**EECE 2323 Digital Logic Design Lab Report**

Lab 2 Partial ALU

| Student Name:Ousmane Toure | Section #: 1 |
| --- | --- |
| Instructor Name: Dr. Aboelela  Lab TA Name: Keshav Bharadwaj Vaidyanathan, Srinidi Somasundaram Subbulakshmi, Yuchao Su | Date: 07/13/22 |

1. **Background & Purpose**:

In this experiment, we designed and implemented a partial Arithmetic and Logic Unit (ALU) responsible for doing addition, bitwise NOT, AND, and OR with the results being displayed on the PYNQ LED. ALU’s are essential because they give central processing units the ability to perform mathematical operations. This is important because it allows the CPU to do functions like addition, division, subtraction etc. The goal of this lab was to utilize verilog code in vivado to represent a partial ALU and virtually testing it before physically implementing it. Furthermore enhancing confidence and proficiency in vivado and computational architecture.

1. **Pre-Lab Response:**

**Table 1:**  2’s Complement Truth Table

| a | b | **s** | f | ovf |
| --- | --- | --- | --- | --- |
| 8’d0  00000000 | 8’d0  00000000 | **2’d0**  **00** | 8’d0  00000000 | 0 |
| -8’d12  11110100 | -8’d34  11011110 | **2’d0**  **00** | -8’d46  11010010 | 0 |
| 8’d100  01100100 | 8’d100  01100100 | **2’d0**  **00** | -8’d56  11001000 | 1 |
| 8’d1  00000001 | 8’d1  00000001 | **2’d1**  **01** | 8’d4  00000100 | 0 |
| -8’d12  11110100 | -8’d12  11110100 | **2’d1**  **01** | 8’d11  00001011 | 0 |
| 8’d100  01100100 | 8’d100  01100100 | **2’d1**  **01** | -8’d101  10011011 | 0 |
| 8’d1  00000001 | 8’d1  00000001 | **2’d2**  **10** | 8’d1  00000001 | 0 |
| 8’d1  00000001 | 8’d3  00000011 | **2’d2**  **10** | 8’d1  00000001 | 0 |
| -8’d4  11111100 | -8’d5  11111011 | **2’d2**  **10** | -8’d8  11111000 | 0 |
| 8’d1  00000001 | 8’d1  00000001 | **2’d3**  **11** | 8’d1  00000001 | 0 |
| 8’d12  00001100 | -8’d16  11110000 | **2’d3**  **11** | -8’d4  11111100 | 0 |
| -8’d2  c | -8’d6  11111010 | **2’d3**  **11** | -8’d2  11111110 | 0 |

1. **Summary of Design Implementation**
   1. **Results and Analysis:**

When conducting this experiment, the partial ALU works by utilizing a 2 bit selector which tells the ALU which operation to perform. After creating our code represented in Appendix A we then created a test bench that would test all of our arithmetic operations. We plugged in values and ran a simulation to verify that an overflow would in fact occur when two negative numbers or two positive numbers were added together, but the sum was the opposite sign. Our results, as stated in Appendix B Figure 1. The results of the test bench verified that our code in fact does work and gives us the green light to program straight into the PYNQ-Z2 board. After programming our board and connecting the add on board, we displayed varying the addition operation and the bitwise AND operation as seen in the two screenshots in Appendix B. Both physical tests resulted in values that were consistent with our pre lab test bench truth table highlighting that our circuit was in fact correct. Errors could occur if you used wrong bitwise vs logical operators when writing your verilog code. For example, using ! instead of ~ resulted in my code not performing correctly. Furthermore, incorrectly setting up your test bench with proper values and operations that reflected your verilog code would result in an error being raised, or your test bench not performing correctly.

* 1. **Conclusion & Recommendations:**

Based on our results, we can conclude that Lab 2 consisted of creating a Partial Arithmetic Logical Unit utilizing verilog code in Vivado. Testing its implementation virtually with a test bench confirmed that our verilog code was correct which gave us the green light to program the PYNQ Board physically. The lab resulted in successfully being able to perform different arithmetic functions like addition as well as bitwise inversion, tools that a Central Processing Unit would need to run varying tasks. Recommendations going forward would be to acquire a bigger add on board with more switches so that more values of B could be tested more securely. Overall, the lab can be seen as a success.

**Appendix A: Design Program Files (Verilog modules, testbenches, etc)**

**Verilog Code:** timescale 1ns / 1ps

//////////////////////////////////////////////////////////////////////////////////

// Company:

// Engineer:

//

// Create Date: 07/12/2022 10:21:52 AM

// Design Name:

// Module Name: called eightbit palu

// Project Name:

// Target Devices:

// Tool Versions:

// Description:

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

//////////////////////////////////////////////////////////////////////////////////

module eightbit\_palu(

input [7:0] a,

input [7:0] b,

input [1:0] sel,

output reg [7:0] f,

output reg ovf

);

always@(a or b or sel)

begin

case(sel)

2'b00: //add

begin

f = a + b;

ovf =((a[7] & b[7]) &!f[7]) | ((!a[7] & !b[7])&f[7]);

end

2'b01: //b inversion

begin

f= ~b;

ovf = 0;

end

2'b10: //AND

begin

f = a \* b;

ovf = 0;

end

2'b11: //OR

begin

f = a | b;

ovf = 0;

end

default:

begin

f = a + b;

ovf =((a[7] & b[7]) &!f[7]) | ((!a[7] & !b[7])&f[7]);

end

endcase

end

endmodule

**TestBench Code:**

`timescale 1ns / 1ps

//////////////////////////////////////////////////////////////////////////////////

// Company:

// Engineer:

//

// Create Date: 07/13/2022 10:04:55 AM

// Design Name:

// Module Name: palu\_unittb

// Project Name:

// Target Devices:

// Tool Versions:

// Description:

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

//////////////////////////////////////////////////////////////////////////////////

module palu\_unittb( );

//inputs

reg [7:0] a = 8'b0;

reg [7:0] b=8'b0;

reg [1:0] sel = 2'd0 ;

wire [7:0] f;

wire ovf;

eightbit\_palu UUT

(

.a(a),

.b(b),

.sel(sel),

.f(f),

.ovf(ovf)

);

initial

begin

sel=2'd0;

a=8'd0;

b=8'd0;

#100;

a=-8'd12;

b=-8'd34;

#100;

a=8'd100;

b=8'd100;

#100;

sel=2'd1;

a=8'd1;

b=8'd1;

#100;

a=-8'd12;

b=-8'd12;

#100;

a=8'd100;

b=8'd100;

#100;

sel=2'd2;

a=8'd1;

b=8'd1;

#100;

a=8'd1;

b=8'd3;

#100;

a=-8'd4;

b=-8'd5;

#100;

sel=2'd3;

a=8'd1;

b=8'd1;

#100;

a=8'd12;

b=-8'd16;

#100;

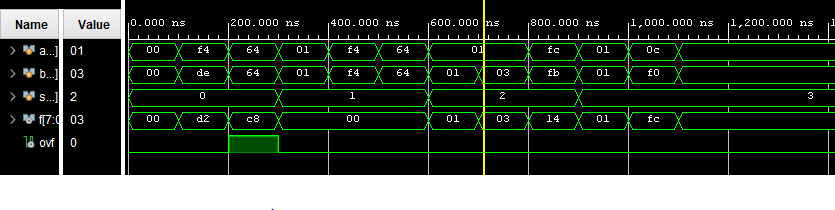
a=-8'd2;

b=-8'd6;

end

endmodule

**Appendix B: Captures of the output screens and simulation waveforms.**



**Figure 1:** Test Bench WaveForm Simulation

**Figure 2:** LED Results

