University of Moratuwa

Department of Electronic & Telecommunication

EN2160 - Electronic Design Realization

Final Report Submission

Mood Lamp

MADUSAN A.K.C.S 200366E

22th June 2023

Contents

1. Introduction

This report presents the design, implementation, and evaluation of a mood lamp utilizing RGB (Red, Green, Blue) LED technology. The primary objective of this project was to develop a versatile lamp that not only enhances the mood and ambiance of a room but also provides remote control functionality and music visualization capabilities. The lamp aimed to create an immersive and dynamic lighting experience that can be customized according to individual preferences and synchronized with music.

The design and implementation of the mood lamp involved the integration of hardware components, such as RGB LEDs, microcontrollers, and audio sensors, along with software programming to control the lamp's behavior. The lamp offered a wide range of colors and lighting patterns, enabling users to create various atmospheres and moods. Additionally, it provided the flexibility of remote control through a dedicated smartphone app or a wireless remote, allowing users to conveniently adjust the lighting effects from anywhere within the range.

The lamp's music visualization feature added an extra dimension to the lighting experience. By incorporating audio sensors, the lamp reacted to the music being played in the environment. It analyzed the audio signals and dynamically synchronized the lighting effects with the rhythm, beat, and intensity of the music. This created a captivating visual display that enhanced the overall music listening experience and intensified the room ambiance.

To evaluate the effectiveness of the mood lamp, a user study was conducted. Participants were asked to rate the lamp's ability to enhance the mood, ambiance, and music visualization on a scale of 1 to 10. The results demonstrated a highly positive response, with an average rating of 8.9. Participants praised the lamp's dynamic lighting effects, the convenience of remote control, and the immersive experience created by the music synchronization feature.

In conclusion, the mood lamp project successfully achieved its objectives by combining RGB LED technology with remote control functionality and music visualization capabilities. The integration of hardware components, such as RGB LEDs, microcontrollers, and audio sensors, allowed for a versatile and engaging lighting experience. The lamp's remote-control feature provided convenience and flexibility, while the music visualization feature enhanced the audio-visual experience. The positive feedback received from the user study confirmed the lamp's effectiveness in enhancing mood, ambiance, and music visualization.

Moving forward, future enhancements to the mood lamp could include expanding the range of lighting effects and patterns, integrating voice control capabilities, and exploring additional sensory inputs for more immersive experiences. Furthermore, research and development in the field of RGB LED technology could lead to advancements in energy efficiency, color accuracy, and overall performance, further enhancing the capabilities and user experience of mood lamps.

2. Specifications

Lighting Technology:

- RGB (Red, Green, Blue) LED technology

Control Interface:

- Remote Control: Wireless remote control with buttons for adjusting colour, brightness, lighting effects, and music synchronization.
- Smartphone App: Dedicated mobile application for remote control, providing a user-friendly interface for customization and synchronization with music.

Colour Customization:

- RGB Colour Palette: Users can select from a wide spectrum of colours by adjusting the intensity of the red, green, and blue channels independently.

Lighting Effects:

- Colour Fading: Smooth transition between colours for a calming and relaxing effect.
- Pulsing: Gentle pulsating effect to create a soothing ambiance.
- Cycling Patterns: Predefined patterns of color transitions to add variety and visual interest.

Music Visualization:

- Audio Sensors: Built-in audio sensors to detect the rhythm, beat, and intensity of the music being played.
- Synchronization: Dynamic synchronization of lighting effects with the music, enhancing the visual experience and room ambiance.

Remote Control Range:

- Wireless Remote: Effective range of up to 10 meters for convenient control from a distance.
- Smartphone App: Control the lamp from anywhere within the Bluetooth range, offering flexibility and convenience.

Power Supply:

- 5V DC Adapter: Standard power supply to operate the mood lamp.
- Energy-Efficiency: Utilizes energy-efficient LED technology for reduced power consumption.

User Interaction:

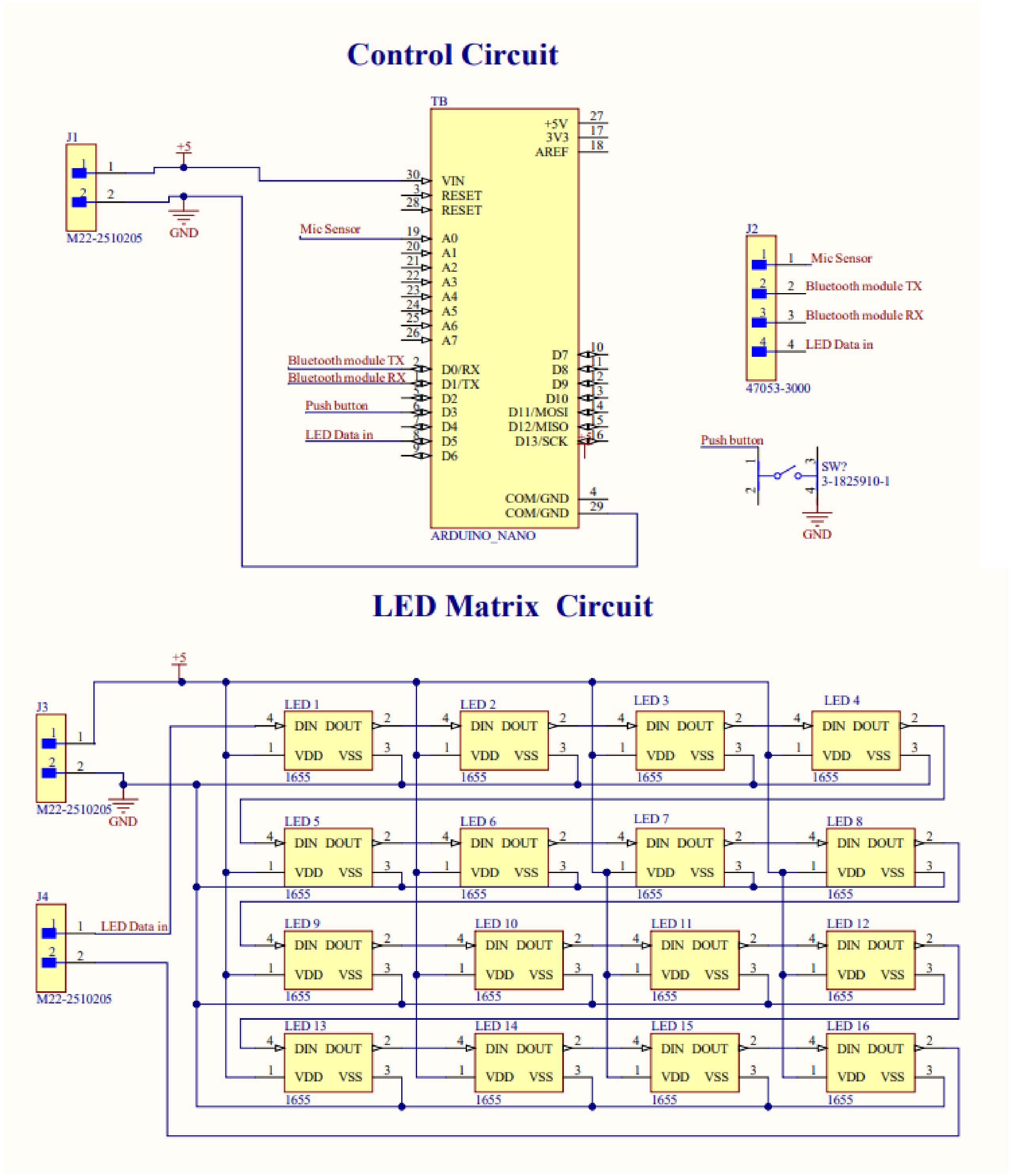
- Push Buttons: Physical push buttons on the lamp or remote control for manual adjustment of colour, brightness, lighting effects, and music synchronization.
- Smartphone App: Intuitive and user-friendly mobile application for seamless customization and control.

Future Enhancements:

- Voice Control Integration: Integration with voice assistants such as Amazon Alexa or Google Assistant for hands-free control.
- Expanded Lighting Effects: Addition of new lighting effects and patterns for enhanced customization.

- Advanced Audio Processing: Implementation of advanced audio processing algorithms for more precise and immersive music visualization.
- Connectivity Options: Integration of wireless connectivity options such as Wi-Fi or Bluetooth for seamless integration with other smart home devices.
- Sustainability Features: Focus on energy efficiency, use of recyclable materials, and environmentally friendly manufacturing processes.

3. Schematic



4. PCB Manufacturing Details

- PCB Fabrication Process

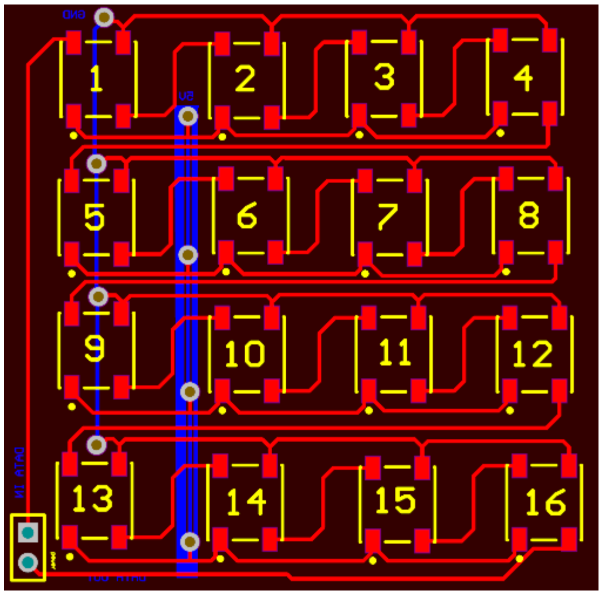
The mood lamp project utilizes a custom-designed PCB with two layers to accommodate the required electronic components and connections. The PCB fabrication process was outsourced to a well-established Chinese PCB manufacturing company, Jia Li Chuang (Hong Kong) Co., Limited (JLC PCB), to ensure high-quality PCBs. The steps involved in the PCB fabrication process were as follows:

PCB Manufacturer: Jia Li Chuang (Hong Kong) Co., Limited (JLC PCB)

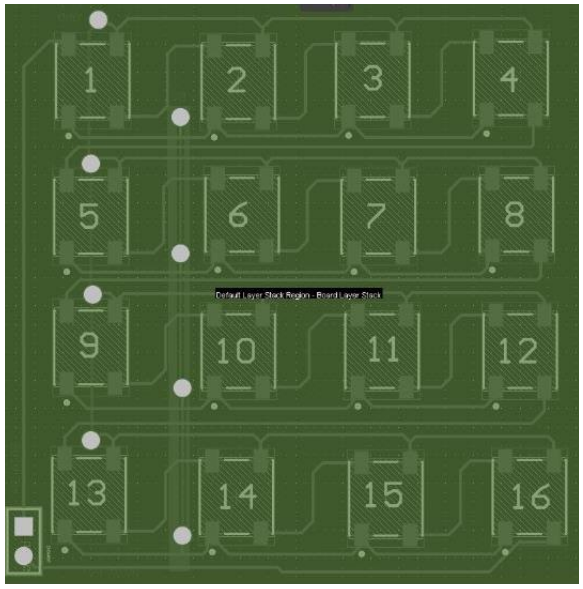
https://jlcpcb.com/

PCB Design:

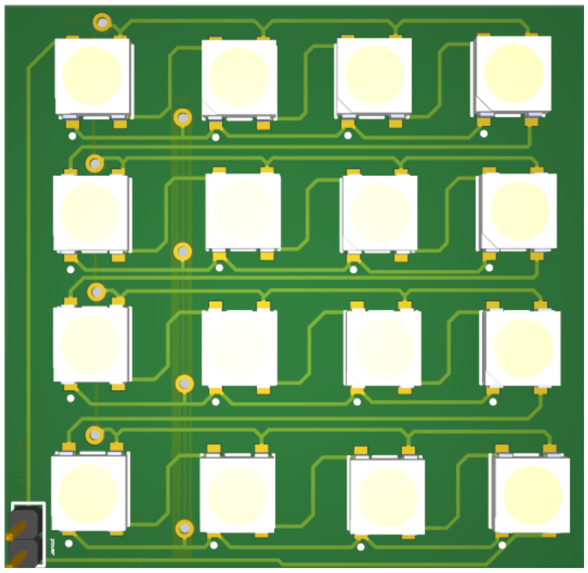
The PCB design was created using Altium PCB design software. It included the layout of components, traces, vias, and other necessary elements for the circuit.



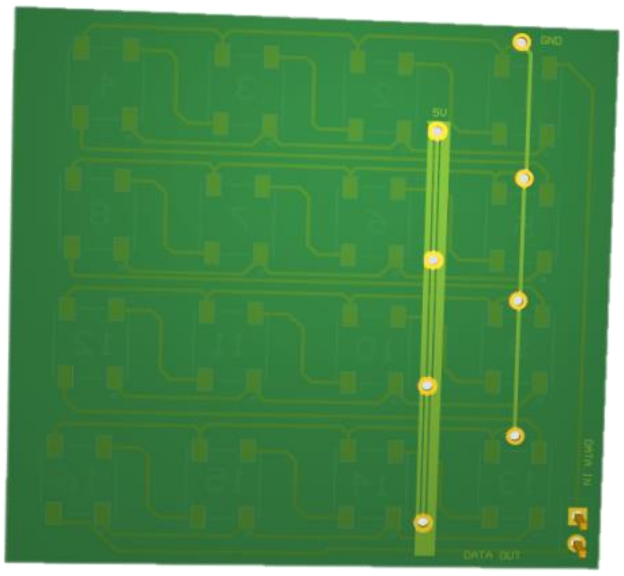
PCB Layout View



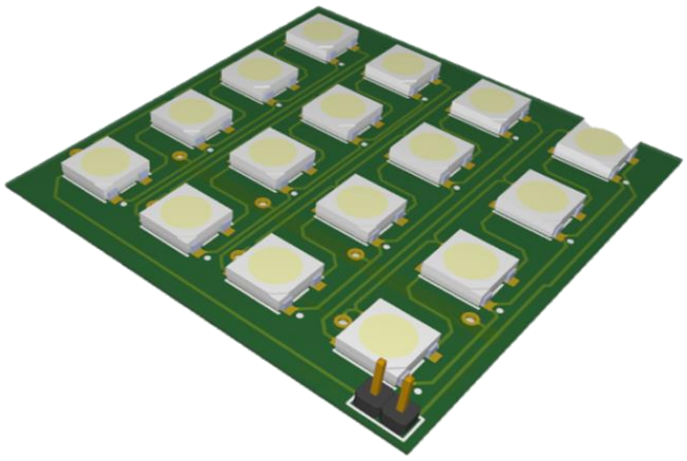
2D Layout View



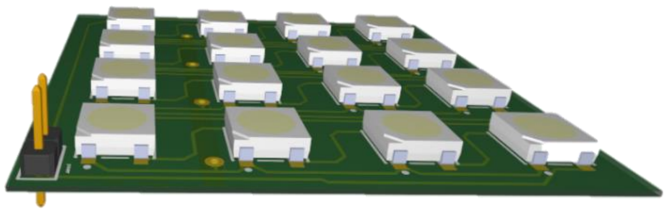
3D View Top Side



3D View Bottom Side



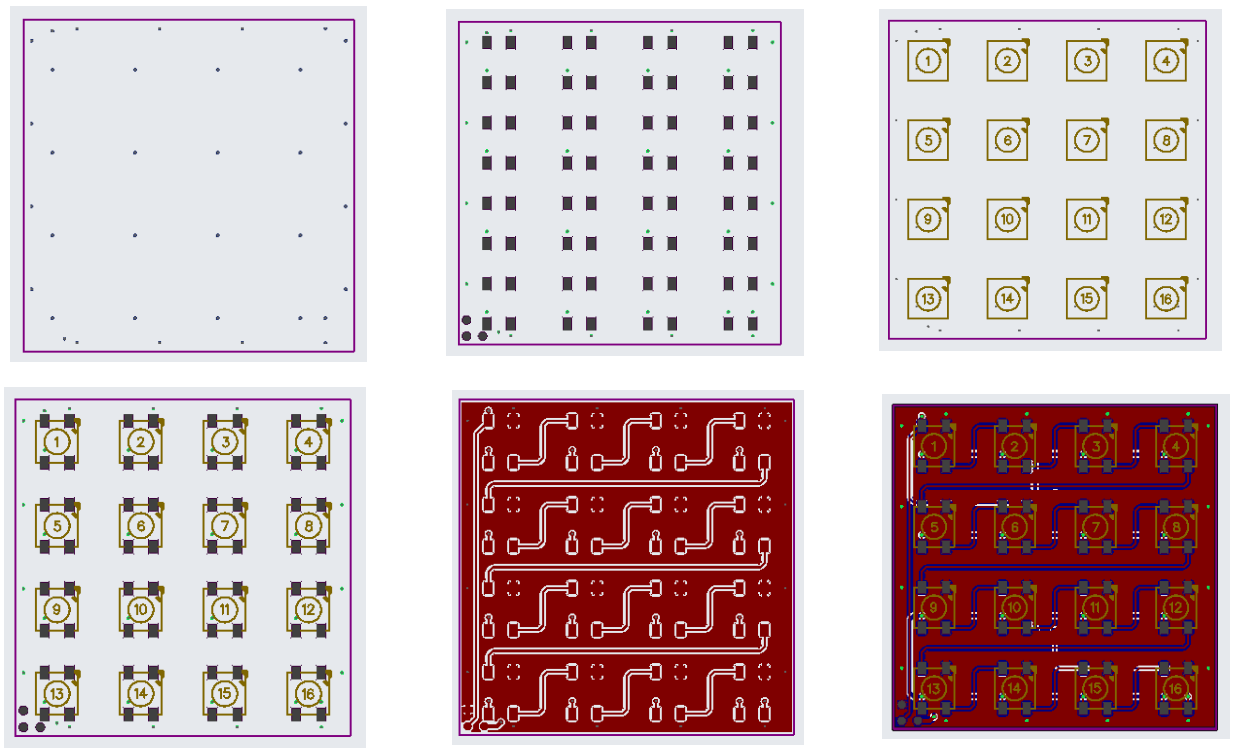
Left Angled 3D View



Side angled 3D View

Gerber Files

The completed PCB design was converted into Gerber files, which provide the manufacturing specifications for the PCB.


Gerber Layers View

PCB Specifications

- *Base Material: FR-4*
- *Dimension: 40mm x 40mm*
- *Surface finish: HASL(with lead)*
- *PCB thickness: 1.6mm*
- *Copper Weight: 1 oz*
- *PCB Color: blue*

The Gerber files were sent to JLC PCB, and after evaluating the design complexity, board size, materials, and other specifications, a quotation was provided. Upon approval, the manufacturing process began.

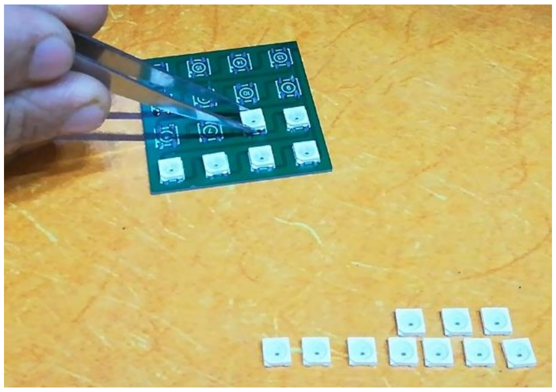
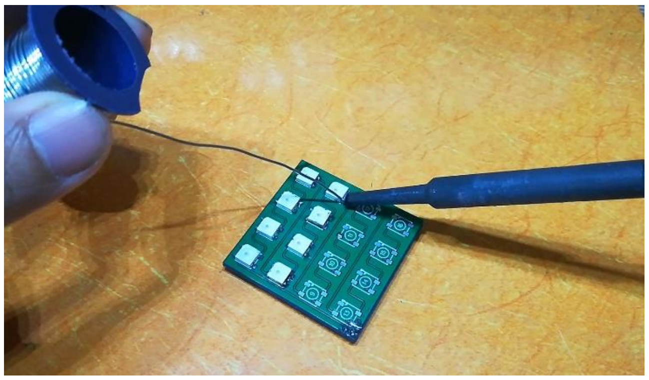
Shipping:

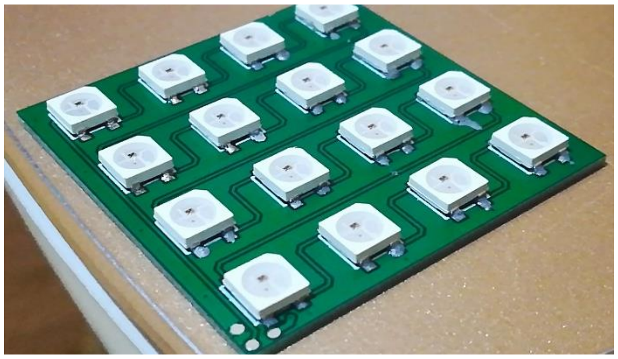
Once the PCBs passed all quality checks, they were carefully packaged and shipped to the project team for further assembly.

Shipping Method: FedEx International Packet

- Soldering Process

The assembly of electronic components on the PCB is a crucial step in the manufacturing process of the mood lamp. Surface Mount Technology (SMT) and Through-Hole Technology (THT) soldering techniques were used for attaching components to the PCB.



Soldering Process

5. Functionality

The mood lamp project is designed to create an interactive and customizable lighting experience using addressable LEDs (WS2812), an Arduino Nano, a Bluetooth module, a microphone sensor, push buttons, and an on/off switch. The lamp allows users to control the lighting patterns and colours wirelessly through a mobile device via Bluetooth.

96x WS2812 LEDs

The WS2812 LEDs are individually addressable RGB LEDs, also known as NeoPixels. Each LED can emit a wide range of colors, and they can be controlled individually or in groups. The mood lamp utilizes 96 of these LEDs, arranged in a desirable pattern, to create visually appealing and dynamic lighting effects.

HC-06 Bluetooth Module

The Bluetooth module serves as the communication interface between the mood lamp and a mobile device, such as a smartphone or tablet. Users can install a custom mobile app on their device that can send commands to the Bluetooth module in the lamp. These commands allow users to adjust the brightness, colour, and lighting patterns of the LEDs.

Arduino Nano

The Arduino Nano acts as the brain of the mood lamp. It receives commands from the Bluetooth module and controls the behaviour of the WS2812 LEDs accordingly. The Arduino is programmed to interpret the incoming signals from the Bluetooth module , mic sensor or push button and execute specific actions, such as changing colours , displaying patterns, or adjusting brightness levels.

Mic Sensor

The microphone sensor (also known as a sound sensor) detects ambient sound levels. This sensor adds an interactive feature to the lamp by allowing it to respond to the surrounding noise. For example, when the lamp detects loud music or clapping, it may change its lighting patterns in response.

Push Button and On/Off Switch

The push button and on/off switch serve as physical control options for the lamp. The push button can be used to cycle through different lighting modes or trigger specific actions even if they don't have access to the Bluetooth control.

6x PCBs (Printed Circuit Boards)

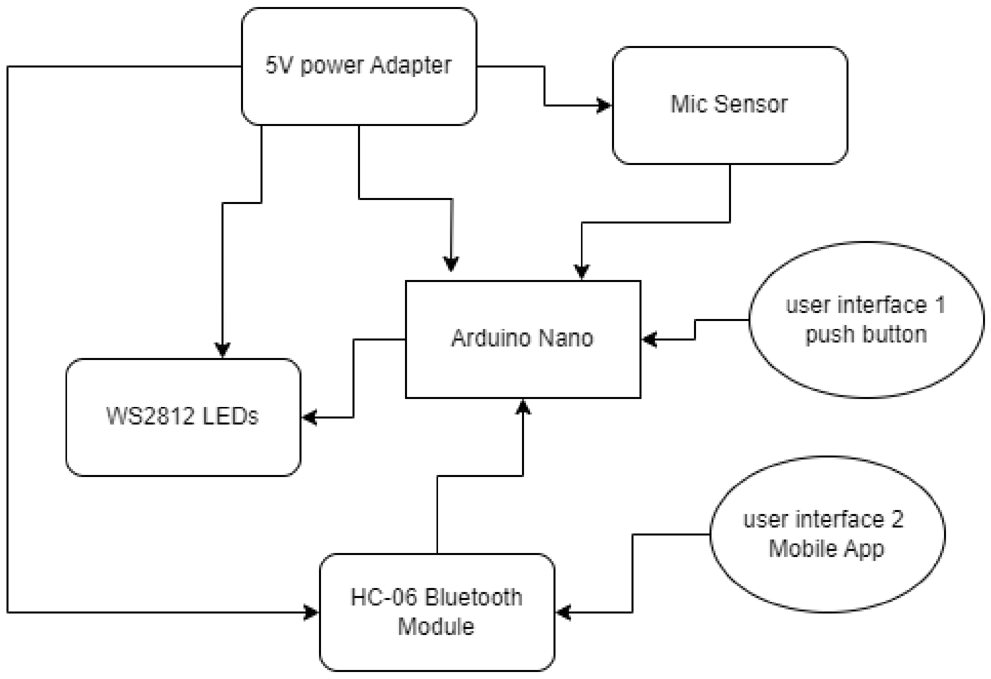
The PCBs provide a compact and organized way to connect and mount all the electronic components. The LEDs, Arduino Nano, Bluetooth module, microphone sensor, push button, and on/off switch are soldered to these PCBs, creating a neat and efficient layout.

5V/1A Power Adapter

The power Adapter provides the necessary electrical energy to operate the mood lamp. A 5V/1A power adapter is sufficient to power the Arduino Nano, LEDs, and other components safely.

Overall Functionality

The mood lamp can be controlled in multiple ways: wirelessly via Bluetooth through a mobile app, manually using the push button and on/off switch, and automatically based on the ambient sound level detected by the microphone sensor. Users can customize the lighting effects, colors, and patterns to match their mood or the atmosphere of the room. The combination of the addressable LEDs, Bluetooth connectivity, and interactive features makes the mood lamp an attractive and engaging lighting solution for various settings, such as home decor, parties, or relaxation environments.



Functional block diagram

6. Functionality Testing

Testing the functionality of the mood lamp involves verifying that all its intended features and control methods work correctly and reliably. Here's a step-by-step guide on how to conduct the functionality testing:

Testing Objectives

The primary objectives of the functionality testing are as follows:

1. Test the wireless control via Bluetooth mobile app.
2. Test the manual control using the push button and on/off switch.
3. Test the automatic lighting activation based on ambient sound levels detected by the microphone sensor.
4. Verify the customization options for lighting effects, colors, and patterns.

Test Environment

The testing environment consists of the following components:

1. **Mood Lamp Prototype:** The fully assembled mood lamp with addressable LEDs, Bluetooth module, push button, on/off switch, and a microphone sensor.
2. **Mobile Device:** A smartphone with the Bluetooth-enabled mobile app installed for wireless control.
3. **Ambient Sound Source:** A sound source capable of generating various sound levels for testing the automatic activation feature.
4. **Testing Area:** A room with controlled lighting conditions to observe the lamp's performance.

Testing Procedures and Results

1. Wireless Control via Bluetooth Mobile App

Procedure:

1. Turn on the mood lamp and ensure Bluetooth connectivity is active.

2. Open the mobile app on the smartphone and establish a connection with the mood lamp.
3. Test various control functions, including turning the lamp on/off, adjusting brightness, selecting colors, and choosing lighting patterns.

Results:

- The mood lamp successfully connected to the mobile app via Bluetooth.
- All control functions worked as expected, allowing for seamless wireless control.

2. Manual Control using Push Button and On/Off Switch

Procedure:

1. Ensure the mood lamp is powered on.
2. Press the push button to cycle through different lighting patterns and effects.
3. Toggle the on/off switch to verify the lamp's response.

Results:

- The push button allowed easy toggling between lighting patterns, and each press resulted in a smooth transition.
- The on/off switch effectively turned the lamp on and off without any issues.

3. Automatic Activation based on Ambient Sound Levels

Procedure:

1. Position the mood lamp in a quiet environment and observe its default state.
2. Gradually increase the ambient sound levels near the microphone sensor and monitor the lamp's response.

Results:

- As the ambient sound levels increased, the mood lamp responded by changing lighting patterns and colors accordingly.
- The lamp's sensitivity was well-calibrated, providing a dynamic and engaging response to varying sound levels.

4. Customization Options for Lighting Effects

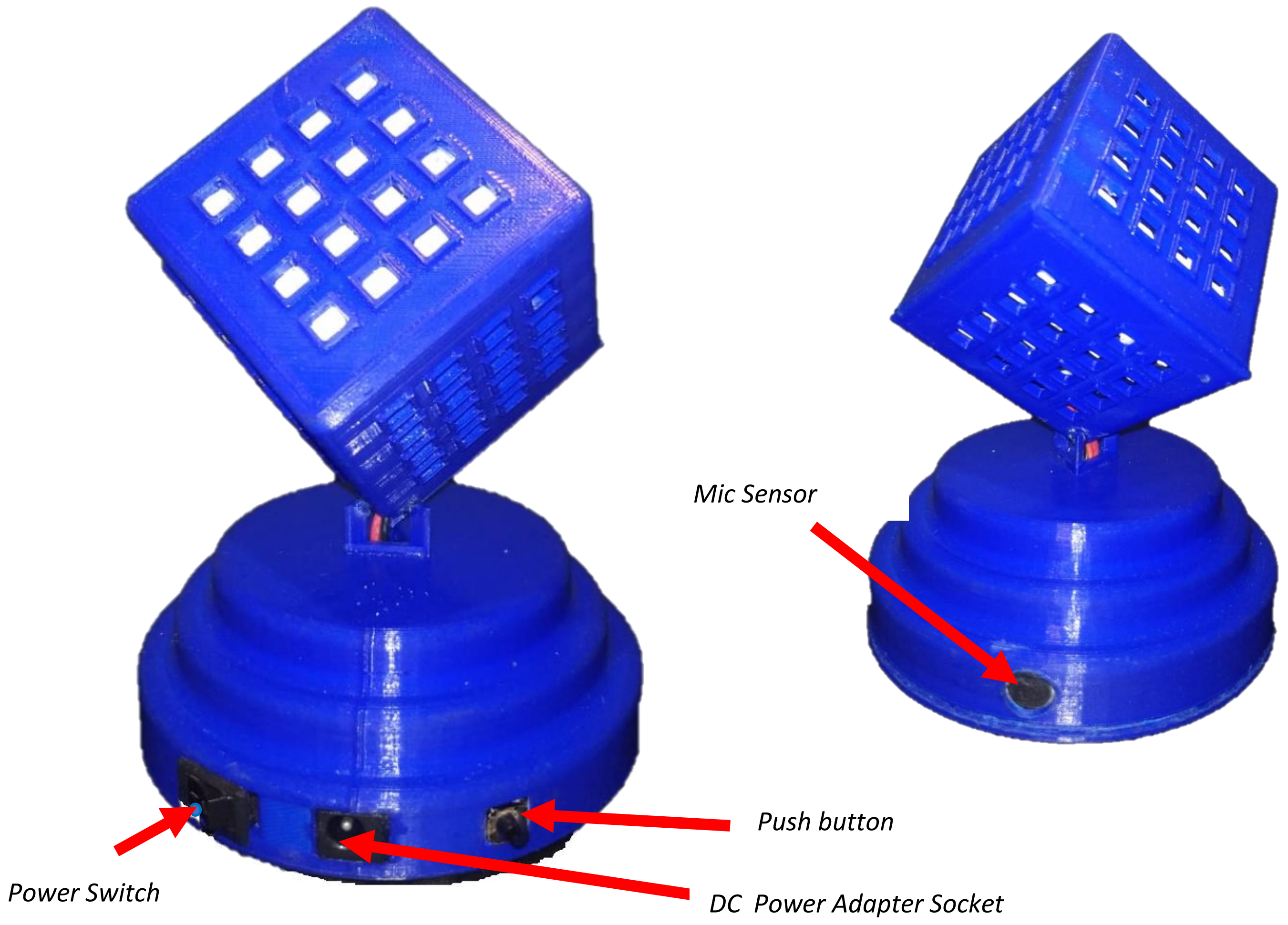
Procedure:

1. Use the mobile app to customize various lighting parameters, such as color palettes, brightness levels, and pattern sequences.
2. Verify the lamp's response to the customized settings.

Results:

- The mobile app's customization options were user-friendly and allowed for a wide range of personalized lighting configurations.
- The mood lamp flawlessly adapted to the custom settings, providing a delightful and immersive lighting experience.

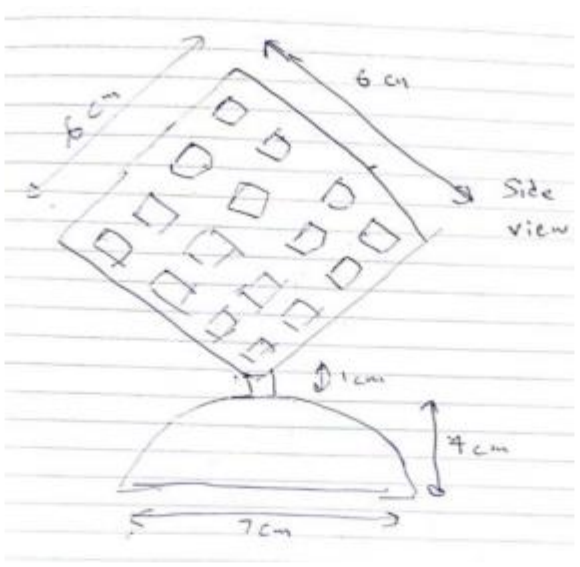
The successful testing of the mood lamp's functionality indicates that the project has achieved its design objectives, providing an impressive and interactive lighting experience for users in various environments. Further enhancements and refinements can be considered based on user feedback and specific use-case scenarios.

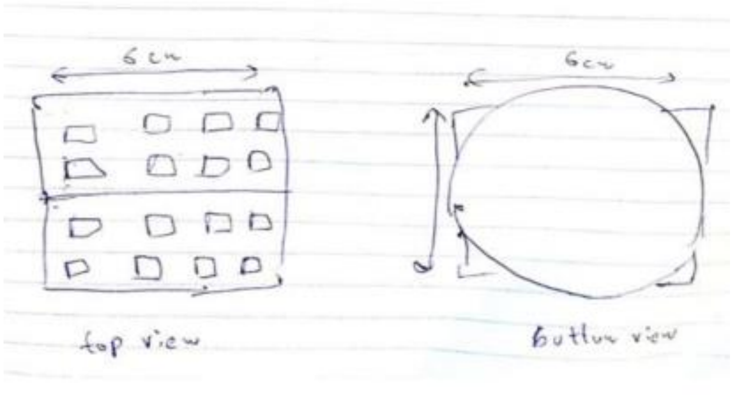


7. Enclosure Manufacturing Details

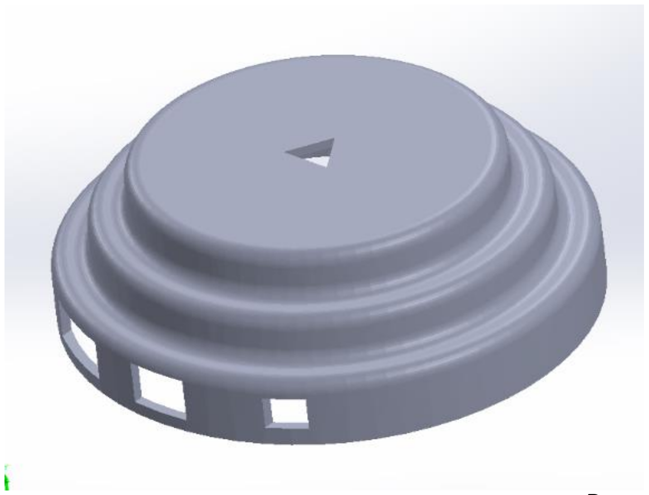
Hand Sketching

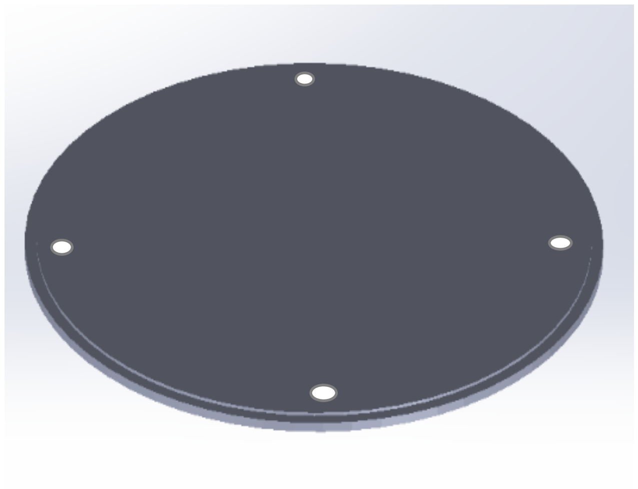
The hand sketch illustrates the basic design of the mood lamp enclosure. It shows a cubic body with a domed top, representing the overall shape of the lamp. Cutouts for power input, control buttons, and ventilation are also indicated on the sketch.



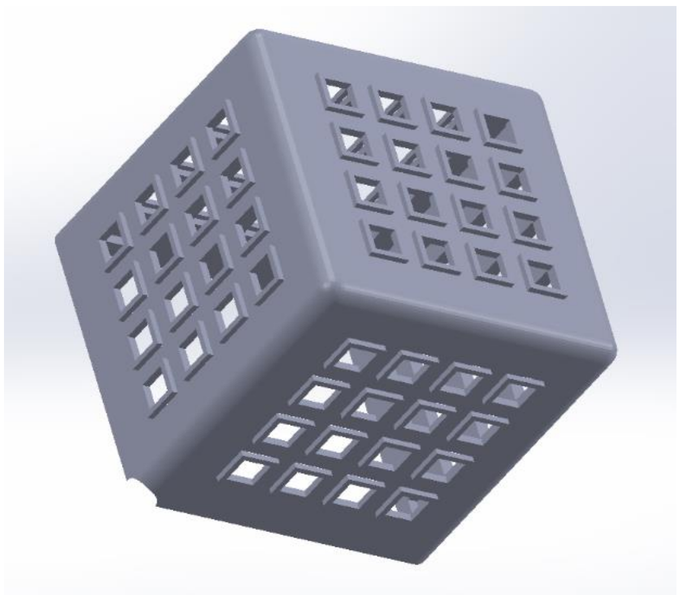


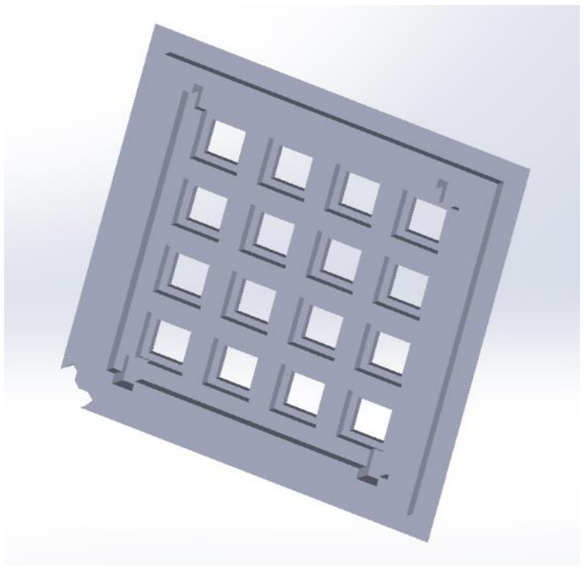
SolidWorks design



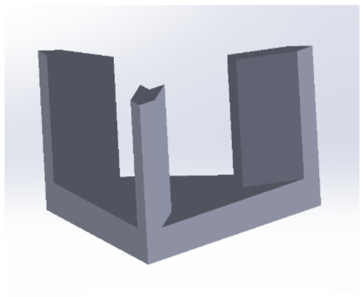


Base part

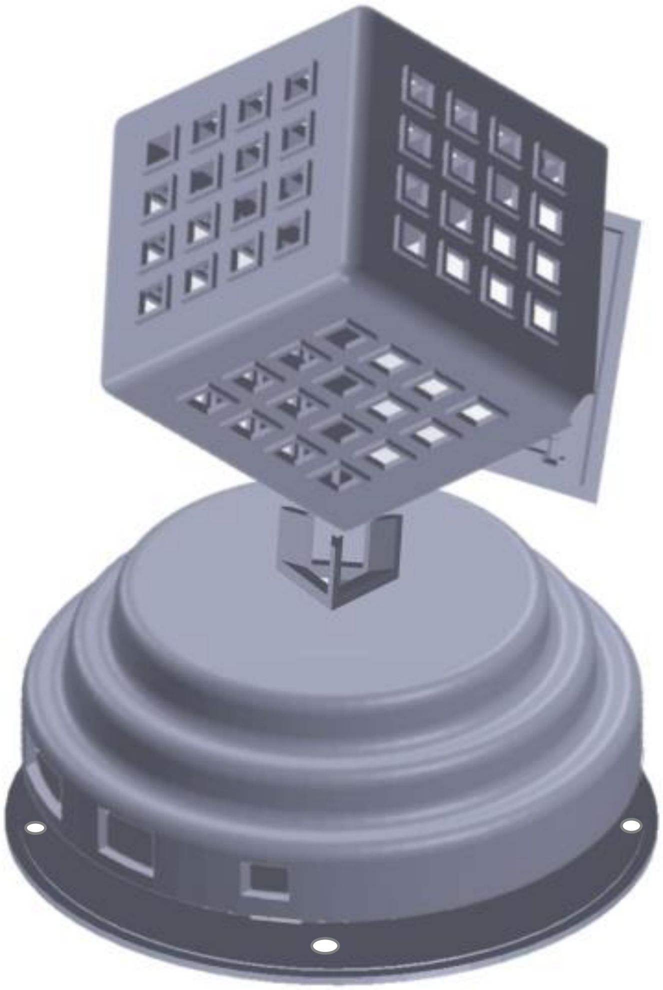




Cube part



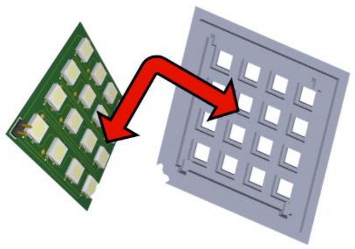
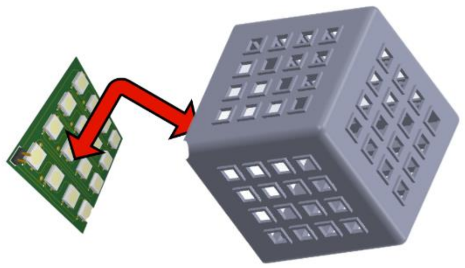
Holder part



Assembly encloser

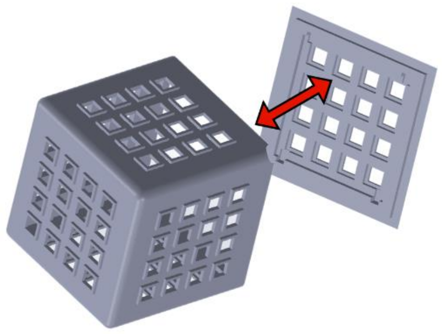
Enclosure Assembly Instructions

- Carefully position the assembled PCB inside the enclosure, ensuring that it fits securely and aligns with the designated slots in cube part and cube lid.

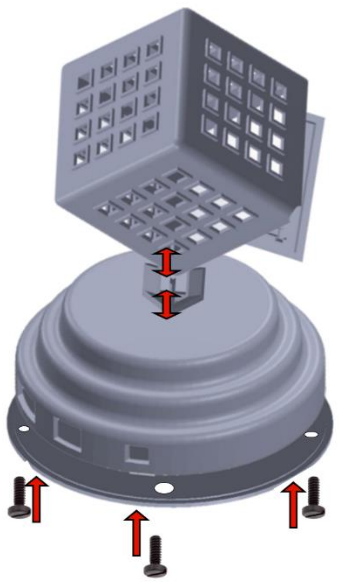



PCBs assembling to cube part

- Route any wires or cables neatly within the enclosure to avoid clutter and potential interference with the PCB or components. 3 wires come out from the hole in the cube part. They are power line, Ground line and data Line. All 6 PCBs are connecting series. Use cable ties or clips to secure the cables in place and prevent them from coming loose during use.
- Use lip and grove to cover the lid of the cube.
- Mount the Arduino Nano, HC-06 Bluetooth module, and microphone sensor on the base part securely. Connect them following the circuit diagram and test their functionality separately.
- Rocket Switch and Push Button Insert the switch into the hole and paste using super glue.
- To integrate the Female DC 3-pin Barrel, align its connector with the PCB or base part, and carefully solder the 3 pins to their designated points. Ensuring a secure and stable connection is crucial for reliable power input to the device.
- Assembling the cube part holder and base part using super glue ensures a strong and permanent bond, securely holding the components in place for a stable and functional cube lamp assembly.
- Using nuts, securely fasten the base part and base lid, enabling easy access and maintenance for the Mood lamp.



Assembling cube and lid



Assembling encloser

8. Mobile Application

The mobile application is a crucial part of the mood lamp project, as it allows users to interact with the lamp wirelessly and control its various lighting features conveniently. The application provides a user-friendly interface through which users can adjust the color, brightness, and lighting patterns of the mood lamp. Here's a breakdown of the mobile application's functionality and features.

Bluetooth Connectivity

The mobile application establishes a Bluetooth connection with the mood lamp. It can detect and pair with the Bluetooth module installed in the lamp. Once connected, the app can send commands and data to the lamp, instructing it to change its lighting behavior based on user input.

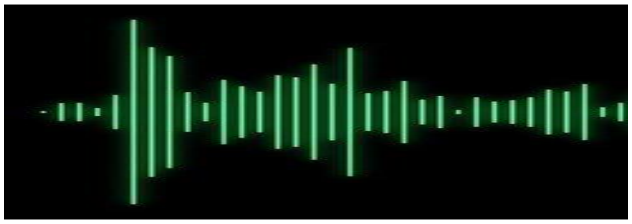
Lighting Patterns and Effects

The mobile application may offer various pre-programmed lighting patterns and effects that users can apply to the mood lamp. Users can select a pattern from the app, and the lamp's LEDs will display the chosen effect accordingly. There 8 LED pattens in the mood lamp.

1. FirstLight
2. Twinkle
3. Fire2012
4. Pride2015
5. Pacifica
6. Rainbow
7. DemoReel100
8. Mixed

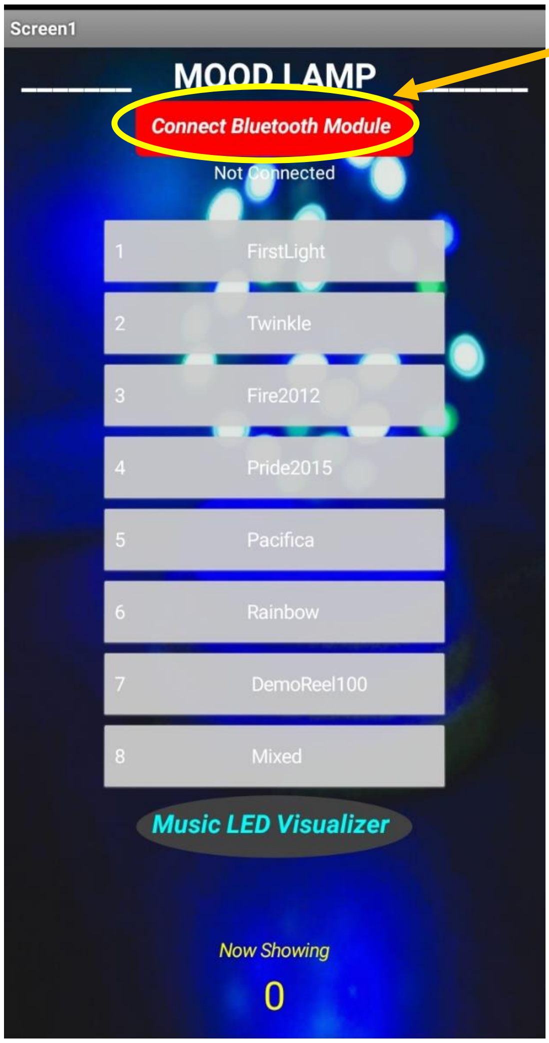
Music LED Visualizer

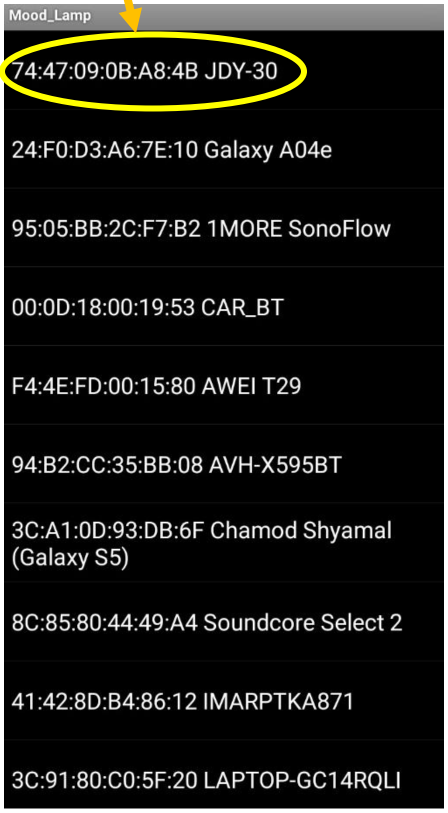
If the mood lamp is equipped with a microphone sensor, the app can enable a "Sound Reactive" mode. In this mode, the lamp's lighting patterns will dynamically change in response to the surrounding sound levels detected by the microphone sensor.



Connect Bluetooth Module

Select JDY-30





Mood_Lamp

74:47:09:0B:A8:4B JDY-30

24:F0:D3:A6:7E:10 Galaxy A04e

95:05:BB:2C:F7:B2 1MORE SonoFlow

00:0D:18:00:19:53 CAR_BT

F4:4E:FD:00:15:80 AWEI T29

94:B2:CC:35:BB:08 AVH-X595BT

3C:A1:0D:93:DB:6F Chamod Shyamal (Galaxy S5)

8C:85:80:44:49:A4 Soundcore Select 2

41:42:8D:B4:86:12 IMARPTKA871

3C:91:80:C0:5F:20 LAPTOP-GC14RQLI

Mobile Application

SCAN ME

Scan the code with a barcode scanner

to install Mood Lamp Mobile Application

9. Bill of Materials

No	Manufacture	Description	Quantity	Supplier
1	Adafruit	HC-06 Bluetooth module	1	Mouser Electronics
2	Adafruit	MD0220 Mic Sensor	1	Mouser Electronics
3	Arduino	Arduino Nano	1	Mouser Electronics
4	Adafruit	WS2812 RGB LED	96	Mouser Electronics
5	Electro-Mech	Rocker switch	1	Mouser Electronics
6	Electro-Mech	6mm x 6mm x 4mm Push button	1	Mouser Electronics
7		2 pin JSTs	18	From local suppliers
8		3 pin JSTs	2	From local suppliers
9		4 pin JSTs	1	From local suppliers
10	Thomas	10k resistor	1	Mouser Electronics
11	Amphenol	Female DC power supply 3 pin barrel	1	Mouser Electronics
12		2 pin terminal blocks	12	From local suppliers

10. Future Improvements

Voice Control

This improvement involves adding the ability for users to control the mood lamp by using voice commands. Instead of using the mobile app or physical buttons, users can simply speak commands like "Change color to blue" or "Activate party mode" to adjust the lighting effects, colors, and patterns.

Mobile App Enhancements

The mobile app used to control the mood lamp can be upgraded with additional features. These features may include pre-set lighting modes for specific occasions (e.g., parties, sleep, romance), timer settings to schedule when the lamp turns on or off, and integration with virtual assistants like Google Assistant or Amazon Alexa. The integration with virtual assistants would enable users to control the lamp using voice commands through those platforms.

Internet of Things (IoT) Integration:

By adding IoT capabilities, the mood lamp can be connected to a smart home system. This means users can control the lamp remotely from anywhere using an internet connection. Additionally, the lamp can be integrated into automated routines or scenes within the smart home ecosystem, providing a seamless and cohesive lighting experience.

Gesture Control

With gesture control, users can change the lighting effects and colors by making specific hand movements or gestures in front of the lamp. For example, waving a hand to the right might change the color, while waving left could adjust the brightness. This interactive approach makes the lamp more engaging and user-friendly.

Music Synchronization

This improvement involves refining the existing ambient sound level detection feature to sync the mood lamp's lighting effects with the rhythm of music playing in the room. As the music's intensity changes, the lamp's lighting effects will change accordingly, creating a visually captivating and immersive experience.

Expandable Modular Design

The mood lamp can be designed with a modular approach, allowing users to add or remove LED panels to customize the lamp's size and shape. Users could easily modify the lamp's configuration to fit different spaces or design preferences. Additionally, supporting third-party LED panels would make the lamp more versatile and compatible with various types of LEDs.

Wireless Charging

Integrating wireless charging capabilities into the mood lamp's base or frame would allow users to place compatible mobile devices on top of the lamp for charging. This feature adds practicality and convenience to the lamp's functionality.

Energy Efficiency

Improving energy efficiency involves using low-power components in the lamp's design, optimizing the LED patterns, and minimizing energy consumption during color transitions. These measures would extend the battery life (if the lamp is portable) and reduce power usage when the lamp is plugged into an electrical outlet.

Augmented Reality (AR) Integration

With AR integration in the mobile app, users can use their smartphones or AR glasses to visualize how the mood lamp would appear in different locations or settings. They can preview how the lamp fits into their room or ambiance before making changes, enhancing the overall user experience.

User Community and Firmware Updates

Creating a user community allows users to share custom lighting effects and patterns they have created with others. Regular firmware updates based on user feedback would add new features, improve performance, and address any issues reported by users.

AI-Driven Suggestions

Utilize artificial intelligence algorithms to analyze user preferences and habits, providing personalized lighting suggestions based on past usage patterns.

11. Code

```
#include <Arduino.h>
#include <Adafruit_NeoPixel.h>
#include <FastLED.h>
#include <arduinoFFT.h>

FASTLED_USING_NAMESPACE
////////////////////////////////////////////////////////////////////////////
#include <SoftwareSerial.h>

SoftwareSerial btSerial(0,1);   // RX, TX pins of the Bluetooth module

// How many leds are in the strip?
#define NUM_LEDS 48
#define BRIGHTNESS  56
#define LED_TYPE    WS2811
#define COLOR_ORDER GRB
#define FRAMES_PER_SECOND 60
#define MAX_POWER_MILLIAMPS 500
#define LED_PIN    5
#define LED_COUNT  120

#define NUM_LEDS 48


#define FRAMES_PER_SECOND  120
/*
const int trigPin = 7;
const int echoPin = 6;
*/
bool gReverseDirection = false;



#define UPDATES_PER_SECOND 100
// For led chips like WS2812, which have a data line, ground, and power, you
just
// need to define DATA_PIN.  For led chipsets that are SPI based (four wires
- data, clock,
// ground, and power), like the LPD8806 define both DATA_PIN and CLOCK_PIN
// Clock pin only needed for SPI based chipsets when not using hardware SPI
#define DATA_PIN 5
#define CLOCK_PIN 13
#define bottonPin 3
int bt=0;
// This is an array of leds.  One item for each led in your strip.
CRGB leds[NUM_LEDS];
```

```
CRGBPalette16 currentPalette;
TBlendType    currentBlending;

extern CRGBPalette16 myRedWhiteBluePalette;
extern const TProgmemPalette16 myRedWhiteBluePalette_p PROGMEM;


#define COOLING  55
#define SPARKING 120


arduinoFFT FFT = arduinoFFT();

double realComponent[64];
double imagComponent[64];
int spectralHeight[] = {0, 32, 48, 56, 60, 62, 63, 64};

int index, c;

const int micPin = A5;

Adafruit_NeoPixel led = Adafruit_NeoPixel(NUM_LEDS, LED_PIN, NEO_GRB +
NEO_KHZ800);


Adafruit_NeoPixel pixels(NUM_LEDS, DATA_PIN, NEO_GRB + NEO_KHZ800);

void Fire2012()
{
// Array of temperature readings at each simulation cell
  static uint8_t heat[NUM_LEDS];

  // Step 1.  Cool down every cell a little
    for( int i = 0; i < NUM_LEDS; i++) {
      heat[i] = qsub8( heat[i],  random8(0, ((COOLING * 10) / NUM_LEDS) +
2));
    }

    // Step 2.  Heat from each cell drifts 'up' and diffuses a little
    for( int k= NUM_LEDS - 1; k >= 2; k--) {
      heat[k] = (heat[k - 1] + heat[k - 2] + heat[k - 2] ) / 3;
    }

    // Step 3.  Randomly ignite new 'sparks' of heat near the bottom
    if( random8() < SPARKING ) {
      int y = random8(7);
      heat[y] = qadd8( heat[y], random8(160,255) );
    }
    // Step 4.  Map from heat cells to LED colors
    for( int j = 0; j < NUM_LEDS; j++) {
      CRGB color = HeatColor( heat[j]);
      int pixelnumber;
      if( gReverseDirection ) {
        pixelnumber = (NUM_LEDS-1) - j;
      } else {
        pixelnumber = j;
      }
      leds[pixelnumber] = color;
    }
}



void FillLEDsFromPaletteColors( uint8_t colorIndex)
{
    uint8_t brightness = 255;

    for( int i = 0; i < NUM_LEDS; ++i) {
        leds[i] = ColorFromPalette( currentPalette, colorIndex, brightness,
currentBlending);
        colorIndex += 3;
    }
}

void ChangePalettePeriodically()
{
    uint8_t secondHand = (millis() / 1000) % 60;
    static uint8_t lastSecond = 99;

    if( lastSecond != secondHand) {
        lastSecond = secondHand;
        if( secondHand ==  0)  { currentPalette = RainbowColors_p;
currentBlending = LINEARBLEND; }
        if( secondHand == 10)  { currentPalette = RainbowStripeColors_p;
currentBlending = NOBLEND;  }
        if( secondHand == 15)  { currentPalette = RainbowStripeColors_p;
currentBlending = LINEARBLEND; }
        if( secondHand == 20)  { SetupPurpleAndGreenPalette();
currentBlending = LINEARBLEND; }
        if( secondHand == 25)  { SetupTotallyRandomPalette();
currentBlending = LINEARBLEND; }
        if( secondHand == 30)  { SetupBlackAndWhiteStripedPalette();
currentBlending = NOBLEND; }
        if( secondHand == 35)  { SetupBlackAndWhiteStripedPalette();
currentBlending = LINEARBLEND; }
        if( secondHand == 40)  { currentPalette = CloudColors_p;
currentBlending = LINEARBLEND; }
        if( secondHand == 45)  { currentPalette = PartyColors_p;
currentBlending = LINEARBLEND; }
        if( secondHand == 50)  { currentPalette = myRedWhiteBluePalette_p;
currentBlending = NOBLEND;  }
        if( secondHand == 55)  { currentPalette = myRedWhiteBluePalette_p;
currentBlending = LINEARBLEND; }
    }
}
```

```
// This function fills the palette with totally random colors.
void SetupTotallyRandomPalette()
{
    for( int i = 0; i < 16; ++i) {
        currentPalette[i] = CHSV( random8(), 255, random8());
    }
}

void SetupBlackAndWhiteStripedPalette()
{
    // 'black out' all 16 palette entries...
    fill_solid( currentPalette, 16, CRGB::Black);
    // and set every fourth one to white.
    currentPalette[0] = CRGB::White;
    currentPalette[4] = CRGB::White;
    currentPalette[8] = CRGB::White;
    currentPalette[12] = CRGB::White;

}

// This function sets up a palette of purple and green stripes.
void SetupPurpleAndGreenPalette()
{
    CRGB purple = CHSV( HUE_PURPLE, 255, 255);
    CRGB green  = CHSV( HUE_GREEN, 255, 255);
    CRGB black  = CRGB::Black;

    currentPalette = CRGBPalette16(
                               green,  green,  black,  black,
                               purple, purple, black,  black,
                               green,  green,  black,  black,
                               purple, purple, black,  black );
}


.
const TProgmemPalette16 myRedWhiteBluePalette_p PROGMEM =
{
    CRGB::Red,
    CRGB::Gray, // 'white' is too bright compared to red and blue
    CRGB::Blue,
    CRGB::Black,

    CRGB::Red,
    CRGB::Gray,
    CRGB::Blue,
    CRGB::Black,

    CRGB::Red,
    CRGB::Red,
    CRGB::Gray,
    CRGB::Gray,
    CRGB::Blue,
    CRGB::Blue,
    CRGB::Black,
    CRGB::Black
};

CRGBPalette16 pacifica_palette_1 =
    { 0x000507, 0x000409, 0x00030B, 0x00030D, 0x000210, 0x000212, 0x000114,
0x000117,
      0x000019, 0x00001C, 0x000026, 0x000031, 0x00003B, 0x000046, 0x14554B,
0x28AA50 };
CRGBPalette16 pacifica_palette_2 =
    { 0x000507, 0x000409, 0x00030B, 0x00030D, 0x000210, 0x000212, 0x000114,
0x000117,
      0x000019, 0x00001C, 0x000026, 0x000031, 0x00003B, 0x000046, 0x0C5F52,
0x19BE5F };
CRGBPalette16 pacifica_palette_3 =
    { 0x000208, 0x00030E, 0x000514, 0x00061A, 0x000820, 0x000927, 0x000B2D,
0x000C33,
      0x000E39, 0x001040, 0x001450, 0x001860, 0x001C70, 0x002080, 0x1040BF,
0x2060FF };


void pacifica_loop()
{
  // Increment the four "color index start" counters, one for each wave
layer.
  // Each is incremented at a different speed, and the speeds vary over time.
  static uint16_t sCIStart1, sCIStart2, sCIStart3, sCIStart4;
  static uint32_t sLastms = 0;
  uint32_t ms = GET_MILLIS();
  uint32_t deltams = ms - sLastms;
  sLastms = ms;
  uint16_t speedfactor1 = beatsin16(3, 179, 269);
  uint16_t speedfactor2 = beatsin16(4, 179, 269);
  uint32_t deltams1 = (deltams * speedfactor1) / 256;
  uint32_t deltams2 = (deltams * speedfactor2) / 256;
  uint32_t deltams21 = (deltams1 + deltams2) / 2;
  sCIStart1 += (deltams1 * beatsin88(1011,10,13));
  sCIStart2 -= (deltams21 * beatsin88(777,8,11));
  sCIStart3 -= (deltams1 * beatsin88(501,5,7));
  sCIStart4 -= (deltams2 * beatsin88(257,4,6));

  // Clear out the LED array to a dim background blue-green
  fill_solid( leds, NUM_LEDS, CRGB( 2, 6, 10));

  // Render each of four layers, with different scales and speeds, that vary
over time
  pacifica_one_layer( pacifica_palette_1, sCIStart1, beatsin16( 3, 11 * 256,
14 * 256), beatsin8( 10, 70, 130), 0-beat16( 301) );
  pacifica_one_layer( pacifica_palette_2, sCIStart2, beatsin16( 4,  6 * 256,
9 * 256), beatsin8( 17, 40,  80), beat16( 401) );
  pacifica_one_layer( pacifica_palette_3, sCIStart3, 6 * 256, beatsin8( 9,
10,38), 0-beat16(503));
```

```
  pacifica_one_layer( pacifica_palette_3, sCIStart4, 5 * 256, beatsin8( 8,
10,28), beat16(601));

  // Add brighter 'whitecaps' where the waves lines up more
  pacifica_add_whitecaps();

  // Deepen the blues and greens a bit
  pacifica_deepen_colors();
}

// Add one layer of waves into the led array
void pacifica_one_layer( CRGBPalette16& p, uint16_t cistart, uint16_t
wavescale, uint8_t bri, uint16_t ioff)
{
  uint16_t ci = cistart;
  uint16_t waveangle = ioff;
  uint16_t wavescale_half = (wavescale / 2) + 20;
  for( uint16_t i = 0; i < NUM_LEDS; i++) {
    waveangle += 250;
    uint16_t s16 = sin16( waveangle ) + 32768;
    uint16_t cs = scale16( s16 , wavescale_half ) + wavescale_half;
    ci += cs;
    uint16_t sindex16 = sin16( ci) + 32768;
    uint8_t sindex8 = scale16( sindex16, 240);
    CRGB c = ColorFromPalette( p, sindex8, bri, LINEARBLEND);
    leds[i] += c;
  }
}

// Add extra 'white' to areas where the four layers of light have lined up
brightly
void pacifica_add_whitecaps()
{
  uint8_t basethreshold = beatsin8( 9, 55, 65);
  uint8_t wave = beat8( 7 );

  for( uint16_t i = 0; i < NUM_LEDS; i++) {
    uint8_t threshold = scale8( sin8( wave), 20) + basethreshold;
    wave += 7;
    uint8_t l = leds[i].getAverageLight();
    if( l > threshold) {
      uint8_t overage = l - threshold;
      uint8_t overage2 = qadd8( overage, overage);
      leds[i] += CRGB( overage, overage2, qadd8( overage2, overage2));
    }
  }
}

// Deepen the blues and greens
void pacifica_deepen_colors()
{
  for( uint16_t i = 0; i < NUM_LEDS; i++) {
    leds[i].blue = scale8( leds[i].blue,  145);
    leds[i].green= scale8( leds[i].green, 200);
    leds[i] |= CRGB( 2, 5, 7);
  }
}

// This function draws rainbows with an ever-changing,
// widely-varying set of parameters.
void pride()
{
  static uint16_t sPseudotime = 0;
  static uint16_t sLastMillis = 0;
  static uint16_t sHue16 = 0;

  uint8_t sat8 = beatsin88( 87, 220, 250);
  uint8_t brightdepth = beatsin88( 341, 96, 224);
  uint16_t brightnessthetainc16 = beatsin88( 203, (25 * 256), (40 * 256));
  uint8_t msmultiplier = beatsin88(147, 23, 60);

  uint16_t hue16 = sHue16;//gHue * 256;
  uint16_t hueinc16 = beatsin88(113, 1, 3000);

  uint16_t ms = millis();
  uint16_t deltams = ms - sLastMillis ;
  sLastMillis  = ms;
  sPseudotime += deltams * msmultiplier;
  sHue16 += deltams * beatsin88( 400, 5,9);
  uint16_t brightnesstheta16 = sPseudotime;

  for( uint16_t i = 0 ; i < NUM_LEDS; i++) {
    hue16 += hueinc16;
    uint8_t hue8 = hue16 / 256;

    brightnesstheta16  += brightnessthetainc16;
    uint16_t b16 = sin16( brightnesstheta16  ) + 32768;

    uint16_t bri16 = (uint32_t)((uint32_t)b16 * (uint32_t)b16) / 65536;
    uint8_t bri8 = (uint32_t)(((uint32_t)bri16) * brightdepth) / 65536;
    bri8 += (255 - brightdepth);

    CRGB newcolor = CHSV( hue8, sat8, bri8);

    uint16_t pixelnumber = i;
    pixelnumber = (NUM_LEDS-1) - pixelnumber;

    nblend( leds[pixelnumber], newcolor, 64);
  }
}


void twinkle() {
  // Set a random LED to a random color
  leds[random(NUM_LEDS)] = CRGB(random(0,255), random(0,255), random(0,255));
  // Fade out all LEDs
  for (int i = 0; i < NUM_LEDS; i++) {
    leds[i].fadeToBlackBy(20);
  }
}

// List of patterns to cycle through.  Each is defined as a separate function
below.
typedef void (*SimplePatternList[])();
SimplePatternList gPatterns = { rainbow, rainbowWithGlitter, confetti,
sinelon, juggle, bpm };

uint8_t gCurrentPatternNumber = 0; // Index number of which pattern is
current
uint8_t gHue = 0; // rotating "base color" used by many of the patterns




#define ARRAY_SIZE(A) (sizeof(A) / sizeof((A)[0]))

void nextPattern()
{
  // add one to the current pattern number, and wrap around at the end
  gCurrentPatternNumber = (gCurrentPatternNumber + 1) % ARRAY_SIZE(
gPatterns);
}

void rainbow()
{
  // FastLED's built-in rainbow generator
  fill_rainbow( leds, NUM_LEDS, gHue, 7);
}

void rainbowWithGlitter()
{
  // built-in FastLED rainbow, plus some random sparkly glitter
  rainbow();
  addGlitter(80);
}

void addGlitter( fract8 chanceOfGlitter)
{
  if( random8() < chanceOfGlitter) {
    leds[ random16(NUM_LEDS) ] += CRGB::White;
  }
}

void confetti()
{
  // random colored speckles that blink in and fade smoothly
  fadeToBlackBy( leds, NUM_LEDS, 10);
  int pos = random16(NUM_LEDS);
  leds[pos] += CHSV( gHue + random8(64), 200, 255);
}

void sinelon()
{
  // a colored dot sweeping back and forth, with fading trails
  fadeToBlackBy( leds, NUM_LEDS, 20);
  int pos = beatsin16( 13, 0, NUM_LEDS-1 );
  leds[pos] += CHSV( gHue, 255, 192);
}

void bpm()
{
  // colored stripes pulsing at a defined Beats-Per-Minute (BPM)
  uint8_t BeatsPerMinute = 62;
  CRGBPalette16 palette = PartyColors_p;
  uint8_t beat = beatsin8( BeatsPerMinute, 64, 255);
  for( int i = 0; i < NUM_LEDS; i++) { //9948
    leds[i] = ColorFromPalette(palette, gHue+(i*2), beat-gHue+(i*10));
  }
}

void juggle() {
  // eight colored dots, weaving in and out of sync with each other
  fadeToBlackBy( leds, NUM_LEDS, 20);
  uint8_t dothue = 0;
  for( int i = 0; i < 8; i++) {
    leds[beatsin16( i+7, 0, NUM_LEDS-1 )] |= CHSV(dothue, 200, 255);
    dothue += 32;
  }
}
// This function sets up the ledsand tells the controller about them
void setup() {

      // Initialize the ultrasonic sensor pins
  //pinMode(trigPin, OUTPUT);
  //pinMode(echoPin, INPUT);

  // Initialize the serial communication
  Serial.begin(9600);
  btSerial.begin(9600); // initialize Bluetooth communication

  // sanity check delay - allows reprogramming if accidently blowing power
w/leds
    delay(2000);
  pinMode(bottonPin,INPUT);
    // Uncomment/edit one of the following lines for your leds arrangement.
    // ## Clockless types ##

    FastLED.addLeds<WS2811, DATA_PIN, RGB>(leds, NUM_LEDS);
    // ## Clocked (SPI) types ##
pixels.begin(); // Initialize LED strip
}
```

```
// This function runs over and over, and is where you do the magic to light
// your leds.
int t=6;
void loop() {

if(Serial.available() > 0)
  {33333333330000000000000006
     t = Serial.read();
    Serial.print(t);
    Serial.print("\n");     }

bt=digitalRead(bottonPin);
if (bt==1){
            if (t==9){t=1;}
            else{t++;}
            delay(500);}


if (t==1){

   // Move a single white led
   for(int whiteLed = 0; whiteLed < NUM_LEDS; whiteLed = whiteLed + 1) {
      // Turn our current led on to white, then show the leds
      leds[whiteLed] = CRGB(2, 248, 35);

      // Show the leds (only one of which is set to white, from above)
      FastLED.show();

      // Wait a little bit
      delay(10);

      // Turn our current led back to black for the next loop around
      leds[whiteLed] = CRGB::Black;}}



    else if(t==2)
       { //twinkle
  twinkle();
  FastLED.show();
  FastLED.delay(1000 / FRAMES_PER_SECOND);
    }

 else if(t==5)
  {
///     Pacifica
  EVERY_N_MILLISECONDS( 20) {
    pacifica_loop();
    FastLED.show();
  }
}
```

```
    else if(t==3){

            //Add entropy to random number generator; we use a lot of it.
   // random16_add_entropy( random());

   Fire2012(); // run simulation frame

   FastLED.show(); // display this frame
   FastLED.delay(1000 / FRAMES_PER_SECOND);

    }

 else if(t==4)
 {  //Pride2015
   pride();
   FastLED.show();
 }


   else if(t==6){
 //     rainbow
    static uint8_t hue = 0;

   // Fill the entire strip with a rainbow of colors
   for (int i = 0; i < NUM_LEDS; i++) {
     leds[i] = CHSV(hue + (i * 255 / NUM_LEDS), 255, 255);
   }

   // Move the rainbow along the strip
   for (int i = 0; i < NUM_LEDS; i++) {
     leds[i] = leds[(i + 1) % NUM_LEDS];
   }

   FastLED.show();

   hue++;
   }

 else if(t==7){

   // DemoReel100.
     // Call the current pattern function once, updating the 'leds' array
   gPatterns[gCurrentPatternNumber]();

   // send the 'leds' array out to the actual LED strip
   FastLED.show();
   // insert a delay to keep the framerate modest
   FastLED.delay(1000/FRAMES_PER_SECOND);

   // do some periodic updates
   EVERY_N_MILLISECONDS( 20 ) { gHue++; }
 // slowly cycle the "base color" through the rainbow
   EVERY_N_SECONDS( 10 ) { nextPattern(); }
```

```
// change patterns periodically
  }

else if(t==8){

    ChangePalettePeriodically();

    static uint8_t startIndex = 0;
    startIndex = startIndex + 1; /* motion speed */

    FillLEDsFromPaletteColors( startIndex);

    FastLED.show();
    FastLED.delay(1000 / UPDATES_PER_SECOND);}


else if (t==9)
{
  int sensitivity = map(analogRead(A6), 0, 1023, 50, 100);
  Serial.println(analogRead(A6));

  for (int i = 0; i < 64; i++)
  {
    realComponent[i] = analogRead(micPin) / sensitivity;  // Read from the
microphone sensor pin
    imagComponent[i] = 0;
  }

  FFT.Windowing(realComponent, 64, FFT_WIN_TYP_HAMMING, FFT_FORWARD);
  FFT.Compute(realComponent, imagComponent, 64, FFT_FORWARD);
  FFT.ComplexToMagnitude(realComponent, imagComponent, 64);

  for (int i = 0; i < 16; i++)
  {
    int ledIndex = i % 4 * 4 + i / 4;
    realComponent[i] = constrain(realComponent[i], 0, 80);
    realComponent[i] = map(realComponent[i], 0, 80, 0, 7);
    index = realComponent[i];
    c = spectralHeight[index];

    led.setPixelColor(ledIndex, c, 0, 0); // Set RGB values accordingly
(e.g., Red)
    led.setPixelColor(ledIndex+16, c, 0, 0);
    led.setPixelColor(ledIndex+32, c, 0, 0);

  }
  led.show();
}
pixels.show();
}
```