# ECE3300L Lab 8

Group B

By Faris Khan (ID #: 012621102)

AND

Nicholas Williams (ID#:016556982)

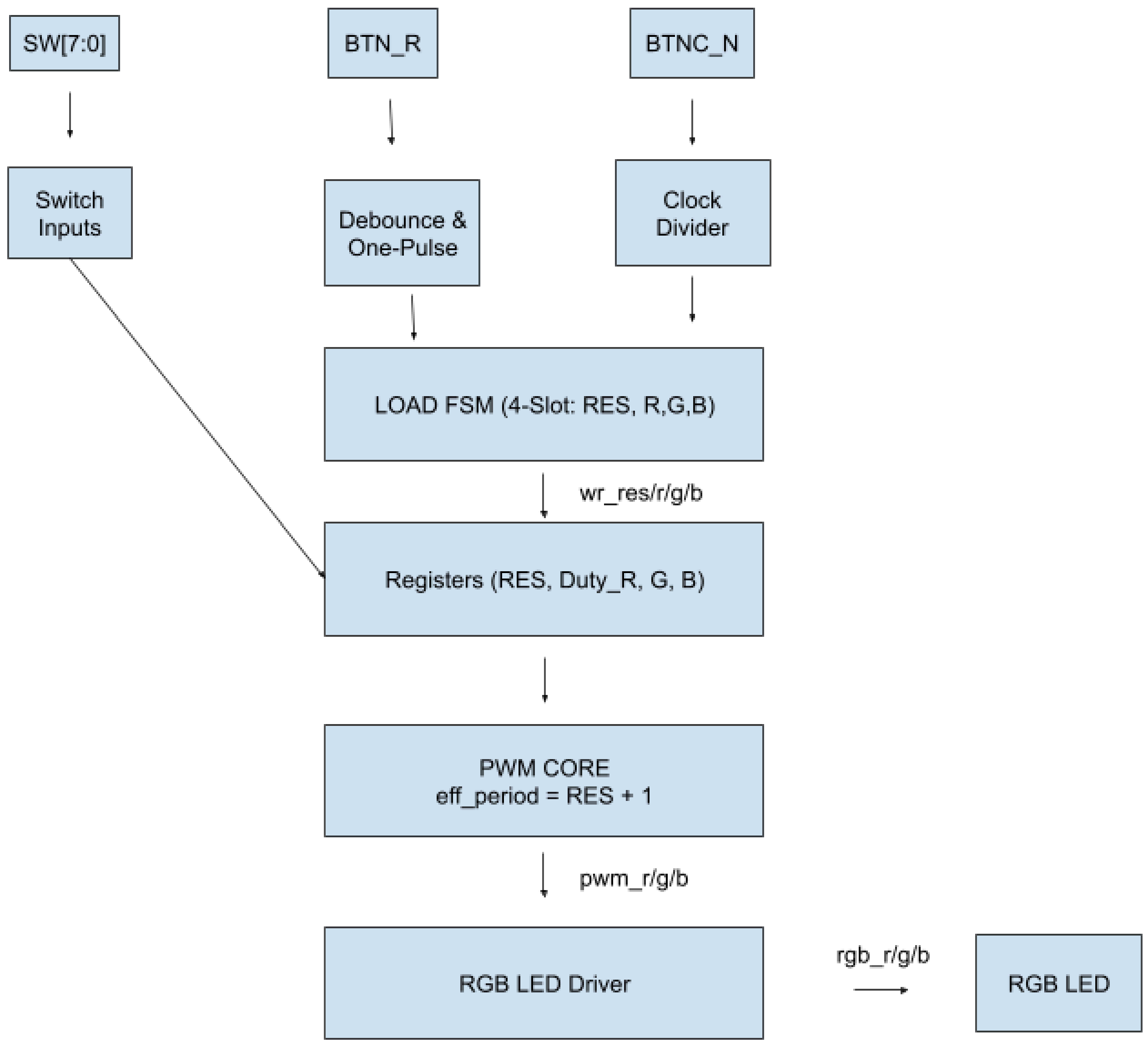
14 August 2025

## Introduction

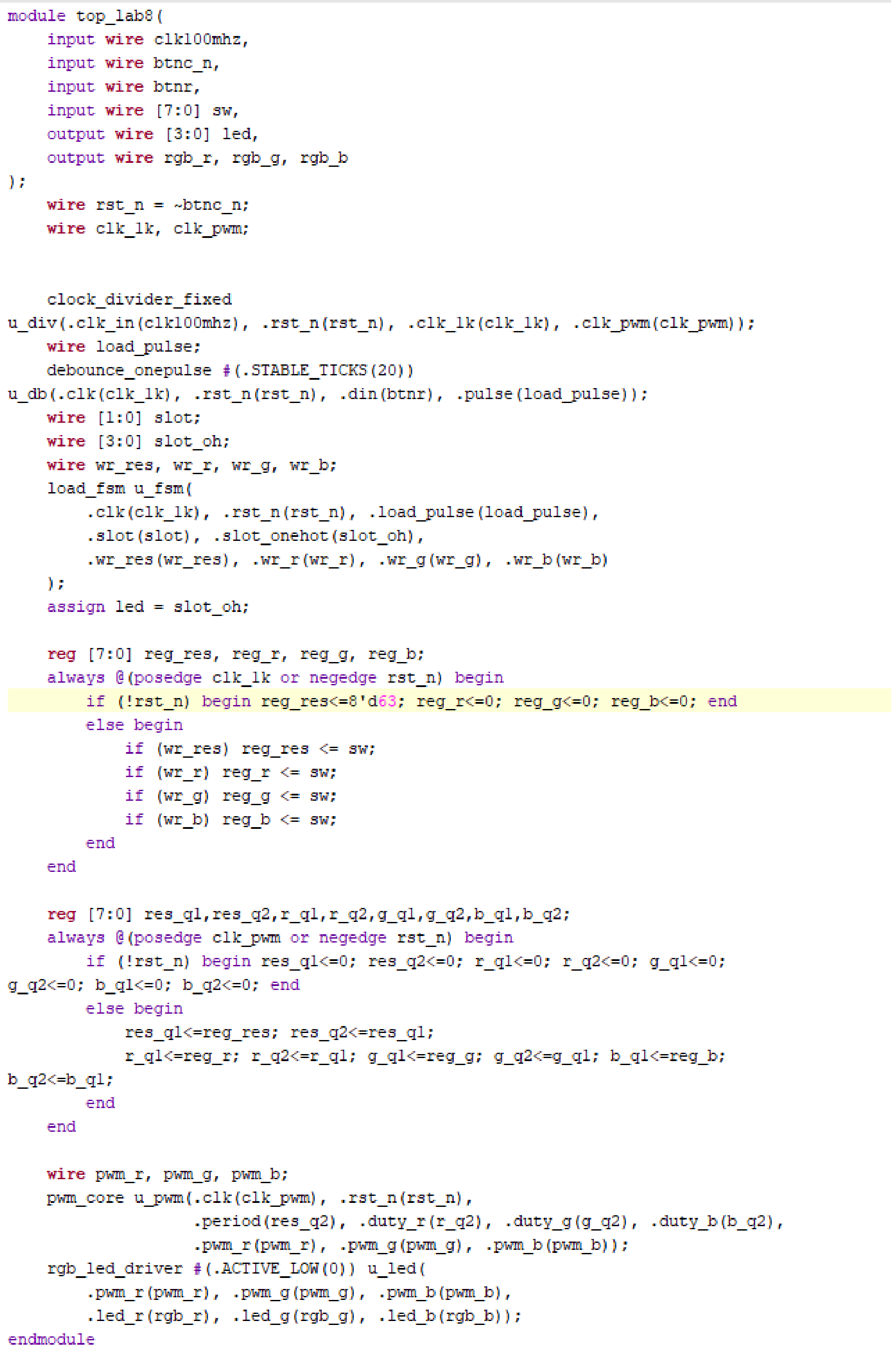
In this lab we were to make a RGB LED PWM controller on our Nexys A7 board. We had to do a total of six Micro-Labs that were used as progress marks to make sure that the code was working. The Micro-Labs lettered A-F. The main focus however was to use PWM which is Pulse Width Modulation to control the brightness of the LED’s by rapidly toggling the output between high and low states at a specific frequency and the Perceived brightness is determined by the duty cycle.

In this Lab we generated precise PWM signals for the red, green, and blue channels of the onboard RGB LED and allowed dynamic adjustments through the Nexys A7 board. We used both a slow and fast clock signal of 1khz and 20khz.

## Block Diagram

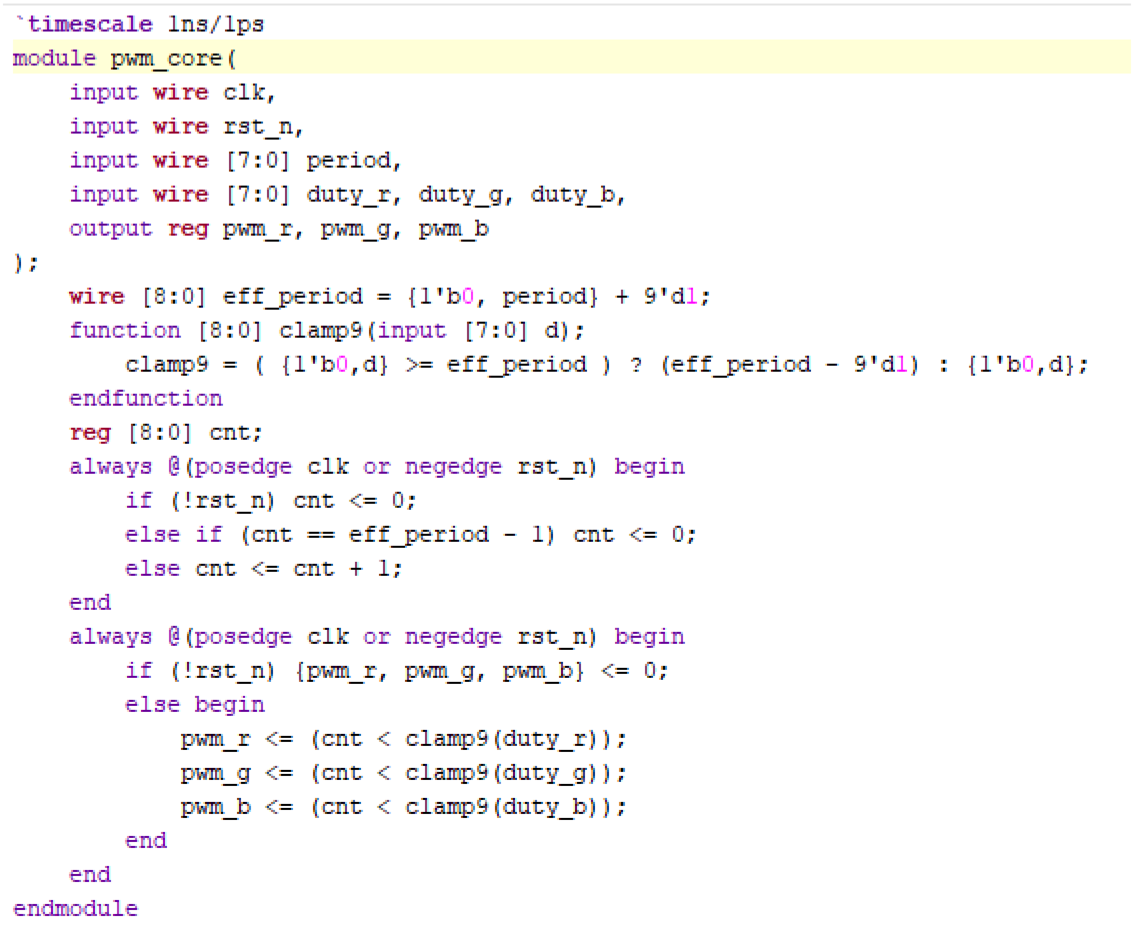


# Code

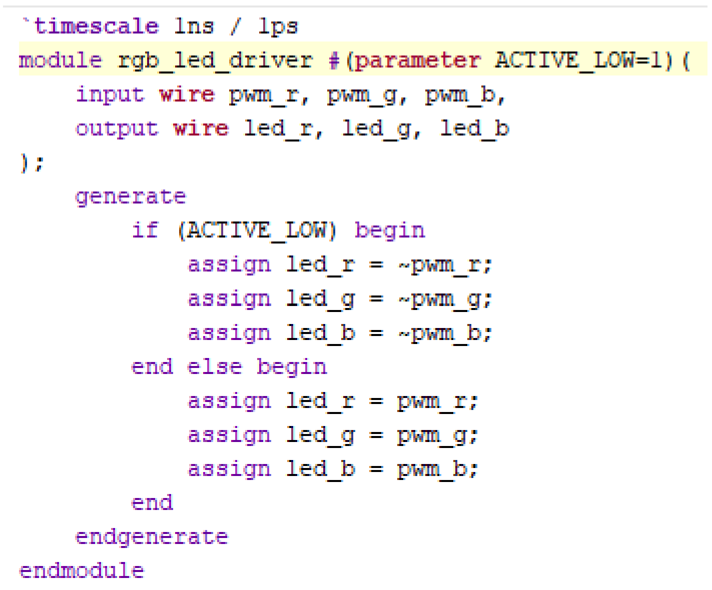


```verilog
module top_lab8(
    input wire clk100mhz,
    input wire btnc_n,
    input wire btnr,
    input wire [7:0] sw,
    output wire [3:0] led,
    output wire rgb_r, rgb_g, rgb_b
);
    wire rst_n = ~btnc_n;
    wire clk_1k, clk_pwm;


    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;

    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

    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_r), .pwm_g(pwm_g), .pwm_b(pwm_b),
        .led_r(rgb_r), .led_g(rgb_g), .led_b(rgb_b));
endmodule
```

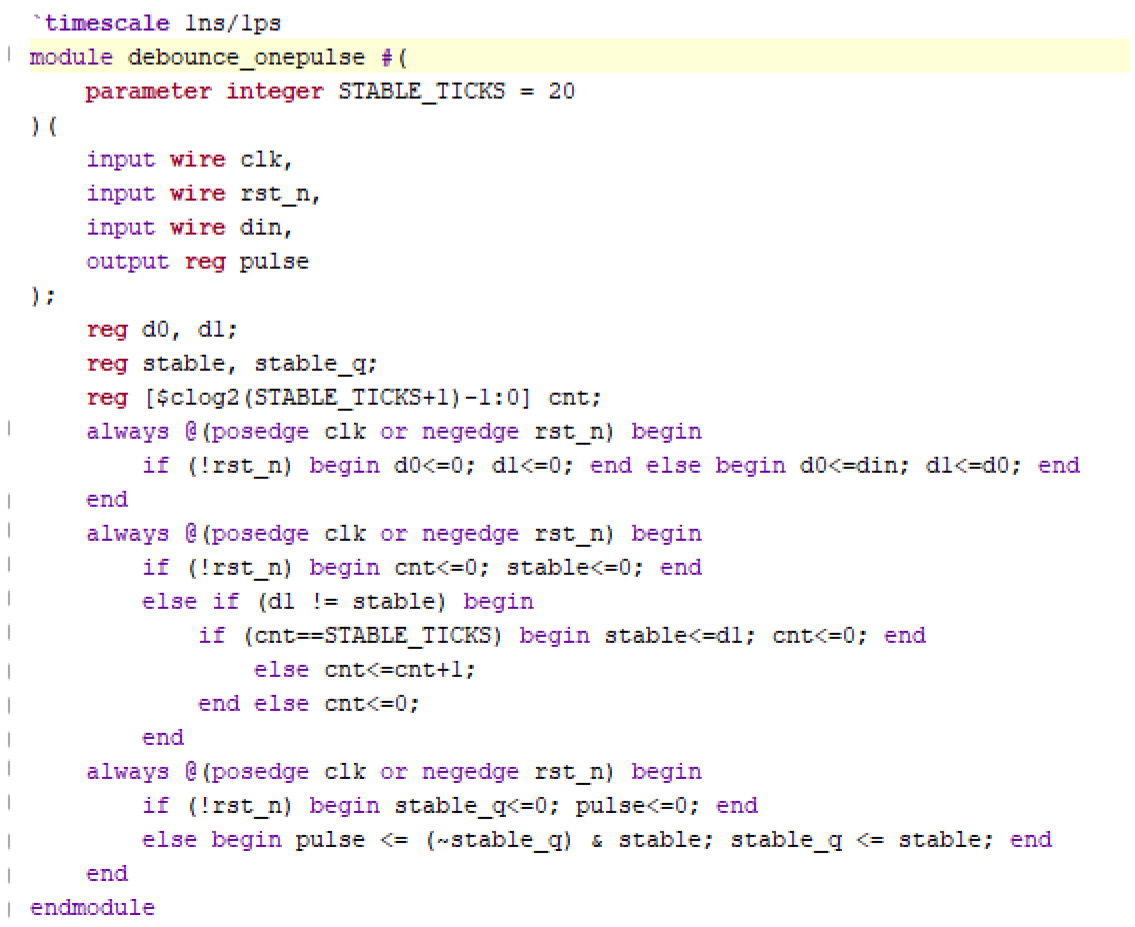
```verilog
`timescale 1ns/1ps
module clock_divider_fixed #(
    parameter integer INPUT_HZ = 100_000_000,
    parameter integer TICK1_HZ = 1_000,
    parameter integer PWM_HZ = 20_000
    )(
    input wire clk_in,
    input wire rst_n,
    output reg clk_1k,
    output reg clk_pwm

);

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

`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, 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
```



```
`timescale 1ns/1ps
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
    always @(posedge clk or negedge rst_n) begin
        if (!rst_n) {pwm_r, pwm_g, pwm_b} <= 0;
        else begin
            pwm_r <= (cnt < clamp9(duty_r));
            pwm_g <= (cnt < clamp9(duty_g));
            pwm_b <= (cnt < clamp9(duty_b));
        end
    end
endmodule
```

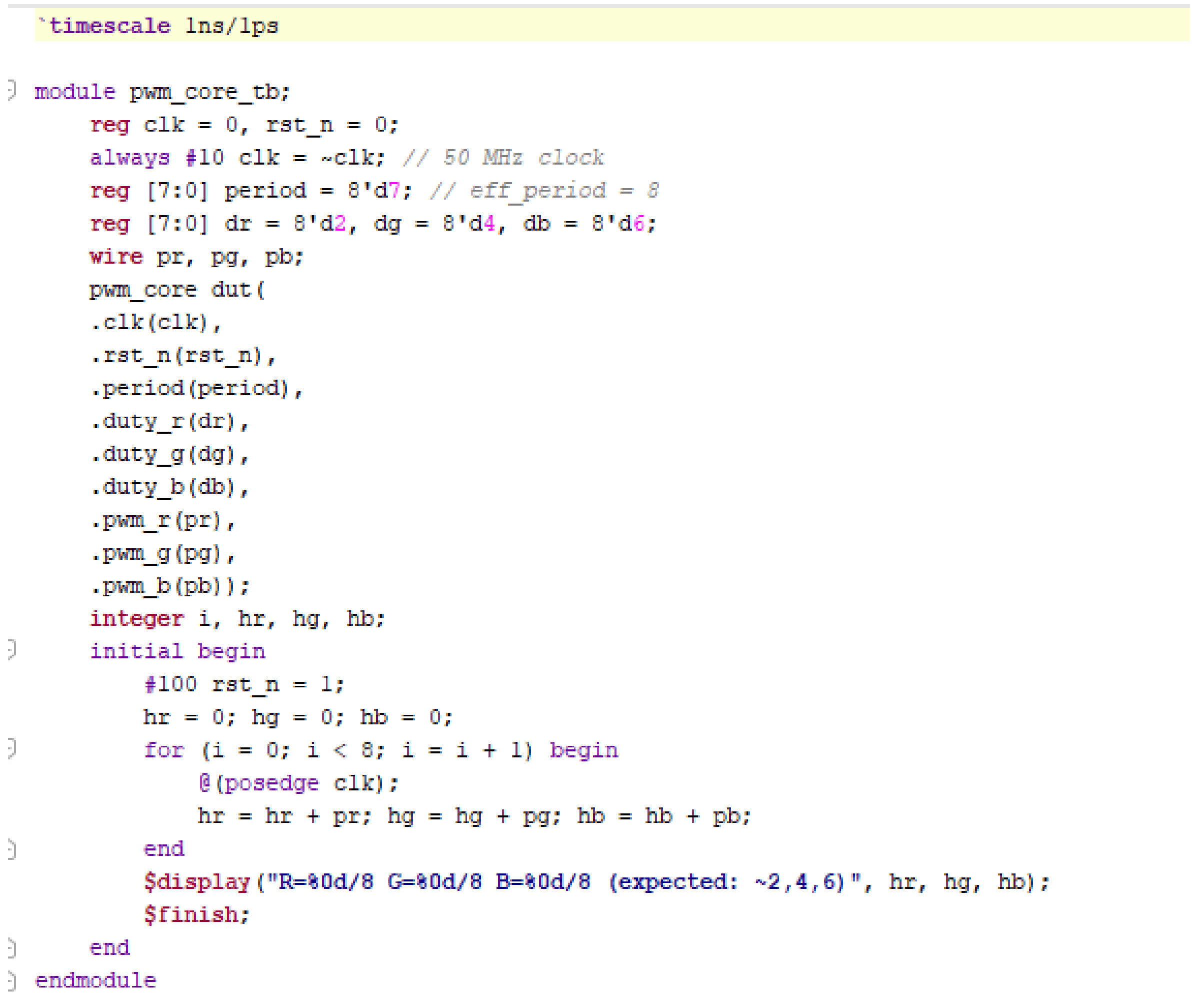


```
`timescale 1ns / 1ps
module rgb_led_driver #(parameter ACTIVE_LOW=1)(
    input wire pwm_r, pwm_g, pwm_b,
    output wire led_r, led_g, led_b
);
    generate
        if (ACTIVE_LOW) begin
            assign led_r = ~pwm_r;
            assign led_g = ~pwm_g;
            assign led_b = ~pwm_b;
   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



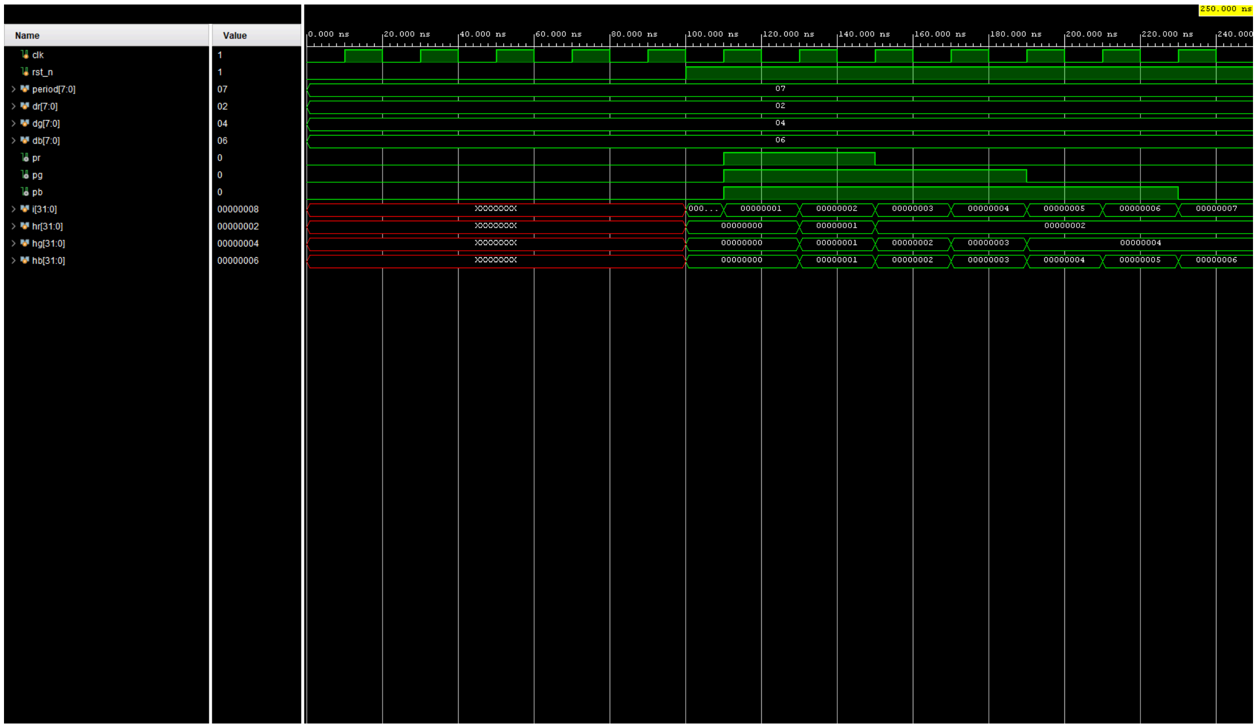
```verilog
`timescale 1ns/1ps
module debounce_onepulse #(
    parameter integer STABLE_TICKS = 20
)(
    input wire clk,
    input wire rst_n,
    input wire din,
    output reg pulse
);
    reg d0, d1;
    reg stable, stable_q;
    reg [$clog2(STABLE_TICKS+1)-1:0] cnt;
    always @(posedge clk or negedge 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
```

## Testbench



```
`timescale 1ns/1ps

module pwm_core_tb;
    reg clk = 0, rst_n = 0;
    always #10 clk = ~clk; // 50 MHz clock
    reg [7:0] period = 8'd7; // eff_period = 8
    reg [7:0] dr = 8'd2, dg = 8'd4, db = 8'd6;
    wire pr, pg, pb;
    pwm_core dut(
    .clk(clk),
    .rst_n(rst_n),
    .period(period),
    .duty_r(dr),
    .duty_g(dg),
    .duty_b(db),
    .pwm_r(pr),
    .pwm_g(pg),
    .pwm_b(pb));
    integer i, hr, hg, hb;
    initial begin
        #100 rst_n = 1;
        hr = 0; hg = 0; hb = 0;
        for (i = 0; i < 8; i = i + 1) begin
            @(posedge clk);
            hr = hr + pr; hg = hg + pg; hb = hb + pb;
        end
        $display("R=%0d/8 G=%0d/8 B=%0d/8 (expected: ~2,4,6)", hr, hg, hb);
        $finish;
    end
endmodule
```

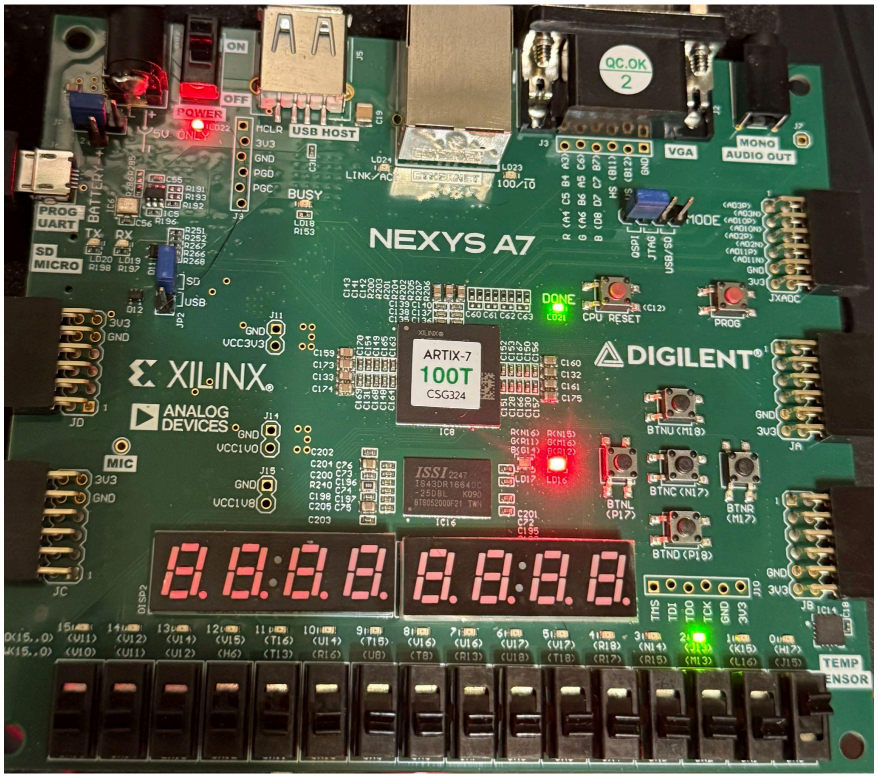
R=2/8 G=4/8 B=6/8 (expected: ~2,4,6)

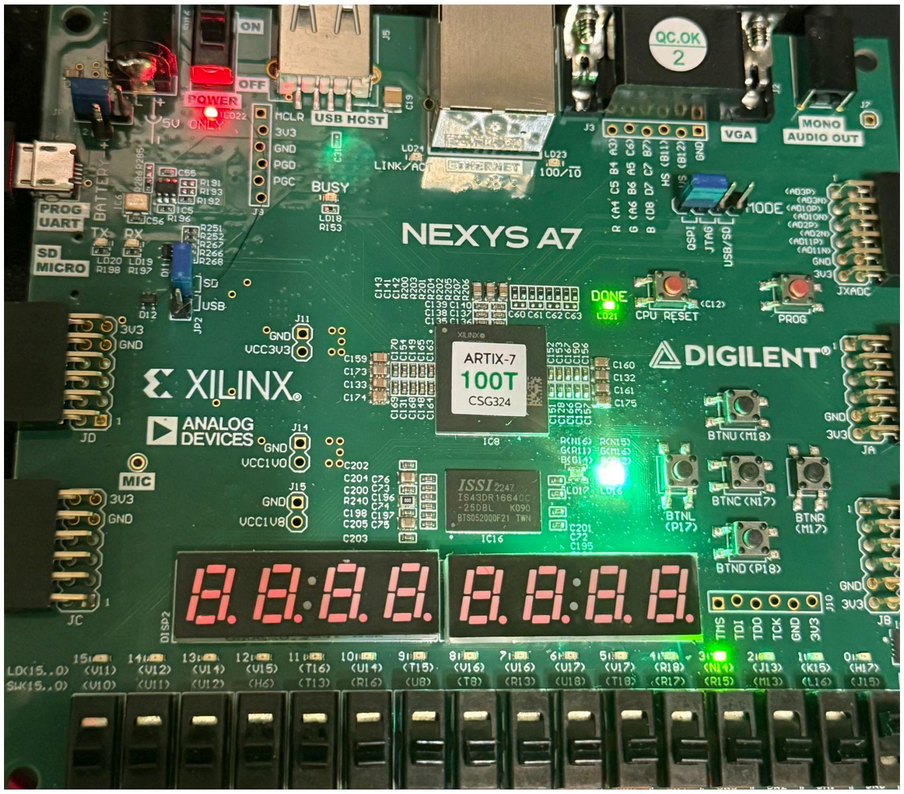


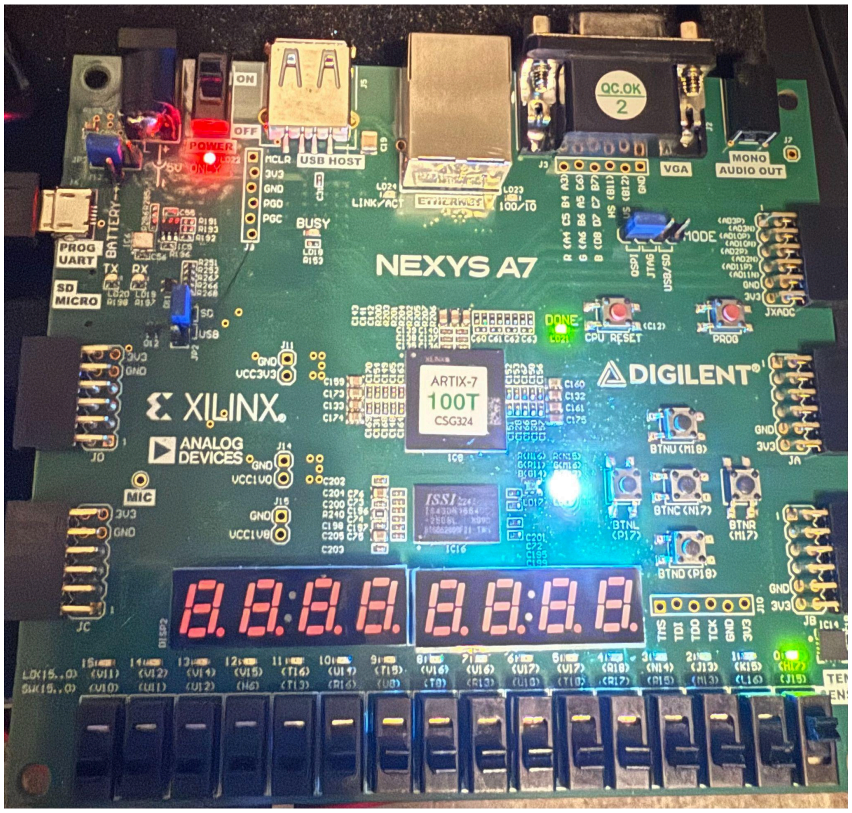
```
Name          Value
clk           1
rst_n         1
period[7:0]   07
dr[7:0]       02
dg[7:0]       04
db[7:0]       06
pr            0
pg            0
pb            0
i[31:0]       00000008
hr[31:0]      00000002
hg[31:0]      00000004
hb[31:0]      00000006
```

## Hardware Results

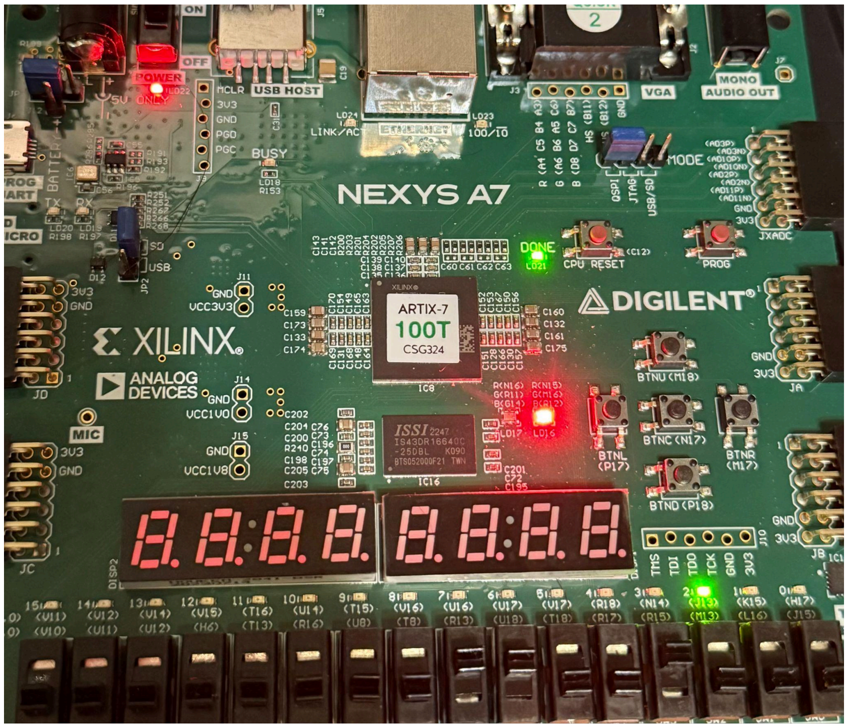
Low RES:

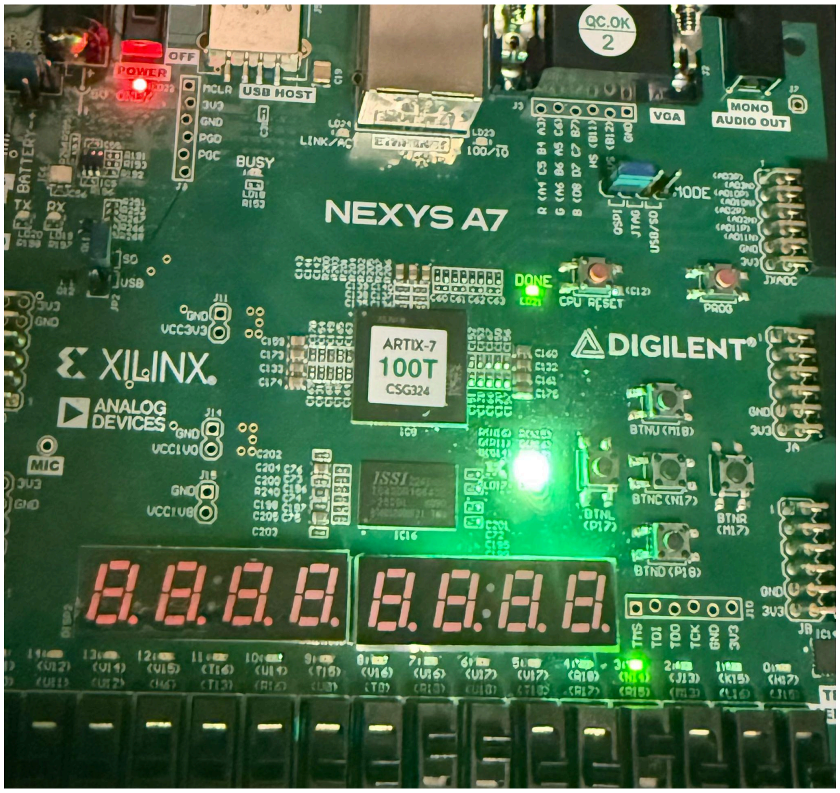


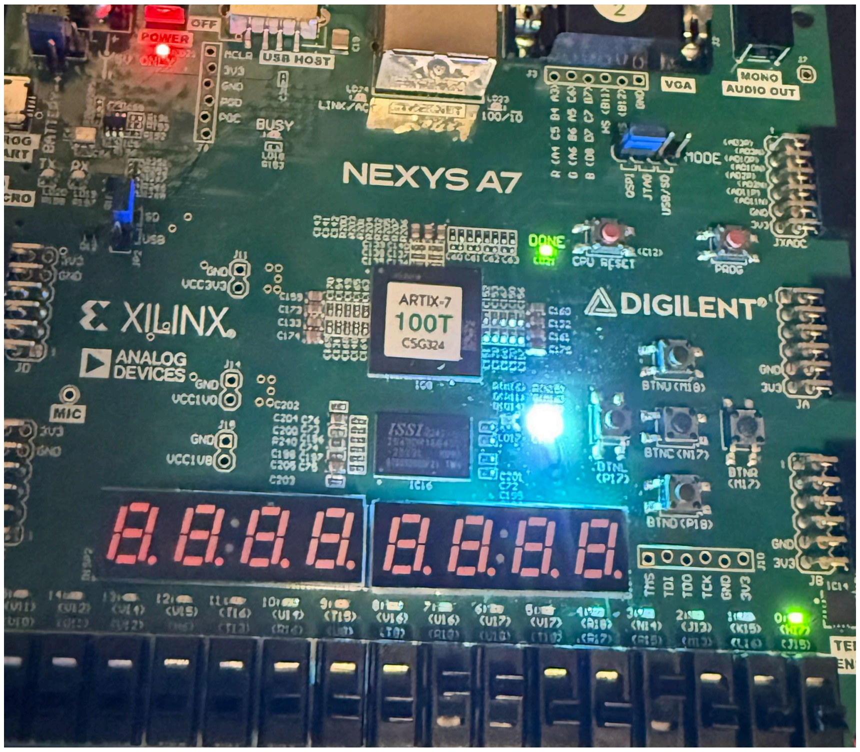



NEXYS A7
DIGILENT
XILINX
ANALOG DEVICES
ARTIX-7
100T
CSG324
POWER
USB HOST
VGA
MONO AUDIO OUT
CPU RESET
PROG
DONE

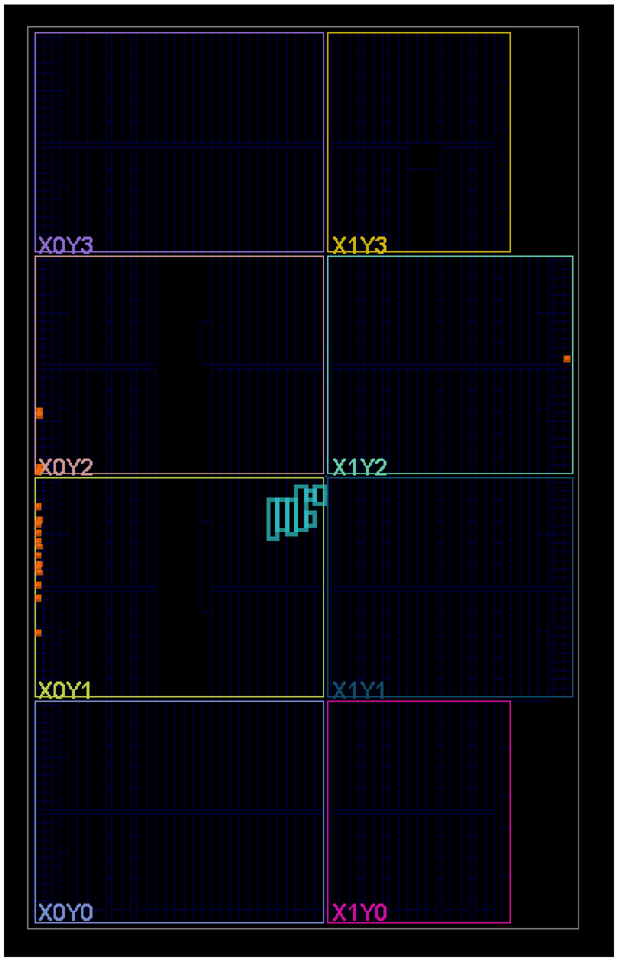
High RES:

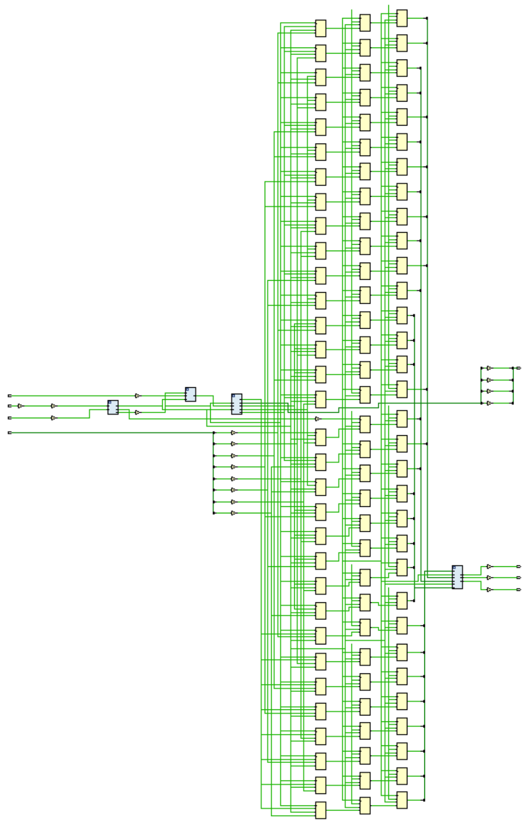

NEXYS A7
DIGILENT
XILINX
ANALOG DEVICES
ARTIX-7
100T
CSG324
POWER
USB HOST
VGA
MONO AUDIO OUT
CPU RESET
PROG
DONE

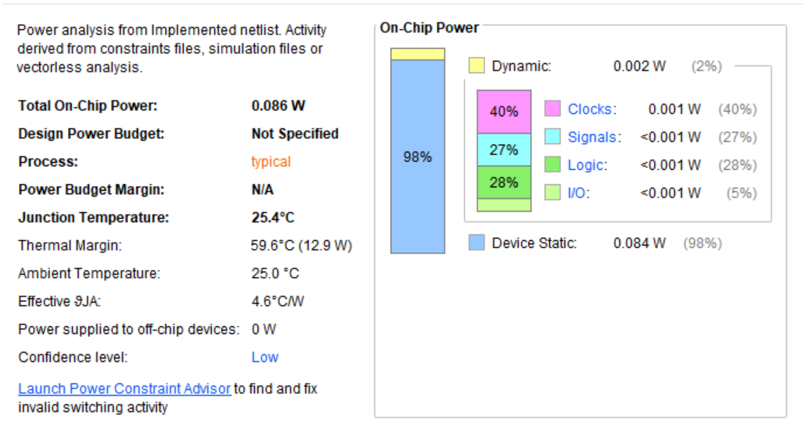

NEXYS A7
XILINX
ANALOG DEVICES
DIGILENT
ARTIX-7
100T
CSG324
DONE
CPU RESET
PROG
BTNU (M18)
BTNC (N17)
BTNL (P17)
BTNR (M17)
BTND (P18)
POWER
USB HOST
VGA
MONO AUDIO OUT
MODE
BUSY
MIC


NEXYS A7
XILINX
ANALOG DEVICES
DIGILENT
ARTIX-7
100T
CSG324
DONE
CPU RESET
PROG
BTNU (M18)
BTNC (N17)
BTNL (P17)
BTNR (M17)
BTND (P18)
POWER
USB HOST
VGA
MONO AUDIO OUT
MODE
BUSY
MIC

## Screenshots





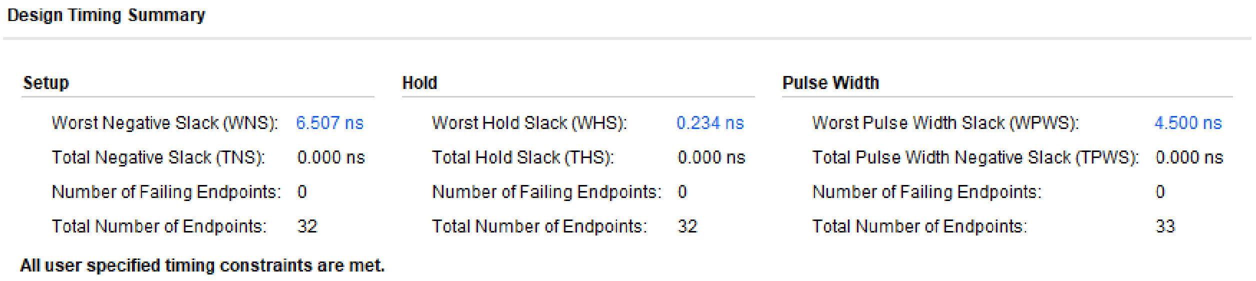


Power analysis from Implemented netlist. Activity derived from constraints files, simulation files or vectorless analysis.

| | |
|---|---|
| **Total On-Chip Power:** | **0.086 W** |
| **Design Power Budget:** | **Not Specified** |
| **Process:** | typical |
| **Power Budget Margin:** | **N/A** |
| **Junction Temperature:** | **25.4°C** |
| Thermal Margin: | 59.6°C (12.9 W) |
| Ambient Temperature: | 25.0 °C |
| Effective ϑJA: | 4.6°C/W |
| Power supplied to off-chip devices: | 0 W |
| Confidence level: | Low |

Launch Power Constraint Advisor to find and fix invalid switching activity

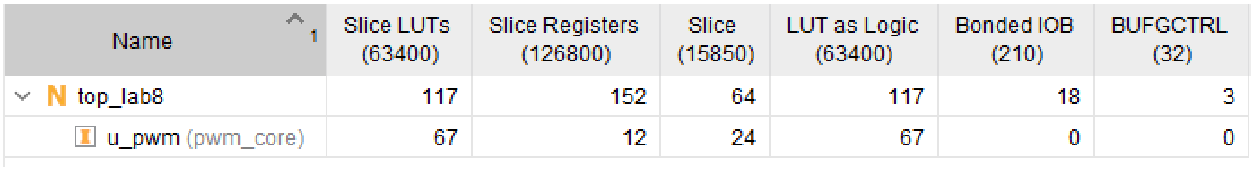
| | | | |
|---|---|---|---|
| 🟨 | Dynamic: | 0.002 W | (2%) |
| 🟪 | Clocks: | 0.001 W | (40%) |
| 🟦 | Signals: | <0.001 W | (27%) |
| 🟩 | Logic: | <0.001 W | (28%) |
| 🟩 | I/O: | <0.001 W | (5%) |
| 🟦 | Device Static: | 0.084 W | (98%) |



**Design Timing Summary**

| **Setup** | | **Hold** | | **Pulse Width** | |
|---|---|---|---|---|---|
| Worst Negative Slack (WNS): | 6.507 ns | Worst Hold Slack (WHS): | 0.234 ns | Worst Pulse Width Slack (WPWS): | 4.500 ns |
| Total Negative Slack (TNS): | 0.000 ns | Total Hold Slack (THS): | 0.000 ns | Total Pulse Width Negative Slack (TPWS): | 0.000 ns |
| Number of Failing Endpoints: | 0 | Number of Failing Endpoints: | 0 | Number of Failing Endpoints: | 0 |
| Total Number of Endpoints: | 32 | Total Number of Endpoints: | 32 | Total Number of Endpoints: | 33 |

**All user specified timing constraints are met.**



| Name | Slice LUTs (63400) | Slice Registers (126800) | Slice (15850) | LUT as Logic (63400) | Bonded IOB (210) | BUFGCTRL (32) |
|---|---|---|---|---|---|---|
| top_lab8 | 117 | 152 | 64 | 117 | 18 | 3 |
| u_pwm (pwm_core) | 67 | 12 | 24 | 67 | 0 | 0 |

The lab design uses a 4-slot load FSM to configure the PWM resolution and duty cycles for each RGB channel using only one pushbutton. Each press of the debounced load button writes the current switch value into the register for the active slot: RES (PWM period), Red duty, Green duty, or Blue duty. It then automatically advances to the next slot in a sequence. The PWM core calculates the effective period as eff_period = period + 1 to account for counting from zero, so a stored value of 7 produces a resolution of 8 ticks per cycle, while 199 produces a resolution of 200. This allows demonstration of duty-cycle control by changing the RES slot value.

## <u>Conclusion</u>

By completing this lab we gained practical experience in digital circuit design through verilog by exploring the fundamentals of PWM based LED system control. We continued to learn how to structure our modules within Vivado to create a project that can be synthesized, implemented, and tested on our FPGA board.

## <u>Contributions</u>

Faris (50%) - Source code, helped with lab report

Nicholas (50%) - Testbench, handled simulations, helped with report, demo