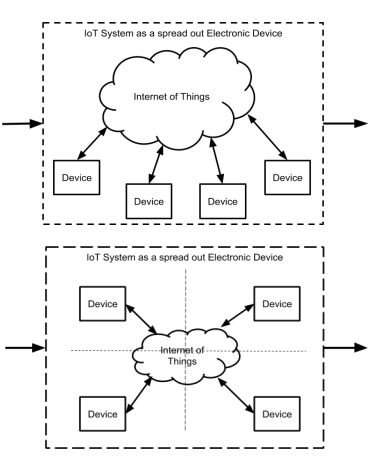
**Varianta 1**

﻿﻿﻿ **1.Structura generică a unui Device loT. Arhitectura generica loT, definiții si descriere generala componente. ﻿﻿﻿**

Dispozitivele IoT sunt componente hardware, cum ar fi senzori, dispozitive de acționare, gadgeturi, aparate sau mașini, care sunt programate pentru anumite aplicații și pot transmite date prin internet sau alte rețele. Acestea pot fi încorporate în alte dispozitive mobile, echipamente industriale, senzori de mediu, dispozitive medicale și multe altele.

**Ex IoT device:** Benzile de fitness,Smart TV,Termostat, Cuptor Smart, Home Automation, Light Bulb and Speaker, Smart Body Analyzer, Finger Reader,

Un dispozitiv IoT tipic are mai multe componente care lucrează împreună pentru a colecta, procesa și transmite date. Aceste componente includ:

**Senzori:** dispozitive care colectează date din mediul fizic (ex. temperatura, umiditatea și presiunea)

**Microcontroler:** este creierul dispozitivului și controlează comportamentul dispozitivului pe baza datelor pe care le primește de la senzori.

**Modul de comunicare:** permite dispozitivului să comunice cu alte dispozitive și sisteme, cum ar fi o rețea locală sau internet.

**Sursa de alimentare:** poate fi o baterie sau o sursă de alimentare externă care oferă dispozitivului puterea necesară pentru a funcționa.

**Interfață cu utilizatorul:** poate include butoane, un afișaj sau alte intrări care permit utilizatorului să interacționeze cu dispozitivul.

În ceea ce privește arhitectura, dispozitivele IoT pot fi împărțite în diferite moduri.

**Arhitectură IoT bazată pe cloud:** în această arhitectură, datele sunt stocate și procesate pe servere cloud. Aceste servere sunt conectate la dispozitivele IoT prin internet și oferă resursele de calcul necesare și capacitatea de stocare pentru dispozitive.

**Arhitectură IoT bazată pe margine:** în această arhitectură, datele sunt procesate pe dispozitivul însuși, în loc să fie trimise în cloud pentru procesare. Acest lucru este util în situațiile în care o conexiune la internet nu este de încredere sau nu este disponibilă.

**Arhitectură hibridă IoT:** aceasta combină atât avantajele arhitecturii cloud, cât și cele bazate pe edge. Datele sunt procesate atât pe dispozitiv, cât și în cloud, în funcție de nevoile specifice ale aplicației.

În general, dispozitivele IoT sunt proiectate pentru a fi de consum redus, costuri reduse și foarte conectate, permițându-le să fie integrate într-o mare varietate de aplicații și sisteme.

O arhitectură IoT generică constă de obicei din mai multe straturi, care lucrează împreună pentru a colecta, transmite și procesa date de pe dispozitivele IoT. Aceste straturi includ:

**Stratul de dispozitiv:** stratul fizic în care sunt amplasate dispozitivele IoT, cum ar fi senzorii și actuatoarele. Aceste dispozitive sunt responsabile pentru colectarea și trimiterea datelor către nivelul următor.

**Stratul de conectivitate:** este responsabil pentru furnizarea unei legături de comunicare între nivelul dispozitivului și cel al rețelei. Se ocupă de protocoalele de comunicare de nivel scăzut, cum ar fi MQTT, CoAP și Zigbee.

**Stratul de rețea:** conectează dispozitivele IoT la internet și oferă servicii de rutare și comunicație pentru dispozitivele IoT. De asemenea, oferă securitate și control al accesului pentru dispozitive.

**Stratul de platformă:** oferă serviciile necesare pentru aplicațiile IoT, cum ar fi stocarea datelor, analizele și managementul. Aici sunt procesate datele, iar acțiunile sunt luate pe baza datelor primite.

**Stratul de aplicație:** oferă interfața cu utilizatorul și aplicațiile care permit utilizatorilor să interacționeze cu dispozitivele IoT și cu datele pe care le colectează.

Este de remarcat faptul că diferite dispozitive și diferite cazuri de utilizare pot avea straturi diferite de arhitectură, dar cele menționate mai sus sunt cele mai comune straturi în arhitectura IoT.

Comunicarea dintre straturi poate fi fie unidirecțională, fie bidirecțională. Comunicarea dintre stratul de dispozitiv și alte straturi superioare este de obicei unidirecțională, în timp ce comunicarea dintre straturile de aplicație și platformă poate fi bidirecțională.

Internetul obiectelor (IoT) este o rețea de dispozitive fizice, vehicule, clădiri și alte elemente încorporate cu electronice, software, senzori și conectivitate care le permite acestor obiecte să colecteze și să schimbe date. Aceste dispozitive se pot conecta la Internet și unele la altele, permițându-le să fie controlate de la distanță și să comunice și să schimbe date.

O arhitectură IoT generică constă de obicei din următoarele componente:

**Senzori și dispozitive:** sunt dispozitivele fizice care colectează date din mediu, cum ar fi senzori de temperatură, camere și dispozitive de acționare.

**Gateway:** un dispozitiv care acționează ca intermediar între senzori și dispozitive și cloud sau alte rețele. Gateway-ul poate efectua sarcini precum preprocesarea datelor, conversia protocolului și securitatea.

**Server cloud sau de rețea:** un server care stochează, gestionează și procesează datele colectate de senzori și dispozitive. De asemenea, poate oferi servicii precum analiza datelor, stocarea și dezvoltarea de aplicații.

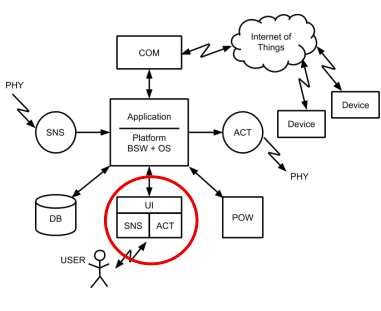
**Stratul de aplicație:** software-ul care rulează pe dispozitive și gateway-uri și oferă interfețe de utilizator și abilitatea de a controla și interacționa cu dispozitivele.

**Rețea:** infrastructura de comunicații care conectează dispozitivele, gateway-urile și serverele. Poate fi o rețea cu fir sau fără fir și poate include o varietate de protocoale și tehnologii diferite, cum ar fi WiFi, Bluetooth, Zigbee sau celular.

**Securitate:** Implementarea unui sistem de securitate este un aspect important al IoT, oferă mecanisme pentru a securiza datele, dispozitivele și sistemele de la actori rău intenționați.

Toate aceste componente lucrează împreună pentru a crea un ecosistem în care dispozitivele pot comunica și partaja date, care pot fi analizate pentru a obține informații și a controla dispozitivele.

**2. Interacțiunea cu utilizatorul – Noțiune generala, clasificare si tipuri de interfețe: binare, tablouri uni si bi-dimensionale, Interfața standard de intrare/ieșire STDIO, Interfețe complexe de interacțiune ﻿﻿﻿**

Interfața cu utilizatorul reprezintă totalitatea componentelor ce facilitează interacțiunea cu utilizatorul sistemului. ca regulă reprezintă o colecție de senzori și dispozitive de acționare specializați.

Interacțiunea cu utilizatorul este un aspect fundamental al sistemelor IoT, deoarece le permite utilizatorilor să controleze și să monitorizeze dispozitivele conectate și să acceseze datele pe care le generează. Această interacțiune poate avea loc printr-o varietate de mijloace diferite, inclusiv:

**Interfețe de utilizator:** acestea pot include interfețe grafice de utilizator (GUI) care sunt afișate pe dispozitive, gateway-uri sau servere, precum și interfețe de linie de comandă (CLI) care pot fi accesate printr-un terminal sau prompt de comandă. Aceste interfețe permit utilizatorilor să controleze și să configureze dispozitivele, precum și să acceseze și să vizualizeze date.

**Asistenți controlați prin voce:** acestea includ dispozitive precum Amazon Echo și Google Home, care utilizează procesarea limbajului natural pentru a recunoaște comenzile vocale ale utilizatorului și pentru a executa sarcini în consecință.

**Aplicații mobile:** Multe sisteme IoT oferă aplicații mobile care pot fi instalate pe smartphone-uri sau tablete. Aceste aplicații oferă o interfață ușor de utilizat pentru a controla și monitoriza dispozitivele și pentru a accesa date.

**Interfețe bazate pe web:** Unele sisteme IoT oferă interfețe web care pot fi accesate printr-un browser web. Aceste interfețe pot fi utilizate pentru a controla dispozitivele, a vizualiza date și a configura setările.

**Notificări prin e-mail și SMS:** Aceste notificări pot fi trimise utilizatorilor pentru a-i informa despre starea unui dispozitiv sau despre un eveniment, cum ar fi o baterie descărcată, o condiție de avertizare sau o alertă.

Toate aceste metode de interacțiune sunt concepute pentru a fi ușor de utilizat și intuitive, permițând utilizatorilor să acceseze și să controleze cu ușurință dispozitivele conectate și datele pe care le generează.

Interacțiunea cu utilizatorul în sistemele IoT poate fi clasificată pe scară largă în mai multe categorii, inclusiv:

**Interacțiune directă:** Acest tip de interacțiune permite utilizatorilor să controleze și să configureze în mod direct dispozitivele folosind interfețe de utilizator, cum ar fi interfețele grafice cu utilizatorul (GUI) sau interfețele de linie de comandă (CLI). Aceasta poate include pornirea sau oprirea unui dispozitiv, ajustarea setărilor sau vizualizarea datelor senzorului.

**Interacțiune indirectă:** acest tip de interacțiune permite utilizatorilor să controleze dispozitivele sau să acceseze date prin intermediari, cum ar fi aplicații mobile, interfețe bazate pe web sau asistenți controlați prin voce. De exemplu, un utilizator ar putea folosi o aplicație mobilă pentru a controla un termostat inteligent sau pentru a vizualiza date de la o cameră conectată.

**Interacțiune automată:** Unele sisteme IoT sunt concepute pentru a interacționa automat cu utilizatorii, fără a fi nevoie de o introducere explicită. Acestea pot include dispozitive care ajustează automat setările în funcție de condițiile de mediu sau de comportamentul utilizatorului. Un exemplu în acest sens este un termostat inteligent care învață preferințele utilizatorilor de temperatură și se reglează automat.

**Interacțiune reactivă:** acest tip de interacțiune are loc atunci când un dispozitiv sau un sistem reacționează la un anumit eveniment sau condiție, cum ar fi declanșarea unei alarme sau citirea unui senzor care depășește un anumit prag. Aceasta poate include notificări trimise utilizatorilor prin e-mail sau SMS sau controlul automat al dispozitivelor pentru a atenua o problemă.

**Interacțiune pasivă:** acest tip de interacțiune este mai informativ și nu necesită o intrare activă din partea utilizatorului, cum ar fi afișarea de informații despre starea dispozitivului, date istorice și performanță pe un ecran sau aplicație mobilă.

În general, clasificarea interacțiunii utilizatorilor în sistemele IoT este destul de largă, iar sistemele diferite pot utiliza o combinație a acestor tipuri diferite de interacțiuni, în funcție de aplicația specifică și de cazul de utilizare.

Categoriile de interfețe :

**• Interfețe binare-** folosește un cod binar, unde intrările și ieșirile sunt reprezentate printr-o secvență de 0 și 1. Acest tip de interfață este adesea folosit în programarea de nivel scăzut sau în comunicarea cu dispozitive care au resurse limitate.

**• Interfețe tablouri uni si bi-dimensionale** - folosește matrice pentru a reprezenta intrările și ieșirile, ar putea fi o matrice unidimensională de valori sau o matrice bidimensională reprezentând o matrice de valori.

**• Interfața standard de intrare/ieșire STDIO** - utilizează fluxurile standard de intrare și ieșire pentru a comunica cu un dispozitiv sau sistem. Este un tip obișnuit de interfață în majoritatea limbajelor de programare, include operații de bază precum citirea de la tastatură și scrierea pe ecran.

**• Interfețe complexe de interacțiune** - concepute pentru interacțiuni avansate, unde utilizatorii pot interacționa cu dispozitivul într-un mod mai natural și mai intuitiv. Exemplele includ recunoașterea gesturilor, recunoașterea facială și recunoașterea vocii. Acestea permit dispozitivelor să înțeleagă intenția utilizatorului și oferă modalități mai sofisticate de interacțiune cu dispozitivul sau sistemul.

În general, alegerea interfeței va depinde de cerințele specifice ale aplicației, de publicul țintă și de resursele disponibile. Unele interfețe pot fi mai potrivite pentru anumite sarcini sau grupuri de utilizatori decât altele și este important să luați în considerare acești factori atunci când alegeți interfața pentru un sistem IoT.

**3. Să se proiecteze o aplicație în baza de MCU pentru detectarea unui cod de la o tastatură 4×4, să verifice codul si sa afișeze mesaj la un LCD. - A se utiliza STDIO pentru scanarea tastaturii și afișare la LCD.**

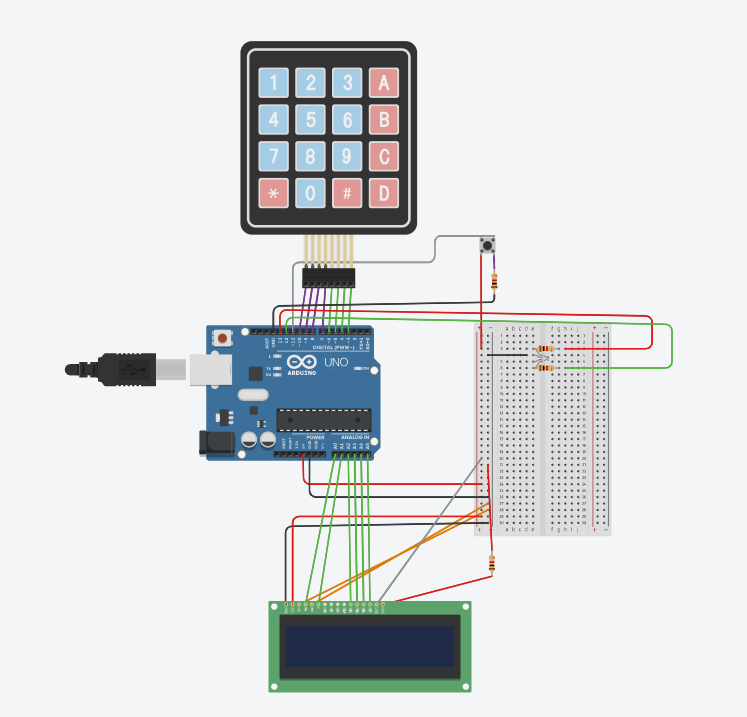
**În explicații să se utilizeze**

**- ﻿﻿Scurta descriere a abordării ﻿﻿**

**- Diagrame conceptuale de interconectare electrica ﻿﻿**

**- Scheme bloc pentru parte funcțională a programului**

**- ﻿﻿Cod sursa cu comentarii relevante în proporții ~1:1**

******

#include <Wire.h>

#include <LiquidCrystal\_I2C.h>

const int ROWS = 4; // numarul de randuri din tastatura

const int COLS = 4; // numarul de coloane din tastatura

// matricea cu pinii de conectare

int keys[ROWS][COLS] = {

{1, 2, 3, 4},

{5, 6, 7, 8},

{9, 10, 11, 12},

{13, 14, 15, 16}

};

// matricea cu valorile tastelor

char keymap[ROWS][COLS] = {

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

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

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

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

};

LiquidCrystal\_I2C lcd(0x27, 2, 1, 0, 4, 5, 6, 7, 3, POSITIVE);

void setup() {

//initializare comunicatie seriala

Serial.begin(9600);

//initializare LCD

lcd.begin(16, 2);

lcd.backlight();

//setare pinilor ca intrari

for (int r = 0; r < ROWS; r++) {

for (int c = 0; c < COLS; c++) {

pinMode(keys[r][c], INPUT);

}

} }

void loop() {

char code[5];

int index = 0;

//scanarea tastaturii

while (index < 4) {

for (int r = 0; r < ROWS; r++) {

for (int c = 0; c < COLS; c++) {

if (digitalRead(keys[r][c]) == LOW) {

code[index] = keymap[r][c];

index++;

delay(250);

}

}

} }

code[4] = '\0';

//verificarea codului

if (strcmp(code, "2222") == 0) {

lcd.clear(); lcd.print("Cod corect!");

}

else {

lcd.clear(); lcd.print("Cod gresit!");

} }

Acest cod este o aplicație Arduino care utilizează o tastatură 4x4(o matrice cu pinii de conectare ai tastaturii și o matrice cu valorile tastelor) și un ecran LCD I2C pentru a permite utilizatorului să introducă un cod. În funcția setup, se inițializează comunicarea serială cu o rată de 9600 baud, se inițializează ecranul LCD, se setează pinii tastaturii ca intrări și se activează iluminarea LCD-ului. În funcția loop, se scanează tastatura, se salvează codul introdus într-un șir de caractere și se verifică dacă acesta corespunde cu un cod prestabilit ("2222"). Dacă codul este corect, se afișează mesajul "Cod corect!" pe ecran, altfel se afișează "Cod gresit!". Biblioteca Wire.h este utilizată pentru comunicarea I2C cu ecranul LCD, iar biblioteca LiquidCrystal\_I2C este utilizată pentru a gestiona ecranul LCD.

**Varianta 2 ﻿﻿﻿**

**1. Structura generica a unui Device loT. Arhitectura generica loT, definiții si descriere generala componente.**

**2. ﻿﻿﻿Senzori - Noțiuni generale, clasificare, achiziții semnal, condiționare semnal, prelucrare semnal achiziționat (filtrare) ﻿﻿﻿**

Senzorii sunt o componentă cheie a sistemelor IoT, deoarece permit dispozitivelor să colecteze date din mediu. Senzorii pot măsura o mare varietate de fenomene fizice, inclusiv temperatura, umiditatea, lumina, sunetul, mișcarea și multe altele.

Există o varietate de modalități în care senzorii pot fi clasificați, în funcție de diferite criterii, cum ar fi:

- După tipul de măsurătoare: Senzorii pot fi clasificați în funcție de tipul de măsurătoare pe care îl efectuează, cum ar fi senzorii de temperatură, senzorii de umiditate, senzorii de presiune și așa mai departe.

- După principiul de funcționare: Senzorii pot fi clasificați în funcție de principiul fizic prin care funcționează, cum ar fi senzorii termici, senzorii piezoelectrici, senzorii optici și așa mai departe.

- După modul de transmisie a semnalului: Senzorii pot fi clasificați în funcție de modul în care transmit semnalul detectat către sistemul de procesare, cum ar fi senzorii analogici, senzorii digitali și senzorii wireless.

- După dimensiunea și forma: Senzorii pot fi clasificați în funcție de dimensiunea și forma lor, cum ar fi senzorii de mare dimensiune, senzorii de dimensiuni reduse, senzorii flexibili și așa mai departe.

- După aplicația sau domeniul de utilizare: Senzorii pot fi clasificați în funcție de aplicația sau domeniul în care sunt utilizați, cum ar fi senzorii pentru automobile, senzorii pentru sănătate, senzorii pentru mediu și așa mai departe.

Există multe tipuri diferite de senzori disponibili, fiecare cu propriile caracteristici și capabilități unice Clasificarea senzorilor se face în funcție de modul în care acestea transformă o cantitate fizică în semnal electric. Există mai multe tipuri de senzori, dar cele mai comune sunt:

**Senzori de temperatură** (ex. termocuplu, termistor) Acești senzori măsoară temperatura și pot fi utilizați într-o gamă largă de aplicații, de la controlul industrial al temperaturii până la monitorizarea temperaturii unui frigider.

**Senzori de umiditate** (ex. higrometru) Acești senzori măsoară conținutul de umiditate din aer și pot fi utilizați pentru a monitoriza umiditatea într-o varietate de medii, cum ar fi sere, depozite și unități de producție.

**Senzori de mișcare** (ex. senzor de mișcare PIR) Acești senzori detectează mișcarea și pot fi utilizați pentru a monitoriza mișcarea persoanelor sau a obiectelor, precum și pentru a detecta mișcarea în sistemele de securitate.

**Senzori de proximitate (**ex. senzor de proximitate inductiv, capacitiv)

**Senzori barometrici**(ex. senzor de presiune absolut, relativ)**:** acești senzori măsoară presiunea aerului din jurul acestuia și pot fi utilizați pentru a detecta schimbările presiunii atmosferice, pentru a determina altitudinea sau pentru a monitoriza tiparele vremii

**Senzori de poziție și de nivel** (ex. senzor LVDT, senzor de nivel cu fir)

**Senzori de iluminat** (ex. senzor fotoresistor) acești senzori măsoară cantitatea de lumină dintr-un mediu dat și pot fi utilizați pentru a controla sistemele de iluminat, pentru a monitoriza consumul de energie sau chiar pentru a detecta mișcarea.

**Senzori de gaze** (ex. senzor de gaz infraroșu)

**Senzori de accelerație** (ex. senzor de accelerație piezoelectric)

**Senzori de sunet:** acești senzori măsoară intensitatea și frecvența sunetului și pot fi utilizați pentru a detecta nivelurile de zgomot, recunoașterea vorbirii și chiar pentru a monitoriza starea de sănătate a echipamentelor.

**Senzori chimici:** Acești senzori pot fi utilizați pentru a detecta prezența sau concentrația anumitor substanțe chimice în mediu. Ele pot fi utilizate în aplicații precum monitorizarea calității aerului, diagnosticarea medicală și controlul proceselor industriale.

Există și senzori mai specializați, cum ar fi senzorii de poziție a roților sau senzorii de gaze toxice.

Acestea sunt doar câteva exemple ale numeroaselor tipuri diferite de senzori care sunt disponibili pentru utilizare în sistemele IoT. Fiecare tip de senzor are propriile caracteristici și capacități unice și poate fi utilizat într-o gamă largă de aplicații.

Achiziția semnalului se referă la procesul prin care un senzor convertește o cantitate fizică în semnalul electric. Aceasta poate fi realizată prin intermediul unui convertor analog-digital (A/D) sau direct prin cablajul senzorului.

Condiționarea semnalului este procesul prin care se îmbunătățește calitatea semnalului achiziționat. Aceasta poate include eliminarea zgomotului sau a interferențelor, amplificarea semnalului, sau adaptarea semnalului la intervalul de măsurare al sistemului de prelucrare.

Prelucrarea semnalului achiziționat se referă la procesul prin care se extrage informația utilă din semnalul achiziționat. Aceasta poate include filtrarea semnalului pentru a elimina zgomotul sau interferențele, prelucrarea statistică a semnalului, sau extragerea caracteristicilor semnalului. Filtrarea poate fi realizată prin utilizarea unui filtru pas-bandă sau pas-trecere pentru a elimina frecvențele neinteresante sau prin utilizarea unui filtru adaptiv pentru a se adapta la condiții variabile.

**Metode de Achiziții semnal**

Există mai multe metode de achiziție a semnalelor, care se bazează pe diverse tehnologii și principii fizice. Alegerea metodei potrivite depinde de caracteristicile semnalului de achiziție și de scopul urmărit.

Iată câteva exemple de metode de achiziție a semnalelor:

-Achiziție analogică: se utilizează o sursă de semnal analogic, cum ar fi un oscilloscop, pentru a măsura valorile semnalului în mod continuu în timpul.

- Achiziție digitală: se utilizează un convertor analog-digital pentru a transforma semnalul analogic într-un semnal digital, care poate fi procesat de un calculator.

- Achiziție prin rețea: se utilizează un sistem de senzori conectați la o rețea pentru a măsura semnalele și a le transmite către un computer pentru procesare.

- Achiziție prin radiofrecvență: se utilizează antene pentru a capta semnalele radiofrecvență și a le converti într-un semnal electric pentru procesare.

- Achiziția semnalelor este importantă pentru că permite măsurarea și înregistrarea diferitelor tipuri de semnale, care pot fi utilizate pentru diverse scopuri, cum ar fi monitorizarea sistemelor, diagnosticarea problemelor și realizarea de studii.

**Condiționarea semnalului** este procesul de prelucrare a unui semnal electronic pentru a îmbunătăți calitatea sa sau pentru a îl adapta pentru utilizare într-un anumit sistem. Există mai multe modalități de a condiționa un semnal, iar alegerea metodei potrivite depinde de scopul dorit și de caracteristicile semnalului de intrare.

Unele exemple de condiționare a semnalului sunt:

- Amplificarea semnalului: se mărește amplitudinea semnalului pentru a îl face mai ușor de detectat sau de procesat.

- Filtrarea semnalului: se elimină anumite frecvențe din spectrul semnalului pentru a îl face mai clar sau pentru a elimina zgomotele de fond.

- Modularea semnalului: se modifică un anumit parametru al semnalului, cum ar fi frecvența sau faza, pentru a îl face mai ușor de transmis sau de receptat.

- Protejarea semnalului: se iau măsuri pentru a proteja semnalul de perturbații sau interferențe externe care ar putea afecta calitatea sa.

- Condiționarea semnalului este importantă pentru că permite ca semnalele să fie prelucrate și utilizate în mod eficient în diferite aplicații, cum ar fi comunicațiile, sistemele de control și senzorii.

**Prelucrarea semnalelor achiziționate** implică procesarea și analizarea semnalelor înregistrate pentru a extrage informații utile sau pentru a le face mai ușor de interpretat. Există mai multe metode de prelucrare a semnalelor, iar alegerea metodei potrivite depinde de caracteristicile semnalului și de scopul urmărit.

Iată câteva exemple de prelucrare a semnalelor:

- Filtrarea semnalelor: se elimină anumite frecvențe din spectrul semnalului pentru a îl face mai clar sau pentru a elimina zgomotele de fond.

- Transformarea Fourier: se utilizează transformarea Fourier pentru a decompora semnalul în componentele sale de frecvență și amplitudine, pentru a îl face mai ușor de analizat.

- Analiza spectrului: se măsoară distribuția amplitudinii semnalului în funcție de frecvență, pentru a obține informații despre componentele de frecvență ale semnalului.

- Analiza timp-frecvență: se analizează modificările semnalului în funcție de timp și frecvență, pentru a obține informații despre variabilitatea semnalului în timp.

Prelucrarea semnalelor este importantă pentru că permite extragerea de informații utile din semnalele achiziționate și pentru a le face mai ușor de interpretat. Acest lucru poate fi util în diverse aplicații, cum ar fi monitorizarea sistemelor, diagnosticarea problemelor și realizarea de studii.

**3.Să se realizeze o aplicație în baza de MCU care va prelua un semnal de la o sursa de semnal analogic, va condiționa semnalul, si va afișa parametrul fizic, si valorile intermediare la un terminal Serial.**

**În explicații să se utilizeze**

**- ﻿﻿Scurta descriere a abordării**

**-﻿﻿Diagrame conceptuale de interconectare electrica**

**- ﻿﻿Scheme bloc pentru parte funcțională a programului**

**- Cod sursa cu comentarii relevante în proporții ~1:1**

Acest cod utilizează pinul A0 pentru a citi semnalul analogic. În funcția loop, se citește valoarea raw a semnalului analogic utilizând funcția analogRead(), apoi se converteste la tensiune prin împărțirea cu raportul de conversie. Apoi se utilizeaza o functie de filtrare pentru a elimina zgomotul din semnalul primit. După aceea, se calculeaza si se afiseaza parametrul fizic si valorile intermediare la un terminal serial prin intermediul functiilor Serial.print() si Serial.println(). La final, se așteaptă o perioadă de 1000ms inainte de a citi din nou semnalul.

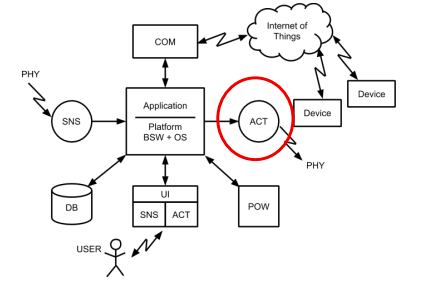
**Varianta 3**

**1. ﻿﻿﻿Structura generica a unui Device loT. Arhitectura generica loT, definitii si descriere generala componente.**

**2. ﻿﻿﻿Actuatori - principii de elaborare, principiu de functionare, modalitate de planificare executiei task-urilor.**

**Actuatorii**reprezinta totalitatea de componente realizate prin inginerie software (SW), inginerie electrică (EE) și inginerie mecanica (ME) care participa la transformarea unui semnal interiorul sistemului într-o acțiune asuprea mediul exterior, reprezentat de o mărime fizică (PHY).

Un actuator este un dispozitiv care transformă energia în mișcare. Există multe tipuri de actuatoare, inclusiv actuatoare electrice, hidraulice și pneumatice, fiecare dintre ele utilizând o formă diferită de energie pentru a produce mișcare. Actuatoarele electrice folosesc un motor electric pentru a produce mișcare, în timp ce actuatoarele hidraulice și pneumatice folosesc presiunea fluidului pentru a produce mișcare. Actuatoarele sunt utilizate într-o gamă largă de aplicații, inclusiv automatizare industrială, robotică și sisteme de control de precizie.

Iată câteva exemple de clasificări ale actuatorilor:

După tipul de energie utilizat:

**Actuatori electrici:** folosesc curent electric pentru a produce semnalul de actiune.

**Actuatori pneumatici:** folosesc aer comprimat pentru a produce semnalul de actiune.

**Actuatori hidraulici:** folosesc ulei sau apă pentru a produce semnalul de actiune.

**Actuatori mecanici:** folosesc o forta mecanica pentru a produce semnalul de actiune.

După modul de montare:

**Actuatori de tip buncăr:** sunt montați în partea inferioară a sistemului și generează o forță verticală.

**Actuatori de tip capotă:** sunt montați în partea superioară a sistemului și generează o forță verticală.

**Actuatori de tip lateral:** sunt montați pe lateralul sistemului și generează o forță orizontală.

După tipul de mișcare:

**Actuatori de translație**: generează o mișcare de translație lineară.

**Actuatori de rotație:** generează o mișcare de rotație.

**Actuatori combinați:** generează o mișcare combinată de translație și rotație.

Clasificarea actuatorilor este importantă pentru că permite alegerea tipului potrivit de actuator pentru o anumită aplicație și pentru a înțelege modul în care acesta funcționează.

**Principii de elaborare**:Actuatoarele pot fi proiectate folosind o varietate de principii, în funcție de tipul de actuator și de aplicația specifică. Câteva principii comune utilizate în proiectarea actuatoarelor includ:

**Mișcare liniară:** multe dispozitive de acționare sunt proiectate pentru a produce mișcare liniară, cum ar fi deplasarea unui piston sau a unui arbore în linie dreaptă. Acest lucru poate fi realizat prin utilizarea unei varietăți de dispozitive mecanice și electromecanice, cum ar fi solenoizi, motoare și șuruburi.

**Mișcare rotativă:** Alte actuatoare sunt proiectate pentru a produce mișcare rotativă, cum ar fi rotirea unei roți sau a unui arbore. Acest lucru poate fi realizat prin utilizarea unor dispozitive precum motoare, angrenaje și came.

**Forță:** actuatoarele pot fi, de asemenea, proiectate pentru a genera o forță sau un cuplu specific. De exemplu, un actuator pneumatic sau hidraulic poate fi proiectat pentru a produce o forță specifică prin controlul presiunii fluidului.

**Controlul poziției și al vitezei:** actuatoarele pot fi proiectate pentru a controla poziția și viteza unei sarcini sau componente. acest lucru se poate realiza prin utilizarea sistemelor de control cu feedback și a senzorilor.

**Eficiență:** actuatoarele pot fi, de asemenea, proiectate pentru a maximiza eficiența, reducând la minimum pierderile de energie și reducând dimensiunea și greutatea actuatorului.

Este important să alegeți tipul potrivit de actuator pentru o anumită aplicație, deoarece aceasta va determina performanța, costul și durata de viață a actuatorului.

**Principiu de functionare:** Actuatoarele convertesc diferite forme de energie în mișcare mecanică pentru a acționa sau deplasa o sarcină sau un sistem de control. Principiul de funcționare al unui actuator depinde de tipul de actuator utilizat. Unele tipuri comune de actuatoare și principiile lor de funcționare includ:

**Actuatoare electrice:** Acestea folosesc un motor electric pentru a converti energia electrică în mișcare mecanică. Ele pot fi împărțite în mai multe tipuri în funcție de tipul de motor utilizat, cum ar fi motoare de curent continuu, motoare pas cu pas și servomotoare. Aceste actuatoare pot fi controlate prin variarea tensiunii sau frecvenței alimentării electrice.

**Actuatoare hidraulice:** Acestea folosesc presiunea fluidului pentru a converti energia unui fluid sub presiune în mișcare mecanică. Ele constau dintr-un cilindru sau un piston care se mișcă ca răspuns la presiunea fluidului, care este furnizat de o pompă. Aceste actuatoare sunt potrivite pentru aplicații cu forță și putere ridicată.

**Actuatoare pneumatice:** Acestea folosesc aer comprimat pentru a transforma energia aerului comprimat în mișcare mecanică. Ele constau dintr-un cilindru sau un piston care se deplasează ca răspuns la presiunea aerului comprimat. Actuatoarele pneumatice sunt utilizate în medii care nu sunt potrivite pentru actuatoarele electrice, precum și pentru aplicații în care este necesară protecția împotriva exploziilor.

**Actuatoare piezoelectrice:** Acestea folosesc efectul piezoelectric, care este capacitatea anumitor materiale de a genera o sarcină electrică ca răspuns la solicitarea mecanică aplicată, de a converti energia electrică în mișcare mecanică. Sunt mici, rapide, precise și potrivite pentru aplicații de micro și nanopoziționare.

**Actuatoare termice:** Acestea folosesc schimbarea formei unui material ca răspuns la o schimbare a temperaturii pentru a converti energia termică în mișcare mecanică.

Fiecare tip de actuator are propriul său set de avantaje și dezavantaje și este important să alegeți actuatorul potrivit pentru o anumită aplicație pe baza unor factori precum forța, viteza, rezoluția, costul și condițiile de mediu.

**Modalitate de planificare executiei task-urilor:** Planificarea și executarea sarcinilor utilizând actuatoare într-un sistem automat implică de obicei mai mulți pași:

**1.Identificați sarcina:** primul pas este să definiți în mod clar sarcina pentru care este folosit actuatorul. Aceasta poate implica identificarea mișcării sau forței specifice pe care trebuie să o producă actuatorul, precum și a oricăror constrângeri sau cerințe pentru sarcină.

**2.Selectați actuatorul:** Odată ce sarcina a fost identificată, următorul pas este selectarea actuatorului potrivit pentru lucrare. Acest lucru va depinde de factori precum tipul de mișcare necesar, sarcina care trebuie mutată, mediul și bugetul.

**3.Proiectați sistemul de comandă:** Odată selectat actuatorul, sistemul de comandă trebuie proiectat. Aceasta implică de obicei selectarea senzorilor adecvați, a algoritmilor de control și a sursei de alimentare pentru actuator.

**4.Implementarea sistemului de control:** Odată ce sistemul de control a fost proiectat, acesta trebuie implementat. Acest lucru poate implica scrierea de software sau controlere de programare pentru a controla actuatorul.

**5.Testare și reglare fină:** După ce sistemul de control a fost implementat, va trebui de obicei testat și reglat fin pentru a se asigura că funcționează corect și că actuatorul îndeplinește sarcina așa cum se dorește.

**6.Monitorizare și întreținere**: Odată ce actuatorul și sistemul de control sunt în funcțiune, este important să le monitorizați performanța pentru a vă asigura că actuatorul funcționează în parametrii de proiectare și că sistemul de control controlează actuatorul așa cum este de așteptat.

Este important să rețineți că acești pași nu sunt strict secvențiali și este adesea necesară repetarea. Mai mult, complexitatea fiecărui pas va varia în funcție de aplicație.

**Metode de Actionare**

Există mai multe metode de actionare a actuatorilor, care se bazează pe diferite principii fizice și tehnologii. Alegerea metodei potrivite depinde de tipul de actuator și de scopul urmărit.

Iată câteva exemple de metode de actionare a actuatorilor:

**Actionarea electrică:** se utilizează un curent electric pentru a actiona un actuator electric. Acest lucru poate fi realizat prin intermediul unui motor electric sau prin intermediul unui solenoiz.

**Actionarea pneumatică:** se utilizează aer comprimat pentru a actiona un actuator pneumatic. Acest lucru poate fi realizat prin intermediul unei valve pneumatic sau prin intermediul unui cilindru pneumatic.

**Actionarea hidraulică:** se utilizează ulei sau apă pentru a actiona un actuator hidraulic. Acest lucru poate fi realizat prin intermediul unei pompe hidraulice sau prin intermediul unui cilindru hidraulic.

**Actionarea mecanică**: se utilizează o forta mecanica pentru a actiona un actuator mecanic. Acest lucru poate fi realizat prin intermediul unei manivele sau prin intermediul unei tije.

Actionarea actuatorilor este importantă pentru că permite controlul sistemelor prin intermediul actuatorilor și pentru a produce diferite mișcări sau acțiuni. Aceasta poate fi utilă în diverse aplicații, cum ar fi sistemele de control automat, sistemele de transport și sistemele industriale.

**3. ﻿﻿﻿Sa se realizeze o aplicatie in baza de MCU care va controlăm motor în curent continuii cu comenzi de setare a puterii motorului intre (-100% .. 100%) adică înainte și înapoi, cu comenzi recepționate de la interfata seriala si raportare si viteza la terminal.**

**În explicații să se utilizeze**

**- ﻿﻿Scurta descriere a abordării ﻿﻿**

**-Diagrame conceptuale de interconectare electrica ﻿﻿**

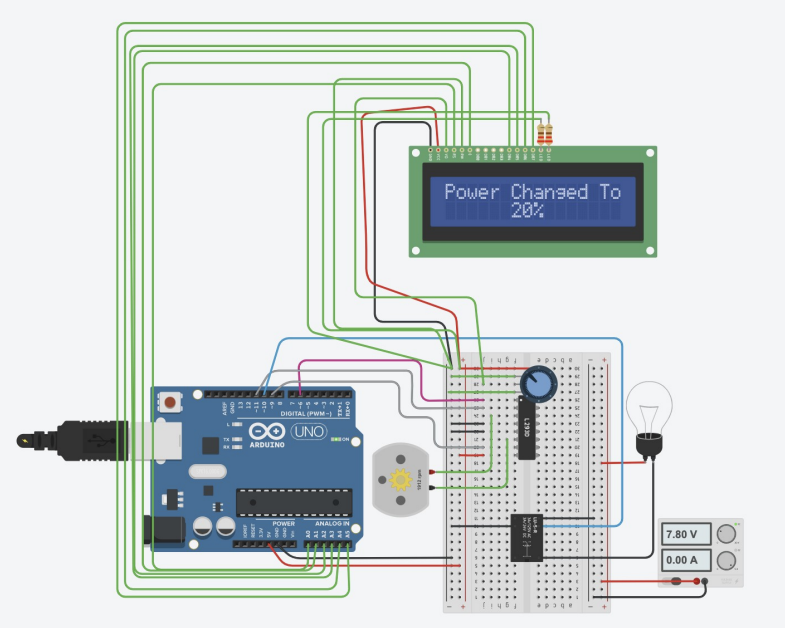
**-Scheme bloc pentru parte funcțională a programului**

**-﻿﻿Cod sursa cu comentarii relevante în proporții ~1:1**

-un motor în curent continuii cu comenzi de setare a puterii motorului între (-100% .. 100%) adica înainte și înapoi, și viteza prin intermediul driverului L298

Driverele de control a periferiilor se vor realiza pe nivele de abstracție.

Materiale utilizate: Visual studio cu Arduino extensie , Kit Arduino, Laptop, Instrument de simulare și validare: Arduino; Bec electric, Releu, Motor în curent continuu, LCD, Rezistori, Potentiometru, Driver pentru motor.

******

Pentru a controla un motor în curent continuu cu comenzi de setare a puterii motorului intre (-100% .. 100%) și raportarea vitezei la terminal, următoarea abordare poate fi utilizată:

Se definește pinul de comandă PWM și pinul de direcție pentru motor în codul C++ Arduino.

În funcția setup, se inițializează comunicarea serială cu o rată de 9600 baud și se setează pinii ca iesiri.

În funcția loop, se verifică dacă există date disponibile la interfata serială. Dacă există, se citesc comenzile de la interfata seriala și se verifică dacă acestea sunt valide(între -100 și 100).

Dacă comanda este valida, se setează direcția motorului (înainte sau înapoi) prin setarea pinului de direcție, iar apoi se setează puterea motorului

#include <LiquidCrystal\_I2C.h>

#include <string.h>

#include <stdio.h>

#include <stdlib.h>

#define PWM\_PIN 9 //pinul de comandă PWM pentru motor

#define DIR\_PIN 8 //pinul de direcție pentru motor

void setup() {

//initializare comunicatie seriala

Serial.begin(9600);

//setare pinii ca iesiri

pinMode(PWM\_PIN, OUTPUT);

pinMode(DIR\_PIN, OUTPUT);

}

void loop() {

if (Serial.available() > 0) {

//citirea comenzii

int power = Serial.parseInt();

//verificarea validitatii comenzii

if (power >= -100 && power <= 100) {

if (power >= 0) {

//setare direcție înainte

digitalWrite(DIR\_PIN, HIGH);

//setare putere motor

analogWrite(PWM\_PIN, power);

Serial.println("Motor înainte la puterea " + String(power) + "%");

}

else {

//setare direcție înapoi

digitalWrite(DIR\_PIN, LOW);

//setare putere motor

analogWrite(PWM\_PIN, abs(power));

Serial.println("Motor înapoi la puterea " + String(abs(power)) + "%");

}

}

else {

Serial.println("Comanda nu este valida. Inserați valoarea intre -100 și 100");

}

}

}

Acest cod utilizează pinul PWM\_PIN pentru a seta puterea motorului și pinul DIR\_PIN pentru a seta direcția motorului. În funcția setup, se inițializează comunicarea serială cu o rată de 9600 baud și se setează pinii ca iesiri. În funcția loop, se verifică dacă există date disponibile la interfata serială. Dacă există, se citesc comenzile de la interfata serială.

**Varianta 4**

**1.﻿﻿﻿Structura generica a unui Device loT. Arhitectura generica loT, definiții si descriere generala componente. ﻿﻿﻿**

**2.Sisteme Secvențiale - principii de elaborare, principiu de funcționare, modalitatea de planificare execuției task-urilor. ﻿﻿﻿**

Sistemele secvențiale sunt sisteme care functioneaza prin executarea unui set de pași sau instrucțiuni într-o ordine predeterminată. Acestea sunt de obicei proiectate pentru a realiza o anumită sarcină sau activitate prin intermediul unei secvențe logice de pași care trebuie urmați într-o anumită ordine.

Sistemele secvențiale pot fi găsite în diverse domenii, cum ar fi automatizarea proceselor industriale, controlul traficului, sistemele de securitate și multe altele. Ele sunt de obicei controlate de către un sistem de operare sau de către un program de control care determină pașii care trebuie urmați și care poate ajusta secvența de pași în funcție de datele de intrare sau de alte condiții.

Sistemele secvenţiale sunt un tip de sistem automat care efectuează o serie de sarcini într-o anumită ordine. Aceste sarcini sunt de obicei efectuate de actuatoare și sunt controlate de un secvențior, care este un program sau controler care controlează ordinea în care sunt efectuate sarcinile.

Un exemplu comun de sistem secvenţial este o linie de asamblare, în care fiecare staţie îndeplineşte o anumită sarcină într-o anumită ordine, iar piesele sunt mutate de la o staţie la alta într-o anumită secvenţă.

Sistemele secvenţiale pot fi împărţite în două categorii:

**Sisteme secvenţiale discrete:** Aceste sisteme efectuează un set specific de sarcini într-o anumită ordine, iar fiecare sarcină este finalizată înainte de a începe următoarea. Un exemplu în acest sens este un robot care efectuează o serie de sarcini de alegere și plasare pe o linie de asamblare.

**Sisteme secvențiale continue:** Aceste sisteme efectuează o serie de sarcini într-o anumită ordine, dar unele sarcini se pot suprapune sau rula concomitent. Un exemplu în acest sens este o fabrică de procesare chimică, în care mai multe procese pot rula în același timp, dar fiecare trebuie să fie finalizat într-o anumită ordine.

În ambele tipuri de sisteme secvențiale, sistemul de control trebuie să se asigure că sarcinile sunt efectuate în ordinea corectă și că sistemul este capabil să gestioneze orice erori sau excepții care pot apărea. Acest lucru se poate face prin utilizarea controlului cu feedback și a senzorilor, precum și a sistemelor de control tolerante la erori.

Sistemele secvenţiale au aplicaţii larg răspândite în industrie, de la producţie la controlul proceselor şi alte domenii, datorită capacităţii de a gestiona operaţiuni şi sarcini complexe şi a fiabilităţii ridicate.

**Principiile proiectării sistemelor secvențiale** presupun crearea unui sistem de control care poate coordona și gestiona ordinea în care sarcinile sunt îndeplinite de către actuatori. Unele principii cheie ale proiectării sistemelor secvențiale includ:

**1.Definirea secvenței:** Primul pas este definirea clară a sarcinilor care trebuie efectuate și a ordinii în care acestea trebuie efectuate. Aceasta este cunoscută sub numele de secvență și este coloana vertebrală a sistemului de control.

**2.Sincronizarea sarcinilor:** Odată ce secvența a fost definită, următorul pas este să vă asigurați că sarcinile sunt sincronizate corect. Acest lucru poate implica utilizarea temporizatoarelor sau contoarelor pentru a se asigura că sarcinile sunt efectuate la momentul potrivit și că durează timpul potrivit.

3.Control de monitorizare și feedback: pentru a se asigura că sistemul funcționează corect, sistemul de control secvenţial trebuie să poată monitoriza performanța actuatoarelor și să ajusteze secvența după cum este necesar. Senzorii și controlul feedback-ului sunt utilizați în acest scop.

4.Gestionarea excepțiilor: Sistemul de control trebuie, de asemenea, să poată gestiona orice excepții sau erori care pot apărea în timpul execuției secvenței. Acest lucru se poate realiza prin utilizarea unor strategii de control tolerante la erori, cum ar fi sistemele de rezervă sau sistemele de siguranță.

5.Programare și implementare: După ce proiectarea sistemului de control este finalizată, următorul pas este programarea și implementarea sistemului. În funcție de complexitatea sistemului, aceasta poate implica scrierea de software sau controlere de programare.

6.testare și validare: înainte ca sistemul să fie implementat în lumea reală, este important să testați sistemul și să-l validăm, pentru a vă asigura că sistemul funcționează conform intenției și că poate face față oricăror situații neașteptate.

7.Întreținere: În cele din urmă, sistemul trebuie întreținut și actualizat în timp, pentru a se asigura că continuă să funcționeze corect și să se adapteze la orice modificări ale mediului sau ale cerințelor aplicației.

Este important să ne amintim că proiectarea sistemelor secvențiale poate fi complexă și poate necesita o combinație de strategii și tehnici de control diferite. Complexitatea și cerințele pot varia, în funcție de aplicația specifică.

Planificarea și executarea sarcinilor în sisteme secvențiale implică de obicei mai mulți pași:

**1.Identificați sarcinile:** primul pas este să definiți clar sarcinile care trebuie efectuate și ordinea în care acestea trebuie efectuate. Aceasta va implica identificarea mișcării sau forței specifice pe care trebuie să le producă actuatoarele, precum și a oricăror constrângeri sau cerințe pentru sarcini.

**2.Creați secvența:** Odată ce sarcinile au fost identificate, următorul pas este crearea secvenței în care vor fi efectuate sarcinile. Aceasta va implica decizia asupra ordinii sarcinilor și a modului în care sarcinile vor fi sincronizate.

**3.Proiectați sistemul de control:** Odată ce secvența a fost creată, sistemul de control trebuie proiectat. Aceasta implică de obicei selectarea senzorilor adecvați, a algoritmilor de control și a sursei de alimentare pentru actuatoare.

**4.Implementarea sistemului de control:** Odată ce sistemul de control a fost proiectat, acesta trebuie implementat. Aceasta poate implica scrierea de software sau programarea controlerelor pentru a controla actuatoarele.

**5.Testare și reglare fină:** După ce sistemul de control a fost implementat, va trebui de obicei testat și reglat fin pentru a se asigura că funcționează corect și că actuatoarele îndeplinesc sarcinile așa cum se dorește.

**6.Monitorizare și întreținere:** Odată ce sistemul este pornit și funcționează, este important să monitorizați performanța acestuia pentru a vă asigura că dispozitivele de acționare funcționează în parametrii de proiectare și că sistemul de control coordonează sarcinile conform așteptărilor. De asemenea, vor fi necesare întreținere și actualizări regulate pentru ca sistemul să funcționeze fără probleme.

**7.Planificați și planificați:** Deoarece aceste sisteme funcționează adesea continuu și concomitent, este important să aveți un plan și un program pentru executarea sarcinilor. Poate implica definirea unui timp specific pentru executarea unei anumite sarcini sau sarcini sau prioritizarea anumitor sarcini față de altele.

Este important să rețineți că acești pași nu sunt strict secvențiali și este adesea necesară repetarea. Mai mult, complexitatea fiecărui pas va varia în funcție de aplicație și de cerințele specifice ale sistemului.

**3.Sa se realizeze o aplicație care va rula 3 task-uri secvențiale in paralel. Task-urile vor avea recurenta si offset diferit, la alegere. Fiecare dintre task-uri reprezinta schimbarea stării unui led in opus**

**În explicații să se utilizeze**

**- ﻿﻿Scurta descriere a abordării ﻿﻿**

**-Diagrame conceptuale de interconectare electrica ﻿﻿**

**-Scheme bloc pentru parte funcțională a programului**

**-﻿﻿Cod sursa cu comentarii relevante în proporții ~1:1**

#include <Arduino\_FreeRTOS.h>

//defineaza pinii ledurilor

#define LED1 2

#define LED2 3

#define LED3 4

void task1(void\* pvParameters) {

while (1) {

//schimba starea ledului 1

digitalWrite(LED1, !digitalRead(LED1));

//asteapta 500ms

vTaskDelay(500 / portTICK\_PERIOD\_MS);

}

}

void task2(void\* pvParameters) {

while (1) {

//schimba starea ledului 2

digitalWrite(LED2, !digitalRead(LED2));

//asteapta 1000ms

vTaskDelay(1000 / portTICK\_PERIOD\_MS);

}

}

void task3(void\* pvParameters) {

while (1) {

//schimba starea ledului 3

digitalWrite(LED3, !digitalRead(LED3));

//asteapta 1500ms

vTaskDelay(1500 / portTICK\_PERIOD\_MS);

}

}

void setup() {

//initializare pinii ledurilor ca iesiri

pinMode(LED1, OUTPUT);

pinMode(LED2, OUTPUT);

pinMode(LED3, OUTPUT);

//creaza taskurile

xTaskCreate(task1, "Task1", 128, NULL, 1, NULL);

xTaskCreate(task2, "Task2", 128, NULL, 1, NULL);

xTaskCreate(task3, "Task3", 128, NULL, 1, NULL);

//porneste schedulerul

vTaskStartScheduler();

}

void loop() {

//taskurile sunt executate de scheduler

}

Acest cod utilizează biblioteca Arduino\_FreeRTOS pentru a gestiona multiple task-uri în paralel pe un sistem Arduino. În acest caz, sunt definite 3 task-uri: task1, task2, task3. Fiecare task are sarcina de a schimba starea unui LED conectat la un pin specificat prin intermediul functiilor digitalWrite() si digitalRead(). Fiecare task are o recurență diferită, specificată prin intermediul vTaskDelay(). În funcția setup, sunt definite pinii ledurilor ca iesiri, se creează task-urile și se pornește schedulerul prin intermediul vTaskStartScheduler(). Funcția loop nu este utilizată în acest caz, deoarece task-urile sunt gestionate de scheduler.

În funcția de setup sunt definite pinii ledurilor, sunt initialize ca iesiri, se creaza taskurile cu diferenta de recurenta si offset, apoi se porneste schedulerul. În funcția loop nu se face nimic, deoarece task-urile sunt gestionate de scheduler. Fiecare task este o buclă infinită care schimbă starea ledului și așteaptă un interval de timp specific (500ms, 1000ms, 1500ms) înainte de a schimba din nou starea ledului. Prin aceasta metoda se pot executa mai multe task-uri în același timp, fiecare cu recurență.

**Varianta 5 ﻿﻿﻿**

**1.Structura generica a unui Device loT. Arhitectura generica loT, definiții si descriere generala componente.**

**2. ﻿﻿﻿Comunicații in sisteme electronice - interfete, protocoale Fizice, protocoale logice**

Comunicarea reprezintă modalitatea de schimb de informație intre interlocutori ."Comunicații in sisteme electronice" se referă la modul în care diferite componente ale unui sistem electronic se comunică între ele pentru a realiza o anumită funcție sau a atinge un anumit obiectiv. Aceasta poate include comunicarea prin intermediul cablurilor sau a undelor radio, precum și codarea și decodarea semnalelor pentru a asigura o transmisie eficientă și sigură a informației. Există mai multe tehnologii și protocoale utilizate în comunicațiile din sistemele electronice, cum ar fi comunicarea serială, paralelă, sau prin intermediul rețelelor, precum Ethernet sau WiFi.

Interfețele de comunicații din sistemele electronice se referă la modul în care diferite componente ale sistemului se conectează fizic pentru a se comunica. De exemplu, aceasta poate include conexiuni fizice precum cabluri sau porturi USB sau wireless cum ar fi Bluetooth sau WiFi.

Protocoalele fizice se referă la regulile și standardele care stabilesc cum informația este transmisă pe aceste interfețe fizice. De exemplu, protocoalele fizice pot specifica caracteristici precum viteza de transmisie, lungimea cablului sau puterea semnalului. Exemple de protocoale fizice: RS-232, RS-485, USB, Ethernet, HDMI

Protocoalele logice, pe de altă parte, se referă la regulile și standardele care controlează cum informația este codată și transmisă între componentele sistemului. Acestea pot include reguli precum ordinea în care bitii sunt transmisi sau metode de detectare și corectare a erorilor.

Exemple de protocoale logice : TCP/IP, HTTP, SMTP

Împreună, interfața, protocoalele fizice și logice formează un mecanism de comunicare complet, care permite diferitelor componente ale unui sistem electronic să comunice eficient și sigur, permitand realizarea obiectivelor sau funcțiilor sistemului respectiv.

**3.﻿﻿﻿Sa se realizeze o aplicație care va implementa un protocol simplu de comunicare prin interfata seriala după cum urmează ﻿﻿indicator de start pachet ﻿﻿indicator de sfârșit ﻿﻿contorizare pachete ﻿﻿date pachet - Payload ﻿﻿suma de control - suma tuturor valorilor numerice din pachet**

**În explicații să se utilizeze**

**- ﻿﻿Scurta descriere a abordării ﻿﻿**

**-Diagrame conceptuale de interconectare electrica ﻿﻿**

**-Scheme bloc pentru parte funcțională a programului**

**-﻿﻿Cod sursa cu comentarii relevante în proporții ~1:1**

Implementeaza un protocol simplu de comunicare prin interfata seriala, care include indicatorii de start si de sfarsit pachet, contorizarea pachetelor, datele pachetului (payload) si suma de control (care este suma tuturor valorilor numerice din pachet):

#include <Arduino.h>

#include <string.h>

#include <stdio.h>

#include <stdlib.h>

int packetCount = 0; // contorizarea pachetelor

char startPacketIndicator = 'S'; // indicatorul de start pachet

char endPacketIndicator = 'E'; // indicatorul de sfarsit pachet

String packetData; // datele pachetului

int packetChecksum = 0; // suma de control

void setup() {

//initializare comunicatie seriala

Serial.begin(9600);

}

void loop() {

if (Serial.available()) {

// primirea datelor pachetului

packetData = Serial.readString();

// calcularea sumei de control

packetChecksum = 0;

for (int i = 0; i < packetData.length(); i++) {

if (isdigit(packetData.charAt(i))) {

packetChecksum += packetData.charAt(i) - '0';

}

}

// crearea si trimiterea pachetului

String packet = String(startPacketIndicator) + String(packetCount) + packetData + String(packetChecksum) + String(endPacketIndicator);

Serial.println("Trimitere pachet: " + packet);

packetCount++;

}

}

Acest cod utilizează un buclu infinit în care se verifică dacă sunt date disponibile pe interfața serială. Dacă sunt, acestea sunt citite și stocate în variabila "packetData", apoi se calculează suma de control prin parcurgerea fiecărui caracter din acest pachet și adăugarea valorii numerice a caracterului la suma de control, dacă caracterul este un număr. După aceea, se creează pachetul final prin concatenarea indicatorului de start pachet, contorizarea pachetelor, datelor pachetului, suma de control și indicatorul de sfârșit pachet. Pachetul final este apoi trimis prin intermediul comunicatiei seriale.

**Varianta 6 ﻿﻿﻿**

**1.Structura generica a unui Device loT. Arhitectura generica loT, definiții si descriere generala componente.**

**﻿﻿﻿2. Control cu Automate finite - principiul de realizare si funcționare**

O mașină cu stări finite (FSM) este un model matematic de calcul care descrie comportamentul unui sistem prin definirea unui set de stări și tranzițiile dintre acele stări. FSM-urile sunt adesea folosite pentru a proiecta și implementa sisteme de control, în special în domeniul sistemelor încorporate, unde sunt utilizate pentru a controla comportamentul dispozitivelor și interfețelor hardware.

Funcționarea de bază a unei mașini cu stări finite este aceea că începe într-o stare inițială și apoi trece la alte stări pe baza intrărilor sau evenimentelor. Fiecare stare reprezintă o condiție sau un mod specific de funcționare a sistemului, iar tranzițiile între stări sunt definite de un set de reguli sau condiții.

În sistemele de control, FSM-urile sunt adesea folosite pentru a controla comportamentul dispozitivelor și interfețelor hardware. De exemplu, un sistem de control al semaforului poate avea trei stări, roșu, galben și verde, iar tranzițiile între aceste stări sunt determinate de durata semnalului de semafor. Sau o mașină de spălat care are diferite stări pentru umplere, spălare, clătire, centrifugare și finisare.

Un FSM poate fi implementat în diferite moduri, de la o simplă logică codificată cu condiții dacă altfel, până la soluții hardware sau software mai complexe. Există diferite instrumente software care pot fi utilizate pentru a proiecta, implementa și simula FSM-uri. Se pot folosi diagrame de stare, tabele de stare, ecuații de stare sau chiar instrumente de glisare și plasare bazate pe grafice pentru a proiecta FSM.

În rezumat, un FSM este un model matematic de calcul care descrie comportamentul unui sistem, prin definirea unui set de stări și tranzițiile dintre acele stări. FSM-urile sunt adesea folosite în sistemele de control, unde sunt folosite pentru a controla comportamentul dispozitivelor și interfețelor hardware și pot fi imp

Controlul cu automate finite se referă la utilizarea de automate finite (AF) pentru a controla un proces sau sistem. Un automat finit este o structură formală care poate fi utilizată pentru a modela și a descrie comportamentul unui sistem. Un automat finit este caracterizat prin faptul că are o serie finită de stări și o serie finită de tranziții între aceste stări.

În controlul cu automate finite, AF este utilizat pentru a modela comportamentul sistemului controlat, iar apoi este utilizat pentru a genera un control automat al sistemului. Automatul finit poate fi utilizat pentru a determina cum trebuie să răspundă sistemul la diferite intrări sau perturbații și poate fi utilizat pentru a implementa un control automat al sistemului.

Acestea sunt utilizate in domenii precum: fabricarea, traficul, robotica, controlul proceselor industriale, controlul traficului aerian, controlul sistemelor automate de transport si altele. Automatele finite joacă un rol important in controlul sistemelor mecanice, electrice, hidraulice și pneumatic sau orice sistem care poate fi descris prin un set finit de stări și tranziții.

Principiul de realizare a controlului cu automate finite se bazează pe utilizarea unui automat finit pentru a modela comportamentul unui sistem și pentru a genera un control automat al acestuia.

Pentru a realiza acest lucru, se urmează urmatorii pasi:

1.Modelarea sistemului: Se construieste un automat finit care descrie comportamentul sistemului controlat, inclusiv toate stările posibile și tranzițiile dintre acestea.

2.Proiectarea controlului: Se proiectează un control automat utilizând automatul finit construit anterior, care va determina cum trebuie să răspundă sistemul la diferite intrări sau perturbații.

3.Implementarea controlului: Se implementează controlul automat proiectat pe sistemul real, utilizând, de exemplu, un sistem de control programabil sau un controler embedded.

4.Monitorizarea si evaluarea sistemului: Se monitorizează sistemul pentru a verifica dacă funcționează corespunzător, si se fac eventualele modificari necesare pentru a se asigura ca sistemul răspunde corect și în mod adecvat la diferite intrări sau perturbații.

În funcționare, automatul finit este utilizat pentru a determina starea curentă a sistemului și pentru a determina acțiunile necesare pentru a răspunde la diferite intrări sau perturbații. Controlul automat generat de automatul finit este apoi implementat pe sistemul real și este utilizat pentru a controla comportamentul acestuia.

Există mai multe modalități de implementare a unei mașini cu stări finite (FSM), în funcție de cerințele specifice ale sistemului de control și de resursele disponibile. Unele metode comune includ:

**Logica hard-coded:** Una dintre cele mai simple moduri de a implementa un FSM este utilizarea logicii hard-coded, cum ar fi instrucțiuni if-else sau instrucțiuni switch-case, pentru a defini în mod explicit tranzițiile de stare și acțiunile din cod. Această metodă este ușor de înțeles și implementat, dar poate deveni greoaie pe măsură ce crește numărul de stări și tranziții.

**Tabele de căutare:** O altă metodă comună este utilizarea tabelelor de căutare pentru a defini tranzițiile și acțiunile de stare. Starea curentă și intrarea sunt folosite pentru a căuta următoarea stare și ieșire în tabel. Această metodă este mai flexibilă și mai scalabilă decât logica hard-coded, dar poate fi mai complex de implementat și întreținut.

**Biblioteci de mașini de stat:** Unele limbaje de programare oferă biblioteci sau cadre care simplifică implementarea FSM-urilor. Aceste biblioteci oferă clase sau funcții FSM prefabricate care pot fi utilizate pentru a defini stări, tranziții și acțiuni și pot gestiona automat tranzițiile de stare și alte detalii.

**Instrumente grafice**: Există câteva instrumente grafice care pot fi utilizate pentru a proiecta, implementa și testa un FSM. Aceste instrumente permit crearea de stări și tranziții prin drag-and-drop, generarea automată a codului și capabilitățile de depanare încorporate. Această metodă poate economisi timp și poate facilita crearea unui FSM pentru non-programatori.

**Bazat pe hardware**: există circuite FSM și circuite integrate specializate care pot fi utilizate pentru implementarea FSM-urilor în hardware. Această metodă poate oferi tranziții de stare mai rapide și mai precise, dar poate fi și mai costisitoare și dificil de implementat.

Metoda de implementare aleasă va depinde de cerințele specifice ale sistemului de control, de resursele disponibile și de nivelul de calificare al membrilor echipei. Logica codificată este potrivită pentru sistemele mici, iar instrumentele grafice sunt bune pentru un membru începător al echipei. Pentru sisteme mai complexe, bibliotecile sau soluțiile bazate pe hardware pot fi mai potrivite.

**﻿﻿﻿3. Să se realizeze o aplicație de control a unui led cu Automat finit - schimbare în opus la detectarea butonului apăsat butonul**

**În explicații să se utilizeze**

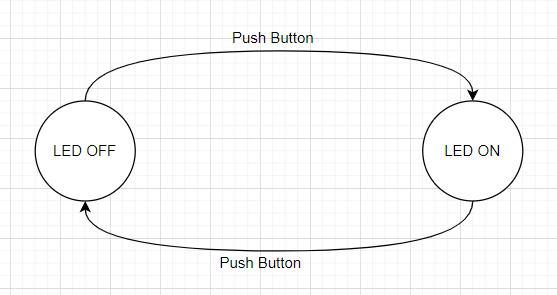
**- ﻿﻿Scurta descriere a abordării ﻿﻿**

**-Diagrame conceptuale de interconectare electrica ﻿﻿**

**-Scheme bloc pentru parte funcțională a programului**

**-﻿﻿Cod sursa cu comentarii relevante în proporții ~1:1**

Diagrama automatului finit pentru a reprezenta modificarea stării butonului LED, și care va fi declanșatorul acestei schimbări. Am definite două stări LED OFF și LED ON.



Reprezentarea schematică a unui automat finit pentru aplicația Button-LED

În cazul de față când este apăsat un buton, LED-ul luminează, sau este stins, în funcție de starea în care este LED-ul în acel moment.

#include <Arduino.h>

#include <string.h>

#include <stdio.h>

#include <stdlib.h>

#define LED\_PIN 13

#define BUTTON\_PIN 2

#define STATE\_LED\_OFF 0

#define STATE\_LED\_ON 1

#define LED\_DELAY 15

#define DELAY\_TIME 10

#define LED\_STATE 2

struct State {

  unsigned long Out;

  unsigned long Time;

  unsigned long Next[LED\_STATE];

  };

  State FSM[LED\_STATE]= {

    { STATE\_LED\_OFF, LED\_DELAY, { STATE\_LED\_OFF, STATE\_LED\_ON }},

    { STATE\_LED\_ON, LED\_DELAY, { STATE\_LED\_ON, STATE\_LED\_OFF }}

    };

int State\_FSM = STATE\_LED\_OFF;

void setup() {

  pinMode(BUTTON\_PIN, INPUT);

  pinMode(LED\_PIN, OUTPUT);

  State\_FSM = STATE\_LED\_OFF;

}

  void loop() {

  int output = FSM[State\_FSM].Out;

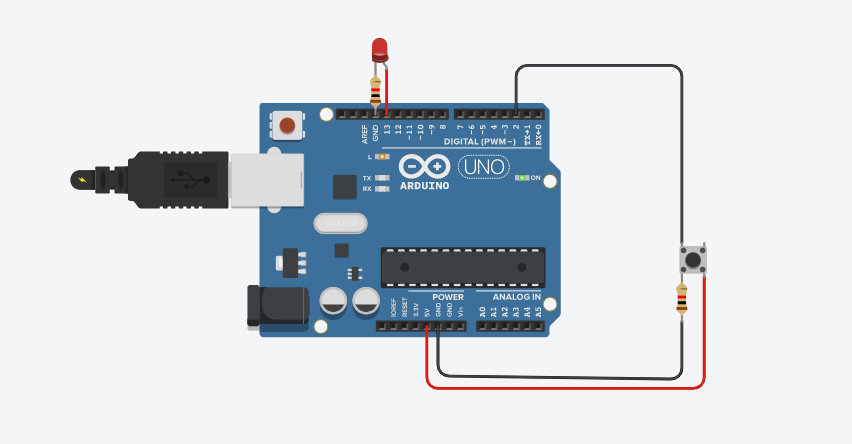
  digitalWrite(LED\_PIN, output);

  delay(FSM[State\_FSM].Time \* DELAY\_TIME);

  int input = digitalRead(BUTTON\_PIN);

  State\_FSM = FSM[State\_FSM].Next[input];

}



In ceea ce priveste diagrama conceptuala de interconectare electrica, aceasta ar arata astfel:

Ledul este conectat la un rezistor si la un pin digital al placii Arduino

Butonul este conectat intre un pin digital al placii Arduino si GND

Rezistorul este utilizat pentru a limita curentul care trece prin led si pentru a preveni suprasarcina pinului digital.

**Varianta 7 ﻿﻿﻿**

**1.Structura generica a unui Device loT. Arhitectura generica loT, definitii si descriere generala componente.**

**﻿﻿﻿2.Control cu Functional - Bucla deschisa, on/off, on/off cu histerezis, PID**

Controlul funcțional se referă la utilizarea unei metode de control bazată pe funcții matematice pentru a controla un sistem. Aceasta se deosebeste de alte metode de control, cum ar fi controlul cu automate finite, care se bazează pe descrierea comportamentului sistemului prin intermediul unui automat finit.

In controlul funcțional se folosește o descriere matematica a sistemului, cum ar fi ecuațiile dinamice sau modelele lui transfer, și apoi se utilizează aceasta descriere pentru a proiecta un controler care poate face sistemul sa urmeze o traiectorie dorită sau sa atinga un anumit obiectiv.

Una din metoda de control funcțional cunoscuta este controlul prin retroalimentare, unde un controler este proiectat pentru a compara un semnal de ieșire dorit cu un semnal de ieșire real și pentru a genera un semnal de comandă care să corecteze orice abatere.

Aceasta metoda este utilizată în domenii precum controlul sistemelor aeronautice, navale, spatii si robotică, controlul proceselor industriale, controlul sistemelor automate de transport si altele, deoarece permite proiectarea unui control mai precis si eficient, si permite adaptarea la conditii variabile ale sistemului.

Functional control poate include tehnici precum controlul prin retroalimentare, controlul prin predictie, sau controlul adaptiv.

Există mai multe tipuri de control funcțional, dar cele mai comune sunt controlul buclă deschisă, controlul on-off, controlul on-off cu histerezis și controlul PID.

**Controlul buclă deschisă** metodă simplă de control în care un semnal de comandă este aplicat direct la un sistem fără a măsura sau a utiliza un semnal de ieșire pentru a regla controlul. Semnalul de comandă este aplicat direct la sistem, fără a utiliza informații despre starea actuală sau comportamentul sistemului. Sistemul răspunde direct la semnalul de comandă, fără a utiliza retroalimentare.

Acest tip de control este utilizat în special în aplicații în care sistemul are un comportament cunoscut și stabil, sau în cazul în care nu este posibil sau necesar să se obțină informații despre starea sau comportamentul sistemului.

De exemplu, în cazul unui sistem de iluminat, un controler poate furniza o comandă pentru a seta o anumită intensitate a luminii, fără a măsura sau a utiliza informații despre intensitatea curentă a luminii.

Controlul buclă deschisă este simplu si economic, dar acesta poate fi mai puțin stabil si poate conduce la erori mari daca sistemul are un comportament neprevizibil sau variabil.

**Controlul on-off** este o metodă de control în care care un dispozitiv sau un proces este fie complet pornit, fie complet oprit în funcție de valoarea unui semnal de intrare. De exemplu, un controler poate comanda un sistem să se oprească sau să pornească dacă o temperatură depășește sau cade sub o anumită valoare.

Acest tip de control este adesea folosit în sistemele simple în care există o alegere binară, cum ar fi un întrerupător care aprinde sau stinge o lumină sau un termostat care pornește sau oprește un sistem de încălzire în funcție de temperatură. Sistemele de control on-off pot fi utilizate și în sisteme mai complexe, cum ar fi procesele industriale sau mașinile, unde scopul este menținerea unei anumite condiții de funcționare, cum ar fi o anumită temperatură sau debit.

**Controlul on-off cu histerezis** se folosește, similar cu cel on-off, este un tip de sistem de control în care un dispozitiv sau un proces este fie complet pornit, fie complet oprit, dar cu adaugarea unui banda de toleranta pentru evitarea comutarii frecvente si stabilizarea sistemului în jurul valorii de referință, în cadrul căreia procesul nu se va porni sau nu se va opri. De exemplu, dacă valoarea de referință pentru un sistem de încălzire este de 38 de grade, sistemul se va porni când temperatura scade la 37 de grade, dar nu se va opri până când temperatura crește la 39 de grade. Intervalul cuprins între 37 și 99 de grade este banda de histerezis, în cadrul căreia sistemul de încălzire nu se va porni sau nu se va opri.

Controlul histerezisului poate fi util în prevenirea ciclismului rapid, care poate duce la o uzură crescută a echipamentelor, la scăderea eficienței și la alte probleme. Este util în special în aplicațiile în care schimbările rapide ale variabilei procesului pot cauza instabilitate a sistemului.

Acest control poate fi implementat prin hardware (mecanic sau electronic) sau software. Bucla de histerezis poate fi implementată programatic prin adăugarea de condiții pentru a detecta dacă procesul este peste sau sub valoarea de referință și apoi sistemul va porni sau opri procesul în consecință.

**Controlul PID (proporțional-integral-derivativ)** este un tip de sistem de control care este utilizat pe scară largă în sistemele de control industrial și în alte aplicații care necesită control modulat continuu. Se numește controler PID deoarece folosește o combinație de trei termeni de control, termenii proporțional, integral și derivat, pentru a controla variabila procesului. O metodă de control avansată care utilizează trei componente pentru a regla un sistem: proporțional, integral și derivativ. Componenta proporțională corectează eroarea în funcție de mărimea acesteia, componenta integrală corectează eroarea pe parcursul timpului și componenta derivativă previzionează eroarea în funcție de viteza de schimbare a acesteia. Acest control este amplu utilizat in industria si in alte domenii unde exista exigențe ridicate pentru precizie și stabilitate.

Termenul proporțional este folosit pentru a aduce variabila de proces la valoarea de referință cât mai repede posibil. Termenul integral este folosit pentru a elimina orice eroare de stare staționară care poate fi prezentă, iar termenul derivat este folosit pentru a anticipa comportamentul viitor al variabilei de proces și pentru a preveni depășirile și oscilațiile. Controlerul poate ajusta ieșirea în funcție de eroarea curentă, eroarea acumulată și rata de modificare a erorii.

Controlerele PID sunt utilizate pe scară largă în sistemele de control industrial, deoarece sunt simplu de implementat, relativ ușor de reglat și extrem de eficiente în controlul proceselor. Controlerul poate fi reglat prin ajustarea constantelor de câștig pentru cei trei termeni de control, cunoscuți ca Kp, Ki și, respectiv, Kd. Aceste constante sunt înmulțite cu eroarea, eroarea acumulată și rata de modificare a erorii pentru a calcula ieșirea de control. Reglarea corectă a acestor constante se poate face prin încercare și eroare sau prin utilizarea unor metode de reglare mai avansate, cum ar fi Ziegler-Nichols, Cohen-Coon sau altele.

Controlerele PID sunt utilizate în principal în aplicații care necesită un control precis al variabilei procesului, cum ar fi controlul temperaturii, controlul debitului și controlul vitezei. Ele pot fi implementate folosind componente electronice sau mecanice și, de asemenea, prin software, cu arhitectură diferită pentru implementare digitală.

**3.﻿﻿﻿Să se realizeze o aplicatie de control a a temperaturii dupã principiul On/Off cu histerezis, unde se considera ca senzorul este realizat prin Interfata GetTemperature() și SetActuatorState(State); si nu necesita implementare.**

**În explicații să se utilizeze**

**- ﻿﻿Scurta descriere a abordării ﻿﻿**

**-Diagrame conceptuale de interconectare electrica ﻿﻿**

**-Scheme bloc pentru parte funcțională a programului**

**-﻿﻿Cod sursa cu comentarii relevante în proporții ~1:1**

#include <Arduino.h>

#include <string.h>

#include <stdio.h>

#include <stdlib.h>

const int TEMPERATURE\_SETPOINT = 22; // setpoint-ul de temperatura dorit

const int TEMPERATURE\_HYSTERESIS = 2; // histerezisul de temperatura

bool actuatorState = false; // starea inițială a actuatorului

void setup() {

//initializare comunicatie seriala

Serial.begin(9600);

}

void loop() {

float temperature = GetTemperature(); // citirea temperaturii

if (actuatorState == false && temperature > TEMPERATURE\_SETPOINT + TEMPERATURE\_HYSTERESIS) {

// daca actuatorul este oprit si temperatura este mai mare decat setpoint + histerezis, se porneste

actuatorState = true;

SetActuatorState(actuatorState);

}

else if (actuatorState == true && temperature < TEMPERATURE\_SETPOINT - TEMPERATURE\_HYSTERESIS) {

// daca actuatorul este pornit si temperatura este mai mica decat setpoint - histerezis, se opreste

actuatorState = false;

SetActuatorState(actuatorState);

}

Serial.print("Temperatura: ");

Serial.print(temperature);

Serial.print(" Actuator status: ");

Serial.println(actuatorState ? "On" : "Off");

delay(1000);

}

Această aplicație folosește principiul On/Off cu histerezis pentru a controla un actuator în funcție de temperatura citită de la un senzor. Funcția GetTemperature() este utilizată pentru a citi temperatura, iar funcția SetActuatorState() este utilizată pentru a seta starea actuatorului. Se ia un setpoint de temperatura dorit și un histerezis de temperatura, care se folosesc pentru a decide când actuatorul trebuie să fie pornit sau oprit. Valorile citite de temperatura si starea actuatorului sunt afisate la terminal serial.

Controlul ON-OFF este cea mai simplă formă de dispozitiv de control al temperaturii. Ieșirea de la dispozitiv este fie pornită, fie oprită, fără stare de mijloc. Un regulator ON-OFF va comuta ieșirea numai atunci când temperatura depășește punctul de referință.

Pentru controlul încălzirii, ieșirea este pornită când temperatura este sub punctul de referință și oprit peste punctul de referință. Deoarece temperatura depășește punctul de referință pentru a schimba starea de ieșire, temperatura procesului va fi ciclată în mod continuu, mergând de la punctul de referință sub punctul de referință la sus și înapoi dedesubt.

În cazurile în care acest ciclu are loc rapid și pentru a preveni deteriorarea releelor ​​și supapelor, se adaugă un diferențial pornit-oprit, sau „histerezis” în cazul nostru, la operațiunile controlerului. Această diferență necesită ca temperatura să depășească valoarea de referință cu o anumită valoare înainte ca ieșirea să se oprească sau să se pornească din nou.