ECE 3300L.01 - Lab 8

RGB LED PWM Controller (Nexys A7)

Professor Mohamed Aly
Group Q: Kevin Tang(015429622) &
Jared Mocling(015215057)

August 14th, 2025

Design:

clock_divider_fixed.v

```
1  // clock_divider_fixed.v
2  `timescale 1ns/1ps
3  module clock_divider_fixed #(
4  parameter integer INPUT_HZ = 100_000_000,
5  parameter integer TICK1_HZ = 1_000,
6  parameter integer PWM_HZ = 20_000
7  )(
8  input wire clk_in,
9  input wire rst_n,
10 output reg clk_1k,
11 output reg clk_pwm
12
13 );
14 localparam integer DIV1H = (INPUT_HZ/TICK1_HZ)/2;
15 localparam integer DIVPMH = (INPUT_HZ/PWM_HZ)/2;
16 reg [$clog2(DIV1H):0] c1;
17 reg [$clog2(DIVPMH):0] c2;
18 always @(posedge clk_in or negedge rst_n) begin
19 if (!rst_n) begin c1 <= 0; clk_1k <= 0; c2 <= 0; clk_pwm <= 0; end
20 else begin
21 if (c1 == DIV1H-1) begin c1 <= 0; clk_1k <= ~clk_1k; end else c1 <=
22 c1+1;
23 if (c2 == DIVPMH-1) begin c2 <= 0; clk_pwm <= ~clk_pwm; end else c2
24 <= c2+1;
25 end
26 end
27 endmodule
```

debounce_onepulse.v

```
1  // debounce_onepulse.v
2  `timescale 1ns/1ps
3  module debounce_onepulse #(
4  parameter integer STABLE_TICKS = 20
5  )(
6  input wire clk,
7  input wire rst_n,
8  input wire din,
9  output reg pulse
10 );
11 reg d0, d1;
12 reg stable, stable_q;
13 reg [$clog2(STABLE_TICKS+1)-1:0] cnt;
14 always @(posedge clk or negedge rst_n) begin
15 if (!rst_n) begin d0<=0; d1<=0; end else begin d0<=din; d1<=d0; end
16 end
17 always @(posedge clk or negedge rst_n) begin
18 if (!rst_n) begin cnt<=0; stable<=0; end
19 else if (d1 != stable) begin
20 if (cnt==STABLE_TICKS) begin stable<=d1; cnt<=0; end
21
22
23 else cnt<=cnt+1;
24 end else cnt<=0;
25 end
26 always @(posedge clk or negedge rst_n) begin
27 if (!rst_n) begin stable_q<=0; pulse<=0; end
28 else begin pulse <= (~stable_q) & stable; stable_q <= stable; end
29 end
30 endmodule
```

load_fsm.v

```
1  // load_fsm.v
2  `timescale 1ns/1ps
3  module load_fsm(
4  input wire clk,
5  input wire rst_n,
6  input wire load_pulse,
7  output reg [1:0] slot,
8  output wire [3:0] slot_onehot,
9  output reg wr_res, wr_r, wr_g, wr_b
10 );
11 assign slot_onehot = 4'b0001 << slot;
12 always @(posedge clk or negedge rst_n) begin
13 if (!rst_n) slot <= 2'd0;
14 else if (load_pulse) slot <= slot + 2'd1;
15 end
16 always @* begin
17 wr_res = 0; wr_r = 0; wr_g = 0; wr_b = 0;
18 case (slot)
19 2'd0: wr_res = load_pulse;
20 2'd1: wr_r = load_pulse;
21 2'd2: wr_g = load_pulse;
22 2'd3: wr_b = load_pulse;
23 endcase
24 end
25 endmodule
```

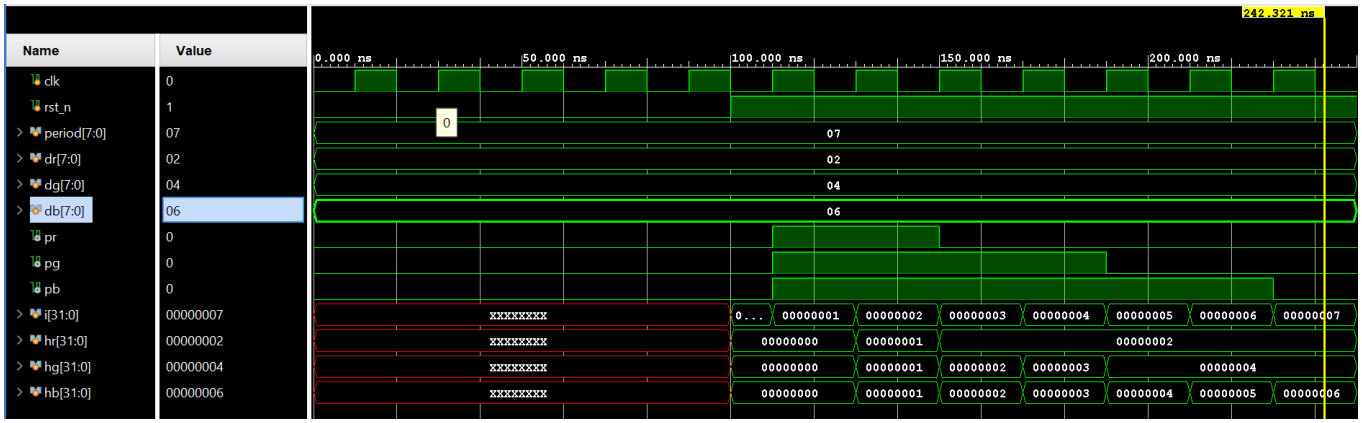
pwm_core.v

```
// 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, 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.v

```
1  // rgb_led_driver.v
2  `timescale 1ns/1ps
3  module rgb_led_driver #(parameter ACTIVE_LOW=0)(
4  input wire pwm_r, pwm_g, pwm_b,
5  output wire led_r, led_g, led_b
6  );
7  generate
8  if (ACTIVE_LOW) begin
9  assign led_r = ~pwm_r;
10 assign led_g = ~pwm_g;
11 assign led_b = ~pwm_b;
12 end else begin
13 assign led_r = pwm_r;
14 assign led_g = pwm_g;
15 assign led_b = pwm_b;
16 end
17 endgenerate
18 endmodule
```

Testbench Waveform:



Utilization/Implementation:

Name	Slice LUTs (63400)	Slice Registers (126800)	Slice (15850)	LUT as Logic (63400)	Bonded IOB (210)	BUFGCTRL (32)
top_lab8	113	152	68	113	18	3
u_pwm (pwm_core)	64	12	27	64	0	0

Power

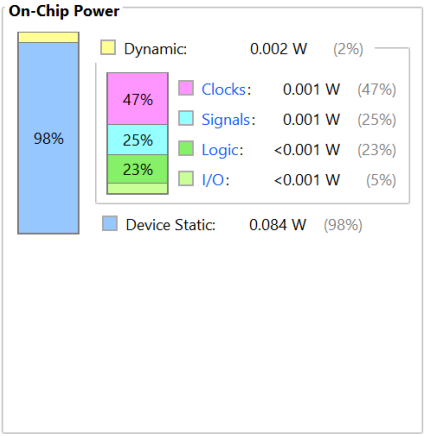
Summary

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

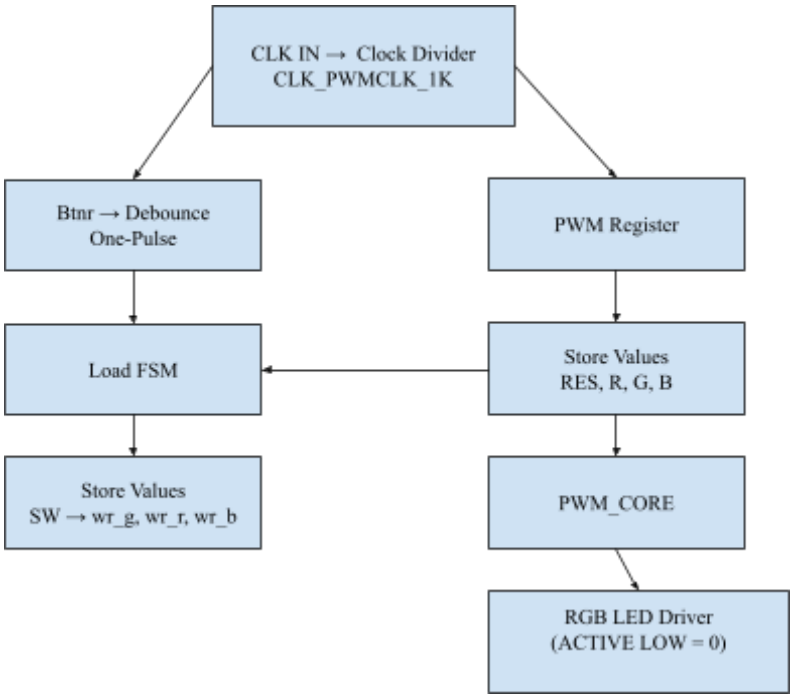
Total On-Chip Power:	**0.086 W**
Design Power Budget:	**Not Specified**
Process:	typical
Power Budget Margin:	**N/A**
Junction Temperature:	**25.4°C**
Thermal Margin:	59.6°C (12.9 W)
Ambient Temperature:	25.0 °C
Effective ϑJA:	4.6°C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Low

Launch Power Constraint Advisor to find and fix invalid switching activity

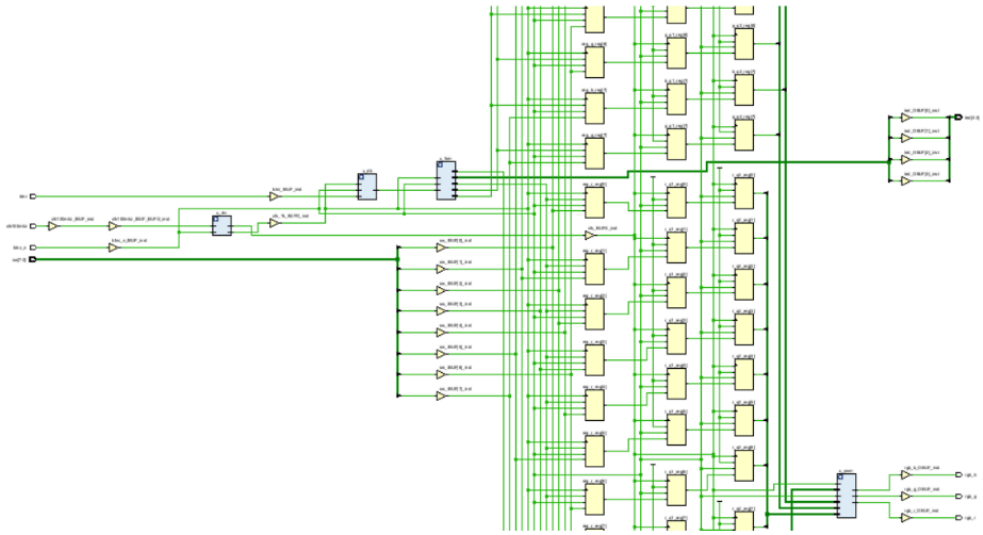
On-Chip Power



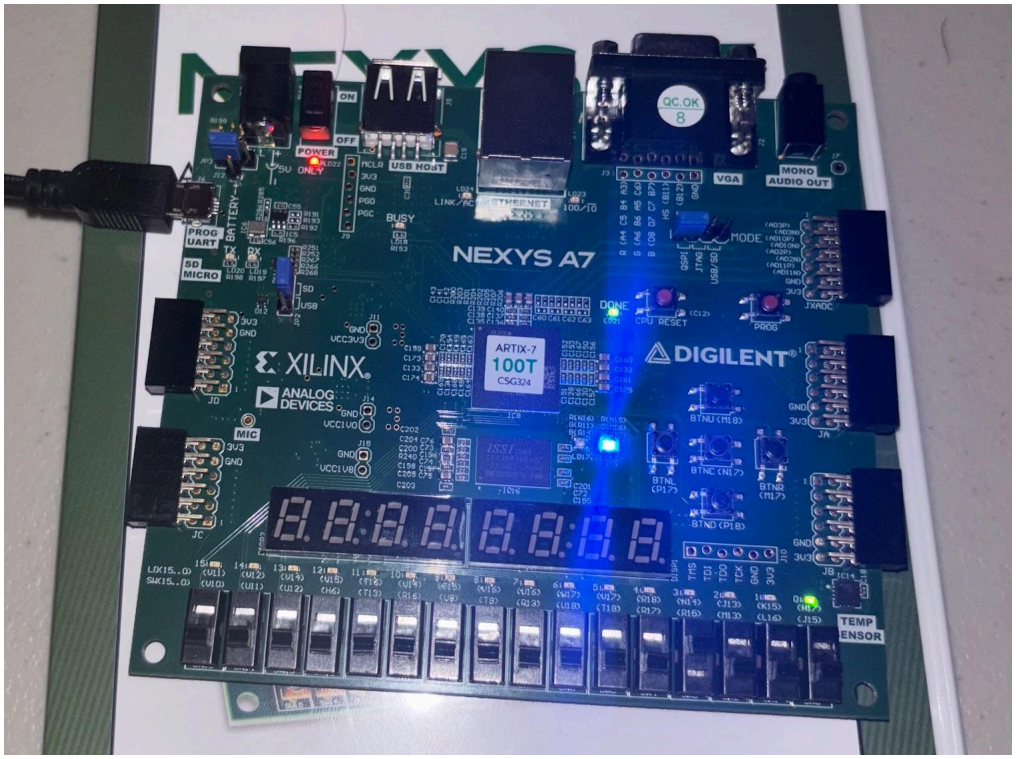
Block Diagram



Schematic:



Board:



Short reflection on 4-slot loading and RES + 1:

In this lab, we used a 4-slot loading system to store separate 8-bit values for the PWM resolution (RES) and red, green, and blue duty cycles (R, G, B). Here we have the load finite state machine (FSM) select each slot in sequence, and when pressing the LOAD button while there is a value on sw[7:0] loads that value into the active slot. Here, Slot 0 holds the RES value, which determines the PWM period, and slots 1 through 3 hold the R, G, and B duty cycle values. Then the PWM core counts from 0 to RES + 1, so the actual period is one clock cycle longer than the value of the RES register. This enables a RES value of 0xFF to produce a full-scale 256-step PWM cycle.

Contributions:

Jared Mocling (50%) - board demo, reflection, report
Kevin Tang (50%)- testbench screenshots, running the simulation, report