Lab2: Behavioral logic design and Verilog Testbench and Simulation

Leonardo Fusser (1946995)

Objectives:

- Create Verilog testbench to perform simulation on design.
- Simulate circuit behavior with Xilinx ISim simulator.
- Familiarize with Verilog behavioral procedural blocks.
- Code combinational logic behaviorally.
- **Implement the mux on the target using top module.**
- Test on the target.

NO full report to submit.

To hand in:

- Listing of the following for each of your designs on LEA:
  1. This sheet with answers and screenshots.
  2. Commented Verilog code modules: mux module, **top module,** and fixture module.
  3. Commented resource summary report copied in a word document (must be commented).
  4. Commented Technology schematics screenshots copied in this word document (must be commented).
  5. Commented RTL schematics screenshots copied in this word document (must be commented).

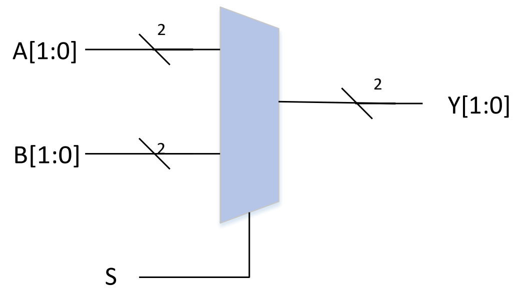
- The main file must include a prolog header: Lab number and name, author, description, version number.
- Each and every module should include a prolog header:

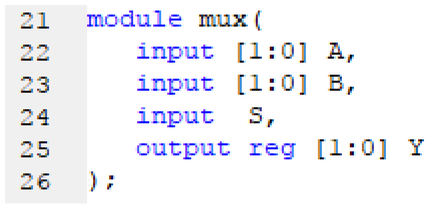
```
/* ************************************************************
*     Module name: counter
*     Description: Divides the input clk by 2 power N where
*                              N is the tapped register number.
*
*     Author             Date         Comments          Revision
*************************************************************
*     Serge Hould        2015-11-25   Original          1.0
*     Serge Hould        2016-11-25   Add comments      1.1
*************************************************************/
```

Pre-lab preparation:

In this lab, you are to create an improved 2-1 multiplexer for 2-bit bus (rather than 1 bit). Also, you are going to code it in behavioral modeling rather than gate-level modeling.

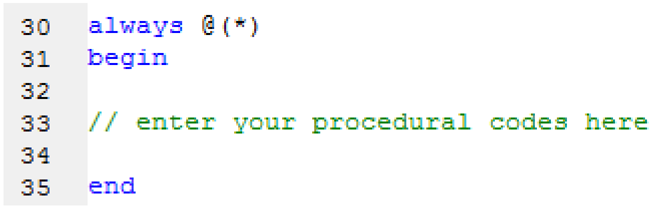
Create a Verilog module called mux_2_1 with input A, B, S and output Y as shown below:





```
module mux(
    input [1:0] A,
    input [1:0] B,
    input  S,
    output reg [1:0] Y
);
```

Code the mux behavior in an always block. You may choose to use “if” statement, or “case” statement, or both to explore the coding options.



```
always @(*)
begin

// enter your procedural codes here

end
```

Your solution:

```
////////////////////////////////////////////////////////////////////////////////
always @(*)

if(S == 0)                      //If select bit = 0...
        begin
                Y = A;          //Output = value stored in A.
        end

else                            //If select bit = 1...
        begin
                Y = B;          //Output = value stored in B.
        end
////////////////////////////////////////////////////////////////////////////////
```

Lab Work:

Part 1: 2-1 mux design and Test Bench simulation

1. Code the mux behavior as per your preparation.

   Create the Verilog module. You **must** use the following I/O names:

```
module mux_2_1(
     input [1:0] A,
     input [1:0] B,
     input S,
     output reg [1:0] Y
);
```

2. Create a Verilog testbench for your design
3. Generate sufficient input stimuli to cover all the test cases. Apart from using delay command, you HAVE to use a *for* loop to generate continuous input bit pattern for simulation.
4. How many loops are required to test all possibilities?

   Explain:

   Regardless of how the for loop is formatted, a total of 32 iterations are needed to test all possibilities. This is because two of the inputs represents two bits each (A = xx and B = xx) and taking into account the select bit (S = x, which is only one bit), the total number of input bits are five bits. Therefore, applying basic math, 2^5 = 32 total possibilities which is why 32 iterations are needed to test possibilities (number of iterations = 2^N, where N = number of input bits).

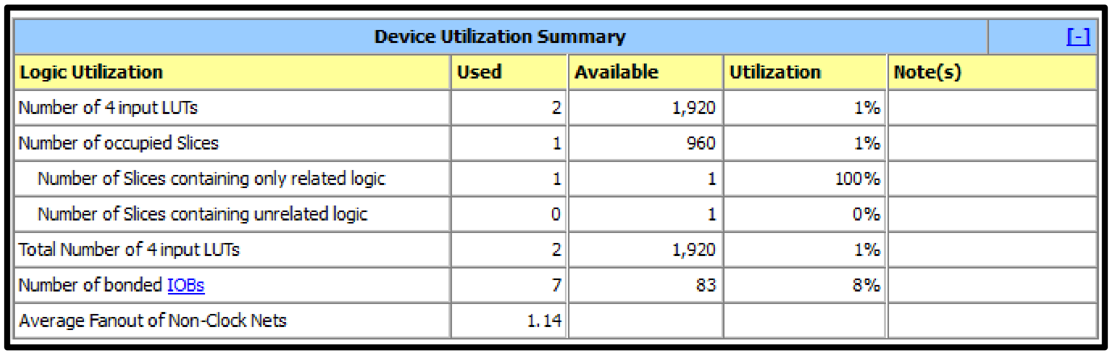
5. Now your design is ready for behavioral simulation using ISim.

Verify your results of your simulation. Is this the same as your truth table?

After running the simulation with Xilinx ISim, the results reveal the same behavior my truth table predicated will happen for the 2-bit 2x1 MUX that was implemented for this Lab.

From the Design Summary/Report, find out how many resources that have been used by your design, including number of LUTs, IOBs, Slices etc. Comment on the resources used. Are they as expected?

From the Design Summary/Report, it could be noted that two 2 input LUTs, seven IOBs and one slice are used to implement the 2-bit 2x1 MUX for our particular FPGA. Complete table showing resources used shown below.



Device Utilization Summary				[-]
Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of 4 input LUTs	2	1,920	1%	
Number of occupied Slices	1	960	1%	
Number of Slices containing only related logic	1	1	100%	
Number of Slices containing unrelated logic	0	1	0%	
Total Number of 4 input LUTs	2	1,920	1%	
Number of bonded IOBs	7	83	8%	
Average Fanout of Non-Clock Nets	1.14			

Figure 1. Complete list and number of resources used to implement the 2-bit 2x1 MUX on our FPGA shown above.

6. Show your simulation results to your instructor.
7. Generate the following two schematics:
   a) RTL schematic:

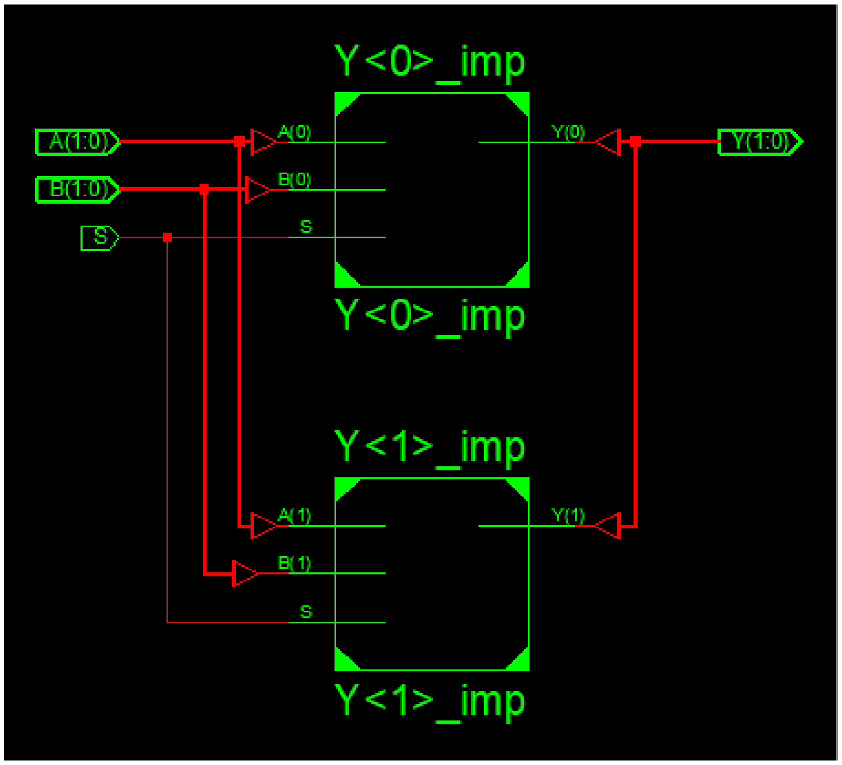


Figure 2. Detailed top-level RTL schematic for 2-bit 2x1 MUX implemented on our particular FPGA shown above. See “After Lab Questions” Q2 for more details.

b) Technology schematic:

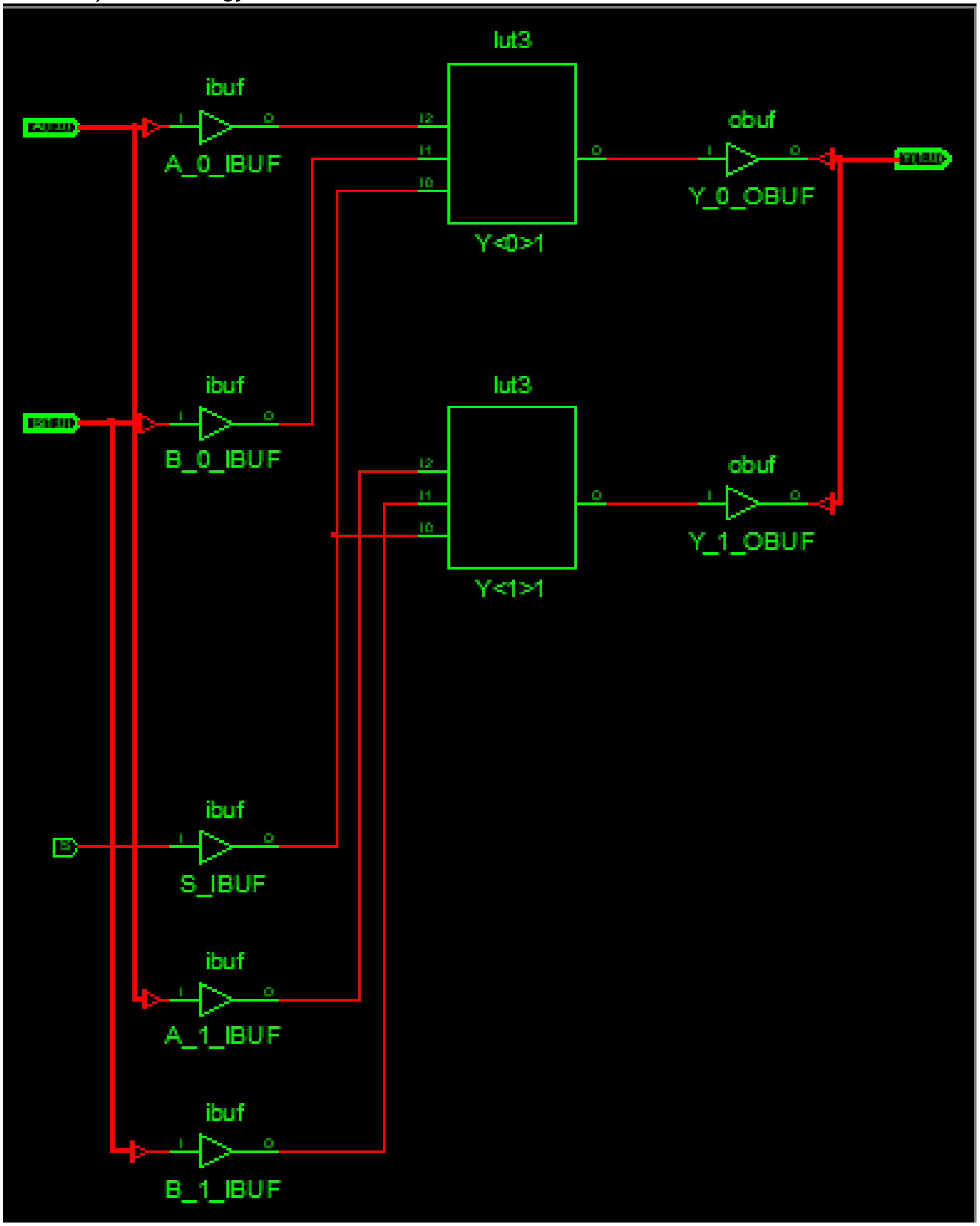


Figure 3. Detailed low-level Technology schematic for 2-bit 2x1 MUX implemented on our particular FPGA shown above. See "After Lab Questions" Q2 for more details.

Part 2 : Implement the mux on the target using top module

Install Adept on your computer:

https://reference.digilentinc.com/reference/software/adept/previous-versions

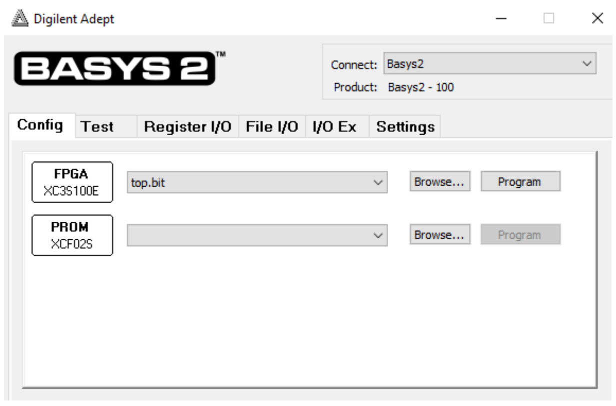
You are required implement the multiplexor using the following I/Os:

- 4 on-board slide switches will be used to provide the data input.
- 1 push buttons will be used as select signals.
- LED 0 and 1 will be used to show the output of the multiplexer.

8. Create a UCF file for your mux design.
   Your mapping of mux's input and output ports should be based on the following:

   A[0] to sw[0], A[1] to sw[1]
   B[0] to sw[2], B[1] to sw[3]
   S to btn[0]
   Y[0] to Led[0], Y[1] to Led[1]

9. Compile and generate programming file for your design. Download to Basys2 board and verify your results. Demonstrate your working board to your instructor.



Connect: Basys2
Product: Basys2 - 100
top.bit

10. Show the board demo to your instructor.

If you don't have the board in hand, you are going to demo next week.

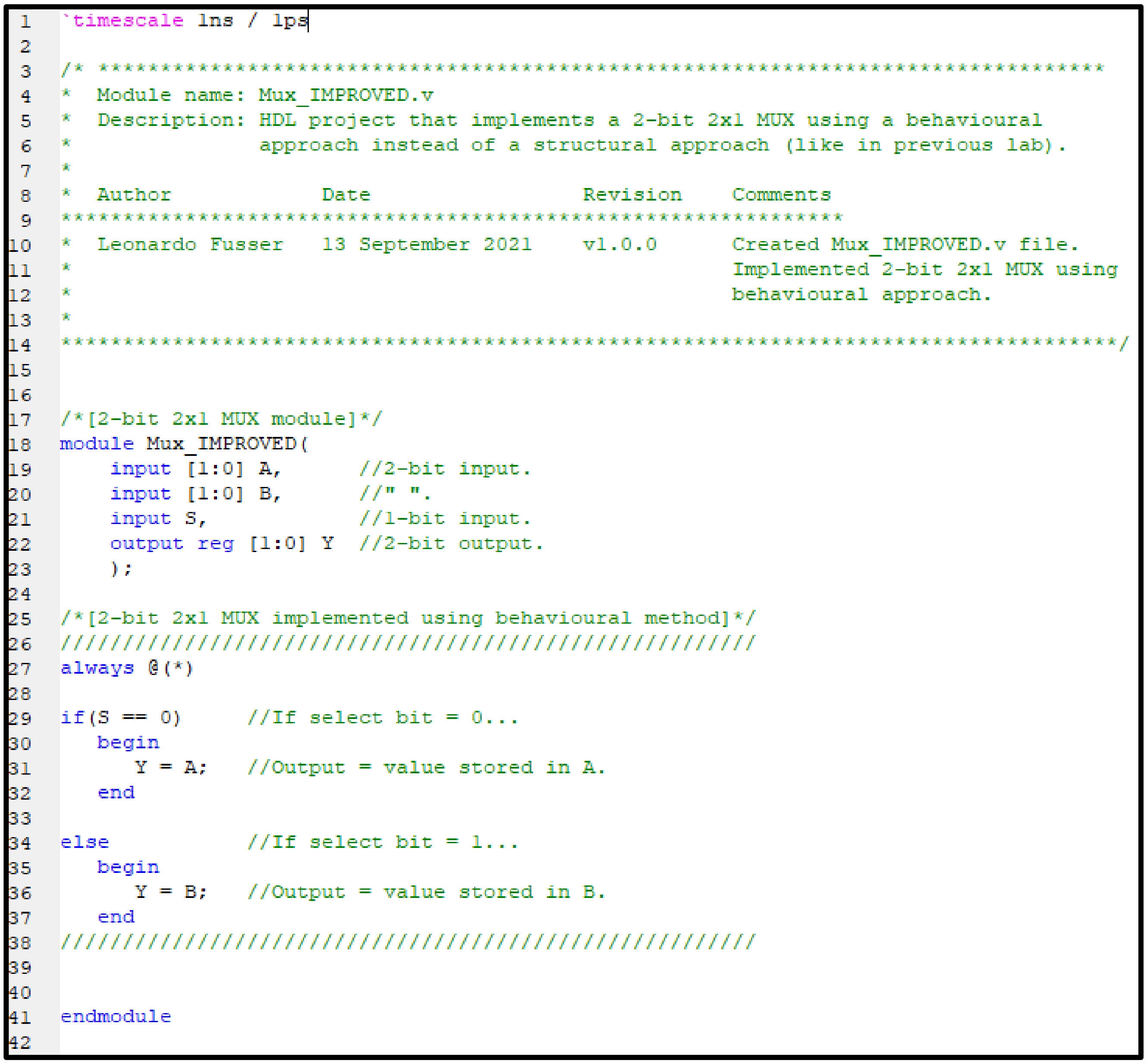
After lab questions:

Q1 - Briefly describe what you have learnt about the behavioral method of programming combinational circuit compared to the structural method, which approach do you prefer and why.

- Comparing the previous lab (used structural approach to implement a 1-bit 2x1 MUX) to this lab, the behavioral method of programming (used in this lab to implement the 2-bit 2x1 MUX) a combinational circuit is different to that of structural method programming. This is because programming based on behavioral approach is relies on the functionality of the circuit and the result is usually more precise (it is less error prone). On the other hand, programming based on structural approach relies on the way different parts of the circuit are connected together and the result could be not as precise (it is error prone). Looking back at the two approaches, behavioral method of programming a combinational circuit is preferred because it is easier to implement and is less time consuming than following the other approach. Structural method of programming a combinational circuit is good when the circuit is relatively small. For large designs, behavioral method is preferred.

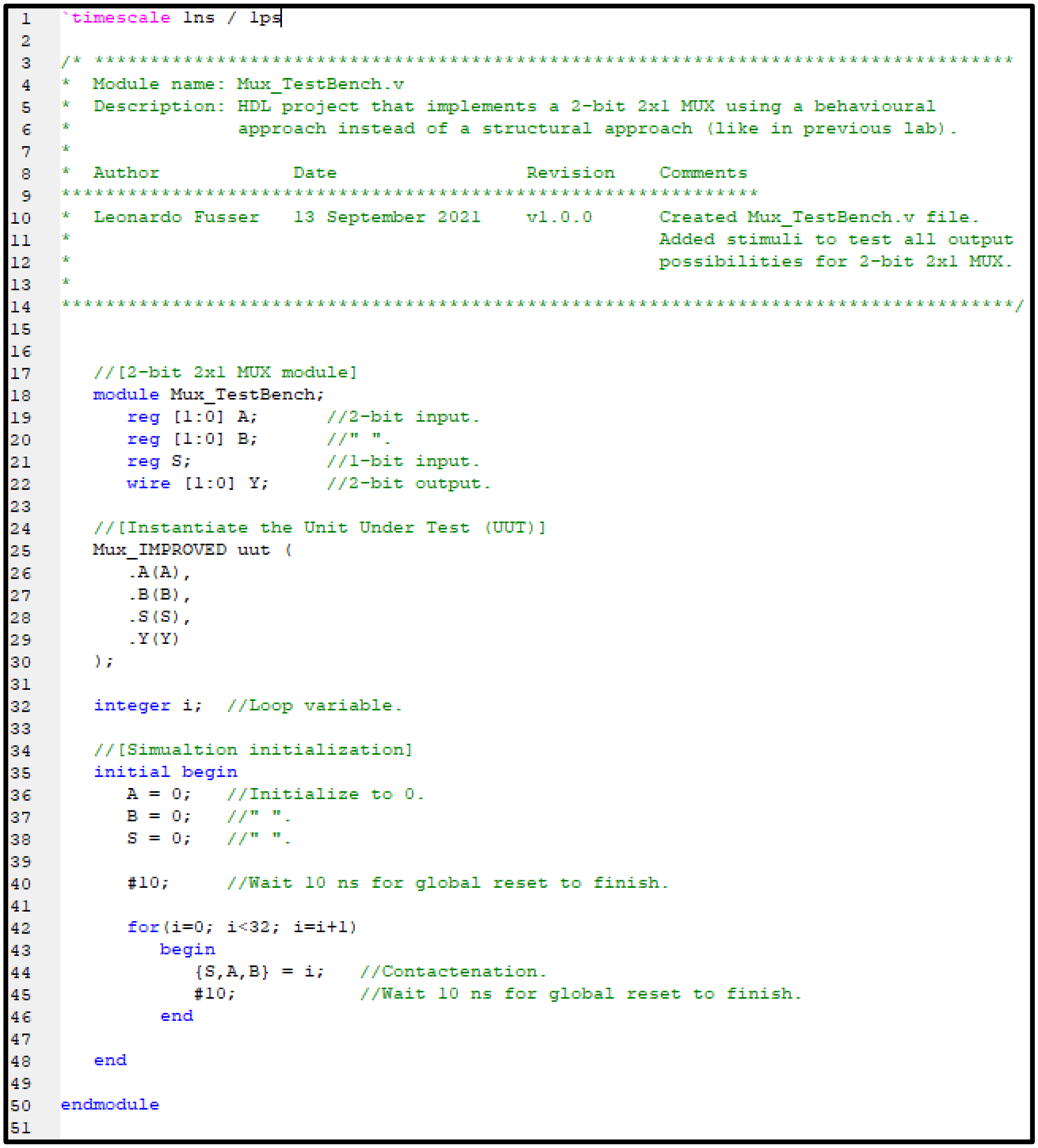
Q2 - Explain the logic circuit from the RTL and Technology schematics.

- The logic circuit from the RTL schematic shows what gets generated after the HDL synthesis process. In this schematic, it shows the representation of the pre-optimized design (for FPGA) in terms of standard logic symbols (such as adders, AND gates, OR gates, etc…) that are independent of the particular FPGA being used. While viewing the RTL schematic, it allows the user to see a high-level (top-level) representation of your HDL circuit before it is optimized for your particular FPGA. On the other hand, the logic circuit from the Technology schematic shows a little more detail in regard to the HDL circuit that is being designed in the Xilinx ISE software. Here, the Technology schematic is generated after the optimization and technology targeting phase of the synthesis process. It shows the representation of the design in terms of logic symbols optimized for the particular FPGA. While viewing the Technology schematic, it allows the user to see a technology-level (low-level) representation of your HDL circuit after it is optimized for your particular FPGA. See figures 2 and 3 above in this document.



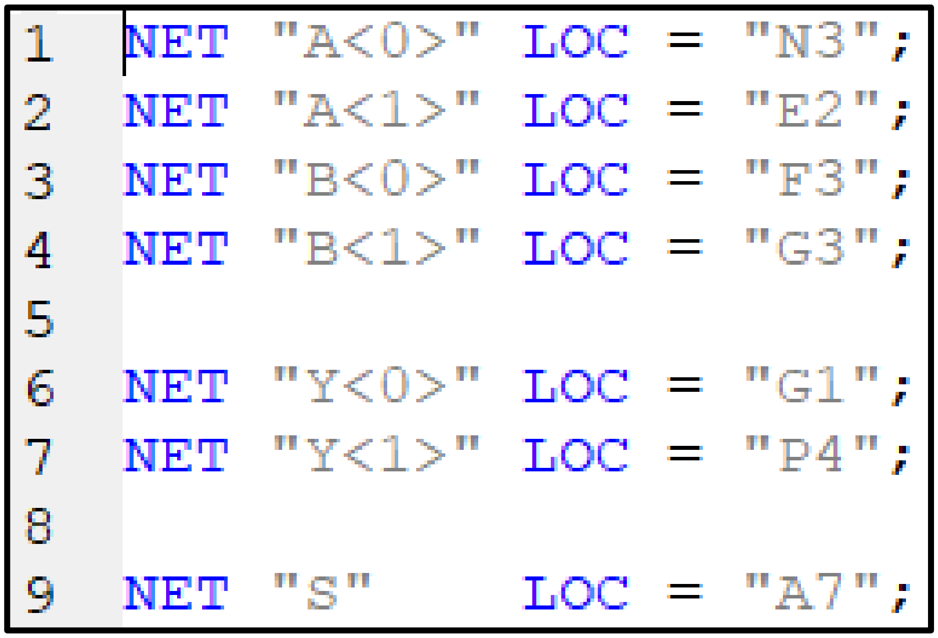
```verilog
`timescale 1ns / 1ps

/* ********************************************************************************
*  Module name: Mux_IMPROVED.v
*  Description: HDL project that implements a 2-bit 2x1 MUX using a behavioural
*               approach instead of a structural approach (like in previous lab).
*
*  Author            Date                 Revision    Comments
**************************************************************
*  Leonardo Fusser   13 September 2021    v1.0.0      Created Mux_IMPROVED.v file.
*                                                     Implemented 2-bit 2x1 MUX using
*                                                     behavioural approach.
*
*********************************************************************************/


/*[2-bit 2x1 MUX module]*/
module Mux_IMPROVED(
    input [1:0] A,      //2-bit input.
    input [1:0] B,      //" ".
    input S,            //1-bit input.
    output reg [1:0] Y  //2-bit output.
    );

/*[2-bit 2x1 MUX implemented using behavioural method]*/
////////////////////////////////////////////////////////
always @(*)

if(S == 0)      //If select bit = 0...
   begin
      Y = A;    //Output = value stored in A.
   end

else            //If select bit = 1...
   begin
      Y = B;    //Output = value stored in B.
   end
////////////////////////////////////////////////////////


endmodule

```

Figure 4. Verilog code used to create 2-bit 2x1 MUX (implemented using behavioral method of programming) shown above.



```
`timescale 1ns / 1ps

/* *************************************************************************************
*  Module name: Mux_TestBench.v
*  Description: HDL project that implements a 2-bit 2x1 MUX using a behavioural
*               approach instead of a structural approach (like in previous lab).
*
*  Author            Date                 Revision    Comments
***************************************************************
*  Leonardo Fusser   13 September 2021    v1.0.0      Created Mux_TestBench.v file.
*                                                     Added stimuli to test all output
*                                                     possibilities for 2-bit 2x1 MUX.
*
**************************************************************************************/


   //[2-bit 2x1 MUX module]
   module Mux_TestBench;
      reg [1:0] A;      //2-bit input.
      reg [1:0] B;      //" ".
      reg S;            //1-bit input.
      wire [1:0] Y;     //2-bit output.

   //[Instantiate the Unit Under Test (UUT)]
   Mux_IMPROVED uut (
      .A(A),
      .B(B),
      .S(S),
      .Y(Y)
   );

   integer i;  //Loop variable.

   //[Simualtion initialization]
   initial begin
      A = 0;   //Initialize to 0.
      B = 0;   //" ".
      S = 0;   //" ".

      #10;     //Wait 10 ns for global reset to finish.

      for(i=0; i<32; i=i+1)
         begin
            {S,A,B} = i;   //Contactenation.
            #10;           //Wait 10 ns for global reset to finish.
         end

   end

endmodule
```

Figure 5. Verliog Testbench used to simulate 2-bit 2x1 MUX (implemented in Verliog code using behavioral method -> see "Figure 4" above) shown above.



```
NET "A<0>" LOC = "N3";
NET "A<1>" LOC = "E2";
NET "B<0>" LOC = "F3";
NET "B<1>" LOC = "G3";

NET "Y<0>" LOC = "G1";
NET "Y<1>" LOC = "P4";

NET "S"    LOC = "A7";
```

Figure 6. Constraints file (.ucf file) used to map locations of Basys 2 board peripherals (LEDs, toggle switches and pushbuttons) to inputs and outputs created in Verilog code (see "Figure 4" above) shown above.

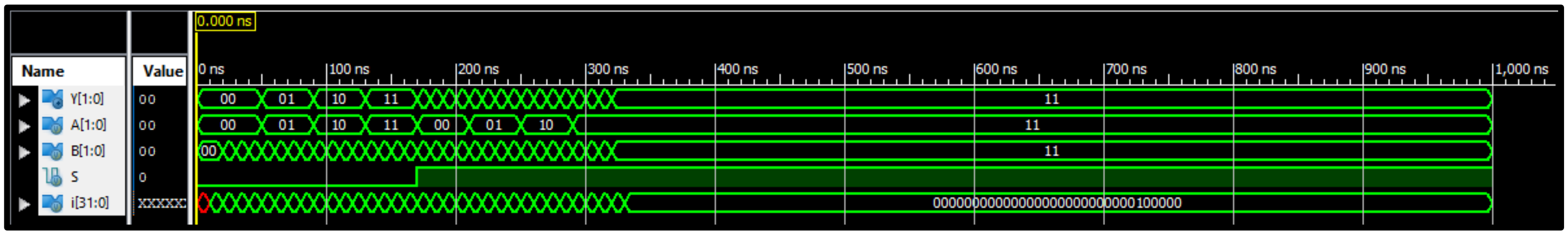


Figure 7. Result from Xilinx ISim simulating 2-bit 2x1 MUX implemented in Verliog code using behavioral approach (see "Figure 4" above) shown above. See truth table on next page.

2-bit 2x1 MUX truth table:

Inputs				Select	Outputs	
A0	**A1**	**B0**	**B1**	**S**	**Y0**	**Y1**
0	0	0	0	0	0	0
…	…	…	…	…	…	…
1	1	1	1	1	1	1

When S = 0, Y0 and Y1 = A0 and A1 respectively.
When S = 1, Y0 and Y1 = B0 and B1 respectively.