**ELECTRICAL & COMPUTER ENGINEERING**

School of Engineering

**EGRE 365 – Digital Systems**

**Laboratory No. 3**

**Name: Luis Barquero**

**Major: Computer Engineering**

**Date: 09/21/17**

**Honor Pledge:** *I have neither given nor received any unauthorized help on this lab. Signed:*

*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

**2) Component Description**

This lab involved the use of a combinational logic circuit that has an 8-bit bus input and 3 one-bit outputs: parity, zero and one. Parity will check count the 1s in the 8-bit number, and if the sum is odd, it will display a ‘1’; otherwise, it will output a ‘0.’ Zero will check the 8-bit number and will output a ‘1’ if all the bits are zero. In a similar fashion, one that will check for the 8-bit input and will display a ‘1’ if all the bits are ‘1.’ We then changed the size of the 8-bit number to a 4-bit number, and performed a parity, zero and one test. Finally, we had constructed the VHDL testbench for both the 8-bit input and the 4-bit input to verify that the results were as expected.

**3) Implementation**

In order to store the l’s and 0’s from the user input number, we assign the 8-bit and the 4-bit bus in arrays with the variable N to denote the size. We then assign parity, zero and one as single bit outputs. Next, we use a loop that will go through the 8-bit input and detect if a ‘1’ is active; if so, another variable named sum will add them up. Once the loop is complete, we check to see if the number of 1s is even or odd. If the result is even, then parity displays a ‘0’; if the result is odd, parity displays a ‘1.’ We then check to see if the sum = 0; if so, zero displays a ‘1’, and if the sum is greater, zero displays a ‘0.’ Finally, we check if the sum = N, or the size of the array. If so, this means all bits are ‘1’, and the variable one is set to ‘1’; otherwise, one will equal ‘0.’

Once we had constructed the model, we wrote the VHDL code for the testbench to verify that the inputs and outputs are as expected. We start by making an input array size (0 to N-1). Next, we set the test values that we want the simulation to run through as the input array, and we set the expected results from the single bit output – parity, zero and one. Next, we construct the DUT port map by linking the constants used to their corresponding signals that will be put under test.

Finally, we construct a monitor process that will verify the result is expected; if not, then, an error message will appear displaying that the output was not as expected, and therefore is incorrect.

**4) VHDL Code**

See Appendix A for the VHDL Code of the 8-bit input and the 4-bit input, and the test bench for both models.

**5) Tests**

In order to successfully test the code, we first input the six examples provided in the lab assignment handout by forcing them into the 8-bit bus. Once these six inputs had passed the test, we created four other inputs, for a total of ten different scenarios. We then tested the same inputs with the testbench and verified that both tests had the same results.

We then constructed another ten examples and forced them into the 4-bit input, and verified that all had the expected output. Once again, we verified the results with the testbench constructed.

The following is a table that illustrates all 20 tests performed, both forced and through the testbench.

***Figure 1 – Figure 1 displays the table of the 10 8-bit numbers entered by the user, the single bit outputs that***

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Trial** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** |
| **8-bit** | **0000 0000** | **0010 0101** | **0100 1011** | **1101 0101** | **1110 1011** | **1111 1111** | **1001 0010** | **1000 1000** | **0011 0011** | **1010 1010** |
| **Parity** | **0** | **1** | **0** | **1** | **0** | **0** | **1** | **0** | **0** | **0** |
| **Zero** | **1** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** |
| **One** | **0** | **0** | **0** | **0** | **0** | **1** | **0** | **0** | **0** | **0** |

***Figure 2 – Figure 2 shows***

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Trial** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** |
| **4-bit** | **0011** | **1010** | **0010** | **1001** | **1000** | **0110** | **0000** | **1111** | **0111** | **1011** |
| **Parity** | **0** | **0** | **1** | **0** | **1** | **0** | **0** | **0** | **1** | **1** |
| **Zero** | **0** | **0** | **0** | **0** | **0** | **0** | **1** | **0** | **0** | **0** |
| **One** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **1** | **0** | **0** |

**6) Simulation Waveforms**

See Appendix B

**7) Problems Encountered**

The main problem encountered was the initializing of the signals in the testbench. When initialized without the : = ‘0’, the signal would not show up on the testbench. Eventually, my partner and I figured it out, after trial and error.

Appendix A

VHDL Code

Appendix B

Simulation Output Waveforms