Ministerul Educației și Cercetării al Republicii Moldova
Universitatea Tehnică a Moldovei
Facultatea Calculatoare, Informatică și Microelectronică
Departamentul Ingineria Software și Automatică

Lucrare de laborator Nr. 2

Disciplina: Internetul lucrurilor (IOT)
Tema: Sisteme de operare

A efectuat: Popa Cătălin, gr. TI-211

A verificat: Cristian Lupan, asist. univ.

Chișinău 2024

2.1 Problema propusă:

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

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.

2.2 Problema propusă:

Realizarea unei aplicații pentru MCU care va rula minim 3 task-uri cu FreeRTOS

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.

Obiective

1. Realizarea unei aplicații pentru MCU care va rula minim 3 taskuri secvențial.
2. Realizarea unei aplicații pentru MCU care va rula minim 3 taskuri cu FreeRTOS.

Introducere

Microcontrolerele permit controlul și monitorizarea dispozitivelor fizice, aducând tehnologia mai aproape de utilizator prin intermediul unei interfețe intuitive și simplificate. Un exemplu clasic de interacțiune între utilizator și dispozitive fizice este controlul unui LED prin apăsarea unui buton sau trimiterea de comenzi printr-o interfață serială. Această tehnologie este utilizată într-o gamă largă de aplicații, de la sisteme de iluminat inteligente până la dispozitive electronice complexe utilizate în automatizări industriale. Unul dintre avantajele microcontrolerelor este costul redus și flexibilitatea lor, permițând implementarea rapidă de prototipuri și teste pentru diverse aplicații IoT. Microcontrolerele precum cele din familia Arduino sau ESP sunt folosite frecvent pentru prototiparea sistemelor IoT datorită ecosistemului extins de biblioteci și comunități de dezvoltatori care susțin aceste platforme .

FreeRTOS este un sistem de operare în timp real (RTOS) conceput pentru a gestiona sarcini multiple într-un mod eficient și fiabil. Este open-source și foarte popular în dezvoltarea de aplicații, având suport pentru o gamă largă de microcontrolere și arhitecturi hardware. Sistemul său de sarcini permite crearea și gestionarea de sarcini ce pot rula simultan, fiecare având propriul set de priorități. Interfața de programare este simplă și bine documentată, ceea ce ajută la dezvoltarea și debugging-ul aplicațiilor. De asemenea, FreeRTOS este optimizat pentru a utiliza resursele sistemului într-un mod eficient, fiind ideal pentru aplicații care funcționează pe hardware cu resurse limitate. Mecanismele de comunicare între sarcini, precum cozi și semafoare, facilitează schimbul de date, îmbunătățind astfel interacțiunea și colaborarea între diferitele sarcini.

Materiale și metode

Arduino UNO - platformă de microcontroler open-source bazată pe microcontrolerul ATmega328P. Este ușor de utilizat și permite conectarea ușoară a componentelor externe prin pini digitali și analogici. Acesta este ideal pentru prototiparea de sisteme IoT și proiecte electronice interactive. Afișarea este în figura 1.1.

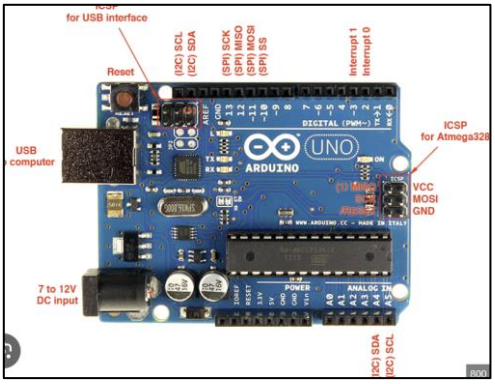


Fig. 1.1. Arduino UNO

- 14 pini digitali I/O (din care 6 sunt PWM);
- 6 intrări analogice;
- memorie flash de 32 KB;
- tensiune de operare: 5V;
- interfață de programare: Arduino IDE.

LED-ul - (diodă emițătoare de lumină) este utilizată pentru a semnaliza starea sistemului în funcție de comenzile primite sau de validitatea codului introdus pe tastatură. Afișarea este în figura 1.2.

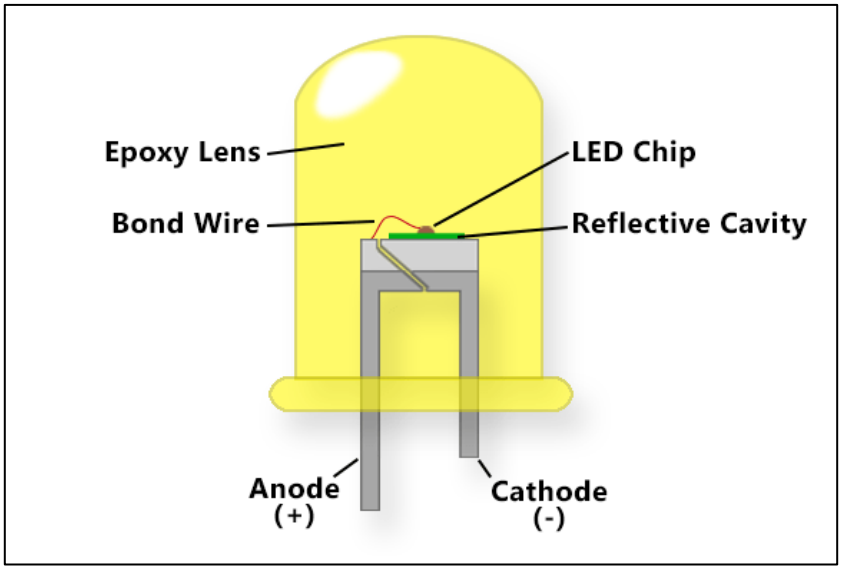


Fig. 1.2. LED

- tensiune de operare: 2V-3V;
- curent: ~20mA;
- LED verde pentru semnalizarea codurilor valide;
- LED roșu pentru semnalizarea codurilor invalide.

Rezistor - Rezistorul este utilizat pentru a limita curentul ce trece prin LED, prevenind arderea acestuia. Pentru a proteja LED-urile, se vor folosi rezistoare de aproximativ 220Ω. Afișarea este în figura 1.3.

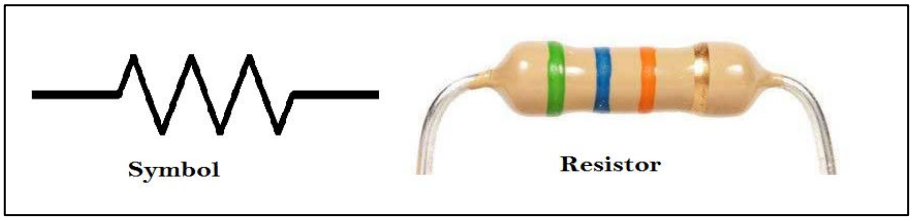


Fig. 1.3. Rezistor

Fire Jumper - Firele jumper sunt utilizate pentru a face legături electrice între diverse componente și plăcuța Arduino. Acestea permit conectarea componentelor fără a fi necesară lipirea acestora. Afișarea este în figura 1.6.

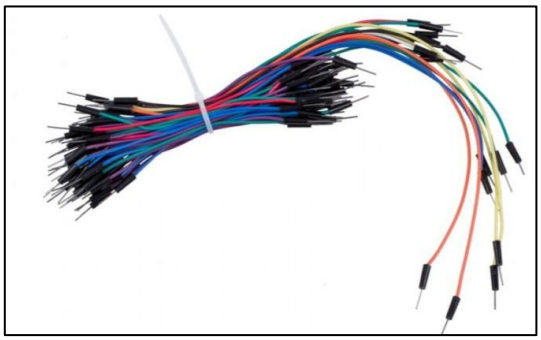


Fig. 1.6. Fire Jumper

Buton - În aplicații de acest gen, un buton (push button) este utilizat pentru a oferi utilizatorului o modalitate de a interacționa cu sistemul printr-un eveniment de apăsare. În această situație, ar putea fi utilizat pentru a reseta sistemul sau pentru a iniția o acțiune nouă. Afișarea este în figura 1.7.

Fig. 1.7. Buton

Metode

Configurarea Arduino UNO: Arduino va fi configurat folosind Arduino IDE. Se va implementa codul pentru a citi inputurile de la tastatura 4x4 și pentru a controla LED-urile și afișarea pe LCD.

Interacțiunea cu utilizatorul: Sistemul va primi comenzi fie prin terminalul serial, fie prin tastatura matricială, și va răspunde prin intermediul LCD-ului și al LED-urilor. Comenzile "led on" și "led off" vor aprinde și stinge LED-ul respectiv, iar codurile introduse vor fi validate, aprinzând LED-ul verde pentru coduri corecte și LED-ul roșu pentru coduri greșite.

Validare în simulator: Codul și conexiunile vor fi testate inițial într-un simulator de circuite, cum ar fi Tinkercad sau Proteus, pentru a verifica funcționalitatea corectă. După simulare, se va testa și pe un circuit real.

2.1

Pentru prima sarcină am avut nevoie de 3 butoane, 2 leduri și pentru sarcina individuală am luat un dispozitiv care redă sunete. Mai întâi am conectat toate componentele necesare, apoi am trecut la partea de cod. Componentele conectate sunt reprezentate în figura 1.8.

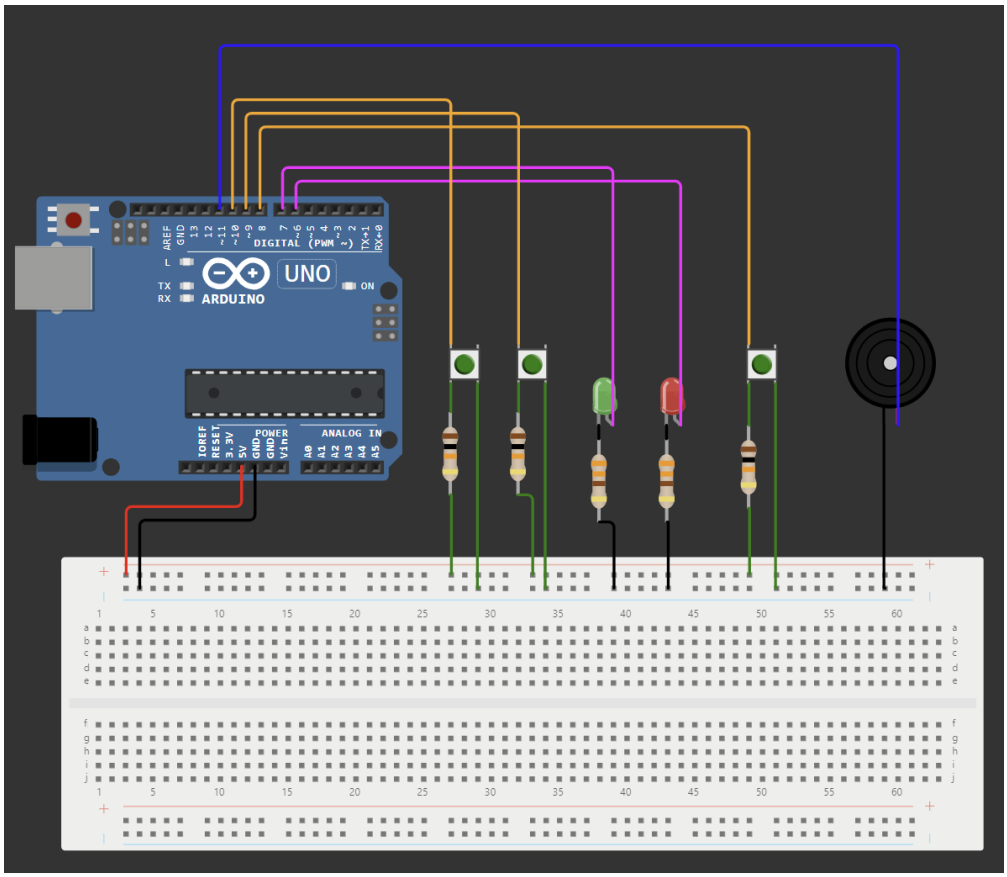


Fig. 1.8. Taskuri secvential

Mai întâi au fost deiniți toți pinii necesari, apoi am trecut la partea cu taskuri. Primul task creat este pentru controlul primului led, cu ajutorul la un buton. Din motiv că BUTTON1_PIN sunt configurați cu pini de intrare care utilizeaza rezisotri de pull-up interni, starea când butonul este LOW, indică că el este apăsat, iar starea HIGH, indică că butonul nu este apăsat. Respectiv, mai întâi se verifică dacă butonul este apăsat apoi ledul își schimbă starea. La fel, se afișează textul precum că ledul este pornit s-au oprit.

```
void controlLED1() {
    if (digitalRead(BUTTON1_PIN) == LOW) {
      led1State = !led1State;
      digitalWrite(LED1_PIN, led1State);
      Serial.print("LED 1 este ");
      Serial.println(led1State ? "Pornit" : "Oprit");
  }
}
```

Al doilea task, a fost creat pentru LED-ul intermitent, dacă LED 1 este oprit. Aici se verifică mai întâi starea la primul led, daca este pornit, atunci ledul 2 nu se apinrde, iar dacă LED 1 este stins, LED 2 va începe a pulsa, în dependență de ce interval este setat.

```
void ledIntermitent() {
  if (digitalRead(LED1_PIN) == LOW && digitalRead(BUTTON1_PIN) == HIGH) {
    unsigned long currentTime = millis();
    if (currentTime - lastBlinkTime >= blinkInterval) {
      lastBlinkTime = currentTime;
      digitalWrite(LED2_PIN, !digitalRead(LED2_PIN));
    }
  } else {
    digitalWrite(LED2_PIN, LOW);
  }
}
```

Ultimul task a fost pentru a incrementa și decrementa valoarea de interval pentru intermitență. Se face mai întâi verificare care buton a fost tastat, apoi valoarea se incrementeaza sau se decrementează cu 100. La final se redă un sunet la 1000 HZ timp de 100 ms.

```
void modificaInterval() {
  if (digitalRead(BUTTON_INC_PIN) == LOW) {
    delay(50);
    if (digitalRead(BUTTON_INC_PIN) == LOW) {
      blinkInterval += 100;
      Serial.print("Interval intermitență: ");
      Serial.println(blinkInterval);
      delay(300);
    }
  }
  if (digitalRead(BUTTON_DEC_PIN) == LOW) {
    delay(50);
    if (digitalRead(BUTTON_DEC_PIN) == LOW && blinkInterval > 100) {
      blinkInterval -= 100;
      Serial.print("Interval intermitență: ");
      Serial.println(blinkInterval);
      delay(300);
    }
  }
}
```

Pentru a realiza task-ul suplimetar, am creat o funcție nouă care va reda un sunet atunci când LED-ul intermintent se aprinde.

```
void getSound(){
 if(digitalRead(LED2_PIN) == HIGH) {
  tone(SONG_PIN, 1000, 100);
 }}
```

La fel, pentru a afișa starea ledurilor, am craet o funcție care afișează dacă ledul este pornit sau oprit. Am pus un delay, pentru ca să se afișeze la fiecare o secundă.

```
void afiseazaStari() {
  Serial.print("LED 1: ");
  Serial.println(led1State ? "Pornit" : "Oprit");
  Serial.print("LED 2: ");
  Serial.println(digitalRead(LED2_PIN) ? "Pornit" : "Oprit");
  delay(1000);
}
```

2.2

În sarcina a doua, am realizat un exemplu clasic de multitasking cu FreeRTOS, unde fiecare task are o responsabilitate clară și utilizează semafoare pentru sincronizare între task-uri. FreeRTOS oferă un control fin asupra task-urilor și permite realizarea unor aplicații cu multiple fire de execuție pe microcontrolere, ceea ce face programul eficient și ușor de extins. Utilizarea semafoarelor pentru sincronizare și raportare asigură că sistemul este bine organizat și evită conflictele între task-uri.

Acest task monitorizează starea unui buton (BUTTON1_PIN) și controlează starea unui LED (LED1_PIN) în funcție de apăsarea acestuia. Când butonul este apăsat, LED-ul își schimbă starea (de la ON la OFF sau invers). Codul include o mică întârziere de debounce (50 ms) pentru a preveni fluctuațiile rapide ale butonului și o întârziere suplimentară (300 ms) pentru a preveni apăsările repetate.

```
void ButtonLedTask(void *pvParameters) {
  pinMode(LED1_PIN, OUTPUT);
  pinMode(BUTTON1_PIN, INPUT_PULLUP);
  while (1) {
    if (digitalRead(BUTTON1_PIN) == LOW) {
      delay(50);  // Debounce simplu
      if (digitalRead(BUTTON1_PIN) == LOW) {
        led1State = !led1State;
        digitalWrite(LED1_PIN, led1State);
```

```
        xSemaphoreGive(reportSemaphore);
        delay(300);
      }
    }
    vTaskDelay(10 / portTICK_PERIOD_MS);
  }
}
```

Task-ul controlează intermitența unui al doilea LED (LED2_PIN) și activează buzzer-ul (BUZZER_PIN) în timpul activării LED-ului. Dacă LED-ul 1 este oprit, LED-ul 2 clipește la un interval variabil (blinkInterval), iar buzzer-ul pornește și se oprește sincronizat cu LED-ul. Dacă LED-ul 1 este pornit, LED-ul 2 și buzzer-ul sunt dezactivate.

```
void BlinkLedTask(void *pvParameters) {
  pinMode(LED2_PIN, OUTPUT);
  unsigned long lastBlinkTime = 0;
  while (1) {
    if (!led1State) {
      unsigned long currentTime = millis();
      if (currentTime - lastBlinkTime >= blinkInterval) {
        lastBlinkTime = currentTime;
        bool led2State = !digitalRead(LED2_PIN);
        digitalWrite(LED2_PIN, led2State);

        if (led2State) {
          tone(BUZZER_PIN, 1000);
        } else {
          noTone(BUZZER_PIN);
        }
      }
    } else {
      digitalWrite(LED2_PIN, LOW);
      noTone(BUZZER_PIN);
```

```
    }
    vTaskDelay(10 / portTICK_PERIOD_MS);
  }
}
```

Controlează creșterea și descreșterea valorii variabilei blinkInterval prin apăsarea a două butoane separate (BUTTON_INC_PIN și BUTTON_DEC_PIN). Când butonul de creștere este apăsat, intervalul crește cu 100 ms, iar când cel de descreștere este apăsat, scade cu 100 ms, dar nu mai puțin de 100 ms. Semaforul este folosit pentru a semnala că valoarea intervalului s-a schimbat.

```
void VariableControlTask(void *pvParameters) {
  pinMode(BUTTON_INC_PIN, INPUT_PULLUP);
  pinMode(BUTTON_DEC_PIN, INPUT_PULLUP);
  while (1) {
    if (digitalRead(BUTTON_INC_PIN) == LOW) {
      delay(50);
      if (digitalRead(BUTTON_INC_PIN) == LOW) {
        blinkInterval += 100;
        xSemaphoreGive(reportSemaphore);
        delay(300);
      }
    }
    if (digitalRead(BUTTON_DEC_PIN) == LOW) {
      delay(50);
      if (digitalRead(BUTTON_DEC_PIN) == LOW && blinkInterval > 100) {
        blinkInterval -= 100;
        xSemaphoreGive(reportSemaphore);
        delay(300);
      }
    }
    vTaskDelay(10 / portTICK_PERIOD_MS);
```

```
  }
}
```

Monitorizează semaforul reportSemaphore și afișează starea actuală a LED-ului 1 și valoarea intervalului LED-ului 2 pe portul Serial. De fiecare dată când un alt task trimite semnalul prin semafor, acest task afișează pe Serial starea curentă a LED-ului 1 și valoarea variabilei blinkInterval.

```
void ReportTask(void *pvParameters) {
  while (1) {
    if (xSemaphoreTake(reportSemaphore, portMAX_DELAY) == pdTRUE) {
      // Afișare stare pe Serial
      Serial.print("LED 1: ");
      Serial.println(led1State ? "Pornit" : "Oprit");
      Serial.print("LED 2 Blink Interval: ");
      Serial.println(blinkInterval);
    }
    vTaskDelay(500 / portTICK_PERIOD_MS);
  }
}
```

ReportSemaphore = xSemaphoreCreateBinary() creează un semafor binar pentru sincronizarea task-urilor, în special pentru raportarea schimbărilor. xTaskCreate() creează și pornește cele patru task-uri: unul pentru controlul LED-ului prin buton, unul pentru intermitența LED-ului 2 și buzzer, unul pentru ajustarea intervalului de intermitență prin butoane și unul pentru raportarea schimbărilor prin Serial. vTaskStartScheduler() pornește sistemul de task-uri gestionat de FreeRTOS.

```
void setup() {
  Serial.begin(9600);
  reportSemaphore = xSemaphoreCreateBinary();
  xTaskCreate(ButtonLedTask, "ButtonLed", 128, NULL, 1, NULL);
  xTaskCreate(BlinkLedTask, "BlinkLed", 128, NULL, 1, NULL);
  xTaskCreate(VariableControlTask, "VariableControl", 128, NULL, 1, NULL);
```

```
  xTaskCreate(ReportTask, "Report", 128, NULL, 1, NULL);

  vTaskStartScheduler();
}
void loop() {
}
```

Concluzii

În cadrul acestui laborator, am realizat un sistem de control bazat pe multitasking folosind FreeRTOS, care implică interacțiunea cu butoane și LED-uri. Proiectul a fost structurat în patru taskuri principale, fiecare având o responsabilitate clar definită, ceea ce a contribuit la organizarea eficientă a codului și la o funcționare coerentă a aplicației. Prin utilizarea FreeRTOS, am reușit să demonstrăm capacitatea de a gestiona mai multe taskuri în mod eficient, asigurând astfel o aplicație bine organizată, ușor de extins și de întreținut. Această abordare de programare este ideală pentru dezvoltarea sistemelor integrate pe microcontrolere, unde resursele sunt limitate, dar necesitățile de funcționalitate sunt variate.

Anexă

2.1

```
#define LED1_PIN 7
#define LED2_PIN 6
#define BUTTON1_PIN 8
#define BUTTON_INC_PIN 9
#define BUTTON_DEC_PIN 10
#define SONG_PIN 11

bool led1State = false;
unsigned long lastBlinkTime = 0;
int blinkInterval = 1000;

void setup() {
  pinMode(LED1_PIN, OUTPUT);
  pinMode(LED2_PIN, OUTPUT);
  pinMode(SONG_PIN, OUTPUT);

  pinMode(BUTTON1_PIN, INPUT_PULLUP);
  pinMode(BUTTON_INC_PIN, INPUT_PULLUP);
  pinMode(BUTTON_DEC_PIN, INPUT_PULLUP);

  Serial.begin(9600);
}

void loop() {
  controlLED1();
```

```
  ledIntermitent();
  getSound();
  modificaInterval();
  afiseazaStari();
}

void controlLED1() {
  if (digitalRead(BUTTON1_PIN) == LOW) {
    led1State = !led1State;
    digitalWrite(LED1_PIN, led1State);
    Serial.print("LED 1 este ");
    Serial.println(led1State ? "Pornit" : "Oprit");
  }
}

void getSound() {
  if (digitalRead(LED1_PIN) == LOW && digitalRead(BUTTON1_PIN) == HIGH) {
    if (digitalRead(LED2_PIN) == HIGH) {
      tone(SONG_PIN, 1000, 100);
    }
  } else {
    noTone(SONG_PIN);
  }
}

void ledIntermitent() {
  if (digitalRead(LED1_PIN) == LOW && digitalRead(BUTTON1_PIN) == HIGH) {
```

```
    unsigned long currentTime = millis();
    if (currentTime - lastBlinkTime >= blinkInterval) {
      lastBlinkTime = currentTime;
      digitalWrite(LED2_PIN, !digitalRead(LED2_PIN));
    }
  } else {
    digitalWrite(LED2_PIN, LOW);
  }
}

void modificaInterval() {
  if (digitalRead(BUTTON_INC_PIN) == LOW) {
    delay(50);
    if (digitalRead(BUTTON_INC_PIN) == LOW) {
      blinkInterval += 100;
      Serial.print("Interval intermitență: ");
      Serial.println(blinkInterval);
      delay(300);
    }
  }

  if (digitalRead(BUTTON_DEC_PIN) == LOW) {
    delay(50);
    if (digitalRead(BUTTON_DEC_PIN) == LOW && blinkInterval > 100) {
      blinkInterval -= 100;
      Serial.print("Interval intermitență: ");
      Serial.println(blinkInterval);
```

```
      delay(300);
    }
  }
}

void afiseazaStari() {
  Serial.print("LED 1: ");
  Serial.println(led1State ? "Pornit" : "Oprit");
  Serial.print("LED 2: ");
  Serial.println(digitalRead(LED2_PIN) ? "Pornit" : "Oprit");
  delay(1000);
}
```

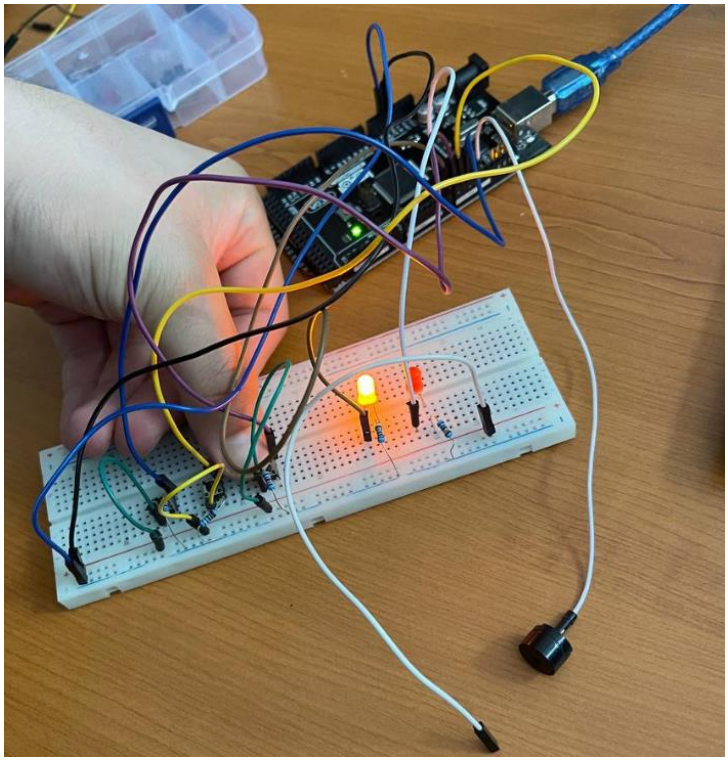
2.2

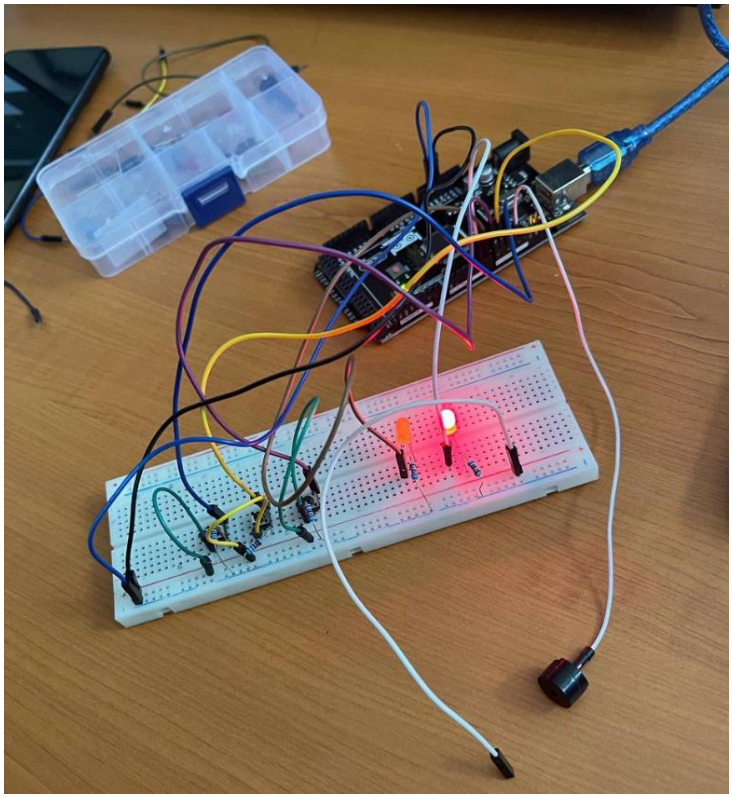
```
#include <Arduino.h>
#include <Arduino_FreeRTOS.h>
#include <semphr.h>

#define LED1_PIN 7
#define LED2_PIN 6
#define BUTTON1_PIN 8
#define BUTTON_INC_PIN 9
#define BUTTON_DEC_PIN 10
#define BUZZER_PIN 11

volatile bool led1State = false;
volatile int blinkInterval = 1000;
SemaphoreHandle_t buttonSemaphore;
SemaphoreHandle_t reportSemaphore;
```

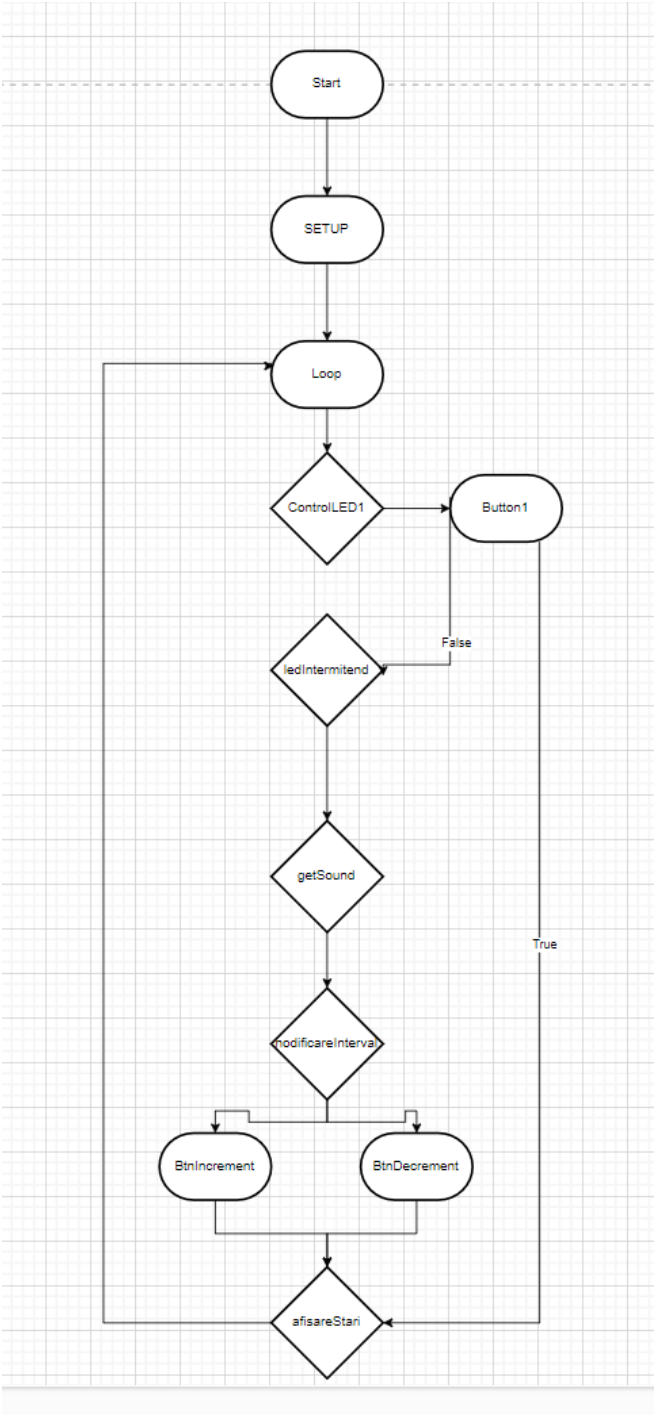
```
void ButtonLedTask(void *pvParameters) {
  pinMode(LED1_PIN, OUTPUT);
  pinMode(BUTTON1_PIN, INPUT_PULLUP);
  while (1) {
    if (digitalRead(BUTTON1_PIN) == LOW) {
      delay(50);  // Debounce simplu
      if (digitalRead(BUTTON1_PIN) == LOW) {
        led1State = !led1State;
        digitalWrite(LED1_PIN, led1State);
        xSemaphoreGive(reportSemaphore);
        delay(300);
      }
    }
    vTaskDelay(10 / portTICK_PERIOD_MS);
  }
}

void BlinkLedTask(void *pvParameters) {
  pinMode(LED2_PIN, OUTPUT);
  unsigned long lastBlinkTime = 0;
  while (1) {
    if (!led1State) {
      unsigned long currentTime = millis();
      if (currentTime - lastBlinkTime >= blinkInterval) {
        lastBlinkTime = currentTime;
        bool led2State = !digitalRead(LED2_PIN);
        digitalWrite(LED2_PIN, led2State);

        if (led2State) {
```

```
          tone(BUZZER_PIN, 1000);
        } else {
          noTone(BUZZER_PIN);
        }
      }
    } else {
      digitalWrite(LED2_PIN, LOW);
      noTone(BUZZER_PIN);
    }
    vTaskDelay(10 / portTICK_PERIOD_MS);
  }
}

void VariableControlTask(void *pvParameters) {
  pinMode(BUTTON_INC_PIN, INPUT_PULLUP);
  pinMode(BUTTON_DEC_PIN, INPUT_PULLUP);
  while (1) {
    if (digitalRead(BUTTON_INC_PIN) == LOW) {
      delay(50);
      if (digitalRead(BUTTON_INC_PIN) == LOW) {
        blinkInterval += 100;
        xSemaphoreGive(reportSemaphore);
        delay(300);
      }
    }
    if (digitalRead(BUTTON_DEC_PIN) == LOW) {
      delay(50);
      if (digitalRead(BUTTON_DEC_PIN) == LOW && blinkInterval > 100) {
        blinkInterval -= 100;
        xSemaphoreGive(reportSemaphore);
```

```
        delay(300);
      }
    }
    vTaskDelay(10 / portTICK_PERIOD_MS);
  }
}

void ReportTask(void *pvParameters) {
  while (1) {
    if (xSemaphoreTake(reportSemaphore, portMAX_DELAY) == pdTRUE) {
      Serial.print("LED 1: ");
      Serial.println(led1State ? "Pornit" : "Oprit");
      Serial.print("LED 2 Blink Interval: ");
      Serial.println(blinkInterval);
    }
    vTaskDelay(500 / portTICK_PERIOD_MS);
  }
}
```






Start
SETUP
Loop
ControlLED1
Button1
False
ledIntermitend
getSound
True
modificareInterval
BtnIncrement
BtnDecrement
afisareStari