TECHNICAL UNIVERSITY OF MOLDOVA
FACULTY OF COMPUTERS, INFORMATICS AND MICROELECTRONICS
DEPARTMENT OF SOFTWARE ENGINEERING AND AUTOMATICS

Laboratory work nr. 4

User interaction - Actuatori - DC Motor + Releu

A elaborat: **Brinza Cristian**
st. gr. FAF-212

A verificat: **Moraru Dumitru**
lect. univ

Chișinău 2024

THE TASK OF THE LABORATORY WORK:

To create an MCU-based application that will control the actuators with commands received from the serial interface and reporting to the LCD.

The actuation devices will be the following:

- an electric bulb through the relay with ON and OFF commands
- a direct current motor with commands to set the motor power between (-100% .. 100%), i.e. forward and backward, and the speed via the L298 driver

Peripheral control drivers will be realized on abstraction levels

To create an MCU-based application that will control the actuation devices with commands received from the serial interface and reporting to the LCD.

Marking:

- 5 - simple device activation application
- +1 - for implementing motor control commands from the serial interface
- +1 - for implementing actuator status display on the LCD
- +1 - for layered implementation of the DC motor control driver
- +1 - for demonstrating evidence of physical implementation

Penalties:

- 1 - penalty for NOT using STDIO
- 1 - penalty for each week late from the deadline
- 1 - penalty for non-compliance with the report format

PROGRESS OF THE WORK

1 Main functions/methods used to execute the task

Explication about this chapterIn this chapter I will explain the functionality of diferent parts of the executed task

In the Main Sketch File:

- *setup():* This function initializes the Arduino's serial communication, sets up the LCD display via the LiquidCrystal_I2C library, and attaches the motor pin for servo control. It ensures that the relay is in the off position at the start, indicating the system's readiness for operations. The setup function plays a pivotal role in preparing the system by configuring the necessary peripherals and variables before entering the main program loop.
- *loop():* Acting as the core of the Arduino sketch, this function continuously monitors for serial commands. Based on the received commands, it executes corresponding actions, such as toggling the bulb state or adjusting the motor's power. The loop function is crucial for interactive control, allowing dynamic responses to serial inputs.

In Device Control and Status Reporting:

- *setBulbState(int state):* Controls the on/off state of the bulb through the relay. It updates the bulb's state, both physically by toggling the relay and visually by updating the LCD display and serial output. This method exemplifies direct device control and feedback mechanisms.
- *setMotorPower(int power):* Adjusts the servo motor's position based on a specified power level. It demonstrates the mapping of abstract command inputs (like power levels) to physical actions (servo positions) and updates the system status on the LCD and serial monitor.
- *handleSerialCommand(String command):* Interprets serial commands to perform specific actions, such as turning the bulb on/off or setting the motor's power. This function is a prime example of how user inputs are translated into control signals for various components of the system.

In Display Management:

- *LiquidCrystal_I2C lcd:* Utilized for initializing and managing the LCD display. Commands such as lcd.init(), lcd.clear(), and lcd.print() are used extensively to provide real-time feedback about the system's status, including the current state of the bulb and the power level of the motor.

Explanation of Chapters

This section has outlined the implementation and functionality of the main sections of our code. Through the setup and loop functions, the sketch manages to establish a responsive system capable of interpreting serial commands for controlling a bulb and a motor. The integration

of direct control methods, along with real-time feedback on the LCD display, exemplifies a practical approach to interactive device management in a laboratory setting.

Device control functions such as setBulbState and setMotorPower highlight the application's ability to affect physical changes through digital inputs, a core aspect of automation and control systems. Meanwhile, the display management techniques employed ensure that the user remains informed about the system's current status, fostering an intuitive interaction between the user and the system.

2 Block Diagram

The diagram is consisting of the following main blocks:

- **Main Program (sketch.ino):** The central point that initializes and controls other components (Keypad, LCD Display, and LEDs) and manages the program flow.
- **KeypadControl Class**: Represents the logic for interfacing with the keypad, including setup, password verification, input handling, and feedback display.
- **LcdDisplay Class:** Handles the initialization and control of the LCD display, including setup and message display.
- **LEDControl Class:** Manages the LED indicators for indicating lock status, including initialization and control.
- **Arduino Board:** The physical layer where the keypad, LCD display, and LEDs are connected

The Figure 1 depicted the UML program flow.

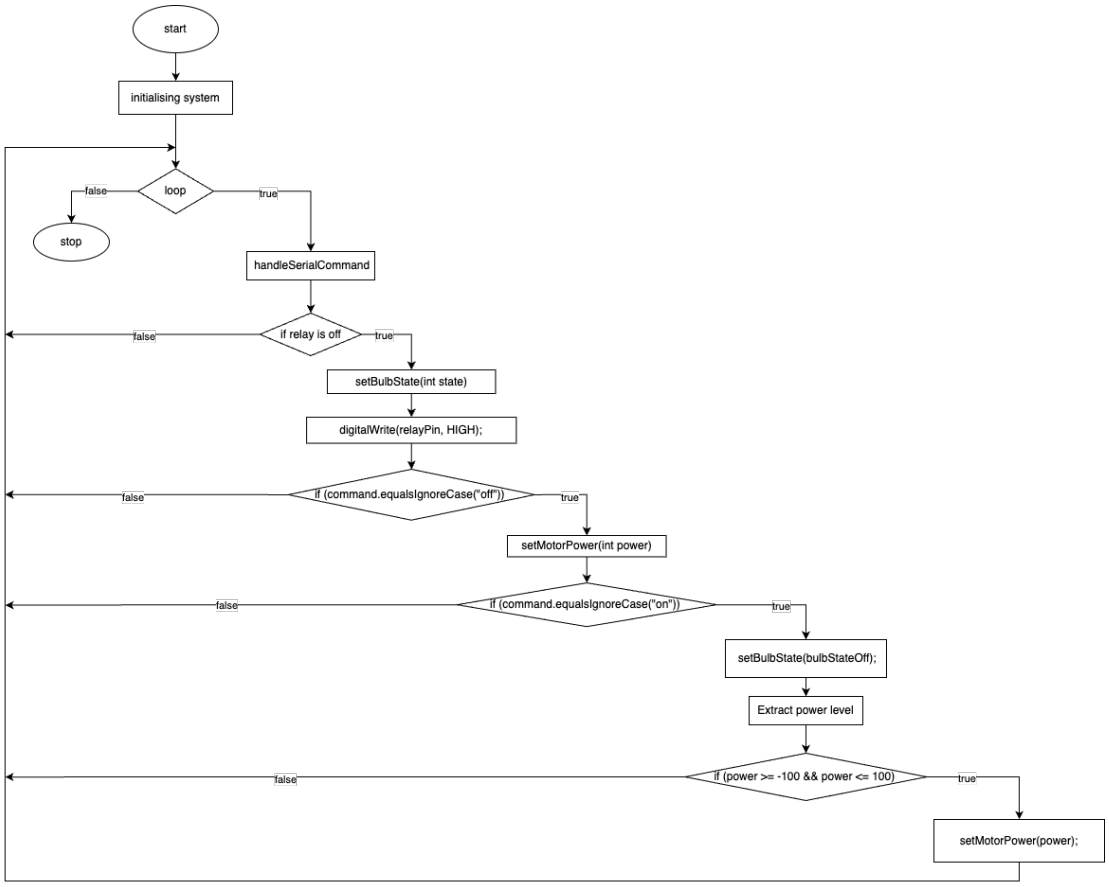

Figure 1 Program schema.

3 Simulated or real assembled electrical schematic diagram :

The Figure 2 depicted the phisical board schema.

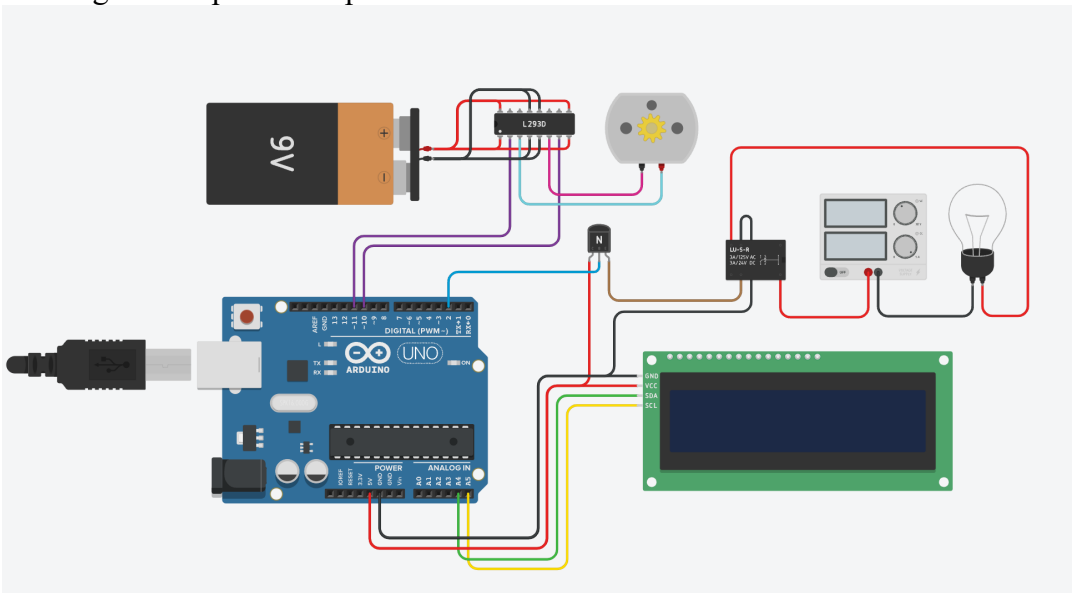

Figure 2 Phisical board schema.

4 Screenshots of the simulation execution:

The Figure 3 depicted a sceenshot of the TINKERCAD simulation :

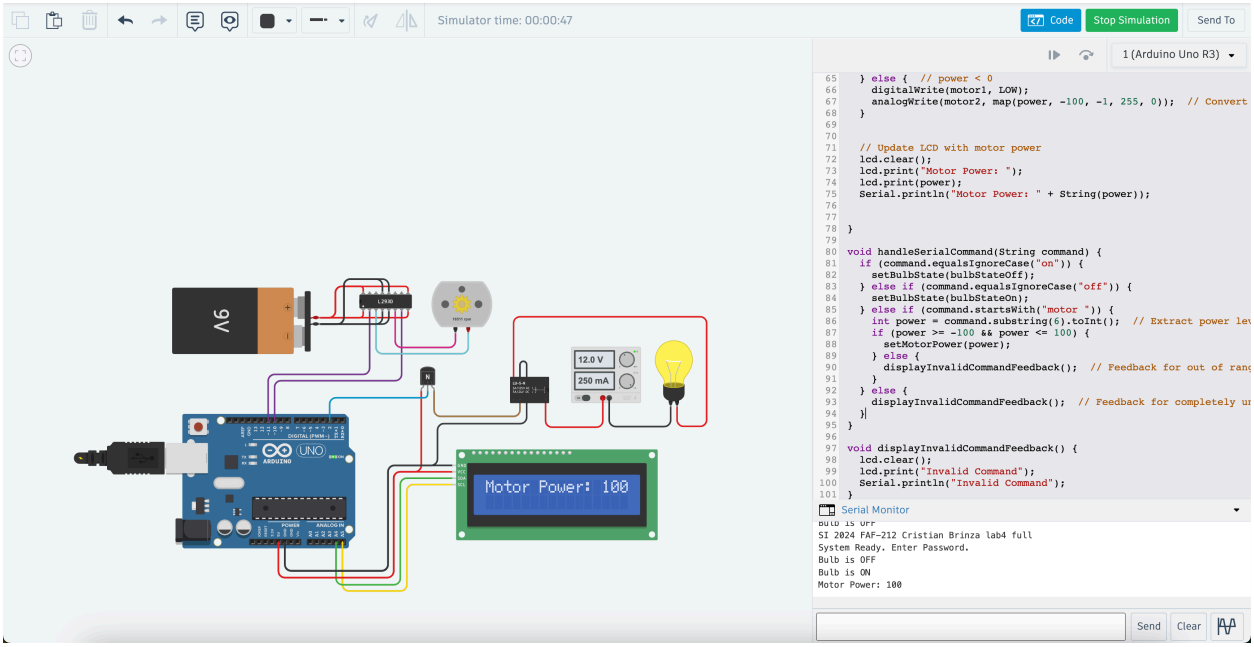

Figure 3 Screenshot of the simulation.

CONCLUSION

This project embarked on the innovative task of creating a dynamic control system, utilizing the versatile Arduino platform to manage a relay-controlled bulb and a servo motor, alongside real-time status updates on an LCD display. The journey from conceptualization to realization has been both challenging and rewarding, offering deep insights into the integration of hardware and software to achieve precise control and feedback mechanisms.

Throughout the development process, we adhered to best coding practices, emphasizing readability and maintainability. The use of clear variable names and structured functions facilitated a seamless debugging experience and ensured that our system was both efficient and scalable. A notable highlight of our project was the successful interpretation and execution of serial commands to control physical devices, demonstrating the power of interactive systems in real-world applications.

The Arduino ecosystem, with its extensive libraries and community support, played a crucial role in the swift development and troubleshooting of our system. This project underscored the importance of understanding both the limitations and capabilities of each component, from the mechanics of the servo motor to the nuances of serial communication.

In conclusion, this project was not just a technical achievement but a comprehensive learning venture that covered the spectrum of embedded system design, from software logic to hardware manipulation. The skills honed and the knowledge gained throughout this project provide a robust foundation for future endeavors in automation and control systems. The successful implementation of this control system stands as a testament to our ability to bridge the gap between theoretical concepts and practical applications, sparking a continued interest in exploring the vast possibilities within the realm of embedded systems.

BIBLIOGRAPHY

1 EDUCBA: *Introduction to Embedded Systems.* Cursuri electronice online ©2020 [quote 10.02.2024] Available: https://www.educba.com/what-is-embedded-systems/

2 ARDUINO: *Arduino UNO.* The official website of Arduino modules, ©2020 [quote 10.02.2024]. Available: https://www.arduino.cc/.

3 TUTORIALSPOINT: *Embedded Systems Tutorial*. Comprehensive guide for beginners and professionals to learn Embedded System basics to advanced concepts, including microcontrollers, processors, and real-time operating systems. ©2023 [quote 10.02.2024] Available: https://www.tutorialspoint.com/embedded_systems/index.htm

4 GURU99: *Embedded Systems Tutorial: What is, History & Characteristics*. A detailed introduction to embedded systems, covering microcontrollers, microprocessors, and the architecture of embedded systems, as well as their applications and advantages. ©2023 [quote 10.02.2024] Available: https://www.guru99.com/embedded-systems-tutorial.html

5 JAVATPOINT: *Embedded Systems Tutorial.* Offers basic and advanced concepts of Embedded System, designed for beginners and professionals. ©2023 [quote 10.02.2024] Available: https://www.javatpoint.com/embedded-systems-tutorial

6 DEEPBLUE*: Embedded Systems Tutorials Introduction* | Embedded Systems Online Course. ©2023 [quote 10.02.2024] Available: https://deepbluembedded.com/embedded-systems-tutorials/

APPENDIX 1

Code of main.ino:

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

// Constants
const int relayPin = 2;
const int lcdAddress = 0x27;
const int bulbStateOff = 0;
const int bulbStateOn = 1;
int motor1=11;
int motor2=10;

// Global variables
int bulbState = bulbStateOff;

// LCD initialization
LiquidCrystal_I2C lcd(lcdAddress, 16, 2);

void setup() {
  pinMode(motor1, OUTPUT);
  pinMode(motor2, OUTPUT);
  pinMode(relayPin, OUTPUT);
  digitalWrite(relayPin, HIGH); // Start with the relay off
  Serial.begin(9600); // Initialize serial communication
  lcd.init(); // Initialize the LCD
  lcd.backlight();
  lcd.print("Bulb is ON");

  // Print lab and system readiness information on the serial monitor
  Serial.println(F("SI 2024 FAF-212 Cristian Brinza lab4 full"));
  Serial.println(F("System Ready. Enter Password."));
}

void loop() {
  if (Serial.available() > 0) {
    String command = Serial.readStringUntil('\n');
    handleSerialCommand(command);
  }
}

void setBulbState(int state) {
  if (state == bulbStateOn) {
    digitalWrite(relayPin, LOW); // Turn on the bulb
    bulbState = bulbStateOn;
    lcd.clear();
    lcd.print("Bulb is OFF");
```

```
    Serial.println("Bulb is OFF");
  } else {
    digitalWrite(relayPin, HIGH); // Turn off the bulb
    bulbState = bulbStateOff;
    lcd.clear();
    lcd.print("Bulb is ON");
    Serial.println("Bulb is ON");
  }
}

void setMotorPower(int power) {
  // Map the power from -100...0 to 0...100 for PWM
  if (power == 0) {
    digitalWrite(motor1, LOW);
    digitalWrite(motor2, LOW);
  } else if (power > 0) {
    analogWrite(motor1, map(power, 1, 100, 0, 255));  // Convert power to PWM
signal
    digitalWrite(motor2, LOW);
  } else {  // power < 0
    digitalWrite(motor1, LOW);
    analogWrite(motor2, map(power, -100, -1, 255, 0));  // Convert power to PWM
signal
  }


  // Update LCD with motor power
  lcd.clear();
  lcd.print("Motor Power: ");
  lcd.print(power);
  Serial.println("Motor Power: " + String(power));


}

void handleSerialCommand(String command) {
  if (command.equalsIgnoreCase("on")) {
    setBulbState(bulbStateOff);
  } else if (command.equalsIgnoreCase("off")) {
    setBulbState(bulbStateOn);
  } else if (command.startsWith("motor ")) {
    int power = command.substring(6).toInt();  // Extract power level
    if (power >= -100 && power <= 100) {
      setMotorPower(power);
    } else {
      displayInvalidCommandFeedback();  // Feedback for out of range values
    }
```

```
  } else {
    displayInvalidCommandFeedback();   // Feedback for completely unrecognized
commands
  }
}

void displayInvalidCommandFeedback() {
  lcd.clear();
  lcd.print("Invalid Command");
  Serial.println("Invalid Command");
}
```