Mauro Lopez

LAB 1 Report

ECEN-449

Section 502

September 13, 2018

Introduction:

For this lab are goal is to apply our knowledge of Verilog and learn how to implement our program into an FPGA board.

Procedure:

To begin the lab, we can start off by login into our terminal and launching Vivado. Once we have Vivado running we can create a new project and call it “Lab 1”. Once we have our project created. We will have several useful tabs. For our first part of the lab we will only focus on adding a new Verilog file and using the text editor to add our code.

Our first assignment is very simple and only requires you to follow our lab1 manual. This first assignment is only mapping each switch to an LEDS. The code below is what its ending result:

```
module switch(
   input [3:0] SWITCHES, //creates a 4 bit input
   output [3:0] LEDS // creates a 4 bit output.
   );
   assign LEDS[3:0] = SWITCHES[3:0];// assign the 4 bits from the SWITCHES
   // to the LEDS
endmodule
```

Now that we have created our Verilog code, we can start to work on our .xdc file. This file is what will embed our program into our FPGA board.

We are given the following code for part 1:

```
##Switches


set_property PACKAGE_PIN R18 [get_ports {BUTTONS[0]}]
set_property IOSTANDARD LVCMOS33 [ get_ports {BUTTONS[0]}]


set_property PACKAGE_PIN P16 [get_ports {BUTTONS[1]}]
set_property IOSTANDARD LVCMOS33 [ get_ports {BUTTONS[1]}]


set_property PACKAGE_PIN V16 [get_ports {BUTTONS[2]}]
set_property IOSTANDARD LVCMOS33 [ get_ports {BUTTONS[2]}]



##LEDS


set_property PACKAGE_PIN M14 [ get_ports {LEDS[0]}]
set_property IOSTANDARD LVCMOS33 [ get_ports {LEDS[0]}]

set_property PACKAGE_PIN M15 [ get_ports {LEDS[1]}]
set_property IOSTANDARD LVCMOS33 [ get_ports {LEDS[1]}]

set_property PACKAGE_PIN G14 [ get_ports {LEDS[2]}]
set_property IOSTANDARD LVCMOS33 [ get_ports {LEDS[2]}]

set_property PACKAGE_PIN D18 [ get_ports {LEDS[3]}]
set_property IOSTANDARD LVCMOS33 [ get_ports {LEDS[3]}]

set_property PACKAGE_PIN L16 [get_ports CLOCK ]
set_property IOSTANDARD LVCMOS33 [ get_ports CLOCK]
```

At a glance it might seemed easy to understand what its happening. We can see that we are mapping the number of our Switches to our variable SWITCHES. The same can be said about our LEDS.

Once we have both our .v and our .xdc, we can generate our bit stream and embed it into our FPGA board.

If you successfully completed the task, you can see that when ever you turn on a switch, an LED will turn on .

PART2:

Now that we have a basic understanding of vivado and our FPGA board we can work on part 2.

Before jumping into the programing, Its very important to have an understanding of what we are supposed to do. This will lead to less time debugging and a deeper understanding of Verilog.

Part 2 of the lab is implementing a 4-bit counter that counts at a rate of 1 unit per second. If our button 0 is pressed it will count up, if our button 1 is pressed it will count down, and also to include a button 2 to reset our counter.

One thing to keep in mind when writing our .v file, is to include a clock divider. The question is how do we go about building this clock divider. We are given that our clock has a frequency of 125MHz. This is way to fast, and if we use this, we wont be able to visualize what is going on. This is where our module for a clock divider comes into play.

My approach to this was trial an error at first. Meaning I used a high binary number to create a new clock cycle. However, that approach was very inefficient. To solve this, I decided to create a counter that would count up to X, once that number was reached, it would send out a high signal, and reset the counter. Once the counter reached X again, it would invert the signal, in our case a low signal. This was the output file final result:

```
module clock_divider(
    output Divider,
    input CLOCK
    );
    reg [29:0] timer=0;
    reg Divider =0;
    always @(posedge CLOCK) begin
    //FFFFFFFFFFFFFF000
    //FFFFFFFFFF
    //7735940 1 second
    //BEBC20 .1 second
    /// h5F5E100
    if(timer<= 29'hBEBC20)begin
    timer<=timer+1 ;
    end

    else begin
    timer<=0;
     if(Divider==1'b1)begin
       Divider<=0;    end
       else if (Divider==0) begin
       Divider<=1;
       end
     end

    end
endmodule
```

with our clock divider created we can use this as our new clock input for our counter.

```
module four_bit_counter(
    output [3:0] LEDS,
    input [2:0] BUTTONS,
    input CLOCK
    //button[0] = up
    //button[1] = down
    );
    reg [3:0] Count=0;
    wire Divider; // our new clock

clock_divider clk(Divider,CLOCK); // we get our new clock signal

 always@(posedge Divider || BUTTONS[2])begin
    if (BUTTONS[2])begin
    Count<=0;// button 2 will create a reset
    end
    else if (BUTTONS[0]==1) begin
             Count=Count + 1; // button 0 will count up
         end

    else if (BUTTONS[1]==1) begin
             Count= Count -1; // button 1 will count down
         end

 end
 assign LEDS[3:0]= Count[3:0]; // our LEDS will reflect our Count
endmodule
```

Here we can see that our buttons are placed in ranks, meaning our button 2 will have a higher priority, since its our reset.

With this complete we can start writing our .xdc file.This file will look familiar to our Switch file since we are still using the same LEDS, but now we are adding Buttons rather than switches.

```
##BUTTONS


set_property PACKAGE_PIN R18 [get_ports {BUTTONS[0]}]

set_property IOSTANDARD LVCMOS33 [ get_ports {BUTTONS[0]}]


set_property PACKAGE_PIN P16 [get_ports {BUTTONS[1]}]

set_property IOSTANDARD LVCMOS33 [ get_ports {BUTTONS[1]}]


set_property PACKAGE_PIN V16 [get_ports {BUTTONS[2]}]

set_property IOSTANDARD LVCMOS33 [ get_ports {BUTTONS[2]}]
```

```
##LEDS

set_property PACKAGE_PIN M14 [ get_ports {LEDS[0]}]
set_property IOSTANDARD LVCMOS33 [ get_ports {LEDS[0]}]

set_property PACKAGE_PIN M15 [ get_ports {LEDS[1]}]
set_property IOSTANDARD LVCMOS33 [ get_ports {LEDS[1]}]

set_property PACKAGE_PIN G14 [ get_ports {LEDS[2]}]
set_property IOSTANDARD LVCMOS33 [ get_ports {LEDS[2]}]

set_property PACKAGE_PIN D18 [ get_ports {LEDS[3]}]
set_property IOSTANDARD LVCMOS33 [ get_ports {LEDS[3]}]

set_property PACKAGE_PIN L16 [get_ports CLOCK ]
set_property IOSTANDARD LVCMOS33 [ get_ports CLOCK]
```

you can see that at the end we added our clock, this is important to add, other wise we will not get any changes.

With this we can now generate our bitstream and embed it into our FPGA board.

Part 3:

Now that we have our clock module, we can re-use it for our part 3 of the lab. This part is creating a simple game of jackpot. We are essentially supposed to guess which LED will light up next, Flip the switch and our switch matches, we get a jackpot and we get flashing lights.

Like before, it’s important to create a design that can work, rather than just jumping right into writing the program.

```verilog
module jackpot(
output [3:0] LEDS,
input [3:0] SWITCHES,
input BUTTONS,
input CLOCK
    );
    wire Divier; // new clock signal
    reg [4:0] Exp= 0; // exponential value
    reg [4:0] Count =0; // our count
    reg Winner=0; // trigger flashing lights.
    reg fail=0; //prevents from leaving it on and winning.
    clock_divider Clk(Divider,CLOCK); // assign our clock signal

  always@(posedge Divider)begin
  if(BUTTONS) begin // reset button.
  Winner<=0;
  end
  if(Winner==1'b1)begin // if winner is high display light show.
    if(Count==4'b1111)begin
    Count<=0; //off
    end
    else
    Count<=4'b1111; //on
    end

  else if(SWITCHES[3:0]==Count[3:0]&& SWITCHES!=4'b0000 && fail!=1'b1)begin
  Winner<=1'b1;
  end
  else begin
  Exp<=Exp+1;
  Count<= 2**Exp; // allows to only light 1 light at a time.
  if(Count==2'b01000)begin
  Exp<=0; //resets counts
  end
  if(SWITCHES!=4'b0000)begin // sends fail signal
  fail=1'b1;
  end
  else
  fail=1'b0;
  end

end
     assign LEDS[3:0]= Count[3:0];
endmodule
```

looking at the code you can see that one of our requirements is to only have 1 LED on at a time. To solve this problem, I made my counter count by the power of two on a binary based scale. This solved the issue, and then all I had to do was make sure my switches match to my LEDs. When I tried to implement this, I got a bug where jackpot was on as soon as the FPGA was on. The reason behind this, was the fact that I didn't exclude the value of zero for the switches. You can also see that I'm using the same clock divider for this except I made it a bit faster.

Once again we have to write our .xdc file, here we can see that we are back to using switches, so it will look similar to our first switch.xdc .

```
##Switches

set_property PACKAGE_PIN G15 [get_ports {SWITCHES[0]}]

set_property IOSTANDARD LVCMOS33 [ get_ports {SWITCHES[0]}]


set_property PACKAGE_PIN P15 [get_ports {SWITCHES[1]}]

set_property IOSTANDARD LVCMOS33 [ get_ports {SWITCHES[1]}]


set_property PACKAGE_PIN W13 [get_ports {SWITCHES[2]}]

set_property IOSTANDARD LVCMOS33 [ get_ports {SWITCHES[2]}]


set_property PACKAGE_PIN T16 [get_ports {SWITCHES[3]}]

set_property IOSTANDARD LVCMOS33 [ get_ports {SWITCHES[3]}]


set_property PACKAGE_PIN R18 [get_ports {BUTTONS}]

set_property IOSTANDARD LVCMOS33 [ get_ports {BUTTONS}]


##LEDS


set_property PACKAGE_PIN M14 [ get_ports {LEDS[0]}]

set_property IOSTANDARD LVCMOS33 [ get_ports {LEDS[0]}]


set_property PACKAGE_PIN M15 [ get_ports {LEDS[1]}]

set_property IOSTANDARD LVCMOS33 [ get_ports {LEDS[1]}]


set_property PACKAGE_PIN G14 [ get_ports {LEDS[2]}]

set_property IOSTANDARD LVCMOS33 [ get_ports {LEDS[2]}]


set_property PACKAGE_PIN D18 [ get_ports {LEDS[3]}]
set_property IOSTANDARD LVCMOS33 [ get_ports {LEDS[3]}]


set_property PACKAGE_PIN L16 [get_ports CLOCK ]

set_property IOSTANDARD LVCMOS33 [ get_ports CLOCK]
```

Once we have both our .v and .xdc file we can generate our bitstream and embed it into our FPGA boards.

Results:

Part 1: switches

The program worked as expected and no bugs were found during the testing process and demo.

Part 2: **4-bit counter**

The program worked according to the specs given in the lab manual.

Part 3: **Jackpot**

The program worked with no bugs or sudden failures. Met all requirements of the assignment.

Conclusion:

In this lab we started by mapping our switches from our FPGA board to it's LEDS. We did this by learning to create a Verilog file and a .xdc file. From this we used our knowledge of Verilog and notes from lectures to create a 4-bit counter, by implementing our own clock divider module. Finally, we combined all of our resources and worked on creating a Jackpot game. This game involved using previous modules such as our clock divider and our counter.

Questions:

How are the use push-buttons wired on the ZYBO board (i.e. what pins on the FPGA do each of them correspond to and are the signals pulled up or down)?

From our lab we learn that our mains 4 buttons are in the pins from left to right on Y16, V16, P16, and R18. The buttons are pull up, since every press on the buttons creates a high signal and goes back to low as soon as its release.

What is the purpose of an edge detection circuit and how should it have been used in this lab?

The purpose of edge detection, is to avoid using a high signal more than once. For instance, an edge detection will only be active for a really short time, compare to being high for a cycle. We can use this in the lab by making our buttons be on positive edge detection, so that the counter reacts faster when the button is pressed.