**Laboratorul 1**

**Definirea problemei:**

1. Să se proiecteze o aplicație în baza de MCU care ar schimba starea unui LED la detectarea apăsării unui buton

2. Să se proiecteze o aplicație în baza de MCU care ar recepționa comenzi de la terminal prin interfața serială pentru a seta starea unui LED

● on pentru aprindere și led off pentru stingere, sistemul trebuie să răspundă cu mesaje text despre confirmrea comenzii

● pentru schimbul d etext prin termina a se utiliza librăria STDIO

3. Să se proiecteze o aplicație în baza de MCU pentru detectarea unui cod de la tastatură

4x4, să verifice codul și să afișeze mesaj la un LCD.

● pentru cod valid să se aprindă un led de culoarea verde, pentru cod invalid, un led de culoarea roșie

● a se utiliza STDIO pentru scanarea tastaturii și afișarea la LCD

**Considerente teoretice:**

Arduino este o platformă electronică open-source bazată pe hardware și software ușor de utilizat. Plăcile Arduino sunt capabile să citească intrări - lumină pe un senzor, un deget pe un buton sau un mesaj Twitter - și să le transforme într-o ieșire - activând un motor, pornind un LED, publicând ceva online.

Arduino este folosit pentru a crea dispozitive electronice cu capacitatea de a primi semnale de la diverși senzori digitali și analogici la care pot fi conectați el și controlează diverse dispozitive executive. Dispozitive bazate pe proiecte pe Arduino, poate lucra singur sau poate interacționa cu software-ul pe computer.

Este evident caci conectarea oricăror dispozitive la placă se realizează prin conectarea la contacte, situat pe placa de control: unul dintre pinii sau pinii digitali sau analogici nutriție.

Un LED simplu poate fi conectat folosind doi pini: masă (GND) și semnal (sau contact de putere).

**Ex.1**

#define BUTTON\_PIN 8

#define LED\_PIN 0

int buttonState = 0;

void setup() {

pinMode(LED\_PIN, OUTPUT);

pinMode(BUTTON\_PIN, INPUT\_PULLUP);

}

void loop() {

buttonState = digitalRead(BUTTON\_PIN);

if(buttonState == LOW)

digitalWrite(LED\_PIN, HIGH);

else

digitalWrite(LED\_PIN, LOW);

}

**Ex.2**

void setup() {

pinMode(13, OUTPUT);

Serial.begin(9600);

while (!Serial);

Serial.println("1 or 0");

}

void loop() {

if (Serial.available()) {

int state = Serial.parseInt();

if (state == 1) {

digitalWrite(13, HIGH);

}

if (state == 0) {

digitalWrite(13, LOW);

}}}

**Laboratorul 2**

**Sarcina:** construiți o aplicație MCU care va rula cel puțin 3 sarcini în două versiuni -secvenţial şi cu FreeRTOS. Sarcinile vor efectua următoarele sarcini:

TaskA - schimbarea stării LED-ului la apăsarea butonului;

TaskB - porniți al doilea LED când LED-ul din prima sarcină

TaskC - crește / micșorează valoarea unei variabile apăsând două butoane care vor fi responsabile de numărul de repetări/timpul în care LED-ul de la prima sarcină va fi în starea activată.

**Mersul lucrării**:

Sarcinile secvențiale sunt implementate folosind funcții void. TaskA, TaskB și TaskC sunt declarate la începutul sau sfârșitul programului și sunt apelate unul după altul în funcție buclă goală ().

Sarcini paralele, de ex. cele care rulează concomitent implementat folosind sistemul de operare în timp real FreeRTOS. Ca majoritatea sistemelor de operare din FreeRTOS au un planificator care, prin alocare o anumită prioritate pentru fiecare fir care rulează, determină ce fir în care moment în timp ar trebui să fie transferat timpul

procesorului (adică ce fir și când ar trebui continuați să executați algoritmul). Pe un sistem FreeRTOS, firul executabil notat cu termenul Sarcină.

Pentru toate sarcinile, este asamblată următoarea schemă: două LED-uri sunt conectate la MCU (roșu și verde), trei butoane și cinci rezistențe. Când faceți clic pe primul buton LED-urile își schimbă starea - roșu se aprinde, verde se stinge și invers.

Al doilea buton este responsabil pentru reducerea frecvenței de clipire a LED-ului roșu, iar al treilea - pentru crește frecvența intermitentului.

Listarea programului cu executarea secvențială a sarcinilor.

#include <stdio.h>

#include <stdlib.h>

const int strButtonPin = 2; // GPIO 2 for the start button

// GPIO 4 pentru butonul care mărește frecvența clipirii

const int incButtonPin = 4;

// GPIO 3 pentru butonul care scade frecvența clipirii

const int decButtonPin = 3;

const int rledPin = 8; // GPIO 8 for the red LED

const int gledPin = 9; // GPIO 9 for the green LED

int lastButtonState = LOW; // Ultima stare a butonului de pornire

int strButtonState = LOW; // Start button current state

int incButtonState = LOW; // Increment button current state

int incLastButtonState = LOW; // Increment button last state

int decButtonState = LOW; // Decrement button current state

int decLastButtonState = LOW; // Decrement button last state

int rledState = LOW; // Red LED state

int gledState = LOW; // Green LED state

int count = 0;

unsigned long previousMillis = 0; //Stochează ultima dată când LED-ul a fost

actualizat

long interval = 500; // Interval la care să clipească (milisecunde)

unsigned long currentMillis = 0;

void setup() {

Serial.begin(9600);

pinMode(rledPin, OUTPUT);

pinMode(gledPin, OUTPUT);

pinMode(strButtonPin, INPUT);

pinMode(incButtonPin, INPUT);

pinMode(decButtonPin, INPUT);

}

void loop() {

// // Returnează numărul de milisecunde trecute de la începutul plăcii Arduino

// running the current program

currentMillis = millis();

strButtonState = digitalRead(strButtonPin);

incButtonState = digitalRead(incButtonPin);

decButtonState = digitalRead(decButtonPin);

if (strButtonState != lastButtonState) { // If the start button was pressed

lastButtonState = strButtonState;

if (strButtonState == LOW) {

count++; increment the counter

Serial.println("Start button is pressed.");

}

}

TaskA(); TaskB(); TaskC();

}

void TaskA() {

// LED-ul roșu este aprins când LED-ul verde este stins și invers

if (count % 2 == 1) { // Dacă contorul este un număr impar

Serial.print("Blink Interval: ");

Serial.println(interval);

if (currentMillis - previousMillis >= interval) {

// save the last time you blinked the LED

previousMillis = currentMillis;

// dacă LED-ul este stins porniți-l și invers

if (rledState == LOW) {

rledState = HIGH;

} else {

rledState = LOW;

}

digitalWrite(rledPin, rledState);

Serial.print("Red button state is ");

Serial.println(rledState);

}

}

else if (count % 2 == 0) // If the counter is an even number (0, 2, 4...)

{

digitalWrite(rledPin, LOW); // turn the red LED off

Serial.println("Red button state is LOW");

count = 0; // reset the counter

} }

void TaskB() {

// The green LED is ON when the red LED is OFF and vice-versa

if (count % 2 == 1) //Dacă contorul este un număr impar

gledState = LOW; // turn the green LED off

if (count % 2 == 0) // In caz contrar,

gledState = HIGH; // // turn the green LED on

digitalWrite(gledPin, gledState);

Serial.print("Green button state is ");

Serial.println(gledState);

}

void TaskC()

{

if (incButtonState != incLastButtonState) {

incLastButtonState = incButtonState;

if (incButtonState == LOW) {// Dacă butonul de creștere a fost apăsat

Serial.println("Increment button is pressed.");

if (interval > 250 && interval < 1000)

{

interval += 50; // increment the value of interval by 50

}

}

}

if (decButtonState != decLastButtonState) {

decLastButtonState = decButtonState;

if (decButtonState == LOW) {// Dacă butonul de decrementare a fost apăsat

Serial.println("Decrement button is pressed.");

if (interval > 250 && interval < 1000)

{

interval -= 50; // decrement the value of interval by 50

}

}

} }

Codul cu utilizarea FreeRTOS

#include <Arduino\_FreeRTOS.h>

// GPIO 2 pentru butonul de pornire care aprinde și stinge LED-ul

const int strButtonPin = 2;

// GPIO 4 pentru butonul care mărește frecvența clipirii

const int incButtonPin = 4;

// GPIO 3 for the button that decreases the blink frequency

const int decButtonPin = 3;

const int rledPin = 8; // GPIO 8 for the red LED

const int gledPin = 9; // GPIO 9 for the green LED

int lastButtonState = LOW; // Start button last state

int strButtonState = LOW; // Start button current stateint incButtonState = scazut; //

Increment button current state

int incLastButtonState = LOW; // Increment button last state

int decButtonState = LOW; // Decrement button current state

int decLastButtonState = LOW; // Decrement button last state

int rledState = LOW; // Red LED state

int gledState = LOW; // Green LED state

int count = 0;

unsigned long previousMillis = 0; // Stochează ultima dată când LED-ul a fost

actualizat

long interval = 500; // Interval la care să clipească (milisecunde)

unsigned long currentMillis = 0;

void Task\_A( void \*pvParameters ); // Task for red LED blinking

void Task\_B( void \*pvParameters ); // Task for green LED blinking

void Task\_C( void \*pvParameters ); // Task for changing blinking frequency

void setup() {

xTaskCreate(

Task\_A // // Indicator către funcția de introducere a sarcinii

, "Task\_A" // Un nume descriptiv pentru sarcină

, 128 // Dimensiunea stivei

, NULL // O valoare care este transmisă ca parametru sarcinii create

, 1 // Priority

, NULL ); // Folosit pentru a distribui mânerul sarcinii create

xTaskCreate(

Task\_B

, "Task\_B"

, 128

, NULL

, 1

, NULL );

xTaskCreate(

Task\_C

, "Task\_C"

, 128

, NULL

, 1

, NULL );

}

void loop() {

//Lucrurile se fac în Sarcini.}

void Task\_A(void \*pvParameters) {

(void) pvParameters;

Serial.begin(9600);

pinMode(rledPin, OUTPUT);

pinMode(strButtonPin, INPUT);

for (;;) //O Sarcină nu va reveni sau ieși niciodată.

{

strButtonState = digitalRead(strButtonPin);

if (strButtonState != lastButtonState) {

lastButtonState = strButtonState;

if (strButtonState == LOW) { // Dacă butonul de pornire a fost apăsat

count++; // increment the counter

Serial.println("Start button is pressed.");

} }

// Returnează numărul de milisecunde trecute de la placa Arduino

// a început să ruleze programul curentcurrentMillis = millis();

// LED-ul roșu este aprins când LED-ul verde este stins și invers

if (count % 2 == 1) { // If the counter is an odd number (1, 3, 5...)

Serial.print("Blink Interval: "); Serial.println(interval);

if (currentMillis - previousMillis >= interval) {

// save the last time you blinked the red LED

previousMillis = currentMillis;

// dacă LED-ul roșu este stins porniți-l și invers

if (rledState == LOW) {

rledState = HIGH;

} else { rledState = LOW;

}

digitalWrite(rledPin, rledState);

Serial.print("Red button state is ");

Serial.println(rledState);

} }

else if (count % 2 == 0) // If the counter is an even number (0, 2, 4...)

{

digitalWrite(rledPin, LOW); // turn the red LED off

Serial.println("Red button state is LOW");

count = 0; // reset the counter

}

vTaskDelay(1); //o întârziere (15 ms) între citiri pentru stabilitate

} }

void Task\_B(void \*pvParameters)

{

(void) pvParameters;

Serial.begin(9600);

pinMode(gledPin, OUTPUT);

pinMode(strButtonPin, INPUT);

for (;;) // O Sarcină nu va reveni sau ieși niciodată.

{

strButtonState = digitalRead(strButtonPin);

if (strButtonState != lastButtonState) {

lastButtonState = strButtonState;

if (strButtonState == LOW) { // If the start button was pressed

count++; // increment the counter

Serial.println("Start button is pressed.");

} }

// LED-ul verde este aprins când LED-ul roșu este stins și invers

if (count % 2 == 1) // If the counter is an odd number

gledState = LOW; // turn the green LED off

if (count % 2 == 0) //In caz contrar

gledState = HIGH; // // turn the green LED on

digitalWrite(gledPin, gledState);

Serial.print("Green button state is ");

Serial.println(gledState);

vTaskDelay(1); // one tick delay (15ms) in between reads for stability

} }

void Task\_C(void \*pvParameters)

{

(void) pvParameters;

Serial.begin(9600);

pinMode(incButtonPin, INPUT);

pinMode(decButtonPin, INPUT);

for (;;) // O sarcină nu va reveni sau ieși niciodată.

{

incButtonState = digitalRead(incButtonPin);

decButtonState = digitalRead(decButtonPin);

if (incButtonState != incLastButtonState) {

incLastButtonState = incButtonState; if (incButtonState == LOW) { // If the

increment button was pressed

Serial.println("Increment button is pressed.");

if (interval > 250 && interval < 1000) {

interval += 50; //măriți valoarea intervalului cu 50

} } }

if (decButtonState != decLastButtonState) { decLastButtonState = decButtonState;

if (decButtonState == LOW) { // If the decrement button was pressed

Serial.println("Decrement button is pressed.");

if (interval > 250 && interval < 1000) {

interval -= 50; // decrement the value of interval by 50

} } }

vTaskDelay(1); // one tick delay (15ms) in between reads for stability

} }

**Laboratorul 3**

Sa se realizeze o aplicație in baza de MCU care va prelua un semnal de la o sursa de

semnal, va condiționa semnalul, si va afișa parametrul fizic la un terminal (LCD si/sau Serial).

Fiecare student va selecta un senzor fie analogic fie digital (nu binar) din PDF atașat sau:

http://www.37sensors.com/

Sa se achiziționeze semnalul de la senzor;

Să se condiționeze semnalul implicând filtre digitale și alte metode;

Să se afișeze datele pe afișor LCD și / sau Serial.

Indicații pentru implementare

 pentru Validare se recomanda utilizarea unui simulator, de ex Proteus.

 funcționalitățile pentru fiecare echipament periferic ( led, buton, lcd, sensor) sa fie

realizate în fișiere separate, in scop de reutilizare in lucrările următoare

 utilizarea regulilor de codare magic number si CamelCase

Introducere

Un Arduino este o placă de dezvoltare microcontroler open-source. Puteți folosi Arduino

pentru a citi senzori și a controla lucruri precum motoare și lumini. Acest lucru vă permite să încărcați

programe în această placă, cu care apoi se poate interacționa cu lucrurile din lumea reală. Cu aceasta,

puteți face dispozitive care răspund și reacționează la lumea în ansamblu.

De exemplu, puteți citi un senzor de umiditate conectat la o plantă în ghiveci și puteți activa un sistem

automat de udare dacă se usucă prea mult. Sau, puteți crea un server de chat stand-alone, care este

conectat la routerul de internet. Sau poți avea un ciripit de fiecare dată când pisica trece printr-o ușă

specială pentru animale. Sau puteți să porniți un ibric cu cafea atunci când alarma dvs. se stinge

dimineața.

La fel un microcontroller reprezintă unitate centrală (CPU) căreia i s-a adăugat, pe

acelaşi chip, memorie şi dispozitive periferice. El poate constitui astfel un sistem de calcul de

sine stătător, realizat la un raport preţ/performanţă optim, orientat pe controlul interacţiunii cu

mediul exterior.

Teorie:

Totalitatea de componente realizate prin ME, EE, si SWE care Transformă mărime fizică

din mediu într-un semnal intern al sistemului • Senzor – simte schimbare din mediu și transforma

în mărime măsurabila • Traductor – transformă mărime măsurabila în semnal electric

Codul sursă:

#include <stdio.h>

#include <pins\_arduino.h>

#include <LiquidCrystal.h>

#define RESOLUTION 1024

#define MAX\_VOLTAGE 5

#define FILTER\_LEN 5

#define NB\_FILTERS 3

#define A0 0

#define A1 1

#define A2 2

const int trigPin = 9;

const int echoPin = 10;

static FILE uartout = {0};

// temperature filters

int adc[FILTER\_LEN];

int adcFiltered[NB\_FILTERS];

int adcFilteredMedium;

// temperature sensor variables

double temperature;

double resistance;

double voltage;

// ultrasonic variables

long duration[FILTER\_LEN];

long durationFiltered;

long durationSum;

int distance;

String distStr;

LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

int my\_putChar(char ch, FILE \*stream) {

lcd.print(ch);

return 0;

}

double adcToVoltage(int adc) {

return (double) adc / RESOLUTION \* MAX\_VOLTAGE;

}

double voltageToResistance(double voltage) {

return voltage / 1000 \* MAX\_VOLTAGE;

}

int compare(const void \*a, const void \*b) {

int int\_a = \*((int \*) a);

int int\_b = \*((int \*) b);

if (int\_a == int\_b) return 0;

else if (int\_a < int\_b) return -1;

else return 1;

}

int filterTempSaltPepper(int pin) {

for (int i = 0; i < FILTER\_LEN; ++i) {

adc[i] = analogRead(pin);

}

qsort(adc, FILTER\_LEN, sizeof(int), compare);

return adc[FILTER\_LEN / 2];

}

int filterTempMedPond(int pond1, int pond2, int pond3) {

return ((adcFiltered[A0] \* pond1) + (adcFiltered[A1] \* pond2) + (adcFiltered[A2] \* pond3)) /

(pond1 + pond2 + pond3);

}

long filterSonicSaltPepper() {

for (int i = 0; i < FILTER\_LEN; ++i) {

digitalWrite(trigPin, LOW);

delayMicroseconds(2);

digitalWrite(trigPin, HIGH);

delayMicroseconds(10);

digitalWrite(trigPin, LOW);

duration[i] = pulseIn(echoPin, HIGH);

}

qsort(duration, FILTER\_LEN, sizeof(int), compare);

return duration[FILTER\_LEN / 2];

}

long filterSonicMedPond(int count) {

durationSum = 0;

for (int k = 0; k < count; k++) {

durationSum += filterSonicSaltPepper();

}

return durationSum / count;

}

void printDistance() {

lcd.setCursor(0, 1);

printf("Dist: %s", distStr.c\_str());

Serial.print("Distance: ");

Serial.println(distance);

}

void setup() {

fdev\_setup\_stream (&uartout, my\_putChar, NULL, \_FDEV\_SETUP\_RW);

stdout = &uartout;

pinMode(trigPin, OUTPUT);

pinMode(echoPin, INPUT);

lcd.begin(16, 2);

printf("Hello world!");

}

void loop() {

delay(1000);

lcd.clear();

printDistance();

// temperature

lcd.setCursor(0, 0);

adcFiltered[A0] = filterTempSaltPepper(A0);

adcFiltered[A1] = filterTempSaltPepper(A1);

adcFiltered[A2] = filterTempSaltPepper(A2);

adcFilteredMedium = filterTempMedPond(20, 30, 50);

voltage = adcToVoltage(adcFilteredMedium);

temperature = (voltage - 0.5) \* 100;

String tempStr = String(temperature) + String((char) 178) + "C";

printf("%s", tempStr.c\_str());

lcd.setCursor(8, 0);

resistance = voltageToResistance(voltage);

String resStr = String(resistance \* 1000) + "mOhm";

printf("%s", resStr.c\_str());

// ultra sonic

durationFiltered = filterSonicMedPond(5);

// Calculating the distance

distance = durationFiltered \* 0.034 / 2;

// Prints the distance on the lcd

distStr = String(distance);

**Laboratorul 4**

Sa 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 bec electric prin intermediul releului cu comenzi de ON si OFF ;

 un motor in curent continuii cu comenzi de setare a puterii motorului intre (-100% .. 100%)

adica inainte si inapoi, si viteza rin intermediul driverului L298 ;

Teorie :

Un microcontroler (MC, MCU) este alcătuit dintr-o unitate centrală (CPU) căreia i s-a adăugat, pe acelaşi chip, memorie şi dispozitive periferice. El poate constitui astfel un sistem de calcul de sine stătător, realizat la un raport preţ/performanţă optim, orientat pe controlul interacţiunii cu mediul exterior.

L298 este un circuit monolitic integrat într-un pachet Multiwatt cu 15 conductoare și PowerSO20. Este un driver dublu full-bridge de înaltă tensiune, curent ridicat conceput să accepte nivelurile logice standard TTL și să conducă sarcini inductive, cum ar fi relee, solenoizi, DC și motoare pas cu pas. Sunt furnizate două intrări de activare pentru a activa sau dezactiva dispozitivul independent de semnalele de intrare.

Emițătorii tranzistoarelor inferioare ale fiecărei punți sunt conectați împreună și terminalul extern corespunzător poate fi utilizat pentru conectarea unui rezistor de detectare extern. Este prevăzută o intrare suplimentară de alimentare, astfel încât logica să funcționeze la o tensiune mai mică.

 un bec electric prin intermediul releului cu comenzi de ON si OFF

 #include <LiquidCrystal\_I2C.h>

#define RELAY\_PIN 11

static FILE uartout = {0} ;

LiquidCrystal\_I2C lcd(0x27, 20, 4);

static int my\_putChar( char ch, FILE \* stream) {

lcd.print( ch );

return 0;

}

char my\_GetChar(FILE \* f)

{

while (! Serial.available());

return Serial.read();

}

void setup() {

Serial.begin(9600);

fdev\_setup\_stream (&uartout, my\_putChar, my\_GetChar,

\_FDEV\_SETUP\_RW);

stdin = stdout = &uartout ;

lcd.init();

lcd.backlight();

pinMode(RELAY\_PIN, OUTPUT);

Serial.print("Start\n");

printf("Start");

}

void loop() {

char command[20] = {};

scanf("%s", command);

if (strcmp(command, "on") == 0) { lcd.clear();

digitalWrite(RELAY\_PIN, HIGH);

Serial.print("BULB is turned ON\n");

printf("BULB is turned ON");

}

else if (strcmp(command, "off") == 0)

{

lcd.clear();

digitalWrite(RELAY\_PIN, LOW);

Serial.print("BULB is turned OFF\n"); printf("BULB is turned OFF");

}

delay(2000);

}

 un motor in curent continuii cu comenzi de setare a puterii motorului intre (-100% .. 100%)

adica inainte si inapoi, si viteza rin intermediul driverului L298 ;

 #include <LiquidCrystal\_I2C.h>

#define IN\_1 2

#define IN\_2 3

#define PWM 10

#define MIN\_VAL -100

#define MAX\_VAL 100

int percentSpeedValue = 0;

int newValue = 0;

int direction = 0;

static FILE uartout = {0};

LiquidCrystal\_I2C lcd(0x27, 20, 4);

static int my\_putChar(char ch, FILE \*stream) {

lcd.print(ch);

return 0;

}

char my\_GetChar(FILE \*f) {

while (!Serial.available());

return Serial.read();

}

int checkLimit(int speed) {

if (speed < MIN\_VAL || speed > MAX\_VAL) {

return 0;

}

return speed;

}

// Layer I

void setPwm(int pwmValue) {

analogWrite(PWM, pwmValue);

}

void pinControl(bool a, bool b) {

digitalWrite(IN\_1, a);

digitalWrite(IN\_2, b);

}

// Layer II

void setSpeed(int percentSpeed) { int pwmSpeed = 0;

pwmSpeed = (int) percentSpeed \* 2.55;

setPwm(pwmSpeed);

}

int setDirection(int direction) {

if (direction < 0) { pinControl(1, 0);

return 1;

}

if (direction > 0) { pinControl(0, 1);

return 2;

}

if (direction == 0) { pinControl(0, 0);

return 0;

}

return -1;

}

// Layer III

void motorControl(int direction, int speed) {

setSpeed(speed);

setDirection(direction);

}

// Layer IV

void displayToLcd(int direction, int speed\_value) {

lcd.clear();

switch (direction) {

case 1:

printf("Inainte: %d", speed\_value);

break;

case -1:

printf("Inapoi: %d", speed\_value);

break;

default:

printf("Stop");

break;

} }

void setup() {

Serial.begin(9600);

fdev\_setup\_stream(&uartout, my\_putChar, my\_GetChar, \_FDEV\_SETUP\_RW);

stdin = stdout = &uartout;

lcd.init();

lcd.backlight();

pinMode(IN\_1, OUTPUT);

pinMode(IN\_2, OUTPUT);

pinMode(PWM, OUTPUT);

Serial.print("Start\n");

printf("Start");

percentSpeedValue = Serial.parseInt();

}

void loop() {

newValue = Serial.parseInt();

if (newValue != 0) {

percentSpeedValue = checkLimit(newValue);

if (percentSpeedValue < 0) {

direction = -1;

percentSpeedValue \*= -1;

} else if (percentSpeedValue > 0) {

direction = 1;

} else {

direction = 0; } }

motorControl(direction, percentSpeedValue);

displayToLcd(direction, percentSpeedValue);

Serial.print("Speed: ");

Serial.println(percentSpeedValue);

}

**Laboratorul 5**

1. Sa se realizeze o aplicatie in baza de MCU care va implementa sisteme de control pentru

a) control temperatura sau umeditate cu aplicarea aplicarea metodei de control On-Off cu histeresis cu actionare prin releu

b) control turatii motor cu aplicarea metodei PID cu un encoder in calitate de sensor, si driver L298 pentru aplicarea puterii la motor.

NOTA: in p (b) se poate alege si la alt parametru de control, cu constrangerea ca actionarea va fi cu o rezolutie de min 8 biti.

2. Set point (valoarea de referinta pentru control) se va seta de la una din surse, la alegere

- un potentiometru

- doua butoane pentru UP/Down

- sensor encoder

- keypad

- interfata serial

3. Valoarea de Setpoint si cea Curenta se vor afisa la LCD.

**Cod sursa part 1 :**

#include <Arduino.h>

#include <../.pio/libdeps/megaatmega2560/Keypad/src/Keypad.h>

#include <../.pio/libdeps/megaatmega2560/LiquidCrystal\_I2C/LiquidCrystal\_I2C.h>

#include <../.pio/libdeps/megaatmega2560/OneWire/OneWire.h>

#include <../.pio/libdeps/megaatmega2560/DallasTemperature/DallasTemperature.h>

// sudo chmod a+rw /dev/ttyACM0

#include <Keypad.h>

#include <stdio.h>

#define RELAY\_PIN 12

#define ONE\_WIRE\_BUS 10

OneWire oneWire(ONE\_WIRE\_BUS);

DallasTemperature sensors(&oneWire);

const byte numRows = 4;

const byte numCols = 4;

static FILE uartout = {0};

LiquidCrystal\_I2C lcd(0x27, 20, 4);

char keymap[numRows][numCols] =

{

{'1', '2', '3', 'A'},

{'4', '5', '6', 'B'},

{'7', '8', '9', 'C'},

{'\*', '0', '#', 'D'}

};

byte rowPins[numRows] = {9, 8, 7, 6}; byte colPins[numCols] = {5, 4, 3, 2};

Keypad myKeypad = Keypad(makeKeymap(keymap), rowPins, colPins, numRows,

numCols);

char command;

float temperature;

int thermostat = 30;

int maxLimit; int minLimit;

bool isEngine = false;

unsigned long previousMillis = 0;

const long interval = 1000;

String tempStr; String thermostatStr; String isEngineStr;

void changeTemp(char command);

static int my\_putChar(char ch, FILE \*stream) {

lcd.print(ch); // change to lcd

return 0;

}

int my\_GetChar(FILE \*f) {

char keypressed = myKeypad.getKey(); return keypressed ? keypressed : NO\_KEY;

}

void setTempLimits(int temp) {

maxLimit = temp + 1;

minLimit = temp - 1;

}

void controlEngine() {

if (isEngine) {

digitalWrite(RELAY\_PIN, HIGH);

} else {

digitalWrite(RELAY\_PIN, LOW);

}

}

void maintainTemperature(int temp) { // Hysteresis

setTempLimits(temp);

if (temperature >= maxLimit) { isEngine = true;

} else if (temperature <= minLimit) { isEngine = false;

}

controlEngine();

}

void changeTemp(char command) {

if (command == NO\_KEY) { return;

} else if (command == '0') { thermostat--;

} else if (command == '#') { thermostat++;

}

}

void printTemp() {

unsigned long currentMillis = millis();

if (currentMillis - previousMillis >= interval) {

previousMillis = currentMillis;

tempStr = String(temperature) + String((char) 178) + "C";

thermostatStr = String(thermostat) + String((char) 178) + "C";

isEngineStr = String(isEngine);

lcd.clear();

printf("Temp: %s", tempStr.c\_str());

lcd.setCursor(0, 1);

printf("Set temp: %s", thermostatStr.c\_str());

lcd.setCursor(0, 2);

printf("%s", isEngineStr.c\_str());

} }

void setup() {

fdev\_setup\_stream (&uartout, my\_putChar, my\_GetChar, \_FDEV\_SETUP\_RW);

stdin = stdout = &uartout;

Serial.begin(9600);

lcd.init();

lcd.backlight();

pinMode(RELAY\_PIN, OUTPUT);

digitalWrite(RELAY\_PIN, LOW);

sensors.begin();

}

void loop() { // comanda de la tastatura

scanf("%c", &command);

changeTemp(command);

sensors.requestTemperatures(); // Send the command to get temperatures

temperature = sensors.getTempCByIndex(0);

maintainTemperature(thermostat);

printTemp();

}**Part 2:**

#include <Arduino.h>

#include <stdio.h>

#include <../.pio/libdeps/uno/LiquidCrystal\_I2C/LiquidCrystal\_I2C.h>

extern "C" {

#include <../.pio/libdeps/uno/PID\_C/src/pid.h>

}

// read rotations of motor

#define ENC1 2

// in1, in2, pwm of motor

#define PWM\_PIN 4

#define IN2 5

#define IN1 6

// constants

#define PULSES\_PER\_ROTATION 110 // pulses per revolution

#define TIME\_UNIT\_SECONDS 60 // seconds in desired time for minutes 60

#define MILLIS\_IN\_SEC 1000

#define MEASURING\_INTERVAL 100

LiquidCrystal\_I2C lcd(0x27, 20, 4);

unsigned long displayLast;

// Rpm variables

volatile unsigned int pulses = 0;

unsigned long millisPrev;

double rpm;

// PID

pid\_t pid;

double setpoint = 135;

double kp = 1;

double ki = 0;

double kd = 0;

double min = 0;

double max = 40;

static FILE uartout = {0};

static int my\_putChar(char ch, FILE \*stream) {

lcd.print(ch);

return 0;

}

void pulsecount() {

pulses++;

}

void readRpm() {

detachInterrupt(digitalPinToInterrupt(ENC1));

rpm = ((double) TIME\_UNIT\_SECONDS \* MEASURING\_INTERVAL /

PULSES\_PER\_ROTATION) /

(millis() - millisPrev) \* pulses \* MILLIS\_IN\_SEC / MEASURING\_INTERVAL;

millisPrev = millis();

pulses = 0;

attachInterrupt(digitalPinToInterrupt(ENC1), pulsecount, RISING);

}

void setup() {

Serial.begin(115200);

lcd.init();

lcd.backlight();

pinMode(ENC1, INPUT); pinMode(PWM\_PIN, OUTPUT);

pinMode(IN2, OUTPUT); pinMode(IN1, OUTPUT);

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

analogWrite(PWM\_PIN, 255);

attachInterrupt(digitalPinToInterrupt(ENC1), pulsecount, RISING);

fdev\_setup\_stream (&uartout, my\_putChar, NULL, \_FDEV\_SETUP\_RW);

stdout = &uartout;

printf("Start");

delay(2000);

lcd.clear();

printf("rpm: ");

lcd.setCursor(0, 1);

printf("setpoint: ");

lcd.setCursor(0, 2);

printf("output: ");

displayLast = millis();

pid\_init(&pid,

setpoint,

kp, ki, kd, min, max);

}

void loop() {

int setSetpoint;

if (Serial.available() > 0) {

setSetpoint = Serial.parseInt();

if (setSetpoint > 0) {

setpoint = setSetpoint;

}

}

if (millis() - millisPrev >= MEASURING\_INTERVAL) {

readRpm();

double output = pid\_compute(&pid, rpm);

analogWrite(PWM\_PIN, output \* 6.375);

}

if (millis() - displayLast >= 500) {

lcd.setCursor(5, 0);

printf("%f", rpm);

lcd.setCursor(10, 1);

printf("%f", setpoint);

Serial.print("rpm: ");

Serial.print(rpm);

Serial.print(" setpoint: ");

Serial.print(setpoint);

Serial.println();

displayLast = millis();

} }

**Laboratorul 6**

Sa se realizeze o aplicatie ce va implementa Automatele finite dupa cum urmeaza:

 Proiectare Automat Finit aplicatie Button-Led.

 Proiectare Automat Finit aplicatie Semafor

Obiective :

 a se utiliza interfata seriala pentru rapoarte de functionare a automatelor

 Reutilizati la maxim solutiile prezentate in laboratoarele precedente

 revizuiti resursele predate la curs

**Teorie :**

Ce este un Automat finit?

Un automat finit (AF) sau o "mașină cu un număr finit de stări" este un model de comportament compus din stări, tranziții și acțiuni. O stare stochează informații despre trecut, adică reflectă schimbările intrării de la inițializarea sistemului până în momentul de față. O tranziție indică o schimbare de stare și este descrisă de o condiție care este nevoie să fie îndeplinită pentru a declanșa tranziția.

O acțiune este o descriere a unei activități ce urmează a fi executată la un anumit moment.

Există câteva tipuri de acțiuni:

 Acțiune de intrare executată la intrarea într-o stare

 Acțiune de ieșire executată la ieșirea dintr-o stare

 Acțiune de intrare de date acțiune executată în funcție de starea prezentă și de datele de intrare

 Acțiune de tranziție acțiune executată în momentul unei tranziții

**Cod sursa :**

#include <Arduino.h>

#include <stdio.h>

#include <../.pio/libdeps/uno/LiquidCrystal\_I2C/LiquidCrystal\_I2C.h>

LiquidCrystal\_I2C lcd(0x27, 20, 4);

// center

#define echoPinC 2

#define trigPinC 3

// left

#define echoPinL 4

#define trigPinL 5

// right

#define echoPinR 6

#define trigPinR 7

// left motor

#define PWM\_L 11

#define EN\_L1 12

#define EN\_L2 13

// right motor

#define EN\_R1 8

#define EN\_R2 9

#define PWM\_R 10

// motor speeds

#define LEFT\_SPEED 225

#define RIGHT\_SPEED 255

#define FILTER\_COUNTER 3

static FILE uartout = {0};

static int my\_putChar(char ch, FILE \*stream) {

lcd.print(ch);

return 0;

}

enum States

{

FORWARD,

TURNING\_LEFT,

TURNING\_RIGHT,

STOP

};

int currState = FORWARD;

double filterSonicMed(int echo, int trigger)

{

double distanceFiltered = 0;

for (int k = 0; k < FILTER\_COUNTER; k++)

{

double distance;

unsigned long duration;

digitalWrite(trigger, LOW);

delayMicroseconds(2);

digitalWrite(trigger, HIGH);

delayMicroseconds(10);

digitalWrite(trigger, LOW);

duration = pulseIn(echo, HIGH);

distance = (double)duration \* 0.034 / 2;

distanceFiltered += distance;

}

return distanceFiltered / FILTER\_COUNTER;

}

void printDistSerial(double center, double left, double right)

{

Serial.print("Center: ");

Serial.print(center);

Serial.print(" cm ");

Serial.print("Left: ");

Serial.print(left);

Serial.print(" cm ");

Serial.print("Right: ");

Serial.print(right);

Serial.println(" cm ");

}

void printDistLcd(double distance, int x, int y)

{

String distanceStr = String((int)distance) + " ";

lcd.setCursor(x, y);

printf("%s", distanceStr.c\_str());

}

int checkState(double c, double l, double r)

{

if (c > 100 && l > 100 && r > 100)

{

return FORWARD;

}

if (c < 10)

{

return STOP;

}

if (currState == TURNING\_LEFT)

{

return TURNING\_LEFT;

}

if (currState == TURNING\_RIGHT)

{

return TURNING\_RIGHT;

}

if (l < c && l < r)

{

return TURNING\_LEFT;

}

if (r < c && r < l)

{

return TURNING\_RIGHT;

}

if (c < l && c < r)

{

return FORWARD;

}

return -1;

}

void goForward()

{

// left

digitalWrite(EN\_L1, LOW);

digitalWrite(EN\_L2, HIGH);

analogWrite(PWM\_L, LEFT\_SPEED);

// right

digitalWrite(EN\_R1, LOW);

digitalWrite(EN\_R2, HIGH);

analogWrite(PWM\_R, RIGHT\_SPEED);

}

void goLeft()

{

// left

digitalWrite(EN\_L1, LOW);

digitalWrite(EN\_L2, LOW);

analogWrite(PWM\_L, 0);

// right

digitalWrite(EN\_R1, LOW);

digitalWrite(EN\_R2, HIGH);

analogWrite(PWM\_R, RIGHT\_SPEED);

}

void goRight()

{

// left

digitalWrite(EN\_L1, LOW);

digitalWrite(EN\_L2, HIGH);

analogWrite(PWM\_L, LEFT\_SPEED);

// right

digitalWrite(EN\_R1, LOW);

digitalWrite(EN\_R2, LOW);

analogWrite(PWM\_R, 0);

}

void stop()

{

// left

digitalWrite(EN\_L1, LOW);

digitalWrite(EN\_L2, LOW);

analogWrite(PWM\_L, 0);

// right

digitalWrite(EN\_R1, LOW);

digitalWrite(EN\_R2, LOW);

analogWrite(PWM\_R, 0);

}

void setup()

{

lcd.init();

lcd.backlight();

fdev\_setup\_stream (&uartout, my\_putChar, NULL, \_FDEV\_SETUP\_RW);

stdout = &uartout;

// ultra sonic

pinMode(trigPinC, OUTPUT);

pinMode(echoPinC, INPUT);

pinMode(trigPinL, OUTPUT);

pinMode(echoPinL, INPUT);

pinMode(trigPinR, OUTPUT);

pinMode(echoPinR, INPUT);

// motor control

pinMode(PWM\_L, OUTPUT);

pinMode(EN\_L1, OUTPUT);

pinMode(EN\_L2, OUTPUT);

pinMode(PWM\_R, OUTPUT);

pinMode(EN\_R1, OUTPUT);

pinMode(EN\_R1, OUTPUT);

Serial.begin(9600);

goForward();

printf("Distance C: ");

lcd.setCursor(0, 1);

printf("Distance L: ");

lcd.setCursor(0, 2);

printf("Distance R: ");

}

void loop()

{

double distanceC = filterSonicMed(echoPinC, trigPinC);

double distanceL = filterSonicMed(echoPinL, trigPinL);

double distanceR = filterSonicMed(echoPinR, trigPinR);

printDistSerial(distanceC, distanceL, distanceR);

printDistLcd(distanceC, 12, 0);

printDistLcd(distanceL, 12, 1);

printDistLcd(distanceR, 12, 2);

// function to veryfy state

currState = checkState(distanceC, distanceL, distanceR);

// switch

switch (currState)

{

case FORWARD:

goForward();

break;

case TURNING\_LEFT:

if (distanceC < distanceL)

{

currState = FORWARD;

}

else

{

goLeft();

}

break;

case TURNING\_RIGHT:

if (distanceC < distanceR)

{

currState = FORWARD;

}

else

{

goRight();

}

break;

case STOP:

stop();

break;

default:

break;

}

**}**

**Laboratorul 7**

1. Protocol fizic de comunicare - Comunicarea intre DOUA Microcontrollere prin interfata I2C

 MCU1 - implementeaza sensorul digital cu interfata I2C pentru sensorul ultrasonic HCS-04,

unde se ececuta colectarea datelor de la interfata sensorului si se retransmite catre interfata I2C

la detectarea unei cereri de citire a datelor.

 MCU2 - executa cererea prin interfata I2C catre sesorul digital ultrasonic (MCU+HCS-04)

2. Protocol logic de comunicare - cererea de date prin interfata serial, in format text respectand un

protocol de comunicare care va avea campurile:

 indicator de start pachet

 indicator de sfarsit

 contorizare pachete

 ID emitator

 ID receptor

 tipul pachetului

 <alte campuri optional>

 date pachet - Payload

 suma de control - suma tuturor valorilor numerice din pachet cererile venite din interfata seriala vor fi verificate dupa patern, si in caz de pachet valid se va intereta comanda. se va raspunde cu un pachet conform aceluias protocol comanda obligatorie pentru implementare este cererea de date de la sensorul digital implementat in

p 1. sa si implementezi inca o camanda la alegere, pentru diversitate.

3. realizarea conexiunii si raportarii datelor achizitionate catre un MQTT server cum ar

fi https://thingsboard.io sau echivalent

**Part1:**

Master (Arduino nr. 1)

// include the library code:

#include <LiquidCrystal.h>

// Includem biblioteca necesara pentru I2C

#include <Wire.h>

long newTime = 0, currentTime = 0;

int interval = 300;

String str = "";

char character;

int red\_pin = 10;

int blue\_pin = 9;

int green\_pin = 8;

// initialize the library with the numbers of the interface pins

LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

void readChars() {

while (Wire.available() > 0) {

character = Wire.read();

str += character;

}

Serial.println(str);

if (str[0] == 'D') //am primit D, deci urmeaza sa citim distanta

{

float temperatura = str.substring(1).toFloat();

Serial.println(temperatura);

lcd.setCursor(0, 1);

lcd.print(temperatura);

}

}

void setup() {

pinMode(red\_pin, OUTPUT);

pinMode(blue\_pin, OUTPUT);

pinMode(green\_pin, OUTPUT);

// Dechidem magistrala I2C ca master

Serial.begin(9600);

Wire.begin(1);

lcd.begin(16, 2);

lcd.print("Master");

}

void loop() {

str = "";

currentTime = millis();

if (currentTime - newTime >= interval) {

newTime = currentTime;

Wire.beginTransmission(2); // transmitem spre device #2

Wire.write('D');

Wire.endTransmission();

Wire.requestFrom(2, 5);

readChars();

**Slave (Arduino nr. 2)**

// Includem biblioteca wire pentru I2C

#include <Wire.h>

#define echoPin 2

#define trigPin 3

int x = 0;

char buffer[6];

float dist;

void receiveEvent(int bytes) {

x = Wire.read(); // citim un character din I2C

}

void requestEvent(){

if(x=='D')

{

Wire.write('D');

dtostrf(dist, 5, 2, buffer);

Serial.println(buffer);

Wire.write(buffer);

}

}

double readDistance() {

double distance;

unsigned long duration;

digitalWrite(trigPin, LOW);

delayMicroseconds(2);

digitalWrite(trigPin, HIGH);

delayMicroseconds(10);

digitalWrite(trigPin, LOW);

duration = pulseIn(echoPin, HIGH);

distance = (double)duration \* 0.034 / 2;

return distance;

}

void setup() {

// Pornim busul I2C ca si slave la adresa 2

Wire.begin(2);

// Atasam o functie care sa se declanseze atunci cand primim ceva

Wire.onRequest(requestEvent);

Wire.onReceive(receiveEvent);

Serial.begin(9600);

pinMode(trigPin, OUTPUT);

pinMode(echoPin, INPUT);

}

void loop() {

dist = readDistance();

}]