Flask Application Controlled LED Music Visualizer

CS 121

Nick Auerbach, Carson Kurtz-Rossi

May 11th, 2021

Table of Contents

Introduction

Inspired by classic audio visualizers, this project will use a Raspberry Pi to make LED strips pulse to sound or display pre-programmed light shows depending on the preference of the user. The ideal application is to provide a light show while listening to music where the greater the volume, the more LEDs in the strip turn on. Additionally, at different volumes the LEDs turn on with different colors to alert the user to the increased music intensity. For example, at low volume the LEDs pulse blue while when the volume increases they change to red. The user can adjust the color and brightness settings of the LEDs and even choose different lighting modes besides audio visualization via a Flask web application. This gives the lights utility in any setting or mood.

The Python-based Flask web application framework is home to the project implementation. The device can be placed anywhere with access to a power outlet and hopefully with a speaker nearby. The goal is to provide users with a versatile and playful light show that responds to the sounds in their environment.

Definitions, Acronyms, and Abbreviations

Acronyms

LED - Light Emitting Diode
SPI - Serial Peripheral Interface
ADC - Analog to Digital Converter
GPIO - General Purpose Input/Out

Definitions

Flask - Python web application framework
LED Strip - A length of interconnected and in this case individually programmable LEDs
Logic Level Converter - Integrated circuit that converts 3.3V signals to 5V signals

Project Detail

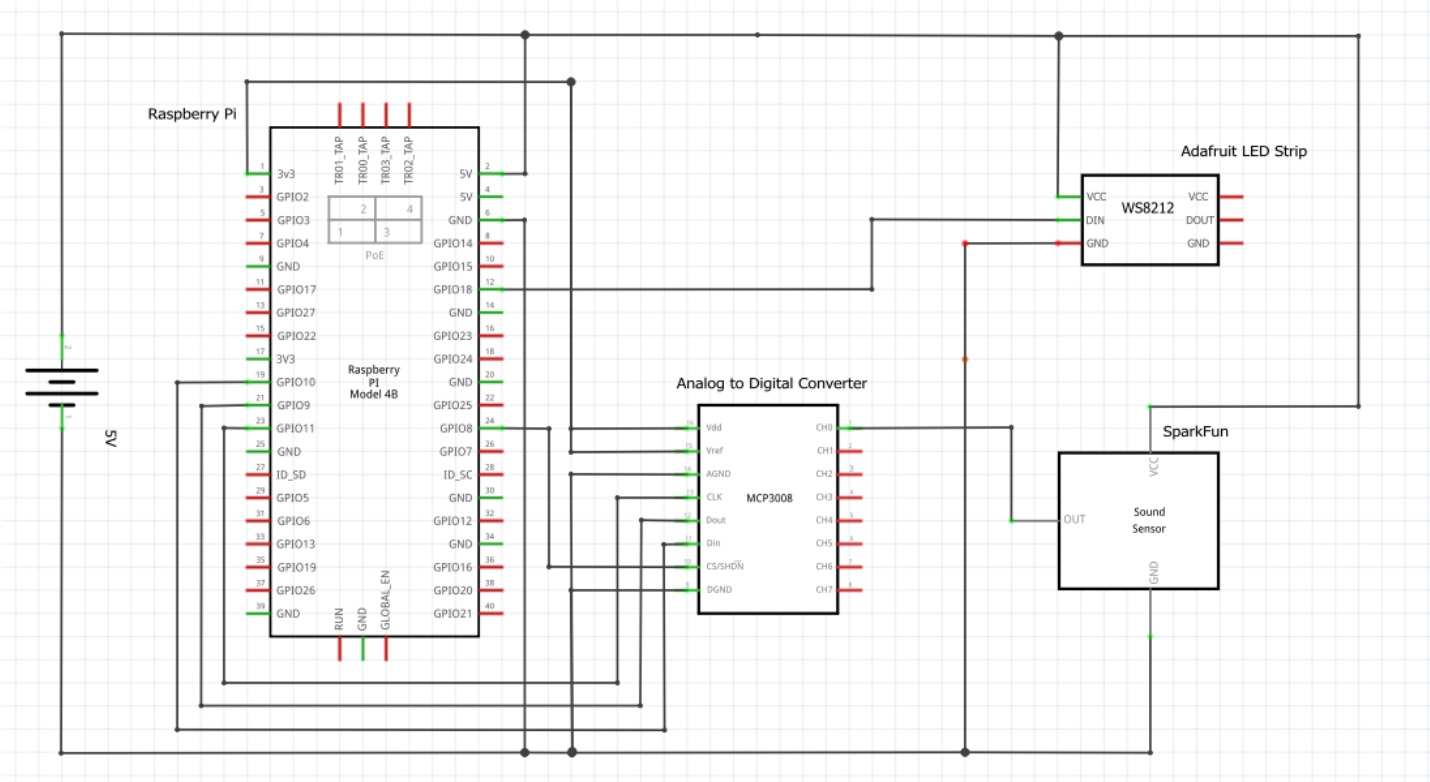


Figure 1 - Circuit Schematic

Figure 1 depicts the project schematic. The Raspberry Pi is used to control multiple LED strips based on inputs from a sound sensor. The output from the sound sensor is a 0V-5V analog signal. Due to the fact that the GPIO pins can only read digital signals an MCP3008 ADC chip is used to provide a digital input to the Raspberry Pi. The spidev Python library allows the Pi to communicate with an SPI device like the MCP3008. The Pi mappes the signal from the sound sensor to a number of LEDs on each LED strip. A higher voltage is mapped to more LEDs so the more of the strip lights up when the sensor detects higher volumes. Although the Raspberry Pi is powered via the 5V GPIO pin in the schematic this was just done to show that there is power to the Pi. In reality, the Pi is powered via the standard power cable and a USB C port.

The GPIO pins output 3.3V but the LED strip expects a 5V digital input signal. When the report was originally written the team expected that this would make it difficult for the Pi to communicate directly with the LEDs and that the 74AHCT125 logic level shifter would be necessary to step up the GPIO 3.3V signal to 5V. Upon testing the lights it became clear that the logic level shifter was not necessary so it was omitted from the final design. Only one LED is depicted in Figure 1 but the prototype includes five strips of 20 LEDs each. This equates to 1.67m of LEDs that draw at most 18W (3.5A @ 5V) of power per meter. This is far too much current for the GPIO pins so an

external 5V 10A power supply was originally recommended to power the LEDs. The final design only uses a 5V 2A power supply because the team already had one available. Some brightness might have been sacrificed, but it was not easily noticeable because the LEDs were not run at max power. Displaying white light requires the most power, but the programmed light modes look much better with more interesting colors so white light was rarely used.

There are five different programmed modes for the lights: Random Mode, Snake Flash, Strobe Party, Illuminate, and Music Mode. Music Mode creates the audio visualization described above. Illuminate lights up all lights on the strip. Strobe Party quickly pulses all lights on, then off. Snake Flash turns all the lights on, one at a time and then turns them off one at a time. Finally, Random Mode lights up the random pixels on each LED strip. The user is able change the LED colors and brightness as well as display mode through the use of a Flask app. Similar to the simple implementations in the CS121 labs, a combination of HTML, JavaScript and Python is implemented to detect button selections on the application that will be different light settings. There is also a color wheel from which the user can select a variety of LED colors. The Flask app is implemented with two JavaScript files titled dashboard.js and colorpicker.js. dashboard.js acts as the event handling for the buttons. colorpicker.js overlaid on a canvas if the user selects the preview button. The user can then drag the cursor to the desired color, and the lights will change to those RGB values. The RGB values are also displayed on the color picker. An HTML file, dashboard.html, includes a cascading style sheet for the sidebar of buttons, as well as formatting for the titles and colorpicker. colorpicker.js sends a JSON list of RGB values to the colorpicker app route in the Flask app, app.py. Global variables for light mode and color allow all functions to access the necessary data to pass to the LEDs through app.py. There are on and off routes for all five of the light mode buttons.

Budget Analysis

Projected

Total Project Budget: $1,958

Total Materials Cost: $78

Total Labor Cost: $1880

Material	Cost	Purchased From	Date Purchased	Date Arrived	Purchaser
Sound Sensor	$6	SunFounder	02/24/21		Carson Kurtz-Rossi
AD/DA Converter PCF8591	$11	SunFounder	02/24/21		Carson Kurtz-Rossi
LED Strip	$56	Adafruit	02/27/21		Carson Kurtz-Rossi
Power Adapter -2.1mm jack to screw	$2	Adafruit	02/27/21		Carson Kurtz-Rossi
2x 74AHCT125	$3	Adafruit	03/26/21		Carson Kurtz-Rossi

Role	Task	Hours	Hourly Rate	Cost
Supply Chain Manager	Ordering materials	1	$30	$30
Software Engineer	Flask web application implementation	20	$60	$1,200
Electrical Engineer	Circuit construction	5	$100	$500
Project Manager	Status Updates	5	$30	$150

Actual

Project Budget Breakdown:

Labor: $3,572.50

- Supply Chain Manager: 1 hours x $30 = $30
- Software Engineer: 21 hours x $60 = $1,260
- Electrical Engineer: 20.5 hours x $100 = $2,050
- Project Manager: 7.75 hours x $30 = $232.50

Materials: $78

Total Cost: $3,650.50

Analysis:

(($3,650.50-$1,958)/$1,958)x100 = 86.44

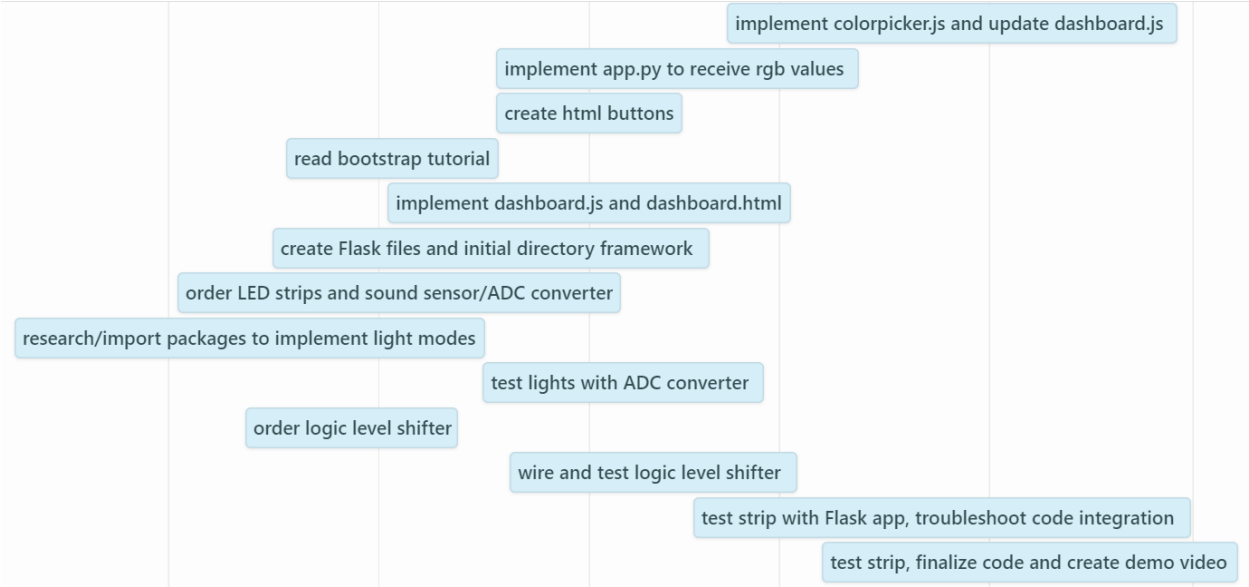
Results: 86.44% over budget

Project Plan

Roles

Project Manager: Nick Auerbach
Duties: Oversee project progress, compile/submit status updates
Supply Chain Manager: Carson Kurtz-Rossi
Duties: Research/purchase necessary materials
Software Engineer: Nick Auerbach
Duties: Implement Flask Web Application
Electrical Engineer: Carson Kurtz-Rossi
Duties: Construct Raspberry Pi LED visualizer

Updated Project Timeline



Group Contributions and Reflections

Nick

Contributions

I focused on the user interface aspect of this project, namely the Flask app implementation using JavaScript event handling and HTML and CSS to provide an appealing dashboard from which the user can select their illumination preferences. This process included considerable HTML and CSS research, as I had no prior exposure to web development other than the labs in CS121. I implemented the event handling for the color wheel, as well as the buttons, ensuring that background processes in the Flask app would not interfere with the completion of the Flask app route calls. I also implemented some simple testing to ensure that the correct data and data types were accessible for Carson to implement the different light modes. Finally, I was the project manager, compiling status updates, timelines, and budgeting, as well as finalizing and submitting our proposal and report.

Reflections

This project pushed me to improve my implementation mind as I had little exposure to HTML or JavaScript. I had a ton of fun exploring tutorials and model implementations and brainstorming how to best construct the interface that we wanted. There were some interesting challenges that came with integrating Carson's implementation. Most notably, the Flask app routes would not complete as there are infinite while loops in the light mode implementations. In order to resolve this and make the button on/off flags functional, I had to include a timeout request that ended the route call 1 second into the infinite while loop. This allowed the while loops to run and the buttons to be functional simultaneously.

This type of debugging while we integrated our systems was the most fun part of the project, and I really enjoyed working through my logic with Carson trying to troubleshoot our problems. I certainly learned how fun and rewarding it can be to troubleshoot problems outside the scope of normal coursework and rely on external documentation and tutorials to guide the implementation of a project.

Carson

Contributions:

I researched and purchased the hardware components that were necessary for this project. I also did the wiring relating to the hardware. This included cutting the LED strip into five sections and soldering each to a prototyping board, along with a power cable connector. I wired the analog to digital converter chip and the sound sensor on a breadboard and made all the necessary connections to interface between the Raspberry Pi, the breadboard, and the protoboard. On the software side I

programmed the different light modes, most notably the Music Mode that makes the lights pulse to voltage signals from the sound sensor.

Reflections:

Overall, I really enjoyed working on this project and I was very pleased with the final outcome. I thought that Nick and I did an excellent job splitting up the work with him focusing on the Flask app and me tackling the hardware and programming of the lights. Nick and I worked independently on each part so it was really great to see what he had come up with once we put everything together towards the end of the semester. I had a vision at the beginning of the semester that I wanted the final product to be something cool that I would actually use as a type of decoration in the future so it was really exciting to see how well it turned out in the end. The lights react to changes in music volume/vibractions really accurately in real time which makes them really great to have as some kind of accessory for a dance party.

While it was convenient for me and Nick to work on aspects of the project separately, the most challenging part was getting everything to work when we had to bring our two halves together. This was where most of the troubleshooting took because we both had to make adjustments to our code to ensure everything worked together correctly. My takeaway relating to this is that in the future I should schedule more time for final build of team projects like these because there can be unforeseen issues that might throw off the schedule.

Project Demonstration and Open-Source Code

Demonstration:
https://www.youtube.com/watch?v=hMBnvxyTZK4

GitHub Repository:
https://github.com/nickauerbach/LED-Music-Visualizer

Target Market

A wide range of music listeners might appreciate this LED visualizer and accompanying application. However, the most likely target demographic is young music listeners who will utilize this visualizer in social settings such as parties and clubs with music genres such as hip hop, rap and trap. The product does not require any connection to music devices, so it is compatible with any speaker on the market. The LED music visualizer is also capable of providing general ambient lighting.

References

Aqib, M. (2020, January 22). Raspberry Pi Analog SENSING: Mcp3008 Raspberry PI Interfacing. https://electronicshobbyists.com/raspberry-pi-analog-sensing-mcp3008-raspberry-pi-interfacing/

Bootstrap 4 Tutorial. (n.d.). https://www.w3schools.com/bootstrap4/

DiCola, T. (2014, September 12). NeoPixels on Raspberry Pi. https://learn.adafruit.com/neopixels-on-raspberry-pi/raspberry-pi-wiring

Hunt-Walker, Nicholas. (2018, April 02). An introduction to the Flask Python web app framework. https://opensource.com/article/18/4/flask

NeoPixels on Raspberry Pi. (2014, September 12). Adafruit Learning System. https://learn.adafruit.com/neopixels-on-raspberry-pi/python-usage

Prikaznov, A. (2012, October 11). Creating Your Own HTML5 Colorwheel. Dzone.Com. https://dzone.com/articles/creating-your-own-html5