**Universitatea “Politehnica” Timişoara**

**Facultatea de Automaticã şi Calculatoare**

**PROIECT SINCRETIC**

**Sistem de monitorizare și reglare a temperaturii cu regulator PID și management al perturbațiilor**

**Coordonator: Prof. Dr. Ing. Nanu Sorin**

**Proiectanți: Mihălțan Șerban, Mandache Eduard  
Anul III, grupa 4.1, specializarea Ingineria Sistemelor**

**2024**

#### Cuprins:

1. **Introducere**
   1. **Context și motivație**
   2. **Natura sincretică**
2. **Obiectivele proiectului**
   1. **Funcţiile realizate de sistem**
   2. **Interfaţa cu utilizatorul**
   3. **Reglarea temperaturii**
   4. **Simularea de perturbaţii**
3. **Metodologia**
4. **Dezvoltarea**
   1. **Conectarea componentelor**
   2. **Scrierea codului pentru citirea și afișarea temperaturii**
   3. **Implementarea funcționalității butoanelor**
   4. **Implementarea codului pentru simularea perturbațiilor**
   5. **Realizarea circuitului de încălzire controlat de tranzistor**
5. **Problemele intâmpinate**
   1. **Probleme de comunicare serială**
   2. **Identificarea unei surse de căldură eficiente**
   3. **Modularizarea codului**
6. **Bibliografie**
7. **Anexe**

## **Introducere**

**1.1 Context și Motivație**

În era actuală, ingineria sistemelor joacă un rol esențial în dezvoltarea de soluții complexe pentru controlul și monitorizarea diferitelor procese, atât industriale, cât și casnice. Proiectul propus a fost ales ca o activitate practică în cadrul facultății, având scopul de a consolida cunoștințele teoretice acumulate și de a le aplica într-un context real de inginerie. Proiectul se situează la intersecția dintre diverse discipline, cum ar fi electronică, programare, și control automat, oferind o oportunitate excelentă de a dezvolta abilități variate și complementare.

**1.2 Natura sincretică**

Natura sincretică a proiectului se reflectă în îmbinarea de cunoștințe și tehnici din diverse domenii precum programarea în C++, electronica, teoria controlului proceselor și algoritmica. Fiecare dintre aceste componente contribuie esențial la funcționarea optimă a sistemului, iar sinergia dintre ele dovedește o abordare multidisciplinară caracteristică ingineriei sistemelor moderne.

Partea software a proiectului presupune dezvoltarea unui cod C++ capabil să gestioneze interfața cu utilizatorul, să stocheze parametri în memoria nevolatilă și să controleze funcționarea regulatorului PID, pentru a obține o stabilitate termică optimă. Partea hardware al proiectului implică integrarea unui microcontroller care controlează senzorul de temperatură, elementul de execuție și componentele de afișare LCD.

Aceste elemente demonstrează cum abordarea sincretică oferă posibilitatea dezvoltării unui sistem complex, cu multiple funcționalități, în care fiecare domeniu contribuie semnificativ la obținerea unui rezultat complet funcțional. Astfel, proiectul devine o experiență valoroasă de învățare practică, pregătindu-mă pentru viitoarele cerințe profesionale din domeniul ingineriei sistemelor.

## **Obiectivele proiectului**

### **2.1 Funcţiile realizate de sistem:**

* încălzirea sistemului până la o temperatura setată **TSET** într-un timp **tîncălzire;**
* menţinerea acestei temperaturi pentru o perioada de timp **tmenținere;**
* răcirea sistemului treptat într-o perioadă de timp **trăcire.**

### **2.2 Interfaţa cu utilizatorul:**

* Sistemul va dispune de un display LCD pe care va fi afişat meniul, iar în timpul rulării

se vor afişa temperatura setată, temperatura curentă şi timpul rămas din etapa

actuală (**tîncălzire, tmenținere, trăcire).**

* Meniul sistemului va permite modificarea următorilor parametri:
  + **TSET;**
  + **tîncălzire;**
  + **tmenținere;**
  + **trăcire;**
  + **KP;**
  + **KI;**
  + **KD.**
* Parametrii vor fi salvaţi în memoria nevolatilă iar repornirea sistemului nu va afecta

parametrii salvaţi.

* Navigarea prin meniul echipamentului se va face prin patru butoane: “**OK**”, “**Cancel**”, “**+**”, “**-**”.

### **2.3 Reglarea temperaturii:**

* Controlul temperaturii va fi asigurat de un regulator de tip PID.
* Elementul de execuţie al sistemului va asigura încălzirea senzorului va fi un bec la o temperatura de minim 50°C
* Se va folosi un tranzistor pentru a comanda pornirea/oprirea alimentării

elementului de execuție ales de către microcontroller.

### **2.4 Simularea de perturbaţii:**

* Se va implementa un generator de numere aleatoare (**PRNG**).
* Sistemul va dispune de un generator de perturbații care va afecta circuitul de

masurare a temperaturii.

* Apăsarea butonului “**OK**” în timpul rulării unui program de reglare a temperaturii va

porni modulul de perturbaţii, iar apăsarea din nou a aceluiaşi buton o va opri.

## **Metodologie**

Metodologia acestui proiect a fost centrată pe aplicarea practică și experimentală a conceptelor teoretice. Metoda principală utilizată a fost experimentarea proprie, combinând testele fizice și simulările virtuale pentru a asigura un proces eficient de proiectare și validare.

Pentru simulările inițiale și verificările rapide ale circuitelor și codului, a fost utilizată platforma **Tinkercad**, un mediu de simulare online care permite crearea și testarea unor montaje electronice de bază cu microcontroller.

De asemenea, pentru testarea cu echipamentele fizice, a fost folosită **placa Arduino UNO**, cu microcontroller-ul ATmega328P, care a servit drept bază pentru montajul experimental. Arduino IDE a fost folosit pentru a scrie, compila și încărca codul în microcontroller, iar funcția **Serial Plotter** integrată în IDE a permis urmărirea în timp real a citirilor a evoluției temperaturii.

## **Dezvoltarea**

Dezvoltarea proiectului a urmat seria de etape enumerate și dezvoltate mai jos, fiecare având un rol specific în crearea sistemului de control al temperaturii:

### **4.**1 **Conectarea componentelor**

Inițial, am conectat senzorul de temperatură **LM35** și display-ul **LCD 16x2** la placa **Arduino UNO**. Pinul de ieșire al **LM35** a fost conectat la un pin analogic al plăcii Arduino pentru a citi valorile de temperatură. LCD-ul a fost conectat la pinii digitali, utilizând pinul **RS** pentru control și pini suplimentari pentru alimentare și contrast. Această configurare a permis afișarea valorilor de temperatură și navigarea în meniul sistemului direct pe display.

### **4.2 Scrierea codului pentru citirea și afișarea temperaturii**

Am implementat codul pentru citirea datelor de temperatură și afișarea lor pe LCD. Datele preluate de la **LM35** au fost procesate pentru a converti tensiunea în valori de temperatură și afișate în timp real pe display.

### **4.3 Implementarea funcționalității butoanelor**

Am adăugat codul pentru controlul butoanelor, permițând utilizatorului să seteze parametrii de temperatură și să navigheze prin meniul sistemului. Aceste butoane ("**OK**", "**Cancel**", "**+**", "**-**") permit ajustarea valorilor și setărilor fără a interveni direct în cod.

### **4.4** **Implementarea codului pentru simularea perturbațiilor**

Am adăugat un modul de generare a perturbațiilor pentru a testa reacția sistemului la schimbări neașteptate ale temperaturii. Acest test a fost realizat cu ajutorul unui algoritm de generare de numere pseudo-aleatoare (**PRNG**) care introduce fluctuații controlate, permițând o evaluare mai realistă a performanțelor **PID**. Am adăugat ulterior un buton pentru start/stop perturbații pentru a facilita controlul sistemului.

### **4.5 R****ealizarea circuitului de încălzire controlat de tranzistor**

Pentru a controla sursa de căldură, am realizat un circuit secundar în care un tranzistor **BC556** controlează alimentarea unui bec cu halogen. Tranzistorul este activat sau dezactivat de Arduino, permițând sistemului să gestioneze temperatura prin comutarea sursei de căldură în funcție de setările **PID**.

### **4.6 Implementarea regulatorului PID în software**

Am dezvoltat algoritmul **PID** pentru a gestiona controlul precis al temperaturii. Acesta utilizează parametrii **Kp**, **Ki** și **Kd** pentru a ajusta ieșirea, asigurând astfel atingerea și menținerea temperaturii dorite, chiar și în condiții de perturbații simulate.

## **Problemele întâmpinate**

### **5.1 Probleme de comunicare serială**

Inițial, LCD-ul era conectat la pinul 11 **(PB3)**, însă acesta a fost mutat pe pinul 9 **(PB1)** pentru a elibera pinul 11. Această modificare ne-a permis utilizarea tranzistorului, și a semnalului PWM de pe Timer2, pentru controlul temperaturii, în paralel cu utilizarea Timer1 folosit pe post de cronometru (counting mode).

### **5.2 Identificarea unei surse de căldură eficiente**

Încă un obstacol a fost găsirea unei surse de căldură care să îndeplinească cerințele de temperatură și stabilitate necesare pentru proiect. După mai multe teste nereușite cu diverse componente, am decis să utilizăm un bec cu halogen folosit în farurile auto, care s-a dovedit a fi o soluție adecvată. Acesta oferă atât o putere termică suficientă pentru a atinge temperaturile dorite, cât și o stabilitate acceptabilă, contribuind astfel la funcționalitatea generală a sistemului. Alegerea acestei soluții a permis finalizarea circuitului și testarea completă a algoritmului de control PID.

## **Bibliografie**

IDE-ul folosit pentru programarea Arduino Uno:

<https://docs.arduino.cc/software/ide/>  
Datasheet Arduino Uno:

<https://docs.arduino.cc/resources/datasheets/ABX00087-datasheet.pdf>  
Datasheet bec far auto cu halogen:  
<https://eprel.ec.europa.eu/screen/product/lightsources/804078>

Datasheet tranzistor:

<https://www.datasheet4u.com/datasheet-pdf/FairchildSemiconductor/2N718A/pdf.php?id=73072>

Datasheet display LCD 2x16:

<https://www.datasheet4u.com/datasheet-pdf/Crystalfontz/CFAH1602A-AGB-JP/pdf.php?id=1109877>

Thomas L. Floyd - ELECTRONIC DEVICES Electron Flow Version Ninth Edition

## **Anexe**

A circuit board with colorful wires

Description automatically generatedSchemă montaj:

Schemă electrică:

A screenshot of a computer

Description automatically generated

Lista cu componentele utilizate în cadrul proiectului:

|  |  |
| --- | --- |
| Componentă | Cantitate |
| Arduino Uno R3 | 1 |
| Display LCD 16x2 CFAH1602A-AGB-JP | 1 |
| Rezistență 1 kΩ | 6 |
| Senzor temperatură LM35 | 1 |
| Buton tactil | 5 |
| Tranzistor NPN 2N718A | 1 |
| Bec far auto cu halogen EL0068374 | 1 |
| Baterie AA 1.5V | 4 |

Codul folosit pentru realizarea sistemului, împărțit pe fișiere:

* Fișierul principal sketch.ino

#include <EEPROM.h>

#include <LiquidCrystal.h>

#include "Time.h"

#include "PID.h"

#include "Menu.h"

#include "Phase.h"

#define TP35\_SENSOR\_CHANNEL 0 // Setare pin senzor de temp A0

#define ADC\_REF\_VOLTAGE 5.0   // Setare tensiune de referinta 5V

// Salvarea parametrilor dupa fiecare modificare(clasa Menu) fct x1

// TEMP,TI,TM,TR,KP,KI,KD

LiquidCrystal lcd\_1(12, 9, 5, 4, 3, 2);

// Declararea variabilelor globale

Time TI, TM, TR, TRAMAS;

float TEMP = 0, TEMPACT;

int perturb = 0;

float KI = 0, KP = 0, KD = 0, lasterror;

Phase phase;

Menu menu;

void init\_adc()

{

    ADMUX |= (1 << REFS0);                                 // Setare tensiune de alimentare ca tensiune de referinta ADC

    ADCSRA |= (1 << ADPS2) | (1 << ADPS1) | (1 << ADPS0); // Setare prescaler clock ADC la 128

    ADCSRA |= (1 << ADEN);                               // Enable ADC

}

float read\_temperature() // Functie de citire a datelor din ADC

{

    ADMUX &= 0xF0;                  // Setarea pinilor din ADMUX pentru citire

    ADMUX |= TP35\_SENSOR\_CHANNEL;  // Setarea pinului la care este legat senzorul de temperatura (A0)

    ADCSRA |= (1 << ADSC);        // Start conversie ADC

    while (ADCSRA & (1 << ADSC)) // Asteptare finalizat conversie ADC

    {}

    uint16\_t adc\_value = ADC;                                     // Calculul temperaturii folosind rezultatul de la ADC

    float voltage = (float)adc\_value \* ADC\_REF\_VOLTAGE / 1024.0; // Transformare valoare analogica in tensiune

    float temperature = voltage \* 100.0;                         // Transformare tensiune in grade celsius (fara offset)

    return temperature;

}

void setup()

{

    lcd\_1.begin(16, 2); // Configurare numar linii si coloane pe LCD

    init\_adc();

    // Citire din memoria nonvolatila a parametrilor

    // TEMP,TI,TM,TR,KP,KI,KD

    Time t(0, 1, 0);

    int adress = 0;

    EEPROM.get(adress, TEMP);

    adress += sizeof(TEMP);

    EEPROM.get(adress, TI);

    adress += sizeof(TI);

    EEPROM.get(adress, TM);

    adress += sizeof(TM);

    EEPROM.get(adress, TR);

    adress += sizeof(TR);

    EEPROM.get(adress, KP);

    adress += sizeof(KP);

    EEPROM.get(adress, KI);

    adress += sizeof(KI);

    EEPROM.get(adress, KD);

    adress += sizeof(KD);

    //TRAMAS = TI = TM = TR = t;  // Variabile folosite pentru simulare

    //TEMP = 50;

    //KP=2.0;

    //KI=5.0;

    //KD=1.0;

    //menu.select();

    phase.setphase(0);

    randomSeed(analogRead(0));

    DDRB |= (1 << DDB3); // Setare pin 9 (PB1) drept output

    // Setare Timer 2 in modul de functionare Fast PWM mode pe pinul 9 (PB1) fara prescaler

    TCCR2A |= (1 << COM2A1) | (1 << COM2B1) | (1 << WGM21) | (1 << WGM20);

    TCCR2B |= (1 << CS20) | (1 << CS20);

     // Setare Timer in modul de functionare normal

    // TCCR1A pentru Timer 1

    TCCR1A = 0;

     // Timer-ul are un prescaler de 1024 care se seteaza folosind bitii de tip CS

    // TCCR1B pentru Timer 1

    TCCR1B = (1 << CS12) | (1 << CS10);

    Serial.begin(9600);

}

void loop()

{

    if (TCNT1 > 15625) // Valoarea incrementata de timer TCNT(0,1,2) si numarul de evenimente intr-o secunda

    {

        lcd\_1.clear();

        TRAMAS--;

        TEMPACT = read\_temperature();

        if (PINC & 1 << 5)

        {

            perturb = perturb ^ 1; // Adaugare perturbatii

        }

        TEMPACT += (float)random(-10, 10) / 5 \* perturb;

        Serial.println(TEMPACT);

        phase.select();

        menu.select();

        TCNT1 = 0; // Resetare variabila timer

       // delay(1000);

    }

}

* Fișierul Menu.h

#ifndef MENU\_H

#define MENU\_H

#include <EEPROM.h>

#include <LiquidCrystal.h>

#include "Time.h"

#include "Phase.h"

extern Phase phase;

class Menu // Meniul de selectie si modificare a parametriilor

{

private:

    int temp;

    float ki, kd, kp;

    Time ti, tm, tr;

public:

    int state;

    Menu();

    void select();

private:

    void base\_mode();

    void mode1();

    void mode2();

    void mode3();

    void mode4();

    void mode5();

    void mode6();

    void mode7();

    void mode11(); // set Temp

    void mode21(); // set timp incalzire

    void mode31(); // set timp mentinere

    void mode41(); // set timp racire

    void mode51(); // set KP

    void mode61(); // set KI

    void mode71(); // set KD

};

#endif

* Fișierul Menu.cpp

#include "Menu.h"

#include "Time.h"

#include "Phase.h"

extern LiquidCrystal lcd\_1;

extern int TEMP;

extern float TEMPACT;

extern float KP, KI, KD;

extern float lasterror;

extern Time TI, TM, TR, TRAMAS;

Menu::Menu()

{

    this->state = 0;

}

void Menu::select()

{

    switch (state)

    {

        case 0: // default

            this->base\_mode();

            break;

        case 1:

            this->mode1();

            break;

        case 2:

            this->mode2();

            break;

        case 3:

            this->mode3();

            break;

        case 4:

            this->mode4();

            break;

        case 5:

            this->mode5();

            break;

        case 6:

            this->mode6();

            break;

        case 7:

            this->mode7();

            break;

        case 11: // set T

            this->mode11();

            break;

        case 21: // set ti

            this->mode21();

            break;

        case 31: // set tm

            this->mode31();

            break;

        case 41: // set tr

            this->mode41();

            break;

        case 51: // set kp

            this->mode51();

            break;

        case 61: // set ki

            this->mode61();

            break;

        case 71: // set kd

            this->mode71();

            break;

        default:

            break;

    }

}

void Menu::base\_mode()

{

    lcd\_1.setCursor(0, 0);

    lcd\_1.print("TA: ");

    lcd\_1.setCursor(3, 0);

    lcd\_1.print(TEMPACT);

    lcd\_1.setCursor(9, 0);

    lcd\_1.print("T: ");

    lcd\_1.setCursor(11, 0);

    lcd\_1.print(TEMP);

    TRAMAS.display\_time(0, 1);

    lcd\_1.setCursor(11, 1);

    lcd\_1.print(phase.get\_symbol());

    if (PINC & 1 << 4) this->state = 1;

}

void Menu::mode1()

{

    lcd\_1.setCursor(0, 0);

    lcd\_1.print("Set temperature");

    temp = TEMP;

    if (PINC & 1 << 4)

        this->state = 11;

    else if (PINC & 1 << 3)

        this->state = 0;

    else if (PINC & 1 << 2)

        this->state = 7;

    else if (PINC & 1 << 1)

        this->state = 2;

}

void Menu::mode2()

{

    lcd\_1.setCursor(0, 0);

    lcd\_1.print("Set tincalzire");

    ti = TI;

    if (PINC & 1 << 4)

        this->state = 21;

    else if (PINC & 1 << 3)

        this->state = 0;

    else if (PINC & 1 << 2)

        this->state = 1;

    else if (PINC & 1 << 1)

        this->state = 3;

}

void Menu::mode3()

{

    lcd\_1.setCursor(0, 0);

    lcd\_1.print("Set tmentinere");

    tm = TM;

    if (PINC & 1 << 4)

        this->state = 31;

    else if (PINC & 1 << 3)

        this->state = 0;

    else if (PINC & 1 << 2)

        this->state = 2;

    else if (PINC & 1 << 1)

        this->state = 4;

}

void Menu::mode4()

{

    lcd\_1.setCursor(0, 0);

    lcd\_1.print("Set tracire");

    tr = TR;

    if (PINC & 1 << 4)

        this->state = 41;

    else if (PINC & 1 << 3)

        this->state = 0;

    else if (PINC & 1 << 2)

        this->state = 3;

    else if (PINC & 1 << 1)

        this->state = 5;

}

void Menu::mode5()

{

    lcd\_1.setCursor(0, 0);

    lcd\_1.print("Set KP");

    kp = KP;

    if (PINC & 1 << 4)

        this->state = 51;

    else if (PINC & 1 << 3)

        this->state = 0;

    else if (PINC & 1 << 2)

        this->state = 4;

    else if (PINC & 1 << 1)

        this->state = 6;

}

void Menu::mode6()

{

    lcd\_1.setCursor(0, 0);

    lcd\_1.print("Set KI");

    ki = KI;

    if (PINC & 1 << 4)

        this->state = 61;

    else if (PINC & 1 << 3)

        this->state = 0;

    else if (PINC & 1 << 2)

        this->state = 5;

    else if (PINC & 1 << 1)

        this->state = 7;

}

void Menu::mode7()

{

    lcd\_1.setCursor(0, 0);

    lcd\_1.print("Set KD");

    kd = KD;

    if (PINC & 1 << 4)

        this->state = 71;

    else if (PINC & 1 << 3)

        this->state = 0;

    else if (PINC & 1 << 2)

        this->state = 6;

    else if (PINC & 1 << 1)

        this->state = 1;

}

void Menu::mode11() // set Temp

{

    lcd\_1.setCursor(0, 0);

    lcd\_1.print("Temp: ");

    lcd\_1.setCursor(0, 1);

    lcd\_1.print(temp);

    if (PINC & 1 << 4)

    {

        this->state = 1;

        TEMP = temp;

        EEPROM.put(0, TEMP);

        !!!!!!!!!!!!!!!

    }

    else if (PINC & 1 << 3)

        this->state = 1;

    else if (PINC & 1 << 2)

        temp--;

    else if (PINC & 1 << 1)

        temp++;

}

void Menu::mode21()   // set timp incalzire

{

    lcd\_1.setCursor(0, 0);

    lcd\_1.print("Timp Incalzire: ");

    ti.display\_time(0, 1);

    if (PINC & 1 << 4)

    {

        this->state = 2;

        TI = ti;

        EEPROM.put(0 + sizeof(TEMP), TI);

    }

    else if (PINC & 1 << 3)

        this->state = 2;

    else if (PINC & 1 << 2)

        ti--;

    else if (PINC & 1 << 1)

        ti++;

}

void Menu::mode31() // set timp mentinere

{

    lcd\_1.setCursor(0, 0);

    lcd\_1.print("Timp mentinere: ");

    lcd\_1.setCursor(0, 1);

    tm.display\_time(0, 1);

    if (PINC & 1 << 4)

    {

        this->state = 3;

        TM = tm;

        EEPROM.put(0 + sizeof(TEMP) + sizeof(TI), TM);

    }

    else if (PINC & 1 << 3)

        this->state = 3;

    else if (PINC & 1 << 2)

        tm--;

    else if (PINC & 1 << 1)

        tm++;

}

void Menu::mode41()  // set timp racire

{

    lcd\_1.setCursor(0, 0);

    lcd\_1.print("Timp racire: ");

    lcd\_1.setCursor(0, 1);

    tr.display\_time(0, 1);

    if (PINC & 1 << 4)

    {

        this->state = 4;

        TR = tr;

        EEPROM.put(0 + sizeof(TEMP) + sizeof(TI) + sizeof(TM), TR);

    }

    else if (PINC & 1 << 3)

        this->state = 4;

    else if (PINC & 1 << 2)

        tr--;

    else if (PINC & 1 << 1)

        tr++;

}

void Menu::mode51()   // set KP

{

    lcd\_1.setCursor(0, 0);

    lcd\_1.print("KP: ");

    lcd\_1.setCursor(0, 1);

    lcd\_1.print(kp);

    if (PINC & 1 << 4)

    {

        this->state = 5;

        KP = kp;

        EEPROM.put(0 + sizeof(TEMP) + sizeof(TI) + sizeof(TM) + sizeof(TR), KP);

    }

    else if (PINC & 1 << 3)

        this->state = 5;

    else if (PINC & 1 << 2)

        kp -= 0.1;

    else if (PINC & 1 << 1)

        kp += 0.1;

}

void Menu::mode61() // set KI

{

    lcd\_1.setCursor(0, 0);

    lcd\_1.print("KI: ");

    lcd\_1.setCursor(0, 1);

    lcd\_1.print(ki);

    if (PINC & 1 << 4)

    {

        this->state = 6;

        KI = ki;

        EEPROM.put(0 + sizeof(TEMP) + sizeof(TI) + sizeof(TM) + sizeof(TR) + sizeof(KP), KI);

    }

    else if (PINC & 1 << 3)

        this->state = 6;

    else if (PINC & 1 << 2)

        ki -= 0.1;

    else if (PINC & 1 << 1)

        ki += 0.1;

}

void Menu::mode71() // set KD

{

    lcd\_1.setCursor(0, 0);

    lcd\_1.print("KD: ");

    lcd\_1.setCursor(0, 1);

    lcd\_1.print(kd);

    if (PINC & 1 << 4)

    {

        this->state = 7;

        KD = kd;

        EEPROM.put(0 + sizeof(TEMP) + sizeof(TI) + sizeof(TM) + sizeof(TR) + sizeof(KP) + sizeof(KI), KD);

    }

    else if (PINC & 1 << 3)

        this->state = 7;

    else if (PINC & 1 << 2)

        kd -= 0.1;

    else if (PINC & 1 << 1)

        kd += 0.1;

}

* Fișierul PID.h

#ifndef PID\_H

#define PID\_H

void PID();

#endif // PID\_H

* Fișierul PID.cpp

#include "PID.h"

#include "Time.h"

extern int TEMP;

extern int TEMPACT;

extern float KP, KI, KD;

extern float lasterror;

extern Time TI, TM, TR, TRAMAS;

void PID()

{

    Time TDelta(50, 0, 0); // Timpul de incalzire pana la TMAX

    int TEMPDelta = 25;//in 50 de secunde se incalzeste de la 28 la 53 de grade

    int output;

    float error;          // Eroare calculata ca diferenta dintre input si setpoint

    float integral;      // Termenul integrator

    float derivative;   // Termenul derivator

    error = TEMP - TEMPACT;

    if (error > 0)

    {

        float proportional = KP \* error;                  // Termenul proportional

        integral += error \* KI;                          // Termenul integrator

        derivative = (error - lasterror) \* KD;          // Termenul derivator

        output = proportional + integral + derivative; // Output PID

        lasterror = error;

        OCR2A = ((output \* TDelta.total\_seconds()) / (TEMPDelta \* TRAMAS.total\_seconds())) \* 255;

        //(output/tramas)/(TEMPDelta/TDelta)\*255

        // Daca in TDelta s-a incalzit cu TempDelta la 255 (un factor de umplere de 1) at se va incalzi cu output in  de TRAMAS la un factor de umplere de x/255

    }

    else OCR2A = 0;

}

* Fișierul Phase.h

#ifndef PHASE\_H

#define PHASE\_H

class Phase

{

private:

    int state;

    char symbol;

public:

    Phase(int state = 0, char symbol = '/');

    void select();

    void incalzire();

    void racire();

    void mentinere();

    char get\_symbol();

    void setphase(int i);

};

#endif // PHASE\_H

* Fișierul Phase.cpp

#include "Phase.h"

#include "Time.h"

#include "PID.h"

extern LiquidCrystal lcd\_1;

extern int TEMP;

extern int TEMPACT;

extern float KP, KI, KD;

extern float lasterror;

extern Time TI, TM, TR, TRAMAS;

Phase::Phase(int state, char symbol)

{

    this->state = state;

    this->symbol = symbol;

}

// Functie pentru determinarea starii curente a sistemului

void Phase::select()

{

    switch (state)

    {

    case 0:

        this->incalzire();

        break;

    case 1:

        this->mentinere();

        break;

    case 2:

        this->racire();

        break;

    default:

        break;

    }

}

void Phase::incalzire()

{

    PID();

    if (TRAMAS.time\_is\_null())

    {

        this->state = 1;

        this->symbol = '-';

        TRAMAS = TM;

    }

}

void Phase::racire()

{

    OCR2A = 0;

    if (TRAMAS.time\_is\_null())

    {

        this->state = 0;

        this->symbol = '/';

        TRAMAS = TI;

    }

}

void Phase::mentinere()

{

    PID();

    if (TRAMAS.time\_is\_null())

    {

        this->state = 2;

        this->symbol = '\\';

        TRAMAS = TR;

    }

}

char Phase::get\_symbol()

{

    return this->symbol;

}

void Phase::setphase(int i)

{

    switch (i)

    {

    case 0:

        this->state = 0;

        this->symbol = '/';

        TRAMAS = TI;

        break;

    case 1:

        this->state = 1;

        this->symbol = '-';

        TRAMAS = TM;

        break;

    case 2:

        this->state = 2;

        this->symbol = '\\';

        TRAMAS = TR;

        break;

    default:

        break;

    }

}

* Fișierul Time,h

#ifndef TIME\_H

#define TIME\_H

#include <LiquidCrystal.h>

class Time {

public:

    int seconds, minutes, hours;

    // Constructor

    Time(int seconds = 0, int minutes = 0, int hours = 0);

    // Metode

    int time\_is\_null();

    void display\_time(int x, int y);

    void update\_time();

    int total\_seconds();

    // Operator de asignare

    Time operator=(const Time &t);

    // Operatori de incrementare

    Time &operator++();

    Time operator++(int);

    // Operatori de decrementare

    Time &operator--();

    Time operator--(int);

};

#endif // TIME\_H

* Fișierul Time,cpp

#include "Time.h"

#include <LiquidCrystal.h>

extern LiquidCrystal lcd\_1;

// Constructor

Time::Time(int seconds, int minutes, int hours)

    : seconds(seconds), minutes(minutes), hours(hours) {}

int Time::time\_is\_null() {

    return (this->seconds == 0 && this->minutes == 0 && this->hours == 0) ? 1 : 0;

}

void Time::display\_time(int x, int y) {

    lcd\_1.setCursor(x, y);

    if (this->hours < 10) lcd\_1.print('0');

    lcd\_1.print(this->hours);

    lcd\_1.print(":");

    if (this->minutes < 10) lcd\_1.print('0');

    lcd\_1.print(this->minutes);

    lcd\_1.print(":");

    if (this->seconds < 10) lcd\_1.print('0');

    lcd\_1.print(this->seconds);

}

void Time::update\_time() {

    if (!this->time\_is\_null()) {

        if (this->seconds >= 60) {

            this->seconds = 0;

            this->minutes++;

            if (this->minutes >= 60) {

                this->minutes = 0;

                this->hours++;

            }

        }

        if (this->seconds < 0) {

            this->seconds = 59;

            this->minutes--;

            if (this->minutes < 0) {

                this->minutes = 59;

                this->hours--;

            }

        }

    }

}

Time Time::operator=(const Time &t) {

    this->seconds = t.seconds;

    this->minutes = t.minutes;

    this->hours = t.hours;

    this->update\_time();

    return \*this;

}

Time &Time::operator++() {

    this->seconds++;

    this->update\_time();

    return \*this;

}

Time Time::operator++(int) {

    Time t = \*this; // Salvează starea curentă

    this->seconds++;

    this->update\_time();

    return t; // Returnează starea salvată

}

Time &Time::operator--() {

    if (!this->time\_is\_null()) this->seconds--;

    this->update\_time();

    return \*this;

}

Time Time::operator--(int) {

    Time t = \*this; // Salvează starea curentă

    if (!this->time\_is\_null()) this->seconds--;

    this->update\_time();

    return t; // Returnează starea salvată

}

int Time::total\_seconds() {

    return hours \* 3600 + minutes \* 60 + seconds;

}