# LABORATORY # 2

# Combinational Logic Design with Verilog

**PART 1[1] (decoder)** **Design of Sprinkler Valve Controller**

**PART 2 (multiplexer)** **Design of Computer Data Bus**

[1] Design Example guided through by TA

## Objectives

**Lab 2** contains 2 parts: **Part 1** – guided design and **Part 2** – individual design. Its purposes are to get familiar with:

**i.** Xilinx Vivado software system usage

**ii.** Design of controller systems based on combinational logic

**iii.** Design entry via text-based HDL (a gate-level schematic accompanies)

**iv.** Generation of testbenches for logic design testing and verification

**v.** Generation of simulation waveforms and analysis

## Equipment

- PC or compatible
- Xilinx Vivado Software

# PART 1. Design of a Sprinkler Valve Controller[2]

In this guided simulation experiment, we will design and test a 3 x 8 decoder (with "enable" switch) for a sprinkler valve controller system:

## Specification

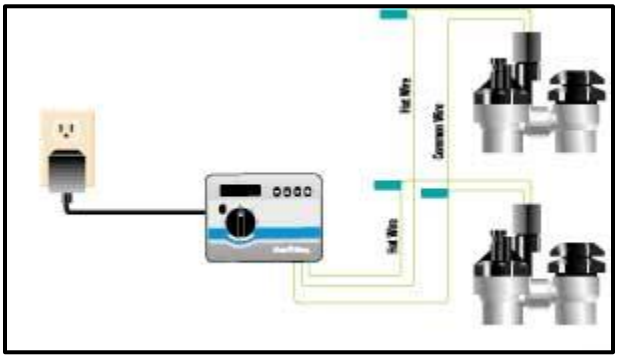

**Figure 1.** Sprinkler Digital Controller System

Automatic lawn sprinkler systems control the opening and closing of water valves. A sprinkler system must support several different zones, such as the backyard, left side yard, right side yard, etc. **Only one zone's valve can be opened at a time** to maintain enough pressure in the sprinklers in that zone. In this design assignment, a sprinkler system must support up to 8 zones. Note that typically a sprinkler system is controlled by a small microcontroller unit (MCU) which executes a program that opens each valve only at specific times of the day and for specific durations. However, we will limit ourselves to a sub-project that is dealing only with opening and closing of the valves. The system must also provide a facility to disable the opening of any valve.

## Analysis and Design

Assuming a microcontroller has only four output pins, a system based on a 3 x 8 decoder (with "enable" switch) will do the job.

[2] Both parts of the lab are based on examples from Frank Vahid's "Digital Design"

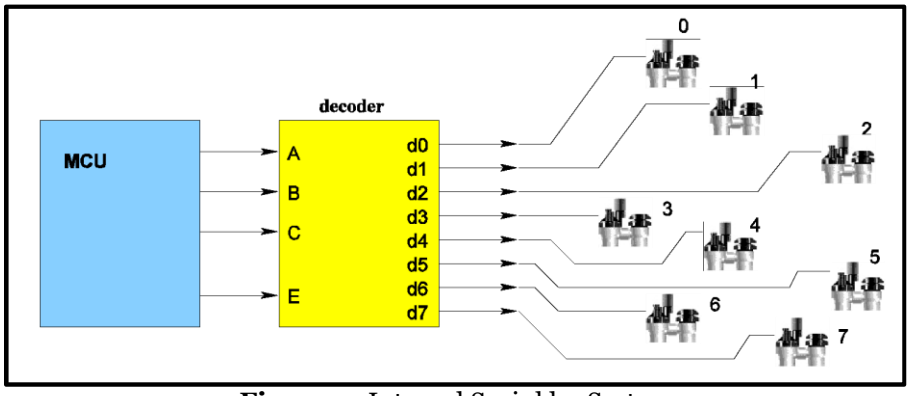


**Figure 2.** Internal Sprinkler System

MCU has one pin to indicate whether the system is active (enabled) and the other three pins indicate the binary number of the valve to be opened. The system is a combinational logic circuit that has 4 inputs: E (enabler) and A, B, C (the binary value of the active zone), and 8 outputs d7, …, d0 (the valve controls). The truth table of the system is shown below.

| E | A | B | C | d0 | d1 | d2 | d3 | d4 | d5 | d6 | d7 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | x | x | x | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

**Table 1.** Truth Table of the Sprinkler System ('x' stands for "don't care")

Following the procedure outlined in zyBook section 1.13, constructing sum-of-products (SOP) minterm based logic expression for each of the data outputs d.

***Step 1*** – Capture the function.

$$d0 = E \,.\, A' \,.\, B' \,.\, C'$$
$$d1 = E \,.\, A' \,.\, B' \,.\, C$$
$$d2 = E \,.\, A' \,.\, B \,.\, C'$$
$$d3 = E \,.\, A' \,.\, B \,.\, C$$
$$d4 = E \,.\, A \,.\, B' \,.\, C'$$
$$d5 = E \,.\, A \,.\, B' \,.\, C$$
$$d6 = E \,.\, A \,.\, B \,.\, C'$$
$$d7 = E \,.\, A \,.\, B \,.\, C$$

***Step 2*** – Convert to equations and/or minimize.
Nothing to do here (already minimized).

***Step 3*** – Implement as a gate based logic circuit.
It is shown below using a non-standard[3] 4-input AND gates and inverters.

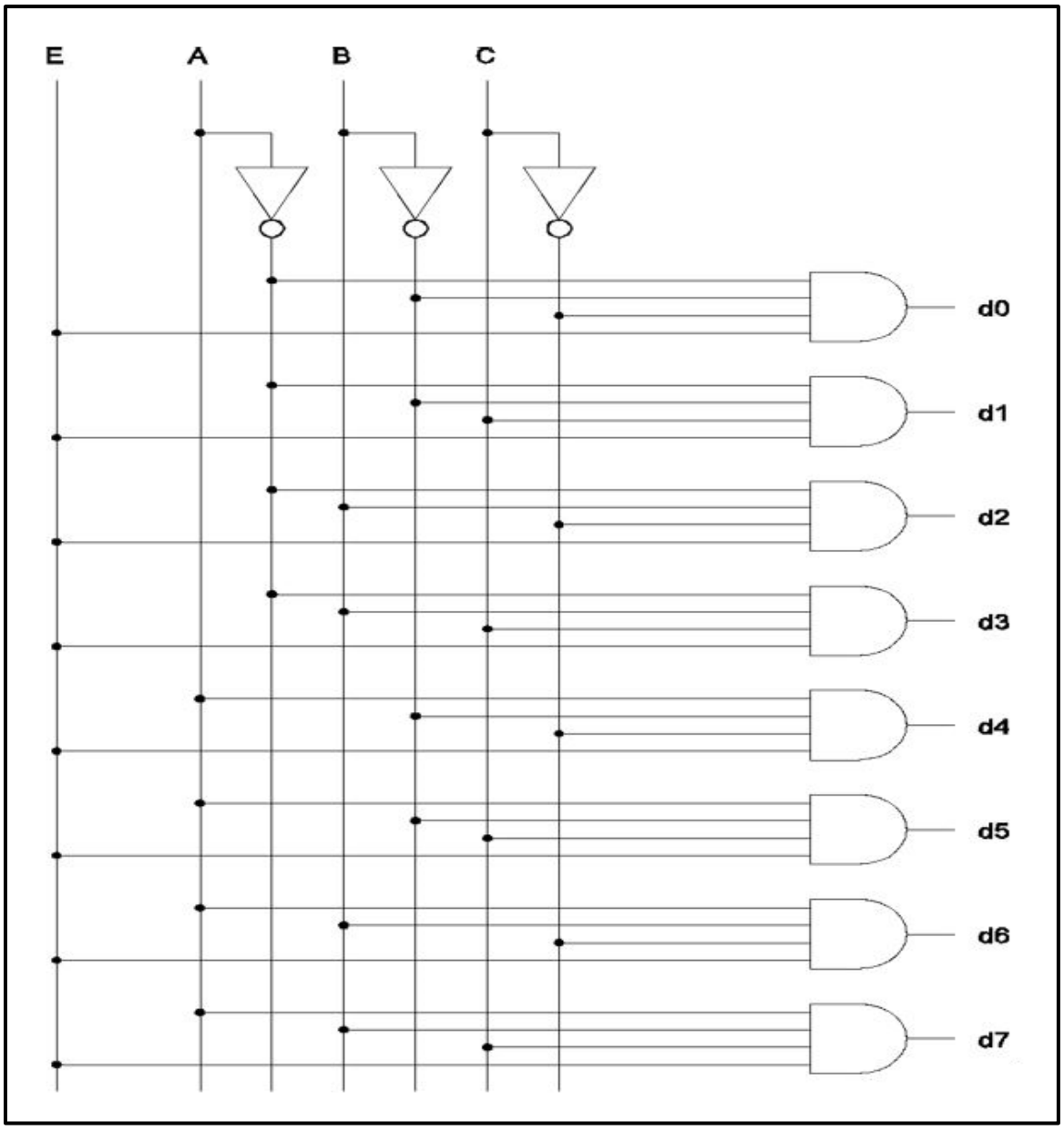


**Figure 3.** Logic Circuit Schematic of the Sprinkler System based on **Table 1**

[3] Usage of non-standard gates largely clarifies a circuit schematic, but in many cases requires a creation of a gate library manually within Logic Design Software Environments such as Xilinx Vivado.

## Design Entry using Text-Based HDL

Our next task is to implement the design above in Verilog, which is a Hardware Description Language and is used for describing electronic circuits and systems in a text-based manner. There are three ways to model circuits in Verilog as discussed below.

(i) ***Structural Modeling*** – This is the gate-level modeling of the circuit. (It most closely resembles the schematic entry and, in a way, replaces a gate symbol with its text-based description.)

*(ii)* ***Dataflow Modeling*** – This is mainly used to describe combinational circuits using the continuous assignment statements. (More on this later).

(iii) ***Behavioral Modeling*** – This is the most high-level modeling technique in Verilog and is used for complex circuits, both sequential and combinational. It allows capturing behavior and lets a synthesis tool to automatically translate it into gates (at a later point).

***Note*** *-> The schematic entry has been popular for many years but is no longer practiced for digital gate-level design. HDL-based design entry has replaced it. However, schematics are required to be submitted as part of the lab report.*

In this lab, our task is to model the circuits using structural and behavioral modeling techniques. Since Part 1 is the guided example, all the Verilog codes will be provided for both tasks. For Part 2, you are required to work out the schematic and equations, and model the circuit structurally as well as behaviorally in Verilog.

### *Part A. Structural Modeling*

***Step 1 (Creating a new project)*** – The first step is to create a new project in Vivado. Go to File -> New Project, to open the wizard for new project creation. The wizard will prompt you to enter Project Name (can be named as Lab_2) and location. Project Type can be selected as RTL Project (and check the box stating Do not specify sources at this time). We will be using Basys3 board in this course, and its part number is *XC7A35TCPG236-1*. Hence, this board needs to be selected in the Project menu. Click on Finish to start the new project. The figures below demonstrate the creation of a new project in Xilinx Vivado.

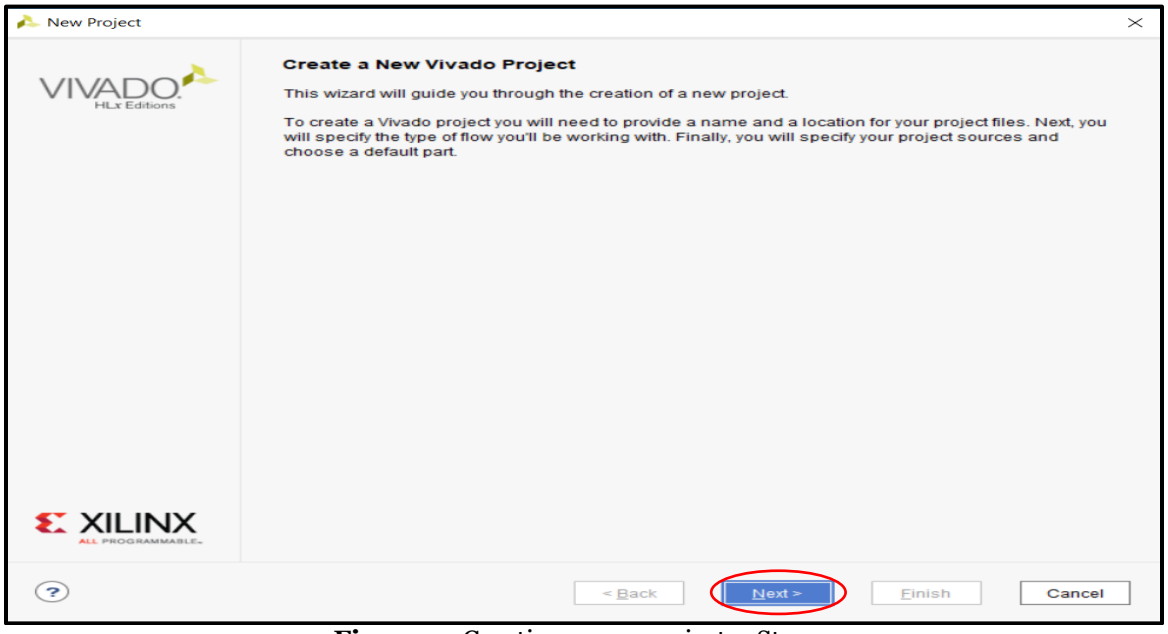


**Figure 4.** Creating a new project – Step 1

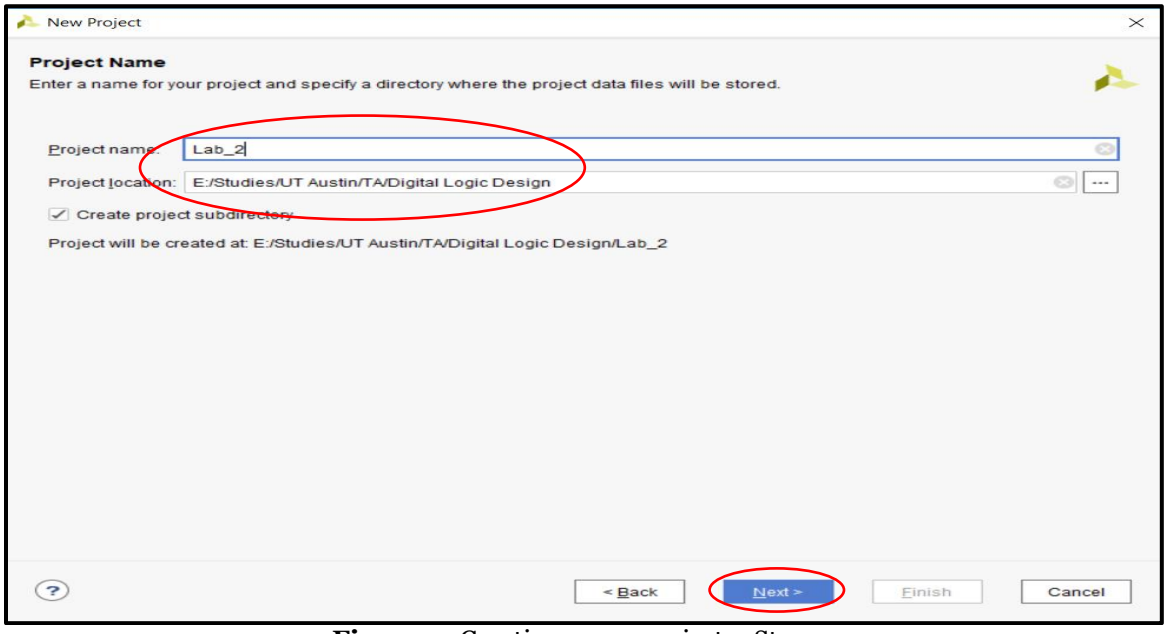


**Figure 5.** Creating a new project – Step 2

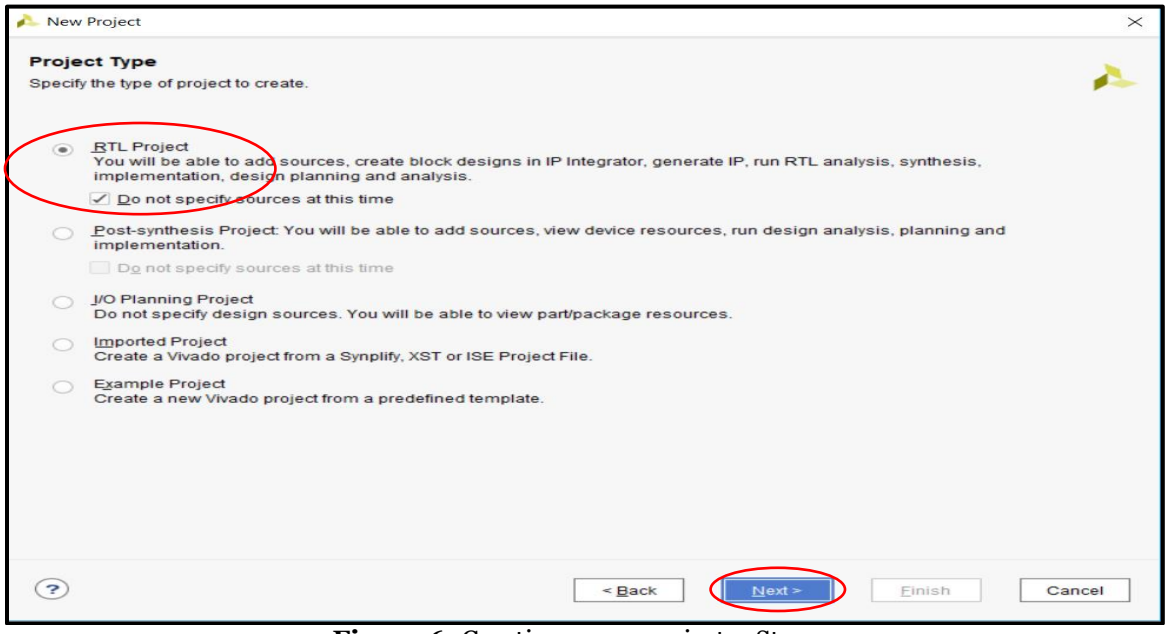


**Figure 6.** Creating a new project – Step 3

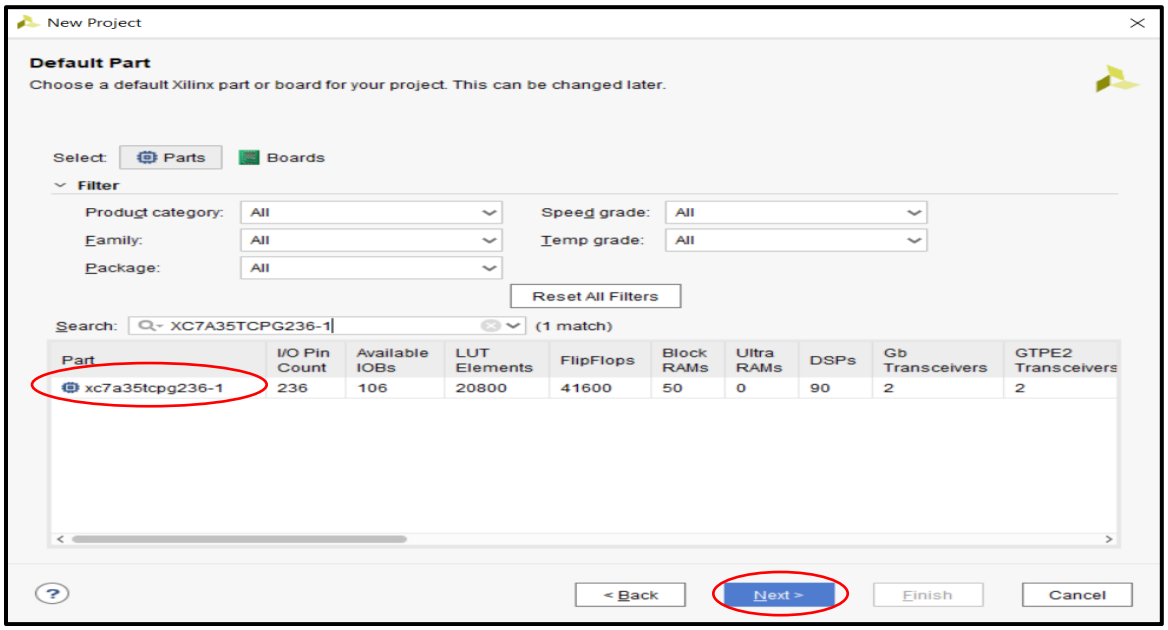


**Figure 7.** Creating a new project – Step 4

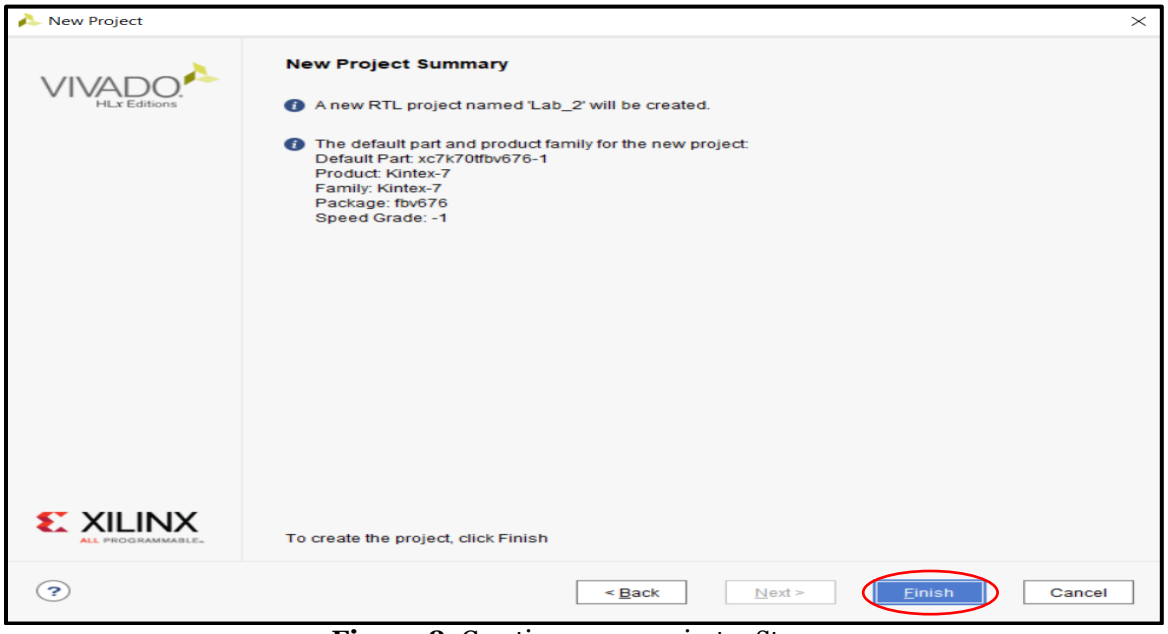


**Figure 8.** Creating a new project – Step 5

***Step 2 (Creating a Verilog file)*** – The next step is to create the Verilog file for the Decoder module. Click on File -> Add Sources to start creating the Verilog module definition. In the Add Sources wizard, select Add or create design sources and click on Next. Click on Create File option to create the new Verilog file. Select File type as Verilog, and File name can be of your choice, but it is a good practice to keep the File name similar to the name of the top module (which in Part A is Decoder_structural and Part B is Decoder_behavioral). File location should be kept as <Local to Project>. Click on Finish to open the Define Module wizard, where the port definitions of the module are defined. For the Decoder design, there are four input ports (e, a, b, c) and eight output ports (d0, d1, …, d7). Add the port names and their corresponding directions in this window (these can be changed later on in the code as well), and click on Ok. This will create a Verilog file under the Design Sources tab, which can be opened by double-clicking on it. The steps to create a new Verilog file in the Vivado project are demonstrated in the figures below.

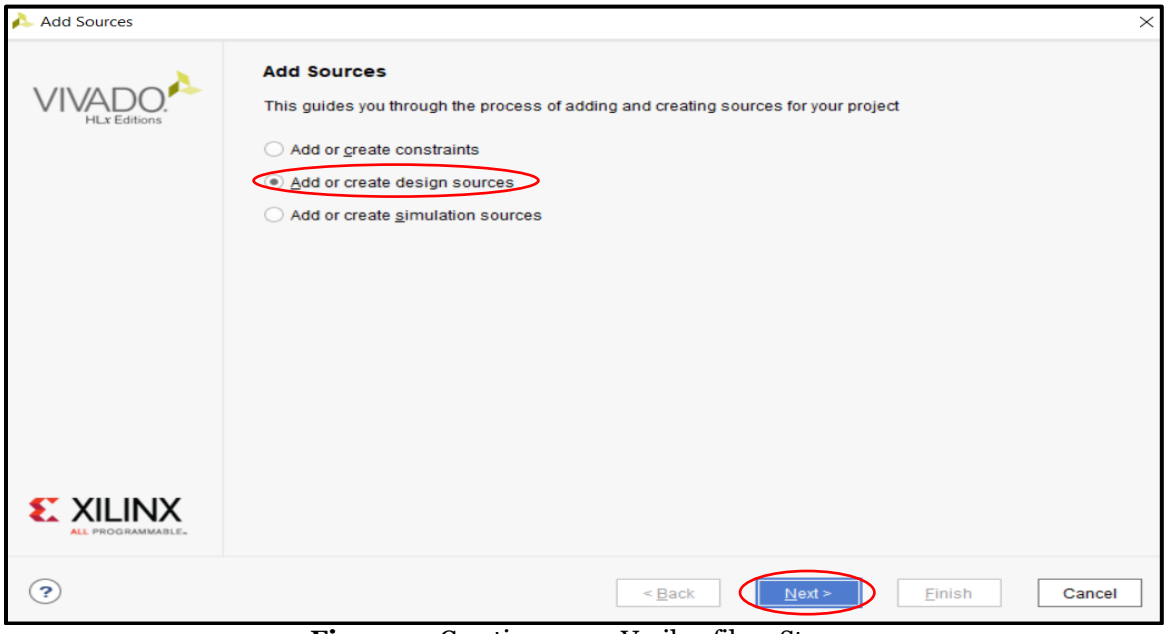


**Figure 9.** Creating a new Verilog file – Step 1

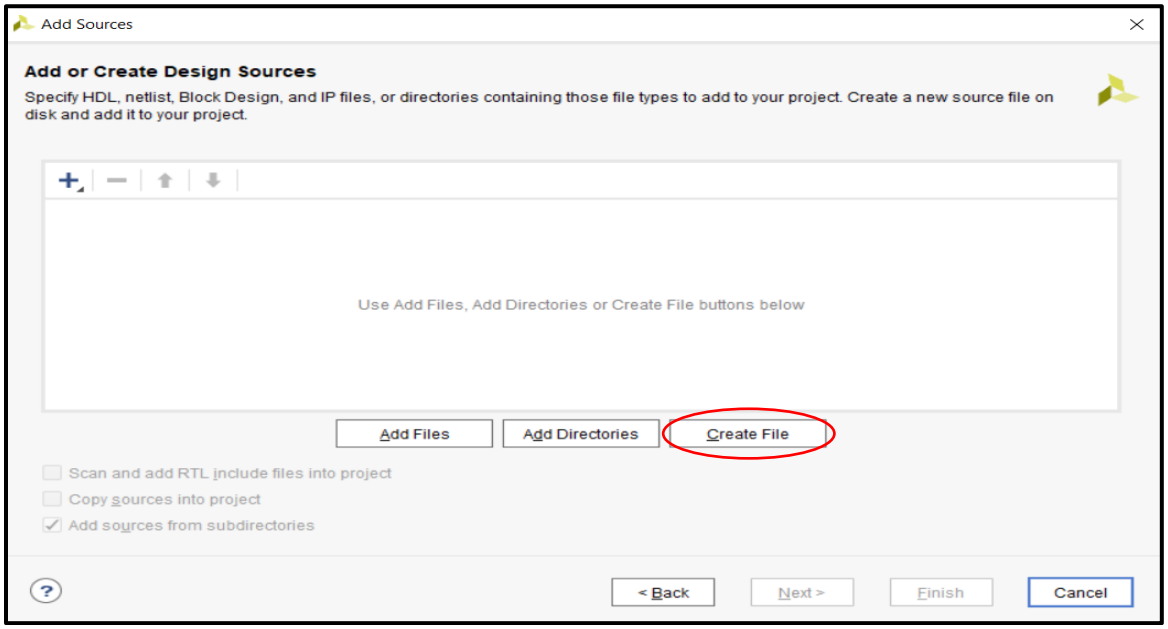


**Figure 10.** Creating a new Verilog file – Step 2

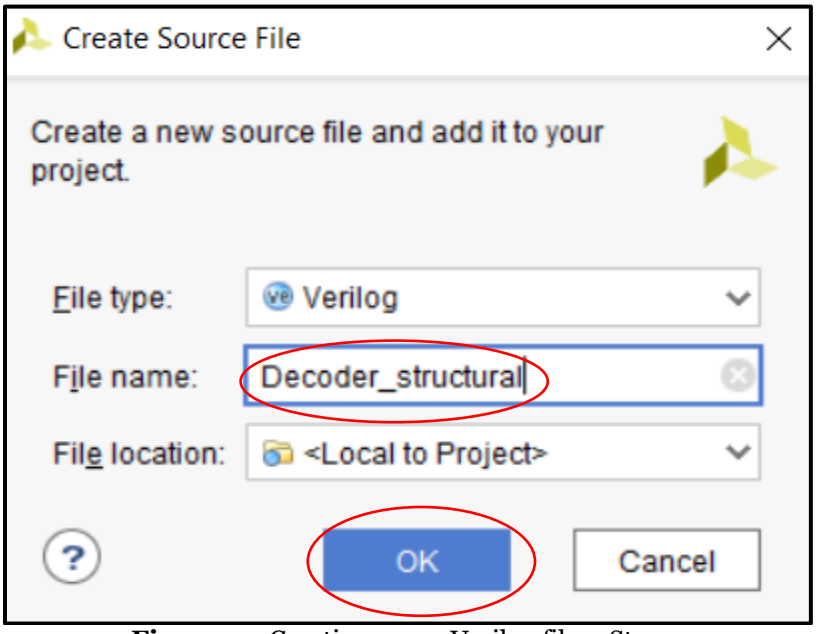


**Figure 11.** Creating a new Verilog file – Step 3

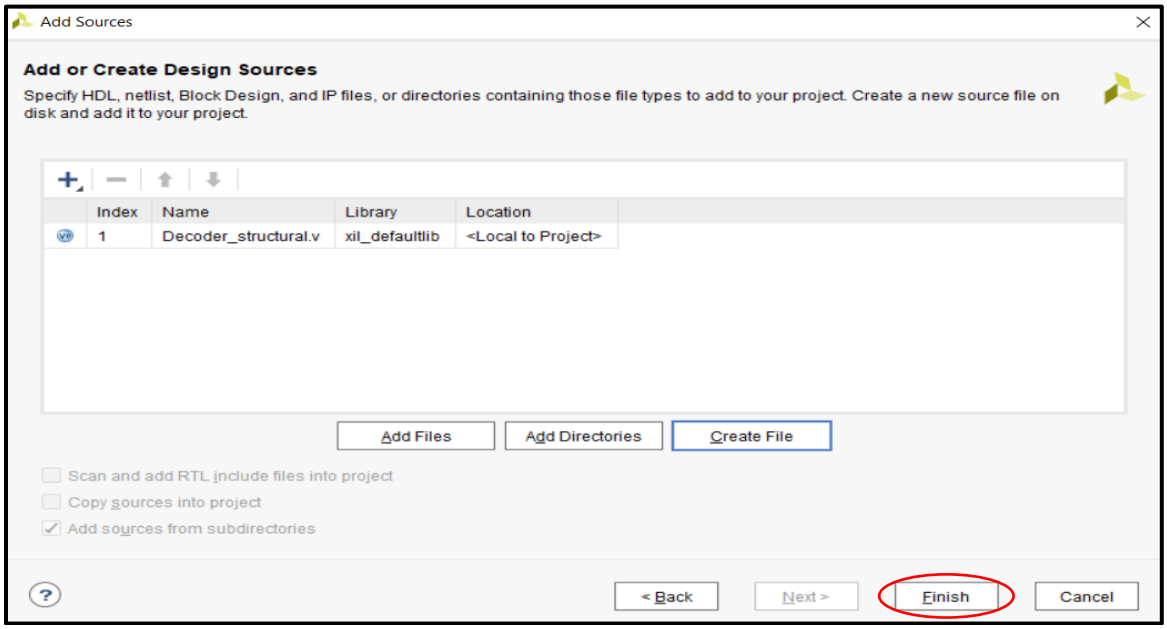


**Figure 12.** Creating a new Verilog file – Step 4

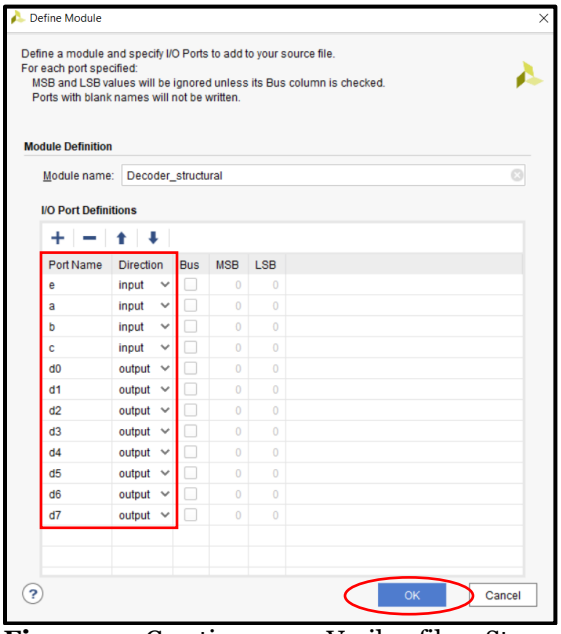


**Figure 13.** Creating a new Verilog file – Step 5

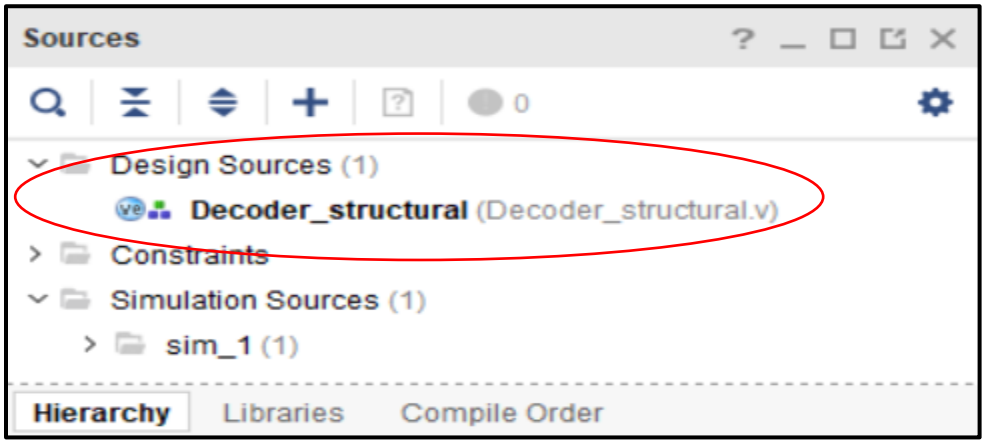


**Figure 14.** Newly created Verilog file – appearing in Design Sources

***Step 3 (Writing Verilog Code)*** – Now, the Verilog code of the module needs to be written in the generated file. The structural code of the Decoder module is written below and can be copied to the generated file.

```
1  `timescale 1ns / 1ps
2
3  module Decoder_structural(
4      input e,
5      input a,
6      input b,
7      input c,
8      output d0,
9      output d1,
10     output d2,
11     output d3,
12     output d4,
13     output d5,
14     output d6,
15     output d7
16     );
17
18     // Defining wires for not signals
19     wire a_not, b_not, c_not;
20
21     // Instantiating Not gates as per the schematic
22     not n0 (a_not, a);
23     not n1 (b_not, b);
24     not n2 (c_not, c);
25
26     // Instantiating And gates as per the schematic
27     and g0 (d0, e, a_not, b_not, c_not);
28     and g1 (d1, e, a_not, b_not, c);
29     and g2 (d2, e, a_not, b, c_not);
30     and g3 (d3, e, a_not, b, c);
31     and g4 (d4, e, a, b_not, c_not);
32     and g5 (d5, e, a, b_not, c);
33     and g6 (d6, e, a, b, c_not);
34     and g7 (d7, e, a, b, c);
35
36 endmodule
```

***Step 4 (Creating a Verilog Testbench)*** – A testbench is used to verify the correctness of the circuit by giving the stimulus (various input combinations) and checking for the output. For smaller circuits, it is good to perform "*exhaustive*" circuit testing, i.e., testing the circuit for all possible input combinations and this will be done for both circuits in this lab. To open a Verilog testbench file, go to File -> Add Sources. In the wizard, select Add or create simulation sources option and click Next. Now, click on the Create File button and give any File name. (A good practice is to name the testbench files as tb_<Module_name> and since the module name for this part is Decoder_structural, the testbench can be named as tb_Decoder_structural). Click on Finish to open the Define Module window. No port definitions are required to be defined for the testbench as the module to be tested will be instantiated inside it, and hence, just click on Ok to create the testbench file. Double click on the file (which will be present under the Simulation Sources tab) to open it. The steps to create a new Verilog testbench file are demonstrated in the figures below.

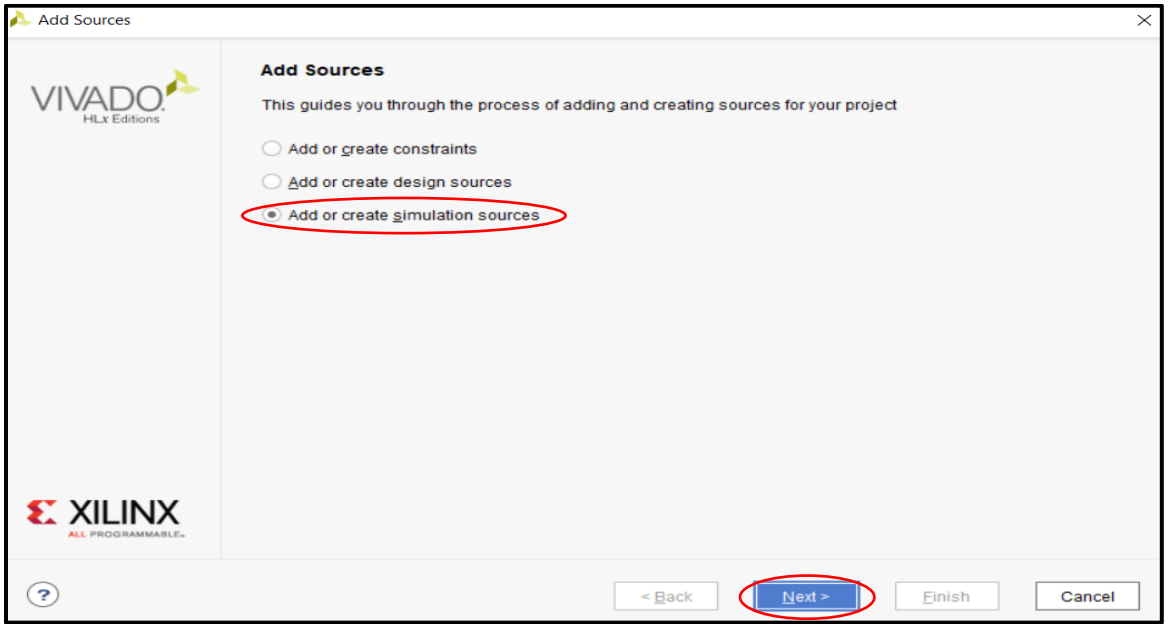


**Figure 15.** Creating a new Verilog testbench – Step 1

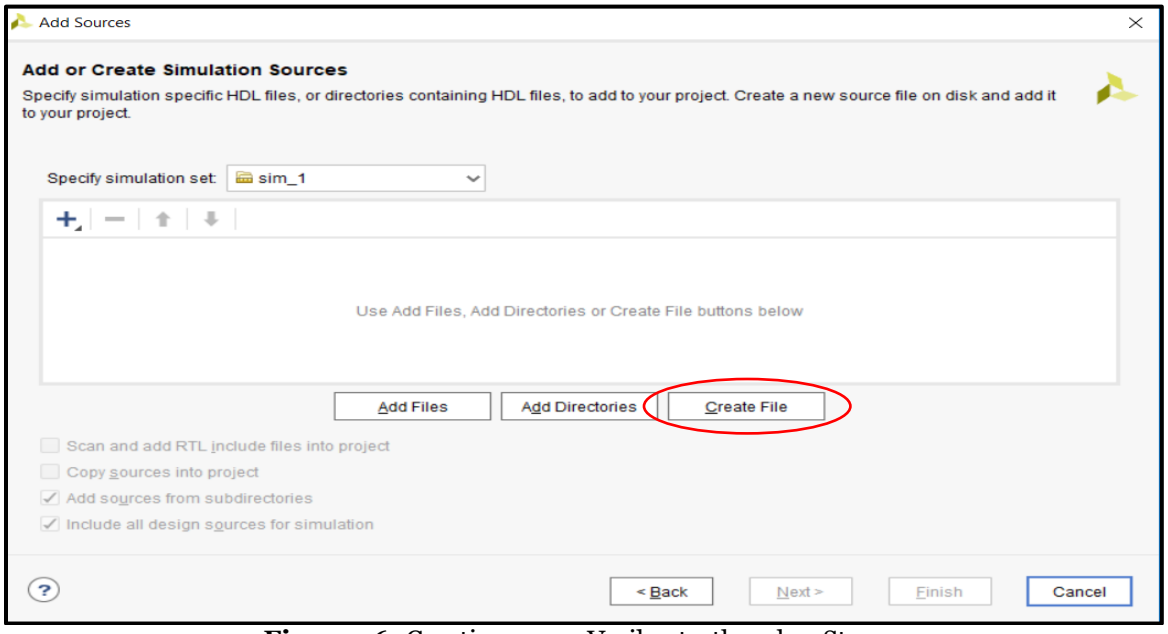


**Figure 16.** Creating a new Verilog testbench – Step 2

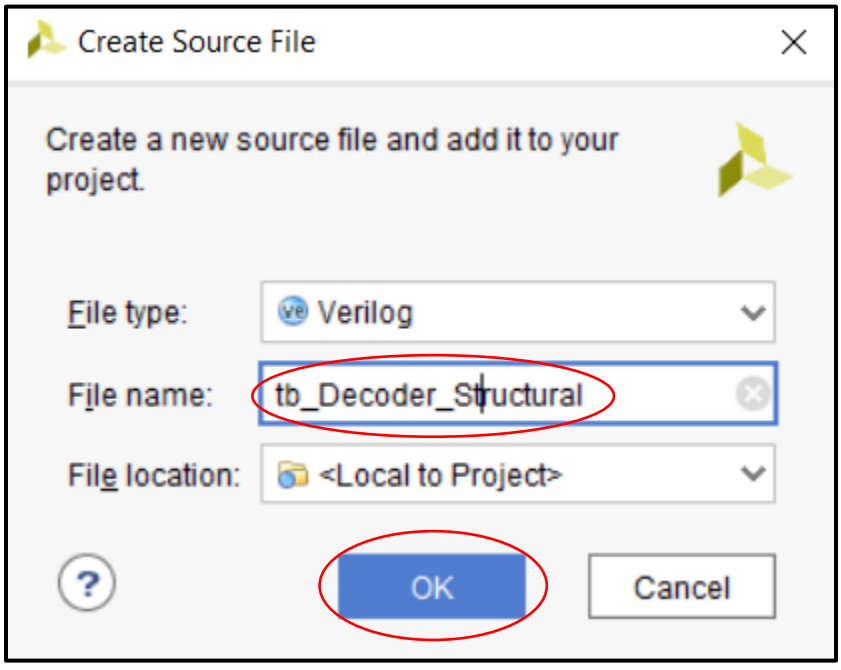


**Figure 17.** Creating a new Verilog testbench – Step 3

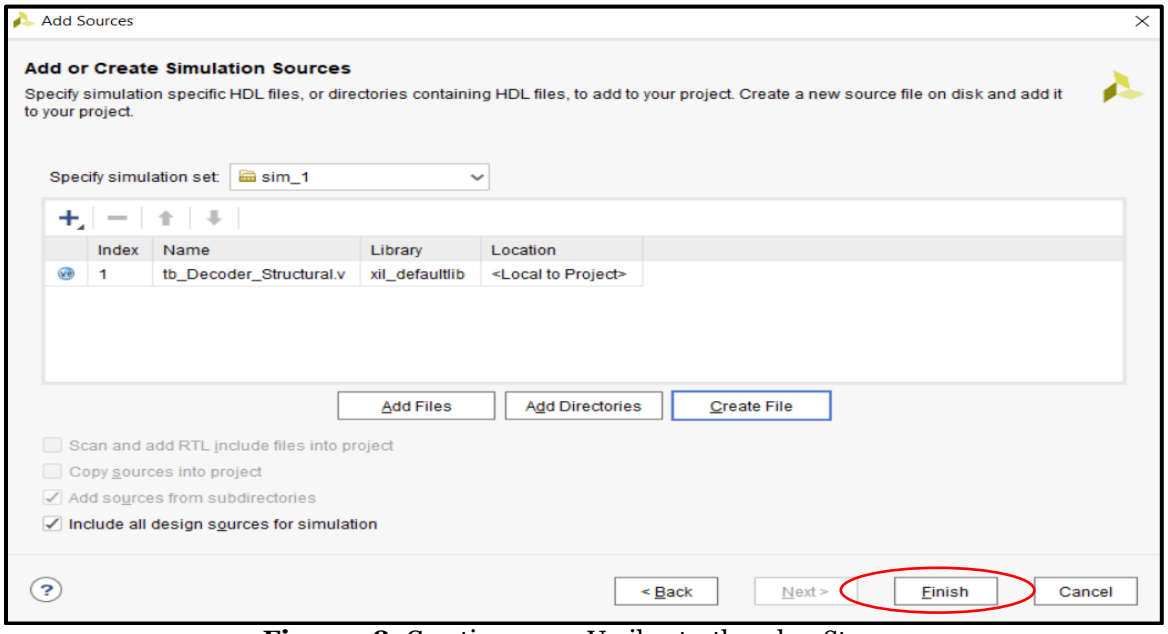


**Figure 18.** Creating a new Verilog testbench – Step 4

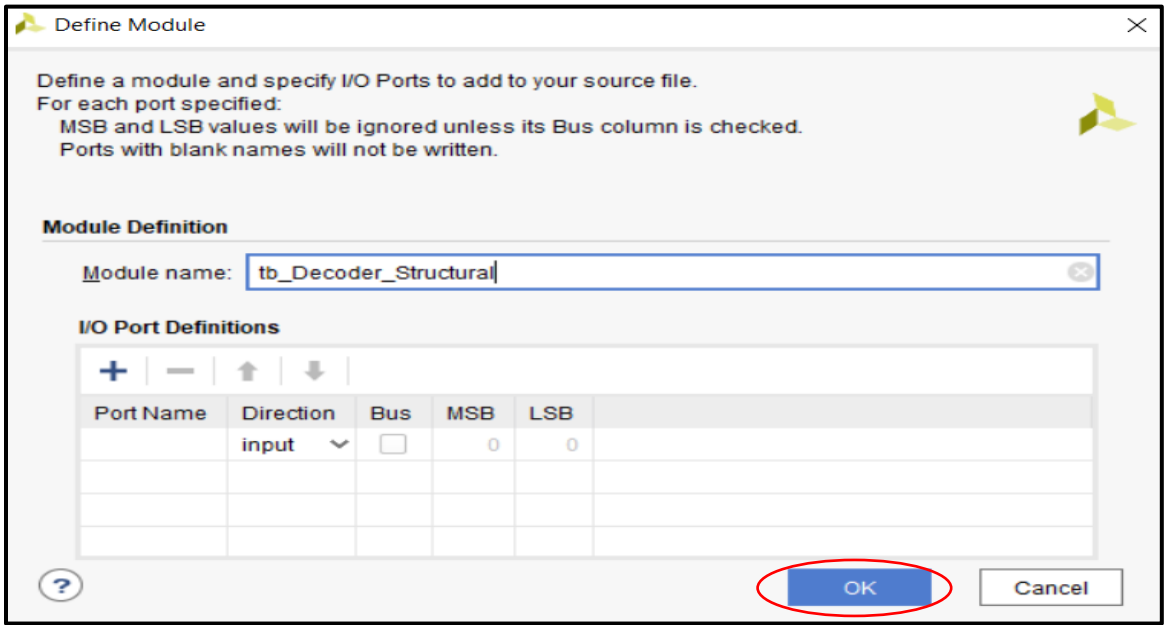


**Figure 19.** Creating a new Verilog testbench – Step 5

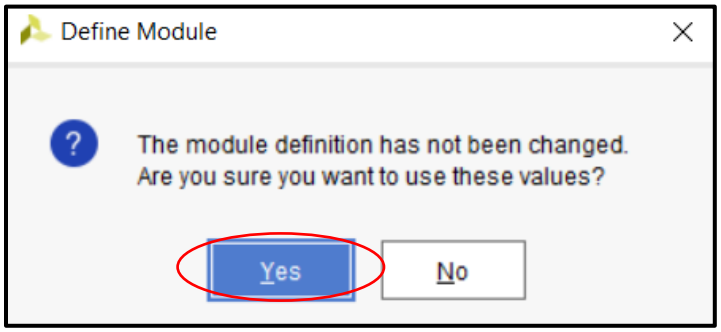


**Figure 20.** Creating a new Verilog testbench – Step 6

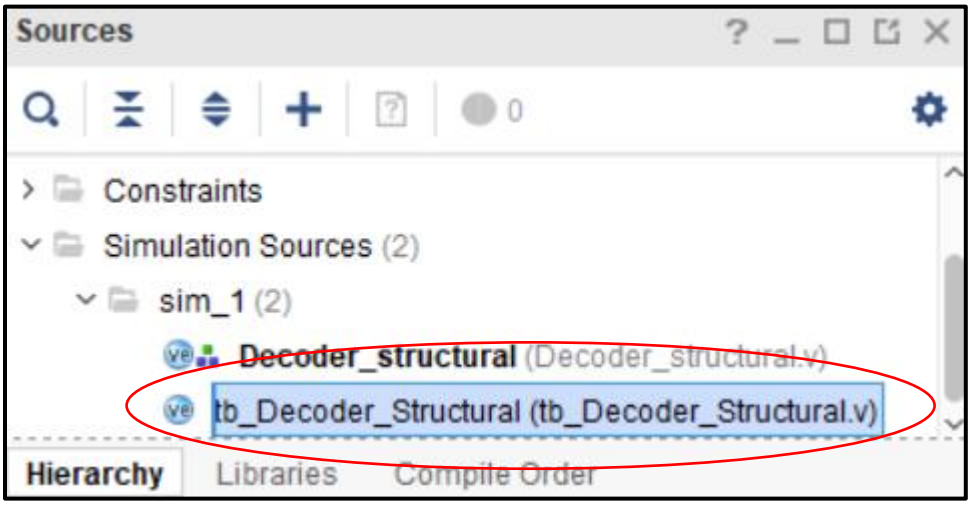


**Figure 21.** Newly created Verilog testbench – appearing in Simulation Sources

***Step 5 (Writing Verilog Testbench)*** – Now, the Verilog testbench needs to be written for the design verification. The code of the testbench is written below and can be copied to the generated file.

```
1  `timescale 1ns / 1ps
2
3  module tb_Decoder_structural;
4
5      // Inputs to be defined as registers
6      reg e;
7      reg a;
8      reg b;
9      reg c;
10
```

```
    // Outputs to be defined as wires
    wire d0;
    wire d1;
    wire d2;
    wire d3;
    wire d4;
    wire d5;
    wire d6;
    wire d7;

    // Instantiate the Unit Under Test (UUT)
    Decoder_structural uut (
        .e(e),
        .a(a),
        .b(b),
        .c(c),
        .d0(d0),
        .d1(d1),
        .d2(d2),
        .d3(d3),
        .d4(d4),
        .d5(d5),
        .d6(d6),
        .d7(d7)
    );

    initial begin
        // Initialize Inputs
        e = 0;
        a = 0;
        b = 0;
        c = 0;

        // Wait 50 ns for global reset to finish
        #50;

```

```
47          // Stimulus - All input combinations followed by some wait time to observe the o/p
48
49          e = 0;
50          a = 0;
51          b = 0;
52          c = 0;
53          $display ("TC01");
54          if ( d0 != 1'b0 ) $display  ("Result is wrong");
55
56          e = 0;
57          a = 0;
58          b = 0;
59          c = 1;
60          #50;
61          $display ("TC02");
62          if ( d1 != 1'b0 ) $display  ("Result is wrong");
63
64          e = 0;
65          a = 0;
66          b = 1;
67          c = 0;
68          #50;
69          $display ("TC03");
70          if ( d2 != 1'b0 ) $display  ("Result is wrong");
71
72          e = 0;
73          a = 0;
74          b = 1;
75          c = 1;
76          #50;
77          $display ("TC04");
78          if ( d3 != 1'b0 ) $display  ("Result is wrong");
79
80          e = 0;
81          a = 1;
82          b = 0;
83          c = 0;
84          #50;
85          $display ("TC05");
86          if ( d4 != 1'b0 ) $display  ("Result is wrong");
87
88          e = 0;
89          a = 1;
90          b = 0;
91          c = 1;
92          #50;
93          $display ("TC06");
94          if ( d5 != 1'b0 ) $display  ("Result is wrong");
95
96          e = 0;
97          a = 1;
98          b = 1;
99          c = 0;
100         #50;
101         $display ("TC07");
102         if ( d6 != 1'b0 ) $display  ("Result is wrong");
103
```

```
105         a = 1;
106         b = 1;
107         c = 1;
108         #50;
109         $display ("TC08");
110         if ( d7 != 1'b0 ) $display  ("Result is wrong");
111
112         e = 1;
113         a = 0;
114         b = 0;
115         c = 0;
116         #50;
117         $display ("TC11");
118         if ( d0 != 1'b1 ) $display  ("Result is wrong");
119
120         e = 1;
121         a = 0;
122         b = 0;
123         c = 1;
124         #50;
125         $display ("TC12 ");
126         if ( d1 != 1'b1 ) $display  ("Result is wrong");
127
128         e = 1;
129         a = 0;
130         b = 1;
131         c = 0;
132         #50;
133         $display ("TC13 ");
134         if ( d2 != 1'b1 ) $display  ("Result is wrong");
135
136         e = 1;
137         a = 0;
138         b = 1;
139         c = 1;
140         #50;
141         $display ("TC14 ");
142         if ( d3 != 1'b1 ) $display  ("Result is wrong");
143
144         e = 1;
145         a = 1;
146         b = 0;
147         c = 0;
148         #50;
149         $display ("TC15 ");
150         if ( d4 != 1'b1 ) $display  ("Result is wrong");
151
152         e = 1;
153         a = 1;
154         b = 0;
155         c = 1;
156         #50;
157         $display ("TC16 ");
158         if ( d5 != 1'b1 ) $display  ("Result is wrong");
159
```

```
160         e = 1;
161         a = 1;
162         b = 1;
163         c = 0;
164         #50;
165         $display ("TC17 ");
166         if ( d6 != 1'b1 ) $display  ("Result is wrong");
167
168         e = 1;
169         a = 1;
170         b = 1;
171         c = 1;
172         #50;
173         $display ("TC18 ");
174         if ( d7 != 1'b1 ) $display  ("Result is wrong");
175
176         end
177
178 endmodule
```

***Step 6 (Circuit Simulation and Verification)*** – The next step is to simulate the testbench created in the previous step and verify the functionality of the circuit. Select the *Run Simulation* option in the left pane and select Run Behavioral Simulation from the drop-down menu. A simulation waveform window will come up, and the messages in $display statements in the testbench will come on the Tcl console. On the simulation window, click on the Zoom fit icon to fit the simulation waveform on the screen. Other options in the menu can be explored as well. Also, different signals can be put on the waveform, by right clicking on the signal names in the Scope and Objects panes.

***Note*** *–> If your computer has an active anti-virus program, it may be required to turn that off before simulation can be done.*

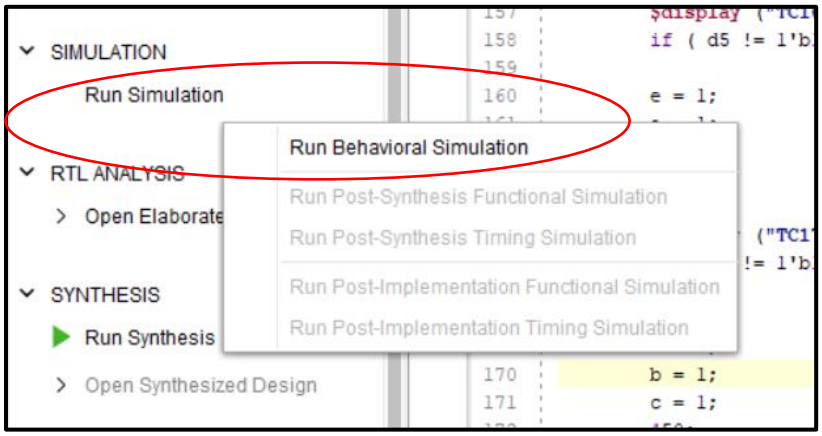


**Figure 22.** Running Simulation

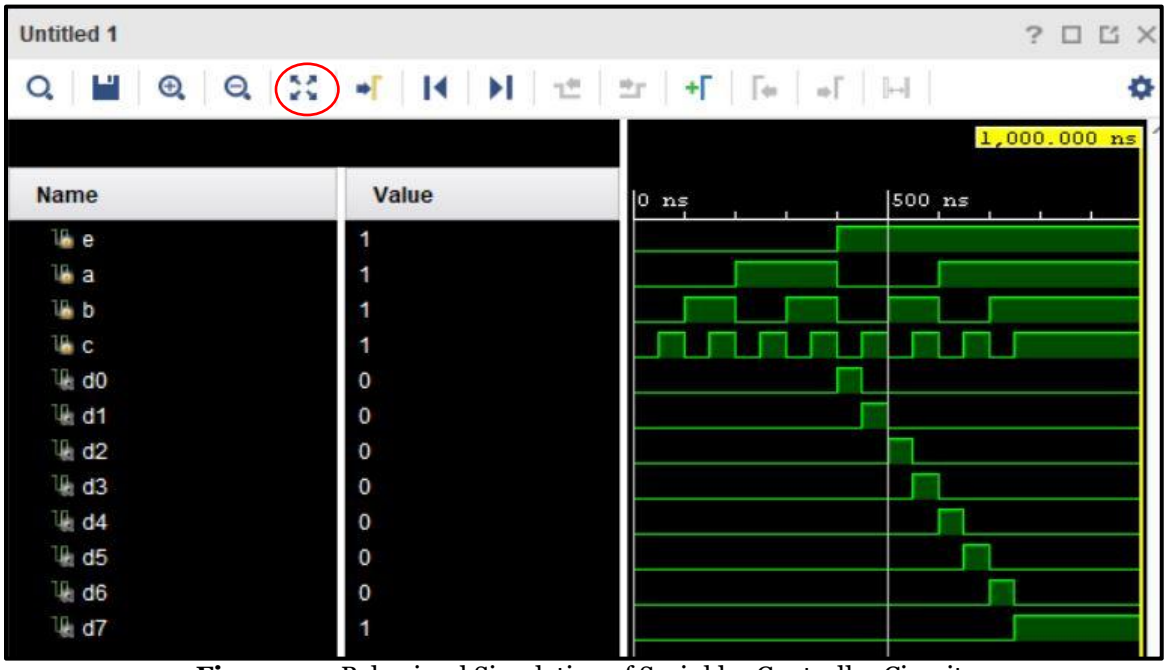


**Figure 23.** Behavioral Simulation of Sprinkler Controller Circuit

The sequence of input and output signals can be re-arranged by pressing and holding the signal line and moving it up or down. Also, signals can be grouped by selecting the signals to be grouped, right clicking and selecting New Group.

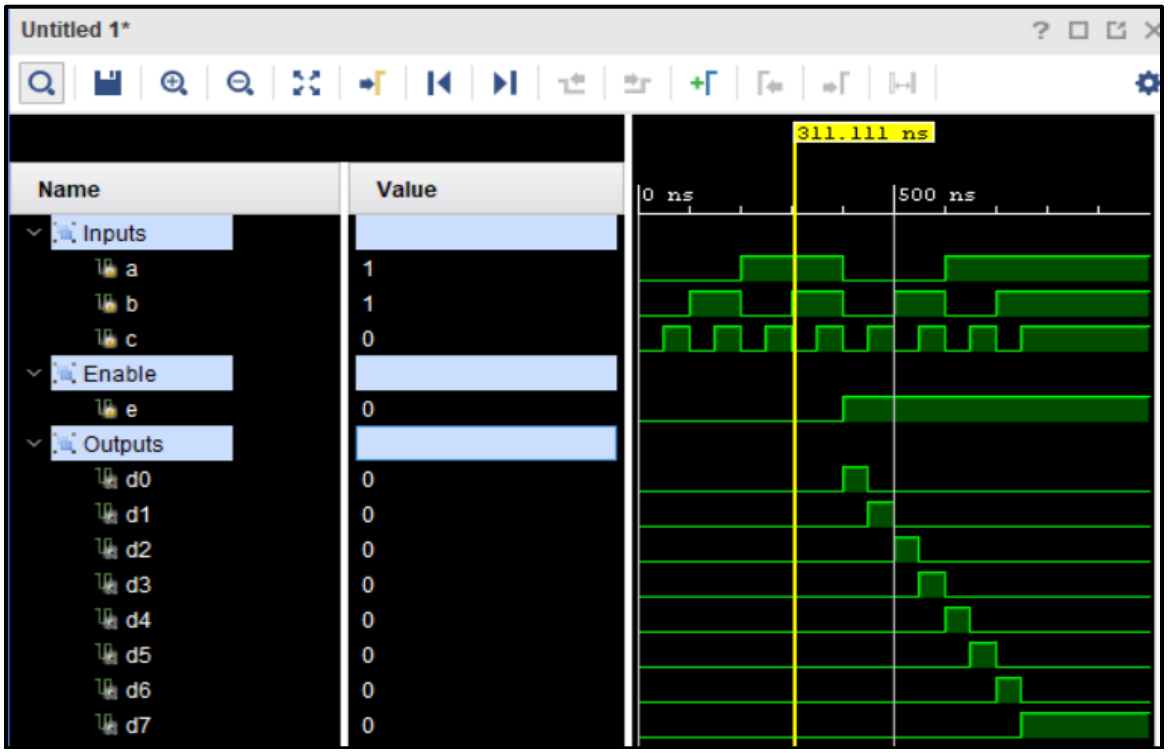


**Figure 24.** Behavioral Simulation Waveform Window with Inputs, Outputs and Enable grouped

It should be noted from the simulation that when the enable is '0', none of the outputs is '1' (meaning that all the valves are closed), but when enable is '1', only one of the outputs is '1' based on the input combination, thus ensuring that only one valve is open at a time as required.

A lot of other customizations like adding cursors, running the simulation for more time, changing signal colors, etc. be done from the waveform window options. It is highly recommended to explore the various options available, as they might be helpful in the future labs. However, for this lab, only the simulation waveforms are required to be submitted.

For closing the simulation, right click on the Simulation tab in the left pane and click on Close Simulation.

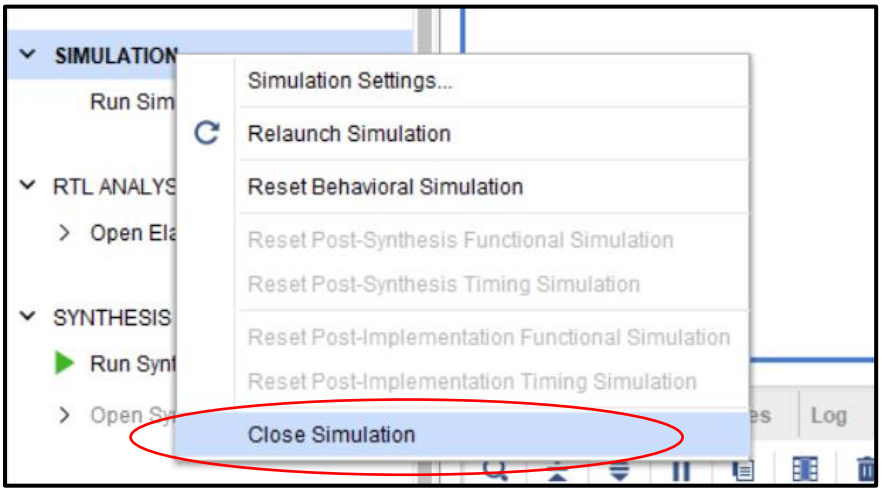


**Figure 25.** Option to close the simulation

## *Part B. Behavioral Modeling*

In this part, the same decoder circuit will be modeled using the behavioral modeling technique. The codes for the module and the testbench are provided below, and the simulation is required to be done following the same steps listed above.

***Note*** *-> It should be noted that the behavioral modeling in this part is done using case statements, which is just one way to model the circuits behaviorally. There are other ways which can be used to model the circuits behaviorally (for example, using "if" statements only).*

### *Verilog Code of the module*

```
`timescale 1ns / 1ps

module Decoder_behavioral(
    input e,
    input a,
    input b,
    input c,
    output reg d0,
    output reg d1,
    output reg d2,
    output reg d3,
    output reg d4,
    output reg d5,
    output reg d6,
    output reg d7
    );

    always @(e,a,b,c)
    begin

    d0=1'b0; d1=1'b0; d2=1'b0; d3=1'b0; d4=1'b0; d5=1'b0; d6=1'b0; d7=1'b0;

    if (e==1'b1)
        case ({a,b,c})
        3'b000 : d0 = 1'b1;
        3'b001 : d1 = 1'b1;
        3'b010 : d2 = 1'b1;
        3'b011 : d3 = 1'b1;
        3'b100 : d4 = 1'b1;
        3'b101 : d5 = 1'b1;
        3'b110 : d6 = 1'b1;
        3'b111 : d7 = 1'b1;
        default :  begin
                   d0=1'b0; d1=1'b0; d2=1'b0; d3=1'b0; d4=1'b0; d5=1'b0; d6=1'b0; d7=1'b0;
                   end
        endcase
    end

endmodule
```

### *Verilog Code of the testbench*

Since the functionality of the decoder is the same no matter how it is modeled (structurally or behaviorally), the same set of stimuli developed in Part A can be used to test the behaviorally modeled testbench as well, just by modifying the testbench name and the instantiation of the UUT. Also, we expect to obtain the same simulation waveforms as Part A, since the functionality of the circuit remains the same.

```
1  `timescale 1ns / 1ps
2
3  module tb_Decoder_behavioral;
4
5      // Inputs to be defined as registers
6      reg e;
7      reg a;
8      reg b;
9      reg c;
10
11     // Outputs to be defined as wires
12     wire d0;
13     wire d1;
14     wire d2;
15     wire d3;
16     wire d4;
17     wire d5;
18     wire d6;
19     wire d7;
20
21     // Instantiate the Unit Under Test (UUT)
22     Decoder_behavioral uut (
23         .e(e),
24         .a(a),
25         .b(b),
26         .c(c),
27         .d0(d0),
28         .d1(d1),
29         .d2(d2),
30         .d3(d3),
31         .d4(d4),
32         .d5(d5),
```

```
33          .d6(d6),
34          .d7(d7)
35      );
36
37      initial begin
38          // Initialize Inputs
39          e = 0;
40          a = 0;
41          b = 0;
42          c = 0;
43
44          // Wait 50 ns for global reset to finish
45          #50;
46
```

```
47          // Stimulus - All input combinations followed by some wait time to observe the o/p
48
49          e = 0;
50          a = 0;
51          b = 0;
52          c = 0;
53          $display ("TC01");
54          if ( d0 != 1'b0 ) $display  ("Result is wrong");
55
56          e = 0;
57          a = 0;
58          b = 0;
59          c = 1;
60          #50;
61          $display ("TC02");
62          if ( d1 != 1'b0 ) $display  ("Result is wrong");
63
64          e = 0;
65          a = 0;
66          b = 1;
67          c = 0;
68          #50;
69          $display ("TC03");
70          if ( d2 != 1'b0 ) $display  ("Result is wrong");
71
72          e = 0;
73          a = 0;
74          b = 1;
75          c = 1;
76          #50;
77          $display ("TC04");
78          if ( d3 != 1'b0 ) $display  ("Result is wrong");
```

```
79
80          e = 0;
81          a = 1;
82          b = 0;
83          c = 0;
84          #50;
85          $display ("TC05");
86          if ( d4 != 1'b0 ) $display  ("Result is wrong");
87
88          e = 0;
89          a = 1;
90          b = 0;
91          c = 1;
92          #50;
93          $display ("TC06");
94          if ( d5 != 1'b0 ) $display  ("Result is wrong");
95
96          e = 0;
97          a = 1;
98          b = 1;
99          c = 0;
100         #50;
101         $display ("TC07");
102         if ( d6 != 1'b0 ) $display  ("Result is wrong");
103
105          a = 1;
106          b = 1;
107          c = 1;
108          #50;
109          $display ("TC08");
110          if ( d7 != 1'b0 ) $display  ("Result is wrong");
111
112          e = 1;
113          a = 0;
114          b = 0;
115          c = 0;
116          #50;
117          $display ("TC11");
118          if ( d0 != 1'b1 ) $display  ("Result is wrong");
119
120          e = 1;
121          a = 0;
122          b = 0;
123          c = 1;
124          #50;
125          $display ("TC12 ");
126          if ( d1 != 1'b1 ) $display  ("Result is wrong");
127
128          e = 1;
129          a = 0;
130          b = 1;
131          c = 0;
132          #50;
133          $display ("TC13 ");
134          if ( d2 != 1'b1 ) $display  ("Result is wrong");
135
```

```
136          e = 1;
137          a = 0;
138          b = 1;
139          c = 1;
140          #50;
141          $display ("TC14 ");
142          if ( d3 != 1'b1 ) $display  ("Result is wrong");
143
144          e = 1;
145          a = 1;
146          b = 0;
147          c = 0;
148          #50;
149          $display ("TC15 ");
150          if ( d4 != 1'b1 ) $display  ("Result is wrong");
151
152          e = 1;
153          a = 1;
154          b = 0;
155          c = 1;
156          #50;
157          $display ("TC16 ");
158          if ( d5 != 1'b1 ) $display  ("Result is wrong");
159
160         e = 1;
161         a = 1;
162         b = 1;
163         c = 0;
164         #50;
165         $display ("TC17 ");
166         if ( d6 != 1'b1 ) $display  ("Result is wrong");
167
168         e = 1;
169         a = 1;
170         b = 1;
171         c = 1;
172         #50;
173         $display ("TC18 ");
174         if ( d7 != 1'b1 ) $display  ("Result is wrong");
175
176         end
177
178 endmodule
```

***Note*** *-> It must be noted that there is no need to create a new project, and the files can be created in the same project. However, the simulation is always done for the top module, and hence, the testbench file of the behavioral model should now be Set as Top by right clicking on it before proceeding with the simulation.*

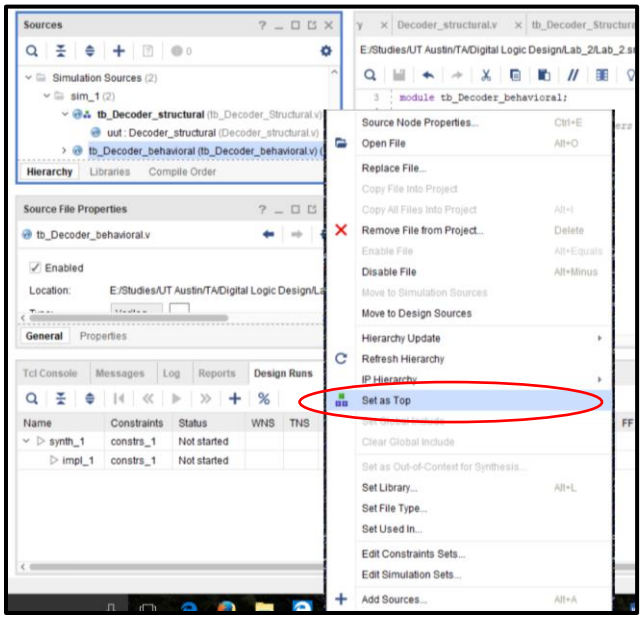


**Figure 26.** Setting a module as Top Module

## Conclusion for Part 1

We have gone through the whole cycle of system design, analysis and circuit logic behavioral verification.

# PART 2. Design of Computer Data Bus[4]

In this assignment, we will design a 4 x 1 multiplexer that will control the flow of data in a single wire data bus and study its basic properties. This technology allows to accomplish, for example, partial serial communication with multiple peripheral devices using just one output pin of a microcontroller.

## Specification

Design a 1-wire data bus controller that performs the following function:

[4] This part must be completed in groups of 2.

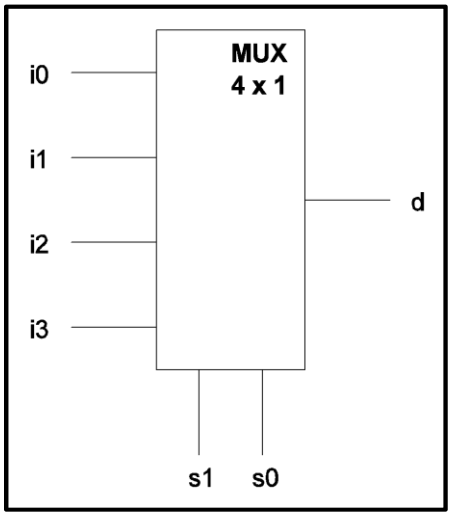


**Figure 27.** Single-wire data bus multiplexer

Inputs s0,s1 control which of the input data i0, i1, i2, i3 is present in the output d, where the binary s1,s0 represents the input pin number.

*Example*: s1,s0 = 1,0 indicates that data in the input line i2 appear in the output d.

Design a logic circuit that will perform this function and verify the logic functionality using Xilinx Vivado software environment. As done in Part 1, the design needs to be modeled structurally as well as behaviorally and both the modules need to be tested using the testbenches.

## Demonstration and Lab Report

During the lab hour, all the simulation waveforms need to be presented to the TAs with the correct explanation of why they are correct.

For Part 1, all the simulation waveforms should be submitted as part of the lab report. For Part 2, the truth table, algebraic expression of the logic function, logic circuit schematic, Verilog codes for modules and testbenches for both structural and behavioral modeling, and simulation waveforms for both structural and behavioral modeling should be submitted as part of the lab report.