Julio Flores (ID# : 016326856)
Victor Perez (ID# : 016196050)

Group I

Session E02

Lab 8

RGB LED PWM Controller
Thursday

August 14, 2025

ECE 3300L

Summer 2025

Objective:

The objective of this lab is to design, implement, and test a RGB LED PWM Controller using VHDL (or Verilog) on the Nexys A7 FPGA development board. The circuit focuses on controlling brightness and blending colors through precise digital logic techniques such as PWM manipulation.

Design(Code):

Clock_Divider_Fixed.V

- Uses a clock to set the speed to a readable speed for 7seg displays and shifter

```verilog
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;
    reg [$clog2(DIV1H):0] c1;
    reg [$clog2(DIVPMH):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
            if (c1 == DIV1H-1) begin c1 <= 0; clk_1k <= ~clk_1k; end else c1 <=c1+1;
            if (c2 == DIVPMH-1) begin c2 <= 0; clk_pwm <= ~clk_pwm; end else c2 <= c2+1;
        end
    end
endmodule
```

debounce_onepulse.v

- Prevents glitches from affecting the buttons to make one clear button press.

```
module debounce_onepulse #(
    parameter integer STABLE_TICKS = 20
)(
    input wire clk,
    input wire rst_n,
    input wire din,
    output reg pulse
);
    reg d0, d1;
    reg stable, stable_q;
    reg [$clog2(STABLE_TICKS+1)-1:0] cnt;

    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

    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 cnt<=cnt+1;
        end else cnt<=0;
    end

    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
```

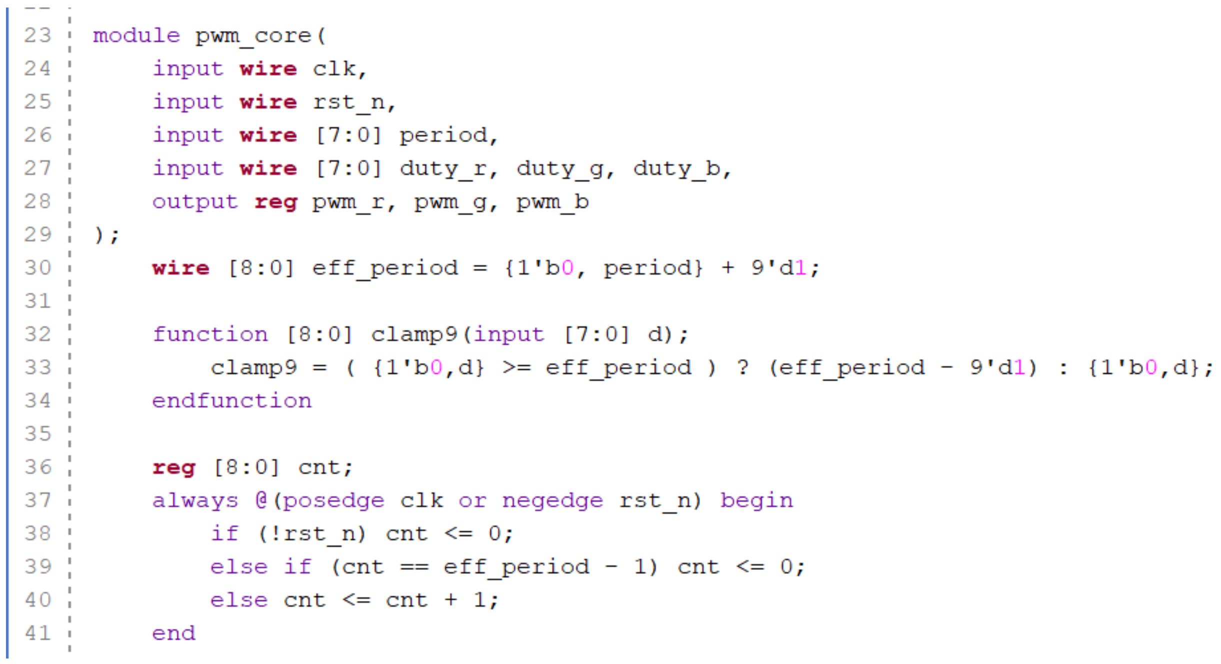
load_fsm.V

- In this FSM there is only 1 button however 4 slots to fill which it cycles through each time a load pulse signal is received. Using RES+1 we can fill the first slot with RES and then continue down the slots filling them with R, G, and B. This guarantees that each color component is updated and controlled in sequential manner without overlap, which is important when timing updates to an FPGA-driven PWM system.

```verilog
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, wr_r, wr_g, wr_b
);
    assign slot_onehot = 4'b0001 << slot;

    always @(posedge clk or negedge rst_n) begin
        if (!rst_n) slot <= 2'd0;
        else if (load_pulse) slot <= slot + 2'd1;
    end

    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;
        endcase
    end
endmodule
```

pwm_core.V

- Converts the stored values into duty cycles for 3 PWM outputs.



```verilog
module pwm_core(
    input wire clk,
    input wire rst_n,
    input wire [7:0] period,
    input wire [7:0] duty_r, duty_g, duty_b,
    output reg pwm_r, pwm_g, pwm_b
);
    wire [8:0] eff_period = {1'b0, period} + 9'd1;

    function [8:0] clamp9(input [7:0] d);
        clamp9 = ( {1'b0,d} >= eff_period ) ? (eff_period - 9'd1) : {1'b0,d};
    endfunction

    reg [8:0] cnt;
    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
```

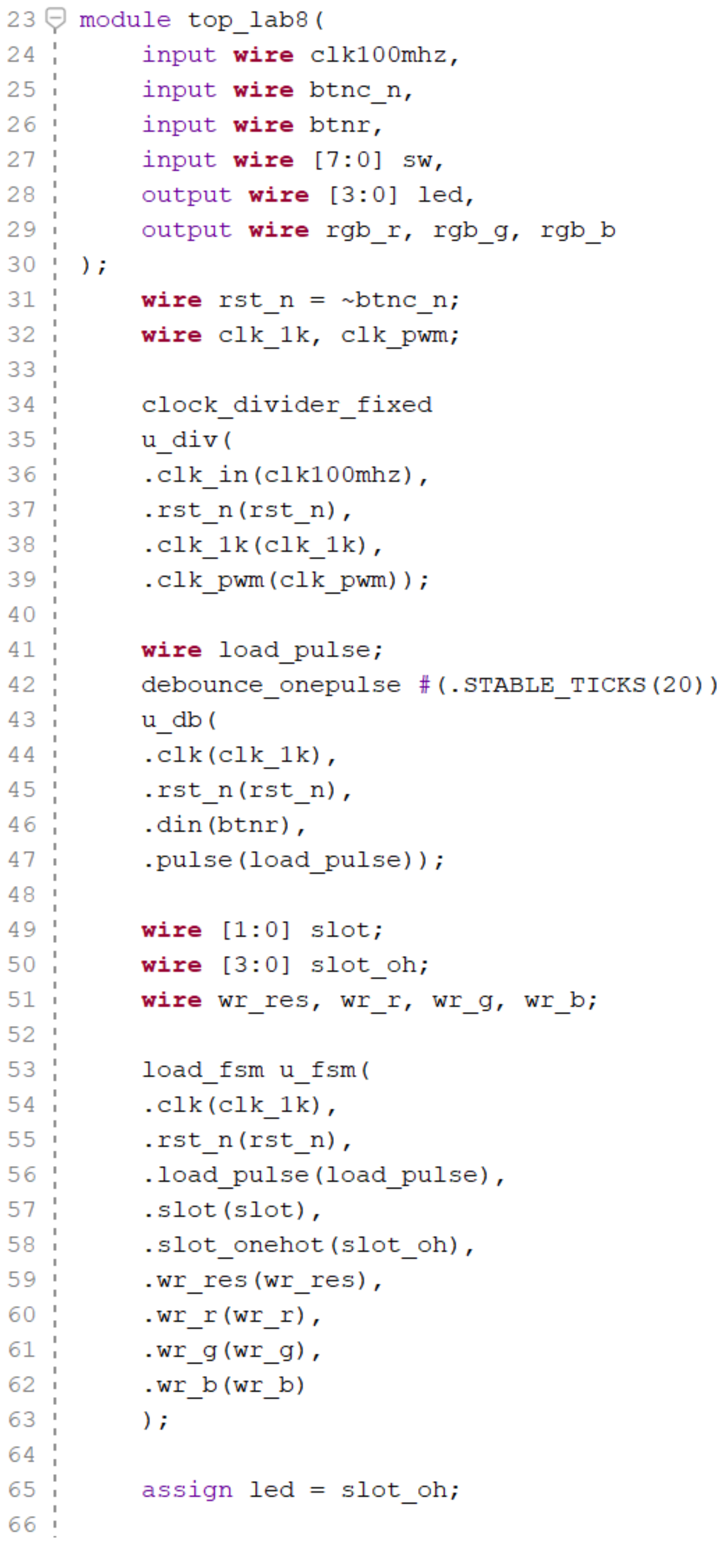
rgb_led_driver.V

- This is the driver for the rgb led driver to make the colors when the signal is active or high depending on the parameter.

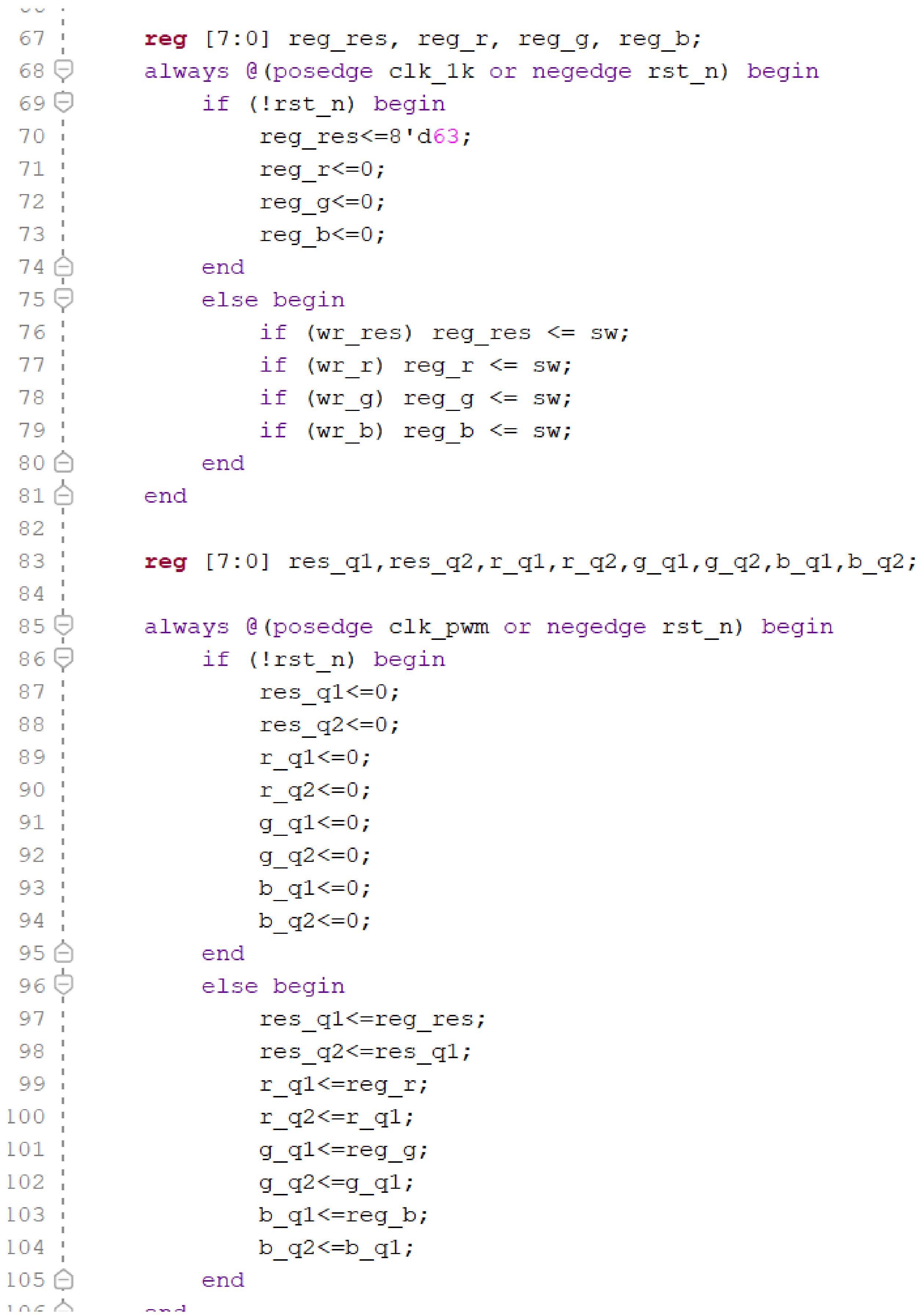
```verilog
module rgb_led_driver#(parameter ACTIVE_LOW=1)(
    input wire pwm_r, pwm_g, pwm_b,
    output wire led_r, led_g, led_b
);
    generate
        if (ACTIVE_LOW) begin
            assign led_r = ~pwm_r;
            assign led_g = ~pwm_g;
            assign led_b = ~pwm_b;
        end else begin
            assign led_r = pwm_r;
            assign led_g = pwm_g;
            assign led_b = pwm_b;
        end
    endgenerate
endmodule
```

top_lab8.V

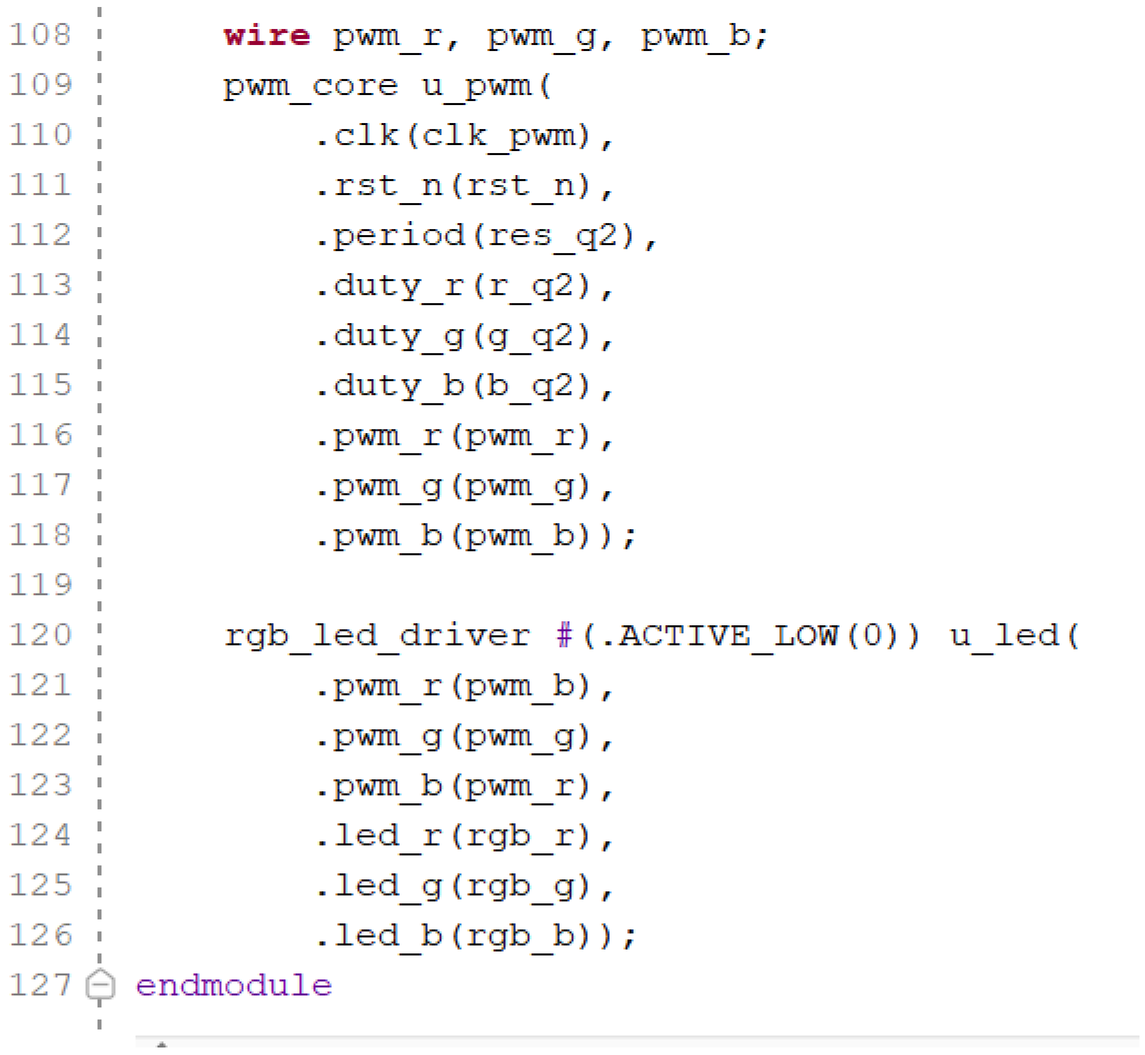
- Calls all modules to the top module for the project to run.



```verilog
module top_lab8(
    input wire clk100mhz,
    input wire btnc_n,
    input wire btnr,
    input wire [7:0] sw,
    output wire [3:0] led,
    output wire rgb_r, rgb_g, rgb_b
);
    wire rst_n = ~btnc_n;
    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));

    wire load_pulse;
    debounce_onepulse #(.STABLE_TICKS(20))
    u_db(
    .clk(clk_1k),
    .rst_n(rst_n),
    .din(btnr),
    .pulse(load_pulse));

    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)
    );

    assign led = slot_oh;

```



```verilog
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<=8'd63;
        reg_r<=0;
        reg_g<=0;
        reg_b<=0;
    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

reg [7:0] res_q1,res_q2,r_q1,r_q2,g_q1,g_q2,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
```

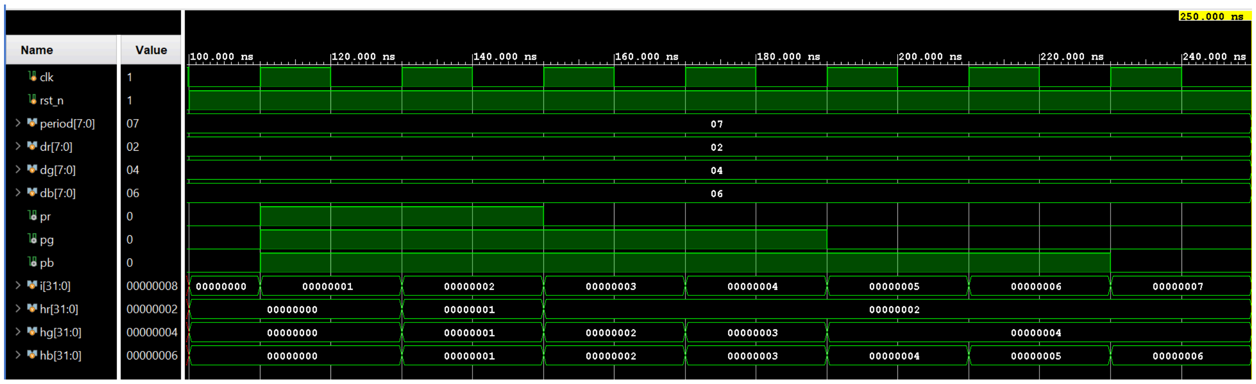


```verilog
    wire pwm_r, pwm_g, pwm_b;
    pwm_core u_pwm(
        .clk(clk_pwm),
        .rst_n(rst_n),
        .period(res_q2),
        .duty_r(r_q2),
        .duty_g(g_q2),
        .duty_b(b_q2),
        .pwm_r(pwm_r),
        .pwm_g(pwm_g),
        .pwm_b(pwm_b));

    rgb_led_driver #(.ACTIVE_LOW(0)) u_led(
        .pwm_r(pwm_b),
        .pwm_g(pwm_g),
        .pwm_b(pwm_r),
        .led_r(rgb_r),
        .led_g(rgb_g),
        .led_b(rgb_b));
endmodule
```

Testbench and Waveform:

pwm_core_tb.V

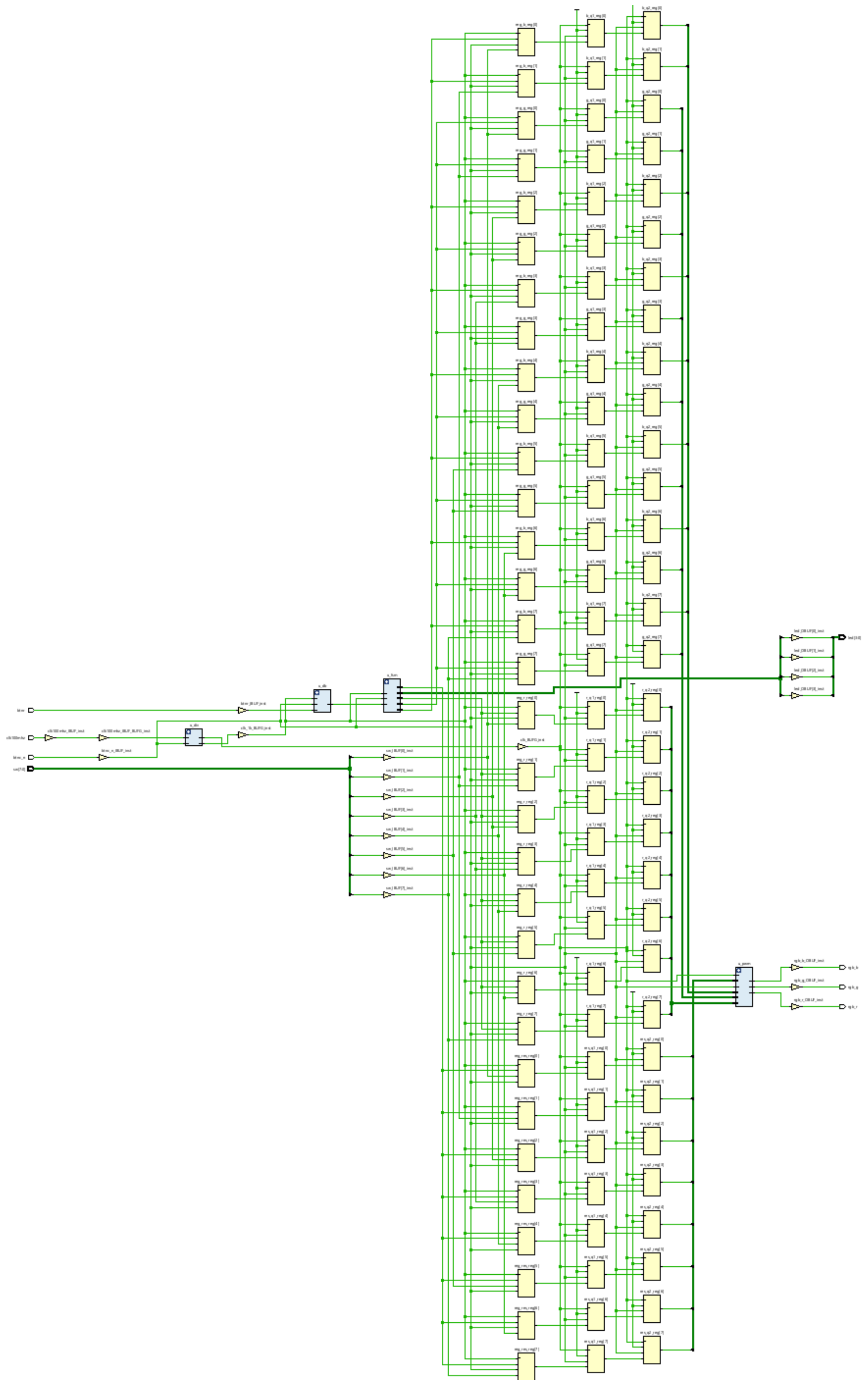
- Shifting the bit values



Implementation:

```
+-------------------------+------+-------+------------+-----------+-------+
|        Site Type        | Used | Fixed | Prohibited | Available | Util% |
+-------------------------+------+-------+------------+-----------+-------+
| Slice LUTs*             |  113 |     0 |          0 |     63400 |  0.18 |
|   LUT as Logic          |  113 |     0 |          0 |     63400 |  0.18 |
|   LUT as Memory         |    0 |     0 |          0 |     19000 |  0.00 |
| Slice Registers         |  152 |     0 |          0 |    126800 |  0.12 |
|   Register as Flip Flop |  152 |     0 |          0 |    126800 |  0.12 |
|   Register as Latch     |    0 |     0 |          0 |    126800 |  0.00 |
| F7 Muxes                |    0 |     0 |          0 |     31700 |  0.00 |
| F8 Muxes                |    0 |     0 |          0 |     15850 |  0.00 |
+-------------------------+------+-------+------------+-----------+-------+
```

Block diagram:



Contributions:

Julio Flores: 50% - Verilog Code (clock_divider, debounce, load_fsm and testbench for these modules) and report and demo

Victor Perez : 50% - Verilog Code (pmw_driver, rgb_led, top_module code and testbench for these modules) and report and demo

Reflection:

In this lab we design a hardware-controlled RGB LED system using Pulse Width Modulation (PWM) on the Nexys A7 FPGA board. The PWM logic was created by comparing a counter against duty cycle values for each color channel. Different duty cycles resulted in varying LED brightness, allowing the generation of multiple colors through additive mixing. Successful testing confirmed that adjusting the duty cycles in real time produced smooth transitions between colors. The implementation demonstrated the practical application of digital design principles, clock division, and modular coding for hardware control.