Name: Alexander Wise **Lab #2 – Arithmetic and Logic Unit**

Instructor Name: Professor Consi **January 30, 2024**

Background

The ALU or Arithmetic Logic Unit is a fundamental component to computers and is often referred to as the heart of the CPU. It is responsible for executing the core operations that drive our computers and other devices. These operations include binary addition, inversion, bitwise functionality, and many other operations for a total of 9. The ALU has 3 inputs: a, b, and s where a and b are both binary inputs to be manipulated and s is the select input to choose which of the 9 operations to utilize. As a crucial component in processors, the ALU will be utilized in further labs down the line as we work to design a processor.

Pre-Lab

3.1.1

1.
- a = 0000 0011 1101 1011, b = 0000 0010 1010 0110
- xor a, b and store in a
  - f = 0000 0001 0111 1101
  - ovf = 0
  - take_branch = 0
- invert b and store in b
  - f = 1111 1101 0101 1001
  - ovf = 0
  - take_branch = 0
- or a and b and store in b
  - f = 1111 1101 0111 1101
  - ovf = 0
  - take_branch = 0
- and b with 7 and store in b
  - f = 0000 0000 0000 0101
  - ovf = 0
  - take_branch = 0
- Sll value in a by b times
  - f = 0010 1111 1010 0000
  - ovf = 0
  - take_branch = 0

2.
- a = 1110 1001, b = 0000 0010
- SRA value in a by b times and store in a
  - f = 1111 1010

    - ovf = 0
    - take_branch = 0
- Add 62 to value in a
    - f = 0011 1000
    - ovf = 1
    - take_branch = 0

- BNEZ a
    - f = 0000 0000
    - ovf = 0
    - take_branch = 0

3.1.2

a	b	s	f	ovf	take_branch
16'hA415	16'hA555	4'b0000	16'h496A	1	0
16'h023D	16'h564B	4'b0001	16'hA9B4	0	0
16'h21AA	16'hB890	4'b0010	16'h2180	0	0
16'E509	16'D79A	4'b0011	16'hF79B	0	0
16'00E9	16'0002	4'b0100	16'h003A	0	0
16'h017D	16'017D	4'b0101	16'h0000	0	0
16'h0001	16'h0000	4'b0110	16'h0001	0	0
16'h0001	16'h0000	4'b0111	16'h0000	0	1
16'h81A3	16'h897B	4'b1000	16'h08D8	0	0
16'hED00	16'hEF03	4'b0000	16'hDC03	0	0
16'hF89B	16'hD77A	4'b0100	16'h0000	0	0
16'hD782	16'h00F3	4'b1000	16'hD771	0	0

Table 1: Pre-Lab Table of Test Values

Design Implementation

3.2 – Entering Your Design

The ALU has three input points, a, b, and s. Inputs a and b are both 16-bit inputs while s is a 4 bit input. Because the ALU has 9 cases, a case statement was utilized to ensure that all operations could be easily accessed depending on the input value provided by *s.* The addition component of the ALU used the 16-bit adder created previously in Lab 1. The inversion, and, or, SRA, SLL and XOR all utilized bitwise operations to easily manipulate the binary value inputted by the *a* and *b* values.

The System Verilog code for the ALU can be found in Appendix A, Figure 1.

4 – Simulating Your Design

To simulate the ALU designed in the previous section, a test bench was designed to test custom values for all cases. The test bench utilized the test cases found in Table 1 as well as other cases from Lab 1 and from part 1 of the pre-lab. The System Verilog code can be found in Appendix A, Figure 2.

Once the test bench was designed, it was run, and a waveform was generated from the outputs. The waveform can be found in Appendix A, Figure 3.

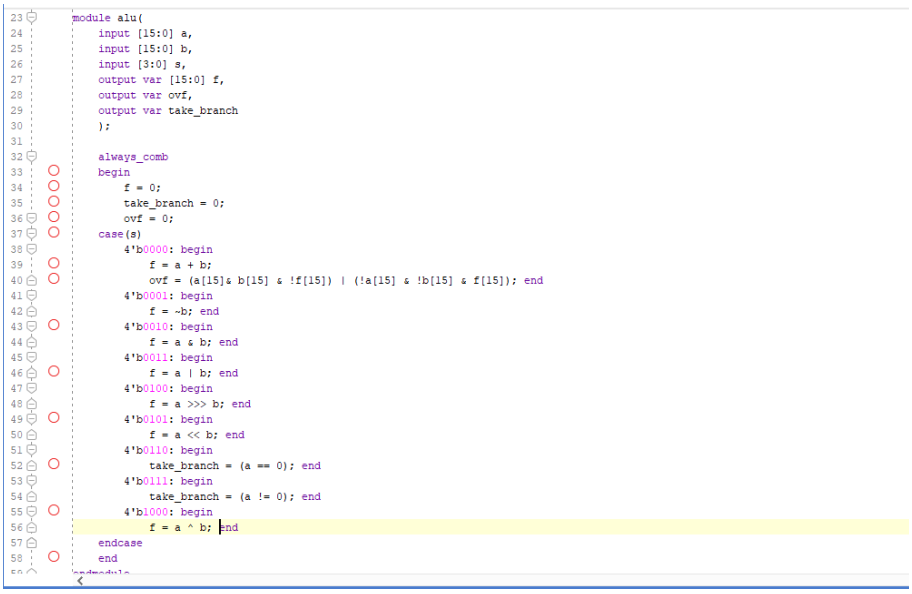
5 – Testing in Hardware

After verifying the results of the simulation against the table of test cases, the ALU was uploaded to the PYNQ-Z2 board to be tested on hardware. An image of the ALU working on hardware can be found in Appendix B, Figure 4.

Conclusion

The ALU is the powerhouse of computers and is a fundamental component to processors. They are arguably the most important component of a computer. As a vital component, learning how to apply System Verilog to designing an ALU will prove to be vital towards future labs as we continue to build off previous labs and work towards the final goal of the course. All cases worked across the 9 functions and various test cases for each function so the ALU works and can be used later throughout the course. Similarly, the lab further reinforced the System Verilog functions that we have been learning in class.

Appendix A – Xilinx Vivado Screen Captures



```
module alu(
    input [15:0] a,
    input [15:0] b,
    input [3:0] s,
    output var [15:0] f,
    output var ovf,
    output var take_branch
    );

    always_comb
    begin
        f = 0;
        take_branch = 0;
        ovf = 0;
    case(s)
        4'b0000: begin
            f = a + b;
            ovf = (a[15]& b[15] & !f[15]) | (!a[15] & !b[15] & f[15]); end
        4'b0001: begin
            f = ~b; end
        4'b0010: begin
            f = a & b; end
        4'b0011: begin
            f = a | b; end
        4'b0100: begin
            f = a >>> b; end
        4'b0101: begin
            f = a << b; end
        4'b0110: begin
            take_branch = (a == 0); end
        4'b0111: begin
            take_branch = (a != 0); end
        4'b1000: begin
            f = a ^ b; end
    endcase
    end
```

Figure 1: ALU System Verilog Code

```
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module alu_tb();
logic [15:0] a;
logic [15:0] b;
logic [3:0] s;
logic [15:0] f;
logic ovf;
logic take_branch;

alu UTT(.a(a), .b(b), .s(s), .f(f), .ovf(ovf), .take_branch(take_branch));

initial begin

    #100 a = 16'hA415; b = 16'hA555; s = 4'b0000;
    #100 a = 16'h023D; b = 16'h564B; s = 4'b0001;
    #100 a = 16'h21AA; b = 16'hB890; s = 4'b0010;
    #100 a = 16'hE509; b = 16'hD79A; s = 4'b0011;
    #100 a = 16'h00E9; b = 16'h0002; s = 4'b0100;
    #100 a = 16'h017D; b = 16'h017D; s = 4'b0101;
    #100 a = 16'h0001; b = 16'h0000; s = 4'b0110;
    #100 a = 16'h0001; b = 16'h0000; s = 4'b0111;
    #100 a = 16'h81A3; b = 16'h897B; s = 4'b1000;
    #100 a = 16'hED00; b = 16'hEF03; s = 4'b0000;

end

endmodule
```

Figure 2: ALU Test Bench code

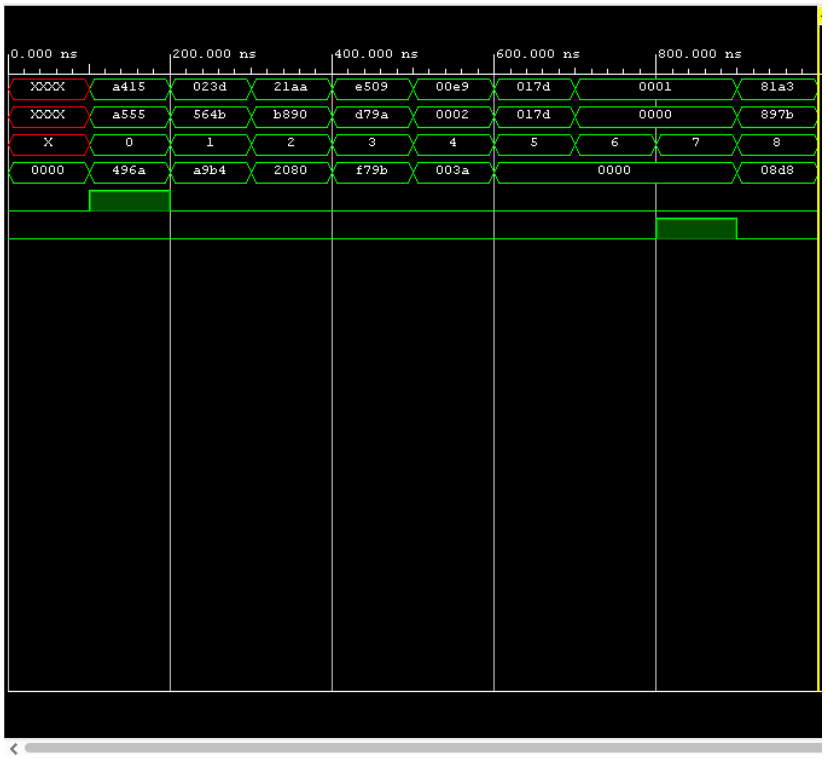


Figure 3: ALU Test Bench waveform

Appendix B – Hardware Images

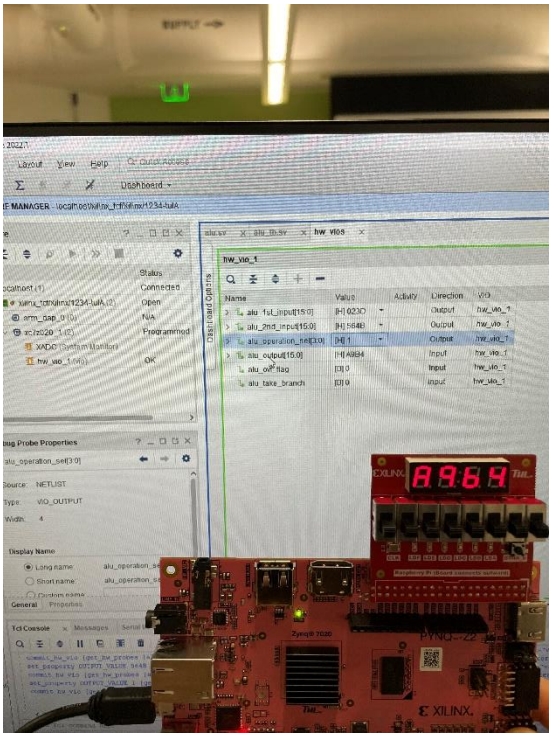


Figure 4: ALU working on hardware