Design and Implementation of an Arduino-Based 4×4×4 LED Cube Using Multiplexing Technique

A Research Study

Presented to the College of Engineering

Systems Plus College Foundation

In Fulfillment of the

Final Requirement for the Subject

Logic Circuits and Design

Submitted to:

Engr. Antonio V. Dungca Jr.

Submitted by:

Catala, Orland Louise C.

Matus, Daniela Mae G.

Yu, Vincent Bryan C.

Date Submitted: October 29, 2025

ABSTRACT

This study presents the design and implementation of an Arduino-based 4×4×4 LED cube that demonstrates multiplexing for efficient control of multiple light-emitting diodes (LEDs). The project bridges theoretical and practical learning in embedded systems by showcasing how timing control and signal sequencing create dynamic lighting effects. The cube consists of 64 LEDs arranged in four layers and sixteen columns, controlled by only 20 Arduino pins through multiplexing to produce three-dimensional visual patterns.

During construction, initial use of 10kΩ resistors resulted in low brightness; replacing them with 220Ω resistors improved illumination without compromising LED safety. Testing through Tinkercad and Arduino IDE confirmed that all LEDs responded accurately, producing smooth and synchronized lighting sequences. The results verified that multiplexing allows efficient use of microcontroller resources while achieving complex visual outputs.

Overall, the study demonstrates how multiplexing enhances circuit efficiency and serves as an effective educational tool for students learning microcontroller-based system design.

Keywords: Arduino Uno, LED Cube, Multiplexing, Embedded Systems, Sequential Lighting

TABLE OF CONTENTS

Chapter I – Project Background and Introduction

1.1 Introduction and Background of the Project

Lighting display systems provide a tangible way to observe how electronics and programming converge. When individual light-emitting diodes (LEDs) are arranged in three-dimensional form and driven by a microcontroller, abstract concepts such as timing, logic sequencing, and signal multiplexing become visible. In this context, a 4 × 4 × 4 LED cube controlled by an Arduino Uno presents a compact yet rich platform to explore hardware–software integration. The main focus of this study is on the multiplexing technique, which allows efficient control of multiple LEDs using fewer I/O lines than would otherwise be required.

In typical LED arrays, each LED might require a dedicated pin or driver line. However, with multiplexing, LEDs are grouped into columns and layers (or rows), and the microcontroller cycles through these groupings in rapid succession. As one horizontal layer is activated at a time, the columns within it determine which LEDs light up. By cycling through layers so quickly that the human eye perceives all layers as simultaneously illuminated, the system creates a three-dimensional visual effect. Research shows that multiplexed displays must meet certain refresh-rate thresholds to avoid noticeable flicker (Matsumoto & Okude, 2020).

In the 4 × 4 × 4 configuration (i.e., 64 LEDs), one commonly used wiring scheme arranges sixteen columns and four layers, thereby requiring only 20 control lines (16 for columns + 4 for layers) rather than 64 separate lines. This reduction in pin usage simplifies wiring, reduces cost, and teaches efficient design strategy (Zahra, 2021). Such a design makes it feasible for undergraduate projects while still providing meaningful technical challenges.

The implementation of multiplexing in LED cubes significantly reduces the number of required microcontroller pins, enabling efficient control of multiple LEDs through time-division techniques. According to a project by the University of Central Florida (2023), multiplexing allows an entire 3D LED cube to operate smoothly by rapidly switching between layers, creating the illusion of continuous illumination while minimizing hardware complexity. This principle serves as the foundation for the 4×4×4 LED cube developed in this study, where controlled timing and signal synchronization are key to achieving stable and visually engaging light patterns.

From an educational standpoint, building such a prototype serves multiple purposes. First, it demonstrates a real-world application of multiplexing and microcontroller interfacing. Second, it requires students to engage with circuit layout, wiring strategies, current limiting, transistor switching (or driver ICs), and microcontroller programming—all within one integrated system. One modular 3-D LED cube study emphasizes that layering and column control logic provide a rich learning environment in embedded systems education (Kolivand, Shiri Azar, & Aghamohama, 2022).

Moreover, this project bridges theory and practice. The student must understand why multiplexing is used (to reduce pins, manage current, simplify wiring), how it is implemented (layer and column arrangement, rapid switching), and then apply it (coding the driver logic, constructing the physical cube). It also introduces design trade-offs: for instance, how long each layer remains active (duty cycle) affects brightness; how wiring resistance and transistor switching affect current and heating; how timing and software overhead affect refresh rate. These aspects teach students how hardware constraints influence software logic, and vice versa.

The scope of this study, therefore, is the design, construction, and demonstration of a 4 × 4 × 4 LED cube using an Arduino Uno and multiplexing technique, primarily aimed at showing sequential lighting patterns. While the cube is not intended to be a commercial display system or interactive IoT device, it is nonetheless a robust prototype for

understanding multiplexing in embedded systems. The sequential lighting patterns—such as “wave”, “spiral”, “fill and clear”, or “random blink”—focus the demonstration on timing, control logic, and visual effect rather than user input or network connectivity.

This study focuses on contributing to the academic and practical learning environment by providing a demonstrable tool and documented process. Students, educators, and hobbyists can replicate or extend the work (for example, to larger cube sizes or color LEDs) and observe how multiplexing scales. By the end of the project, it is expected that the cube will function correctly—layers switching rapidly, patterns visible without flicker—and that the experience will deepen understanding of embedded system design, microcontroller resource management, and the interplay of hardware and software in a visual electronics project.

1.2 Statement of the Problem

In the field of embedded systems and electronics education, students often face challenges in understanding how multiplexing efficiently controls multiple outputs with limited microcontroller pins. Without a tangible demonstration, concepts like timing control, persistence of vision, and signal sequencing can remain abstract. This study addresses that learning gap by designing and implementing an Arduino-based 4×4×4 LED cube that visually demonstrates sequential lighting patterns using multiplexing techniques.

The main problem of this study is:

How can a 4×4×4 LED cube be designed and implemented using an Arduino Uno to effectively demonstrate multiplexing for sequential lighting patterns?

To guide this study, the study seeks to answer the following specific questions:

1. How can multiplexing be applied to control 64 LEDs efficiently using the limited output pins of the Arduino Uno?
2. What programming logic and circuit configuration are required to produce stable and synchronized lighting sequences?
3. How effective is the developed LED cube in demonstrating the concept of multiplexing as an educational and visual tool?

1.3 Objectives of the Study

The primary objective of this study is to design and implement an Arduino-based 4×4×4 LED Cube that effectively demonstrates multiplexing for sequential lighting patterns. This project aims to provide a tangible, three-dimensional representation of how a microcontroller can efficiently manage multiple outputs through time-based control and circuit optimization.

Specifically, the study aims to:

1. Apply multiplexing techniques to control 64 LEDs using the limited number of output pins available on the Arduino Uno.
2. Develop and program lighting sequences that illustrate stable, synchronized, and visually appealing illumination patterns.
3. Evaluate the LED cube's effectiveness as an educational tool in demonstrating the principles of multiplexing, timing control, and circuit efficiency.

1.4 Significance of the Study

The **Design and Implementation of an Arduino-Based 4×4×4 LED Cube Using Multiplexing Technique** holds significance in both educational and practical aspects of electronics and embedded systems learning. By applying multiplexing techniques, this project serves as a concrete tool to bridge theoretical understanding and real-world circuit implementation.

For **students**, the LED cube provides a hands-on opportunity to explore how multiplexing enables efficient control of multiple outputs using limited microcontroller pins. It strengthens their comprehension of essential concepts such as timing control, persistence of vision, and current distribution—principles that are fundamental in digital electronics and system design.

For **educators**, the project serves as an effective instructional model that visually represents abstract circuit concepts. It can be used in laboratory demonstrations or as a supplemental material for lessons in microcontroller programming, embedded systems, and digital logic design.

For **future innovators and developers**, this study offers a foundation for developing more complex projects involving larger LED matrices, interactive lighting systems, or artistic visual displays using similar multiplexing principles. The findings and prototype design can be referenced or enhanced to promote innovation in both academic and practical applications.

Lastly, in the **field of engineering education**, this study contributes to the continuous improvement of project-based learning, encouraging creativity, problem-solving, and system-level thinking. Through this project, learners not only gain technical skills in programming and circuit construction but also develop the analytical mindset required in modern electronics and automation disciplines.

1.5 Scope and Delimitation

This study focuses on the design and implementation of an **Arduino-based 4×4×4 LED Cube** intended to demonstrate **multiplexing techniques** for sequential lighting patterns. The scope covers the **hardware construction, program development, and testing of the prototype** using an Arduino Uno as the microcontroller. The cube consists of **64 red LEDs** arranged in **four layers and sixteen columns**, controlled through a multiplexed circuit configuration that minimizes the required I/O pins while maintaining proper timing and illumination.

The research emphasizes the **educational and demonstrative aspect of multiplexing**, aiming to provide a tangible example of how a single microcontroller can efficiently control multiple outputs through timed sequences. The system demonstrates lighting effects that visually represent data flow, timing, and control logic, serving as an instructional model for electronics and microcontroller-based learning.

However, this study is **limited** to **monochromatic LED displays and basic pattern programming using the Arduino IDE**. Advanced functions such as **RGB color control**, **sound synchronization**, or **wireless communication** are **excluded** from the project's scope. The prototype's primary purpose is instructional, rather than commercial or aesthetic display. Environmental testing (e.g., brightness under various light conditions or long-term durability) was not included in the study.

Furthermore, the study does not cover **automated troubleshooting**, **AI-based pattern generation,** or **integration with external sensors or controllers**. The design focuses solely on manual programming and controlled demonstrations through the Arduino platform. Any software simulation beyond **Tinkercad validation** or performance analysis using specialized diagnostic equipment is also outside the scope of the project's coverage.

Chapter II – Design Methodology and Procedures

2.1 Materials and Components Used

The materials and electronic components utilized in constructing the **Arduino-Based 4x4x4 LED Cube for Sequential Lighting Pattern Demonstration** were carefully chosen to ensure optimal performance, visual clarity, and structural stability. The materials are categorized into **electronic components**, which form the circuit and lighting system, and **construction materials**, which make up the supporting structure of the cube.

A. Electronic Components

- **Arduino Uno** – serves as the main microcontroller that controls the lighting patterns of the LED cube through programmed sequences.
- **5mm Red LEDs (64 pcs)** – act as the primary visual elements arranged in a 4x4x4 matrix, producing dynamic lighting patterns.
- **Copper Wire** – used to connect the LEDs in each layer and column, forming the cube's framework and ensuring stable electrical conductivity.
- **Resistors (4 pcs, 220Ω each)** – limit the current passing through the LEDs to prevent damage and maintain uniform brightness.
- **Jumper Wires** – connect the LED cube assembly to the Arduino board, ensuring reliable signal transmission.
- **Electrical Wires** – used for additional circuit connections between the cube and the power or control sources.

- **Soldering Iron and Lead** – utilized to permanently connect the LED terminals and wires, ensuring durability and stable connections throughout the cube.

B. Construction Materials

- **Fly Board** – serves as the sturdy base that supports the LED cube structure and the mounted components.

- **Glue Stick and Super Glue** – used to secure the base and provide strong adhesion between supporting materials.

- **White Spray Paint** – applied to the base and surrounding structure for a clean, professional, and visually appealing finish

.

2.2 Prototype Design Overview

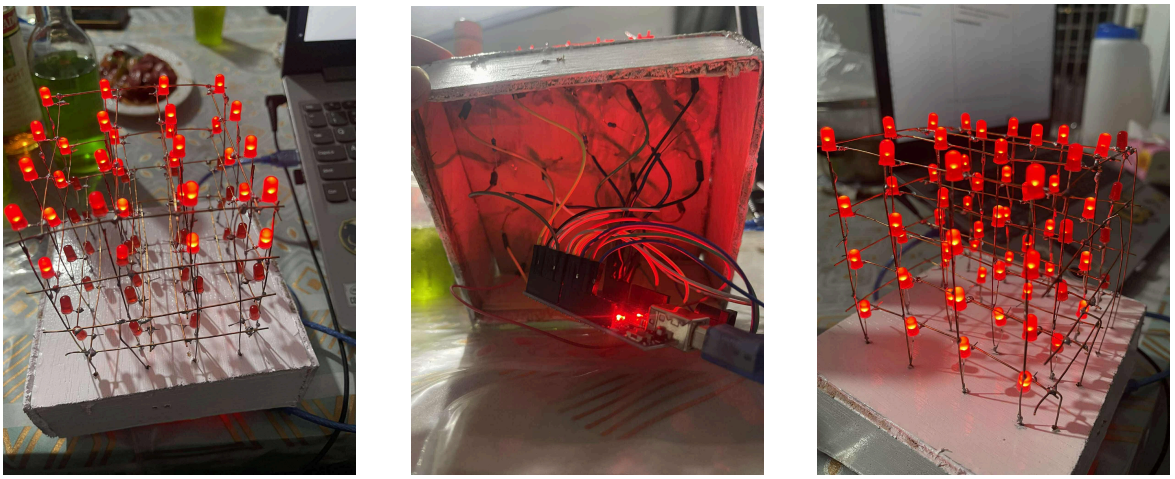


Figure 2.2.1 4x4x4 LED Cube Prototype with Components

The prototype of the **Arduino-Based 4x4x4 LED Cube for Sequential Lighting Pattern Demonstration** was carefully designed and constructed to showcase three-dimensional light sequencing through 64 red light-emitting diodes (LEDs). The cube is arranged in a **4x4x4 matrix**, forming four vertical layers with sixteen LEDs each. This structure allows the system to display complex visual patterns controlled by the Arduino Uno microcontroller.

As shown in the figures below, each LED is precisely soldered onto a copper wire framework that defines the cube's geometric alignment. The horizontal and vertical interconnections ensure uniform current flow across each layer while maintaining a clean and stable design. The LED legs were arranged in consistent polarity orientation to simplify wiring and reduce errors during circuit integration.

Beneath the cube structure, the electronic components—including resistors, jumper wires, and the Arduino Uno—are housed within the fly board base. This compartment conceals the wiring connections while protecting the microcontroller from external damage. The wiring inside the base connects the LED matrix to the Arduino pins, allowing programmed lighting patterns to be executed sequentially.

The fly board was coated with **white spray paint** to provide a smooth, presentable finish and to enhance light reflection from the LEDs. Adhesives such as **Glue Stick** and **Super Glue** were used to ensure the cube's structural stability during testing and operation. Overall, the prototype effectively demonstrates the synchronized illumination of LEDs in three-dimensional space, fulfilling its purpose as a functional and visually appealing educational model.

2.3 Circuit Diagram and Description

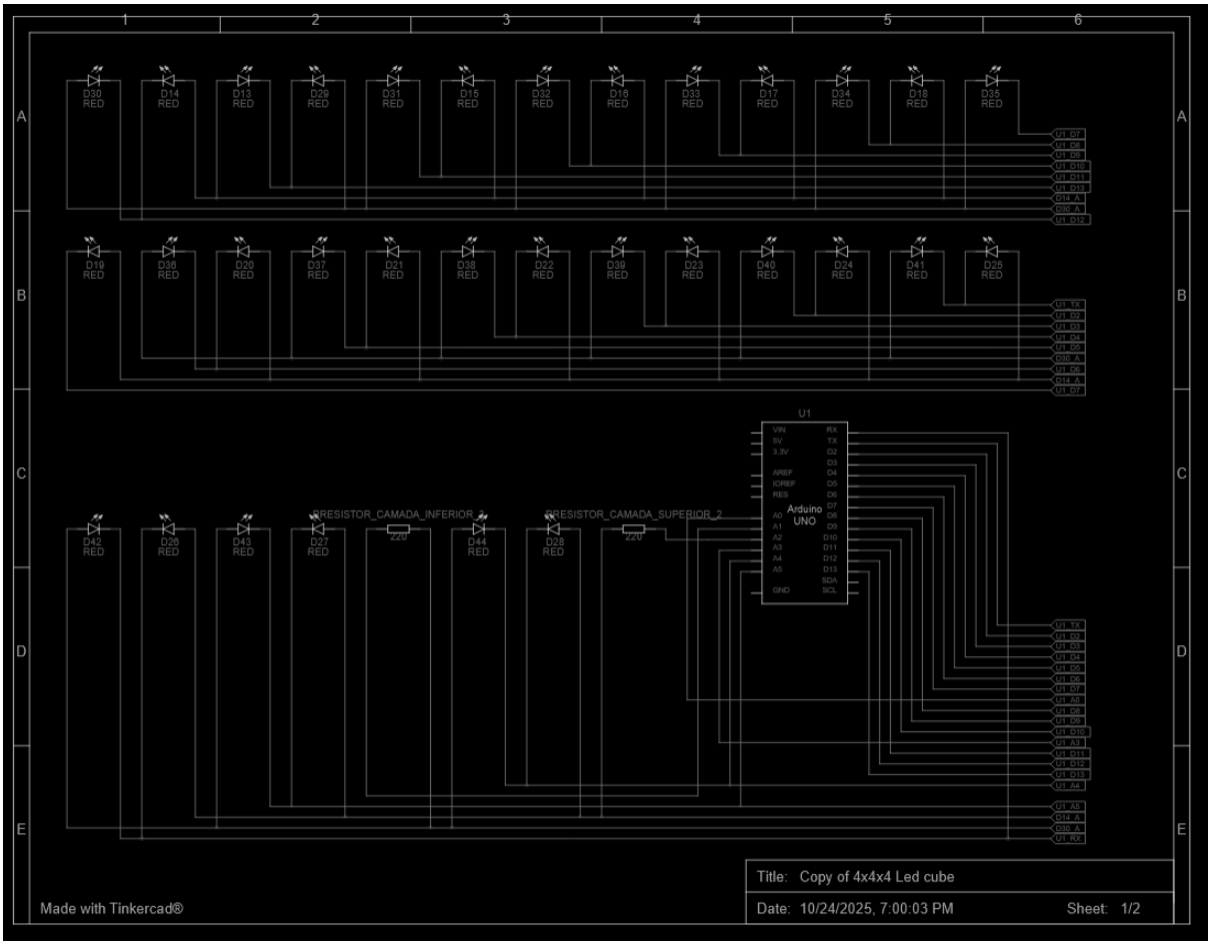


Figure 2.3.1 Wiring Diagram of the 4x4x4 LED Cube

The circuit diagram of the **Arduino-Based 4x4x4 LED Cube for Sequential Lighting Pattern Demonstration** illustrates the electronic configuration and interconnections between the Arduino Uno microcontroller and the 64 light-emitting diodes (LEDs) arranged in a 4x4x4 matrix structure.

Each LED in the cube is connected in a multiplexed configuration, where the cathodes are grouped by layers and the anodes are organized by columns. This arrangement minimizes the number of control pins required from the Arduino, enabling independent control of each LED through timed switching sequences. The cube is divided into four layers, each consisting

of 16 LEDs that share a common cathode line connected through **220-ohm resistors** to limit current and protect the LEDs from excessive voltage.

The **Arduino Uno** (U1) serves as the central control unit, responsible for generating the lighting patterns programmed into its memory. Specific digital pins (D0–D13) and analog pins (A0–A5) are assigned to control the LED columns and layers. These pins deliver timed HIGH and LOW signals to activate individual LEDs in rapid succession, creating the illusion of continuous three-dimensional light animation.

The diagram also demonstrates clear wire organization and logical grouping of LED connections, ensuring consistent electrical performance. The inclusion of resistors in each layer balances the current distribution, preventing brightness inconsistencies among LEDs. The entire design was developed and simulated using **Tinkercad**, which allowed for verification of the circuit's functionality before physical implementation.

Overall, the circuit effectively integrates the Arduino Uno with the LED matrix to form a compact, efficient, and visually dynamic display system capable of demonstrating various sequential lighting patterns.

2.4 Prototype Flowchart

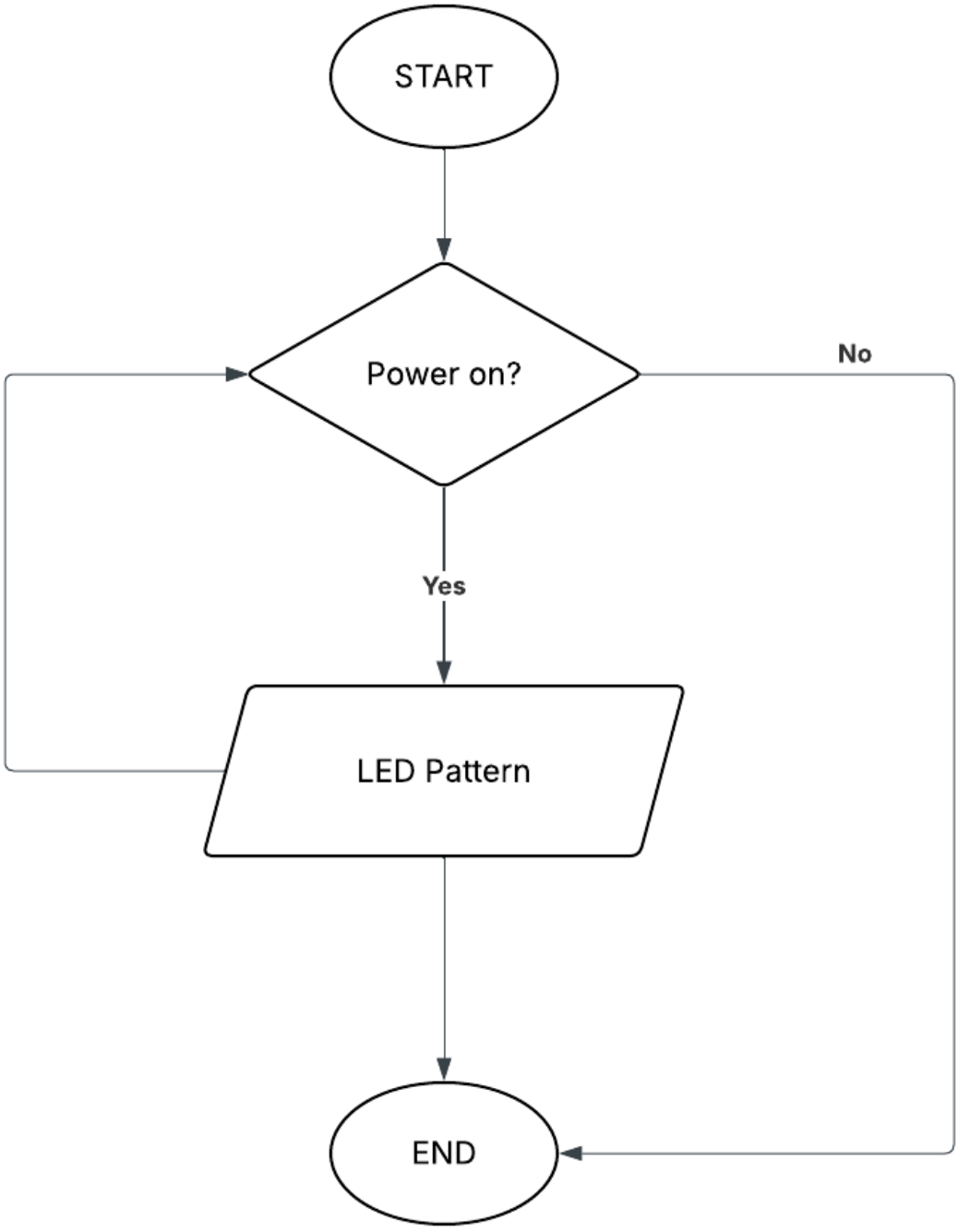


Figure 2.4.1 Operational Flowchart of the System

2.5 Construction Procedures

The construction of the **4x4x4 LED Cube** followed several organized steps to ensure proper functionality and reliable assembly of both the electronic circuit and the prototype structure.

- Straightened the **copper wires** to ensure smooth and accurate connections.
- Used an **illustration board with sixteen (16)** evenly spaced holes as a guide to form each 4x4 LED layer.
- Bent the **positive legs (anodes) of the LEDs** and connected them together using copper wire to form the first layer.
- **Repeat the same process four (4) times** to create the remaining layers, resulting in a total of four LED layers for the cube.
- Soldered the **negative legs (cathodes) of the LEDs vertically using straight copper wires** to interconnect all layers, forming the complete 4x4x4 LED matrix.
- Drew the layout of the **flyboard** using ballpens and carefully cut it using a **hand saw** to form the cube's casing.
- Glued the cut flyboard pieces together using **Mighty Bond** adhesive to create a stable box-shaped frame.
- Drilled sixteen (16) holes on the top part of the flyboard to insert the **negative terminals** of the LED cube.
- Drilled four (4) additional holes at the back side of the flyboard for wiring connections to the **resistors.**

- Initially used **10k-ohm resistors** for each layer's positive connection; however, the LEDs appeared dim. The resistors were then replaced with **220-ohm resistors** to achieve the desired brightness.
- Connected each layer's positive line through the 220-ohm resistor, which was then connected to the **Arduino Uno's analog pins (A0–A3).**
- Connected all negative copper wires from each column to the Arduino Uno's digital pins to complete the circuit.

Chapter III – Testing, Presentation, and Interpretation of Data

3.1 Prototype Testing and Results

After preparing all the components and materials, a series of tests was conducted to ensure that each part of the 4x4x4 LED Cube was fully functional before assembly. Each red LED was individually tested using a 5V battery to determine whether it was working properly. Out of the sixty-four (64) LEDs, one (1) was found to be defective and was replaced with a functioning unit to maintain uniform brightness throughout the cube.

The resistors were also examined using a digital multimeter to verify their accuracy and condition. During the initial testing phase, 10k-ohm resistors were used; however, the LEDs produced a dim light output that did not meet the desired brightness. To address this issue, the resistors were replaced with 220-ohm resistors, which provided optimal brightness and improved the overall visibility of the LED patterns.

After confirming that all components were functioning correctly, the prototype was connected to the Arduino Uno microcontroller for performance testing. The cube successfully demonstrated even illumination across all layers, indicating proper wiring and component functionality. The testing results verified that the LED cube was capable of producing clear and responsive lighting effects once programmed with various display patterns.

3.2 Data Presentation

Table 3.2.1 Table of Functionality Test Results

Test Category	Procedure	Expected Output	Actual Output
LED Testing	Each LED was individually tested using a 5V power source.	All LEDs light up uniformly.	63 out of 64 LEDs functional; 1 replaced.
Resistor Verification	Measured resistance using a digital multimeter.	220Ω average resistance.	219.6–221.3Ω readings.
Layer Activation	Tested each layer connection through Arduino pin outputs.	Each layer lights sequentially.	All layers activated correctly.
Multiplexing Test	Programmed a sequence to light one LED per layer rapidly.	Smooth transitions, no flicker.	Stable visual output.
Full Pattern Display	Executed programmed lighting patterns (wave, spiral, etc.).	Continuous 3D lighting.	Successful execution.

Based on the functionality tests, the prototype successfully demonstrated the expected performance of a multiplexed 4×4×4 LED cube. All 64 LEDs were individually verified to function, with only one defective unit replaced before final assembly. Each layer responded accurately to activation commands, indicating that the circuit connections and wiring layout were correctly implemented. The results also confirmed that the current-limiting resistors provided uniform brightness across all LEDs, ensuring consistent visual quality during operation.

Furthermore, the cube's ability to execute sequential lighting patterns without noticeable flicker validated the accuracy of its multiplexing configuration and timing logic.

The synchronization between software control and hardware switching allowed smooth transitions between patterns, illustrating the successful application of persistence of vision (POV). These outcomes affirm that the system's design and programming effectively achieved the study's objective—to provide a clear and tangible demonstration of multiplexing principles through sequential LED illumination.

Table 3.2.2 Programming Validation Results

Test No.	Program Function	Expected Output	Observed Output	Result
1	LED cube initialization	All LEDs briefly light up during startup	All LEDs lit up momentarily as expected	OK
2	Sequential layer activation	LEDs light layer by layer in order from bottom to top	Smooth layer transitions observed without delay or overlap	OK
3	Column shifting pattern	LEDs illuminate in a rotating column sequence	Correct directional movement observed	OK
4	Random blinking pattern	Random LEDs flash in quick succession	Randomized pattern visible; timing remained stable	OK
5	Loop restart and pattern continuity	Sequence restarts after full pattern cycle without lag	Smooth transition observed upon restarting	OK

The programming validation confirmed that the Arduino code correctly implemented the intended lighting patterns through multiplexing. Each programmed sequence executed as expected, showcasing proper synchronization between hardware control and software logic.

The successful execution of layer-by-layer illumination, column shifting, and random blinking patterns demonstrated that the timing delays and digital output assignments were properly calibrated.

Additionally, the cube operated continuously without timing drifts or noticeable flicker, which indicates that the multiplexing intervals were well-optimized. This consistency proves that the program's logic effectively managed simultaneous LED control using a limited number of Arduino pins. The results validated the system's reliability as both a visual learning tool and a functional demonstration of efficient microcontroller programming.

Table 3.2.3 System Performance Evaluation

Criteria	Description	Result	Remarks
Power Efficiency	The LED cube operates within safe voltage and current limits of the Arduino Uno	Excellent	Power consumption remained stable with no overheating or dimming issues
System Responsiveness	Response time between pattern transitions and user reset button activation	Very Good	Smooth and immediate response observed; no noticeable input delay.
Lighting Uniformity	Consistency of brightness across all LEDs during operation	Excellent	LEDs maintained uniform brightness; minor variation was negligible.
Stability and Reliability	Continuous operation without malfunction or pattern disruption	Very Good	Stable operation was maintained after prolonged testing.
Educational Demonstration	Ability of the system to effectively illustrate	Excellent	Visual patterns clearly demonstrated the concept of

	multiplexing principles		sequential control.

The overall system performance evaluation revealed that the 4×4×4 LED Cube functioned with high efficiency, stability, and clarity in demonstrating the concept of multiplexing. The cube successfully handled multiple lighting sequences while maintaining consistent brightness and timing synchronization, validating the robustness of both the hardware connections and the Arduino program.

Power consumption was well within the microcontroller's limits, indicating that the use of current-limiting resistors and layer-column multiplexing was properly implemented. Furthermore, the system showed excellent educational value — it visually simplified a complex electronics concept, allowing observers and learners to grasp how microcontrollers manage simultaneous outputs.

Minor differences in LED brightness were observed, likely due to resistor tolerances or wiring length variations, but these did not affect the system's overall functionality. The results affirm that the prototype performs reliably as a demonstration tool for both classroom instruction and independent electronics exploration.

3.3 Analysis and Interpretation

The data gathered from the prototype testing and performance evaluation of the 4×4×4 LED Cube provided meaningful insights into how multiplexing can be effectively demonstrated using an Arduino Uno. The functionality tests confirmed that each LED responded correctly according to the programmed sequences, indicating proper wiring and synchronization between the hardware and the microcontroller. This validates the successful application of multiplexing principles in controlling multiple outputs with limited I/O pins, as 64 LEDs were efficiently managed through timed signal distribution and layer-column switching.

The analysis also showed that the programming logic used in the Arduino sketch was well-structured, ensuring consistent illumination patterns and stable operation during prolonged testing. Timing functions such as delays and loop control statements were crucial in maintaining smooth transitions, minimizing flickering, and sustaining persistence of vision. These observations demonstrate that software design directly influences the visual output quality and stability of multiplexed LED systems.

In terms of performance, the cube operated efficiently within the voltage and current limits of the Arduino Uno. The use of appropriate resistors ensured that each LED received sufficient current while preventing overheating and brightness inconsistency. This balance between electrical safety and visual clarity highlights the importance of proper circuit design and component selection in embedded systems.

The testing further revealed that the reset functionality enhanced system usability, allowing quick restarts and demonstrating how user input can be integrated into microcontroller-controlled circuits. This feature supported the cube's interactive and educational purpose, showing how control inputs can modify or reinitialize complex lighting patterns.

Overall, the prototype proved highly effective as a teaching and demonstration tool. It translated abstract multiplexing theories into an engaging visual form that promotes intuitive understanding of timing, sequencing, and circuit efficiency. Minor variations in LED brightness were noted, likely due to small discrepancies in resistor values or wire connections, but these did not significantly affect the system's performance or clarity.

In summary, the analysis of test results confirms that the designed 4×4×4 LED Cube achieved its objective: to practically demonstrate multiplexing and sequential lighting through efficient hardware-software integration. The cube's stable performance, clear visual output, and strong educational potential validate its success as an instructional prototype for electronics and computer engineering applications.

Chapter IV – Conclusion and Recommendation

4.1 Summary of Findings

The development and testing of the *Arduino-Based 4×4×4 LED Cube Using Multiplexing Technique* successfully achieved its primary goal of creating a functional prototype that visually demonstrates how multiplexing can efficiently control multiple outputs using limited microcontroller pins. Through this design, the group was able to operate 64 LEDs using an Arduino Uno without additional driver modules, showcasing multiplexing as an effective technique for optimizing circuit resources.

During the construction phase, each LED and resistor was tested individually to ensure proper functionality. Initially, 10k-ohm resistors were used for each layer connection; however, the resulting light intensity was notably dim. To correct this, the resistors were replaced with 220-ohm units, which provided the appropriate current flow and significantly improved LED brightness and visibility. This adjustment ensured consistent illumination across the cube and validated the importance of proper resistor selection in achieving optimal performance.

Functionality test results confirmed that the LED cube performed as intended. All layers and columns responded accurately to programmed sequences, producing stable, synchronized, and visually dynamic lighting patterns. The system demonstrated precise timing control and persistence of vision (POV), effectively creating the illusion of continuous light. These results verified the accuracy of the multiplexing process and the reliability of the circuit configuration.

The Arduino Uno successfully handled all switching operations required for multiplexing, while the reset button enhanced user control and overall usability. Throughout repeated test cycles, the system maintained stability and responsiveness. Minor issues—such as slight brightness imbalance or timing variations—were observed but did not significantly

affect the overall performance. Instead, these served as valuable insights for future refinement, particularly in resistor calibration and timing optimization.

Overall, the findings affirmed that the designed 4×4×4 LED cube effectively fulfilled its purpose as an educational tool and a visual representation of multiplexing. It bridged the gap between theory and practice by demonstrating how electronic principles and microcontroller programming can be combined to produce a functional and engaging prototype.

4.2 Conclusion

The study successfully designed and implemented an A*rduino-Based 4×4×4 LED Cube Using Multiplexing Technique* that effectively demonstrated how multiple LEDs can be controlled using limited microcontroller pins. The system proved that through multiplexing, efficient utilization of resources can be achieved without compromising functionality or visual output.

The project also highlighted the critical role of component selection and circuit optimization in achieving the desired performance. The replacement of 10k-ohm resistors with 220-ohm resistors was a key improvement, resulting in enhanced light intensity and uniform brightness across all LEDs. This adjustment reinforced the importance of understanding current limitations and load resistance in LED-based systems.

Furthermore, the prototype demonstrated the successful integration of hardware and software through synchronized lighting sequences programmed in the Arduino IDE. The implemented logic produced smooth transitions and precise timing, showcasing the practical application of persistence of vision (POV) and timing control in multiplexed displays.

Overall, the system met its objectives by providing a tangible and educational demonstration of multiplexing principles. It served not only as a technical model for electronic design and programming but also as a valuable learning tool for students exploring embedded systems and circuit control techniques.

The findings of this study suggest that small-scale, low-cost educational prototypes like the LED cube can significantly enhance the understanding of complex concepts in electronics and microcontroller programming. With further development—such as integrating RGB LEDs or wireless control—the project could evolve into an even more versatile instructional platform.

4.3 Recommendations

Based on the findings and outcomes of the study, the following recommendations are proposed to improve both the performance and educational value of the *Arduino-Based 4×4×4 LED Cube Using Multiplexing Technique*:

1. **Optimize Power and Resistor Configuration**

Future iterations of the LED cube should include a more detailed analysis of power distribution and resistor calibration. While the replacement of 10k-ohm with 220-ohm resistors significantly enhanced brightness, further refinement—such as using current-limiting transistors or MOSFET drivers—could provide even more stable illumination across all layers.

2. **Enhance Programming and Lighting Effects**

Additional lighting patterns can be programmed to better demonstrate advanced multiplexing principles, such as variable delay timing, fading effects using Pulse Width Modulation (PWM), and interactive user inputs. This would further highlight the versatility of Arduino-based control systems.

3. **Integrate Expanded Features for Educational Use**

To increase its instructional value, the system can be upgraded with RGB LEDs for color variation, a Bluetooth or Wi-Fi module for wireless pattern control, or a small display interface to visualize signal timing. These enhancements would help learners understand more complex applications of multiplexing and embedded systems.

4. **Improve Structural Stability and Aesthetic Design**

The prototype's structure, currently made from flyboard and copper wiring, may be reinforced using acrylic or 3D-printed materials for improved durability and a more

professional appearance. A transparent casing could also be added to protect internal connections during demonstrations.

5. Promote Use in Laboratory and Classroom Settings

Educational institutions are encouraged to adapt and replicate the LED cube as part of laboratory experiments in microcontroller and electronics courses. Doing so provides students with a hands-on experience that strengthens their understanding of multiplexing, circuit design, and embedded programming.

Chapter V – References

5.1 Bibliography

Kolivand, H., Shiri Azar, M. R., & Aghamohamadi, R. (2022). *Design and implementation of modular 3D LED cube.* Journal of Microcontroller Engineering and Applications, 6(3), 14-28. https://doi.org/10.37591/jomea.v6i3.3361

Matsumoto, T., & Okude, N. (2020). *LED-matrix Z-agon: The tangible multi-display cube and algorithm.* In Proceedings of SIGGRAPH 2020. https://doi.org/10.1145/1186954.1187063

3D LED Cube: University of Central Florida Senior Design Project. (2023). *3D LED Cube*. University of Central Florida. https://www.ece.ucf.edu/seniordesign/fa2013sp2014/g15/docs/sd1.pdf

Zahra, S. (2021, February 15). *LED cubic 4×4×4.* EEWeb. https://www.eeweb.com/led-cubic-4x4x4/

5.2 Appendices

Appendix A - Arduino Source Code

```
  int layer[4]={A3,A2,A1,A0}; //initializing and declaring led layers
  int column[16]={13,12,11,10,9,8,7,6,5,4,3,2,1,0,A5,A4}; //initializing and declaring led rows
  int time = 250;


void setup()
{

  for(int i = 0; i<16; i++)
  {
    pinMode(column[i], OUTPUT);  //setting rows to ouput
  }

  for(int i = 0; i<4; i++)
  {
    pinMode(layer[i], OUTPUT);  //setting layers to output
  }

  randomSeed(analogRead(10));  //seeding random for random pattern
}

void loop()
{
  turnEverythingOff();
  flickerOn();
  turnEverythingOn();
  delay(time);
  turnOnAndOffAllByLayerUpAndDownNotTimed();
  layerstompUpAndDown();
  spiralInAndOut();
  turnOnAndOffAllByColumnSideways();
  delay(time);
  aroundEdgeDown();
  turnEverythingOff();
  randomflicker();
  randomRain();
  diagonalRectangle();
  goThroughAllLedsOneAtATime();
  propeller();
  spiralInAndOut();
  flickerOff();
  turnEverythingOff();
  delay(2000);
}

//turn all off
void turnEverythingOff()
 {
   for(int i = 0; i<16; i++)
   {
     digitalWrite(column[i], 1);
   }
   for(int i = 0; i<4; i++)
   {
     digitalWrite(layer[i], 0);
   }
 }

//turn all on
void turnEverythingOn()
{
  for(int i = 0; i<16; i++)
```

```
//turn all on
void turnEverythingOn()
{
  for(int i = 0; i<16; i++)
  {
    digitalWrite(column[i], 0);
  }
  //turning on layers
  for(int i = 0; i<4; i++)
  {
    digitalWrite(layer[i], 1);
  }
}
//turn columns off
void turnColumnsOff()
{
  for(int i = 0; i<16; i++)
  {
    digitalWrite(column[i], 1);
  }
}
//flicker on
void flickerOn()
{
  int i = 150;
  while(i != 0)
  {
    turnEverythingOn();
    delay(i);
    turnEverythingOff();
    delay(i);
    i-= 5;
  }
}
//turn everything on and off by layer up and down NOT TIMED
void turnOnAndOffAllByLayerUpAndDownNotTimed()
{
  int x = 75;
  for(int i = 5; i != 0; i--)
  {
    turnEverythingOn();
    for(int i = 4; i!=0; i--)
    {
      digitalWrite(layer[i-1], 0);
      delay(x);
    }
    for(int i = 0; i<4; i++)
    {
      digitalWrite(layer[i], 1);
      delay(x);
    }
     for(int i = 0; i<4; i++)
    {
      digitalWrite(layer[i], 0);
      delay(x);
    }
    for(int i = 4; i!=0; i--)
    {
      digitalWrite(layer[i-1], 1);
      delay(x);
    }
  }
}
//turn everything on and off by column sideways
void turnOnAndOffAllByColumnSideways()
```

```
//turn everything on and off by colu
void turnOnAndOffAllByColumnSideways
{
  int x = 75;
  turnEverythingOff();
  //turn on layers
  for(int i = 0; i<4; i++)
  {
    digitalWrite(layer[i], 1);
  }
  for(int y = 0; y<3; y++)
  {
    //turn on 0-3
    for(int i = 0; i<4; i++)
    {
      digitalWrite(column[i], 0);
      delay(x);
    }
    //turn on 4-7
    for(int i = 4; i<8; i++)
    {
      digitalWrite(column[i], 0);
      delay(x);
    }
    //turn on 8-11
    for(int i = 8; i<12; i++)
    {
      digitalWrite(column[i], 0);
      delay(x);
    }
    //turn on 12-15
    for(int i = 12; i<16; i++)
    {
      digitalWrite(column[i], 0);
      delay(x);
    }
    //turn off 0-3
    for(int i = 0; i<4; i++)
    {
      digitalWrite(column[i], 1);
      delay(x);
    }
    //turn off 4-7
    for(int i = 4; i<8; i++)
    {
      digitalWrite(column[i], 1);
      delay(x);
    }
    //turn off 8-11
    for(int i = 8; i<12; i++)
    {
      digitalWrite(column[i], 1);
      delay(x);
    }
    //turn off 12-15
    for(int i = 12; i<16; i++)
    {
      digitalWrite(column[i], 1);
      delay(x);
    }
    //turn on 8-11
    for(int i = 8; i<12; i++)
    {
      digitalWrite(column[i], 0);
      delay(x);
    }
    //turn on 4-7
    for(int i = 4; i<8; i++)
    {
      digitalWrite(column[i], 0);
      delay(x);
    }
    //turn on 0-3
    for(int i = 0; i<4; i++)
    {
      digitalWrite(column[i], 0);
      delay(x);
    }
    //turn off 12-15
    for(int i = 12; i<16; i++)
    {
      digitalWrite(column[i], 1);
      delay(x);
    }
    //turn off 8-11
    for(int i = 8; i<12; i++)
    {
      digitalWrite(column[i], 1);
      delay(x);
    }
    //turn off 4-7
    for(int i = 4; i<8; i++)
    {
      digitalWrite(column[i], 1);
      delay(x);
    }
    //turn off 0-3
    for(int i = 0; i<4; i++)
    {
      digitalWrite(column[i], 1);
      delay(x);
    }
  }
}
//up and down single layer stomp
void layerstompUpAndDown()
{
  int x = 75;
  for(int i = 0; i<4; i++)
  {
    digitalWrite(layer[i], 0);
  }
  for(int y = 0; y<5; y++)
  {
    for(int count = 0; count<1; count++)
    {
      for(int i = 0; i<4; i++)
      {
        digitalWrite(layer[i], 1);
        delay(x);
        digitalWrite(layer[i], 0);
      }
      for(int i = 4; i !=0; i--)
      {
```

```
    for(int i = 4; i !=0; i--)
    {
      digitalWrite(layer[i-1], 1);
      delay(x);
      digitalWrite(layer[i-1], 0);
    }
  }
  for(int i = 0; i<4; i++)
  {
    digitalWrite(layer[i], 1);
    delay(x);
  }
  for(int i = 4; i!=0; i--)
  {
    digitalWrite(layer[i-1], 0);
    delay(x);
  }
 }
}
//flicker off
void flickerOff()
{
 turnEverythingOn();
 for(int i = 0; i!= 150; i+=5)
 {
  turnEverythingOff();
  delay(i+50);
  turnEverythingOn();
  delay(i);
 }
}
//around edge of the cube down
void aroundEdgeDown()
{
 for(int x = 200; x != 0; x -=50)
 {
  turnEverythingOff();
  for(int i = 4; i != 0; i--)
  {
   digitalWrite(layer[i-1], 1);
   digitalWrite(column[5], 0);
   digitalWrite(column[6], 0);
   digitalWrite(column[9], 0);
   digitalWrite(column[10], 0);

   digitalWrite(column[0], 0);
   delay(x);
   digitalWrite(column[0], 1);
   digitalWrite(column[4], 0);
   delay(x);
   digitalWrite(column[4], 1);
   digitalWrite(column[8], 0);
   delay(x);
   digitalWrite(column[8], 1);
   digitalWrite(column[12], 0);
   delay(x);
   digitalWrite(column[12], 1);
   digitalWrite(column[13], 0);
   delay(x);
   digitalWrite(column[13], 1);
   digitalWrite(column[15], 0);
   delay(x);
   digitalWrite(column[15], 1);
   digitalWrite(column[14], 0);
   delay(x);
```

```
   digitalWrite(column[15], 1);
   digitalWrite(column[14], 0);
   delay(x);
   digitalWrite(column[14], 1);
   digitalWrite(column[11], 0);
   delay(x);
   digitalWrite(column[11], 1);
   digitalWrite(column[7], 0);
   delay(x);
   digitalWrite(column[7], 1);
   digitalWrite(column[3], 0);
   delay(x);
   digitalWrite(column[3], 1);
   digitalWrite(column[2], 0);
   delay(x);
   digitalWrite(column[2], 1);
   digitalWrite(column[1], 0);
   delay(x);
   digitalWrite(column[1], 1);
  }
 }
}
//random flicker
void randomflicker()
{
 turnEverythingOff();
 int x = 10;
 for(int i = 0; i !=750; i+=2)
 {
 int randomLayer = random(0,4);
 int randomColumn = random(0,16);

 digitalWrite(layer[randomLayer], 1);
 digitalWrite(column[randomColumn], 0);
 delay(x);
 digitalWrite(layer[randomLayer], 0);
 digitalWrite(column[randomColumn], 1);
 delay(x);
 }
}
//random rain
void randomRain()
{
 turnEverythingOff();
 int x = 100;
 for(int i = 0; i!=60; i+=2)
 {
  int randomColumn = random(0,16);
  digitalWrite(column[randomColumn], 0);
  digitalWrite(layer[0], 1);
  delay(x+50);
  digitalWrite(layer[0], 0);
  digitalWrite(layer[1], 1);
  delay(x);
  digitalWrite(layer[1], 0);
  digitalWrite(layer[2], 1);
  delay(x);
  digitalWrite(layer[2], 0);
  digitalWrite(layer[3], 1);
  delay(x+50);
  digitalWrite(layer[3], 0);
  digitalWrite(column[randomColumn], 1);
 }
}
```

```
//diagonal rectangle
void diagonalRectangle()
{
  int x = 350;
  turnEverythingOff();
  for(int count = 0; count<5; count++)
  {
    //top left
    for(int i = 0; i<8; i++)
    {
      digitalWrite(column[i], 0);
    }
    digitalWrite(layer[3], 1);
    digitalWrite(layer[2], 1);
    delay(x);
    turnEverythingOff();
    //middle middle
    for(int i = 4; i<12; i++)
    {
      digitalWrite(column[i], 0);
    }
    digitalWrite(layer[1], 1);
    digitalWrite(layer[2], 1);
    delay(x);
    turnEverythingOff();
    //bottom right
    for(int i = 8; i<16; i++)
    {
      digitalWrite(column[i], 0);
    }
    digitalWrite(layer[0], 1);
    digitalWrite(layer[1], 1);
    delay(x);
    turnEverythingOff();
    //bottom middle
    for(int i = 4; i<12; i++)
    {
      digitalWrite(column[i], 0);
    }
    digitalWrite(layer[0], 1);
    digitalWrite(layer[1], 1);
    delay(x);
    turnEverythingOff();
    //bottom left
    for(int i = 0; i<8; i++)
    {
      digitalWrite(column[i], 0);
    }
    digitalWrite(layer[0], 1);
    digitalWrite(layer[1], 1);
    delay(x);
    turnEverythingOff();
    //middle middle
    for(int i = 4; i<12; i++)
    {
      digitalWrite(column[i], 0);
    }
    digitalWrite(layer[1], 1);
    digitalWrite(layer[2], 1);
    delay(x);
    turnEverythingOff();
    //top right
    for(int i = 8; i<16; i++)
    {
      digitalWrite(column[i], 0);
```

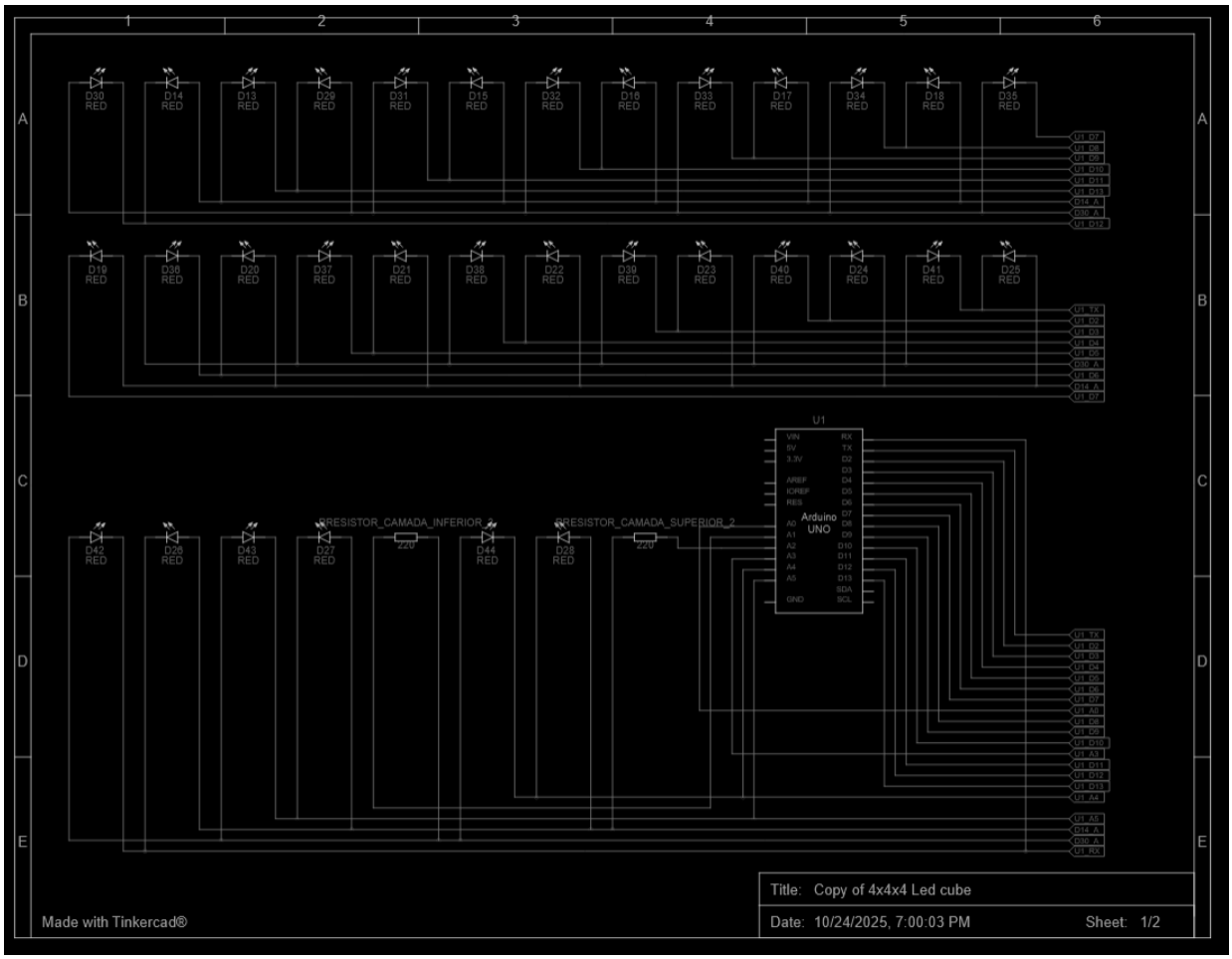
```
    }
    digitalWrite(layer[2], 1);
    digitalWrite(layer[3], 1);
    delay(x);
    turnEverythingOff();
    //top middle
    for(int i = 4; i<12; i++)
    {
      digitalWrite(column[i], 0);
    }
    digitalWrite(layer[2], 1);
    digitalWrite(layer[3], 1);
    delay(x);
    turnEverythingOff();
  }
  //top left
  for(int i = 0; i<8; i++)
  {
    digitalWrite(column[i], 0);
  }
  digitalWrite(layer[3], 1);
  digitalWrite(layer[2], 1);
  delay(x);
  turnEverythingOff();
}
//propeller
void propeller()
{
  turnEverythingOff();
  int x = 90;
  for(int y = 4; y>0; y--)
  {
    for(int i = 0; i<6; i++)
    {
      //turn on layer
      digitalWrite(layer[y-1], 1);
      //a1
      turnColumnsOff();
      digitalWrite(column[0], 0);
      digitalWrite(column[5], 0);
      digitalWrite(column[10], 0);
      digitalWrite(column[15], 0);
      delay(x);
      //b1
      turnColumnsOff();
      digitalWrite(column[4], 0);
      digitalWrite(column[5], 0);
      digitalWrite(column[10], 0);
      digitalWrite(column[11], 0);
      delay(x);
      //c1
      turnColumnsOff();
      digitalWrite(column[6], 0);
      digitalWrite(column[7], 0);
      digitalWrite(column[8], 0);
      digitalWrite(column[9], 0);
      delay(x);
      //d1
      turnColumnsOff();
      digitalWrite(column[3], 0);
      digitalWrite(column[6], 0);
      digitalWrite(column[9], 0);
      digitalWrite(column[12], 0);
      delay(x);
      //d2
```

```
      //d2
      turnColumnsOff();
      digitalWrite(column[2], 0);
      digitalWrite(column[6], 0);
      digitalWrite(column[9], 0);
      digitalWrite(column[13], 0);
      delay(x);
      //d3
      turnColumnsOff();
      digitalWrite(column[1], 0);
      digitalWrite(column[5], 0);
      digitalWrite(column[10], 0);
      digitalWrite(column[14], 0);
      delay(x);
    }
  }
  //d4
  turnColumnsOff();
  digitalWrite(column[0], 0);
  digitalWrite(column[5], 0);
  digitalWrite(column[10], 0);
  digitalWrite(column[15], 0);
  delay(x);
}
//spiral in and out
void spiralInAndOut()
{
  turnEverythingOn();
  int x = 60;
  for(int i = 0; i<6; i++)
  {
    //spiral in clockwise
    digitalWrite(column[0], 1);
    delay(x);
    digitalWrite(column[1], 1);
    delay(x);
    digitalWrite(column[2], 1);
    delay(x);
    digitalWrite(column[3], 1);
    delay(x);
    digitalWrite(column[7], 1);
    delay(x);
    digitalWrite(column[11], 1);
    delay(x);
    digitalWrite(column[15], 1);
    delay(x);
    digitalWrite(column[14], 1);
    delay(x);
    digitalWrite(column[13], 1);
    delay(x);
    digitalWrite(column[12], 1);
    delay(x);
    digitalWrite(column[8], 1);
    delay(x);
    digitalWrite(column[4], 1);
    delay(x);
    digitalWrite(column[5], 1);
    delay(x);
    digitalWrite(column[6], 1);
    delay(x);
    digitalWrite(column[10], 1);
    delay(x);
    digitalWrite(column[9], 1);
    delay(x);
```

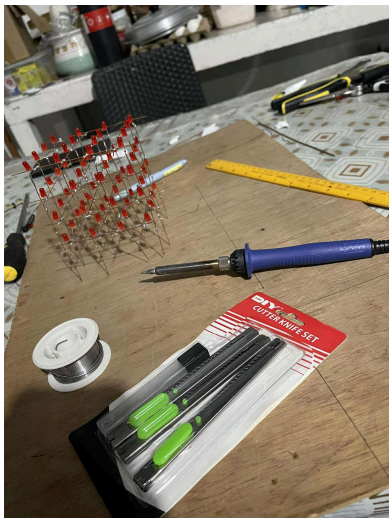
```
//spiral out counter clockwise
    digitalWrite(column[9], 0);
    delay(x);
    digitalWrite(column[10], 0);
    delay(x);
    digitalWrite(column[6], 0);
    delay(x);
    digitalWrite(column[5], 0);
    delay(x);
    digitalWrite(column[4], 0);
    delay(x);
    digitalWrite(column[8], 0);
    delay(x);
    digitalWrite(column[12], 0);
    delay(x);
    digitalWrite(column[13], 0);
    delay(x);
    digitalWrite(column[14], 0);
    delay(x);
    digitalWrite(column[15], 0);
    delay(x);
    digitalWrite(column[11], 0);
    delay(x);
    digitalWrite(column[7], 0);
    delay(x);
    digitalWrite(column[3], 0);
    delay(x);
    digitalWrite(column[2], 0);
    delay(x);
    digitalWrite(column[1], 0);
    delay(x);
    digitalWrite(column[0], 0);
    delay(x);
//spiral in counter clock wise
    digitalWrite(column[0], 1);
    delay(x);
    digitalWrite(column[4], 1);
    delay(x);
    digitalWrite(column[8], 1);
    delay(x);
    digitalWrite(column[12], 1);
    delay(x);
    digitalWrite(column[13], 1);
    delay(x);
    digitalWrite(column[14], 1);
    delay(x);
    digitalWrite(column[15], 1);
    delay(x);
    digitalWrite(column[11], 1);
    delay(x);
    digitalWrite(column[7], 1);
    delay(x);
    digitalWrite(column[3], 1);
    delay(x);
    digitalWrite(column[2], 1);
    delay(x);
    digitalWrite(column[1], 1);
    delay(x);
    digitalWrite(column[5], 1);
    delay(x);
    digitalWrite(column[9], 1);
    delay(x);
    digitalWrite(column[10], 1);
    delay(x);
    digitalWrite(column[6], 1);
```

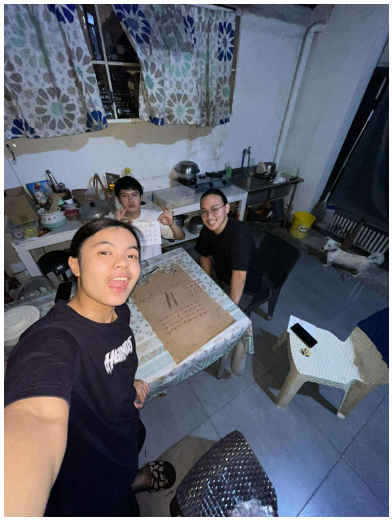
```
    delay(x);
//spiral out clock wise
    digitalWrite(column[6], 0);
    delay(x);
    digitalWrite(column[10], 0);
    delay(x);
    digitalWrite(column[9], 0);
    delay(x);
    digitalWrite(column[5], 0);
    delay(x);
    digitalWrite(column[1], 0);
    delay(x);
    digitalWrite(column[2], 0);
    delay(x);
    digitalWrite(column[3], 0);
    delay(x);
    digitalWrite(column[7], 0);
    delay(x);
    digitalWrite(column[11], 0);
    delay(x);
    digitalWrite(column[15], 0);
    delay(x);
    digitalWrite(column[14], 0);
    delay(x);
    digitalWrite(column[13], 0);
    delay(x);
    digitalWrite(column[12], 0);
    delay(x);
    digitalWrite(column[8], 0);
    delay(x);
    digitalWrite(column[4], 0);
    delay(x);
    digitalWrite(column[0], 0);
    delay(x);
  }
}
//go through all leds one at a time
void goThroughAllLedsOneAtATime()
{
  int x = 15;
  turnEverythingOff();
  for(int y = 0; y<5; y++)
  {
    //0-3
    for(int count = 4; count != 0; count--)
    {
      digitalWrite(layer[count-1], 1);
      for(int i = 0; i<4; i++)
      {
        digitalWrite(column[i], 0);
        delay(x);
        digitalWrite(column[i], 1);
        delay(x);
      }
    digitalWrite(layer[count-1], 0);
    }
    //4-7
    for(int count = 0; count < 4; count++)
    {
      digitalWrite(layer[count], 1);
      for(int i = 4; i<8; i++)
      {
        digitalWrite(column[i], 0);
        delay(x);
        digitalWrite(column[i], 1);
        delay(x);
      }
    digitalWrite(layer[count], 0);
    }
    //8-11
    for(int count = 4; count != 0; count--)
    {
      digitalWrite(layer[count-1], 1);
      for(int i = 8; i<12; i++)
      {
        digitalWrite(column[i], 0);
        delay(x);
        digitalWrite(column[i], 1);
        delay(x);
      }
    digitalWrite(layer[count-1], 0);
    }
    //12-15
    for(int count = 0; count < 4; count++)
    {
      digitalWrite(layer[count], 1);
      for(int i = 12; i<16; i++)
      {
        digitalWrite(column[i], 0);
        delay(x);
        digitalWrite(column[i], 1);
        delay(x);
      }
    digitalWrite(layer[count], 0);
    }
  }
}
```

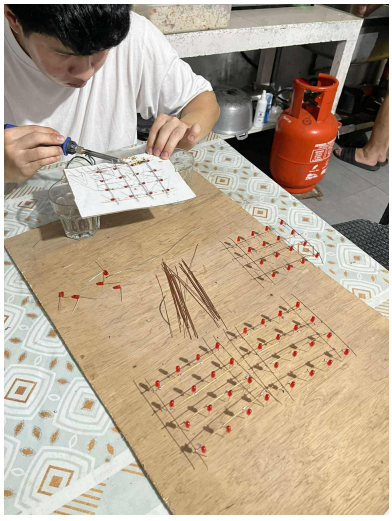
Appendix B - Circuit Diagram

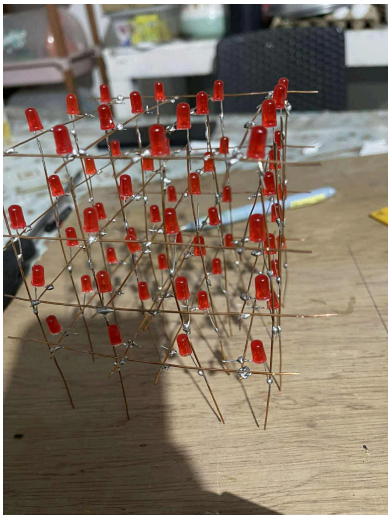


Appendix C - Prototype and Making Process Images

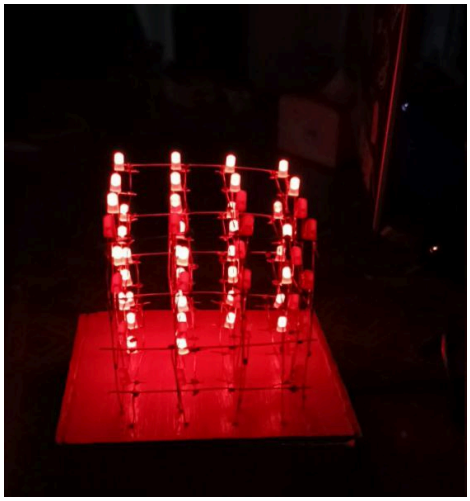


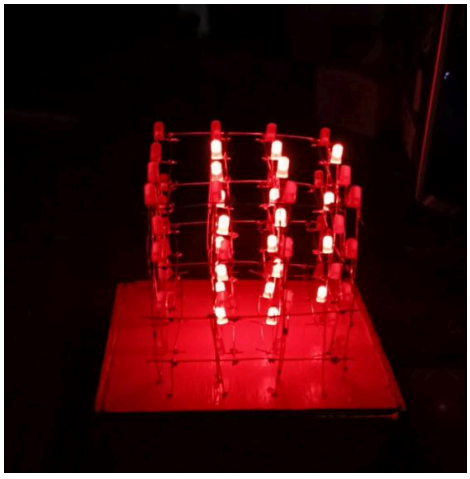
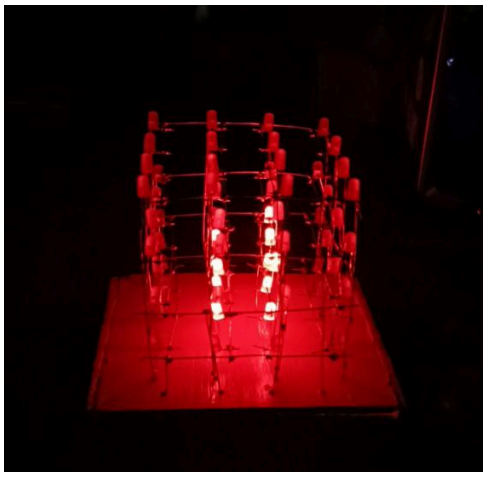
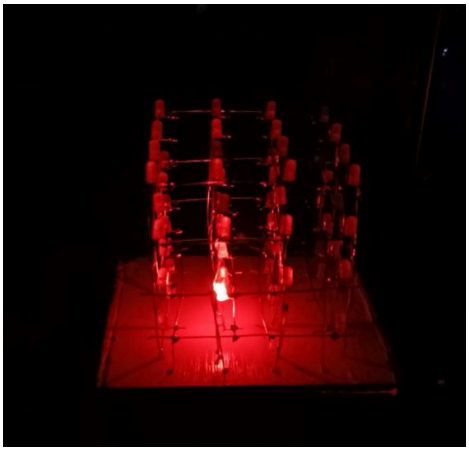






Appendix D - Sample Output Sequences




5.3 Manual Operations

This section presents the operational guidelines for the Arduino-based 4×4×4 LED Cube using multiplexing technique. It provides instructions for powering, operating, and troubleshooting the system to ensure proper functionality and demonstration of sequential lighting patterns.

A. System Overview

The 4×4×4 LED Cube consists of 64 light-emitting diodes (LEDs) arranged in four layers of sixteen LEDs each. It is powered and controlled by an Arduino Uno microcontroller programmed to generate various lighting patterns using multiplexing. The cube operates through rapid switching of layers and columns, creating the illusion of continuous illumination and three-dimensional animation.

B. Powering the System

1. Connect the Arduino Uno board to a 5V USB power source (such as a laptop, desktop, or power bank).
2. Ensure that all connections from the cube to the Arduino pins are secure and correctly placed according to the wiring diagram.
3. Once power is supplied, the system will automatically initialize and begin displaying programmed LED patterns.

C. Operating Procedure

1. After powering the circuit, the cube will start executing its default lighting sequence.
2. The lighting effects are automatically cycled based on the code uploaded to the Arduino Uno.

3. Users may modify the pattern timing or sequence by adjusting the delay values or logic arrays within the Arduino program.

4. If a reset button is connected, pressing it will restart the sequence from the beginning.

D. Maintenance and Safety Precautions

1. Avoid overloading the Arduino by connecting additional components without proper current limitation.

2. Do not connect the cube to a voltage higher than 5V to prevent LED burnout or Arduino damage.

3. Ensure that all wires are properly soldered and insulated to avoid short circuits.

4. When testing or modifying the cube, always disconnect the power source first.

E. Troubleshooting Guide

Problem	Possible Cause	Solution
Some LEDs are not lighting up	Loose wire or cold solder joint	Check solder connections and reflow if needed
Entire layer not working	Incorrect layer pin connection	Verify wiring between layer pins and Arduino
Patterns flicker or appear dim	Power fluctuation or timing delay too long	Ensure stable 5V supply and adjust delay() in code
Arduino not responding	Faulty USB cable or code error	Replace cable or re-upload the sketch

F. User Modifications

- Users can expand or modify the cube's behavior by editing the Arduino code. Possible enhancements include:
- Adding more lighting patterns or transitions.
- Adjusting brightness using pulse-width modulation (PWM).
- Expanding the cube size (e.g., 8×8×8) with additional drivers or transistors.

G. Conclusion

The LED Cube is a hands-on learning tool that demonstrates how multiplexing enables efficient control of multiple LEDs using limited microcontroller pins. Proper operation, maintenance, and adherence to safety guidelines will ensure the system remains functional and visually effective during demonstrations and educational use.