Lab 6:Introduction to Behavioral Verilog and Logic Synthesis

ECEN 248 - 505

TA: Younggyun Cho

Date: October 13, 2020

Objectives

The main goal of this lab is to implement the behavioral modeling techniques in Verilog to be able to test higher level logic gates. This lab also introduces the process of logic synthesis through programming, and implementing it on Zybo.

Design

Experiment 1 - The first experiment in this lab is an introduction to the hardware behaviour in Verilog. This behavioral process is used to run simulations for a four-bit 2:1 MUX and a four-bit 4:1 MUX.

TwoOne Mux

```
//UIN 528000252
`timescale 1ns/1ps
`default_nettype none
module two_one_mux(Y, A, B, S);
//declare output and input ports
        output reg Y;
        input wire A, B, S; //A and B are 1-bit wide wires
        //block triggered whenever A, B, or S changes
        always@(A or B or S)
        begin //block constructs together
        if(S==1'b0) //one bit binary value of zero
        Y=A;    //if S=0 then Y=A
        else            //any other value
        Y=B;    //if S=1 then Y=B
        end
endmodule
```

FourBit Mux

```
`timescale 1 ns/ 1 ps
`default_nettype none

module four_bit_mux(Y, A, B ,S);
//four bit 2:1 mux
        output reg [3:0] Y;         //output is 4-bit wide reg
        input wire [3:0] A, B;    //inputs for A and B are 4-bit wires
        input wire S;               //select is 1-bit wide

        always@(A or B or S)
        begin
        if(S==1'b0)       //zeo is one bit
        Y=A;              //if S=0 then Y=A
        else              //any other value
        Y=B;           //if S=1 then Y=B
        end
```

```
endmodule
```

FourBit 4to1 Mux

```
`timescale 1 ns/ 1 ps
`default_nettype none

module mux_4bit_4to1(Y, A, B, C, D, S);

        output reg [3:0] Y; //four-bit wide output register
        input wire [3:0] A, B, C, D;//four-bit wide input wire A,B,C,D
        input wire [1:0] S;//two-bit wide switch S

        always@(*)
        case (S)
        2'b00: Y = A;//if S = 2'b00, the Y = A
        2'b01: Y = B;//if S = 2'b01, the Y = B
        2'b10: Y = C;//if S = 2'b10, the Y = C
        2'b11: Y = D;//if S = 2'b11, the Y = D
        endcase //end case statement
endmodule
```

Experiment 2- For experiment 2 in this lab, a 2:4 decoder, a 4:2 encoder, and a priority encoder are coded in Verilog to run a simulation.

TwoFour Decoder

```
`timescale 1ns / 1ps
`timescale 1ns / 1ps
`default_nettype none
/*This module describes a 2:4 decoder using behavioral constructs in Verilog HDL. */

/*module interface for the the 2:4 decoder*/
module two_four_decoder(
  input wire [1:0] W,
  input wire En,
  output reg [3:0] Y //Y should be a 4 bit output of type reg
);
  always@(En or W) // trigger when E or W changes
    begin //not necessary because if is single clause but looks better
      if(En == 1'b1) //can put case within if clause!
        case(W) // selection based on W
          2'b00: Y = 4'b0001; //4'b signifies a 4 bit binary values
          2'b01: Y = 4'b0010; //light up y[1]
```

```
        2'b10: Y = 4'b0100; //light up y[2]
        2'b11: Y = 4'b1000; //light up y[3]
      endcase // designates the end of a case statement
    else // if not Enable
      Y = 4'b0000; // disable all outputs;
  end
endmodule
```

FourTwo Encoder

```
`timescale 1ns/1ps
`default_nettype none
//this module describes a 4:2 encoder
module four_two_encoder(
        input wire [3:0] W,//W is input wire of 4-bits
        output wire zero,//zero is output wire of 1-bit
        output reg [2:0] Y //Y is output wire of type reg
);
        assign zero = (W == 4'b0000);//zero is defined as four-bit 0000
        always@(W)//triggerd when W changes
        begin//start case statement
        case(W)//selection based on W
        4'b0001: Y = 2'b00;//W[0] lights up
        4'b0010: Y = 2'b01;//W[1] lights up
        4'b0100: Y = 2'b10;//W[2] lights up
        4'b1000: Y = 2'b11;//W[3] lights up
        default Y = 2'bXX;//covers cases not listed
        endcase//end of case statement
        end
endmodule
```

Priority Encoder

```
//UIN 528000252
`timescale 1ns / 1ps
`timescale 1ns / 1ps
`default_nettype none
/*This module describes a 2:4 decoder using behavioral constructs */

/*module interface for the 4:2 encoder*/
module priority_encoder(
  input wire [3:0] W,
  output wire zero,
  output reg [1:0] Y
);
```

```
  /*can mix levels of abstraction!*/
  assign zero = (W == 4'b0000); //a zero test! notice the use of == rather than =

  /*behavioral portion*/
  always@(W) //trigger when W changes
    begin // not necessary because case is single clause but looks better
casex(W) //select based on W
       4'b0001: Y = 2'b00; //2'b signifies a 2 bit binary value
       4'b001X: Y = 2'b01; //w[1] is lit up
       4'b01XX: Y = 2'b10; //w[2] is lit up
       4'b1XXX: Y = 2'b11; //w[3] is lit up
       default: Y = 2'bXX; //default covers cases not listed!
     endcase // designates the end of a case statement
   end
endmodule //ends the module
```

Experiment 3 -For the third experiment, the designs used for the 4:2 encoder, the 2:4 decoder, and the priority encoder are displayed on the zybo board. The student first had to implement the data by "Generating Bitstream", then program the board and run the test cases.

Result

The results for this lab are based on accomplishing the learning objectives over having quantifiable results. The behavioral verilog code ran all the right simulations along with implementing the data on the zybo correctly. The logic units were 2:4 decoder, 4:2 encoder, and the priority encoder. These hardware pieces were implemented and run on the FPGA board, and it all corresponds to each gate through respective xdc files.

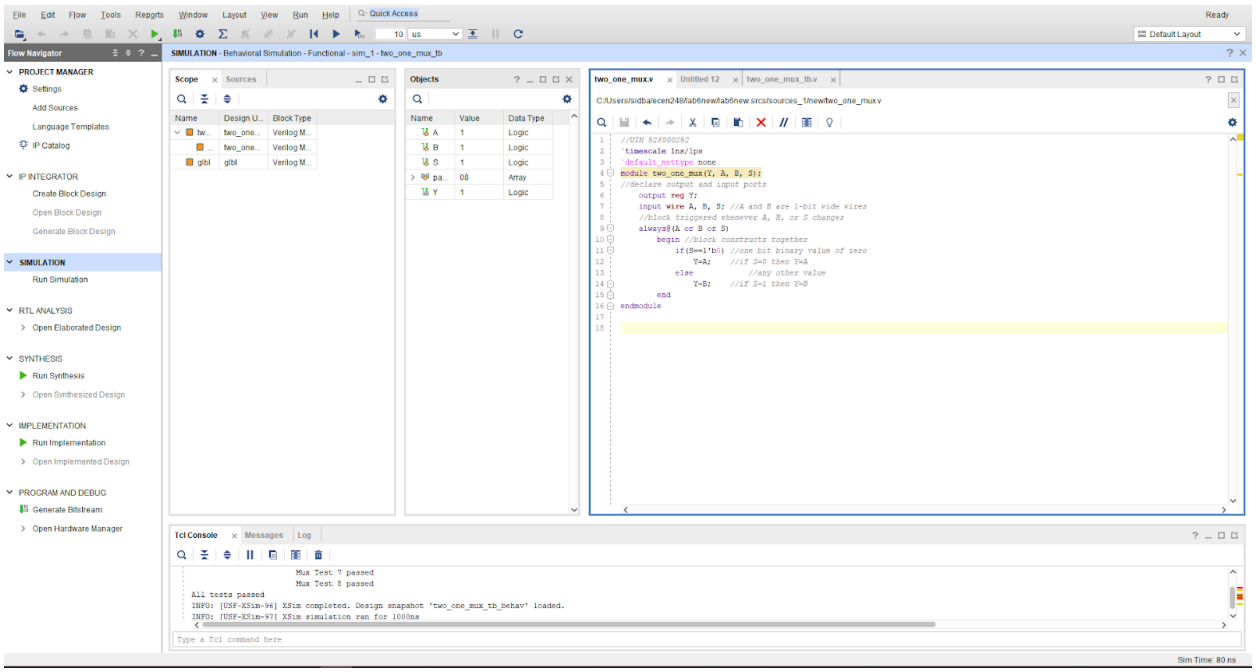
Conclusion

This lab effectively introduced the students of behavioral techniques in Verilog, along with the logic synthesis that was performed through the FPGA board. While writing the code, and performing this lab it was especially helpful to have the gate-schematics. This helped in understanding the logic flows in each circuit. This also introduced the basic parts to the zybo and how to power and set one up.

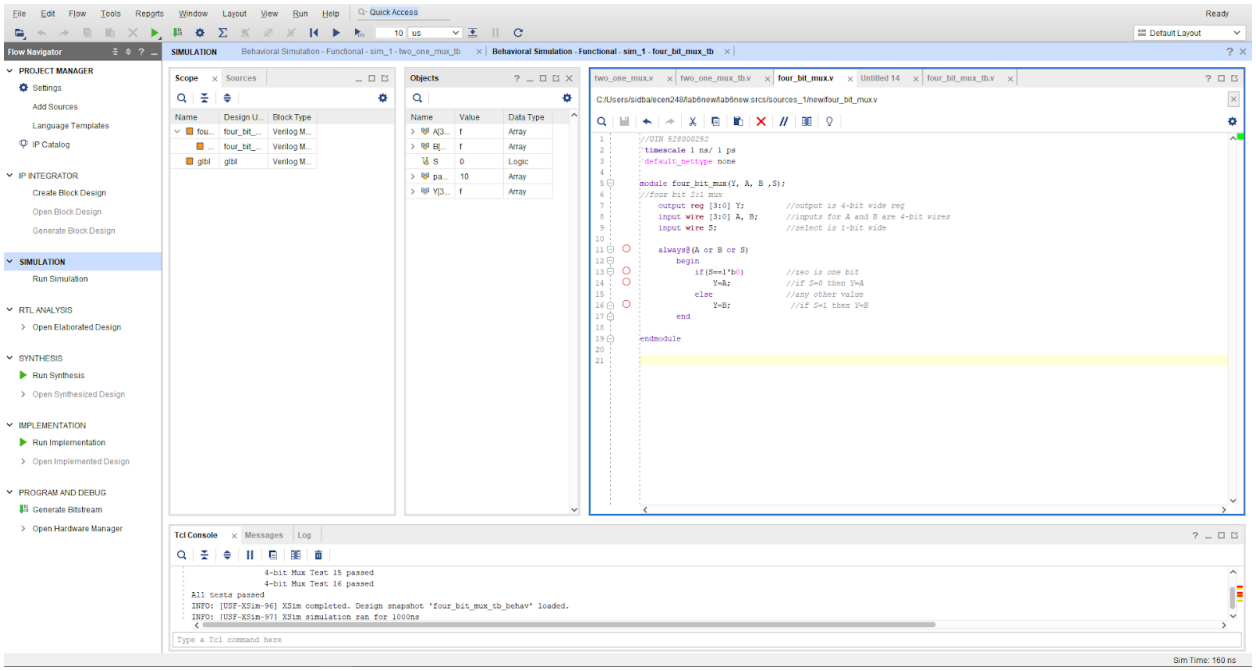
Post - Lab Questions

1. Include the source code with comments for all modules you simulated. You do not have to include test bench code. Code without comments will not be accepted!

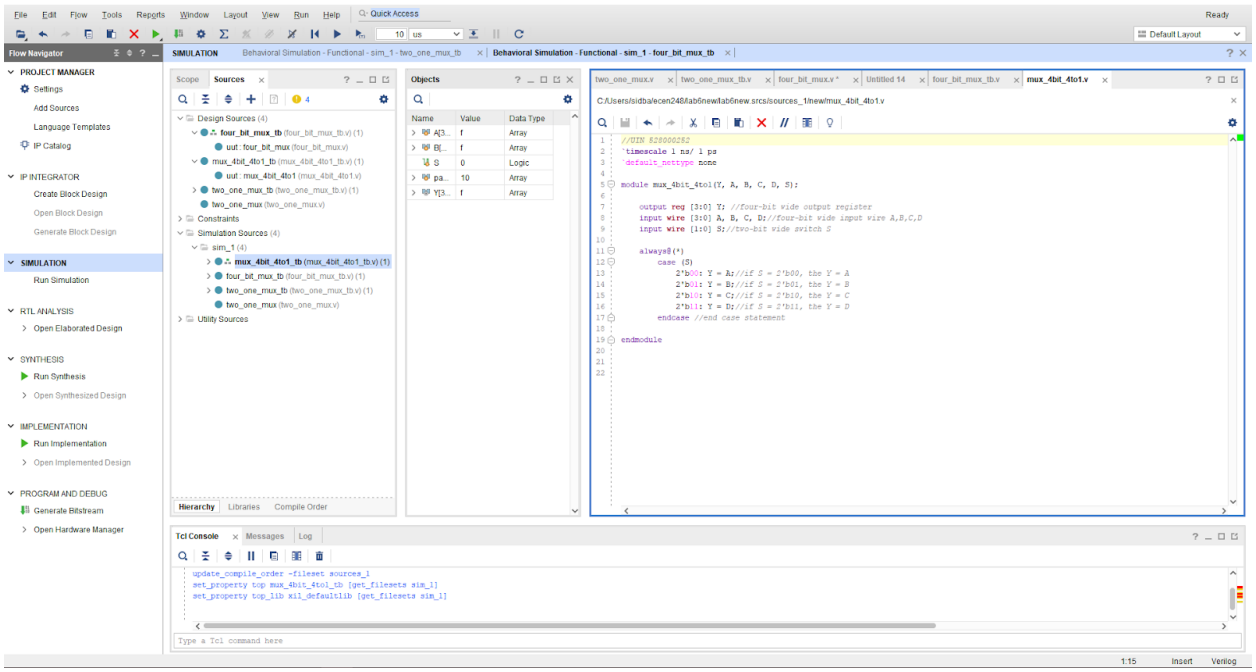
TwoOne Mux



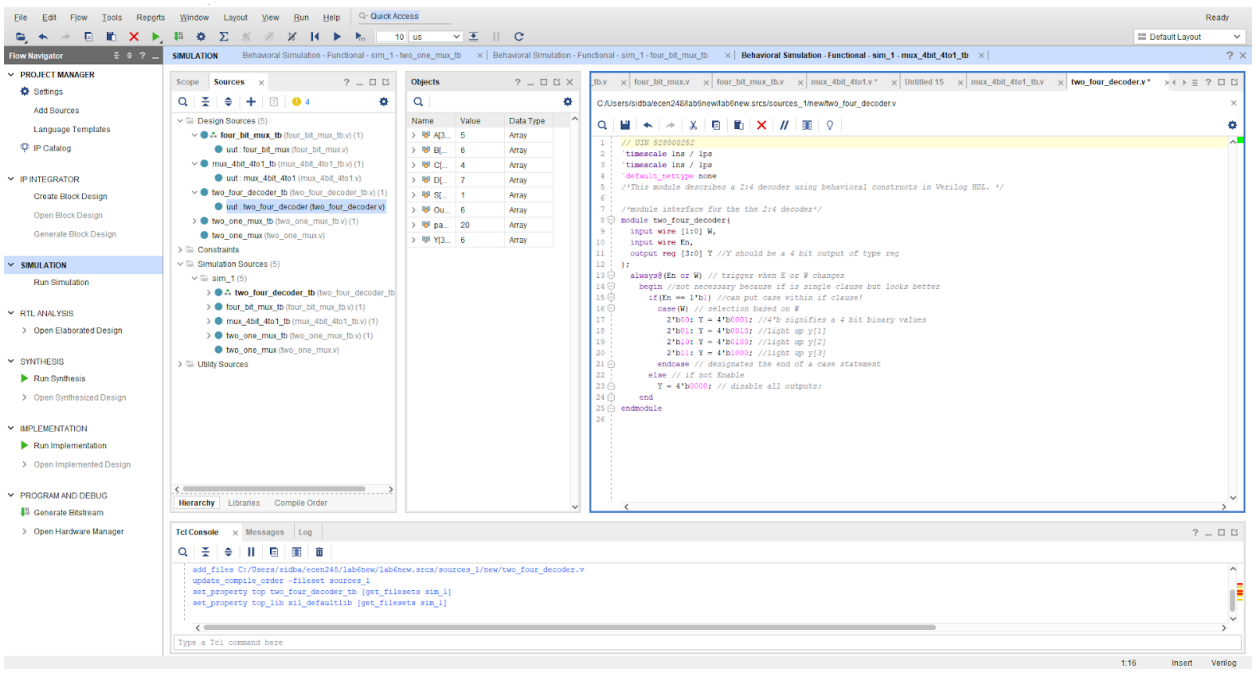
FourBit Mux



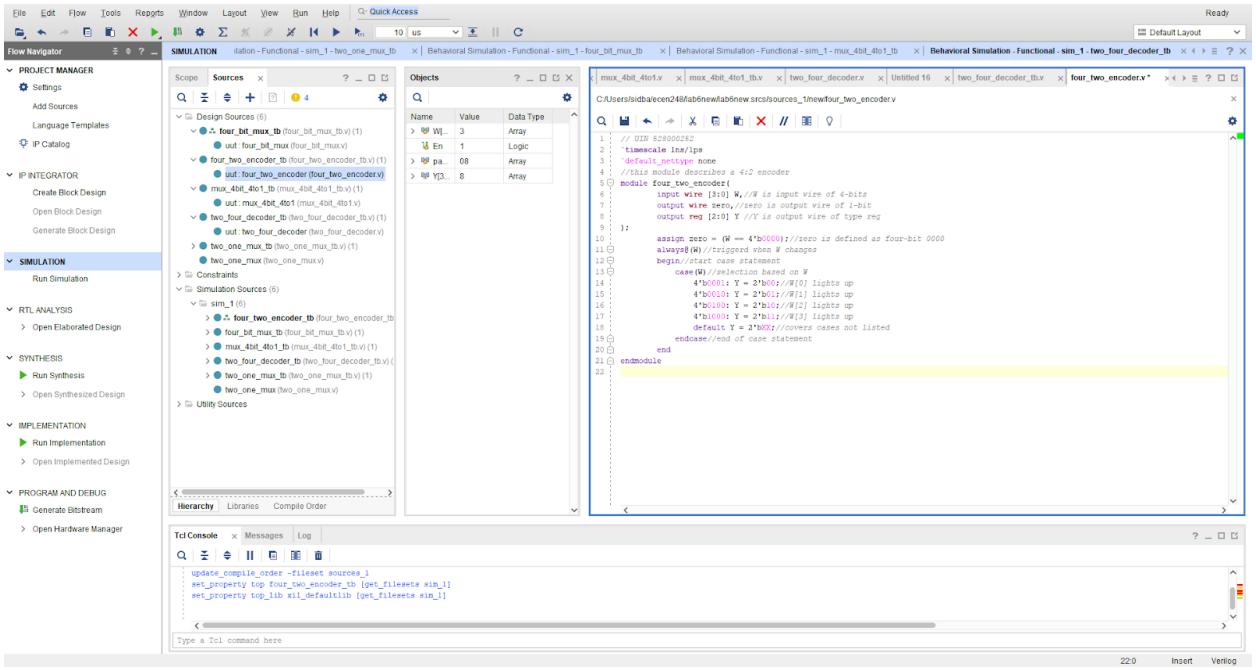
FourBit 4to1 Mux



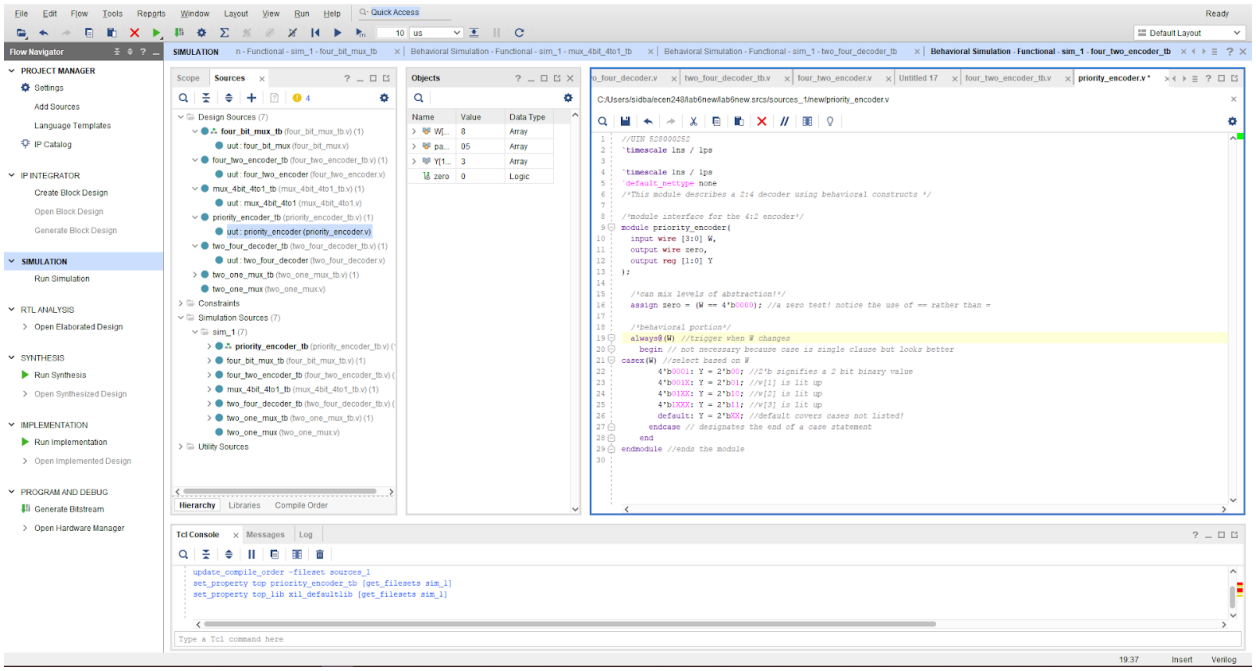
TwoFour Decoder



FourTwo Encoder

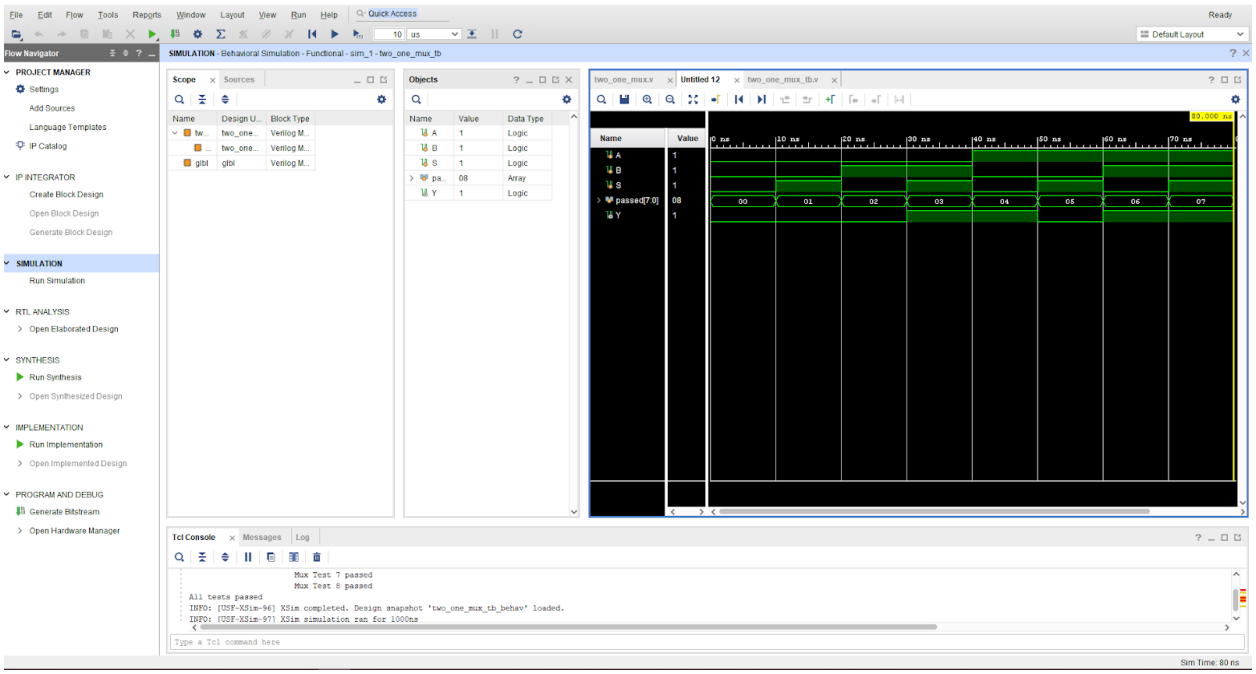


Priority Encoder

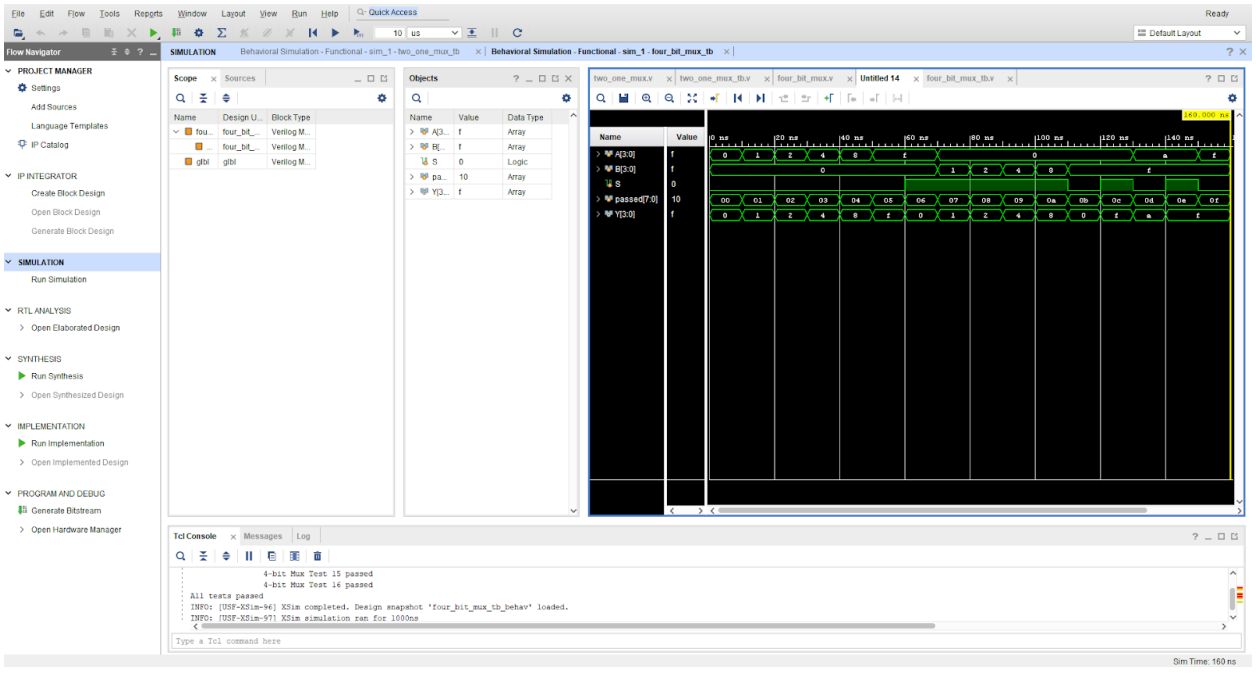


2. Include screenshots of all waveforms captured during simulation in addition to the test bench console output for each test bench simulation.

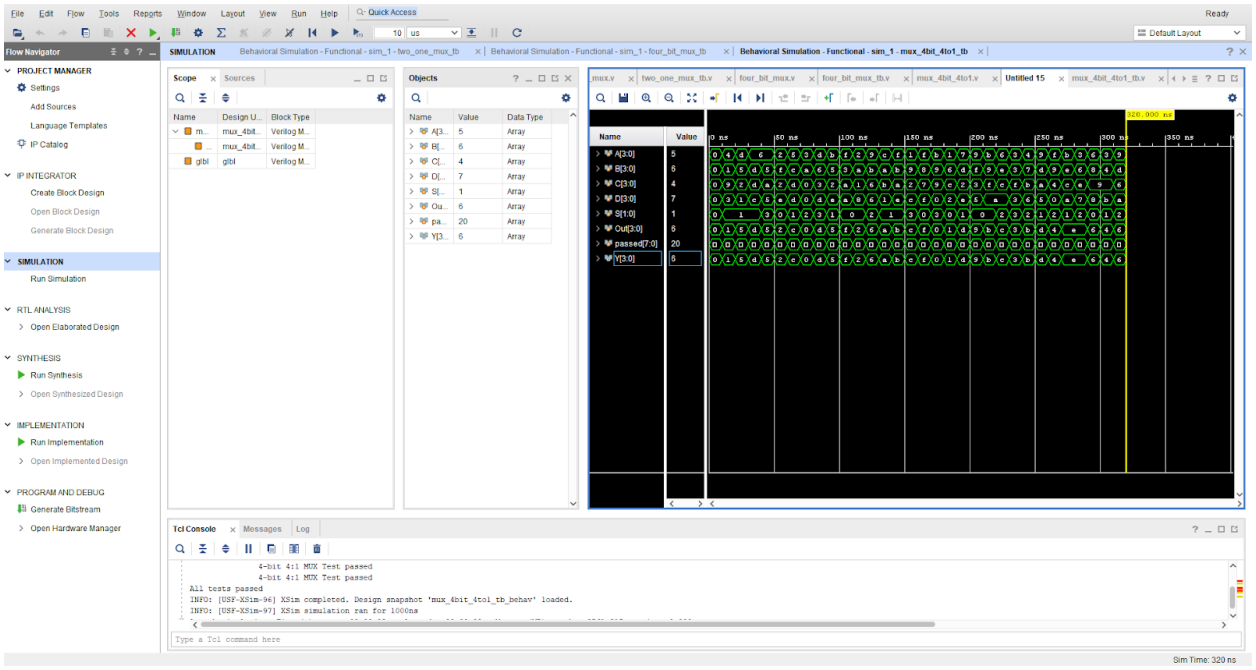
TwoOne Mux



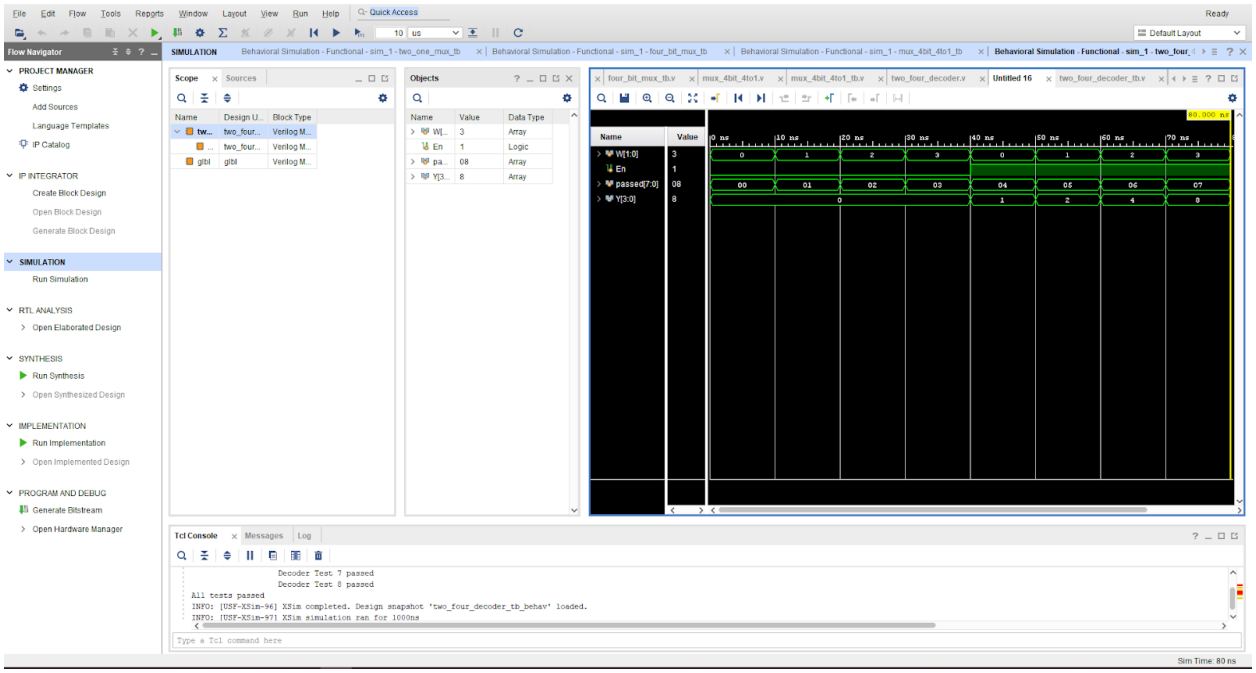
FourBit Mux



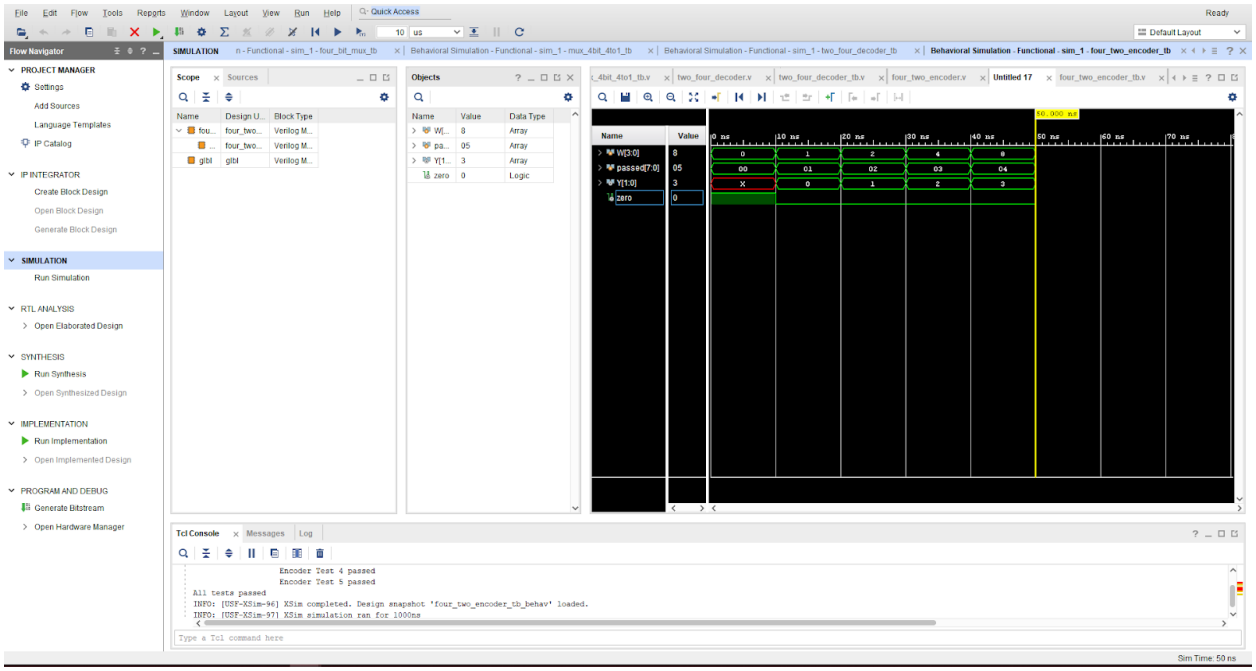
FourBit 4to1 Mux



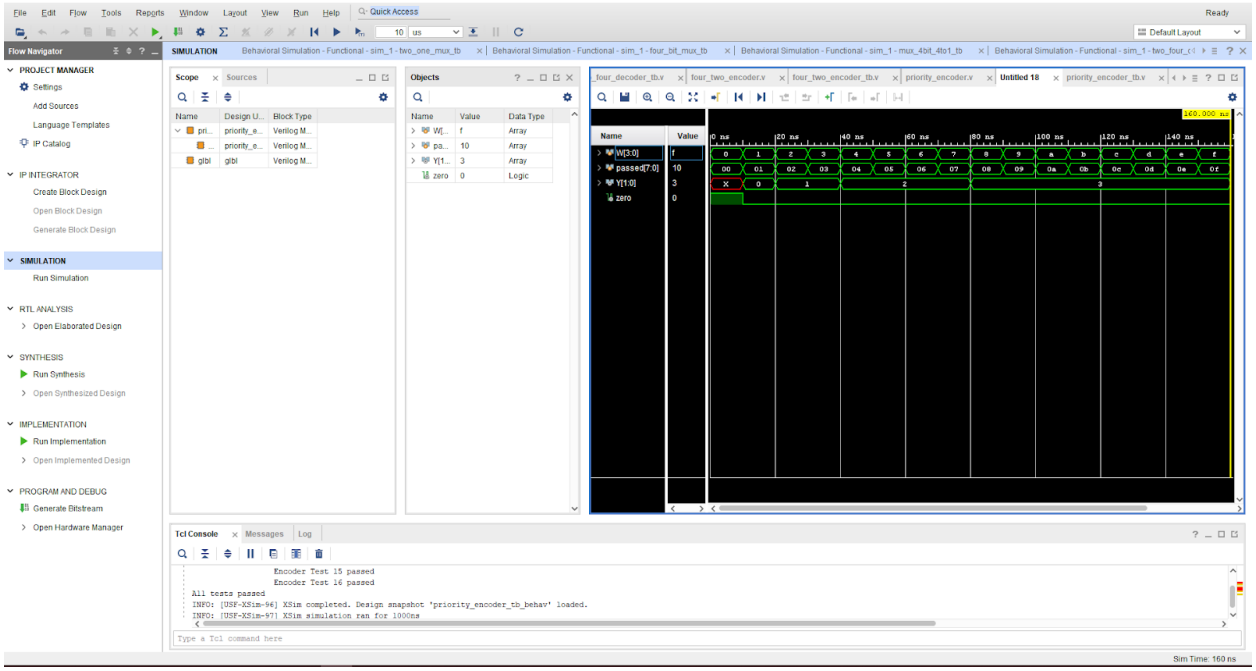
TwoFour Decoder



FourTwo Encoder



Priority Encoder



3. Provide a comparison between behavioral Verilog used in this week's lab and the structural and dataflow Verilog used in last week's lab. What might be the advantages and disadvantages of each.

The structural and dataflow approaches in Verilog are based on the outputs of specific functions, which are useful in the design of circuits with given gate schematics. These approaches are also very cost effective when designing low to mid level logic circuits , considering that it operates on a given number of transistors. On the other hand, these approaches are very tedious when applied to projects that require

implementing multiple logic gates. Programming on the verilog is much more effective when it comes to complex circuits. It also does not take into consideration the path in which outputs occur, however this is not ideal for the design of simple circuits that require fewer logic gates.

4. Compare the process of using a breadboard to implementing a digital circuit on an FPGA. State some advantages and disadvantages of each. Which process do you prefer?

Both the FPGA and breadboard have their own advantages and disadvantages. In order to gain a good fundamental knowledge of logic flows within a circuit the breadboard is better to use. All the inputs are connected by wires, and debugging is just more visual. However, where there's a more complex circuit a breadboard is not ideal, because it requires too many wires and more space is required. On the other hand, the FPGA is difficult to visually interpret, it's more helpful for complex designs. If the Verilog codes run correctly through the simulator, the display on the FPGA board should always be right. I prefer using the FPGA because it is much easier to set up and debugging doesn't get too complex for larger circuits, however without learning the basics on a breadboard first, I wouldn't have had a good fundamental understanding.

Feedback

1. What did you like most about the lab assignment and why? What did you like least about it and why?

I enjoyed learning how to use the Verilog, and observing the results cleared up my understanding of the logic gates. My least favorite part for the behavioral description for the MUXs.

2. Were there any sections of the lab manual that were unclear? If so, what was unclear? Do you have any suggestions for improving the clarity?

Overall the lab manual was very detailed, however clarifying how to fix errors during implementation could be helpful.

3. What suggestions do you have to improve the overall lab assignment?

This lab was very easy to understand using the lab manual, hence I wouldn't change anything about the lab .