Universitatea Tehnică a Moldovei

Facultatea Calculatoare Informatică şi Microelectronică

Departamentul Ingineria Software și Automatică

**RAPORT**

Lucrarea de laborator nr. 2.2

La disciplina “Internetul Lucrurilor”

**Tema:** **Sisteme de Operare – FreeRtos**

A efectuat: st. gr. SI-211 Adrian Chihai

A verificat: Valentina Astafi

**Chișinău – 2024**

**Definire Problema**

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.

Indicații pentru implementare

1. sa se implementeze comunicarea intre Taskuri ca provider consumer, adica:

* - task-ul care generează date, provider, stochează rezultatele într-o variabila globala/semnal, **instaleaza un semafor SemaforGive**
* - task-ul care utilizează aceste date, consumer, citește aceasta variabila/semnal, **reseteaza semaforul**

*de ex: task de UI (LCD sau Serial) preia informația din niște variabile-semnale globale și raportează*

 2. A se urma principiile prezentate la curs Sisteme Secvențiale pentru recurenta si offset

* stabilirea rezonabila a recurentei  pentru a diminua incarcarea procesorului - prin **xTaskDelayUntil**( delay) **in** bucla infinita.
* stabilirea ofsetului, intru a activa în ordinea cuvenita task urile - prin **xTaskDelay**( delay) **inainte** de bucla infinita.
* pentru **xTaskDelay** se utiliza referinta: https://www.freertos.org/a00127.html

 3: Task-ul de raportare pentru Secvențial cu utilizare STDIO printf() catre LCD va fi rulat in in FreeRtos - un task separat.

# Procesul efectuării lucrării

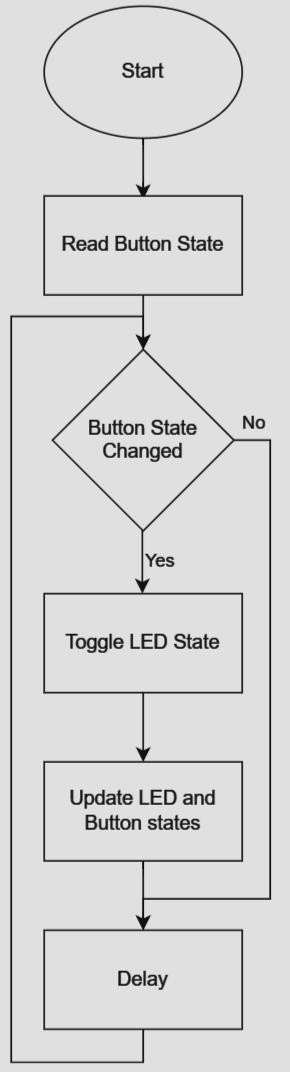
Progresul implementării codului FreeRTOS depinde de o înțelegere aprofundată a conceptelor sale și de gestionarea eficientă a resurselor. Sarcinile trebuie structurate cu claritate, fiecare îndeplinind o funcție specifică, de la comutarea LED-urilor la gestionarea intrărilor de butoane. Prioritizarea corespunzătoare a sarcinilor asigură executarea în timp util a funcțiilor critice.

Când am început lucrările de laborator, m-am concentrat mai întâi pe inițializarea componentelor hardware necesare și pe configurarea cronometrului pentru a asigura o sincronizare precisă. Apoi, am implementat bucle de sarcini pentru a gestiona funcționalitatea butoanelor și a LED-urilor. Ulterior, am integrat o funcționalitate pentru a ajusta intervalul de clipire al unuia dintre LED-uri pe baza apăsării butonului, permițând un control dinamic asupra comportamentului acestuia. În timpul perioadelor de inactivitate, am configurat mecanisme de monitorizare pentru a detecta și raporta orice modificare a stării LED-urilor, a apăsării butoanelor și a intervalelor de clipire.

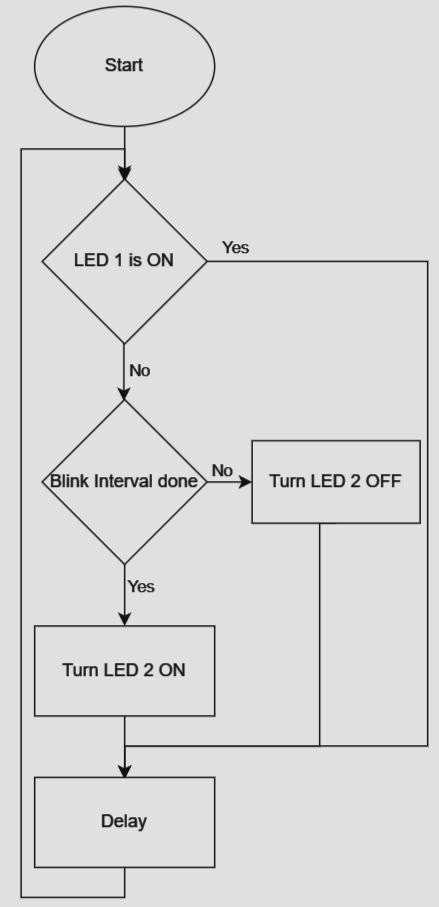
FreeRTOS permite crearea de sarcini multiple care pot rula simultan. În funcția setup(), am creat patru sarcini utilizând funcția xTaskCreate(). Fiecare sarcină are propria prioritate, dimensiune a stivei și funcție de intrare. Planificatorul FreeRTOS determină ce sarcină ar trebui să ruleze la un moment dat pe baza priorităților lor. Sarcinile cu prioritate mai mare vor prelua sarcinile cu prioritate mai mică atunci când acestea sunt gata de execuție.

# Diagrama bloc

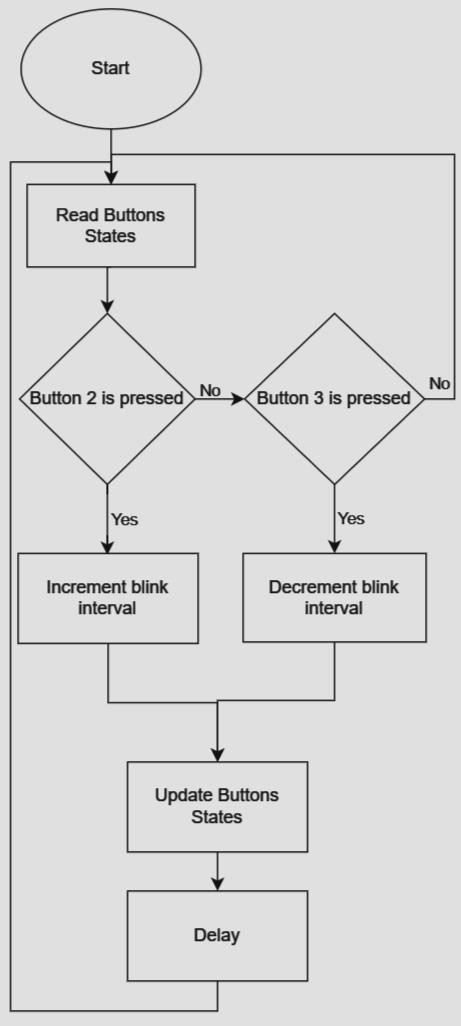
**Figura 1. Diagrama bloc a programului**



**Figura 2. Diagrama bloc pentru taskul 1**

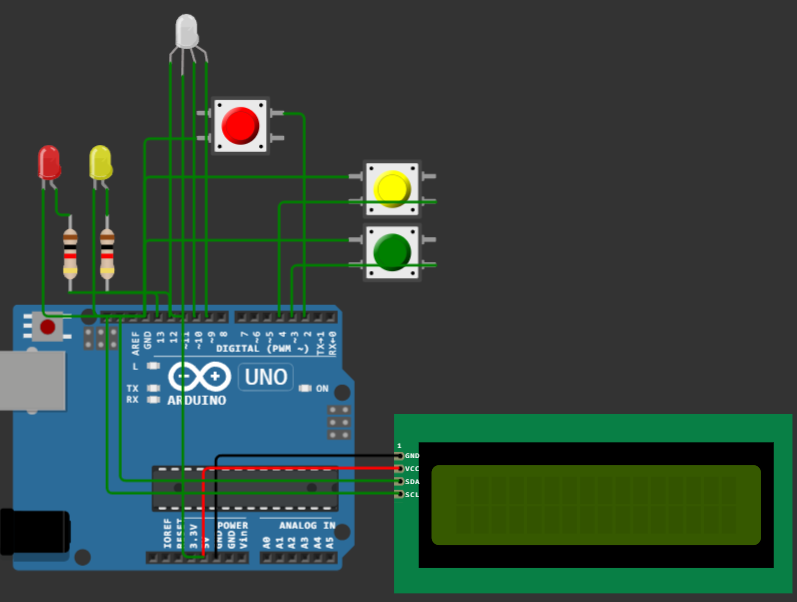


**Figura 3. Diagrama block pentru taskul 2**



**Figura 4. Diagrama block pentru task-ul 3**

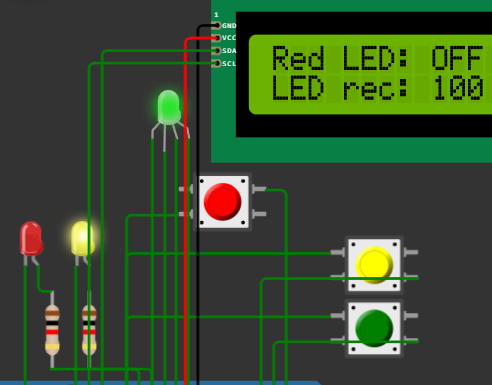
# Schema electrică

Pentru a simula schema electrică pentru acest proiect, am început prin a plasa o placă Arduino Uno în centru. Am conectat apoi două LED-uri, unul roșu și unul galben, la rezistențe de 1kΩ. În plus, am adăugat trei butoane și un led RGB suplimentar. Conexiunile la masă pentru butoane au fost realizate la pinul de masă (GND) al Arduino. În cele din urmă, am conectat catodul fiecărui LED la pinul respectiv de masă al Arduino pentru a finaliza circuitul. Această schemă asigură interacțiunea corectă între placa Arduino, LED-urile și butoanele din proiect.****

**Figura 5. Circuitul electric**

# Simularea circuitului

In imaginea de mai jos putem observa o simulare a circuitului:



**Figura 6. Rularea circuitului**

# Concluzie

Acest proiect a presupus proiectarea unei aplicații MCU utilizând FreeRTOS pentru a gestiona cel puțin trei sarcini secvențiale. Aceste sarcini au inclus comutarea unui LED ca răspuns la apăsarea unui buton, activarea unui LED secundar intermitent atunci când primul LED era oprit și ajustarea intervalului de intermitență a LED-ului prin intermediul intrărilor de buton. În plus, am implementat o sarcină inactiv pentru a gestiona stările programului, cum ar fi afișarea stării LED-urilor și gestionarea mesajelor pentru apăsarea butoanelor, utilizând un mecanism furnizor/consumator. Integrarea FreeRTOS a vizat crearea unui mediu multitasking eficient pentru MCU.

Prin intermediul acestui proiect, am dobândit experiență practică în gestionarea sarcinilor, alocarea resurselor și gestionarea evenimentelor în timp real. Gestionarea mai multor sarcini m-a ajutat să aprofundez înțelegerea multitasking-ului în sistemele integrate, în timp ce sarcina de inactivitate a evidențiat flexibilitatea FreeRTOS în gestionarea stărilor și interacțiunilor sistemului. Acest laborator a oferit o perspectivă valoroasă asupra dezvoltării aplicațiilor RTOS, oferindu-mi competențe esențiale pentru viitoarele proiecte care necesită multitasking în sistemele integrate..

# Anexa A

#include <Arduino.h>

#include <Wire.h>

#include <stdio.h>

#include <Arduino\_FreeRTOS.h>

#include <LiquidCrystal\_I2C.h>

#include <task.h>

#include <semphr.h>

#include "LED.h"

#include "Button.h"

#define Task\_Delay 15

#define LED\_RED 13 // Red LED

#define BUTTON\_RED 2 // Button for Red LED

#define BUTTON\_INC 3 // Button for increasing the recurrency

#define BUTTON\_DEC 4 // Button for decreasing the recurrency

#define LED\_YELLOW 12 // Blinking LED

#define LCD\_COL 16

#define LCD\_ROW 2

#define LED\_ON HIGH

#define LED\_OFF LOW

#define BUTTON\_PRESSED HIGH

#define BUTTON\_RELEASED LOW

#define RECURRENCY\_MIN 100

#define RGBRED\_LED 11

#define GREEN\_LED 10

#define BLUE\_LED 9

const int i2c\_addr = 0x27; // I2C address of the LCD

int led\_state = 0; // 0: off, 1: on RED

int blinkled\_state = 0; // 0: off, 1: on BLINK

int button\_state = LOW;

int lastButtonState = LOW;

int led\_rec = 100;

int recurrency\_number = 100;

int redLEDState;

// LCD object

LiquidCrystal\_I2C lcd(i2c\_addr, LCD\_COL, LCD\_ROW);

SemaphoreHandle\_t dataSemaphore;

// function prototypes

void LCD\_setup();

int get\_led\_state(int pin);

void blinking\_led(void \*pvParameters);

void red\_led\_state(void \*pvParameters);

void blink\_counter(void \*pvParameters);

void state\_output(void \*pvParameters);

void schedule\_funcs();

void draw(void \*pvParameters);

// LEDs

Led red\_led(LED\_RED);

Led yellow\_led(LED\_YELLOW);

Led rgbred\_led(RGBRED\_LED);

Led rgbgreen\_led(GREEN\_LED);

Led rgbblue\_led(BLUE\_LED);

// Buttons

Button button\_red(BUTTON\_RED);

Button button\_inc(BUTTON\_INC);

Button button\_dec(BUTTON\_DEC);

void setup()

{

LCD\_setup();

dataSemaphore = xSemaphoreCreateBinary();

schedule\_funcs();

vTaskStartScheduler();

}

void loop() {}

void schedule\_funcs()

{

xTaskCreate(red\_led\_state, "Button Red LED", 128, NULL, 1, NULL);

xTaskCreate(blinking\_led, "LED Blink", 128, NULL, 3, NULL);

xTaskCreate(blink\_counter, "Count Blinks", 128, NULL, 2, NULL);

xTaskCreate(state\_output, "Output State", 128, NULL, 4, NULL);

xTaskCreate(draw, "Draw", 128, NULL, 5, NULL);

// // xTaskCreate(

// timer\_task, // Function that should be called

// "Timer Task", // Name of the task (for debugging)

// 128, // Stack size (words)

// NULL, // Parameter to pass

// 1, // Task priority

// NULL // Task handle

// );

}

// change RED LED state

void red\_led\_state(void \*pvParameters)

{

(void)pvParameters;

for (;;)

{

button\_state = button\_red.getState();

if (button\_state != lastButtonState)

{

if (!button\_red.isPressed())

{

led\_state = !led\_state;

if (led\_state == LED\_ON)

{

red\_led.on();

yellow\_led.off();

}

else

{

red\_led.off();

yellow\_led.on();

}

xSemaphoreGive(dataSemaphore);

}

lastButtonState = button\_state;

}

vTaskDelay(Task\_Delay);

}

}

// second blinking led

void blinking\_led(void \*pvParameters)

{

(void)pvParameters;

for (;;)

{

if (xSemaphoreTake(dataSemaphore, portMAX\_DELAY))

{

if (led\_state == LED\_OFF) // Red LED is off

{ // make it blink

if (blinkled\_state == LED\_OFF) // if the state is off

{

yellow\_led.on(); // turn on the LED if it is off

blinkled\_state = LED\_ON; // set the state to on

}

else

{

yellow\_led.off(); // turn off the LED if it is on

blinkled\_state = LED\_OFF; // set the state to off

}

}

}

vTaskDelay(led\_rec / portTICK\_PERIOD\_MS);

}

}

// modify the recurrency number of the blinking led

void blink\_counter(void \*pvParameters)

{

(void)pvParameters;

int lastButtonIncState = BUTTON\_RELEASED; // Initialize last state of inc button

int lastButtonDecState = BUTTON\_RELEASED; // Initialize last state of dec button

for (;;)

{

int button\_inc\_state = button\_inc.getState();

int button\_dec\_state = button\_dec.getState();

if (button\_inc\_state == BUTTON\_PRESSED && lastButtonIncState == BUTTON\_RELEASED)

{

led\_rec += recurrency\_number;

lastButtonIncState = BUTTON\_PRESSED; // Update last state of inc button

}

else if (button\_inc\_state == BUTTON\_RELEASED)

{

lastButtonIncState = BUTTON\_RELEASED; // Update last state of inc button

}

if (button\_dec\_state == BUTTON\_PRESSED && lastButtonDecState == BUTTON\_RELEASED && led\_rec > RECURRENCY\_MIN)

{

led\_rec -= recurrency\_number;

lastButtonDecState = BUTTON\_PRESSED; // Update last state of dec button

}

else if (button\_dec\_state == BUTTON\_RELEASED)

{

lastButtonDecState = BUTTON\_RELEASED; // Update last state of dec button

}

vTaskDelay(Task\_Delay);

xSemaphoreGive(dataSemaphore);

}

}

void draw(void \*pvParameters){

(void)pvParameters;

for (;;)

{

led\_color(0,255,255);

delay(1000);

led\_color(255,0,255);

delay(1000);

led\_color(255,255,0);

delay(1000);

vTaskDelay(50);

}

}

// output state

void state\_output(void \*pvParameters)

{

(void)pvParameters;

for (;;)

{

lcd.setCursor(0, 0);

lcd.print("Red LED: ");

lcd.print(led\_state == LED\_ON ? "ON " : "OFF");

lcd.setCursor(0, 1);

lcd.print("LED rec: ");

lcd.print(led\_rec);

lcd.print(" ");

vTaskDelay(1000 / portTICK\_PERIOD\_MS);

}

}

void led\_color(int red\_value, int green\_value,int blue\_value)

{

analogWrite(RGBRED\_LED,red\_value);

analogWrite(GREEN\_LED,green\_value);

analogWrite(BLUE\_LED,blue\_value);

}

// setup LCD

void LCD\_setup()

{

lcd.begin(LCD\_COL, LCD\_ROW, 0x00);

lcd.backlight();

}

# BIBLIOGRAPHY

1. xTaskCreate, TASK CREATION, [Quoted: 10.03.2024], acces link: https://[www.freertos.org/a00125.html](http://www.freertos.org/a00125.html)
2. vTaskDelay, TASK CONTROL, [Quoted: 10.03.2024], acces link: https://[www.freertos.org/a00127.html](http://www.freertos.org/a00127.html)
3. Basic exemple of FreeRTOS with Arduino, void loop Robotech & Automation*,* [Quoted: 9.03.2024], acces link https://[www.youtube.com/watch?v=BuRGD3x-](http://www.youtube.com/watch?v=BuRGD3x-) QDM&list=PLOYsAys6a6mmoyI2l440Wm5JwYhmtci8g&index=2