```verilog
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company: Worcester Polytechnic Institute
// Engineer:
//  Adam Grabowski
// Create Date: 10/02/2022 07:24:45 PM
// Design Name: ECE 3829 Lab 3
// Module Name: top_lab3
// Project Name: ECE 3829 Lab 3
// Target Devices: Basys 3 Artix-7 Development Board
// Tool Versions: Verilog 2021.1
// Description:
//  Specifies the inputs and outputs of the top module
// Dependencies:
//  None
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//  None
//////////////////////////////////////////////////////////////////////////////////
module top_lab3(
    input clk, // 100 MHz clock
    input btnC, // active-high reset
    input JA3, // ALS sdo pin
    output JA1, // ALS cs_n pin
    output JA4, // ALS sclk pin
    output [6:0] seg, // seven segment value
    output [3:0] an // display anodes
    );

    // Parameters
    parameter [3:0] wpi_id_a = 4'b1000; // display A value
    parameter [3:0] wpi_id_b = 4'b0110; // display B value

    // Internal signals
    wire clk_10MHz; // 10 MHz clock
    wire reset_n; // active-low reset
    wire [7:0] sensor_val; // sensor value

    // Instantiate clock generation module
    clock_gen clock_geni(
    .clk_100MHz(clk),
    .reset(btnC),
    .clk_10MHz(clk_10MHz),
    .reset_n(reset_n));

    // Instantiate ALS PMOD interface and control module
    als_pmod_int als_pmod_inti(
    .clk(clk_10MHz),
    .reset_n(reset_n),
    .sdo(JA3),
    .sclk(JA4),
    .cs_n(JA1),
    .value(sensor_val));

    // Instantiate seven segment display module
    seven_seg seven_segi(
    .clk(clk_10MHz),
    .reset_n(reset_n),
    .a(wpi_id_a),
    .b(wpi_id_b),
    .c(sensor_val[7:4]),
    .d(sensor_val[3:0]),
    .seg(seg),
    .an(an));

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company: Worcester Polytechnic Institute
// Engineer:
//  Adam Grabowski
// Create Date: 10/02/2022 07:28:26 PM
// Design Name: ECE 3829 Lab 3
// Module Name: als_pmod_int
// Project Name: ECE 3829 Lab 3
// Target Devices: Basys 3 Artix-7 Development Board
// Tool Versions: Verilog 2021.1
// Description:
//  Specifies the inputs and outputs of the ALS PMOD interface and control module
// Dependencies:
//  None
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//  None
//////////////////////////////////////////////////////////////////////////////////


module als_pmod_int
    #(parameter SAMPLE_RATE = 10000000)( // 1 sec sample rate
    input clk, // 10 MHz clock
    input reset_n, // active-low reset
    input sdo, // ALS PMOD sdo
    output reg sclk, // ALS PMOD sclk
    output cs_n, // ALS PMOD cs_n
    output reg [7:0] value // display value
    );

    // Parameters
    localparam TERM_COUNT_SCLK = 4'd9; // sclk terminal count
    localparam RISE_EDGE = 4'd0; // rising edge set count
    localparam FALL_EDGE = 4'd5; // falling edge set count
    localparam TERM_COUNT_CS = 5'd19; // cs_n terminal count
    localparam LOW_COUNT_CS = 5'd15; // cs_n set low count
    localparam S_IDLE = 2'b00; // idle state encoding
    localparam S_WAIT = 2'b01; // wait state encoding
    localparam S_READ = 2'b10; // read state encoding
    localparam S_DONE = 2'b11; // done state encoding

    // Internal signals
    reg [31:0] count_1sec; // 1 sec count
    reg [3:0] count_sclk; // sclk count
    wire rising_edge; // rising edge enable
    wire falling_edge; // falling edge enable
    reg [4:0] count_cs; // chip select count
    reg [1:0] current_state = S_IDLE; // current state
    reg [7:0] data; // read data

    // 1 sec sample rate counter
    always @ (posedge clk) begin
        if (reset_n == 1'b0) begin
            count_1sec <= 32'd0;
        end else begin
            if (current_state == S_IDLE) begin
                count_1sec <= 32'd0;
            end else begin
                count_1sec <= count_1sec + 32'd1;
            end
        end
    end

    // Rising and falling edge enable assigns
    assign rising_edge = {count_sclk == RISE_EDGE} ? 1'b1 : 1'b0;
    assign falling_edge = {count_sclk == FALL_EDGE} ? 1'b1 : 1'b0;

    // 4-bit sclk counter
```

```
always @ (posedge clk) begin
    if (reset_n == 1'b0) begin
        count_sclk <= 4'd0;
    end else begin
        if (count_sclk == TERM_COUNT_SCLK) begin
            count_sclk <= 4'd0;
        end else begin
            count_sclk <= count_sclk + 4'd1;
        end
    end
end

// Set sclk on rising and falling edges
always @ (posedge clk) begin
    if (reset_n == 1'b0) begin
        sclk <= 1'b0;
    end else begin
        if (rising_edge == 1'b1) begin
            sclk <= 1'b1;
        end else if (falling_edge == 1'b1) begin
            sclk <= 1'b0;
        end
    end
end

// Active-low chip select assign
assign cs_n = (count_cs > LOW_COUNT_CS) ? 1'b1 : 1'b0;

// 5-bit chip select counter
always @ (posedge clk) begin
    if (reset_n == 1'b0) begin
        count_cs <= 5'd0;
    end else begin
        if (falling_edge == 1'b1) begin
            if (count_cs == TERM_COUNT_CS) begin
                count_cs <= 5'd0;
            end else begin
                count_cs <= count_cs + 5'd1;
            end
        end
    end
end

// Read data on rising edge when chip select low
always @ (posedge clk) begin
    if (reset_n == 1'b0) begin
        data <= 8'd0;
    end else begin
        if (cs_n == 1'b0 && rising_edge == 1'b1 && count_cs > 5'd3 && count_cs < 5'd12) begin
            data[7:0] <= {data[6:0], sdo};
        end
    end
end

// State machine with 4 states: idle, wait, read, and done
always @ (posedge clk) begin
    if (reset_n == 1'b0) begin
        current_state <= S_IDLE;
    end else begin
        if (falling_edge == 1'b1) begin
            case (current_state[1:0])
                S_IDLE: begin
                    current_state <= S_WAIT;
                end S_WAIT: begin
                    if (count_1sec >= SAMPLE_RATE && count_cs == TERM_COUNT_CS) begin
                        current_state <= S_READ;
                    end
                end S_READ: begin
                    if (count_cs == LOW_COUNT_CS) begin
                        current_state <= S_DONE;
                        end
                    end S_DONE: begin
                        value[7:0] <= data[7:0];
                        current_state <= S_IDLE;
                    end
                endcase
            end
        end
    end

endmodule
```

```verilog
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company: Worcester Polytechnic Institute
// Engineer:
//  Adam Grabowski
// Create Date: 10/02/2022 07:28:57 PM
// Design Name: ECE 3829 Lab 3
// Module Name: seven_seg
// Project Name: ECE 3829 Lab 3
// Target Devices: Basys 3 Artix-7 Development Board
// Tool Versions: Verilog 2021.1
// Description:
//  Specifies the inputs and outputs of the seven segment display module
// Dependencies:
//  None
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//  None
//////////////////////////////////////////////////////////////////////////////////


module seven_seg(
    input clk, // 10 MHz clock
    input reset_n, // active-low reset
    input [3:0] a, // display A value
    input [3:0] b, // display B value
    input [3:0] c, // display C value
    input [3:0] d, // display D value
    output reg [6:0] seg, // seven segment value
    output reg [3:0] an // display anodes
    );

    // Parameters
    parameter TERM_COUNT = 8'd255; // display update terminal count
    parameter ZERO = 7'b0000001; // seven segment 0 encoding
    parameter ONE = 7'b1001111; // seven segment 1 encoding
    parameter TWO = 7'b0010010; // seven segment 2 encoding
    parameter THREE = 7'b0000110; // seven segment 3 encoding
    parameter FOUR = 7'b1001100; // seven segment 4 encoding
    parameter FIVE = 7'b0100100; // seven segment 5 encoding
    parameter SIX = 7'b0100000; // seven segment 6 encoding
    parameter SEVEN = 7'b0001111; // seven segment 7 encoding
    parameter EIGHT = 7'b0000000; // seven segment 8 encoding
    parameter NINE = 7'b0000100; // seven segment 9 encoding
    parameter TEN = 7'b0001000; // seven segment 10 encoding
    parameter ELEVEN = 7'b1100000; // seven segment 11 encoding
    parameter TWELVE = 7'b0110001; // seven segment 12 encoding
    parameter THIRTEEN = 7'b1000010; // seven segment 13 encoding
    parameter FOURTEEN = 7'b0110000; // seven segment 14 encoding
    parameter FIFTEEN = 7'b0111000; // seven segment 15 encoding

    // Internal signals
    reg [7:0] count; // display update count
    reg [2:0] dis; // selected display
    reg [3:0] sv; // selected value
    wire update_en; // update enable

    // Enable display update
    assign update_en = (count == TERM_COUNT) ? 1'b1 : 1'b0;

    // 8-bit display update counter
    always @ (posedge clk) begin
        if (reset_n == 1'b0) begin
            count <= 8'd0;
        end else begin
            if (count == TERM_COUNT) begin
                count <= 8'd0;
            end else begin
                count <= count + 8'd1;
            end
        end
    end

    // Change display if update enabled
    always @ (posedge clk) begin
        if (reset_n == 1'b0) begin
            dis <= 3'b000;
        end else begin
            if (update_en == 1'b1) begin
                case (dis[2:0])
                    3'b100: dis <= 3'b101;
                    3'b101: dis <= 3'b110;
                    3'b110: dis <= 3'b111;
                    3'b111: dis <= 3'b100;
                    default: dis <= 3'b100;
                endcase
            end
        end
    end

    // Mux to select value and update display anodes
    always @ (*) begin
        if (reset_n == 1'b0) begin
            sv = 4'b0000;
            an = 4'b1111;
        end else begin
            case (dis[2:0])
                3'b100: begin
                    sv = a;
                    an = 4'b0111;
                end 3'b101: begin
                    sv = b;
                    an = 4'b1011;
                end 3'b110: begin
                    sv = c;
                    an = 4'b1101;
                end 3'b111: begin
                    sv = d;
                    an = 4'b1110;
                end default: begin
                    sv = 4'b0000;
                    an = 4'b1111;
                end
            endcase
        end
    end

    // Decoder to convert selected value to seven segment value
    always @ (*) begin
        if (reset_n == 1'b0) begin
            seg = ZERO;
        end else begin
            case (sv[3:0])
                4'b0000: seg = ZERO;
                4'b0001: seg = ONE;
                4'b0010: seg = TWO;
                4'b0011: seg = THREE;
                4'b0100: seg = FOUR;
                4'b0101: seg = FIVE;
                4'b0110: seg = SIX;
                4'b0111: seg = SEVEN;
                4'b1000: seg = EIGHT;
                4'b1001: seg = NINE;
                4'b1010: seg = TEN;
                4'b1011: seg = ELEVEN;
                4'b1100: seg = TWELVE;
                4'b1101: seg = THIRTEEN;
                4'b1110: seg = FOURTEEN;
                4'b1111: seg = FIFTEEN;
                endcase
            end
        end

    endmodule
```

```verilog
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company: Worcester Polytechnic Institute
// Engineer:
//  Adam Grabowski
// Create Date: 10/02/2022 07:26:11 PM
// Design Name: ECE 3829 Lab 3
// Module Name: clock_gen
// Project Name: ECE 3829 Lab 3
// Target Devices: Basys 3 Artix-7 Development Board
// Tool Versions: Verilog 2021.1
// Description:
//  Specifies the inputs and outputs of the clock generation module
// Dependencies:
//  None
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//  None
//////////////////////////////////////////////////////////////////////////////////


module clock_gen(
    input clk_100MHz, // 100 MHz clock
    input reset, // active-high reset
    output clk_10MHz, // 10 MHz clock
    output reg reset_n // active-low reset
    );

    // Internal signals
    reg reset_n_d; // active-low reset delayed 1 clock cycle
    wire reset_n_dd; // active-low reset delayed 2 clock cycles

    // Instantiate MMCM clock module
    clk_mmcm_wiz clk_mmcm_wizi(
    .clk_in1(clk_100MHz),
    .reset(reset),
    .clk_10MHz(clk_10MHz),
    .locked(reset_n_dd));

    // 2 consecutive flip flops syncronize active-low reset
    always @ (posedge clk_100MHz) begin
        reset_n_d <= reset_n_dd;
        reset_n <= reset_n_d;
    end

endmodule
```