Ministerul Educaţiei, Culturii și Cercetării al Republicii Moldova
Universitatea Tehnică a Moldovei
Departamentul Ingineria Software și Automatică

RAPORT

Lucrare de laborator Nr.4.2
Disciplina: IoT
Tema: Actuatori cu interfată analogică. Motor în curent continuu

A efectuat: st.gr.TI-212,
Muntean Mihai

A verificat : asist. univ.
Lupan Cristian

Chișinău 2024

Definirea problemei:

Să se realizeze o aplicatie in baza de MCU care va controla dispozitivele de actionare cu comenzi receptionate de la interfata seriala si raportare catre LCD.

Dispozitivele de actionare vor fi urmatoarele:

- un motor in curent continuii cu comenzi de setare a puterii motorului intre (-100% .. 100%) adica inainte si inapoi, si viteza prin intermediul driverului L298

Driverele de control a periferiilor se vor realiza pe nivele de abstractie

Obiective:

1. Implementarea comunicării seriale pentru recepționarea comenzilor;
2. Controlul motorului prin intermediul driverului L298;
3. Afișarea stării pe LCD.

INTRODUCERE

Actuatorii reprezintă componente esențiale în sisteme automatizate, având rolul de a transforma un semnal de control (electronic sau mecanic) într-o acțiune fizică. În esență, un actuator este un dispozitiv care primește o comandă de la un sistem de control (precum un microcontroler) și o convertește într-o mișcare sau într-o altă formă de energie utilă, cum ar fi activarea unui releu pentru a aprinde un bec sau pentru a mișca un motor electric.

Într-un sistem automatizat, actuatoarele permit interacțiunea cu mediul fizic, permițând funcții precum mișcarea mecanică, controlul poziției, aprinderea sau stingerea unui dispozitiv și multe altele. Controlul precis al actuatorilor este important în aplicații variate, de la automatizarea industrială și robotică până la dispozitive inteligente de uz casnic.

Exemple de actuatori:

- Releele – permit controlul dispozitivelor de putere (ex. becuri, motoare) printr-un semnal de control de joasă tensiune. Un releu poate fi acționat pentru a închide sau deschide un circuit electric.
- Motoarele electrice – utilizate pentru mișcarea componentelor, fiind acționate în funcție de intensitatea și direcția curentului.

Modulul driver de motor L298N este un component util pentru controlul motoarelor DC cu un Arduino, permițând controlul vitezei și al direcției. Poate controla până la două motoare DC sau un motor pas cu pas (stepper). Modulul folosește un circuit H-bridge, care permite controlul bidirecțional al rotației motorului, iar controlul vitezei se realizează prin Modularea Pulsului în Lățime (PWM), care ajustează tensiunea medie prin variația lățimii impulsului.

EFECTUAREA LUCRĂRII

Materiale necesare:

- **L298N motor driver:** Acesta controlează alimentarea și direcția motorului.
- **Microcontroler (Arduino Mega**): Pentru a controla L298N prin semnale logice (pini de control digitali).
- **Motor DC:** Motorul care va fi controlat de driver.
- **Surse de alimentare separate și fire de conectare:** L298N necesită o sursă de alimentare pentru motor și alta pentru Arduino.
- **Display LCD I2C** pentru a afișa starea becului (aprins sau stins). Interfața I2C simplifică conexiunile necesare pentru LCD, reducând numărul de pini necesari pe Arduino Mega;

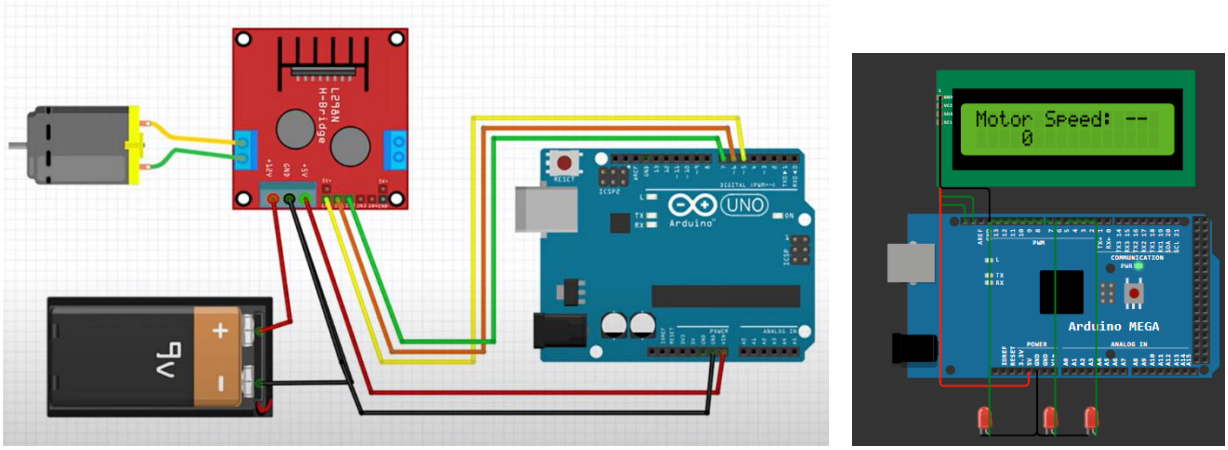

Figura 1 – Ansamblarea circuitului virtual

Modul de lucru

Utilizând această aplicație în bază de MCU, putem gestiona starea unui Motor viteza și direcția de deplasare, utilizând L298 comandat de microcontroler prin intermediul terminalului **Serial.**

```
#include "lcd_utils.h"
#include "serial_utils.h"
#include "L298.h"

int16_t enterSpeed();
void setSpeed(uint8_t speed);

void setup() {
  initializeSerial(11500);
  initializeLcd();
  initializeDcMotor();
}
```

Mai sus are loc instanțierea stărilor primare a componentelor, configurând microcontrolerul pentru a putea controla motorul și LCD-ul.

```
void loop() {
  int16_t speed = 0;
  char buffer[20];

  printf("Enter SPEED: ");
  speed = enterSpeed();          // Citirea vitezei de la utilizator

  dtostrf(speed, 5, 2, buffer);
  printf("%s\n", buffer);

  setSpeed(speed);               // Setarea vitezei motorului în funcție de valoarea introdusă
}
```

// Funcție pentru citirea și validarea vitezei introduse de utilizator prin interfața serială

```
int16_t getSpeed() {
  bool valid = false;
  int16_t speed;

  while (!valid) {
    speed = getNumericValue();
    if (speed < -255 || speed > 255) {
      valid = false;
      printf("Value out of range. Please enter a value between -255 and 255.\n");
    } else {
      valid = true;
    }
  }
```

```
  return speed;
}
```

// Funcție pentru setarea direcției și vitezei motorului DC

```
void setSpeed(int16_t speed) {
  if (speed < 0) {              // Dacă viteza este negativă, motorul va roti spre stânga
    spinLeft(abs(speed));
  }
  else if (speed > 0) {         // Dacă viteza este pozitivă, motorul va roti spre dreapta
    spinRight(abs(speed));
  }
  else {
    stopSpinning();             // Dacă viteza este zero, oprim motorul
  }
}
```

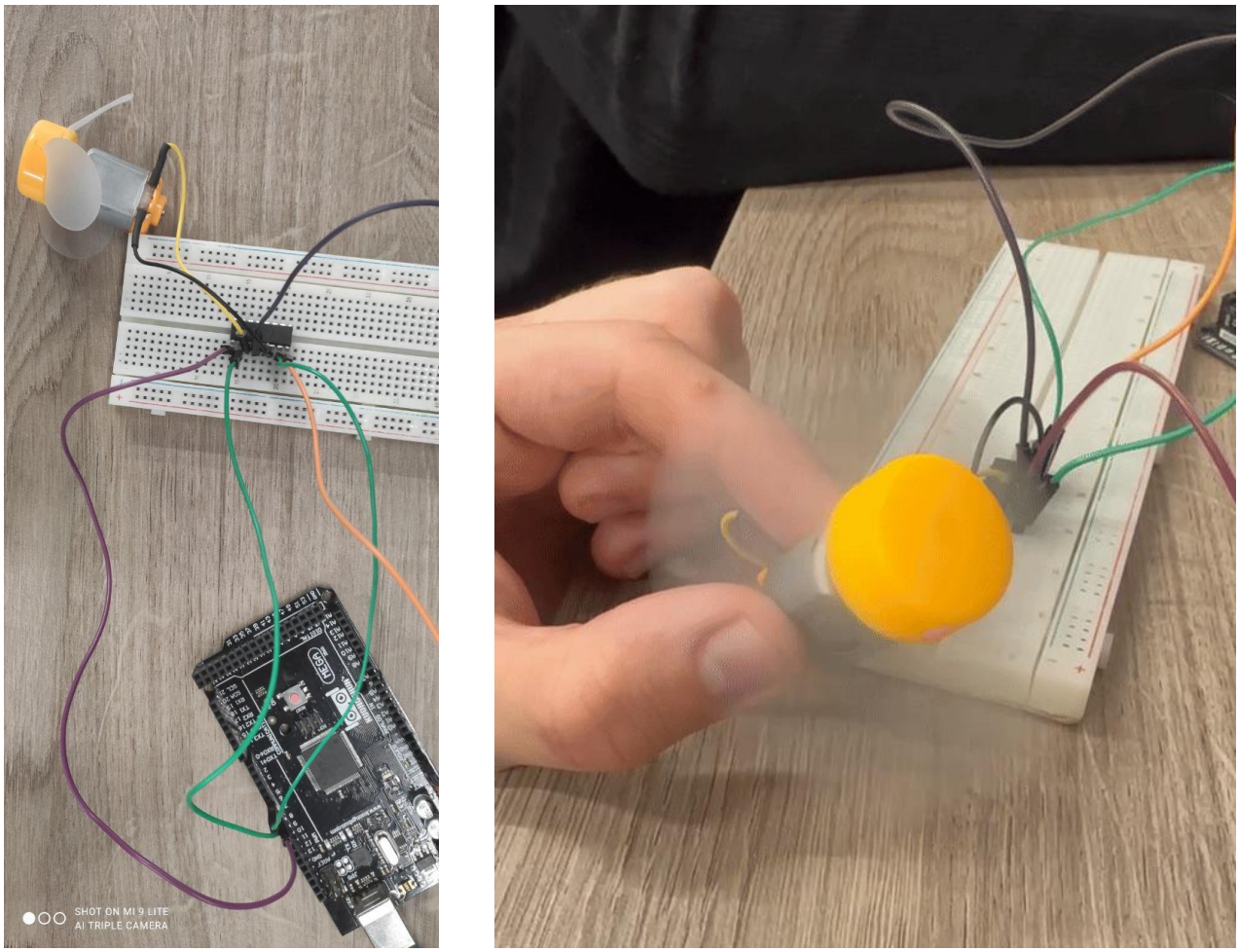

Figura 2 – Aplicația pe circuit fizic

CONCLUZIE

În cadrul acestei lucrări de laborator, am implementat un sistem de control pentru un motor în curent continuu folosind un driver L298N, interfațat cu un microcontroler Arduino și un ecran LCD pentru afișarea stării motorului. Aplicația permite ajustarea vitezei și direcției motorului prin intermediul comenzilor recepționate serial, oferind o soluție simplă, dar eficientă pentru controlul actuatorilor.

Lucrarea a subliniat importanța actuatorilor în sistemele automatizate, evidențiind capacitatea lor de a interacționa fizic cu mediul înconjurător. Modul de lucru include implementarea comenzilor seriale, utilizarea PWM pentru controlul vitezei și un circuit H-bridge pentru controlul bidirecțional al motorului. Rezultatele experimentale au demonstrat succesul implementării unui sistem integrat de comandă a motorului, completat de o interfață vizuală care facilitează monitorizarea stării motorului în timp real.

Astfel, această lucrare a pus în practică principiile fundamentale ale controlului actuatoarelor într-un sistem IoT, utilizând comunicarea serială, interfețele hardware și afișarea grafică pe LCD.

BIBLIOGRAFII

1. Resursa electronică: https://docs.wokwi.com/?utm_source=wokwi – Regim de acces;
2. Resursa electronică: https://else.fcim.utm.md/course/view.php?id=343 – Regim de acces;
3. Resursa electronică: https://forum.arduino.cc/t/serial-print-and-printf/146256/14 - Regim de acces;
4. Interface L298N DC Motor Driver Module with Arduino. Resursa electronica: https://lastminuteengineers.com/l298n-dc-stepper-driver-arduino-tutorial/ - Regim de acces;
5. Proiectul pe GitHub, Resursa electronică: https://github.com/MunMihai/Anul4/tree/8c7af8c43b8c728ff28e1ee6c75a63f997659015/Semestrul_7/Internetul_Lucrurilor%20(IoT)/Laborator/Laborator4 - Regim de acces;

Anexa 1

Fișierul stdinout.h

```
#ifndef _STDINOUT_H
#define _STDINOUT_H

// no need to make an instance of this yourself
class initializeSTDINOUT
{
        static size_t initnum;
public:
        // Constructor
        initializeSTDINOUT();
};

// Call the constructor in each compiled file this header is included in
// static means the names won't collide
static initializeSTDINOUT initializeSTDINOUT_obj;

#endif
```

Fișierul stdinout.cpp

```
#if ARDUINO >= 100
#include "Arduino.h"
#else
#include "WProgram.h"
#endif
#include <stdio.h>
#include "stdinout.h"

// Function that printf and related will use to print
static int serial_putchar(char c, FILE *f)
{
  if (c == '\n') {
    serial_putchar('\r', f);
  }

  return Serial.write(c) == 1 ? 0 : 1;
}
// Function that scanf and related will use to read
static int serial_getchar(FILE *)
{
  // Wait until character is avilable
  while (Serial.available() <= 0) { ; }

  return Serial.read();
}

static FILE serial_stdinout;

static void setup_stdin_stdout()
```

```
{
  // Set up stdout and stdin
  fdev_setup_stream(&serial_stdinout, serial_putchar, serial_getchar, _FDEV_SETUP_RW);
  stdout = &serial_stdinout;
  stdin  = &serial_stdinout;
  stderr = &serial_stdinout;
}

// Initialize the static variable to 0
size_t initializeSTDINOUT::initnum = 0;

// Constructor that calls the function to set up stdin and stdout
initializeSTDINOUT::initializeSTDINOUT()
{
  if (initnum++ == 0) {
    setup_stdin_stdout();
  }
}
```

Anexa 2

main.ino

```
#include "lcd_utils.h"
#include "serial_utils.h"
#include "L298.h"

int16_t getSpeed();
void setSpeed(int16_t speed);

void setup() {
  initializeSerial(11500);
  initializeLcd();
  initializeDcMotor();
}

void loop() {
  int16_t speed = 0;
  char buffer[20];

  printf("Enter SPEED: ");
  speed = getSpeed();

  dtostrf(speed, 5, 2, buffer);
  printf("%s\n", buffer);

  setSpeed(speed);
}

int16_t getSpeed() {
  bool valid = false;
  int16_t speed;

  while (!valid) {
```

```
    speed = getNumericValue();
    if (speed < -255 || speed > 255) {
      valid = false;
      printf("Value out of range. Please enter a value between -255 and 255.\n");
    } else {
      valid = true;
    }
  }
  return speed;
}

void setSpeed(int16_t speed) {
  if (speed < 0) {
    spinLeft(abs(speed));
  }
  else if (speed > 0) {
    spinRight(abs(speed));
  }
  else {
    stopSpinning();
  }
}
```

lcd_utils.h

```
#ifndef LCD_UTILS_H
#define LCD_UTILS_H

#include <LiquidCrystal_I2C.h>

#define I2C_ADDR    0x27
#define COLUMNS 16
#define ROWS   2

void initializeLcd();
void resetLcd(uint8_t speed, const char* direction);

#endif // LCD_UTILS_H
```

lcd_utils.cpp

```
#include "lcd_utils.h"
#include <stdio.h>
LiquidCrystal_I2C lcd(I2C_ADDR, COLUMNS, ROWS);

void initializeLcd(){
  lcd.begin(16, 2);
  lcd.backlight();
  resetLcd(0, "--");
}

void resetLcd(uint8_t speed, const char* direction) {
```

```
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("Motor Speed:");
  lcd.setCursor(13, 0);
  lcd.print(direction);
  lcd.setCursor(4, 1);
  lcd.print(speed);
}
```

serial_utils.h

```
#ifndef SERIAL_UTILS_H
#define SERIAL_UTILS_H

#include <Arduino.h>

void initializeSerial(int64_t baud);
int16_t getNumericValue();

#endif // SERIAL_UTILS_H
```

serial_utils.cpp

```
#include "serial_utils.h"

void initializeSerial(int64_t baud){
  Serial.begin(baud);
  printf("Serial is rady!\n");
}

int16_t getNumericValue() {
  char input[10];  // Șir pentru a stoca inputul utilizatorului
  int16_t speed = 0;
  bool valid = false;

  while (!valid) {
    // Citim inputul ca un șir de caractere
    scanf("%s", input);

    // Verificăm dacă inputul este numeric și poate începe cu '-' sau '+'
    valid = true;  // Presupunem că este valid până la dovada contrară
    int i = 0;
    if (input[0] == '-' || input[0] == '+') {
      i = 1;  // Dacă începe cu '-' sau '+', începem verificarea de la următorul caracter
    }

    // Verificăm restul caracterelor să fie doar cifre
    for (; input[i] != '\0'; i++) {
      if (input[i] < '0' || input[i] > '9') {
        valid = false;
        break;
      }
```

```
    }

    // Dacă inputul este valid, convertim la număr și verificăm intervalul
    if (valid) {
      speed = atoi(input);  // Convertim șirul la număr întreg

    } else {
      printf("Invalid input. Please enter a numeric value.\n");
    }
  }
  return speed;  // Returnăm valoarea introdusă
}
```

L298.h

```
#ifndef L298_H
#define L298_H

#include <Arduino.h>

//MotorPins
#define IN1 5
#define IN2 6
#define ENA 7 // for speed control

void initializeDcMotor();
void stopSpinning();
void spinRight(uint8_t);
void spinLeft(uint8_t);

#endif // L298_H
```

L298.cpp

```
#include "L298.h"
#include "lcd_utils.h"

void initializeDcMotor(){
  pinMode(IN1, OUTPUT);
  pinMode(IN2, OUTPUT);
  pinMode(ENA, OUTPUT);

  digitalWrite(IN1, LOW);
  digitalWrite(IN2, LOW);
}

void stopSpinning(){
  digitalWrite(IN1, LOW);
  digitalWrite(IN2, LOW);
  analogWrite(ENA, 0);
  resetLcd(0, "--");
```

```
}

void spinRight(uint8_t speed){
  digitalWrite(IN1, HIGH);
  digitalWrite(IN2, LOW);
  analogWrite(ENA, speed);
  resetLcd(speed, "->");
}

void spinLeft(uint8_t speed){
  digitalWrite(IN1, LOW);
  digitalWrite(IN2, HIGH);
  analogWrite(ENA, speed);
  resetLcd(speed, "<-");
}
```