Universitatea Tehnică a Moldovei
Facultatea Calculatoare Informatică şi Microelectronică
Departamentul Ingineria Software și Automatică

RAPORT

Lucrarea de laborator nr. 2.1
La disciplina “Internetul Lucrurilor”
Tema: Sisteme de Operare - Secventiale

A efectuat: st. gr. SI-211 Chirita Stanislav
A verificat: Valentina Astafi

Chișinău – 2024

Lucrare de laborator nr. 2.2

Codul din Anexa 1 este un exemplu de utilizare a sistemului de operare în timp real FreeRTOS pe o placă Arduino. Se descrie în mod general fiecare parte și metodă pentru a înțelege funcționarea codului.

Declarații și Inițializări:

- Se definesc pinii și variabilele utilizate în program (BUTTON_PIN, BUTTON_LED_INCR, BUTTON_LED_DECR, LED_PIN, LED_BLINK_PIN, etc.).

- Se definește și se inițializează variabilele de stare pentru LED-uri și butoane (ledState, led2State, buttonState, etc.).

Task-ul taskLedWithButton

- Acest task citeste starea unui buton și controlează starea unui LED în funcție de apăsarea butonului.

- Dacă butonul este apăsat, se inversează starea LED-ului și se actualizează pinul LED-ului.

- Dacă LED-ul este deja aprins, se oprește LED-ul de blink asociat altui pin.

Task-ul taskLedBlink

- Acest task controlează blink-ul unui al doilea LED în funcție de starea unui alt LED și un delay specificat (timeDelay).

- Dacă primul LED este stins, se controlează blink-ul celui de-al doilea LED în funcție de delay-ul specificat.

Task-ul taskButton23

- Acest task monitorizează butoanele pentru a ajusta delay-ul utilizat pentru blink-ul celui de-al doilea LED.

- Dacă butoanele sunt apăsate, delay-ul (timeDelay) este modificat și se afișează pe portul serial.

Metoda setup

- Configurează pinii ca ieșiri sau intrări în funcție de necesități.

- Inițializează portul serial pentru comunicarea cu computerul.

Metoda loop: Nu este utilizată în mod obișnuit în FreeRTOS, dar aici se apelează vTaskDelete(NULL) pentru a preveni crearea unui loop infinit.

Definirea mânerelor de sarcini:

TaskHandle_t ledWithButtonTaskHandle;

TaskHandle_t ledBlinkTaskHandle;

TaskHandle_t button23TaskHandle;

Se definesc mânerele pentru sarcinile create cu FreeRTOS.

Crearea task-ului: **xTaskCreate(taskLedWithButton, "LedWithButtonTask", 100, NULL, 1, &ledWithButtonTaskHandle);**

taskLedWithButton:

- Este funcția pe care o dorim să fie executată de această sarcină. În acest caz, este funcția taskLedWithButton.

"LedWithButtonTask":

- Este un șir de caractere care reprezintă numele sarcinii.

- Acest nume este util în scopuri de diagnosticare și urmărire pentru a identifica sarcina în sistemul de operare în timpul execuției.

100:

- Reprezintă dimensiunea stivei pentru sarcina creată, exprimată în cuvinte (de regulă, cuvinte de 4 octeți pentru Arduino).

NULL:

- Este parametrul de inițializare, dar în acest caz, este setat la NULL pentru că nu avem nevoie de un parametru specific pentru această sarcină.

1:

- Este prioritatea atribuită sarcinii.

- Cu cât numărul este mai mare, cu atât prioritatea este mai mică. Sarcinile cu prioritate mai mare rulează înaintea celor cu prioritate mai mică.

&ledWithButtonTaskHandle:

- Este adresa unei variabile de tip TaskHandle_t în care funcția xTaskCreate() va stoca mânerul (handle) pentru sarcină.

- Acest mâner (handle) poate fi folosit pentru a gestiona și monitoriza sarcina creată.

În ansamblu, acest cod folosește FreeRTOS pentru a gestiona sarcinile concurente (task-urile) care controlează LED-urile și interacționează cu butoanele, permițând astfel un comportament reactiv și sincronizat al dispozitivului Arduino.

Sistemul în real

În programarea secvențială, codul rulează într-o singură linie de execuție. Adică, instrucțiunile sunt executate una după alta, într-o secvență liniară.Nu există gestionare a sarcinilor sau a concurenței.Programele secvențiale sunt simplu de scris și de înțeles, dar pot să nu utilizeze eficient resursele hardware, în special în sistemele cu mai multe nuclee de procesor.

FreeRTOS este un sistem de operare în timp real care oferă funcționalități avansate pentru gestionarea sarcinilor concurente.În FreeRTOS, puteți crea mai multe task-uri care rulează în paralel, fiecare cu propriile lor priorități și resurse alocate.FreeRTOS gestionează programarea și sincronizarea sarcinilor, oferind control și planificare mai granulară.

Cu toate acestea, utilizarea FreeRTOS necesită o învățare suplimentară și o mai mare complexitate în comparație cu programarea secvențială, deoarece impune gestionarea concurenței și a planificării. Alegerea între cele două abordări depinde de cerințele specifice ale proiectului și de complexitatea acestuia. În cazul aplicație cu cerințe de gestionare a butoanelor și LED-urilor, utilizarea FreeRTOS ar putea aduce beneficii semnificative pentru gestionarea evenimentelor concurente și a responsivității sistemului.

In figura 1 este prezentata schema circuitului

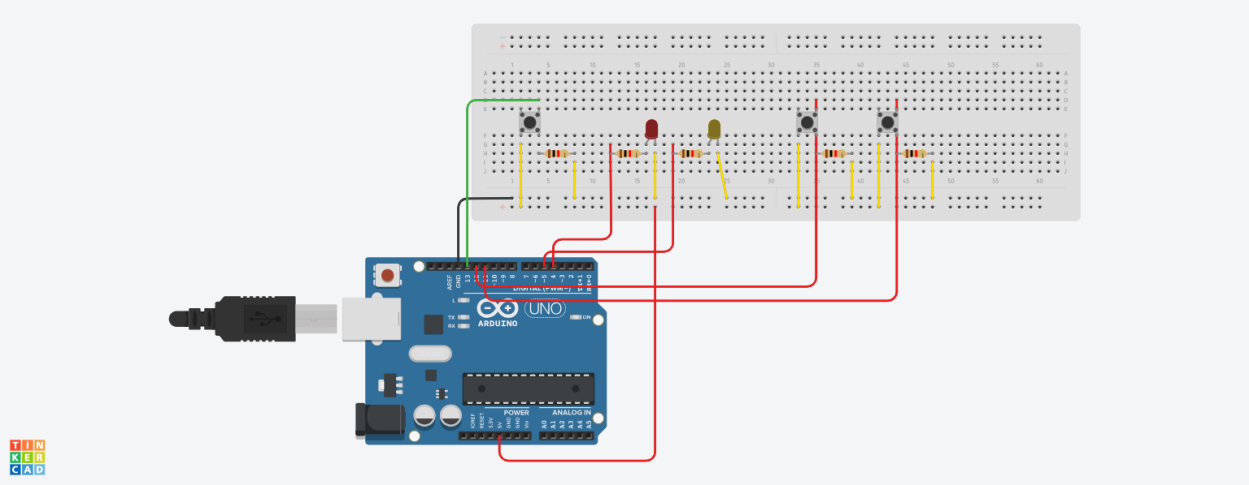

Figura 1 – Schema circuitului

Concluzie:

La efectuarea acestui laborator s-a abordat subiectul dezvoltarii și implementarii sistemelor de operare secvențiale și utilizarea FreeRTOS ,pentru gestionarea sarcinilor a oferit oportunitatea de a înțelege în profunzime conceptele și tehnologiile fundamentale legate de sistemele de operare încorporate. Această experiență a relevat importanța utilizării unui sistem de operare în dezvoltarea aplicațiilor pentru dispozitive încorporate, precum și beneficiile aduse de un planificator de tip preemptive precum FreeRTOS.

Anexa 2. Cod cu FreeRTOS

```
#include <Arduino_FreeRTOS.h>

#include <stdio.h>


#define BUTTON_PIN 13

#define BUTTON_LED_INCR 12

#define BUTTON_LED_DECR 11

#define LED_PIN 4

#define LED_BLINK_PIN 5


int led2Rec = 0; // Variabilă pentru a stoca starea LED-ului 2 din task-ul button23

int ledState = 0; // Variabilă pentru a stoca starea LED controlată de apăsarea butonului

int led2State = 0; // Variabilă pentru a stoca starea LED-ului 2 controlată de task-ul de blink al LED-ului

int buttonState = 0; // Starea curentă a butonului

int lastButtonState = 1; // Starea anterioară a butonului

int timeDelay = 500; // Intervalul de întârziere pentru intervalul de clipire a LED-ului


TaskHandle_t ledWithButtonTaskHandle;

TaskHandle_t ledBlinkTaskHandle;

TaskHandle_t button23TaskHandle;


void taskLedWithButton(void* pvParameters)

{

     (void)pvParameters;


     for (; ; )

     {

          buttonState = digitalRead(BUTTON_PIN);


          // Verifică dacă butonul a fost apăsat și eliberat
```

```
          if (buttonState == 1 && lastButtonState == 0)
          {
               ledState = !ledState; // Inversează starea LED-ului
               digitalWrite(LED_PIN, ledState); // Actualizează starea LED
          }

          if (ledState == 1)
          {
               digitalWrite(LED_BLINK_PIN, LOW); // Oprește LED-ul 2 când LED-ul 1 este pornit
          }

          lastButtonState = buttonState; // Stochează starea curentă a butonului pentru
următoarea iterație

          vTaskDelay(10); // Întârziere pentru a evita înfometarea task-ului
     }
}

void taskLedBlink(void* pvParameters)
{
     (void)pvParameters;

     static unsigned long previousBlinkTime = 0;

     for (; ; )
     {
          if (ledState == 0)
          { // Verifică dacă LED-ul 1 este stins
               unsigned long currentTime = millis();
               if (currentTime - previousBlinkTime >= timeDelay)
               {
```

```
                    previousBlinkTime = currentTime;

                    if (led2State == 0)
                    {
                         led2State = 1;
                         digitalWrite(LED_BLINK_PIN, HIGH); // Porniți LED-ul 2
                    }
                    else
                    {
                         led2State = 0;
                         digitalWrite(LED_BLINK_PIN, LOW); // Oprește LED-ul 2
                    }
               }
          }

          vTaskDelay(10); // Întârziere pentru a evita înfometarea task-ului
     }
}

void taskButton23(void* pvParameters)
{
     (void)pvParameters;

     for (; ; )
     {
          if (timeDelay >= 400)
          {
               if (digitalRead(BUTTON_LED_INCR) == 1)
               {
                    timeDelay = timeDelay + 100; // Crește intervalul de clipire a LED-ului
                    Serial.print("Butonul pentru incrementare a fost apăsat! ");
```

```
                    Serial.println(timeDelay); // Afișează intervalul actualizat
                }
                if (digitalRead(BUTTON_LED_DECR) == 1 && timeDelay > 400)
                {
                    timeDelay = timeDelay - 100; // Scade intervalul de clipire a LED-ului (cu
o valoare minimă de 400)
                    Serial.print("Butonul pentru decrementare a fost apăsat! ");
                    Serial.println(timeDelay); // Afișează intervalul actualizat
                }
            }

            vTaskDelay(10); // Întârziere pentru a evita înfometarea task-ului
        }
}

void setup()
{
    pinMode(LED_PIN, OUTPUT);
    pinMode(LED_BLINK_PIN, OUTPUT);
    pinMode(BUTTON_PIN, INPUT);
    pinMode(BUTTON_LED_DECR, INPUT);
    pinMode(BUTTON_LED_INCR, INPUT);
    Serial.begin(9600); // Inițializează comunicarea Serial

    // Creează trei task-uri pentru controlul LED-urilor și intrarea butonului
    xTaskCreate(taskLedWithButton, "LedWithButtonTask", 100, NULL, 1,
&ledWithButtonTaskHandle);
    xTaskCreate(taskLedBlink, "LedBlinkTask", 100, NULL, 2, &ledBlinkTaskHandle);
    xTaskCreate(taskButton23, "Button23Task", 100, NULL, 3, &button23TaskHandle);
}

void loop()
```

```
{

     // Loop-ul principal nu face nimic, deoarece toate activitățile sunt gestionate de task-
uri

}
```