Lab 2: Melodies and LED’s, but it's not a Rave.

Mason Wheeler, 2032634
Joseph Pirich, 2042211 12-Apr-2023
Assignment: ECE474 Lab 2

Abstract

This lab report presents the implementation and results of driving digital outputs to display the location of a thumbstick toggle to a 8x8 LED matrix as well as produce a tone on a speaker using an Arduino Mega microcontroller board. The project aimed to enhance our understanding of low-level hardware manipulation, specifically controlling hardware registers/bits without relying on existing libraries, and coordinating multiple concurrent tasks through round-robin scheduling. We successfully implemented a LED flashing sequence, generated a specific tone sequence on a speaker, and created an interactive XY LED display matrix that responded to thumbstick inputs. The project showcased our ability to manage multiple tasks concurrently and demonstrated their proficiency in manipulating hardware registers/bits directly.

1. Introduction

Working as a team we took on the challenge of controlling an 8x8 LED matrix and producing a tone on a speaker using an Arduino Mega microcontroller board. The main goal of this project was to help us better understand low-level hardware manipulation, programming, and managing multiple tasks at once.

For this project, we were required to create three different functions: (1) a flashing LED sequence with 3 LED’s using bit operations. (2) a specific tone sequence representing Mary had a Little Lamb using a square tone played on a speaker, and (3) an interactive XY LED display matrix that reacted to thumbstick inputs. To add to the challenge, we weren't allowed to use pre-built libraries and had to manipulate the hardware registers and bits directly.

In this report, we'll discuss the basic knowledge we needed to understand before starting the project, the steps we took to achieve the objectives, the technical aspects and challenges we encountered, and the results we obtained. This project was an excellent opportunity for us to apply what we've learned in class to a real-world situation, helping our team grow our skills and understanding of microcontrollers.

2. Methods and Techniques:

In this lab report, we used Arduino programming, LED control, and speaker control to develop a system that can both light up a sequence of three LEDs and play the "Mary Had a Little Lamb" tune. We took advantage of the millis() function to manage multiple tasks concurrently, allowing the LEDs and speaker to work at the same time without interrupting each other.

For controlling the speaker, we applied pulse-width modulation (PWM) to create different frequencies corresponding to the musical notes of the song. This technique allowed us to generate the desired melody accurately.
Furthermore, we expanded the project by integrating an LED matrix and a thumbstick, which together formed an interactive visual display. The thumbstick inputs were used to manipulate the LED matrix, providing an engaging and dynamic experience for the user. Throughout the development process, we ensured that the added components did not interfere with the ongoing operation of the LED sequence and speaker functionality.

3. Procedures and Results

1.2:
Procedure:
In this part, we implemented code to control three LEDs connected to pins 47, 48, and 49 on an Arduino board.

1. We start by defining three constants to represent the LED pins: ledPin1, ledPin2, and ledPin3. They are assigned the values 47, 48, and 49 respectively.
2. In the setup() function, we set the pinMode for each LED pin to OUTPUT using pinMode(). This configures the pins to be used as output pins to control the LEDs.
3. In the loop() function, we use digitalWrite() to turn each LED on and off sequentially. We turn on an LED by setting its corresponding pin to HIGH and turn it off by setting the pin to LOW.
4. We introduce a delay of 333 milliseconds between turning on and off each LED using the delay() function. This creates a pattern where each LED is lit for 333 milliseconds before moving on to the next one.

Results:
When the code is uploaded to the Arduino, the LEDs connected to pins 47, 48, and 49 light up sequentially for 333 milliseconds each. After the last LED turns off, the cycle repeats.
1.4:
Procedure:
In this part, we implemented a similar LED control pattern as in 1.2, but using lower-level programming with direct port manipulation.

1. In the setup() function, we set pins 47, 48, and 49 as output pins by manipulating the DDR (Data Direction Register) for port L. We use the bitwise OR operator (|=) to set the corresponding bits for the LED pins without affecting other pins.
2. In the loop() function, we use the PORT register for port L to control the LEDs. We turn an LED on by setting its corresponding bit using the bitwise OR operator (|=) and turn it off by clearing the bit using the bitwise AND operator (&=) with the bitwise NOT operator (~).

3. We introduce a delay of 333 milliseconds between turning on and off each LED using the delay() function, as in the previous example.

Results:
When the code is uploaded to the Arduino, the LEDs connected to pins 47, 48, and 49 light up sequentially for 333 milliseconds each, just like in the previous example. However, the code in this part uses direct port manipulation, which provides faster and more efficient control of the pins.

Procedure for Task 3.2:

1. Modify the provided code for Task A and Task B to run concurrently, while maintaining the same functionality for each individual task.
2. Add global variables to track the completion status of Task A and Task B.
3. Implement a state machine in the controlTasks() function to manage the execution of both tasks in different phases.
4. Modify the runTaskA() and runTaskB() functions to run only when enabled by the controlTasks() function.
5. Test the modified code to ensure that Task A and Task B run concurrently, and the desired functionality is achieved.

Results for Task 3.2:
The modified code successfully demonstrated the simultaneous operation of Task A and Task B. The controlTasks() function managed the different phases of execution and enabled the tasks accordingly. The LED sequence (Task A) and the melody (Task B) were both performed concurrently as expected.
Procedure for Task 4:

1. Modify the given code to include a thumbstick-controlled LED matrix as an interactive display.
2. Add global variables and function prototypes to handle the LED matrix and thumbstick inputs.
3. Implement the spiTransfer() function to transfer data to the LED matrix.
4. Implement the convertToIndex() function to convert the thumbstick value to a row or column index for the LED matrix.
5. Modify the loop() function to include reading the thumbstick inputs and controlling the LED matrix based on the inputs.
6. Test the modified code to ensure that the interactive display works as expected, in addition to the simultaneous operation of Task A and Task B.

Results for Task 4:
The modified code successfully integrated a thumbstick-controlled LED matrix as an interactive display. The thumbstick inputs were accurately translated to row and column indices for the LED matrix, and the display responded accordingly. The LED matrix functionality did not interfere with the concurrent operation of Task A and Task B.

4. **Code Documentation:**

The provided code is designed to set up and manage three LEDs, a speaker, and an LED matrix connected to a thumbstick. The core functionality of the code is split into two distinct tasks, referred to as Task A and Task B. Task A is responsible for managing the sequential illumination of the LEDs, while Task B takes charge of playing the "Mary Had a Little Lamb" tune using the speaker.
The controlTasks() function plays a crucial role in overseeing the progression of both tasks through their respective phases, ensuring smooth transitions and proper execution. Meanwhile, the LED matrix and the attached thumbstick are managed within the main loop of the code. This arrangement allows for continuous input from the thumbstick, which in turn manipulates the LED matrix display.
Additionally, the code includes a conversion function that effectively maps the thumbstick readings to row and column indices on the LED matrix. This mapping enables the user to interact with the visual display, providing an engaging and dynamic experience while maintaining the functionality of the LED sequence and the speaker.

5. **Overall Performance Summary:**

The code performs well in achieving the desired outcome: sequentially lighting up the LEDs and playing the "Mary Had a Little Lamb" tune. The LEDs operate smoothly and transition without any noticeable delay. The speaker plays the tune with the correct note durations and maintains the melody throughout the sequence. The LED matrix and thumbstick provide a responsive and engaging visual display, with no noticeable lag or inaccuracies.

6. **Teamwork Breakdown:**

We worked together as a team, dividing the tasks among ourselves and regularly communicating our progress. We collaboratively designed the overall structure of the code and then split the implementation of Task A, Task B, and the LED matrix/thumbstick control. Each team member contributed to debugging and refining the code, ensuring a smooth integration of all components.

7. **Discussion:**

Our project showcases the ability to control multiple hardware components simultaneously using Arduino programming. The integration of millis() allowed us to manage concurrent tasks without resorting to delay(), enabling the parallel operation of LEDs and the speaker. The use of an LED matrix and thumbstick added an interactive aspect to the project, enhancing its appeal and engagement.

8. **Conclusion:**

In conclusion, our team effectively designed and implemented a system that controls three LEDs and a speaker, playing a tune while also offering an interactive visual display. Through collaboration and clear communication, we successfully met our project goals and developed an engaging final product.
This project helped us enhance our skills in Arduino programming and showed us the potential of this technology for creative and interactive applications. We are excited about our

accomplishments and eager to explore further possibilities in the future, applying the knowledge and experience we gained from this project to new challenges and endeavors.

Labs/Lab02/LED_sequence/1.2.ino

```
/*
Place the LEDs and resistors on the breadboard:
Insert the anode (longer leg) of the first LED into a row on the breadboard, and connect its cathode (shorter leg) to another row.
Connect a 500-250 Ohm resistor to the cathode row of the LED and connect the other end of the resistor to the ground rail of the breadboard.
Repeat these steps for the second and third LEDs, leaving some space between them for easy identification and cable management.
*/

const int ledPin1 = 47;
const int ledPin2 = 48;
const int ledPin3 = 49;

void setup() {
  pinMode(ledPin1, OUTPUT);
  pinMode(ledPin2, OUTPUT);
  pinMode(ledPin3, OUTPUT);
}

void loop() {
  digitalWrite(ledPin1, HIGH);
  delay(333);
  digitalWrite(ledPin1, LOW);

  digitalWrite(ledPin2, HIGH);
  delay(333);
  digitalWrite(ledPin2, LOW);

  digitalWrite(ledPin3, HIGH);
  delay(333);
  digitalWrite(ledPin3, LOW);
}
```

Labs/Lab02/LED_sequence/1.4.ino

```
void setup() {
  // Set pins 47, 48, 49 as outputs
  DDRL |= (1 << LED_PIN_47_BIT) | (1 << LED_PIN_48_BIT) | (1 << LED_PIN_49_BIT);
}

void loop() {
  // Turn on LED at pin 47
  PORTL |= (1 << LED_PIN_47_BIT);
  delay(333);
  // Turn off LED at pin 47
  PORTL &= ~(1 << LED_PIN_47_BIT);

  // Turn on LED at pin 48
  PORTL |= (1 << LED_PIN_48_BIT);
  delay(333);
  // Turn off LED at pin 48
  PORTL &= ~(1 << LED_PIN_48_BIT);

  // Turn on LED at pin 49
  PORTL |= (1 << LED_PIN_49_BIT);
  delay(333);
  // Turn off LED at pin 49
  PORTL &= ~(1 << LED_PIN_49_BIT);
}
```

Labs/Lab02/Concurrent/Concurrent.ino

```
// Global variables
unsigned long previousMillisA = 0;
unsigned long previousMillisB = 0;
unsigned long previousMillisC = 0;
unsigned long noteStartTime = 0;

// Add two new global variables
bool taskACompleted = false;
bool taskBCompleted = false;

// Add a new global variable
unsigned long noteGapStartTime = 0;
bool gapState = false;


#define LED_PIN_47_BIT 0
#define LED_PIN_48_BIT 1
#define LED_PIN_49_BIT 2

#define SPEAKER_PIN 6

const unsigned long intervalA = 333;
const unsigned long intervalB[] = {2000, 10000, 1000}; // Task B durations
const unsigned long noteDurations[] = {500, 500, 500, 500, 500, 500, 1000, 500, 500,
1000, 500, 500, 1000, 500, 500, 500, 500, 500, 500, 1000, 500, 500, 500, 500, 500, 1000};
// Note durations
uint8_t currentNote = 0;

int phase = 0;
bool taskAEnabled = false;
bool taskBEnabled = false;

// Frequencies for "Mary Had a Little Lamb"
uint16_t frequencies[] = {494, 440, 392, 440, 494, 494, 494, 440, 440, 440, 494, 587,
587, 494, 440, 392, 440, 494, 494, 494, 494, 440, 440, 494, 440, 392};

void setup() {
  // Task A setup
  DDRL |= (1 << LED_PIN_47_BIT) | (1 << LED_PIN_48_BIT) | (1 << LED_PIN_49_BIT);

  // Task B setup
  pinMode(SPEAKER_PIN, OUTPUT);
  TCCR4A = (1 << COM4A1) | (1 << WGM41);
  TCCR4B = (1 << WGM43) | (1 << WGM42) | (1 << CS41);
  ICR4 = 40000;

  // Initialize Task A
  taskAEnabled = true;
  taskBEnabled = false;
}

void loop() {
  controlTasks();
```

```
  runTaskA(); // This will run continuously
  runTaskB();
}

void controlTasks() {
  unsigned long currentMillisC = millis();

  switch (phase) {
    case 0:
      taskAEnabled = true;
      taskBEnabled = false;
      if (taskACompleted) {
        taskACompleted = false;
        previousMillisC = currentMillisC;
        phase = 1;
      }
      break;
    case 1:
      // Add an extra intervalA duration for the third LED to stay on
      if (currentMillisC - previousMillisC >= intervalA) {
        taskAEnabled = false;
        taskBEnabled = true;
        if (taskBCompleted) {
          taskBCompleted = false;
          previousMillisC = currentMillisC;
          phase = 2;
        }
      }
      break;
    case 2:
      taskAEnabled = true;
      taskBEnabled = true;
      if (taskACompleted && taskBCompleted) {
        taskACompleted = false;
        taskBCompleted = false;
        previousMillisC = currentMillisC;
        phase = 3;
      }
      break;
    case 3:
      taskAEnabled = false;
      taskBEnabled = false;
      if (currentMillisC - previousMillisC >= 1000) {
        previousMillisC = currentMillisC;
        phase = 0;
      }
      break;
  }
}

void runTaskA() {
  if (!taskAEnabled) {
    PORTL &= ~((1 << LED_PIN_47_BIT) | (1 << LED_PIN_48_BIT) | (1 << LED_PIN_49_BIT)); //
Turn off all LEDs
    return;
  }
```

```
    static uint8_t ledState = 0;

    unsigned long currentMillisA = millis();
    if (currentMillisA - previousMillisA >= intervalA) {
      previousMillisA = currentMillisA;

      updateLEDs(ledState);
      ledState = (ledState + 1) % 3;

      // Set taskACompleted to true when the LED sequence has completed 3 cycles
      if (ledState == 0) {
        taskACompleted = true;
      }
    }
  }

  // runTaskB() function
  void runTaskB() {
    if (!taskBEnabled) {
      OCR4A = 0; // Set duty cycle to 0% to silence the speaker
      return;
    }

    unsigned long currentMillisB = millis();

    if (gapState) {
      if (currentMillisB - noteGapStartTime >= 100) { // 100 ms gap between notes
        gapState = false;
        play_tone(frequencies[currentNote], noteDurations[currentNote]);
        noteStartTime = currentMillisB;
      }
    } else {
      if (currentMillisB - noteStartTime >= noteDurations[currentNote]) {
        silence();
        noteGapStartTime = currentMillisB;
        gapState = true;
        currentNote = (currentNote + 1) % (sizeof(noteDurations) /
  sizeof(noteDurations[0]));

        // Set taskBCompleted to true when the song has completed
        if (currentNote == 0) {
          taskBCompleted = true;
        }
      }
    }
  }

  void updateLEDs(uint8_t ledState) {
    PORTL &= ~((1 << LED_PIN_47_BIT) | (1 << LED_PIN_48_BIT) | (1 << LED_PIN_49_BIT)); //
  Turn off all LEDs

    switch (ledState) {
      case 0:
        PORTL |= (1 << LED_PIN_47_BIT); // Turn on LED at pin 47
        break;
      case 1:
        PORTL |= (1 << LED_PIN_48_BIT); // Turn on LED at pin 48
```

```
      break;
    case 2:
      PORTL |= (1 << LED_PIN_49_BIT); // Turn on LED at pin 49
      break;
  }
}

void play_tone(uint16_t frequency, uint32_t duration) {
  ICR4 = F_CPU / (8 * frequency); // Calculate the TOP value based on the frequency
  OCR4A = ICR4 / 2; // Set the duty cycle to 50%
  noteStartTime = millis(); // Store the start time of the note
}

void silence() {
  OCR4A = 0; // Set the duty cycle to 0% to silence the speaker
}
```

Labs/Lab02/Final_Task/Final_Task.ino

```
#include <Arduino.h>
#define LED_PIN_47_BIT 0
#define LED_PIN_48_BIT 1
#define LED_PIN_49_BIT 2
#define OP_DECODEMODE  8
#define OP_SCANLIMIT   10
#define OP_SHUTDOWN    11
#define OP_DISPLAYTEST 14
#define OP_INTENSITY   10

#define SPEAKER_PIN 6
// Global variables
unsigned long previousMillisA = 0;
unsigned long previousMillisB = 0;
unsigned long previousMillisC = 0;
unsigned long noteStartTime = 0;

// Add a new global variable
unsigned long noteGapStartTime = 0;
bool gapState = false;

// LED matrix and thumbstick control variables
int DIN = 22; // Changed from 47
int CS =  24; // Changed from 49
int CLK = 26; // Changed from 51

int THUMBSTICK_X = A0;
int THUMBSTICK_Y = A1;
byte spidata[2];


// Function prototypes
void spiTransfer(volatile byte opcode, volatile byte data);
int convertToIndex(int value, bool invert = false);

// Function to transfer data to the LED matrix
void spiTransfer(volatile byte opcode, volatile byte data){
  int offset = 0;
  int maxbytes = 2;

  // Clear the SPI data buffer
  for(int i = 0; i < maxbytes; i++) {
    spidata[i] = (byte)0;
  }

  // Load SPI data
  spidata[offset+1] = opcode+1;
  spidata[offset] = data;

  // Send SPI data
  digitalWrite(CS, LOW);
  for(int i=maxbytes;i>0;i--)
```

```
      shiftOut(DIN,CLK,MSBFIRST,spidata[i-1]);
    digitalWrite(CS,HIGH);
  }


  // Function to convert the thumbstick value to a row or column index
  int convertToIndex(int value, bool invert) {
    if (invert) {
      value = 1023 - value;
    }
    int index = (int)((value / 1023.0) * 8);
    // Limit the index to be within the valid range (0-7)
    index = min(max(index, 0), 7);
    return index;
  }

  // Add two new global variables
  bool taskACompleted = false;
  bool taskBCompleted = false;

  const unsigned long intervalA = 333;
  const unsigned long intervalB[] = {2000, 10000, 1000}; // Task B durations
  const unsigned long noteDurations[] = {500, 500, 500, 500, 500, 500, 1000, 500, 500,
  1000, 500, 500, 1000, 500, 500, 500, 500, 500, 500, 1000, 500, 500, 500, 500, 500, 1000};
  // Note durations
  uint8_t currentNote = 0;

  int phase = 0;
  bool taskAEnabled = false;
  bool taskBEnabled = false;

  // Frequencies for "Mary Had a Little Lamb"
  uint16_t frequencies[] = {494, 440, 392, 440, 494, 494, 494, 440, 440, 440, 494, 587,
  587, 494, 440, 392, 440, 494, 494, 494, 494, 440, 440, 494, 440, 392};

  void setup() {
    // Task A setup
    DDRL |= (1 << LED_PIN_47_BIT) | (1 << LED_PIN_48_BIT) | (1 << LED_PIN_49_BIT);

    // Task B setup
    pinMode(SPEAKER_PIN, OUTPUT);
    TCCR4A = (1 << COM4A1) | (1 << WGM41);
    TCCR4B = (1 << WGM43) | (1 << WGM42) | (1 << CS41);
    ICR4 = 40000;

    // Initialize Task A
    taskAEnabled = true;
    taskBEnabled = false;

    pinMode(DIN, OUTPUT);
    pinMode(CS, OUTPUT);
    pinMode(CLK, OUTPUT);
    digitalWrite(CS, HIGH);

     // Initialize the LED matrix
    spiTransfer(OP_DISPLAYTEST,0);
    spiTransfer(OP_SCANLIMIT,7);
```

```
    spiTransfer(OP_DECODEMODE,0);
    spiTransfer(OP_SHUTDOWN,1);

    // Clear the display
    for (int i = 0; i < 8; i++) {
      spiTransfer(i, 0);
    }

  }

  void loop() {
    controlTasks();
    runTaskA(); // This will run continuously
    runTaskB();

    int row = convertToIndex(analogRead(THUMBSTICK_Y));
    int col = convertToIndex(analogRead(THUMBSTICK_X), true);

    // Light up the LED at the specified row and column
    spiTransfer(row, 1 << col);

    delay(50); // Add this delay to allow the LED to turn on completely

    // Turn off the LED at the specified row and column
    spiTransfer(row, 0);
  }


  void controlTasks() {
    unsigned long currentMillisC = millis();

    switch (phase) {
      case 0:
        taskAEnabled = true;
        taskBEnabled = false;
        if (taskACompleted) {
          taskACompleted = false;
          previousMillisC = currentMillisC;
          phase = 1;
        }
        break;
      case 1:
        // Add an extra intervalA duration for the third LED to stay on
        if (currentMillisC - previousMillisC >= intervalA) {
          taskAEnabled = false;
          taskBEnabled = true;
          if (taskBCompleted) {
            taskBCompleted = false;
            previousMillisC = currentMillisC;
            phase = 2;
          }
        }
        break;
      case 2:
        taskAEnabled = true;
        taskBEnabled = true;
```

```
        if (taskACompleted && taskBCompleted) {
          taskACompleted = false;
          taskBCompleted = false;
          previousMillisC = currentMillisC;
          phase = 3;
        }
        break;
      case 3:
        taskAEnabled = false;
        taskBEnabled = false;
        if (currentMillisC - previousMillisC >= 1000) {
          previousMillisC = currentMillisC;
          phase = 0;
        }
        break;
    }
  }

  void runTaskA() {
    if (!taskAEnabled) {
      PORTL &= ~((1 << LED_PIN_47_BIT) | (1 << LED_PIN_48_BIT) | (1 << LED_PIN_49_BIT)); //
  Turn off all LEDs
      return;
    }

    static uint8_t ledState = 0;

    unsigned long currentMillisA = millis();
    if (currentMillisA - previousMillisA >= intervalA) {
      previousMillisA = currentMillisA;

      updateLEDs(ledState);
      ledState = (ledState + 1) % 3;

      // Set taskACompleted to true when the LED sequence has completed 3 cycles
      if (ledState == 0) {
        taskACompleted = true;
      }
    }
  }

  // runTaskB() function
  void runTaskB() {
    if (!taskBEnabled) {
      OCR4A = 0; // Set duty cycle to 0% to silence the speaker
      return;
    }

    unsigned long currentMillisB = millis();

    if (gapState) {
      if (currentMillisB - noteGapStartTime >= 100) { // 100 ms gap between notes
        gapState = false;
        play_tone(frequencies[currentNote], noteDurations[currentNote]);
        noteStartTime = currentMillisB;
      }
    } else {
```

```
      if (currentMillisB - noteStartTime >= noteDurations[currentNote]) {
        silence();
        noteGapStartTime = currentMillisB;
        gapState = true;
        currentNote = (currentNote + 1) % (sizeof(noteDurations) /
sizeof(noteDurations[0]));

        // Set taskBCompleted to true when the song has completed
        if (currentNote == 0) {
          taskBCompleted = true;
        }
      }
    }
  }

  void updateLEDs(uint8_t ledState) {
    PORTL &= ~((1 << LED_PIN_47_BIT) | (1 << LED_PIN_48_BIT) | (1 << LED_PIN_49_BIT)); //
  Turn off all LEDs

    switch (ledState) {
      case 0:
        PORTL |= (1 << LED_PIN_47_BIT); // Turn on LED at pin 47
        break;
      case 1:
        PORTL |= (1 << LED_PIN_48_BIT); // Turn on LED at pin 48
        break;
      case 2:
        PORTL |= (1 << LED_PIN_49_BIT); // Turn on LED at pin 49
        break;
    }
  }

  void play_tone(uint16_t frequency, uint32_t duration) {
    ICR4 = F_CPU / (8 * frequency); // Calculate the TOP value based on the frequency
    OCR4A = ICR4 / 2; // Set the duty cycle to 50%
    noteStartTime = millis(); // Store the start time of the note
  }

  void silence() {
    OCR4A = 0; // Set the duty cycle to 0% to silence the speaker
  }
```

Labs/Lab02/LED_Matrix/LEDS.ino

```
#include <Arduino.h>

#define OP_DECODEMODE  8
#define OP_SCANLIMIT   10
#define OP_SHUTDOWN    11
#define OP_DISPLAYTEST 14
#define OP_INTENSITY   10

int DIN = 47;
int CS =  49;
int CLK = 51;
int THUMBSTICK_X = A0;
int THUMBSTICK_Y = A1;

byte spidata[2];

// Function prototypes
void spiTransfer(volatile byte opcode, volatile byte data);
int readThumbstickValue(int pin);
int convertToIndex(int value, bool invert = false);

// Setup function
void setup(){
  // Configure pins for the LED matrix
  pinMode(DIN, OUTPUT);
  pinMode(CS, OUTPUT);
  pinMode(CLK, OUTPUT);
  digitalWrite(CS, HIGH);

  // Initialize the LED matrix
  spiTransfer(OP_DISPLAYTEST,0);
  spiTransfer(OP_SCANLIMIT,7);
  spiTransfer(OP_DECODEMODE,0);
  spiTransfer(OP_SHUTDOWN,1);

  // Clear the display
  for (int i = 0; i < 8; i++) {
    spiTransfer(i, 0);
  }

  // Initialize serial communication
  Serial.begin(9600);
}

// Main loop function
void loop(){
  // Read the thumbstick values and convert them to row and column indices
  int row = convertToIndex(readThumbstickValue(THUMBSTICK_Y));
  int col = convertToIndex(readThumbstickValue(THUMBSTICK_X), true);

  // Print the row and column values to the serial monitor
  Serial.print("Row: ");
```

```
    Serial.print(row);
    Serial.print(", Col: ");
    Serial.println(col);

    // Light up the LED at the specified row and column
    spiTransfer(row, 1 << col);
    delay(50);

    // Turn off the LED at the specified row and column
    spiTransfer(row, 0);
  }

  // Function to transfer data to the LED matrix
  void spiTransfer(volatile byte opcode, volatile byte data){
    int offset = 0;
    int maxbytes = 2;

    // Clear the SPI data buffer
    for(int i = 0; i < maxbytes; i++) {
      spidata[i] = (byte)0;
    }

    // Load SPI data
    spidata[offset+1] = opcode+1;
    spidata[offset] = data;

    // Send SPI data
    digitalWrite(CS, LOW);
    for(int i=maxbytes;i>0;i--)
      shiftOut(DIN,CLK,MSBFIRST,spidata[i-1]);
    digitalWrite(CS,HIGH);
  }

  // Function to read the thumbstick value
  int readThumbstickValue(int pin) {
    return analogRead(pin);
  }

  // Function to convert the thumbstick value to a row or column index
  int convertToIndex(int value, bool invert) {
    if (invert) {
      value = 1023 - value;
    }
    return (int)((value / 1023.0) * 8);
  }
```