IT-BUSINESS COLLEGE

DEPARTMENT: COMPUTER SCIENCE

PROJECT TITLE: LUMEN CONTROL

GROUP NAME: LAN'S GROUP

Team Members:

Askarbekov Bayel - 238715065
Esenbekova Nurai - 238715072
Anarbekov Adilkan - 238715070
Bekzhanov Emir - 238711028
Instructor: Nurbekov Mirlan
Group: SCA-23B

Remote lighting control system Lumen Control

The journey from design to smart lighting model in action.

The Lumen Control project team - Ala-Too International University.
Bishkek, Chuy Region, Kyrgyzstan.
Date: 11 December 2025.

Table of Contents

Introduction

Modern campuses and office buildings boast hundreds or thousands of light fixtures. Most of these types of fixtures are controlled by manual switches on the wall in corridors, classrooms, and offices. Since sometimes people forget to turn the lights off or leave an area unattended for some time and in fact lighting provides a significantly high load on the entire power system. According to industry studies, office lighting control systems can reduce energy consumption by up to 30 per cent, and a company that modernizes its buildings has seen savings in their energy consumption of around 70 percent. Smart lighting not only lowers work costs but also promotes safety and productivity with its provision of light at the perfect spot when it is required.

Our research, Lumen Control, intends to investigate the use of microcontroller-based automation in university lighting. We have also adapted an Arduino system using the common design tools available, but also developed custom software to enable our software to be scalable and a versatile solution with remote control over multiple light sources. We began with basic circuits with a couple of LEDs,and slowly

developed more complicated control methods including IR remotes and Bluetooth communication. Each prototype taught us a new lesson, about circuitry, programming, user experience, and scalability. Lastly, we were able to make a micro-model of three zones in the building, all powered by phone app and IR remote.

This report draws on our experience from the initial breadboard tests to the almost complete system. You can read more over why we chose these hardware and software solutions; and the reasoning behind why each prototype was required and then explain lessons we learned from this experience. Illustrations and diagrams (figures 1.1-4.1) augment the narrative and help the reader visualize the designs. A full appendix with example code of Arduino and Flutter is given, followed by assessment of the system, evaluation of the challenges and future work.

Motivation and Problem Statement

When designing any engineered approach, the problem must be recognized when designing the engineering project. The two related issues behind our initiative are: (1) energy demand and (2) operational convenience. According to research, lighting accounts for 30% of overall in a building's electricity use, and automation-controlled systems reduce this number significantly. Conventional lighting systems consume electricity by turning on vacant rooms or failing to respond to natural light. Aside from the energy waste, manually switching dozens of lights is inefficient for staff and out of reach for large facilities. Therefore, the following problem statement was developed.

1.Energy efficiency: Lights need to be enabled only when rooms are occupied or when there is a lack of natural light. Savings on excess usage lower energy bills and reduces wear and tear on lamps.

2.Convenience and safety: Security personnel and facility managers must keep lights on without going through every door. And as we know, remote control allows emergency response quickly without adding any foot traffic.

3.Scalability: The system will allow for a variety of lighting zones without any modifications. Replicating modules rather than reinventing the architecture is the only way to scale the network from a few LEDs to dozens of lamps.

4.Affordability: Parts and software must be cheap and easy to get. This is reproducible by students and hobbyists on small budgets and uses the Arduino platform and open-source libraries.

In order to meet them, we decided to gradually progress in building the system. Each prototype described one facet of the problem: basic controls, memory and state, wireless communication, physical model, and user interface. By building complexity we could test a feature at a time and test for ease before adding new features, it was ensured to be reliable.

Prototype 1: Manual LED Control

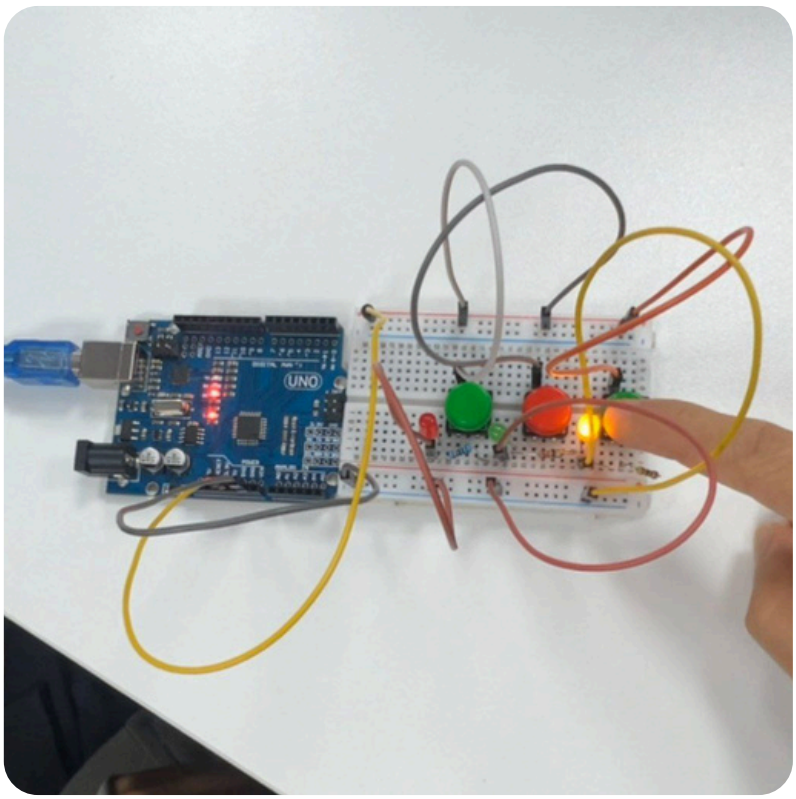

Figure 1.1 - prototype with hree LEDs operated with three push buttons

We began with the simplest circuit we could find: three LEDs operated with three push buttons. When the button pressed down the LED lit up and when the button was released the LED turned off. A prototype was formulated in order to validate elementary electrical connections, verify that the Arduino is able to read both digital inputs and output signals, and to teach others the basics of the breadboard wiring.

Circuit Design

In Prototype 1 all the LEDs were connected in series with a 220 Ω resistor to minimize current and damages associated with this logic. LEDs anodes were connected in series to Arduino digital pins 9-11, and cathodes to the ground rail. Every push button was wired to 5 V when pushed and a ground in a pull-down resistor when released. Wiring Diagram developed in Tinkercad is shown in figure 1.1 The jumper wires will be of different colors that represent different signals; orange wires will deliver the button signals, and yellow, blue, and cyan wires will serve LEDs. The diagram illustrates good practices for a breadboard: lining everything up in one row and using the positive and negative rails on the rail uniformly.

Physical Assembly

We constructed the circuit on a solderless breadboard (See Figure 1.2), where we wired the wires to it and connected them as well. An associated photo illustrates the arrangement and their controlled arrangement. The design was built to prevent tangling and inadvertent disconnections. We began with simple and simple design which helped us identify and debug physical problem such as loose connections, improperly seated or bad jumper wires. This step was critical because a minor hardware problem leads to erratic behavior or sporadic failures that are difficult to track at the high end of the scale. Once the circuit assembled and was powered with USB, we observed that the LEDs were responding accurately and consistently to the button presses, verifying our programme-based digital input and output. This practical testing was not only the first time we were actually seeing the design of the circuit, but it helped us improve upon the fundamentals of electronics such as voltage, current flow and signal propagation along the breadboard.

Lessons Learned
The first prototype taught us many important lessons which would lead us as we proceeded with development. It highlighted the importance of the necessity of current-limiting resistors in the protection of LEDs and to protect the components from excessive current and to enable a safe and predictable operating environment for the Arduino. An Arduino function was used from here and through experimental work, we obtained hands-on experience. pinMode and digitalRead, with also digitalWrite as the basis for setting up digital inputs and outputs. While remote control capabilities were not implemented in this first circuit, they did not undermine the foundation from which it originated.

Finally, when the model was tested by other people in future laboratory experiments, we learned first hand the basic principles of the system. The accompanying photo made it so easy to trace connections, detect any errors, etc. reliably replicate the setup. This not only got us to be faster when debugging, it validated that the design could be reproduced or expanded without any confusion. Finally, selection using the momentary switches as input devices we created an opportunity in the next step to explore stateful logic. Reiteration of the project, preparing for advanced control features, and showing how early design decisions can then shape the following phases.

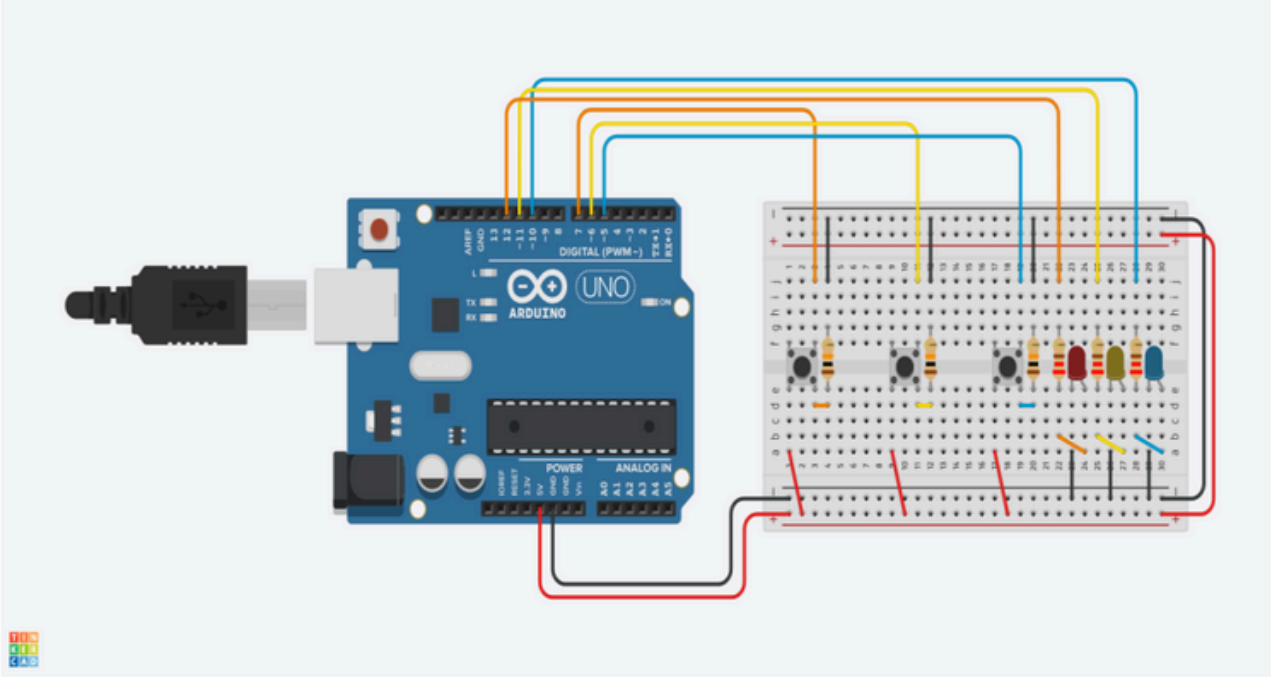

Figure 1.11 - Wiring diagram of Prototype 1 with three buttons and LEDs

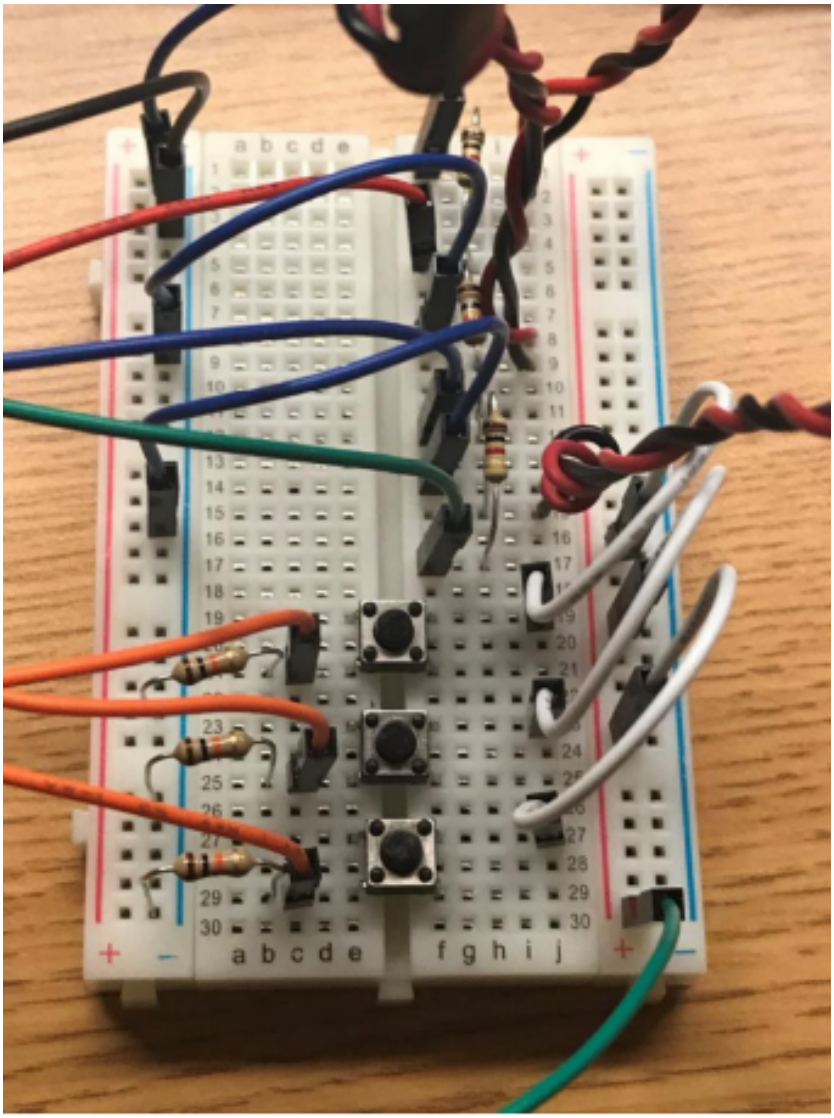

Figure 1.2 - Photograph of the first prototype assembled on a breadboard

Prototype 2: On/Off Button Logic

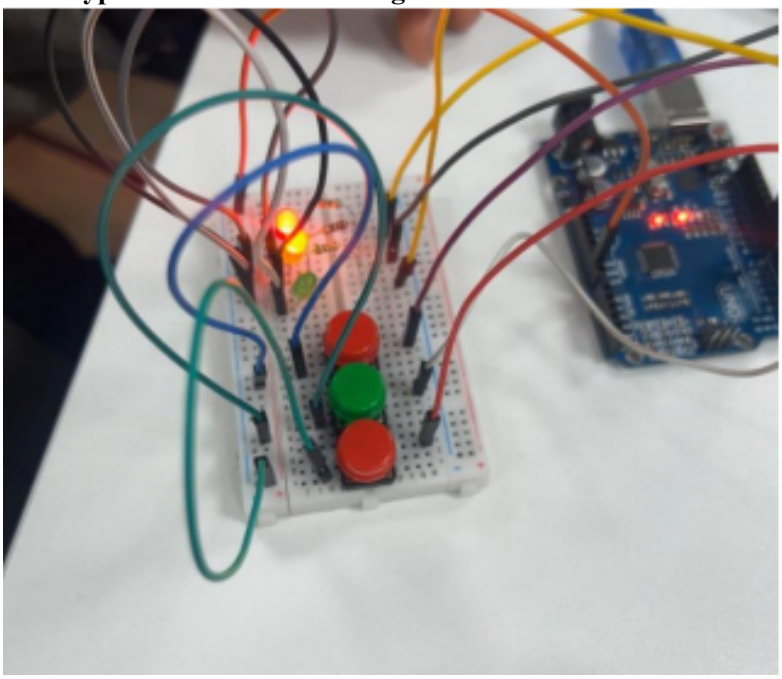

Figure 1.2.2 - Photograph of the prototype with On/Off logic

We confirmed the hardware works properly and worked on enhancing the user experience using the system. The LEDs in the first prototype lit only while the buttons were pressing it, which impeded the realism and feasibility of setup. That is for real lighting systems. A light must be allowed to turn on with just one command and then remain on until a separate command powers it off. To achieve this, we changed the software such that each button could switch it on and off. Assisted the end-user to switch on or off the LED, which provided a more intuitive and accurate control procedure.

While this adjustment may appear trivial, it provided various key considerations. Has to track the state of each LED in software, making sure that the system remembered which lights were on or off at any given time. Furthermore, we had to write debounce logic so as to avoid any future problems such as these. Multiple unintended toggles can arise from a single button press, caused by the mechanical bounce of switch contacts. This experience emphasised how much hardware behavior and software are closely tied together. Design: minor changes in user interaction may at times require thoughtful considerations of the program's logic, timing, and reliability. In the end, this enhancement did not only render the prototype more workable and easy-to-use but also enhanced our comprehension of state management and input in interactive electronic systems.

Software Design
Prototype 2 showed that every LED state (on/off) in the Arduino sketch was stored as Boolean array. In a button press the code flipped out the state of the button and read the input output using digitalWrite. Fired up this behaviour reliably meant filtering out noise created by mechanical switch contacts. For us to make this behaviour consistently worked, we had to reduce noise from mechanical switch contacts. Mechanical buttons do not flip between HIGH and LOW switches perfectly. They bounce for several milliseconds. One press without debouncing looks multiple rapid presses, making the LED flicker. To help us out it, we tested the button state at intervals of time and disregarded all transitions that took place within 50 ms of having a successful press.

Why This Prototype?
With the aim of practicality when it comes to lighting, toggle logic was the impetus. System lights are to stay on without any constant input with the user, and switch off every second command. We avoided errors later when inputs come in due to early debouncing implementation and testing from an infrared receiver or Bluetooth module. In addition, Prototype 2 introduced the concept of state machines; each LED’s behaviour depended on what had previously been brought by it and what came from the input now. This prototype showed how software enhancements could dramatically improve functionality without changing the physical circuit.

Reflections
It was during this round that we realised that structuring our code makes a difference and keeping the variables in the loop to keep the state throughout. They have to arrange their program and so, as described by: Tackling program issues. Using information about each LED status, we were able to make reliable behavior based toggling that enabled an adaptable toggling system to be used that ensured the foundation for continued operation, providing the required and predictable operation. As well as debouncing not implementing correct debouncing techniques were not to remove unintended multiple triggers from one, but this only improved the speed and user experience. It not only provided a seamless button press, but also set the groundwork for future wireless commands. This ensured that subsequent inputs, from an IR remote or Bluetooth module, could be understood accurately without interference or erratic behavior.

Beyond this prototype achieving successful implementation, however, we received more than functional validation; it was confidence to overcome the problems of physical buttons. Having enforced strong state and input handling, we were now ready to look into remote control features, broadening the system's scope and usability. This variant showed that while the original concept was to maintain the simplicity of tasks by having simple programming principles, attention to software development is, and always will be, necessary potentially directly improve the stability and scalability of the project, making it work even better down the line and user-friendly interactions in future versions.

Prototype 3: IR Remote Control

To remove the need for physical contact, and to make an even larger number of lights work with it, we integrated an infrared (IR) receiver. One standard IR remote was used to send commands up from inside the system and an IR receiver to control it. IR remotes are used in consumer electronics and capable of transmitting dozens of distinctive codes, all of which are able to be recognized and interpreted by a microcontroller. After implementing this feature, we got to experiment with wireless communication to learn how signals can be correctly received, decoded, and, for that matter, correspond to precise behaviors of the Arduino program.

The prototype gave us not only a better and more flexible user interface but also lights were as real as possible controlled without forcing buttons, and also drastically increasing the number of outputs that could be controlled. Every single unique IR code could be assigned to a different LED, or a system of LEDs, practically doubling down on the system's potential without needing more hardware resources. Furthermore, engagement with the IR receiver brought practical implications such as signal interference as well as line-of-sight requirements, and code filtering, giving us crucial insights into some of the difficulties in wireless control on the ground real-world environments. Taken together this version represented an important achievement to construct a more advanced lighting control system.

Hardware Integration

We deployed an infrared receiver module (e.g., VS1838B) to the breadboard. The receiver's OUT pin was attached to an Arduino input (digital pin 2), and the VCC and GND pins linked to the appropriate rails. The individual resistors connected to pins 3-8 were six LEDs. The updated wiring diagram was shown in Figure 2.1. The right-hand side of the diagram contains an IR remote with labeled buttons; each button emits a distinct hexadecimal code. We decoded these signals via the IRremote Arduino library. Pressing a button presented a 32-bit code to the microcontroller, which we matched against a lookup table to figure out which LED to toggle.

Decoding IR Signals

IR commands, for example, taught us about NEC and Sony, and we had to learn about their processes. Each remote button sends a series of pulses at a carrier frequency of 38 kHz, modulated to represent zeros and ones. The IRremote library simplifies this procedure by handling the timing details. In our code we called irrecv.decode() to read a code into a decode_results structure, extract the value field and compared it against known codes. For instance, pressing the red button on our remote returned a code that toggled LED 3. As in Prototype 2, we also stored the state of each LED and debounced commands by neglecting repeated transmissions (most remotes will send a repetitive code when a button is held).

Prototyping Challenges

For example, using an IR remote introduced new challenges. The receiver needed an unobstructed line of sight; the signal was blocked by putting our hand in front of the sensor. Ambient infrared noise from fluorescent lights sometimes triggered false alarms. We used two approaches to alleviate this issue: shielding the receiver and ignoring codes that did not match any expected value. Managing multiple LED states was another problem. Using six outputs instead of three made our code more complicated, a fact to note of modular design. Yet the prototype worked: we could switch on any of six LEDs from across the room without touching the breadboard.

Visualising the system

Figure 2.2 demonstrates a photograph of the IR receiver, LEDs on our breadboard, and remote control. The receiver's distinctive black package with metallic cross helps focus the 38 kHz carrier signal. In Figure 2.3, the infrared emitter on the remote control glows faintly when viewed through a camera, illustrating how IR communication functions (the light is invisible to our eyes but visible to digital sensors). By visualising these signals, we were able to have the confidence of knowing the system was operating correctly and have a real-world showcase for our students.

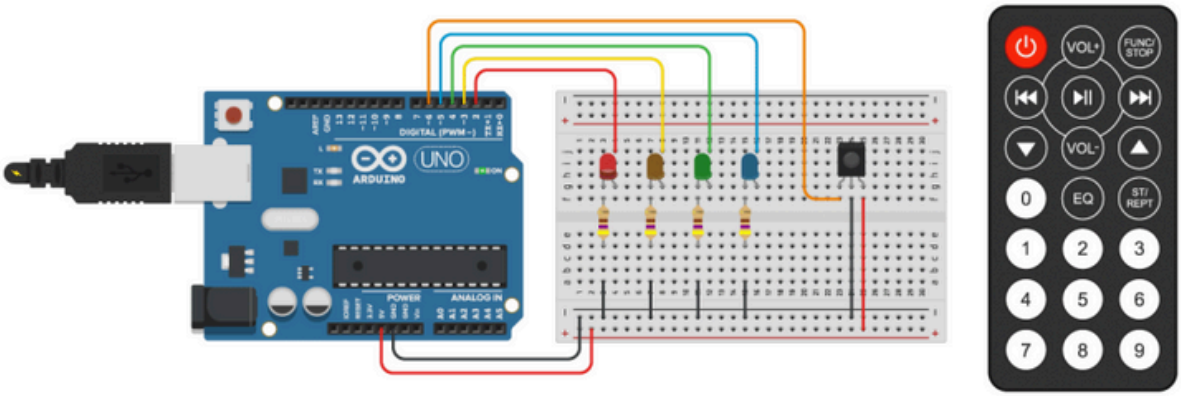

Figure 2.1 Curcuit diagram showing the Arduino , six LEDs, an IR receiver and remote control

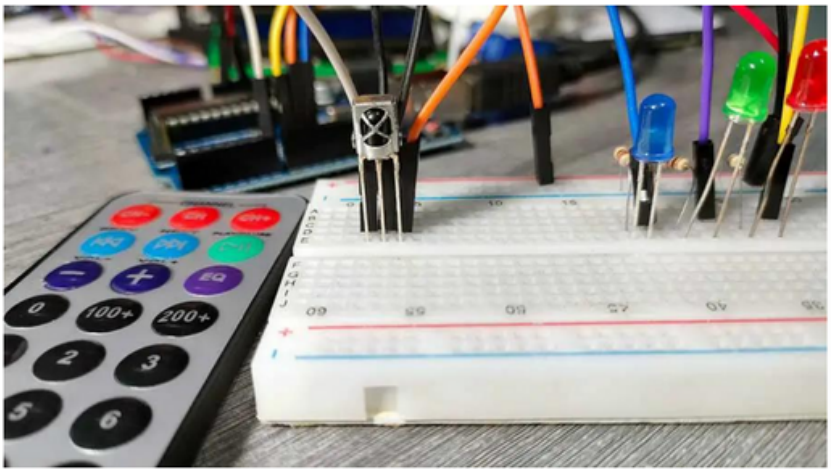

Figure 2.2 Photograph of the IR receiver and LEDs with the remote control

Figure 2.3 Camera captured image of an IR remote emitting infrared light

First lighting block prototype design
You should understand that controlling individual LEDs on a breadboard is a great way to teach you digital fundamentals, electronics and microcontroller programming, but buildings in the real world are seldom single-light operated in isolation. In most architectural scenarios, the lights are assembled in the same building room, hallway or larger zone, and control systems must process these clusters efficiently. We also designed a small lighting block containing six LEDs laid out in a straight line. Once the lighting and equipment was all set up, it was possible to change the angle of two LED lines in a straight line as you would in architectural thinking. It was used to show only one room, or one corridor in what we imagined, to allow us to guess how several lights could collectively be controlled in a small space. To maintain a connected user interface we maintained the IR control paradigm introduced in Prototype 3, and using the already existing experience of wireless communications. Now pressing a number key on the remote toggled the corresponding LED in the block, an intuitive map between input and output. This setup was useful for playing with the grouped lighting control without changing the configuration, manageable on the breadboard. We learned the big picture when we used clusters instead of individual LEDs: scalability, usability/scalability, user interaction and spatial representation, all of which are key factors for building lighting systems for live buildings in real spaces.

Model Construction
We attached six LEDs to a piece of cardboard, with every LED grouped within a simple foamboard frame, simulating the walls of a room or corridor. There was an LED assigned to zones within the block, and we can represent areas of light in a compact, visual way. Wires from the LEDs were fed back into the breadboard and the necessary resistors and IR receiver were connected thereafter, following the layout in the previous prototypes. This setup allowed for a clear, manageable way to test the system while keeping room to either change or grow the setup as necessary. The housing not only allowed us to imagine where lights might be placed and clustered in a realistic scene, it also facilitated the demonstration of the remote control system to others. When buttons on the remote were pressed, the corresponding LEDs in the block would be lit directly, giving a near perfect simulating effect of real lighting behavior. Those observers could respond well to this interactive demonstration, which they did see the lights act dynamically and respond without having anyone reach in and physically interact with the model. The setup thus was a visual representation of both a performance review of the control system and a way to visually test our control system as well as to play with the model! Theory and a practical, comprehensible outcome.

Reason for This Prototype
That was a crucial bridge for the project between a purely electronic experiment and a solid physical model of an architecture. We could evaluate by placing the LEDs inside a small “room” to the practical side of things like aesthetics, beyond the workings of the system itself: the type of light, the intensity necessary for effective illumination, and the appropriateness of the IR control scheme in a spatial sense. The block symbolizing a room influenced our thinking on that process. We thought about the cabling and wire handling, trivial, simple problems in small breadboard setups that you’d be tempted to consider insignificant, and the significance of larger models/real buildings in scale of the system. Moreover, the development of this demonstration block also made communicating the project to a nontechnical audience easier. Facility managers, building planners, or stakeholders who may think about using the system could directly observe how lights would behave in real-world spaces, rather than attempting to understand abstract circuits or codes. It represented the tangible, visual. This stage has been a bridge between technical development and showing a more practical view on how the lighting system works in real-world, practical implementation, to ensure that the user was well aware of both functionality and experience.

Lessons Learned

From this prototype testing we gleaned a few practical lessons in the deployment in the field. First, we noted that enclosures can make themselves hot significantly if multiple LEDs are in operation at the same time, emphasizing the value of good spacing and ventilation in any real installation. Proper thermal management is important not only for the efficient operation of the LEDs, but also to keep surrounding components and the system's safety. Second, we indicated that the IR receiver's line of sight dependence could become inconvenient to apply with more complex or obstructed layouts. IR works great for basic configurations, but objects blocking the way between the remote and the receiver might inhibit signals from being collected, leading to disuse and annoyance for users in larger or more complicated installations. These observations, taken all together, encouraged us to focus on Bluetooth control in the next phase of development. Bluetooth provides the benefits of wireless communication without the rigid line-of-sight obligations, allowing you to more easily position both the controller and the lights. Knowledge of these constraints at the IR stage of this phase helped us to design a more efficient and usable control system, as the project progressed, making a more robust and easy control system as an asset to plan in the years after.

Mobile App Development - Phase 1

The IR remote gave one easier wireless control yet has drawbacks as well. In particular, it necessitated a line of sight to the receiver, which could be limiting in more complex layouts or larger models. Not only does a standard IR remote have few buttons, but this limits the number of lights or zones which could be controlled independently. Recognising these limitations and considering modern user expectations, we decided to develop a mobile application that could offer greater flexibility and convenience. To achieve this, we chose Flutter, Google's cross-platform UI framework, which allowed us to design and build a single application capable of running on both Android and iOS devices. Flutter's architecture made it much easier to design its interactive, responsive interface and maintain compatibility across various platforms. No longer must separate apps be developed for different operating systems. The application communicates with the Arduino via a Bluetooth HC-05 module, which allows wireless control of the lighting system without the limitations imposed by IR line-of-sight. This transition marked the start of a significant step forward in making the system more user-friendly, scalable, and aligned with modern expectations for smart device control.

Connecting the HC-05 module

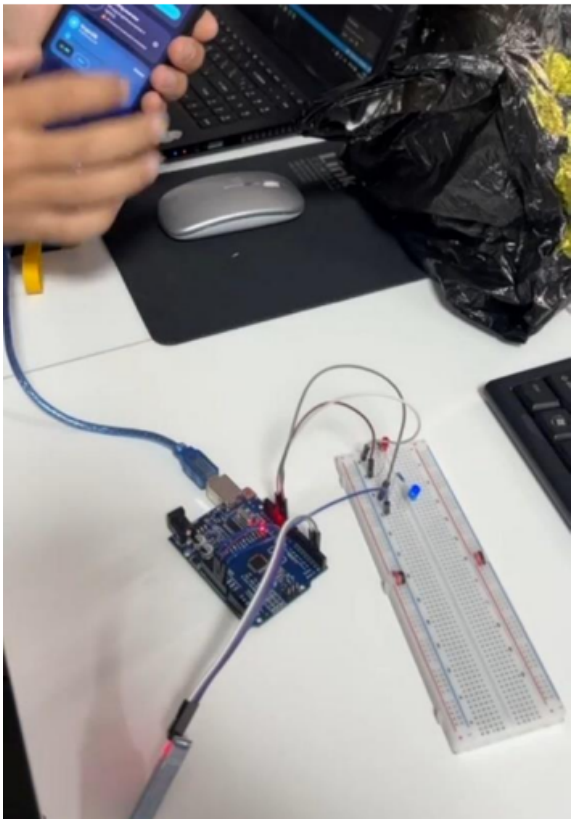

Figure 3 -Connecting the HC-05 module

The HC-05 module is a popular Bluetooth Serial Port Profile (SPP) device that enables the Arduino to both send and receive data from a smartphone or other Bluetooth-enabled device. Its popularity stems from its simplicity, low cost, and reliable performance in hobbyist and prototyping projects, making it an industry standard tool for beginners and prototypes. For our system, we wired the module into the Arduino Uno as depicted in Figure 3.1, carefully following established wiring conventions to ensure proper operation.

The module's VCC pin was connected to the Arduino's 5V output, providing power, and the GND was connected to the Arduino ground to complete the circuit. The TXD (transmit) pin of the HC-05 was attached to the Arduino's RX pin and the RXD (receive) pin of the module was connected to the Arduino's TX pin. With this setup, the communication is bidirectional, and the Arduino can both receive commands from the smartphone and provide status or feedback data back to the app.

Since the HC-05 operates with 3.3 V logic on its RXD pin, some tutorials and manufacturer documentation recommend adding a voltage divider or logic level shifter to prevent damage from the Arduino's 5 V TX signals. This advice ensures long-term reliability of the module and prevents electrical overstress. Consistent with this advice, the project documentation we consulted emphasized connecting GND to Arduino ground, the red VCC pin to 5 V, TX to Arduino RX, and RX to Arduino TX, thereby upholding the standard wiring approach that enables safe and effective communication with the Bluetooth module.

Basic Flutter App

Our main Flutter application was built to provide a simple yet effective way to control the lighting system through Bluetooth. The app first displayed a list of available Bluetooth devices, allowing the user to select and connect to the HC-05 module associated with the Arduino. Once pairing was complete, the interface presented a panel of buttons labelled Zone 1 through Zone 6, each corresponding to a specific LED or group of LEDs in the lighting block. Pressing one of these buttons sent a single character over the serial connection—for example, pressing the button for Zone 1 transmitted the character 'A', Zone 2 sent 'B', and so on—enabling the Arduino to interpret the command and toggle the appropriate LED.

On the Arduino side, we used the SoftwareSerial library to communicate with the HC-05 module, enabling the microcontroller to receive data on a separate serial port without interfering with the main USB connection. Similar to IR remote codes, the incoming characters were decoded in a manner consistent with our previous control schemes, simplifying the logic for toggling LEDs.

To improve user experience, the mobile application provided immediate visual feedback by highlighting the selected zone, allowing users to see which lights were currently on or off. The result was an intuitive, interactive interface linking hardware and software in real-time, demonstrating that a mobile app was superior to IR controls for this approach. Taken as a whole, this prototype signalled a significant step forward in making the system far more user-friendly, responsive, and scalable

Prototype Rationale

This prototype served many functions. It tested Bluetooth connectivity between a smartphone and the Arduino, verified that our command protocol worked over the serial link, and introduced the concept of remote control without line-of-sight. Flutter was chosen because it enabled rapid development with expressive layouts and because the flutter_bluetooth_serial package simplified the Bluetooth interface. The GitHub repository we referenced when learning to implement Bluetooth communications in Flutter provided a valuable starting point and served as a great guide and reminder for understanding asynchronous streams and device discovery.

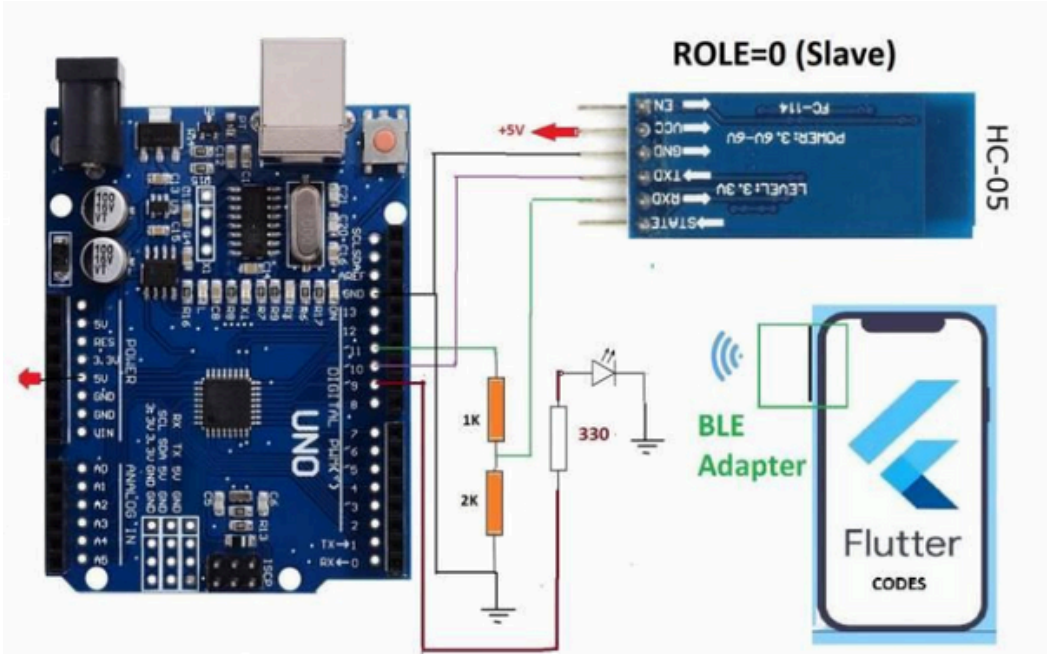

Figure 3.1 Wiring diagram showing the connection between the Arduino Uno and HC-05 module with a smartphone app

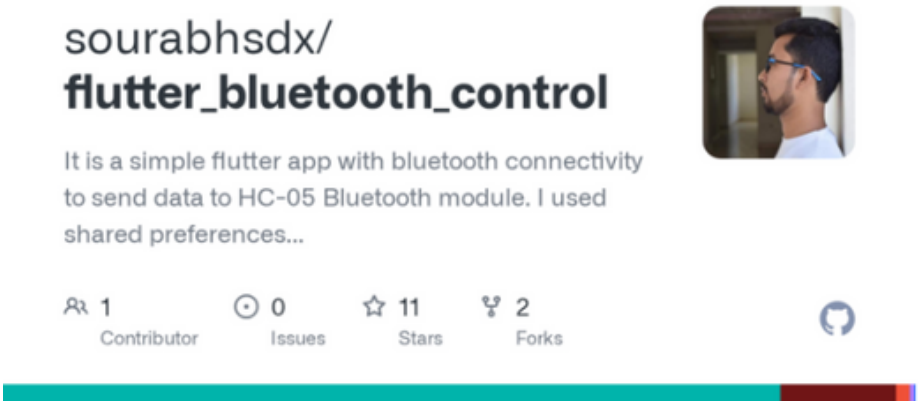

Figure 3.2 Screenshot of an open-source Flutter-Bluetooth control project

Mobile App Development: The Next Generation - Part 2
While working on the mobile app, we faced many challenges dealing with pairing and in general user convenience. The HC-05 Bluetooth module by default requires manual pairing of the smartphone; this is necessary before any interaction can occur, however by Bluetooth settings on the device, an important user must take care during this process. Straightforward for experienced users, can get confused or be too much work for others as the system designed with user-intuitive user experience should be easy to navigate and used. The expectations of many users are that a modern application should manage that works together automatically for the device to integrate, so you don't even have to scroll through the devices until they become part of the set up and connecting to or using these, and are easily manageable. menu or settings.

This limitation showed that the app needs to be developed with user experience in mind, even a technically functional system can feel as simple as connecting to the module and could end up feeling clumsy or inconvenient, if the interface interaction is limited to only basic settings. This challenge eventually became a main area of improvement to the mobile interface for smooth, reliable connectivity and less steps &. For end-user there exists potential for errors.

Automatic Pairing and Unique Device Identification User Interface and Feedback
The UI was much better in the second version of the app. We grouped the six zone buttons into three blocks to reflect our physical model, added icons to illustrate their current state and had a status bar in case of connection status. The app also sent the correct character to the Arduino when the zone button was clicked and modified its colour to yellow for ON or grey for OFF. A log area displayed confirmation messages from the microcontroller. This two-way dialogue was helpful for us to debug the system and reassured users their commands were accepted. We also implemented error support for lost connections and a Reconnect button.

Why This Step Was Important
This phase ended with an application that we could deliver a similar user experience to commercial smart home appliances. The automatic pairing mechanism eliminated the friction related to manual Bluetooth configuration so users can be easily connected to the HC-05 module without going through system settings. On the other hand improved user interface makes operation intuitive, with clear visual feedback, labelled controls, and responsive interactions, allowing users to take over the lighting system efficiently requiring minimal intervention.

These refinements were critical for adoption in real world scenarios. Facility managers, building staff or other non-technical users are far more likely to accept a system that is easy to use and operate, requiring no electronics or programming knowhow. Besides immediate usability, the new app was also the beginning of more advanced features in later incarnations like automated lights scheduling, monitoring in real-time conditions, and possibly integration into wider smart-building platforms. This phase is important not only to assess usability but scalability as well, so the system is not just here and now functional, practical, reliable, well suited for deployment in the real world.

Final Prototype: Multi-block Lumen Control

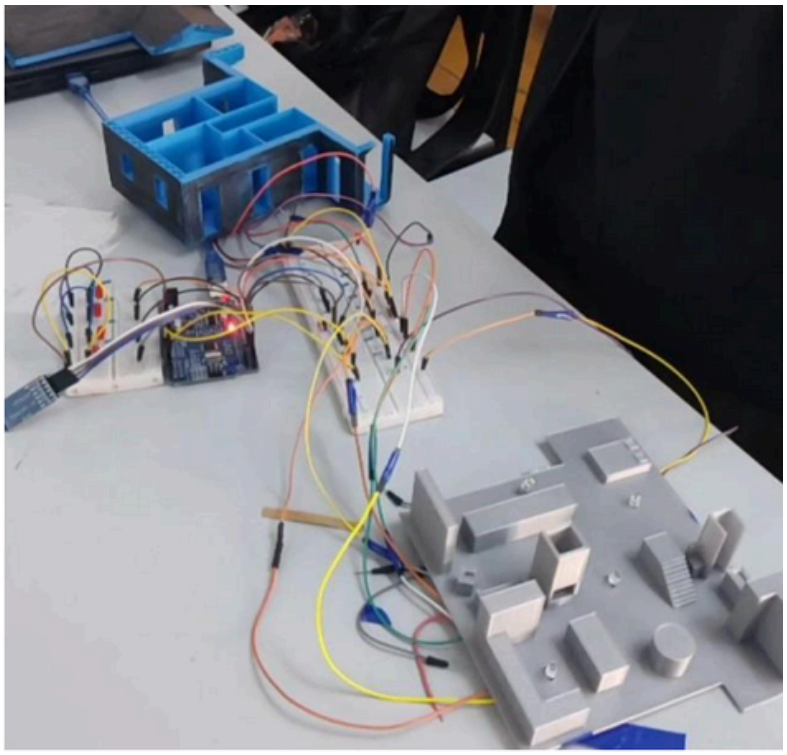

Figure 3.3 -Final Prototype: Multi-block Lumen Control

Our final project had culminated in producing a prototype which melded all the pieces-IR control, Bluetooth control and a physical model—all integrated into one system. We constructed a prototype from three building zones, each containing six LEDs, for a total of eighteen light sources. These were independent zones, halls or hallways in one single building. Foamboard walls were used for the model, a base upon which the electronics were mounted, and little plastic trees for a realistic look. A final model is depicted in Figure 4.1 connected to the Arduino and power bank.

System Operation

The microcontroller controlled LED status and input from IR interface. The microcontroller performed to read input of all the LEDs. Remote and Bluetooth module commands updated the same internal state array and added to the changes to the output pins. Since IR remote and smartphone app used different encoding methods (hexadecimal vs ASCII code) we designed different functions to interpret each piece of input. The system also sent feedback to the smartphone to tell it the state of all lights after each command.

Users could thus decide how to control the entire model in three ways:

1. Physical Buttons (legacy) - Although no longer necessary, the original buttons remained connected for fallback control or demonstration purposes.
2. Infrared Remote - Useful when the user has line of sight to the model and prefers physical keys.
3. Mobile App - Offers the most flexibility, can work through walls and across short distances as long as the Bluetooth signal is available

Results and Demonstrations

In demonstrations the system reliably responded to both IR and Bluetooth commands. Members of the audience enjoyed the option to turn on many rooms at once with a few taps on the app. The physical model, with its brightly lit windows and miniature landscape, helped people realize how the project could grow into a real building. Because the wiring was modular, more LEDs or blocks could be added without rewriting the code; thus our scalability requirement was met. It was a final model that perfectly represents the essence of a smart-lighting system: adaptive control, variety of input methods and a clear visual representation of the state of the building.

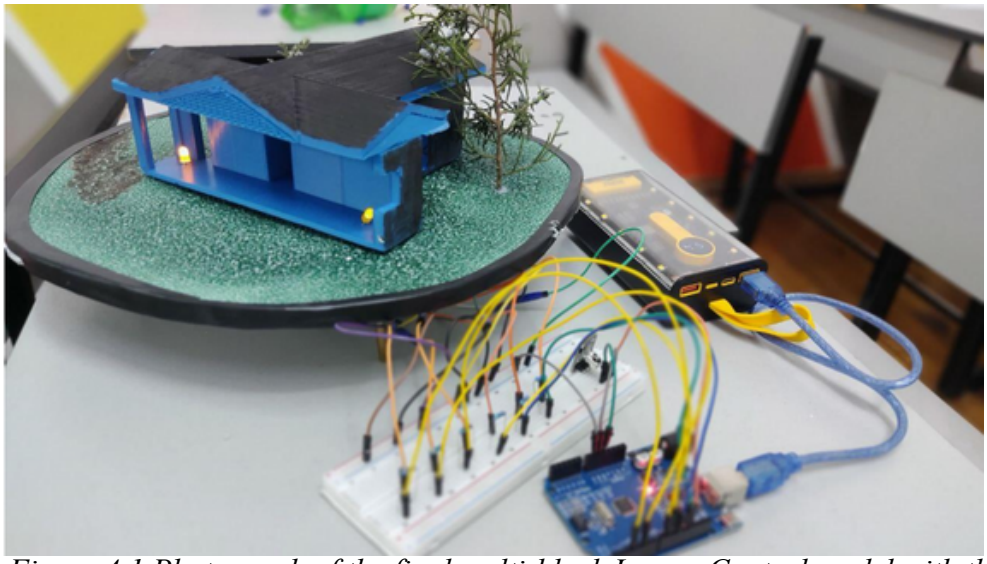

Figure 4.1 Photograph of the final multi-block Lumen Control model with three lighting zones

System Architecture: Hardware

Knowing what hardware architecture it was designed using, explains how the prototypes became a whole system. A table will be provided outlining the basic parts of the Lumen Control project, including their roles. Some parts are shown only in select prototypes (e.g physical buttons in initial phase), others are found throughout.

Component	Role	Quantity
Arduino Uno	Central controller, reads inputs, runs control logic and drives LEDs	1
Breadboard	Prototyping platform, allows quick wiring changes without soldering	1
LEDs (5 mm)	Represent individual light fixtures in each zone	18 total (6 per block)
Resistors (220 Ω)	Limit current through LEDs to prevent burnout	18
Push Buttons	Provide manual input for testing and fallback control	3
IR Receiver (e.g., VS1838B)	Receives infrared commands from the remote control	1
IR Remote Control	Transmits codes to toggle lights without physical contact	1
Bluetooth Module (HC-05)	Enables wireless communication with the smartphone app	1
Power Source (USB / Power Bank)	Supplies 5 V to the Arduino and peripherals	1

The wiring was basic, each LED connected to a dedicated digital pin through a resistor, and the common cathodes were linked to ground. IR receiver required a mere three wires (signal, VCC and GND). Four wires (VCC, GND, TX and RX) were connected to establish the HC-05 module with a voltage divider on the RX pin to reduce Arduino's 5V signal to the 3.3 V with which module is allowed to operate. Such Page16. level shifting can be critical to prevent causing damage to the Bluetooth module. The wiring was already highly modular and the modules themselves had the modular character of the wiring, so new lights simply meant connecting them to unused digital pins.

System Architecture: Software
Software development for the Lumen Control was simultaneously developed, in order to model the evolution of the program, increasing complexity and utility of the project. It consists of two critical parts, namely that of Arduino firmware, responsible for controlling LEDs in hardware, processing incoming commands, and ensuring consistent, reliable operation of the system; and the Flutter mobile application for user interface. Software components were built for modularity.

Arduino firmware is engineered in that manner so that new features, for example new zones, or new types of input devices can be added without need for substantial changes to the current codebase. Also, the mobile app was created based on a scalable interface and flexible communication logic so that additional controls, feedback mechanisms, or new lighting patterns can be effectively incorporated. This modularization makes it possible to make this system capable of being adapted to future requirements, be it through augmenting the number of controllable outputs or implementing alternate wireless protocols, or improving the UI, and limiting the likelihood of getting bugs or disrupting existing functionality.

Arduino Firmware

Arduino firmware written as a C/C++ dialect of Arduino library is:
• **IRremote** - Decodes infrared signals and provides the irrecv.decode() function used in Prototype 3 and later. It manages the complicated timing of IR protocols, which enables us to focus on mapping codes to actions.
• **SoftwareSerial** - Generates a serial interface on arbitrary pins for connecting with HC-05 module. To do that, we use RX and TX pins 10 and 11 on Uno. This frees up the hardware serial port for debugging.
• **EEPROM** (optional) - Provides non-volatile storage for settings or the current state of lights over power cycles. While not implemented in this project, it is available for future enhancements.

The firmware centers around a state machine. The state of each LED is stored in a Boolean array. The loop() does one polling from the IR receiver and Bluetooth module, updating the state array based on commands, it writes the new states to the output pins. Debouncing logic avoids false triggers from mechanical buttons or through noise signals. The firmware switches on the proper LED when a valid command arrives and it returns a confirmation message through the serial connection to the mobile app.

Here, an illustration of the control logic:

```
void loop() {
  if (irrecv.decode(&results)) {
    uint32_t code = results.value;
    handleIRCode(code);
    irrecv.resume();
  }
  if (btSerial.available()) {
    char cmd = btSerial.read();
    handleBTCommand(cmd);
  }
  for (int i = 0; i < NUM_LEDS; i++) {
    digitalWrite(LED_PINS[i], ledState[i]);
  }
}
```

Flutter Application
The mobile application is developed in Dart language using the Flutter framework. It relies on flutter_bluetooth_serial for device discovery, pairing and data transfer. The app architecture is MVC: it records the state of the zones, the view shows buttons and status indicators and the controller communicates with the Arduino. When the user clicks a zone button, the controller sends an ASCII character over Bluetooth. For example, you tap Zone 1 and you get character A, which the Arduino interprets as a request to toggle the first LED. The controller then waits for a response string (such as A1 for on or A0 for off) from the firmware. This response is used to update the model and the view rebuilds to reflect the new state.

The pseudo-code below explains how:

```
void onZonePressed(int zone) async {
  String cmd = String.fromCharCode(65 + zone);
  await bluetoothConnection.output.add(cmd.codeUnits);
  bluetoothConnection.input.listen((data) {
    String response = String.fromCharCodes(data);
    updateZoneState(response);
  });
}
```

What we loved about Flutter: quick fire reloading for quick UI modifications, cross-platform deployment, and a rich set of widgets. On the other hand, we also had to deal with asynchronous streams, as well as the ability to keep the Bluetooth connection in good order to support the app moving to the background. We tackled these issues and created a responsive mobile control panel.

Testing and Evaluation
To identify the success or failure of the Lumen Control system to accomplish the purpose, we prepared the final prototype and took a series of tests. This evaluation concentrated on functionality, reliability, range and, of course, user experience. The major tests and results are summarized in the table below:

Test	Procedure	Result
Button responsiveness	Press each physical button repeatedly and verify that its LED toggles exactly once per press	All buttons toggled reliably; no missed presses or false triggers were observed
IR command recognition	Stand at varying distances (0.5–5 m) and angles (up to 45°) while sending commands	Commands were recognised up to 4.5 m with a clear line of sight. Beyond 5 m or with obstacles, some commands were missed
Bluetooth range	Move the smartphone away from the model while sending commands every 5 s	Reliable communication up to ~10 m indoors; beyond this, the signal weakened and the connection dropped
State persistence	Toggle various LEDs, power cycle the system and observe if lights return to previous states	Without EEPROM saving the lights reset to off; with EEPROM enabled they restored correctly

Test	Procedure	Result
Simultaneous control	Send commands via IR and Bluetooth alternately to different zones	Firmware handled both inputs without conflict; states updated correctly
User satisfaction	Ask test participants to operate the system via the app and remote	Users found the app intuitive and preferred it over the IR remote; comments suggested adding labels to the physical model for clarity

The system generally did well. The IR remote worked nicely and generated quick responses but needed line-of-sight. Bluetooth provided a higher degree of flexibility but had less range than expected because of indoor obstacles. The combination of them meant that users always had a way to control the lights. We observed that the physical model facilitated users comprehending which zone they controlled, highlighting the benefit of visual feedback.

Challenges and Solutions

The project was marred by several problems throughout the development pace and stability of our system. But they went from hardware-related troubles to all manner of other reasons, software integration challenges and environmental interference also caused problems. Although some of these initial hurdles slowed us, identifying and overcoming these initial problems became an integral part of refining the prototype. All those challenges forced us to scrutinize our design, reconsider our assumptions, and implement more robust technical solutions. Implementing the improvements we made not just resolved the immediate problems, but helped us to improve their workflow significantly and improved the stability of the system. We have provided some clearer solutions by the solutions we created. We made clear the guidelines for our recommendations moving forward, facilitated us in creating a more stable architecture, and informed our suggested areas of further work.

Mechanical Bounce and Debouncing

Challenge: While designing the prototype, we found a common problem of how the system processed input from physical buttons. Each button was intended to accomplish a single clean transition-for instance, turning on an LED or switching it off. From what was supposed to be a single input, multiple rapid transitions occurred instead of one. This caused the LEDs to flicker, switch unpredictably, or perform several actions from a single input. At first, this seemed like a coding error or an electrical fault. We tested the wiring, examined resistors, and checked the logic controlling the LED outputs, but the real trouble remained. Further investigation revealed that mechanical switch bounce was the driving factor. When a physical button was pressed, the metal contacts inside don't immediately connect; they instead vibrate and bounce for short periods, generating a series of fast on/off signals. Even though this takes place within a few milliseconds, the Arduino can read each of these fluctuations. As a result, one physical press was interpreted by the microcontroller as multiple inputs, resulting in unstable and unpredictable behavior from the LED.

Solution: We overcame this by applying a debouncing function that eliminated accidental input signals. The logic worked by recording the timestamp of the last valid button press and then ignoring any subsequent changes within the next 50 milliseconds. This brief delay was long enough to allow the button contacts to settle into a stable state while still keeping the system responsive. With debouncing in place, each button press yielded exactly one reliable transition, eliminating flickering and accidental multi-triggering.

Moreover, once this solution worked well for physical buttons, we generalized the same logic to other inputs (e.g., IR remote commands, Bluetooth-based interactions, etc.). Although these inputs do not experience mechanical bounce, they may occasionally produce rapid repeated signals because of transmission noise and prolonged button holds on remote devices. Reusing the debouncing logic guaranteed consistency across all input sources, avoided unexpected behavior, and reduced the overall control system's unpredictability, resulting in considerable stability and predictability.

Ambient IR Noise

Challenge: We also found in the testing stage that the IR receiver was occasionally registering commands even when no remote control was used. Initially, these false triggers appeared random, but after several tests under different lighting conditions, we identified a pattern: fluorescent lights and natural sunlight were emitting infrared radiation that interfered with the receiver. Because IR sensors are designed to detect specific wavelengths of light, any strong ambient IR source-whether from overhead lamps or direct sunlight entering through a window-could overwhelm the sensor or cause it to misinterpret environmental radiation as an intentional signal. This caused the system to react to commands that were never actually sent. The issue was especially noticeable when working in brightly lit rooms or testing near windows during daytime, making the receiver unreliable in uncontrolled lighting environments.

Solution: To prevent this interference, we used a physical and software-based method. First, we shielded the IR receiver by placing it inside a small tube, which served as a protective barrier against external infrared noise. This tube dramatically reduced the sensor's field of view, ensuring it detected primarily signals coming directly from the remote rather than from surrounding light sources. In addition to the physical shielding, we also built in a filtering feature in the software: the system ignored any IR codes that were not recognized or did not match a set of valid, predefined commands. This prevented unwanted or corrupted signals from triggering any actions. Finally, during regular operation, we made sure to position the receiver away from direct exposure to strong light sources—especially direct sunlight or the immediate glare of fluorescent bulbs. With these measures combined, the IR receiver became much more stable, greatly reducing the number of accidental triggers and improving the overall reliability of the system.

Bluetooth Pairing Friction

Challenge: When using the HC-05 Bluetooth module, we repeatedly encountered a problem: the mobile app was unable to reliably connect to the module. Initially, the problem seemed to be software, since the application kept failing to establish a reliable connection. Despite many attempts and troubleshooting, it was found that the true culprit was the module's default setup. Out of the box, the HC-05 needed users to manually pair it via the phone's Bluetooth settings, which wasn't just laggy for the connectivity system-but it also annoyed users because they needed to go through the process multiple times. Furthermore, the module's default name complicated its detection among other nearby Bluetooth devices, which only compounded the confusion and led some to believe the app "couldn't find" the correct device.

Solution: In order to fix this, we went to AT mode on the HC-05 to fully configure it before integrating it with the app. We set a catchy, easily recognisable name for the module to enable people to instantly identify it as being ours. This alone helped make the pairing process well-defined and avoided the risk of testers selecting the wrong device. We added a password to this connection so it was secured and recorded the MAC address of the module as a unique identifier. With the above MAC address, we created the Flutter application to search for and connect to its respective module automatically and without any manual pairing from the user. These adjustments made the Bluetooth workflow much smoother, faster, and more intuitive, which resolved the previous connection struggle.

Level Shifting for HC-05

Challenge: In the course of hardware integration, we discovered a critical technical problem in electrical compatibility with the HC-05 Bluetooth module's RXD pin. The RXD input here is optimized for working with 3.3-volt logic levels, and the Arduino board outputs signals at 5 volts. The HC-05 is tolerant enough for short exposure to higher voltages, but constant operation at 5 V poses a very real risk of permanent damage, signal instability, or even complete failure of the module. This challenge was particularly relevant because the RXD line is responsible for all incoming serial communication from the Arduino. Without proper voltage matching, the module could behave unpredictably, fail to respond to commands, or degrade over time. Therefore, we needed to ensure that the interface between these two devices was safe and stable to avoid hardware failures and guarantee reliable communication.

Solution: So we added a voltage divider to solve this problem with a straightforward but effective voltage-shifting method directly on the RXD line. We built a passive circuit using two resistors-1 kΩ and 2 kΩ-which lowers the Arduino's 5 volts to a safe 3.3 volts required by the HC-05. This is a well-documented approach and recommended in Arduino hardware specifications-and offers a stable, low-cost solution without requiring extra components, like dedicated logic-level converters. As demonstrated in Figure 3.1 and explained in the Arduino Project Hub, the resistor divider guarantees that every signal sent from the Arduino to the Bluetooth module remains within the acceptable voltage range. After introducing this modification, the resulting communication was completely reliable, effectively eliminating risks that could have damaged the HC-05, which will help maintain overall system stability.

Scaling the System

Challenge: When the project expanded and we started adding more LEDs and building blocks to represent different parts of the system, we suddenly hit a wall with scalability. Each successive LED had to have its own dedicated digital pin on the Arduino, and each additional block required writing more repetitive code to manage individual light states. This process left the program getting messier, more demanding, and more prone to errors. Manually controlling every LED wasted too many pins and made the codebase more brittle because any structural adjustment-such as increasing the number of LEDs or rearranging the system-entailed rewriting parts of the logic. It made clear the challenge: we had the restriction of controlling individual outputs one at a time, particularly as the prototype grew beyond the minimal initial setup.

Solution: To cope with this scalability problem, we revamped the LED control logic by using arrays and loops, which enabled us to handle multiple outputs in a much more methodical and efficient manner. Instead of writing separate code blocks for each LED, we used a single array to store all of the LED pins and updated their states using simple loops collectively. This not only reduced redundancy but also made the logic much easier to change. If we ever needed to adjust the number of LEDs, we just needed to change one constant, NUM_LEDS, and the rest of the code would adapt automatically. For long-term scalability, we also explored hardware-based solutions like shift registers and I²C port expanders, which enable dozens of LEDs or blocks to be driven using only a few Arduino pins. While we have not fully implemented these options in the current prototype, considering them at this stage gave us a roadmap for continued development and assured us that the system could be extended significantly without redesigning the core architecture.

Power Consumption and Heat
Challenge:
We extended the prototype to the number of concurrently lit LEDs and faced additional hurdles: electrical and thermal challenges. The system utilized many LEDs and their overall current draw was much higher. Not only did this increase the requirement for power supply but also caused the physical model to warm up noticeably during operation. Arduino and LEDs could withstand moderate currents, but high currents for extended periods of time can overheat components, leading to reduced lifetime and system instability. Electrical load and heat dissipation was critical to sustaining the prototype by ensuring it remained safe whilst running safely and adequately on continuous demand.

Solution:
So you needed a USB power bank that can offer more than 1 A of current, which supplied each LED with a stable and adequate current without putting the Arduino's onboard regulator to the test. In addition, we also implemented some spacing and ventilation in the model to give the heat at the LEDs space to dissipate where required and avoid hotspots as well. 220 Ω resistors also prevented current from flowing through the LEDs so that each LED worked within the safe ranges of electrical conductivity and there was no excess of thermal temperature at the component level. This included not only the possibility to operate multiple LEDs simultaneously, but temperatures as safe as possible, as well as the maintenance to ensure the life and well-being of the components. Each of these problems provided us the opportunity to explore working design in practice and the example of how environmental noise, convenience for end users and electrical limitations directly affect system design. Each obstacle that we encountered (e.g., fluctuating input signals, Bluetooth connectivity issues, voltage mismatches, thermal challenges) forced us to investigate how hardware and software interact in service and how it affects the user experience. But we were able to mitigate some of these problems and the system continued to be reliable and stable and work with all sorts of environments. At the same time, the usability of the solution improved and users had an improved experience with the interface. Overcoming these challenges not only made the current prototype more user friendly but highlighted the underlying problems that we had to overcome in real world scalability, reliability, stability and user friendly hardware-software systems.

Future Work and Enhancements
Although our main goals were met through the deliverable prototype, we can consider several directions for future improvement:

• **Wireless Networking**: Replace the HC-05 module with an ESP8266 or ESP32 microcontroller, which would provide Wi-Fi connectivity with the codebase you have developed. This would allow the lights to be controlled across campus from anywhere, via the internet to be linked to voice assistant or online dashboard.

• **Sensor Addition**: Integrate motion detectors and light sensors for automatically controlling light sources as per occupancy and ambient light levels. That would save energy in conference rooms and private offices with occupancy sensors by up to 30 percent. When the lighting gets dimmed, based upon a feature such as dimming based on daylight harvesting, the use would be cut down by 20 to 40 percent.

• **Scheduling and Timer Features**: Integrate a real-time clock and scheduling capabilities in the Arduino firmware. Lights could be turned on and off at specific times (e.g. when classes start or end) without any user intervention at all.

• **Networked Control Panels**: Have a web-based dashboard that indicates the status of each zone and allows admin-level users to set up scenes or lock controls. A boon to larger campuses, first and foremost.

• **Hardware Miniaturisation**: Develop custom printed circuit boards (PCBs) and/or modules that include the microcontroller, power regulation and communication interfaces. This would result in decreased wire clutter, increased reliability, and easier installation.

• **Energy Monitoring**: Track actual power usage and give feedback on savings using current sensors. It can be logged and analysed to optimise the system.

With these improvements, the prototype would eventually mature into a fully fledged smart-lighting system which could be deployed in actual buildings.

Conclusion

The Lumen Control project shows how an iterative approach can transform a simple idea for remote control of LEDs to a full-blown smart-lighting prototype. Thanks to an easy and straightforward three-LED array and necessary software sophistication, infrared/Bluetooth communication, model construction and cross-platform mobile app building are realized. At the end of the prototype, people could control several zones with a smartphone or IR remote - and the system could scale up to more lights without needing to be refined anew.

Our effort tackles the top challenges of a contemporary facility: energy efficiency, user experience and potential scaling. The application demonstrates the benefits of the open-source tools (Arduino, Flutter, etc.) with rapid development and multi-level cooperation. Along the way, we faced and coped with problems such as switch bouncing and wireless pairing as well as learned several lessons in resiliency and usability to date.

We would like this report to be a source for students and practitioners who want to build their own intelligent lighting systems. For improved technology and connected hardware-Wi-Fi, sensors and energy tracking-lighting up the Lumen Control system could be successful in schools, offices, and domestic settings to make things more sustainable and practical.

Appendix A: Arduino Code & Diagrams

In this appendix we will show abridged versions of the Arduino sketches of our prototypes. Full source code is available upon request.

A.1 – Prototype 2 Toggle Logic

```
const int btnPins[3] = {4, 5, 6};
const int ledPins[3] = {9, 10, 11};
bool ledState[3] = {false, false, false};
unsigned long lastDebounce[3] = {0, 0, 0};
const unsigned long debounceDelay = 50;

void setup() {
  for (int i = 0; i < 3; i++) {
    pinMode(btnPins[i], INPUT);
    pinMode(ledPins[i], OUTPUT);
  }
}

void loop() {
  for (int i = 0; i < 3; i++) {
    int reading = digitalRead(btnPins[i]);
    if (reading == HIGH && (millis() - lastDebounce[i] > debounceDelay)) {
      ledState[i] = !ledState[i];
      digitalWrite(ledPins[i], ledState[i]);
```

```
      lastDebounce[i] = millis();
    }
  }
}
```

A.2-IR remote control(Prototype 3)

```
#include <IRremote.h>
const int recvPin = 2;
IRrecv irrecv(recvPin);
decode_results results;
const int ledPins[6] = {3,4,5,6,7,8};
bool ledState[6] = {false};

void setup(){
  Serial.begin(9600);
  irrecv.enableIRIn();
  for(int i=0;i<6;i++){ pinMode(ledPins[i],OUTPUT); }
}

void loop(){
  if(irrecv.decode(&results)){
    switch(results.value){
      case 0xFFA25D: toggleLED(0); break; // Button 1
      case 0xFF629D: toggleLED(1); break; // Button 2
      // add cases for other buttons
    }
    irrecv.resume();
  }
}

void toggleLED(int idx){
  ledState[idx] = !ledState[idx];
  digitalWrite(ledPins[idx], ledState[idx]);
}
```

A.3 Bluetooth control integration

```
#include <SoftwareSerial.h>
SoftwareSerial btSerial(10, 11); // RX, TX
const int ledPins[6] = {3,4,5,6,7,8};
bool ledState[6] = {false};

void setup(){
  Serial.begin(9600);
  btSerial.begin(9600);
  for(int i=0;i<6;i++) pinMode(ledPins[i], OUTPUT);
}

void loop(){
  if(btSerial.available()){
    char cmd = btSerial.read();
    int idx = cmd - 'A';
    if(idx >=0 && idx <6){ toggleLED(idx); }
```

```
  }
}

void toggleLED(int idx){
  ledState[idx] = !ledState[idx];
  digitalWrite(ledPins[idx], ledState[idx]);
  // send feedback
  btSerial.print((char)('A'+idx));
  btSerial.println(ledState[idx]);
}
```

Appendix B: Flutter app code

Below is a simplified version of the Flutter code used to connect to the HC-05 and send commands. The full project is available on our repository.

```
import 'package:flutter/material.dart';
import 'package:flutter_bluetooth_serial/flutter_bluetooth_serial.dart';

class LumenControlApp extends StatefulWidget {
  @override
  _LumenControlAppState createState() => _LumenControlAppState();
}

class _LumenControlAppState extends State<LumenControlApp> {
  BluetoothConnection? _connection;
  final String targetAddress = '00:21:13:01:23:45';
  List<bool> zoneStates = List<bool>.filled(6, false);

  Future<void> _connect() async {
    try {
      _connection = await BluetoothConnection.toAddress(targetAddress);
      _connection!.input?.listen((data) {
        String resp = String.fromCharCodes(data).trim();
        int idx = resp.codeUnitAt(0) - 65;
        setState(() {
          zoneStates[idx] = resp.endsWith('1');
        });
      });
    } catch (e) {
      print('Cannot connect: \$e');
    }
  }

  void _toggleZone(int index) {
    if (_connection != null) {
      String cmd = String.fromCharCode(65 + index);
      _connection!.output.add(Uint8List.fromList(cmd.codeUnits));
    }
  }

  @override
  Widget build(BuildContext context) {
```

```
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(title: Text('Lumen Control')),
        body: Column(
          children: [
            ElevatedButton(
              onPressed: _connect,
              child: Text('Connect'),
            ),
            GridView.builder(
              shrinkWrap: true,
              gridDelegate: SliverGridDelegateWithFixedCrossAxisCount(
                crossAxisCount: 3,
                mainAxisSpacing: 10,
                crossAxisSpacing: 10,
              ),
              itemCount: 6,
              itemBuilder: (context, index) {
                return ElevatedButton(
                  style: ElevatedButton.styleFrom(
                    primary: zoneStates[index] ? Colors.amber : Colors.grey,
                  ),
                  onPressed: () => _toggleZone(index),
                  child: Text('Zone \${index + 1}'),
                );
              },
            ),
          ],
        ),
      ),
    );
  }
}
```

References

1. PacLights Learning Center. Why Office Light Control Systems Are a Game-Changer for Energy Efficiency. This article explains that lighting accounts for roughly 30 % of commercial energy use and that smart control systems can reduce consumption by 30 – 70 %.

2. Arduino Project Hub. Arduino Uno HC-05 Bluetooth Controlled LED Using Mobile. Provides wiring guidance for connecting the HC-05 module to the Arduino, including using the GND, VCC, TX and RX pins.

Youtube Video Link : https://youtu.be/wXa9NTvVRF8?si=UsivvuLzPYdZjx2B

GitHub Repositoryy QR code: :