Lab 5: FPGAs

Deolian Domawa

ECE 361L

November 19, 2024

Abstract: This lab introduces the class to simulating SystemVerilog designs in Xilinx Vivado, synthesizing the designs and programming them onto an FPGA development board. By the end of the lab, students should be able to navigate Vivado to simulate and design circuits, configure constraints, synthesize the design, and program it onto a Basys 3 board.

1 Introduction

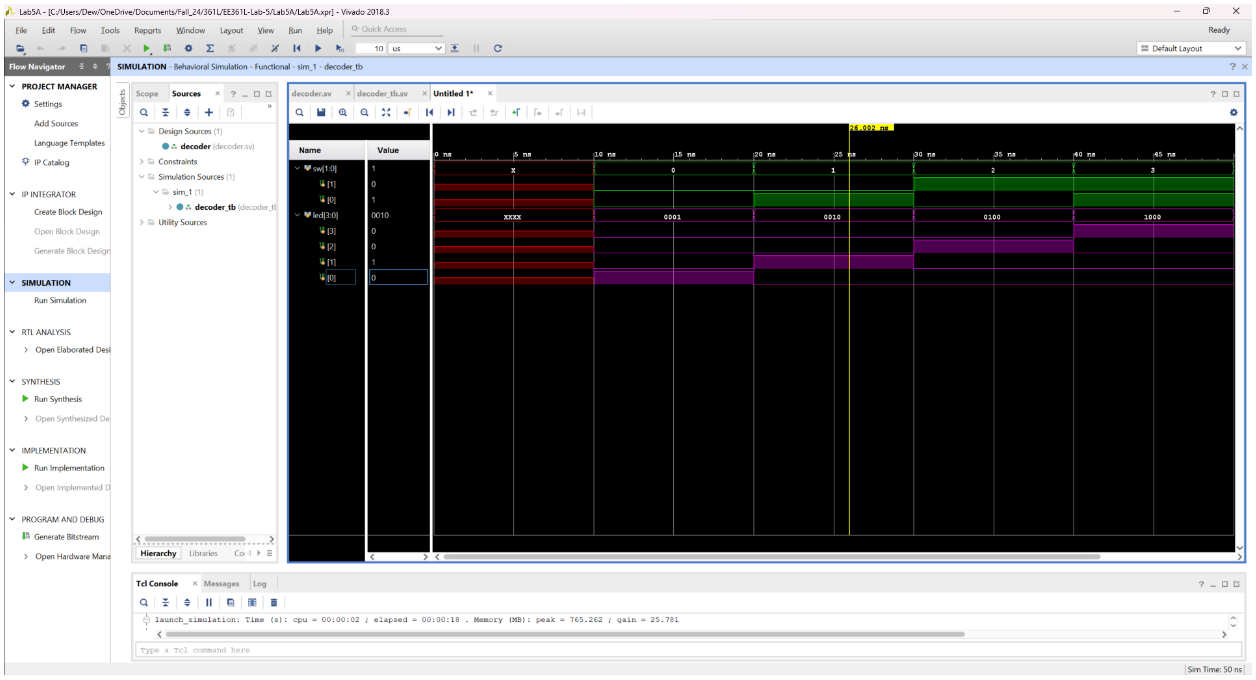
The purpose of ECE 361L Lab 5 [1] lab is to become familiar with Electronic Design Automation using Xilinx Vivado, and programming it onto an FPGA development board.

The main tools for this lab are a laptop, Xilinx Vivado `2018.3` HL WebPACK software edition, and a Digilent Basys 3 board with a USB cord. Section 2 covers the installation process of the software, the simulation of a decoder, and the behavior of the FPGA after being programmed. Section 3 covers programming a sequential circuit made of two modules onto the FPGA. Section 4 covers the design process of an ALU circuit and the behavior of the FPGA after it was programmed. The final section presents the conclusion of this lab report.

2 Decoder

The main portion of this section was setting up the Xilinx Vivado software. This involved having to create a Xilinx account before downloading and installing its archived version `2018.3`.

After installing its HL WebPACK edition with default configurations, an RTL project of a combination circuit, a decoder, was created. Following the lab instructions, the design provided by the manual was programmed in SystemVerilog, including the testbench it was simulated with.


Lab5A - [C:/Users/Dew/OneDrive/Documents/Fall_24/361L/EE361L-Lab-5/Lab5A/Lab5A.xpr] - Vivado 2018.3
SIMULATION - Behavioral Simulation - Functional - sim_1 - decoder_tb
Name
Value
sw[1:0]
led[3:0]
x
0
1
2
3
XXXX
0001
0010
0100
1000
Sim Time: 50 ns

```
launch_simulation: Time (s): cpu = 00:00:02 ; elapsed = 00:00:18 . Memory (MB): peak = 765.262 ; gain = 25.781
```

Figure 1: The waveform output from the decoder simulation

From Fig. 1, it can be seen that the simulation is consistent with how the decoder was designed. The state of the LEDs are unknown for the first 10 nanoseconds due to the switch statement in the design lacking a default case statement. The states change every 10 nanoseconds because the testbench is designed to increment the value of the switch by 1 at that interval. This behavior is observable in the waveform, where each bit of `sw` and `led` changes accordingly. When a bit is true, the waveform rises to a high level, and the area beneath the waveform is highlighted. This makes it visually clear when a bit is active or inactive. As it is a combinational circuit, there are no clocks and thus, the LEDs immediately make a response, behaving like a decoder. In accordance to the testbench, it takes 40 nanoseconds for the switch to reach the binary value of 3 which outputs the LEDs as `1000`.

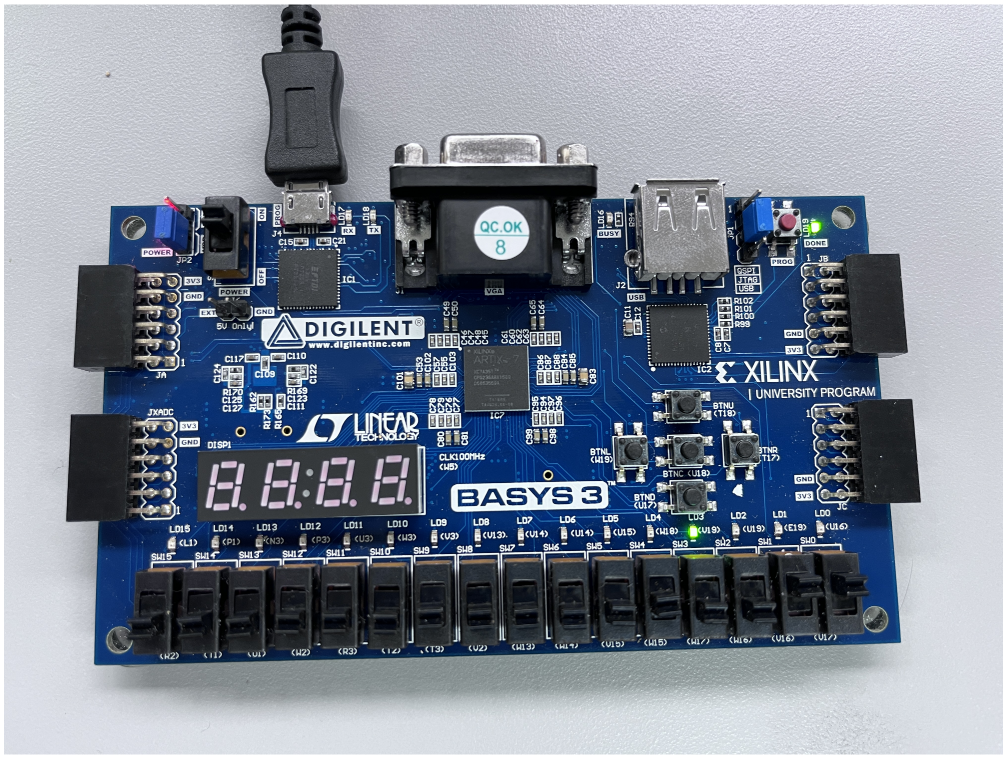


Figure 2: Basys 3 board programmed with the decoder designed in Fig. 1 on the preceding page

Following the constraints instructed by the manual, the decoder was successfully synthesized and programmed onto the FPGA. With the four LEDs on the right of Fig. 2 configured as `led` and the two switches on the right as `sw`, the leftmost led is on when both switches are on.

3 Sequential Circuit

This section is similar to the previous with the exception of the design provided being a sequential circuit. The circuit consists of a 2-bit counter with a clock input and 2-bit output LED. Unlike the previous project, this project was split into two modules, a counter and a timer. The counter increments the LEDs and the timer keeps track if half a second has elapsed using the on-board clock of the FPGA.

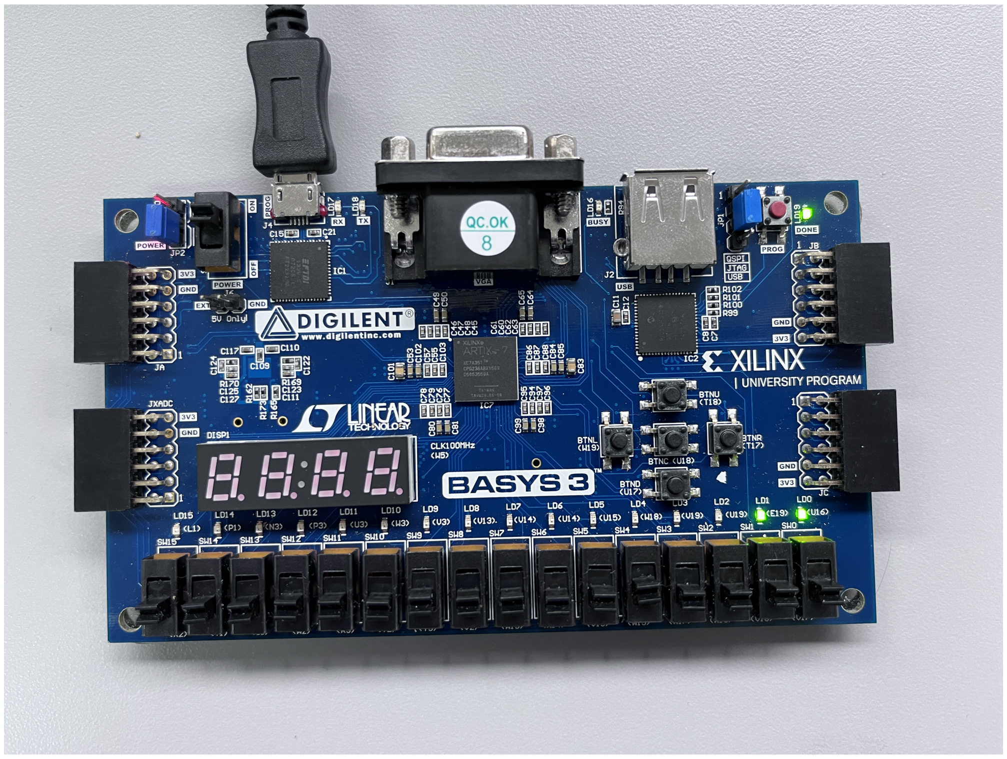


Figure 3: Basys 3 board programmed with the sequential circuit

After the design was synthesized, it was programmed onto the FPGA, and the state of the LEDs incremented like a 2-bit binary without the need for any switches due to the timer that was designed to use the on-board clock as input. When the button that was configured as the reset was pushed, the LEDs restarted and then continued to increment.

4 ALU

This section of the lab implemented an ALU circuit onto the FPGA. It was the longest among all the sections as it involved creating our own design of the circuit.

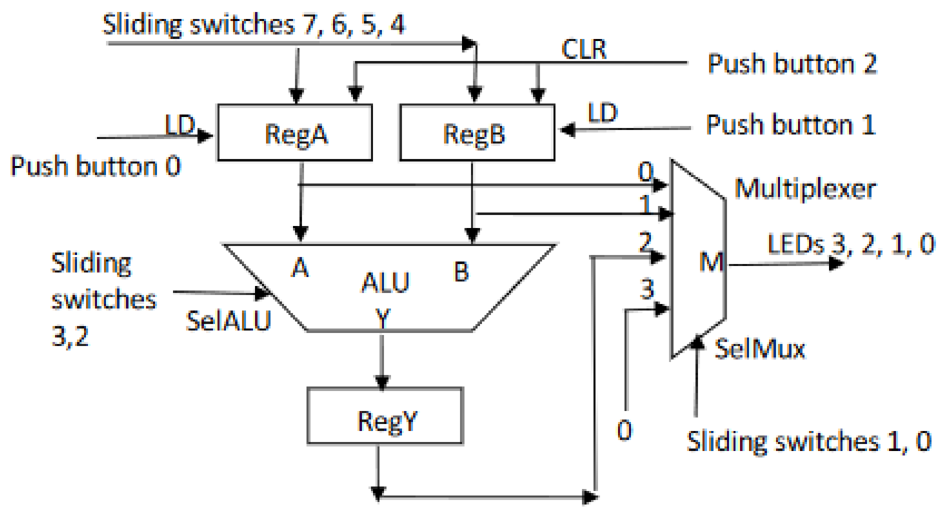


Figure 4: The ALU circuit that was designed

The ALU circuit consisted of three 4-bit registers, one 4-bit ALU with two 4-bit inputs, and a 4:1 multiplexer with each input being 4-bit.

During the design process, a module of the registers were first made. With `RegA` and `RegB` having the same function, only one module was needed as two instances of it can be made in the top module. After it was implemented in the top module, the original code for the register was automatically converted to `RegA` and `RegB` by Vivado.

```
module register (
  input logic [3:0] data_in,
  input logic [1:0] control,  // HOLD, LOAD, CLEAR
  input logic clk,
  output logic [3:0] data_out
);

  logic [3:0] reg_data;

  always_ff @(posedge clk)
    begin
      case (control)
        2'b00: reg_data <= reg_data; // HOLD
        2'b01: reg_data <= data_in; // LOAD
        2'b10: reg_data <= 4'b0000; // CLEAR
        2'b11: reg_data <= 4'b0000; // CLEAR
        default: reg_data <= 4'b0000;
      endcase

      data_out <= reg_data;
    end

endmodule
```

Listing 1: The module for RegA and RegB

With RegY having a more simple function, behaving similarly to a D flip flop, it was made in a separate module.

```
module register_Y (
  input logic clk,
  input logic [3:0] d,
  output logic [3:0] q
);

  always_ff @(posedge clk)
    begin
      q <= d;
  end
endmodule
```

Listing 2: The module for RegY

The third module that was designed was the multiplexer. It was configured to take four 4-bit inputs and select one of them as the single 4-bit output.

```
module mux (
  input logic [3:0] reg_A, reg_B, reg_Y,
  input logic [1:0] selMux,
  output logic [3:0] M
);

  always_comb
    begin
```

```
      case (selMux)
        2'b00: M = reg_A;
        2'b01: M = reg_B;
        2'b10: M = reg_Y;
        2'b11: M = 4'b0000;
        default: M = 4'b0000;
      endcase
    end

endmodule
```

Listing 3: The module for the multiplexer

The final module to be designed was the ALU. This was easily implemented through a behavioral design.

```
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;
        default: Y = 4'b0000;
      endcase
    end

endmodule
```

Listing 4: The module for the ALU

With the submodules designed, the top module was created to connect all of them together, forming the complete ALU circuit.

```
module top (
  input logic [7:0] sw,
  input logic btn0, btn1, btn2,
  input logic clk,
  output logic [3:0] led
);

  wire [1:0] controlA;
  wire [1:0] controlB;
  wire [3:0] regA_A;
  wire [3:0] regB_B;
  wire [3:0] Y_regY;
  wire [3:0] regY_mux;

  assign controlA = {btn2, btn0};
  assign controlB = {btn2, btn1};

  register reg_A (
    .data_in(sw[7:4]),
    .clk(clk),
    .control(controlA),
    .data_out(regA_A)
  );

  register reg_B (
    .data_in(sw[7:4]),
    .clk(clk),
    .control(controlB),
    .data_out(regB_B)
  );

  alu alu (
    .A(regA_A),
    .B(regB_B),
    .selALU(sw[3:2]),
    .Y(Y_regY)
  );

  register_Y reg_Y (
    .clk(clk),
    .d(Y_regY),
    .q(regY_mux)
  );

  mux mux (
    .reg_A(regA_A),
    .reg_B(regB_B),
    .reg_Y(regY_mux),
    .selMux(sw[1:0]),
    .M(led)
  );

endmodule
```

Listing 5: The top module for the complete ALU circuit

The final step before the circuit could be synthesized and programmed onto the FPGA was to configure the constraints such that the ports are connected to the right pins.

```
## Clock
set_property PACKAGE_PIN W5 [get_ports clk]
set_property 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 {sw[0]}]
set_property -dict { PACKAGE_PIN V16 IOSTANDARD LVCMOS33 } [
   get_ports {sw[1]}]
set_property -dict { PACKAGE_PIN W16 IOSTANDARD LVCMOS33 } [
   get_ports {sw[2]}]
```

```
set_property -dict { PACKAGE_PIN W17 IOSTANDARD LVCMOS33 } [
   get_ports {sw[3]}]
set_property -dict { PACKAGE_PIN W15 IOSTANDARD LVCMOS33 } [
   get_ports {sw[4]}]
set_property -dict { PACKAGE_PIN V15 IOSTANDARD LVCMOS33 } [
   get_ports {sw[5]}]
set_property -dict { PACKAGE_PIN W14 IOSTANDARD LVCMOS33 } [
   get_ports {sw[6]}]
set_property -dict { PACKAGE_PIN W13 IOSTANDARD LVCMOS33 } [
   get_ports {sw[7]}]

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

## Buttons
set_property -dict { PACKAGE_PIN U18 IOSTANDARD LVCMOS33 } [
   get_ports btn0]
set_property -dict { PACKAGE_PIN T18 IOSTANDARD LVCMOS33 } [
   get_ports btn1]
set_property -dict { PACKAGE_PIN W19 IOSTANDARD LVCMOS33 } [
   get_ports btn2]

## Configuration options, can be used for all designs
set_property CONFIG_VOLTAGE 3.3 [current_design]
set_property CFGBVS VCCO [current_design]

## 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]
```

Listing 6: The constraints for the ALU FPGA design

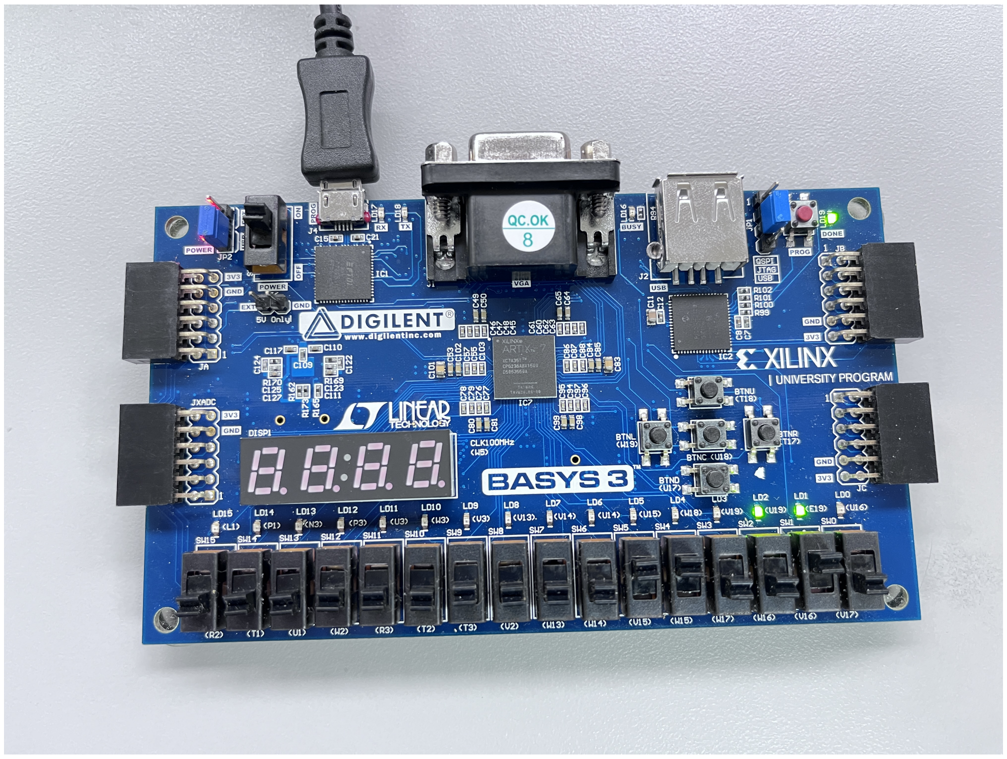


Figure 5: Basys 3 board programmed with the ALU circuit

In Fig. 5, a value of 3 was loaded into registers A and B. The multiplexer was set to select the output from register Y, and the operator selected by the ALU was the addition operator. According to Fig. 4 on page 6, the output of the circuit should be the binary value `0110`. As displayed above, the LEDs that are turned on comply with our design.

5 Conclusion

I successfully completed the assignment and gained a solid understanding of using Xilinx Vivado to simulate our designs, and synthesize and program them onto an FPGA. I also acquired a stronger grasp of designing large circuits through smaller modules. I did not encounter any issues throughout the entirety of the labs and was satisfied with my learning experience.

References

[1] “EE 361L, Lab 5 FPGAs,” Laboratory handout for ECE 361L, University of Hawaii, Fall 2024.