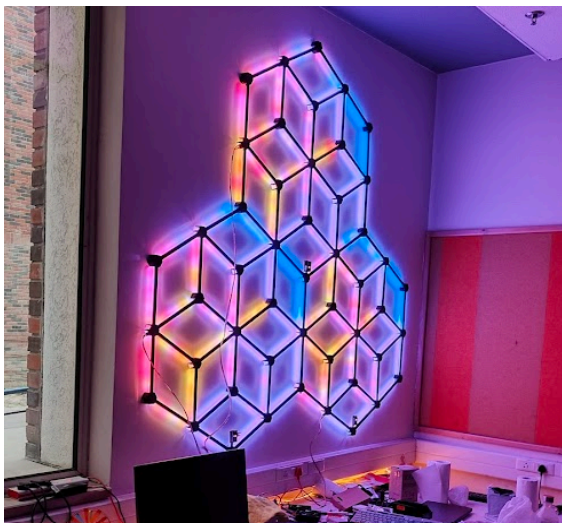
Rapid Prototyping and Experimentation - I
ISM: CS - IS - 3070 - I
Professor Debayan Gupta
With Sai Khurana
Madhav Gupta
4 May, 2024 (Spring ‘24)

LED Graph Communication Layer(s) - Final Project Report

Github Repository
https://github.com/xSpectre9/Rapid-Protopying-and-Experimation-I
This contains final build code as well as an ‘Archive’ folder that showcases some code as part of the journey.

Background
The Hexagonal LED project has made significant progress but has been on hold due to the lack of a software layer that can send Pranav Iyengar’s ISM/Capstone work to the physically built hexagons. While he drew out software simulations of a graph algorithm (Dijkstra's, for instance) and there was a corresponding hardware module built for it as well, the two did not work synchronously.



Three-Hexagon-LED-Array

Touch-Sensor-LED-Hexagon

Aim

(1) To write software that is integrable into Pranav’s code that can transfer any software-generated pattern to the physical LED modules. This includes:
  (a) The three hexagon build displaying any light pattern generated on the led-graph-mono codebase.
  (b) The touch sensor fitted hexagon build displaying any light pattern generated by the led-graph-mono codebase.

(2) To detect a touch input on the physical touch sensor hexagon and send those touch sensor inputs to a python script. This would be used in the two-way communication layer between the hexagon and Pranav's codebase.

Journey Mapping

Here I am going to be describing all the tasks I undertook (highlighted with learning objectives) in order to complete the aim of the project.

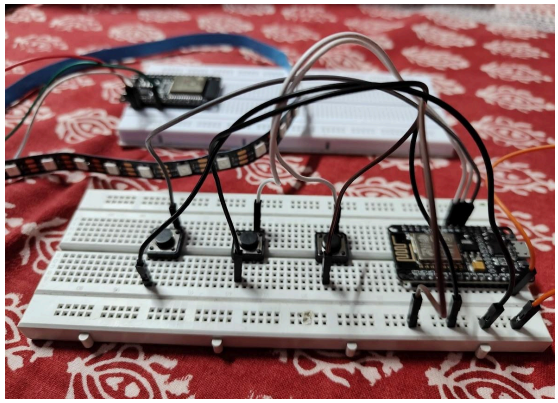
Pre-work:

I started by learning the basics of hardware. To speed up the learning process, I was undertaking a task-based approach to get me up to speed on the basics of the hardware build so that I can start writing the requisite software. The tasks were given and moderated by Sai and Deepraj. Here is a list of things I had to learn and implement as pre-work:

1. Breadboards, jumper cables and microcontrollers (Arduino Uno, ESP32 and ESP8266). My specific task with these was to light up a strip of NeoPixel LEDs via Arduino IDE. The learning objective was understanding how NeoPixels are programmed, and how they are represented in an array.
2. Servers and Client Requests - I had to figure out how server-client calls work, using Spotify's API and RestAPI. The learning objective here was how a python script can host a server, and how it can pull and edit information from a hosted database.

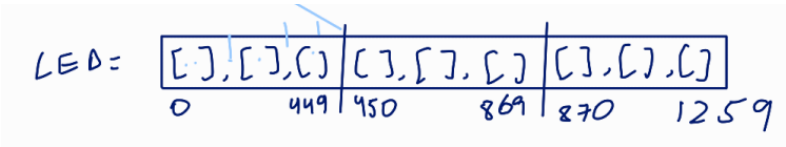
Making 2 ESPs communicate:

This was the final pre-learning task before I jumped into the final project. It involved reading three push button inputs connected to one ESP, and then having another ESP light up an LED strip based on which button was pressed.



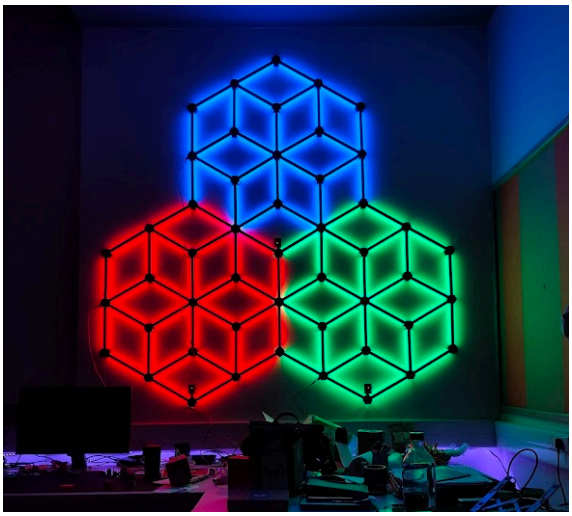
Make the first hexagon light up using a python script:

This was the major part comprising the communication layer. Going into technical detail, the wiring of the three hexagons has been done to make them one contiguous array of length 1260. Here, one unit of the array contains a tuple, which is the [R, G, B] values that constitute one single pixel on the NeoPixel strip, shown below:



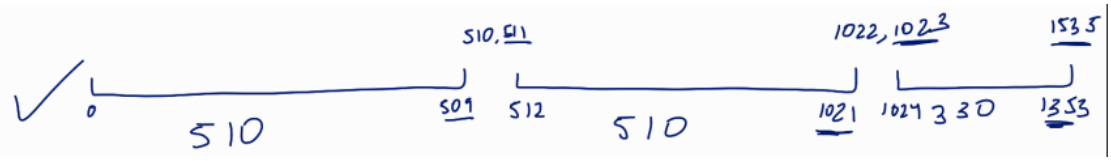
In the figure above, 0 to 449 represents the amount of LEDs being controlled by the first 1 ESP. One of the tuples in this contains the [R, G, B] values for the pixel. There were three such ESPs that controlled the entire three-hexagon module and had to be run at the same time.

WLED, or the software that the ESPs are loaded with that control the light strips, read inputs one DMX universe at a time. One DMX universe is constituted of 512 DMX channels, and three DMX channels represent the [R, G, B] values of one pixel.



This meant that for lighting up just the bottom left hexagon red, I had to devise an algorithm that:

1. Sections the first 450 LEDs into three universes (slicing)
2. Adds the RGB values correctly into the three universes
3. Make sure to not to write any extra LED values into the same universe (padding).

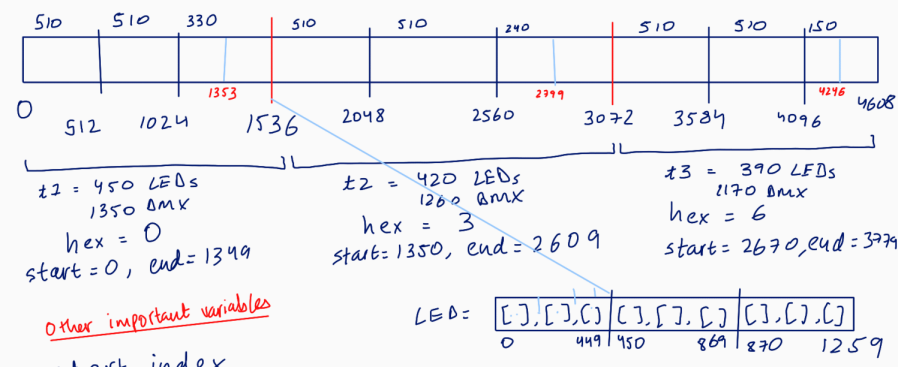


The figure above represents the universe-split of the first 450 LEDs (the bottom left hexagon). Since 512 (how many channels one universe sends) isn't divisible by three, the last couple of elements of the universe had to be padded. Extra care also had to be taken to ensure that the last universe being sent to the ESP only has RGB values until where the first hexagon ends.

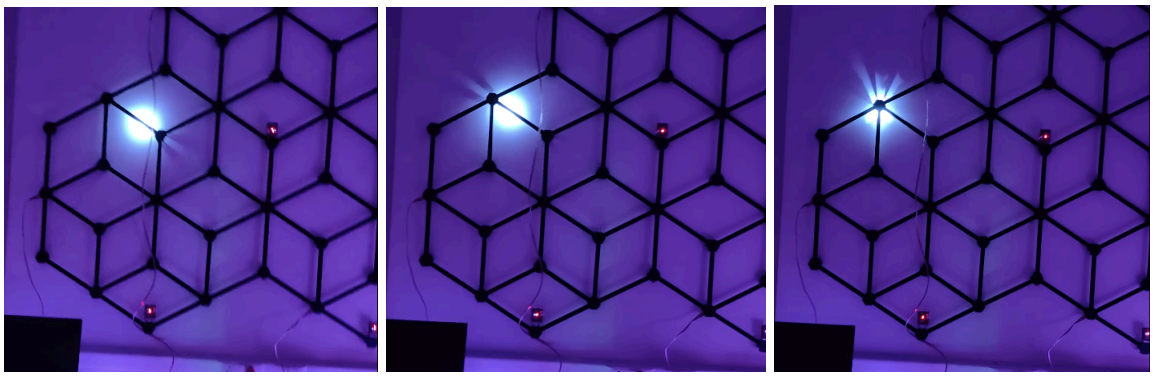
After a lot of iterations of writing logic into code, I finally got to the stage where I could control the lighting on the bottom left hexagon. This was functionally identical to the touch sensor hexagon, which also has 450 LEDs. However, when I tried to test if my algorithm worked, the lights were glitching out quite terribly. I realized that I need to implement multithreading into my

code, wherein one thread was responsible for continuously sending data to the WLED loaded ESP, and the other for reading through the tuples array and writing its contents into universes. The codes for this can be found in the Archives folder in my GitHub repository.

Send universes to the entire three-hexagon module:
After implementing this, I had to scale my code up to light up all three hexagons. This involved being able to read the entire 1260 LED array, running three threads (each for one hexagon) that each break down the array to their respective universes and then transmit the data.



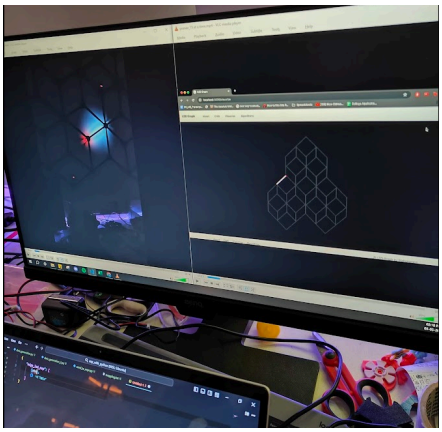
After a lot more troubleshooting and array manipulation, I managed to get all three hexagons lit up. However, until now, these were only displaying static effects. To be able to display a dynamic effect, it had to be able to read multiple frames. Using iterating through the entire three hexagon module as an example, I had to generate and then read a text file that has 1260x1260 rows, with 1260 rows representing one “frame” (or one pixel being lit up).

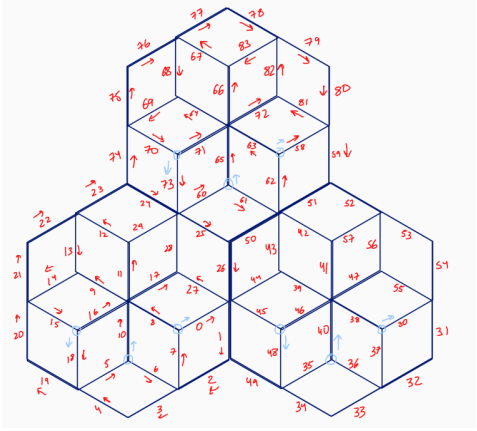


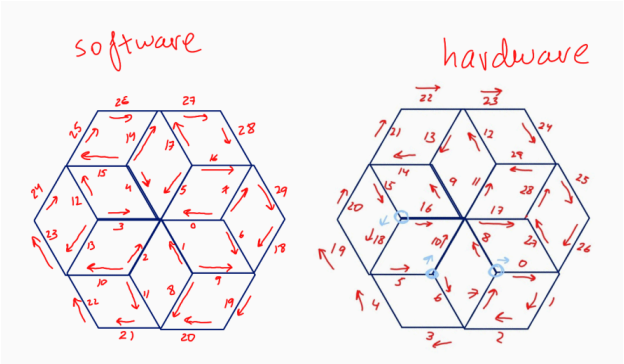
After this was done, my code could finally handle taking a text file input that has the entire pattern of what pixel lights up when, and lighting them up, universe-by-universe. Sai Khurana and Kanishk Singh were instrumental in giving guidance and support through all the troubleshooting steps and helping me with designing the architectural code changes.

Actual mapping:
The source of these text files is being generated by led-graph-mono, the codebase created by Pranav Iyengar as part of his ISM converted Capstone Project. Now, the first pixel that lit up in the software simulation didn’t equate to the same first pixel lighting up in the actual hardware. This meant that a 1:1 mapping had to be done that equates the first LED in software to the first

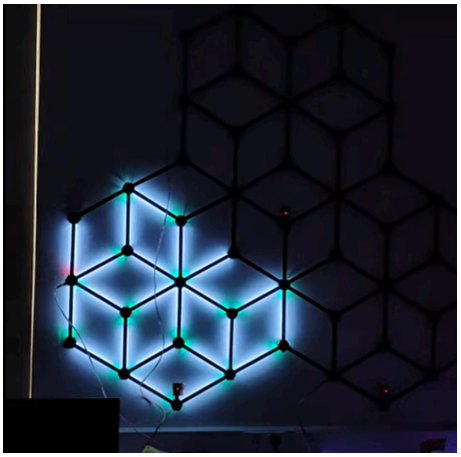
in hardware. Doing this 1260 times was going to be painstaking, so I devised a method that lights up one edge (15 LEDs) at a time, and notes the direction vector of the edge as well.

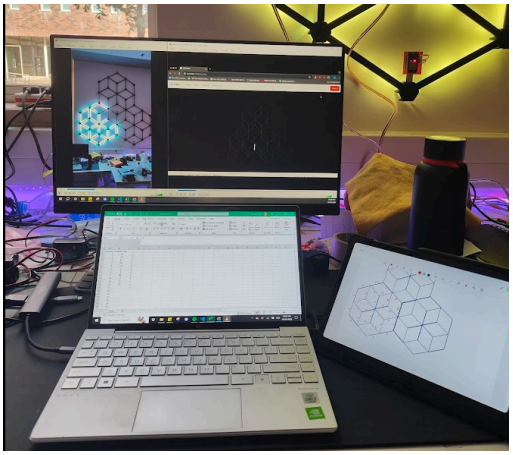






By lighting up one edge at a time, I only had to map 84 edges, which had to be done painstakingly - one edge and direction at a time. I created a JSON file that mapped each hardware edge to match the software ones, for both the three hexagon modules as well as the touch sensor hexagon module.

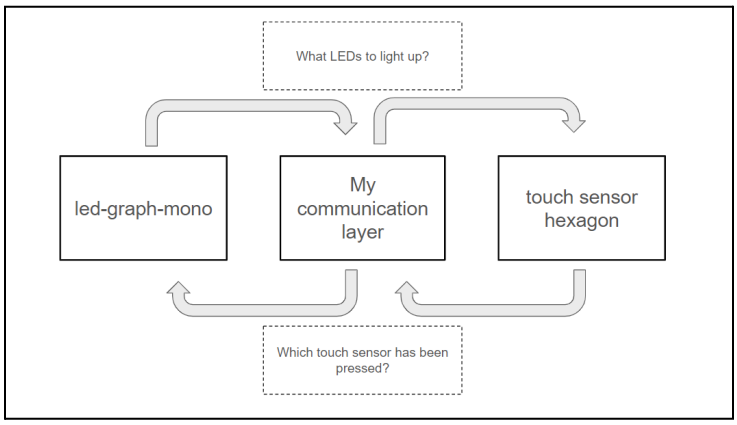




I had tried using excel at first, but switched to JSON for the completeness of programming. After the mapping fun was completed, the lighting portion of my project was done. My code now has

to be integrated into the text files generated by led-graph-mono to display the visual effects (which it can)!

ESP32 touch sensor inputs and mapping:
The final part of my ISM is writing a python script that completes the separate two-way channel between the touch sensor hexagon and led-graph-mono.



Channel 1 is sending what lights have to be lit up. Channel 2 is detecting which of the 19 touch sensors have been pressed on the graph. The lighting part of this channel was completed above. I now had to be able to read which touch sensor is pressed on the hexagon (an ESP-32 is loaded with the code that detects and sends this via a server it is hosting). I learned the important skills of soldering and crimping while setting up this part of the process.

```
gachihyper@LAPTOP-LGBUR42Q:~/Rapid Prototyping and Expermination I$ /bin/python3 "/home/gachihyper/Rapid Prototyping and Expermination I/Touch Sensor Hexagon/touch_sensor_ser
ver.py"
Response from ESP32: No sensors touched
Response from ESP32: No sensors touched
Response from ESP32: No sensors touched
Response from ESP32: Sensor 17 touched
No sensors touched
Response from ESP32: No sensors touched
Response from ESP32: Sensor 2 touched
No sensors touched
Response from ESP32: No sensors touched
Response from ESP32: No sensors touched
Response from ESP32: No sensors touched
```

The python script reads which sensor has been pressed and makes a note of it, giving the information to led-graph-mono to then read the data and transmit the corresponding lighting effect.

Major Challenges, Pitfalls and Set-Backs

(1) Technical Proficiency – as an economics major who has only done computer science in the capacity of a minor subject, a lot of concepts in programming and systems designs were unknown to me. This meant it often took me more effort to understand and learn some basics to implement and complete the project.
(2) Time Commitments – as I was working into an already existing project, I had to be cautious of both mine and other stakeholder time commitments. I often needed material and understanding from said stakeholders which resulted in delays.
(3) I was unable to run the led-graph-mono codebase on my own machine, which meant that I have to wait beyond the time-scope of the ISM report to start mapping the physical touch sensors to what is being simulated in led-graph-mono.

Results and Accomplishments

(1) The codebase can break down any LED pattern generated into universes and display them on the corresponding physical hexagon.
(2) A 1:1 mapping of all software LED pixels to hardware pixels has been finished. It has been done for both the 1260 LED three-hexagon module and the 450 LED touch-sensor hexagon module.
(3) A python script can read which of the 19 touch sensors have been pressed on the touch-sensor hexagon, transmitted by the ESP-32 attached.

Next Steps

(1) Touch sensor mapping.
(2) Performance profiling: Any generated effect displayed by my mapping code will not be as smooth as something like what WLED can natively do. This requires meticulous performance optimization to ensure that all the components of the core script are running smoothly.
(3) Improvement on functionality: the codebase needs careful intervention to fit into another codebase that wants to transmit to the hexagons. When considering the future scope of the application, which is scaling beyond one or three hexagons, it needs to be made more modular and easy to read.