ECEN 749: Microprocessor Systems Design

Lab 1: Using Vivado

Manav Gurumoorthy

830000011

Section 603

Table of Contents

Introduction

The objective of the lab is to familiarize oneself with Vivado and the steps involved in developing Verilog code targeted at deployment on an FPGA platform. In this lab the Zybo Z7-10 development platform was used. The Zybo board has a Xilinx AP SoC architecture.

The lab was divided into three tasks. The first was to make on board LEDs light up based on inputs given through the onboard switches. The next task was to implement a 4 bit up and down counter using the LEDs to display the current state of the counter. This task requires downscaling the on-board clock from 125 MHz to a lower frequency so that the switching of the LEDs is distinguishable. Three push buttons were also interfaced; two were used to select the direction of the counter and the third was used as a reset. The last task was to design a jackpot game. The functioning of the game is described in detail later.

Procedure

Initial Configuration

The following steps describe how to setup Vivado for use in the lab exercises. The following steps are to be executed in a command terminal on the university's UNIX server (Hera, Olympus, Zeus etc):

i. Sourcing Vivado:
   >source /opt/coe/Xilinx/Vivado/2015.2/setting64.sh
ii. Running Vivado:
   >vivado
iii. Once in Vivado, create a project and add the required with Verilog as the Target language and Simulator language as Mixed.
iv. The next step is to pick the right hardware for our implementation. In this lab we use the Zybo Z7-10 board.
v. Lastly, we use the Define Module window to set the ports for the Verilog design.

The steps mentioned above are common to all the tasks in the lab.

Blinking an LED

This task requires that the onboard LEDs turn on according to inputs on the switches. The basic idea is to set the state of the LEDs as equal to the state of the switches. The module is as follows:

Verilog code for blinking LEDs

```
`timescale 1ns / 1ps

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

    assign LEDS[3:0] = SWITCHES[3:0];
endmodule
```

After the module is written and checked for syntax errors (**File -> Save All Files**) we need to create another file called a Xilinx Design Constraints (XDC) file. This file describes where the signals and ports defined in the Verilog file must be mapped to on the FPGA board. The file is shown below.

XDC file for blinking LEDs

```
## 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]}]
## 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]}]
```

After the XDC file is created and saved we add it to the list of constraints in the sources window in Vivado. This is done by right clicking the **Constraints Folder** and selecting **Add Files,** in the window chose **Add or create constraints** and add the XDC file that was created.

Now that both the XDC and Verilog file are created, proceed to synthesize the code and generate the bitstream that is to be loaded on to the FPGA board. In the Flow Navigator we select Program and Debug and select Generate Bitstream. This launches the Synthesis and implementation. Upon successful completion, open the hardware manager to load the code onto the FPGA. Select **Open Target** and select **Open New Hardware Target**, click next and select the **'Local Server'** option. Select 'xilinx_tcf' in the Hardware targets and 'xc7z010_1' in Hardware Devices. Then click on 'Program Device' and select the FPGA that is now listed.

At this point the Verilog code is successfully implemented on the hardware FPGA and we see the LEDs light up based on the input.

Up Down Counter:

A counter as the name suggests is used to "count". In this implementation pulses from the onboard clock to are used trigger a count. The LEDs present on board are used as a display to visually represent the state of the counter. The counter is 4 bits wide and hence counts from 0-15 before being reset to 0.

The frequency of the clock is 125Mhz, having the LEDs refresh at this rate makes it impossible to discern the change in states. Hence a simple clock divider is used that reduces the frequency of the clock from 125Mhz to approximately 1Hz which is much easier to observe.

The requirement was to create a counter that counts in both ascending and descending order. In order to set the direction two push buttons are used. When neither of these buttons are pressed the counter waits at its current value and glows the corresponding LED(s). Lastly, another switch was used as a RESET trigger, which can be used to reset the counter to 0.

The Verilog code for the same follows.

Verilog code for Up-Down Counter

```
`timescale 1ns / 1ps
module jackpot(
    output [3:0] LEDS,
    input [3:0] SWITCH,
    input CLK,
    input RST
    );

 reg [3:0] LEDS = 4'b000;
 reg divC = 0;
 reg [22:0] count = 23'b0;
 reg [4:0] state = 5'b0000;
 reg swRST = 0;

always @(posedge CLK) begin
    count <= count+1;
    if (count == 23'hDD00000) divC <= !divC;
 end

 always @ (posedge divC) begin
    if (LEDS [3:0] == 4'b0000) LEDS <= 4'b0001;
    else if(LEDS[3:0]== 4'b1000) LEDS <=4'b0000;
    else if(LEDS[3:0] == 4'b1111) LEDS <=4'b1111;
    else LEDS [3:0] <= LEDS << 1;
    end
always @ (posedge divC) begin
       if(LEDS[3] && ~SWITCH[3]) swRST <=1;
       else if(LEDS[2] && ~SWITCH[2]) swRST <=1;
       else if(LEDS[1] && ~SWITCH[1]) swRST <=1;
       else if(LEDS[0] && ~SWITCH[0]) swRST <=1;
       end
always @ (posedge swRST) LEDS [3:0] = 4'b1111;
always @ (posedge RST) LEDS [3:0] = 4'b0000;
```

```
endmodule
```

Just like the previous task, an XDC file was created for this code that showed clearly the pin mappings from the code to the physical board. Below is the XDC file for the 4-bit up-down counter. `## Switches`

XDC file for Up-Down Counter

```
set_property PACKAGE_PIN G15 [get_ports {SWITCH[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {SWITCH[0]}]
set_property PACKAGE_PIN P15 [get_ports {SWITCH[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {SWITCH[1]}]
set_property PACKAGE_PIN W13 [get_ports {SWITCH[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {SWITCH[2]}]
set_property PACKAGE_PIN T16 [get_ports {SWITCH[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {SWITCH[3]}]
set_property PACKAGE_PIN K18 [get_ports {RST}]
set_property IOSTANDARD LVCMOS33 [get_ports {RST}]
##set_property PACKAGE_PIN T16 [get_ports {SWITCHES[3]}]
##set_property IOSTANDARD LVCMOS33 [get_ports {SWITCHES[3]}]
## 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 K17 [get_ports {CLK}]
set_property IOSTANDARD LVCMOS33 [get_ports {CLK}]
```

The other setup and execution steps for this is identical to the previous task.

Jackpot game:

This task requires the design of a simple jackpot game. The rules of the game are simple. The LEDs glow sequentially in a One-Hot fashion. Each LED is tied to a corresponding switch. The LEDs are made to switch at a high rate and the objective is to trigger one of the on-board DIP switches at the same instant of time that the corresponding LED is on. If this is done the player is said to have one a jackpot and all the LEDs flash.

The bit of this task that was a little complicated was to be able to accurately detect the case when the state of the LEDs matched the state of the Switches. This required being able to detect the edge of the instant when the Switch was triggered.

Verilog code for Jackpot game

```
`timescale 1ns / 1ps
////////////////////////////////////////////////////////////////////////
//////////
// Company:
// Engineer:
//
// Create Date: 09/10/2019 03:52:03 PM
// Design Name:
// Module Name: jackpot
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
```

```
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
////////////////////////////////////////////////////////////////////////
///////////
module jackpot(
    output [3:0] LEDS,
    input [3:0] SWITCH,
    input CLK,
    input RST
    );

 reg [3:0] LEDS;
 reg divC = 0;
 reg [22:0] count = 23'b0;
 reg [3:0] state = 4'b0001;
 reg [3:0] swTrg  = 4'b0000;

always @(posedge CLK) begin
    count <= count+1;
    if (count == 23'hDD00000) divC <= !divC;
 end

 always @ (posedge divC, posedge RST) begin
       if (RST)  begin
        LEDS[3:0] <= 4'b0000;
        state [3:0] <= 4'b0001;
       end
       else if (LEDS [3:0] == 4'b1111) begin
                LEDS[3:0] <= 4'b1111;
                state[3:0] <= 4'b1100;
            end
      else begin
      if(state == 4'b0001) begin
            LEDS [3:0] <= state [3:0];
                if (SWITCH [3:0] == state [3:0] && swTrg [3:0]
==4'b0000)
                    LEDS [3:0] <= 4'b1111;
                else state <= 4'b0010;
        end
     else if (state == 4'b0010)begin
       LEDS [3:0] <= state [3:0];
       if (SWITCH [3:0] == state [3:0] && swTrg [3:0] == 4'b0000)
             LEDS [3:0] <= 4'b1111;
        else state <= 4'b0100;
     end
     else if (state == 4'b0100) begin
        LEDS [3:0] <= state [3:0];
        if (SWITCH [3:0] == state [3:0] && swTrg [3:0] ==4'b0000)
        LEDS [3:0] <= 4'b1111;
        else state <= 4'b1000;
     end

     else begin
        LEDS [3:0] <= state [3:0];
        if (SWITCH [3:0] == state [3:0] && swTrg [3:0] ==4'b0000)
        LEDS [3:0] <= 4'b1111;
        else state <= 4'b0001;
     end
     swTrg <= SWITCH;
  end
 end
endmodule
```

XDC file for the Jackpot game

```
## Switches
set_property PACKAGE_PIN G15 [get_ports {SWITCH[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {SWITCH[0]}]
set_property PACKAGE_PIN P15 [get_ports {SWITCH[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {SWITCH[1]}]
set_property PACKAGE_PIN W13 [get_ports {SWITCH[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {SWITCH[2]}]
set_property PACKAGE_PIN T16 [get_ports {SWITCH[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {SWITCH[3]}]
set_property PACKAGE_PIN K18 [get_ports {RST}]
set_property IOSTANDARD LVCMOS33 [get_ports {RST}]
##set_property PACKAGE_PIN T16 [get_ports {SWITCHES[3]}]
##set_property IOSTANDARD LVCMOS33 [get_ports {SWITCHES[3]}]
## 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 K17 [get_ports {CLK}]
set_property IOSTANDARD LVCMOS33 [get_ports {CLK}]
```

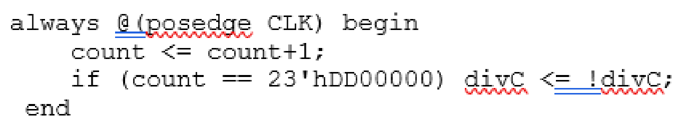
Results

Blinking LEDs

This task was straightforward and there weren't any challenges that I faced in this. I did however, learn about how the XDC file is used in vivado and what it needs to contain in order to realise Verilog code on the FPGA hardware.

4-bit Up-Down Counter

This task as explained earlier was to implement a 4-bit counter. The difficulty in the implementation arose from the fact that the onboard clock was to be used as the trigger for the counter. As the frequency of the clock is 125Mhz it was impossible to distinguish the counting action. To rectify this a clock division circuit was built that divided the clock from 125 Mhz to a much smaller value. Writing the clock division was something that was quite challenging but, in the end, it managed to be just 3 lines of code.



```
always @(posedge CLK) begin
    count <= count+1;
    if (count == 23'hDD00000) divC <= !divC;
 end
```

Figure 1 Clock Division circuit

Jackpot Game

The jackpot game proved to be quite difficult. It required combining all the skills from the two previous task and adding a little flourish to make it work. It was broken it down into three parts. First was getting the one hot encoding going. The initial thought was to use some sort of modulo operation (mod 15) along with a left shift operation. But instead the final code uses a much simpler if-else if ladder to achieve the same. The second step was to prevent a win from being triggered from a previous switch trigger. The third and final part was to hold the game in the win state and adding a reset switch.

```
       end
       else if (LEDS [3:0] == 4'b1111) begin
                LEDS[3:0] <= 4'b1111;
                state[3:0] <= 4'b1100;
            end
      else begin
      if(state == 4'b0001) begin
            LEDS [3:0] <= state [3:0];
                if (SWITCH [3:0] == state [3:0] && swTrg [3:0]
==4'b0000)
                    LEDS [3:0] <= 4'b1111;
                else state <= 4'b0010;
        end
     else if (state == 4'b0010)begin
       LEDS [3:0] <= state [3:0];
       if (SWITCH [3:0] == state [3:0] && swTrg [3:0] == 4'b0000)
             LEDS [3:0] <= 4'b1111;
        else state <= 4'b0100;
     end
     else if (state == 4'b0100) begin
        LEDS [3:0] <= state [3:0];
        if (SWITCH [3:0] == state [3:0] && swTrg [3:0] ==4'b0000)
        LEDS [3:0] <= 4'b1111;
        else state <= 4'b1000;
     end

     else begin
        LEDS [3:0] <= state [3:0];
        if (SWITCH [3:0] == state [3:0] && swTrg [3:0] ==4'b0000)
        LEDS [3:0] <= 4'b1111;
        else state <= 4'b0001;
     end
```

Each of these If-Else statements are used to transition from one state to the next and to check if the winning state has occurred.

Figure 2 Excerpt from the code describing the main logic of the jackpot game.

Conclusion

The tasks in Lab 1 helped to familiarise oneself with Vivado and the Zybo platform. It was also a good revision on some of the important concepts in Verilog like blocking and non-blocking statements through the coding exercises.

Questions

1. The push buttons are wired to the Zybo Z7-10 through the pins Y16, K19, P16 and K18. The switches are pulled down to ground.
2. The edge detection circuit is used to detect the instant at which an event has taken place. Edge detection is of two forms; Rising or positive edge detection and Falling or negative edge detection. In context to this lab, edge detection is used to trigger the counter which updates its value at every rising edge of the divided clock. In fact edge detection is also used in the generation of the divided clock. Edge detection was also used in the detection of the winning combination in the jackpot game. This would track the switch and get triggered at the rising of it.