MCTA 3203: MECHATRONICS SYSTEM INTEGRATION

SECTION 2

SEMESTER 1, 2024/2025

INSTRUCTOR:

Wahju Sediono *(Dr.)*

Zulkifli Bin Zainal Abidin *(Assoc. Prof. Eur. Ing. Ir. Ts. Gs. Inv. Dr.)*

REPORT ON MIDTERM

PREPARED BY:

MUHAMMAD BASIL BIN ABDUL HAKIM (2215315)

MUHAMMAD HAFIZ HAMZAH BIN FANSURI (2212803)

Introduction

In this midterm assessment, we worked on integrating two systems by connecting and communicating between two Arduino Uno boards using the (Inter-Integrated Circuit) I2C protocol. The setup included a master Arduino and a slave Arduino, working together to control input and output devices.

The master Arduino used a push button and a potentiometer as input devices. The push button sent a signal to the slave Arduino to turn an LED on or off. At the same time, the potentiometer sent a signal to the slave Arduino to control the rotation of a servo motor.

Then, we managed to use Serial Plotter to display the input Data visualisation on Task 2. After that, on Task 3, we control the output via User Input Control on the IDE, without signal from the master Arduino.

Lastly, for Task 4 (troubleshooting) we are required spontaneously by the instructor to replace the LED with a buzzer as an output.

Coding

Task 1 and Task 2 Master

```
#include <Wire.h> // Include the Wire library for I2C communication

int potValue;    // Variable to store potentiometer value
int buttonState; // Variable to store button state
const int buttonPin = 7; // Button connected to pin 7

void setup() {
  Serial.begin(9600);    // Initialize serial communication for debugging and plotting
  Wire.begin();          // Start I2C communication as master
  pinMode(buttonPin, INPUT_PULLUP); // Set button pin as input with pull-up resistor

  Serial.println("Potentiometer\tButton"); // Column labels for clarity in Serial Plotter
}
```

```
void loop() {
  // Read the button state (active low)
  buttonState = digitalRead(buttonPin);
  int ledState = (buttonState == LOW) ? 1 : 0; // If button is pressed, set ledState to 1

  // Read the potentiometer value
  potValue = analogRead(A0);
  int servoPosition = map(potValue, 0, 1023, 0, 180); // Map to servo angle (0-180)

  // Send potentiometer and button state to Serial Plotter
  Serial.print(potValue);   // Print potentiometer value
  Serial.print("\t");       // Tab separates the data series
  Serial.println(ledState); // Print button state on the same line

  // Send data to the slave for LED and servo control
  Wire.beginTransmission(8); // Address of the slave (8)
  Wire.write(ledState);      // Send LED state
  Wire.write(servoPosition); // Send servo position
  Wire.endTransmission();    // End transmission

  // Debugging output (optional)
  Serial.print("Button: ");
  Serial.print(ledState);
  Serial.print(" | Servo: ");
  Serial.println(servoPosition);

  delay(1000); // Short delay for smoother plotting
}
```

Slave code

```
#include <Wire.h> // Include the Wire library for I2C communication
#include <Servo.h> // Include the Servo library

Servo myServo; // Create a servo object
```

```
int ledPin = 7; // LED connected to pin 7
int ledState = 0;
int servoPosition = 0;

void setup() {
  Wire.begin(8);          // Start I2C communication with address 8
  Wire.onReceive(receiveEvent); // Define function to handle received data
  pinMode(ledPin, OUTPUT);  // Set LED pin as output
  myServo.attach(5);       // Attach servo to pin 9
}

void loop() {
  digitalWrite(ledPin, ledState); // Control LED based on received data
  myServo.write(servoPosition);   // Set servo position based on received data
  delay(10);                      // Short delay
}

// Function to handle data received from master
void receiveEvent(int howMany) {
  if (Wire.available() >= 2) { // Ensure at least 2 bytes of data are available
    ledState = Wire.read();    // Read LED state
    servoPosition = Wire.read(); // Read servo position
  }
}
```

Task 3 Master

```
#include <Wire.h> // Include the Wire library for I2C communication

int ledState = 0;   // Variable to store LED state (from serial input)
int servoPosition = 90; // Default servo position

void setup() {
```

```
  Serial.begin(9600); // Initialize serial communication for Serial Monitor
  Wire.begin();       // Start I2C communication as master
  Serial.println("Enter 'L' for LED control and 'S' for Servo position:");
}

void loop() {
  // Check if data is available in Serial Monitor
  if (Serial.available() > 0) {
    char command = Serial.read(); // Read the first character (command)

    if (command == 'L') {
      Serial.println("Enter LED state (0 for OFF, 1 for ON):");
      while (Serial.available() == 0); // Wait for input
      ledState = Serial.parseInt();   // Read LED state
      Serial.print("LED State: ");
      Serial.println(ledState);
    }
    else if (command == 'S') {
      Serial.println("Enter Servo position (0-180):");
      while (Serial.available() == 0); // Wait for input
      servoPosition = Serial.parseInt(); // Read servo position
      Serial.print("Servo Position: ");
      Serial.println(servoPosition);
    }
    else {
      Serial.println("Invalid command. Use 'L' or 'S'.");
    }

    // Send data to the slave
    Wire.beginTransmission(8); // Address of the slave (8)
    Wire.write(ledState);      // Send LED state
    Wire.write(servoPosition); // Send servo position
    Wire.endTransmission();    // End transmission
  }

  delay(100); // Short delay to avoid spamming
}
```

Task 4 Master Code

In the master code, you will read the button state. If the button is pressed, the master will send a signal to the slave to turn on the buzzer. When the button is not pressed, the master will send a signal to turn off the buzzer.

```
#include <Wire.h> // Include the Wire library for I2C communication

const int buttonPin = 7;  // Button connected to pin 7
int buttonState = 0;      // Variable to store button state (pressed or not pressed)

void setup() {
  Serial.begin(9600);    // Initialize serial communication
  Wire.begin();          // Start I2C communication as master
  pinMode(buttonPin, INPUT_PULLUP); // Set button pin as input with pull-up resistor
}

void loop() {
  // Read the button state (active low, pressed when button is low)
  buttonState = digitalRead(buttonPin);

  // Send the button state to the slave via I2C
  Wire.beginTransmission(8);  // Start communication with slave (address 8)
  Wire.write(buttonState);    // Send button state (0 = button not pressed, 1 = button
pressed)
  Wire.endTransmission();     // End transmission

  // Debugging output
  if (buttonState == LOW) {
    Serial.println("Button Pressed - Sending signal to turn on buzzer");
  } else {
    Serial.println("Button Released - Sending signal to turn off buzzer");
  }

  delay(100);  // Small delay to avoid spamming the slave
}
```

Task 4 Slave Code

The slave will receive the button state from the master. When the button is pressed (i.e., the received signal is 1), the slave will turn the buzzer ON using the tone() function. When the button is not pressed (i.e., the signal is 0), the buzzer will be turned OFF using the noTone() function.

```
#include <Wire.h>  // Include the Wire library for I2C communication

const int buzzerPin = 6;  // Pin connected to the buzzer
int buzzerState = 0;      // Variable to store buzzer state (on or off)

void setup() {
  Wire.begin(8);          // Join I2C bus with address 8 (slave address)
  Wire.onReceive(receiveEvent);  // Register the receive event
  pinMode(buzzerPin, OUTPUT);   // Set buzzer pin as output
}

void loop() {
  // If buzzer state is 1, turn on the buzzer
  if (buzzerState == 1) {
    tone(buzzerPin, 1000);  // Play 1 KHz tone on the buzzer
  } else {
    noTone(buzzerPin);      // Stop the buzzer tone
  }
}

void receiveEvent(int howMany) {
  // Read the button state sent by the master
  if (Wire.available()) {
    buzzerState = Wire.read();  // 0 = button not pressed, 1 = button pressed
  }
}
```

Image of our setup