

**Department of Electrical Engineering**

**Lab Report 3: Design a 3-to-8 decoder using 2-to-4 decoders**

Name: Socheath Sok, ID: 014470701

Partner: Ashley Tran

Group: 11

Professor: Dr. Ray Wang

Class: EE 301

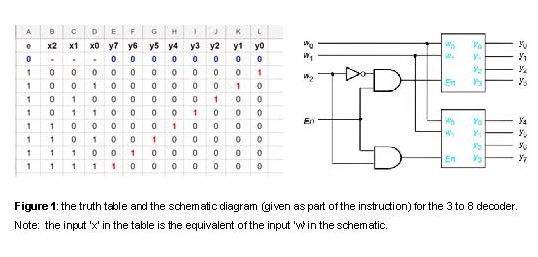
Date Performed: September 27, 2017

**Abstract**

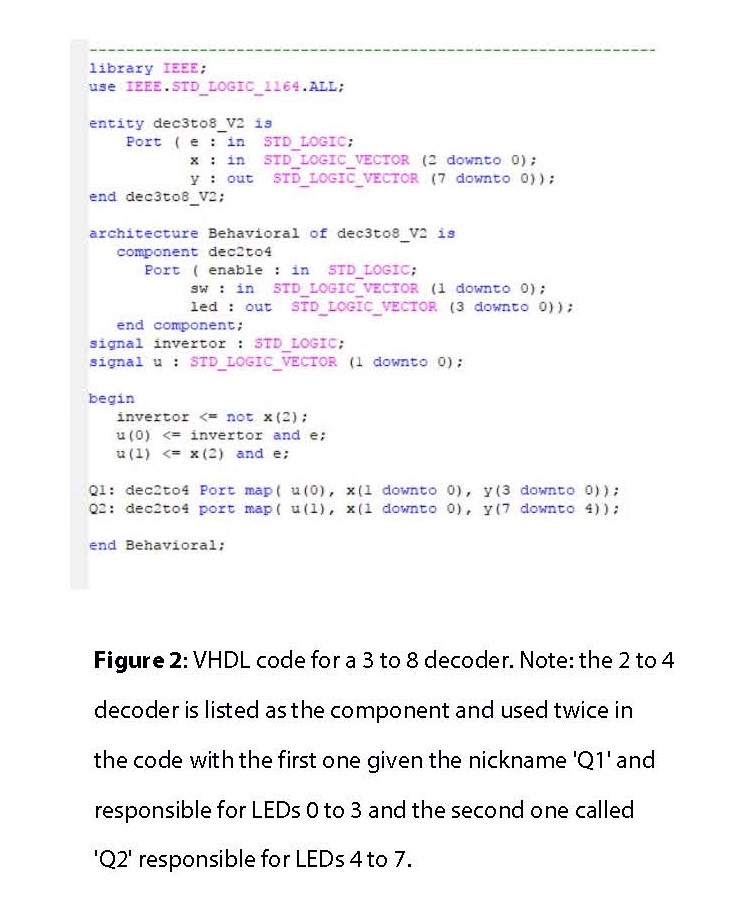
The goal of this lab is to design a three to eight decoder using the two to four decoder code created in the previous lab as a component. From the schematic diagram provided as part of the instruction, we identified the additional input, outputs, gates, and inverter needed to make our decoder. After that, we went on to create the truth table, the VHDL code for 3 to 8 decoder, the testbench code for generating the timing diagram , and the constraint file for transferring our design to our Basys2 board. Lastly, we checked all the combination on the board and compared it to the truth table.

**Introduction**

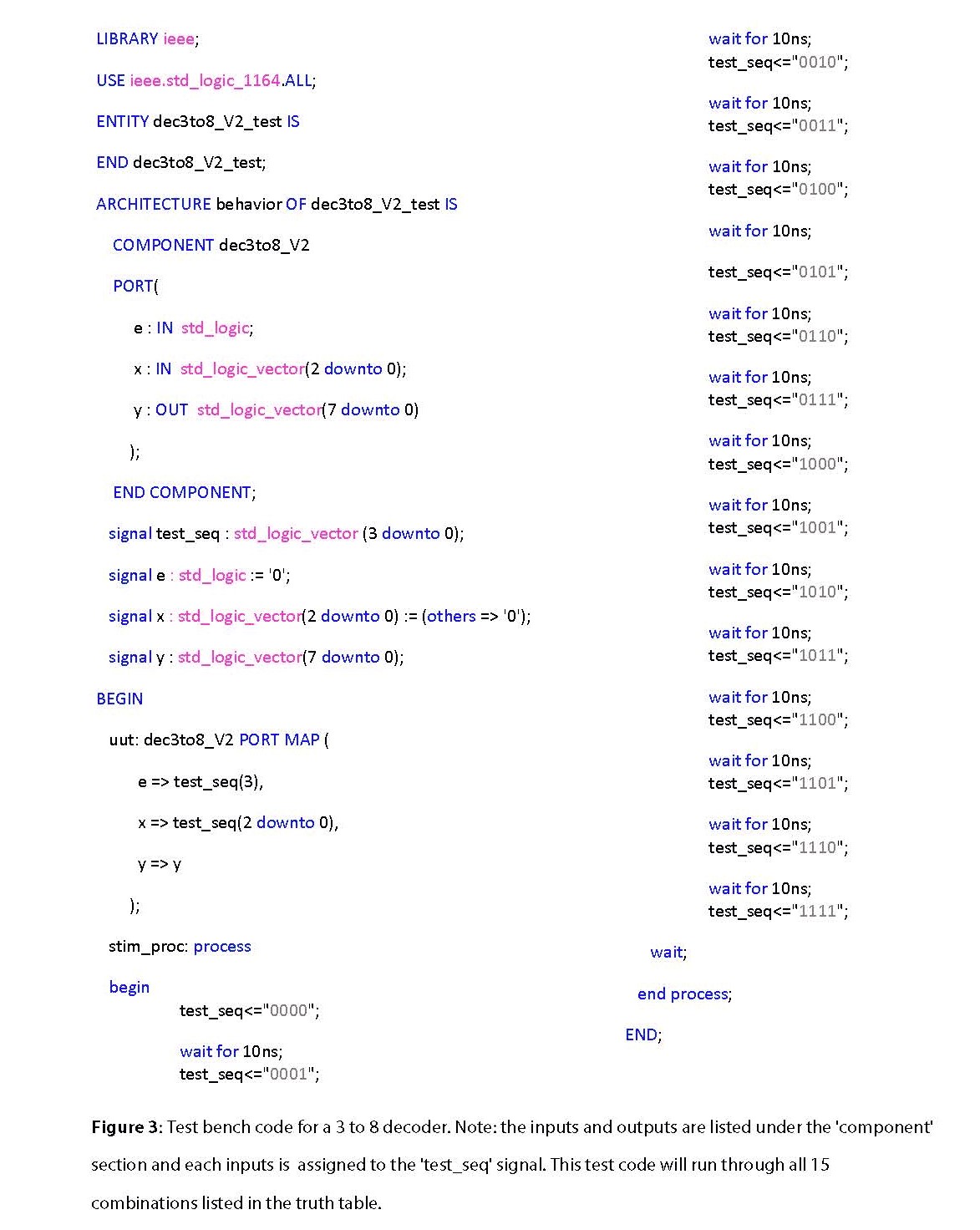
For the 3 to 8 decoder, based on the schematic given, we need an extra input, two AND gates, and an inverter for connecting the two 2 to 4 decoders together, and the result is a total of eight outputs. Next, we created the truth table in **Figure 1** with 3 inputs, 1 enable, and 8 outputs. From the table, we can see that the LEDs only light up when the enable is on (1000 to 1111) and both decoders will not work at the same time. The addition input, which we called ‘x(2)’is the one that determined which decoders will work and which will not. When ‘x(2)’ is high, the top decoder that responsible for ‘y(0)’ to ‘y(3)’ will turn off and the bottom decoder will produce high outputs. The opposite will happens when ‘x(2)’ is low.

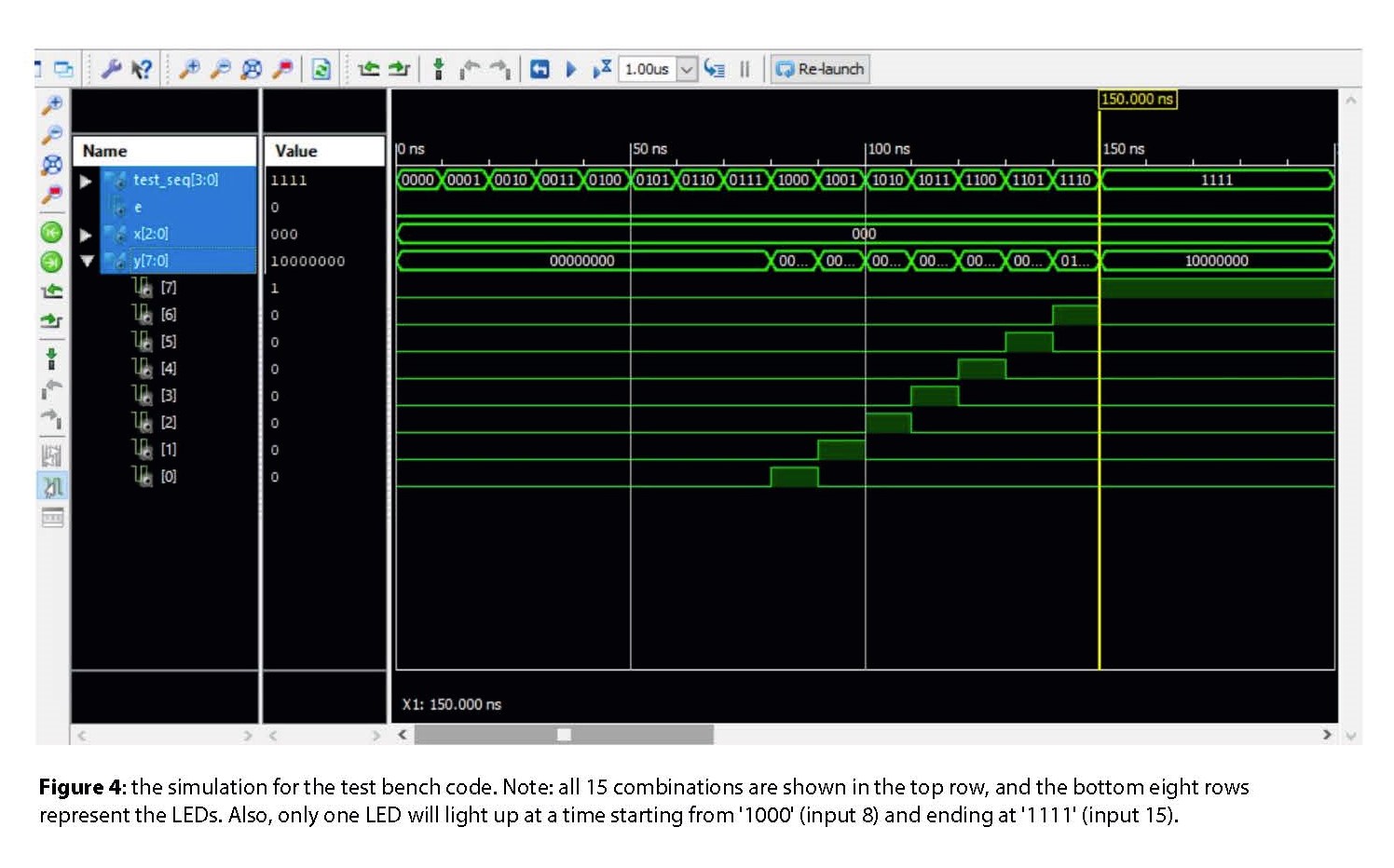


Based on the information we got from the table, we created the VHDL code in **Figure 2**. In the code, we assigned ‘e’ as our enable, vector ‘x’ as our three inputs, and vector ’y’ as our eight outputs. Then, we imported in the 2 to 4 decoder, and put it as the component under the ‘architecture Behavioral of dec3to8\_V2 is’ section. To avoid importing in another 2 to 4 decoder, we assigned the ‘Q1’ and ‘Q2’ as the nicknames, so we can use the same decoder twice. Also, since the ‘Port map’ statements won’t allow us to have any operation in them, we created signals called ‘inverter’ and vector ‘u’ to assign all operation involving gates and inverter before using them in the ‘Q1’ and ‘Q2’ statements.

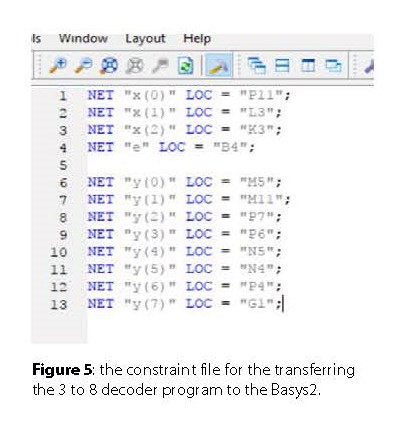


The next step is to create a timing diagram by creating a testbench code for our 3 to 8 decoder code. In the test code, shown in **Figure 3**, we added a new vector signal called ‘test\_seq’ to the skeleton code (created by the software) and assigning it to the inputs (‘e’ and ‘x’ vector). Under the ‘stimulus process’, we put in all the possible combinations and assigning a ‘10ns’ delay between each one. We put ‘200ns’ for our ‘Simulation Run Time’ , so that all inputs are account for and the result of the testbench code is the timing diagram shown in **Figure 4**. At the point marked by the yellow line, we can see that our code is right because the input combination is , and the output is, which is the same as in the truth table.





Lastly, we created a constraint file shown in **Figure 5** and generated the programming file for our board. Also, we changed the FTAG start-up clock to ‘JTAG clock’ ,so that the program will run on PC. Opening the ‘Adept’ software, we transferred our design and checked all the combinations.



**Discussion**

The only problem we encountered in this lab was that we were not sure how to write the ‘Port map’ statements. After a few trials and errors, we found that all we needed to do was giving each statement a nickname, so that the program knows that we want to use two 2 to 4 decoders to make the 3 to 8. Overall, the component/port map method is really useful and saved us a lot of time because we did not have to rewrite the 2 to 4 decoder code. Others than that, we had no problem because the testbench code and the constraints file were made using the exact same procedure as the previous lab. Lastly, every combination we tried on the board produced the same outputs as stated in the truth table, so our structural code is right.