ECE 437 Post\_Lab 3

Q1. a. Theledoutput (andthere for led\_register) is floating (undefined) initially and requires the first falling edge to become valid in the post implementation timing simulation while in behavior simulation it is valid immediately, this showcases signal setup time.

b. All bits in the led output does not update at the same time, they change at different time due to different signal delays.

c. Signals also does not update perfectly aligned with clock edges in post implementation timing sim, while signals edges aligned perfectly with clock edges in behavioral sim

Q2. It takes roughly 25ns for LEDs to change its output, there is some erroneous state between state transition, this is due to each bit of the leds output going through different amount of levels of combinational logic to be updated thus having different delays, updating at different times

Q3. It only takes 15 ns for behavioral sim to update, this is possibally due to that change the button is pressed right at the edge of clock, the FPGA might not be able to capture this change in button resulting the change being captured 1 clock cycle later (10ns) than behavioral sim where rising clock while perfectly capture everything.

Milestone1 did not require code changes thus it is unlisted.

Milestone2 Code:

FSM.V:

`timescale 1ns / 1ps

module FSM

#( parameter one\_sec = 100000000,

half\_sec = 50000000)

(

input wire clk,

input wire[31:0] pedestrian,

output wire[7:0] led

);

localparam STATE\_Y1 = 3'd1;

localparam STATE\_G1 =3'd2;

localparam STATE\_Y2 = 3'd3;

localparam STATE\_G2 =3'd4;

localparam STATE\_PE1 = 3'd5;

localparam STATE\_PE2 = 3'd6;

reg [3:0]

reg [7:0]

reg [27:0]

reg

reg

event

state = 0;

led\_register = 0; //R1, Y1, G1, R2, Y2, G2, R3, G3.

counter = 0;

cross = 0;

pedestrain\_reg = 1; // initialized to 1 to prevent USB set up triggering pedestrian

assign led = ~led\_register; //map led wire to led\_register

always @(posedge clk)

begin

pedestrain\_reg <= pedestrian[0];

if ((pedestrain\_reg == 1'b0) && (pedestrian[0] == 1'b1))

begin

cross <= 1'b1;

end

case (state)

STATE\_G1 : begin

led\_register <= 8'b00110010;

if (counter >= one\_sec)

begin

state <= STATE\_Y1;

counter <= 0;

end

else

begin

counter <= counter + 1;

end

end

STATE\_Y1 : begin

led\_register <= 8'b01010010;

if (counter >= half\_sec && cross)

begin

state <= STATE\_PE1;

counter <= 0;

cross <= 0;

end

else if (counter >= half\_sec)

begin

state <= STATE\_G2;

counter <= 0;

end

else

begin

counter <= counter + 1;

end

end

STATE\_G2 : begin

led\_register <= 8'b10000110;

if (counter >= one\_sec)

begin

state <= STATE\_Y2;

counter <= 0;

end

else

begin

counter <= counter + 1;

end

end

STATE\_Y2 : begin

led\_register <= 8'b10001010;

if (counter >= half\_sec && cross)

begin

state <= STATE\_PE2;

counter <= 0;

cross <= 0;

end

else if (counter >= half\_sec)

begin

state <= STATE\_G1;

counter <= 0;

end

else

begin

counter <= counter + 1;

end

end

STATE\_PE1 : begin

led\_register <= 8'b10010001;

if (counter >= one\_sec)

begin

state <= STATE\_G2;

counter <= 0;

end

else

begin

counter <= counter + 1;

end

end

STATE\_PE2 : begin

led\_register <= 8'b10010001;

if (counter >= one\_sec)

begin

state <= STATE\_G1;

counter <= 0;

end

else

begin

counter <= counter + 1;

end

end

default: state <= STATE\_G1;

endcase

end

endmodule

lab3\_example.v:

`timescale 1ns / 1ps

module lab3\_example(

input wire [4:0] okUH,

output wire [2:0] okHU,

inout wire [31:0] okUHU,

inout wire okAA,

output [7:0] led,

input sys\_clkn,

input sys\_clkp

);

wire [31:0] pedestrian;

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 clk;

IBUFGDS osc\_clk(

.O(clk),

.I(sys\_clkp),

.IB(sys\_clkn)

);

okHost hostIF (

.okUH(okUH),

.okHU(okHU),

.okUHU(okUHU),

.okClk(okClk),

.okAA(okAA),

.okHE(okHE),

.okEH(okEH)

);

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(pedestrian));

FSMFSM(.clk(clk),

.pedestrian(pedestrian),

.led(led));

endmodule

lab3\_TestBench.v:

`timescale 1ns / 1ps

module lab3\_TestBench();

//Declare wires and registers that will interface with the module under test

//Registers are initilized to known states. Wires cannot be initilized.

reg clk = 1;

wire [7:0] led;

reg [31:0]button;

//Invoke the module that we like to test

FSM#(.one\_sec(100),.half\_sec(50)) ModuleUnderTest (.pedestrian(button),.led(led),.clk(clk));

// Generate a clock signal. The clock will change its state every 5ns.

//Remember that the test module takes sys\_clkp and sys\_clkn as input clock signals.

//From these two signals a clock signal, clk, is derived.

//The LVDS clock signal, sys\_clkn, is always in the opposite state than sys\_clkp.

always begin

#5 clk = ~clk;

end

initial begin

#0 button <=0;

#4000 button <= 1;

#500 button <= 0;

#2000

button <= 1;

end

endmodule FSM.V:

`timescale 1ns / 1ps

module FSM

#( parameter one\_sec = 100000000,

half\_sec = 50000000)

(

input wire clk,

input wire[31:0] pedestrian,

output wire[7:0] led

);

localparam STATE\_Y1 = 3'd1;

localparam STATE\_G1 =3'd2;

localparam STATE\_Y2 = 3'd3;

localparam STATE\_G2 =3'd4;

localparam STATE\_PE1 = 3'd5;

localparam STATE\_PE2 = 3'd6;

reg [3:0]

reg [7:0]

reg [27:0]

reg

reg

event

state = 0;

led\_register = 0; //R1, Y1, G1, R2, Y2, G2, R3, G3.

counter = 0;

cross = 0;

pedestrain\_reg = 1; // initialized to 1 to prevent USB set up triggering pedestrian

assign led = ~led\_register; //map led wire to led\_register

always @(posedge clk)

begin

pedestrain\_reg <= pedestrian[0];

if ((pedestrain\_reg == 1'b0) && (pedestrian[0] == 1'b1))

begin

cross <= 1'b1;

end

case (state)

STATE\_G1 : begin

led\_register <= 8'b00110010;

if (counter >= one\_sec)

begin

state <= STATE\_Y1;

counter <= 0;

end

else

begin

counter <= counter + 1;

end

end

STATE\_Y1 : begin

led\_register <= 8'b01010010;

if (counter >= half\_sec && cross)

begin

state <= STATE\_PE1;

counter <= 0;

cross <= 0;

end

else if (counter >= half\_sec)

begin

state <= STATE\_G2;

counter <= 0;

end

else

begin

counter <= counter + 1;

end

end

STATE\_G2 : begin

led\_register <= 8'b10000110;

if (counter >= one\_sec)

begin

state <= STATE\_Y2;

counter <= 0;

end

else

begin

counter <= counter + 1;

end

end

STATE\_Y2 : begin

led\_register <= 8'b10001010;

if (counter >= half\_sec && cross)

begin

state <= STATE\_PE2;

counter <= 0;

cross <= 0;

end

else if (counter >= half\_sec)

begin

state <= STATE\_G1;

counter <= 0;

end

else

begin

counter <= counter + 1;

end

end

STATE\_PE1 : begin

led\_register <= 8'b10010001;

if (counter >= one\_sec)

begin

state <= STATE\_G2;

counter <= 0;

end

else

begin

counter <= counter + 1;

end

end

STATE\_PE2 : begin

led\_register <= 8'b10010001;

if (counter >= one\_sec)

begin

state <= STATE\_G1;

counter <= 0;

end

else

begin

counter <= counter + 1;

end

end

default: state <= STATE\_G1;

endcase

end

endmodule

lab3\_example.v:

`timescale 1ns / 1ps

module lab3\_example(

input wire [4:0] okUH,

output wire [2:0] okHU,

inout wire [31:0] okUHU,

inout wire okAA,

output [7:0] led,

input sys\_clkn,

input sys\_clkp

);

wire [31:0] pedestrian;

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 clk;

IBUFGDS osc\_clk(

.O(clk),

.I(sys\_clkp),

.IB(sys\_clkn)

);

okHost hostIF (

.okUH(okUH),

.okHU(okHU),

.okUHU(okUHU),

.okClk(okClk),

.okAA(okAA),

.okHE(okHE),

.okEH(okEH)

);

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(pedestrian));

FSMFSM(.clk(clk),

.pedestrian(pedestrian),

.led(led));

endmodule

lab3\_TestBench.v:

`timescale 1ns / 1ps

module lab3\_TestBench();

//Declare wires and registers that will interface with the module under test

//Registers are initilized to known states. Wires cannot be initilized.

reg clk = 1;

wire [7:0] led;

reg [31:0]button;

//Invoke the module that we like to test

FSM#(.one\_sec(100),.half\_sec(50)) ModuleUnderTest (.pedestrian(button),.led(led),.clk(clk));

// Generate a clock signal. The clock will change its state every 5ns.

//Remember that the test module takes sys\_clkp and sys\_clkn as input clock signals.

//From these two signals a clock signal, clk, is derived.

//The LVDS clock signal, sys\_clkn, is always in the opposite state than sys\_clkp.

always begin

#5 clk = ~clk;

end

initial begin

#0 button <=0;

#4000 button <= 1;

#500 button <= 0;

#2000

button <= 1;

end

endmodulend end STATE\_G2 : begin led\_register <= 8'b10000110; if (counter >= one\_sec) begin state <= STATE\_Y2; counter <= 0; end else begin counter <= counter + 1; end end STATE\_Y2 : begin led\_register <= 8'b10001010; if (counter >= half\_sec && cross) begin state <= STATE\_PE2; counter <= 0; cross <= 0; end else if (counter >= half\_sec) begin state <= STATE\_G1; counter <= 0; end else begin counter <= counter + 1; end end STATE\_PE1 : begin led\_register <= 8'b10010001; if (counter >= one\_sec) begin state <= STATE\_G2; counter <= 0; end else begin counter <= counter + 1; end end STATE\_PE2 : begin led\_register <= 8'b10010001; if (counter >= one\_sec) begin state <= STATE\_G1; counter <= 0; end else begin counter <= counter + 1; end end default: state <= STATE\_G1; endcase end endmodule lab3\_example.v: `timescale 1ns / 1ps module lab3\_example( input wire [4:0] okUH, output wire [2:0] okHU, inout wire [31:0] okUHU, inout wire okAA, output [7:0] led, input sys\_clkn, input sys\_clkp ); wire [31:0] pedestrian; 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 clk; IBUFGDS osc\_clk( .O(clk), .I(sys\_clkp), .IB(sys\_clkn) ); okHost hostIF ( .okUH(okUH), .okHU(okHU), .okUHU(okUHU), .okClk(okClk), .okAA(okAA), .okHE(okHE), .okEH(okEH) ); 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(pedestrian)); FSMFSM(.clk(clk), .pedestrian(pedestrian), .led(led)); endmodule lab3\_TestBench.v: `timescale 1ns / 1ps module lab3\_TestBench(); //Declare wires and registers that will interface with the module under test //Registers are initilized to known states. Wires cannot be initilized. reg clk = 1; wire [7:0] led; reg [31:0]button; //Invoke the module that we like to test FSM#(.one\_sec(100),.half\_sec(50)) ModuleUnderTest (.pedestrian(button),.led(led),.clk(clk)); // Generate a clock signal. The clock will change its state every 5ns. //Remember that the test module takes sys\_clkp and sys\_clkn as input clock signals. //From these two signals a clock signal, clk, is derived. //The LVDS clock signal, sys\_clkn, is always in the opposite state than sys\_clkp. always begin #5 clk = ~clk; end initial begin #0 button <=0; #4000 button <= 1; #500 button <= 0; #2000 button <= 1; end endmodule