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.

### 1.1 Diagrama bloc

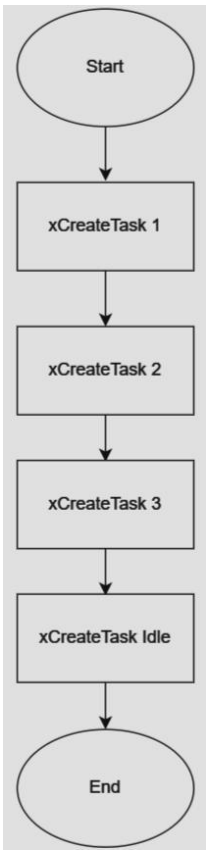


**Figura 1. Diagrama bloc a programului**

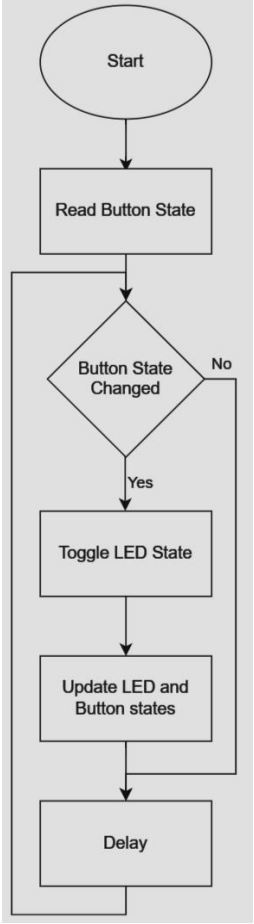


**Figura 2. Diagrama bloc pentru taskul 1**

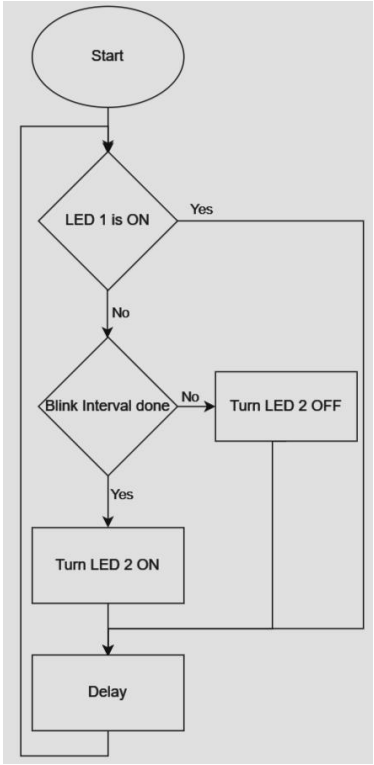


**Figura 3. Diagrama block pentru taskul 2**

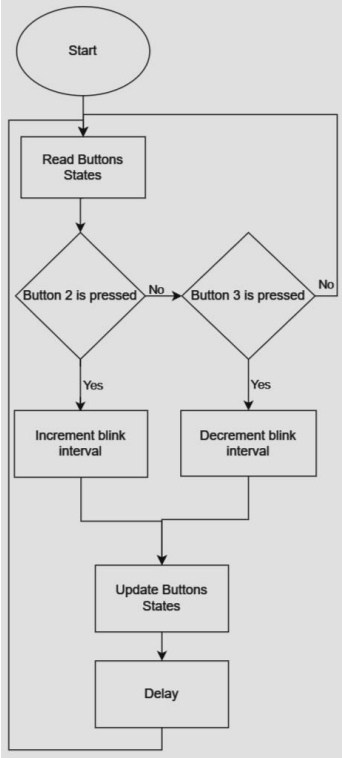


**Figura 4. Diagrama block pentru task-ul 3**

### 1.2 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.

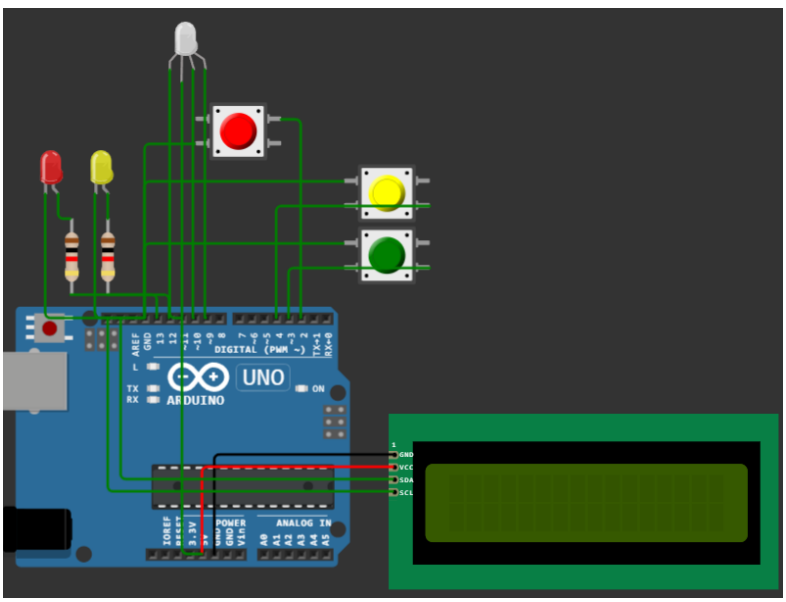


**Figura 5. Circuitul electric**

### 1.3 Simularea circuitului

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

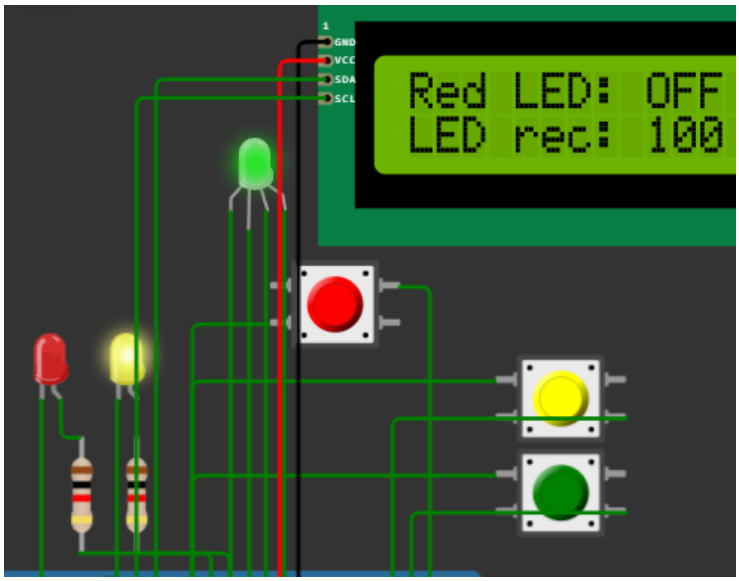


**Figura 6. Rularea circuitului**

**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

2. vTaskDelay, TASK CONTROL, [Quoted: 10.03.2024], acces link: https://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-QDM&list=PLOYsAys6a6mmoyI2l440Wm5JwYhmtci8g&index=2