VIT®

Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

HANDS-ON COURSE ON VEGA PROCESSORS AND ECOSYSTEM

VIT, CHENNAI

February 22, 2025 – April 14, 2025

Faculty Name:

Dr Vydeki D
Dr Jagannath M

LOKESH R (23BEC1009)

GAUTHAM R (23BEC1069)

MATHESH V (23BEC1152)

CONTENTS

EXAMPLE CODES

Ex. No.	Title	Example pathway
01	Hello World	UART -> Hello_world
02	Internal LED	Basics -> Blink
03	External LED	GPIO -> Blink
04	Serial LED	GPIO -> Series
05	IR Sensors	GPIO -> IR_Sensor_HW201
06	4-Digit Display	GPIO -> Four_Digit_Display
07	Piezo Buzzer	GPIO -> PIEZO_Buzzer
08	PWM Piezo Buzzer	PWM -> PIEZOBuzzer_PWM
09	Ultrasonic Sensor	GPIO -> UltrasonicSensor_HC-SR04
10	GPIO - RYG LED Strip	GPIO -> RYG_LED_GPIO
11	PWM – RYG LED Strip	PWM -> RYG_LED_PWM
12	Touch Sensor	GPIO -> touch_sensor
13	LDR Sensor	ADC -> LDR_Sensor
14	Servo Test	PWM -> Servo Motor
15	RFID Module	SPI -> RC522_ReadRFID
16	RTC Module	Wire -> RTC -> RTC_ReadTime
17	BMP180	BMP180_pre_temp_sensor
18	Bluetooth Module	UART -> Bluetooth_HC05

ACTIVITY 01
DISTANCE MEASUREMENT

PROBLEM STATEMENT

If the IR sensor detects an object, show "SAFE" on the display. If no object is detected, show "DEAD" on the display.

COMPONENTS USED

Aries Development Board v3, USB Cable, IR sensor, 4-bit display, jumper wires

PIN CONNECTIONS

Seven segment display:

DIO → GPIO 0

CLK → GPIO 1

VCC → 3V3

GND → GND

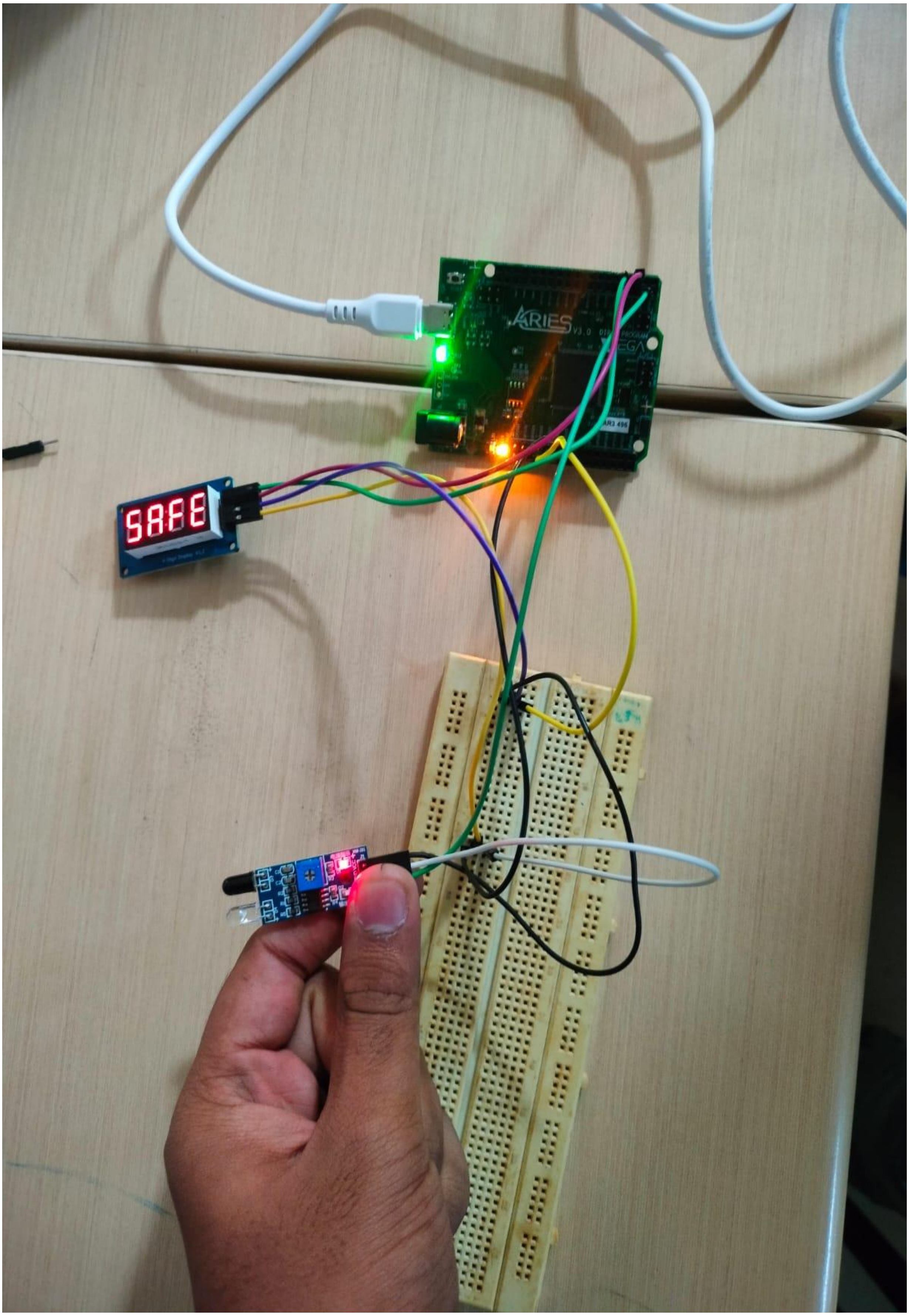
IR Sensor:

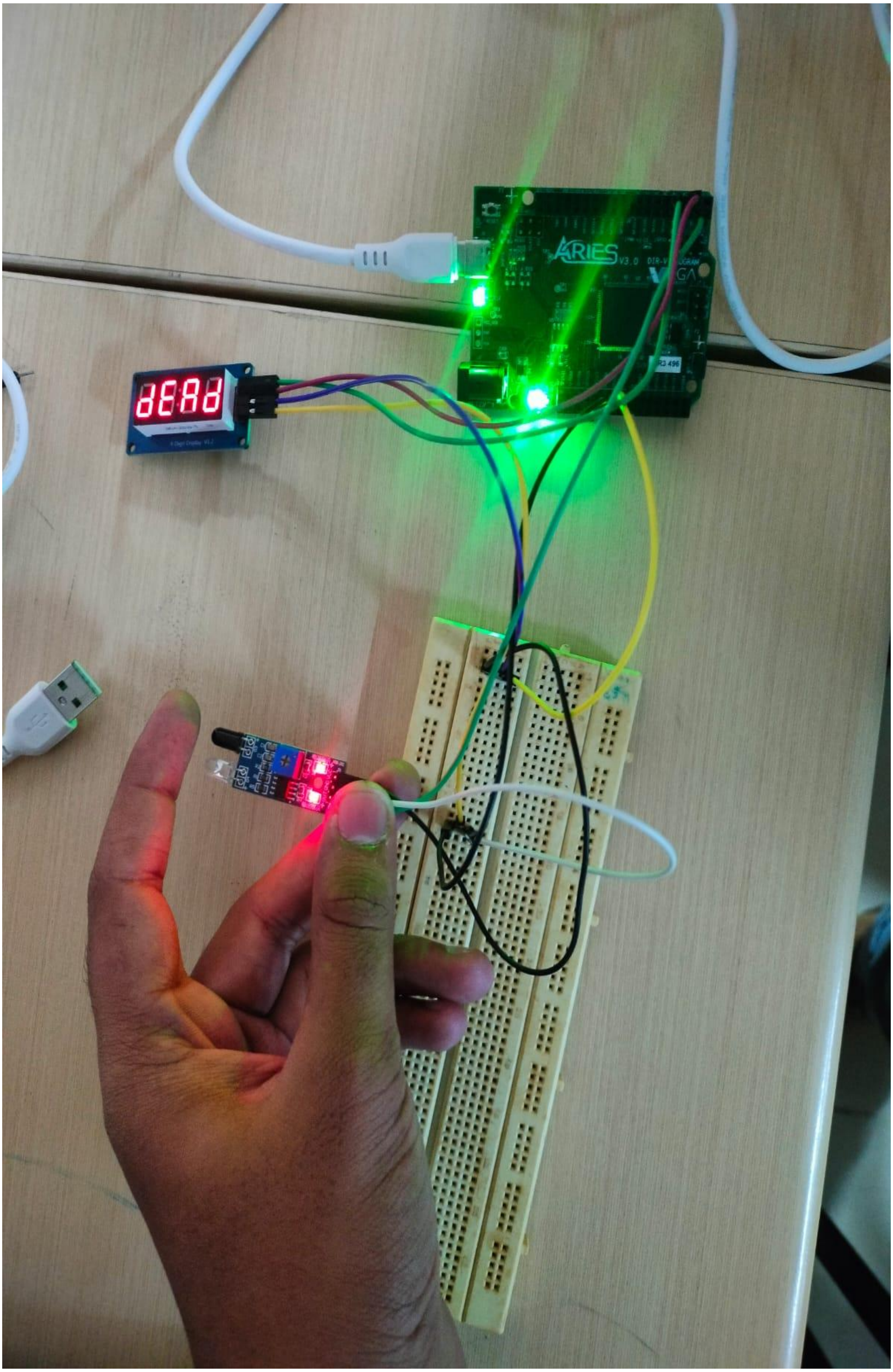
VCC → 3V3

GND → GND

OUT → GPIO 4

SNAPSHOTS OF THE OUTPUT




dEAd
ARIES
V3.0
VEGA

CODE

```
#include <TM1637.h>


int IRSensor = 4;

int LED = 22;

int CLK = 1; //CLK of TM1637 is connected to GPIO-1 pin of Aries Board

int DIO = 0;  //DI0 of TM1637 is connected to GPIO-0 pin of Aries Board


TM1637 tm(CLK,DIO);


void setup() {

 pinMode (IRSensor, INPUT);


  tm.init();

  tm.set(2); //set brightness; 0-7

}


void loop() {

  int statusSensor = digitalRead (IRSensor);


  if (statusSensor == 1){

    digitalWrite(LED, HIGH); // LED OFF
    tm.display(0,5);
    tm.display(1,10);
    tm.display(2,15);
    tm.display(3,14);
  }
  else {
    digitalWrite(LED, LOW); // LED ON
    tm.display(0,13);
    tm.display(1,14);
    tm.display(2,10);
    tm.display(3,13);
  }
}
```

ACTIVITY 02
OBJECT COUNTER

PROBLEM STATEMENT

- Use the IR sensor to detect objects passing in front of it.
- Display the count on the 4-digit 7-segment display
- Reset the count when push button is pressed

COMPONENTS USED

Aries Development Board v3, USB Cable, IR sensor, 4-bit display, jumper wires

PIN CONNECTIONS

Seven segment display:

DIO → GPIO 0

CLK → GPIO 1

VCC → 3V3

GND → GND

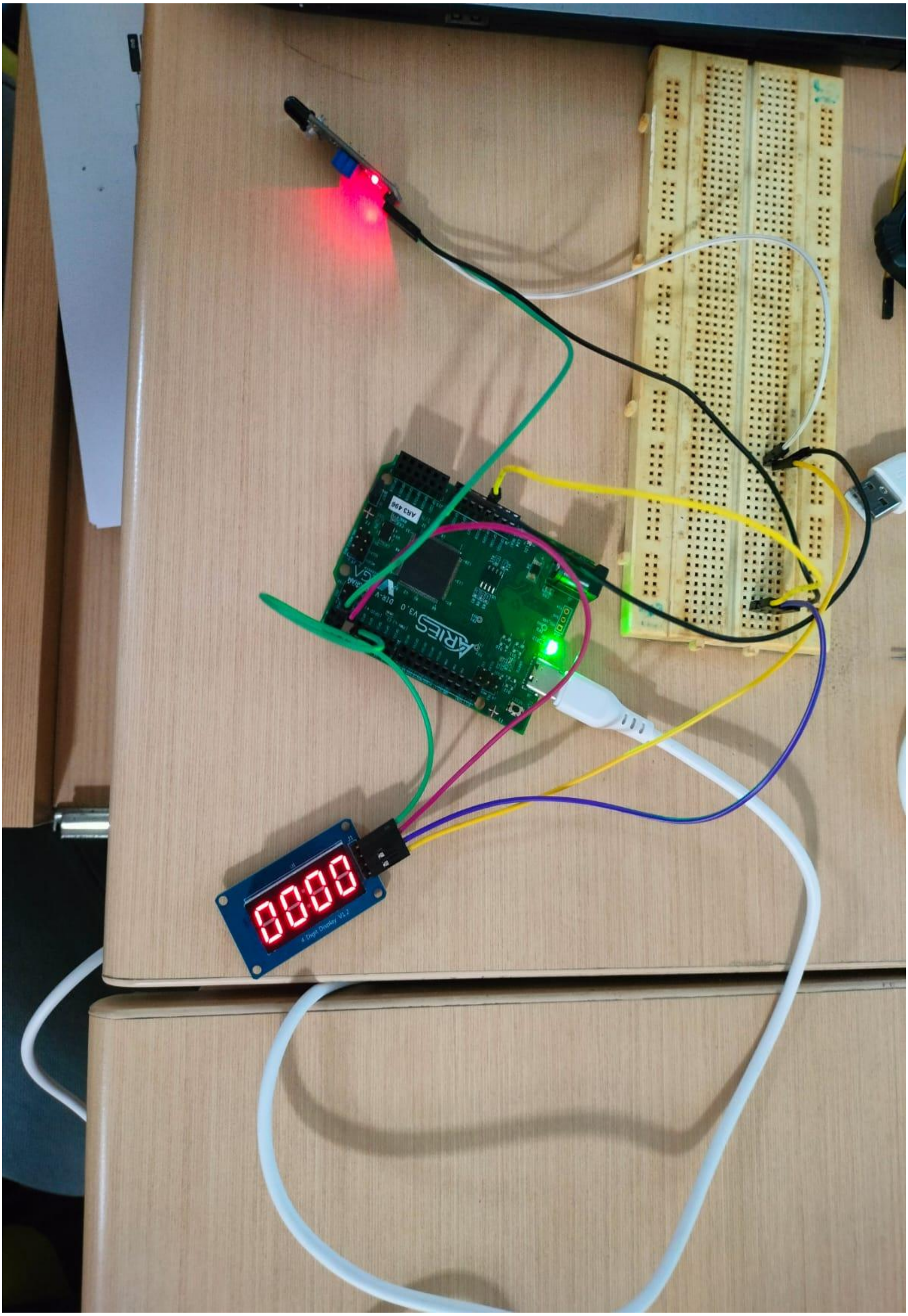
IR Sensor:

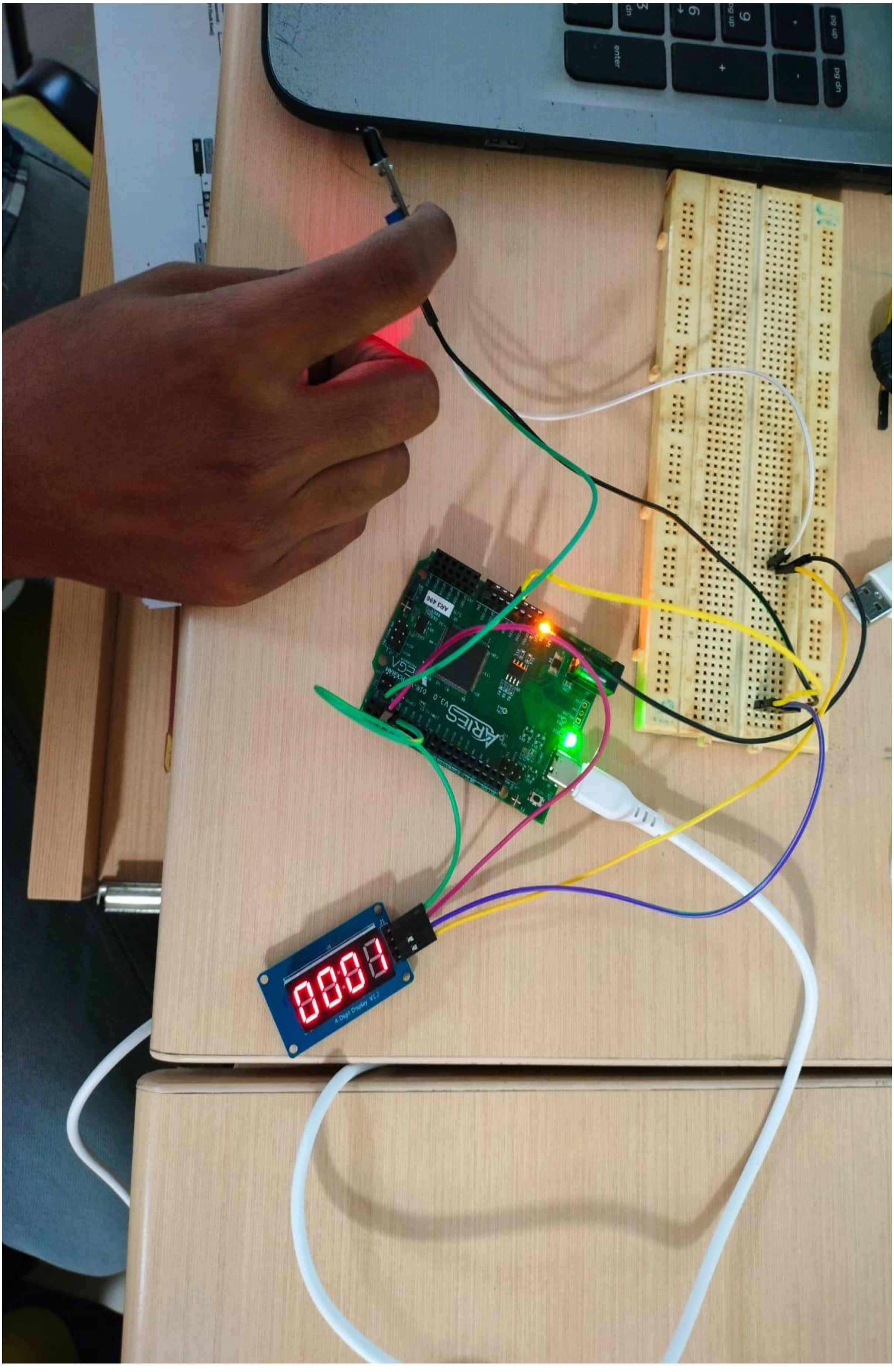
VCC → 3V3

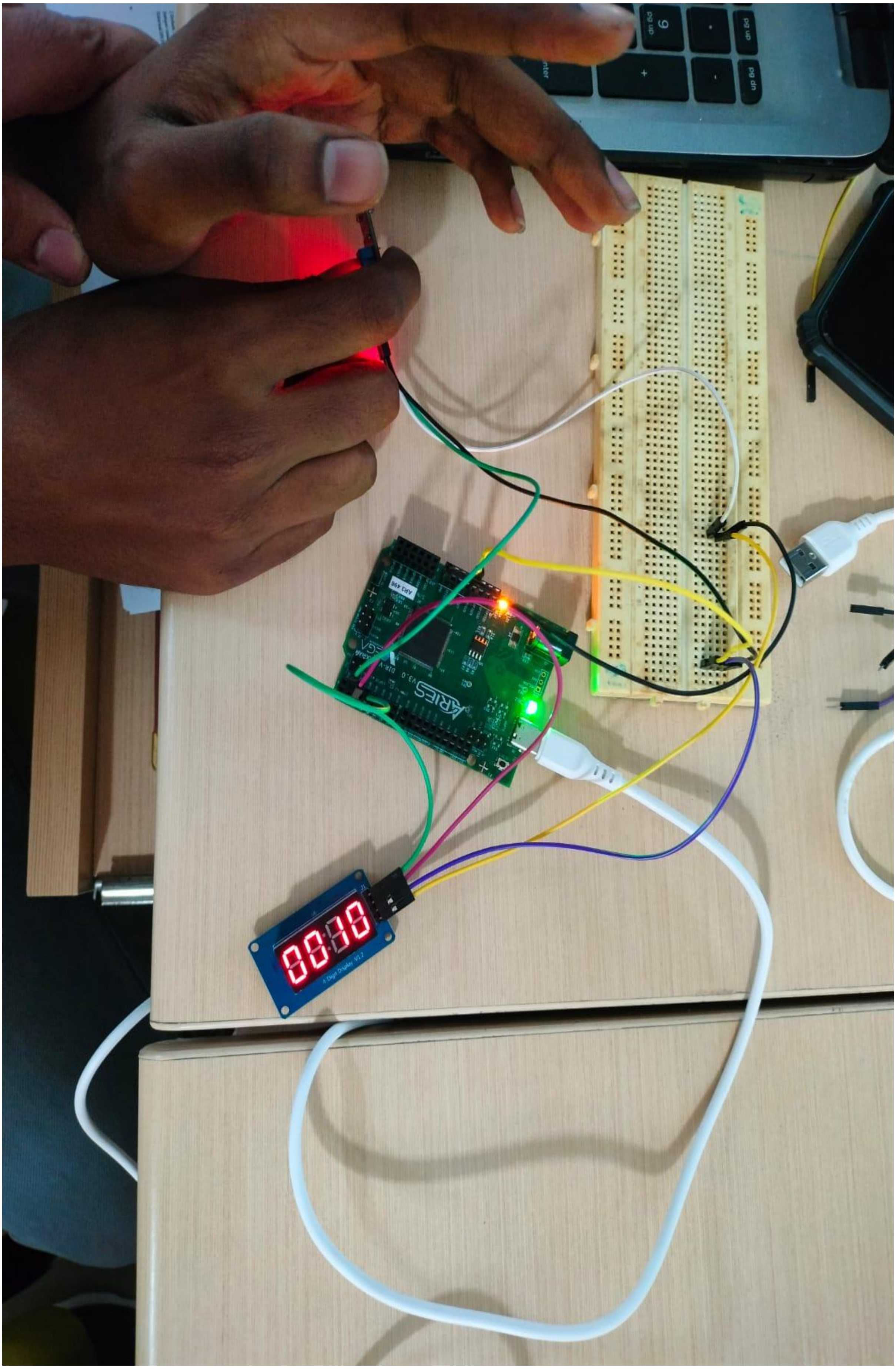
GND → GND

OUT → GPIO 4

SNAPSHOTS OF THE OUTPUT




0001


ARIES
V3.0
0010

CODE

```
#include <TM1637.h>
int IRSensor = 4;
int CLK = 1;  // CLK of TM1637 is connected to GPIO-1 pin of Aries Board
int DIO = 0;  // DIO of TM1637 is connected to GPIO-0 pin of Aries Board

TM1637 tm(CLK, DIO);
int count = 0;
int lastState = HIGH;   // Assume no object detected at start
void setup() {
  pinMode(IRSensor, INPUT);
  tm.init();
  tm.set(2);
  updateDisplay(count);
}

void loop() {
  delay(100);
  int statusSensor = digitalRead(IRSensor);
  if (statusSensor == LOW && lastState == HIGH) {
    count++;
    updateDisplay(count);
  }
  lastState = statusSensor;
}
// Function to update 4-digit 7-segment display
void updateDisplay(int num) {
  tm.display(3, num % 10);
  tm.display(2, (num / 10) % 10);
  tm.display(1, (num / 100) % 10);
  tm.display(0, (num / 1000) % 10);
}
```

ACTIVITY 03
SLOT MACHINE GAME WITH IR TRIGGER

PROBLEM STATEMENT

- When the IR sensor detects a hand movement, generate random numbers (0000-9999).
- Display the result on the 7-segment display.
- Flash LEDs for a “winning” number.

COMPONENTS USED

Aries Development Board v3, USB Cable, IR sensor, 4-bit display, jumper wires

PIN CONNECTIONS

Seven segment display:

DIO → GPIO 0

CLK → GPIO 1

VCC → 3V3

GND → GND

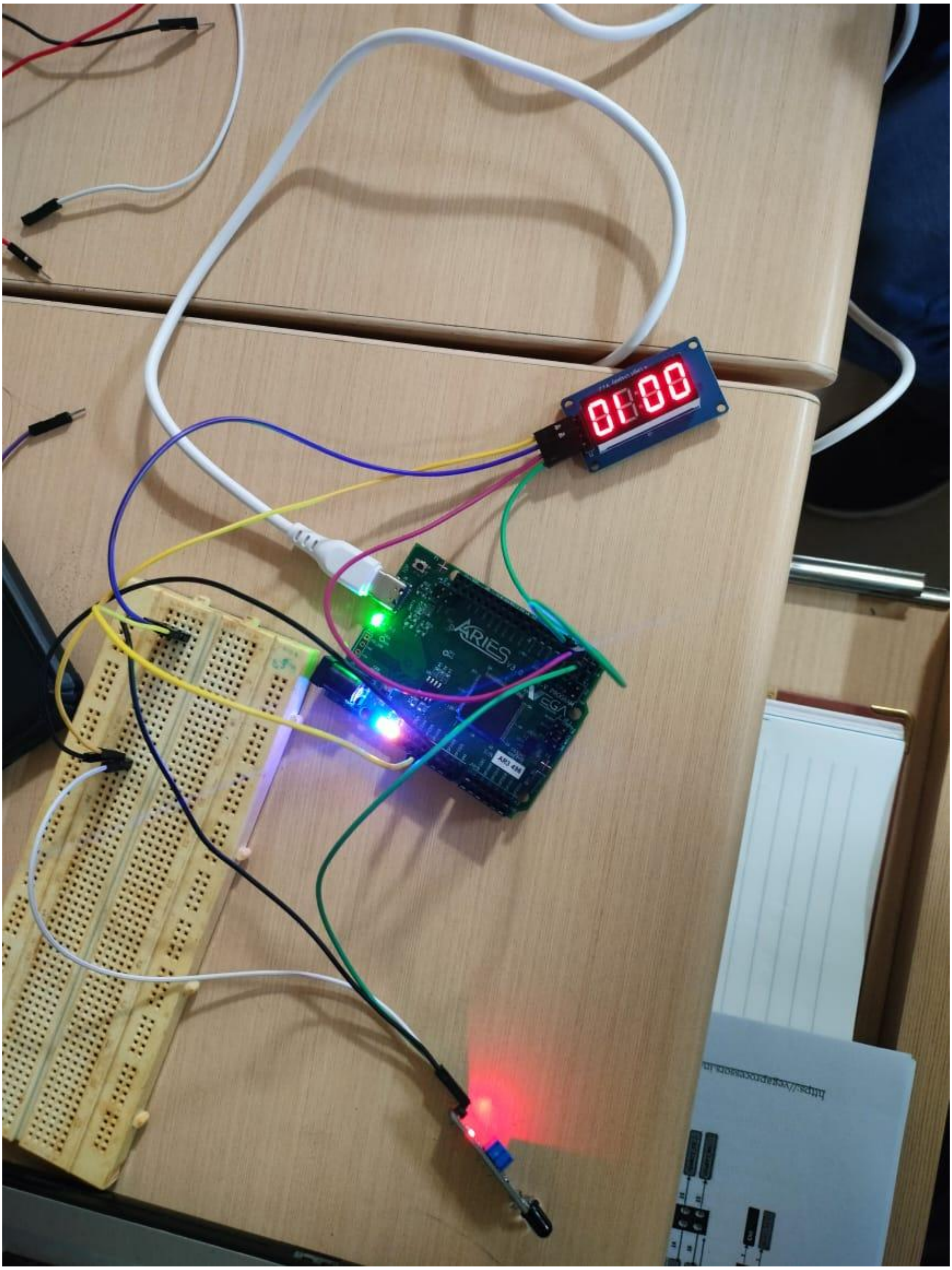
IR Sensor:

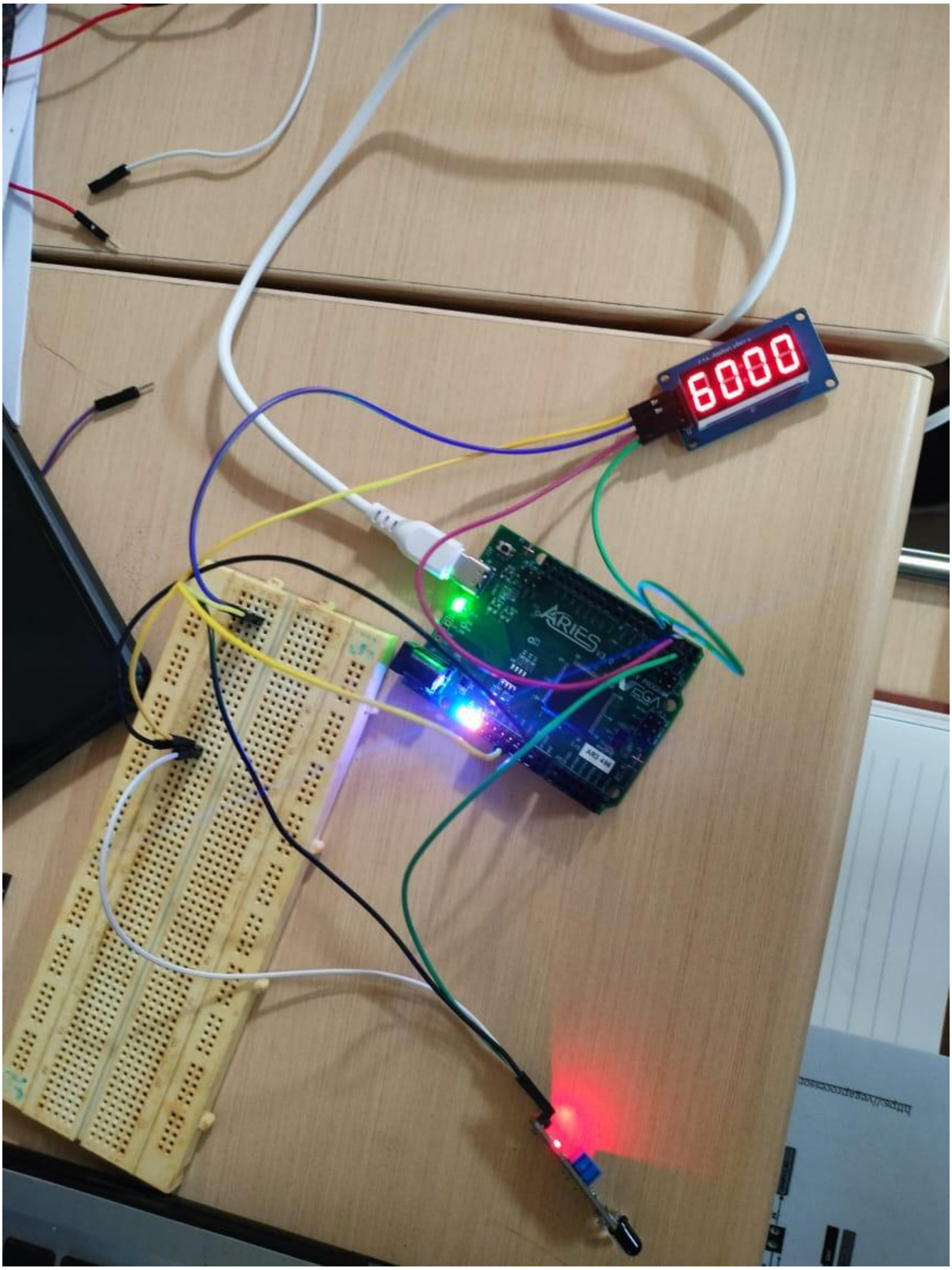
VCC → 3V3

GND → GND

OUT → GPIO 4

SNAPSHOTS OF THE OUTPUT




6000
ARIES
VEGA

CODE

```
#include <TM1637.h>

int IRSensor = 4;
int LED = 23;
int CLK = 1;       // TM1637 Clock Pin
int DIO = 0;       // TM1637 Data Pin

TM1637 tm(CLK, DIO);

int lastState = HIGH;

void setup() {
  pinMode(IRSensor, INPUT);
  pinMode(LED, OUTPUT);
  digitalWrite(LED, LOW);

  tm.init();
  tm.set(5);
  updateDisplay(0000);
```

```
}

void loop() {
  int statusSensor = digitalRead(IRSensor);

  if (statusSensor == LOW && lastState == HIGH) {
    int slotNumber = random(0, 11);
    updateDisplay(slotNumber);

    if (slotNumber == 10) {
      flashWinLED();
    }
  }

  lastState = statusSensor;
}

void updateDisplay(int num) {
  tm.display(3, num % 10);          // Ones place
  tm.display(2, (num / 10) % 10);   // Tens place
  tm.display(1, (num / 100) % 10);  // Hundreds place
  tm.display(0, (num / 1000) % 10); // Thousands place
```

```
}

void flashWinLED() {
  for (int i = 0; i < 5; i++) {
    digitalWrite(LED, HIGH);
    delay(300);
    digitalWrite(LED, LOW);
    delay(300);
  }
}
```

ACTIVITY 04A
BUZZER TUNES

PROBLEM STATEMENT

- Create a simple tune using Buzzer and delays
- Play "Twinkle Twinkle little star" with buzzers.
- Play a random song by defining the frequencies and notes.

COMPONENTS USED

Aries Development Board v3, USB Cable, Buzzer, jumper wires

PROBLEM STATEMENT 1: BASIC BUZZER

PIN CONNECTIONS

Buzzer:

VCC → 3V3

GND → GND

IN → GPIO 0

CODE

```
#define BUZZER 1 // connect INPUT pin of buzzer to GPIO-0


// the setup function runs once when you press reset or power the board
```

```
void setup() {
  // initialize digital pin 0 as an output.
  pinMode(BUZZER, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
  // switching buzzer on and off rapidly
  digitalWrite(BUZZER, HIGH); //turn on the buzzer
  delay(250);
  digitalWrite(BUZZER, LOW);  //turn off the buzzer
  delay(750);
  digitalWrite(BUZZER, HIGH); //turn on the buzzer
  delay(125);
  digitalWrite(BUZZER, LOW);  //turn off the buzzer
  delay(125);
  digitalWrite(BUZZER, HIGH); //turn on the buzzer
  delay(125);
  digitalWrite(BUZZER, LOW);  //turn off the buzzer
  delay(125);
}
```

PROBLEM STATEMENT 2 : TWINKLE TWINKLE ON PIEZO BUZZER

PIN CONNECTIONS

Buzzer:

VCC → 3V3

GND → GND

IN → GPIO 0

CODE

```
/*
  @file PIEZO_Buzzer.ino
  @brief Play "Twinkle Twinkle Little Star" on Piezo Buzzer
  @detail Using different frequencies to create musical notes

  Useful Links:
    Official Site: https://vegaprocessors.in/
    Development Boards: https://vegaprocessors.in/devboards/
    Blogs : https://vegaprocessors.in/blog/

   *** Piezoelectric buzzer ***
   Connections:
   Buzzer    Aries Board
   VCC     -   3.3V
  GND     -   GND

  IN      -   GPIO0

*/


#define BUZZER 0 // connect INPUT pin of buzzer to GPIO-0


// Define frequencies for musical notes

#define NOTE_C4  262

#define NOTE_D4  294

#define NOTE_E4  330

#define NOTE_F4  349

#define NOTE_G4  392

#define NOTE_A4  440

#define NOTE_B4  494

#define NOTE_C5  523


// Twinkle Twinkle Little Star melody

int melody[] = {

  NOTE_C4, NOTE_C4, NOTE_G4, NOTE_G4, NOTE_A4, NOTE_A4,
NOTE_G4,

  NOTE_F4, NOTE_F4, NOTE_E4, NOTE_E4, NOTE_D4, NOTE_D4,
NOTE_C4
```

```
};


// Note durations in milliseconds
int durations[] = {
  250, 250, 250, 250, 250, 250, 500,
  250, 250, 250, 250, 250, 250, 500
};


// Define the number of notes in our melody
const int noteCount = 14;


void setup() {
  pinMode(BUZZER, OUTPUT);
}


void loop() {
  // Play the melody once
  for (int i = 0; i < noteCount; i++) {
    playTone(melody[i], durations[i]);


    // Brief pause between notes
    delay(50);
  }


  // Pause before playing again
  delay(1000);
}


// Function to play a tone of specific frequency and duration
void playTone(int frequency, int duration) {
  // For very low frequencies, just use a simple delay approach
  long period = 1000000 / frequency;
  long elapsedTime = 0;


  while (elapsedTime < duration * 1000) {
    digitalWrite(BUZZER, HIGH);
    delayMicroseconds(period / 2);
    digitalWrite(BUZZER, LOW);
    delayMicroseconds(period / 2);
    elapsedTime += period;
  }
}
```

PROBLEM 04C : PLAYING A RANDOM SONG

PIN CONNECTIONS

Buzzer:

VCC → 3V3

GND → GND

IN → GPIO 1

CODE

```
#define BUZZER 1 // Connect INPUT pin of buzzer to GPIO-1


// Define frequencies for musical notes
#define NOTE_C4  262
#define NOTE_Cs4 277
#define NOTE_D4  294
#define NOTE_Ds4 311
#define NOTE_E4  330
#define NOTE_F4  349
#define NOTE_Fs4 370
#define NOTE_Gs4 415
#define NOTE_A4  440
#define NOTE_As4 466
#define NOTE_B4  494
#define NOTE_C5  523
```

```
// Melody sequence
int melody[] = {
  NOTE_A4, NOTE_B4, NOTE_A4, NOTE_Gs4, NOTE_A4,
  NOTE_A4, NOTE_B4, NOTE_A4, NOTE_Gs4, NOTE_A4,
  NOTE_A4, NOTE_B4, NOTE_C5, NOTE_B4, NOTE_A4,
  NOTE_A4, NOTE_B4, NOTE_A4, NOTE_Gs4, NOTE_Fs4,

  NOTE_A4, NOTE_B4, NOTE_A4, NOTE_Gs4, NOTE_A4,
  NOTE_A4, NOTE_B4, NOTE_A4, NOTE_Gs4, NOTE_A4,
  NOTE_A4, NOTE_B4, NOTE_C5, NOTE_B4, NOTE_A4,
  NOTE_A4, NOTE_B4, NOTE_A4, NOTE_Gs4, NOTE_Fs4,

  NOTE_E4, NOTE_Fs4, NOTE_Fs4, NOTE_Gs4, NOTE_Gs4,
  NOTE_Gs4, NOTE_Fs4, NOTE_Fs4, NOTE_Gs4, NOTE_Gs4,
  NOTE_A4, NOTE_A4, NOTE_B4, NOTE_B4, NOTE_C5,

  NOTE_E4, NOTE_Fs4, NOTE_Fs4, NOTE_Gs4, NOTE_Gs4,
  NOTE_Gs4, NOTE_Fs4, NOTE_Fs4, NOTE_Gs4, NOTE_Gs4,
  NOTE_A4, NOTE_A4, NOTE_B4, NOTE_B4, NOTE_C5
};

// Adjusted durations for a faster tempo
```

```
int durations[] = {
  100, 100, 100, 100, 200,
  100, 100, 100, 100, 200,
  100, 100, 100, 100, 200,
  100, 100, 100, 100, 200,

  100, 100, 100, 100, 200,
  100, 100, 100, 100, 200,
  100, 100, 100, 100, 200,
  100, 100, 100, 100, 200,

  100, 200, 100, 200, 300,
  100, 100, 100, 200, 100,
  200, 100, 200, 100, 200,

  100, 200, 100, 200, 300,
  100, 100, 100, 200, 100,
  200, 100, 200, 100, 200,
};

// Number of notes in the melody
const int noteCount = sizeof(melody) / sizeof(melody[0]);
```

```
void setup() {
  pinMode(BUZZER, OUTPUT);
}

void loop() {
  // Play the melody
  for (int i = 0; i < noteCount; i++) {
    playTone(melody[i], durations[i]);

    // Shorter pause between notes
    delay(30);
  }

  // Shorter pause before repeating
  delay(500);
}

// Function to play a tone with specific frequency and duration
void playTone(int frequency, int duration) {
  long period = 1000000 / frequency;
  long elapsedTime = 0;
  while (elapsedTime < duration * 1000) {
    digitalWrite(BUZZER, HIGH);
    delayMicroseconds(period / 2);
    digitalWrite(BUZZER, LOW);
    delayMicroseconds(period / 2);
    elapsedTime += period;
  }
}
```

ACTIVITY 04B
Ultrasonic Triggered Beats

PROBLEM STATEMENT

- Create a system of two ultrasonic sensors, RYG LED strip and Piezo Buzzer
- Each ultrasonic sensor triggers its corresponding LED and unique buzzer tune when the distance is less than 10cm.

COMPONENTS USED

Aries Development Board v3, USB Cable, Buzzer, jumper wires, RYG LED set, Two ultrasonic sensors

PIN CONNECTIONS

Ultrasonic Sensor 2:

VCC - 5V, GND - GND, Trig - GPIO3, Echo - GPIO4

Ultrasonic Sensor 3:

VCC - 5V, GND - GND, Trig - GPIO5, Echo - GPIO6

LEDs:

GND - GND

YELLOW - GPIO8

GREEN - GPIO7

Piezoelectric buzzer:

VCC - 3.3V

GND - GND

IN - GPIO15

CODE

```
// Define Ultrasonic Sensor pins

#define TRIG_PIN2 3

#define ECHO_PIN2 4

#define TRIG_PIN3 5

#define ECHO_PIN3 6


// Define LED pins

#define YELLOW_LED 8

#define GREEN_LED 7


// Define Buzzer pin
```

```
#define BUZZER 12

// Variables for distance measurement
long duration2, duration3;
int distance2, distance3;
int threshold = 9; // Threshold distance in cm
int minValidDistance = 2; // Minimum valid distance (cm) to filter out false
readings
boolean specialMode = false; // For special patterns when multiple sensors
are triggered

// Variables for buzzer control
unsigned long lastBuzzerTime = 0;
int currentTune = 0; // 0: no sound, 2: tune2, 3: tune3, 4: special tune

void setup() {
  // Initialize ultrasonic sensor pins
  pinMode(TRIG_PIN2, OUTPUT);
  pinMode(ECHO_PIN2, INPUT);
  pinMode(TRIG_PIN3, OUTPUT);
  pinMode(ECHO_PIN3, INPUT);
  // Initialize LED pins
  pinMode(YELLOW_LED, OUTPUT);
  pinMode(GREEN_LED, OUTPUT);

  // Initialize buzzer pin
  pinMode(BUZZER, OUTPUT);
  digitalWrite(BUZZER, LOW);

  // Turn off all LEDs initially
  digitalWrite(YELLOW_LED, LOW);
  digitalWrite(GREEN_LED, LOW);

  // Test the buzzer
  playTone(100, 3);

  // Initialize serial communication
  Serial.begin(115200);
  Serial.println("Two Ultrasonic Sensors Controlling LEDs with Buzzer Feedback");
  Serial.println("Place hand in front of any sensor within 10cm to trigger its LED and buzzer");
  Serial.println("Buzzer test complete");
```

```
}

// Function to measure distance from an ultrasonic sensor
int measureDistance(int trigPin, int echoPin) {
  // Clear the trigPin
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);

  // Set trigPin high for 10 microseconds
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);

  // Read the echoPin, returns sound wave travel time in microseconds
  long duration = pulseIn(echoPin, HIGH, 25000); // Add timeout of 25ms

  // Check if reading timed out (returned 0)
  if (duration == 0) {
    return 400; // Return a large value (no object detected)
  }

  // Calculate distance
  int distance = duration * 0.034 / 2; // Speed of sound wave divided by 2 (go and back)

  // Filter out unreasonable values
  if (distance < minValidDistance || distance > 400) {
    return 400; // Return a large value for invalid readings
  }

  return distance;
}

void allLedsOff() {
  digitalWrite(YELLOW_LED, LOW);
  digitalWrite(GREEN_LED, LOW);
}

// Function to play a single tone
void playTone(int duration, int repetitions) {
  for (int i = 0; i < repetitions; i++) {
    digitalWrite(BUZZER, HIGH);
    delay(duration);
    digitalWrite(BUZZER, LOW);
    delay(duration);
  }
}

// Function for Sensor 2 tune (Yellow LED) - Double beep pattern
void playTune2() {
  unsigned long currentTime = millis();
  if (currentTime - lastBuzzerTime > 500) {
    lastBuzzerTime = currentTime;
    digitalWrite(BUZZER, HIGH);
    delay(80);
    digitalWrite(BUZZER, LOW);
    delay(80);
    digitalWrite(BUZZER, HIGH);
    delay(80);
    digitalWrite(BUZZER, LOW);
  }
}

// Function for Sensor 3 tune (Green LED) - Long single beep
void playTune3() {
  unsigned long currentTime = millis();
  if (currentTime - lastBuzzerTime > 800) {
    lastBuzzerTime = currentTime;
    digitalWrite(BUZZER, HIGH);
    delay(200);
    digitalWrite(BUZZER, LOW);
  }
}

// Function for special tune (All sensors) - Victory melody
void playSpecialTune() {
  unsigned long currentTime = millis();
  if (currentTime - lastBuzzerTime > 1000) {
    lastBuzzerTime = currentTime;
    // Play ascending notes
    for (int i = 50; i <= 150; i += 25) {
      digitalWrite(BUZZER, HIGH);
      delay(i);
      digitalWrite(BUZZER, LOW);
      delay(50);
    }
  }
}
```

```
void loop() {
  // Measure distance from each sensor
  distance2 = measureDistance(TRIG_PIN2, ECHO_PIN2);
  distance3 = measureDistance(TRIG_PIN3, ECHO_PIN3);

  // Print distances to serial monitor for debugging
  Serial.print("Distance 2: ");
  Serial.print(distance2);
  Serial.print(" cm | Distance 3: ");
  Serial.print(distance3);
  Serial.println(" cm");

  // Turn off all LEDs first
  allLedsOff();

  // Reset buzzer state
  currentTune = 0;

  // Check if both sensors are triggered simultaneously
  if ((distance2 < threshold && distance2 >= minValidDistance) &&
      (distance3 < threshold && distance3 >= minValidDistance)) {
```

```
    specialMode = true;
    // Special mode: all LEDs on steadily
    digitalWrite(YELLOW_LED, HIGH);
    digitalWrite(GREEN_LED, HIGH);
    // Play special tune
    currentTune = 4;
    Serial.println("SPECIAL MODE: Both sensors triggered!");
  }
  else {
    specialMode = false;

    // Check sensor 2 (Yellow LED)
    if (distance2 < threshold && distance2 >= minValidDistance) {
      digitalWrite(YELLOW_LED, HIGH);
      currentTune = 2; // Set tune for sensor 2
      Serial.println("YELLOW LED ON - Sensor 2 triggered");
    }
    // Check sensor 3 (Green LED)
    else if (distance3 < threshold && distance3 >= minValidDistance) {
      digitalWrite(GREEN_LED, HIGH);
      currentTune = 3; // Set tune for sensor 3
      Serial.println("GREEN LED ON - Sensor 3 triggered");
    }
  }
  // Play the selected tune
  switch(currentTune) {
    case 2:
      playTune2();
      break;
    case 3:
      playTune3();
      break;
    case 4:
      playSpecialTune();
      break;
    default:
      // No sound
      digitalWrite(BUZZER, LOW);
      break;
  }

  // Short delay before next reading
  delay(20); // Reduced delay for better tune responsiveness
}
```

ACTIVITY 04C
Ultrasonic and IR based Music system

PROBLEM STATEMENT

A) Uses two ultrasonic sensors for notes and an IR sensor to switch between music modes – normal and disco.

B) Same as Problem 6A but uses two separate buzzers for generating the music.

COMPONENTS USED

Aries Development Board v3, USB Cable, Two Buzzers, jumper wires, RYG LED set, Two ultrasonic sensors

PROBLEM 06A : USING ONE BUZZER

PIN CONNECTIONS

Ultrasonic Sensor 2:

VCC - 5V, GND - GND, Trig - GPIO3, Echo - GPIO4

Ultrasonic Sensor 3:

VCC - 5V, GND - GND, Trig - GPIO5, Echo - GPIO6

IR Sensor:

VCC - 5V, GND - GND, OUT - GPIO11

LEDs:

GND - GND

RED - GPIO9

YELLOW - GPIO8

GREEN - GPIO7

Piezoelectric buzzer:

VCC - 3.3V

GND - GND

IN - GPIO12

CODE

```
// Define Ultrasonic Sensor pins
#define TRIG_PIN2 3
#define ECHO_PIN2 4
#define TRIG_PIN3 5
#define ECHO_PIN3 6

// Define IR sensor pin
#define IR_SENSOR 11
```

```
// Define LED pins
#define RED_LED 9    // Add back the Red LED
#define YELLOW_LED 8
#define GREEN_LED 7

// Define Buzzer pin
#define BUZZER 1

// Variables for distance measurement
long duration2, duration3;
int distance2, distance3;
int threshold = 9; // Threshold distance in cm
int minValidDistance = 2; // Minimum valid distance (cm) to filter out false
readings

// Variables for mode control and timing
boolean irDetected = false;
unsigned long lastLEDToggle = 0;
unsigned long lastArpeggioChange = 0;
boolean ledState = false;
unsigned long discoInterval = 100;
unsigned long arpeggioInterval = 150;
int activeNote = 0; // 0: no note, 1: note from sensor 2, 2: note from sensor 3

// Variable for disco mode LED sequencing
int discoStep = 0;  // 0: Red, 1: Yellow, 2: Green
unsigned long lastDiscoStep = 0;
unsigned long discoStepInterval = 150;  // Time between LED changes in disco mode

// Variables for pulsing tones when IR detected
unsigned long lastPulseChange = 0;
boolean pulseState = false;
unsigned long pulseOnTime = 200;   // Tone on time
unsigned long pulseOffTime = 100;  // Tone off time

// Variables for tone frequencies
int freq2 = 500;  // Lower frequency for sensor 2 (Hz)
int freq3 = 1000; // Higher frequency for sensor 3 (Hz)

void setup() {
  // Initialize ultrasonic sensor pins
  pinMode(TRIG_PIN2, OUTPUT);
  pinMode(ECHO_PIN2, INPUT);

  pinMode(TRIG_PIN3, OUTPUT);

  pinMode(ECHO_PIN3, INPUT);


  // Initialize IR sensor pin

  pinMode(IR_SENSOR, INPUT);


  // Initialize LED pins

  pinMode(RED_LED, OUTPUT);    // Add Red LED initialization

  pinMode(YELLOW_LED, OUTPUT);

  pinMode(GREEN_LED, OUTPUT);


  // Initialize buzzer pin

  pinMode(BUZZER, OUTPUT);

  digitalWrite(BUZZER, LOW);


  // Turn off all LEDs initially

  digitalWrite(RED_LED, LOW);   // Add Red LED

  digitalWrite(YELLOW_LED, LOW);

  digitalWrite(GREEN_LED, LOW);


  // Initialize serial communication
```

```
  Serial.begin(115200);
  Serial.println("Enhanced Interactive Musical Light System");
  Serial.println("Two modes based on IR sensor detection:");
  Serial.println("1) IR detected: Pulsing tones and disco light sequence");
  Serial.println("2) IR not detected: Steady tones with corresponding LEDs");

  // Startup sequence to test components
  testComponents();
}

void testComponents() {
  // Test LEDs in sequence
  digitalWrite(RED_LED, HIGH);
  delay(200);
  digitalWrite(RED_LED, LOW);
  digitalWrite(YELLOW_LED, HIGH);
  delay(200);
  digitalWrite(YELLOW_LED, LOW);
  digitalWrite(GREEN_LED, HIGH);
  delay(200);
  digitalWrite(GREEN_LED, LOW);
  // Test buzzer with quick ascending notes
  for (int i = 100; i <= 300; i += 100) {
    digitalWrite(BUZZER, HIGH);
    delay(i);
    digitalWrite(BUZZER, LOW);
    delay(50);
  }
}

// Function to measure distance from an ultrasonic sensor
int measureDistance(int trigPin, int echoPin) {
  // Clear the trigPin
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);

  // Set trigPin high for 10 microseconds
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);

  // Read the echoPin, returns sound wave travel time in microseconds
  long duration = pulseIn(echoPin, HIGH, 25000); // Add timeout of 25ms
  // Check if reading timed out (returned 0)
  if (duration == 0) {
    return 400; // Return a large value (no object detected)
  }

  // Calculate distance
  int distance = duration * 0.034 / 2; // Speed of sound wave divided by 2 (go and back)

  // Filter out unreasonable values
  if (distance < minValidDistance || distance > 400) {
    return 400; // Return a large value for invalid readings
  }

  return distance;
}

// Function to play a tone with specific frequency
void playTone(int frequency, int duration) {
  // Simple tone generation (approximation)
  // More precise frequencies would require a library or PWM
  int period = 1000000 / frequency; // Period in microseconds
  for (long i = 0; i < duration * 1000L / period; i++) {
    digitalWrite(BUZZER, HIGH);
    delayMicroseconds(period / 2);
    digitalWrite(BUZZER, LOW);
    delayMicroseconds(period / 2);
  }
}

// Function to turn off all LEDs
void allLedsOff() {
  digitalWrite(RED_LED, LOW);
  digitalWrite(YELLOW_LED, LOW);
  digitalWrite(GREEN_LED, LOW);
}

// Function to run the disco light sequence (when IR detects)
void runDiscoSequence() {
  unsigned long currentMillis = millis();

  if (currentMillis - lastDiscoStep > discoStepInterval) {
    lastDiscoStep = currentMillis;
    // Turn off all LEDs
    allLedsOff();

    // Move to next LED in sequence
    discoStep = (discoStep + 1) % 3;

    // Turn on the current LED in the sequence
    switch (discoStep) {
      case 0:
        digitalWrite(RED_LED, HIGH);
        break;
      case 1:
        digitalWrite(YELLOW_LED, HIGH);
        break;
      case 2:
        digitalWrite(GREEN_LED, HIGH);
        break;
    }
  }
}
```

```
// Function to generate pulsed tones based on active sensor(s)
void playPulsedTones() {
  unsigned long currentMillis = millis();

  // Determine which tone to play based on detected sensors
  if (distance2 < threshold && distance2 >= minValidDistance &&
      distance3 < threshold && distance3 >= minValidDistance) {
    // Both sensors - play alternating tones
    if (currentMillis - lastPulseChange > pulseOffTime) {
      if (!pulseState) {
        playTone(freq2, 50);  // Play short tone at frequency 2
        pulseState = true;
        lastPulseChange = currentMillis;
      } else {
        playTone(freq3, 50);  // Play short tone at frequency 3
        pulseState = false;
        lastPulseChange = currentMillis;
      }
    }
  }
  else if (distance2 < threshold && distance2 >= minValidDistance) {
    // Sensor 2 - pulse at lower frequency
    if (currentMillis - lastPulseChange > (pulseState ? pulseOnTime :
pulseOffTime)) {
      pulseState = !pulseState;
      lastPulseChange = currentMillis;

      if (pulseState) {
        playTone(freq2, 50);  // Play short tone at frequency 2
      }
    }
  }
  else if (distance3 < threshold && distance3 >= minValidDistance) {
    // Sensor 3 - pulse at higher frequency
    if (currentMillis - lastPulseChange > (pulseState ? pulseOnTime :
pulseOffTime)) {
      pulseState = !pulseState;
      lastPulseChange = currentMillis;

      if (pulseState) {
        playTone(freq3, 50);  // Play short tone at frequency 3
      }
    }
  }
  else {
    // No sensors - silence
    digitalWrite(BUZZER, LOW);
  }
}

void loop() {
  // Measure distances from ultrasonic sensors
  distance2 = measureDistance(TRIG_PIN2, ECHO_PIN2);
  distance3 = measureDistance(TRIG_PIN3, ECHO_PIN3);

  // Check IR sensor - NOTE: Logic is now inverted from previous implementation
  irDetected = digitalRead(IR_SENSOR) == LOW; // Assuming IR sensor outputs LOW when object detected

  // Print debug info
  Serial.print("Distance 2: ");
  Serial.print(distance2);
  Serial.print(" cm | Distance 3: ");
  Serial.print(distance3);
  Serial.print(" cm | IR: ");
Serial.println(irDetected ? "Detected" : "Not Detected");

// Turn off all LEDs initially
allLedsOff();

// MODE 1: IR object detected - Disco lights and pulsed tones
if (irDetected) {
  // Run disco light sequence
  runDiscoSequence();

  // Play pulsed tones based on which sensors are triggered
  playPulsedTones();

  Serial.println("MODE 1: Disco lights and pulsed tones");
}
// MODE 2: No IR object - Steady lights and continuous tones
else {
  // Check sensors and play appropriate continuous tones
  if (distance2 < threshold && distance2 >= minValidDistance &&
      distance3 < threshold && distance3 >= minValidDistance) {
    // Both sensors - turn on both LEDs, play higher frequency
    digitalWrite(YELLOW_LED, HIGH);
    digitalWrite(GREEN_LED, HIGH);
    playTone(freq3, 50);  // Play higher frequency
    Serial.println("MODE 2: Both LEDs ON - Playing higher tone");
  }
  else if (distance2 < threshold && distance2 >= minValidDistance) {
    // Sensor 2 only - turn on Yellow LED, play lower frequency
    digitalWrite(YELLOW_LED, HIGH);
    playTone(freq2, 50);  // Play lower frequency
    Serial.println("MODE 2: YELLOW LED ON - Playing lower tone");
  }
  else if (distance3 < threshold && distance3 >= minValidDistance) {
    // Sensor 3 only - turn on Green LED, play higher frequency
    digitalWrite(GREEN_LED, HIGH);
    playTone(freq3, 50);  // Play higher frequency
    Serial.println("MODE 2: GREEN LED ON - Playing higher tone");
  }
  else {
    // No sensors triggered - silence
    digitalWrite(BUZZER, LOW);
  }
}
```

```
  // Short delay for stability
  delay(5);
}
```

PROBLEM 06B : COMBINATION OF TWO BUZZERS

PIN CONNECTIONS

Ultrasonic Sensor 2:

VCC - 5V, GND - GND, Trig - GPIO3, Echo - GPIO4

Ultrasonic Sensor 3:

VCC - 5V, GND - GND, Trig - GPIO5, Echo - GPIO6

IR Sensor:

VCC - 5V, GND - GND, OUT - GPIO11

LEDs:

GND - GND

RED - GPIO9

YELLOW - GPIO8

GREEN - GPIO7

Piezoelectric buzzers:

VCC - 3.3V

GND - GND

IN - GPIO12, GPIO13

CODE

```
// Define Ultrasonic Sensor pins
#define TRIG_PIN2 3
#define ECHO_PIN2 4
#define TRIG_PIN3 5
#define ECHO_PIN3 6

// Define IR sensor pin
#define IR_SENSOR 11

// Define LED pins
#define RED_LED 9    // Add back the Red LED
#define YELLOW_LED 8
#define GREEN_LED 7

// Define Buzzer pins
#define BUZZER1 12  // First buzzer for Sensor 2
#define BUZZER2 13  // Second buzzer for Sensor 3
```

```
// Define musical notes for better melodies
#define NOTE_C4  262
#define NOTE_D4  294
#define NOTE_E4  330
#define NOTE_F4  349
#define NOTE_G4  392
#define NOTE_A4  440
#define NOTE_B4  494
#define NOTE_C5  523
#define NOTE_D5  587
#define NOTE_E5  659
#define NOTE_F5  698
#define NOTE_G5  784

// Melody for Sensor 2 (Yellow LED) - "Charge" fanfare
const int yellowMelodySize = 6;
const int yellowMelody[yellowMelodySize] = {NOTE_C4, NOTE_F4,
NOTE_G4, NOTE_A4, NOTE_F4, NOTE_C5};
const int yellowDurations[yellowMelodySize] = {100, 100, 100, 100, 100, 200};
int yellowNoteIndex = 0;
unsigned long lastYellowNoteTime = 0;
```

```
// Melody for Sensor 3 (Green LED) - "Star Wars" theme hint
const int greenMelodySize = 5;
const int greenMelody[greenMelodySize] = {NOTE_G4, NOTE_G4,
NOTE_G4, NOTE_D4, NOTE_B4};
const int greenDurations[greenMelodySize] = {130, 130, 130, 100, 200};
int greenNoteIndex = 0;
unsigned long lastGreenNoteTime = 0;

// Harmony patterns for dual buzzer mode
const int harmonySize = 4;
const int harmonyMelody1[harmonySize] = {NOTE_C4, NOTE_E4,
NOTE_G4, NOTE_C5};
const int harmonyMelody2[harmonySize] = {NOTE_E4, NOTE_G4,
NOTE_C5, NOTE_E5};
const int harmonyDurations[harmonySize] = {120, 120, 120, 200};
int harmonyIndex = 0;
unsigned long lastHarmonyTime = 0;

// Variables for distance measurement
long duration2, duration3;
int distance2, distance3;
int threshold = 9; // Threshold distance in cm
int minValidDistance = 2; // Minimum valid distance (cm) to filter out false readings

// Variables for mode control and timing
boolean irDetected = false;
unsigned long lastLEDToggle = 0;
unsigned long lastArpeggioChange = 0;
boolean ledState = false;
unsigned long discoInterval = 100;
unsigned long arpeggioInterval = 150;
int activeNote = 0; // 0: no note, 1: note from sensor 2, 2: note from sensor 3

// Variable for disco mode LED sequencing
int discoStep = 0;  // 0: Red, 1: Yellow, 2: Green
unsigned long lastDiscoStep = 0;
unsigned long discoStepInterval = 150;  // Time between LED changes in disco mode

// Variables for pulsing tones when IR detected
unsigned long lastPulseChange = 0;
boolean pulseState = false;
```

```
unsigned long pulseOnTime = 200;   // Tone on time
unsigned long pulseOffTime = 100;  // Tone off time

// Variables for tone frequencies
int freq2 = 500;  // Lower frequency for sensor 2 (Hz)
int freq3 = 1000; // Higher frequency for sensor 3 (Hz)

void setup() {
  // Initialize ultrasonic sensor pins
  pinMode(TRIG_PIN2, OUTPUT);
  pinMode(ECHO_PIN2, INPUT);
  pinMode(TRIG_PIN3, OUTPUT);
  pinMode(ECHO_PIN3, INPUT);

  // Initialize IR sensor pin
  pinMode(IR_SENSOR, INPUT);

  // Initialize LED pins
  pinMode(RED_LED, OUTPUT);    // Add Red LED initialization
  pinMode(YELLOW_LED, OUTPUT);
  pinMode(GREEN_LED, OUTPUT);
  // Initialize buzzer pins
  pinMode(BUZZER1, OUTPUT);
  pinMode(BUZZER2, OUTPUT);
  digitalWrite(BUZZER1, LOW);
  digitalWrite(BUZZER2, LOW);

  // Turn off all LEDs initially
  digitalWrite(RED_LED, LOW);   // Add Red LED
  digitalWrite(YELLOW_LED, LOW);
  digitalWrite(GREEN_LED, LOW);

  // Initialize serial communication
  Serial.begin(115200);
  Serial.println("Dual Buzzer Interactive Musical Light System");
  Serial.println("Two modes based on IR sensor detection:");
  Serial.println("1) IR detected: Disco lights and stereo music effects");
  Serial.println("2) IR not detected: Steady lights and dedicated buzzer
melodies");

  // Startup sequence to test components
  testComponents();
}
```

```
void testComponents() {
  // Test LEDs in sequence
  digitalWrite(RED_LED, HIGH);
  delay(200);
  digitalWrite(RED_LED, LOW);
  digitalWrite(YELLOW_LED, HIGH);
  delay(200);
  digitalWrite(YELLOW_LED, LOW);
  digitalWrite(GREEN_LED, HIGH);
  delay(200);
  digitalWrite(GREEN_LED, LOW);

  // Test both buzzers
  Serial.println("Testing Buzzer 1");
  playTone(BUZZER1, 440, 200);
  delay(300);

  Serial.println("Testing Buzzer 2");
  playTone(BUZZER2, 587, 200);
  delay(300);
  Serial.println("Testing Dual Buzzer Harmony");
  playDualTone(BUZZER1, 440, BUZZER2, 659, 300);
  delay(100);
}

// Function to measure distance from an ultrasonic sensor
int measureDistance(int trigPin, int echoPin) {
  // Clear the trigPin
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);

  // Set trigPin high for 10 microseconds
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);

  // Read the echoPin, returns sound wave travel time in microseconds
  long duration = pulseIn(echoPin, HIGH, 25000); // Add timeout of 25ms

  // Check if reading timed out (returned 0)
  if (duration == 0) {
    return 400; // Return a large value (no object detected)
  }

  // Calculate distance
  int distance = duration * 0.034 / 2; // Speed of sound wave divided by 2 (go
and back)

  // Filter out unreasonable values
  if (distance < minValidDistance || distance > 400) {
    return 400; // Return a large value for invalid readings
  }

  return distance;
}

// Function to play a tone on a specific buzzer
void playTone(int buzzer, int frequency, int duration) {
  // Simple tone generation with specified buzzer
  int period = 1000000 / frequency; // Period in microseconds
  for (long i = 0; i < duration * 1000L / period; i++) {
    digitalWrite(buzzer, HIGH);
    delayMicroseconds(period / 2);
    digitalWrite(buzzer, LOW);
    delayMicroseconds(period / 2);
  }
}

// Function to play two tones simultaneously on both buzzers (harmony)
void playDualTone(int buzzer1, int freq1, int buzzer2, int freq2, int duration) {
  // Approximate dual-tone generation
  long cycles = duration * 1000L / 2000; // Number of 2ms cycles
  int period1 = 1000000 / freq1;
  int period2 = 1000000 / freq2;

  for (long i = 0; i < cycles; i++) {
    // Generate a short burst of each frequency
    for (int j = 0; j < 10; j++) {
      digitalWrite(buzzer1, HIGH);
      delayMicroseconds(period1 / 2);
      digitalWrite(buzzer1, LOW);
      delayMicroseconds(period1 / 2);
    }

    for (int j = 0; j < 10; j++) {
      digitalWrite(buzzer2, HIGH);
      delayMicroseconds(period2 / 2);
      digitalWrite(buzzer2, LOW);
      delayMicroseconds(period2 / 2);
    }
  }
}

// Function to play the yellow sensor melody on buzzer 1
void playYellowMelody() {
  unsigned long currentTime = millis();

  if (currentTime - lastYellowNoteTime > 200) {
    lastYellowNoteTime = currentTime;

    // Play current note on buzzer 1
    playTone(BUZZER1, yellowMelody[yellowNoteIndex],
yellowDurations[yellowNoteIndex]);

    // Move to next note or reset to beginning
    yellowNoteIndex = (yellowNoteIndex + 1) % yellowMelodySize;
  }
}
```

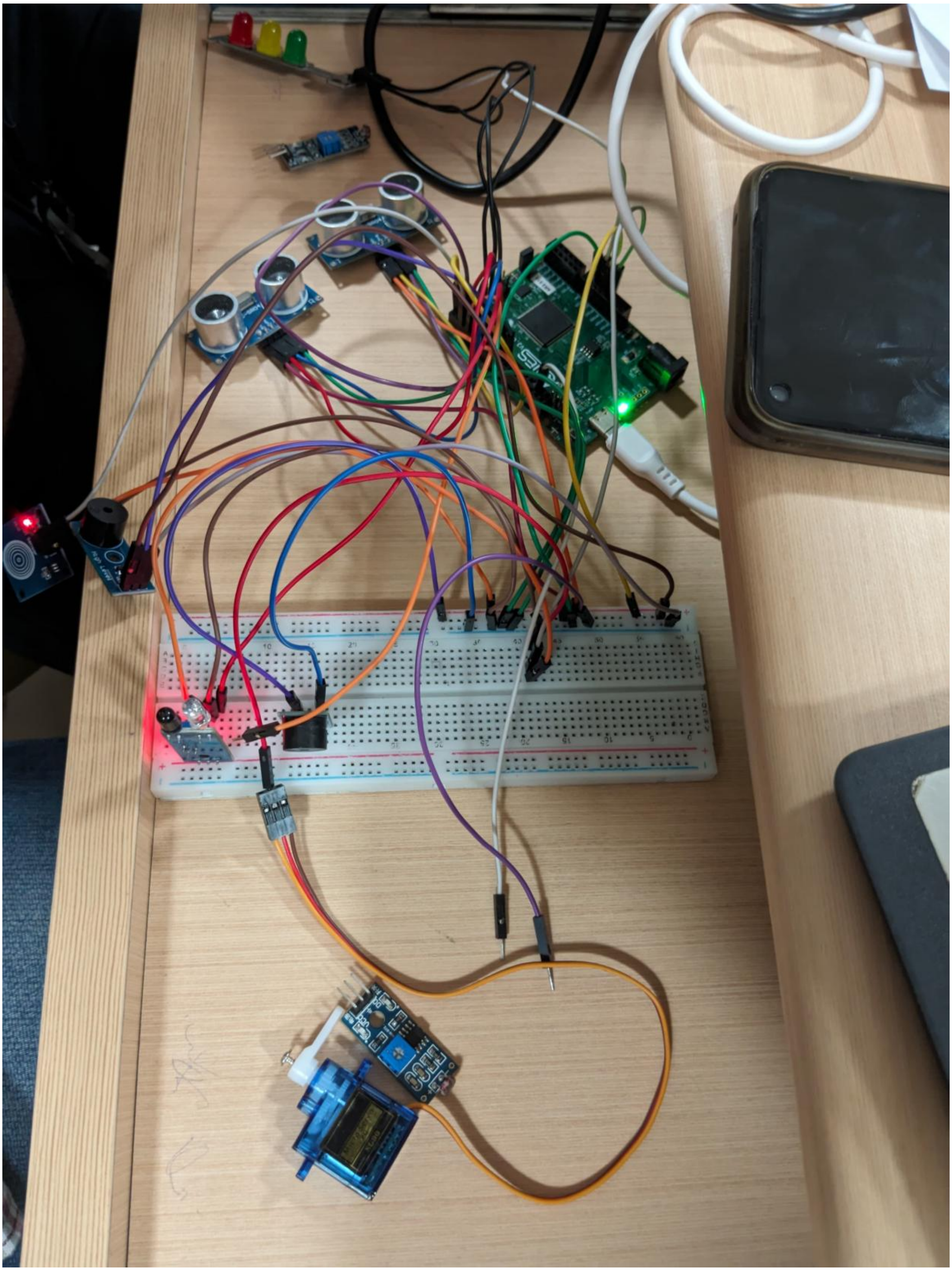
```
// Function to play the green sensor melody on buzzer 2
void playGreenMelody() {
  unsigned long currentTime = millis();

  if (currentTime - lastGreenNoteTime > 200) {
    lastGreenNoteTime = currentTime;

    // Play current note on buzzer 2
    if (greenMelody[greenNoteIndex] > 0) {
      playTone(BUZZER2, greenMelody[greenNoteIndex],
greenDurations[greenNoteIndex]);
    }

    // Move to next note or reset to beginning
    greenNoteIndex = (greenNoteIndex + 1) % greenMelodySize;
  }
}

// Function to play harmony when both sensors are triggered
void playHarmonyMelody() {
  unsigned long currentTime = millis();
  if (currentTime - lastHarmonyTime > 300) {
    lastHarmonyTime = currentTime;

    // Play harmony notes on both buzzers
    playDualTone(BUZZER1, harmonyMelody1[harmonyIndex],
                 BUZZER2, harmonyMelody2[harmonyIndex],
                 harmonyDurations[harmonyIndex]);

    // Move to next note pair
    harmonyIndex = (harmonyIndex + 1) % harmonySize;
  }
}

// Function to turn off all LEDs
void allLedsOff() {
  digitalWrite(RED_LED, LOW);
  digitalWrite(YELLOW_LED, LOW);
  digitalWrite(GREEN_LED, LOW);
}

// Function to run the disco light sequence (when IR detects)
```

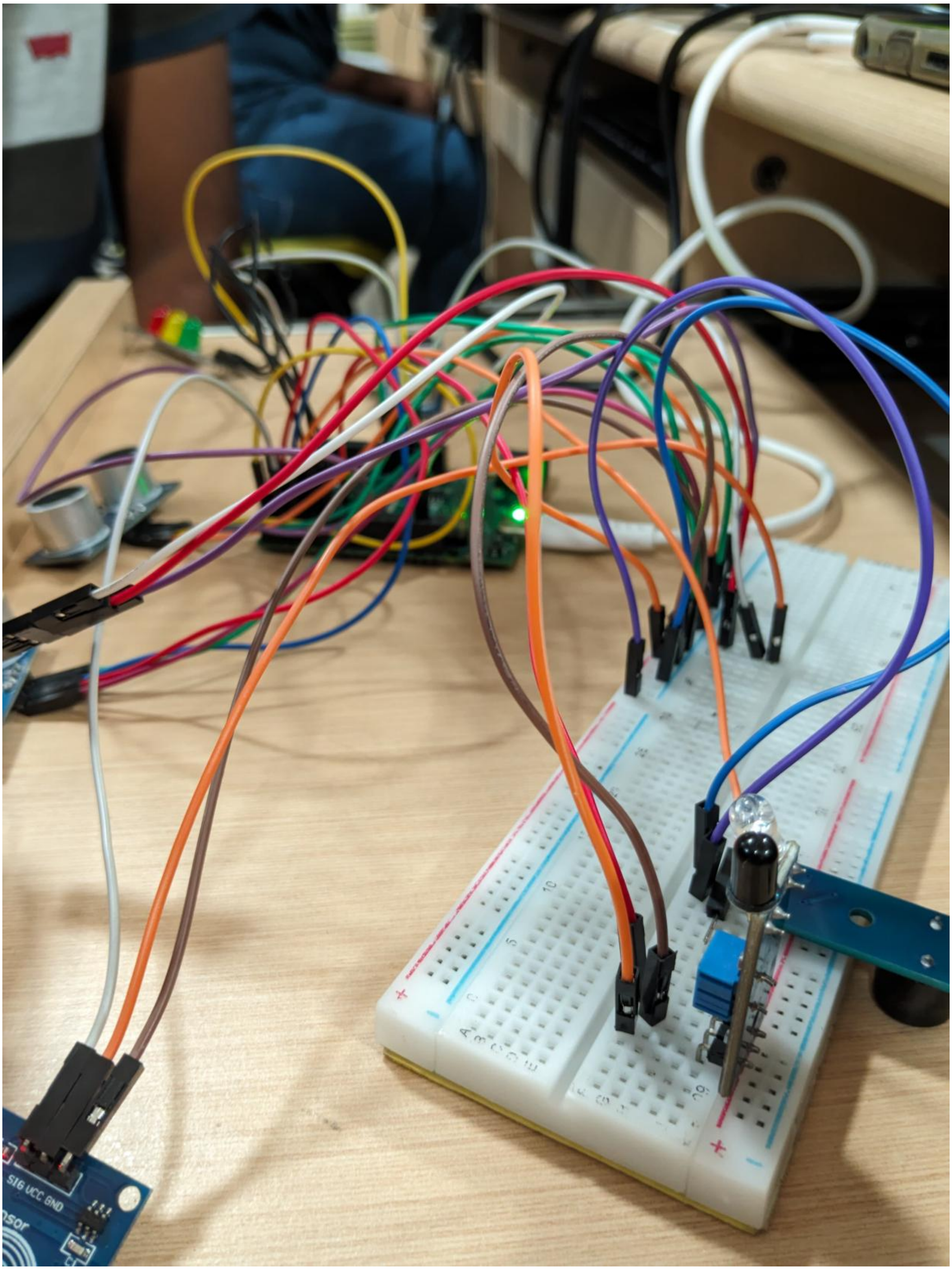
```
void runDiscoSequence() {
  unsigned long currentMillis = millis();

  if (currentMillis - lastDiscoStep > discoStepInterval) {
    lastDiscoStep = currentMillis;

    // Turn off all LEDs
    allLedsOff();

    // Move to next LED in sequence
    discoStep = (discoStep + 1) % 3;

    // Turn on the current LED in the sequence
    switch (discoStep) {
      case 0:
        digitalWrite(RED_LED, HIGH);
        break;
      case 1:
        digitalWrite(YELLOW_LED, HIGH);
        break;
      case 2:
        digitalWrite(GREEN_LED, HIGH);
        break;
    }
  }
}

// Function for stereo pulsed tones in disco mode
void playStereoPulsedTones() {
  unsigned long currentMillis = millis();

  // Check which sensors are triggered
  if (distance2 < threshold && distance2 >= minValidDistance &&
      distance3 < threshold && distance3 >= minValidDistance) {
    // Both sensors - stereo ping-pong effect
    if (currentMillis - lastPulseChange > pulseOffTime) {
      lastPulseChange = currentMillis;

      if (!pulseState) {
        // Left to right
        playTone(BUZZER1, freq2, 50);
        delay(50);
        playTone(BUZZER2, freq3, 50);
      } else {
        // Right to left
        playTone(BUZZER2, freq3, 50);
        delay(50);
        playTone(BUZZER1, freq2, 50);
      }

      pulseState = !pulseState;
    }
  }
  else if (distance2 < threshold && distance2 >= minValidDistance) {
    // Only sensor 2 - pulse buzzer 1
    if (currentMillis - lastPulseChange > (pulseState ? pulseOnTime :
pulseOffTime)) {
      pulseState = !pulseState;
      lastPulseChange = currentMillis;

      if (pulseState) {
        playTone(BUZZER1, freq2, 50);
      }
    }
  }
  else if (distance3 < threshold && distance3 >= minValidDistance) {
    // Only sensor 3 - pulse buzzer 2
    if (currentMillis - lastPulseChange > (pulseState ? pulseOnTime : pulseOffTime)) {
      pulseState = !pulseState;
      lastPulseChange = currentMillis;

      if (pulseState) {
        playTone(BUZZER2, freq3, 50);
      }
    }
  }
}

void loop() {
  // Measure distances from ultrasonic sensors
  distance2 = measureDistance(TRIG_PIN2, ECHO_PIN2);
  distance3 = measureDistance(TRIG_PIN3, ECHO_PIN3);

  // Check IR sensor - NOTE: Logic is now inverted from previous implementation
  irDetected = digitalRead(IR_SENSOR) == LOW; // Assuming IR sensor outputs LOW when object detected
```

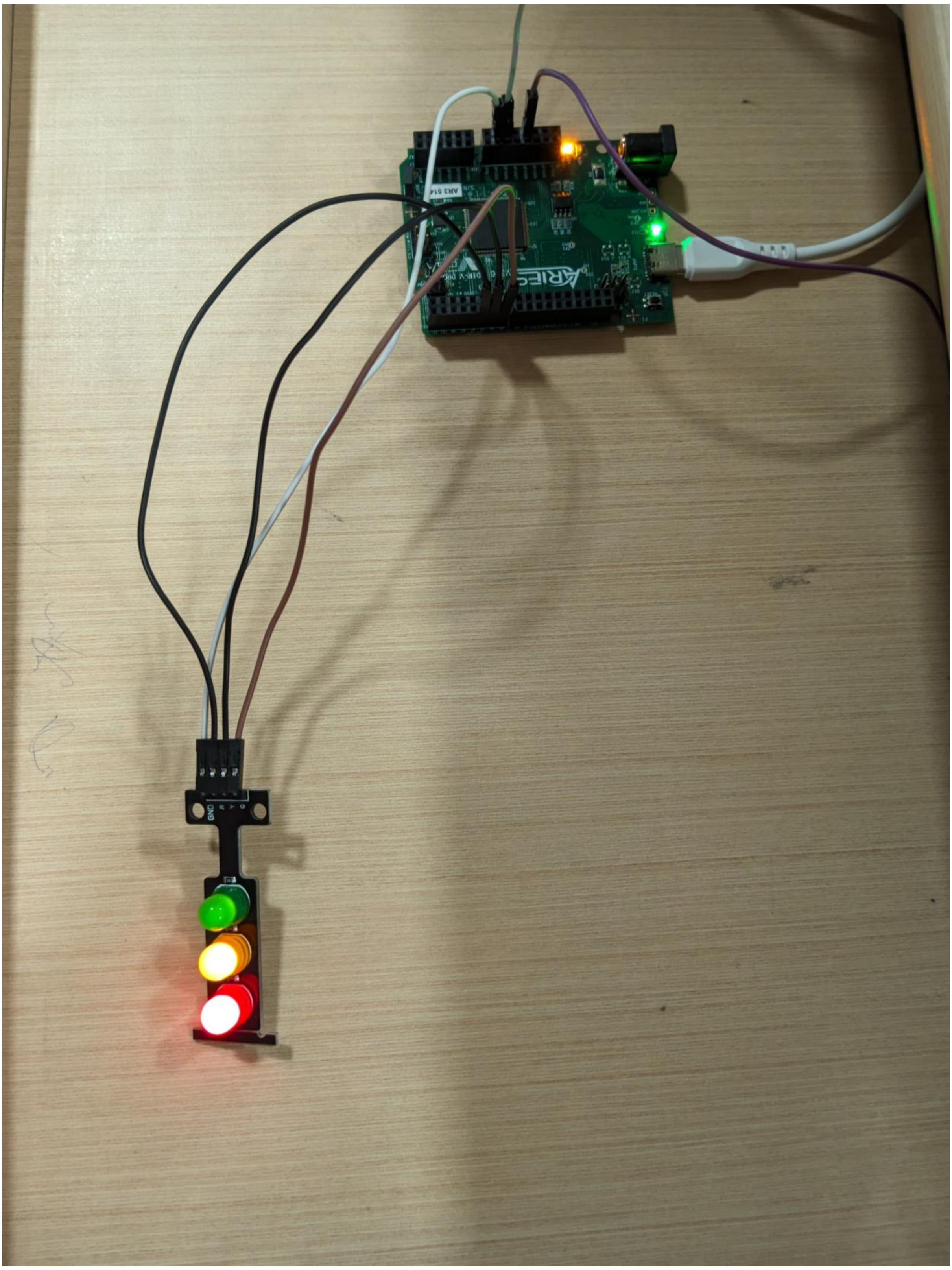
```
// Print debug info
Serial.print("Distance 2: ");
Serial.print(distance2);
Serial.print(" cm | Distance 3: ");
Serial.print(distance3);
Serial.print(" cm | IR: ");
Serial.println(irDetected ? "Detected" : "Not Detected");

// Turn off all LEDs initially
allLedsOff();

// MODE 1: IR object detected - Disco lights and stereo pulsed tones
if (irDetected) {
  // Run disco light sequence
  runDiscoSequence();

  // Play stereo pulsed tones based on which sensors are triggered
  playStereoPulsedTones();

  Serial.println("MODE 1: Disco lights and stereo pulsed tones");
}
```

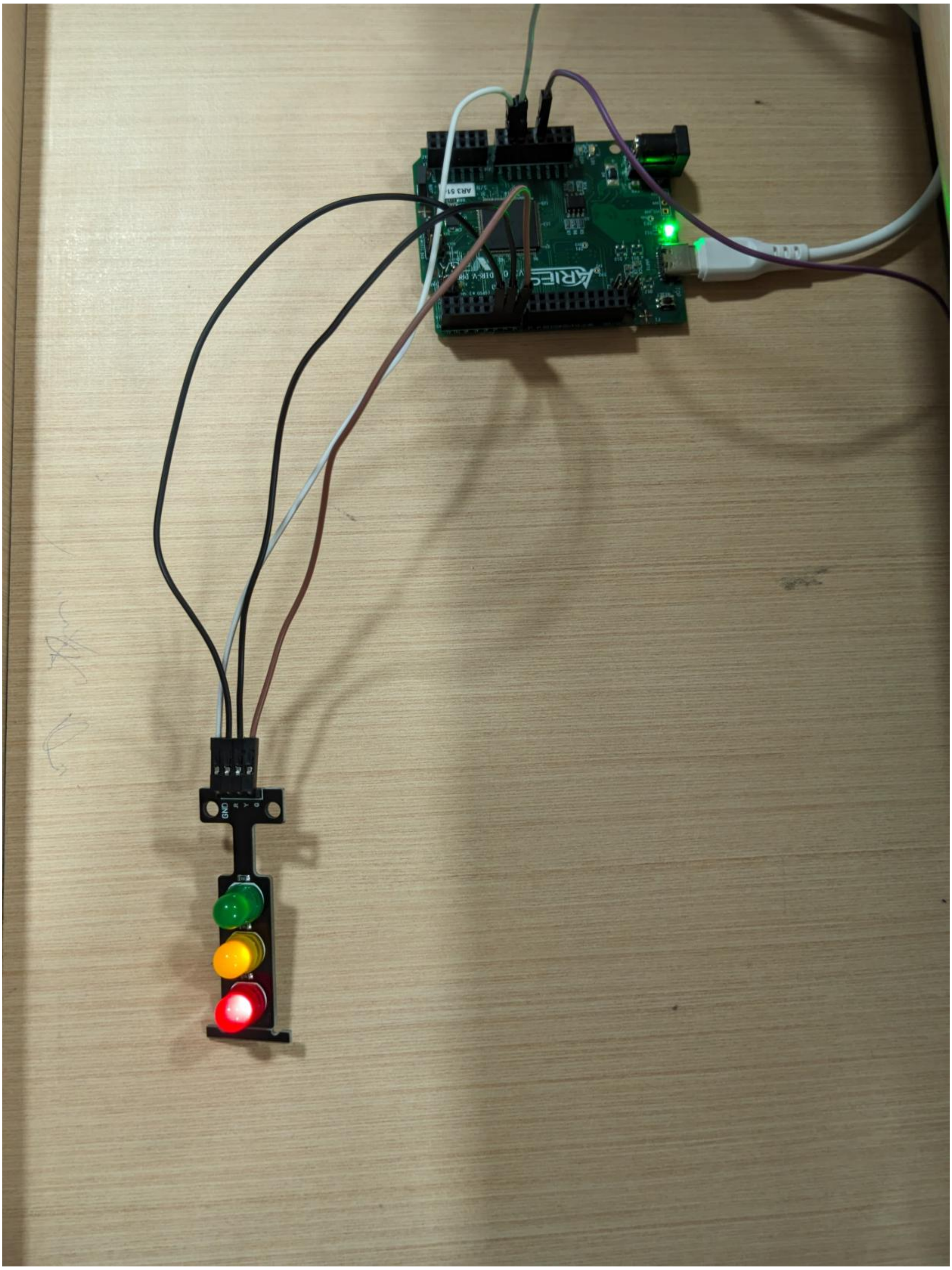
```
  // MODE 2: No IR object - Steady lights and continuous melodies
  else {
    // Check sensors and play appropriate continuous tones
    if (distance2 < threshold && distance2 >= minValidDistance &&
        distance3 < threshold && distance3 >= minValidDistance) {
      // Both sensors - turn on both LEDs, play harmony
      digitalWrite(YELLOW_LED, HIGH);
      digitalWrite(GREEN_LED, HIGH);
      playHarmonyMelody();
      Serial.println("MODE 2: Both LEDs ON - Playing harmony on both
buzzers");
    }
    else if (distance2 < threshold && distance2 >= minValidDistance) {
      // Sensor 2 only - turn on Yellow LED, play yellow melody on buzzer 1
      digitalWrite(YELLOW_LED, HIGH);
      playYellowMelody();
      Serial.println("MODE 2: YELLOW LED ON - Playing melody on buzzer
1");
    }
    else if (distance3 < threshold && distance3 >= minValidDistance) {
      // Sensor 3 only - turn on Green LED, play green melody on buzzer 2
      digitalWrite(GREEN_LED, HIGH);
      playGreenMelody();
      Serial.println("MODE 2: GREEN LED ON - Playing melody on buzzer
2");
    }
    else {
      // No sensors triggered - silence
      digitalWrite(BUZZER1, LOW);
      digitalWrite(BUZZER2, LOW);
    }
  }

  // Short delay for stability
  delay(5);
}
```

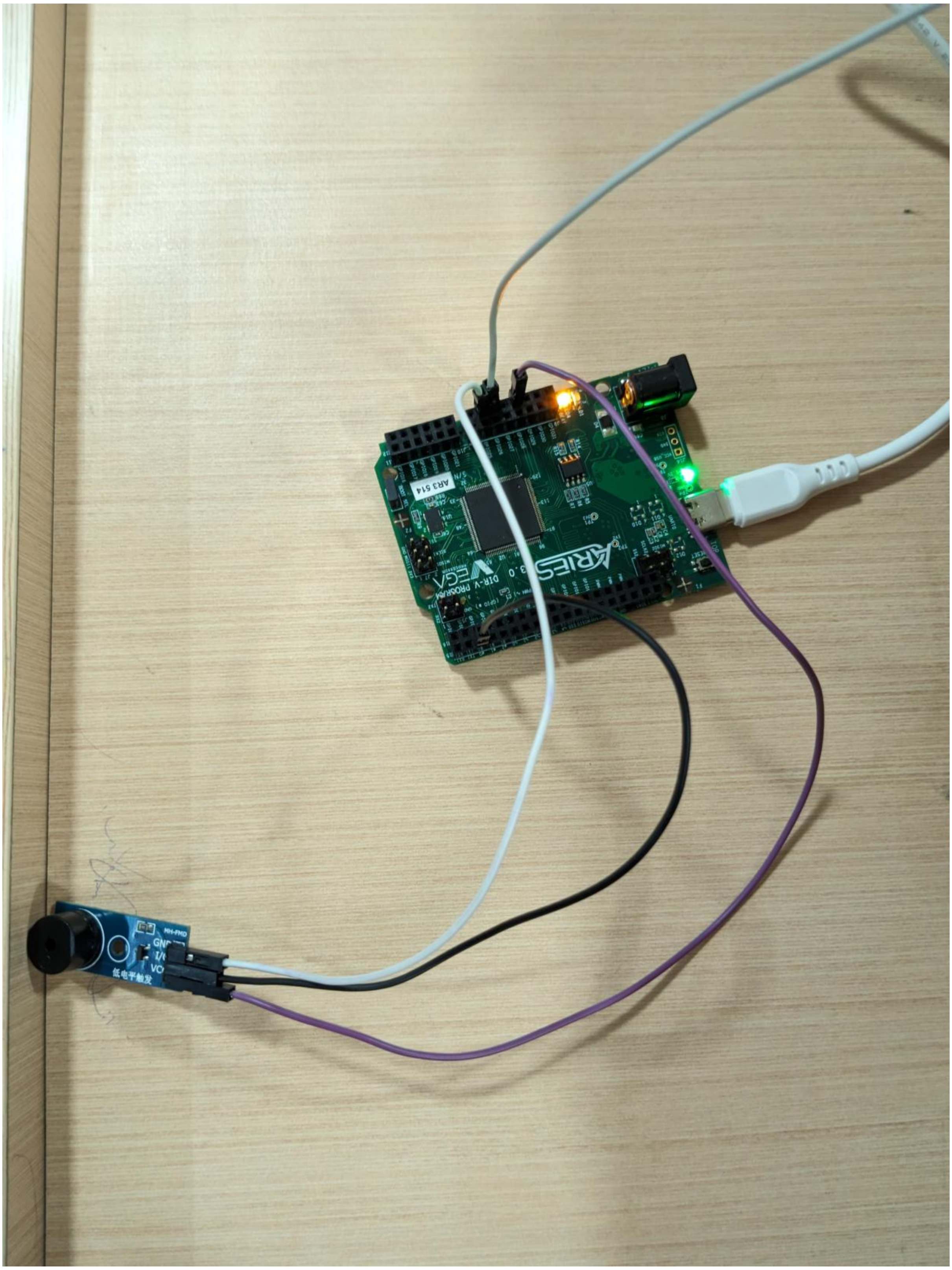
SNAPSHOTS OF THE WEEK : ACTIVITY 04




SIG VCC GND
nsor




GND
R
Y
G
ARIES


AR3 514
ARIES 3.0
VEGA
MH-FMD
GND
低电平触发

ACTIVITY 05
RFID ATTENDANCE CHECK

PROBLEM STATEMENT

If no card is shown, no LED in the RYG strip should glow. When a card is shown, if valid, Green should glow. If not valid, yellow should glow. If more than 5 attempts of invalid attendance, red would glow until board is reset.

COMPONENTS USED

Aries Development Board v3, USB Cable, RYG LED Strip, RFID module, RFID tag(s), jumper wires

PIN CONNECTIONS

RFID to Aries V3:

MISO → MISO 0

MOSI → MOSI 0

3V3 → 3V3

GND → GND

SCK -> SCLK 0

SDA / SS -> GPIO - 10

RYG Strip to Aries V3:

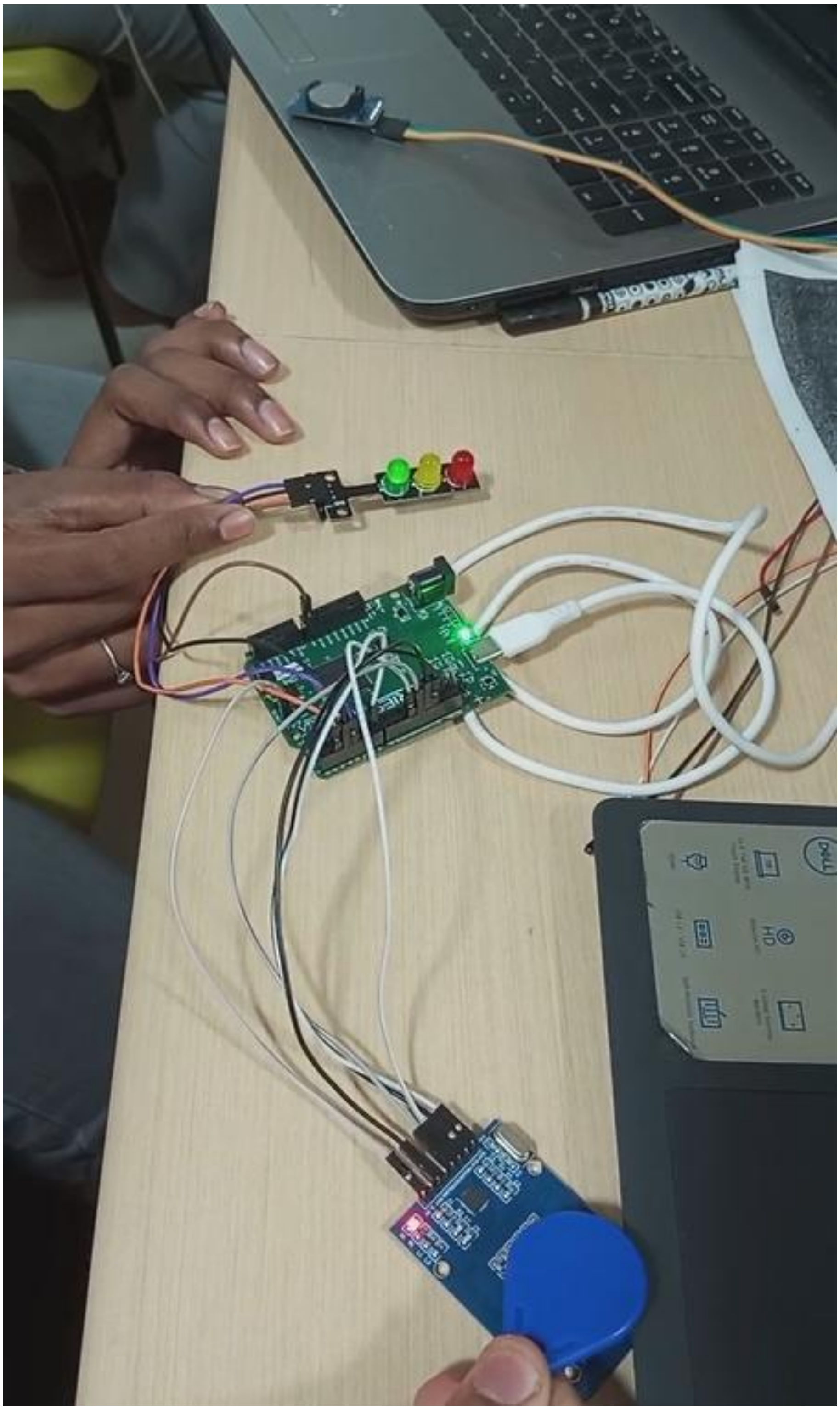
GND → GND

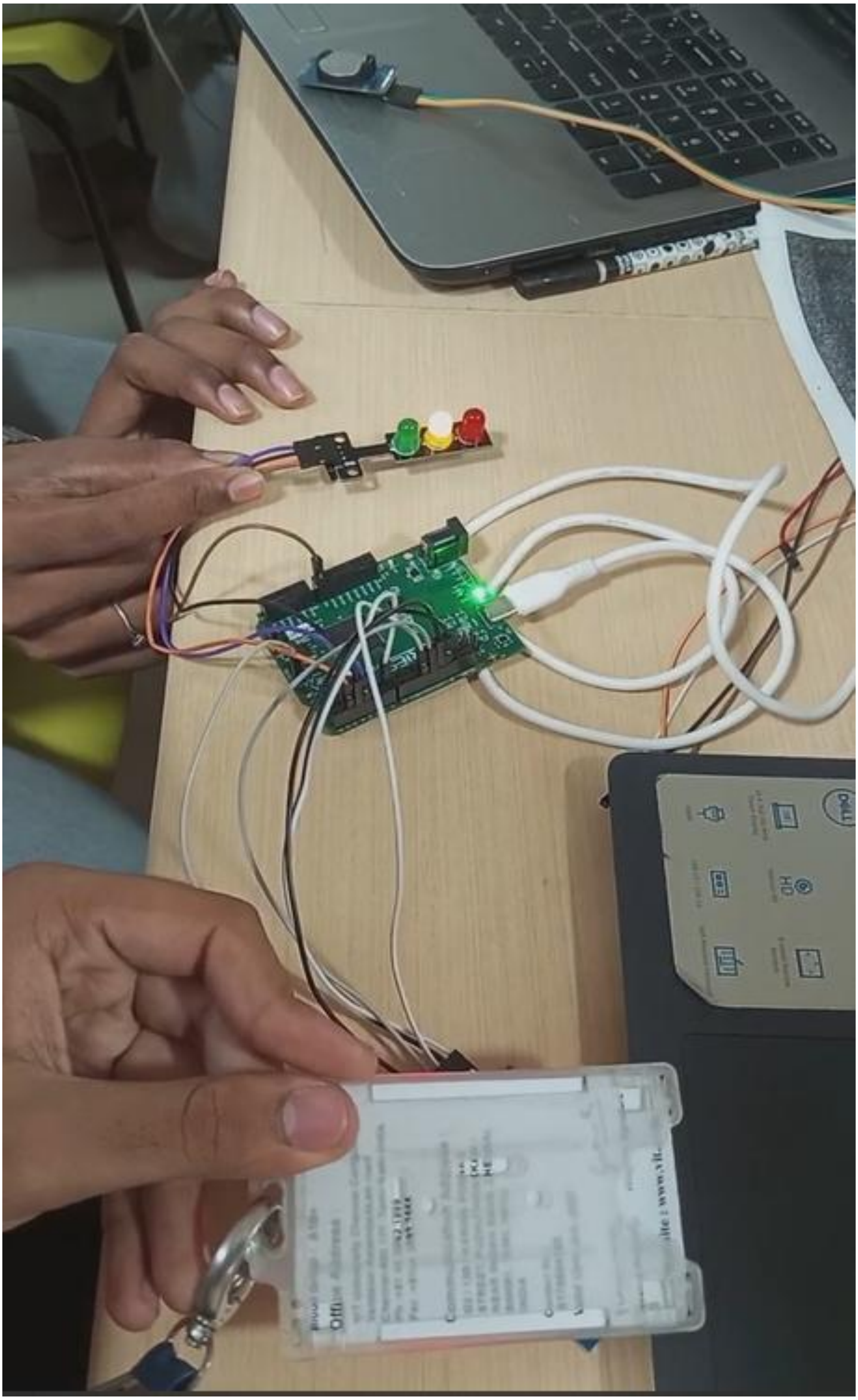
R -> GPIO 0

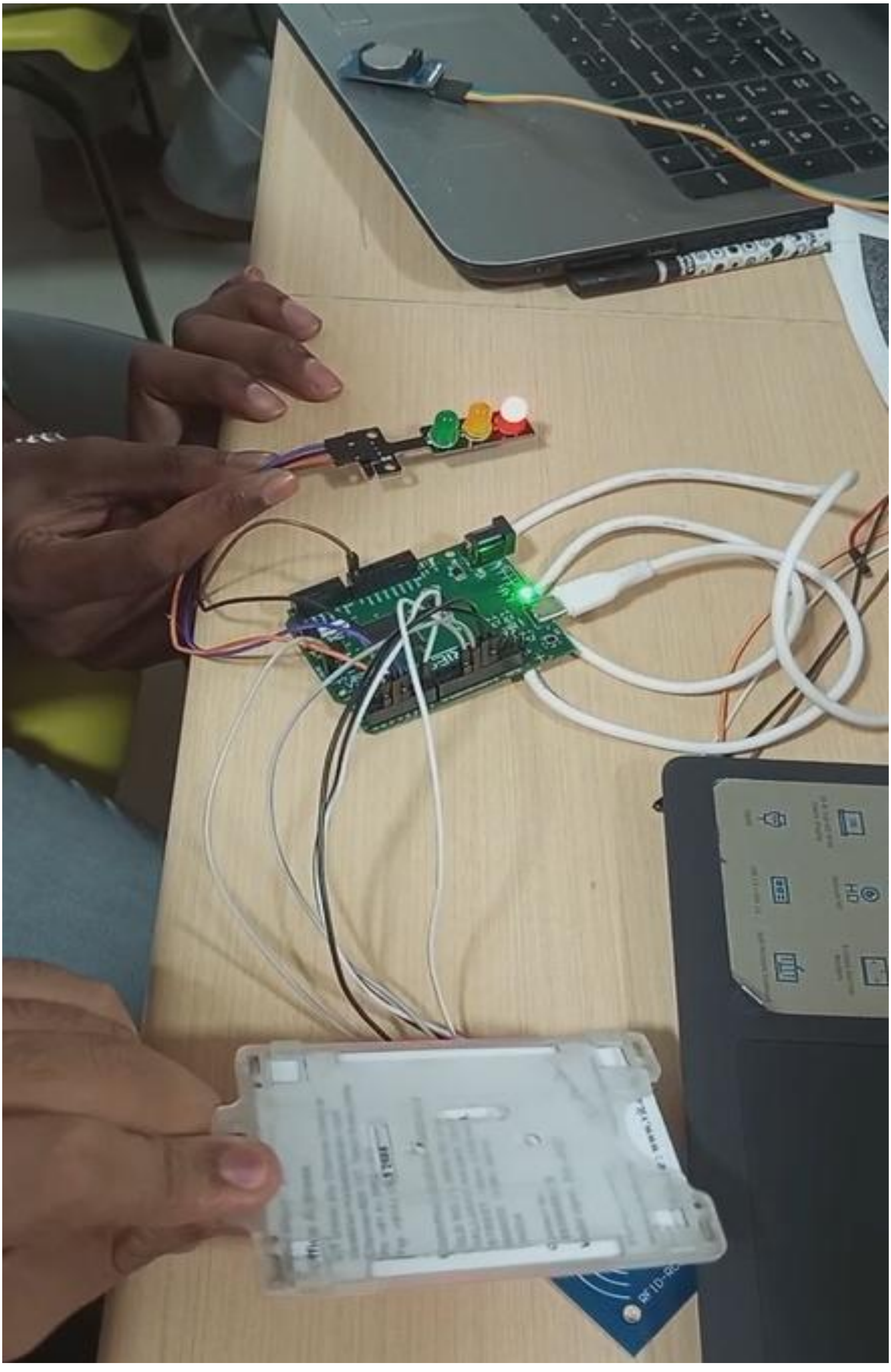
Y -> GPIO 1

G -> GPIO 2

SNAPSHOTS






RFID-RC

CODE

```
#include <SPI.h>
#include <MFRC522.h>

// RFID pins definition
#define SS_PIN 10     // Connect SDA/SS pin of RFID to GPIO-10
#define RST_PIN 9     // Not required but defined for MFRC522 library

// LED pins definition
#define RED_LED     0  // Connect Red LED pin to GPIO-0
#define YELLOW_LED  1  // Connect Yellow LED pin to GPIO-1
#define GREEN_LED   2  // Connect Green LED pin to GPIO-2

// RFID initialization
SPIClass SPI(0);
MFRC522 rfid(SS_PIN, RST_PIN);
MFRC522::MIFARE_Key key;
// Define valid card UIDs (maximum 5 cards)
// First valid card from example: A8 AB B1 12 (hex)
const byte VALID_CARDS[][4] = {
  {0xA8, 0xAB, 0xB1, 0x12},  // Card 1
  // Add more valid cards here as needed
```

```
};
const int NUM_VALID_CARDS = 1;  // Update when adding more cards


// Security tracking variables
int invalidAttempts = 0;
const int MAX_INVALID_ATTEMPTS = 3;  // Changed from 5 to 3
bool systemLocked = false;


// LED feedback timing
unsigned long lastCardTime = 0;
const unsigned long FEEDBACK_DURATION = 3000;  // 3 seconds


void setup() {
  // Initialize serial communication
  Serial.begin(115200);


  // Initialize SPI bus and RFID reader
  SPI.begin();
  rfid.PCD_Init();


  // Initialize LED pins
  pinMode(RED_LED, OUTPUT);
  pinMode(YELLOW_LED, OUTPUT);
  pinMode(GREEN_LED, OUTPUT);

  // Turn all LEDs off initially
  allLedsOff();

  // Initialize RFID key
  for (byte i = 0; i < 6; i++) {
    key.keyByte[i] = 0xFF;
  }

  Serial.println(F("RFID Access Control System"));
  Serial.println(F("Present your card for authentication"));
}

void loop() {
  // System locked state (after too many invalid attempts)
  if (systemLocked) {
    allLedsOff();              // Clear all LEDs first
    digitalWrite(RED_LED, HIGH); // Turn on ONLY red LED
    return;                    // Exit loop until reset
  }
  // Check if we're currently in LED feedback mode (valid/invalid card was just shown)
  if (millis() - lastCardTime < FEEDBACK_DURATION && lastCardTime > 0) {
    // Still in feedback period, don't do anything
    return;
  } else if (lastCardTime > 0) {
    // Feedback period has ended, turn off all LEDs
    allLedsOff();
    lastCardTime = 0;
  }

  // Reset the loop if no new card present on the sensor/reader
  if (!rfid.PICC_IsNewCardPresent())
    return;

  // Verify if the NUID has been read
  if (!rfid.PICC_ReadCardSerial())
    return;

  // Check if the card UID matches any valid card
```

```
bool isValidCard = false;
for (int i = 0; i < NUM_VALID_CARDS; i++) {
  if (memcmp(rfid.uid.uidByte, VALID_CARDS[i], 4) == 0) {
    isValidCard = true;
    break;
  }
}

if (isValidCard) {
  Serial.println(F("Valid card detected."));
  digitalWrite(GREEN_LED, HIGH);  // Turn on green LED
  lastCardTime = millis();        // Start feedback timer
  invalidAttempts = 0;            // Reset invalid attempts
} else {
  Serial.println(F("Invalid card detected."));
  invalidAttempts++;
  Serial.print(F("Invalid attempts: "));
  Serial.println(invalidAttempts);

  if (invalidAttempts >= MAX_INVALID_ATTEMPTS) {
    Serial.println(F("SECURITY ALERT: System locked!"));
    allLedsOff();                 // Clear all LEDs
```

```
      digitalWrite(RED_LED, HIGH);   // Turn on ONLY red LED
      systemLocked = true;           // Lock the system
    } else {
      digitalWrite(YELLOW_LED, HIGH); // Turn on yellow LED for feedback
      lastCardTime = millis();        // Start feedback timer
    }
  }

  // Halt PICC
  rfid.PICC_HaltA();

  // Stop encryption on PCD
  rfid.PCD_StopCrypto1();
}
/**
 * Turn off all LEDs.
 */
void allLedsOff() {
  digitalWrite(RED_LED, LOW);
  digitalWrite(YELLOW_LED, LOW);
  digitalWrite(GREEN_LED, LOW);
}
```

ACTIVITY 06
ALARM WITH RTC

PROBLEM STATEMENT

Set an alarm for a given hour:minute (24 hour format) using RTC and Buzzer

COMPONENTS USED

Aries Development Board v3, USB Cable, IR sensor, 4-bit display, jumper wires

PIN CONNECTIONS

Aries connection to DS1307:

VCC -> 3.3V

GND -> GND

SDA -> SDA0

SCL -> SCL0

Aries connection to Buzzer:

VCC -> 3.3V

GND -> GND

IN -> GPIO 0

SCL -> SCL0

CODE

```
#include <TimeLib.h>
#include <DS1307RTC.h>

#define BUZZER_PIN 0       // Connect buzzer to GPIO-0

// EASY ALARM CONFIGURATION - SET YOUR ALARM TIME HERE
const uint8_t ALARM_HOUR = 14;    // 24-hour format (e.g., 7 for 7AM, 15
for 3PM)
const uint8_t ALARM_MINUTE = 58; // Minutes (0-59)

TwoWire Wire(0); // I2C-0

bool parse = false;
bool config = false;
bool alarmActive = false;
unsigned long lastBeepChange = 0;  // For tracking beep pattern timing
uint8_t beepState = 0;             // For tracking position in beep pattern

const char *monthName[12] = {
  "Jan", "Feb", "Mar", "Apr", "May", "Jun",
  "Jul", "Aug", "Sep", "Oct", "Nov", "Dec"
```

```
};


tmElements_t tm;

tmElements_t alarmTime;  // To store the alarm time


// the setup function runs once when you press reset or power the board

void setup() {

  // initialize serial communication

  Serial.begin(115200);

  pinMode(BUZZER_PIN, OUTPUT);  // Set buzzer pin as output

  digitalWrite(BUZZER_PIN, LOW); // Ensure buzzer is off initially


  delay(1000);

  Serial.println("RTC Alarm Clock with Buzzer");


  // Set RTC time using compiler time

  char timeString[9];

  char dateString[12];

  strcpy(timeString, __TIME__);

  strcpy(dateString, __DATE__);


  // get the date and time the compiler was run
  if (getDate(dateString) && getTime(timeString)) {
    parse = true;
    // and configure the RTC with this info
    if (RTC.write(tm)) {
      config = true;
    }
  }

  // Set the alarm using predefined hour and minute
  if (config) {
    setAlarmExact(ALARM_HOUR, ALARM_MINUTE);
  }
}

// the loop function runs over and over again forever
void loop() {
  // Read current time from RTC
  if (RTC.read(tm)) {
    displayTime();
    checkAlarm();
  } else {
    Serial.println("Failed to read RTC!");
  }

  delay(1000); // Update every second
}

// Display the current time on serial monitor
void displayTime() {
  Serial.print("Current time: ");
  Serial.print(tm.Hour);
  Serial.print(":");
  if (tm.Minute < 10) Serial.print("0");
  Serial.print(tm.Minute);
  Serial.print(":");
  if (tm.Second < 10) Serial.print("0");
  Serial.print(tm.Second);
  Serial.print(" ");
  Serial.print(tm.Day);
  Serial.print(" ");
  Serial.print(monthName[tm.Month-1]);
  Serial.print(" ");
  Serial.println(tmYearToCalendar(tm.Year));
}
```

```
// Set alarm to a specific hour and minute
void setAlarmExact(uint8_t hour, uint8_t minute) {
  alarmTime.Hour = hour;
  alarmTime.Minute = minute;

  Serial.print("Alarm set for: ");
  Serial.print(alarmTime.Hour);
  Serial.print(":");
  if (alarmTime.Minute < 10) Serial.print("0");
  Serial.println(alarmTime.Minute);
}

// Set alarm to trigger after specified minutes from current time
void setAlarm(int minutesFromNow) {
  if (RTC.read(alarmTime)) {
    // Set alarm time based on current time + minutesFromNow
    alarmTime.Minute += minutesFromNow;
    if (alarmTime.Minute >= 60) {
      alarmTime.Minute -= 60;
      alarmTime.Hour += 1;
      if (alarmTime.Hour >= 24) {
        alarmTime.Hour -= 24;
      }
    }

    Serial.print("Alarm set for: ");
    Serial.print(alarmTime.Hour);
    Serial.print(":");
    if (alarmTime.Minute < 10) Serial.print("0");
    Serial.println(alarmTime.Minute);
  }
}

// Check if current time matches alarm time
void checkAlarm() {
  // Check if alarm should trigger (match hour and minute)
  if (tm.Hour == alarmTime.Hour && tm.Minute == alarmTime.Minute) {
    if (!alarmActive) {
      alarmActive = true;
      Serial.println("ALARM TRIGGERED!");
      lastBeepChange = millis(); // Initialize beep timing
      beepState = 0;            // Start pattern from beginning
    }
    // Sound the buzzer with pattern
    soundAlarmPattern();
  } else {
    // Turn off buzzer and reset alarm flag if time no longer matches
    if (alarmActive) {
      alarmActive = false;
      digitalWrite(BUZZER_PIN, LOW);
    }
  }
}

// Create a more complex alarm sound pattern
void soundAlarmPattern() {
  // SOS pattern (... --- ...) with timing
  // Short beeps are 100ms, long beeps are 300ms, pauses are 100ms
  // Sequence pause is 500ms

  const uint8_t PATTERN_LENGTH = 19;  // Total states in pattern
  const uint16_t timings[PATTERN_LENGTH] = {
    100, 100, 100, 100, 100, 100, // Three short beeps (S)
    300, 100, 300, 100, 300,      // Three long beeps (O)
    100, 100, 100, 100, 100, 100, // Three short beeps (S)
    500, 500              // Longer pause before repeating
  };

  const bool states[PATTERN_LENGTH] = {
    HIGH, LOW, HIGH, LOW, HIGH, LOW, // Three short beeps (S)
    HIGH, LOW, HIGH, LOW, HIGH,      // Three long beeps (O)
    HIGH, LOW, HIGH, LOW, HIGH, LOW, // Three short beeps (S)
    LOW, LOW                         // Longer pause before repeating
  };

  unsigned long currentTime = millis();

  if ((currentTime - lastBeepChange) > timings[beepState]) {
    beepState = (beepState + 1) % PATTERN_LENGTH;
    digitalWrite(BUZZER_PIN, states[beepState]);
    lastBeepChange = currentTime;
  }
}

// Original simple alarm pattern (kept for reference)
void soundAlarm() {
  // Create a beeping pattern for the alarm
  if ((tm.Second % 2) == 0) {

    // On during even seconds

    digitalWrite(BUZZER_PIN, HIGH);

  } else {

    // Off during odd seconds

    digitalWrite(BUZZER_PIN, LOW);

  }

}


bool getTime(char *str) {

  // get the time by seperating the string:

  char* time[3];

  int count = 0;

  char* token = strtok(str, ":");

  while (token != NULL) {

    time[count] = token;

    count++;

    token = strtok(NULL, ":");

  }


  if(count == 3) {

    tm.Hour = atoi(time[0]);
    tm.Minute = atoi(time[1]);
    tm.Second = atoi(time[2]);
    return true;
  }
  return false;
}

bool getDate(char *str) {
  // get the date by seperating the string:
  char* date[3];
  uint8_t monthIndex;

  int count = 0;
  char* token = strtok(str, " ");
  while (token != NULL) {
    date[count] = token;
    count++;
    token = strtok(NULL, " ");
  }
  if(count != 3) return false;
  for (monthIndex = 0; monthIndex < 12; monthIndex++) {
    if (strcmp(date[0], monthName[monthIndex]) == 0) break;
  }
  if (monthIndex >= 12) return false;
  tm.Day = atoi(date[1]);
  tm.Month = monthIndex + 1;
  tm.Year = CalendarYrToTm(atoi(date[2]));
  return true;
}
```

SNAPSHOTS OF THE CIRCUIT