Ministerul Educației și Cercetării al Republicii Moldova

Universitatea Tehnică a Moldovei

Facultatea Calculatoare, Informatică şi Microelectronică

Departamentul Inginerie Software și Automatică

Report

Laboratory work nr. 4.2

Actuatori Analogici – Control Motor DC

Embedded Systems

Prepared by:

Berco Andrei, FAF-221

Checked by:

Martîniuc Alexei, *university assistant*

Chișinău – 2025

1. OBJECTIVES

Main purpose of the work: Developing a modular application for a microcontroller (MCU) to control a DC motor using the L298 driver. Commands will be received through the STDIO interface (serial terminal or keypad), while also reporting the motor's parameters via the serial interface and/or LCD display.

The objectives of the work:

- Understanding the principles of DC motor control using the L298 driver
- Implementing analog control (PWM) for regulating power applied to a DC motor
- Using the STDIO library for receiving user commands and reporting state parameters
- Developing peripheral drivers (L298, motor, LCD) with clear abstraction levels
- Documenting the software architecture and presenting block diagrams and electrical schematics as part of the project design methodology

Problem Definition
Develop an MCU-based application that controls a DC motor via an L298 driver while meeting the following requirements:

- Commands received through STDIO (serial or keypad):
    - motor set [-100 .. 100] - Direct setting of motor power and direction
    - motor stop - Immediate motor stop
    - motor max - Setting power to maximum (100%) in the current direction
    - motor inc - Incremental increase of motor power by +10%
    - motor dec - Incremental decrease of motor power by -10%

- PWM motor control for proportional power regulation
- Periodic or event-triggered reporting of motor parameters
- Modular structure with clear abstraction levels for peripheral drivers

2. DOMAIN ANALYSIS

1. Technology stack

In order to complete this laboratory work, I used the following hardware components:

- Arduino UNO microcontroller
- DC Motor
- L298N motor driver module
- External power supply (6-12V DC)
- Jumper wires
- Breadboard
- USB connection for programming and serial communication

For the software layer, I used:

- Visual Studio Code with PlatformIO extension for embedded development
- FreeRTOS for task management
- Standard Arduino libraries
- Custom STDIO redirection for serial communication

2. Use cases

A potential application of this DC motor control system is in robotics or automated manufacturing systems. For example, in a conveyor belt system, precise speed control of motors is essential to maintain consistent product flow. The system allows operators to adjust the speed and direction of conveyor belt motors through simple commands, making it easy to adapt to different product types or production rates.

In a robotics application, this system could control the wheels of a mobile robot, allowing for precise movement control with variable speeds in both forward and backward directions. The ability to incrementally adjust speed helps in fine-tuning the robot's movement for different terrains or tasks.

Additionally, this motor control system could be used in home automation for controlling motorized curtains, blinds, or gates, where different speeds might be required depending on the time of day or user preferences.

Architectural Design

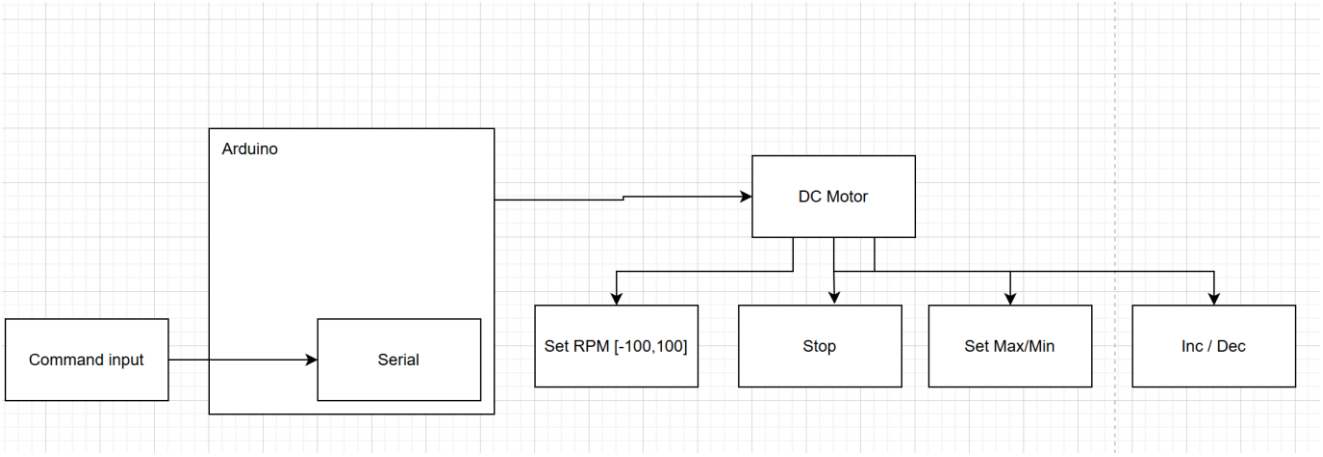
The architectural design of this system aims to create a modular, scalable, and maintainable framework for controlling binary actuators (such as lights or fans) through a relay, while providing feedback and user interaction via a serial interface or keypad. The system is built around the core principles of modularity, abstraction, and clarity, ensuring that it can be easily modified or extended in the future.

- Microcontroller Unit (MCU): The MCU is the heart of the system, managing input/output (I/O) operations, processing user commands, and controlling peripherals. It handles all of the logic, such as receiving commands, controlling the relay, and providing status feedback.
- Relay Module: The relay serves as the physical interface between the MCU and the actuator (light, fan, etc.). The relay can switch high-voltage or high-current devices on and off based on the signals from the MCU, which are generated via digital output pins.
- Keypad (Optional): The keypad serves as an alternative input device, allowing the user to input commands (e.g., relay on or relay off) via button presses. The keypad module scans the pressed keys and passes the corresponding command to the MCU.
- LCD Display (Optional): The LCD display shows real-time feedback to the user. It displays

the status of the actuator (whether it's on or off) and other relevant information, such as confirmation messages or errors.

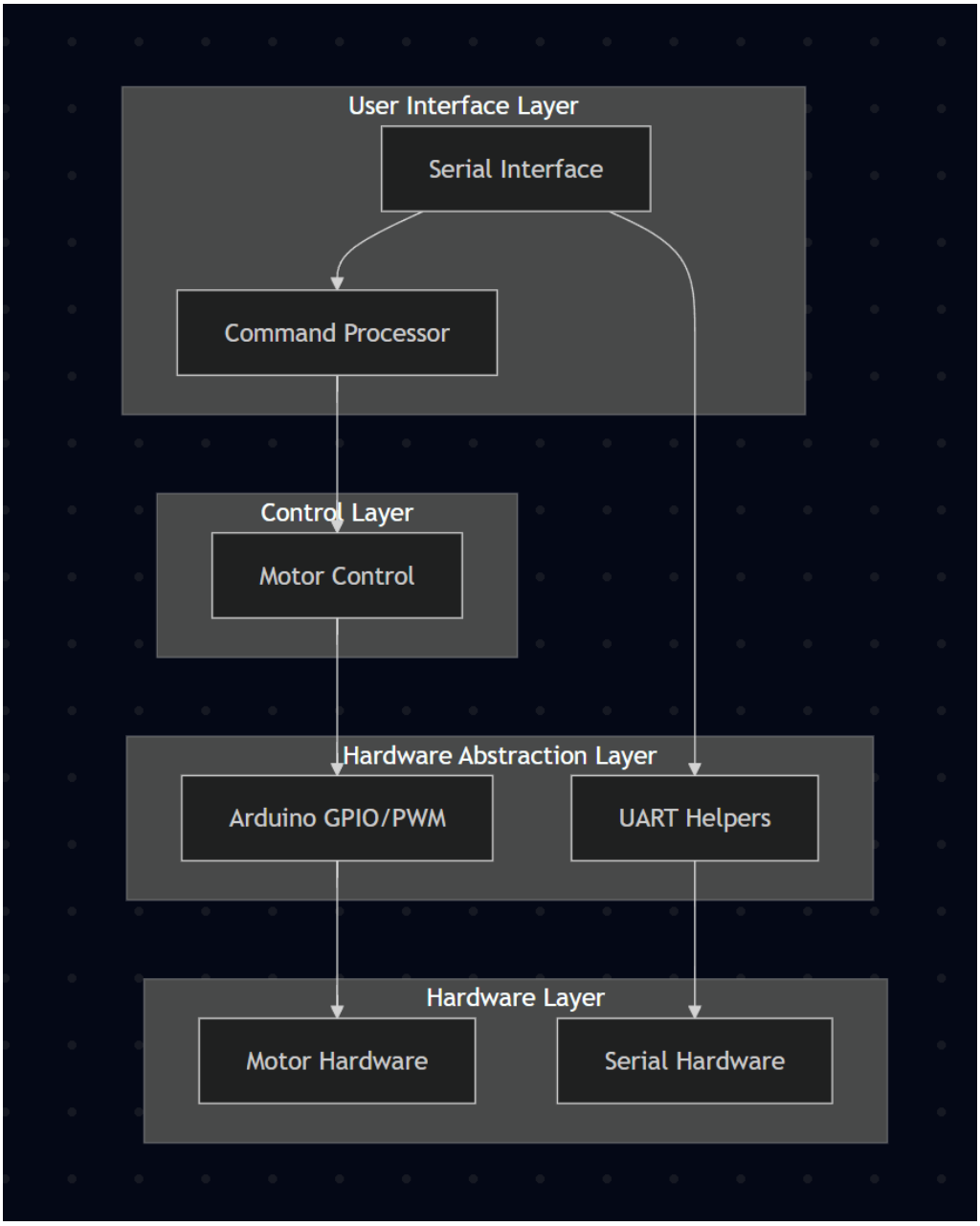
- Serial Interface: The serial interface (usually connected to a PC or terminal) allows the user to communicate with the MCU through text-based commands. This provides an alternate way to control the relay, making the system suitable for debugging and general use.
- DC Motor

Hardware Diagram



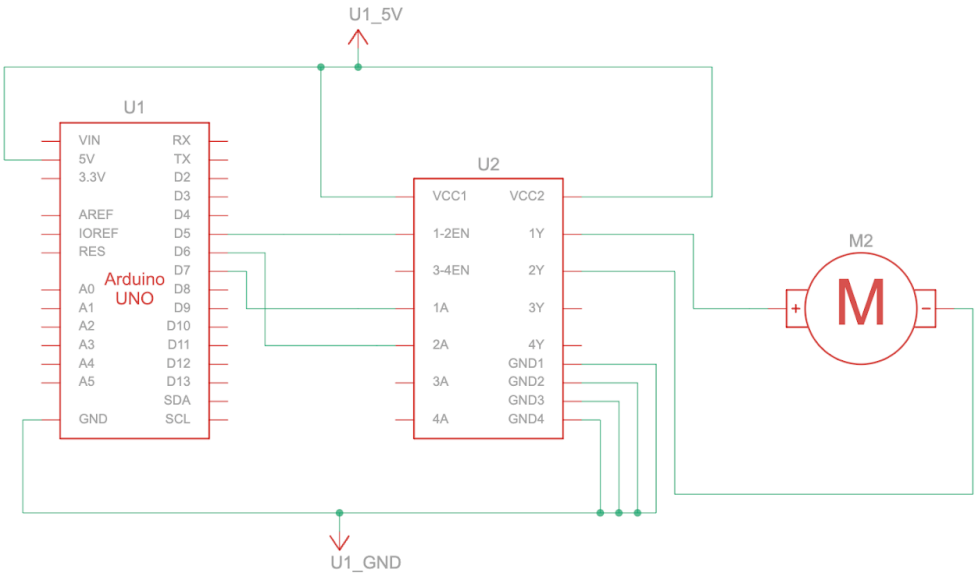
Serial receives commands from user, motor set *int*, motor stop, motor inc/dec and processes it. Next, according to the received command, the DC Motor executes it. For example, if motor receives the motor max command, the RPMs of the motor are set to the maximal value, that beinig 100%.

Layer Diagram

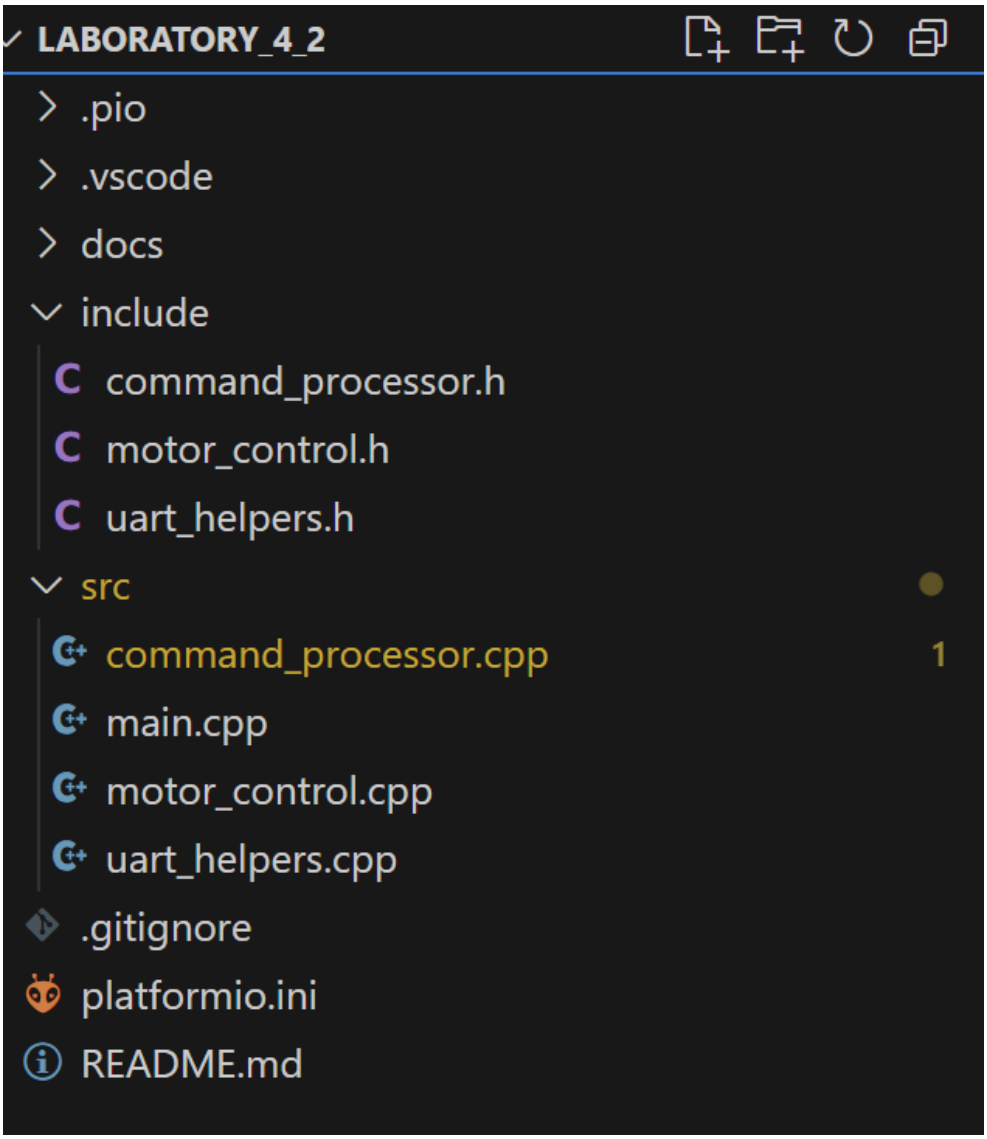


This diagram illustrates the system's architecture in layers. At the top, we have the User Interface Layer handling user commands through the Serial Interface and Command Processor. The Control Layer contains the Motor Control module which manages motor behavior. The Hardware Abstraction Layer provides interfaces to the hardware components through UART Helpers and Arduino GPIO/PWM functions. Finally, the Hardware Layer represents the physical components: the motor and serial hardware.

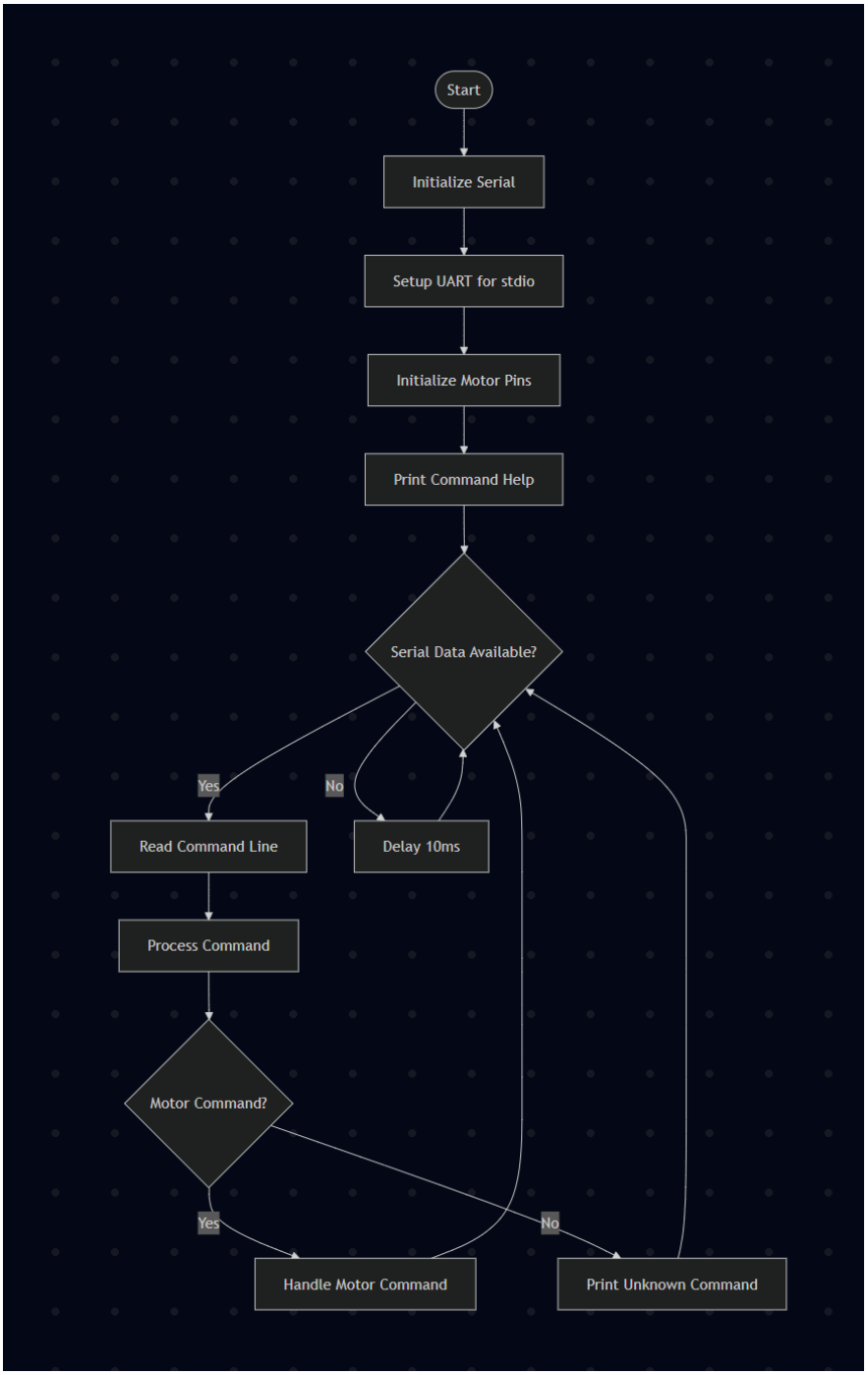
Electrical Scheme


U1_5V
U1
VIN
5V
3.3V
AREF
IOREF
RES
A0
A1
A2
A3
A4
A5
GND
RX
TX
D2
D3
D4
D5
D6
D7
D8
D9
D10
D11
D12
D13
SDA
SCL
Arduino
UNO
U2
VCC1
VCC2
1-2EN
1Y
3-4EN
2Y
1A
3Y
2A
4Y
GND1
GND2
GND3
GND4
3A
4A
M2
M
+
-
U1_GND

Project Structure and Modular Implementation

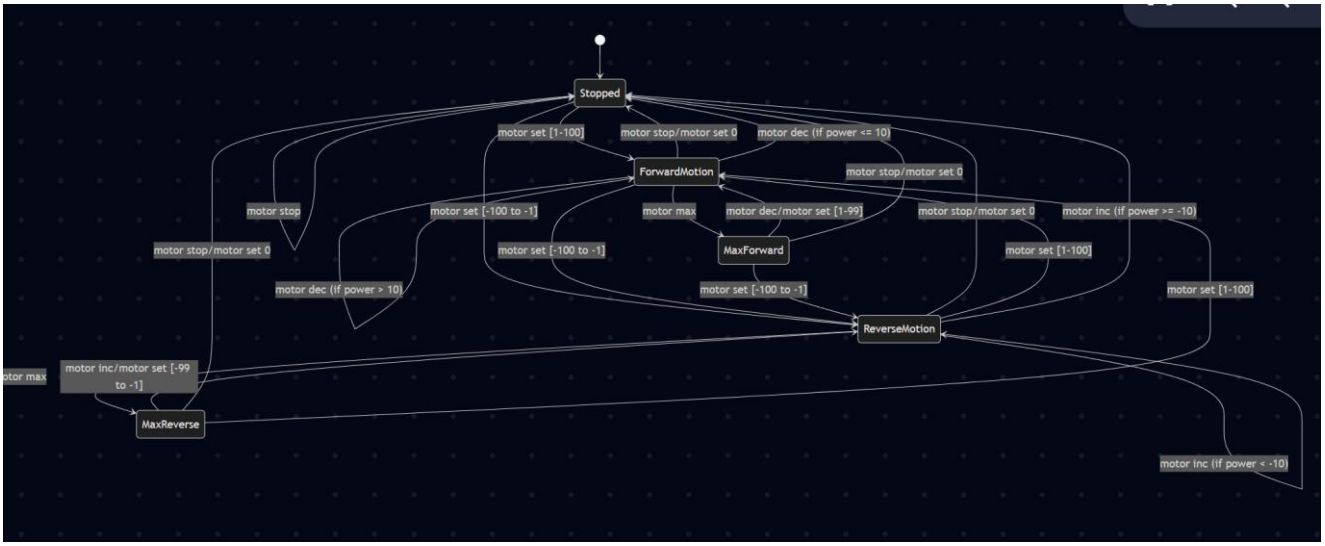


Block Diagram of System Behaviour



This flowchart shows the overall program flow. The system initializes the Serial connection, sets up UART for standard I/O, initializes the motor pins, and prints the command help. It then enters the main loop where it continuously checks for serial data. When data is available, it reads the command line, processes the command, and handles motor-specific commands or prints an error for unknown commands. A 10ms delay prevents CPU overload.

Functional Block Diagrams

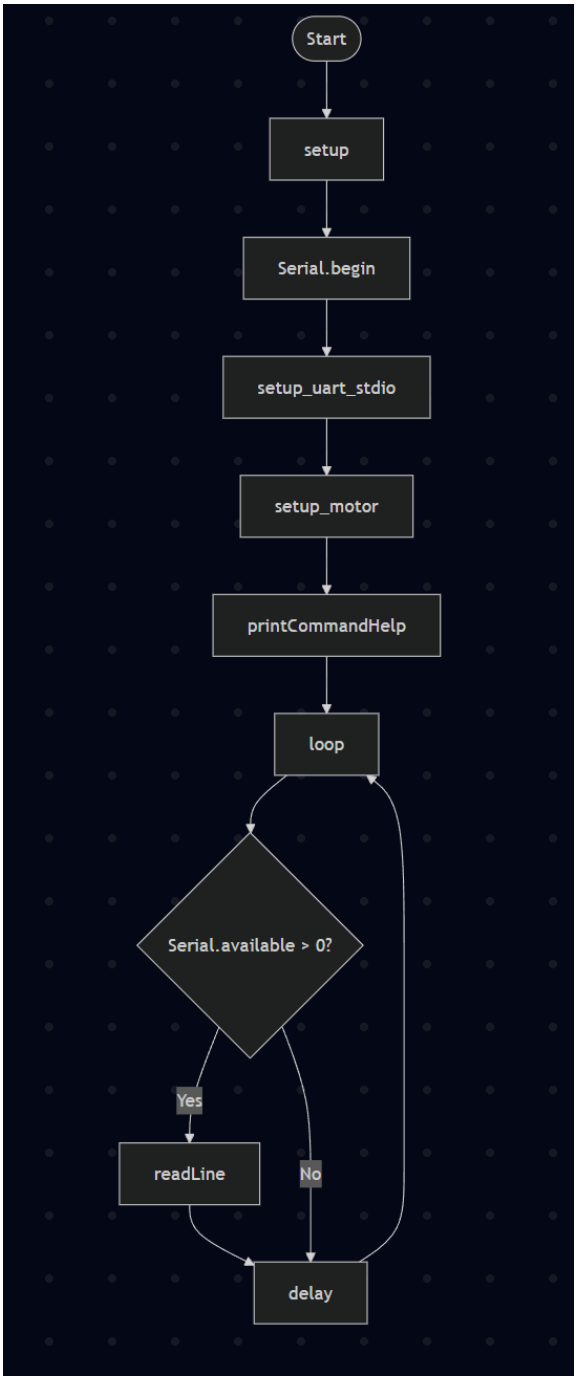


Description:

This finite state machine diagram illustrates the different states of the motor control system and the transitions between them. The four main states are Stopped (no motion), ForwardMotion (positive power), MaxForward (power at 100%), ReverseMotion (negative power), and MaxReverse (power at -100%). Transitions occur in response to commands like "motor set", "motor stop", "motor inc", "motor dec", and "motor max". The diagram shows how these commands affect the motor's state based on the current power level and direction.

Code flowchart:

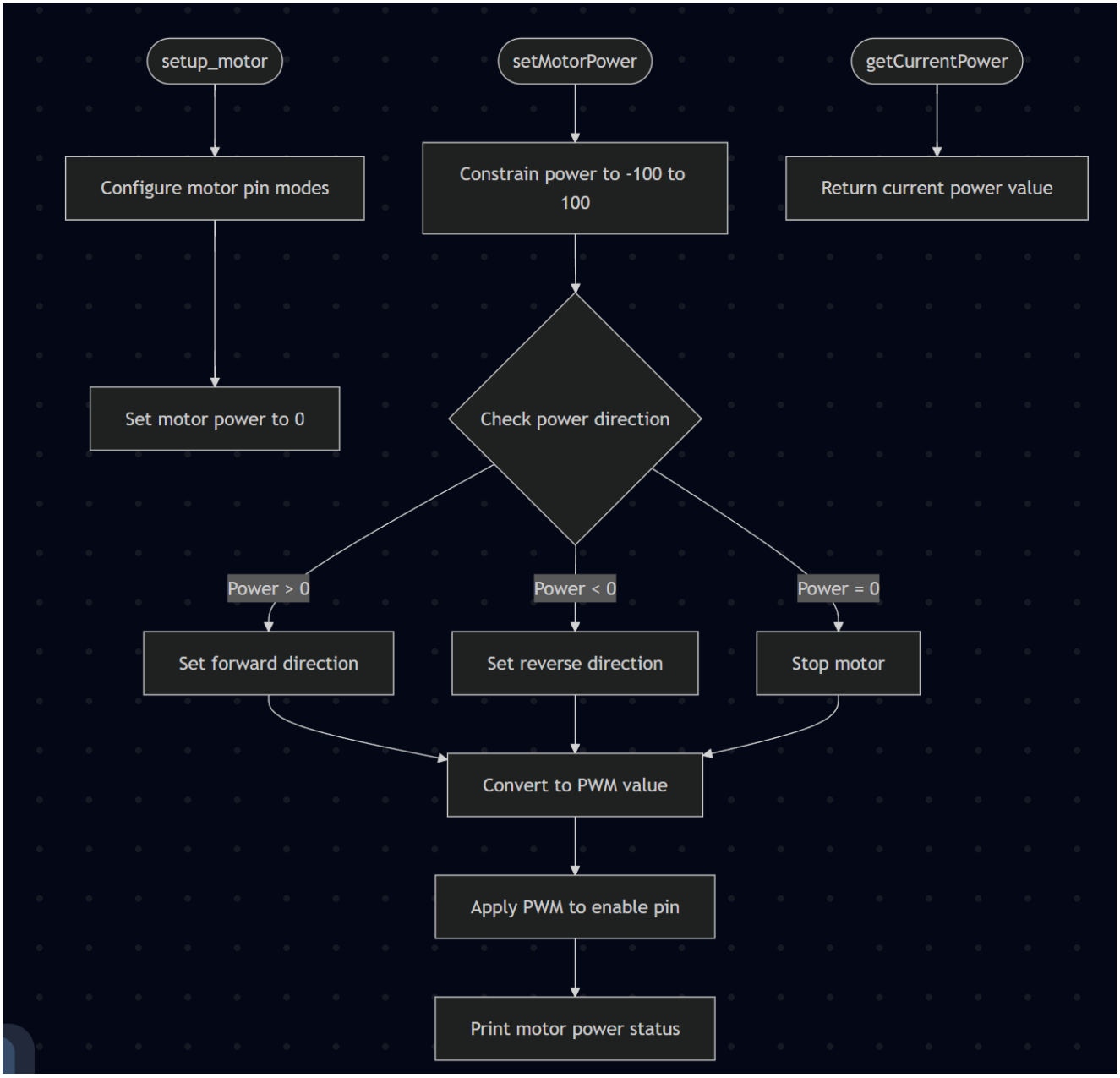
Main.cpp



Description:

This flowchart illustrates the program flow in main.cpp. It starts with the setup() function that initializes the Serial communication at 9600 baud, sets up UART for standard I/O, initializes the motor, and prints command help information. Then it enters the loop() function which continuously checks if serial data is available. If data is available, it calls readLine() to process the input. A small 10ms delay prevents CPU overload. The loop continues indefinitely, constantly checking for new commands.

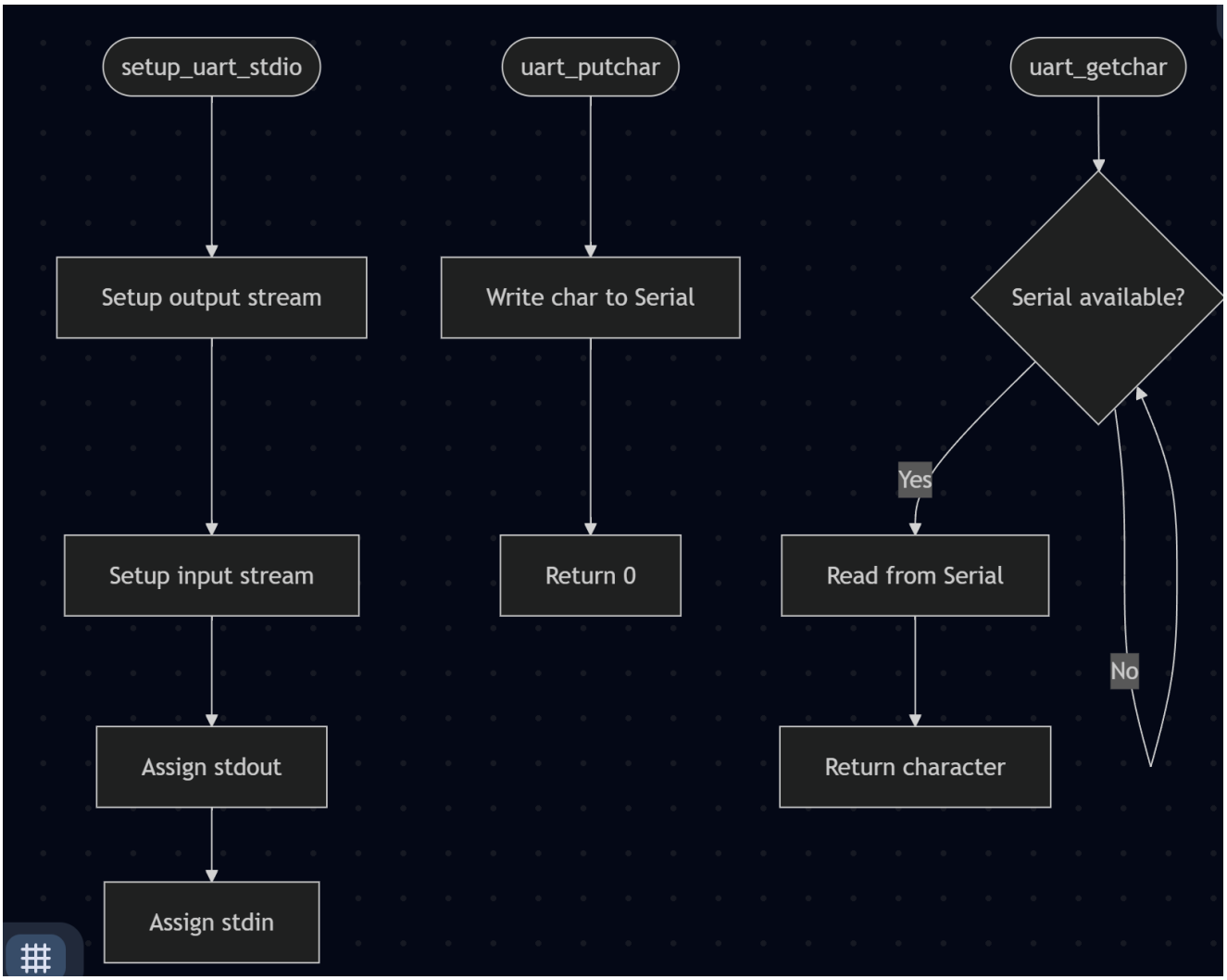
Motor_control.cpp



Description:

This flowchart details the functionality in motor_control.cpp. The setup_motor() function configures the motor pins and initializes the motor to a stopped state. The setMotorPower() function is the core function that controls the motor. It constrains the power to the valid range (-100 to 100), sets the direction pins based on whether the power is positive (forward), negative (reverse), or zero (stop), converts the power percentage to a PWM value, applies it to the enable pin, and prints the status. The getCurrentPower() function simply returns the current power value.

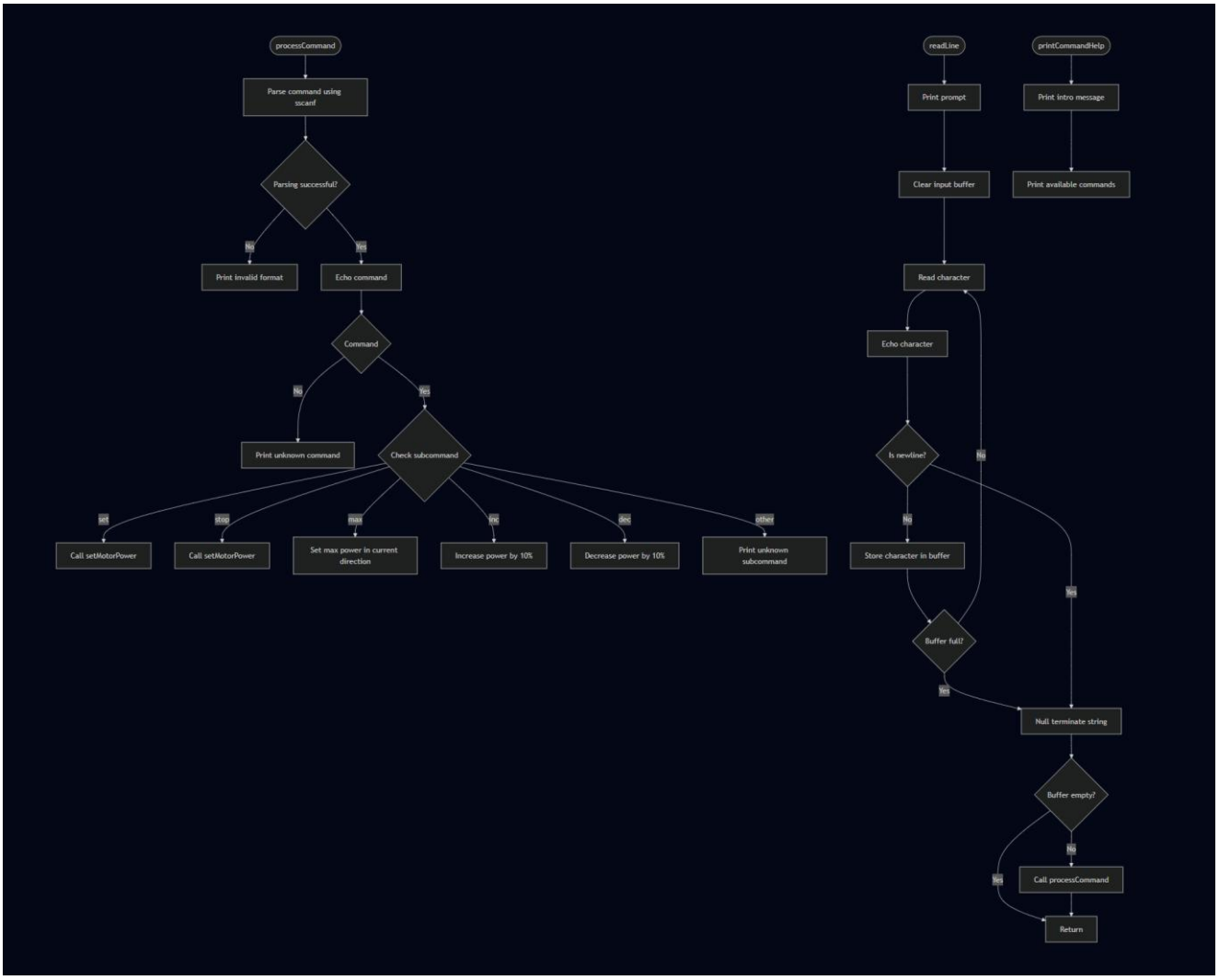
Uart_helpers.cpp



Description:

This flowchart shows the UART helper functions. The setup_uart_stdio() function configures the standard I/O streams by setting up output and input file streams and assigning them to stdout and stdin. The uart_putchar() function is a callback that writes a character to the Serial port and returns 0. The uart_getchar() function waits in a loop until data is available on the Serial port, then reads and returns the character. These functions enable the use of standard C I/O functions like printf() and getchar() with the Arduino Serial interface.

Command_processor.cpp



This detailed flowchart illustrates the command processing system. The processCommand() function parses commands using sscanf and handles motor-specific commands like "set", "stop", "max", "inc", and "dec", calling the appropriate motor control functions. The readLine() function manages the input buffer, reading characters one by one until it encounters a newline or the buffer is full. It echoes characters back to the user and processes the completed command. The printCommandHelp() function displays an introduction message and lists the available commands for the user. This module serves as the interface between user input and the motor control system.

Conclusion

In this laboratory work, we successfully designed and implemented a modular system to control binary actuators (fans) using a relay, driven by a microcontroller (Arduino Uno). The system receives commands from the user through a serial interface, processes these commands, and toggles the state of the relay accordingly.

During the preparation of this report, the author used Grok for generating/strengthening the content. The resulting information was reviewed, validated, and adjusted according to the requirements of the laboratory work.

References

1. IoT-24 : Lab 3 2. - https://www.youtube.com/watch?v=ODJjs82LrtM
2. Arduino Documentation about Seria - https://docs.arduino.cc/language-reference/en/functions/communication/serial/
3. PlatformIO Documentation - https://docs.platformio.org/en/latest/integration/ide/pioide.html

Appendix - Source Code

Main.cpp

```
#include <Arduino.h>
#include "uart_helpers.h"
#include "motor_control.h"
#include "command_processor.h"

void setup() {
  // Initialize serial
  Serial.begin(9600);

  // Initialize stdio
  setup_uart_stdio();

  // Initialize motor
  setup_motor();

  // Print command help
  printCommandHelp();
}

void loop() {
```

```
  if (Serial.available() > 0) {
    readLine();
  }
  delay(10); // Small delay to prevent CPU overload
}
```

Command_processor.cpp

```
#include "command_processor.h"
#include "motor_control.h"
#include <string.h>

char inputBuffer[64];

void processCommand(const char* commandStr) {
  char command[32];
  char subCommand[32] = "";
  int value = 0;

  // Parse the command using sscanf
  if (sscanf(commandStr, "%s %s %d", command, subCommand, &value) >= 1) {
    printf("> %s", command);
    if (strlen(subCommand) > 0) printf(" %s", subCommand);
    if (value != 0 || strcmp(subCommand, "set") == 0) printf(" %d", value);
    printf("\n");

    if (strcmp(command, "motor") == 0) {
      if (strcmp(subCommand, "set") == 0) {
        setMotorPower(value);
      }
      else if (strcmp(subCommand, "stop") == 0) {
        setMotorPower(0);
      }
      else if (strcmp(subCommand, "max") == 0) {
        // Set to max power in current direction
        int currentPower = getCurrentPower();
        if (currentPower >= 0) {
          setMotorPower(100);
```

```
      } else {
        setMotorPower(-100);
      }
    }
    else if (strcmp(subCommand, "inc") == 0) {
      int currentPower = getCurrentPower();
      setMotorPower(currentPower + 10);
    }
    else if (strcmp(subCommand, "dec") == 0) {
      int currentPower = getCurrentPower();
      setMotorPower(currentPower - 10);
    }
    else {
      printf("Unknown sub-command\n");
    }
  }
  else {
    printf("Unknown command\n");
  }
 }
 else {
  printf("Invalid command format\n");
 }
}

// Read a line from Serial using stdio functions
void readLine() {
 int i = 0;
 char c;

 printf("Enter command: ");

 // Clear the buffer
 memset(inputBuffer, 0, sizeof(inputBuffer));

 // Read until newline or buffer is full
 while (i < sizeof(inputBuffer) - 1) {
```

```
    c = getchar();

    // Echo the character back
    putchar(c);

    if (c == '\n' || c == '\r') {
      break;
    }

    inputBuffer[i++] = c;
  }

  inputBuffer[i] = '\0';  // Null terminate the string

  if (strlen(inputBuffer) > 0) {
    processCommand(inputBuffer);
  }
}

void printCommandHelp() {
  printf("Motor Control Ready\n");
  printf("Available commands:\n");
  printf("  motor set [-100..100] - Set power and direction\n");
  printf("  motor stop - Stop the motor\n");
  printf("  motor max - Set to max power in current direction\n");
  printf("  motor inc - Increase power by 10%%\n");
  printf("  motor dec - Decrease power by 10%%\n");
}
```

Motor_control.cpp

```
#include "motor_control.h"
#include <stdio.h>

// Motor pins
int motor1pin1 = 6;
int motor1pin2 = 7;
int enablePin = 5;
```

```
int currentPower = 0; // Range: -100 to 100

void setup_motor() {
  // Configure motor pins
  pinMode(motor1pin1, OUTPUT);
  pinMode(motor1pin2, OUTPUT);
  pinMode(enablePin, OUTPUT);

  // Initial state: motor stopped
  setMotorPower(0);
}

void setMotorPower(int power) {
  // Constrain power to -100 to 100 range
  currentPower = constrain(power, -100, 100);

  // Set direction based on power sign
  if (currentPower > 0) {
    digitalWrite(motor1pin1, HIGH);
    digitalWrite(motor1pin2, LOW);
  } else if (currentPower < 0) {
    digitalWrite(motor1pin1, LOW);
    digitalWrite(motor1pin2, HIGH);
  } else {
    // Stop if power is 0
    digitalWrite(motor1pin1, LOW);
    digitalWrite(motor1pin2, LOW);
  }

  // Convert percentage (0-100) to PWM (0-255)
  int pwmValue = map(abs(currentPower), 0, 100, 0, 255);
  analogWrite(enablePin, pwmValue);

  printf("Motor power set to: %d\n", currentPower);
}

int getCurrentPower() {
```

```
  return currentPower;
}
```

Uart_helpers.cpp

```
#include "uart_helpers.h"

// Setup FILE streams for stdin and stdout
FILE uart_output;
FILE uart_input;

// Function to redirect printf to Serial
int uart_putchar(char c, FILE *stream) {
  Serial.write(c);
  return 0;
}

// Function to redirect stdin to Serial
int uart_getchar(FILE *stream) {
  while (!Serial.available());
  return Serial.read();
}

void setup_uart_stdio() {
  // Initialize stdio
  fdev_setup_stream(&uart_output, uart_putchar, NULL, _FDEV_SETUP_WRITE);
  fdev_setup_stream(&uart_input, NULL, uart_getchar, _FDEV_SETUP_READ);
  stdout = &uart_output;
  stdin = &uart_input;
}
```