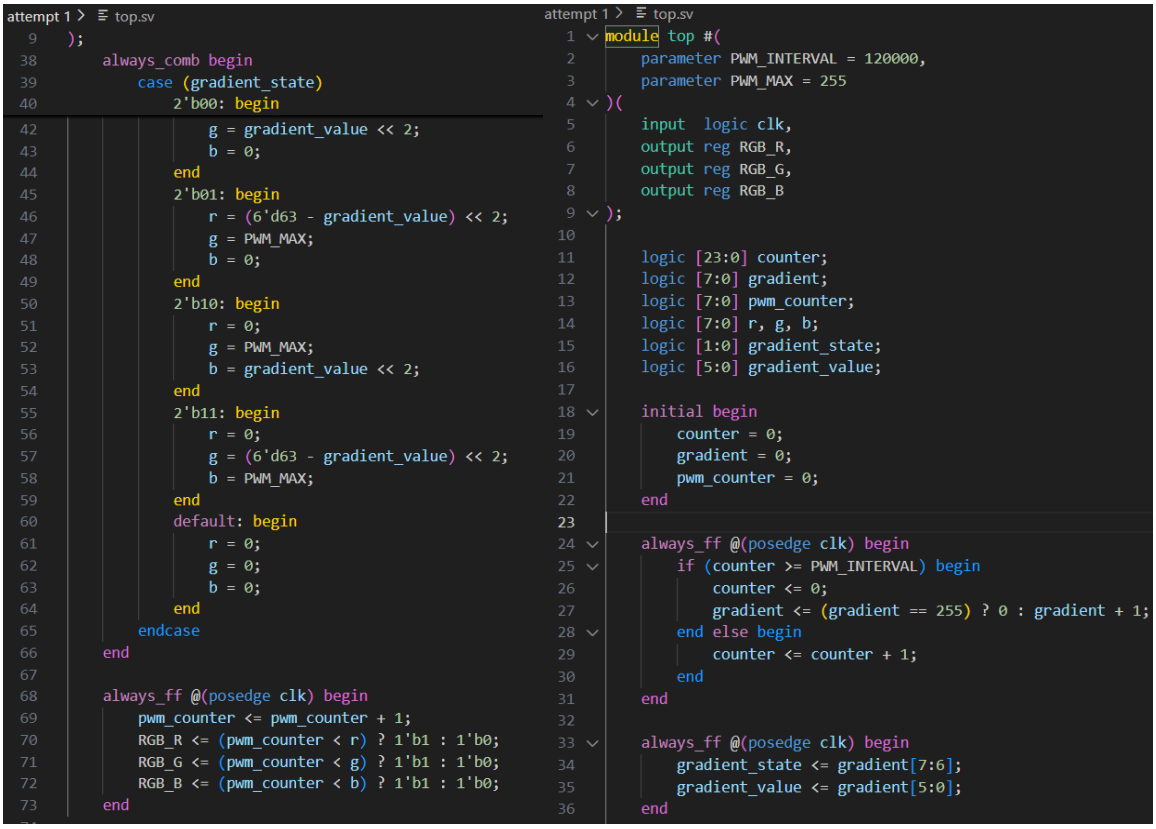
Project 2

My first attempt was to cycle though the colors and creating fade by increasing and decreasing the bit by left shit 2. But after receiving the email and did the proper gtkwave I realized the wave is wrong



```
attempt 1 > top.sv
 9  );
38      always_comb begin
39          case (gradient_state)
40              2'b00: begin
42                  g = gradient_value << 2;
43                  b = 0;
44              end
45              2'b01: begin
46                  r = (6'd63 - gradient_value) << 2;
47                  g = PWM_MAX;
48                  b = 0;
49              end
50              2'b10: begin
51                  r = 0;
52                  g = PWM_MAX;
53                  b = gradient_value << 2;
54              end
55              2'b11: begin
56                  r = 0;
57                  g = (6'd63 - gradient_value) << 2;
58                  b = PWM_MAX;
59              end
60              default: begin
61                  r = 0;
62                  g = 0;
63                  b = 0;
64              end
65          endcase
66      end
67
68      always_ff @(posedge clk) begin
69          pwm_counter <= pwm_counter + 1;
70          RGB_R <= (pwm_counter < r) ? 1'b1 : 1'b0;
71          RGB_G <= (pwm_counter < g) ? 1'b1 : 1'b0;
72          RGB_B <= (pwm_counter < b) ? 1'b1 : 1'b0;
73      end
```

```
attempt 1 > top.sv
1  module top #(
2      parameter PWM_INTERVAL = 120000,
3      parameter PWM_MAX = 255
4  )(
5      input  logic clk,
6      output reg RGB_R,
7      output reg RGB_G,
8      output reg RGB_B
9  );
10
11     logic [23:0] counter;
12     logic [7:0] gradient;
13     logic [7:0] pwm_counter;
14     logic [7:0] r, g, b;
15     logic [1:0] gradient_state;
16     logic [5:0] gradient_value;
17
18     initial begin
19         counter = 0;
20         gradient = 0;
21         pwm_counter = 0;
22     end
23
24     always_ff @(posedge clk) begin
25         if (counter >= PWM_INTERVAL) begin
26             counter <= 0;
27             gradient <= (gradient == 255) ? 0 : gradient + 1;
28         end else begin
29             counter <= counter + 1;
30         end
31     end
32
33     always_ff @(posedge clk) begin
34         gradient_state <= gradient[7:6];
35         gradient_value <= gradient[5:0];
36     end
```

Second attempt

RGB LED controller that uses PWM to control the brightness of red, green, and blue LEDs. The design includes three main modules: top, fade, and pwm. From the given example

Top Module:

- The top module integrates the fade and pwm modules to control the RGB LEDs.
- It receives a clock input (clk) and outputs signals to control the RGB LEDs (RGB_R, RGB_G, RGB_B).

- The fade modules generate PWM values that gradually change over time to create a fading effect for each color.
- The pwm modules use these PWM values to generate PWM signals for each color.
- The final RGB LED output signals are obtained by inverting the PWM signals (~red_pwm, ~green_pwm, ~blue_pwm).

Fade Module:

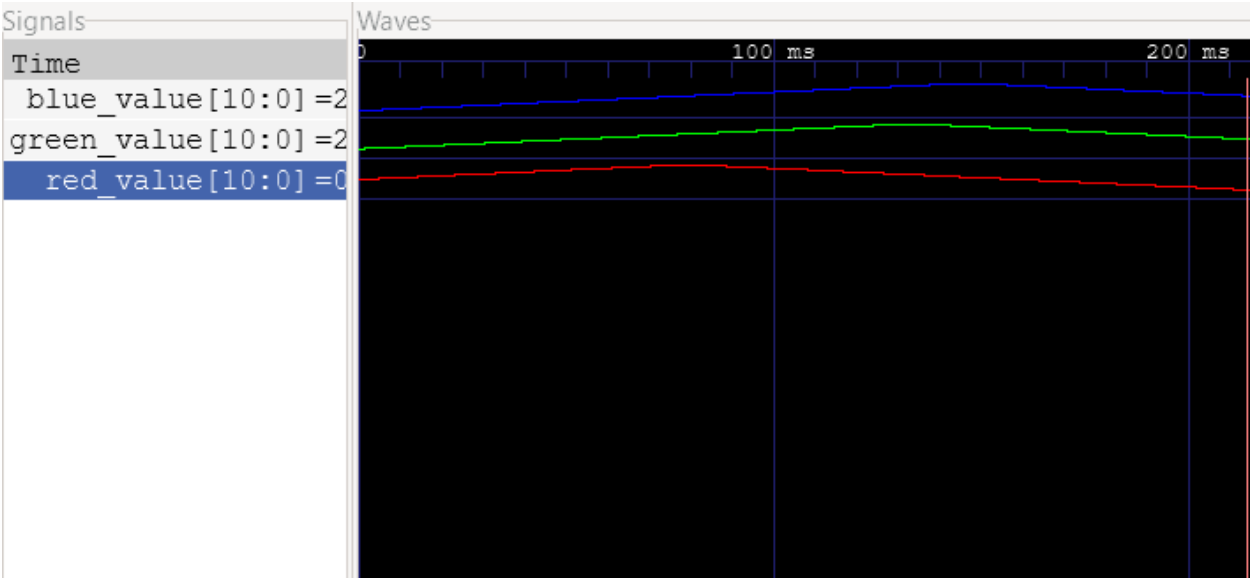
- The fade module generates PWM values that gradually increase and decrease to create a fading effect.
- It includes a state machine with two states: increment (PWM_INC) and decrement (PWM_DEC). And the phase shift to shift the starting point
- The module uses counters to determine when to change the PWM value and when to switch states.

PWM Module:

- The pwm module generates a PWM signal based on the given PWM value.
- It uses a counter (pwm_count) to create the PWM signal by comparing the counter value with the PWM value.
- The output signal (pwm_out) is high when the counter value is less than the PWM value, creating the desired PWM effect.

Testbench:

- The testbench (fade_tb) verifies the functionality of the top module.
- It generates a clock signal and runs the simulation to test the RGB LED controller.



I am unable to do the two part delay when it reaches the peak/zero

Github:

zhi41/P2