Professor Bhavnagarwala Anik Barua

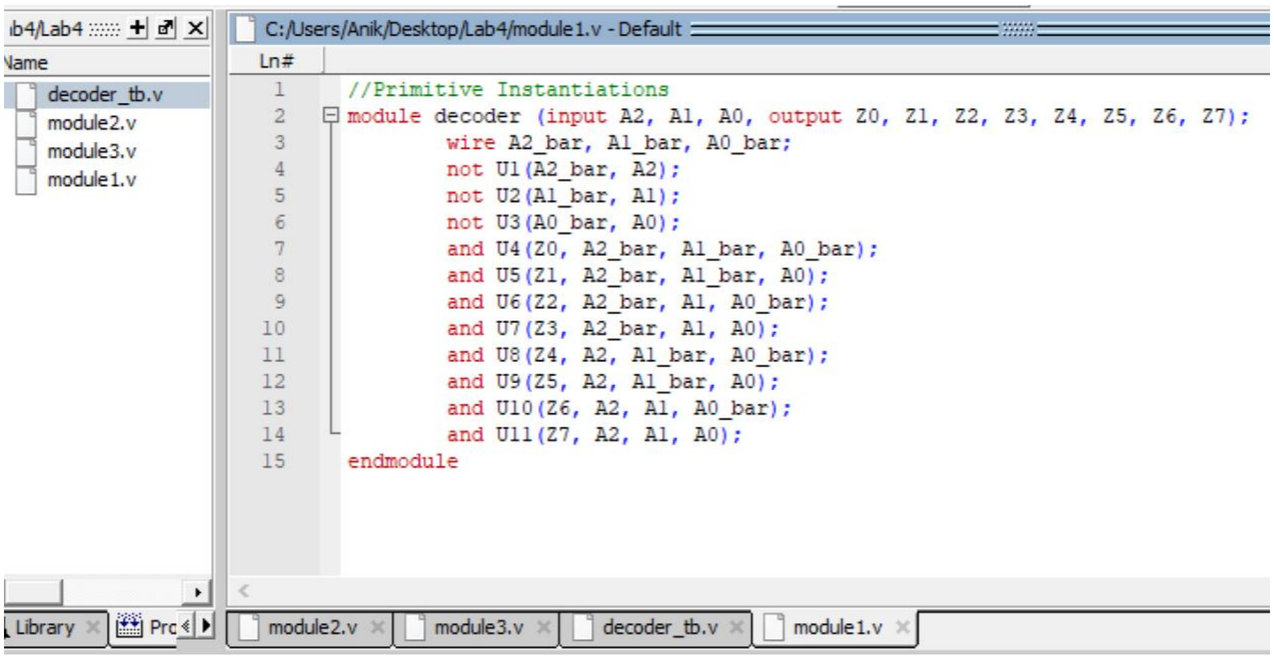
CS 2204 - Section C 10-01-2021

Lab 4: Building Basic Verilog Coding Skills

Today’s Lab is about writing Verilog Module for a 3-input decoder using primitive instantiations, Continuous assignment statements and Conditional statements. Then we will write a testbench to simulate the 3 modules and verify using the waveforms. Verilog is a hardware description language for describing hardware from transistor level to behavioral. Verilog description consist of modules which is basic unit of hierarchy that describes boundaries (module to endmodule), input and output and behavioral codes (conditional statements, continuous assignments) etc.

Primitive instantiations:

Verilog basic logic gates are called primitives and to describe the components using primitives, we use the constructor called Primitive instantiation. Examples of primitives - and, not, wire, input, output etc. that we will be using for our decoder.



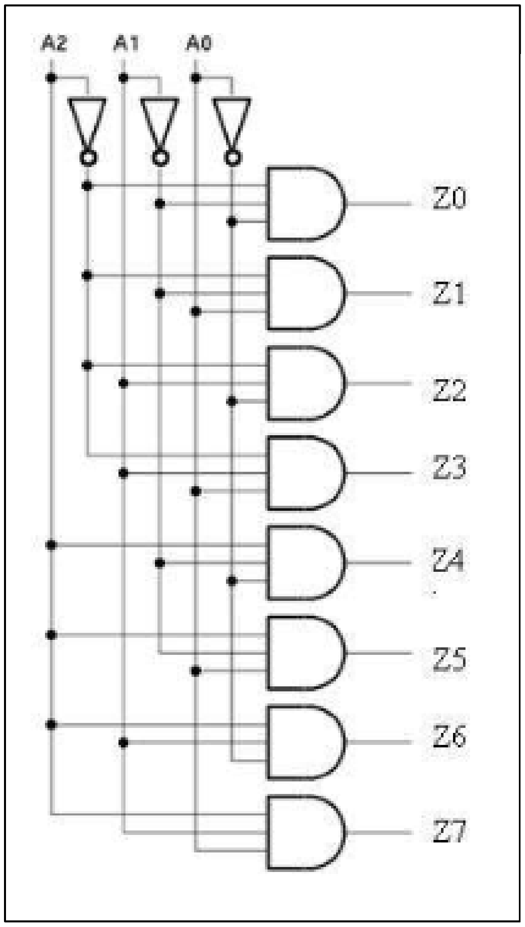
```
//Primitive Instantiations
module decoder (input A2, A1, A0, output Z0, Z1, Z2, Z3, Z4, Z5, Z6, Z7);
        wire A2_bar, A1_bar, A0_bar;
        not U1(A2_bar, A2);
        not U2(A1_bar, A1);
        not U3(A0_bar, A0);
        and U4(Z0, A2_bar, A1_bar, A0_bar);
        and U5(Z1, A2_bar, A1_bar, A0);
        and U6(Z2, A2_bar, A1, A0_bar);
        and U7(Z3, A2_bar, A1, A0);
        and U8(Z4, A2, A1_bar, A0_bar);
        and U9(Z5, A2, A1_bar, A0);
        and U10(Z6, A2, A1, A0_bar);
        and U11(Z7, A2, A1, A0);
endmodule
```

A basic module starts with boundaries (module to endmodule). All our code will go inside that boundary. Now we will put a module name based on our choice but naming something what the module does is preferable. Next inside the bracket we first right the inputs and outputs from the given logic schematic. Then we declare wire and reg for variables (based on schematic). We use wire for connecting different elements and reg (register) for storing data. Then based on logic schematic we use and, not, or gate and the syntax is -

not <u>name</u> (<u>input</u> , <u>output)</u>

and <u>name</u> (<u>input</u> , <u>output)</u>

We get the input and output from the logic schematic for example - for output Z1 we have input A2_bar (A2 inverter), A1_bar, A0. Similarly we can create it for all output.



$$Z0 = \overline{A_2} \cdot \overline{A_1} \cdot \overline{A_0}$$

$$Z1 = \overline{A_2} \cdot \overline{A_1} \cdot A_0$$

$$Z2 = \overline{A_2} \cdot A_1 \cdot \overline{A_0}$$

$$Z3 = \overline{A_2} \cdot A_1 \cdot A_0$$

$$Z4 = A_2 \cdot \overline{A_1} \cdot \overline{A_0}$$

$$Z5 = A_2 \cdot \overline{A_1} \cdot A_0$$

$$Z6 = A_2 \cdot A_1 \cdot \overline{A_0}$$

$$Z7 = A_2 \cdot A_1 \cdot A_0$$

Continuous assignment statements -

In continuous assignments, we use boolean expressions to describe the components. We use the assign statements for assigning results to the expressions for various outputs based on our input.

Syntax -

assign Z0 = (~A2 & ~A1 & ~A0)   (~ means not)

Conditional operators -

In Verilog, we can use conditional expressions to assign values to output. It works like an if-then-else behavior of a software and very useful for describing complex functions.

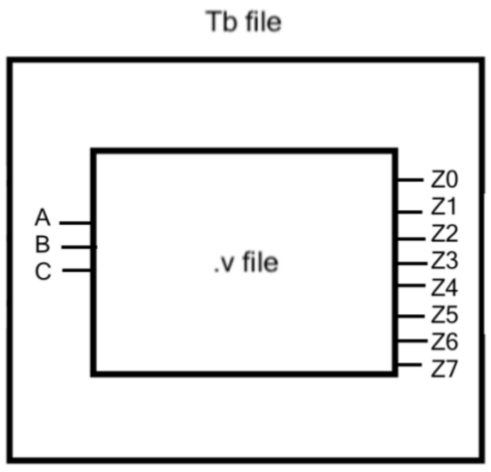
Syntax -

assign Z0 = (~A2 & ~A1 & ~A0) ? 1b'1 : 1'b0

This means if the inside bracket condition is true then assign 1 to Z0, if false we assign 0 to Z0. To write binary number in Verilog, we first write the number of bit, let's say 1 then 'b' for binary and the ' sign followed by the binary number.

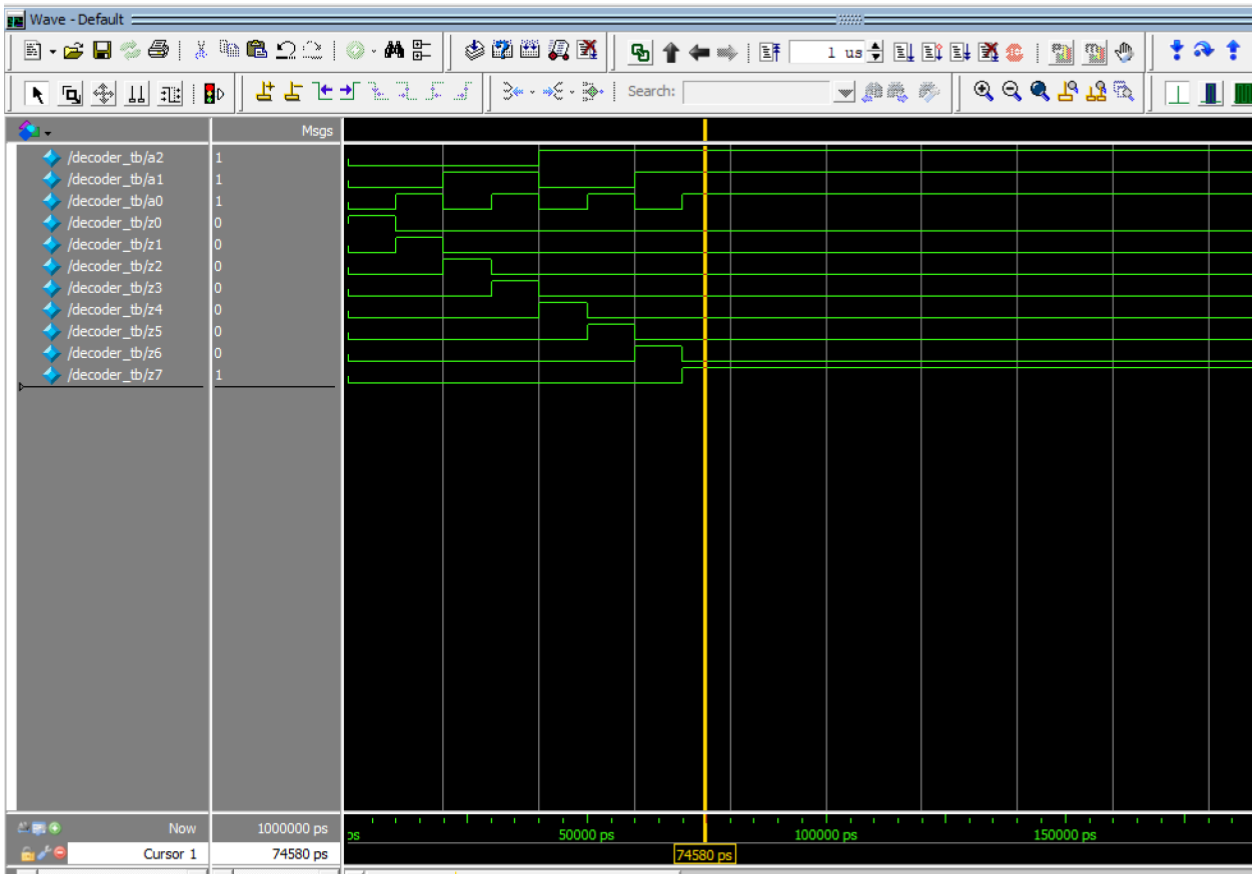
Testbench -

We use testbench to simulate our modules to make sure they are working correctly and we can verify it using the waveform. In our test bench files, we don't have any additional input or output.



But we create a temporary reg for inputs and temporary wire for outputs that will be connected to source file input and output. We force value the input and we get output based on the source file. To test a module, we write the module name followed by 'dut' and inside bracket we write all the input and outputs. Then we start 'initial begin' that follows by the 'end' keyword, and inside we write the time delay (#10) for each force value input. The force value for input is basically the truth table like if we have 3 inputs, it will have 2^3 = 8 possible values. Now if we simulate it we will get the waveforms.

Waveform explanations -

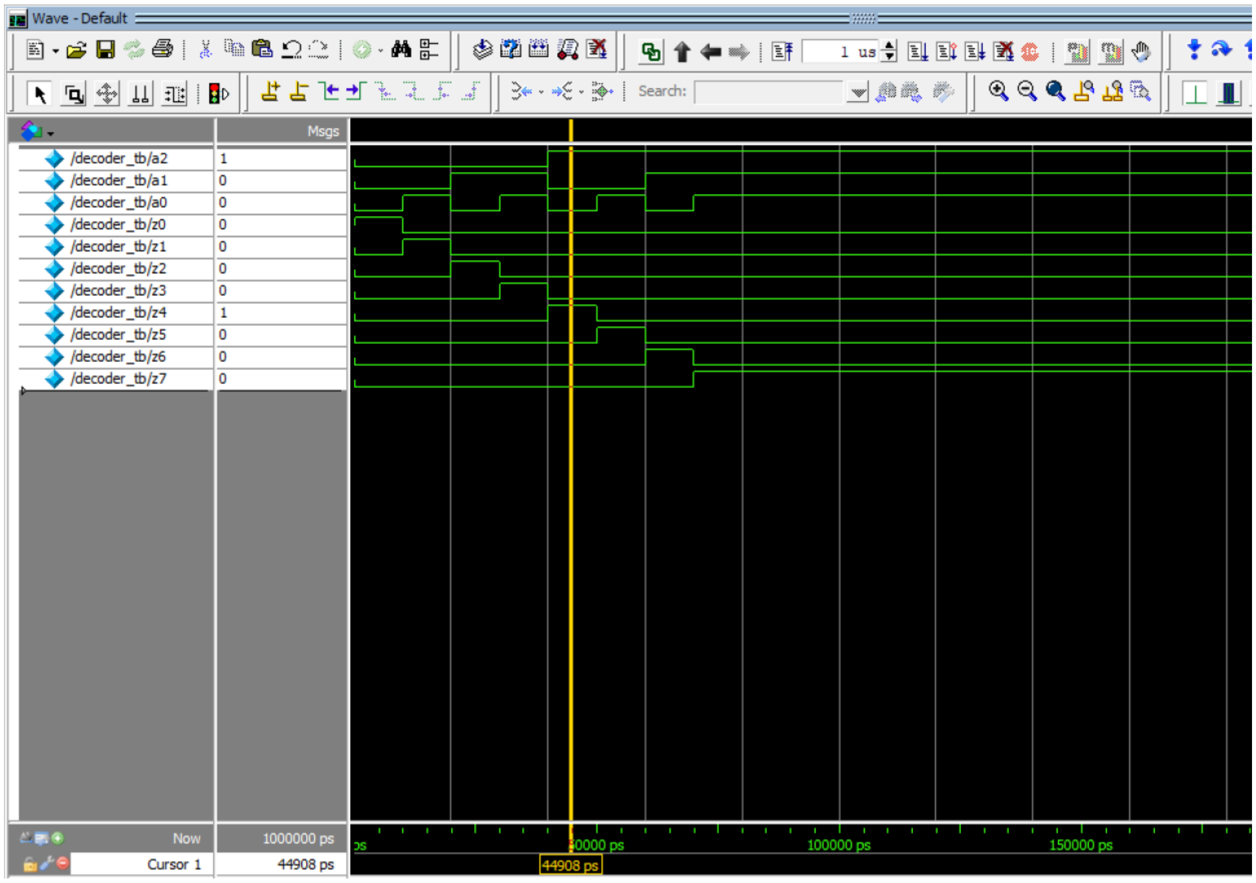


Since we are writing the same decoder in 3 different ways, our waveforms will be the same. This waveform is from decoder 1 (Primitive instantiation). We can see when A2 = 0, A1 = 0 and A0 =

1, we get Z1 = 1. We can see that it true using the expression Z1 = ~A2 & ~A1 & A0 where we get 1 for the A2 = 0, A1 = 0, A0 = 1 values. Another example - if A2 = 0, A1 = 1 and A0 = 1, we see Z3 = 1. It is true because for Z3 expression (~A2 & A1 & A0) we get 1 for A2 = 0, A1 = 1 and A0 = 1. When A2 = 1, A1 = 1 and A0 = 1, we see Z7 is 1 which is also true using Z7 expression A2 & A1 & A0 = 1. So our waveforms shows that decoder for primitive instantiation is correct. And I get the same waveform for decoder 2 (continuous assignment) and decoder 3 (Conditional operator).

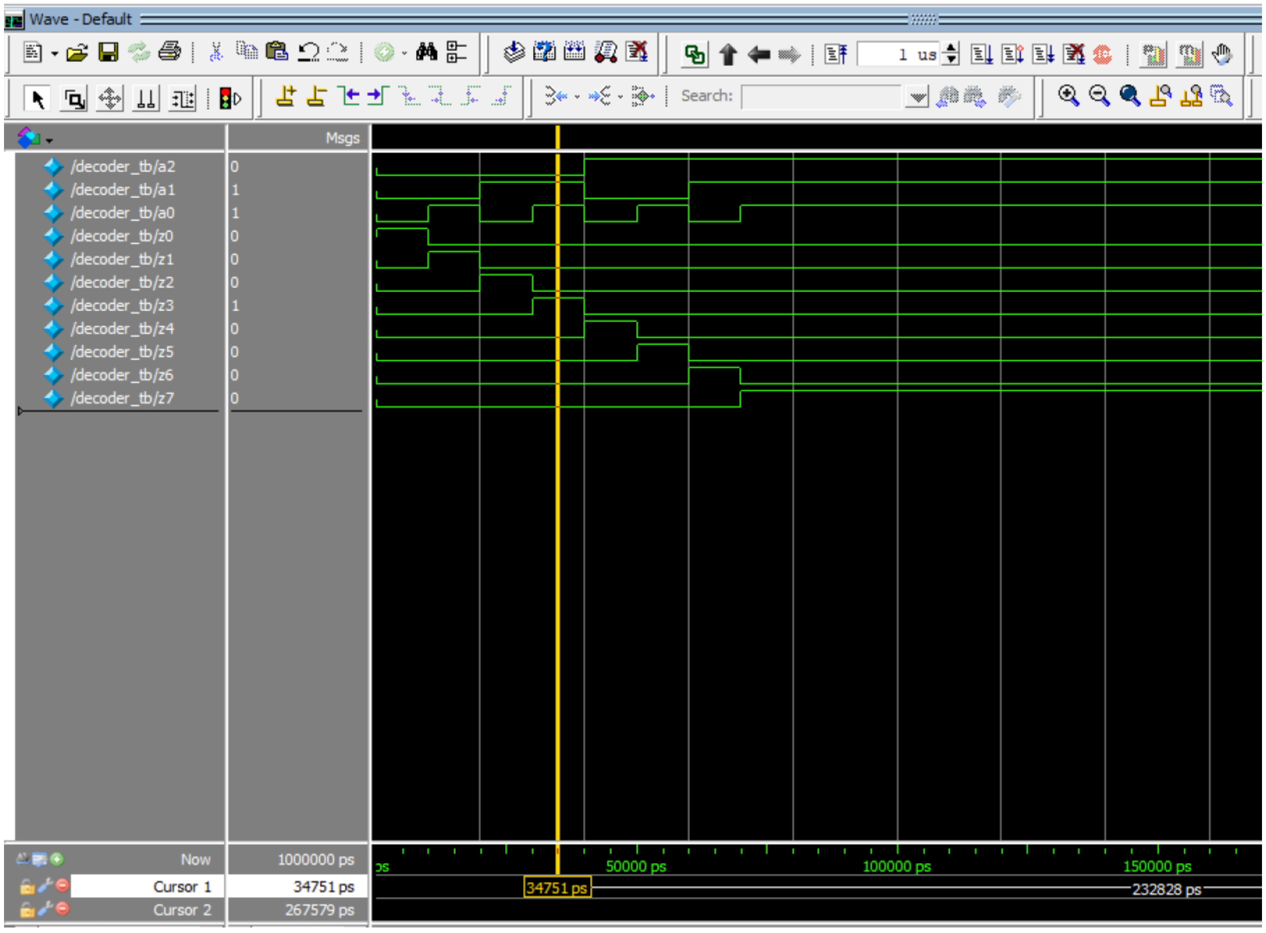
Screenshots of waveforms -

<u>Decoder 2 (continuous assignment)</u>

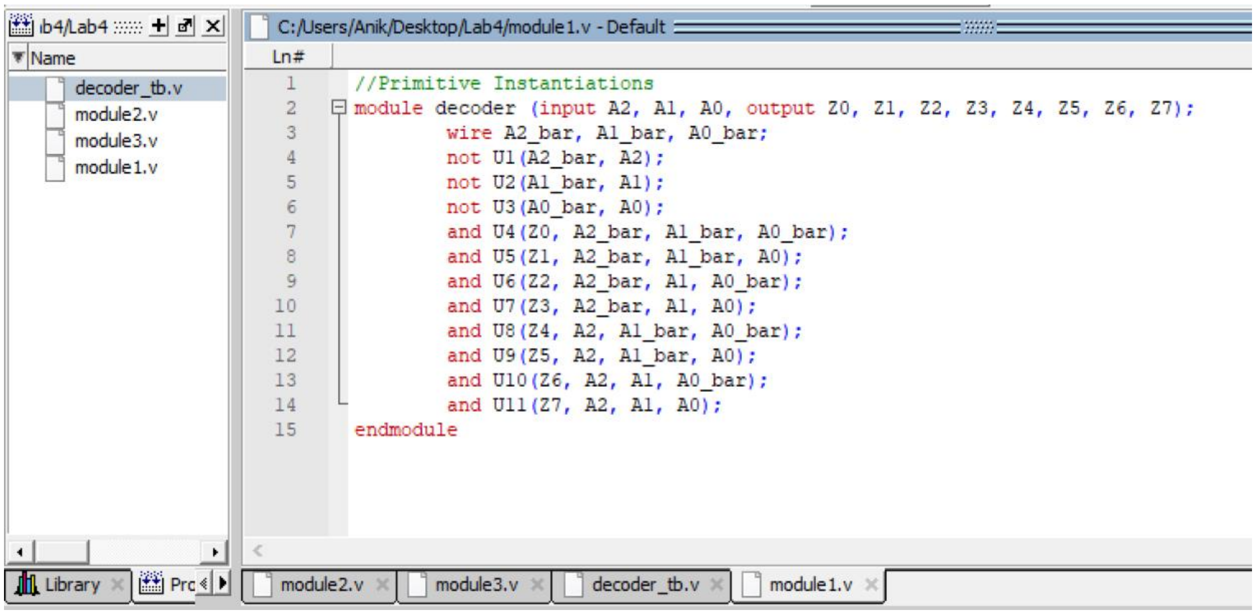


	Msgs
/decoder_tb/a2	1
/decoder_tb/a1	0
/decoder_tb/a0	0
/decoder_tb/z0	0
/decoder_tb/z1	0
/decoder_tb/z2	0
/decoder_tb/z3	0
/decoder_tb/z4	1
/decoder_tb/z5	0
/decoder_tb/z6	0
/decoder_tb/z7	0
Now	1000000 ps
Cursor 1	44908 ps

Decoder 3 (Conditional operator)

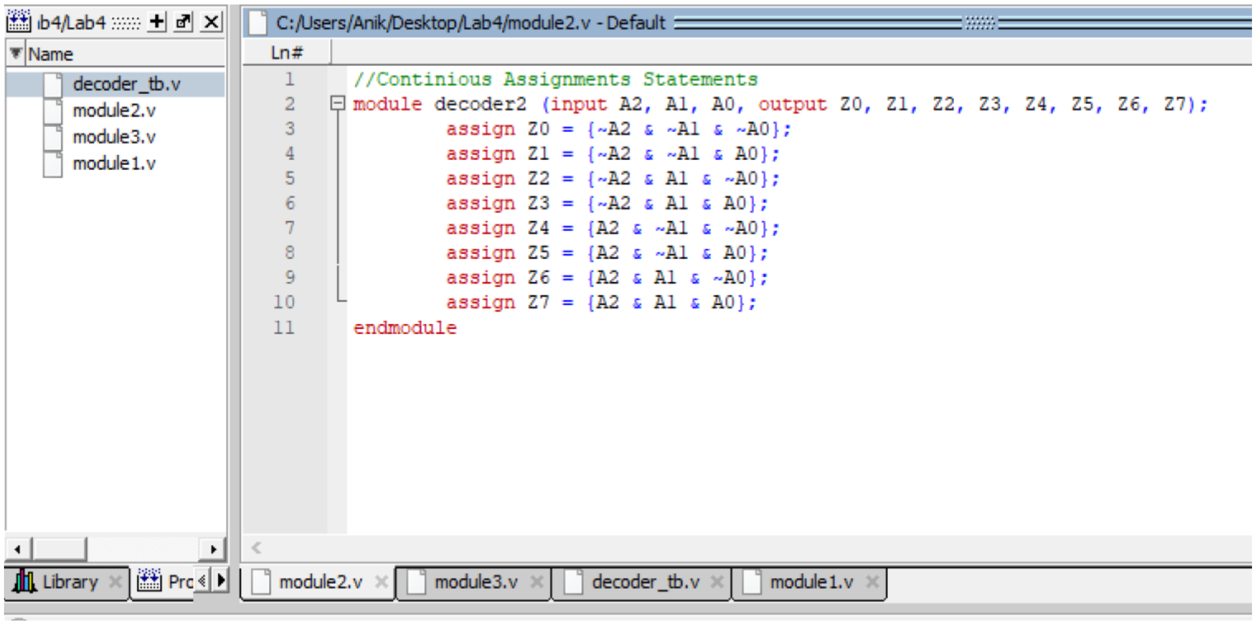


Module for decoder 1 (primitive instantiations)



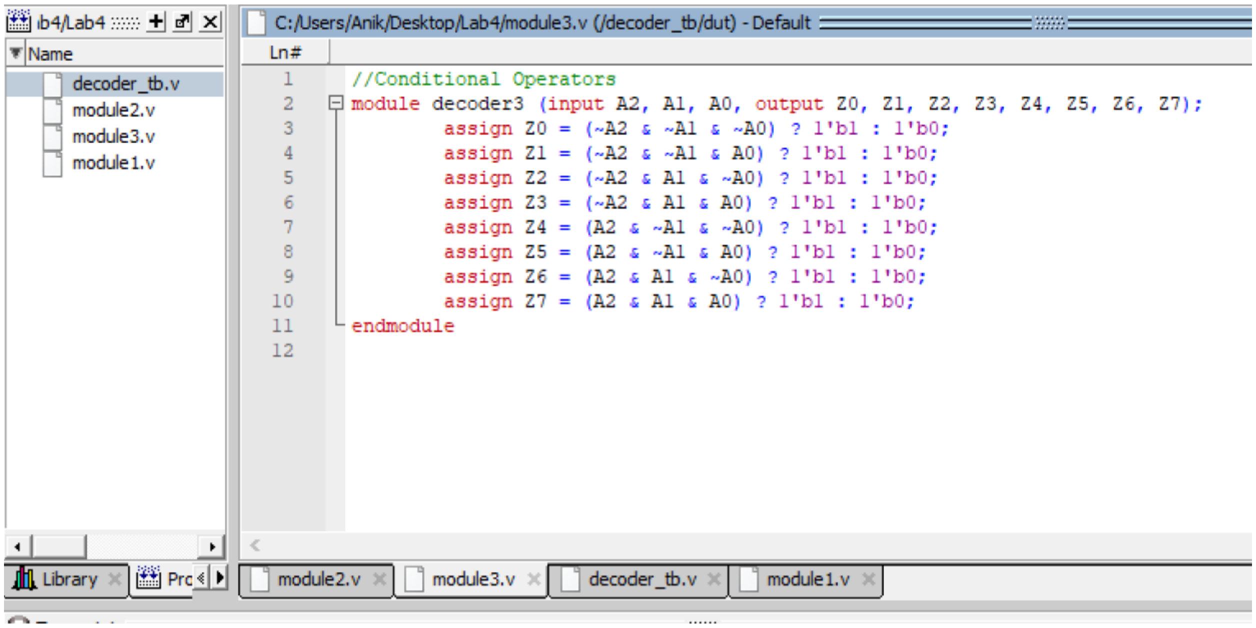
```
//Primitive Instantiations
module decoder (input A2, A1, A0, output Z0, Z1, Z2, Z3, Z4, Z5, Z6, Z7);
        wire A2_bar, A1_bar, A0_bar;
        not U1(A2_bar, A2);
        not U2(A1_bar, A1);
        not U3(A0_bar, A0);
        and U4(Z0, A2_bar, A1_bar, A0_bar);
        and U5(Z1, A2_bar, A1_bar, A0);
        and U6(Z2, A2_bar, A1, A0_bar);
        and U7(Z3, A2_bar, A1, A0);
        and U8(Z4, A2, A1_bar, A0_bar);
        and U9(Z5, A2, A1_bar, A0);
        and U10(Z6, A2, A1, A0_bar);
        and U11(Z7, A2, A1, A0);
endmodule
```

Module for decoder 2 (continuous assignment)



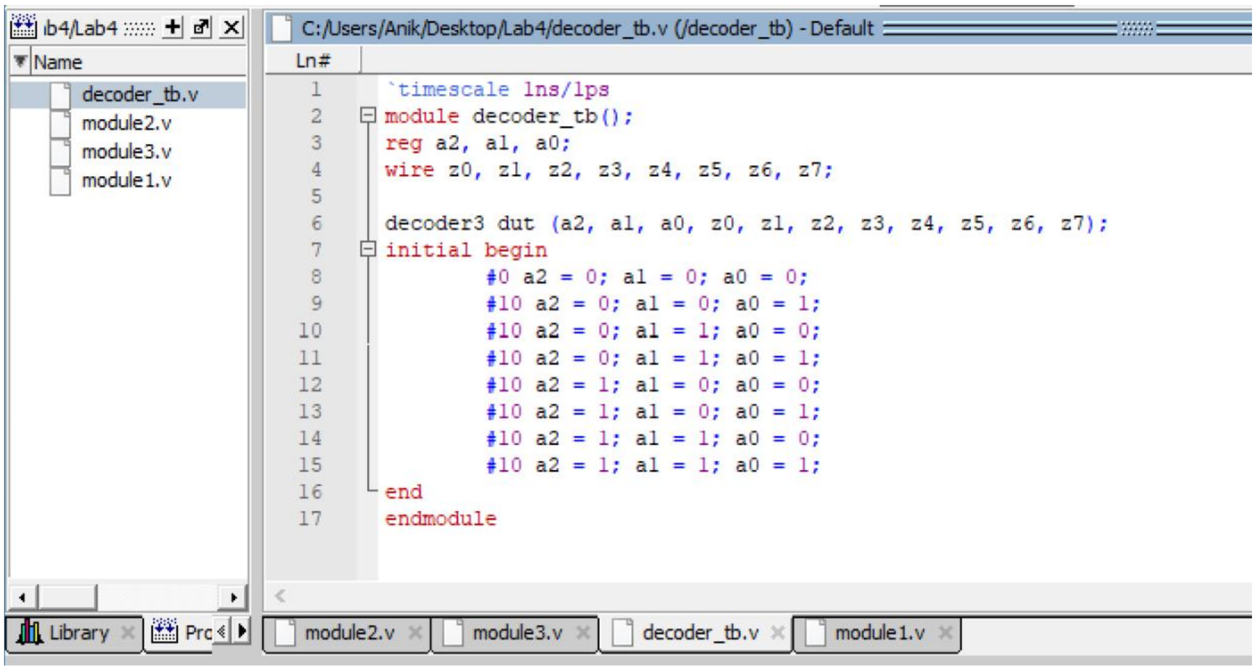
```verilog
//Continious Assignments Statements
module decoder2 (input A2, A1, A0, output Z0, Z1, Z2, Z3, Z4, Z5, Z6, Z7);
        assign Z0 = {~A2 & ~A1 & ~A0};
        assign Z1 = {~A2 & ~A1 & A0};
        assign Z2 = {~A2 & A1 & ~A0};
        assign Z3 = {~A2 & A1 & A0};
        assign Z4 = {A2 & ~A1 & ~A0};
        assign Z5 = {A2 & ~A1 & A0};
        assign Z6 = {A2 & A1 & ~A0};
        assign Z7 = {A2 & A1 & A0};
endmodule
```

Module for decoder 3 (conditional operator)



```verilog
//Conditional Operators
module decoder3 (input A2, A1, A0, output Z0, Z1, Z2, Z3, Z4, Z5, Z6, Z7);
        assign Z0 = (~A2 & ~A1 & ~A0) ? 1'b1 : 1'b0;
        assign Z1 = (~A2 & ~A1 & A0) ? 1'b1 : 1'b0;
        assign Z2 = (~A2 & A1 & ~A0) ? 1'b1 : 1'b0;
        assign Z3 = (~A2 & A1 & A0) ? 1'b1 : 1'b0;
        assign Z4 = (A2 & ~A1 & ~A0) ? 1'b1 : 1'b0;
        assign Z5 = (A2 & ~A1 & A0) ? 1'b1 : 1'b0;
        assign Z6 = (A2 & A1 & ~A0) ? 1'b1 : 1'b0;
        assign Z7 = (A2 & A1 & A0) ? 1'b1 : 1'b0;
endmodule
```

Testbench Module -



```
`timescale 1ns/1ps
module decoder_tb();
reg a2, a1, a0;
wire z0, z1, z2, z3, z4, z5, z6, z7;

decoder3 dut (a2, a1, a0, z0, z1, z2, z3, z4, z5, z6, z7);
initial begin
        #0 a2 = 0; a1 = 0; a0 = 0;
        #10 a2 = 0; a1 = 0; a0 = 1;
        #10 a2 = 0; a1 = 1; a0 = 0;
        #10 a2 = 0; a1 = 1; a0 = 1;
        #10 a2 = 1; a1 = 0; a0 = 0;
        #10 a2 = 1; a1 = 0; a0 = 1;
        #10 a2 = 1; a1 = 1; a0 = 0;
        #10 a2 = 1; a1 = 1; a0 = 1;
end
endmodule
```