**ELECTRICAL & COMPUTER ENGINEERING**

School of Engineering

**EGRE 365 – Digital Systems**

**Laboratory No. 4**

**Name: Luis Barquero**

**Major: Computer Engineering**

**Date: 09/28/17**

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

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

**2) Component Description**

This lab involved modeling an ALU using the std\_logic type and the numberic\_std package. The ALU contained 5 parts: Mode, OE (Enabler), C, Zero, Cout/Parity. The purpose of the ALU was to implement 9 modes; 8 when the Enabler was set to ‘1’, and one mode for when the enabler was set to ‘0.’ The 8 modes implemented when the Enabler was set to ‘1’ were: adding, subtracting, two’s complement, logical left shift, AND operation, OR operation, XOR operation, and the NOT operation. The final operation was when the Enabler was set to ‘0’, which resulted in the output displaying “zzzzzz”. Besides calculating the 9 operations, we had to also to implement a Carry/Parity and a Zero operation. The Carry operation would display a ‘1’ for three scenarios:

1) after doing addition and/or subtraction, if the result acquired a carry bit in the left most bit

2) if, after performing the -A operation, all the bits were 0s

3) and if, when performing the left logical shift, the sign has changed

For Parity, the ALU displays a ‘1’ if after performing logical operations – AND, OR, XOR, NOT – if the number of 1s in C is odd, then we get a ‘1’; otherwise, we get a ‘0.’

For Zero, the ALU displays a ‘1’ if all bits in C – the result – are ‘0’; otherwise, Zero displays a ‘0.’

After constructing the VHDL Model, we wrote a testbench that would simulate 4 cases for each mode, for a total of 36 case: 32 for Modes 0-7, and 4 cases for when Enable = ‘0.’

**3) Implementation**

We first begin by using the ieee library and the standard logic to initialize inputs and outputs as either standard logic vectors – inputs A and B(size 16), Mode(size 3), and C(size 16) – or simply of type standard logic, such as with EN, Carry/Parity and Zero.

Next, we broke down the logic for the ALU in terms of its 9 cases, or Modes, as illustrated by table 1:

|  |  |  |
| --- | --- | --- |
| **Mode** | **Enable** | **C** |
| **000** | **1** | **C <= A+B** |
| **001** | **1** | **C <= A-B** |
| **010** | **1** | **C <= -A** |
| **011** | **1** | **C <= SLL(A)** |
| **100** | **1** | **C <= A • B** |
| **101** | **1** | **C <= A | B** |
| **110** | **1** | **C <= A (XOR) B** |
| **111** | **1** | **C <= NOT A** |
| **…** | **0** | **“zzzzzz…zzz”** |

***Table 1 references all 9 cases performed in the ALU, where Mode and Enable determine the case.***

**Addition (Mode 000, EN 1):**

For the addition, we first performed the XOR operation on A and B, since its truth table best simulates adding 1s and 0s.

**Subtraction (Mode 001, EN 1):**

When implementing the subtraction operation, we first began by performing two’s complement on B, and then performing the addition as before.

**-A (Mode 010, EN 1):**

For the negative A, or -A, operation, this is the equivalent of performing two’s complement on A.

**SLL(A) (Mode 011, EN 1):**

When performing the left logical shift, we found that the best way to achieve is with the following logic:

VECT <= VECT (14 DOWNTO 0) AND '0';

In this code, we are preserving the first 15 bits of A and appending the last bit with a ‘0,’ which results in the shifting of A to the left by 1 bit.

**AND/OR/XOR/NOT (Modes 100-110, EN 1)**

Finally, we finished all cases when EN = ‘1’ by performing four logical operations – AND, OR, XOR, NOT – and since they are already built in to Modelsim, we just had to simply implement them.

**ZZZZZZZZZZZZ (Modes 111, EN 1:**

The final cases for the ALU was the enabler = ‘0,’ which results in C displaying “zzzzz.”

**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 our code, we decided to perform four cases for each one of the modes in the ALU, for a total of 36 cases: 32 for Modes 0-7, and 4 cases for when Enable = ‘0.’

In each one of the four cases, inputs A and B were set to arbitrary values, varying from each other, in order to ensure the ALU worked in all cases. Next, in order to maintain consistency, we decided to maintain the same values set for A and B across all 36 cases. Table 2 displays the values set for inputs A and B.

***Table 2 displays the inputs set for A and B that will be used across all 36 cases***

|  |  |
| --- | --- |
| Case | A |
| 1 | 0000 0000 0000 0001 |
| 2 | 0000 0000 0000 0110 |
| 3 | 1111 1111 1111 1111 |
| 4 | 0000 0111 1111 1100 |

|  |  |
| --- | --- |
| Case | B |
| 1 | 0111 1111 1100 0001 |
| 2 | 0000 0000 0000 0110 |
| 3 | 0010 0000 0001 0111 |
| 4 | 1100 0000 0011 1110 |

We then performed all 9 cases, with the following tables showcasing the results for each case.

***Table 3 displays the results acquired for Mode 000, Enable 0***

|  |  |  |
| --- | --- | --- |
| **C** | **Carry/Parity** | **Zero** |
| **‘zzzz zzzz zzzz zzzz’** | **0** | **0** |
| **‘zzzz zzzz zzzz zzzz’** | **0** | **0** |
| **‘zzzz zzzz zzzz zzzz’** | **0** | **0** |
| **‘zzzz zzzz zzzz zzzz’** | **0** | **0** |

**At this point, we set Enable = ‘1’ for the remaining 32 cases**

***Table 4 displays the results acquired for Mode 000***

|  |  |  |
| --- | --- | --- |
| **C** | **Carry/Parity** | **Zero** |
| **0111 1111 1100 0010** | **0** | **0** |
| **0000 0000 0000 1100** | **0** | **0** |
| **0010 0000 0001 0110** | **0** | **0** |
| **1100 1000 0011 1010** | **1** | **0** |

***Table 5 displays the results acquired for Mode 001***

|  |  |  |
| --- | --- | --- |
| **C** | **Carry/Parity** | **Zero** |
| **1000 0000 0100 0000** | **0** | **0** |
| **0000 0000 0000 0000** | **0** | **0** |
| **1101 1111 1110 1000** | **1** | **0** |
| **0100 0111 1011 1110** | **0** | **0** |

***Table 6 displays the results acquired for Mode 010***

|  |  |  |
| --- | --- | --- |
| **C** | **Carry/Parity** | **Zero** |
| **1111 1111 1111 1111** | **0** | **0** |
| **1111 1111 1111 1010** | **0** | **1** |
| **0000 0000 0000 0001** | **1** | **0** |
| **1111 1000 0000 0100** | **0** | **0** |

***Table 6 displays the results acquired for Mode 011***

|  |  |  |
| --- | --- | --- |
| **C** | **Carry/Parity** | **Zero** |
| **0000 0000 0000 0010** | **0** | **0** |
| **0000 0000 0000 1100** | **0** | **0** |
| **1111 1111 1111 1110** | **1** | **0** |
| **1111 1111 1111 1000** | **0** | **0** |

***Table 6 displays the results acquired for Mode 100***

|  |  |  |
| --- | --- | --- |
| **C** | **Carry/Parity** | **Zero** |
| **0000 0000 0000 0001** | **1** | **0** |
| **0000 0000 0000 0110** | **0** | **0** |
| **0010 0000 0001 0111** | **1** | **0** |
| **0000 0000 0011 1100** | **0** | **0** |

***Table 7 displays the results acquired for Mode 101***

|  |  |  |
| --- | --- | --- |
| **C** | **Carry/Parity** | **Zero** |
| **0111 1111 1100 0001** | **0** | **0** |
| **0000 0000 0000 0110** | **0** | **0** |
| **1111 1111 1111 1111** | **0** | **0** |
| **1100 0111 1111 1110** | **0** | **0** |

***Table 8 displays the results acquired for Mode 110***

|  |  |  |
| --- | --- | --- |
| **C** | **Carry/Parity** | **Zero** |
| **0111 1111 1100 0000** | **1** | **0** |
| **0000 0000 0000 0000** | **0** | **1** |
| **1101 1111 1110 1000** | **1** | **0** |
| **1100 0111 1100 0010** | **0** | **0** |

***Table 9 displays the results acquired for Mode 111***

|  |  |  |
| --- | --- | --- |
| **C** | **Carry/Parity** | **Zero** |
| **1111 1111 1111 1110** | **1** | **0** |
| **1111 1111 1111 1001** | **0** | **0** |
| **0000 0000 0000 0000** | **0** | **1** |
| **1111 1000 0000 0011** | **1** | **0** |

**6) Simulation Waveforms**

See Appendix B

**7) Problems Encountered**

The main problem encountered was that implementation of standard logic, since in certain cases, VHDL would tell us that we were using a std\_logic\_vector but we were trying to get a result as a std\_logic\_boolean or just a std\_logic.

We also had an issue with the testbench, where at we couldn’t get the implementation of a loop to increase Mode by 1 in order to run through all 9 modes. Eventually, after trial and error, we decided to get rid of the loop and implement all cases as illustrated by the Example Lab Write-Up on Blackboard.

Our final issue was the monitoring process during the testbench. In the assertion portion of the monitoring process, we tested all signals to make sure the values were correct. However, for some unknown reason, the kept receiving assertion error message explaining that there were incorrect values for our C signal, even though it was the correct value. We tried fixing it by having the monitoring process wait on Mode instead of Input A, but we still had one assertion error message regarding the C value. Our fix to this was to get rid of the monitoring process altogether, since it wasn’t necessary for the lab and we didn’t know how to fix the issue.

Appendix A

VHDL Code

Appendix B

Simulation Output Waveforms