Lab 1: Using Vivado

Abhinav Subramanian

ECEN 449-502

Due date: 1/28/2020

Introduction:

The aim of this lab was to refresh our memory of Verilog and to get us used to the Vivado development environment. It also taught us how to program the ZYBO FPGA board using Vivado. In the end, we used this knowledge to make a 4 bit counter and a “jackpot” game.

Procedure:

- Open Vivado and create a project called lab1. Create a new file called switch.v.
- Copy the code given in the manual to the file.
- Copy the .xdc code to the file and add it as a design constraint. Now follow the steps given in the manual to program the zybo.
- Copy the four bit counter referenced in appendix to a file fourbitcounter.v, and copy the top level counter to toplevelcounter.v.
- Program the zybo and verify that the design is working correctly.
- Copy the code under the jackpot section of appendix to a file jackpot.v
- Program the zybo and verify that the design is working correctly.

Results:

The programs worked more or less as expected. One issue I did run into was implementing an edge detector to prevent a false win. I wrote several designs that were syntactically correct, but would not synthesize. I also wrote a design that implemented a separate edge detector module, and although it synthesized, it didn’t seem to be doing anything when I flipped the switches. Ultimately, I was unable to fix the issue.

As far as screenshots go, I only tested my design on the FPGA and never actually simulated it. This is probably not the best idea going forward as it takes time to generate a bitstream and program the ZYBO, but these designs were simple enough that following this practice was entirely feasible.

Conclusion:

In this lab, I got a good refresher on using Verilog. I also learned that even Verilog that might be syntactically correct and simulate correctly might not actually synthesize, because these envisioned designs are ambiguous for the bitstream to generate in real life.

Postlab questions:

a) Buttons 0, 1, 2 and 3 correspond to pins K18, P16, K19 and Y16 on the FPGA, respectively. They are pulled down because when the switches are open, the output is low.

b) An edge detection circuit is used to detect whenever an input changes from 0 to 1/1 to 0. It should have been used to prevent false wins on the jackpot (i.e. flipping the switch ahead of time and waiting for the LEDs to reach it, thus winning the game).

Appendix:

Toplevelcounter.v:

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 01/21/2020 11:39:04 AM
// Design Name:
// Module Name: toplevelcounter
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
```

```
module toplevelcounter(
   input CLOCK,
   input [3:0] BUTTONS,
   output [3:0] LEDS
   );
   reg dividedClock;
   reg [25:0] clockcount;
   initial begin
    dividedClock<=0;
    clockcount<=26'b0;
   end
   //clock divider
   always @(posedge CLOCK) begin
    if(clockcount==26'b11101110011010110010100000) begin
      dividedClock=~dividedClock;
      clockcount=26'b0;
    end
    else
      clockcount=clockcount+1;
   end
   fourbitcounter count0(.CLOCK(dividedClock), //instantiate counter.
                  .BUTTONS(BUTTONS),
                  .LEDS(LEDS));

endmodule
```

fourbitcounter.v:

```
module fourbitcounter(
```

```
    input CLOCK,
    input [3:0] BUTTONS,
    output reg [3:0] LEDS
    );
    always @ (posedge CLOCK) begin
      if(BUTTONS[2])//reset
        LEDS=4'b0000;
      else if(BUTTONS[0]) //count forward
        LEDS=LEDS+1;
      else if (BUTTONS[1]) //count backward
        LEDS=LEDS-1;
    end
endmodule
```

jackpot.v:

```
module jackpot(
    input [3:0] SWITCHES,
    output [3:0] LEDS,
    input CLOCK,
    input [3:0] BUTTONS
    );
    reg dividedClock;
    reg [24:0] clockcount;
    initial begin
      dividedClock<=0;
      clockcount<=25'b0000000;
    end
    //clock divider
```

```
always @(posedge CLOCK) begin
  if(clockcount==25'b11111110100000100100000000) begin
    dividedClock=~dividedClock;
    clockcount=25'b0000000;
  end
  else
    clockcount=clockcount+1;
end
reg [3:0] count;
initial begin
  count=4'b0001;
end

always@(negedge dividedClock or posedge SWITCHES) begin
  if((count==4'b1111)&&(BUTTONS[0]==1'b1)) //reset
    count<=4'b0001;
  else if(count==4'b1111) //lock in the win
    count<=4'b1111;
  else if((SWITCHES==count)) //register a win
    count<=4'b1111;
  else if(count==4'b0001) //shift the LEDs
    count<=4'b0010;
  else if(count==4'b0010)
    count<=4'b0100;
  else if(count==4'b0100)
    count<=4'b1000;
  else if(count==4'b1000)
```

```
        count<=4'b0001;
    end

    assign LEDS=count;


endmodule
```