SCOMP LED Peripheral for Perceptual Brightness Control

Pranav Eada

Submitted
April 22, 2025

Introduction

This document summarizes the design and purpose of our custom SCOMP peripheral, which provides gamma-corrected PWM control for up to 10 independent LEDs. It allows a programmer to control brightness and enable status via memory-mapped I/O, supporting smooth, flicker-free light transitions.

Our team prioritized simplicity, perceptual accuracy, and modularity. We chose count-and-compare architecture with a shared 8-bit counter and gamma correction using a lookup table. This removed the need for extra timers and kept the design efficient. We structured our work around early planning and regular testing, dividing tasks based on SCOMP familiarity. The peripheral met all the functional goals that our client specified, and key design decisions added flexibility and clarity for users who may extend or repurpose the system.

Device Functionality

This peripheral enables real-time brightness control of up to 10 LEDs using gamma-corrected Pulse Width Modulation (PWM). A SCOMP programmer interacts with the peripheral through:

- A 10-bit LED enable bitmask to specify active LEDs.
- An 8-bit brightness value for each enabled LED.

The register map below defines the memory layout:

Address (Hex)	**Register Name**	**IO Data (MSB:LSB)**
0x01	LED_EN	IO_Data(9:0)
0x20	LED0_BRIGHT	IO_Data(7:0)
0x21	LED1_BRIGHT	IO_Data(7:0)
0x22	LED2_BRIGHT	IO_Data(7:0)
0x23	LED3_BRIGHT	IO_Data(7:0)
0x24	LED4_BRIGHT	IO_Data(7:0)
0x25	LED5_BRIGHT	IO_Data(7:0)
0x26	LED6_BRIGHT	IO_Data(7:0)
0x27	LED7_BRIGHT	IO_Data(7:0)

0x28	LED8_BRIGHT	IO_Data(7:0)
0x29	LED9_BRIGHT	IO_Data(7:0)

Note: IO_Data is 16 bits wide in SCOMP; only the specified bits are used per register.

To further clarify how the peripheral integrates within the SCOMP system, below is a high-level block diagram of the LED controller's inputs and outputs:

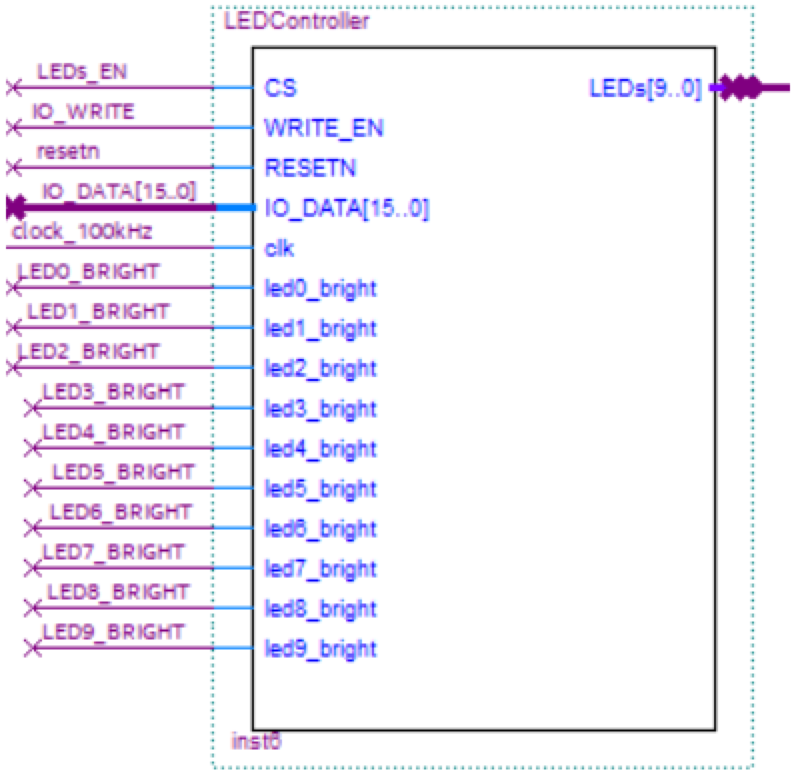

Figure 1. Peripheral block diagram symbol.

Internally, a shared 8-bit counter increments every clock cycle. Each brightness value is gamma-corrected using a lookup table (LUT) and compared to the counter. If the counter is less than the corrected value, the LED turns on; otherwise, it stays off.

This generates a 10-bit gamma bitmask each cycle. ANDing it with the enable bitmask ensures only selected LEDs are active. The result is sent to output pins, enabling smooth, flicker-free brightness changes without manual timing or interrupts.

Design Decisions and Implementation

We used a count-and-compare model to generate PWM signals using a single shared counter, simplifying hardware while preserving control over all 10 LEDs. The 10-bit LED enable mask mirrors Lab 8's format, offering a familiar structure.

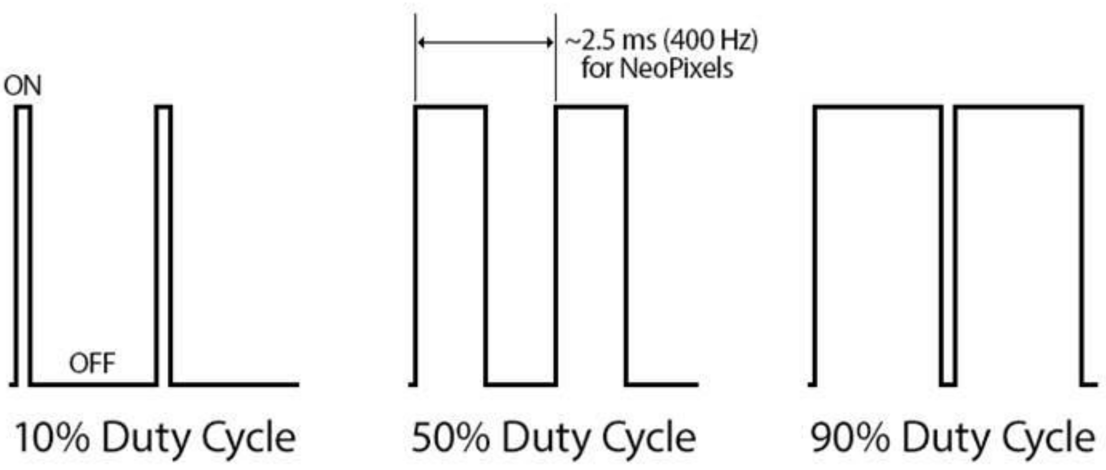

Figure 2. PWM signals at varying duty cycles. Higher duty cycles yield brighter LEDs.

Each LED has an 8-bit brightness register. To align output with human perception, we applied gamma correction using a lookup table. As shown in *Figure 3*, this compresses low-end values and expands the upper range for smoother transitions.

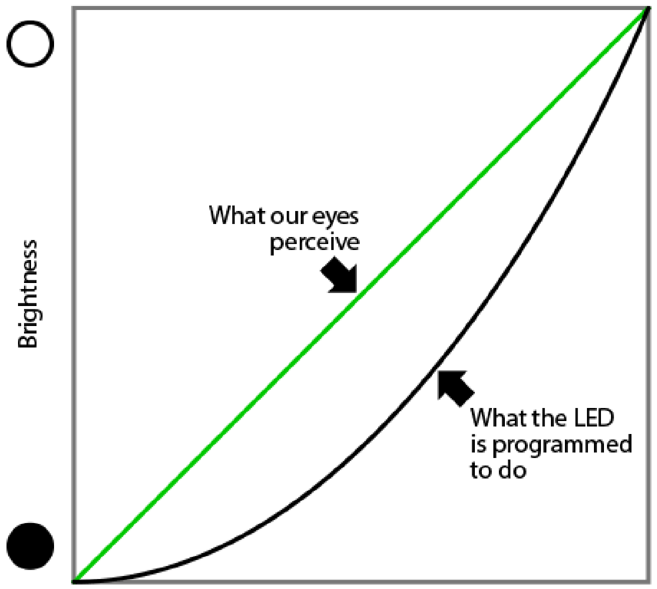

Figure 3. Gamma Correction curve for LED brightness distribution

An 8-bit counter increments each clock cycle and acts as the core timing mechanism for generating PWM signals. On every cycle, the system compares this counter to the gamma-corrected brightness values to determine whether each LED should be on or off. We chose this count-and-compare model for its simplicity and efficiency, eliminating the need for multiple timers or complex logic.

Each comparison yields a 10-bit bitmask representing the real-time LED output states. This is ANDed with the static enable mask to ensure only selected LEDs are active. By decoupling brightness from enablement, the design supports flexible patterns (like blinking) without altering stored brightness values.

Real-time computation also avoids waveform storage or added memory. This ensures compact design and smooth visual transitions. We briefly explored using separate timers per LED but rejected the idea due to high resource costs and increased complexity.

Conclusion

Our final design successfully achieved perceptual brightness control, individual LED support and familiar memory-mapped I/O: key goals of our proposal. This project demonstrated how thoughtful hardware design can balance simplicity, efficiency, and perceptual quality.

If revisiting the project, we would explore supporting dynamic PWM frequency or debug readbacks for easier testing. Still, the current design offers a strong foundation for future projects and serves as a clear example of building perceptually aware, resource-conscious SCOMP peripherals.