Report for Fall 2023 EE 361L Lab 5

Author: Justine Ponce

Lab Partner: Michael Lau

Date: October 28, 2023

Abstract: In this lab, we have two assignments. The first is just a tutorial on how to use a Web-pack software tool called Xilinx Vivado and a device called Field Programmable Gate Arrays or FPGA board. We are assigned to learn how to use the Web-pack tool so that we can configure the FPGA board. The second assignment is to implement an arithmetic logic unit to ALU and then implement the ALU to the FPGA board by using the Web-pack tool.

1) Introduction

As described in the abstract we have two tasks. Three PDF files are given to us.

- EE361L-Lab5-FPGA [1]: This will be our main lab manual it talks about our tasks/assignments.
- Xilinx Vivado Installation Instructions [2]: This file contains the instructions on how to install Xilinx Vivado on your personal computer.
- TutorialArtix-7Basys3 [3]: This is a tutorial on how to use the webpack software tool, Xilinx Vivado, to code SystemVerilogs, and implement it to the FPGA.

The first task will be my second paragraph, the second task coding part will be the third paragraph, the FPGA board interaction will be my fourth paragraph and my last will be the conclusion and references.

2) Vivado and FPGA

This task only consists of us following the tutorial files given to us. The first tutorial is the installation of Xilinx Vivado [2]. You don't necessarily need to download it if you are going to use the school computers since it's already been pre-installed. This tutorial is only if you want it on your personal computer. Just letting you know, it's around 20 gigabytes. The second PDF tutorial [3], is a must-read file. This talks about how to set up our lab in Vivado. The settings we have to choose, how to write and implement SystemVerilog codes, how to connect it to the FPGA so that the switches, LEDs, and buttons, work as intended. So, the tutorial files are a must-read to successfully do this lab.

3) ALU Circuit

Upon looking at the lab manual [1], we are given an ALU to build in SystemVerilog. Refer to Figure 1[1] for the circuit diagram.

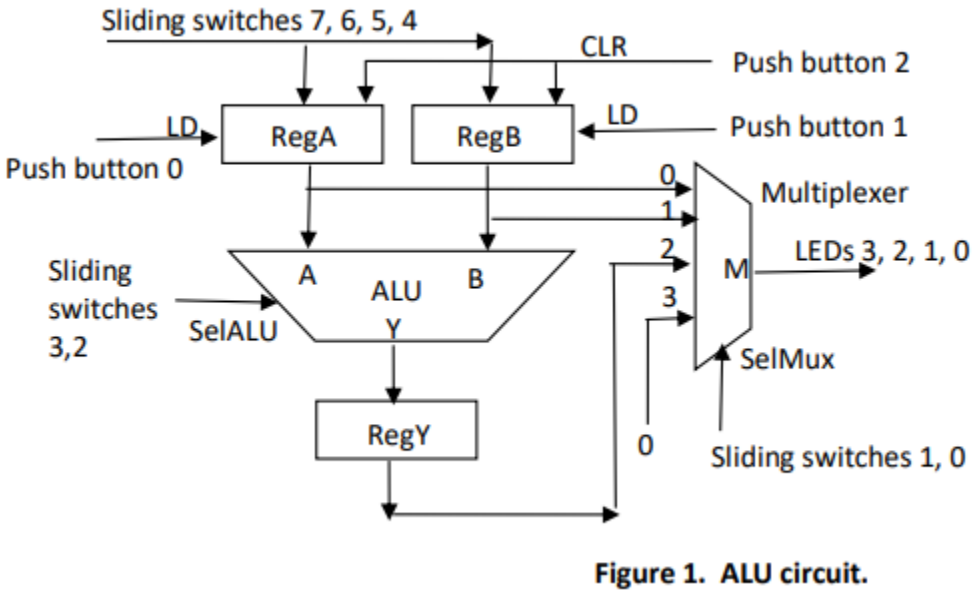

Figure 1. ALU circuit.

[1] Figure 1: ALU Circuit

Now looking at the circuit there a 3 different circuits within the circuit, the Register circuits, an ALU circuit, and a Multiplexer circuit.

There are three three registers that we need, RegA, RegB, and RegY. All three of them synchronize with the clock signal of the FPGA board. RegY will always load a value for the Mux input, so it behaves like a 4D flip-flop. RegA and RegB will follow Figure 2 [1] table below.

CLR	LD	Operation
1	X	Clear
0	1	Load
0	0	Hold

[1] Figure 2: 4-bit Registers

From the table, we know that every time the input ‘CLR’ is 1, the register is cleared. If ‘CLR’ is 0 and ‘LD’ is 1 then the register will load a new value. If both are 0, then the register holds its current value. Upon knowing this, we can build a Register Function, refer to Figure 3.

```
module register(
input logic [3:0] in,
input logic clk, load, clear,
output logic [3:0] out
    );
    always_ff @(posedge clk)
    begin
        if(clear == 1)
            out<=0;
        else if (load == 1)
            out <= in;
     end
endmodule
```

Figure 3: Register Function

This adds a register function, with a 4-bit input and output, along with a clock, load, and clear inputs. We will have a clock that every time it is positive, will execute the sequential logic. The sequential logic is a condition statement, that if clear is equal to 1, the output will be 0 clearing the register. If load is equal to 1, then the output will be updated with the value of the input and then loaded to the register.

The next circuit that we need to build would be the ALU. It is a combinational circuit and we are given a table to follow. Refer to Figure 4[1].

SelALU (2-bit select input)	Operation (4-bit operations)	Description
00	Y = A + B	Addition
01	Y = A - B	Subtraction
10	Y = A | B	Bit-wise OR
11	Y = A & B	Bit-wise AND

[1] Figure 4: ALU Table

On the table, we know that we have a 2-bit select input, 4-bit A and B input, and 4-bit Y output. At each select input(SelALU), it will be doing addition, subtraction, bit-wise OR, and bit-wise AND, to the A and B inputs to get the Y output. Refer to Figure 5 for the code.

```
module alu(
input logic [3:0] A, B,
input logic [1:0] SelALU,
output logic [3:0] Y
    );

    always_comb
        begin
            case(SelALU)
            2'b00: Y=A+B;
            2'b01: Y=A-B;
            2'b10: Y=A|B;
            2'b11: Y=A&B;
       endcase
     end
endmodule
```

Figure 5: Simple ALU in SystemVerilog

As we can see above, we have the input and outputs. The select input (SelALU) will be choosing which operations to use using a case statement. When SelALU is ‘00’ it will run ‘Y=A+B’, ‘01’ will run ‘Y=A-B’, ‘10’ will run ‘Y=A|B’, and lastly ‘11’ will run ‘Y=A&B’. Ultimately, outputting Y.

The last circuit that we need is the 4:1 multiplexer circuit. This circuit will have its output connected to the LEDs of the FPGA board by following the table given in Figure 6 [1].

SelMux	Operation
00	M = RegA (input 0)
01	M = RegB (input 1)
10	M = RegY (input 2)
11	M = 0 (input 3)

[1] Figure 6: 4:1 Multiplexer

Here we have another 2-bit select input for mux(SelMUX) and an output of 4-bit M. The registers A, B, and Y from the very first circuit that I made from this will also become an input. Refer to Figure 7 for the code.

```
module mux(
input logic[3:0] A,B,Y,
input logic[1:0] SelMux,
output logic [3:0] M
);
always_comb
        begin
            case(SelMux)
            2'b00: M=A;
            2'b01: M=B;
            2'b10: M=Y;
            2'b11: M=0;
       endcase
     end
endmodule
```

Figure 7: 4:1 MUX LEDs

So we have the inputs of 3-bit A, B, and Y, a 2-bit input of select input(SelMux), and a 4-bit output of M. Using the same logic as the last paragraph, the select input will be choosing which of the Registers A, B, and Y will equal M such as SelMux is ‘00’, output M will be Register A, ‘01’ output M will be Register B, ‘10’ output M will be Register Y, and ‘11’ output will be 0. If The output M is 1, then the LED corresponding to it will turn on, if 0 then it will be off.

Now that all the circuits are made, we have to connect them all so that they function as the circuit shown in Figure 1[1]. As you can see, Register A and B outputs will be an input of ALU and MUX circuits. ALU circuit output will be the input of Register Y. Register Y will be the input of the MUX circuit, and then the output of the MUX circuit will connect to the LEDs of the FPGA board, which will turn on or off depending on the output. Refer to Figure 8 for the code.

```
module topModule(
input logic clk,
input logic [2:0] Pb,
input logic [7:0] s,
output logic [3:0] LEDs
    );
    wire [3:0] w1, w2, w3, w4;

    register regA(s[7:4],clk, Pb[0], Pb[2], w1);

    register rebB(s[7:4],clk, Pb[1], Pb[2], w2 );

    alu ALU(w1, w2, s[3:2], w3);

    register regY(w3, clk, 1, 0, w4 );

    mux MUX( w1, w2, w4, s[1:0], LEDs);

endmodule
```

Figure 8: ALU Circuit Connections

I made a topModule() function as the main module of all the circuits. It has the clock input(clk), a 2-bit push button(Pb) input, an 8-bit switch(s) input, and a 4-bit LEDs output. The wires declaration will connect the circuits acting as the input and output of the circuits.

So to get Register A and Register B, we must call the Register function twice and instantiate a differing name such as regA and regB so we know what they are. As you can see in Figure 1[1], both registers are identical, they have 4 switches from switch 7 to 4, push button 2, and a clock because back on the register circuit it has a clock input that synchronizes with the FPGA board, so this is a must-have so it works properly. The

difference is that regA has a push button 0 and the output will be wire 1(w1), while regB has a push button 1 with the output of wire 2(w2).

Again we call the function alu and instantiate a different name like ALU (all capitalized). Since we know the output of regA and regB are part of the input of ALU, we just add them as w1 and w2 as their input, along with switch 3 to 2 inputs, and then we can call the output as wire 3(w3).

In the same format, call the function register again and instantiate a name regY. The input is the output of ALU which is w3 then add the clock. Now I put ‘1’ as an input to connect it to the load input from the register function and to indicate that the register should load the input value of w3 when the signal is asserted. The ‘0’ input connects to the clear input of the register function indicating that the register should not be cleared.

Lastly, we call the mux function and called it MUX. From Figure 1[1] we know that the w1, w2, and w4 are part of the input along with switch 1 and switch 0 outputting 4-bit LEDs. The LEDs 1 to 4 should turn on or off depending on the buttons and switches.

4) Field Programmable Gate Arrays Board

If you followed the PDF “TutorialArtix-7Basys3”[3], then you should know that we had to download a folder with an important file called Basys-3-Master.xdc[3] on page 12. This is added to the constraints folder in Vivado. Basically, this file will reflect the inputs and outputs of the topModule function. It contains all the constraints that we need, and all we have to do is uncomment them. Refer to Figure 9 for the buttons.

```
## This file is a general .xdc for the Basys3 rev B board
## To use it in a project:
## - uncomment the lines corresponding to used pins
## - rename the used ports (in each line, after get_ports) according to the top level signal names in the project

## Clock signal
set_property -dict { PACKAGE_PIN W5   IOSTANDARD LVCMOS33 } [get_ports clk]
create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports clk]


## Switches
set_property -dict {PACKAGE_PIN V17 IOSTANDARD LVCMOS33} [get_ports {s[0]}]
set_property -dict {PACKAGE_PIN V16 IOSTANDARD LVCMOS33} [get_ports {s[1]}]
set_property -dict { PACKAGE_PIN W16   IOSTANDARD LVCMOS33 } [get_ports {s[2]}]
set_property -dict { PACKAGE_PIN W17   IOSTANDARD LVCMOS33 } [get_ports {s[3]}]
set_property -dict { PACKAGE_PIN W15   IOSTANDARD LVCMOS33 } [get_ports {s[4]}]
set_property -dict { PACKAGE_PIN V15   IOSTANDARD LVCMOS33 } [get_ports {s[5]}]
set_property -dict { PACKAGE_PIN W14   IOSTANDARD LVCMOS33 } [get_ports {s[6]}]
set_property -dict { PACKAGE_PIN W13   IOSTANDARD LVCMOS33 } [get_ports {s[7]}]


## LEDs
set_property -dict {PACKAGE_PIN U16 IOSTANDARD LVCMOS33} [get_ports {LEDs[0]}]
set_property -dict {PACKAGE_PIN E19 IOSTANDARD LVCMOS33} [get_ports {LEDs[1]}]
set_property -dict {PACKAGE_PIN U19 IOSTANDARD LVCMOS33} [get_ports {LEDs[2]}]
set_property -dict {PACKAGE_PIN V19 IOSTANDARD LVCMOS33} [get_ports {LEDs[3]}]




##Buttons
set_property -dict { PACKAGE_PIN U18   IOSTANDARD LVCMOS33 } [get_ports Pb[0]]
set_property -dict { PACKAGE_PIN T18   IOSTANDARD LVCMOS33 } [get_ports Pb[1]]
set_property -dict { PACKAGE_PIN W19   IOSTANDARD LVCMOS33 } [get_ports Pb[2]]



## SPI configuration mode options for QSPI boot, can be used for all designs
set_property BITSTREAM.GENERAL.COMPRESS TRUE [current_design]
set_property BITSTREAM.CONFIG.CONFIGRATE 33 [current_design]
set_property CONFIG_MODE SPIx4 [current_design]
```

Figure 9: Setting Constraints for FPGA

As you can see all the names of the switches, buttons, clock, and LEDs are exactly the same as the input and output of the topModule function. It has 8 switches which correspond to the 8-bit ‘s’ input, 3 push buttons correspond to the 3-bit input Pb, it has the clock as clk, and 4 LEDs which correspond to the 4-bit output LEDs.

Now when you don’t push any button or switch any switches on the FPGA, you get no lights as shown in Figure 10.

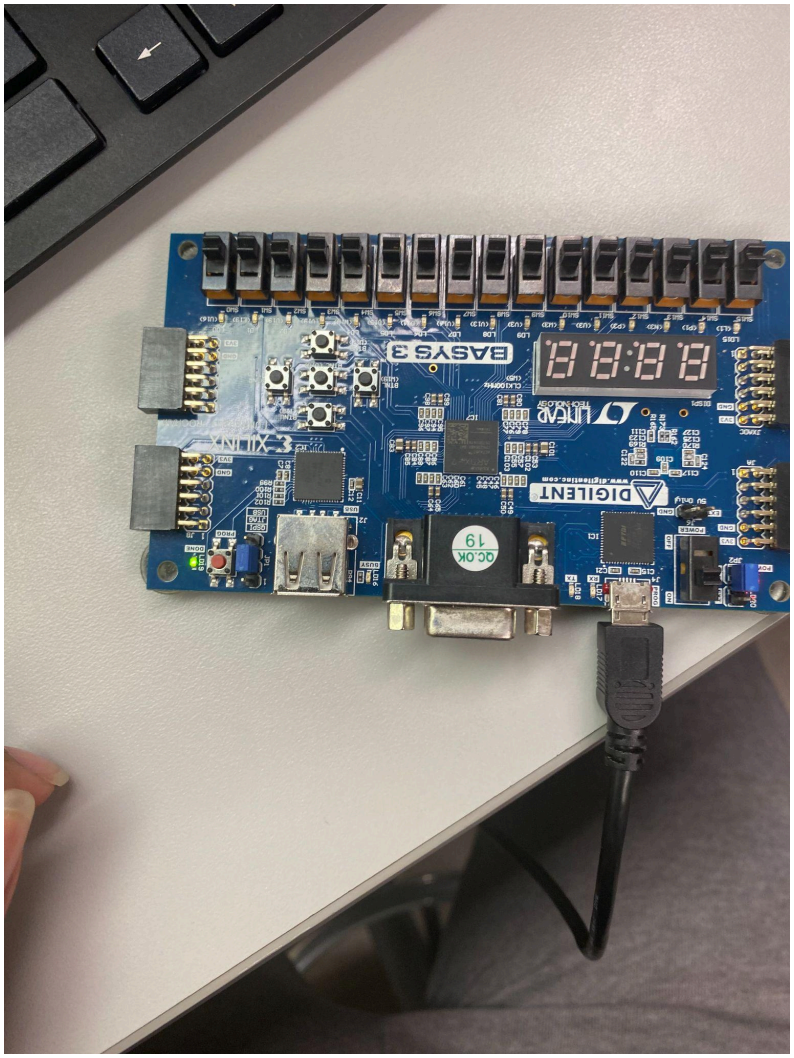

Figure 10: FPGA All Switch Off

That's because if you look at Figure 1[1], all the registers would be at 0 and the switches are off. Now if you flip the switch 2 to 5, LEDs[0] and LEDs[1] are turned on. See Figure 11.

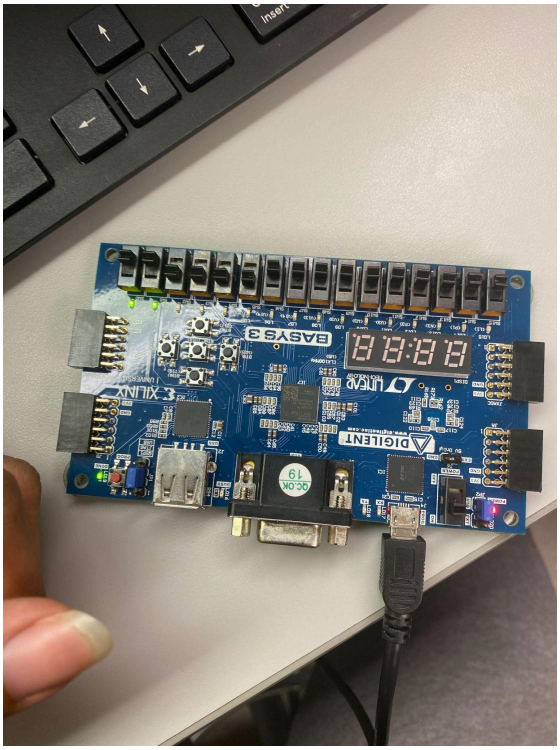

Figure 11: Two LEDs are On

When switches 2 to 5 are flipped, registers A and B will output 1 while register Y will output 0. In the MUX function, when M = 1, the LED corresponding to the M will turn on. Now if you turn on switches 3 to 6, you get the LEDs turned on shown in Figure 12.

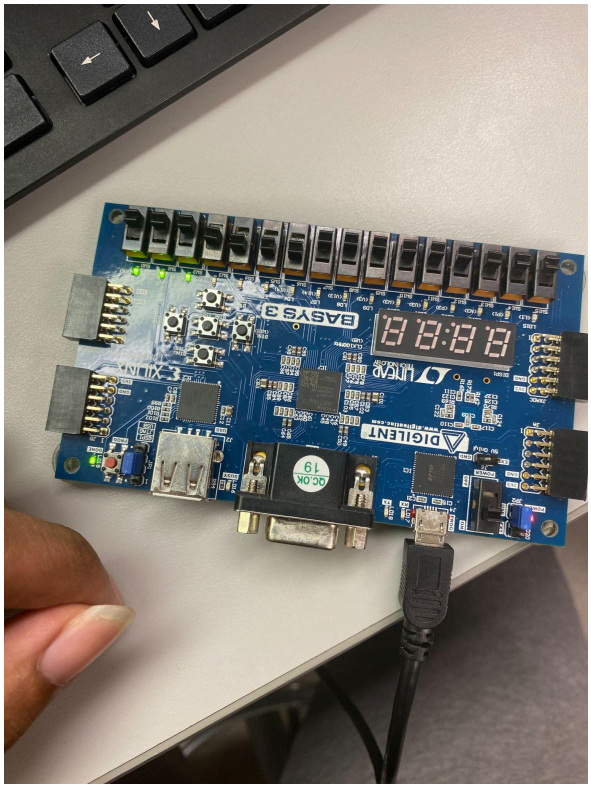

Figure 12: Three LEDs are On

The switches 3 to 6 turn on LEDs[0], LEDs[1], and LEDs[2]. Meaning that the output of regA, regB, and regY is equal to 1. However, the last LED is always off because back in the mux function, the very last M is equal to 0. And now we have finished the lab.

5) Conclusion

Summarizing what we just did, we learned how to use Xilinx Vivado and all its glory. It has many options to choose from including its own SystemVerilog Compiler. You can run simulations of your code and look at the input and output values in a waveform configuration. We learned how to use an FPGA board and set up the buttons, switches, and LEDs using Vivado. We are tasked to write ALU circuits by creating register, ALU, and MUX circuits and wiring them all up to follow the diagram we are given. Then implement the ALU circuit that we just created to the FPGA board so that the LEDs will turn on or off depending on the switches we flip or the buttons we press.

Overall, this Lab was very interesting. I found that the FPGAs were very cool to learn. It's cool to see how the code that we build is implemented into something that we can interact with in real life and not just get a printout of the output. It's a huge step up from our previous labs because all of them are just coding after coding just to get a printout. The Lab manuals weren't that messy, it was very organized. The tutorials and the lab manual are separated into other PDF files which did a great favor of organizing the actual lab manual. In conclusion, this lab was amazing compared to other labs and I hope we get to implement the codes we build or modify into real-life technology.

6) References

1) “EE361L-Lab5-FPGA”, Laboratory handout for EE 361L, University of Hawaii, Fall 2023
2) “Xilinx Vivado Installation Instructions”, Laboratory handout for EE 361L, University of Hawaii, Fall 2023
3) “TutorialArtix-7Basys3”, Laboratory handout for EE 361L, University of Hawaii, Fall 2023.