

MINISTERUL EDUCAȚIEI ȘI CERCETĂRII AL REPUBLICII MOLDOVA
Universitatea Tehnică a Moldovei
Facultatea Calculatoare, Informatică și Microelectronică
Departamentul Ingineria Software și Automatică
Programul de studii: Tehnologia informației



RAPORT

Disciplina „IoT – Internetul Lucrurilor”
Tema: Sisteme de Operare - Secvențiale

Student(ă): _____ **Vlașițchi Ștefan , TI-212**

Coordonator universitate: _____ **Lupan Cristian, asist.univ.**

Chișinău, 2024

Scopul: Realizarea unei aplicații pentru MCU care va rula minim 3 task-uri - *Secvential*

Obiective:

Aplicația va rula minim 3 task-uri printre care

1. Button Led - Schimbare stare LED la detecția unei apăsări pe buton.
2. un al doilea Led Intermitent în faza în care LED-ul de la primul Task e stins
3. Incrementare/decrementare valoare a unei variabile la apăsarea a doua butoane care va reprezenta numărul de recurențe/timp în care ledul de la al doilea task se va afla într-o stare
4. Task-ul de Idle se va utiliza pentru afișarea stărilor din program, cum ar fi, afișare stare LED, și afișare mesaj la detecția apăsării butoanelor, o implementare fiind ca la apăsarea butonului sa se seteze o variabila, iar la afișare mesaj - resetare, implementând mecanismul provider/consumer.

Introducere

Internetul Lucrurilor (IoT) reprezintă interconectarea dispozitivelor fizice prin intermediul internetului, facilitând astfel schimbul de date și interacțiunea automatizată între ele. În cadrul acestui laborator, utilizarea microcontrolerelor (MCU) pentru a implementa interacțiunea cu utilizatorul printr-o interfață serială este esențială pentru prototiparea și testarea aplicațiilor IoT. O problemă actuală abordată în acest context este controlul secvențial al echipamentelor periferice, precum LED-uri, prin comenzi trimise prin terminal, utilizând butoane pentru a permite dezvoltatorilor să gestioneze eficient aceste dispozitive. Printre referințele studiate se numără documentația STUDIO pentru microcontrolere, precum și tutorialele de configurare a mediului de dezvoltare (IDE) pentru dezvoltarea și simularea aplicațiilor.

Materiale utilizate:

- Microcontroler cu suport pentru interfața serială (Arduino).
- LED-uri, butoane și rezistori.
- IDE cu suport pentru Arduino și multiple fișiere, precum Arduino IDE.
- Simulator Wokwi pentru validarea proiectului.

Metodologie:

1. Implementarea comunicării între task-uri ca provider-consumer, astfel:
 - Task-ul care generează date, numit provider, stochează rezultatele într-o variabilă globală/semnal.

- Task-ul care utilizează aceste date, numit consumer, citește această variabilă/semnal. De exemplu, task-ul de interfață utilizator (UI), fie el LCD sau serial, preia informația din variabile-semnale globale și o raportează utilizatorului.
2. Urmarea principiilor prezentate în cursul de Sisteme Secvențiale:
 - Stabilirea unei frecvențe rezonabile pentru a diminua încărcarea procesorului.
 - Stabilirea unui ofset pentru a activa în ordinea cuvenită task-urile.
 3. Task-ul de raportare pentru secvențial, care utilizează STDIO cu printf() către LCD, va fi rulat într-o buclă infinită/IDLE. Aceasta este bazată pe un spin lock și ar putea bloca întreruperile, deci este esențială implementarea secvențială clasică — utilizând printf și delay în bucla principală.

Rezultate

Pentru acest laborator am creat o aplicație pentru MCU care rulează 4 taskuri consecutive secvențiale (Vezi Fig 1).

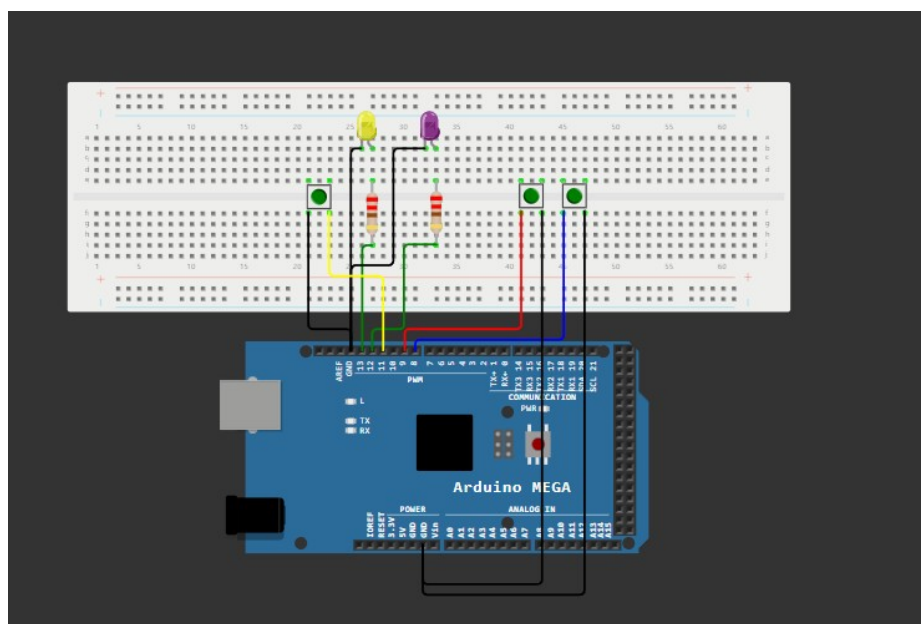


Figura 1 – Schema circuitului Arduino

Fragment cod sursa **main.ino**

```
void loop() {
  currentTime = millis();
  task3(); // Task-ul pentru butoanele de incrementare/decrementare
  task1(); // Task-ul pentru LED-ul principal
  task2(); // Task-ul pentru LED-ul intermitent
}
```

```

task4(); // Task-ul pentru consumul de energie
idleTask(); // Task-ul de Idle pentru afișarea stărilor
}

```

Primul și al doilea task corelează între ele. La început, când butonul este apăsat, primul LED începe să alterneze între stins și aprins la fiecare 2 secunde. Dacă butonul este apăsat din nou, primul LED se stinge definitiv, iar un al doilea LED se aprinde și rămâne aprins până la o nouă apăsare a butonului. La următoarea apăsare, primul LED reia alternarea stării la intervalul inițial stabilit (Vezi Fig 2).

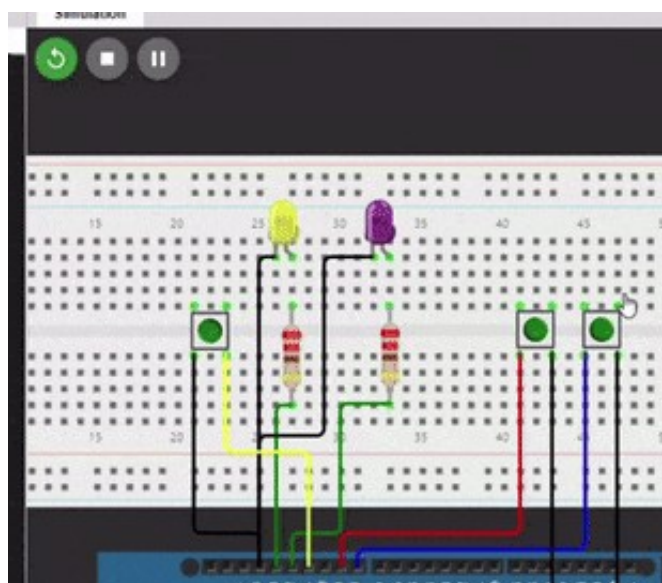


Figura 2 – Rezultatul taskului 1 și 2

Următorul task gestionează incrementarea și decrementarea unei variabile prin apăsarea a două butoane separate. Această variabilă reprezintă numărul de recurențe sau intervalul de timp în care LED-ul din cel de-al doilea task rămâne într-o anumită stare, fie aprins, fie stins. La apăsarea butonului de incrementare, valoarea variabilei crește, mărinnd astfel intervalul de timp pentru schimbarea stării LED-ului. În mod similar, apăsarea butonului de decrementare va reduce valoarea variabilei, scurtând intervalul de timp al LED-ului (Vezi Fig 3).

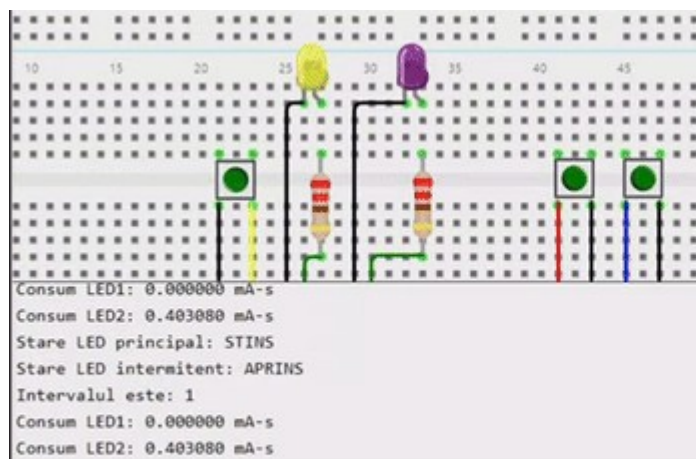


Figura 3 – Rezultatul taskului 3

Task-ul 4 se ocupă cu monitorizarea timpului de aprindere pentru cele două LED-uri, precum și cu calcularea consumului estimat de energie. Atunci când LED-ul principal este aprins, se măsoară timpul total în care acesta a fost activ, acumulându-se în variabila `led1OnTime`. De asemenea, se calculează și timpul de aprindere pentru LED-ul intermitent, care se activează în funcție de starea sa. Consumurile energetice pentru fiecare LED sunt apoi estimate folosind formula corespunzătoare, ținând cont de timpul de aprindere și de curentul consumat de LED-uri.

Rezultatele sunt convertite într-un format adecvat pentru afișare și sunt prezentate utilizând funcția `printf`, care permite o formatare clară a datelor. Astfel, utilizatorul poate observa în mod clar consumul de energie al fiecărui LED, iar actualizarea acestor informații se realizează la fiecare secundă.

```

Consum LED1: 0.000000 mA-s
Consum LED2: 0.736120 mA-s

```

Figura 4 – Rezultatul taskului 4

Task-ul Idle are rolul de a gestiona și afișa informațiile legate de interacțiunile utilizatorului și de starea LED-urilor. În cazul în care butonul principal a fost apăsător, un mesaj informativ este afișat, indicând acest lucru. Task-ul continuă prin monitorizarea stării LED-urilor, prezentând dacă LED-ul principal și cel intermitent sunt aprinse sau stinse.

De asemenea, intervalul setat pentru LED-ul intermitent este afișat, oferind utilizatorului o imagine de ansamblu asupra funcționării sistemului. La fel ca în Task-ul 4, informațiile sunt prezentate utilizând funcția `printf`, asigurând o comunicare eficientă și clară a stării curente a sistemului, actualizându-se la fiecare secundă.

```
Stare LED principal: STINS  
Stare LED intermitent: STINS  
Intervalul este: 1
```

Figura 5 – Rezultatul taskului Idle

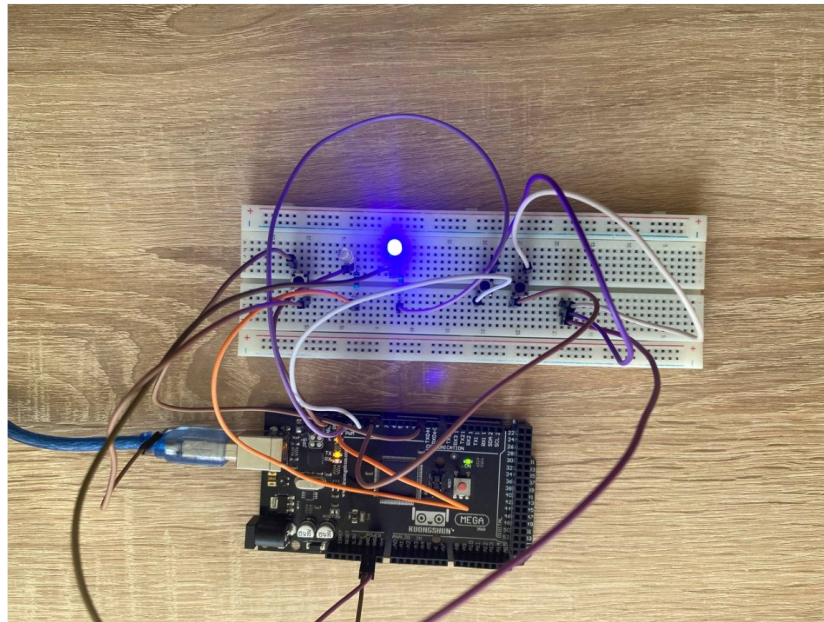


Figura 6 – Schema circuitului fizic

Concluzie

Tema laboratorului a constatat în realizarea unei aplicații pentru microcontrolere (MCU) care să implementeze cel puțin trei task-uri secvențiale. Prin dezvoltarea acestui proiect, s-a demonstrat abilitatea de a gestiona interacțiunea între diferite componente hardware, precum LED-uri și butoane, utilizând o structură de task-uri care comunică între ele.

Aplicația a evidențiat importanța organizării corecte a codului și a gestionării eficiente a timpului, prin implementarea unor funcționalități precum controlul stării LED-urilor în funcție de apăsarea butoanelor și monitorizarea consumului de energie. De asemenea, utilizarea unei interfețe seriale a permis vizualizarea stărilor și a consumului energetic, facilitând astfel debugging-ul și testarea aplicației.

În concluzie, laboratorul a oferit oportunitatea de a aprofunda cunoștințele despre microcontrolere și de a aplica conceptele teoretice în dezvoltarea practică a unui sistem funcțional, promovând abilități esențiale în programarea și gestionarea echipamentelor electronice. Aceasta experiență a consolidat fundamentul necesar pentru explorarea ulterioară a tehnologiilor IoT și a arhitecturii sistemelor integrate.

Anexa A

stdinout.h

```
#ifndef _STDINOUT_H
#define _STDINOUT_H

// no need to make an instance of this yourself
class initializeSTDINOUT
{
    static size_t initnum;
```

```

public:
    // Constructor
    initializeSTDINOUT();
};

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

#endif

```

stdnout.cpp

```

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

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

    return Serial.write(c) == 1 ? 0 : 1;
}

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

    return Serial.read();
}

static FILE serial_stdnout;

static void setup_stdin_stdout()
{
    // Set up stdout and stdin
    fdev_setup_stream(&serial_stdnout, serial_putchar, serial_getchar, _FDEV_SETUP_RW);
    stdout = &serial_stdnout;
    stdin = &serial_stdnout;
    stderr = &serial_stdnout;
}

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

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

```

Anexa B

Lab2.1.ino

```

// Definirea pinilor și variabilelor pentru consumul de energie
const float ledCurrent = 0.02; // Consum curent al LED-urilor în amperi (20 mA)
unsigned long ledOnTime = 0;

```



```

unsigned long led2OnTime = 0;
unsigned long prevLed1Time = 0;
unsigned long prevLed2Time = 0;

// Definirea pinilor pentru LED-uri și butoane
const int led1Pin = 13; // Pin pentru LED1
const int led2Pin = 12; // Pin pentru LED2
const int buttonPin = 11; // Pin pentru buton principal
const int incButtonPin = 9; // Pin pentru buton de incrementare
const int decButtonPin = 8; // Pin pentru buton de decrementare

#define MAIN_LED_PIN led1Pin
#define INTERMITENT_LED_PIN led2Pin
#define MAIN_BUTTON_PIN buttonPin
#define INC_BUTTON_PIN incButtonPin
#define DEC_BUTTON_PIN decButtonPin

// Variabile globale pentru timpi și stări
unsigned int currentTime = 0;
unsigned int prevTime_T1 = 0;
unsigned int delayTime = 1000; // Timpul implicit pentru LED-ul intermitent
byte ledState = LOW;
byte interLedState = LOW;
bool buttonPressed = false; // Variabila de stare a butonului (provider/consumer)
int interval = 1; // Variabila pentru incrementare/decrementare

// Prototipuri de funcții
void task1();
void task2();
void task3();
void task4();
void idleTask();

void setup() {
    // Inițializarea pinilor
    pinMode(MAIN_LED_PIN, OUTPUT);
    pinMode(INTERMITENT_LED_PIN, OUTPUT);
    pinMode(MAIN_BUTTON_PIN, INPUT_PULLUP); // Folosim pull-up pentru buton
    pinMode(INC_BUTTON_PIN, INPUT_PULLUP); // Buton de incrementare
    pinMode(DEC_BUTTON_PIN, INPUT_PULLUP); // Buton de decrementare

    Serial.begin(9600); // Pentru configurarea serialului
}

void loop() {
    currentTime = millis();
    task3(); // Task-ul pentru butoanele de incrementare/decrementare
    task1(); // Task-ul pentru LED-ul principal
    task2(); // Task-ul pentru LED-ul intermitent
    task4(); // Task-ul pentru consumul de energie
    idleTask(); // Task-ul de Idle pentru afișarea stărilor
}

void task1() {
    if (digitalRead(MAIN_BUTTON_PIN) == LOW) {
        while (digitalRead(MAIN_BUTTON_PIN) == LOW); // Așteptăm eliberarea butonului
        ledState = !ledState; // Comutăm starea LED-ului principal
        digitalWrite(MAIN_LED_PIN, ledState);
        buttonPressed = true; // Setăm variabila la apăsarea butonului
    }
}

void task2() {
    if (currentTime - prevTime_T1 > delayTime * interval) { // Ajustăm durata cu intervalul
        if (ledState == HIGH) {
            interLedState = LOW;
            digitalWrite(INTERMITENT_LED_PIN, interLedState);
        } else {
            interLedState = !interLedState; // Comutăm starea LED-ului intermitent
            digitalWrite(INTERMITENT_LED_PIN, interLedState);
        }
    }
}

```

```

    prevTime_T1 = currentTime; // Actualizăm timpul anterior
}
}

void task3() {
    // Verificăm butonul de incrementare
    if (digitalRead(INC_BUTTON_PIN) == LOW) {
        while (digitalRead(INC_BUTTON_PIN) == LOW); // Așteptăm eliberarea butonului
        interval++; // Incrementăm valoarea intervalului
        if (interval > 10) interval = 10; // Limita superioară
        buttonPressed = true; // Setăm variabila la apăsarea butonului
    }

    // Verificăm butonul de decrementare
    if (digitalRead(DEC_BUTTON_PIN) == LOW) {
        while (digitalRead(DEC_BUTTON_PIN) == LOW); // Așteptăm eliberarea butonului
        interval--; // Decrementăm valoarea intervalului
        if (interval < 1) interval = 1; // Limita inferioară
        buttonPressed = true; // Setăm variabila la apăsarea butonului
    }
}

void task4() {
    // Calculăm timpul de aprindere pentru LED1
    if (ledState == HIGH) {
        led1OnTime += millis() - prevLed1Time;
    }
    prevLed1Time = millis();

    // Calculăm timpul de aprindere pentru LED2 (intermitent)
    if (interLedState == HIGH) {
        led2OnTime += millis() - prevLed2Time;
    }
    prevLed2Time = millis();

    // Afișăm consumul estimat de energie
    float led1Energy = (led1OnTime / 1000.0) * ledCurrent; // Energie în mA-s
    float led2Energy = (led2OnTime / 1000.0) * ledCurrent;

    char buffer[20]; // Buffer for storing converted float to string
    dtostrf(led1Energy, 6, 6, buffer); // Convert float to string
    printf("Consum LED1: %s mA-s\n", buffer); // Print converted string

    dtostrf(led2Energy, 6, 6, buffer); // Convert float to string
    printf("Consum LED2: %s mA-s\n", buffer); // Print converted string

    delay(1000); // Actualizăm la fiecare secundă
}

void idleTask() {
    // Afișăm mesaj dacă butonul principal a fost apăsător
    if (buttonPressed) {
        printf("Butonul principal a fost apăsător!\n"); // Print instead of Serial.println
        buttonPressed = false; // Resetăm variabila după afișarea mesajului
    }

    // Afișăm stările LED-urilor în mod continuu
    printf("Stare LED principal: %s\n", ledState == HIGH ? "APRINS" : "STINS");
    printf("Stare LED intermitent: %s\n", interLedState == HIGH ? "APRINS" : "STINS");
    printf("Intervalul este: %d\n", interval);

    delay(1000); // Afișăm stările o dată pe secundă
}

```

Lab2.2.ino

```

#include <Arduino.h>
#include <Arduino_FreeRTOS.h>
#include "stdiostream.h" // Include your stdiostream header

// Definirea pinilor și variabilelor pentru consumul de energie
const float ledCurrent = 0.02; // Consum curent al LED-urilor în amperi (20 mA)
unsigned long led1OnTime = 0;
unsigned long led2OnTime = 0;
unsigned long prevLed1Time = 0;
unsigned long prevLed2Time = 0;

// Definirea pinilor pentru LED-uri și butoane
const int led1Pin = 13; // Pin pentru LED1
const int led2Pin = 12; // Pin pentru LED2
const int buttonPin = 11; // Pin pentru buton principal
const int incButtonPin = 9; // Pin pentru buton de incrementare
const int decButtonPin = 8; // Pin pentru buton de decrementare

#define MAIN_LED_PIN led1Pin
#define INTERMITENT_LED_PIN led2Pin
#define MAIN_BUTTON_PIN buttonPin
#define INC_BUTTON_PIN incButtonPin
#define DEC_BUTTON_PIN decButtonPin

// Variabile globale pentru timpi și stări
unsigned int currentTime = 0;
unsigned int prevTime_T1 = 0;
unsigned int delayTime = 1000; // Timpul implicit pentru LED-ul intermitent
byte ledState = LOW;
byte interLedState = LOW;
bool buttonPressed = false; // Variabila de stare a butonului (provider/consumer)
int interval = 1; // Variabila pentru incrementare/decrementare

// Prototipuri de funcții
void task1(void* parameters);
void task2(void* parameters);
void task3(void* parameters);
void task4(void* parameters);
void idleTask(void* parameters);

void setup() {
    // Inițializarea pinilor
    pinMode(MAIN_LED_PIN, OUTPUT);
    pinMode(INTERMITENT_LED_PIN, OUTPUT);
    pinMode(MAIN_BUTTON_PIN, INPUT_PULLUP); // Folosim pull-up pentru buton
    pinMode(INC_BUTTON_PIN, INPUT_PULLUP); // Buton de incrementare
    pinMode(DEC_BUTTON_PIN, INPUT_PULLUP); // Buton de decrementare

    Serial.begin(9600); // Pentru inițializarea Serial

    // Crearea task-urilor
    xTaskCreate(task1, "Task 1", 1000, NULL, 1, NULL);
    xTaskCreate(task2, "Task 2", 1000, NULL, 1, NULL);
    xTaskCreate(task3, "Task 3", 1000, NULL, 1, NULL);
    xTaskCreate(task4, "Task 4", 1000, NULL, 1, NULL);
    xTaskCreate(idleTask, "Idle Task", 1000, NULL, 1, NULL);
}

void loop() {
    // Nu este necesară utilizarea funcției loop, FreeRTOS se ocupă de execuția task-urilor
}

// Task pentru LED-ul principal
void task1(void* parameters) {
    for (;;) {
        if (digitalRead(MAIN_BUTTON_PIN) == LOW) {
            vTaskDelay(50 / portTICK_PERIOD_MS); // Debounce
            while (digitalRead(MAIN_BUTTON_PIN) == LOW); // Așteptăm eliberarea butonului
            ledState = !ledState; // Comutăm starea LED-ului principal
        }
    }
}

```

```

        digitalWrite(MAIN_LED_PIN, ledState);
        buttonPressed = true; // Setăm variabila la apăsarea butonului
    }
    vTaskDelay(100 / portTICK_PERIOD_MS);
}

// Task pentru LED-ul intermitent
void task2(void* parameters) {
    for (;;) {
        currentTime = millis();
        if (currentTime - prevTime_T1 > delayTime * interval) { // Ajustăm durata cu intervalul
            interLedState = (ledState == HIGH) ? LOW : !interLedState; // Comutăm starea LED-ului
intermitent
            digitalWrite(INTERMITENT_LED_PIN, interLedState);
            prevTime_T1 = currentTime; // Actualizăm timpul anterior
        }
        vTaskDelay(100 / portTICK_PERIOD_MS);
    }
}

// Task pentru gestionarea butoanelor de incrementare/decrementare
void task3(void* parameters) {
    for (;;) {
        // Verificăm butonul de incrementare
        if (digitalRead(INC_BUTTON_PIN) == LOW) {
            vTaskDelay(50 / portTICK_PERIOD_MS); // Debounce
            while (digitalRead(INC_BUTTON_PIN) == LOW); // Așteptăm eliberarea butonului
            interval++; // Incrementăm valoarea intervalului
            if (interval > 10) interval = 10; // Limita superioară
            buttonPressed = true; // Setăm variabila la apăsarea butonului
        }

        // Verificăm butonul de decrementare
        if (digitalRead(DEC_BUTTON_PIN) == LOW) {
            vTaskDelay(50 / portTICK_PERIOD_MS); // Debounce
            while (digitalRead(DEC_BUTTON_PIN) == LOW); // Așteptăm eliberarea butonului
            interval--; // Decrementăm valoarea intervalului
            if (interval < 1) interval = 1; // Limita inferioară
            buttonPressed = true; // Setăm variabila la apăsarea butonului
        }
        vTaskDelay(100 / portTICK_PERIOD_MS);
    }
}

// Task pentru calculul consumului de energie
void task4(void* parameters) {
    for (;;) {
        unsigned long currentMillis = millis(); // Capture the current time at the start of the loop

        // Calculăm timpul de aprindere pentru LED1
        if (ledState == HIGH) {
            led1OnTime += currentMillis - prevLed1Time; // Add the elapsed time since last check
        }
        prevLed1Time = currentMillis; // Update previous time

        // Calculăm timpul de aprindere pentru LED2 (intermitent)
        if (interLedState == HIGH) {
            led2OnTime += currentMillis - prevLed2Time; // Add the elapsed time since last check
        }
        prevLed2Time = currentMillis; // Update previous time

        // Afișăm consumul estimat de energie
        float led1Energy = (led1OnTime / 1000.0) * ledCurrent; // Energie în mA-s
        float led2Energy = (led2OnTime / 1000.0) * ledCurrent;

        char buffer[20]; // Buffer for storing converted float to string
        dtostrf(led1Energy, 6, 6, buffer); // Convert float to string
        printf("Consum LED1: %s mA-s\n", buffer); // Print converted string

        dtostrf(led2Energy, 6, 6, buffer); // Convert float to string
    }
}

```

```

    printf("Consum LED2: %s mA-s\n", buffer); // Print converted string

    vTaskDelay(1000 / portTICK_PERIOD_MS); // Actualizăm la fiecare secundă
}

// Task pentru afișarea stărilor
void idleTask(void* parameters) {
    for (;;) {
        // Afișăm mesaj dacă butonul principal a fost apăsăat
        if (buttonPressed) {
            printf("Butonul principal a fost apăsăat!\n");
            buttonPressed = false; // Resetăm variabila după afișarea mesajului
        }

        // Afișăm stările LED-urilor în mod continuu
        printf("Stare LED principal: %s\n", (ledState == HIGH ? "APRINS" : "STINS"));
        printf("Stare LED intermitent: %s\n", (interLedState == HIGH ? "APRINS" : "STINS"));
        printf("Intervalul este: %d\n", interval);

        vTaskDelay(1000 / portTICK_PERIOD_MS); // Afișăm stările o dată pe secundă
    }
}

```