**ELECTRICAL & COMPUTER ENGINEERING**

School of Engineering

**EGRE 365 – Digital Systems**

**Laboratory No. 2**

**Name: Luis Barquero**

**Major: Computer Engineering**

**Date: 09/14/17**

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

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

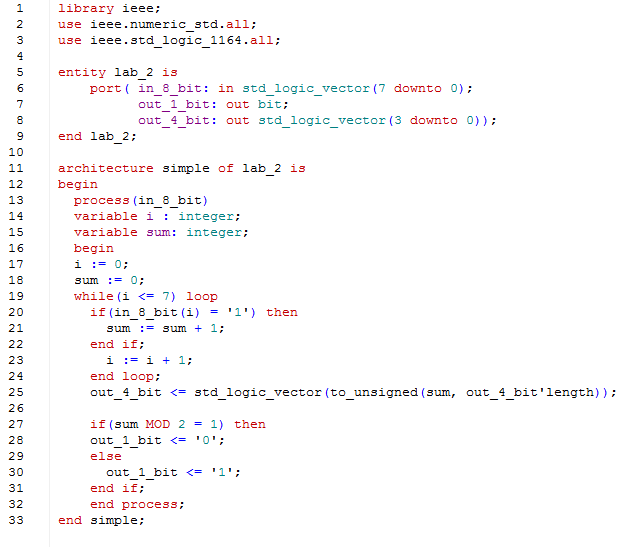
**2) Component Description**

This lab involved the use of a combinational logic circuit that has an 8-bit bus input, a 4-bit bus output and a single bit output. The 8-bit bus would take an 8-bit binary number, and the 4-bit bus counts how many 1’s were present. Finally, the single bit will output a ‘1’ if the number of 1’s is even and a ‘0’ if the number of 1’s is odd.

**3) Implementation**

In order to store the l’s and 0’s from the user input number, we assign both the 8-bit and the 4-bit bus in arrays. Next, we use a loop that will go through the 8-bit bus and detect if a ‘1’ is active; if so, another variable named sum will add them up. Once the loop is complete, a conversion factor will happen where the variable sum will be converted to a bit number and stored into the 4-bit output. Finally, a mod-2 operation will occur on sum, and if the remainder is 0, then the single-bit bus will output a ‘1’; otherwise, the output will be ‘0.’

**4) VHDL Code**

****

**5) Tests**

In order to successfully test the code, we first input the four examples provided in the lab assignment handout by forcing them into the 8-bit bus. Once these 4 had passed the test, we created 16 other inputs, for a total of 20 different scenarios. The following is a table that lists all 20 inputs with their correct outputs.

***Figure 1 – Figure 1 displays the table of the 20 8-bit numbers entered by the user, the 4-bit output that counts the 1s, and the single bit out that displays a ‘1’ or ‘0’ according to the number of 1s in the original input.***

|  |  |  |  |
| --- | --- | --- | --- |
| Trial | 8-Bit Input | 4-Bit output | Single Bit Output |
| 1 | 0000 0000 | 0000 | 1 |
| 2 | 0010 0101 | 0011 | 0 |
| 3 | 1101 0101 | 0101 | 0 |
| 4 | 1111 1111 | 1000 | 1 |
| 5 | 1011 0110 | 0101 | 0 |
| 6 | 0101 1101 | 0101 | 0 |
| 7 | 0000 0001 | 0001 | 0 |
| 8 | 1110 0010 | 0100 | 1 |
| 9 | 1111 0001 | 0101 | 0 |
| 10 | 1100 1100 | 0100 | 1 |
| 11 | 0001 1101 | 0100 | 1 |
| 12 | 1111 1110 | 0111 | 0 |
| 13 | 1010 1010 | 0100 | 1 |
| 14 | 1011 1011 | 0110 | 1 |
| 15 | 1011 1110 | 0110 | 1 |
| 16 | 1000 0001 | 0010 | 1 |
| 17 | 1001 0010 | 0011 | 0 |
| 18 | 1000 1000 | 0010 | 1 |
| 19 | 0110 0110 | 0100 | 1 |
| 20 | 0000 1011 | 0011 | 0 |

**6) Simulation Waveforms**

See Appendix A

**7) Problems Encountered**

The main problem encountered was converting the 4-bit output into an integer, since we needed to perform a mod-2 operation on it in order to display a ‘1’ or ‘0.’ Once we determined the syntax for converting and realizing that we needed certain libraries initialized at the beginning of the code, we encountered no problems.

Appendix A

Simulation Output Waveforms