Computer Architecture MP 2 Report

Chang Jun Park

February 28, 2025

1 Introduction

This report presents the design and implementation of an RGB LED controller circuit for the iceBlinkPico FPGA board. The controller creates a smooth and continuous transition through the color spectrum by modulating the duty cycles of three PWM signals that drive the red, green, and blue LED components. The circuit implements a full 360 degree color wheel, providing an aesthetically pleasing visual demonstration of digital control systems and PWM techniques.

2 Design Overview

The system architecture consists of three primary components:

1. **Main RGB LED Controller Module (`rgb_led_controller.sv`)**: Manages the color transitions by generating appropriate PWM duty cycle values for each color channel based on the current position in the color wheel.
2. **PWM Generator Module (`pwm.sv`)**: Three instances of this module convert the 8-bit duty cycle values into PWM signals for each color channel (red, green, and blue).
3. **Top-level Module (`top.sv`)**: Interfaces the controller with the physical hardware, including the necessary configuration for the RGB LED on the iceBlinkPico board.

3 Circuit Operation

3.1 Color Transition Algorithm

The design implements a full RGB color wheel by dividing it into six 60-degree segments, each representing a transition between two primary colors:

- **Segment 0 (0°-60°)**: Red = 100%, Blue = 0%, Green ramps from 0% to 100% (Red → Yellow)
- **Segment 1 (60°-120°)**: Green = 100%, Blue = 0%, Red ramps from 100% to 0% (Yellow → Green)
- **Segment 2 (120°-180°)**: Green = 100%, Red = 0%, Blue ramps from 0% to 100% (Green → Cyan)
- **Segment 3 (180°-240°)**: Blue = 100%, Red = 0%, Green ramps from 100% to 0% (Cyan → Blue)
- **Segment 4 (240°-300°)**: Blue = 100%, Green = 0%, Red ramps from 0% to 100% (Blue → Magenta)
- **Segment 5 (300°-360°)**: Red = 100%, Green = 0%, Blue ramps from 100% to 0% (Magenta → Red)

This approach ensures smooth transitions between colors while maintaining high color saturation in operation.

3.2 Timing Control

The color wheel cycle is controlled by the following parameters:

$$\text{CLK\_FREQ} = 12,000,000 \text{ Hz (12 MHz clock from the iceBlinkPico board)} \tag{1}$$

$$\text{TOTAL\_STEPS} = 360 \text{ steps to complete a full color wheel (1 step per degree)} \tag{2}$$

$$\text{STEP\_SIZE} = 60 \text{ steps per segment (matching the 60-degree segments)} \tag{3}$$

$$\text{CYCLES\_PER\_STEP} = \frac{\text{CLK\_FREQ}}{\text{TOTAL\_STEPS}} = 33,333 \text{ clock cycles per step} \tag{4}$$

These parameters determine how quickly the LED cycles through the color spectrum. The step counter increments after CYCLES_PER_STEP clock cycles, and the color values are updated based on the current step position in the color wheel.

3.3 PWM Implementation

Each color channel uses an 8-bit PWM implementation:

- **Resolution**: 8 bits (256 brightness levels from 0 to 255)
- **Counter Range**: 0 to 255, rolling over to create the PWM period
- **Duty Cycle Control**: Output is high when the counter is less than the specified duty cycle value

4 Simulation Results

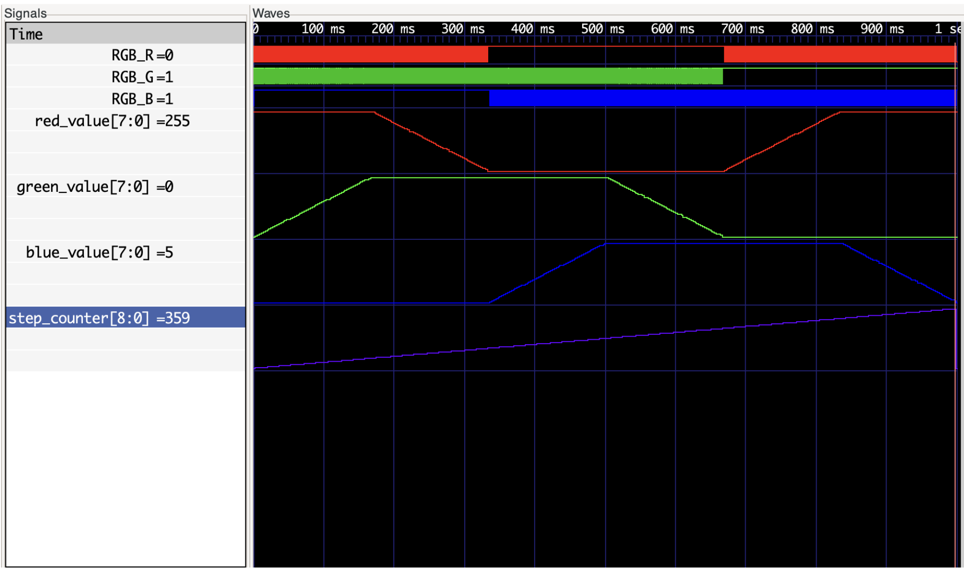

Figure 1: RGB Signal Simulation in GTKWave

The above GTKWave simulation plot demonstrates that the RGB signals change over time as the controller cycles through the color wheel. The plot shows:

1. The step counter gradually increasing up to 360 step size
2. The red, green, and blue PWM duty cycle values changing according to the current segment
3. How only one or two color components change at any given time while others remain at 0% or 100%

The simulation confirms the expected behavior: smooth transitions between colors, with one color ramping up while another ramps down in each segment, creating the full spectrum of colors in the RGB color wheel.

5 Hardware Implementation

The RGB LED on the iceBlinkPico board requires the PWM signals to be inverted:

RGB LED

```
assign RGB_R = ~red_led;
assign RGB_G = ~green_led;
assign RGB_B = ~blue_led;
```

The circuit uses the board's built-in 12 MHz clock as the timing reference.

6 Source Code

6.1 RGB LED Controller Module

RGB LED Controller Module (rgb_led_controller.sv)

```
module rgb_led_controller#(
    parameter CLK_FREQ = 12_000_000
) (
    input  logic clk,
    output logic red_led,
    output logic green_led,
    output logic blue_led
);

    localparam TOTAL_STEPS = 360;
    localparam STEP_SIZE   = 60;
    localparam CYCLES_PER_STEP = CLK_FREQ / TOTAL_STEPS;

    logic [$clog2(CYCLES_PER_STEP)-1:0] cycle_counter = 0;
    logic [8:0]  step_counter  = 0;  // 0 to 359

    // PWM values for RGB
    logic [7:0] red_value   = 8'd255;
    logic [7:0] green_value = 8'd0;
    logic [7:0] blue_value  = 8'd0;

    // Step counter logic
    always_ff @(posedge clk) begin
        if (cycle_counter >= CYCLES_PER_STEP - 1) begin
            cycle_counter <= 0;
            if (step_counter >= TOTAL_STEPS - 1)
                step_counter <= 0;
            else
                step_counter <= step_counter + 1;
        end
        else begin
            cycle_counter <= cycle_counter + 1;
        end
    end

    always_comb begin
        case (step_counter / STEP_SIZE)
            // Segment 0: 0 to 60 degrees
            // Red = 100%, Blue = 0%, Green ramps 0% to 100%
            0: begin
                red_value   = 8'd255;
                blue_value  = 8'd0;
                green_value = ((step_counter % STEP_SIZE) * 255) / STEP_SIZE;
            end

```

```
            // Segment 1: 60 to 120 degrees
            // Green = 100%, Blue = 0%, Red ramps 100% down to 0%
            1: begin
                green_value = 8'd255;
                blue_value  = 8'd0;
                red_value   = 8'd255 - ((step_counter % STEP_SIZE) * 255) / STEP_SIZE;
            end

            // Segment 2: 120 to 180 degrees
            // Green = 100%, Red = 0%, Blue ramps 0% to 100%
            2: begin
                green_value = 8'd255;
                red_value   = 8'd0;
                blue_value  = ((step_counter % STEP_SIZE) * 255) / STEP_SIZE;
            end

            // Segment 3: 180 to 240 degrees
            // Blue = 100%, Red = 0%, Green ramps 100% down to 0%
            3: begin
                blue_value  = 8'd255;
                red_value   = 8'd0;
                green_value = 8'd255 - ((step_counter % STEP_SIZE) * 255) / STEP_SIZE;
            end

            // Segment 4: 240 to 300 degrees
            // Blue = 100%, Green = 0%, Red ramps 0% to 100%
            4: begin
                blue_value  = 8'd255;
                green_value = 8'd0;
                red_value   = ((step_counter % STEP_SIZE) * 255) / STEP_SIZE;
            end

            // Segment 5: 300 to 360 degrees
            // Red = 100%, Green = 0%, Blue ramps 100% down to 0%
            5: begin
                red_value   = 8'd255;
                green_value = 8'd0;
                blue_value  = 8'd255 - ((step_counter % STEP_SIZE) * 255) / STEP_SIZE;
            end

            default: begin
                red_value   = 8'd255;
                green_value = 8'd0;
                blue_value  = 8'd0;
            end
        endcase
    end

    pwm red_pwm (
        .clk(clk),
        .duty_cycle(red_value),
        .pwm_out(red_led)
    );

    pwm green_pwm (
        .clk(clk),
        .duty_cycle(green_value),
        .pwm_out(green_led)
    );

    pwm blue_pwm (
```

```
        .clk(clk),
        .duty_cycle(blue_value),
        .pwm_out(blue_led)
    );

endmodule
```

6.2 PWM Module

PWM Module (pwm.sv)

```
module pwm (
    input logic clk,
    input logic [7:0] duty_cycle,
    output logic pwm_out
);

    logic [7:0] counter = 0;

    always_ff @(posedge clk) begin
        counter <= counter + 1;
    end

    assign pwm_out = (counter < duty_cycle);

endmodule
```

6.3 Top-level Module

Top-level Module (top.sv)

```
`include "pwm.sv"
`include "rgb.sv"

module top (
    input logic clk,
    output logic RGB_R,
    output logic RGB_G,
    output logic RGB_B
);
    logic red_led;
    logic green_led;
    logic blue_led;

    rgb_led_controller rgb_controller (
        .clk(clk),
        .red_led(red_led),
        .green_led(green_led),
        .blue_led(blue_led)
    );

    assign RGB_R = ~red_led;
    assign RGB_G = ~green_led;
    assign RGB_B = ~blue_led;
endmodule
```