ECEN 749

Laboratory Exercise #1

Using Vivado

Name: Pranav Anantharam

UIN: 734003886

Course: ECEN 749 Microprocessor System Design

Section: 603

Introduction:

The first laboratory exercise was designed to introduce us to the Xilinx FPGA design process using Vivado through a basic example. The goal is to create hardware using Vivado that interacts with the ZYBO Z7-10 board's LEDs based on the state of its on-board DIP switches. This hardware will be implemented using an FPGA and programmed in Vivado using Verilog. By completing this exercise, we will gain practical experience with FPGA design and be prepared to undertake further tasks, such as creating a simple counter and a jackpot game independently. This lab is important as it provides hands-on experience with FPGA design, which is essential for understanding and working with advanced digital systems.

Procedure:

1) Load the environment variables on the workstation using appropriate commands and open Vivado 2022.1.
2) Create a new project and add sources to the project.
3) Select hardware from the 'Default Part' window. Filter the parts available using the below device properties:
    a. Family: Zynq-7000
    b. Package: clg400
    c. Speed: -1
4) Select the device with part number 'xc7z010clg400-1'.
5) Using the 'Define Module' window, define the ports for the hardware module.
6) From the 'Sources' window, open the desired source file and add/edit the code to implement the program to control the LEDs using on-board DIP switches.
7) Create the 'Xilinx Design Constraints (XDC)' file containing the location of the DIP switches and LEDs on the Zybo Z7-10 board using a text editor. Save the constraints file.
8) Add the constraints file to the project.
9) Select the design source file and generate bitstream. The obtained bitstream can be downloaded onto the FPGA board.
10) Turn on the power to Zybo Z7-10 board and connect the board to the workstation over USB cable to download the bitstream onto the board.
11) Using the 'Open Hardware Manager' window, select 'Open New Hardware Target' to detect and connect to the Zybo Z7-10 board.
12) Finally, select 'Program device' to program the Zybo Z7-10 board with the bitstream generated.
13) Repeat steps 5-12 after making changes in the design source file and constraints files to implement 4-bit counter and jackpot game. (Design source changes and Constraints file changes given in Appendix Section)

Results:

A simple example project was created and developed to light up the LEDs on the Zybo Z7-10 board based on the state of the on-board DIP switches. The bitstream for the example program was generated and downloaded onto the board, and the example was tested manually. It was a simple example in which registers holding switch states were assigned continuously to registers holding LED states. This example allowed us to gain a better understanding of the Xilinx FPGA design flow.

4-Bit Counter:

Following this example, a 4-bit counter program was developed and demonstrated using the LEDs on the Zybo Z7-10 board. The count value was updated approximately every 1 second and buttons were configured to control the direction of the count. The working of the program is as follows:

a) First, we divide the input clock (125MHz) using the clock divider module to obtain a modified input clock of 1 Hz frequency.
b) The clock divider module is a simple module which involves a counter variable which outputs a 1 when the counter variable is equal to 125 million, otherwise outputs a 0. Hence, it divides the input clock of 125 MHz.
c) In the counter module, at every positive edge of the modified clock, check the state of the Buttons to determine the direction of the counter.
d) Increment or decrement the counter value accordingly and assign the value of the counter to the LED register to update the state of the on-board LEDs.

The constraints file needed to be updated to map the button inputs in the counter module to the pins on the FPGA. This required us to refer to the Zybo reference manual to find the appropriate pins for use.

The outputs of the program were demonstrated using the on-board LEDs of the Zybo Z7-10 board.

(Screenshots of design source file and constraints files given in Appendix section)

Jackpot Game:

Next, a jackpot game was designed in which the frequency of the LED transitions were increased to a higher rate, and users attempt to turn on the on-board DIP switch corresponding to the glowing LED, which would result in a Jackpot win. All the LEDs start glowing in response to a jackpot win.

Changes were made to the clock divider module to increase the LED transition rate; the divided clock output was set to 1 when the counter variable reached 40 million, and the output was set to 0 otherwise.

The working of the jackpot program is as follows:

a) Initialize 4-bit registers to track the current state of the LEDs and previous state of the on-board DIP switches.
b) At every positive edge of the divided clock input, we first check the game_finish flag to determine whether is already over.
c) If the game_finish flag is false, then we proceed to check if the previous state of the switch is 0 and the current state of the switch is 1 (to detect a OFF-> ON switch transition) and if the corresponding LED is also glowing for any of the 4 LEDs.
d) If the above condition is true, set the current LED state register to 4'b1111 and set the game_finish flag to 1 indicating a Jackpot win.
e) If the game_finish flag is not set, then update the current LED state register in a cyclic manner.
f) Update the previous switch state register with the current switch state register.
g) Continuously assign the led_state register to the LEDS register to update the state of the on-board LEDs.

The constraints file needed to be updated to map the switch and button inputs in the jackpot module to the pins on the FPGA. This required us to refer to the Zybo reference manual to find the appropriate pins for use.

(Screenshots of design source file and constraints files given in Appendix section)

Conclusion:

The purpose of the laboratory exercise is to provide practical experience in FPGA design using Vivado software, starting with a foundational example of hardware interaction with the ZYBO Z7-10 board's LEDs via DIP switches. Building upon this, the development of a 4-bit counter and a jackpot game deepens our understanding of FPGA design principles and their applications. Through these examples, we explore concepts such as clock management, signal processing, gaining hands-on experience in designing synchronous digital circuits and user interaction. These exercises reinforce our understanding of FPGA design and also demonstrate its versatility in solving real-world problems, laying a solid foundation for tackling more complex FPGA-based projects in the future.

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:

The Zybo Z7-10 board has four push-buttons. There are also two push-buttons connected directly to the processing system (PS) via Multiplexed I/O pins. The push-buttons are connected to the Zynq via series resistors to prevent damage from inadvertent short circuits. The push-buttons normally generate a low output when they are at rest, and a high output only when they are pressed. All the push-button signals are pulled down to GND. The pin mapping for push-buttons on the FPGA are given below:

1. BTN0 – K18
2. BTN1 – P16
3. BTN2 – K19
4. BTN3 – Y16
5. BTN4 – MIO50
6. BTN5 – MIO51

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

Answer:

An edge detection circuit is designed to detect changes in the signal level, specifically transitions from low to high (rising edge) or high to low (falling edge). The purpose of such a circuit is to trigger actions or events based on these signal changes. In the context of this lab, an edge detection circuit could have been used to detect transitions in the state of the on-board DIP switches or push-buttons on the ZYBO Z7-10 board.

In the lab, the push-buttons and switches are used as inputs to control various functionalities, such as changing the direction of the counter and triggering actions in the jackpot game. By incorporating an edge detection circuit, the FPGA can be programmed to respond specifically to changes in the state of the DIP switch and push-button rather than continuously monitor their values. This would enable more efficient and accurate detection of user inputs, ensuring that actions are triggered precisely when the DIP switch/push-button is enabled.

For example, in the counter example, an edge detection circuit can be utilized to detect when the push-buttons change from a released to a pressed state, signaling the FPGA to change the direction of the counter accordingly

(increment or decrement). Similarly, in the jackpot game, edge detection can be employed to detect when the player turns on the DIP switch corresponding to the glowing LED, triggering the jackpot win event.

Appendix:

(Screenshots of design source and constraints files)

clock_divider.v: (Clock Divider Module)

```
module clock_divider(output reg div_clock, input clock, input reset);

    reg [31:0] counter;

    parameter freq_count = 125000000;

    always @(posedge clock) begin

        if( counter == freq_count ) begin
            div_clock <= 1;
            counter <= 0;
        end
        else begin
            div_clock <= 0;
            counter <= counter + 1;
        end
    end
endmodule
```

counter.v: (4-bit Counter Module)

```
module counter(output [3:0] LEDS, input [1:0] BUTTONS, input clock, input reset);

    reg [3:0] counter_val;
    reg dir;

    wire div_clock;

    clock_divider(div_clock, clock, reset);

    always @(posedge div_clock) begin
        if(reset) begin
            counter_val <= 0;
        end
        else begin
            if( BUTTONS[0] == 1 ) begin
                dir = 1;
            end
            else if( BUTTONS[1] == 1 )begin
                dir = 0;
            end

            if( dir == 1 ) begin
                counter_val <= counter_val + 1;
            end
            else if( dir == 0 ) begin
                counter_val <= counter_val - 1;
            end

        end
    end

    assign LEDS[3:0] = counter_val[3:0];

endmodule
```

counter.xdc: (4-bit Counter Constraints File)

```
##Buttons
set_property PACKAGE_PIN Y16 [get_ports {BUTTONS[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {BUTTONS[0]}]

set_property PACKAGE_PIN K19 [get_ports {BUTTONS[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {BUTTONS[1]}]

##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]}]

##Reset
set_property PACKAGE_PIN P16 [get_ports {reset}]
set_property IOSTANDARD LVCMOS33 [get_ports {reset}]

##Clock
set_property PACKAGE_PIN K17 [get_ports {clock}]
set_property IOSTANDARD LVCMOS33 [get_ports {clock}]
```

jackpot.v: (Jackpot Game Module)

```
module jackpot(output [3:0] LEDS, input [3:0] SWITCHES, input clock, input reset);

    wire div_clock;

    reg game_finish = 0;
    reg [3:0] led_state = 4'b0001;
    reg [3:0] prev_switch_state = 4'b0000;

    clock_divider(div_clock, clock, reset);

    always @(posedge div_clock) begin

        if( reset ) begin
            led_state = 4'b0001;
            prev_switch_state = 4'b0000;
            game_finish = 0;
        end
        else begin
            if( game_finish != 1 ) begin
                if( (prev_switch_state[0] == 0 && SWITCHES[0] == 1 && led_state == 4'b0001) ||
                        (prev_switch_state[1] == 0 && SWITCHES[1] == 1 && led_state == 4'b0010) ||
                        (prev_switch_state[2] == 0 && SWITCHES[2] == 1 && led_state == 4'b0100) ||
                        (prev_switch_state[3] == 0 && SWITCHES[3] == 1 && led_state == 4'b1000)
                    ) begin
                        led_state <= 4'b1111;
                        game_finish = 1;
                    end

                    if( game_finish != 1 ) begin
                        if( led_state == 4'b1000 ) begin
                            led_state <= 4'b0001;
                        end
                        else begin
                            led_state <= (led_state << 1);
                        end
                    end
                end
            end

        // Update previous switch state
        prev_switch_state[3:0] = SWITCHES[3:0];
    end

    assign LEDS[3:0] = led_state[3:0];

endmodule
```

clock_divider.v: (Modified Clock Divider Module for faster LED transitions)

```
module clock_divider(output reg div_clock, input clock, input reset);

    reg [31:0] counter;

    parameter freq_count = 40000000;

    always @(posedge clock) begin

        if( counter == freq_count ) begin
            div_clock <= 1;
            counter <= 0;
        end
        else begin
            div_clock <= 0;
            counter <= counter + 1;
        end
    end
endmodule
```

jackpot.xdc: (Jackpot Game Constraints File)

```
##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]}]


##Reset
set_property PACKAGE_PIN Y16 [get_ports {reset}]
set_property IOSTANDARD LVCMOS33 [get_ports {reset}]


##Clock
set_property PACKAGE_PIN K17 [get_ports {clock}]
set_property IOSTANDARD LVCMOS33 [get_ports {clock}]
```