CalPolyPomona
College of Engineering

Lab 8 Report

by

Jonathan Huynh #016137719

Adam Godfrey #015981472

Instructor: Dr. Mohamed Aly

Class: ECE 3300L .E02-OU - Verilog Design

August 14th, 2025

Code with Explanation:

Clock Divider

Explanation

The clock_divider_fixed module generates two slower clocks for us, 1k at 1 kHz and the pwm at 20 kHz via dividing the two. Since they have independent counters, both counters are reset when rst_n is active low, which helps create a clean output. It also will toggle each output when its respective counter reaches target count. Through this, we allow separate timing signals for control and PWM driving with a single CLK input.

Code

```
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

Explanation

The debounce module filters out the rapid on/off transitions caused by mechanical switch bouncing, ensuring a clean signal for processing. Without debouncing, unwanted multiple triggers could occur.

Code

```
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

Explanation

Load_fsm is a finite state machine that cycles through our four slots, it activates one at a time when we see a load_pulse. It outputs a 2-bit slot index and one encoded. While it also will generate individual signals for our reset, red, green, and blue channels depending on the input.

Code

```
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

Explanation

With this, we have a module drive three independent PWM outputs in R,G,B, using a shared counter that repeats every period + 1 ticks. Each channel is clamped to ensure it never exceeds effective period, preventing errors and oddity outputs. Each clock, the counter will reset at end of period and the PWM output is high while cnt is less than clamped.

Code

```
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
```

RGB LED Driver

Explanation

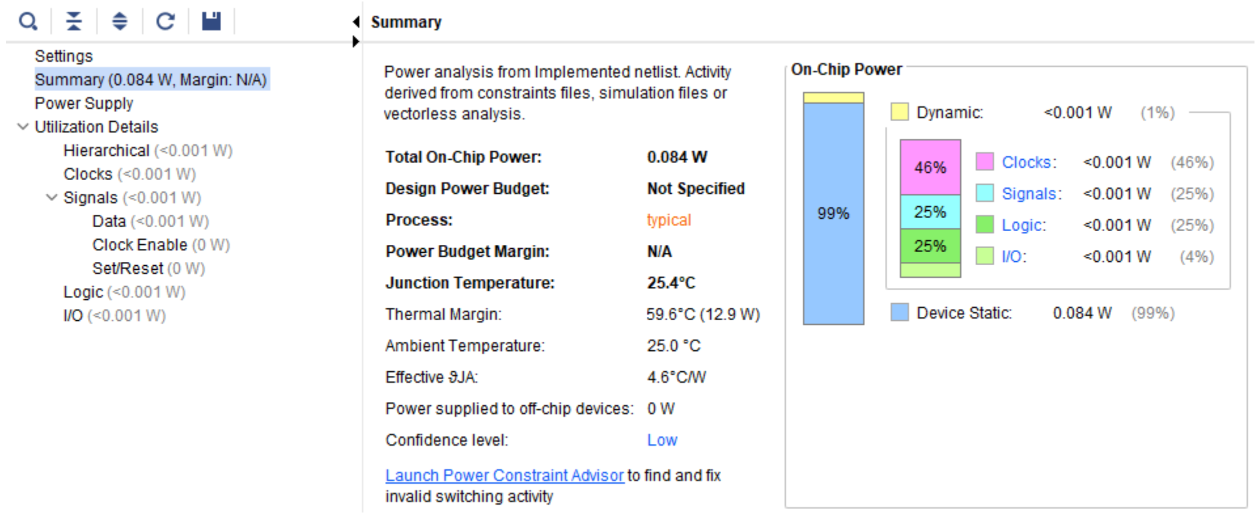
Our module routes the PWM to red,green,and blue channels here. It has option to invert as well if active-low. Through this, we give the behavior ability to adapt to different LEW configs without changing core logic, which makes it more simple and flexible.

Code

```
module rgb_led_driver #(parameter ACTIVE_LOW=0)(
    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
```

Screenshot Proofs:

Resource Utilization Table:



- Settings
- Summary (0.084 W, Margin: N/A)
- Power Supply
- Utilization Details
  - Hierarchical (<0.001 W)
  - Clocks (<0.001 W)
  - Signals (<0.001 W)
    - Data (<0.001 W)
    - Clock Enable (0 W)
    - Set/Reset (0 W)
  - Logic (<0.001 W)
  - I/O (<0.001 W)

Summary

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

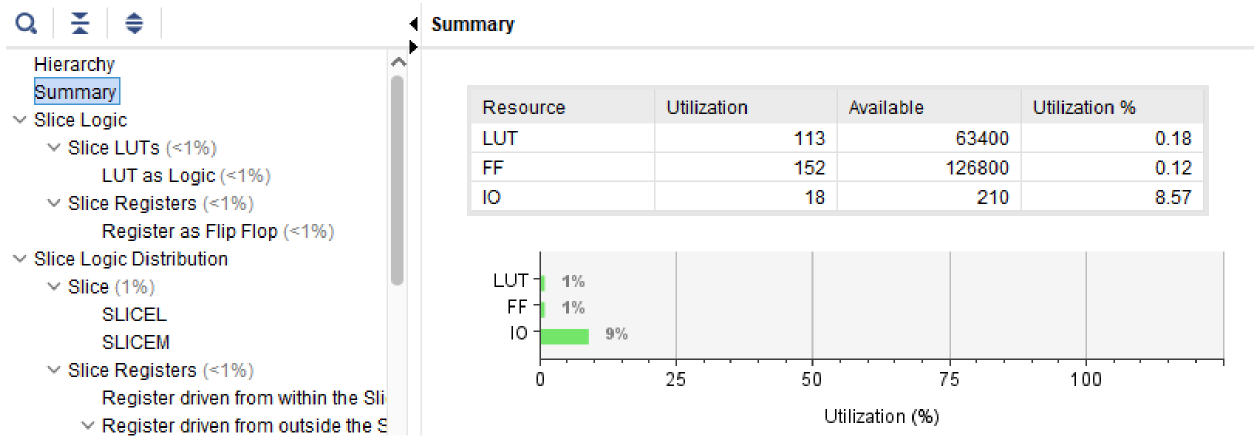
Total On-Chip Power:	**0.084 W**
Design Power Budget:	**Not Specified**
Process:	typical
Power Budget Margin:	**N/A**
Junction Temperature:	**25.4°C**
Thermal Margin:	59.6°C (12.9 W)
Ambient Temperature:	25.0 °C
Effective ϑJA:	4.6°C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Low

Launch Power Constraint Advisor to find and fix invalid switching activity

On-Chip Power

Dynamic:	<0.001 W	(1%)	
	Clocks:	<0.001 W	(46%)
	Signals:	<0.001 W	(25%)
	Logic:	<0.001 W	(25%)
	I/O:	<0.001 W	(4%)
Device Static:	0.084 W	(99%)	

Power Utilization:

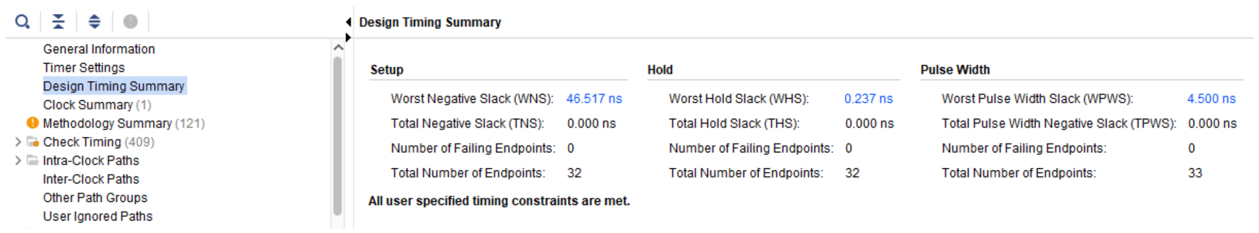


- Hierarchy
- Summary
- Slice Logic
  - Slice LUTs (<1%)
    - LUT as Logic (<1%)
  - Slice Registers (<1%)
    - Register as Flip Flop (<1%)
- Slice Logic Distribution
  - Slice (1%)
    - SLICEL
    - SLICEM
  - Slice Registers (<1%)
    - Register driven from within the Sli
    - Register driven from outside the S

Summary

Resource	Utilization	Available	Utilization %
LUT	113	63400	0.18
FF	152	126800	0.12
IO	18	210	8.57

Timing Summary:



- General Information
- Timer Settings
- Design Timing Summary
- Clock Summary (1)
- Methodology Summary (121)
- Check Timing (409)
- Intra-Clock Paths
- Inter-Clock Paths
- Other Path Groups
- User Ignored Paths

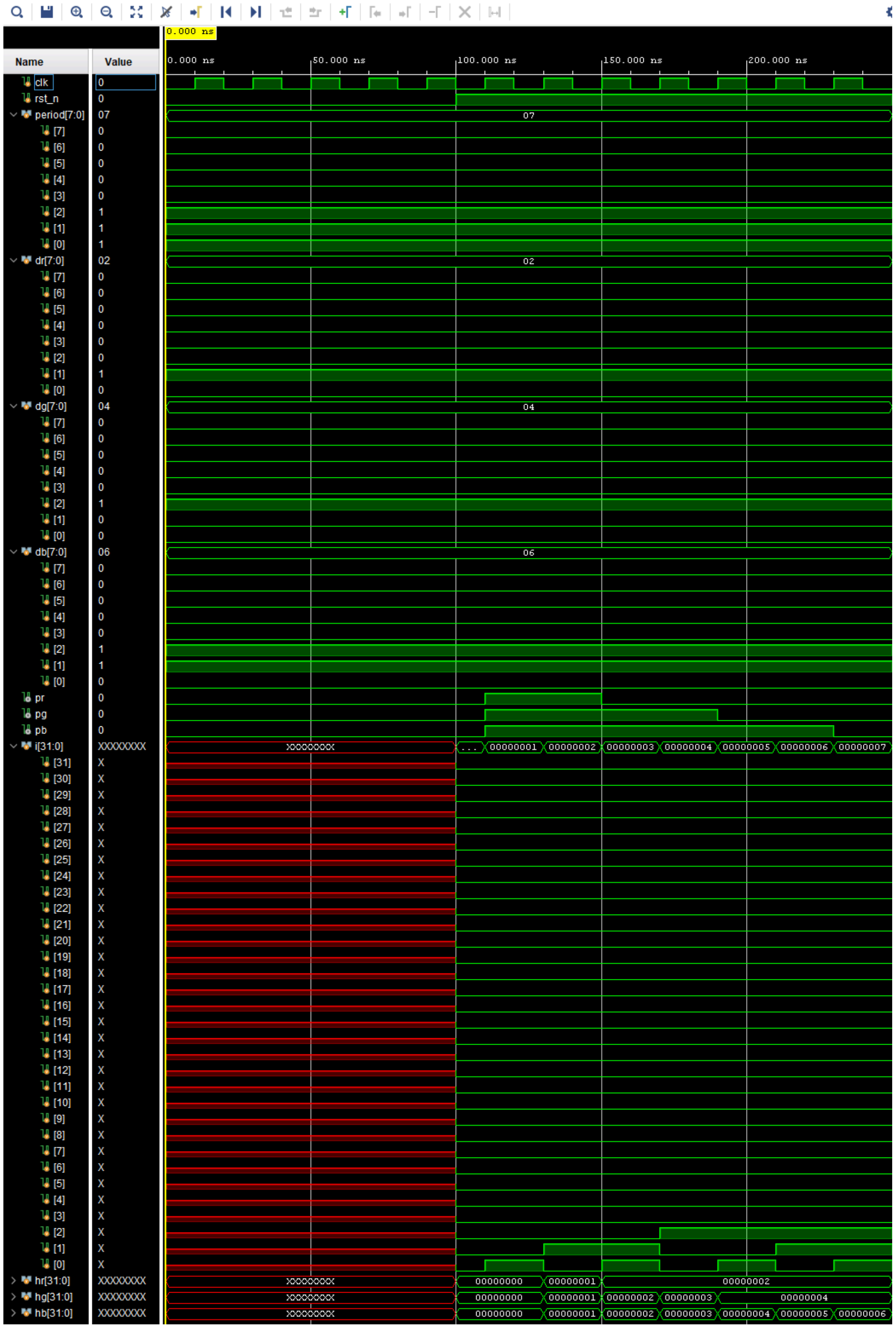
Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	46.517 ns	Worst Hold Slack (WHS):	0.237 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.

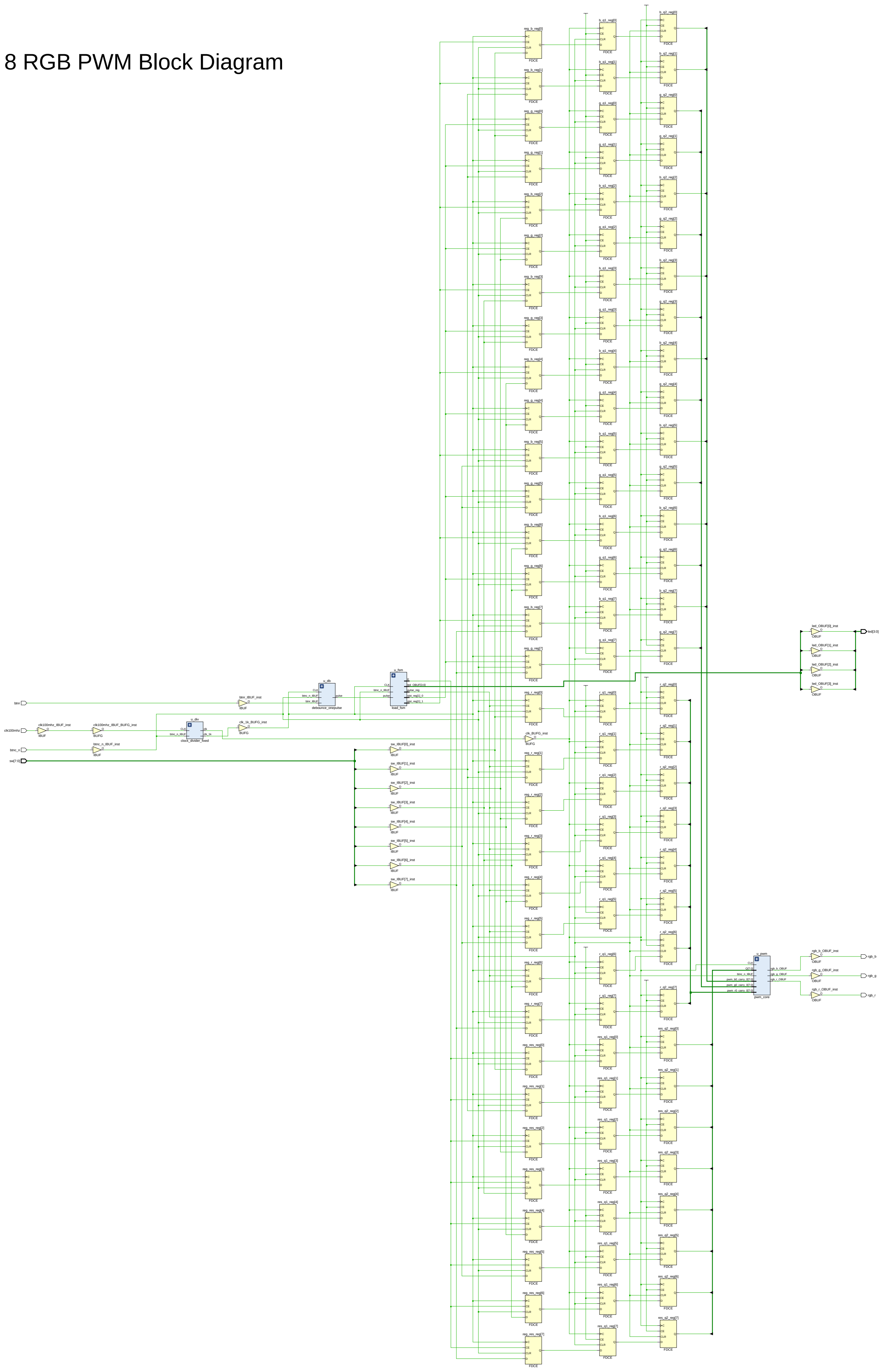
Simulation Waveforms:

PWM Core:

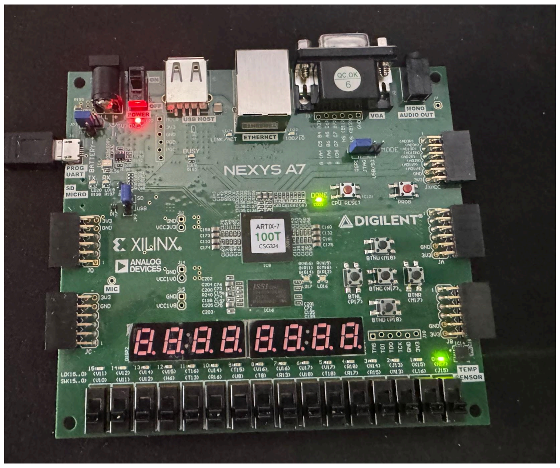
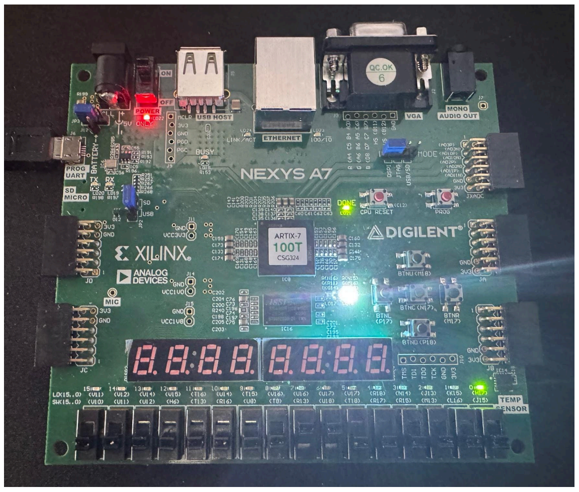
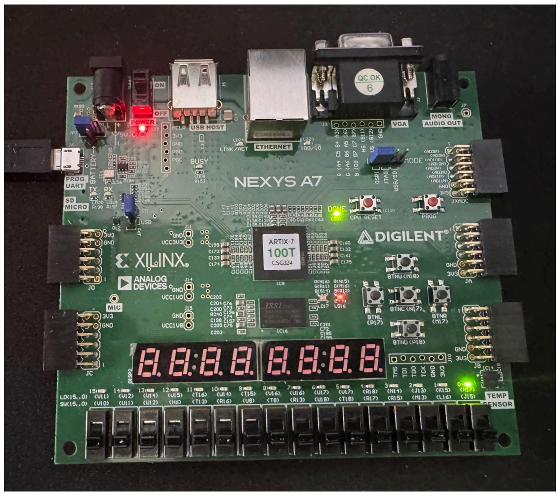
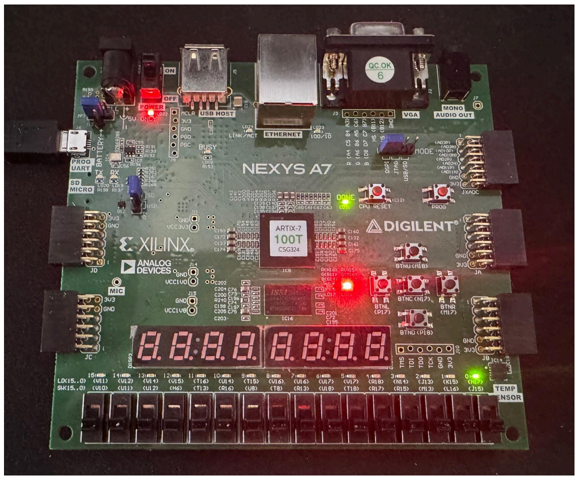
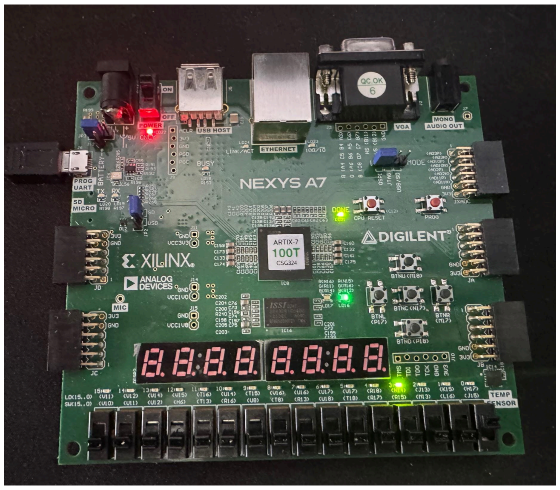
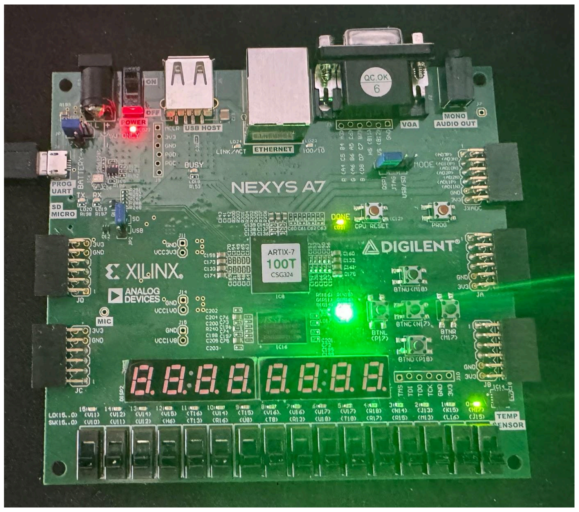


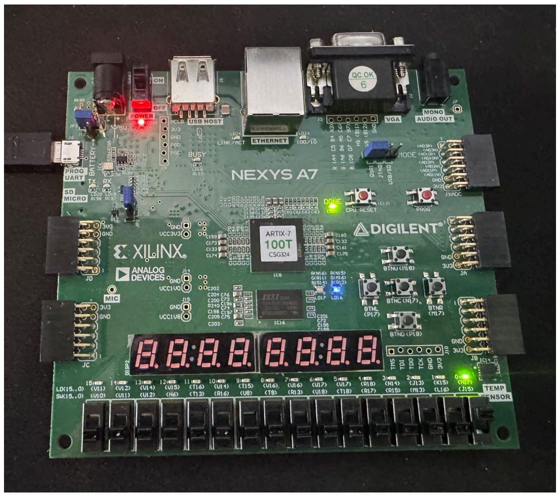
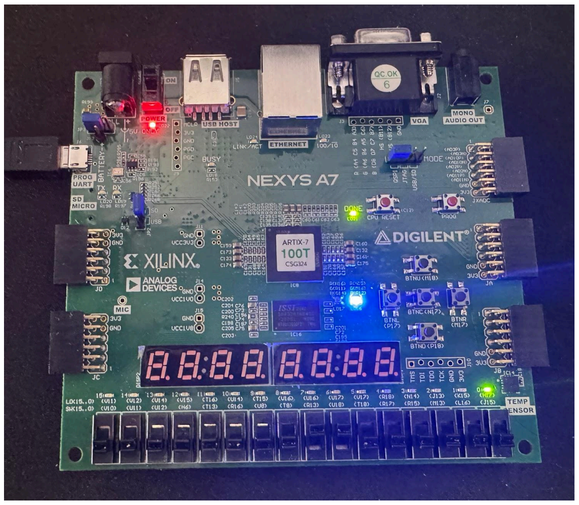
Name	Value
clk	0
rst_n	0
period[7:0]	07
[7]	0
[6]	0
[5]	0
[4]	0
[3]	0
[2]	1
[1]	1
[0]	1
dr[7:0]	02
[7]	0
[6]	0
[5]	0
[4]	0
[3]	0
[2]	0
[1]	1
[0]	0
dg[7:0]	04
[7]	0
[6]	0
[5]	0
[4]	0
[3]	0
[2]	1
[1]	0
[0]	0
db[7:0]	06
[7]	0
[6]	0
[5]	0
[4]	0
[3]	0
[2]	1
[1]	1
[0]	0
pr	0
pg	0
pb	0
i[31:0]	XXXXXXXX
[31]	X
[30]	X
[29]	X
[28]	X
[27]	X
[26]	X
[25]	X
[24]	X
[23]	X
[22]	X
[21]	X
[20]	X
[19]	X
[18]	X
[17]	X
[16]	X
[15]	X
[14]	X
[13]	X
[12]	X
[11]	X
[10]	X
[9]	X
[8]	X
[7]	X
[6]	X
[5]	X
[4]	X
[3]	X
[2]	X
[1]	X
[0]	X
hr[31:0]	XXXXXXXX
hg[31:0]	XXXXXXXX
hb[31:0]	XXXXXXXX

Lab 8 RGB PWM Block Diagram



Board Photo:

	Low Brightness	*High Brightness*
Fully On/Off		
Red		
Green		

Blue		

Partner Contributions:

Team Member	**Contribution**	**% Effort**
Jonathan Huynh	**Code:** [Clock Divider, Debounce, Load FSM] **Testbench:** [Clock Divider, Debounce, Load FSMr] **Additional:** Synthesis/Implementation and Demo	50%
Adam Godfrey	**Code:** [Top Module, RGB Driver, PWM Core] **Testbench:** [Top Module, RGB Driver, PWM Core] **Additional:** Simulation and Lab Report	50%