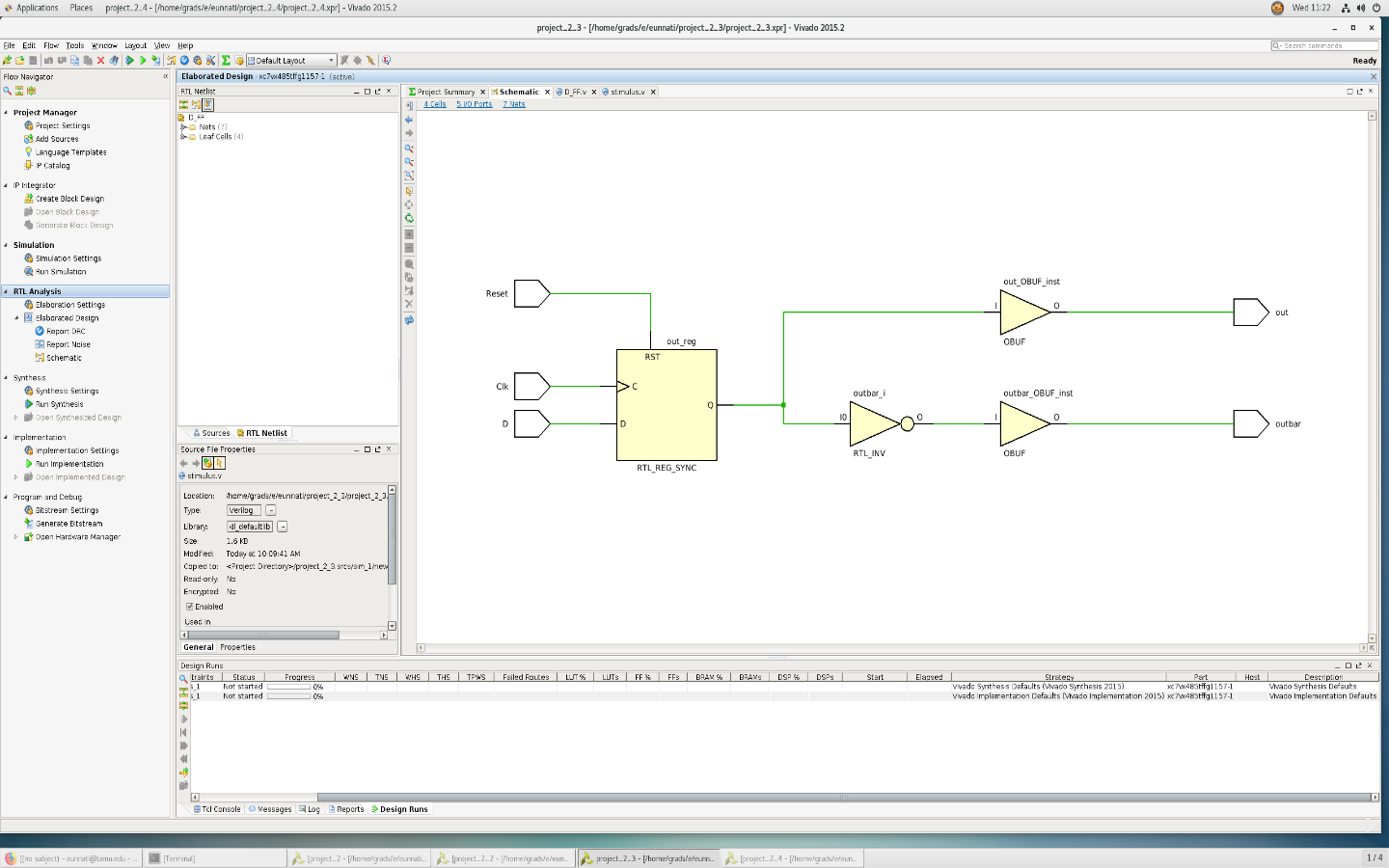


Truth Table for JK Flip Flop

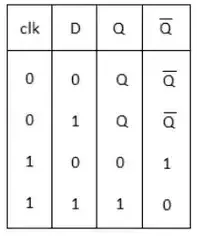
1. D Flip Flop



A D Flip-Flop constructed with NAND gates

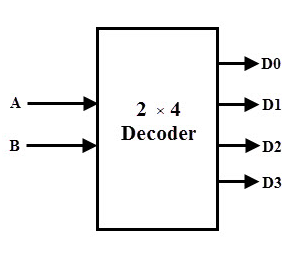


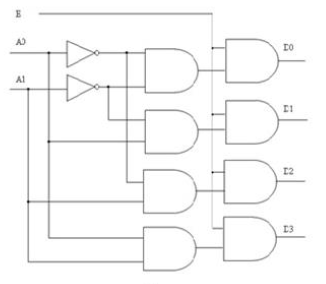
Schematic of D Flip Flop following Behavioural Verilog



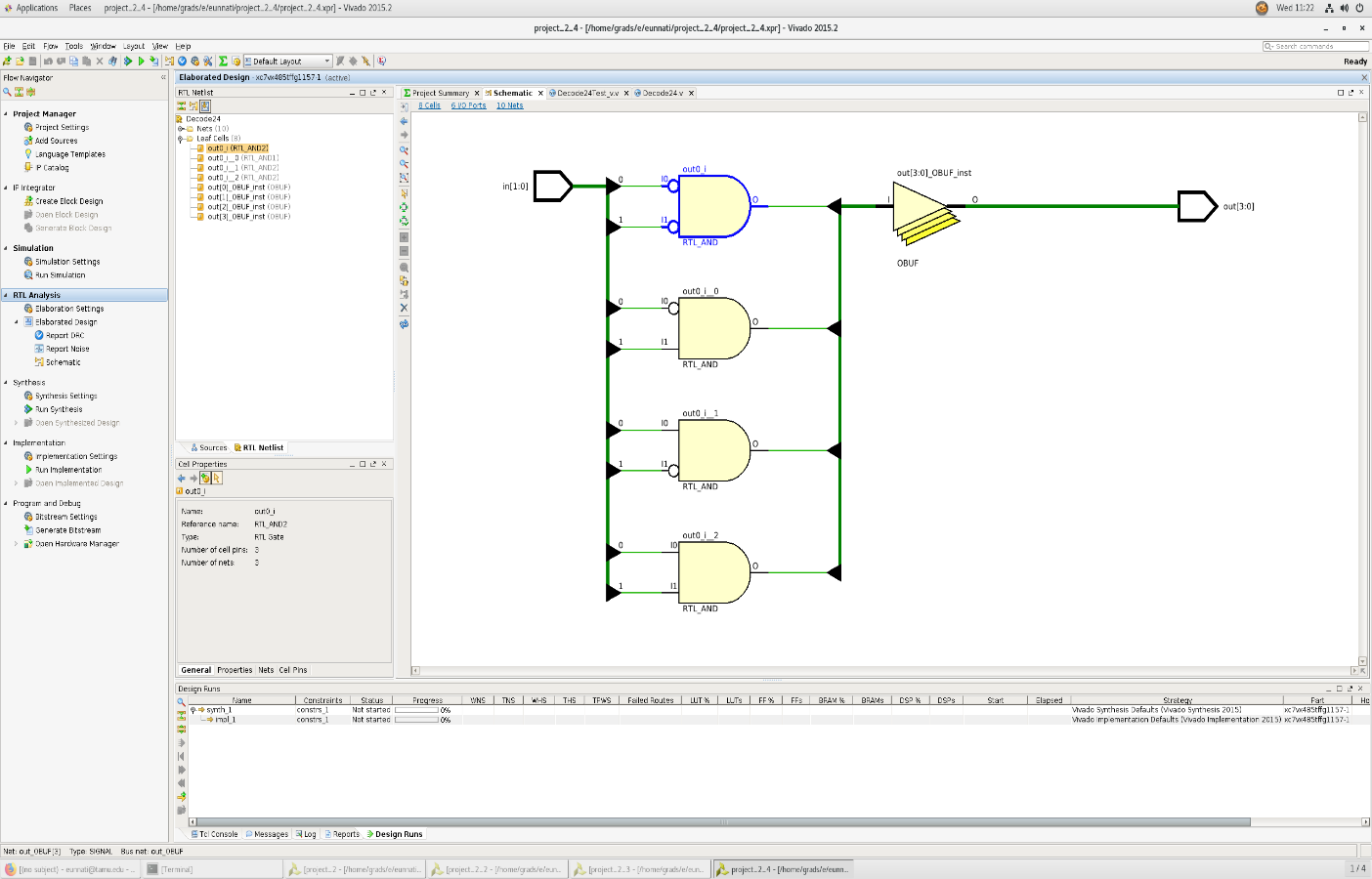
Truth Table for D Flip Flop

1. 2-to-4 Decoder

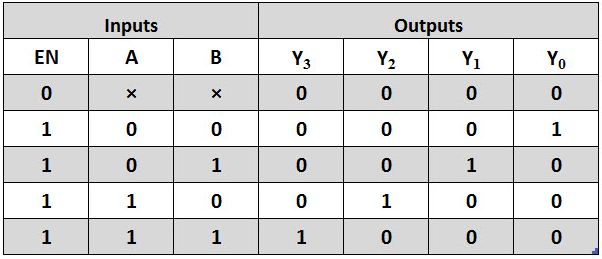




A 2-to-4 Decoder constructed with AND gates



Schematic of 2-to-4 Decoder following Dataflow Verilog



Truth Table for 2-to-4 Decoder

**QUESTIONS**

(a)In behavioral Verilog, two types of assignment statements exists. What are they, and when would you use one over the other?

When an assign statement is used to assign the given net with some value, it is called explicit assignment. Verilog also allows an assignment to be done when the net is declared and is called implicit assignment.

Eg : wire [1:0] a;

Assign a = x & y; \\ Explicit

Wire [1:0] a = x & y;

(b)Compare and contrast the structural verses behavioral implementation of the JK flip-flop. Which level of abstraction might you use for a processor design and why?

Verilog supports abstract behavioural modeling, so can be used to model the functionality of a system at a high level of abstraction. This is useful at the system analysis and partitioning stage.

However, customarily, **structural** refers to describing a design using module instances (especially for the lower-level building blocks such as AND gates and flip-flops), whereas **behavioral** refers to describing a design using always blocks.

Gate netlists are always **structural**, and RTL code is typically **behavioral**. It is common for RTL to have instances of clock gates and synchronizer cells.