MECHATRONICS SYSTEM INTEGRATION (MCTA 3203)

SECTION 2, SEMESTER 1, 2024/2025

FINAL PROJECT REPORT

INSTRUCTOR:

Wahju Sediono *(Dr.)*

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

WEEK 14:

INTEGRATION SYSTEMS (WASHING MACHINE)

No	Name (Group 6)	Matric No.
1	MUHAMMAD BASIL BIN ABDUL HAKIM	2215315
2	MUHAMMAD SYAFIQ BIN NOR AZMAN	2213187
3	MUHAMMAD HAFIZ HAMZAH BIN FANSURI	2212803
4	MUHAMMAD AMMAR ZUHAIR BIN NOR AZMAN SHAH	2110259
5	MUHAMMAD RAZIQ BIN KAHARUDDIN	2120225

INTRODUCTION

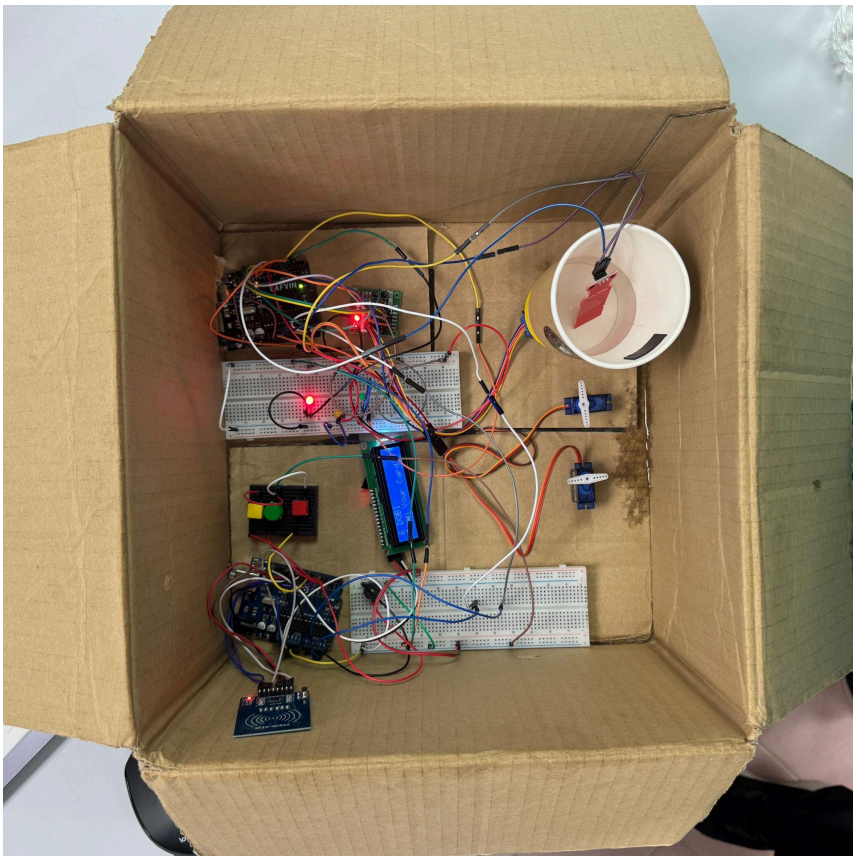
This project is all about building a smart washing machine prototype that focuses on bringing different systems together to work as one or what we called system integration. We're using two Arduino microcontrollers—one as the master and the other as the slave. They communicate with each other using the I2C protocol, which helps them share data and work in sync. The master takes care of user inputs and decides what needs to be done, while the slave handles tasks like running motors and reading sensors.

By focusing on how all these parts work together, this project shows how modern technology can make everyday appliances smarter and more efficient.

EQUIPMENTS AND INSTRUMENTS

- Microcontrollers, Arduino Uno, 2 units
- RFID reader
- RFID ID card
- LCD with I2C adapter
- PIXY camera
- Servo motors
- Push buttons
- Breadboards
- Stepper motor
- Motor module, ULN2003 driver
- Water sensor for Arduino
- Jumper wires
- Piezo buzzer

PROTOTYPE DIAGRAMS



PROTOTYPE WORKFLOW

Prototype Workflow

The smart washing machine prototype operates through the following sequence of steps:

1. **LCD Display**: A simple and user-friendly LCD interface provides real-time updates on each stage of the process, ensuring better user interaction.
2. **RFID Authorization**: The user taps an RFID card on the reader. If the card is authorized, a buzzer beeps to confirm, and the system proceeds to the next step.
3. **Start Command**: The user presses the start button to initiate the washing process.
4. **Lid Locking**: A servo motor rotates 90 degrees, acting as a lever to lock the washing machine lid securely.
5. **Detergent Dispensing**: Another servo motor rotates 360 degrees for three revolutions, simulating the action of a detergent dispenser motor to release detergent into the tub.
6. **Water Level Detection**: A water sensor monitors the water level in the tub, with three LEDs indicating the level:
    - **Green LED**: High water level
    - **Yellow LED**: Medium water level
    - **Red LED**: Low water level

      Based on the detected water level, the system determines the duration required for the washing machine to complete:
    - **1 Wash Cycle**
    - **1 Rinse Cycle**
    - **1 Spin Cycle**
7. **Pause Button for Safety**: A pause button acts as an emergency stop, immediately halting the rotation of the washing tub. This feature enhances safety by allowing users to interrupt the process if needed.
8. **Completion and Unlocking**: Once all cycles are completed, the lid lock servo motor rotates -90 degrees, returning to its initial position and unlocking the lid.

This workflow highlights the integration of RFID, sensors, indicators, and user-friendly controls, ensuring an efficient and safe operation of the washing machine prototype.

CODING

Integration of the systems being made via I2C protocol therefore there are codes for master Arduino (where the inputs are connected) and slave Arduino (where the outputs are connected).

Master coding

```
#include <SPI.h>
#include <MFRC522.h>
#include <LiquidCrystal_I2C.h>
#include <Wire.h>

#define NOTE_B4  494
#define NOTE_B5  523
#define NOTE_FS5 740
#define NOTE_DS5 622
#define REST      0

#define SLAVEP 8
#define SLAVEM 9
#define RST_PIN 5
#define SS_PIN 10

#define STRTBUTTON 3
#define STPBUTTON 8
#define RSMBUTTON 4

MFRC522 mfrc522(SS_PIN, RST_PIN);
LiquidCrystal_I2C lcd(0x27, 2, 1, 0, 4, 5, 6, 7, 3, POSITIVE);

int buzzer = 6;
```

```
int tempo = 105;
String MasterTag = "538FD52C"; // Replace with your Tag ID
String tagID = "";

int melody[] = {
  NOTE_B4, 16, NOTE_B5, 16, NOTE_FS5, 16, NOTE_DS5, 16
};

int notes = sizeof(melody) / sizeof(melody[0]) / 2;
int wholenote = (60000 * 4) / tempo;
int divider = 0, noteDuration = 0;

void setup() {
  Serial.begin(9600);
  Wire.begin();
  pinMode(STRTBUTTON, INPUT_PULLUP);
  pinMode(STPBUTTON, INPUT_PULLUP);
  pinMode(RSMBUTTON, INPUT_PULLUP);

  SPI.begin();
  mfrc522.PCD_Init();
  lcd.begin(16, 2);

  lcd.print("Welcome to");
  lcd.setCursor(0, 1);
  lcd.print("6 DOBI");
  delay(3000);

  lcd.clear();
  lcd.print("6 DOBI");
  lcd.setCursor(0, 1);
  lcd.print("Scan Your Card>>");
  Serial.println("Tap the card");
}

void loop() {
  if (getID()) {
    lcd.clear();
    lcd.setCursor(0, 0);
```

```
    if (tagID == MasterTag) {
      lcd.print("Access Granted!");
      song();
      Serial.println("Access Granted!");
      delay(3000);

      lcd.clear();
      lcd.print("Press START...");
      waitForStart();

      while (true) {
        checkButtons();
      }
    } else {
      lcd.print("Access Denied!");
      lcd.setCursor(0, 1);
      lcd.print("Use other card!");
      Serial.println("Access Denied!");
      delay(3000);

      lcd.clear();
      lcd.print("6 DOBI");
      lcd.setCursor(0, 1);
      lcd.print("Scan Your Card>>");
      Serial.println("Tap the card");
    }
  }
}

boolean getID() {
  if (!mfrc522.PICC_IsNewCardPresent()) return false;
  if (!mfrc522.PICC_ReadCardSerial()) return false;

  tagID = "";
  for (uint8_t i = 0; i < 4; i++) {
    tagID.concat(String(mfrc522.uid.uidByte[i], HEX));
  }
  tagID.toUpperCase();
  mfrc522.PICC_HaltA();
  return true;
```

```
}

void song() {
  for (int thisNote = 0; thisNote < notes * 2; thisNote += 2) {
    divider = melody[thisNote + 1];
    if (divider > 0) {
      noteDuration = wholenote / divider;
    } else if (divider < 0) {
      noteDuration = wholenote / abs(divider);
      noteDuration *= 1.5;
    }

    tone(buzzer, melody[thisNote], noteDuration * 0.9);
    delay(noteDuration);
    noTone(buzzer);
  }
}

void waitForStart() {
  while (true) {
    if (digitalRead(STRTBUTTON) == LOW) {
      sendCommandToSlave(SLAVEP, "START");
      lcd.clear();
      lcd.print("Running...");
      delay(40000);
      lcd.clear();
      lcd.print("Cycle complete");
      return;
    }
  }
}

void checkButtons() {
  if (digitalRead(STPBUTTON) == LOW) {
    sendCommandToSlave(SLAVEP, "STOP");
    lcd.clear();
    lcd.print("Stopped!");
    delay(3000);
    resetSystem();
  } else if (digitalRead(RSMBUTTON) == LOW) {
```

```
    sendCommandToSlave(SLAVEP, "RESUME");
    lcd.clear();
    lcd.print("Restart...");
    delay(500);
  } else if (digitalRead(STRTBUTTON) == LOW) {
    sendCommandToSlave(SLAVEP, "PAUSE");
    lcd.clear();
    lcd.print("Paused...");
    delay(500);
  }
}

void resetSystem() {
  lcd.clear();
  lcd.print("6 DOBI");
  lcd.setCursor(0, 1);
  lcd.print("Scan Your Card>>");
  tagID = "";  // Reset RFID state
}

void sendCommandToSlave(int address, const char *command) {
  Wire.beginTransmission(address);
  Wire.write(command);
  Wire.endTransmission();
}
```

Slave coding

```
#include <Wire.h>
#include <Stepper.h>
#include <Servo.h>

#define stepsPerRevolution 2038
#define lowWaterLED 2
#define mediumWaterLED 3
#define highWaterLED 4
#define SENSOR_PIN A0
```

```
#define SENSOR_POWER_PIN 7

Servo myServo1;  // Continuous Servo
Servo myServo2;  // Normal Servo
Stepper myStepper(stepsPerRevolution, 8, 10, 9, 11);

int waterLevel = 0;
float rotationMultiplier = 1.0;
volatile bool motorRunning = false;
volatile bool paused = false;
volatile bool stopped = false;

void setup() {
  Serial.begin(9600);
  myServo1.attach(6);
  myServo2.attach(5);
  myServo2.write(0);
  myServo1.write(90);
  myStepper.setSpeed(15);
  pinMode(lowWaterLED, OUTPUT);
  pinMode(mediumWaterLED, OUTPUT);
  pinMode(highWaterLED, OUTPUT);
  pinMode(SENSOR_POWER_PIN, OUTPUT);
  digitalWrite(lowWaterLED, LOW);
  digitalWrite(mediumWaterLED, LOW);
  digitalWrite(highWaterLED, LOW);
  digitalWrite(SENSOR_POWER_PIN, LOW);  // Start with sensor off
  Wire.begin(8);  // Slave address is 8
  Wire.onRequest(requestEvent);
  Wire.onReceive(receiveEvent);
  Serial.println("Slave Initialized...");
}

void loop() {
  if (stopped) {
    stopAll();
    return;  // Halt further execution
  }

  if (motorRunning && !paused) {
```

```
    performCycle();
  }
}

void performCycle() {
  rotateServo2(0);
  rotateServo2(90);
  spinServo1(5);
  rotateServo2(0);

  if (stopped) return;  // Stop mid-cycle if STOP is pressed

  waterLevel = readWaterLevel();
  operateStepperMotor(waterLevel);
  operateLEDs(waterLevel);

  if (stopped) return;  // Stop mid-cycle if STOP is pressed

  delay(5000);  // Pause before rinse cycle
  operateStepperMotor(waterLevel);
  motorRunning = false;  // Stop after the cycle
  Serial.println("Cycle complete. Waiting for START command.");
}

void rotateServo2(int angle) {
  if (stopped) return;  // Stop immediately
  myServo2.write(angle);
  delay(1000);  // Wait for servo to move
}

void spinServo1(int seconds) {
  if (stopped) return;  // Stop immediately
  myServo1.write(180);
  delay(seconds * 1000);
  myServo1.write(90);  // Stop the servo
}

int readWaterLevel() {
  digitalWrite(SENSOR_POWER_PIN, HIGH);  // Power on the sensor
  delay(100);  // Allow sensor to stabilize
```

```
  int level = analogRead(SENSOR_PIN);
  digitalWrite(SENSOR_POWER_PIN, LOW);  // Power off the sensor
  return level;
}

void operateStepperMotor(int level) {
  if (stopped) return;  // Stop immediately
  int speed = (level <= 200) ? 5 : (level <= 500) ? 10 : 15;
  myStepper.setSpeed(speed);
  myStepper.step(stepsPerRevolution * rotationMultiplier);
  delay(1000);
  myStepper.step(-stepsPerRevolution * rotationMultiplier);
  delay(1000);
  Serial.println("Stepper motor cycle complete.");
}

void operateLEDs(int level) {
  if (level <= 200) {
    digitalWrite(lowWaterLED, HIGH);
    digitalWrite(mediumWaterLED, LOW);
    digitalWrite(highWaterLED, LOW);
    Serial.println("Low water level detected. Red LED ON.");
  } else if (level > 200 && level <= 500) {
    digitalWrite(lowWaterLED, LOW);
    digitalWrite(mediumWaterLED, HIGH);
    digitalWrite(highWaterLED, LOW);
    Serial.println("Medium water level detected. Yellow LED ON.");
  } else {
    digitalWrite(lowWaterLED, LOW);
    digitalWrite(mediumWaterLED, LOW);
    digitalWrite(highWaterLED, HIGH);
    Serial.println("High water level detected. Green LED ON.");
  }
}

void stopAll() {
  motorRunning = false;
  paused = false;

  // Stop all motors and reset LEDs
```

```
  myServo1.write(90);  // Stop continuous servo
  myServo2.write(0);   // Reset servo to 0 degrees
  myStepper.setSpeed(0);  // Stop stepper
  digitalWrite(lowWaterLED, LOW);
  digitalWrite(mediumWaterLED, LOW);
  digitalWrite(highWaterLED, LOW);
  Serial.println("System stopped.");
}

void requestEvent() {
  Wire.write(motorRunning ? 1 : 0);
}

void receiveEvent(int bytes) {
  char command[10];
  int index = 0;
  while (Wire.available() > 0 && index < sizeof(command) - 1) {
    command[index++] = Wire.read();
  }
  command[index] = '\0';

  if (strcmp(command, "START") == 0) {
    motorRunning = true;
    paused = false;
    stopped = false;
    Serial.println("Motor started.");
  } else if (strcmp(command, "PAUSE") == 0) {
    motorRunning = false;
    paused = true;
    Serial.println("Motor paused.");
  } else if (strcmp(command, "RESUME") == 0) {
    motorRunning = true;
    paused = false;
    Serial.println("Motor resumed.");
  } else if (strcmp(command, "STOP") == 0) {
    stopped = true;
    Serial.println("Emergency stop activated!");
  } else {
    Serial.println("Unknown command received.");
  }
```

```
}
```

1. Purpose

- The code implements a control system for motors, water-level sensors, and LED indicators.
- It uses I2C communication to receive commands and respond with the system's status.

2. Libraries Used

- Wire.h: Enables I2C communication for external device interfacing.
- Stepper.h: Controls the stepper motor with precise movements.
- Servo.h: Manages the movement of two servos (continuous and normal).

3. Hardware Components

- LED Indicators:
  - Show water levels (low, medium, high).
- Water Level Sensor:
  - Measures the water level using an analog signal.
- Stepper Motor:
  - Executes rotational tasks with variable speed and direction.
- Two Servos:
  - Perform specific mechanical actions.
- I²C Communication:
  - Receives commands (START, STOP, etc.) and sends the motor state.

4. Key Constants and Variables

- Constants: stepsPerRevolution, pin numbers for components, and default servo angles.
- Variables:

    - waterLevel: Stores the water sensor reading.
    - motorRunning, paused, stopped: Boolean flags for system states.

5. Setup (setup Function)

- Initializes components (motors, LEDs, I2C serial communication).
- Configures pins as input or output.
- Ensures components are in a safe starting state.

6. Main Logic (loop Function)

- Checks system states (stopped, paused, motorRunning) and executes actions accordingly.
- If stopped, halts execution and resets the system.
- If motorRunning and not paused, performs a complete operational cycle.

7. Core Functions

- performCycle:
    - Executes a sequence of servo rotations, stepper motor operations, and water level checks.
- rotateServo2:
    - Adjusts the normal servo to a specific angle.
- spinServo1:
    - Operates the continuous servo for a set duration.
- readWaterLevel:
    - Activates the sensor, reads the analog value, and deactivates the sensor.
- operateStepperMotor:
    - Controls the stepper motor speed and direction based on water levels.
- operateLEDs:
    - Illuminates the appropriate LED based on the current water level.

8. Emergency Stop (stopAll Function)

- Stops all motors and resets LEDs.
- Resets the system to a safe idle state.

9. I2C Communication

- requestEvent: Sends the motorRunning status when queried by an external device.
- receiveEvent: Processes commands (START, PAUSE, RESUME, STOP) to control the system.

10. Safety and Efficiency

- Safety Features:
    - Immediate stopping of motors during emergencies.
    - Prevents operations when the system is paused or stopped.
- Efficiency:
    - Powers the water level sensor only when in use.
    - Adapts motor speeds dynamically based on water level readings.

11. Applications

- Automated control systems for water-based processes.
- Demonstrations of multitasking and real-time responsiveness in embedded systems.
- Educational projects showcasing motor and sensor integration.

RESULTS

The prototype of the Arduino-based smart washing machine was successfully built, using the I2C communication protocol to connect a Master and a Slave Arduino. The Master Arduino handled user interactions, including an RFID system for authentication, an LCD screen for real-time updates, and a buzzer for sound alerts. It also processed button inputs for starting, stopping, and resuming the washing process. The Master's main role was to send instructions to the Slave Arduino to carry out washing functions.

The Slave Arduino controlled the machine's mechanical parts and water levels. It measured the water level with a sensor and used LED lights to indicate the amount of water present. It also operated a stepper motor for drum rotation, a continuous servo motor for water flow control, and a standard servo motor for releasing detergent. The Slave received and executed commands from the Master, ensuring smooth operation of the washing cycle.

The I2C communication between the two Arduinos was stable and efficient, allowing them to exchange data quickly with minimal delay. The Master Arduino successfully sent commands for different washing stages, while the Slave responded with updates on motor operation and water levels. During testing, the system worked reliably, with no major communication errors.

In conclusion, the prototype successfully demonstrated an automated washing machine system using Arduino. The combination of RFID authentication, motor control, and water level monitoring created a functional and easy-to-use design. Future improvements could include wireless connectivity, smartphone integration, and AI-based washing cycle optimization to enhance efficiency and user experience.

RECOMMENDATION

While the current prototype demonstrates effective system integration, there are several enhancements that could be implemented to further improve its functionality:

1. **Color Detection with Pixy Camera or Color Sensor**:
    - A Pixy Camera or color sensor could be incorporated to detect white clothing. The system could be trained to recognize white clothes and issue a warning on the LCD display if white garments are detected mixed with colored clothes.
    - Since the Pixy Camera shares the same communication bus as the RFID module, an external microcontroller could be used to manage the camera's operation and ensure smooth communication within the system.

2. **Water Temperature Sensing**:
    - A temperature sensor such as the LM35 or DS18B20 could be added to monitor the water temperature. This feature would allow the system to suggest suitable water temperature modes to the user based on the detected temperature, enhancing usability and providing better care for fabrics.

3. **Graphical User Interface (GUI)**. Instead of relying solely on an LCD display, a dedicated **smartphone application** could be developed to provide a more user-friendly interface for controlling and monitoring the washing machine. This app could display washing cycle progress, allow users to select washing modes, and send notifications when the cycle is complete.

    To establish communication between the washing machine and the smartphone app, **Bluetooth or Wi-Fi** connectivity could be integrated. **Bluetooth** would be a suitable option for short-range, low-energy communication, while **Wi-Fi** could enable remote access, allowing users to control the washing machine from anywhere. By implementing this feature, the system would become more convenient and accessible, improving the overall user experience.

These enhancements would expand the system's capabilities and make the washing machine smarter, more user-friendly, and better equipped to handle various laundry needs.

CONCLUSION

The washing machine system was successfully built and worked well, showing how I2C communication can be used effectively in embedded systems. By using a master controller to manage several slave devices, the system ran smoothly and efficiently.

This project demonstrates how embedded systems can create smart and practical solutions for everyday tasks. The results show that modular design and I2C communication are effective for managing data exchange between devices. Future improvements could focus on making communication faster, improving sensor accuracy, and making the system more scalable for complex situations. While the system is operated well, any unexpected dangers should be taken into account with some safety features that can be improved by time.

In conclusion, the project achieved its goals and created a working smart washing machine prototype, opening the door for future development and innovation in embedded systems.

REFERENCES

Master Slave I2C Connection
https://projecthub.arduino.cc/PIYUSH_K_SINGH/master-slave-i2c-connection-31a095

ACKNOWLEDGEMENTS

Special thanks to Dr. ZULKIFLI BIN ZAINAL ABIDIN and Dr.WAHJU SEDIONO for their guidance and support during this project especially and throughout this whole course.