IE2070
Embedded Systems
2nd Year, 2nd Semester

Assignment

Assignment (Individual)

Submitted to

Sri Lanka Institute of Information Technology

In partial fulfilment of the requirements for the

Bachelor of Science Special Honors Degree in Information Technology

04/05/2024

Declaration

I certify that this report does not incorporate without acknowledgement, any material previously submitted for a degree or diploma in any university, and to the best of my knowledge and belief, it does not contain any material previously published or written by another person, except where due reference is made in text.

Registration Number: IT22360496

Name: D R Wickrama Arachchi

Table of Contents

1. Introduction 1

2. Design Methodology 2

2.1. Power Supply 2

2.2. Microcontroller 2

2.3. LED Configuration 2

2.4. Remote Control Interface 2

2.5. Connectivity 3

3. Code Implementation 4

4. Circuit Diagram 16

5. Testing and Validation 18

6. Discussion 20

6.1. Future Improvements 20

6.2. Key Challenges 20

References 21

1. Introduction

This report details the design, development, and testing of a circuit utilising an infrared (IR) remote controller and an Arduino UNO microcontroller to control an LED lamp remotely. The lamp consists of one multicolor LED and four white LEDs, all powered by a battery. Using the volume-up, volume-down, and channel-down buttons on the remote, users can adjust the lamp's brightness and state, whether that be on or off. In addition, the lamp operates in a particular order, which is detailed in the Design Methodology section.

2. Design Methodology

The design process involved several key decisions to ensure functionality and ease of use:

2.1. Power Supply

The LED lamp and Arduino UNO microcontroller will be powered by a 9-volt battery with a snap connector clip. The use of a battery provides flexibility and portability.

2.2. Microcontroller

An Arduino UNO R3 was used as the microcontroller, programmed via a USB cable. Being compatible and simple to program, the Arduino UNO can interface with a wide range of electronic components, including LED drivers and infrared receivers.

2.3. LED Configuration

As previously stated, the LED lamp will include four white LEDs and one multicolor LED. The common-cathode multicolor LED will provide additional functionalities and aesthetic elements; however, for general illumination, white LEDs will be used. Resistors will be affixed to each LED anode to control the current to a level suitable for each LED.

Seven resistors will be utilised: one of 2.2kΩ connected to the VCC pin of the IR sensor, and six 330Ω resistors—one each for the six anodes of the white LEDs and the common-cathode multicolor LED.

2.4. Remote Control Interface

The project will use an IR sensor and a standard IR remote controller to operate the LED lamp. The volume-up, volume-down, and channel-down buttons on the remote will be used to operate the lamp in the following sequence:

The LED lamp will be inactive at the outset. Upon the first press of the volume-up button, the first white LED will illuminate. Each subsequent press of the volume-up button will activate the next white LED in sequence. Once all four white LEDs are lit, with each additional press of the volume-up button, the multicolor LED will endlessly cycle through its three colours one at a time. Pressing the volume-down button will turn off the most recently activated light or colour, continuing the sequence in reverse until all the LEDs have been turned off. While the multicolor LED is not turned on , the channel-down button is designated to decrease the brightness of the most recently activated white LED.

2.5. Connectivity

All the components mentioned above will be connected as demonstrated in the Circuit Diagram section on a dot board and soldered together with the use of a soldering gun. Connections from the battery to the Arduino UNO board and connections from the Arduino UNO board to the IR sensor and LEDs will utilise male-to-female jumper wires and male header pins soldered onto the dot board. All connections to the ground for the relevant components will be made by forming a common ground for the components and using jumper wires.

3. Code Implementation

The C programming language was used to write the project's code in Microchip Studio. The code followed the necessary steps to complete the task. The Arduino UNO is able to read infrared signals from the remote controller, allowing the adjustment of the LEDs accordingly. The code uses an organised approach to handle various remote inputs and LED statuses, ensuring smooth functioning, and was thoroughly tested to ensure it was operating as intended. The code utilises the IRremote.h library written by Liviu Istrate for receiving remote controller codes.

```
#define F_CPU 16000000UL


#include <avr/io.h>

#include <avr/interrupt.h>

#include <util/delay.h>

#include <IRremote.h>


// Pin definitions for LEDs

#define RED   PD2

#define GREEN PD4

#define BLUE  PD7

#define WHITE_1   PD3

#define WHITE_2   PD5

#define WHITE_3   PD6

#define WHITE_4   PB3


// Pin definition for IR sensor
```

```
#define IR_SENSOR PB0


// IR Remote button codes

#define VOLUME_UP_CODE 21

#define VOLUME_DOWN_CODE 7

#define CHANNEL_DOWN_CODE 69


// Declare variables

int currentLEDState = 0;

bool isColorToggling = false;

int colorToggleCounter = 1;

int ledBrightness[4] = {255, 255, 255, 255};

int brightnessReduced[4] = {0, 0, 0, 0};

int loopCounter = 0;


// Function to switch an LED on

void switchLEDsOn() {

     currentLEDState++;

     if (currentLEDState > 7) {

          loopCounter++;

          currentLEDState = 7;

          isColorToggling = true;

          colorToggleCounter = 1;
     }
}

// Function to switch an LED off
void switchLEDsOff() {
     currentLEDState--;
     if (currentLEDState <= 0) {
          currentLEDState = 0;
     }
     ledBrightness[currentLEDState] = 255;
}

// Function to toggle between RGB colors
void toggleColors() {
     colorToggleCounter++;
     if (colorToggleCounter > 3) {
          colorToggleCounter = 1;
          loopCounter++;
     }
}

// Function to switch RGB LED off
void exitRGB() {
     currentLEDState--;
     if (currentLEDState <= 0) {
          currentLEDState = 0;
     }
}

// Function to exit toggling mode of RGB LED
void exitToggling() {
     colorToggleCounter--;
     if (colorToggleCounter < 1 && loopCounter > 0) {
          colorToggleCounter = 3;
          loopCounter--;
          if (loopCounter <= 0) {
               isColorToggling = false;
               loopCounter = 0;
          }
     }
}

// Function to switch off all the LEDs
void setAllLEDsLow() {
     OCR2B = OCR0B = OCR0A = OCR2A = 0;
     DDRD &= ~((1 << WHITE_1) | (1 << WHITE_2) | (1 << WHITE_3));
```

```
    DDRB &= ~(1 << WHITE_4);

    PORTD &= ~((1 << RED) | (1 << GREEN) | (1 << BLUE));

}


// Function to switch on and set brightness of the white LEDs

void setWhiteLEDs(int count) {

    if (count >= 1) {

        DDRD |=  _BV(DDD3);

        TCCR2A |= (_BV(COM2B1) | _BV(WGM21) | _BV(WGM20));

        TCCR2B |=  (_BV(CS20));

        OCR2B = 255;


        if (brightnessReduced[0] == 1) {

            OCR2B = ledBrightness[0];

            _delay_ms(10);

        }


    } else {

            OCR2B = 0;

            DDRD &= ~(1 << WHITE_1);

    }


    if (count >= 2) {
        DDRD |=  _BV(DDD5);
        TCCR0A |= (_BV(COM0B1) | _BV(WGM01) | _BV(WGM00));
        TCCR0B |=  (_BV(CS00));
        OCR0B = 255;

        if (brightnessReduced[1] == 1) {
            OCR0B = ledBrightness[1];
            _delay_ms(10);
        }

    } else {
        OCR0B = 0;
        DDRD &= ~(1 << WHITE_2);
    }

    if (count >= 3) {
        DDRD |=  _BV(DDD6);
        TCCR0A |= (_BV(COM0A1) | _BV(WGM01) | _BV(WGM00));
        TCCR0B |=  (_BV(CS00));
        OCR0A = 255;

        if (brightnessReduced[2] == 1) {
            OCR0A = ledBrightness[2];
          _delay_ms(10);
     }

} else {
     OCR0A = 0;
     DDRD &= ~(1 << WHITE_3);
}

if (count >= 4) {
     DDRB |=  _BV(DDB3);
     TCCR2A |= (_BV(COM2A1) | _BV(WGM21) | _BV(WGM20));
     TCCR2B |= (_BV(CS20));
     OCR2A = 255;

     if (brightnessReduced[3] == 1) {
          OCR2A = ledBrightness[3];
          _delay_ms(10);
     }

} else {
     OCR2A = 0;
     DDRB &= ~(1 << WHITE_4);
}
```

```
    PORTD &= ~((1 << RED) | (1 << GREEN) | (1 << BLUE));
}

// Function to switch RGB colour on
void setRGBLED(int ledPin) {
    PORTD &= ~((1 << RED) | (1 << GREEN) | (1 << BLUE));
    PORTD |= (1 << ledPin);
}

// Handle the LEDs
void updateLEDs() {
    if (!isColorToggling) {
        // Handle white LEDs and RGB LEDs for non-color toggle
mode
        switch (currentLEDState) {
            case 0: setAllLEDsLow(); break; // All off
            case 1: setWhiteLEDs(1); break;
            case 2: setWhiteLEDs(2); break;
            case 3: setWhiteLEDs(3); break;
            case 4: setWhiteLEDs(4); break;
            case 5: setRGBLED(RED); break; // Red LED on
            case 6: setRGBLED(GREEN); break; // Green LED on
```

```
                case 7: setRGBLED(BLUE); break; // Blue LED on
            }
        } else {
            // Handle RGB LED for color toggle mode
            switch (colorToggleCounter) {
                case 0: setWhiteLEDs(4); break; // All off
                case 1: setRGBLED(RED); break; // Red LED on
                case 2: setRGBLED(GREEN); break; // Green LED on
                case 3: setRGBLED(BLUE); break; // Blue LED on
            }
        }
}

// Action of volume up button
void volumeUp() {
        if (isColorToggling) {
            toggleColors();
        } else {
            switchLEDsOn();
        }
        updateLEDs();
        _delay_ms(10); // Debounce delay
}
```

```
// Action of volume down button
void volumeDown() {
    if (currentLEDState > 4 && !isColorToggling) {
        exitRGB();
    } else if (isColorToggling) {
        exitToggling();
    } else {
        switchLEDsOff();
    }
    updateLEDs();
    _delay_ms(10); // Debounce delay
}

// Action of channel down button
void channelDown() {
    if (currentLEDState == 0) {
        return;
    }
    ledBrightness[currentLEDState - 1] = 30;
    brightnessReduced[currentLEDState - 1] = 1;
    updateLEDs();
}
```

```
// Main function
int main(void) {
    uint16_t address = 0; // Variable to store IR address
    uint16_t command = 0; // Variable to store IR command

    // Initialize white LEDs as outputs
    DDRD |= ((1 << WHITE_1) | (1 << WHITE_2) | (1 << WHITE_3));
    DDRB |= (1 << WHITE_4);
    // Initialize RGB LEDs as outputs
    DDRD |= ((1 << RED) | (1 << GREEN) | (1 << BLUE));

    IR_init(); // Initialize IR remote control

    while(1) {
        if (IR_codeAvailable()) { // Check if IR code is
available
            if (!IR_isRepeatCode()) { // Check if IR code is
not a repeat
                IR_getCode(&address, &command); // Get IR
address and command
                if (command == VOLUME_UP_CODE) {
                    volumeUp();
                } else if (command == VOLUME_DOWN_CODE) {
                    volumeDown();
                } else if (command == CHANNEL_DOWN_CODE) {
                    channelDown();
                }
            }
        }
    }
}
```

4. Circuit Diagram

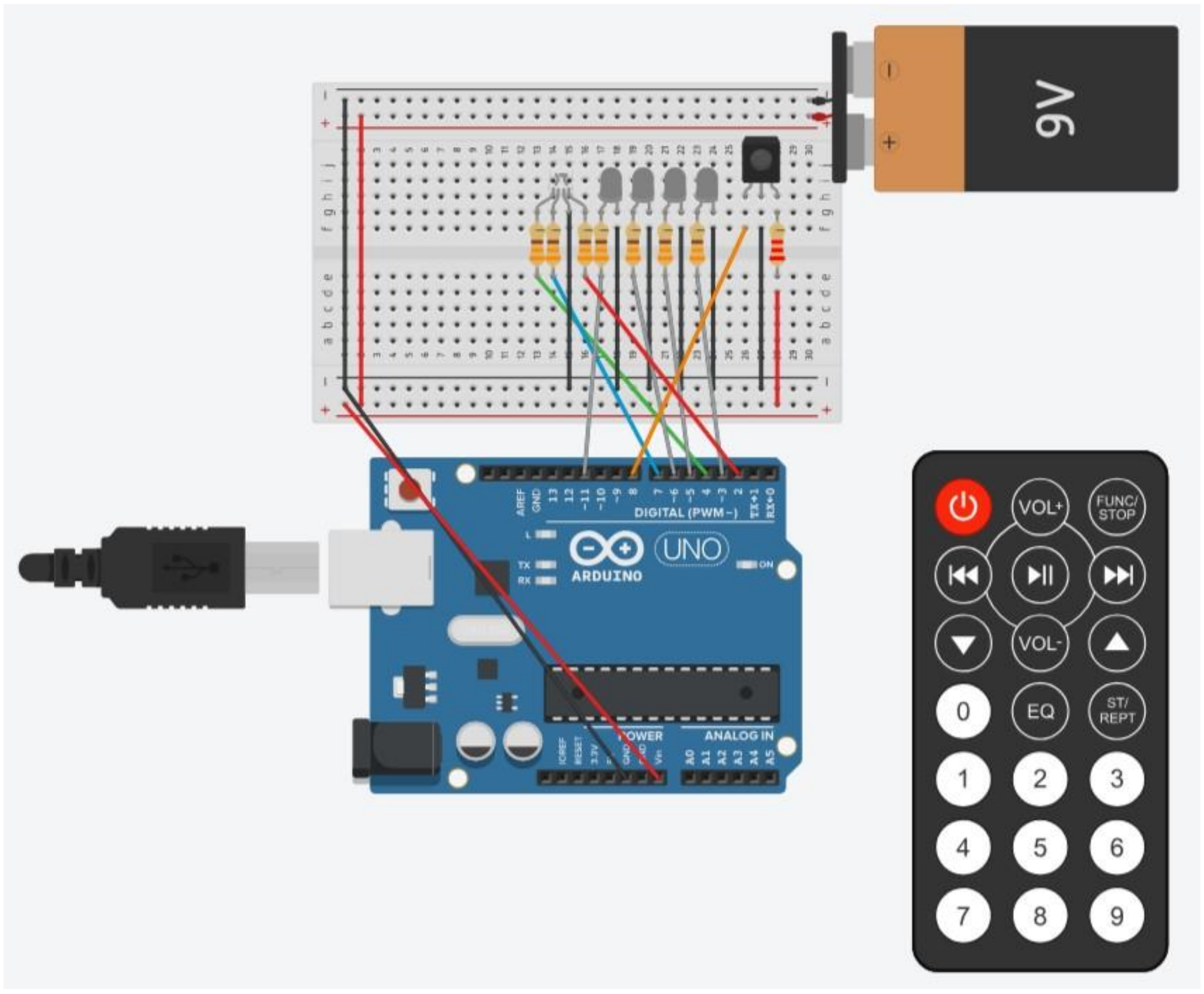


Figure 4.1: Breadboard circuit view

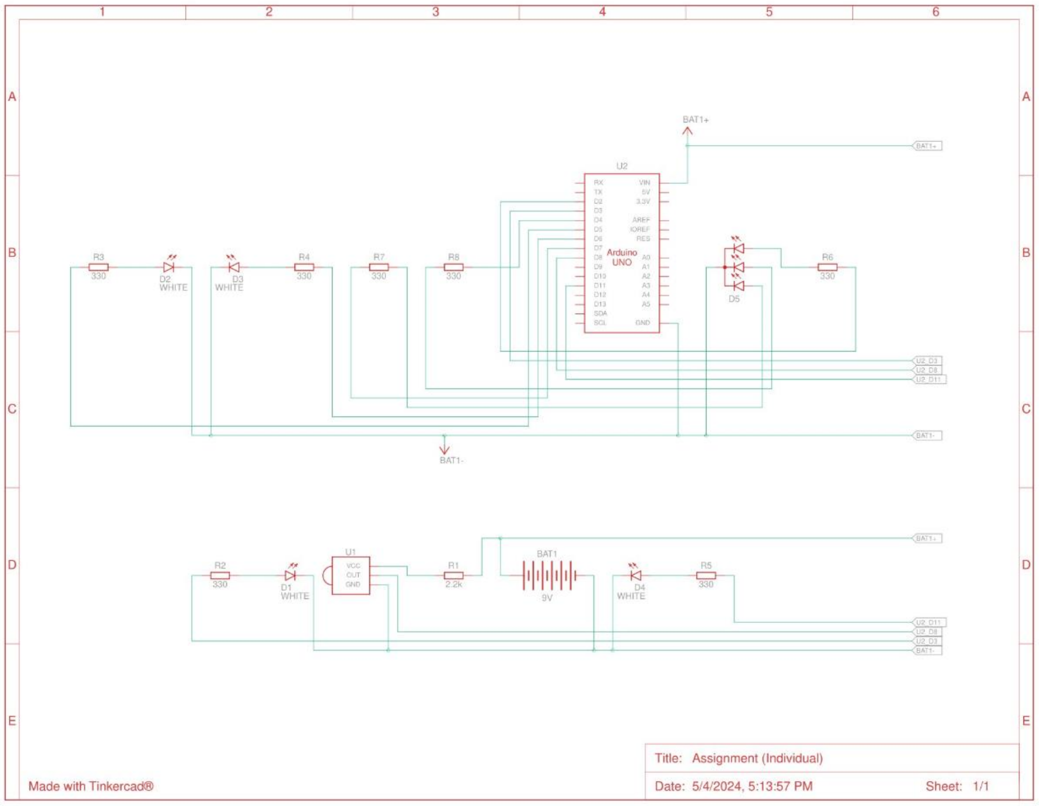


Figure 4.2: Schematic view

5. Testing and Validation

To ensure it met the required standards, the circuit underwent thorough testing. The testing process involved the following steps:

1. Checking the power supply.

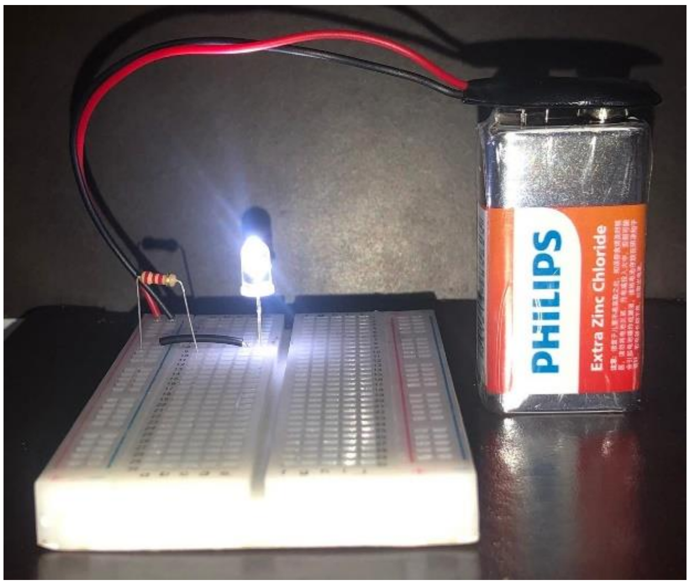
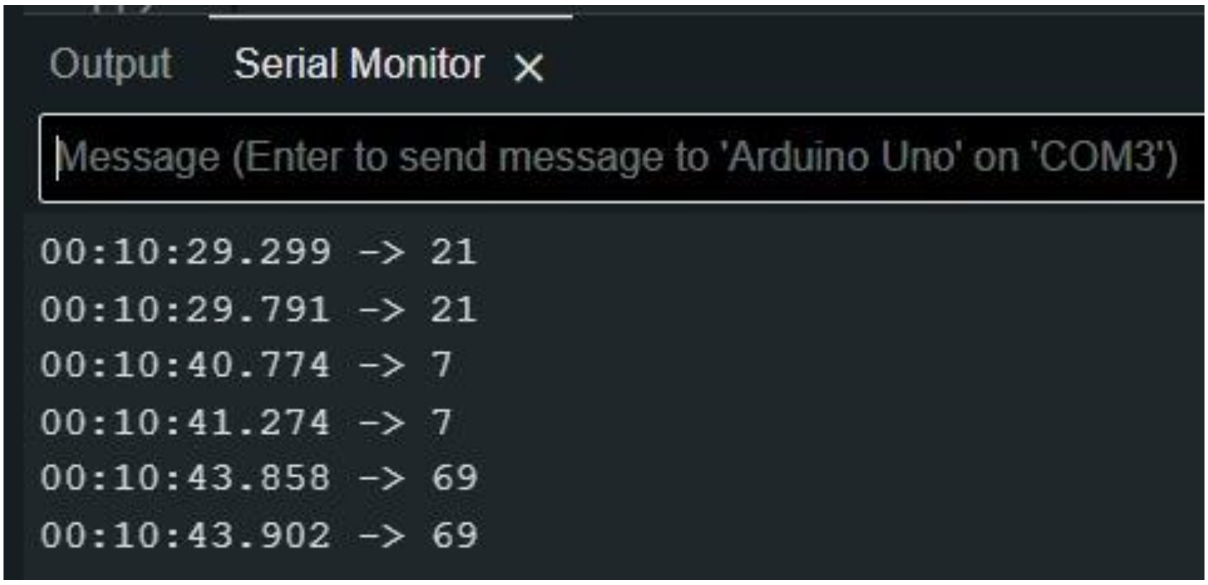


Figure 5.1: Verifying the power supply

2. Verifying the button mapping and operation of the IR remote control.



```
Message (Enter to send message to 'Arduino Uno' on 'COM3')
00:10:29.299 -> 21
00:10:29.791 -> 21
00:10:40.774 -> 7
00:10:41.274 -> 7
00:10:43.858 -> 69
00:10:43.902 -> 69
```

Figure 5.2: Testing IR remote functionality and button mapping

3. Confirming that the LED control sequence matches the specifications.

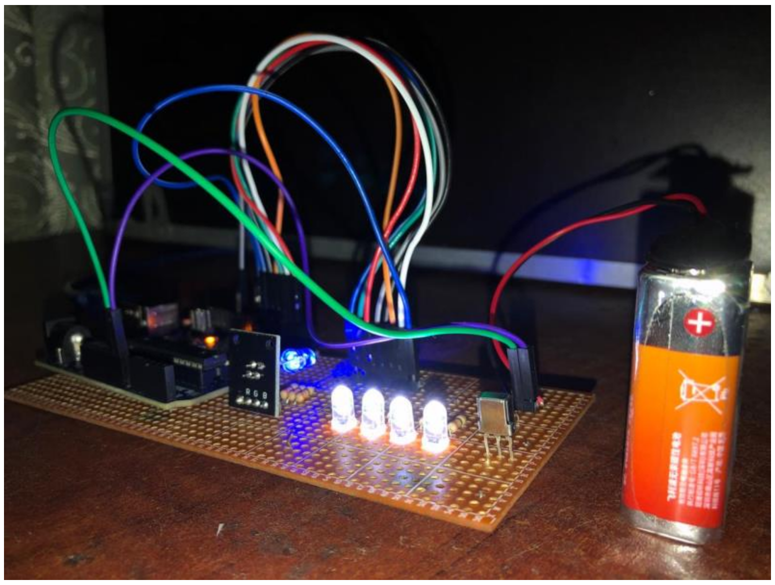


Figure 5.3: LED control sequence aligned with specifications

4. Using the channel-down button to test the brightness adjustment.

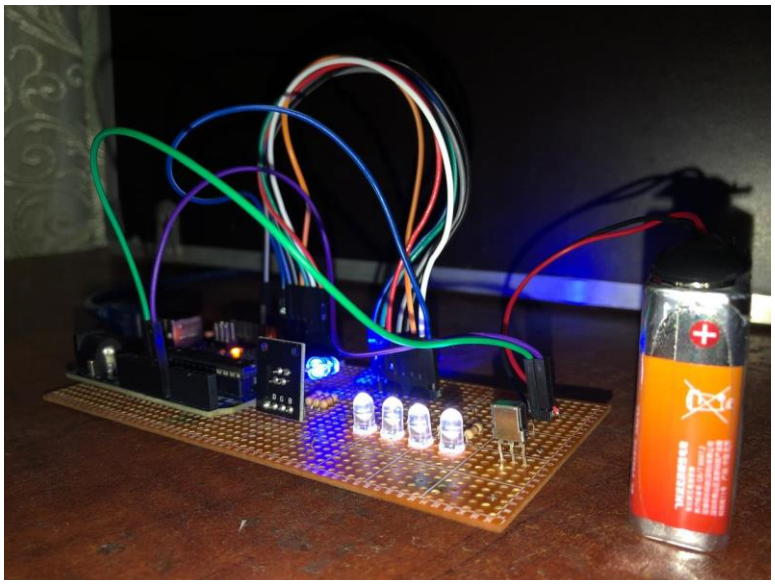


Figure 5.4: Testing brightness adjustment

The LED lamp performed as expected, with the LEDs turning on and off in the proper order and the brightness changing in response to the remote-control input.

6. Discussion

The project successfully created a remote-controlled LED lamp using an Arduino UNO microcontroller and an IR remote. This demonstrated the use of microcontrollers, LED control, and a remote interface design. The lamp followed the specific sequence of operations required for the project. The use of a common IR remote controller adds convenience and user-friendliness to the lamp and showcases the potential for integrating common household technology with do-it-yourself electronics projects. This project emphasises the importance of careful design considerations, code implementation, and testing.

6.1. Future Improvements

1. The code can be modified to accept commands from different IR remotes by learning and storing multiple IR codes.

2. The functionalities can be expanded to include features like blinking patterns or colour mixing for the multicolor LED.

3. An enclosure can be designed and built for the lamp to improve aesthetics and portability.

6.2. Key Challenges

Some key challenges encountered during the project included integrating the IR receiver library and decoding the remote-control signals accurately. Additionally, coordinating the LED control sequence with the remote-control input was another challenge.

Overall, the project was a successful learning experience, allowing the development of practical skills in embedded system design and implementation.

References

[1]
"ARDUINO: IR REMOTE CONTROL OF LEDS," *www.youtube.com*. https://www.youtube.com/watch?v=zLR8EevE5_A&ab_channel=MYTECTUTOR (accessed Apr. 01, 2024).

[2]
Tinkercad, "Tinkercad | From mind to design in minutes," *www.tinkercad.com*, 2023. https://www.tinkercad.com/dashboard

[3]
D. Wood, "Adjust the brightness of an LED with an Arduino," *Arduino Uno tutorials*. https://www.codemahal.com/adjust-the-brightness-of-an-led (accessed Apr. 04, 2024).

[4]
J. Réjaud, "Arduino to AVR-C Reference Guide," *Medium*, May 03, 2016. https://medium.com/@jrejaud/arduino-to-avr-c-reference-guide-7d113b4309f7 (accessed Apr. 07, 2024).

[5]
L. Istrate, "IR remote control library for AVR microcontrollers," Aug. 28, 2022. https://www.programming-electronics-diy.xyz/2022/08/ir-remote-control-library-for-avr.html (accessed Apr. 08, 2024).

[6]
“Learning AVR-C Episode 7: PWM,” *www.youtube.com*.
https://www.youtube.com/watch?v=ZhIRRyhfhLM&ab_channel=humanHardDrive
(accessed Apr. 11, 2024).

[7]
A. Corporation, “ATmega328P 8-bit AVR Microcontroller with 32K Bytes In-System Programmable Flash,” Atmel Corporation, San Jose, California, USA, 2015. Accessed: Apr. 11, 2024. [Online]. Available: https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf

[8]
“AVR_C/Library/analogWrite.c at main · lkoepsel/AVR_C,” *GitHub*.
https://github.com/lkoepsel/AVR_C/blob/main/Library/analogWrite.c (accessed Apr. 12, 2024).