Power Management IC Digital Logic

Implemented on FPGA

Chase A. Lotito
E.E. Undergraduate, Southern Illinois University Carbondale

The following design lab explores the use of HDL programming (Verilog) to implement a finite state machine (FSM) to control the timing operations for a proposed power management integrated circuit. This includes 3.3V, 2.5V, and 1.2V regulator enables and low power management.

Acknowledgements:

The author would like to thank Dr. Haibo Wang and TA Kalyan Burugu for providing guidance and the Xilinx Artix-7 hardware for this design.

Section 0: Planning

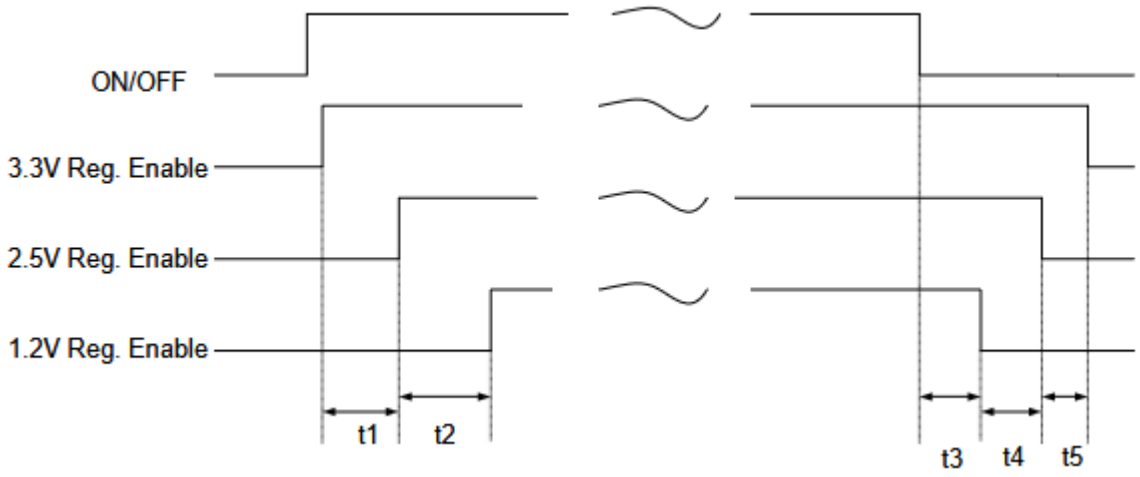

Figure 1: PMIC Timing Scheme

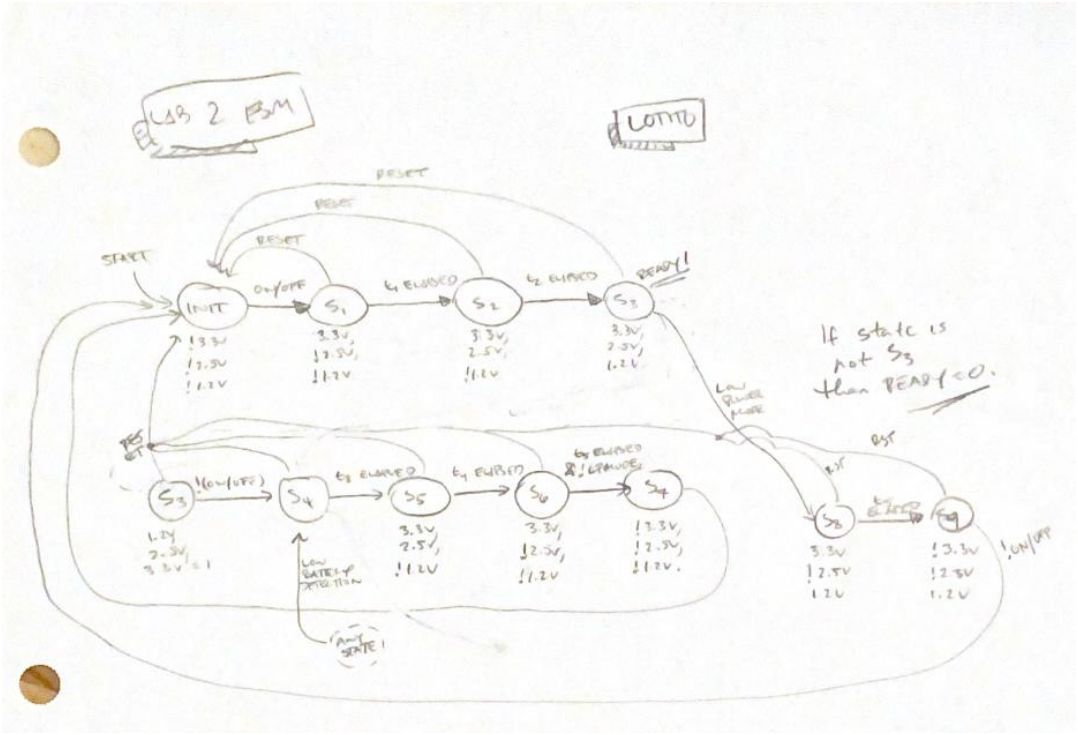

Figure 2: Design FSM

The following FSM flows through the various states which are associated with the "Turning On" sequence, "Turning Off" sequence, and "Low Power" stages.

The FSM always begins in the INITIAL state. Once the circuit is turned on, then the management circuit goes through states S_1 through S_3 which turn on the 3.3V, 2.5V, and 1.2V regulators (in that order) with timing parameters t_1 and t_2. The circuit only allows a user to put the device in "Low Power Mode" when the circuit is in the ready state, i.e. the current state is S_3.

Then, if the circuit is turned off, the circuit flows through states S_4 to S_7, which turn off the 1.2V, 2.5V, and 3.3V regulators (in that order) with timing parameters t_3, t_4, and t_5.

Then, if the circuit is ready, as in it's in state S_3, and a user puts the circuit into "Lowe Power Mode", then the state jumps to state S_8 which immediately shuts off the 2.5V regulator, and after a time duration of t_5 shuts off the 3.3V regulator. This just leaves the 1.2V regulator on. If a low battery is detected, then the circuit turns off immediately in this stage.

At any other state in the circuit, if a low battery is detected, then it shifts to state S_4 and the "Turn Off" sequence.

Inside each state, we will set the timing parameters accordingly, and only transition once we know the countdown timer has reached zero! The timer is loaded each transition, where a loading register is used to prevent multi-driving a single counter register.

Design Implementation

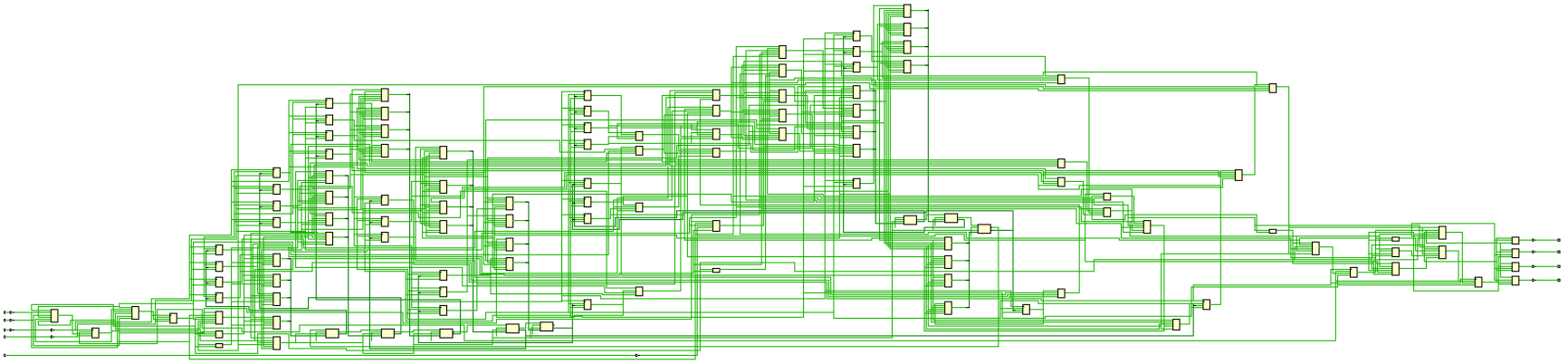

Figure 3: Design Schematic

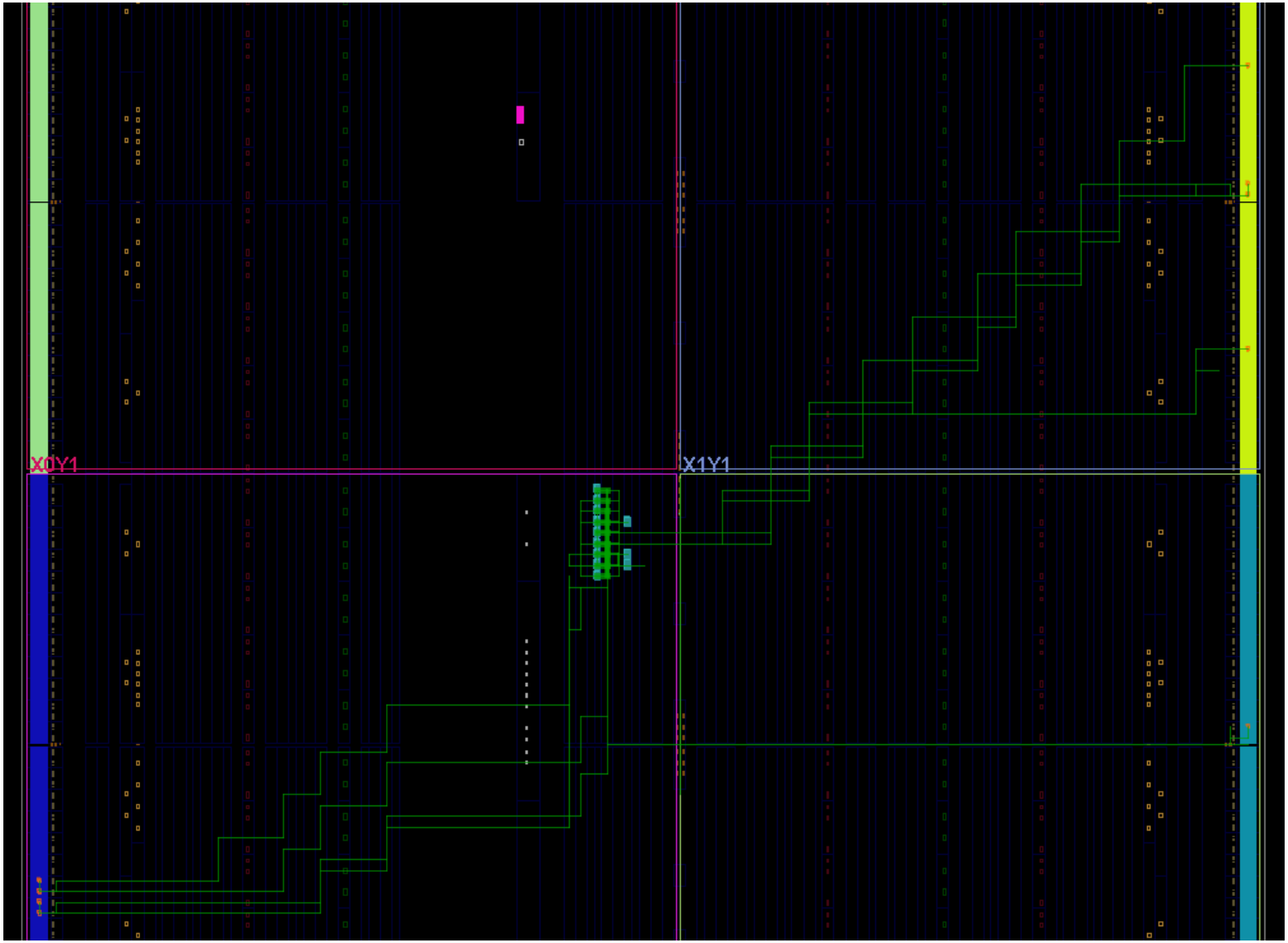

Figure 4: Implemented Circuit

Simulated Wave Forms

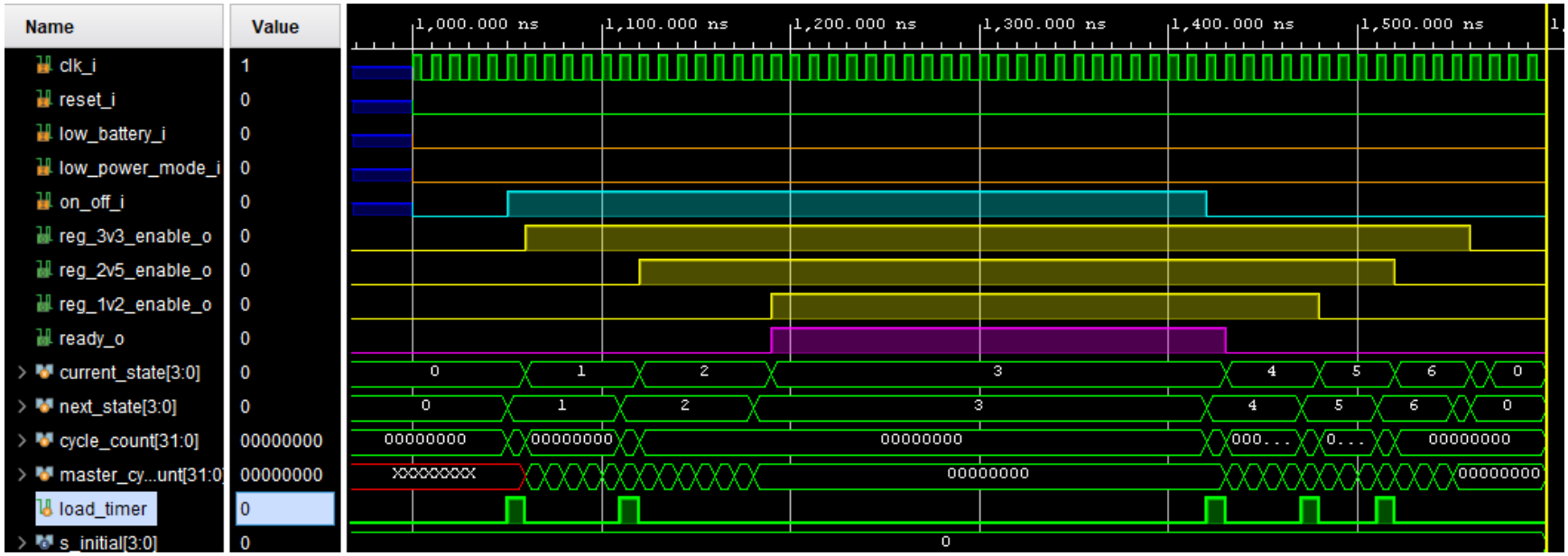

Figure 5: On and Off Sequences

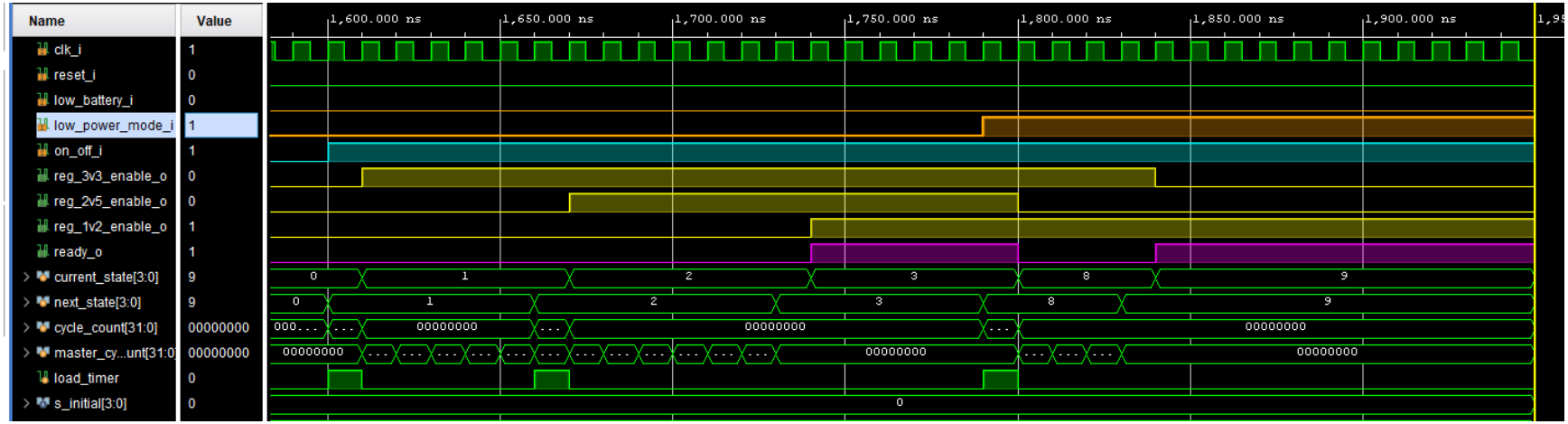

Figure 6: Low Power Mode Sequence

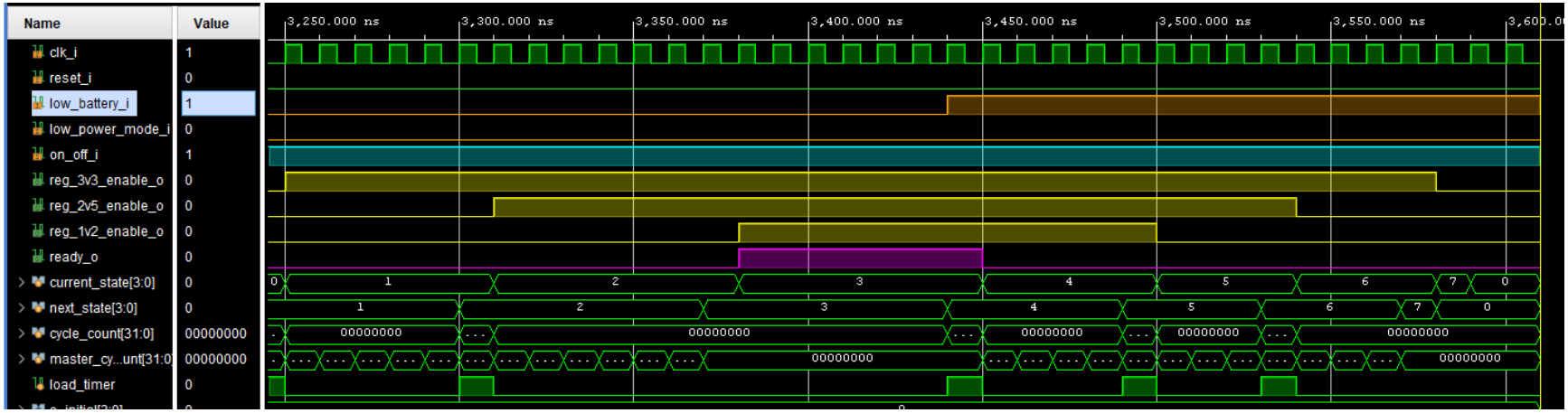

Figure 7: Low Battery Sequence

APPENDIX A: CODE

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company: Southern Illinois University Carbondale
// Engineer:
//
// Create Date: 04/20/2025 06:01:13 PM
// Design Name: Power Management IC Module
// Module Name: pmic_digital_module
// Project Name: ECE428 Lab 2
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////



module pmic_digital_module(
    input clk_i,
```

```
    input reset_i,
    input low_battery_i,
    input low_power_mode_i,
    input on_off_i,
    output reg_3v3_enable_o,
    output reg_2v5_enable_o,
    output reg_1v2_enable_o,
    output ready_o
    );

    // Define parameters for FSM
    parameter s_initial = 4'b0000;
    parameter       s_1 = 4'b0001;
    parameter       s_2 = 4'b0010;
    parameter       s_3 = 4'b0011;
    parameter       s_4 = 4'b0100;
    parameter       s_5 = 4'b0101;
    parameter       s_6 = 4'b0110;
    parameter       s_7 = 4'b0111;
    parameter       s_8 = 4'b1000;
    parameter       s_9 = 4'b1001;

    // Define timer parameters
    parameter       t1 = 5;
    parameter       t2 = 6;
    parameter       t3 = 5;
    parameter       t4 = 3;
```

```
    parameter       t5 = 3;

    // FSM State Registers
    reg [3:0] current_state;
    reg [3:0] next_state;

    // Timing Registers
    reg [31:0] cycle_count;
    reg [31:0] master_cycle_count;
    reg load_timer;

    // Initialize FSM (for testing)
    initial begin
        current_state <= 0;
        cycle_count <= 0;
        load_timer <= 0;
    end

    // Sequential FSM Logic
    always @(posedge clk_i) begin
        if (reset_i) begin
            current_state <= s_initial;
            master_cycle_count <= 0;
        end else begin
            current_state <= next_state;

            if (load_timer) begin
```

```
            master_cycle_count <= cycle_count;
        end else if (master_cycle_count != 0) begin
            master_cycle_count = master_cycle_count - 1;
        end
    end

end

// Next State FSM Logic
always @(*) begin
    // Default
    next_state = current_state;
    load_timer = 0;
    cycle_count = 0;

    case (current_state)
        s_initial: if (on_off_i && !low_battery_i) begin
                        next_state = s_1;
                        load_timer = 1;
                        cycle_count = t1;
                   end
                   else next_state = s_initial;

             s_1: if (master_cycle_count == 0) begin
                        next_state = s_2;
                        load_timer = 1;
                        cycle_count = t2;
```

```
            end else if (low_battery_i) begin
                next_state = s_4;
                load_timer = 1;
                cycle_count = t3;
            end
        s_2: if (master_cycle_count == 0) begin
                next_state = s_3;
            end else if (low_battery_i) begin
                next_state = s_4;
                load_timer = 1;
                cycle_count = t3;
            end

        s_3: if (!on_off_i || low_battery_i) begin
                next_state = s_4;
                load_timer = 1;
                cycle_count = (t3-1);
            end else if (low_power_mode_i) begin
                next_state = s_8;
                load_timer = 1;
                cycle_count = t5;
            end

        s_4: if (master_cycle_count == 0) begin
                next_state = s_5;
                load_timer = 1;
                cycle_count = t4;
```

```
                    end

                s_5: if (master_cycle_count == 0) begin
                        next_state = s_6;
                        load_timer = 1;
                        cycle_count = t5;
                    end

                s_6: if (master_cycle_count == 0) begin
                        next_state = s_7;
                    end

                s_7: next_state = s_initial;

                s_8: if (master_cycle_count == 0) begin
                        next_state = s_9;
                    end

                s_9: if (!on_off_i || low_battery_i) begin
                        next_state = s_initial;
                    end else next_state = s_9;

            default: next_state = s_initial;
        endcase
    end

    // Output Combinational Logic
```

```
    // Assignment Logic for 3.3V Register Enable
    assign reg_3v3_enable_o = (current_state == s_1) ||
                              (current_state == s_2) ||
                              (current_state == s_3) ||
                              (current_state == s_4) ||
                              (current_state == s_5) ||
                              (current_state == s_6) ||
                              (current_state == s_8);
    // Assignment Logic for 2.5V Register Enable
    assign reg_2v5_enable_o = (current_state == s_2) ||
                              (current_state == s_3) ||
                              (current_state == s_4) ||
                              (current_state == s_5);
    // Assignment Logic for 1.2V Register Enable
    assign reg_1v2_enable_o = (current_state == s_3) ||
                              (current_state == s_4) ||
                              (current_state == s_8) ||
                              (current_state == s_9);

    // Assignment for Ready Signal
    assign ready_o = (current_state == s_3) || (current_state == s_9);

endmodule
```