ECE 2031 Proposal

Jane Lee, Rachel Mittal, Apuroop Mutyala, Austin Rowland

Introduction

We are creating a SCOMP peripheral that gives the user control over specific aspects of a NeoPixel LED string. With our peripheral, our user is able to:

- Control individual pixels in a NeoPixel string
- Set a 24-bit or 16-bit color for an individual pixel
    - Set a 16-bit color for ALL pixels simultaneously
- Have pixel colors move up the string from a starting pixel

Additional Functions

We, as a team, also plan on implementing extra features to our peripheral alongside the aforementioned functionalities. These new features include:

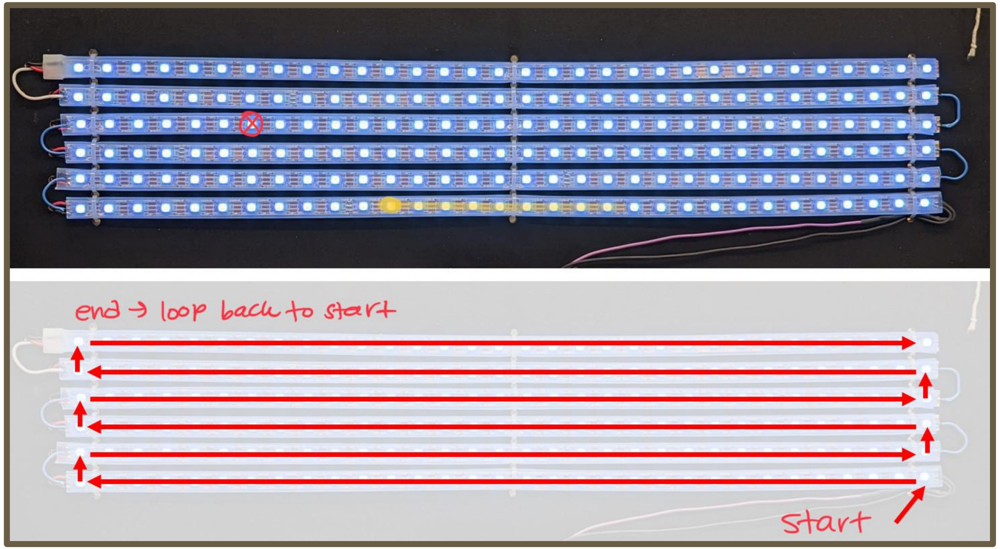
- Allowing the selected color to create a wave pattern
- Creating a gradient with selective colors
- Implementing a Cyclone game in assembly that features different functionalities and features of our product

Cyclone Game

The game we plan on implementing is a version of the popular arcade game: **Cyclone**. The game's functionalities will include:

- Moving and controlling LEDs
- Matching LEDs to a certain point
- Implementing a looping LED pattern
- Controlling the color of LED pixels in the assembly file
- Implementing win/lose conditions

Example of Cyclone game



Technical Approach- Base functionalities

- Control up to 256 Pixels
- Set a 24-bit color for any pixel
- Set a 16-bit color for any pixel using a single OUT
- Simultaneously set all pixels to a 16-bit color with a single OUT
- "Address auto-increment" functionality

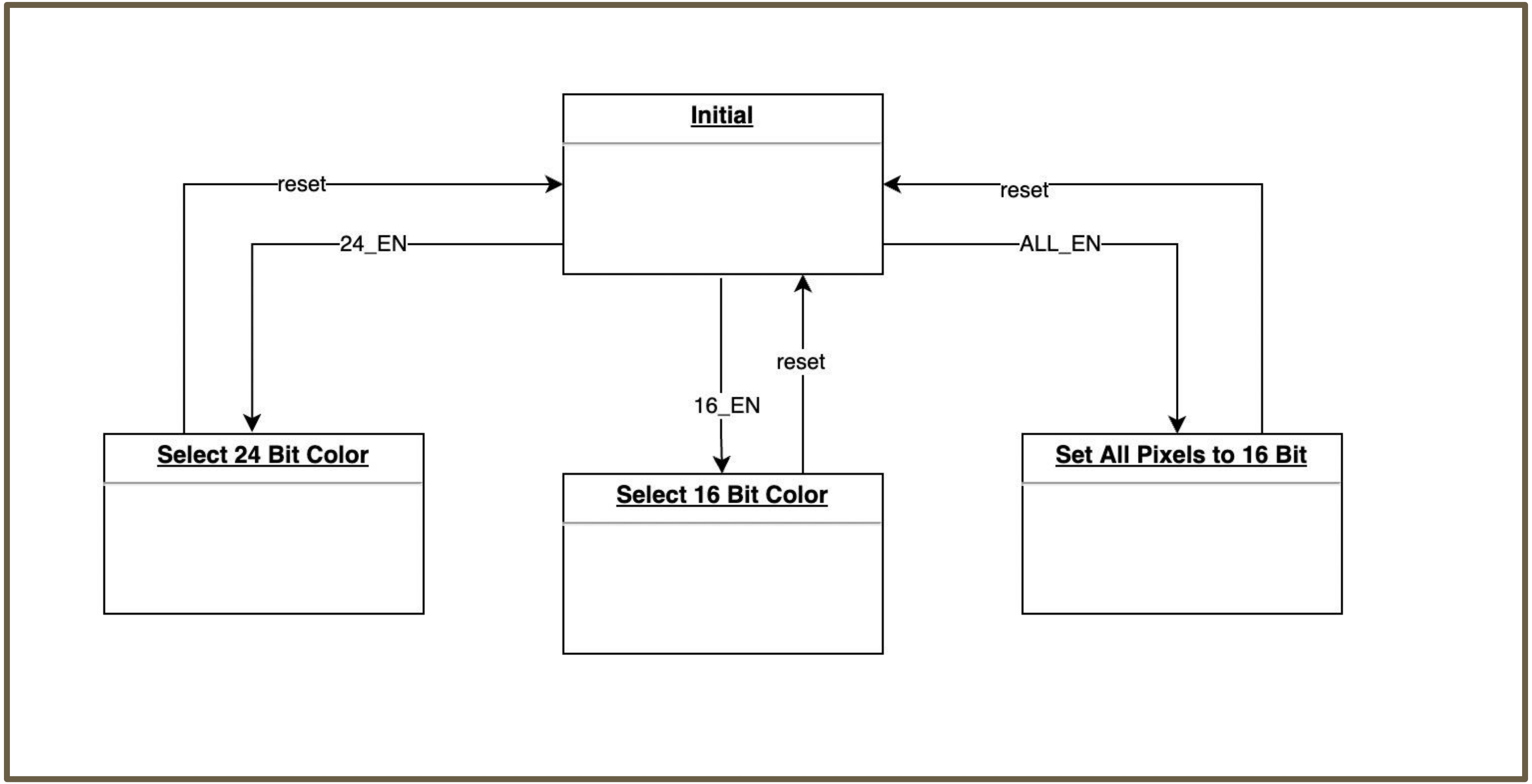
Implementation and Reasoning

We are implementing different states or “modes” to switch between different functionalities.

We are planning on implementing:

- Initial state to select mode
- States for setting either a 24 or 16 bit color to one of the 256 pixel addresses
- State for sending a 16-bit color to all pixels at once

UML Diagram of Base Functionality States

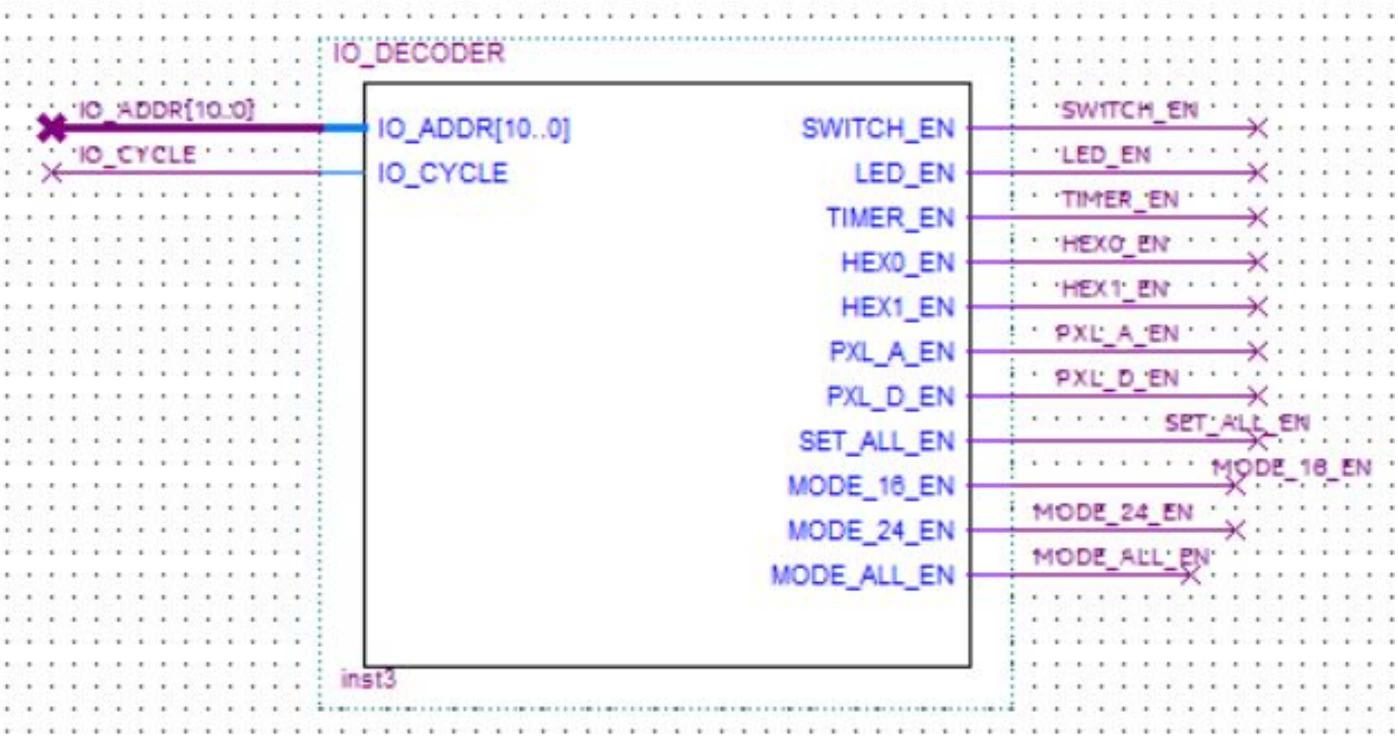


Specifications of State Machine

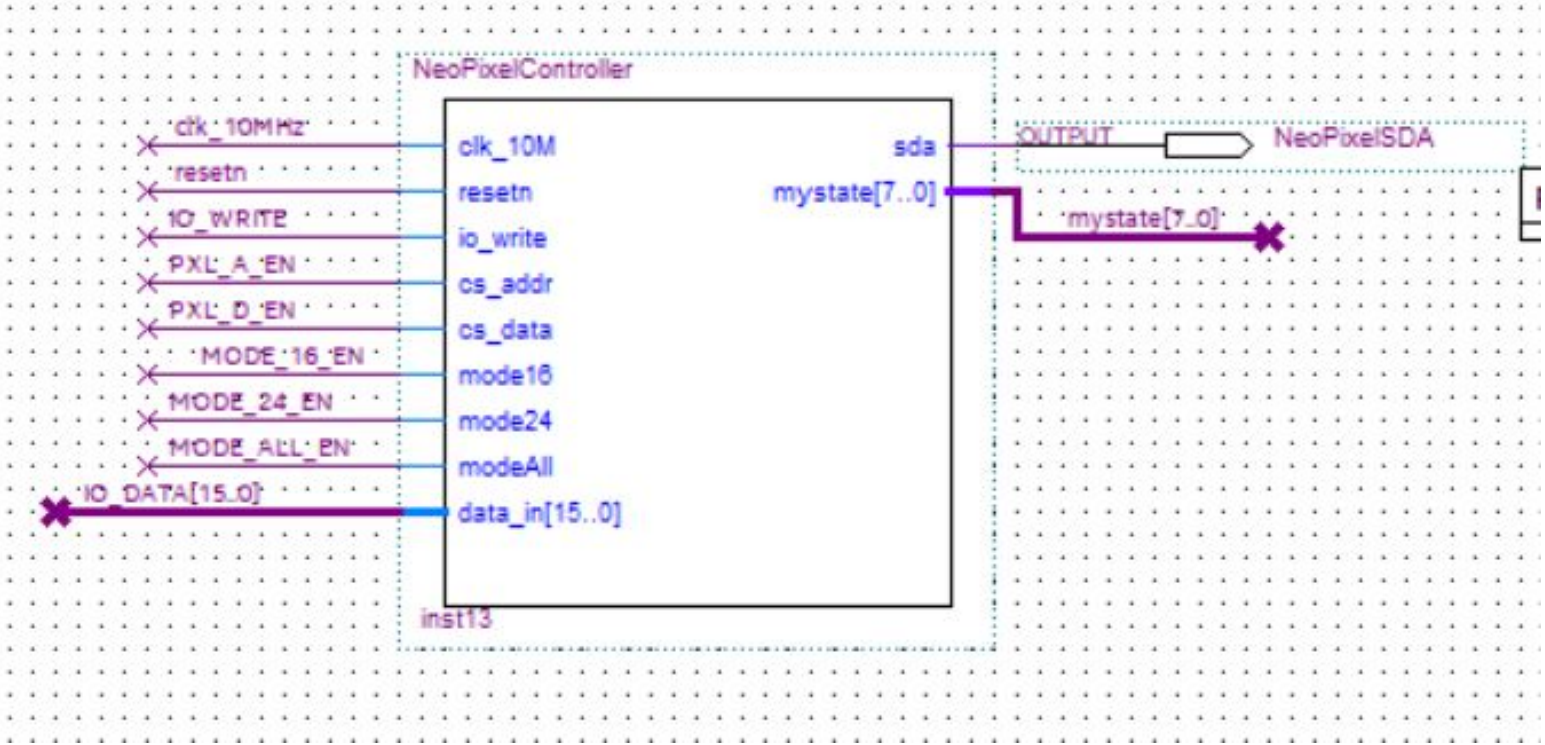
We are modifying the NeoPixelController.vhd file in order to implement our different states. We have also modified our SCOMP and IO_Decoder in order to account for our additional variables.

We chose this way because implementing a state machine was the most easiest and most intuitive way for our group to handle the problem.

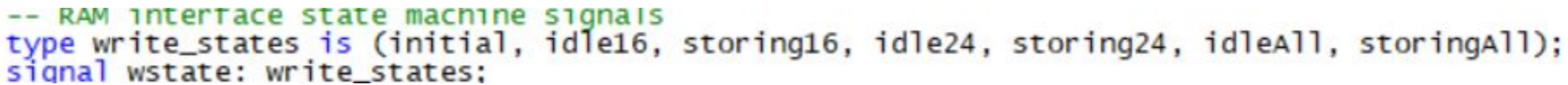
System Explanation



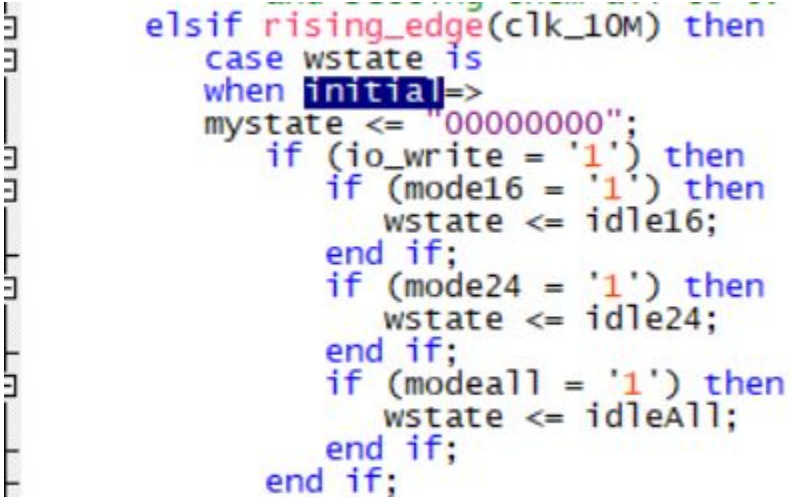
System Explanation



System Explanation- States



```
-- RAM interface state machine signals
type write_states is (initial, idle16, storing16, idle24, storing24, idleAll, storingAll);
signal wstate: write_states;
```



```
elsif rising_edge(clk_10M) then
   case wstate is
   when initial=>
   mystate <= "00000000";
      if (io_write = '1') then
         if (mode16 = '1') then
            wstate <= idle16;
         end if;
         if (mode24 = '1') then
            wstate <= idle24;
         end if;
         if (modeall = '1') then
            wstate <= idleAll;
         end if;
      end if;
```

System Explanation- 24 Bit Color

Implemented using 3 different OUT statements

- One vector for each red, green, and blue
- Once red is written to, then records green, then blue
- Once blue is recorded, outputs to pixel

System Explanation- Outputting to all pixels

Implemented using auto-increment feature

- Auto-increment simply implemented through same method as Lab 8
  - After storing, increment address
- Increment through pixel addresses quickly for all 256 and set each to same color

System Explanation- Wave formation

User inputs a color, then:

- Certain number of pixels will turn on
- Counter will increment with a delay to turn the first pixel in the series off and the next pixel on
- Loops until the user ends it
- Features 16 or 24 bit color and pixel control

System Explanation- Gradient

User selects a color and:

- Auto-increment feature will be used to increment through pixels
- Adjusts the color values by a certain number each time to make gradual change
- Features auto-increment and 16 or 24-bit color control

XXXXX XXXXXX XXXXX

XXXXXXXX XXXXXXXX XXXXXXXX

System Explanation- Cyclone Game

Implement using auto-increment feature and a timer

- Keep one pixel on, increment through pixel addresses with a timer delay and turn each one on and off, then freeze when a button/switch/input is signalled and check if it’s the correct one
- Features pixel control and auto-increment feature

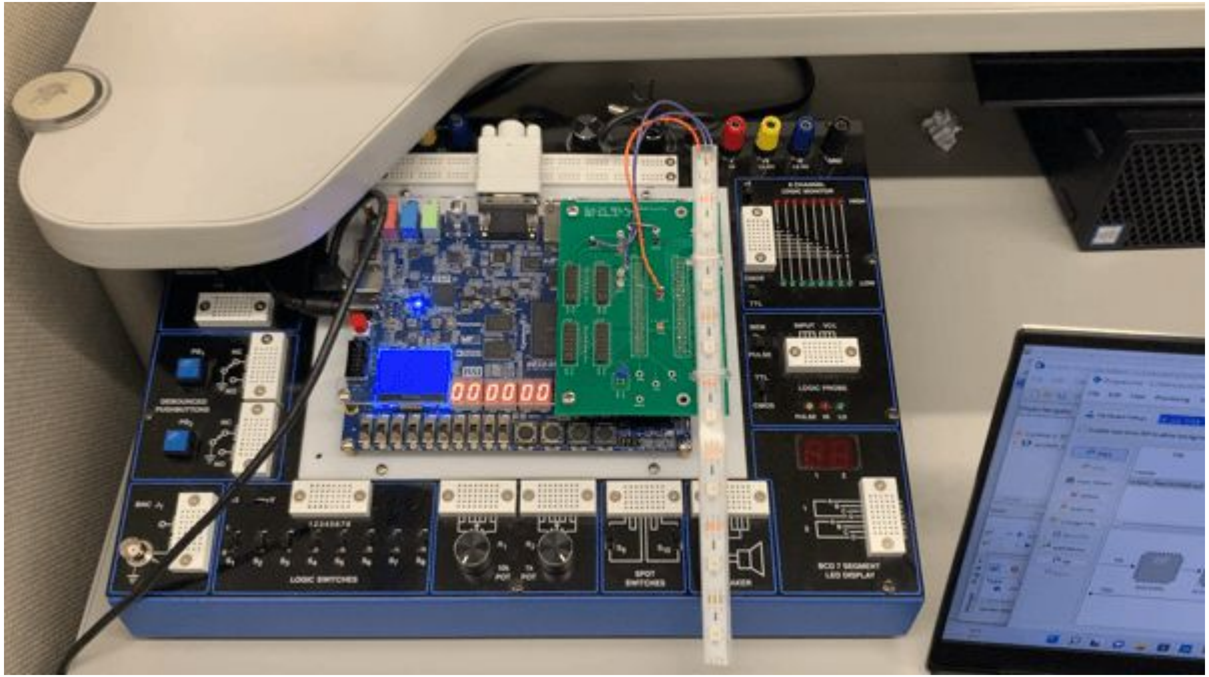
Feasibility

We are confident we can complete this project on time and effectively because:

- Previous experience in labs
- Progress in coding the current base functionalities
- The technical plans we've created for the additional functionality build off of base functionality
    - instead of introducing completely new concepts

Feasibility

Feature Demo: Being able to control any one pixel's color using an rgb565 color input that is provided using the switches.



Feasibility

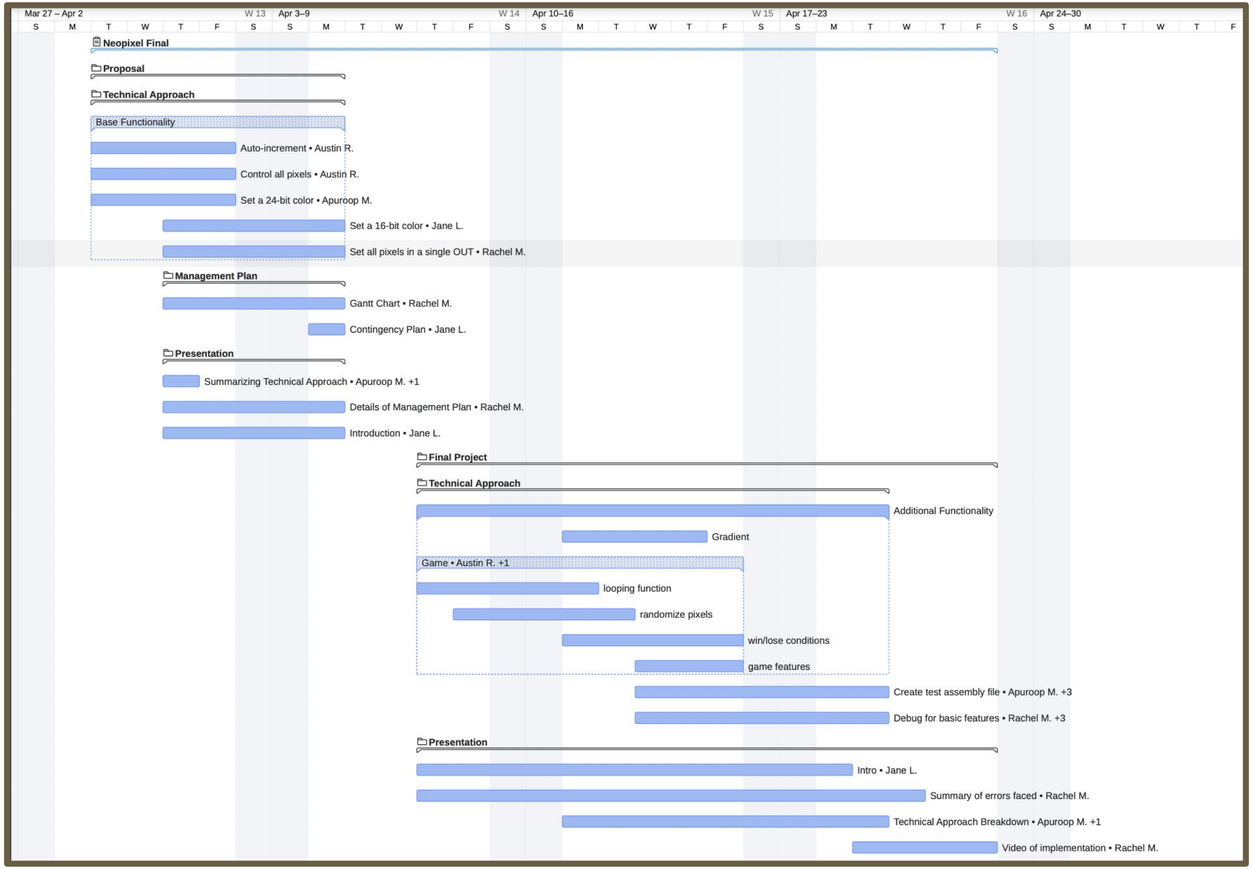
Feature Demo: Testing the auto-increment for pixel addresses. Determined this feature works in conjunction with the single color feature, except the auto-increment continues to set all pixels to the same color.

Demonstration of Peripheral Plan

We will be using our additional functionalities in order to demonstrate working function of our peripheral.

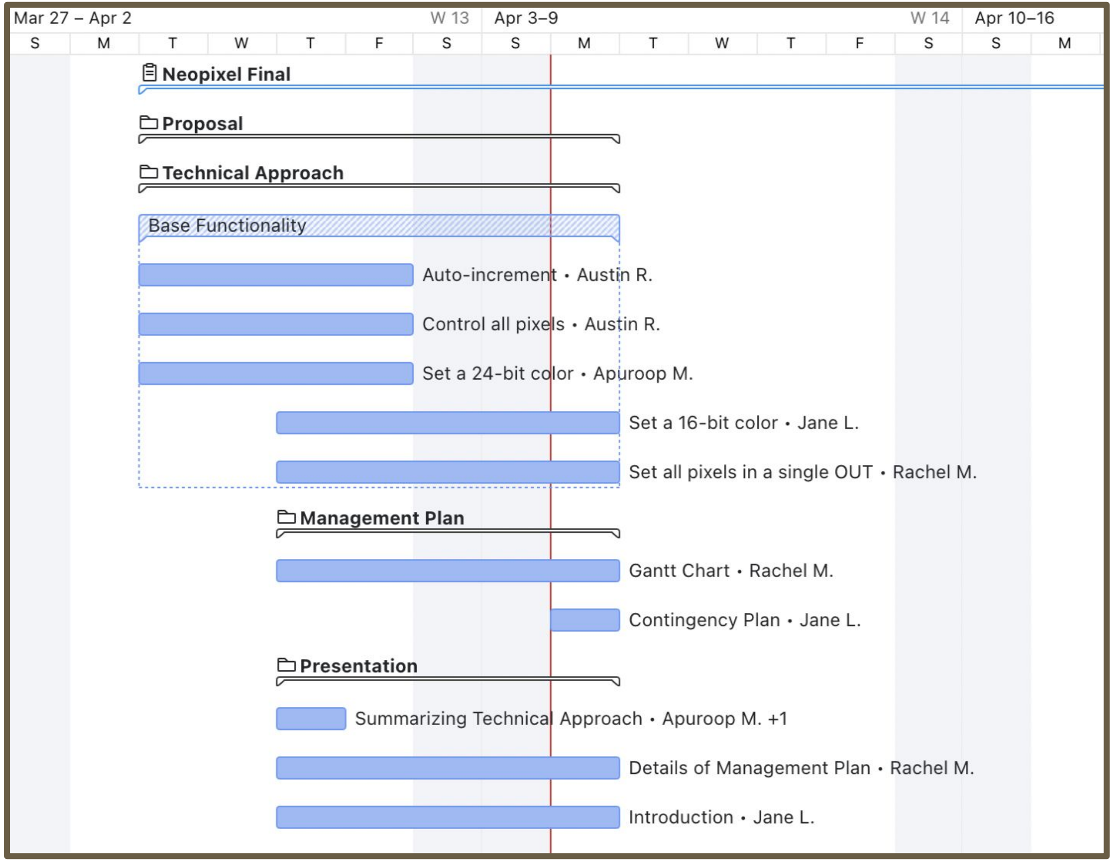
- Our gradient will demonstrate the peripheral’s ability to set any 16-bit color to any selected pixel
- Our Cyclone game will demonstrate the peripheral’s abilities to set a 16-bit color to a pixel and the ability to control any pixel (by automatic movement and manual selection)

Gantt Chart



Mar 27 – Apr 2
W 13
Apr 3–9
W 14
Apr 10–16
W 15
Apr 17–23
W 16
Apr 24–30
S M T W T F S S M T W T F S S M T W T F S S M T W T F S S M T W T F
Neopixel Final
Proposal
Technical Approach
Base Functionality
Auto-increment • Austin R.
Control all pixels • Austin R.
Set a 24-bit color • Apuroop M.
Set a 16-bit color • Jane L.
Set all pixels in a single OUT • Rachel M.
Management Plan
Gantt Chart • Rachel M.
Contingency Plan • Jane L.
Presentation
Summarizing Technical Approach • Apuroop M. +1
Details of Management Plan • Rachel M.
Introduction • Jane L.
Final Project
Technical Approach
Additional Functionality
Gradient
Game • Austin R. +1
looping function
randomize pixels
win/lose conditions
game features
Create test assembly file • Apuroop M. +3
Debug for basic features • Rachel M. +3
Presentation
Intro • Jane L.
Summary of errors faced • Rachel M.
Technical Approach Breakdown • Apuroop M. +1
Video of implementation • Rachel M.

Proposal Timeline



Neopixel Final

Proposal

Technical Approach

Base Functionality

Auto-increment • Austin R.

Control all pixels • Austin R.

Set a 24-bit color • Apuroop M.

Set a 16-bit color • Jane L.

Set all pixels in a single OUT • Rachel M.

Management Plan

Gantt Chart • Rachel M.

Contingency Plan • Jane L.

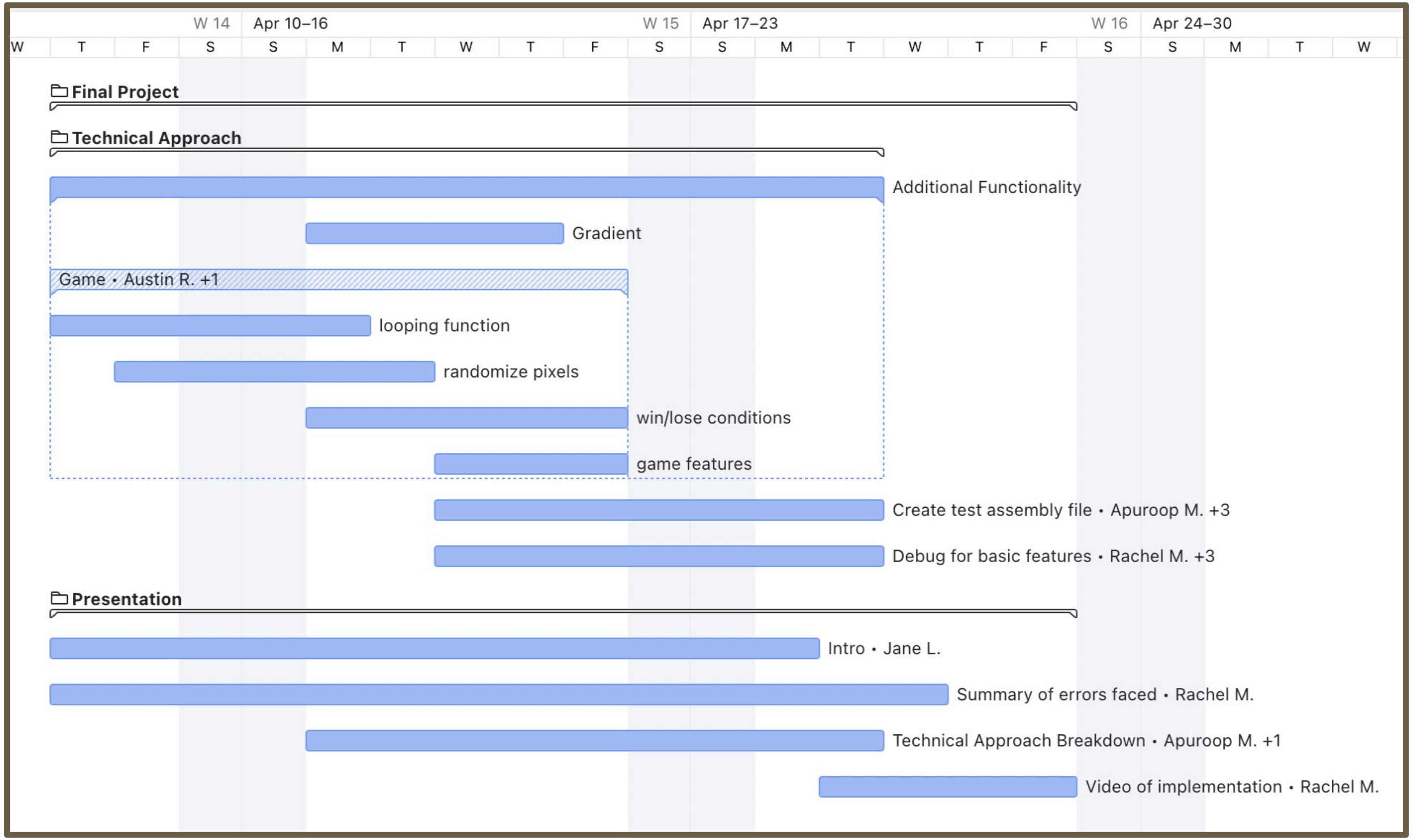
Presentation

Summarizing Technical Approach • Apuroop M. +1

Details of Management Plan • Rachel M.

Introduction • Jane L.

Final Project Timeline


W 14
Apr 10–16
W 15
Apr 17–23
W 16
Apr 24–30
W T F S S M T W T F S S M T W T F S S M T W
Final Project
Technical Approach
Additional Functionality
Gradient
Game • Austin R. +1
looping function
randomize pixels
win/lose conditions
game features
Create test assembly file • Apuroop M. +3
Debug for basic features • Rachel M. +3
Presentation
Intro • Jane L.
Summary of errors faced • Rachel M.
Technical Approach Breakdown • Apuroop M. +1
Video of implementation • Rachel M.

Contingency Plan

We will implement a contingency plan in case certain aspects or goals of our project become unattainable due to time constraints or other reasons.

- **Cyclone Game** -> We will implement another game that is similar to Cyclone, but does not implement looping and/or matching.
- **Gradient** -> We will implement a gradient for fixed colors rather than user-selected colors. We could also present a completely random gradient that would at least show control over 16-bit colors.
- **Wave Pattern** -> We will use another pattern as a demonstration of our peripheral. Examples include an alternating pattern, a looping pattern, etc.

Other Contingencies

Other contingencies-

- If we don’t finish necessary tasks by the end of a week/meeting
  - Distribute/assign them to finish by the next meeting
  - Update Gantt chart and timeline
- If someone is unable to make a meeting
  - Let the group know ahead of time
  - Give any work they’ve accomplished and a written explanation of it or commented code
- If we’re unable to get an additional functionality working
  - Pivot or simplify

Conclusion

Our project:

- Is feasible
- Satisfies the customer’s needs
- Allows ease of use and customizability(API)
- Implements creative additional features
- Includes a detailed plan and contingencies