Syeda Samia (2123536)
Muhammad Eiqbal bin Hasbollah (2216911)

We connected a pushbottun and a potentiometer to an Arduino Uno which is acting as the master Arduino to control the Arduino Mega connected by RX TX. The Arduino Mega is connected to an LED pin and a Servo motor. When the pushbotton is pressed the LED turns on whereas when released it turns off. On the other hand, when turning the potentiometer knob we control the servo's angle. Moreover, we displayed the values of the outputs in a graph whiles also displaying the output and input values on python.

Integration of sensors and actuators with 2 Arduinos

Arduino code for Master Arduino:

```
//uno master
#include <Wire.h>

const int buttonPin = 10;
const int potPin = A0;  // Potentiometer connected to analog pin A0
const int slaveAddress = 8;  // I2C address of the slave Arduino

int buttonState = 0;  // Variable for reading the pushbutton status
int potValue = 0;     // Variable for reading the potentiometer value

void setup() {
  pinMode(buttonPin, INPUT_PULLUP);

  // Initialize I2C communication as master
  Wire.begin();
  Serial.begin(9600);
}

void loop() {
  buttonState = digitalRead(buttonPin);
  potValue = analogRead(potPin);  // Read the potentiometer value


  // Send the state of the pushbutton and potentiometer value to the slave
  Wire.beginTransmission(slaveAddress);
  Wire.write(buttonState);        // Send the button state to the slave
  Wire.write(potValue >> 8);      // Send the high byte of pot value
  Wire.write(potValue & 0xFF);    // Send the low byte of pot value
  Wire.endTransmission();

  delay(100);  // A short delay to avoid overwhelming the I2C bus
}
```

Explanation: The code sets up an Arduino Uno as an I2C master to communicate with a slave device at address 8. It reads the state of a pushbutton (connected to pin 10 with an internal pull-up resistor) and the value of a potentiometer (connected to analog pin A0). In the setup function, it initializes the button pin, I2C communication, and serial communication. In the loop function, it reads the button state and potentiometer value, then sends these values to the I2C slave. The potentiometer value, a 10-bit number, is split into high and low bytes for transmission. A 100ms delay prevents I2C bus overload.

Arduino code for Slave Arduino:

```
//mega slave
#include <Wire.h>
#include <Servo.h>

const int ledPin = 10;
Servo myServo;        // Create a servo object
int buttonState = 0;  // Variable for storing the received pushbutton status
int potValue = 0;     // Variable for storing the received potentiometer value

void setup() {
  pinMode(ledPin, OUTPUT);

  // Initialize I2C communication as slave
  Wire.begin(8);  // Join I2C bus with address #8
  Wire.onReceive(receiveEvent);  // Register event handler for receiving data

  myServo.attach(9);
  Serial.begin(9600);
}

void loop() {
    if (buttonState == LOW) {
    digitalWrite(ledPin, HIGH);
  } else {
    digitalWrite(ledPin, LOW);
  }

  // Control the servo motor based on potentiometer value
  int servoPos = map(potValue, 0, 1023, 0, 180);
  myServo.write(servoPos);

  // Serial output
  Serial.print("Pot Value: ");
  Serial.print(potValue);
  Serial.print(" Servo Position: ");
  Serial.println(servoPos);

  delay(50);  // A short delay to avoid rapid LED state changes
```

```
}

// Function that executes whenever data is received from master
void receiveEvent(int howMany) {
  if (howMany >= 3) {
    buttonState = Wire.read(); // Read the button state
    int highByte = Wire.read(); // Read the high byte of pot value
    int lowByte = Wire.read();  // Read the low byte of pot value
    potValue = (highByte << 8) | lowByte; // Combine high and low bytes

    // Print the received button state and potentiometer value to the Serial Monitor
    Serial.print("Received Button State: ");
    Serial.print(buttonState);
    Serial.print(" Pot Value: ");
    Serial.println(potValue);
  }
}
```

Explanation :This Arduino code sets up an I2C slave device using an Arduino Mega, which controls an LED and a servo motor based on data received from an I2C master. The LED is connected to pin 10 and the servo to pin 9. In the `setup()`, the I2C communication is initialized with address 8, and an event handler `receiveEvent` is registered to handle incoming data. The `loop()` continuously updates the LED and the servo position based on the received button state and potentiometer value. The `receiveEvent` function reads the button state and the potentiometer value, processes the data, and prints it to the Serial Monitor for debugging.

Displaying values on computer

Python code:

```
import serial
import time
ser = serial.Serial('COM12', 9600)
time.sleep(2)  # Wait for the serial connection to initialize
def read_from_arduino():
 while True:
  if ser.in_waiting > 0:
   data = ser.readline().decode('utf-8').strip()
   print(data)

try:
 read_from_arduino()
except KeyboardInterrupt:
 print("Program stopped")
finally:
 ser.close()
```

Explanation: The code establishes a serial connection to an Arduino on 'COM12' at 9600 baud. It reads and prints incoming data continuously from the Arduino. The loop runs indefinitely, but can be interrupted with a keyboard command (Ctrl+C), ensuring the serial connection is closed properly afterward.

Real-Time Arduino Servo Monitoring and Control with Python

```
import serial
import time
import matplotlib.pyplot as plt
import matplotlib.animation as animation

ser = serial.Serial('COM12', 9600)  # Replace 'COM13' with your actual serial port
time.sleep(2)  # Wait for the serial connection to initialize

# Variables to store data
xs = []
ys = []

def read_from_arduino():
 if ser.in_waiting > 0:
  data = ser.readline().decode('utf-8').strip()
  print(data)
  if "Servo Angle: " in data:
   try:
    angle_str = data.split("Servo Angle: ")[1]
    angle = int(angle_str)
    return angle
   except (IndexError, ValueError):
    pass
 return None

def send_to_arduino(led_state, servo_angle):
 ser.write(bytes([led_state]))
 ser.write(bytes([servo_angle]))

def animate(i, xs, ys):
 angle = read_from_arduino()
 if angle is not None:
  xs.append(time.time())
  ys.append(angle)
  # Limit the lists to the last 40 items
  xs = xs[-40:]
  ys = ys[-40:]
  # Clear and plot the data
  ax.clear()
  ax.plot(xs, ys)
  plt.xticks(rotation=45, ha='right')
  plt.subplots_adjust(bottom=0.30)
  plt.title('Servo Motor Angle Over Time')
  plt.ylabel('Angle (degrees)')
```

```
# Set up plot to call animate() function periodically
fig, ax = plt.subplots()
ani = animation.FuncAnimation(fig, animate, fargs=(xs, ys), interval=100, save_count=100)

try:
 while True:
  # Read and print data from Arduino
  read_from_arduino()
  # Example to send LED state and servo angle from Python to Arduino
  led_state = 1  # Change to 0 to turn off the LED
  servo_angle = 90  # Example angle, you can set it to any value between 0 and 180
  send_to_arduino(led_state, servo_angle)
  # Show the plot
  plt.show()

except KeyboardInterrupt:
 print("Program stopped")
finally:
 ser.close()
```

Explanation: This Python script establishes a serial connection with an Arduino board to collect real-time data. It continuously reads servo angle data from the Arduino and updates a live plot using Matplotlib to visualize the angle over time. Additionally, the script sends commands to control both an LED and the servo motor position based on specific states or inputs. The communication with the Arduino is handled through the serial library, while Matplotlib facilitates the dynamic plotting of the data. Overall, this script creates a comprehensive system for monitoring and controlling servo motor behavior in real-time.

Controlling using computer

Arduino code:

```
#include <Servo.h>
#define SER 7
Servo servo;
int mssg;

void setup() {
  servo.attach(SER);
  Serial.begin(9600);
}

void loop() {
  if (Serial.available() > 0) {
    mssg = Serial.parseInt();
    servo.write(mssg);
    delay(50);
  }
}
```

Explanation: This Arduino code initializes a servo motor connected to pin 7 and establishes serial communication at 9600 baud. In the loop function, it continuously checks for incoming serial data. If data is available, it reads an integer from the serial buffer and assigns it to the variable mssg. Then, it commands the servo motor to move to the angle specified by mssg using the servo.write() function. A small delay is added to ensure smooth servo movement. This setup allows external devices to control the servo motor's position via serial communication.

Python code:

```
import serial
from time import sleep

try:
    ser = serial.Serial('COM9', 9600)
    sleep(5)  # Waiting for Arduino to initialize
    print("Serial connection established successfully.")
except serial.SerialException:
    print("Failed to establish serial connection. Make sure the port is correct and Arduino is
connected.")
    exit()

print("Enter a number between 0-180, or 'e' to exit the program\n")

while True:
    enter = input("Enter a number between 0-180: ")
```

```
    if enter == 'e':
        break

    try:
        angle = int(enter)
        if 0 <= angle <= 180:
            ser.write(str(angle).encode())
            print("Sent angle:", angle)
        else:
            print("Angle must be between 0 and 180")
    except ValueError:
        print("Invalid input. Please enter a number or 'e' to exit.")
```

Explanation: The code establishes a serial connection to an Arduino on 'COM9' at 9600 baud, waiting 5 seconds for initialization. It prompts the user to enter a number between 0 and 180 or 'e' to exit. If a valid number is entered, it's sent to the Arduino via the serial connection; otherwise, an error message is displayed. If 'e' is entered, the loop breaks, and the program terminates. The program handles exceptions to ensure the serial connection is correctly established and user inputs are valid, enhancing robustness and usability.