MIE438 Project Final Report

8*8*8 LED Cubic Game Console

Department of Mechanical and Industrial Engineering

Qilong (Jerry) Cheng 1003834103
Jiaming (Jason) Dong 1003181594

October 29, 2025

1 Introduction and Motivation

Console games have been around for decades, from the first introduction of Computer Space to Modern Nintendo. The market value for console games is worth 42.9 billion[1] and more than 5 million[2] console games have been created. However, the games developed are all restricted by the 2D display, even with the introduction of augmented reality (AR) and virtual reality (VR), true 3D life experienced games have rarely been introduced. The team came up with the idea of using a customized 8*8*8 3D LED cube as the platform to redevelop classic console games. This can break the traditional 2D experience, mimicking holograms, and bringing the games to life. The quest of reviving old console games inspired the team to design our own embedded system and give users an end-to-end next-level playing experience as our project.

2 Initial Problem Framing and Scoping

Objective: Be able to program animations, retro games on a customized 8*8*8 LED cube, and control it via a PS2 controller.
Scope:

1. Design and build the LED cube hardware, including soldering the LEDs, CAD and 3D printing a base for the cube.
2. Design a customized PCB to eliminate disorganized wiring.
3. Program a library to easily display contents onto the LED, and can be imported and reused for later games/animations.
4. Make simple animation effects: waves, fireworks etc.
5. Program at least one game on the cube: 3D snake game; pong; fighter jet, etc.

3 Final Design and Goals Achieved

The final design achieved almost all the preliminary goals set in the project proposal. However, there were changes made and future work is needed to polish the project. In a nutshell, the goals achieved and changes are listed below:

- Designed the high-level architecture of the LED wiring, soldered the cube and built the PCB prototype. Due to the high voltage difference, the team switched from the previous 3mm LED to 5mm LED for its better current and voltage handling. Current limiting resistors have been omitted due to the relatively low current.
- 3D printed the base, and an acrylic enclosure was added for protection.
- Modified the circuit design. Originally, anodes and cathodes were controlled separately by the microcontroller, whereas the current design daisy-chained all the shift registers and eliminated additional wiring to the microcontroller. In addition, instead of using serial communication between the shift registers and the microcontroller, SPI is used to send out the buffer data for its faster speed.
- MOSFETs have been used to replace the BJTs as switches to control LED cube's cathodes ON/OFF – connected to ground or not.
- Designed the PCB for the LED cube. However, due to the time constrain and some last-minute circuit changes, the team did not get the chance to shift the existing prototype onto a printed PCB board. The final design remained to be wired on the breadboard.
- Used push buttons as external interrupts and two joysticks for gaming control. The team had previously attempted to use a PS2 controller for these purposes. However, the team was not able to adapt to the PS2 controller's communication protocol and was not able to use it for triggering external interrupts to receive user inputs. Instead, the team was able to build our own controller using two joysticks and one push button. The joysticks are wired to four analog pins that constantly pull data from the main loop. The push button was used as an external interrupt to cut off the program and bring it back to the "Home" menu.
- Added an additional active buzzer for playing music, and an OLED display for displaying menu, texts, and animations.
- Programmed seven animation effects to demonstrate the three-dimensional characteristic of the LED cube.
- Programmed two classic console games – Snake and Pong

4 High Level Controller Design

The overall architecture design consists of three inputs: two joysticks, one push button and an IMU; as well as three outputs: a OLED display, a buzzer and the LED cube itself (Figure 1).

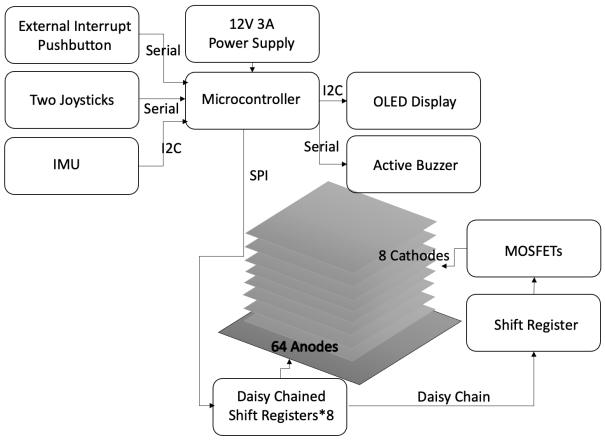

Figure 1: High Level Architecture Design

At the high level of the project, 8*8*8 LEDs (512 in total) were controlled via SPI by 9 shift registers using an Arduino Mega. On each layer, the LEDs' cathodes are connected together to a common ground. One each column, 8 LEDs' anodes are vertically connected to an output pin of a shift register.

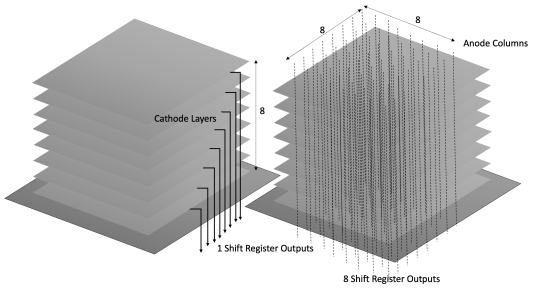

Figure 2: LED Connection Illustration

By using this connection method, the number of output signals can be reduced from 512 down to 64 using multiplexing. This also reduced the number of shift registers needed from 64+1 to 8+1.

5 Hardware Design

5.1 Mechanical Part

The external design is shown below. The LED cube is enclosed inside an acrylic box to prevent damage. The bottom base act both as a support and a housing for all the sensors and circuits. The OLED display cut-out is design so that the display can be integrated with the external housing. In addition, openings are made for the power connector and the USB connector.

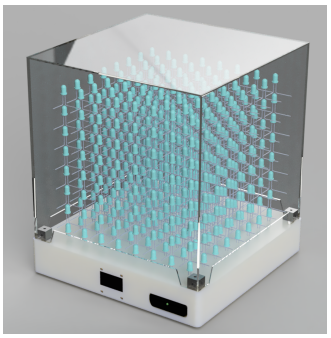

Figure 3: Front Render of the Design

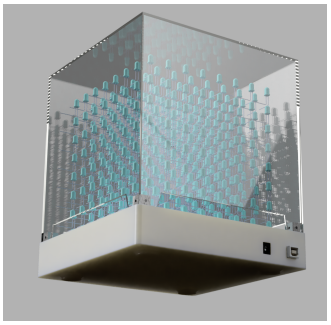

Figure 4: Back Render of the Design

All LEDs are soldered in place by the team and the housing is 3D printed. The final product is showcased below:

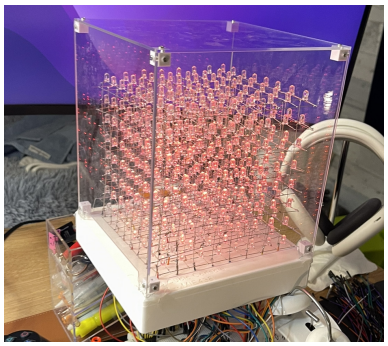

Figure 5: Final built of the LED cube hardware

5.2 Circuits

The overall design of the circuit built using Fritzing is shown in Appendix A, and the most crucial part of the embedded circuit is shown below in figure 6:

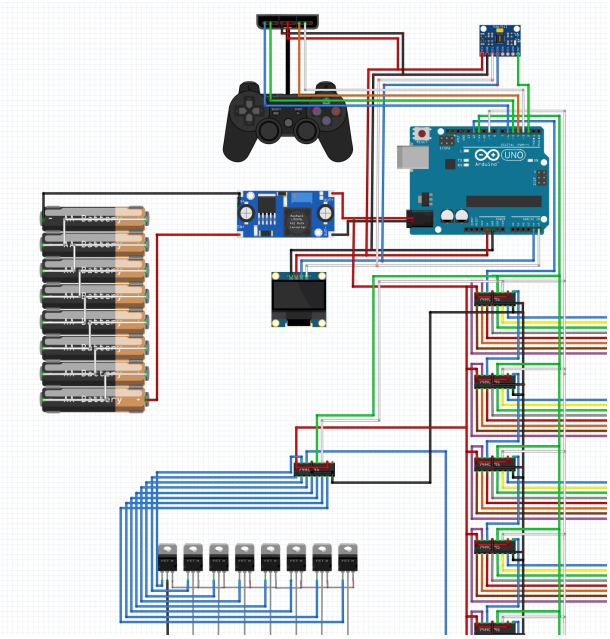

Figure 6: Most Important Part of the Embedded System Circuit Diagram (Full Circuit Diagram Refer to Appendix A)

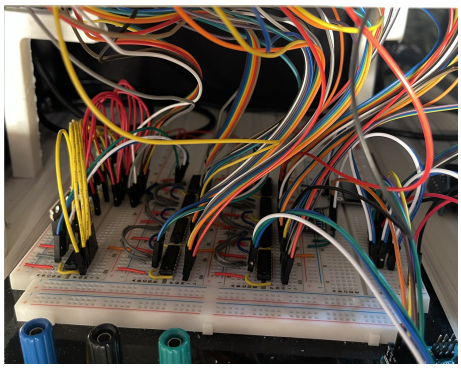

Figure 7: Prototype Final Circuit Layout

5.2.1 Microcontroller

The team initially decided to use the Arduino UNO, but one of the major roadblock was limited memory capacity. Compared to the UNO, Mega contains 256KB of Flash Memory rather than 32KB on the UNO; 8KB RAM instead of 2KB on the UNO. In addition, there are 54 GPIO pins and 15 PWM pins on the Mega rather than 14 GPIOs and 6 PWMs on the UNO, allowing more connectivity with other devices. Therefore, Arduino Mega is chosen for the project for its accessibility, ease of use, and overall better functionality.

Despite the Arduino Mega working well with the current scope of the project, for future iterations, ESP32 should be used to replace the Mega due to its limitations:

- Limited Total Memory: In total, six open libraries were implemented to realize the project – Adafruit_GFX; Adafruit_SSD1306; SPI; WIRE, TimerOne and PS2X_lib. Although the team had optimized the code for efficiency, it still used over 72% of dynamic memory and 24% of the program memory. Although the cube controller and its library only occupied around 5% of its memory, the additional animations for the OLED display took up much of the program memories.
- Limited Number of SPI and I2C Lines: The current embedded system only occupies one SPI bus line to control the LED cube. If in the future, the LEDs were to be upgraded to RGB LEDs, the current SPI lines will not be sufficient using the configuration adopted in this design.
- Limited Clock Cycle: The clock cycle is fundamental in this project as it determines how fast the cube can be refreshed. To be discussed in Section 6.1, the cube uses multiplexing to reduce the total number of output pins needed. This was done by refreshing each level at a refreshing rate faster than what the human eyes can register (around 30Hz). Although, there was no issue using the Arduino's 16MHz clock rate, a faster clock rate would offer a smoother and brighter display.
- Lack of Bluetooth and WIFI Supports: Due to the lack of native supports for WIFI and Bluetooth, more advanced functionalities for the LED cube were abandoned.

ESP32 operates up to 160-240MHz and contains 520KB of RAM, it also contains 4 SPI buses, two I2C controllers, 34 programmable GPIOs, as well as WIFI and Bluetooth system. Overall, ESP32 is a much more superior microcontroller compared to the Mega, and should be adopted in future designs.

5.2.2 Shift Registers

The use of shift registers was the heart of the project. Since there were 512 LEDs, it was unrealistic to have one data output for each individual LEDs. The serial-in parallel-out shift register was able to take in 1 byte of data and store the data until a latch signal was sent to release the stored 8 bits. Eight shift registers were used to accomplish the control of 64 LEDs. An additional shift register coupled with multiplexing was able to achieve the ON/OFF control of each individual LEDs.

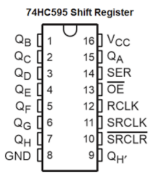

Figure 8: Shift Register Pinout [3]

In total, 9 shift registers are daisy-chained to the microcontroller and are connected with only 3 pins – data pin (SER), latch pin (RCLK), and clock pin(SRCLK). All the shift registers share the same latch and clock lines that come directly from the Arduino. The data line starts from the Arduino output pin to the first shift register's data input pin. The data is then fed into the second shift register's data input pin from the first shift register's data output pin. The same wiring applies to all the 9 shift registers. The output enable pin (OE) is held LOW to be activated, and the master reset pin (SRCLR) is held HIGH to be deactivated.

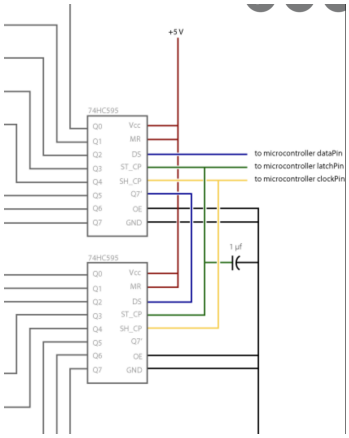

Figure 9: Example of the Two Daisy-Chained Shift Registers Wiring [5]

When the latch pin is set HIGH, data starts shifting in and is stored inside the shift registers. Since there are 9 shift registers, 9 bytes of data are shifted in at once before the latch pin is pulled LOW for release. A simple signal wave form is displayed below:

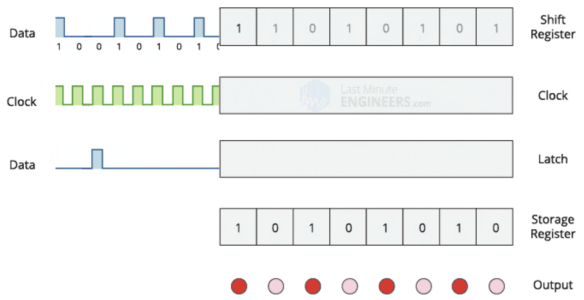

Figure 10: Shift Register Signal Control Illustrations [5]

5.2.3 MOSFET

By implementing MOSFET, the team was able to avoid the flickering LED and non-stable current problems. Differ from BJTs, MOSFET is triggered when the gate reaches the threshold voltage. As the shift register pins can only supply 20mA of current, resistors are needed to create a transistor switch using BJTs. Originally, Darlington transistor array IC and PN2222 were implemented by the team. However, due to the unstable current from the shift registers (purchased from AliExpress), the transistors shifted between the ON/OFF states, causing the LEDs to flicker. Hence, MOSFETs were used in the final version of the design. The shift register pins can output a sustainable voltage between 4.5V to 5.5V, which lay above the minimum threshold voltage of the IRF520 MOSFET used, which is 4V. Moreover, the maximum current from the drain-to-source for the MOSFET we chose (IRF520) is rated at 9.2A, which is more than enough to handle the 64*20mA = 1.28A of current from one level of LEDs.

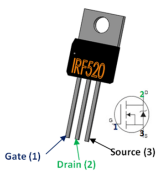

Figure 11: IRF520 N-channel MOSFET Pinout [6]

5.2.4 OLED Display

Different displays were considered by the team such as the LCD screen, various types of OLED displays, and even an e-ink display. In the end, a 128*64 resolution OLED was chosen. This OLED display was chosen for its relatively higher resolution compared to the LCD, and also for its compactness and low cost. Unlike the LCD panel, which requires an external driver to run via I2C, the chosen OLED display is able to communicate with the microcontroller via I2C natively. This greatly reduces the circuit wiring difficulty and the difficulty we would encounter in the high-level coding.

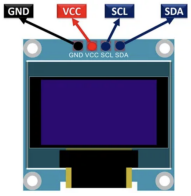

Figure 12: OLED Display Pinout [4]

The team was able to utilize the relatively high resolution of the display to achieve the following.

- **Display Animations:**

  When the user presses the pulse or play button, or when the program goes to the "Home Page", a specific animation will be played on the screen. First, the animation video is transformed into .gif format, each frame is then taken to be formatted into .bmp file. The .bmp files are then fed into a program written by VolosR (Appendix D), which transforms the files into bytes for Arduino to display.

- **Text:**

  Score of the game and the type of the game/animation are also displayed on the OLED.

5.2.5 Buzzer

An active buzzer module is used to add more effects to the games. The wiring of the buzzer is relatively simple, contains a VCC, a GND and a data line.

Figure 13: Active Buzzer Module Pinout [7]

Since there is a built-in transistor (PN2222) inside the module, the active buzzer module turns the sound generation on or off using an additional data line. Since there are no frequency signals to control the pitch or the length of the sound, the music generation is done completely in the software. The built-in Arduino function is used to achieve this goal.

```
tone(pin, frequency, duration)
```

This function automatically generates square waves with a 50% duty cycle. The frequency is simulated by modifying the switching rate of the square wave. the faster it switches, the higher the pitch goes, and vice versa. The length of the square wave controls the length of the note, the longer the duration is, the longer the square wave will be sent. Finally, for the built-in Arduino function tone(), between each time it is called, a short delay is added to separate the notes which controls how fast the song is being played. The delay time is set to be the same as the duration each tone is played.

6 Software Design

The software aspect was the most challenging part of the project. The team struggled to debug many low-level programming issues related to the embedded system. The main software was done mostly on the Arduino IDE in C++, and the libraries for the program were done in VS Code.

There are four main aspects in the software designs:

- LED Cube Display Library Creation
- Animation Effects
- ISR For Receiving User Inputs and Update Buffer
- 3D Retro Game Creations

Since the LED cube was fully customized by the team, there was no open library to be implement with. However, the team did find some great resources online regarding the software design (refer to Appendix D). The software was heavily modified to suit the hardware design of the project. All the firmware had been open-sourced and can be found in the GitHub repository in Appendix C.

6.1 LED Cube Display Library

The design of the 3D Cube display library is similar to a 2D LCD screen, where the data is stored as bytes in an array. For our 3D cube, a two dimensional array called cube [8][8] was used to store the buffer data. In total, 64 bytes of memories were used to store the ON/OFF value of each LED.

The first thing the team had done was to create several low-level helper functions that can set the ON/OFF for each individual LED and get the ON/OFF states of the targeted ones. Each LED is visualized into an x-y-z Cartesian coordinate ranging from (0,0,0) to (7,7,7).

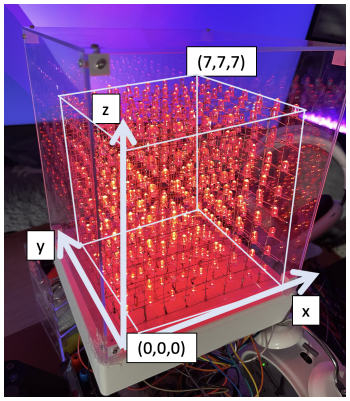

Figure 14: LED Coordinates Illustration

The value-changes for the LEDs were done using bit Math.

- **Turn ON one single LED**

```
void setVoxel(int x, int y, int z):
    cube[7 - y][7 - z] |= (0x01 << x);
```

- **Turn OFF one single LED**

```
void clearVoxel(int x, int y, int z):
     cube[y][z] &= ~(0x01 << x);
```

- **Flip one single LED**

```
void flipVoxel(int x, int y, int z):
    cube[7 - y][7 - z] ^= (0x01 << x);
```

- **Get the ON/OFF state of a single LED**

```
bool getVoxel(int x, int y, int z):
    return (cube[7 - y][7 - z] & (0x01 << x)) == (0x01 << x);
```

Once we were able to control and communicate with each individual pixel (a single LED), manipulations in the macro level could then be done to achieve multiple LEDs lighting up at the same time. For example, to light up the entire plane, a function called **setPlane()** would be used:

```
void setPlane(uint8_t axis, uint8_t i) {
  for (uint8_t j = 0; j < 8; j++) {
    for (uint8_t k = 0; k < 8; k++) {
      if (axis == XAXIS) {
        setVoxel(i, j, k);
      } else if (axis == YAXIS) {
        setVoxel(j, i, k);
      } else if (axis == ZAXIS) {
        setVoxel(j, k, i);
      }
    }
  }
}
```

Similar logic and bit shifting can be mixed to achieve visually stunning animations.

Finally, the most important function, **renderCube()** was used to send the bytes stored in the buffer, cube[8][8] via SPI lines to the shift registers to light up the cube.

```
void renderCube() {
  for (uint8_t i = 0; i < 8; i++) {
    digitalWrite(SS, LOW);
    SPI.transfer(0x01 << i);
    for (uint8_t j = 0; j < 8; j++) {
      SPI.transfer(cube[i][j]);
    }
    digitalWrite(SS, HIGH);
  }
}
```

This function sends 8 bytes of data from one level to another; in addition, an extra byte is used to control which level is lighting up. This multiplexing method at the software level allowed us to avoid using 64+1 shift registers, and instead only requires 9 shift registers to control 512 LEDs individually. The execution of the for loop can be done at a very fast rate as we had set the SPI clock cycle at 16MHz. The human eyes will not able to distinguish the ON/OFF sequences at this rate.

6.2 Interrupt Design

This was the most challenging part of the project. Two methods were considered by the team.

- ISR for user input and update the buffer
- ISR for refreshing the LED cube

Initially, the team attempted the first method. However, for each iteration in the main loop, reading the user input and calculating the logic takes a substantial amount of time, thus the cube only lights up for a small amount of time compared to the time it takes to calculate the buffer. Moreover, this method prevents us from easily modifying the refresh/update rate of the cube, increasing the difficulty of changing the update rate of the game or the animation.

Hence an internal timer interrupt was used to update the cube every 300µs. When the set time is reached, it will stop the main program and jump into the ISR – renderCube() to light up the cube. The 300µs is tuned carefully by the team. When it is too small, it might create a race condition where there is not enough time for the buffer data to be computed and sent out to the cube. When it is too large, it could cause the LED cube to be too dim as there is not enough refresh happening during each loop. The internal timer interrupt is done using the Arduino library TimerOne, and can be initialized using:

```
Timer1.initialize(300);
Timer1.attachInterrupt(renderCube);
```

In addition to the timer interrupt, external interrupts triggered by a push button was also implemented. For the software UI design, the team hoped for a button that functions as the "Home button" to return to the start whenever

it is pressed. Since there are delays for each refreshes that the program cannot stop without finishing delay program execution, external interrupts are used to make the program jump back to the start of the loop. Those external interrupts are also setup using the Arduino provided function:

```
attachInterrupt(digitalPinToInterrupt(pin), ISR, mode)
```

6.3 Input Device Decision and Related Firmware

The main control panel designed for the user includes two joysticks and one push button. As mentioned above, the push button was used as an external interrupt trigger to force the program to jump back to"Home". The two joysticks are designed so that there is enough axis for the user to visualize the control in a three dimensional environment. Similar to how drones are controlled, one joystick controls the planar motion of the moving object. In the context of a Snake Game, it controls the moving directions of the snake; while in the context of a Pong Game, it controls the moving directions of the puddle. Another joystick controls the UP/DOWN motion of the controlled object, allowing the player to break into a new plane level. Finally, the push buttons inside the two joysticks are used to select the type of games and the type of animations the user would like to play or see.

Problems occurred however due to the 0.4s refresh time delays for each loop. The input from the user cannot be obtained until the delay() function finishes execution. This was problematic because it requires the user to keep on holding the joystick for at least 0.4s before the input can be registered. Hence, millis() function is used to replace any delay() in the main loop. Instead of directly putting delay(400) for a 0.4s refresh cycle, a conditional check would be used to check if the (currentTime-initTime) is greater than 400.

Unlike the interrupt push button, the joystick's analog and digital data are obtained by a constant pulling from each loop. Although this method does not respond as fast as an interrupt, it prevents the program from slowing down the cubeRender() function and does not have any significant visual delay while playing the game.

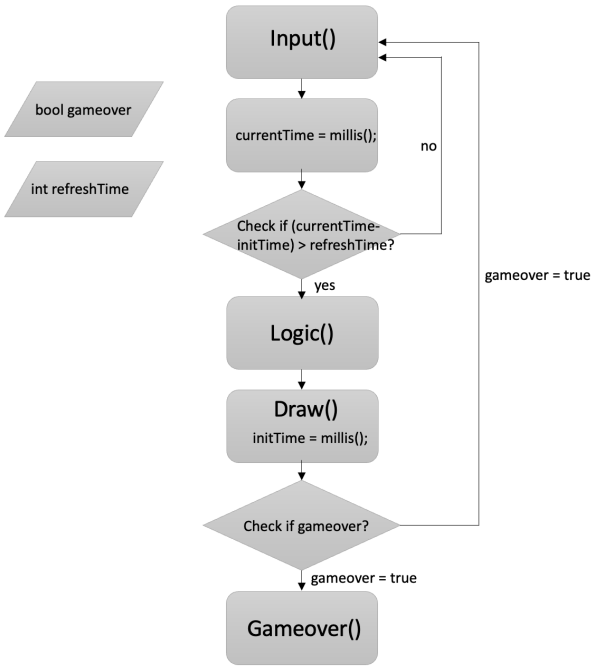

Figure 15: Flow Diagram on Game Input Updates and Logic

6.4 Data Communication Design

This project utilized all the major communication protocols, including direct serial communication, I2C, and SPI.

- **SPI:** SPI communication is generally much faster than the serial ports, hence native SPI support on the Arduino was chosen over the normal serial prots. However, since it is a one-way communication (from master to slave), only three lines are used in our project: MOSI(D51), SS(D53), and SCLK(D52). The built-in Arduino SPI library is used to set up the communication as shown below:

```
SPI.beginTransaction(SPISettings(16000000, MSBFIRST, SPI_MODE0));
```

In order to send the data to the cube as fast as possible, the clock rate is maxed out at 16MHz. And Mode0 represents the falling edge for data capture, and the rising edge for the output signal.

- **Serial Communication:**

  Serial communication is used for receiving the user inputs from the analog pins and digital pins that are connected to the joystick. In addition, the serial port is also used for Serial.print() commands for debugging purposes.

- **I2C:**

  Both the OLED display and the IMU sensor use the I2C communication protocol with an address of 0x3C and 0x86 respectively. The SDA and SCL pins are wired to the D21 and D22 pins on the Mega for data and clock communication. The Arduino Wire library is used to handle the communication, hence data is easily received and sent via the predefined functions.

6.5 State Machine

A state machine diagram of the project is illustrated below for the understanding of the high level of the program. The details in the snake game and pong game have been omitted and simplified into one single state with input conditions Gameover=true to exit, and Gameover=false to stay inside the state.

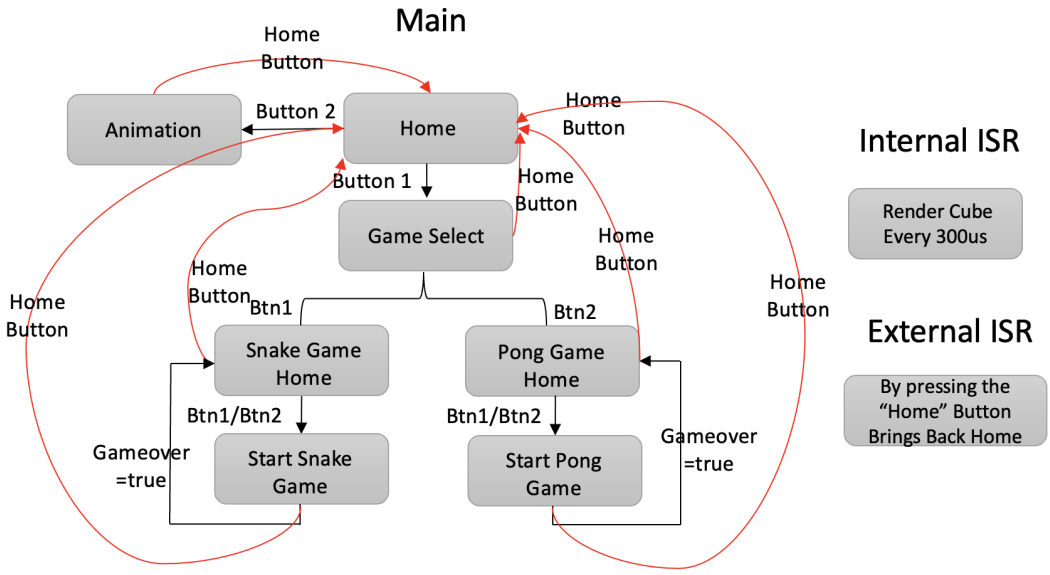

Figure 16: State Machine Illustration

6.6 Code Optimization

The program optimization was crucial for the project to mitigate the potential race conditions mentioned above. Shorter and faster programs allow enough time for the program to store data into the buffer memory location before jumping into the ISRs. The optimization techniques used in this project are listed below:

- **Direct control of the registers in Arduino:** For pin mode setup, instead of using the built-in Arduino library, we controlled them directly using the respective registers. For instance: PORTD - The Port D Data Register; DDRD - The Port D Data Direction Register; PIND - The Port D Input Pins Address.

```
DDRD = B11111111; // set PORTD (digital 7~0) to outputs
PORTD = B11110000; // digital 4~7 HIGH, digital 3~0 LOW
```

- **Bit math instead of decimal math:** Instead of using number 0-7 to represent each coordinate, bits are used for its easier manipulations when it comes to setting and receiving each LED's state. Only one line of code is required to complete this operation.

```
cube[7-y][7-z] |= (0x01 << x);
cube[y][z] &= ~(0x01 << x);
```

- **Using goto statements for switching between different states:** Instead of checking each if statement every time the program loops through, goto and switch statements are used to jump to the targeted program directly to reduce execution time.

7 Results and Outlook

In conclusion, the team learnt a great deal about embedded system throughout this project. We were able to program the LED cube from the lowest-level and build our own library. Animation effects were programmed in order to take full advantage of the three dimensional characteristic of the LED cube. Classic Snake Game and Pong Game were created to achieve our project objective of reinventing retro games.

Future improvements would be to program more console games like: Breakout, Fighter Jet, etc. In addition, we could design a better user interface on the OLED display for switching between different games and modes. The current Arduino Mega could be replaced by an ESP32 for its superior features and higher clock rate. Additional sensors could be added to provide more diverse inputs to the game. For instance, an IMU can help detect the orientation of the cube, so that the games can add in gravitational effects. Finally, a PCB and a portable battery can be used to replace the existing breadboard and laptop(for power supply). Those additions can eventually make the cube become a truly portable 3D game console like the Nintendo Switch on the market.

References

[1] Arduino Project Hub. 2022. How to Interface PS2 Wireless Controller w/ Arduino. [online] Available at: https://create.arduino.cc/projecthub/electropeak/how-to-interface-ps2-wireless-controller-w-arduino-a0a813 [Accessed 22 March 2022].

[2] », M., 2022. 74HC595 Shift Register Demystified. [online] Instructables. Available at: https://www.instructables.com/74HC595-Shift-Register-Demistified [Accessed 22 March 2022].

[3] "74HC595 Shift Register", Adafruit Learning System, 2022. [Online]. Available: https://learn.adafruit.com/74hc595/pinouts. [Accessed: 14- Apr- 2022].

[4] "In-Depth: Interface OLED Graphic Display Module with Arduino", Last Minute Engineers, 2022. [Online]. Available: https://lastminuteengineers.com/oled-display-arduino-tutorial/. [Accessed: 14- Apr- 2022].

[5] "Shift Registers - 74HC595 74HC165 with Arduino", DroneBot Workshop, 2022. [Online]. Available: https://dronebotworkshop.com/shift-registers/. [Accessed: 14- Apr- 2022].

[6] "IRF520 N-Channel Power MOSFET", Components101, 2022. [Online]. Available: https://components101.com/mosfets/irf520-pinout-datasheet-features. [Accessed: 14- Apr- 2022].

[7] "How to Interface A Passive Buzzer Module with Arduino - ElectroPeak", Electropeak, 2022. [Online]. Available: https://electropeak.com/learn/interfacing-passive-buzzer-module-with-arduino/. [Accessed: 14- Apr- 2022].

Appendices

A Circuit Breadboard Diagram on Fritzing

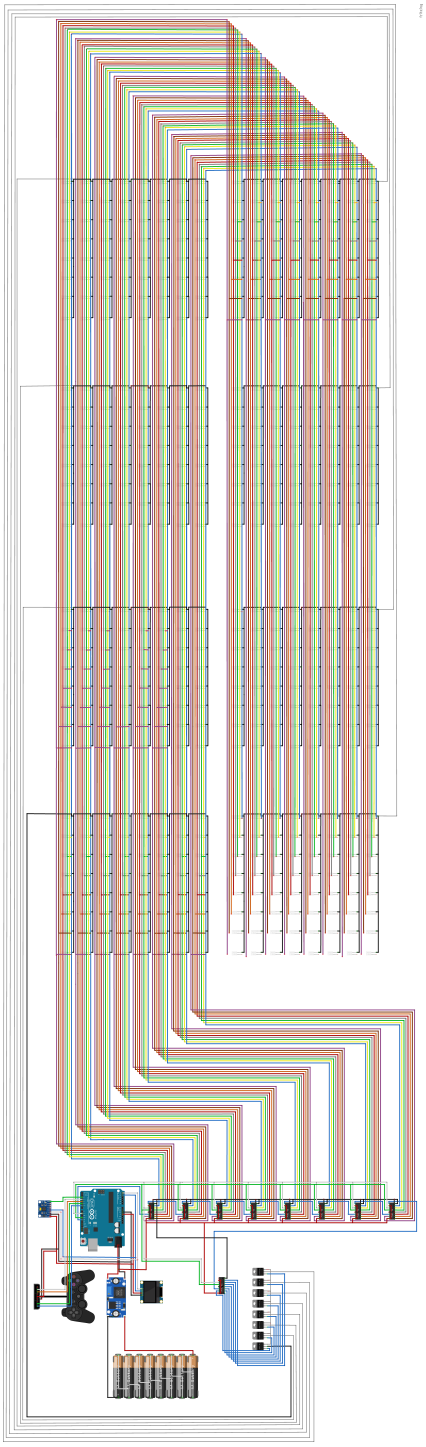

Figure 17: Full Circuit Diagram on Fritzing

B Circuit Schematics on KiCAD

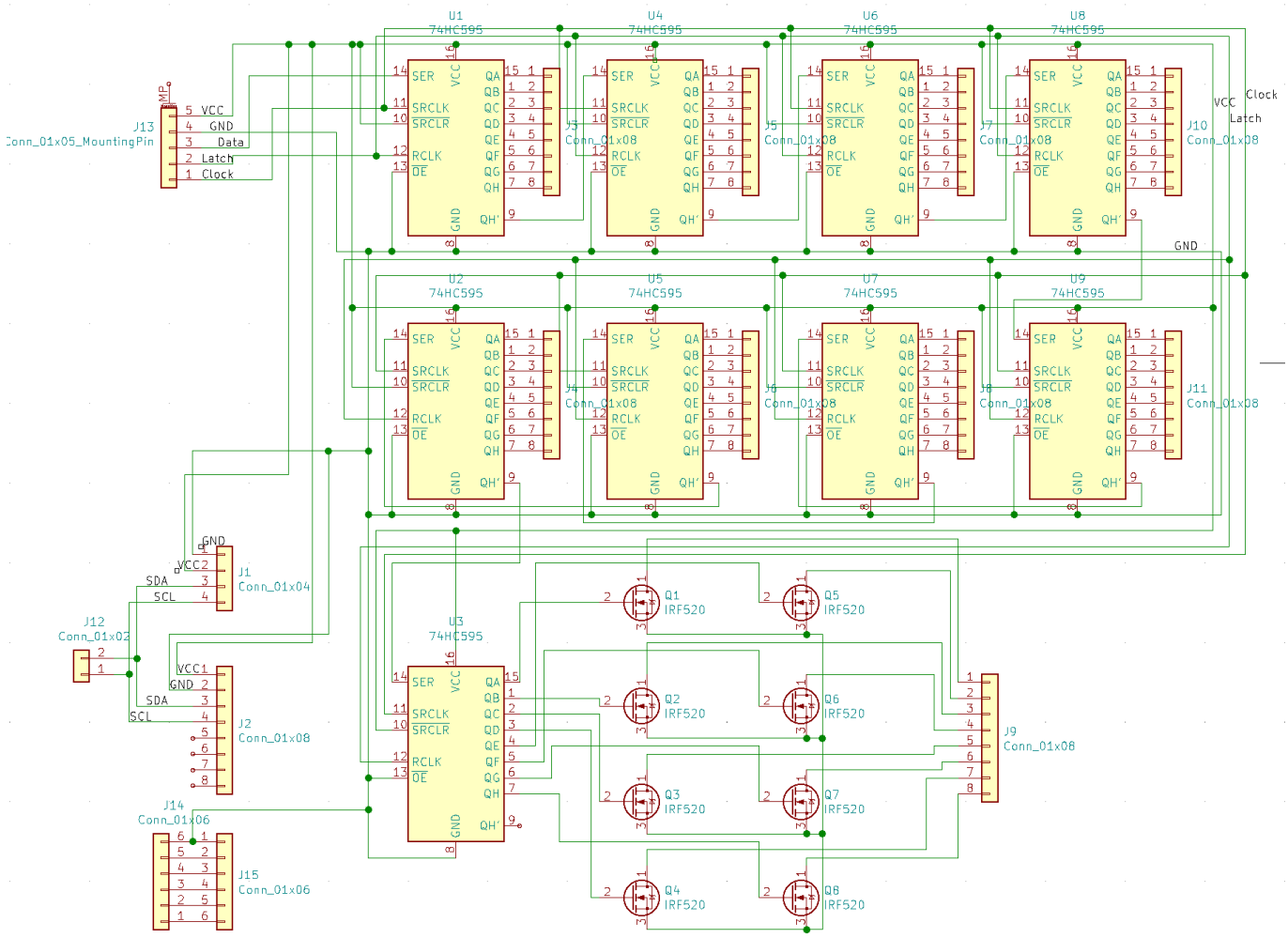

Figure 18: Circuit Schematics Design

C PCB Board Layout on KiCAD

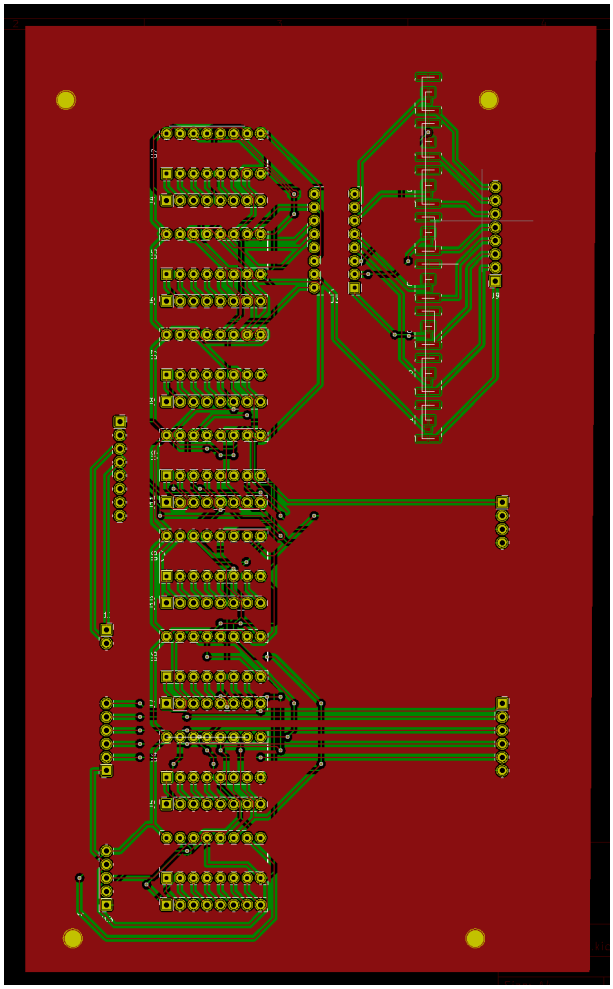

Figure 19: Circuit PCB Design

D Bit Map Generator Zip File By VolosR

Custom application for OLED animations:

E LED Cubic Console Source Code

Link to Team's GitHub Page

F Links to Other Similar LED Cube Projects Team Have Studied From

Link to Cornell University Student Project
YouTube Tutorial on How to Construct an LED Cube
YouTube Video on Theory Behind GRB LED Cube

G Link to Published YouTube Demo Video

Link to Demo Video