Smart Sports Scoreboard

By

Michael Manning

Max Mitchell

Matthew Rosenbaum

Final Report for ECE 445, Senior Design, Spring 2021

TA: Daniel Vargas

02 May 2021

Project No. 49

Abstract

This project entails the creation and application of a smart sports scoreboard composed through an addressable LED matrix system. The system provides an intuitive and interactive interface, allowing users to view results of any of their favorite teams both quickly and accurately. The scoreboard currently has the capability to support 11 of the major sports leagues across the world so that users everywhere can enjoy the product. Inspiration for the product comes from the increased use of smart devices and the corresponding negative effects of blue light that come along with it. Our hardware and software implementation allows users to feel connected without being connected to their smart devices. Within this report, you can find all details surrounding the creation of our project, our subsystems, and the results

Contents

1. Introduction

1.1 Objective

Smartphones have become a central part of our society in America over the past decade. As of last year, 96% of Americans owned a cell phone, and 81% of the population owned a smartphone [1]. As a result, we have become nearly dependent on using smartphones to obtain information, stay connected to social circles, and communicate. As of 2019, people have been found to check their phones an average of 96 times a day, a number that has increased by 20% over two years prior [2]. Among these occurrences, sports apps are among the most common - a Capgemini study found that 69% of sports fans utilize smartphone technology to enhance their sports following [3]. Although, this great ability has brought with it a few negatives, specifically with the exposure to blue light. Studies have found that an excessive amount of blue light exposure can lead to daytime fatigue and a disruption of one's circadian rhythms. Furthermore, blue light can cause damage to the retina, leading to macular degeneration. The same studies found that children and adolescents are most susceptible to this damage, since their eyes are not as developed [4].

At the start of the project, our team set out to provide fans with a fast and real-time sports experience without the use of a device that emits blue light. It was crucial for us to create such a product while still performing as well as a sports app such as ESPN. We successfully completed this by creating a wall-mounted LED display that is interfaced with a Wi-Fi-supported microcontroller. Using a local network connection, the device can scrape the score and game information from the internet. Additionally, a user interface web application was developed, allowing users to easily choose their favorite teams and customizations.

1.2 Background

Many media companies, such as ESPN, Yahoo!, and FOX, have developed their own sports apps within the last 10 years. These apps allow the user to check the scores or even stream the game directly within the given app. As stated according to [3], 69% of sports followers utilize these platforms to follow their teams. The ESPN app itself has 70 million downloads and 2 million daily active users [7]. At the start of this project, there were no viable products on the market that enable a user to follow their teams outside of an app or website. The functioning prototype of our scoreboard has the potential to change that.

Since the beginning, there was an excellent opportunity to give fans a new method for following their favorite teams because of the ability to utilize an API to grab sports scores.

We have tested our system modulary, checking that information from an actual sports game is being delivered. The result is an end-to-end project that displays information accurately and quickly. With this scoreboard, every sports fan can have a unique and high-quality fandom experience without needing to pick up their phone.

1.3 Physical Product

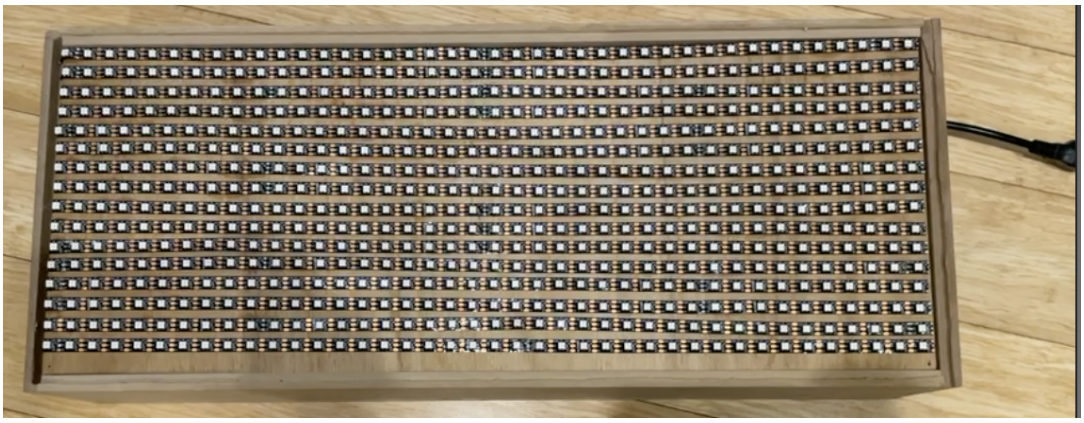

Fig. 1. Front of Scoreboard Appearance

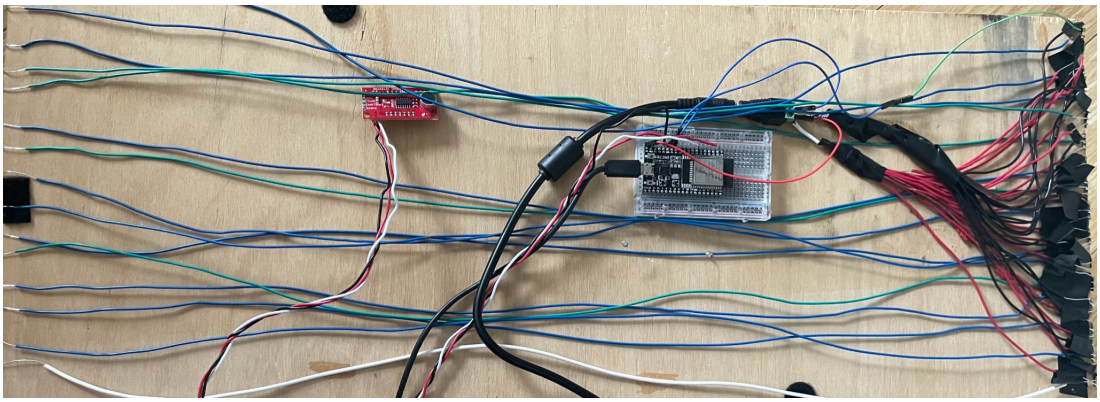

Fig. 2. Physical Design of Back Side of LEDs

Fig. 3. Physical Appearance of Inside of Scoreboard

After talking with the machine shop, the physical design of our scoreboard was determined to be constructed of wood. The LED Display has dimensions of 26”x10” and there is a 1” wood barrier on each side of the scoreboard to allow for access to the LED’s in order to replace or perform work on them. The power supply we used for the device has dimensions of 8”x4.5”x2” and is connected through the back of the device. All components within the device are accessible through the back. With the current design, we have a workspace inside of the scoreboard with dimensions 3 ¾”x10”x26”. Within these dimensions, we have to store the microphone, the microcontroller, and power supply. The machine shop stated that they need to visualize our parts before they can come up with a definitive drawing of how the device will look. Figures 1, 2 and 3 show how the physical scoreboard looks from external and internal points of view.

1.4 High-Level Requirements

- The scoreboard operates at a latency no greater than 30s [11] with a 2.4 GHz band, and it displays the correct score of the game at run-time.

- The scoreboard system will be able to operate up to 15 amps to fit standard electrical outlets in the U.S. The Bluetooth LE protocol must transmit data at a speed of at least 1 Mb/s.

- The scoreboard system must have a UI that is easy to navigate for the user. It must allow the user to select their favorite team and customize LED interfaces using team themes.

1.5 Block Diagram

In our design, the ESP32 acts as the brains of the design, enabling a web interface through Wi-Fi, enough processing power to drive our LEDs and react to noise from the microphone in real time, and a small form factor to fit within our enclosure. Our power system is sufficient enough to power all of our LEDs and the sensors. Our enclosure, while not displayed in the block diagram, is another crucial component. It looks clean while also being light enough to be mounted on a wall. After fully implementing our project, there is not much deviation from the original block diagram. The only difference is that our system does not feature any Bluetooth. It was outside the scope of this course to send Wi-Fi information via Bluetooth, as this would require the development of an Android application.

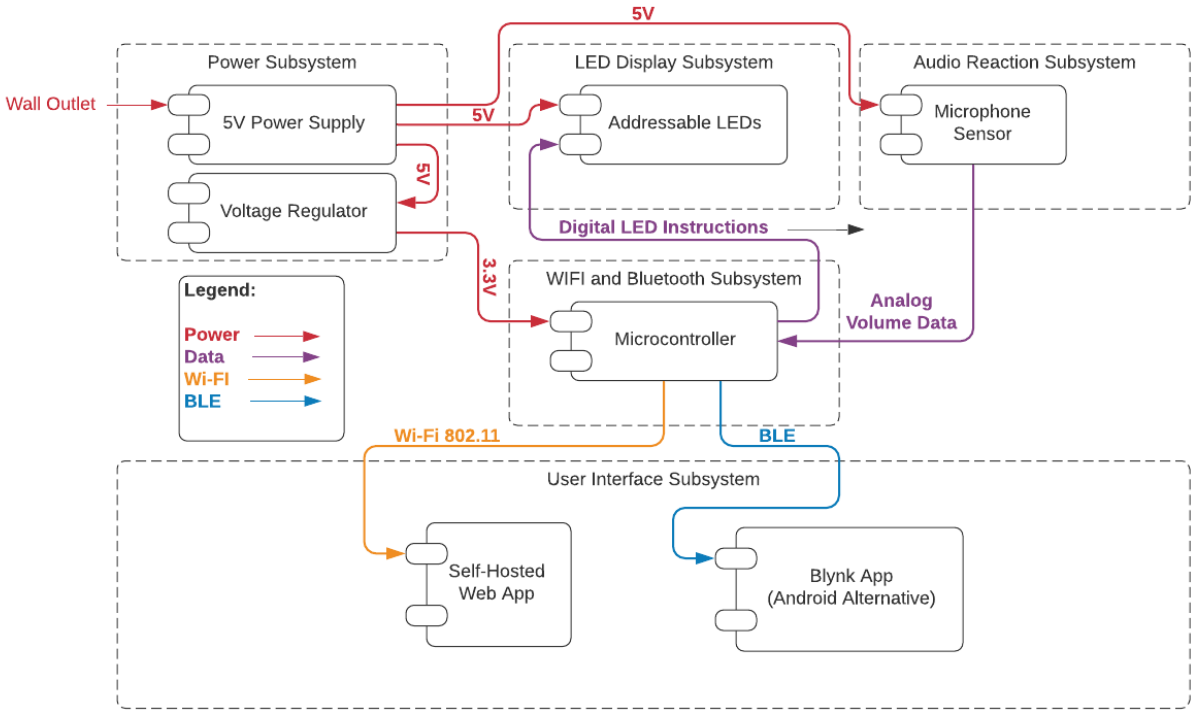

Fig. 4. Block Diagram

2 Design

2.1 Power Subsystem

A power supply is required to power the LEDs, the microcontroller, and the microphone sensor. We have 640 LEDs with a maximum power draw of 60mA per LED, for a total power draw of 38.4A. This assumes that all of our LEDs are on at max intensity, which will never be the case in our project. The ESP32 and microphone draw a negligible amount in comparison. Therefore a 15A power supply is suitable, with the note that only 39% of our LEDs can be on at maximum power at any given time.

2.1.1 5V Power Supply

We used a power supply with a built-in AC to DC converter that plugs into a standard US wall outlet, just as planned. This supplied 5V power to each LED strip, the microcontroller, and the microphone sensor. The power supply can be seen as the black box in Figure 3.

2.1.2 Voltage Regulator

We planned and attempted to use a voltage regulator as part of our PCB design, but due to issues with the PCB the voltage regulator was never required. The ESP32 DevKit we used in our final design accepts 5V voltage directly (it has a built in voltage regulator), making this subcomponent irrelevant. The circuit design we used on the PCB can be seen below.

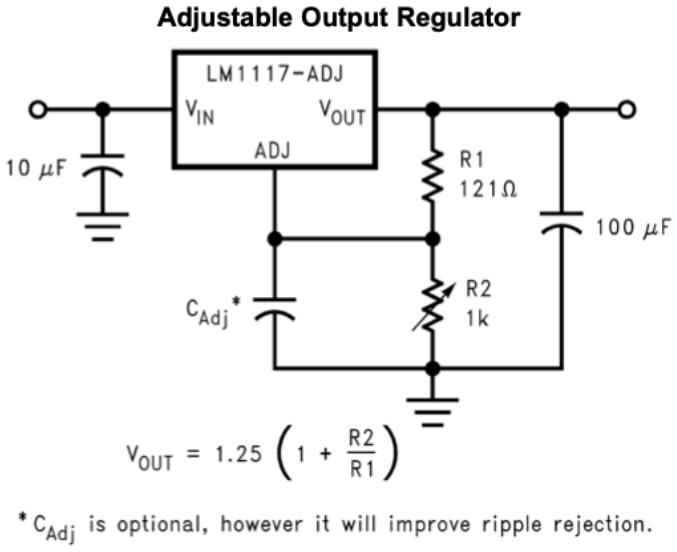

Fig. 5. Circuit Schematic for TI LM1117 Linear Voltage Regulator[6]

2.2 LED Display Subsystem

The LED Display acts as the main visual interface that displays the current score of the game or record of the user's favorite team. This subsystem interfaces with the power supply to receive 5V power, and with the ESP32 to receive digital lighting instructions through a GPIO pin.

2.2.1 WS2812B Addressable LEDs

We originally planned to use 600 WS2812B LEDs arranged in a 40x15 grid, but later modified our design to include 640 LEDs in a 40x16 grid. We made this design change due to having extra space at the bottom of the enclosure, and to give us a top and bottom border with two rows of 5x7 text in the middle. The data pins were connected in serial as planned, allowing us to easily map to control the lights as if it was a solid 40x16 display. Each strip was connected to the 5V power supply in parallel, minimizing the power loss from resistance. The final matrix can be seen back in Figure 1.

2.3 Audio Reaction Subsystem

The audio reaction subsystem makes the scoreboard an interactive highlight of the room in group settings. This is utilized in our "sound border" animation setting, in which the top and bottom borders light up relative to the volume of the room.

2.3.1 Microphone Sensor

We used the SparkFun Sound Detector to detect the room's noise level. The sensor was connected to the 5V power supply, and the audio pin was connected to one of the ESP32's GPIO pins. This gives us the raw audio signal, which we use to control how many of the top and bottom row LEDs are turned on. The circuit for the detector can be seen below.

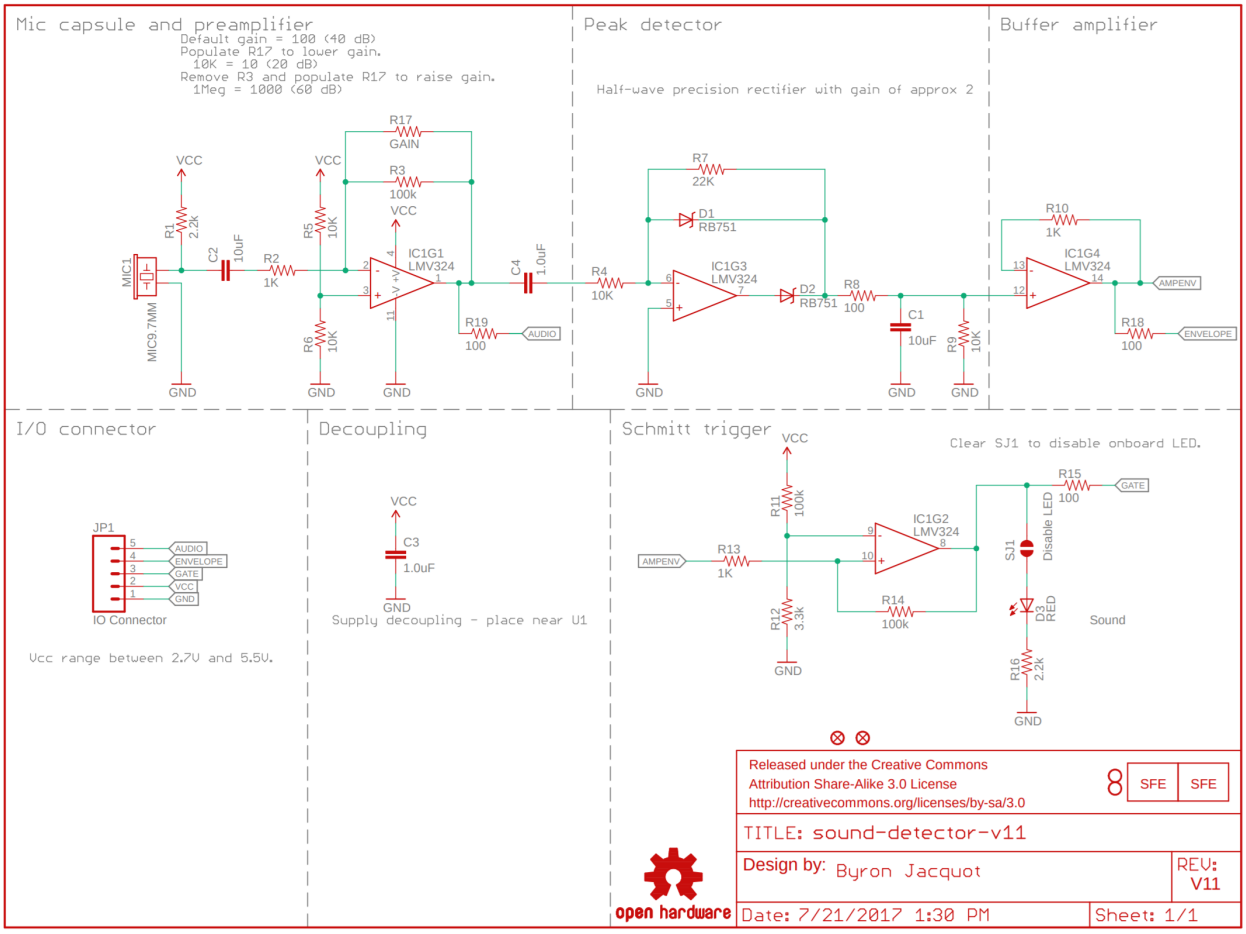

Fig. 6. Circuit Schematic for SparkFun Sound Detector

2.4 Wi-Fi and Bluetooth Subsystem

While called the Wi-Fi and Bluetooth subsystem, this subsystem also acts as the driving microcontroller to control the lights, interpret microphone data, and interact with the user through the web application and bluetooth connection. This subsystem interfaces with every other subsystem. It interfaces with the power subsystem to receive 5V power, with the LED display subsystem to send lighting information over the GPIO pins, with the audio reaction subsystem to receive analog volume data over a GPIO pin, and with the user interface subsystem through Wi-Fi 802.11.

2.4.1 ESP32

We decided to use an ESP32 chip as it has support for both Wi-Fi and Bluetooth built in, and it has a dual core processor to handle both the connectivity and the application code. It is limited to 520 kB of on-chip SRAM, but has support for external Flash and SRAM should our code be larger. While we designed a PCB to incorporate the ESP32 chip (seen below in Figure 7), we had difficulties getting the PCB to work properly. Our final design ended up

using an ESP32 DevKit instead. Another change from our design stems from the Bluetooth compatibility. While we planned to use BLE to retrieve the Wi-Fi information from the user, we instead decided to just hard code the information as the Blynk app we planned to use did not function properly.

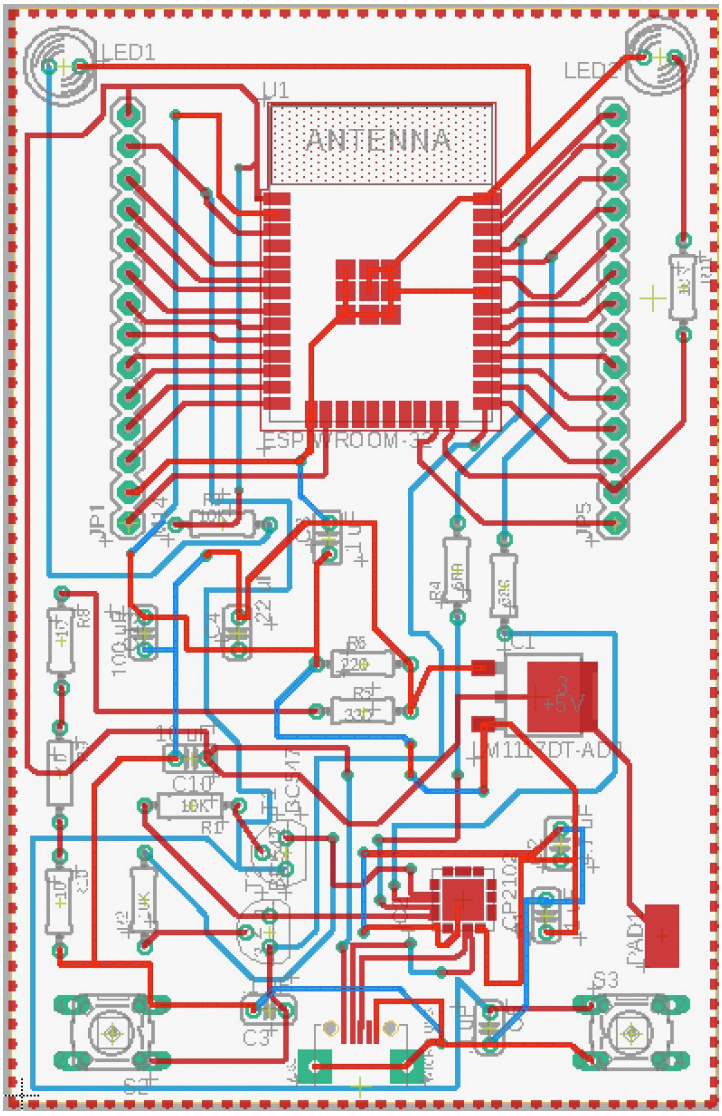

Fig. 7. EPS32 PCB Circuit Design

2.5 User Interface Subsystem

This subsystem is fully in software, and is how the user interacts with the device to change their favorite team and control the visual settings. We originally planned for two user interfaces, the main one being a web application connected to the ESP32 over Wi-Fi to control the settings, and the second one a phone app called Blynk connected to the ESP32 over Bluetooth Low Energy to retrieve the Wi-Fi password for the network it will be connected to. Our final design included only the web application.

2.5.1 Web Application

We created an intuitive web interface for users to communicate with the device. This is where users tell the device who their favorite team is, what sport information they want displayed, and what border style they want. We support 11 sports leagues, can display the current game score, previous game score, or the team's record, and have three border options (sound, static, or bouncing ball). The application runs on the ESP32's server, and users have to be on the same Wi-Fi network as the ESP32 to connect to it. The user interface can be seen below in Figure 8.

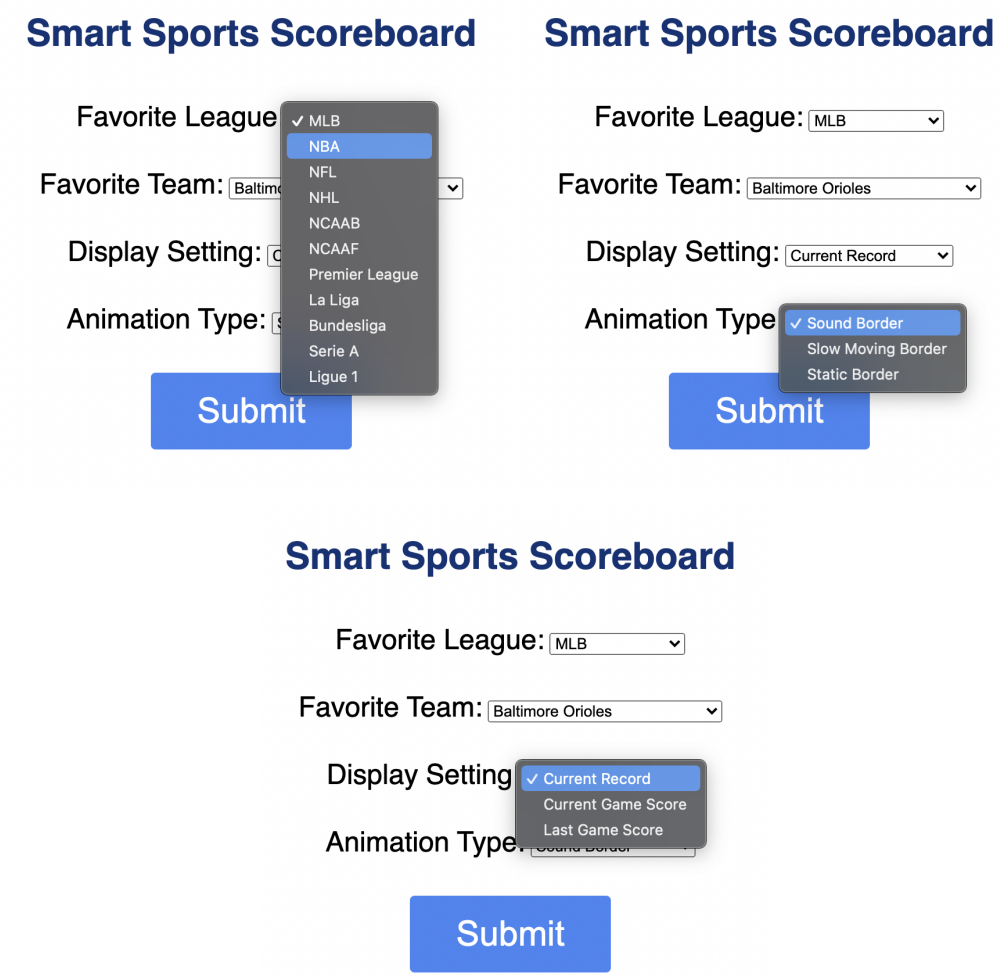

Fig. 8. Web Application Screenshots

2.6 Software

The software is critical for retrieving and displaying the sports data. Originally, we planned on accessing the ESPN API directly from the ESP32 chip and parsing the response there, but we quickly ran into memory errors due to the large size of the responses and the limited SRAM. We decided to create our own API server using Flask, which parses the ESPN API and creates a small JSON file with only the information we need to send to the ESP32. This flask server is what the ESP32 interacts with, as seen in Figure 9 below.

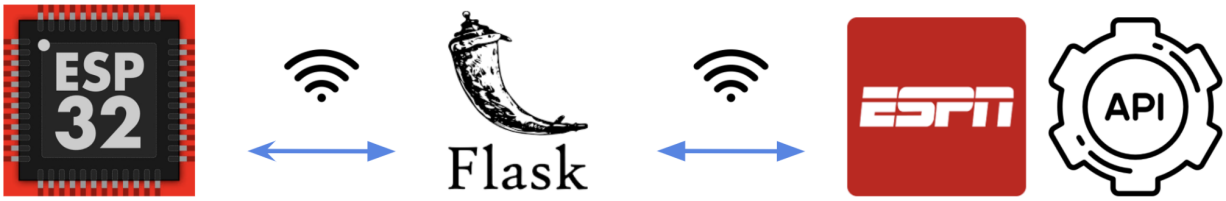

Fig. 9. Sports Data Flow

2.6.1 Sports Data Scraper

Our Flask server receives the league and team names from the ESP32 in the form of a GET request, and uses this information to hit the necessary ESPN API endpoints. In this backend we parse the long JSON to isolate the relevant information for our scoreboard. We return the team's primary and secondary colors, their record, the team they most recently played and that game's score, the team they previously played and that game's score, and the period/inning/quarter of these games. This resulting JSON is small enough for the ESP32 to parse without running into memory errors.

2.6.2 LED Driver

Once the data is obtained from our custom API, it must be used to program the LEDs. Our LED Driver allows us to write two rows of eight letters, with the top and bottom LED rows being used as a border. This is the perfect length to display game scores and team abbreviations. The driver software uses a bitmap dictionary of 5x7 characters to know which LEDs to turn on for each character. The code logic for using the bitmap to draw a character can be seen in Figure 10. Another important note is that our code keeps track of the number of LEDs lit to ensure we don't light more than 39%, satisfying our power supply limitations.

```
ledIndexX = position % 8
ledIndexY = (position // 8) * 7
ledIndex = 40 + (ledIndexY * 40 + (ledIndexX * 5))

for y in range(7):
    for x in range(5):
        curIndex = ledIndex + y*40 + x
        prevLit = (self.np[curIndex] != (0, 0, 0))

        if int('{0:08b}'.format(chars[char][x])[7 - y]):
            self.np[curIndex] = color
            if not prevLit:
                self.numLit += 1
        else:
            self.np[curIndex] = (0, 0, 0)
            if prevLit:
                self.numLit -= 1

self.np.write()
```

Fig. 10. LED Driver - Single Character Logic

3 Verification

3.1 Successful Verification

3.1.1 LED Display Subsystem

Table 5 in Appendix A refers to the requirements and verification process for the LED Display Subsystem. We were able to verify that all LEDs can display the full spectrum of colors and be individually controlled by the ESP32. We wrote one script that lights the matrix up with a looping rainbow gradient. Another script, shown in Figure 11, was written to individually power each LED and make them red. The successful result is shown in Figure 12, showing the matrix after the script has finished.

```
224  from machine import Pin
225  from time import sleep
226  from neopixel import NeoPixel
227
228  np = neoPixel(Pin(22), 640)
229
230  for i in range(0,640):
231      np[i] = (255,0,0)
232      np.write()
```

Fig. 11. LED Display Verification Test Script

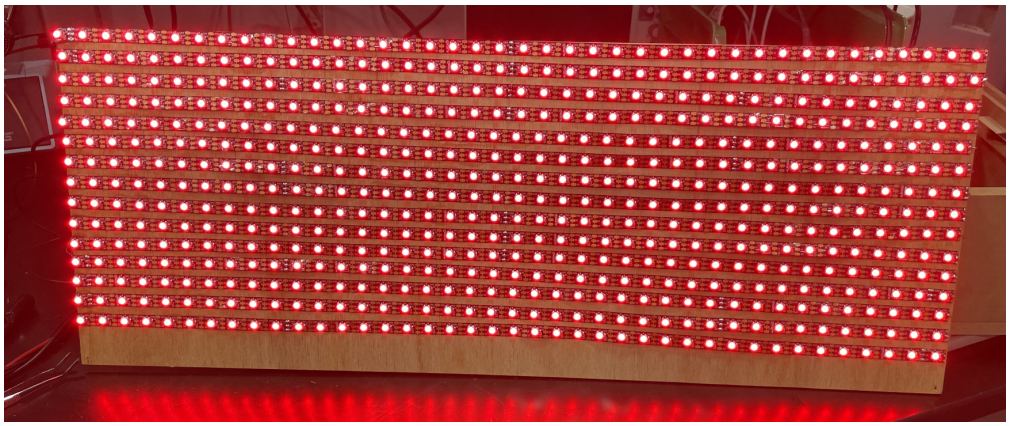

Fig. 12. LED Display Verification Result

3.1.2 Audio Reaction Subsystem

Table 4 in Appendix A refers to the requirements and verification process for the Audio Reaction Subsystem. To properly determine if the audio sensor could properly differentiate between different noise levels, a test script was written on the ESP32 that interfaces with the LED matrix. Specifically, the test script programs a series of 40 LEDs to light up based on the measured noise level. At the end of the script, the decibel level and number of LEDs are printed. Figure 13 is the piece of code written to perform this test and Figure 14 is the corresponding results. It is clear that the number of LEDs lit up is directly proportional to the decibel level. It is also shown that the LEDs are only lit up only when a fan raises their voice higher than normal talking volume. This successfully verified that the audio sensor could differentiate between differing noise levels.

```
def sound_border(self, color, audio_level, decibels):
    for i in range(40):
        if i < audio_level:
            self.np[i] = color
            self.np[i+600] = color
        else:
            self.np[i] = (0,0,0)
            self.np[i+600] = (0,0,0)
    if audio_level > 0:
        print("Decibel value: " + str(decibels))
        print("Number of LEDs lit: " + str(audio_level))

    self.np.write()
```

Fig. 13. Audio Reaction Verification Test Script

```
Decibel value: 69.36103
Number of LEDs lit: 40
Decibel value: 67.25719
Number of LEDs lit: 30
Decibel value: 67.74423
Number of LEDs lit: 40
Decibel value: 67.54613
Number of LEDs lit: 38
Decibel value: 68.41232
Number of LEDs lit: 40
Decibel value: 68.22903
Number of LEDs lit: 40
Decibel value: 66.22236
Number of LEDs lit: 4
Decibel value: 66.15848
Number of LEDs lit: 3
Decibel value: 66.31521
Number of LEDs lit: 6
Decibel value: 66.21387
Number of LEDs lit: 4
Decibel value: 68.23239
Number of LEDs lit: 40
Decibel value: 67.7157
Number of LEDs lit: 40
Decibel value: 66.56351
Number of LEDs lit: 12
Decibel value: 68.64655
Number of LEDs lit: 40
Decibel value: 66.15848
Number of LEDs lit: 3
Decibel value: 66.45679
Number of LEDs lit: 10
Decibel value: 66.62045
Number of LEDs lit: 14
```

Fig. 14. Audio Reaction Verification Data

3.1.3 User Interface Subsystem

Table 3 in Appendix A refers to the requirements and verification process for the User Interface Subsystem. To ensure that this subsystem allows for customization in terms of sports and animations, each league was tested three times with three separate teams. Each test run had a separate display between team record, current score and previous score. Additionally, each test run had one of three animation modes between slow moving, static, and sound border. The results displayed in Figures 15 and 16 are evidence that the scoreboard system can properly display all teams from every league with every customization option. The last column indicates a binary yes or no for a correct and accurate display during the given test run. The following three images, Figures 17, 18 and 19, are three example displays of the different customization modes for baseball, basketball and soccer.

League + Team	Customization Mode	Animation Mode	Correct
MLB + White Sox	Team Record	Sound border	Y
MLB + Dodgers	Current Game	Slow border	Y
MLB + Mets	Previous Game	Static border	Y
NBA + Bulls	Team Record	Sound border	Y
NBA + Thunder	Current Game	Slow border	Y
NBA + Nets	Previous Game	Static border	Y
NFL + Bears	Team Record	Sound border	Y
NFL + Steelers	Current Game	Slow border	Y
NFL + Chiefs	Previous Game	Static border	Y
NHL + Blackhawks	Team Record	Sound border	Y
NHL + Jets	Current Game	Slow border	Y
NHL + Blue Jackets	Previous Game	Static border	Y
NCAAB + Illini	Team Record	Sound border	Y
NCAAB + UCLA	Current Game	Slow border	Y
NCAAB + Baylor	Previous Game	Static border	Y
NCAAF + Illini	Team Record	Sound border	Y

Fig. 15. User Interface Verification Data - 1 of 2

League + Team	Customization Mode	Animation Mode	Correct
NCAAF + Alabama	Current Game	Slow border	Y
NCAAF + Vandy	Previous Game	Static border	Y
EPL + ManUnited	Team Record	Sound border	Y
EPL + ManCity	Current Game	Slow border	Y
EPL + Chelsea	Previous Game	Static border	Y
La Liga + Barcelona	Team Record	Sound border	Y
La Liga + Barcelona	Current Game	Slow border	Y
La Liga + Madrid	Previous Game	Static border	Y
Bundasliga + Bayern	Team Record	Sound border	Y
Bundasliga + Bayern	Current Game	Slow border	Y
Bundasliga + Leipzig	Previous Game	Static border	Y
Serie A + Inter	Team Record	Sound border	Y
Serie A + Milan	Current Game	Slow border	Y
Serie A + Juventus	Previous Game	Static border	Y
Ligue 1 + PSG	Team Record	Sound border	Y
Ligue 1 + PSG	Current Game	Slow border	Y
Ligue 1 + Monaco	Previous Game	Static border	Y

Fig. 16. User Interface Verification Data - 2 of 2

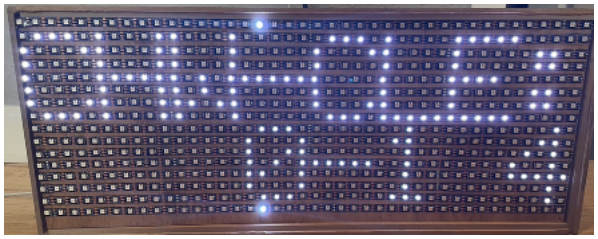

Fig. 17. Current Game Score and Slow Moving Border

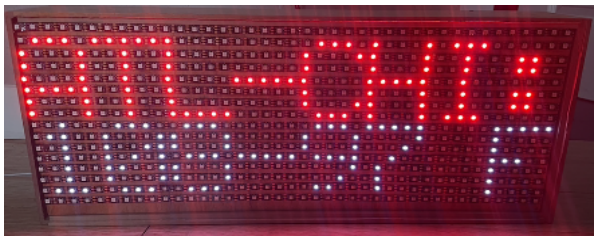

Fig. 18. Previous Game Score and Sound Border

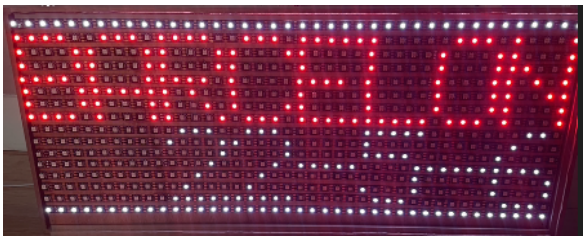

Fig. 19. Team Record and Static Border

3.2 Unsuccessful Verification

3.2.1 Wi-Fi Subsystem

The Wi-Fi Subsystem represents the PCB that was designed to function the same way that an ESP32-DevKit does. In the successful verifications of our project, we utilized the ESP32-DevKit to host the UI server, drive the LEDs and program the audio sensor. We went through three PCB rounds without success, including two rounds that were ordered individually. In the first round, we were unable to solder the ESP32 module onto the PCB. We realized that the main ground pin in the middle of the board was not properly aligned, preventing a connection with the solder. In the second round, we determined that there was no true ground on the PCB and power was never dissipating. This was discovered by probing different power pads on the PCB. The longer the ESP32 was powered, the hotter it would get. The third and final PCB arrived a day before our demo. The team worked as hard as possible with a self-purchased soldering iron to solder the ESP32 chip onto the board, but the iron was extremely thick. Ultimately, there was not enough time to complete the final soldering. There was promising progress made, as the PCB was detected by the computer when plugged in via micro-USB. However, we were unable to successfully flash the ESP32 with MicroPython and were forced to go back to the ESP32-DevKit. Figure 20 shows the final PCB attempt, with all of the components soldered onto the board.

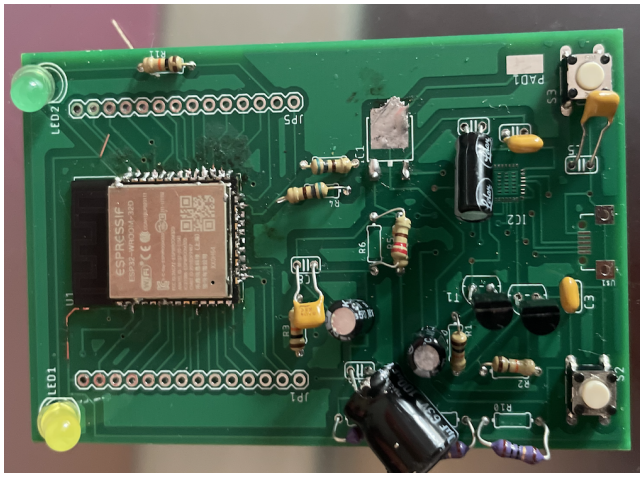

Fig. 20. Final PCB Attempt

4. Costs

The grand total for this project is a combination of the costs for parts and labor. Our grand total for making this project is approximately **$30,250.00.**

4.1 Parts

The costs for the parts of this project are broken down in Table 1. As with many projects, our group did not have all of the parts that we needed originally. Mistakes happen, and new findings result in the continuous order process for parts in projects of this scale. Therefore, our total for parts will not match the total for parts from the design document. Our final cost of parts was $245.50. Our initial estimate for this project was that parts would be $124.45. We therefore spent about double what we had budgeted, but we are happy with our results and this is not a problem.

Table 1. Parts Costs

Part Name	Description	Manufacturer	Part Number	Quantity	Cost per part
AC/DC Power Adapter	5V 15 A AC/DC Power Supply Adapter	ALITOVE	5V-15A-BK	1	$28.99
ESP32 Microcontroller	ESP32 Dev Board with WiFi Microcontroller	MELIFE	B07Q576VWZ	2	$14.99
SparkFun Sound Detector	Microphone for Audio	SparkFun Electronics	SEN-12642	1	$10.49
Individually Adressable LED Strip Light	LED's for Matrix	ALITOVE	8128300WPBK-FBA	2	$34.99
Breadboard	Breadboard for Testing	ECE Shop	N/A	1	Previously Acquired
Assorted Resistors	Resistor set for building circuits	ECE Shop	Varies	1	Previously Acquired
Assorted Capacitors	Capacitor set for building circuits	ECE Shop	Varies	1	Previously Acquired
Individually Ordered PCBs	PCB's because 3rd round did not arrive	PCB Way	N/A	2	$26.00
Micro USB Breakout Board	Breakout board instead of soldering small piece	HiLetgo	CP2102	1	$6.49
Individually Adressable LED Strip Light	LED's for Matrix	ALITOVE	B018X04ES2	2	$17.99
Gorilla Glue	Glue for fixing matrix	Gorilla Glue	7400202	2	$5.79
					Total Parts Costs: $245.50

4.2 Labor

In Table 2, the labor costs of the project are broken down on a person by person basis and summed up at the end to give us the total cost of labor for the group. We assumed a rate of $40/hour for the project as there is a large software engineering component to our work. We assumed that for the rest of the 10 weeks, we will all be contributing about 10 hours/week to the project. Using these statistics, we came up with our total labor costs to be $30,000.

Table 2. Labor Costs

Partner	Hourly Rate	Hours/Week	Weeks	2.5x Multiplier	Total/Person
Michael	40	10	10	2.5	$10,000
Max	40	10	10	2.5	$10,000
Matthew	40	10	10	2.5	$10,000
					Total Labor Costs: $30,000

5. Conclusion

5.1 Accomplishments

Our team was able to put together a fully functioning smart sports scoreboard, composed of multiple systems integrated together as described above. Our device is able to display the scores, both past and present, and records of teams across 11 leagues. We have achieved end-to-end functionality from an ESPN API to Flask to ESP32 server to LED matrix driver that results in a user being able to get real time updates for their favorite team without having to check their mobile device. We have also made this device to have a smart and intuitive user interface which is easy to understand, customize, and use for individuals anywhere in the world.

5.2 Uncertainties

As with any project challenges occurred during this project. Challenges present opportunities to learn, and this is the way our group chose to attack these. Some of the notable challenges we faced were with PCB design, Bluetooth, and low on chip memory. Our group used an ESP32 Dev Kit in the final project and was consequentiality unable to use a PCB which was individually designed. We express ordered 2 boards with our own money, but were unable to get either working for the demo. Regardless of the outcome our group learned about using Eagle software and soldering in depth. These were great takeaways that will help our group in the future. Bluetooth was initially a portion of our design that we eventually removed after we realized that its incorporation was outside of the scope of this

project. Low on chip memory initially resulted in occasional crashing of the board, but our group was able to find workarounds so that this would not be a problem for the user.

5.3 Ethical considerations

We as a group, in accordance with the IEEE Code of Ethics, understand that it is our responsibility to commit ourselves to the highest ethical and professional standards in creating this device. In particular, our device is responsible for "hold(ing) paramount the safety, health, and welfare of the public" [11]. As a Smart Sports Scoreboard, further versions of the product may include sports gambling features so that users may stay up to date with betting odds on games that they are following. Any sports betting features do not have the ability to make wagers. These features are specifically for entertainment purposes. We as a group only condone sports gambling where it is done legally in states that allow it, and only when it is done responsibly by the individuals making the wagers.
Our device also strived "to treat all persons fairly and with respect" [11]. The purpose of our device is to provide real-time information in a visually appealing way to the user, so that they may spend less time on their smart devices and more time honed in on what is happening around them. We, in no way, intend to alter the information the user receives, and only hope to provide the user with true information. Therefore, we strived to treat all persons, teams, and players being represented on our device fairly and with the utmost respect.

It is our responsibility as a group to hold each other accountable in "striv(ing) to ensure this code is upheld by colleagues" [11]. We, in accordance with the code of ethics, supported our teammates and continually followed up with each other to ensure that we upheld conduct of the highest standard. No member of the group behaved unethically, and thus no retaliation was present against individuals since no one was to report a violation. It is our responsibility in creating a device to make sure that positive value is provided to the world, and this is what we did.

5.4 Future work

There are three areas of future work which the group sees as a possibility for expansion of the project in the future. The first is implementing a cloud server. Right now, Flask needs to be running locally for the device to work, but this cloud server would allow for a much more easily accessible experience from the user, where they could use the device from anywhere. The second is displaying individual statistics. ESPN APIs that we used had the information for individual statistics and in the future we could display this information on the board. The third is being able to show the whole horizontal name of a team. For example, Liverpool gets cut off at "Liverpoo". We could add an animation to scroll through the team's letters to avoid confusion for users. It is important to note that the second and

third were not done for this project because at the end of our project timeline, the group shifted focus away from software development and totally towards PCB design in attempt to create a working board.

References

[1] Pew Research Center, *Mobile Fact Sheet*, Pew Research Center, June 12, 2019. Accessed on February 15, 2021. [Online]. Available:https://www.pewresearch.org/internet/fact-sheet/mobile/

[2] Asurion, *Americans Check Their Phones 96 Times a Day*, Asurion, November 21, 2019. Accessed on February 14, 2021. [Online]. Available: https://www.asurion.com/about/press-releases/americans-check-their-phones-96-times-a-day/

[3] C. Gough, *Sports fans using mobile apps worldwide 2019, by age*, Statista, March 20, 2020. Accessed on February 14, 2021. [Online]. Available: https://www.statista.com/statistics/1100567/sports-content-mobile-apps/

[4] F. Yanoga, *Does blue light from electronic devices damage your eyes?*, Wexner Medical Center, June 13, 2019. Accessed on March 3, 2021. [Online]. Available: https://wexnermedical.osu.edu/blog/blue-light-and-vision

[5] Random Nerd Tutorials, *ESP32 Bluetooth Low Energy (BLE) on Arduino IDE*, Random Nerd Tutorials, June 4, 2019. Accessed on March 3, 2021. [Online]. Available: https://randomnerdtutorials.com/esp32-bluetooth-low-energy-ble-arduino-ide/

[6] Texas Instruments, *LM1117 800-mA, Low-Dropout Linear Regulator*, LM1117 datasheet, February 2000. [Online]. Available: https://www.ti.com/lit/ds/symlink/lm1117.pdf?ts=1615985306907&ref_url=https%253A%252F%252Fwww.google.com%252F

[7] K. Draper, *ESPN Tries to Get With a Mobile, App-Driven World*, The New York Times, April 12, 2018. Accessed on February 15, 2021. [Online]. Available: https://www.nytimes.com/2018/04/12/sports/espn-app.html

[8] OSHA, "CONSTRUCTION SAFETY & HEALTH." [Online]. Accessed: 01-Mar-2021. Available: https://www.osha.gov/sites/default/files/2018-12/fy07_sh-16586-07_4_electrical_safety_participant_guide.pdf.

[9] IEEE.org, *IEEE Code of Ethics,* IEEE, 2021. Accessed on February 13, 2021. [Online]. Available: https://www.ieee.org/about/corporate/governance/p7-8.html

[10] Espressif Systems, *ESP32 Series*, ESP32-WROOM-32 datasheet, Jan 2021. [Online]. Available: https://www.espressif.com/sites/default/files/documentation/esp32-wroom-32_datasheet_en.pdf

[11] H. Regan, *Spoiler Alert: Low Latency Crucial for Sports Apps*, Wowza Media Systems, May 22, 2017. Accessed on March 3, 2021. [Online]. Available: https://www.wowza.com/blog/spoiler-alert-low-latency-crucial-for-sports-apps

[12] L. Podkalicki, *eagle-libraries*, ver 1.7. Accessed on March 4, 2021. [Online]. Available: https://github.com/lpodkalicki/eagle-libraries

[13] B. Stegner, "Do Monitor Refresh Rates Matter? Everything You Need to Know," *MUO*, 12-Feb-2021. [Online]. Available: https://www.makeuseof.com/tag/60hz-vs-144hz/. [Accessed: 05-Mar-2021].

Appendix A

Table 3. Web Application Requirements and Verifications

Requirements	Verification	Verified (Y/N)
1. The user must be able to select from all available teams from the following eleven leagues: NFL, NBA, MLB, NHL, College Basketball, College Football, Premier League, La Liga, Bundasliga, Serie A, and Ligue 1.	1. a. Write software for the web app that properly parses the API response containing the list of teams, storing the teams alphabetically in the form of a list. b. One-by-one, select one team from each sport listed. The scoreboard must display the final score of each team's previous game.	Y
2. The user must be able to toggle between three customization modes: display the current score (if there is a game in progress), most recent score, or team's record.	2. a. There will be a dropdown menu that allows the user to select between modes. Select each mode, one-by-one, ensuring that each mode displays accordingly on the LEDs. b. Repeat step (a) for all eleven leagues for quality assurance.	Y

Table 4. Audio Reaction Requirements and Verifications

Requirements	Verification	Verified (Y/)
1. Must be able to differentiate between normal room talking volumes (~60-65dB) and yelling volumes (~65-80dB).	1. a. Test the sensor at a baseline volume level (60-65dB) and record the sensor output. b. Test the sensor at a yelling volume level (65dB+) and record the sensor output. c. Ensure that there is an observable (+/- 10%) difference in the recorded sensor output.	Y

Table 5. Addressable LEDs Requirements and Verifications

Requirements	Verification	Verified (Y/N)
1. Each of the 600 LEDs can operate at 36mA and are visible from 5m away.	1. a. Deliver 36 mA to the LED and measure out 5 meters. Verify that the LED is visible.	Y
2. Each of the LEDs can display the full spectrum of RGB colors.	2. a. Write a test script to display a looping rainbow gradient. Verify that the entire spectrum is displayed on each LED.	Y
3. The LEDs can be individually controlled by the ESP32 chip.	3. a. Write a test script to light up each LED individually with the ESP32 and ensure each light successfully turns on.	Y