

# **ECEN 449 - Lab Report**

**Lab Number: 1**

**Lab Title: Using Vivado**

**Section Number: 511**

**Student's Name: Kris Gavvala**

**Student's UIN: 929000158**

**Date: 2/12/2025**

**TA: Gautham Nemani**

## **Introduction:**

The lab provided hands-on experience with Xilinx FPGA design flow through the Vivado development environment. Using an FPGA (ZYBO Z7-10 board), basic hardware implementation was developed to control LED outputs based on DIP switch inputs, and buttons. Using Verilog HDL, the control logic for a counter and a jackpot game was developed and then simulated using the ZYBO Z7-10.

## Procedure:

### Part 1: Switch – Setting up Vivado, FPGA connection and implementing LED switches.

#### 1. Initial Setup and Project Creation

In Vivado, create a new RTL project with Verilog as target language. Select the Zynq7000 FPGA device. Define module ports: 4-bit input (SWITCHES) and 4-bit output (LEDS). Verify board connection through Hardware Manager.

#### 2. Code Implementation

Create the switch.v design and edit the module to assign LED outputs based on switch inputs with a continuous statement: `assign LEDS = SWITCHES`. Create and add switch.xdc constraint file with FPGA pin assignments using the manual for the ZYBO Z7-10. LEDS[3:0] go to FPGA pins M14, M15, G14, D18. SWITCHES[3:0] go to pins G15, P15, W13, and T16.

#### 4. Bitstream Generation and Programming

Generate bitstream through Vivado. Program FPGA with the generated bitstream. Verify that each LED lights up when its assigned switch is flipped.

### Part 2: 4-Bit Counter

- Implement a 4-bit counter with:
  - 1-second update rate
  - Up/down counting controlled by BTN0/BTN1 ◦ Reset functionality with BTN2

Create the counter.v design in vivado and declare the i/o ports for BUTTONS, LEDS, CLOCK, and RESET. Then create the counter.xdc file and connect the FPGA pins for the clock (K17), BUTTONS[1:0] and RESET (K18, P16, K19 respectively), along with the LEDs.

To update the leds every second, the CLOCK signal from the FPGA has to be divided from 125 MHz down to 1 Hz signal.

Clock implementation:

- Always block triggered by a positive edge of 125 MHz CLOCK signal
  - Flip a Boolean reg (clk) every 62.5 million cycles. A signal that rises (goes to 1) every second is desired, so flipping clk every 125 million cycles gives a signal that goes high every two seconds and  $\left(\frac{125 \cdot 10^6}{2}\right) = 6.25 \cdot 10^7$  so the clk signal should flip every 62.5 million CLOCK cycles.
- Have a count reg to store the current count for the LEDs
- Use an Always block triggered by a positive edge of the new clk signal to check whether button 0, button 1, or reset (button 2) is pressed with an if-else block statement. Increase, decrease or zero the count every second if button 0, button 1, or reset respectively are pressed.
- Use continuous logic to assign leds to the count reg: `assign LEDS = count`.

## Part 3: Jackpot Game

- Create a "Jackpot" game where:
  - LEDs light sequentially
  - Players must flip the switch matching the lit LED ◦ Winning triggers all LEDs to light up

Implementation:

- Use a clock edge to iteratively turn on a single LED at a specific time interval.
  - Use an always block to divide the clock down to a slower speed that is observable. Use the same implementation as the counter except flip the desired clk signal every 30 million CLOCK cycles so that you get around 2 Hz for clk signal.
- Keep track of game state variables using
  - Lites: to keep track of which LED is currently lit.
  - count: A counter that cycles through the LED pattern.
  - game\_active: Indicates whether the game is running or paused.
  - jackpot\_mode: boolean signal to indicate if the player won the game (activates jackpot mode).
- Implement the game logic with the new clk signal
  - Use the count reg to cycle from 1 to 4 continuously
  - Use bit shifting by bitwise shifting ( $\ll 1$ ) the lites register
  - If the player hasn't won yet, the LEDs keep shifting left every 0.5 seconds. When count == 3 (after 4 shifts), it resets back to 0001, looping the pattern.
  - If game\_active is high it checks if the player pressed the correct switch (SWITCHES[3:0] matches LEDS[3:0]).
    1. If correct: All LEDs light up (1111), and the game enters jackpot mode.
    2. If incorrect: The LED pattern continues shifting.
  - If the player has already won (jackpot\_mode is high), wait for the player to reset the switches (SWITCHES == 0000).
  - If all switches get turned off, rest goes high and the game restarts, setting lites back to 0001 and count to 0.

## Results:

The results of this lab can be shown through the demonstration. For part 1, every LED turned on when a switch was turned on. For part 2 we had to learn how the clock signal worked and how to implement a clock divided with behavioral Verilog. There was a good amount of information learned about how to create synthesizable code like where you can or can't use regs and nonblocking vs blocking assignments. For the last problem I also learned about the bit shift Verilog function.

## Conclusion:

This lab provided a hands-on approach for applying introductory knowledge of Verilog HDL. It also introduced the FPGA as a method to test logic implementation. Through this lab, we became familiar with the Xilinx FPGA design flow and Vivado to implement and debug Verilog modules. We practiced using behavioral and continuous Verilog to create sequential and combinational logic functions and learned good practices for

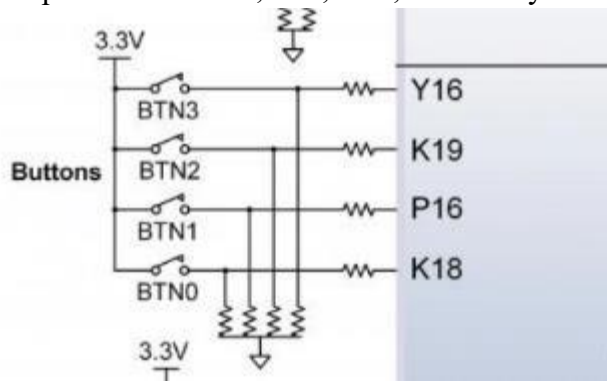
creating synthesizable verilog. We also became familiar with using the clock to synchronize events and implement edge-triggered logic.

### Post-lab Deliverables:

3. [4 points.] Answers to the following questions:

(a) How are the user push-buttons wired on the ZYBO Z7-10 board (i.e. what pins on the FPGA do each of them correspond to and are the signals pulled up or down)? You will have to consult the Master XDC file for this information.

Answer: button pins 0-3 are K18, P16, K19, Y16. They are all in a pull-down configuration.



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

An edge detection circuit is used to detect when a signal transitions from one state to another like from low to high (rising edge) or from high to low (falling edge). This is to ensure that certain actions like triggering an event, counting pulses, or registering button presses only occur once per transition. In this lab, where push-buttons are used as inputs, edge detection circuits were implemented to detect button presses accurately and trigger actions only on the positive rising edge of a clock to provide synchronization for events.

## Appendix

### Part 1 code: switch.v and switch.xdc

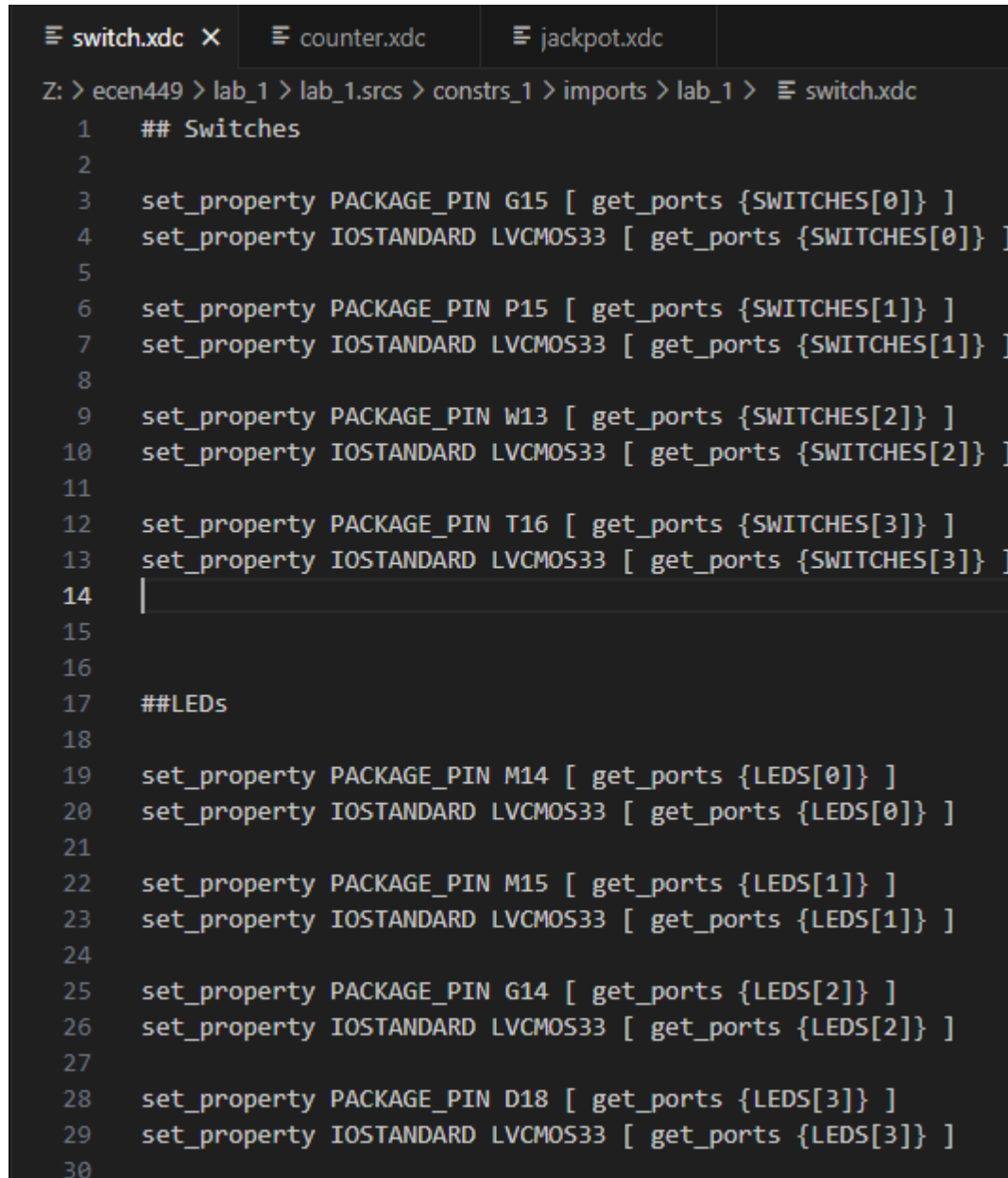
Figure 1: switch.v

```
module switch(SWITCHES,LEDS);
    input [3:0] SWITCHES;
    output [3:0] LEDES;

    assign LEDES[3:0] = SWITCHES[3:0];

endmodule
```

Figure 2: switch.xdc



```
switch.xdc X counter.xdc jackpot.xdc
Z: > ecen449 > lab_1 > lab_1.srcs > constrs_1 > imports > lab_1 > switch.xdc
1  ## Switches
2
3  set_property PACKAGE_PIN G15 [ get_ports {SWITCHES[0]} ]
4  set_property IOSTANDARD LVCMOS33 [ get_ports {SWITCHES[0]} ]
5
6  set_property PACKAGE_PIN P15 [ get_ports {SWITCHES[1]} ]
7  set_property IOSTANDARD LVCMOS33 [ get_ports {SWITCHES[1]} ]
8
9  set_property PACKAGE_PIN W13 [ get_ports {SWITCHES[2]} ]
10 set_property IOSTANDARD LVCMOS33 [ get_ports {SWITCHES[2]} ]
11
12 set_property PACKAGE_PIN T16 [ get_ports {SWITCHES[3]} ]
13 set_property IOSTANDARD LVCMOS33 [ get_ports {SWITCHES[3]} ]
14
15
16
17 ##LEDs
18
19 set_property PACKAGE_PIN M14 [ get_ports {LEDES[0]} ]
20 set_property IOSTANDARD LVCMOS33 [ get_ports {LEDES[0]} ]
21
22 set_property PACKAGE_PIN M15 [ get_ports {LEDES[1]} ]
23 set_property IOSTANDARD LVCMOS33 [ get_ports {LEDES[1]} ]
24
25 set_property PACKAGE_PIN G14 [ get_ports {LEDES[2]} ]
26 set_property IOSTANDARD LVCMOS33 [ get_ports {LEDES[2]} ]
27
28 set_property PACKAGE_PIN D18 [ get_ports {LEDES[3]} ]
29 set_property IOSTANDARD LVCMOS33 [ get_ports {LEDES[3]} ]
30
```

## Part 2: counter1.v and counter.xdc

```
module counter1(CLOCK,RES,BUTTONS,LEDS);  
    input CLOCK;                // FPGA clock  
    input RES;                  // Button 2 on FPGA  
    input [1:0] BUTTONS;        // Buttons 0 and 1 on the FPGA  
    output [3:0] LEDS;          // LEDs 0-3 on the FPGA  
    reg [31:0] clockc = 0;       //clock counter to count up to 62.5 million  
    reg [3:0] count = 4'b0000;   //intermediate reg to calculate led value  
    reg clk1hz = 0;              //desired clock signal  
  
    always@ (posedge(CLOCK))    //clock divider  
  
    begin  
        clockc <= clockc + 1;  
  
        if (clockc == 62500000) begin  
  
            clk1hz <= ~clk1hz;    //on every 62.5 E6 positive edges of CLOCK, turn clk1hz  
            clockc <= 0;          // zero the clock counter to count for the next turn of clk1hz  
  
        end  
  
    end  
  
    end  
  
    always @( posedge(clk1hz) )begin    // on every positive edge of clk1hz calculate the led values  
        if (RES)begin                  // if reset(B2) is active, reset the count for leds to 0
```

```

        count <= 4'b0000;

end

else if (BUTTONS[0]) begin           //if B0 is pressed increment the count
    count <= count + 1;
end

else if (BUTTONS[1]) begin           // if B1 is pressed decrement the count
    count <= count - 1;
end

end

assign LEDS[3:0]= count[3:0];        // assign leds with continuous statement to count

endmodule

```

Figure 3: counter.xdc

```

1  #Buttons
2
3  set_property PACKAGE_PIN K18 [ get_ports {BUTTONS[0]} ]
4  set_property IOSTANDARD LVCMOS33 [ get_ports {BUTTONS[0]} ]
5
6  set_property PACKAGE_PIN P16 [ get_ports {BUTTONS[1]} ]
7  set_property IOSTANDARD LVCMOS33 [ get_ports {BUTTONS[1]} ]
8
9  set_property PACKAGE_PIN K19 [ get_ports {RES} ]
10 set_property IOSTANDARD LVCMOS33 [ get_ports {RES} ]
11
12 ##Clock
13
14 set_property PACKAGE_PIN K17 [ get_ports CLOCK ]
15 set_property IOSTANDARD LVCMOS33 [ get_ports CLOCK ]
16
17
18 ##LEDs
19
20 set_property PACKAGE_PIN M14 [ get_ports {LEDS[0]} ]
21 set_property IOSTANDARD LVCMOS33 [ get_ports {LEDS[0]} ]
22
23 set_property PACKAGE_PIN M15 [ get_ports {LEDS[1]} ]
24 set_property IOSTANDARD LVCMOS33 [ get_ports {LEDS[1]} ]
25
26 set_property PACKAGE_PIN G14 [ get_ports {LEDS[2]} ]
27 set_property IOSTANDARD LVCMOS33 [ get_ports {LEDS[2]} ]
28
29 set_property PACKAGE_PIN D18 [ get_ports {LEDS[3]} ]
30 set_property IOSTANDARD LVCMOS33 [ get_ports {LEDS[3]} ]
31

```

### Part 3: jackpot.v and jackpot.xdc

```
module jackpot(SWITCHES, LEDS, CLOCK);  
    input [3:0] SWITCHES;          // fpga switches  
    input CLOCK;                   // fpga clock  
    output reg [3:0] LEDS;         // fpga leds  
  
    reg [31:0] clockc = 0;         //clock counter to count up to 62.5 million  
    reg [3:0] lites = 4'b0001;     //intermediate reg to assign led value  
    reg [2:0] count = 0;           // reg to track which leds are lit  
    reg clk1hz = 0;                // desired clock signal  
    reg game_active = 1;           // Boolean to check if the game is running  
    reg jackpot_mode = 0;          // Boolean to see if a jackpot has been made  
  
    always@ (posedge(CLOCK))       // clock divider  
  
    begin  
        clockc <= clockc + 1;  
  
        if (clockc == 30000000) begin  
  
            clk1hz <= ~clk1hz;     // flip clk1hz roughly every quarter second so its high every 0.5 seconds  
            clockc <= 0;  
  
        end  
  
    end  
  
end
```



```

always@ (posedge(clk1hz)) begin          // led logic

    if (jackpot_mode) begin              // if jackpot is true and all switches are down, restart the game
        if (SWITCHES == 4'b0000) begin
            jackpot_mode <= 0;            // reset jackpot_mode for the next game
            game_active <= 1;             // set game_active high
            lites <= 4'b0001;             // initialize leds for cycling sequence
            count <= 0;                   // set count back to zero indicating led0 is lit
        end
    end

end

else if (game_active) begin              // while the game is still running and no jackpot has been detected
    if (SWITCHES == lites) begin
        LEDS <= 4'b1111;
        jackpot_mode <= 1;               // if switches equal led values, jackpot is high and game is suspended
        game_active <= 0;
    end

end

else begin                                //normal led shifts while the game is running and switches != lites
    if (count == 3) begin
        lites <= 4'b0001;                // if the count reaches 3, cycle back so the only led1 is high and count is 0
        count <= 0;
    end

    else begin
        lites <= lites << 1;             // if count < 3, keep bit-shifting the lites reg and increment the count
        count <= count + 1;
    end

    LEDS[3:0] <= lites[3:0];             // assign the lites value to the led output

end

end

```

end

endmodule

Figure 4: jackpot.xdc

```
ecen449 > lab_1 > lab_1.srcs > constrs_1 > new > ≡ jackpot.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]} ]

##Clock
set_property PACKAGE_PIN K17 [ get_ports CLOCK ]
set_property IOSTANDARD LVCMOS33 [ get_ports CLOCK ]

##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]} ]
```

