ECE 437 Post Lab 2

Ryan Libiano

September 2023

Post-Lab Questions

1. **In checkpoint 1 of the lab, you printed simulation results from "Behavior Simulation" and "Post-Implementation Timing Simulation". List three differences that you observe between the results from the "Behavior Simulation" and "Post-Implementation Timing Simulation"? What are the reasons for these differences between the two simulations? Use the zoom button to help you find these differences in the simulation windows. (15 pts).**
   One clear difference is that the Behavioral Simulation's timed events happen on integer multiples of the clock period, but the Post-Implementation Timing Simulation's events happen at timing intervals that aren't integer multiples of the clock period. Moreover, the Behavioral Simulation doesn't show interconnect delay or propogation delay, it only takes into account gate delay. This can be seen by the Post-Implementation timing taking a bit longer for events to happen as compared to the Behavorial Simulation. A final distinct difference between the two is that the Behavioral simulation doesn't show phenomena like meta-stability. You can see this phenomona in the Post-Implementation Timing Simulation, some states go through multiple changes before settling to a final values.

2. **In checkpoint 1 of the lab, look at the waveform results at time 220ns from the start of the simulation. This is the time when the buttons change from state 4b'1111 to 4b'1011. Use the zoom in button to closely examine the results. How long does it take for the LEDs to change its output state measured from the time the buttons change from state 4b'1111 to 4b'1011? Do you see any erroneous states that the LEDs transition through? Use the zoom button and take a snapshot of these states. Include them in the report. Why do you have these erroneous states? (10 points)**
   It takes 25 ns from when the button is pressed for the LED's to finally change to their next value. There are erronous states, which can be seen in Figure 5. These are characterized by the different LED states stuttering through incorrect states before coming across the final state. This could

be due to physical behavior in the FPGA, since Post-Implementation takes into account the physical fitting and routing of elements.

3. **Compare the timing results from question 2 to the ones from the "Behavioral simulation". Report the time it takes for the LEDs to change its output measured from the onset the buttons change from state 4b'1111 to 4b'1011. Why is this time different compared to the one from the "Post-Implementation Timing Simulation"? (10 points)**
   It takes 15 ns from when the button is pressed for the LED's to finally change to their next value. This difference between the two simulations is probably due to the different considerations taken in each simulation. The behavioral simulation seems to only take into account gate delay/logical delay, however, in the post implementation timing simulation, propagation delay and fanout is also taken into affect. This is most likely the cause between the differences in the times.

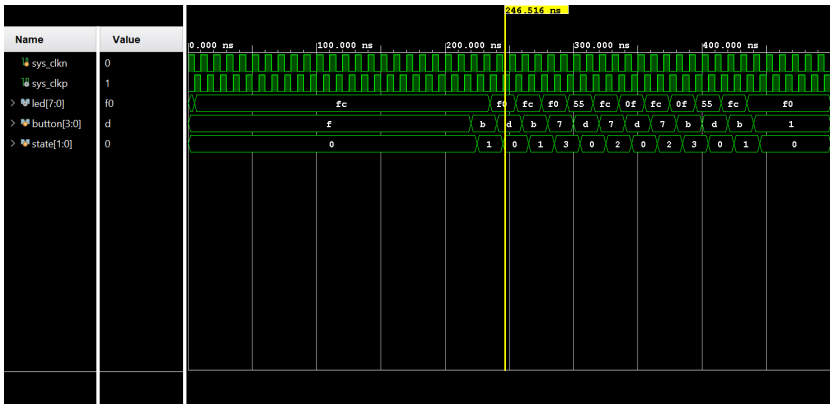

Figure 1: Milestone 1 Behavioral Simulation

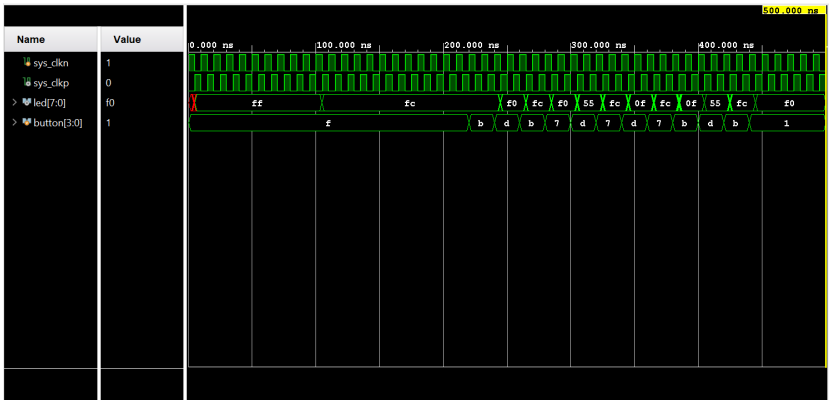

Figure 2: Milestone 1 Post-Implementation Timing Simulation

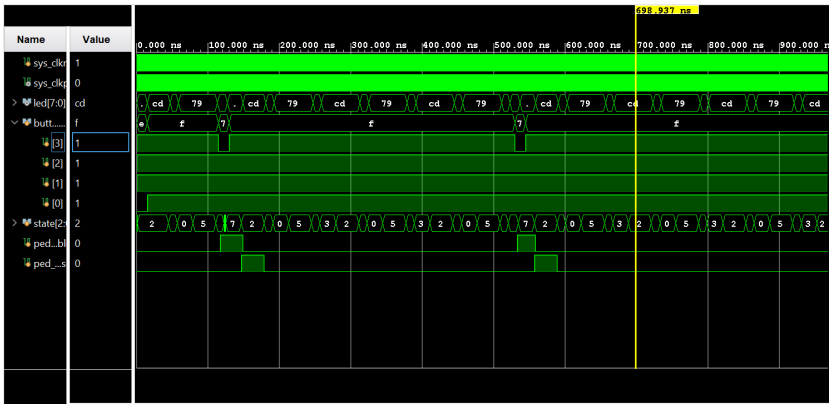

Figure 3: Milestone 2 Behavioral Simulation

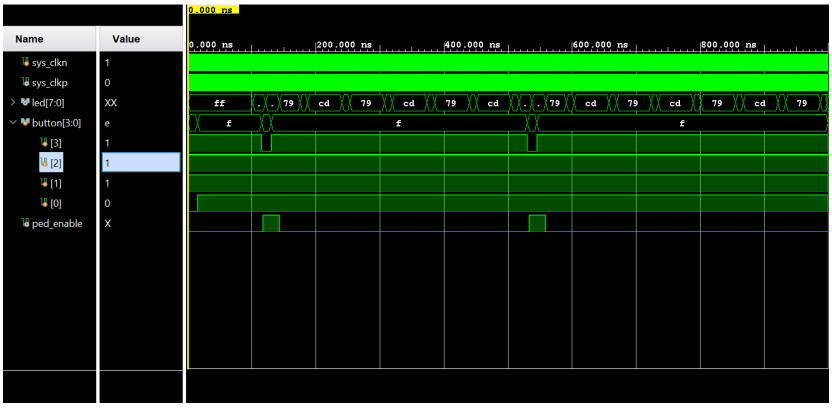

Figure 4: Milestone 2 Post Implementation Timing Simulation

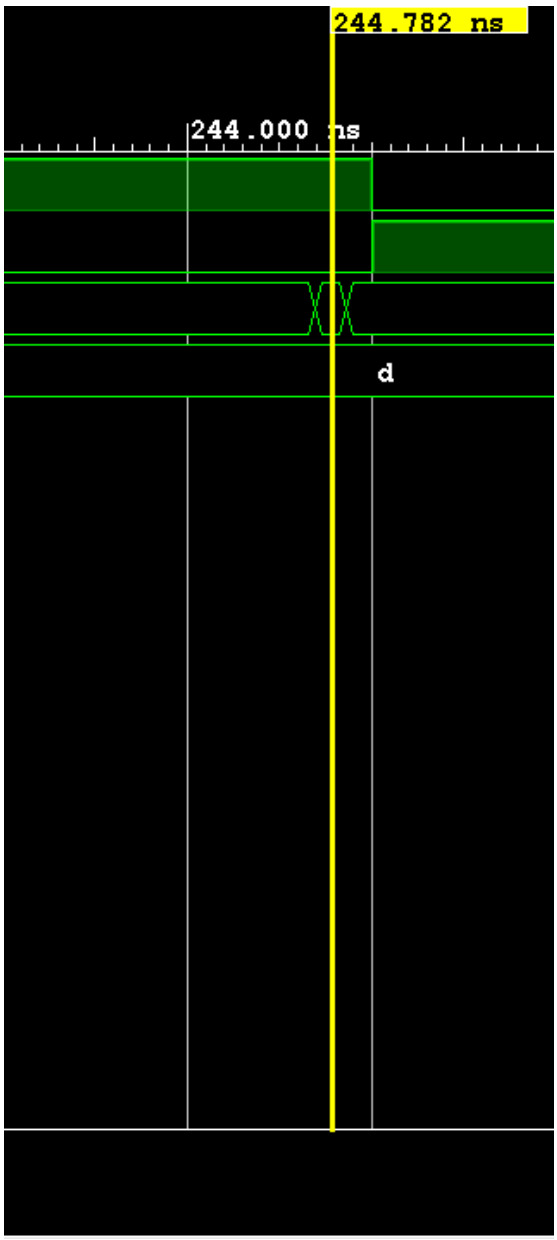

Figure 5: Erroneous Behavior in Post Implementation Timing Simulation

Include a printout of both codes with the final report.

Verilog Code Milestone 1

```
`timescale 1ns / 1ps
    module lab3_example(
    input [3:0] button,
    output [7:0] led,
    input sys_clkn,
    input sys_clkp
    );

    reg [1:0] state = 0;
    reg [7:0] led_register = 0;
    reg [3:0] button_reg;

    wire clk;
    IBUFGDS osc_clk(
        .O(clk),
        .I(sys_clkp),
        .IB(sys_clkn)
    );

    assign led = ~led_register; //map led wire to led_register
    localparam STATE_INIT       = 2'd0;
    localparam STATE_ALPHA      = 2'd1;
    localparam STATE_BRAVO      = 2'd2;
    localparam STATE_CHARLIE    = 2'd3;

    always @(posedge clk)
    begin
        button_reg = ~button;
        if (button_reg [3:0] == 4'b1110) state <= STATE_INIT;
        else
        begin
            case (state)
                STATE_INIT : begin
                    if (button_reg == (4'b0100)) state <= STATE_ALPHA;
                    else if (button_reg == 4'b1000) state <= STATE_BRAVO;
                    else led_register <= 8'b00000011;
                end

                STATE_ALPHA : begin
                    if (button_reg == 4'b1000) state <= STATE_CHARLIE;
                    else if (button_reg == 4'b0010) state <= STATE_INIT;
                    else led_register <= 8'b00001111;
                end
```

```
                STATE_BRAVO : begin
                    if (button_reg == 4'b0100) state <= STATE_CHARLIE;
                    else if (button_reg == 4'b0010) state <= STATE_INIT;
                    else led_register <= 8'b11110000;
                end

                STATE_CHARLIE : begin
                    if (button_reg == 4'b0010) state <= STATE_INIT;
                    else led_register <= 8'b10101010;
                end

                default: state <= STATE_INIT;

            endcase
        end
    end
endmodule
```

Verilog Code Milestone 2

```
    `timescale 1ns / 1ps
module lab3_example(input   wire    [4:0] okUH,
        output  wire    [2:0] okHU,
        inout   wire    [31:0] okUHU,
        inout   wire    okAA,
    input [3:0] button,
    output [7:0] led,
    input sys_clkn,
    input sys_clkp
    );
    //Front Panel Stuff
    wire okClk;             //These are FrontPanel wires needed to IO communication
    wire [112:0]    okHE;   //These are FrontPanel wires needed to IO communication
    wire [64:0]     okEH;   //These are FrontPanel wires needed to IO communication
    wire [31:0] ped_but;

    //This is the OK host that allows data to be sent or recived
    okHost hostIF (
           .okUH(okUH),
           .okHU(okHU),
           .okUHU(okUHU),
           .okClk(okClk),
           .okAA(okAA),
           .okHE(okHE),
           .okEH(okEH)
```

```
   );

//parameters
  localparam  endPt_count = 1;
  wire [endPt_count*65-1:0] okEHx;
  okWireOR # (.N(endPt_count)) wireOR (okEH, okEHx);

 okWireIn wire10 (   .okHE(okHE),
                     .ep_addr(8'h00),
                     .ep_dataout(ped_enable));

reg [2:0] state = 0;
reg [7:0] led_register = 0;
reg [3:0] button_reg;
reg [31:0] count = 0;
reg [31:0] ped_count = 0;
reg ped_enable;
reg ped_reset = 0;
wire clk;
IBUFGDS osc_clk(
    .O(clk),
    .I(sys_clkp),
    .IB(sys_clkn)
);
always @ ( posedge clk) begin
    if (ped_reset) begin
        ped_enable <= 1'b0;
    end
     else if (ped_but == 32'hFFFF) begin
        ped_enable <= 1'b1;
    end
end
assign led = ~led_register; //map led wire to led_register
localparam R1 = 3'd0;
localparam Y1 = 3'd1;
localparam G1 = 3'd2;
localparam R2 = 3'd3;
localparam Y2 = 3'd4;
localparam G2 = 3'd5;
localparam C_N_S = 3'd6;
localparam C_E_W = 3'd7;

always @(posedge clk)
begin
    button_reg = ~button;
    if (button_reg == 4'b0001) begin
```

```
    state <= G1;
    count <= 32'd0;
    ped_reset <= 0;
end
else
begin
    case (state)
        R1 : begin
            if (count == 32'd10) begin
                count <= 32'd0;
                state <=  G2;
            end

            else if (ped_enable == 1'b1) begin
                count <= 32'd0;
                ped_count <= 32'd0;
                state <= C_N_S;

            end

            else begin
                led_register <= 8'b10000110;
                count <= count + 1;
            end
        end

        Y1 : begin
            if (count == 32'd5) begin
                count <= 32'd0;
                state <=  R1;
            end

            else begin
                led_register <= 8'b01010010;
                count <= count + 1;
            end
        end

        G1 : begin
            if (count == 32'd15) begin
                count <= 32'd0;
                state <=  Y1;
                ped_reset <= 0;
            end

            else begin
```

```
        led_register <= 8'b00110010;
        count <= count + 1;
    end
end

R2 : begin
    if (count == 32'd10) begin
        count <= 32'd0;
        state <=  G1;
    end

    else if (ped_enable == 1'b1) begin
        count <= 32'd0;
        ped_count <= 32'd0;
        state <= C_E_W;
    end

    else begin
        led_register <= 8'b00110010;
        count <= count + 1;
    end
end

Y2 : begin
    if (count == 32'd5) begin
        count <= 32'd0;
        state <=  R2;
    end

    else begin
        led_register <= 8'b10001010;
        count <= count + 1;
    end
end

G2 : begin
    if (count == 32'd15) begin
        count <= 32'd0;
        state <=  Y2;
        ped_reset <= 0;
    end

    else begin
        led_register <= 8'b10000110;
        count <= count + 1;
    end
```

```
                end

                C_N_S : begin
                    if (ped_count == 32'd10) begin
                        ped_reset <= 1;
                        state <= G2;
                        ped_count <= 32'd0;
                    end
                    else begin
                        led_register <= 8'b10010001;
                        ped_count <= ped_count + 1;
                    end
                end

                C_E_W : begin
                    if (ped_count == 32'd10) begin
                        ped_reset <= 1;
                        state <= G1;
                        ped_count <= 32'd0;
                    end
                    else begin
                        led_register <= 8'b10010001;
                        ped_count <= ped_count + 1;
                    end
                end

                default: state <= G1;

            endcase
        end
    end
endmodule
```

Python Code

```
    # -*- coding: utf-8 -*-

#%%
# import various libraries necessary to run your Python code
import time   # time related library
import sys,os    # system related library
ok_sdk_loc = "C:\\Program Files\\Opal Kelly\\FrontPanelUSB\\API\\Python\\x64"
ok_dll_loc = "C:\\Program Files\\Opal Kelly\\FrontPanelUSB\\API\\lib\\x64"
```

```
sys.path.append(ok_sdk_loc)   # add the path of the OK library
os.add_dll_directory(ok_dll_loc)

import ok     # OpalKelly library

# Define FrontPanel device variable, open USB communication and
# load the bit file in the FPGA
dev = ok.okCFrontPanel()  # define a device for FrontPanel communication
SerialStatus=dev.OpenBySerial("")      # open USB communication with the OK board
ConfigStatus=dev.ConfigureFPGA("lab2_example.bit"); # Configure the FPGA with this bit file

# Check if FrontPanel is initialized correctly and if the bit file is loaded.
# Otherwise terminate the program
print("----------------------------------------------------------")
if SerialStatus == 0:
    print ("FrontPanel host interface was successfully initialized.")
else:
    print ("FrontPanel host interface not detected. The error code number is:" + str(int(Ser
    print("Exiting the program.")
    sys.exit ()

if ConfigStatus == 0:
    print ("Your bit file is successfully loaded in the FPGA.")
else:
    print ("Your bit file did not load. The error code number is:" + str(int(ConfigStatus)))
    print ("Exiting the progam.")
    sys.exit ()
print("----------------------------------------------------------")
print("----------------------------------------------------------")

#%%
# Define the two variables that will send data to the FPGA
# We will use WireIn instructions to send data to the FPGA
on = 4294967295
off = 0
dev.SetWireInValue(0x00, off) #Input data
dev.UpdateWireIns()  # Update the WireIns

#%%
# We will read data from the FPGA in two different variables
# Since we are using a slow clock on the FPGA to compute the results
# we need to wait for the result to be computed

# First receive data from the FPGA by using UpdateWireOuts
while(1) :
    time.sleep(.10)
```

```
    dev.SetWireInValue(0x00, off)
    dev.UpdateWireIns()
    button = 0;
    button = int(input('Input 1 to press walk'))
    print(result_difference)
    if (button == 1):
        time.sleep(0.05)
        dev.SetWireInValue(0x00, on)
        dev.UpdateWireIns()  # Update the WireIns
    else :
        time.sleep(0.05)
        dev.SetWireInValue(0x00, off)
        dev.UpdateWireIns()

dev.Close

#%%
```