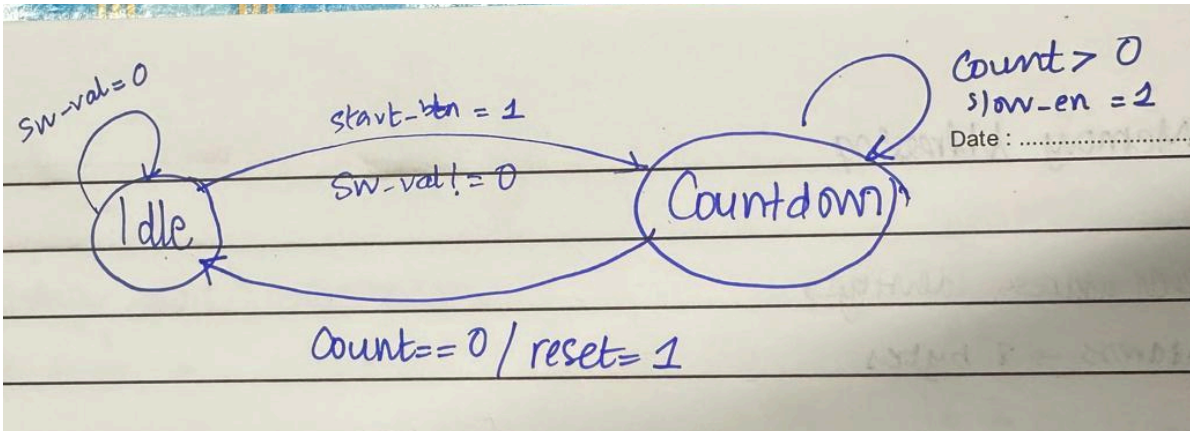
Lab 05 – Worksheet

Name: Myrah Hussain Romaisa Shazad	ID: mh10074 rb10107	Section:

Note: Assumptions and logics should be explained separately in tasks after the task results.

Task 1

FSM:



Comments

*T*he FSM consists of two states: **IDLE** and **COUNTDOWN**.

IDLE State: The FSM begins in IDLE after reset. It remains in IDLE (self-loop) as long as sw_val = 0. When start_btn = 1 and sw_val != 0, the FSM transitions to COUNTDOWN, loading the switch value into the counter.

COUNTDOWN State: The FSM remains in COUNTDOWN (self-loop) while count > 0 and slow_en = 1, decrementing the counter by one on each slow clock tick. The FSM returns to IDLE when either count = 0 or reset = 1, clearing the counter immediately.

Reset Behavior: Regardless of the current state, when reset = 1, this immediately forces the FSM back to IDLE and clears the counter to zero without waiting for a clock edge.

Task 2

Modules + Testbench:

```
module fsm_counter (
    input clk, rst,         // clock and reset
    input start_btn,        // from the debouncer
    input [15:0] sw_val,    // timer value from physical switches
    input slow_en,          // the 1-second metronome tick
    output reg [15:0] count // what we show on the LEDs
);

    reg state; // 0 is IDLE, 1 is COUNTDOWN

    always @(posedge clk or posedge rst) begin
        if (rst) begin
            state <= 0;
            count <= 0;
        end else begin
            case (state)
                0: begin // IDLE: wait for button
                    if (start_btn) begin
                        count <= sw_val; // load '5'
                        state <= 1;
                    end
                end

                1: begin // COUNTDOWN
                    if (count == 0) begin
                        state <= 0;      // back to idle
                    end else if (slow_en) begin
                        // only subtract when the 1-sec tick hits
                        count <= count - 1;
                    end
                end
            endcase
        end
    end
endmodule
```

Debouncer Module:

```
module debouncer(
    input clk,          // system clock
    input pbin,         // raw button input
    output reg pbout    // clean button output signal
);

   reg [19:0] timer;
     reg [1:0] sync; // sync the human button press to the computer clock to prevent logic errors

    always @(posedge clk) begin
        // Shift register for syncing
        sync <= {sync[0], pbin};  // for every clock tick it pulls the current state of pbin

        // If output matches input, reset timer. Otherwise, count up.
          if (sync[1] == pbout)     //checks if current button state mathces the last stable output
            timer <= 0;           //if they match, reset the timer

        else if (timer == 20'hFFFFF)      // timer reached max value
            pbout <= sync[1]; // Final stable state reached
        else
            timer <= timer + 1;
    end
endmodule
```

Switches Module:

```
`timescale 1ns / 1ps

module switches(
    input clk,              // system clock
    input rst,              // reset signal
    input [31:0] writeData,    // data for writes (unused)
    input writeEnable,         // enable for writing
    input readEnable,          // enable for reading
    input [29:0] memAddress,   // address from bus
    output reg [31:0] readData = 0, // data sent to bus
    output reg [15:0] leds     // physical switch status
);

    // handle bus reads
```

```
    always @(posedge clk) begin
        if (readEnable) begin
            readData <= {16'b0, leds}; //Pad with 0s to fit into 32 bit
        end
    end

endmodule
```

LEDs Module:

```
module leds(
    input clk, rst,              // clock and reset
    input [15:0] btns,           // button inputs
    input [31:0] writeData,      // data to write
    input writeEnable,           // write ctrl
    input readEnable,            // read ctrl
    input [29:0] memAddress,     // address bus
    input [15:0] switches,       // switch inputs
    output reg [31:0] readData,  // data sent back to bus
    output [15:0] leds           // physical led output
);

    // send count or switch data to leds
    assign leds = readData[15:0]; // connect lower bits to lights

    // bus read logic
    always @(posedge clk) begin
        if (readEnable)
            readData <= {16'b0, switches}; // pad 16-bit switches to 32-bit bus
    end
endmodule
```

Top Level Module:

```
module top_module (
    input clk,
    input rst_pin,
    input start_pin,
    input [15:0] physical_switches,
    output [15:0] physical_leds
);

    // internal wires
    wire debounced_start;
    wire [15:0] current_count;
    reg start_delay;
```

```
wire start_pulse;
wire [31:0] sw_bus_data;

// clock divider
reg [26:0] clk_div;
wire slow_en;

always @(posedge clk) begin
    if (rst_pin) clk_div <= 0;
    else if (clk_div >= 100_000_000) clk_div <= 0;
    else clk_div <= clk_div + 1;
end
assign slow_en = (clk_div == 100_000_000);

// clean button noise
debouncer u_debouncer (
    .clk(clk),
    .pbin(start_pin),
    .pbout(debounced_start)
);

// edge detector
always @(posedge clk) begin
    start_delay <= debounced_start;
end
assign start_pulse = debounced_start && !start_delay;

// Switch Interface: Grabs physical switches and puts them on sw_bus_data
switches sw_inst (
    .clk(clk),
    .rst(rst_pin),
    .writeData(32'b0),
    .writeEnable(1'b0),
    .readEnable(1'b1),
    .memAddress(30'b0),
    .readData(sw_bus_data),     // This is the 32-bit output
    .leds(physical_switches)    // This is the 16-bit input from board
);

//  FSM Counter: Takes switch values from the bus and counts down
fsm_counter u_fsm (
    .clk(clk),
    .rst(rst_pin),
    .slow_en(slow_en),
    .start_btn(start_pulse),
```

```
    .sw_val(sw_bus_data[15:0]), // Use the data from the switch interface
    .count(current_count)       // Output current value
  );

  // (c) LED Interface: Displays the 'current_count' on physical LEDs
  leds u_leds (
    .clk(clk),
    .rst(rst_pin),
    .btns(16'b0),
    .writeData(32'b0),
    .writeEnable(1'b0),
    .readEnable(1'b1),
    .memAddress(30'b0),
    .switches(current_count),   // Connect FSM output to LED input
    .readData(),
    .leds(physical_leds)        // Drive the actual board lights
  );

endmodule
```

Testbench:

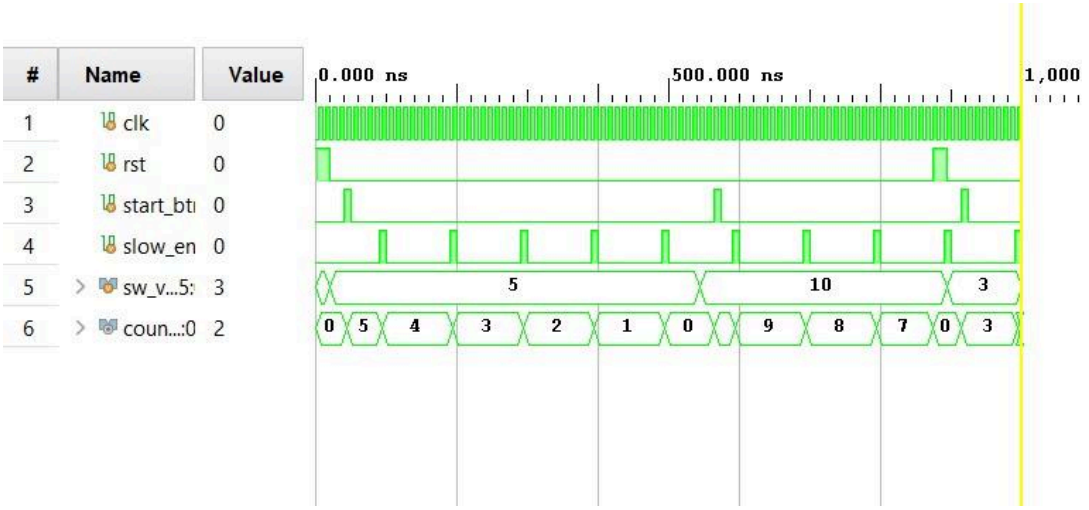
```
`timescale 1ns / 1ps

module tb_fsm();
  reg clk;
  reg rst;
  reg start_btn;
  reg slow_en;
  reg [15:0] sw_val;
  wire [15:0] count;

  // Unit Under Test
  fsm_counter uut (
    .clk(clk),
    .rst(rst),
    .start_btn(start_btn),
    .sw_val(sw_val),
    .slow_en(slow_en),
    .count(count)
  );

  // 100MHz clock (10ns period)
  always #5 clk = ~clk;
```

```
// Fix the Red 'X': Initialize slow_en and generate pulses
initial begin
    slow_en = 0; // Essential initialization
    forever begin
        #90 slow_en = 1; // Pulse high for 1 clock cycle
        #10 slow_en = 0;
    end
end

initial begin
    // --- 1. Initialization ---
    clk = 0; rst = 1; start_btn = 0; sw_val = 0;
    #20 rst = 0; // Release reset

    // --- 2. Test First Switch Instance (Count from 5) ---
    sw_val = 16'd5;
    #20 start_btn = 1;
    #10 start_btn = 0;

    // Wait for it to finish naturally to show "Versatility"
    wait(count == 0);
    #50;

    // --- 3. Test Second Switch Instance & Mid-Count Reset ---
    // Load a larger value like 10
    sw_val = 16'd10;
    #20 start_btn = 1;
    #10 start_btn = 0;

    // Let it count down a few steps (e.g., until it hits 7)
    #300;

    // Apply Reset during active countdown
    // This proves "Safety/Control": the counter must drop to 0 immediately
    $display("Applying reset mid-count...");
    rst = 1;
    #20 rst = 0;

    // --- 4. Test Recovery ---
    // Verify we can start a new countdown immediately after the reset
    sw_val = 16'd3;
    #20 start_btn = 1;
    #10 start_btn = 0;

    wait(count == 0);
```

```
        #100;

        $display("Simulation Finished Successfully");
        $finish;
    end
endmodule
```

Results:

Waveform:



The waveform shows three test cases used to verify the FSM works correctly.

Test 1 — Normal Countdown (sw_val = 5): sw_val is loaded with 5 and start_btn is pulsed to begin the countdown. The counter decrements from 5 → 4 → 3 → 2 → 1 → 0 on each slow_en pulse, then the FSM automatically returns to IDLE once it hits 0.

Test 2 — Mid-Countdown Reset (sw_val = 10): sw_val is then loaded with 10 and start_btn is pulsed again. The counter begins decrementing from 9 → 8 → 7, and while still active, rst (reset) is asserted mid-countdown. The counter immediately drops to 0, confirming that the asynchronous reset works correctly — it does not wait for a slow_en pulse, it reacts instantly.

Test 3 — Recovery After Reset (sw_val = 3): After reset is released, sw_val is loaded with 3 and start_btn is pulsed once more. The FSM recovers successfully and begins a fresh countdown from 3 → 2, confirming that after a mid-countdown reset the system is fully functional and can run a new countdown without any issues.

The slow_en signal acts as the 1-second tick that controls when each decrement happens, and start_btn is pulsed once at the start of each test case to transition the FSM from IDLE into COUNTDOWN.

Task 3

XDC file

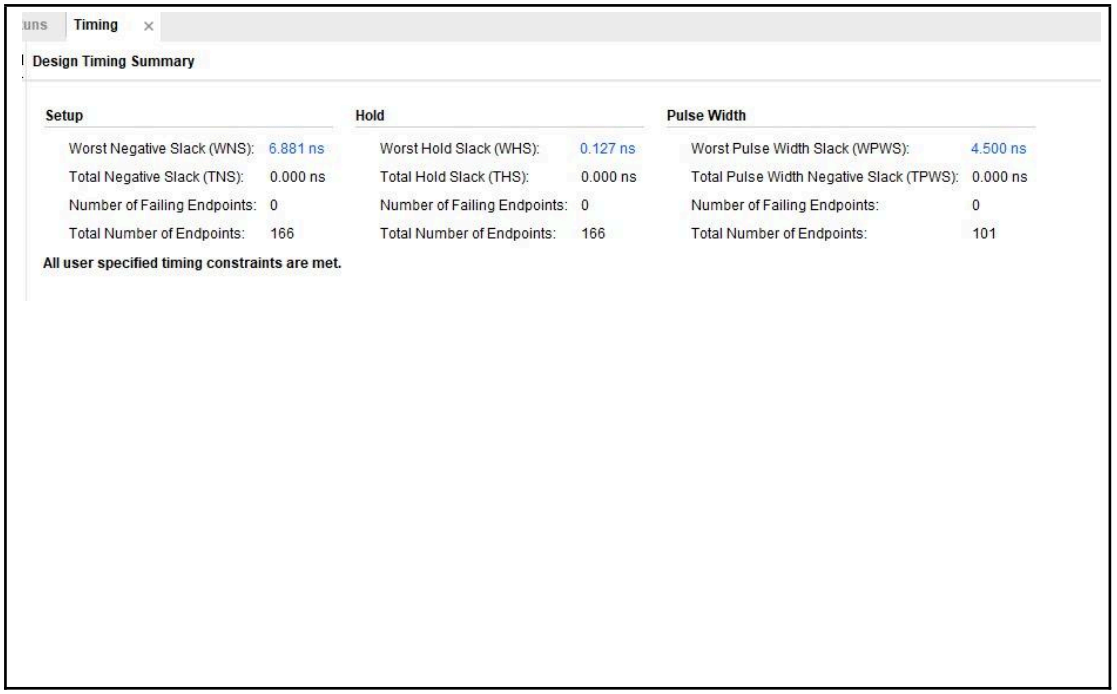
```
## ------------------------------------------------------------------------
## CLOCK SIGNAL (100 MHz)
## ------------------------------------------------------------------------
set_property PACKAGE_PIN W5 [get_ports clk]

        set_property IOSTANDARD LVCMOS33 [get_ports clk]
        # Create a 10ns period clock (100MHz) for the internal timing analysis
        create_clock -add -period 10.00 -name sys_clk_pin -waveform {0 5}
[get_ports clk]

## ------------------------------------------------------------------------
## BUTTONS (Using Center and Upper Buttons)
## ------------------------------------------------------------------------
# Center Button (U18) for rst_pin
set_property PACKAGE_PIN U18 [get_ports rst_pin]

        set_property IOSTANDARD LVCMOS33 [get_ports rst_pin]

# Upper Button (T18) for start_pin
set_property PACKAGE_PIN T18 [get_ports start_pin]

        set_property IOSTANDARD LVCMOS33 [get_ports start_pin]

## ------------------------------------------------------------------------
## PHYSICAL SWITCHES (SW0 to SW15)
## ------------------------------------------------------------------------
set_property PACKAGE_PIN V17 [get_ports {physical_switches[0]}]

        set_property IOSTANDARD LVCMOS33 [get_ports
{physical_switches[0]}]
set_property PACKAGE_PIN V16 [get_ports {physical_switches[1]}]

        set_property IOSTANDARD LVCMOS33 [get_ports
{physical_switches[1]}]
set_property PACKAGE_PIN W16 [get_ports {physical_switches[2]}]

        set_property IOSTANDARD LVCMOS33 [get_ports
{physical_switches[2]}]
set_property PACKAGE_PIN W17 [get_ports {physical_switches[3]}]

        set_property IOSTANDARD LVCMOS33 [get_ports
{physical_switches[3]}]
```

```
set_property PACKAGE_PIN W15 [get_ports {physical_switches[4]}]

        set_property IOSTANDARD LVCMOS33 [get_ports
{physical_switches[4]}]
set_property PACKAGE_PIN V15 [get_ports {physical_switches[5]}]

        set_property IOSTANDARD LVCMOS33 [get_ports
{physical_switches[5]}]
set_property PACKAGE_PIN W14 [get_ports {physical_switches[6]}]

        set_property IOSTANDARD LVCMOS33 [get_ports
{physical_switches[6]}]
set_property PACKAGE_PIN W13 [get_ports {physical_switches[7]}]

        set_property IOSTANDARD LVCMOS33 [get_ports
{physical_switches[7]}]
set_property PACKAGE_PIN V2 [get_ports {physical_switches[8]}]

        set_property IOSTANDARD LVCMOS33 [get_ports
{physical_switches[8]}]
set_property PACKAGE_PIN T3 [get_ports {physical_switches[9]}]

        set_property IOSTANDARD LVCMOS33 [get_ports
{physical_switches[9]}]
set_property PACKAGE_PIN T2 [get_ports {physical_switches[10]}]

        set_property IOSTANDARD LVCMOS33 [get_ports
{physical_switches[10]}]
set_property PACKAGE_PIN R3 [get_ports {physical_switches[11]}]

        set_property IOSTANDARD LVCMOS33 [get_ports
{physical_switches[11]}]
set_property PACKAGE_PIN W2 [get_ports {physical_switches[12]}]

        set_property IOSTANDARD LVCMOS33 [get_ports
{physical_switches[12]}]
set_property PACKAGE_PIN U1 [get_ports {physical_switches[13]}]

        set_property IOSTANDARD LVCMOS33 [get_ports
{physical_switches[13]}]
set_property PACKAGE_PIN T1 [get_ports {physical_switches[14]}]

        set_property IOSTANDARD LVCMOS33 [get_ports
{physical_switches[14]}]
```

```
set_property PACKAGE_PIN R2 [get_ports {physical_switches[15]}]

        set_property IOSTANDARD LVCMOS33 [get_ports
{physical_switches[15]}]

## ------------------------------------------------------------------------
## PHYSICAL LEDs (LED0 to LED15)
## ------------------------------------------------------------------------
set_property PACKAGE_PIN U16 [get_ports {physical_leds[0]}]

        set_property IOSTANDARD LVCMOS33 [get_ports {physical_leds[0]}]
set_property PACKAGE_PIN E19 [get_ports {physical_leds[1]}]

        set_property IOSTANDARD LVCMOS33 [get_ports {physical_leds[1]}]
set_property PACKAGE_PIN U19 [get_ports {physical_leds[2]}]

        set_property IOSTANDARD LVCMOS33 [get_ports {physical_leds[2]}]
set_property PACKAGE_PIN V19 [get_ports {physical_leds[3]}]

        set_property IOSTANDARD LVCMOS33 [get_ports {physical_leds[3]}]
set_property PACKAGE_PIN W18 [get_ports {physical_leds[4]}]

        set_property IOSTANDARD LVCMOS33 [get_ports {physical_leds[4]}]
set_property PACKAGE_PIN U15 [get_ports {physical_leds[5]}]

        set_property IOSTANDARD LVCMOS33 [get_ports {physical_leds[5]}]
set_property PACKAGE_PIN U14 [get_ports {physical_leds[6]}]

        set_property IOSTANDARD LVCMOS33 [get_ports {physical_leds[6]}]
set_property PACKAGE_PIN V14 [get_ports {physical_leds[7]}]

        set_property IOSTANDARD LVCMOS33 [get_ports {physical_leds[7]}]
set_property PACKAGE_PIN V13 [get_ports {physical_leds[8]}]

        set_property IOSTANDARD LVCMOS33 [get_ports {physical_leds[8]}]
set_property PACKAGE_PIN V3 [get_ports {physical_leds[9]}]

        set_property IOSTANDARD LVCMOS33 [get_ports {physical_leds[9]}]
set_property PACKAGE_PIN W3 [get_ports {physical_leds[10]}]

        set_property IOSTANDARD LVCMOS33 [get_ports {physical_leds[10]}]
set_property PACKAGE_PIN U3 [get_ports {physical_leds[11]}]

        set_property IOSTANDARD LVCMOS33 [get_ports {physical_leds[11]}]
```

```
set_property PACKAGE_PIN P3 [get_ports {physical_leds[12]}]

        set_property IOSTANDARD LVCMOS33 [get_ports {physical_leds[12]}]
set_property PACKAGE_PIN N3 [get_ports {physical_leds[13]}]

        set_property IOSTANDARD LVCMOS33 [get_ports {physical_leds[13]}]
set_property PACKAGE_PIN P1 [get_ports {physical_leds[14]}]

        set_property IOSTANDARD LVCMOS33 [get_ports {physical_leds[14]}]
set_property PACKAGE_PIN L1 [get_ports {physical_leds[15]}]

        set_property IOSTANDARD LVCMOS33 [get_ports {physical_leds[15]}]
```

Timing Summary



Interpretation:

The timing diagram indicates that the design meets all timing requirements.

The timing analysis indicates that our design meets all timing requirements. Slack refers to the difference between the time a signal is required to arrive and the time it actually arrives. A positive slack means the signal arrives early with margin to spare, while a negative slack indicates a timing violation where the signal is too slow and the design would malfunction.

Our setup analysis shows a worst slack of **+6.881 ns**, with zero total negative slack and no failing endpoints across 166 paths. This indicates that all data paths meet the required setup timing constraints with a very high margin, meaning our logic comfortably completes its operations well within the 10 ns clock period imposed by the 100 MHz clock.

Our hold analysis reports a worst hold slack of **+0.127 ns**, with zero total hold slack and no failing endpoints. This confirms that our data remains stable after clock edges and no hold violations exist. The hold slack is smaller than the setup slack, which is expected and acceptable as hold timing is determined by the minimum

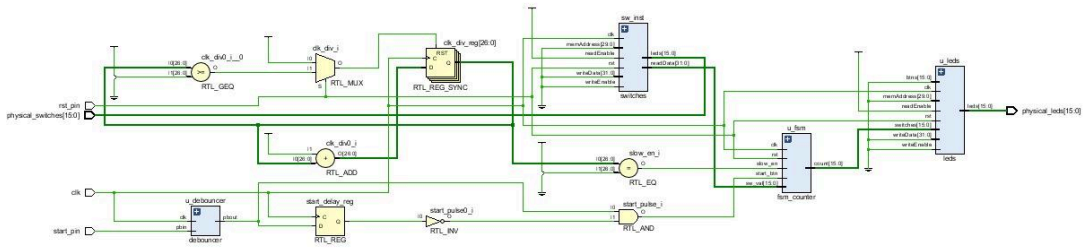
path delay through the logic rather than the maximum; as long as it remains positive, our design is safe from race conditions.

The pulse width analysis also shows positive slack (**+4.500 ns**) across 101 endpoints with no violations, meaning the clock signal meets the minimum high and low pulse width requirements for the FPGA hardware.

Since all slack values are positive and no failing endpoints exist across any analysis, our design has no timing violations. This means the synthesized hardware will operate reliably on the FPGA at 100 MHz without any risk of unpredictable behavior under normal operating conditions. Overall, our design is fully timing-clean and operates safely at 100 MHz with no risk of failure under normal temperature or voltage variation.

Task 4

RTL Schematic:



The RTL schematic confirms the correct connectivity of all modules as defined in the top-level module. The clk signal is distributed globally to all modules. The rst_pin signal feeds into the debouncer (u_debouncer) which outputs a clean pbout signal that is then passed through a register and edge detector circuit — consisting of RTL_REG, RTL_INV, and RTL_AND — to produce a single-cycle start_pulse from the start_pin input.

The physical_switches[15:0] feed directly into the switches module (sw_inst) which outputs the synchronized switch value onto the internal bus. This value is passed to

the FSM counter (u_fsm) as sw_val[15:0], along with slow_en generated by the clock divider and start_btn from the edge detector.

The clock divider is implemented using a 27-bit register (RTL_REG_SYNC), a comparator, and an adder that together generate the slow_en pulse once every 100,000,000 clock cycles, acting as the 1-second metronome tick.

The FSM processes its inputs and drives count[15:0] into the LEDs module (u_leds) through its switches port, which then outputs to physical_leds[15:0]. The schematic validates that the signal flow matches the intended design: switches feed the FSM, the FSM drives the LEDs, reset is debounced before reaching any module, and the start button is edge-detected to ensure a single clean trigger per press.

Results

Lab 05 – Designing FSM Using FPGA Switches and LEDs
Assessment Rubrics

Points Distribution

Task No.	LR2 Code	LR 5 Results	LR11 Design	AR7
Prelab		/15		/10 & /10
Task 1	-	-	/10	
Task 2	/15	/10		
Task 3		/15		
Task 4		/15		
Total Points	/80			/20
CLO Mapped	CLO 1			

For description of different levels of the mapped rubrics, please refer the provided Lab Evaluation Assessment Rubrics and Affective Domain Assessment Rubrics.

#	Assessment Elements	Level 1: Unsatisfactory Points 0-1	Level 2: Developing Points 2	Level 3:Good Points 3	Level 4:Exemplary Points 4
LR2	**Program/Code / Simulation Model/ Network Model**	Program/code/simulation model/network model does not implement the required functionality and has several errors. The student is not able to utilize even the basic tools of the software.	Program/code/simulation model/network model has some errors and does not produce completely accurate results. Student has limited command on the basic tools of the software.	Program/code/simulation model/network model gives correct output but not efficiently implemented or implemented by computationally complex routine.	Program/code/simulation /network model is efficiently implemented and gives correct output. Student has full command on the basic tools of the software.
LR5	**Results & Plots**	Figures/ graphs / tables are not developed or are poorly constructed with erroneous results. Titles, captions, units are	Figures, graphs and tables are drawn but contain errors. Titles, captions, units are not accurate. Data presentation is not too clear.	All figures, graphs, tables are correctly drawn but contain minor errors or some of the details are missing.	Figures / graphs / tables are correctly drawn and appropriate titles/captions and proper units are mentioned. Data presentation is systematic.

		not mentioned. Data is presented in an obscure manner.			
AR9	**Report**	All the in-lab tasks are not included in report and / or the report is submitted too late.	Most of the tasks are included in report but are not well explained. All the necessary figures / plots are not included. Report is submitted after due date.	Good summary of most the in-lab tasks is included in report. The work is supported by figures and plots with explanations. The report is submitted timely.	Detailed summary of the in-lab tasks is provided. All tasks are included and explained well. Data is presented clearly including all the necessary figures, plots and tables.