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);
  }
```