Ministry of Education and Research of the Republic of Moldova

Technical University of Moldova

Department of Software and Automation Engineering

REPORT

Laboratory work No. 2.2

Discipline: Embedded Systems

Student: Revenco Victor, FAF - 221

Checked: asist. univ., Martiniuc A.

Chișinău 2025

Analysis of the Situation in the Field

1. Description of the Technologies Used and Application Context

This lab implements a modular application for a microcontroller using FreeRTOS to demonstrate real-time scheduling, synchronization, and execution of multiple tasks. The system addresses a common embedded systems scenario where:

1. A physical input (button press) triggers a system event
2. The event needs to be processed and communicated to other parts of the system
3. Multiple operations need to occur with specific timing requirements
4. Resources must be shared efficiently between multiple processes

Hardware Components

- **Microcontroller**
- **LEDs**
- **Resistors (220Ω)**
- **Push Buttons**
- **Breadboard**
- **Jumper Wires**
- **USB Power Supply**

Software Components

- **PlatformIO with Visual Studio Code**: Integrated Development Environment (IDE)
- **C++ for Embedded Systems**: Programming language used for implementation
- **STDIO Functions (printf)**: For system reporting and monitoring
- **Serial Monitor**: For displaying system status

2. System Architecture Explanation and Solution Justification

The solution follows a modular architecture with clearly defined responsibilities for each component:

The system is organized into three main tasks:

1. **ButtonLedTask (10ms recurrence)**:
   - Monitors the button state
   - Controls the first LED (turns on for 1 second after button press)
   - Signals the synchronous task via semaphore

    - Ensures precise timing using `xTaskDelayUntil()`
2. **SynchronousTask (50ms recurrence)**:
    - Waits for semaphore from ButtonLedTask
    - Increments counter N
    - Generates and sends N bytes to a shared buffer
    - Controls the second LED to blink N times (ON: 300ms, OFF: 500ms)
3. **AsynchronousTask (200ms recurrence)**:
    - Reads from the shared buffer
    - Formats and displays data through UART (serial terminal)

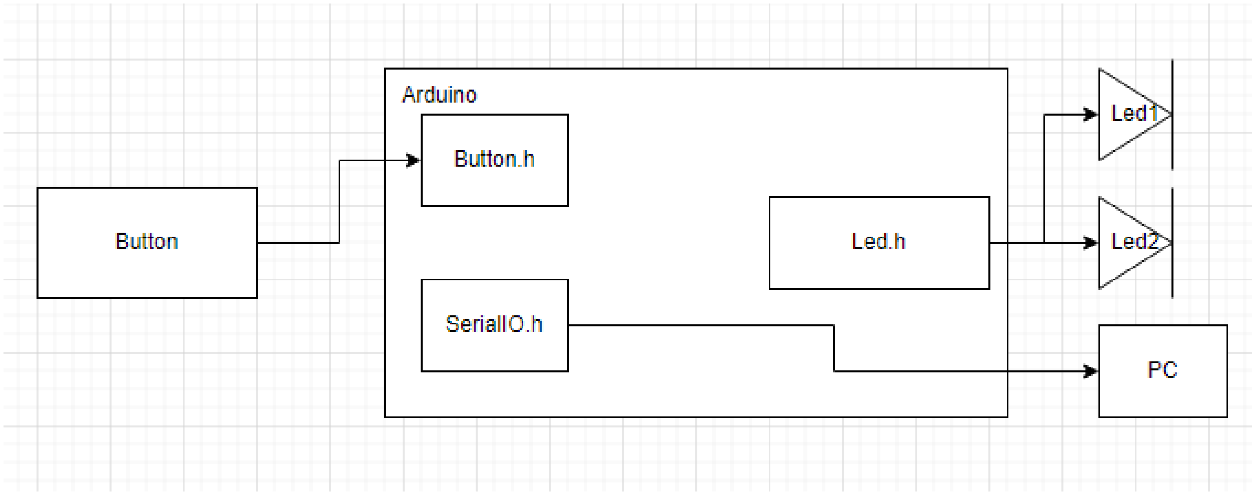
3. Case Study: LED Control in an Industrial Automation System

A home automation company needs to develop a reliable controller for managing multiple home devices. The controller must handle user inputs (physical and remote), process sensor data, and control various appliances while maintaining responsive performance in all conditions.

4. Design

1. Architectural Sketch and Component Interconnection

- Button connected to digital input pins
- LEDs connected to output pins with resistors
- Microcontroller executing sequential task scheduling
- Serial Monitor for system status output

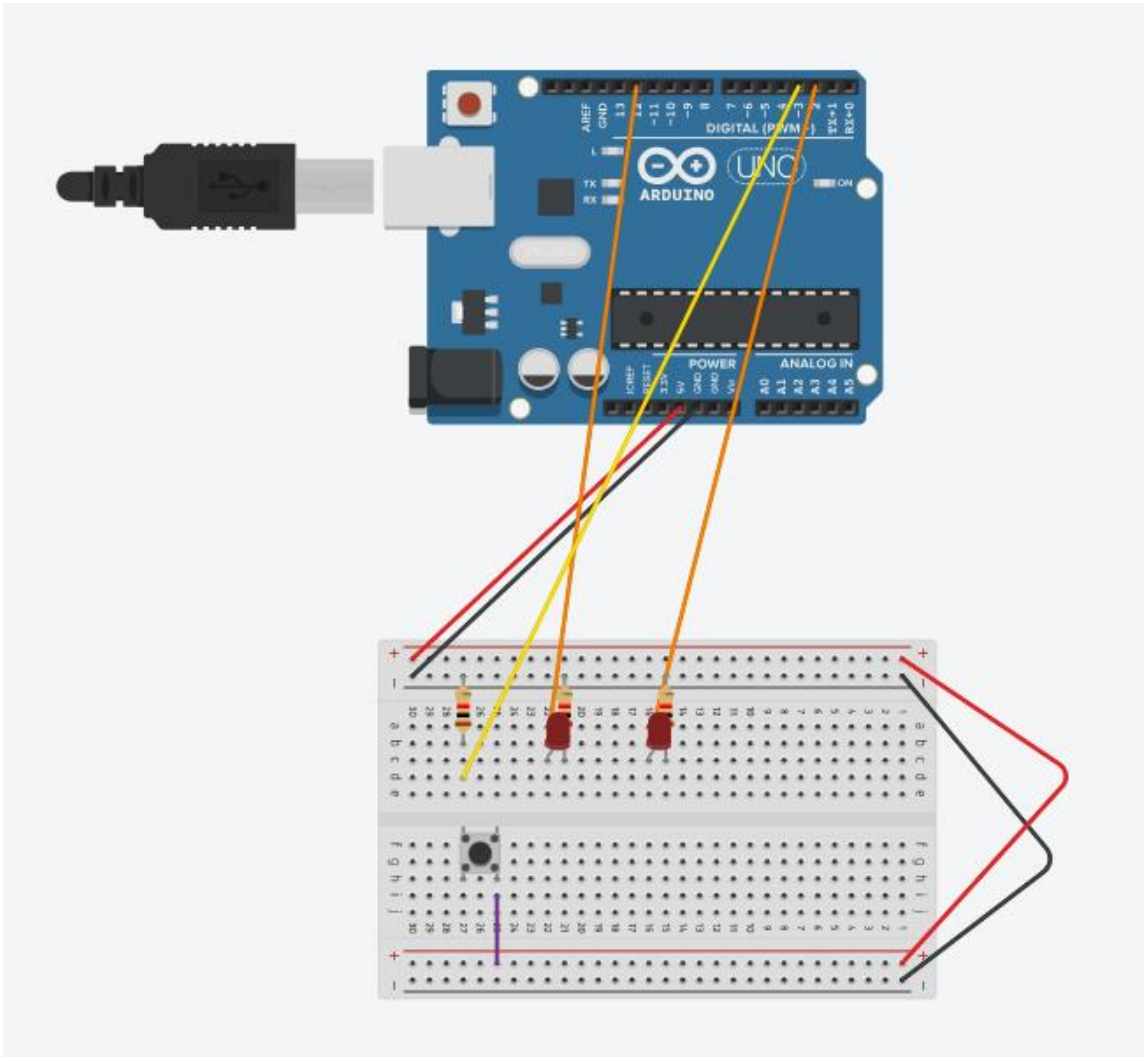


Component Description and Their Roles:

- **Microcontroller**: Core processing unit executing tasks

- **LEDs**: Provide visual feedback for task states
- **Buttons**: User input for system state changes
- **Resistors**: Current limiting for LED protection

The sketch shows how the components interact.



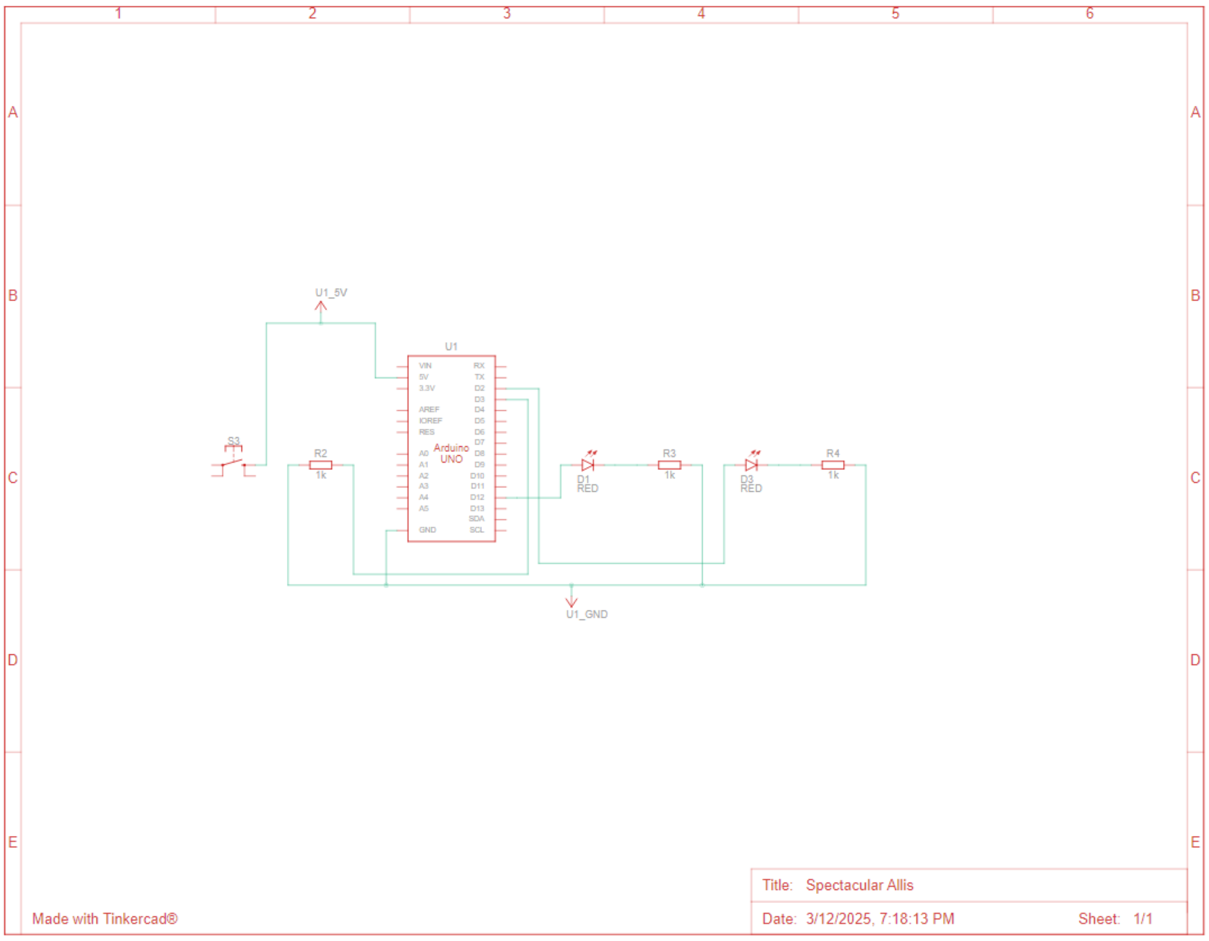


Figure 1, 2 Electrical schematic

This image shows an **Arduino Uno** connected to a breadboard circuit, which includes LEDs, resistors, and a push button. Here's a breakdown of the components and their connections:

1. Microcontroller (Arduino Uno)

- The **Arduino Uno** is the main processing unit, providing power and controlling the circuit.

2. LEDs (Light Emitting Diodes)

- There are **three LEDs** placed on the breadboard.
- Each LED is connected to a **current-limiting resistor** to prevent excessive current.
- The **other end of the resistors is connected to digital output pins** on the Arduino.

3. Push Buttons

- The breadboard includes **three push buttons**. These are used for **user input**, such as toggling LED states or adjusting variables.
- The buttons are connected in a **pull-down configuration**, meaning they pull the input **LOW (0V) when not pressed** and go **HIGH (5V) when pressed**.
- Wires connect the buttons to digital input pins on the Arduino.

4. Power Connections

- The Arduino **5V pin** is connected to the **red power rail** of the breadboard.
- The Arduino **GND (ground) pin** is connected to the **black power rail** of the breadboard.
- The components use these rails to share power.

2. Schematic diagrams

To understand the system's behavior, a Flowchart and a Finite State Machine (FSM) are used.

Flowchart – Serial Command Processing

The FSM includes the following states:

1. Idle – Waits for commands from the user.
2. Processing – Processes the received command.
3. Led on – Turns on the LED.
4. Led off – Turns off the LED.
5. Counter++ – Increments counter.
6. Counter-- - Decrements counter.

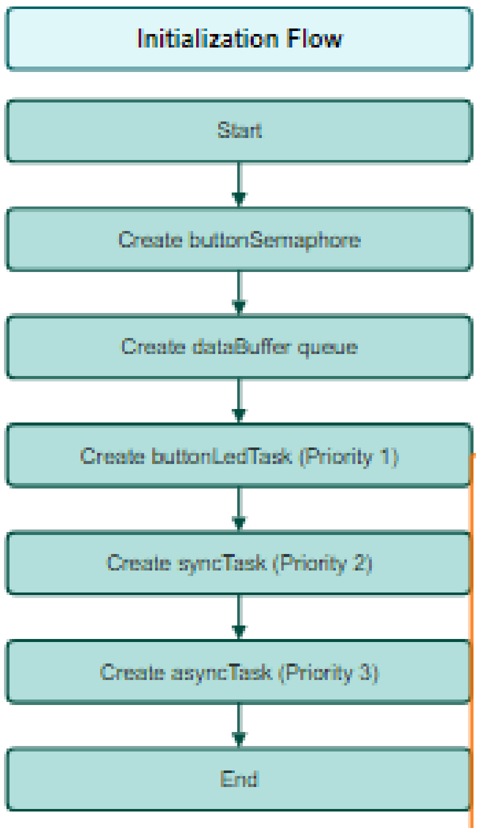

Initialization Flow
Start
Create buttonSemaphore
Create dataBuffer queue
Create buttonLedTask (Priority 1)
Create syncTask (Priority 2)
Create asyncTask (Priority 3)
End

Figure 3 Initialization flow

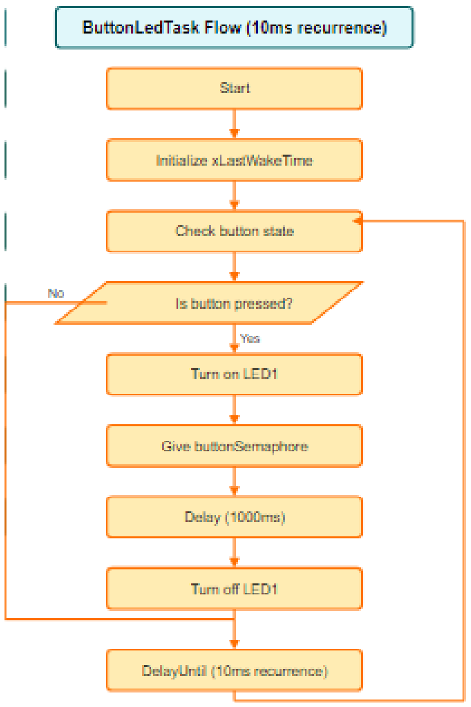


Figure 4 ButtonLedTask

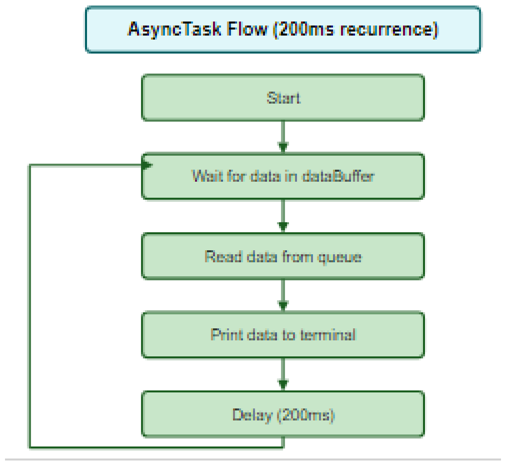

AsyncTask Flow (200ms recurrence)
Start
Wait for data in dataBuffer
Read data from queue
Print data to terminal
Delay (200ms)

Figure 5 AsyncTask

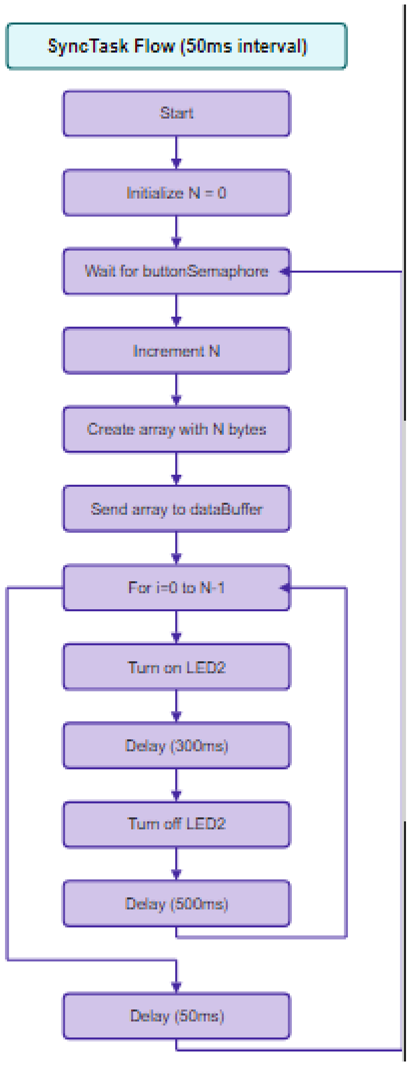


Figure 6 SyncTask

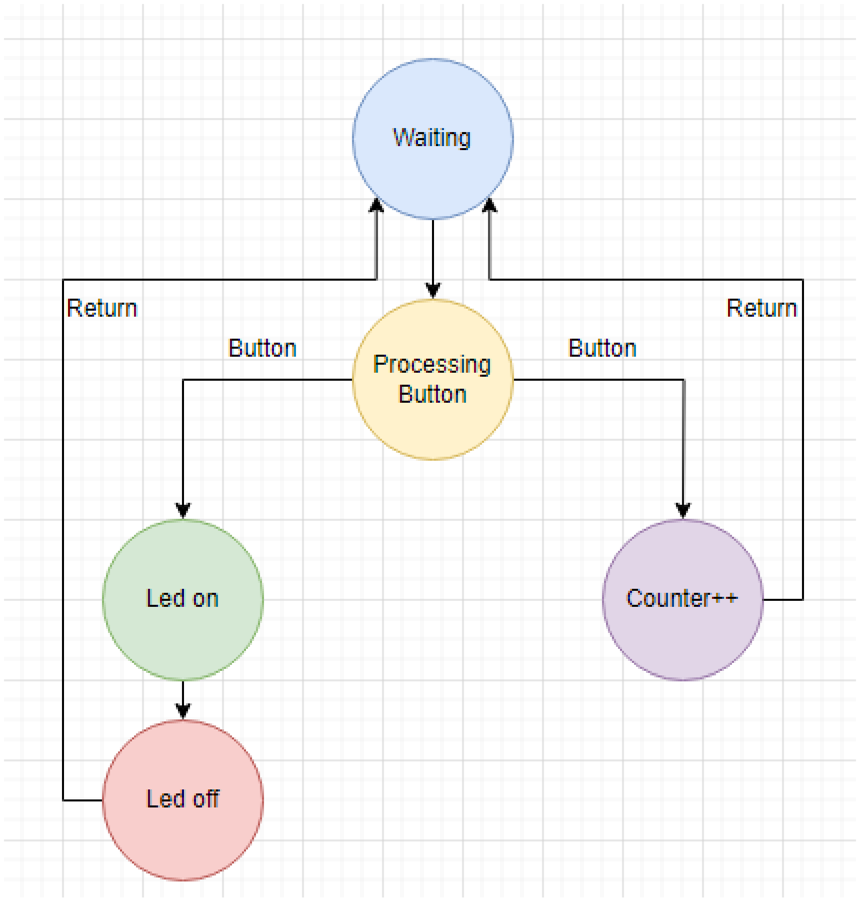
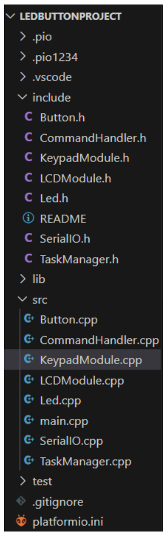


Figure 7 FSM diagram

3. Modular implementation



```
LEDBUTTONPROJECT
  > .pio
  > .pio1234
  > .vscode
  v include
     Button.h
     CommandHandler.h
     KeypadModule.h
     LCDModule.h
     Led.h
     README
     SerialIO.h
     TaskManager.h
  > lib
  v src
     Button.cpp
     CommandHandler.cpp
     KeypadModule.cpp
     LCDModule.cpp
     Led.cpp
     main.cpp
     SerialIO.cpp
     TaskManager.cpp
  > test
  .gitignore
  platformio.ini
```

Figure 8 Project organization

This header file, named *Led.h*, defines the interface for controlling an LED. It declares 3 functions without providing their implementations:

1. toggle() – This function is used to change the state of the LED (State: ON or OFF)
2. turnOn() – This function is used to change the state of the LED to ON
3. turnOff() – This function is used to change the state of the LED to OFF

In the SerialIO.h we have the following methods for reading and writing the text:

1. serial_putc(char c, FILE *stream) – it uses Serial.write to write the text
2. serial_getc(FILE *stream) – it uses Serial.read serial is not available

In the file Led.cpp we implement all methods in the Led.h:

1. toggle() – it changes the state using '!', and writes to the pin the state HIGH if its LOW and vice versa
2. turnOn() – sets the state to true and sets the pin to HIGH
3. turnOff() – sets the state to false and sets the pin to LOW

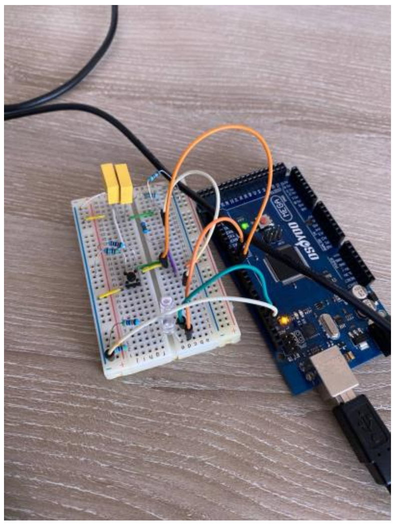
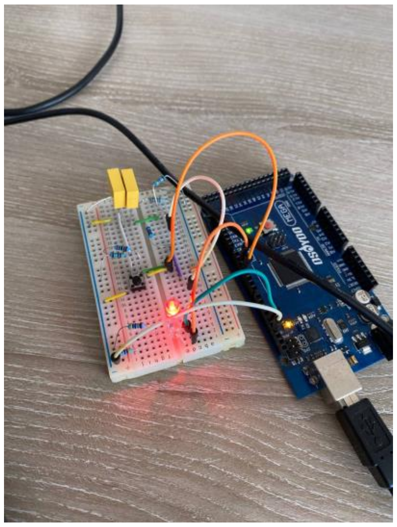
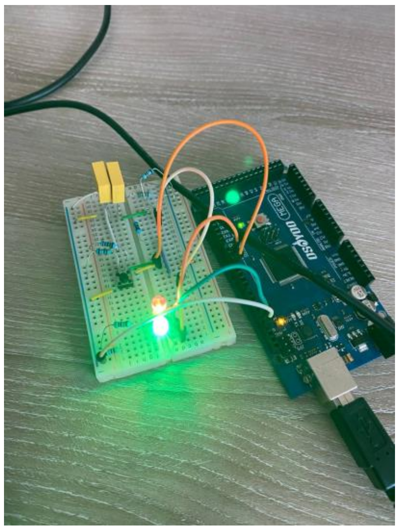
In the Button.h we have the following methods for reading button pressed:

1. Constructor – read button input

In the TaskManager.h we have the following methods for initialization:

1. void initTaskManager() – initialize TaskManager

Results

https://drive.google.com/file/d/12mvF65avXQekcbEKVsEbpSIzJcAf1r2s/view?usp=sharing

Conclusions

This lab demonstrates the implementation of a real-time embedded system using FreeRTOS. The solution effectively showcases key RTOS concepts:

1. Task Management: Three independent tasks running concurrently

2. Synchronization: Using binary semaphores for event notification

3. Communication: Using queues for thread-safe data exchange

4. Timing Control: Precise scheduling with different recurrence rates

The modular design ensures each component has a single responsibility, making the system easy to understand, maintain, and extend. The implementation follows best practices for embedded systems development, including proper resource management, clear interfaces between components, and robust error handling.

This architecture could be extended to more complex scenarios by adding additional tasks, implementing more sophisticated communication patterns, or introducing priority-based scheduling to handle critical operations.

Bibliography

1. Official Arduino Documentation
    - Arduino Reference – Serial Communication https://www.arduino.cc/reference/en/#communication
    - Arduino Mega 1280 Pinout & Datasheet https://docs.arduino.cc/hardware/mega-1280
2. PlatformIO Official Documentation
    - PlatformIO for Arduino Development https://docs.platformio.org/en/latest/platforms/atmelavr.html

3. TUM Courses
    - Introducere în Sistemele Embedded şi Programarea Microcontrolerelor
    - Principiile comunicației seriale şi utilizarea interfeței UART
4. https://drive.google.com/file/d/1mq9aQYGayB3Bn766FlOnAlqN_RGh9hlB/view?usp=sharing

Appendix

1. **GitHub**: https://github.com/Kipitokisk/SI_Lab

```
#include <Arduino.h>
#include "Button.h"
#include "LED.h"
#include "SerialIO.h"

// Button objects for task 1 and task 3
Button button1(4);  // Button for Task 1 (LED toggle)
Button button2(3);  // Button for Task 3 (Increment)
Button button3(2);  // Button for Task 3 (Decrement)

// LED objects for task 1 and task 2
Led led1(13);  // LED for Task 1 (Button LED)
Led led2(11);  // LED for Task 2 (Intermittent LED)

// Global variables
volatile bool led1State = false;  // State of LED 1 (Task 1)
volatile int blinkCount = 0;      // Number of blinks for Task 2
volatile bool led2State = false;  // State of LED 2 (Task 2)

// Task 1: Button LED
void taskButtonLED() {
    if (button1.isPressed()) {
        led1State = !led1State;  // Toggle LED 1 state
        if (led1State) {
            led1.turnOn();
        } else {
            led1.turnOff();
        }
    }
}

void taskLEDIntermittent() {
    static unsigned long previousMillis = 0;

    if (!led1State && blinkCount > 0) {
        unsigned long currentMillis = millis();
        int blinkRate = max(50, 1000 / blinkCount);  // Min delay of 50ms

        if (currentMillis - previousMillis >= blinkRate) {
            previousMillis = currentMillis;
            led2State = !led2State;  // Toggle LED state
            led2.toggle();
        }
    } else {
        led2.turnOff();  // Ensure LED stays off if blinkCount is 0
    }
}


// Task 3: Variable State (Increment/Decrement)
void taskVariableState() {
    if (button2.isPressed()) {
        blinkCount++;  // Increment blink count
    } else if (button3.isPressed()) {
        blinkCount--;  // Decrement blink count
    }
}

// Idle Task: Reporting System State
void taskIdle() {
    // Reporting current system state via serial monitor
    printf("LED 1 State: %s\n", led1State ? "ON" : "OFF");
    printf("LED 2 Blink Count: %d\n", blinkCount);
    delay(200);  // Delay to avoid flooding the serial monitor
}

void setup() {
    // Initialize Serial communication
    serialInit();

    // Initialize LEDs to be turned off initially
    led1.turnOff();
    led2.turnOff();
}

void loop() {
    // Call the tasks in sequence
```

```cpp
    taskButtonLED();          // Task 1: Button LED
    taskLEDIntermittent();    // Task 2: LED Intermittent
    taskVariableState();      // Task 3: Variable State
    taskIdle();               // Idle Task: Reporting state
}

#include "SerialIO.h"

int serial_putchar(char c, FILE* f) {
    Serial.write(c);
    return c;
}

int serial_getchar(FILE* f) {
    while (!Serial.available());
    return Serial.read();
}

FILE serial_stdout;

void serialInit() {
    Serial.begin(115200);
    while (!Serial);

    fdev_setup_stream(&serial_stdout, serial_putchar, serial_getchar, _FDEV_SETUP_RW);
    stdout = &serial_stdout;
    stdin = &serial_stdout;
}

#include "Led.h"

Led::Led(uint8_t pin) {
    this->pin = pin;
    this->state = false;
    pinMode(pin, OUTPUT);
}

void Led::toggle() {
    state = !state;
    digitalWrite(pin, state ? HIGH : LOW);
}

void Led::turnOn() {
    state = true;
    digitalWrite(pin, HIGH);
}

void Led::turnOff() {
    state = false;
    digitalWrite(pin, LOW);
}

#include "Button.h"

Button::Button(uint8_t pin) {
    this->pin = pin;
    pinMode(pin, INPUT_PULLUP);
}

bool Button::isPressed() {
    return digitalRead(pin) == HIGH;
}

#include "TaskManager.h"

// Task handles
TaskHandle_t buttonLedTaskHandle = NULL;
TaskHandle_t syncTaskHandle = NULL;
TaskHandle_t asyncTaskHandle = NULL;

// Synchronization and communication
SemaphoreHandle_t buttonSemaphore = NULL;
QueueHandle_t dataBuffer = NULL;

// Objects
Button button(BUTTON_PIN);
Led led1(LED1_PIN);
Led led2(LED2_PIN);

void initTaskManager() {
    // Create semaphore for button press synchronization
    buttonSemaphore = xSemaphoreCreateBinary();
    // Create queue for data communication between tasks
    dataBuffer = xQueueCreate(10, sizeof(uint8_t[10]));

    xTaskCreate(buttonLedTask,"ButtonLedTask",128,NULL, 1, NULL);
    xTaskCreate(syncTask,"SyncTask",128,NULL,2,NULL);
    xTaskCreate(asyncTask,"AsyncTask",128,NULL,3,NULL);
}
```

```

// Task 1: Button LED - Checks button state and signals with semaphore
void buttonLedTask(void *pvParameters) {
    TickType_t xLastWakeTime;
    xLastWakeTime = xTaskGetTickCount();

    while (1) {
        if (button.isPressed()) {
            led1.turnOn();
            xSemaphoreGive(buttonSemaphore);
            vTaskDelay(pdMS_TO_TICKS(BUTTON_LED_DURATION_MS));
            led1.turnOff();
        }

        vTaskDelayUntil(&xLastWakeTime, pdMS_TO_TICKS(TASK1_PERIOD_MS));
    }
}

// Task 2: Synchronized Task - Waits for semaphore, increments N, sends N bytes
void syncTask(void *pvParameters) {
    static uint8_t N = 0;
    while (1) {
        // Wait for semaphore from Task 1
        if (xSemaphoreTake(buttonSemaphore, portMAX_DELAY) == pdTRUE) {
            N++;
            uint8_t data[N];
            for (uint8_t i = 0; i < N; i++) {
                data[i] = i + 1;
            }
            xQueueSendToFront(dataBuffer, &data, portMAX_DELAY);
            for (uint8_t i = 0; i < N; i++) {
                led2.turnOn();
                vTaskDelay(pdMS_TO_TICKS(LED_ON_DURATION_MS));
                led2.turnOff();
                vTaskDelay(pdMS_TO_TICKS(LED_OFF_DURATION_MS));
            }
        }
        vTaskDelay(pdMS_TO_TICKS(TASK2_INTERVAL_MS));
    }
}

```

```
// Task 3: Asynchronous Task - Reads buffer every 200ms and displays data
void asyncTask(void *pvParameters) {
    uint8_t receivedData[10];
    while (1)
    {
        if (xQueueReceive(dataBuffer, &receivedData, portMAX_DELAY) == pdTRUE) {
            printf("Data: ");
            for (uint8_t i = 0; i < sizeof(receivedData); i++)
            {
                if (receivedData[i] == 0) {
                    printf("\n");
                    break;
                }
                printf("%d", receivedData[i]);
                printf(" ");
            }
        }
        vTaskDelay(pdMS_TO_TICKS(TASK3_PERIOD_MS));
    }
}
```