Traffic Cube

Daniel Saghbini & Michael Nyugen

California State University, San Marcos

Abstract

This project emulates a 4-way intersection based on a predetermined traffic light sequence. It consists of a 3x3x3 LED cube to represent the traffic lights of the 4-way intersection, 2 of the 74HC595N shift registers to control the LED cube, 3 of the 100 ohm resistor to prevent LED blowout, and a F401RE NUCLEO board to send serial output to the shift registers and control the sequence of the traffic lights. Determined to build this, we planned out the cube to include red, yellow, and green lights. Each level of the cube holds the respective color of a traffic light. So all around, you have the bottom layer acting as a green light (ours is a red light), the middle layer as a yellow light, and the top layer as a red light (ours is a green light). Designating the sides of the cube as north, south, east and west, we simply look at the middle and red LEDs, no matter the level, to determine the lane. So facing the appropriate side, we get to see that the middle LED is the only left turn lane and the right LED is the lane that lets the driver go straight and turn right. Now we gave it a predetermined traffic sequence to turn on the respective lights assuming where the cars are waiting. And because of this, we achieved an embedded system that is efficient and effectively designed to capture all the lights that can be on in unison.

Introduction

The aim of this project is to apply our knowledge of embedded systems within the scope of creating a traffic light system. The relevance of this project can be found in how real life traffic lights work. In order for a traffic light system to be successful, it had to have efficient throughput of traffic while not causing any motor accidents. Drawing up the schematics of how this would work, we formed a simple diagram detailing the paths that have to be blocked when a certain light goes green to prevent crashes. The expectation at a traffic light is your green light is a guaranteed safe time to go.

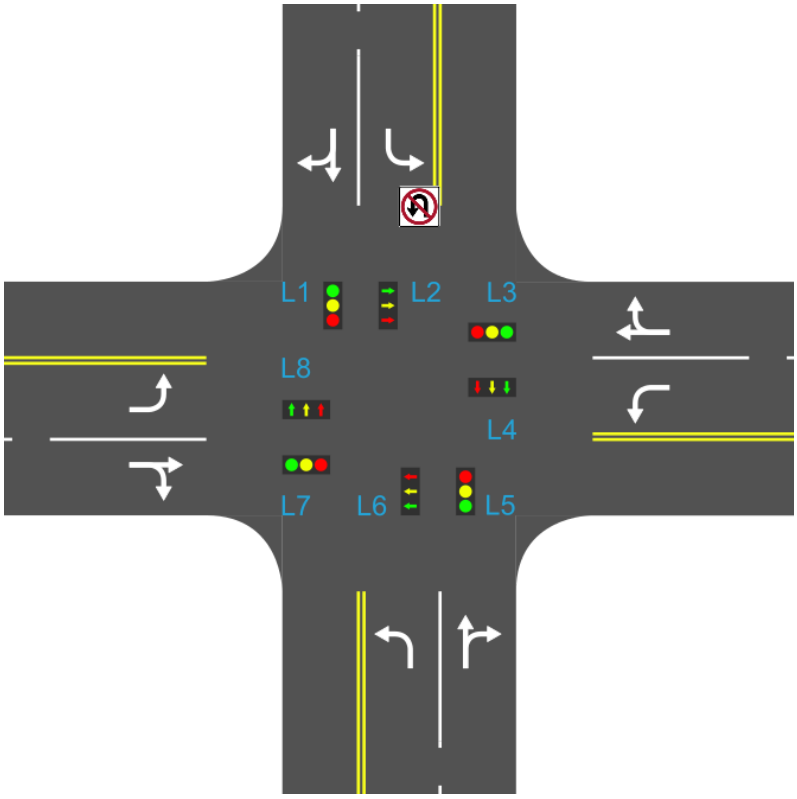

Figure TL1 - The hypothetical layout of the traffic lights in a specific 4-way intersection.

This is relevant because we engage with traffic lights all the time and a cubed form of it can be used for educational purposes in children and can teach them the importance of traffic coordination as a way to build engineering literacy. This would help set up many bright futures in engineering and inspire the next generation of engineers. Many more use cases can be executed depending on the need to simulate a traffic light system.

These uses of the LED cube as a traffic light system, as well as the hardware setup we use, separates it from other embedded system projects that make use of a LED cube. Our LED cube hardware setup is similar to one such project found on the SolderingMind.com website, but that project uses no shift registers, uses an Arduino board, and the code is written in the Arduino Programming Language [1]. Another example of a LED cube project by RGBFreak on the Instructables.com website makes use of transistors, pin headers, and a perfboard to solder all the components on to as well as also using an Arduino board [2]. Unlike those projects, our Traffic Cube lights up for a specific purpose instead of for show, uses 2 of the 74HC595N shift registers, uses the F401RE NUCLEO board, and puts all components on a breadboard.

A F401RE NUCLEO board only has so many pins for LED manipulation, so it would be unwieldy or impossible to connect the 27 digital out pins and the 3 digital in out pins required to control each individual LED of the 3x3x3 LED cube. To reduce the

amount of pins required, we can solder the LED cube in a way that would only need 9 digital out pins and 3 of the digital in out pins to control which column of LEDs and which layer of LEDs can be lit. By controlling the specific column, then controlling the specific layer, we can control the individual LED inside of the cube. This is done by soldering all anodes of a column of LEDs together in series, and soldering all cathodes of a layer of LEDs together in series as specified in [6, Figure CD1]. Through this method, we cannot light multiple layers at once or else a column of LEDs can have multiple LEDs on, which is not useful for the intent of the Traffic Cube project. For the human eye, there exists a phenomenon called the “persistence of vision” in which light lingers or persists for a bit within human vision [3]. We can make use of this phenomenon by quickly flashing the lights of each layer, so that in human vision it appears that all layers of the LED cube have their lights on. Even with the 27 + 3 (30) pins required by the LED cube reduced to 9 + 3 (12) pins connected to the NUCLEO board, it would still be an unwieldy mess of wires and an improper application of what we learned in embedded systems.

To solve this, shift registers can be used to convert 3 NUCLEO pin connections into an effectively 8 pin output as specified by the 74HC595N shift register [4]. However, we need 9 pins to control the 3x3x3 LED cube, so an additional shift register is required. One feature of shift registers is the ability to cascade to another shift register, writing a serial output that can overflow to the next shift register. In the 74HC595N shift register, it is possible to cascade data out of the Q7’ serial output pin by sending a low active signal to the OE output enable pin. We can then send the data from the Q7’ serial output pin of the first shift register to the DS serial input pin of the second shift register [4]. This effectively doubles the shift register output to 16 pins total; of which we only need 9 pins for the LED cube.

The barest order of events within our project is defined in [4, Figure PDL], in which inside of a forever loop, the code will iterate through a sequence of traffic lights and write that sequence to the SPI for the shift registers and LED cube control [5].

```
BEGIN
    Initialize SPI
    WHILE 1
      FOR i < Length of Light Sequence, i++
           Read current Light Sequence
           Convert current Light Sequence to Binary
           Write Binary to SPI
      END FOR
    END WHILE
END
```

Figure PDL - The Program Design Language of how the project would work

To be more specific in the function of our project, we can initialize a traffic light sequence as a 2D array, each element of a row representing a specific light, as in [10, Figure TL2]. Then our program must be able to communicate with the shift registers through MBED's SPI, so the SPI must be initialized [5]. From there, within the forever loop, we can loop through every row of the traffic light sequence. We can pass that specific row sequence to a function that will be able to write to the LED cube what lights are meant to be lit.

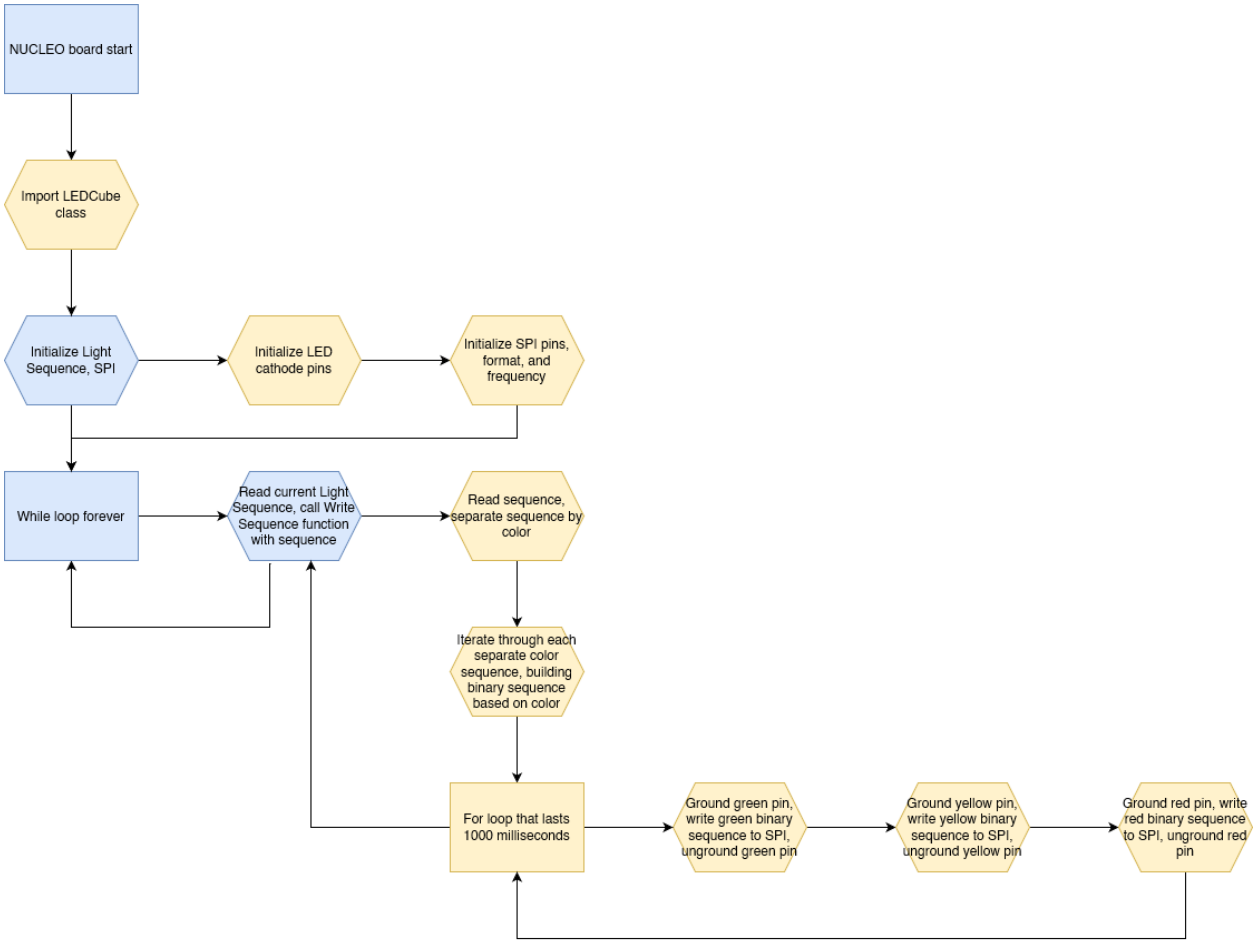

Figure TLF - The top level flow-chart that illustrates the sequence of the code

Implementation

The design for the Traffic Cube is based on a 3x3x3 LED cube connected to 2 of the 74HC595N shift registers and the F401RE NUCLEO board. The anodes of the LEDs are connected in series for each column and are controlled by connecting them to the serial output (Q0 through Q7) pins of the 74HC595N shift registers. The cathodes of each layer of the LED cube are connected in series to 3 separate 100 ohm resistors per

layer and PA_0, PA_1, and PA_4 (A0, A1, A2) pins of the NUCLEO board per layer. As a singular 74HC595N shift register only supports an 8 bit serial output and we require the ability to control 9 pins, 2 of the 74HC595N shift registers are connected so that one cascades the serial output to the other. On both shift registers, the 16 (Vcc) and 10 (MR) pins are connected to a 3.3v power source, the 8 (GND) pins are connected to ground, and the 12 (ST_CP) and 11 (SH_CP) pins are connected to D10 and D11 pins of the NUCLEO board respectively. The first shift register has its 13 (OE) pin connected to the 3.3v power source, its 14 (DS) pin connected to the D11 pin of the NUCLEO board, and the 9 (Q7') pin connected to the 14 (DS) pin of the next shift register. The second shift register has its 13 (OE) pin connected to ground and the 9 (Q7') pin unconnected to anything. A circuit diagram of the hardware setup can be found in [6, Figure CD1] and the actual real life hardware setup can be found in [7, Figure HD1]

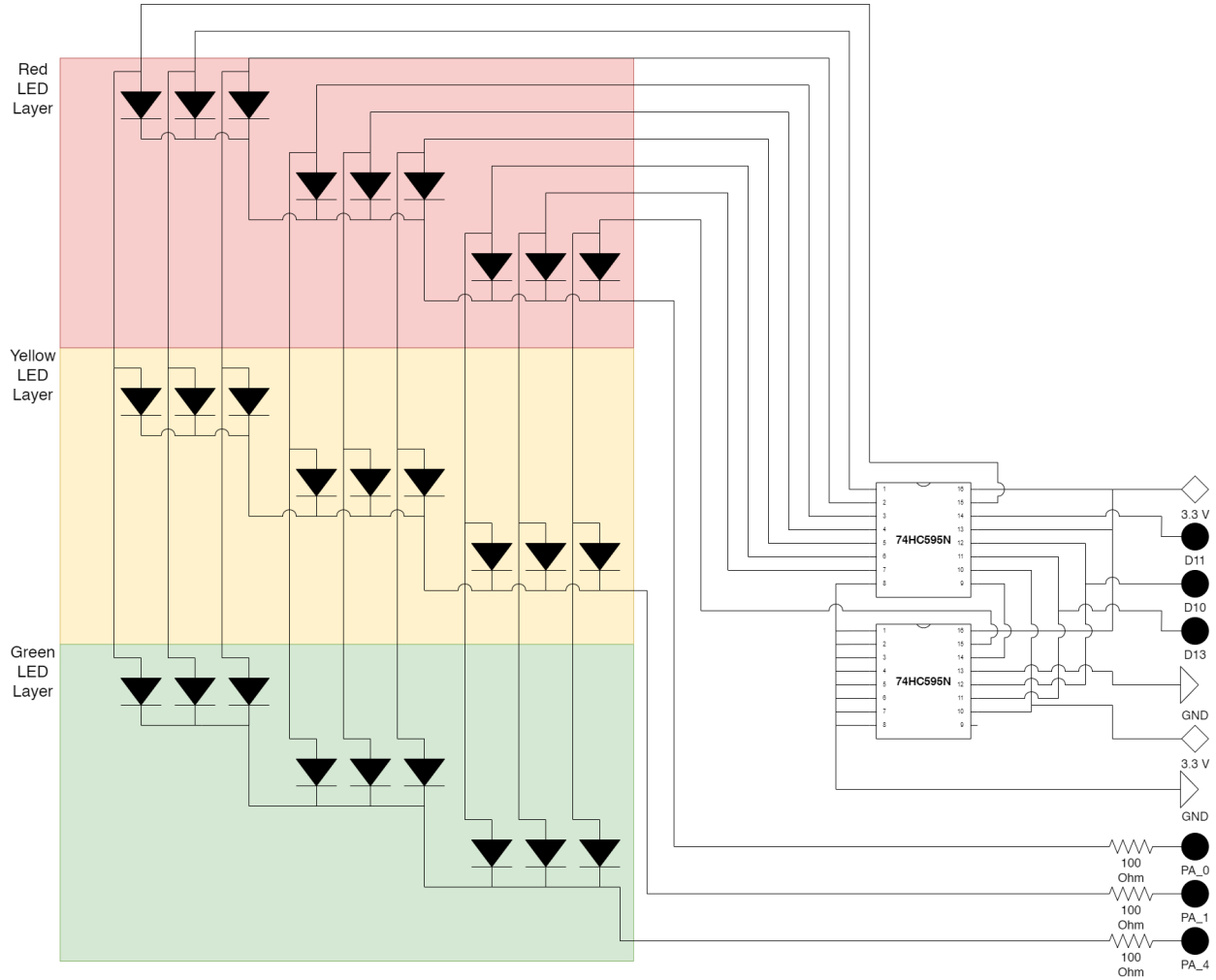

Figure CD1 - The circuit diagram for the Traffic Cube project

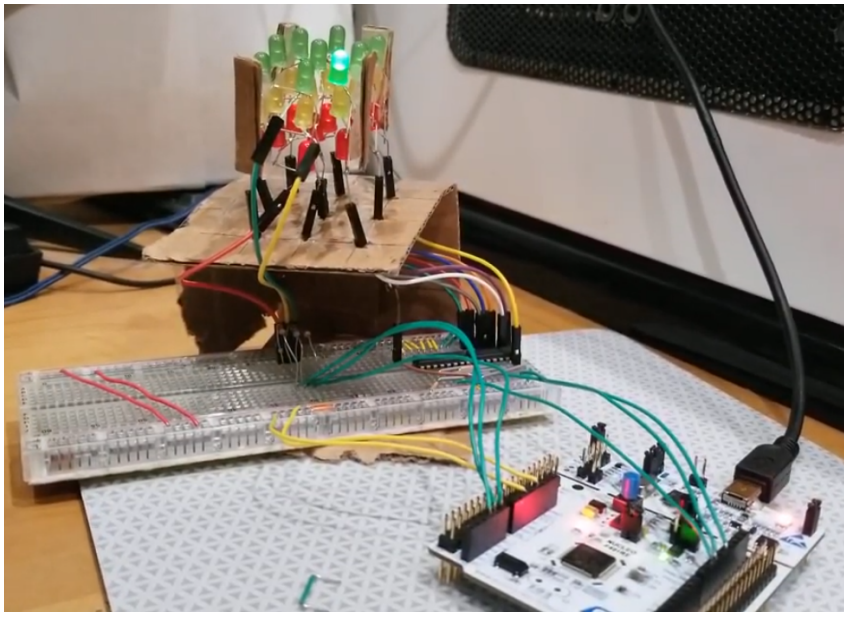

Figure HD1 - The real life hardware setup for the Traffic Cube Project

The requisite supplies for the project that were not provided by the university's NUCLEO kit were 100 pieces of 5mm LED light diodes to be the lights and 20 pieces of 3 inch Female/Male jumper wires to connect the LEDs with the breadboard. The 100 piece set of LEDs contained 20 of yellow, red, blue, green, and white colored LEDs. From those colors, only 9 of the yellow, 9 of the red, and 9 of green were used to create a 3x3 layer for each layer of the 3x3x3 LED cube. In each layer of the LED cube, the LEDs had their cathodes soldered together, while each individual LED of the layer had their anodes soldered to the anodes of the other layers as seen in [Fig CD1]. To align the LEDs properly, they were placed into a 3x3 grid of holes poked into a piece of cardboard so that all LEDs placed into the hole would be evenly distributed with other LEDs of the layer. The cathode wires of the LEDs were bent so that they would connect with the other LED cathodes in a spiral shape. The anode wires were bent slightly so that they could clear the space occupied by the LED below them and connect to the next anode wire in the series. Each color layer was soldered separately then soldered one on top of the other, connecting through their anode wires, creating the 3x3x3 LED cube. It was then mounted on a cube like cardboard construction with the Female/Male jumper wires superglued through holes in the cardboard to act as the connection points of the LED cube. This was so that the LED cube could remain upright and to provide space to make connections to the breadboard and the NUCLEO board. To reinforce the visibility of only certain traffic lights, other pieces of cardboard were cut and applied to the 3x3x3 LED cube using double sided tape to hide the lights from certain directions.

The code is broken up into 3 separate files: the main.cpp that begins the Traffic Cube sequence, the LEDCube.h that holds function prototypes, and the LEDCube.cpp that provides all the functions for manipulating the LED cube. The overall order and

general summary of the code can be found in [4, Figure PDL], and in [5, Figure TLF]. The main.cpp code starts with the importation of the mbed.h library, LEDCube library, and the c++ chrono library. Then it initializes and defines the traffic light sequence within a 24 by 8 length 2D character array as a global variable, with G, Y, and R, in the array representing the green, yellow, and red light respectively. Within the main function, it calls the initSPI function from LEDCube to initialize the LED layer cathodes and the MBED SPI class for the shift register serial output. Then, within a WHILE loop that lasts forever, it starts a FOR loop that loops through the 24 rows of the LED sequence array. Within that FOR loop it calls the writeSeq function from LEDCube, passing the current row of the LED sequence array as the function argument. Within the writeSeq function, it creates 3 separate 9 length arrays that represent what lights are affected by the 3 colors of green, yellow, and red. Each element of the 9 length array represents the lights determined in [1, Figure TL1] and in[10, Figure TL2]. It will use a FOR loop to iterate through every element of the current light sequence row, writing a 1 to the respective element of the color arrays to represent that light being that color at that time, with 0 representing the light being off at that time. As the middle column of the LED cube is unused, the 4th element of each color array is set to 0 so that the middle column would never light up. When each color array has their specific sequence of which LED lights to control, the arrays are then converted to the binary sequence used as output by the shift registers to control the LED lights. The color arrays are iterated through by a FOR loop, turning the array of 1s and 0s to a binary sequence. This is done by using bit shift operations to move the 1 to its specific bit, then adding that to the initially empty binary number. With a binary sequence for each color, a FOR loop that lasts 1000 milliseconds will ground the color layer's specific cathode, flash the color by writing the binary sequence to the SPI for 1 millisecond, unground the color layer's specific cathode, then do the same for the rest of the colors. The cathodes are grounded by defining pins as DigitalInOut, setting them to open drain mode, and setting their output to 0. This puts the pin's potential at zero, grounding the pin [6]. Once the 1000 milliseconds of flashing through the color passes, it returns to the FOR loop in main.cpp and moves to the next row of the light sequence.

Results

In trying to find the perfect system for the Traffic Cube there were several factors to consider: what lights represent what traffic action, what kind of road the traffic lights are for, and the sequence of which lights should be on and which should be off. There were several iterations of what was decided for the Traffic Cube road, such as dedicated turning lanes, 3-way intersections, 2 lanes going both ways, and the lights

representing different directions. The final version of road layout for the Traffic Cube was decided as a 4-way intersection road, with a dedicated left turning lane, and there only being 1 lane for both directions. To determine the sequence of the traffic lights, a logic table was created as seen in [10, Figure TL2]. Each light (for reference see [1, Figure TL1]) would only have a specific color per the moment of the sequence, the moment which will be referred to as a tick. Each tick was decided to be 1000 milliseconds which is 1 second. This gives each light about 3 seconds of being green, with smaller periods of being green when appropriate. If a certain light was green for go, that means certain other lights would not be able to be also green at the same time, such as a left turn light colliding with the approach of an opposite straight turning light. That means that during the creation of the sequence table, every light at that tick had to be checked to make sure it did not cause collisions with other lights. To improve efficiency and throughput of traffic through the lights, it was determined that certain lights could be green at the same time as other lights and allow for brief periods of time for lights that have not been green for a while to be green. This meant that the middle column of lights in the LED cube would remain unused. From this sequence table, an 2D array could be generated and used inside of the project's program to determine the sequence the LEDs light up in.

To create a better understanding of how the SPI serial output would affect the 3x3x3 LED cube during the prototyping phase, a for loop was created to increment through a 16 bit binary sequence as well as writing specific predetermined values to the SPI. During testing, it revealed failures in the hardware setup of the shift registers as certain LEDs would not light up even if the binary sequence was full. These failures led to more experimenting with the hardware setup of the shift registers until the results were what was expected of certain binary sequences written to the SPI. As observed, each bit with the number of 1 in the binary sequence would light up a specific LED in the cube. The bit was mapped to a certain output pin, and adjustments were made to the LED cube hardware connections and project code in order to align their interactions cleanly. From that, we determined the order of which part of the sequence as determined in [10, Figure TL2] affected which LED and the logic to convert that sequence to the proper binary for SPI writing.

Tick	L1	L2	L3	L8	L4	L7	L6	L5
1	G	R	R	R	R	R	R	G
2	G	R	R	R	R	R	R	Y
3	G	G	R	R	R	R	R	R
4	Y	G	R	R	R	R	R	R
5	R	G	R	R	R	R	G	R
6	R	Y	R	R	R	R	Y	R
7	R	R	G	R	R	G	R	R
8	R	R	G	R	R	Y	R	R
9	R	R	G	R	G	R	R	R
10	R	R	Y	R	G	R	R	R
11	R	R	R	G	G	R	R	R
12	R	R	R	Y	Y	R	R	R
13	G	R	R	R	R	R	R	G
14	Y	R	R	R	R	R	R	G
15	R	R	R	R	R	R	G	G
16	R	R	R	R	R	R	G	Y
17	R	G	R	R	R	R	G	R
18	R	Y	R	R	R	R	Y	R
19	R	R	G	R	R	G	R	R
20	R	R	Y	R	R	G	R	R
21	R	R	R	G	R	G	R	R
22	R	R	R	G	R	Y	R	R
23	R	R	R	G	G	R	R	R
24	R	R	R	Y	Y	R	R	R

Figure TL2 - A logic table of what color the lights would be at a certain tick.

Lessons

Project management consisted of the two group members separating project responsibilities and communicating with each other when needed. Michael Nguyen was mainly responsible for the hardware implementation. Daniel Saghbini was mainly responsible for research and project refinement. Both team members contributed to working on the project code, writing the report, and for coming up with the project. For communication, both team members used Discord, an online messaging platform, to

organize and communicate with each other on project matters as well as occasionally meeting in person on the CSUSM campus.

The SPI lab in class was not very effective in teaching the practical aspects of using the MBED SPI class and the 74HC595N shift register. This was due to the hardware setup already being provided, the bulk of the code required being already written and provided, as well as the code being specifically written for the NHD_0216HZ LCD screen [8]. This did not lead to a better understanding of how to practically apply the concept of shift registers to the physical 74HC595N shift register. In order to be successful in this project, the datasheet for the 74HC595N had to be studied as well as experimentation in the hardware setup for the shift register to produce the output needed to control the LED cube. Alternatives in MBED pin functions had to be researched to determine how to control if a NUCLEO board pin is grounded, with the DigitalInOut used for its open drain mode [6].

Our initial challenge was in expanding on the initial idea of an LED cube so that it would have real world use and allow us to apply what we learned in embedded systems. This was compounded by the haphazard formation of the group and the late start in the project. The biggest challenges during this project were bug fixing hardware and software. Due to how intertwined the hardware was with the project code, it was difficult to determine when an issue was from the improper set up of the hardware or the improper implementation of the code. If we had more time to work on the project, we could introduce pedestrian crossing buttons that would modify the traffic light sequence to allow for pedestrians to walk across the intersections.

Summary

By establishing the idea to solder an LED cube and transform it to a traffic system, we were able to successfully wire the cube to the shift register and wire the shift register to the NUCLEO board. Then we took up new knowledge by specifying the commands to the shift register to then output the correct lights when navigating a 4-way intersection. Through it all, we managed to find out how to take something we have learned and applied it to something new. Though we were self-taught, it was worth it in the end.

Bibliography

[1] Admin. (2020, May 20). *Arduino led cube 3x3x3 circuit with code* [Blogpost]. Available: https://solderingmind.com/arduino-led-cube-3x3x3/
[2] JColvin91. (2017, October 15). *3x3x3 led Cube* [Blogpost]. Available: https://www.instructables.com/3x3x3-LED-Cube-2/
[3] J. Ma and D. Bjanes, "Persistent of Vision Display," Dept. Electr. Eng., Cornell University, New York City, NY, 2005. Available: https://people.ece.cornell.edu/land/courses/ece4760/FinalProjects/s2012/jm787_dab355/jm787_dab355/index.html.
[4] Philips Semiconductors, "74HC595; 74HCT595 8-bit serial-in, serial or parallel-out shift register with output latches; 3-state," 2003, June 25. Available: https://www.digchip.com/datasheets/download_datasheet.php?id=86130&part-number=74HC595N
[5] ARM, (2023). *SPI* [API]. Available: https://os.mbed.com/docs/mbed-os/v6.16/apis/spi.html
[6] ST. "STM32 microcontroller GPIO hardware settings and low-power consumption," AN4899 Rev 3, 2022, March. Available: https://www.st.com/resource/en/application_note/an4899-stm32-microcontroller-gpio-hardware-settings-and-lowpower-consumption-stmicroelectronics.pdf
[7] M. Najjar. (2023). Serial Communication [PDF]. Available: https://csusm.instructure.com/courses/22284/files/2625740?module_item_id=1776745
[8] M. Najjar. (2023). LAB 5: SERIAL COMMUNICATION [PDF]. Available: https://csusm.instructure.com/courses/22284/assignments/270997?module_item_id=1776747

Code Listings

main.cpp

```
/* mbed Microcontroller Library
 * Copyright (c) 2019 ARM Limited
 * SPDX-License-Identifier: Apache-2.0
 */

#include "mbed.h"
#include "LEDCube.h"
#include <chrono>
```

```
int lightSeq[24][8] = {
	/*
	Four Way Intersection Light Layout:

	L1 L2
		L3
	L8	L4
	L7
	L6 L5

	Light Sequence Array:
	red = R, yellow = Y, green = G
	{L1, L2, L3, L8, L4, L7, L6, L5} // tick #
	*/
	{'G', 'R', 'R', 'R', 'R', 'R', 'R', 'G'}, // 0
	{'G', 'R', 'R', 'R', 'R', 'R', 'R', 'Y'}, // 1
	{'G', 'G', 'R', 'R', 'R', 'R', 'R', 'R'}, // 2
	{'Y', 'G', 'R', 'R', 'R', 'R', 'R', 'R'}, // 3
	{'R', 'G', 'R', 'R', 'R', 'R', 'G', 'R'}, // 4
	{'R', 'Y', 'R', 'R', 'R', 'R', 'Y', 'R'}, // 5
	{'R', 'R', 'G', 'R', 'R', 'G', 'R', 'R'}, // 6
	{'R', 'R', 'G', 'R', 'R', 'Y', 'R', 'R'}, // 7
	{'R', 'R', 'G', 'R', 'G', 'R', 'R', 'R'}, // 8
	{'R', 'R', 'Y', 'R', 'G', 'R', 'R', 'R'}, // 9
	{'R', 'R', 'R', 'G', 'G', 'R', 'R', 'R'}, // 10
	{'R', 'R', 'R', 'Y', 'Y', 'R', 'R', 'R'}, // 11
	{'G', 'R', 'R', 'R', 'R', 'R', 'R', 'G'}, // 12
	{'Y', 'R', 'R', 'R', 'R', 'R', 'R', 'G'}, // 13
	{'R', 'R', 'R', 'R', 'R', 'R', 'G', 'G'}, // 14
	{'R', 'R', 'R', 'R', 'R', 'R', 'G', 'Y'}, // 15
	{'R', 'G', 'R', 'R', 'R', 'R', 'G', 'R'}, // 16
	{'R', 'Y', 'R', 'R', 'R', 'R', 'Y', 'R'}, // 17
	{'R', 'R', 'G', 'R', 'R', 'G', 'R', 'R'}, // 18
	{'R', 'R', 'Y', 'R', 'R', 'G', 'R', 'R'}, // 19
	{'R', 'R', 'R', 'G', 'R', 'G', 'R', 'R'}, // 20
	{'R', 'R', 'R', 'G', 'R', 'Y', 'R', 'R'}, // 21
	{'R', 'R', 'R', 'G', 'G', 'R', 'R', 'R'}, // 22
	{'R', 'R', 'R', 'Y', 'Y', 'R', 'R', 'R'}  // 23
};
```

```
Ticker ticker;
const chrono::seconds lightTimeSec(1);

int main() {
	initSPI();
	printf("Begin\n\r");
	while(1){
	printf("Loop Start\n\r");
	for(int i = 0; i < 24; i++){
	writeSeq(lightSeq[i]);
	//ThisThread::sleep_for(lightTimeSec);
	}
	}

}
```

LEDCube.h

```
void initSPI(void);
void writeRaw(int data);
void writeSeq(int seq[8]);
```

LEDCube.cpp

```
#include "mbed.h"
#include "LEDCube.h"
#include <chrono>

DigitalInOut green(PA_0);
DigitalInOut yellow(PA_1);
DigitalInOut red(PA_4);

DigitalOut slave(D10);//ds (pin 14)
SPI spi(D11, D12, D13);
//d11 -> st_cp (pin 12), d13 -> sh_cp (pin 11)


void initSPI(){
	green.output();
```

```
	yellow.output();
	red.output();
	green.mode(OpenDrain);
	yellow.mode(OpenDrain);
	red.mode(OpenDrain);
	red = 1;
	yellow = 1;
	green = 1;

	slave = 1;//deselect register
	spi.format(16, 3);//16bit data, hi clock
	spi.frequency(100000);//1MHz clock rate
	slave = 0;//select register
	spi.write(0b0000000000000000);//turn off all LEDs
	slave = 1;
	ThisThread::sleep_for(chrono::seconds(1));
}

void writeRaw(int data){

	slave = 0;//select register
	spi.write(data);
	slave = 1;//deselect register
}


void writeSeq(int seq[8]){
	/*
	Four Way Intersection Light Layout:

	L1 L2
		L3
	L8	L4
	L7
	L6 L5

	Light Sequence Array:
	red = R, yellow = Y, green = G
	{L1, L2, L3, L8, L4, L7, L6, L5} // tick #
```

```
*/
int redIndex[9];
int yellowIndex[9];
int greenIndex[9];
//parallel colors
for (int i = 0; i < 9; i++){
//printf("%c, ", seq[i]);
if (i == 4){//skip middle led
redIndex[i] = 0;
yellowIndex[i] = 0;
greenIndex[i] = 0;
}
else if (i < 4){
switch(seq[i]) {
case 'R':
        redIndex[i] = 1;
        yellowIndex[i] = 0;
        greenIndex[i] = 0;
        break;
case 'Y':
        redIndex[i] = 0;
        yellowIndex[i] = 1;
        greenIndex[i] = 0;
        break;
case 'G':
        redIndex[i] = 0;
        yellowIndex[i] = 0;
        greenIndex[i] = 1;
        break;
default:
        redIndex[i] = 0;
        yellowIndex[i] = 0;
        greenIndex[i] = 0;
}
}
else if (i > 4){
switch(seq[i]) {
case 'R':
        redIndex[i] = 1;
```

```
        yellowIndex[i] = 0;
        greenIndex[i] = 0;
        break;
case 'Y':
        redIndex[i] = 0;
        yellowIndex[i] = 1;
        greenIndex[i] = 0;
        break;
case 'G':
        redIndex[i] = 0;
        yellowIndex[i] = 0;
        greenIndex[i] = 1;
        break;
default:
        redIndex[i] = 0;
        yellowIndex[i] = 0;
        greenIndex[i] = 0;
}
}
}
//printf("\n\n\r");
//build binary write data for shift registers
int redData = 0b0;
int yellowData = 0b0;
int greenData = 0b0;
for (int i = 0; i < 9; i++){
if(redIndex[i] == 1){
//printf("r[%d]:%d\n\r", i, redIndex[i]);
redData += (0b00000001 << (8 - i));// bitshift to get as leftmost as possible
}
if(yellowIndex[i] == 1){
//printf("y[%d]:%d\n\r", i, yellowIndex[i]);
yellowData += (0b00000001 << (8 - i));// bitshift to get as leftmost as possible
}
if(greenIndex[i] == 1){
//printf("g[%d]:%d\n\r", i, greenIndex[i]);
greenData += (0b00000001 << (8 - i));// bitshift to get as leftmost as possible
}
/*
```

```
        printf("%d\n\r", i);
        printf("r:%d\n\r", redData);
        printf("y:%d\n\r", yellowData);
        printf("g:%d\n\r", greenData);
        */
        }
        //printf("Write Data Done\n\r");
        /*
        Ticker tick;
        tick.attach(&lightLED, chrono::milliseconds(1));
        while(1){
        ThisThread::sleep_for(chrono::seconds(2));
        }
        */
        for(int i = 0; i < 1000; i++){
        green=0;
        writeRaw(greenData);
        writeRaw(0);
        ThisThread::sleep_for(chrono::milliseconds(1));
        green=5;

        yellow=0;
        writeRaw(yellowData);
        writeRaw(0);
        ThisThread::sleep_for(chrono::milliseconds(1));
        yellow=5;

        red=0;
        writeRaw(redData);
        writeRaw(0);
        ThisThread::sleep_for(chrono::milliseconds(1));
        red=5;




        }
}
```

Video Link

https://drive.google.com/file/d/1e2O-bvrWwr8zNWOc5ntWjk6rj8jMhh1l/view?usp=sharing