Name: Nabeeha Aamir	Student ID: na10076	Section: T-4
Divya Lakhani	dl10267	
Areesha Ashfaq	aa09879	

Lab 5: Designing FSM Using FPGA Switches and LEDs

Objective

The objective of this lab is to design, implement, and verify a Finite State Machine (FSM) on an FPGA using input switches, LEDs, and a reset button. Students will learn how to interface user inputs with hardware logic, implement state-based control, and manage synchronous counters using Verilog HDL.

By the end of this lab, students will be able to:

- Design an FSM using a state diagram
- Interface FPGA switches and LEDs with control logic
- Implement a decrement counter controlled by FSM states
- Integrate provided switch, button debouncing, and LED interface modules
- Verify FSM behavior through FPGA synthesis and on-board testing

System Description

The system behavior is defined as follows:

- The system continuously monitors the FPGA switches for input.
- If the switch input value is **zero**, the system remains in the input-waiting state.
- When the switch input becomes **non-zero**, the FSM:
  - Captures the switch value
  - Displays the value on the LEDs
  - Starts a decrement counter from the captured value down to zero
- While counting down:
  - The LEDs reflect the current counter value
  - Switch inputs are ignored
- When the counter reaches zero:
  - The FSM automatically returns to the input-waiting state
- At any time during counting, pressing the **reset button**:
  - Immediately returns the FSM to the input-waiting state
  - Clears the counter and LED output

Note: The switch interface, button debouncing interface, and LED interface are provided and must be used without modification.

Task 1: FSM Design and State Diagram

Objective

To design an FSM that controls system behavior based on switch input, counter value, and reset signal.

Procedure

(a) Identify the required FSM states, such as:

- Input Waiting State
- Countdown State
- Reset State (if implemented explicitly)

(b) Draw a complete FSM state diagram showing:

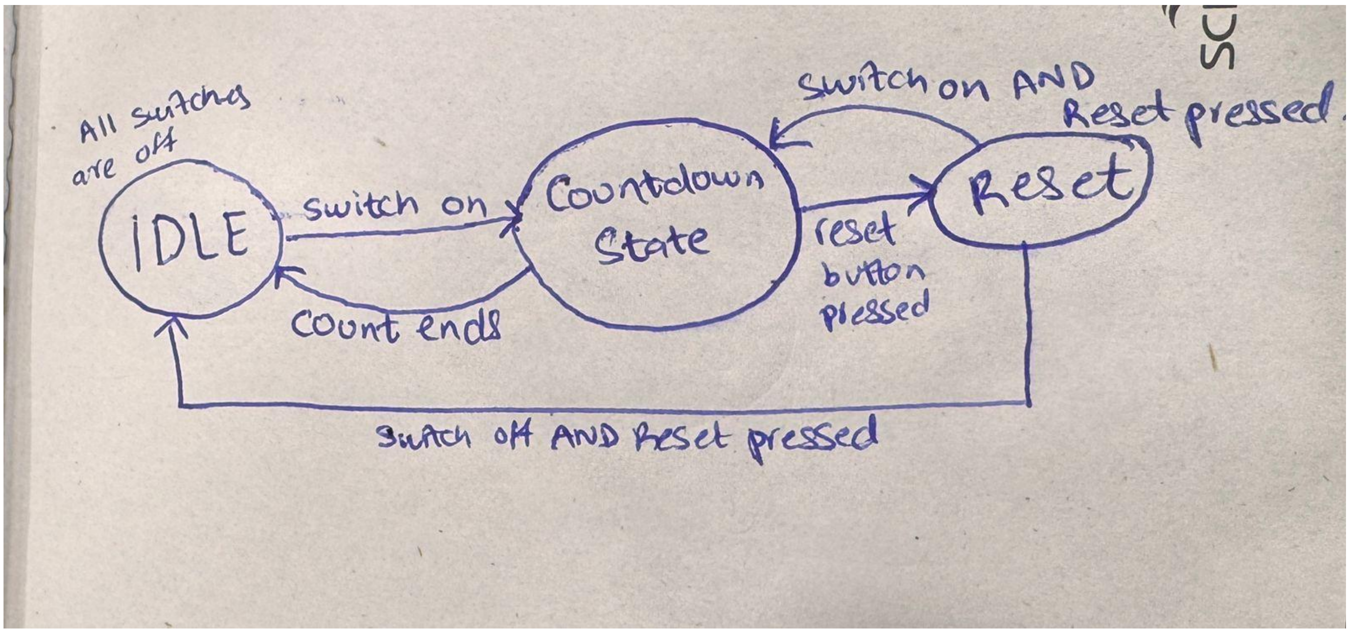
- State transitions
- Conditions for transitions (switch input, counter reaching zero, reset)
- Outputs associated with each state

(c) Clearly indicate:

- How switch input is latched
- When the counter is enabled or disabled
- How LEDs are driven in each state

Deliverables

- FSM state diagram with labeled transitions and states



Task 2: Verilog FSM and Counter Implementation

Objective

To implement the FSM and decrement counter in Verilog HDL and verify functionality through simulation.

Procedure

(a) Implement the FSM in Verilog:

- Use synchronous state transitions triggered by the system clock
- Include an asynchronous or synchronous reset as specified

(b) Implement a decrement counter:

- Load the counter with the switch input value
- Decrement the counter on each clock cycle while enabled

- Stop counting when the counter reaches zero

(c) Integrate the provided modules:

- Switch interface:

```
module switches(
    input clk,
    input rst,
    input [31:0] writeData,
    input writeEnable,
    input readEnable,
    input  [29:0]  memAddress,

    output reg  [31:0] readData = 0, // not to be read
    output reg [15:0] leds
);
...

endmodule
```

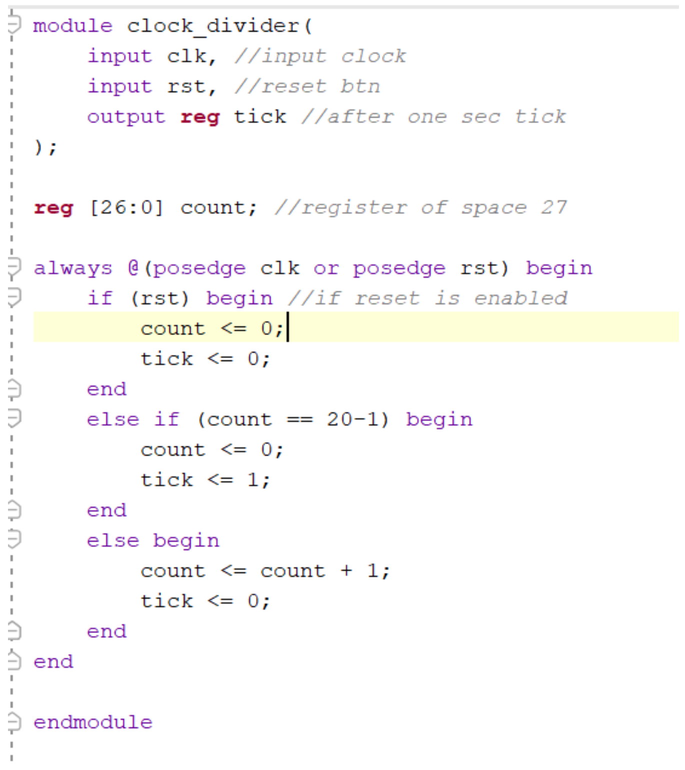
- Button debouncing interface:

```
module debouncer(
    input clk,
    input pbin,
    output pbout
```

- LED interface:

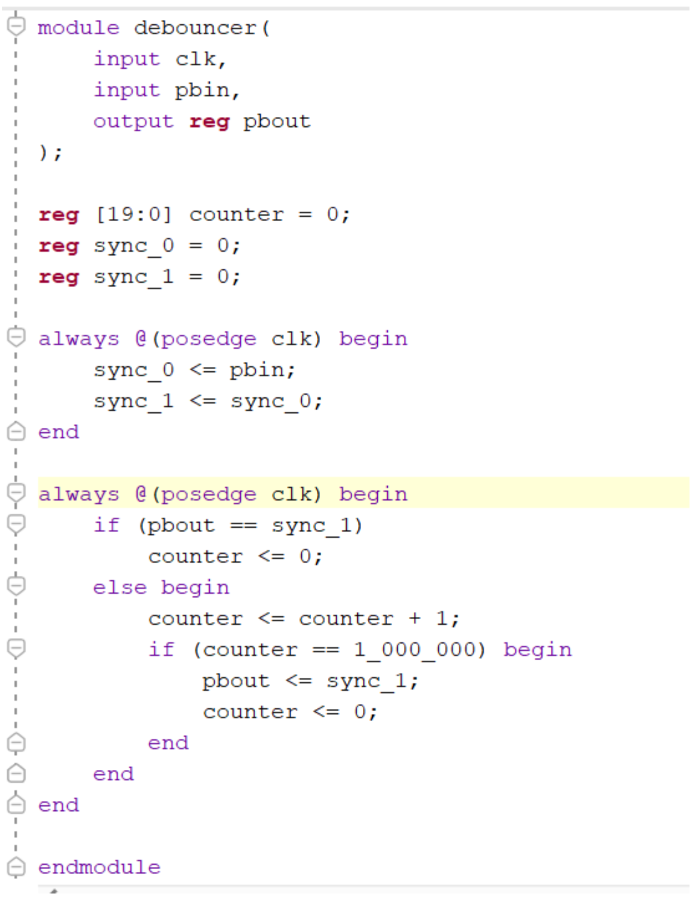
```
module leds(
    input clk, rst,
    input [15:0] btns,
    input [31:0] writeData, //    not to be written
    input writeEnable, // not to be used
    input readEnable,
    input  [29:0]  memAddress,
    input [15:0] switches,

    output reg  [31:0] readData
```

Clock divider module:



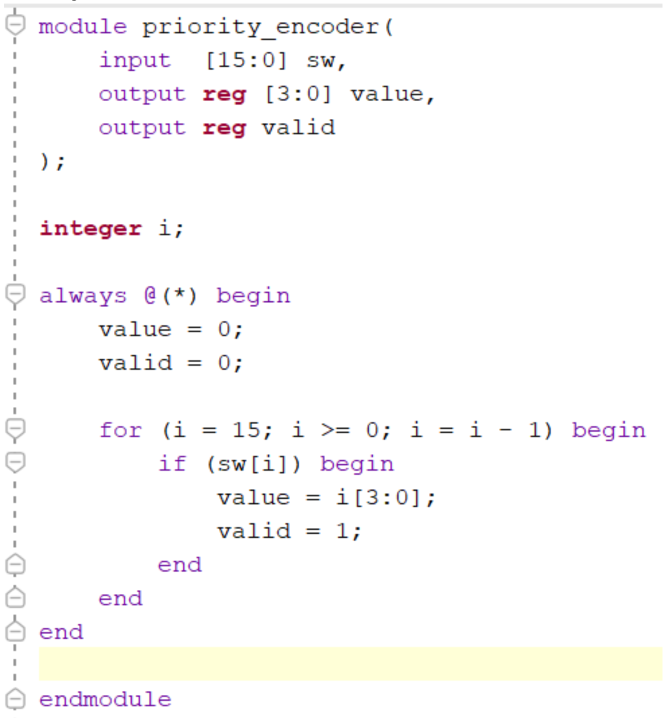
```
module clock_divider(
    input clk, //input clock
    input rst, //reset btn
    output reg tick //after one sec tick
);

reg [26:0] count; //register of space 27

always @(posedge clk or posedge rst) begin
    if (rst) begin //if reset is enabled
        count <= 0;
        tick <= 0;
    end
    else if (count == 20-1) begin
        count <= 0;
        tick <= 1;
    end
    else begin
        count <= count + 1;
        tick <= 0;
    end
end

endmodule
```

Debouncer module:



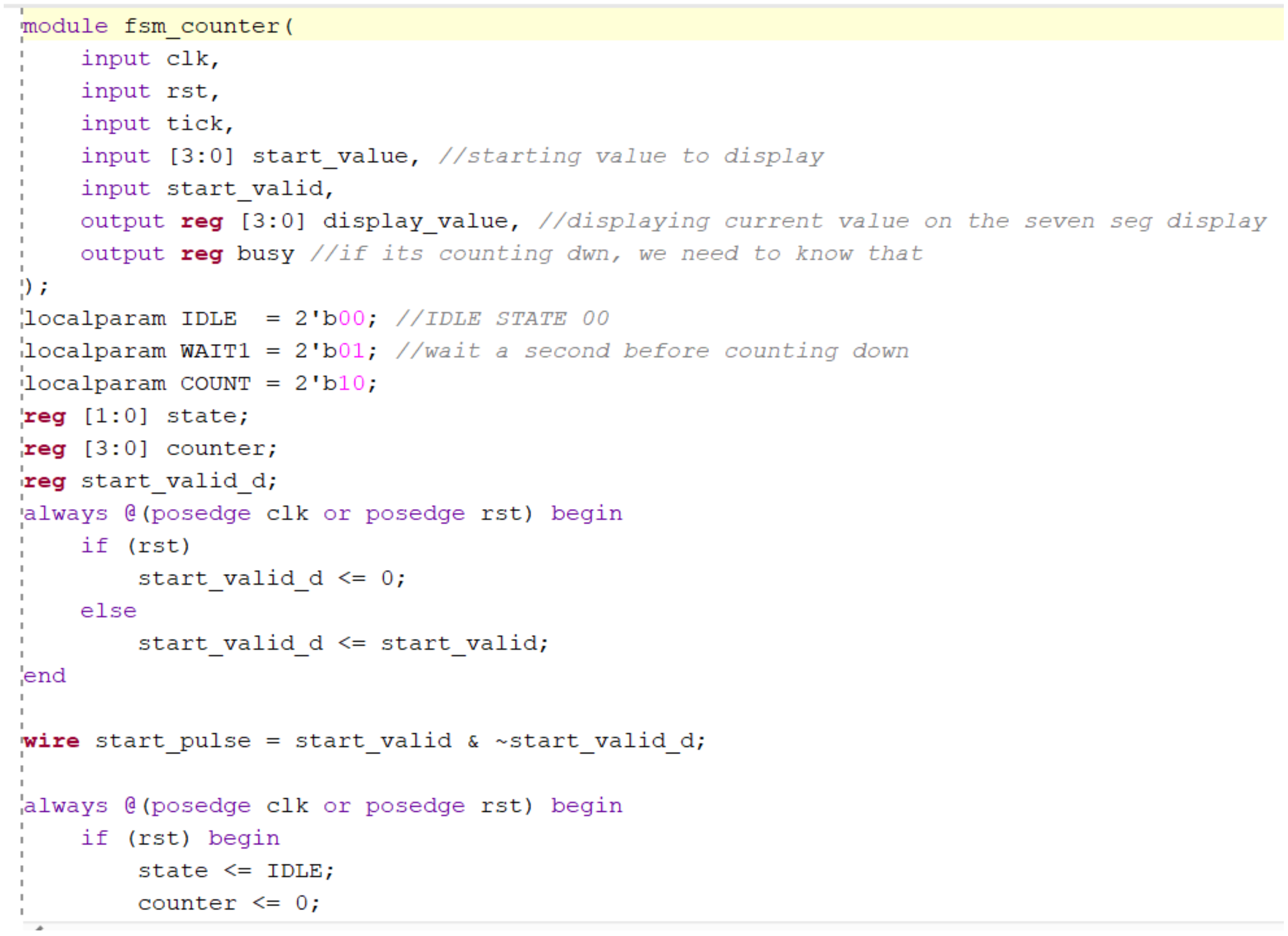
```
module debouncer(
    input clk,
    input pbin,
    output reg pbout
);

reg [19:0] counter = 0;
reg sync_0 = 0;
reg sync_1 = 0;

always @(posedge clk) begin
    sync_0 <= pbin;
    sync_1 <= sync_0;
end

always @(posedge clk) begin
    if (pbout == sync_1)
        counter <= 0;
    else begin
        counter <= counter + 1;
        if (counter == 1_000_000) begin
            pbout <= sync_1;
            counter <= 0;
        end
    end
end

endmodule
```

Priorty encoder module:

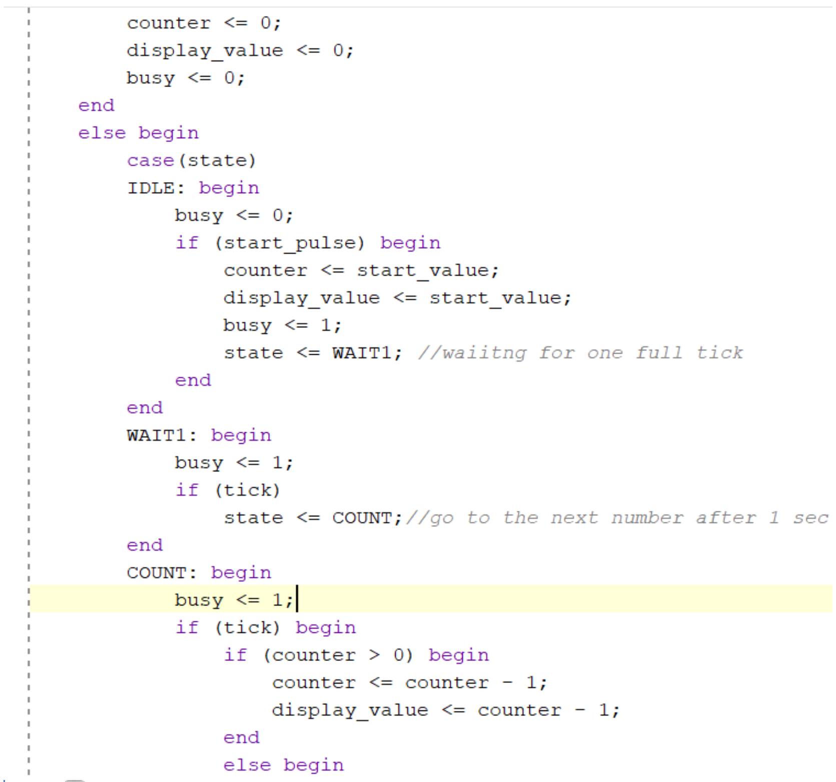


```
module priority_encoder(
    input  [15:0] sw,
    output reg [3:0] value,
    output reg valid
);

integer i;

always @(*) begin
    value = 0;
    valid = 0;

    for (i = 15; i >= 0; i = i - 1) begin
        if (sw[i]) begin
            value = i[3:0];
            valid = 1;
        end
    end
end

endmodule
```

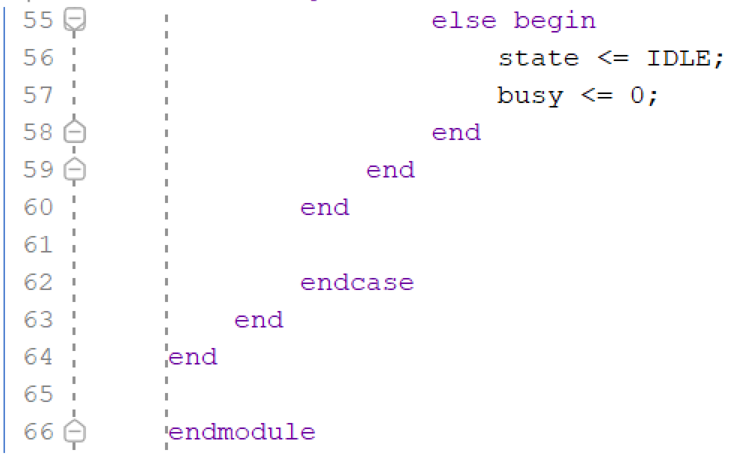
Fsm counter module:



```
module fsm_counter(
    input clk,
    input rst,
    input tick,
    input [3:0] start_value, //starting value to display
    input start_valid,
    output reg [3:0] display_value, //displaying current value on the seven seg display
    output reg busy //if its counting dwn, we need to know that
);
localparam IDLE  = 2'b00; //IDLE STATE 00
localparam WAIT1 = 2'b01; //wait a second before counting down
localparam COUNT = 2'b10;
reg [1:0] state;
reg [3:0] counter;
reg start_valid_d;
always @(posedge clk or posedge rst) begin
    if (rst)
        start_valid_d <= 0;
    else
        start_valid_d <= start_valid;
end

wire start_pulse = start_valid & ~start_valid_d;

always @(posedge clk or posedge rst) begin
    if (rst) begin
        state <= IDLE;
        counter <= 0;
```

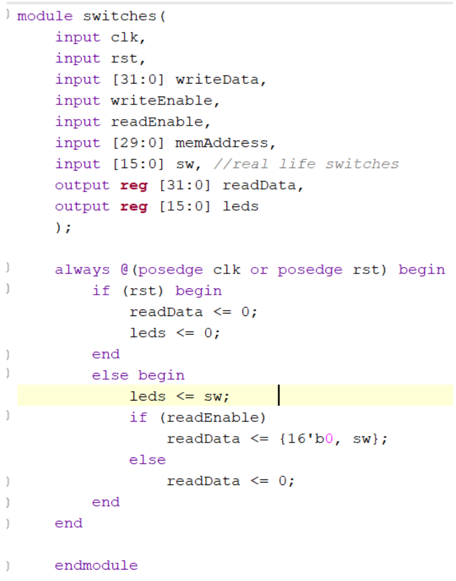


```
            counter <= 0;
            display_value <= 0;
            busy <= 0;
        end
        else begin
            case(state)
            IDLE: begin
                busy <= 0;
                if (start_pulse) begin
                    counter <= start_value;
                    display_value <= start_value;
                    busy <= 1;
                    state <= WAIT1; //waiitng for one full tick
                end
            end
            WAIT1: begin
                busy <= 1;
                if (tick)
                    state <= COUNT;//go to the next number after 1 sec
            end
            COUNT: begin
                busy <= 1;
                if (tick) begin
                    if (counter > 0) begin
                        counter <= counter - 1;
                        display_value <= counter - 1;
                    end
                    else begin
```



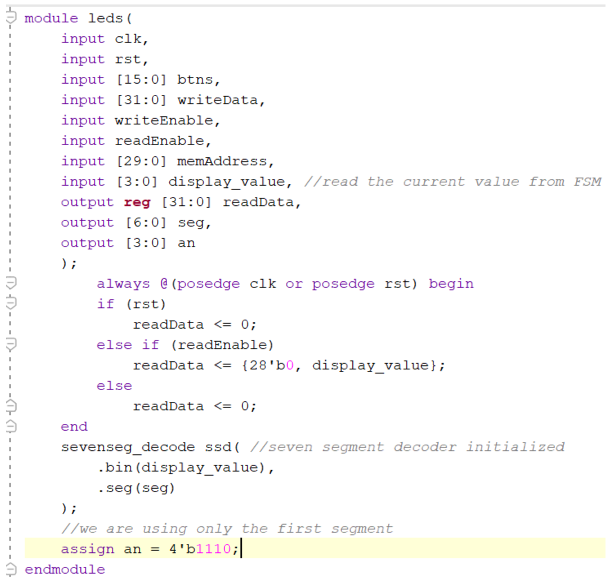
```
                    else begin
                        state <= IDLE;
                        busy <= 0;
                    end
                end
            end

            endcase
        end
    end

    endmodule
```

Switches module:



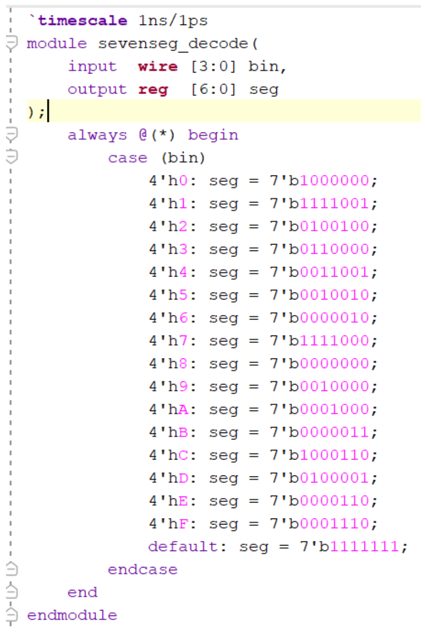
```
module switches(
    input clk,
    input rst,
    input [31:0] writeData,
    input writeEnable,
    input readEnable,
    input [29:0] memAddress,
    input [15:0] sw, //real life switches
    output reg [31:0] readData,
    output reg [15:0] leds
    );

    always @(posedge clk or posedge rst) begin
        if (rst) begin
            readData <= 0;
            leds <= 0;
        end
        else begin
            leds <= sw;
            if (readEnable)
                readData <= {16'b0, sw};
            else
                readData <= 0;
        end
    end

    endmodule
```

LEDs module:



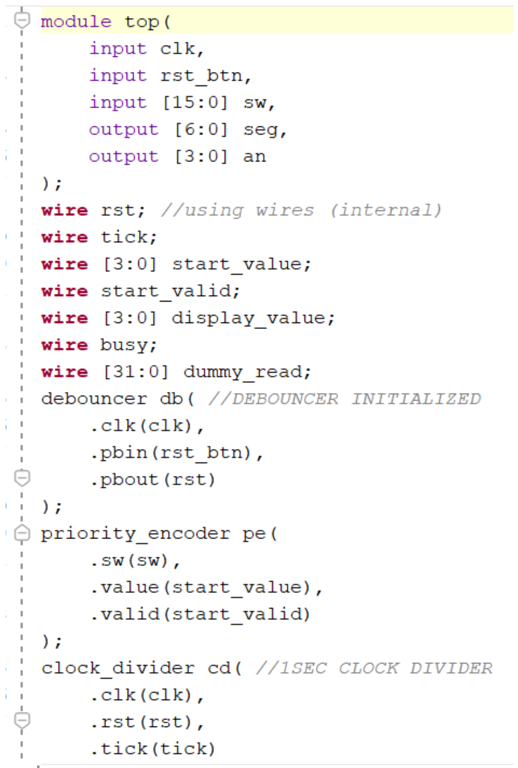
```
module leds(
    input clk,
    input rst,
    input [15:0] btns,
    input [31:0] writeData,
    input writeEnable,
    input readEnable,
    input [29:0] memAddress,
    input [3:0] display_value, //read the current value from FSM
    output reg [31:0] readData,
    output [6:0] seg,
    output [3:0] an
    );
        always @(posedge clk or posedge rst) begin
        if (rst)
            readData <= 0;
        else if (readEnable)
            readData <= {28'b0, display_value};
        else
            readData <= 0;
    end
    sevenseg_decode ssd( //seven segment decoder initialized
        .bin(display_value),
        .seg(seg)
    );
    //we are using only the first segment
    assign an = 4'b1110;
endmodule
```

Seven segment decoder:

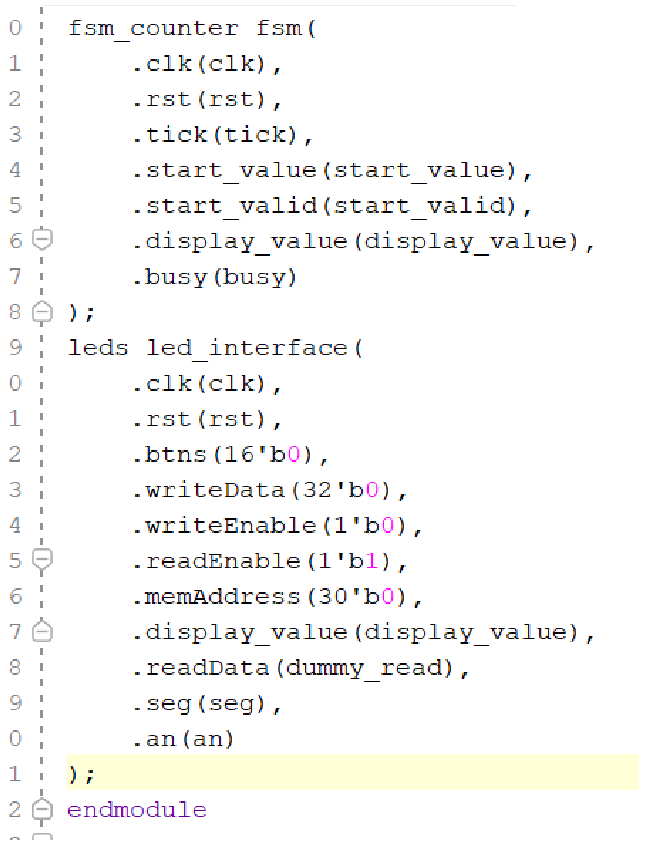


```
`timescale 1ns/1ps
module sevenseg_decode(
    input  wire [3:0] bin,
    output reg  [6:0] seg
);
    always @(*) begin
        case (bin)
            4'h0: seg = 7'b1000000;
            4'h1: seg = 7'b1111001;
            4'h2: seg = 7'b0100100;
            4'h3: seg = 7'b0110000;
            4'h4: seg = 7'b0011001;
            4'h5: seg = 7'b0010010;
            4'h6: seg = 7'b0000010;
            4'h7: seg = 7'b1111000;
            4'h8: seg = 7'b0000000;
            4'h9: seg = 7'b0010000;
            4'hA: seg = 7'b0001000;
            4'hB: seg = 7'b0000011;
            4'hC: seg = 7'b1000110;
            4'hD: seg = 7'b0100001;
            4'hE: seg = 7'b0000110;
            4'hF: seg = 7'b0001110;
            default: seg = 7'b1111111;
        endcase
    end
endmodule
```

TOP MODULE:



```verilog
module top(
    input clk,
    input rst_btn,
    input [15:0] sw,
    output [6:0] seg,
    output [3:0] an
);
wire rst; //using wires (internal)
wire tick;
wire [3:0] start_value;
wire start_valid;
wire [3:0] display_value;
wire busy;
wire [31:0] dummy_read;
debouncer db( //DEBOUNCER INITIALIZED
    .clk(clk),
    .pbin(rst_btn),
    .pbout(rst)
);
priority_encoder pe(
    .sw(sw),
    .value(start_value),
    .valid(start_valid)
);
clock_divider cd( //1SEC CLOCK DIVIDER
    .clk(clk),
    .rst(rst),
    .tick(tick)
```

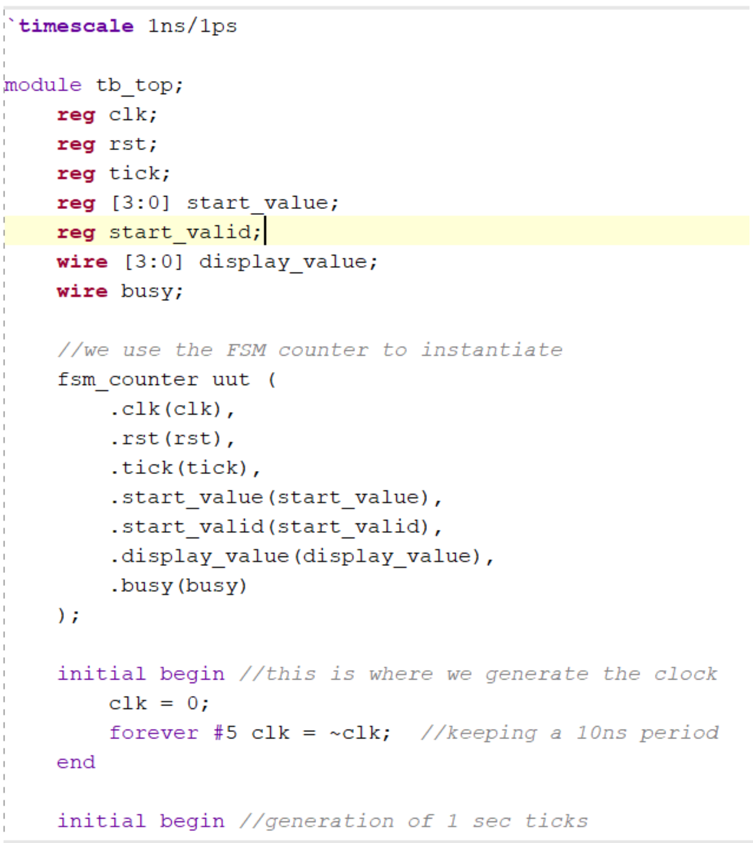


```verilog
fsm_counter fsm(
    .clk(clk),
    .rst(rst),
    .tick(tick),
    .start_value(start_value),
    .start_valid(start_valid),
    .display_value(display_value),
    .busy(busy)
);
leds led_interface(
    .clk(clk),
    .rst(rst),
    .btns(16'b0),
    .writeData(32'b0),
    .writeEnable(1'b0),
    .readEnable(1'b1),
    .memAddress(30'b0),
    .display_value(display_value),
    .readData(dummy_read),
    .seg(seg),
    .an(an)
);
endmodule
```

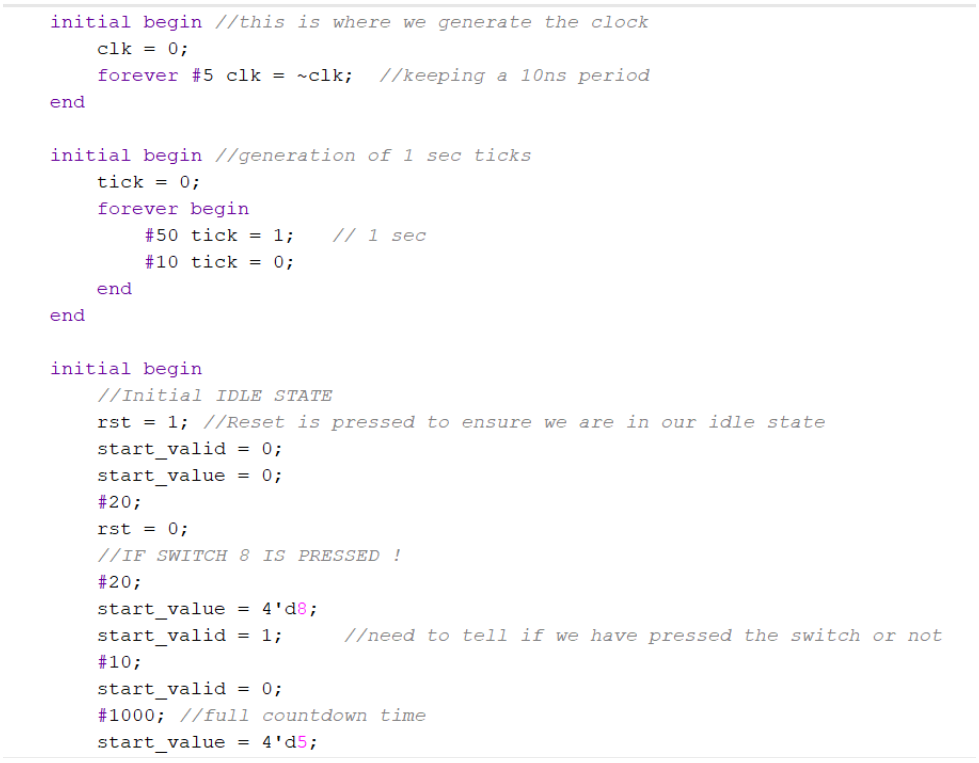
Develop a testbench to:

- Apply different switch input values
- Verify FSM transitions
- Confirm correct countdown behavior
- Verify reset functionality

Test bench :

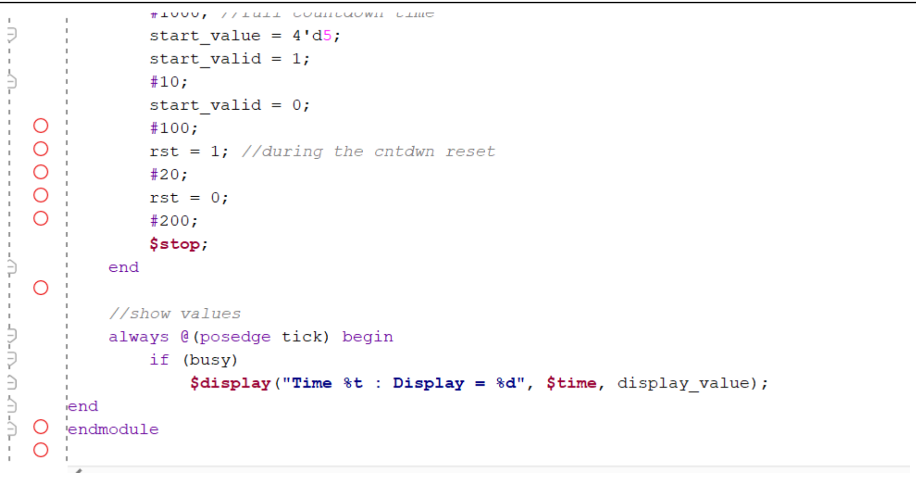


```
`timescale 1ns/1ps

module tb_top;
    reg clk;
    reg rst;
    reg tick;
    reg [3:0] start_value;
    reg start_valid;
    wire [3:0] display_value;
    wire busy;

    //we use the FSM counter to instantiate
    fsm_counter uut (
        .clk(clk),
        .rst(rst),
        .tick(tick),
        .start_value(start_value),
        .start_valid(start_valid),
        .display_value(display_value),
        .busy(busy)
    );

    initial begin //this is where we generate the clock
        clk = 0;
        forever #5 clk = ~clk;  //keeping a 10ns period
    end

    initial begin //generation of 1 sec ticks
```



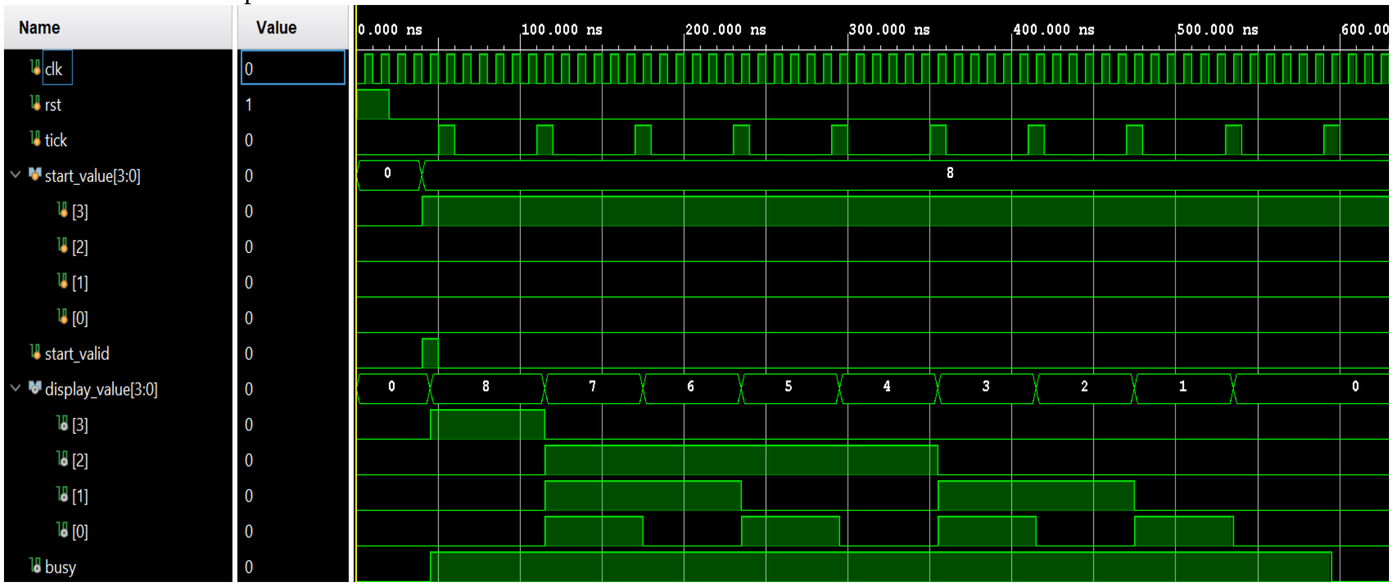
```
    initial begin //this is where we generate the clock
        clk = 0;
        forever #5 clk = ~clk;  //keeping a 10ns period
    end

    initial begin //generation of 1 sec ticks
```

```
        tick = 0;
        forever begin
            #50 tick = 1;   // 1 sec
            #10 tick = 0;
        end
    end

    initial begin
        //Initial IDLE STATE
        rst = 1; //Reset is pressed to ensure we are in our idle state
        start_valid = 0;
        start_value = 0;
        #20;
        rst = 0;
        //IF SWITCH 8 IS PRESSED !
        #20;
        start_value = 4'd8;
        start_valid = 1;     //need to tell if we have pressed the switch or not
        #10;
        start_valid = 0;
        #1000; //full countdown time
        start_value = 4'd5;
```



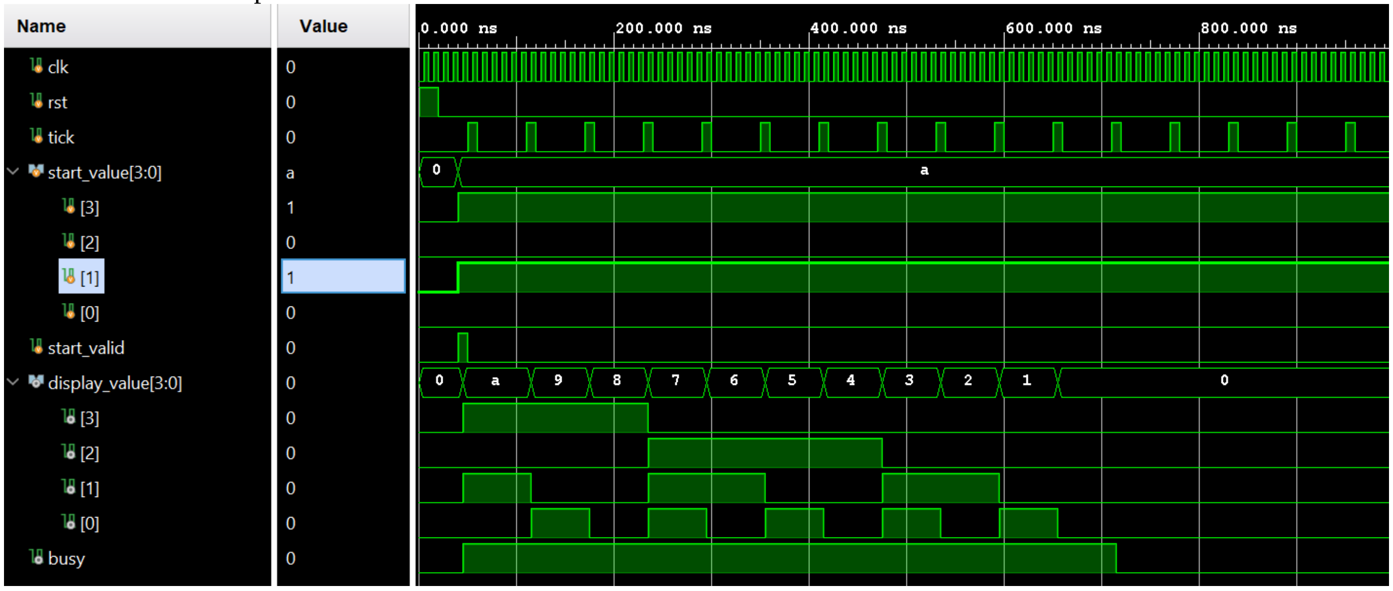
```
        #1000; //full countdown time
        start_value = 4'd5;
        start_valid = 1;
        #10;
        start_valid = 0;
        #100;
        rst = 1; //during the cntdwn reset
        #20;
        rst = 0;
        #200;
        $stop;
    end

    //show values
    always @(posedge tick) begin
        if (busy)
            $display("Time %t : Display = %d", $time, display_value);
end
endmodule
```

Case 1 : Switch 8 is pressed



Name	Value
clk	0
rst	1
tick	0
start_value[3:0]	0
[3]	0
[2]	0
[1]	0
[0]	0
start_valid	0
display_value[3:0]	0
[3]	0
[2]	0
[1]	0
[0]	0
busy	0

Case 2 : switch 10 is pressed:



Name	Value
clk	0
rst	0
tick	0
start_value[3:0]	a
[3]	1
[2]	0
[1]	1
[0]	0
start_valid	0
display_value[3:0]	0
[3]	0
[2]	0
[1]	0
[0]	0
busy	0

Example Pin Constraint (Switch)

Map Switch Input to FPGA Pin

```
set_property PACKAGE_PIN V17 [get_ports {switches[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {switches[0]}]
```

Explanation:

- PACKAGE_PIN V17 specifies the physical FPGA pin
- IOSTANDARD LVCMOS33 defines voltage level (3.3V CMOS)

Example Pin Constraint (LED)

Map LED Output to FPGA Pin

```
set_property PACKAGE_PIN U16 [get_ports {leds[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {leds[0]}]
```

Clock Constraint

Define System Clock (100 MHz)

```
create_clock -period 10.000 -name sys_clk [get_ports clk]
```

Explanation: The clock period of 10 ns corresponds to 100 MHz. All logic must complete within this time.

Understanding Timing Slack

$$\text{Slack} = \text{Required Time} - \text{Actual Path Delay}$$

Slack Value	Meaning	Status
Positive	Timing met	PASS
Zero	Critical timing	WARNING
Negative	Timing violation	FAIL

Real Timing Report Examples

Example 1 – Good Design (PASS)

```
Slack (MET) : 3.215ns
Required Time : 10.000ns
Path Delay : 6.785ns
```

Interpretation: Design runs safely at 100 MHz.

Example 2 – Marginal Design (CRITICAL)

```
Slack (MET) : 0.042ns
Required Time : 10.000ns
Path Delay : 9.958ns
```

Interpretation: Design barely meets timing and may fail under temperature/voltage variation.

Example 3 – Failing Design (VIOLATION)

```
Slack (VIOLATED) : -1.372ns
Required Time : 10.000ns
Path Delay : 11.372ns
```

Interpretation: Design cannot operate at 100 MHz — must be optimized or clock slowed.

Procedure

(a) Add an XDC file to your project

(b) Assign pin constraints to all switches, LEDs, reset, and clock

(c) Add a clock constraint

(d) Run synthesis

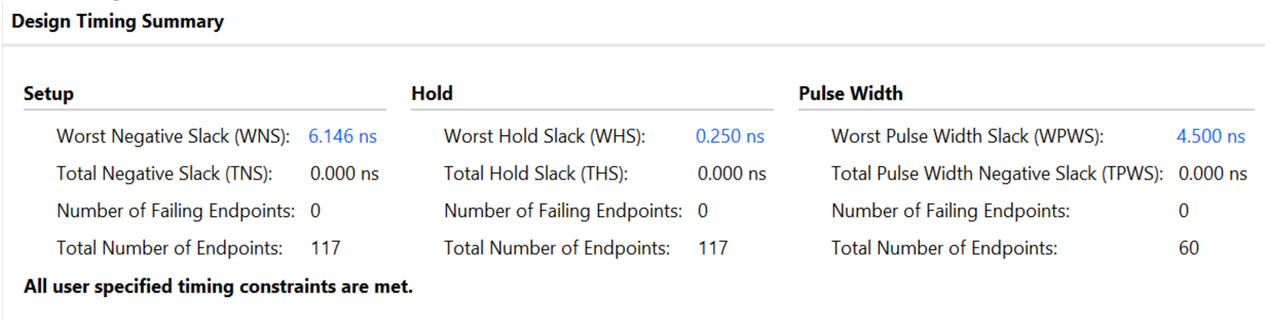
(e) Report your Worst Negative Slack (WNS)

Deliverables

- XDC file
- Screenshot of Vivado Timing Summary
- WNS value and interpretation (PASS / MARGINAL / FAIL)

```
set_property IOSTANDARD LVCMOS33 [get_ports {an[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {an[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {an[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {an[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {led[15]}]
set_property IOSTANDARD LVCMOS33 [get_ports {led[14]}]
set_property IOSTANDARD LVCMOS33 [get_ports {led[13]}]
set_property IOSTANDARD LVCMOS33 [get_ports {led[12]}]
set_property IOSTANDARD LVCMOS33 [get_ports {led[11]}]
set_property IOSTANDARD LVCMOS33 [get_ports {led[10]}]
set_property IOSTANDARD LVCMOS33 [get_ports {led[9]}]
set_property IOSTANDARD LVCMOS33 [get_ports {led[8]}]
set_property IOSTANDARD LVCMOS33 [get_ports {led[7]}]
set_property IOSTANDARD LVCMOS33 [get_ports {led[6]}]
set_property IOSTANDARD LVCMOS33 [get_ports {led[5]}]
set_property IOSTANDARD LVCMOS33 [get_ports {led[4]}]
set_property IOSTANDARD LVCMOS33 [get_ports {led[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {led[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {led[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {led[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg[6]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg[5]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg[4]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {sw[15]}]
set_property IOSTANDARD LVCMOS33 [get_ports {sw[14]}]
set_property IOSTANDARD LVCMOS33 [get_ports {sw[13]}]
set_property IOSTANDARD LVCMOS33 [get_ports {sw[12]}]
set_property IOSTANDARD LVCMOS33 [get_ports {sw[11]}]
set_property IOSTANDARD LVCMOS33 [get_ports {sw[10]}]
set_property IOSTANDARD LVCMOS33 [get_ports {sw[9]}]
set_property IOSTANDARD LVCMOS33 [get_ports {sw[8]}]
set_property IOSTANDARD LVCMOS33 [get_ports {sw[7]}]
set_property IOSTANDARD LVCMOS33 [get_ports {sw[6]}]
set_property IOSTANDARD LVCMOS33 [get_ports {sw[5]}]
set_property IOSTANDARD LVCMOS33 [get_ports {sw[4]}]
set_property IOSTANDARD LVCMOS33 [get_ports {sw[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {sw[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {sw[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {sw[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports clk]
set_property IOSTANDARD LVCMOS33 [get_ports rst_btn]
set_property PACKAGE_PIN U18 [get_ports rst_btn]
set_property PACKAGE_PIN W5 [get_ports clk]
set_property PACKAGE_PIN V17 [get_ports {sw[0]}]
set_property PACKAGE_PIN V16 [get_ports {sw[1]}]
set_property PACKAGE_PIN W16 [get_ports {sw[2]}]
set_property PACKAGE_PIN W17 [get_ports {sw[3]}]
set_property PACKAGE_PIN W15 [get_ports {sw[4]}]
set_property PACKAGE_PIN V15 [get_ports {sw[5]}]
set_property PACKAGE_PIN W14 [get_ports {sw[6]}]
set_property PACKAGE_PIN W13 [get_ports {sw[7]}]
set_property PACKAGE_PIN V2 [get_ports {sw[8]}]
set_property PACKAGE_PIN T3 [get_ports {sw[9]}]
set_property PACKAGE_PIN T2 [get_ports {sw[10]}]
set_property PACKAGE_PIN R3 [get_ports {sw[11]}]
set_property PACKAGE_PIN W2 [get_ports {sw[12]}]
set_property PACKAGE_PIN U1 [get_ports {sw[13]}]
set_property PACKAGE_PIN T1 [get_ports {sw[14]}]
set_property PACKAGE_PIN R2 [get_ports {sw[15]}]
set_property PACKAGE_PIN W7 [get_ports {seg[0]}]
set_property PACKAGE_PIN W6 [get_ports {seg[1]}]
set_property PACKAGE_PIN U8 [get_ports {seg[2]}]
set_property PACKAGE_PIN V8 [get_ports {seg[3]}]
set_property PACKAGE_PIN U5 [get_ports {seg[4]}]
set_property PACKAGE_PIN V5 [get_ports {seg[5]}]
set_property PACKAGE_PIN U7 [get_ports {seg[6]}]
set_property PACKAGE_PIN U2 [get_ports {an[0]}]
set_property PACKAGE_PIN U4 [get_ports {an[1]}]
set_property PACKAGE_PIN V4 [get_ports {an[2]}]
set_property PACKAGE_PIN W4 [get_ports {an[3]}]
set_property PACKAGE_PIN U16 [get_ports {led[0]}]
set_property PACKAGE_PIN E19 [get_ports {led[1]}]
set_property PACKAGE_PIN U19 [get_ports {led[2]}]
set_property PACKAGE_PIN V19 [get_ports {led[3]}]
set_property PACKAGE_PIN W18 [get_ports {led[4]}]
set_property PACKAGE_PIN U15 [get_ports {led[5]}]
set_property PACKAGE_PIN U14 [get_ports {led[6]}]
set_property PACKAGE_PIN V14 [get_ports {led[7]}]
set_property PACKAGE_PIN V13 [get_ports {led[8]}]
set_property PACKAGE_PIN V3 [get_ports {led[9]}]
set_property PACKAGE_PIN W3 [get_ports {led[10]}]
set_property PACKAGE_PIN U3 [get_ports {led[11]}]
set_property PACKAGE_PIN P3 [get_ports {led[12]}]
set_property PACKAGE_PIN N3 [get_ports {led[13]}]
set_property PACKAGE_PIN P1 [get_ports {led[14]}]
set_property PACKAGE_PIN L1 [get_ports {led[15]}]
```

Worst negative slack:



Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	6.146 ns	Worst Hold Slack (WHS):	0.250 ns	Worst Pulse Width Slack (WPWS):	4.500 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	117	Total Number of Endpoints:	117	Total Number of Endpoints:	60

All user specified timing constraints are met.

The Timing Summary report shows:

- Worst Negative Slack (WNS) = **6.146 ns**
- Total Negative Slack (TNS) = **0.000 ns**
- Number of Failing Endpoints = **0**

Since the WNS is positive and TNS is zero, all timing constraints are satisfied. This indicates that the design meets the required 100 MHz timing specification and operates without setup or hold violations.

Task 4: FPGA Synthesis and On-Board Verification

Objective

To synthesize the FSM design on an FPGA and verify functionality using physical switches, LEDs, and reset button.

Procedure

(a) Integrate the FSM module into the top-level FPGA design.

(b) Assign FPGA pins for:
- Switch inputs
- LED outputs
- Reset button

(c) Synthesize the design and analyze:
- RTL schematic
- State transitions
- Counter implementation

(d) Program the FPGA and perform on-board testing:
- Apply zero and non-zero switch values
- Observe LED countdown behavior
- Press reset during countdown to verify immediate reset

(e) Demonstrate correct operation to the lab instructor or RA.

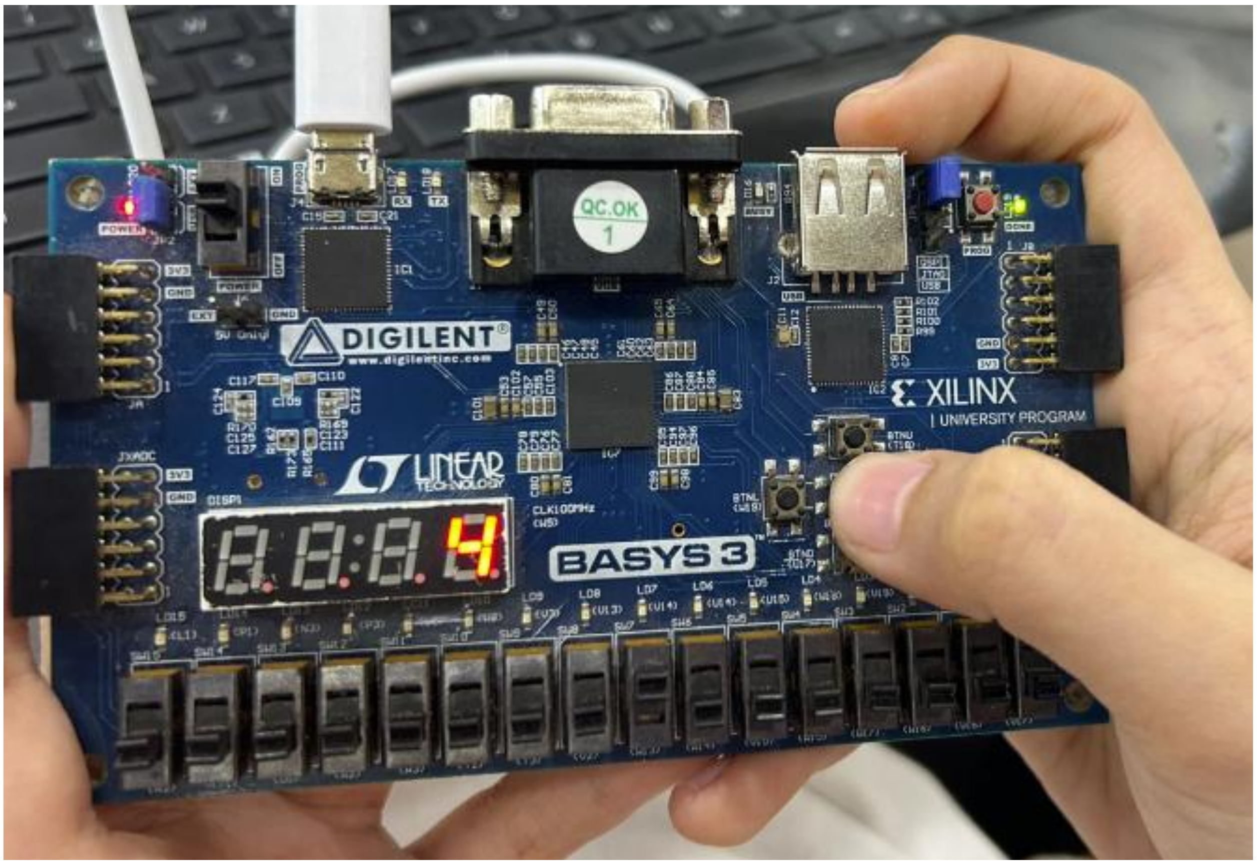
Deliverables

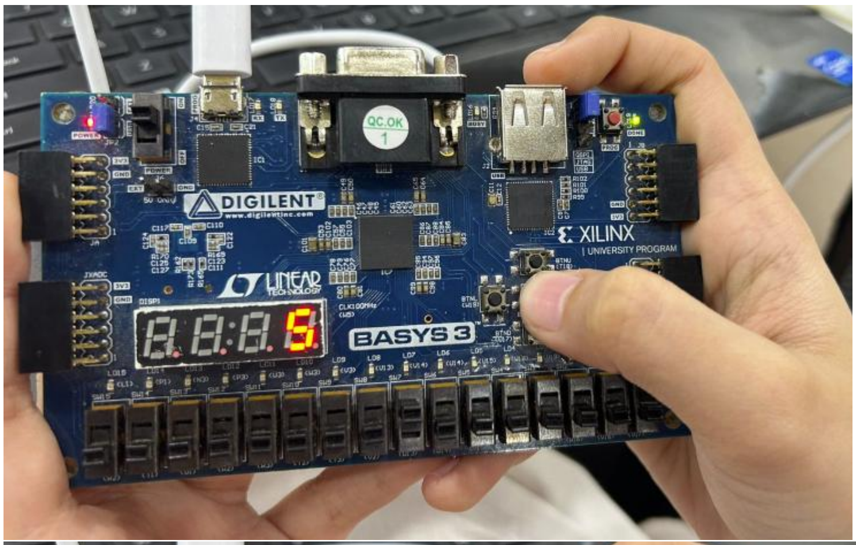
- RTL schematic screenshots
- Successful FPGA demonstration

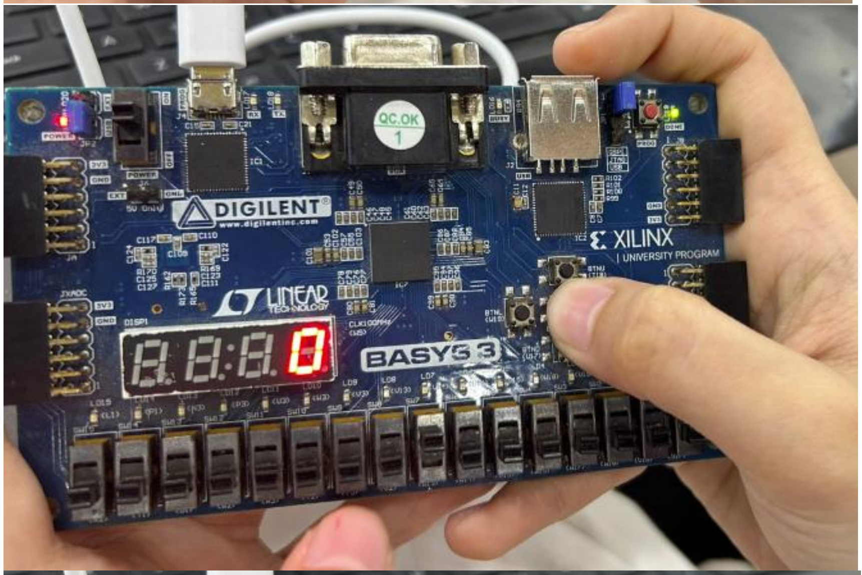
Expected Learning Outcomes

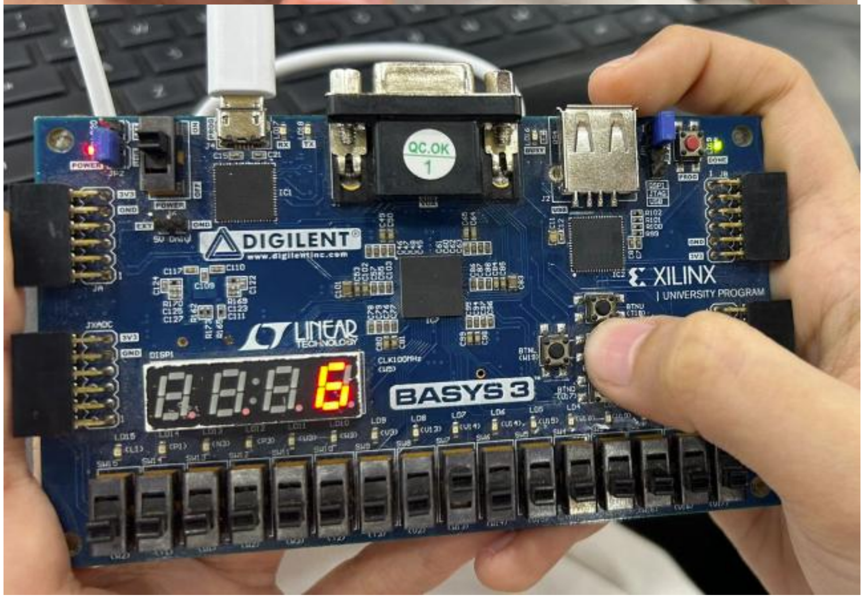
Upon successful completion of this lab, students will be able to:

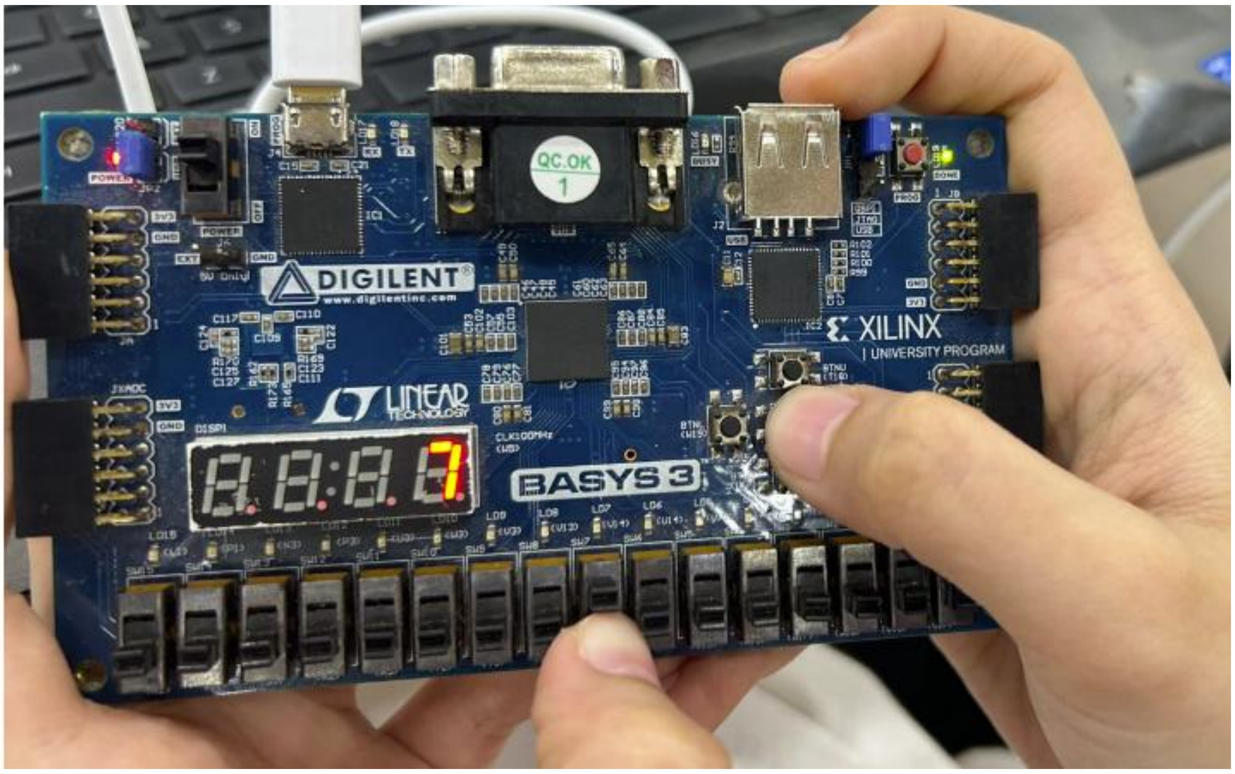
- Design FSMs for hardware control applications
- Interface user inputs with synchronous digital systems
- Implement counters controlled by FSM logic
- Debug and verify FPGA-based designs using real hardware




DIGILENT
XILINX
BASYS 3
5


DIGILENT
XILINX
BASYS 3
0


DIGILENT
XILINX
BASYS 3
6


DIGILENT
XILINX
UNIVERSITY PROGRAM
LINEAR TECHNOLOGY
BASYS 3
QC.OK 1