

```

`timescale 1ns / 1ps

// =====
=====

// Top Level Module
// =====
=====

module digital_clock_top(
    input clk,           // 100 MHz System Clock
    input rst,           // Reset Button
    input btn_h,         // Increment Hour Button
    input btn_m,         // Increment Minute Button
    output [6:0] seg,   // 7-segment cathode segments (a-g)
    output [3:0] an     // 7-segment anode enables
);

// Internal signals
wire one_hz_enable;      // 1 second pulse
wire refresh_clk;         // Display refresh pulse
wire btn_h_clean;         // Debounced Hour button
wire btn_m_clean;         // Debounced Minute button

// Time registers
reg [5:0] seconds = 0;
reg [5:0] minutes = 0;
reg [4:0] hours = 0;

// --- 1. Clock Divider Instantiation ---
// Generates 1Hz signal for clock and ~500Hz-1kHz for display
refreshing
clk_divider dividers (
    .clk(clk),
    .rst(rst),
    .one_hz_enable(one_hz_enable),
    .refresh_clk(refresh_clk)
);

// --- 2. Button Debouncers ---
debounce db_h (
    .clk(clk),
    .btn_in(btn_h),
    .btn_out(btn_h_clean)
);

debounce db_m (
    .clk(clk),

```

```

    .btn_in(btn_m),
    .btn_out(btn_m_clean)
);

// --- 3. Time Counting Logic ---
always @(posedge clk or posedge rst) begin
    if (rst) begin
        seconds <= 0;
        minutes <= 0;
        hours <= 0;
    end else begin

        // Standard Time Counting (1Hz tick)
        if (one_hz_enable) begin
            if (seconds == 59) begin
                seconds <= 0;
                if (minutes == 59) begin
                    minutes <= 0;
                    if (hours == 23)
                        hours <= 0;
                    else
                        hours <= hours + 1;
                end else begin
                    minutes <= minutes + 1;
                end
            end else begin
                seconds <= seconds + 1;
            end
        end else begin
            end
        end
    end
end
end

// Manual Adjustment Logic (Edge Detection on Debounced Signals)
// This logic runs independently of the 1Hz counter
reg btn_h_prev, btn_m_prev;
always @(posedge clk) begin
    btn_h_prev <= btn_h_clean;
    btn_m_prev <= btn_m_clean;

    if (!rst) begin
        // Increment Hour on rising edge of button press
        if (btn_h_clean && !btn_h_prev) begin
            if (hours == 23) hours <= 0;
            else hours <= hours + 1;
        end

        // Increment Minute on rising edge of button press
        if (btn_m_clean && !btn_m_prev) begin

```

```

        if (minutes == 59) minutes <= 0;
        else minutes <= minutes + 1;
    end
end

// --- 4. 7-Segment Display Controller ---
seven_segment_display display_driver (
    .clk(clk),
    .refresh_clk(refresh_clk),
    .rst(rst),
    .digit0(minutes % 10),      // Ones of Minutes
    .digit1(minutes / 10),      // Tens of Minutes
    .digit2(hours % 10),        // Ones of Hours
    .digit3(hours / 10),        // Tens of Hours
    .seg(seg),
    .an(an)
);

endmodule

// =====
=====

// Sub-Module: Clock Divider (Generates 1Hz and Refresh Clock)
// =====
=====

module clk_divider(
    input clk,
    input rst,
    output reg one_hz_enable,
    output reg refresh_clk
);

    // Parameters for 100 MHz clock
    parameter COUNTER_1HZ_MAX = 100000000;
    parameter COUNTER_REFRESH_MAX = 100000; // ~1 kHz refresh rate

    integer ctr_1hz = 0;
    integer ctr_refresh = 0;

    always @(posedge clk or posedge rst) begin
        if (rst) begin
            ctr_1hz <= 0;
            one_hz_enable <= 0;
            ctr_refresh <= 0;

```

```

        refresh_clk <= 0;
    end else begin
        // 1 Hz Generation
        if (ctr_1hz >= COUNTER_1HZ_MAX - 1) begin
            ctr_1hz <= 0;
            one_hz_enable <= 1; // Pulse for one clock cycle
        end else begin
            ctr_1hz <= ctr_1hz + 1;
            one_hz_enable <= 0;
        end

        // Refresh Clk Generation
        if (ctr_refresh >= COUNTER_REFRESH_MAX - 1) begin
            ctr_refresh <= 0;
            refresh_clk <= ~refresh_clk; // Toggle for simpler
usage
        end else begin
            ctr_refresh <= ctr_refresh + 1;
        end
    end
endmodule

```

```

// =====
=====

// Sub-Module: Button Debounce (Filters noise from mechanical buttons)
// =====
=====

module debounce(
    input clk,
    input btn_in,
    output reg btn_out
);

    // Shift register to filter noise (20 bits for a few hundred ms
filtering)
    reg [19:0] shift_reg;

    always @(posedge clk) begin
        shift_reg <= {shift_reg[18:0], btn_in};

        // If all bits are 1, button is definitely pressed
        if (&shift_reg)
            btn_out <= 1;
        // If all bits are 0, button is definitely released
    end
endmodule

```

```

        else if (~|shift_reg)
            btn_out <= 0;
        // Otherwise, hold previous state
        // This is not necessary as `btn_out` is a `reg` and holds its
value by default
    end
endmodule

// =====
=====

// Sub-Module: 7-Segment Display Driver (Multiplexed)
// =====
=====

module seven_segment_display(
    input clk,
    input refresh_clk,
    input rst,
    input [3:0] digit0, // Rightmost (Minutes Ones)
    input [3:0] digit1, // Minutes Tens
    input [3:0] digit2, // Hours Ones
    input [3:0] digit3, // Leftmost (Hours Tens)
    output reg [6:0] seg, // Cathodes (A-G)
    output reg [3:0] an // Anodes (Active Low)
);

reg [1:0] digit_select = 0;
reg [3:0] current_digit_val;

// Anode Switching Logic (Fast cycling)
always @(posedge refresh_clk or posedge rst) begin
    if (rst)
        digit_select <= 0;
    else
        digit_select <= digit_select + 1;
end

// Digit Selection (Output which anode is active and select the
correct value)
always @(*) begin
    case (digit_select)
        2'b00: begin
            an = 4'b1110; // Turn on Digit 0
            current_digit_val = digit0;
        end
        2'b01: begin

```

```

        an = 4'b1101; // Turn on Digit 1
        current_digit_val = digit1;
    end
    2'b10: begin
        an = 4'b1011; // Turn on Digit 2
        current_digit_val = digit2;
    end
    2'b11: begin
        an = 4'b0111; // Turn on Digit 3
        current_digit_val = digit3;
    end
    default: begin
        an = 4'b1111; // All off
        current_digit_val = 0;
    end
endcase
end

// 7-Segment Decoder (Common Anode: 0 is ON, 1 is OFF)
// Mapping: gfedcba
always @(*) begin
    case (current_digit_val)
        4'h0: seg = 7'b1000000; // 0
        4'h1: seg = 7'b1111001; // 1
        4'h2: seg = 7'b0100100; // 2
        4'h3: seg = 7'b0110000; // 3
        4'h4: seg = 7'b0011001; // 4
        4'h5: seg = 7'b0010010; // 5
        4'h6: seg = 7'b0000010; // 6
        4'h7: seg = 7'b1111000; // 7
        4'h8: seg = 7'b0000000; // 8
        4'h9: seg = 7'b0010000; // 9
        default: seg = 7'b1111111; // All off/blank
    endcase
end

endmodule

```