California Polytechnic State University, Pomona
Department of Electrical and Computer Engineering

Digital Circuit Design Using Verilog Laboratory
ECE 3300L

Lab 8

RGB LED PWM Controller

By

Jetts Crittenden (ID# 015468128) and
Evan Tram(#ID 016404570)

8/14/2025

Design:

This design is based on a 4-slot loading mechanism that allows the user to use the same switches for setting the RGB values, plus setting the resolution register value. It sequentially moves through the first slot, resolution, and ends up rotating through red, green, and lastly blue. The resolution slot indicates the resolution of the PWM period values for the RGB segments of the LED. The resolution is based on the set resolution + 1, so that the duty cycle can cleanly move through 0 to RES + 1. Overall, while the control scheme of the lab can be somewhat complicated, it provides deeper insight into how LED control systems work. Especially with limited hardware and IO.

Code:

Active Low = 0;

top_lab8.v

```
`timescale 1ns/1ps
module top_lab8 #(
    parameter integer DEFAULT_PERIOD = 8'd63,
    parameter integer DEBOUNCE_TICKS = 20,
    parameter ACTIVE_LOW = 0
)(
    input wire clk100mhz,
    input wire btnc_n,     // Active-low reset
    input wire btnr,       // Load trigger button
    input wire [7:0] sw,   // Switch input for register loading
    output wire [3:0] led, // One-hot slot indicator
    output wire rgb_r, rgb_g, rgb_b // RGB LED outputs
);

    // Reset signal
    wire rst_n = ~btnc_n;

    // Clock division: generate 1kHz control clock and 20kHz PWM
clock
    wire clk_1k, clk_pwm;
    clock_divider_fixed u_div (
        .clk_in(clk100mhz),
        .rst_n(rst_n),
        .clk_1k(clk_1k),
```

```
        .clk_pwm(clk_pwm)
    );

    // Debounce and one-pulse generator for btnr
    wire load_pulse;
    debounce_onepulse #(.STABLE_TICKS(DEBOUNCE_TICKS)) u_db (
        .clk(clk_1k),
        .rst_n(rst_n),
        .din(btnr),
        .pulse(load_pulse)
    );

    // FSM: cycles through 4 slots (res, r, g, b)
    wire [1:0] slot;
    wire [3:0] slot_oh;
    wire wr_res, wr_r, wr_g, wr_b;
    load_fsm u_fsm (
        .clk(clk_1k),
        .rst_n(rst_n),
        .load_pulse(load_pulse),
        .slot(slot),
        .slot_onehot(slot_oh),
        .wr_res(wr_res),
        .wr_r(wr_r),
        .wr_g(wr_g),
        .wr_b(wr_b)
    );

    // Display active slot on LEDs
    assign led = slot_oh;

    // Register bank: stores values from switches
    reg [7:0] reg_res, reg_r, reg_g, reg_b;
    always @(posedge clk_1k or negedge rst_n) begin
        if (!rst_n) begin
            reg_res <= DEFAULT_PERIOD;
            reg_r   <= 8'd0;
            reg_g   <= 8'd0;
            reg_b   <= 8'd0;
```

```
        end else begin
            if (wr_res) reg_res <= sw;
            if (wr_r)   reg_r   <= sw;
            if (wr_g)   reg_g   <= sw;
            if (wr_b)   reg_b   <= sw;
        end
    end

    // Double-flop synchronization into clk_pwm domain
    reg [7:0] res_q1, res_q2;
    reg [7:0] r_q1, r_q2;
    reg [7:0] g_q1, g_q2;
    reg [7:0] b_q1, b_q2;
    always @(posedge clk_pwm or negedge rst_n) begin
        if (!rst_n) begin
            res_q1 <= 0; res_q2 <= 0;
            r_q1   <= 0; r_q2   <= 0;
            g_q1   <= 0; g_q2   <= 0;
            b_q1   <= 0; b_q2   <= 0;
        end else begin
            res_q1 <= reg_res; res_q2 <= res_q1;
            r_q1   <= reg_r;   r_q2   <= r_q1;
            g_q1   <= reg_g;   g_q2   <= g_q1;
            b_q1   <= reg_b;   b_q2   <= b_q1;
        end
    end

    // PWM core: generates RGB PWM signals
    wire pwm_r, pwm_g, pwm_b;
    pwm_core u_pwm (
        .clk(clk_pwm),
        .rst_n(rst_n),
        .period(res_q2),
        .duty_r(r_q2),
        .duty_g(g_q2),
        .duty_b(b_q2),
        .pwm_r(pwm_r),
        .pwm_g(pwm_g),
        .pwm_b(pwm_b)
```

```
    );

    // LED driver: maps PWM to physical RGB pins with polarity
control
    rgb_led_driver #(.ACTIVE_LOW(ACTIVE_LOW)) u_led (
        .pwm_r(pwm_r),
        .pwm_g(pwm_g),
        .pwm_b(pwm_b),
        .led_r(rgb_r),
        .led_g(rgb_g),
        .led_b(rgb_b)
    );

endmodule
```

clock_divider_fixed

```
`timescale 1ns/1ps
module clock_divider_fixed #(
    parameter integer INPUT_HZ = 100_000_000,
    parameter integer TICK1_HZ = 1_000,
    parameter integer PWM_HZ   = 20_000
)(
    input wire clk_in,
    input wire rst_n,
    output reg clk_1k,
    output reg clk_pwm
);

localparam integer DIV1H  = (INPUT_HZ / TICK1_HZ) / 2;
localparam integer DIVPMH = (INPUT_HZ / PWM_HZ) / 2;

localparam integer WIDTH1  = $clog2(DIV1H);
localparam integer WIDTHPM = $clog2(DIVPMH);

reg [WIDTH1-1:0]  c1;
reg [WIDTHPM-1:0] c2;

always @(posedge clk_in or negedge rst_n) begin
```

```
    if (!rst_n) begin
        c1       <= 0;
        clk_1k   <= 0;
        c2       <= 0;
        clk_pwm  <= 0;
    end else begin
        // 1 kHz clock toggle
        if (c1 == DIV1H - 1) begin
            c1     <= 0;
            clk_1k <= ~clk_1k;
        end else begin
            c1 <= c1 + 1;
        end

        // 20 kHz clock toggle
        if (c2 == DIVPMH - 1) begin
            c2      <= 0;
            clk_pwm <= ~clk_pwm;
        end else begin
            c2 <= c2 + 1;
        end
    end
end

endmodule
```

Debounce_onepulse.v

```
`timescale 1ns/1ps
module debounce_onepulse #(
    parameter integer STABLE_TICKS = 20
)(
    input wire clk,
    input wire rst_n,
    input wire din,
    output reg pulse
);
```

```
localparam CNT_WIDTH = $clog2(STABLE_TICKS + 1);
reg [CNT_WIDTH-1:0] cnt;

reg d0, d1;
reg stable, stable_q;

// Input synchronizer
always @(posedge clk or negedge rst_n) begin
    if (!rst_n) begin
        d0 <= 0;
        d1 <= 0;
    end else begin
        d0 <= din;
        d1 <= d0;
    end
end

// Stability counter
always @(posedge clk or negedge rst_n) begin
    if (!rst_n) begin
        cnt    <= 0;
        stable <= 0;
    end else if (d1 != stable) begin
        if (cnt == STABLE_TICKS) begin
            stable <= d1;
            cnt    <= 0;
        end else begin
            cnt <= cnt + 1;
        end
    end else begin
        cnt <= 0;
    end
end

// One-pulse generator
always @(posedge clk or negedge rst_n) begin
    if (!rst_n) begin
        stable_q <= 0;
```

```
        pulse    <= 0;
    end else begin
        pulse    <= (~stable_q) & stable;
        stable_q <= stable;
    end
end

endmodule
```

load_fsm.v

```
`timescale 1ns/1ps
module load_fsm (
    input wire clk,
    input wire rst_n,
    input wire load_pulse,
    output reg [1:0] slot,
    output wire [3:0] slot_onehot,
    output reg wr_res,
    output reg wr_r,
    output reg wr_g,
    output reg wr_b
);

// One-hot encoding of slot
assign slot_onehot = 4'b0001 << slot;

// Slot counter
always @(posedge clk or negedge rst_n) begin
    if (!rst_n)
        slot <= 2'd0;
    else if (load_pulse)
        slot <= slot + 2'd1; // wraps naturally at 2'd3
end

// Write strobes based on slot
always @* begin
    wr_res = 0;
    wr_r   = 0;
```

```
    wr_g   = 0;
    wr_b   = 0;

    case (slot)
        2'd0: wr_res = load_pulse;
        2'd1: wr_r   = load_pulse;
        2'd2: wr_g   = load_pulse;
        2'd3: wr_b   = load_pulse;
        default: ; // optional, for synthesis completeness
    endcase
end

endmodule
```

pwm_core.v

```
`timescale 1ns/1ps
module pwm_core (
    input wire clk,
    input wire rst_n,
    input wire [7:0] period,
    input wire [7:0] duty_r,
    input wire [7:0] duty_g,
    input wire [7:0] duty_b,
    output reg pwm_r,
    output reg pwm_g,
    output reg pwm_b
);

localparam CNT_WIDTH = 9;
wire [CNT_WIDTH-1:0] eff_period = {1'b0, period} + 9'd1;

// Clamp function to prevent duty > period
function [CNT_WIDTH-1:0] clamp9(input [7:0] d);
    begin
        clamp9 = ({1'b0, d} >= eff_period) ? (eff_period - 1) :
{1'b0, d};
    end
```

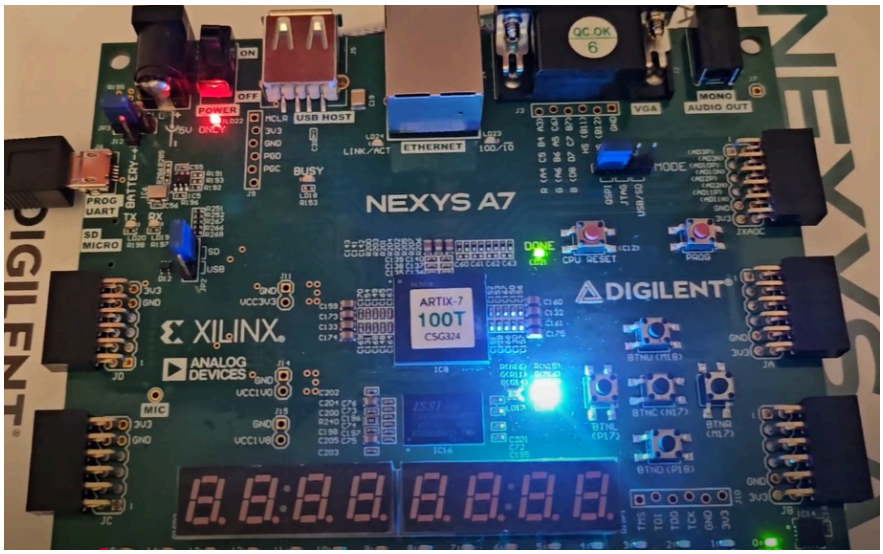
```
endfunction

reg [CNT_WIDTH-1:0] cnt;

// Counter logic
always @(posedge clk or negedge rst_n) begin
    if (!rst_n)
        cnt <= 0;
    else if (cnt == eff_period - 1)
        cnt <= 0;
    else
        cnt <= cnt + 1;
end

// PWM output logic
always @(posedge clk or negedge rst_n) begin
    if (!rst_n) begin
        pwm_r <= 0;
        pwm_g <= 0;
        pwm_b <= 0;
    end else begin
        pwm_r <= (cnt < clamp9(duty_r));
        pwm_g <= (cnt < clamp9(duty_g));
        pwm_b <= (cnt < clamp9(duty_b));
    end
end

endmodule
```

rgb_led_driver.v

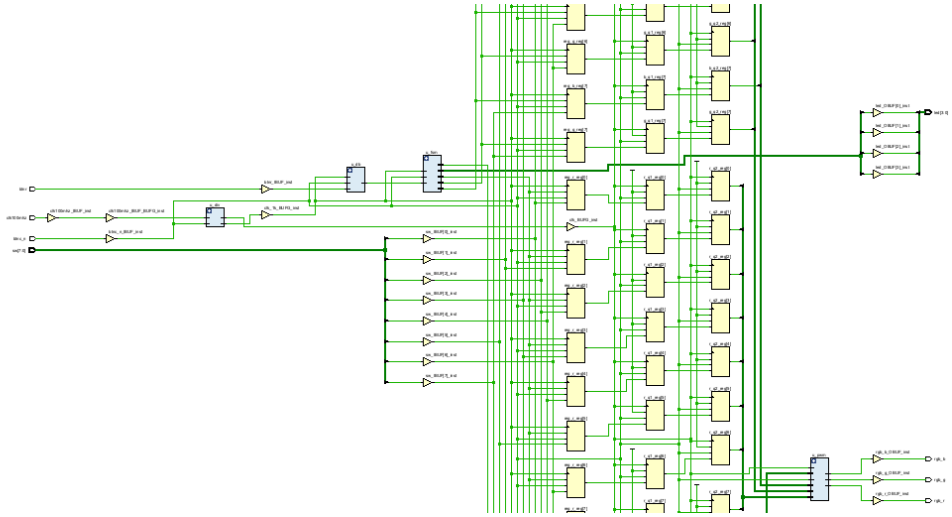
```
`timescale 1ns/1ps
module rgb_led_driver #(
    parameter ACTIVE_LOW = 0
)(
    input wire pwm_r,
    input wire pwm_g,
    input wire pwm_b,
```

```
    output wire led_r,
    output wire led_g,
    output wire led_b
);

// Conditional inversion based on polarity
assign led_r = ACTIVE_LOW ? ~pwm_r : pwm_r;
assign led_g = ACTIVE_LOW ? ~pwm_g : pwm_g;
assign led_b = ACTIVE_LOW ? ~pwm_b : pwm_b;

endmodule
```

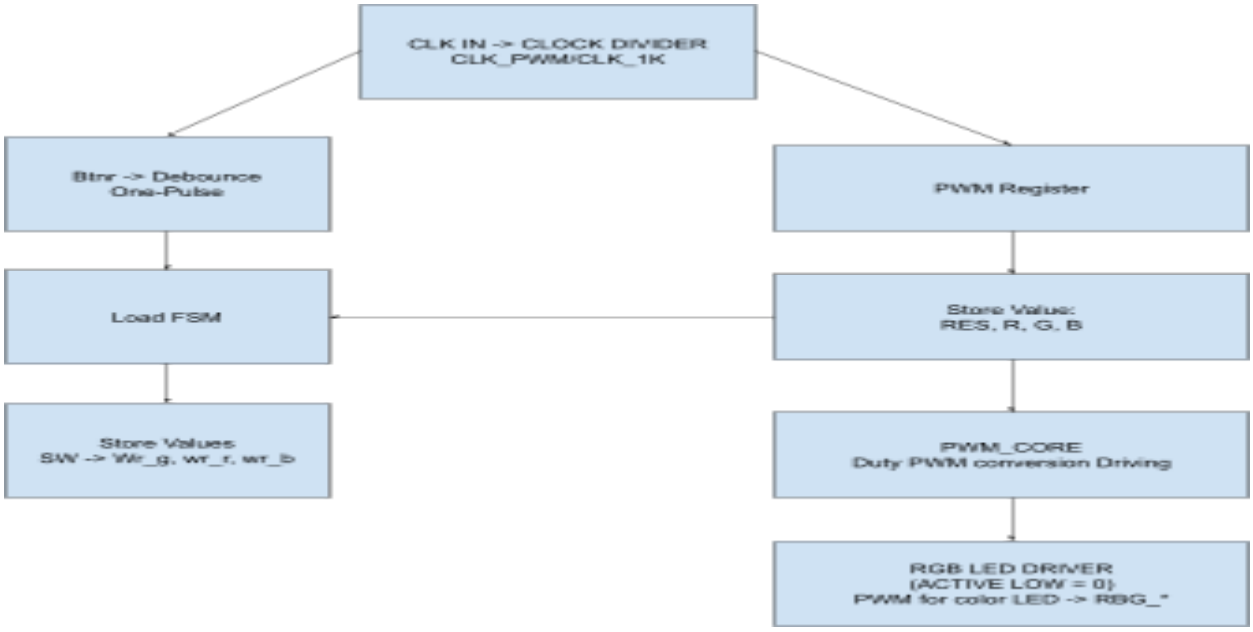
Board:



Hardware Schematic:



Block Diagram:



XDC Snippet:

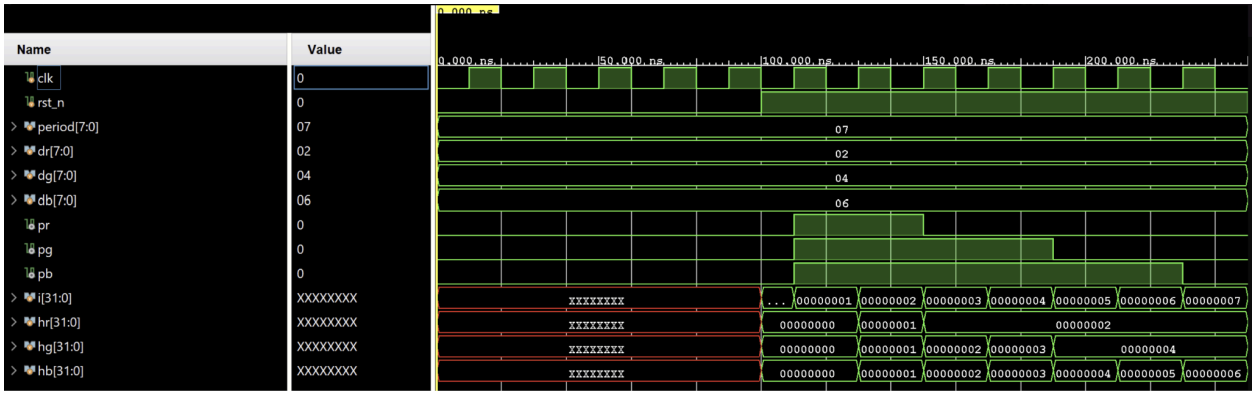
```
## Clock signal
set_property -dict { PACKAGE_PIN E3    IOSTANDARD LVCMOS33 }
[get_ports { clk100mhz }]; #IO_L12P_T1_MRCC_35 Sch=clk100mhz
create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5}
[get_ports {clk100mhz}];


##Switches

set_property -dict { PACKAGE_PIN J15   IOSTANDARD LVCMOS33 }
[get_ports { sw[0] }]; #IO_L24N_T3_RS0_15 Sch=sw[0]
set_property -dict { PACKAGE_PIN L16   IOSTANDARD LVCMOS33 }
[get_ports { sw[1] }]; #IO_L3N_T0_DQS_EMCCLK_14 Sch=sw[1]
set_property -dict { PACKAGE_PIN M13   IOSTANDARD LVCMOS33 }
[get_ports { sw[2] }]; #IO_L6N_T0_D08_VREF_14 Sch=sw[2]
set_property -dict { PACKAGE_PIN R15   IOSTANDARD LVCMOS33 }
[get_ports { sw[3] }]; #IO_L13N_T2_MRCC_14 Sch=sw[3]
set_property -dict { PACKAGE_PIN R17   IOSTANDARD LVCMOS33 }
[get_ports { sw[4] }]; #IO_L12N_T1_MRCC_14 Sch=sw[4]
set_property -dict { PACKAGE_PIN T18   IOSTANDARD LVCMOS33 }
[get_ports { sw[5] }]; #IO_L7N_T1_D10_14 Sch=sw[5]
set_property -dict { PACKAGE_PIN U18   IOSTANDARD LVCMOS33 }
```

```
[get_ports { sw[6] }]; #IO_L17N_T2_A13_D29_14 Sch=sw[6]
set_property -dict { PACKAGE_PIN R13   IOSTANDARD LVCMOS33 }
[get_ports { sw[7] }]; #IO_L5N_T0_D07_14 Sch=sw[7]

## LEDs

set_property -dict { PACKAGE_PIN H17   IOSTANDARD LVCMOS33 }
[get_ports { led[0] }]; #IO_L18P_T2_A24_15 Sch=led[0]
set_property -dict { PACKAGE_PIN K15   IOSTANDARD LVCMOS33 }
[get_ports { led[1] }]; #IO_L24P_T3_RS1_15 Sch=led[1]
set_property -dict { PACKAGE_PIN J13   IOSTANDARD LVCMOS33 }
[get_ports { led[2] }]; #IO_L17N_T2_A25_15 Sch=led[2]
set_property -dict { PACKAGE_PIN N14   IOSTANDARD LVCMOS33 }
[get_ports { led[3] }]; #IO_L8P_T1_D11_14 Sch=led[3]


set_property -dict { PACKAGE_PIN R12   IOSTANDARD LVCMOS33 }
[get_ports { rgb_b }]; #IO_L5P_T0_D06_14 Sch=led16_b
set_property -dict { PACKAGE_PIN M16   IOSTANDARD LVCMOS33 }
[get_ports { rgb_g }]; #IO_L10P_T1_D14_14 Sch=led16_g
set_property -dict { PACKAGE_PIN N15   IOSTANDARD LVCMOS33 }
[get_ports { rgb_r }]; #IO_L11P_T1_SRCC_14 Sch=led16_r

##Buttons

#set_property -dict { PACKAGE_PIN C12   IOSTANDARD LVCMOS33 }
[get_ports { CPU_RESETN }]; #IO_L3P_T0_DQS_AD1P_15 Sch=cpu_resetn

set_property -dict { PACKAGE_PIN N17   IOSTANDARD LVCMOS33 }
[get_ports { btnc_n }]; #IO_L9P_T1_DQS_14 Sch=btnc
#set_property -dict { PACKAGE_PIN M18   IOSTANDARD LVCMOS33 }
[get_ports { BTNU }]; #IO_L4N_T0_D05_14 Sch=btnu
#set_property -dict { PACKAGE_PIN P17   IOSTANDARD LVCMOS33 }
[get_ports { BTNL }]; #IO_L12P_T1_MRCC_14 Sch=btnl
set_property -dict { PACKAGE_PIN M17   IOSTANDARD LVCMOS33 }
[get_ports { btnr }]; #IO_L10N_T1_D15_14 Sch=btnr
```

Simulation:

The testbench code was withheld from the report to avoid distracting from the content. All testbench code is found in the GitHub repository.

pwm_core_tb.v



Implementation: Resource utilization table(s):

Slice Logic Utilization:

LUTs Used: **105 used 0.17% Utilization**

Registers Used: 150 used 0.12% Utilization

Muxes:

F7: 0 used

F8: 0 used

Registers:

Registers used as Flip-flops

IO: 18 IO used , 8.57% Utilization

Timing:

Worst Negative Slack: 6.640 ns, 30 endpoints

Worst Hold Slack: 0.140 ns, 30 endpoints

Worst Pulse Width Slack: 4.500 ns, 31 endpoints

Timing constraints are met, providing adequate slack.

Group video link:

https://youtu.be/p4dp1OhqBbs?si=4OprMyaDio6E4T-M

Contributions:

The contributions of this lab were equal across the board. Most of the heavy development of this lab was already handled and given by the project, so collaborative strategies were used to understand and implement the existing code. The demo was recorded by Jetts Crittenden and the testing was executed by Evan Tram. The analysis and writing the paper were conducted by both members. Overall, this lab marked a good conclusion to the lab unit and was able to gave both members good information regarding PWM systems in Verilog.