ECE 2031 NeoPixel Controller VHDL Peripheral

Jadon Grossberg

Submitted
December 7, 2021

Introduction

This document describes the architecture and proper use of an SCOMP peripheral capable of driving up to 256 NeoPixels at ~87Hz. The peripheral uses an FSM to display a variety of patterns depending on an opcode. We prioritized a scalable architecture with an intuitive and low friction path for future development. The peripheral can set 16- and 24-bit color to a specified LED and set 24-bit color to all LEDs. Using a non-defined opcode interprets the data as 16-bit color to be assigned to all LEDs in a single data latch. This makes a small fraction of the colors unattainable. This limitation is caused by a simplification made during development to avoid latch signal timing issues.

Device Functionality

The peripheral can control a string of up to 256 NeoPixels by receiving instructions via SCOMP's 16-bit IO data bus. The number of controllable LEDs can be specified by changing the parameter "num_leds" in the block diagram.

Peripheral Architecture

The peripheral consists of a finite state machine that transitions when data is latched to it.

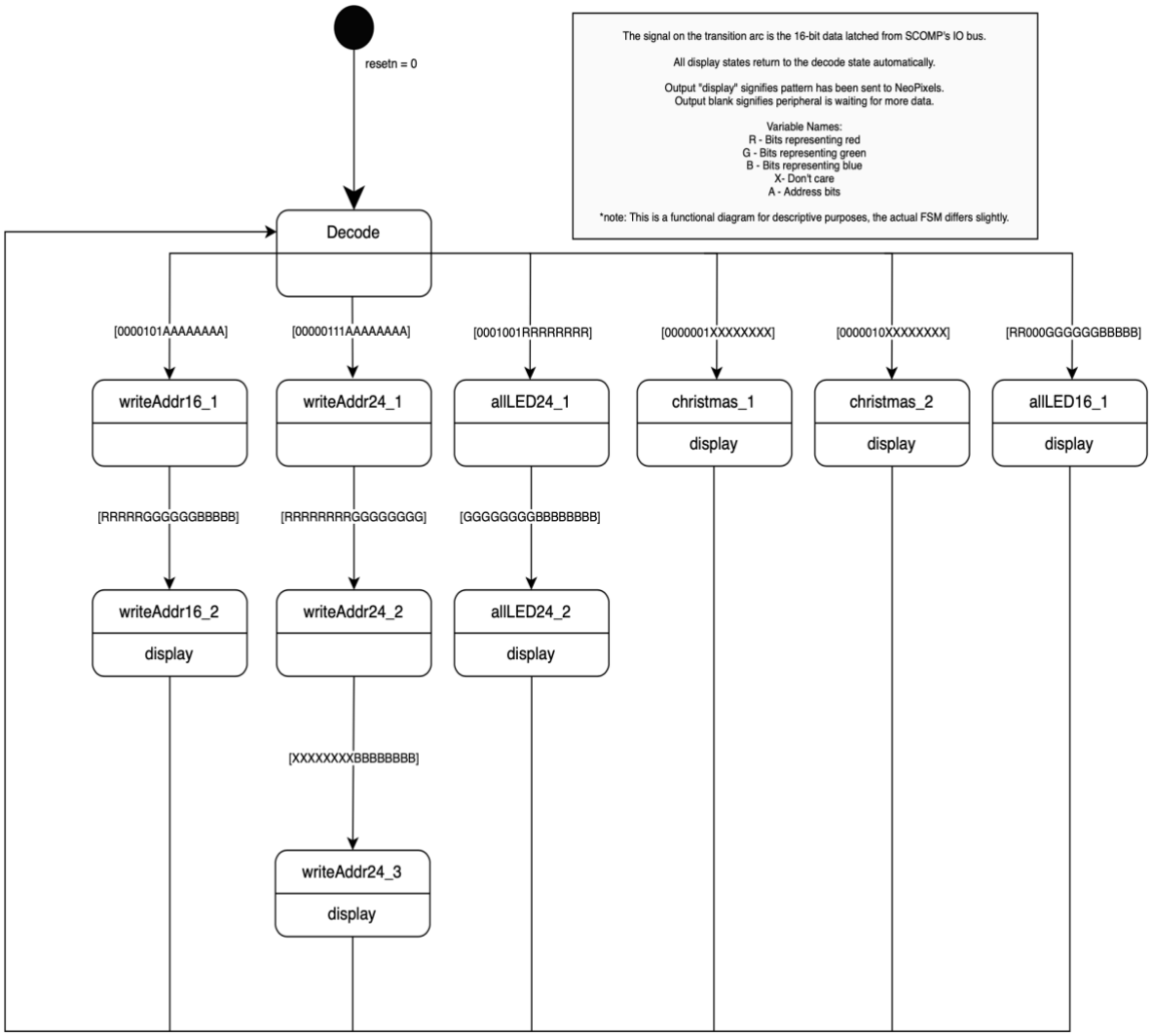

Figure 1. UML diagram describing peripheral's finite state machine.

Like SCOMP's control unit, the peripheral receives an opcode that determines what next state (and thus what pattern) to execute. This enables the programmer to enter arbitrarily large amounts of data through a sequence of data latches.

Refresh Rate

To demonstrate that the peripheral supports refreshing strings of all supported lengths at 30 Hz or faster, the team performed the following calculation:

1 / ((Max time to load 1 bit) * (24 bits per NeoPixel) * (256 NeoPixels) + (Reset pulse time) * (1/(10 MHz reset clock))) = 1/ (1.85 * 10^{-6} * 24 * 256 + 1000 * 10^{-7}) Hz ≈ **87 Hz for refreshing 256 LEDs**

Patterns

In order to display a pattern, an 8-bit opcode is added to the input data. Then, the user can supply the rest of the data (for example, a 16-bit color) in subsequent data latches. For the data latch visual, 'A' is the binary representation of the address (specific LED) that is changed, and 'R', 'G', and 'B' represent Red, Green, and Blue values respectively.

Christmas 1 (Opcode 00000001)

The Christmas 1 pattern displays alternating green and red lights and comprises of one data latch:

0	0	0	0	0	0	0	1	X	X	X	X	X	X	X	X

Christmas 2 (Opcode 00000010)

Christmas 2 displays alternating green and red lights in the opposite order of Christmas 1 and comprises of one data latch:

0	0	0	0	0	0	1	0	X	X	X	X	X	X	X	X

16-bit color at an address (Opcode 00000101)

This operation allows the user to write a 16 bit color to an LED without overwriting other LEDs in two data latches.

Data latch 1 (Opcode and address):

0	0	0	0	0	1	0	1	A	A	A	A	A	A	A	A

Data latch 2 (16-bit color):

R	R	R	R	R	G	G	G	G	G	G	B	B	B	B	B

24-bit color at an address (Opcode 00000111)

This operation allows the user to write a 24 bit color to an LED without overwriting other LEDs in three data latches.

Data latch 1 (Opcode and address):

0	0	0	0	0	1	1	1	A	A	A	A	A	A	A	A

Data latch 2 (Red and green bits):

R	R	R	R	R	R	R	R	G	G	G	G	G	G	G	G

Data latch 3 (Blue bits):

X	X	X	X	X	X	X	X	B	B	B	B	B	B	B	B

24-bit color to all LEDs (Opcode 00001001)

This operation will overwrite all LEDs with a specified 24-bit color value in two data latches.

Data Latch 1 (Opcode and red bits):

0	0	0	0	1	0	0	1	R	R	R	R	R	R	R	R

Data Latch 2 (Green and blue bits):

G	G	G	G	G	G	G	G	B	B	B	B	B	B	B	B

16-bit color to all LEDs (Any non-existing opcode)

This pattern will overwrite all LEDs with a 16-bit color in one OUT. The upper 8 bits of the color may **NOT** overlap with any defined opcodes, otherwise the corresponding procedure is performed.

R	R	R	R	R*	G*	G*	G*	G	G	G	B	B	B	B	B

*- bits at risk of overlapping an opcode.

Design Decisions and Implementation

The design has always centered around an FSM. The implementation of the “16-bit color to all LEDs" pattern has drastically changed over the design process, starting out as a separate peripheral and becoming integrated into a single peripheral for simplicity. Otherwise, the peripheral’s architecture has remained constant throughout the design process.

Conclusions

The peripheral integrates common concepts such as opcodes and finite state machines which make its use intuitive for engineers. It is recommended to declare opcodes as variables with readable names and to make use of the 24-bit color to all LEDs for better color accuracy and avoid opcode overlap. In future iterations, debugging the process statement that determines which latch signal was pulled high

will fix the 16-bit color to all LEDs. Overall, the peripheral exceeded expectations in ease of use and scalability but basic functionality could be optimized.