TECHNICAL UNIVERSITY OF MOLDOVA
FACULTY OF COMPUTERS, INFORMATICS AND MICROELECTRONICS
DEPARTMENT OF SOFTWARE ENGINEERING AND AUTOMATICS

Report of laboratory work №2

Theme: Operational Systems

Fulfilled: st. gr. FAF-202 **Popa Eugeniu**

Controlled: univ. lecturer **Moraru Dumitru**

Chișinău 2023

THE TASKS OF THE LABORATORY WORK

Main task: To design an MCU-based application that would run at least 3 tasks in two versions: Sequentially and with FreeRTOS.

Task 1: LED button – LED state change upon detection of a button press.

Task 2: A second intermittent LED when the LED from the first task is off.

Task 3: Increment/decrement the value of a variable when pressing two buttons that will represent the number of recurrences/time during which the led from the second task will be in a state.

Task 4: The Idle task will be used to display the states in the program, such as LED status display, and message display when the buttons are pressed, an implementation being that when the button is pressed, a variable is set, and when the message is displayed - reset, implementing the provider/consumer mechanism.

THE PROGRESS OF THE WORK

Sequential Implementation

This code is written in C++ and controls two LEDs (green and red) and three buttons, and it is divided into four tasks that handle different functionalities.

The setup() function (Fig. 1.1) runs only once at the beginning of the program and initializes the pins for each button and LED, as well as the Serial communication for debugging purposes.

```
void setup()
{
  button1.setPin(button1Pin);
  button1.setup();

  button2.setPin(button2Pin);
  button2.setup();

  button3.setPin(button3Pin);
  button3.setup();

  redLed.setPin(redLedPin);
  redLed.setup();

  greenLed.setPin(greenLedPin);
  greenLed.setup();

  Serial.begin(9600);
  Serial.println("This is laboratory work nr.2");
}
```

Fig 1.1 setup() function

The task1() function (Fig 1.2) checks the state of the first button and toggles the green LED state accordingly. If the button is pressed, the green LED state is changed, and the LED is turned on/off.

```
void task1()
{
  bool button1State = button1.readButton();
  if (button1State != lastButton1State)
  {
    lastButton1State = button1State;
    if (button1State == false)
    {
      if (greenLedState == true)
      {
        if (blinksPerSecond == 0)
        {
          greenLedState = false;
          delay(500);
        }
        else
        {
          Serial.println("Decrease the frequency to 0 to turn off the green led");
        }
      }
      else
      {
        greenLedState = true;
        delay(500);
      }
      greenLed.switchLight(greenLedState);
    }
  }
}
```

Fig 1.2 task1() function

The task2() function (Fig 1.3) monitors the green LED's state and turns on/off the red LED according to it. If the green LED is off, the red LED is turned on.

```
void task2()
{
  if (lastGreenLedState != greenLedState)
  {
    lastGreenLedState = greenLedState;
    if (greenLedState == false)
    {
      redLed.switchLight(true);
    }
    else
    {
      redLed.switchLight(false);
    }
  }
}
```

Fig 1.3 task2() function

The task3() function (Fig 1.4) handles the two remaining buttons. If the second button is pressed, it increases the number of blinks per second of the green LED, while if the third button is pressed, it decreases the blinks per second. The green LED then blinks with the updated frequency.

```
void task3()
{
  bool button2State = button2.readButton();
  if (button2State == lastButton2State && (millis() - lastButton2Time) > 50)
  {
    lastButton2State = button2State;
    lastButton2Time = millis();
    if (button2State == false)
    {
      blinksPerSecond++;
    }
  }

  bool button3State = button3.readButton();
  if (button3State == lastButton3State && (millis() - lastButton3Time) > 50)
  {
    lastButton3State = button3State;
    lastButton3Time = millis();
    if (button3State == false)
    {
      if (blinksPerSecond > 0)
      {
        blinksPerSecond--;
      }
      else if (blinksPerSecond == 0)
      {
        greenLed.switchLight(true);
      }
    }
  }

  if (blinksPerSecond > 0)
  {
    int interval = 1000 / blinksPerSecond;
    greenLed.switchLight(false);
    delay(interval / 2);
    greenLed.switchLight(true);
    delay(interval / 2);
  }
}
```

Fig 1.4 task3() function

The task4() function (Fig 1.5) monitors the state of the green LED and displays information on the Serial Monitor if it changes. Additionally, it checks if the number of blinks per second is updated by any of the buttons and prints the new value on the Serial Monitor.

```
void task4()
{
  if (lastGreenLedState2 != greenLedState)
  {
    lastGreenLedState2 = greenLedState;
    if (greenLedState == true)
    {
      Serial.println("Monitoring leds:");
      Serial.println("Green led status: on");
      Serial.println("Red led. Status: off");
    }
    else
    {
      Serial.println("Monitoring leds:");
      Serial.println("Green led status: off");
      Serial.println("Red led status: on");
    }
  }

  if (counter != blinksPerSecond)
  {
    if (counter < blinksPerSecond)
    {
      Serial.print("Button 2 is pressed. Incrementing the number of blinks per second: ");
      Serial.println(blinksPerSecond);
    }
    else if (counter > blinksPerSecond)
    {
      Serial.print("Button 3 is pressed. Decrementing the number of blinks per second: ");
      Serial.println(blinksPerSecond);
    }
    counter = blinksPerSecond;
  }
}
```

Fig 1.5 task4() function

The loop() function (Fig 1.6) is the main function of the program, which repeatedly calls each task in a sequential manner.

```
void loop()
{
  task1();
  task2();
  task3();
  task4();
}
```

Fig 1.4 loop() function

FreeRTOS Implementation

This is a program written in C++ using the Arduino framework and FreeRTOS. The program controls two LEDs and three buttons. One of the buttons turns on and off a green LED, while the other two buttons increase or decrease the frequency at which the green LED blinks.

The task1() function (Fig 2.1) monitors the state of button 1 and toggles the green LED accordingly. If the button is pressed, it toggles the state of the green LED. If the LED is on, it turns it off. If it's off, it turns it on. The state of the LED is protected by a mutex to avoid conflicts between tasks that access it.

```
void task1(void *pvParameters)
{
  while(1)
  {
    bool button1State = button1.readButton();
    if (button1State != lastButton1State)
    {
      lastButton1State = button1State;
      if (button1State == false)
      {
        if (greenLedState == true)
        {
          if (blinksPerSecond == 0)
          {
            greenLedState = false;
          }
          else
          {
            Serial.println("Decrease the frequency to 0 to turn off the green led");
          }
        }
        else
        {
          greenLedState = true;
        }
        xSemaphoreTake(greenLedStateMutex, portMAX_DELAY);
        greenLed.switchLight(greenLedState);
        xSemaphoreGive(greenLedStateMutex);
      }
    }
    vTaskDelay(100 / portTICK_PERIOD_MS);
  }
}
```

Fig 2.1 task1() function

The task2() function (Fig 2.2) monitors the state of the green LED and turns on the red LED if the green LED is off. This is done to provide a visual cue that the green LED is off. The state of the green LED is protected by a mutex to avoid conflicts between tasks that access it.

```
void task2(void *pvParameters)
{
  while(1)
  {
    bool currentGreenLedState = greenLedState;
    if (lastGreenLedState != currentGreenLedState)
    {
      lastGreenLedState = currentGreenLedState;
      if (currentGreenLedState == false)
      {
        xSemaphoreTake(greenLedStateMutex, portMAX_DELAY);
        redLed.switchLight(true);
        xSemaphoreGive(greenLedStateMutex);
      }
      else
      {
        xSemaphoreTake(greenLedStateMutex, portMAX_DELAY);
        redLed.switchLight(false);
        xSemaphoreGive(greenLedStateMutex);
      }
    }
    vTaskDelay(100 / portTICK_PERIOD_MS);
  }
}
```

Fig 2.2 task2() function

The task3() function (Fig 2.3) monitors the state of button 2 and button 3. Button 2 increases the blink rate of the green LED, and button 3 decreases it. If the blink rate is set to 0, button 3 turns the green LED on. The blink rate is stored in a variable that is protected by a mutex to avoid conflicts between tasks that access it. When the blink rate is greater than 0, the green LED blinks at the specified rate.

```
void task3(void *pvParameters)
{
  while(1)
  {
    bool button2State = button2.readButton();
    if (button2State == lastButton2State && (millis() - lastButton2Time) > 50)
    {
      lastButton2State = button2State;
      lastButton2Time = millis();
      if (button2State == false)
      {
        blinksPerSecond++;
      }
    }

    bool button3State = button3.readButton();
    if (button3State == lastButton3State && (millis() - lastButton3Time) > 50)
    {
      lastButton3State = button3State;
      lastButton3Time = millis();
      if (button3State == false)
      {
        if (blinksPerSecond > 0)
        {
          blinksPerSecond--;
        }
        else if (blinksPerSecond == 0)
        {
          greenLed.switchLight(true);
        }
      }
    }

    if (blinksPerSecond > 0)
    {
      int interval = 1000 / blinksPerSecond;
      greenLed.switchLight(false);
      delay(interval / 2);
      greenLed.switchLight(true);
      delay(interval / 2);
    }
    vTaskDelay(100 / portTICK_PERIOD_MS);
  }
}
```

Fig 2.3 task3() function

The task4() function (Fig 2.4) monitors the state of the green and red LEDs and prints messages to the serial monitor to indicate their status. It also prints the blink rate when it changes.

```
void task4(void *pvParameters)
{
  while(1)
  {
    if (lastGreenLedState2 != greenLedState)
    {
      lastGreenLedState2 = greenLedState;
      if (greenLedState == true)
      {
        Serial.println("Monitoring leds:");
        Serial.println("Green led status: on");
        Serial.println("Red led. Status: off");
      }
      else
      {
        Serial.println("Monitoring leds:");
        Serial.println("Green led status: off");
        Serial.println("Red led status: on");
      }
    }

    if (counter != blinksPerSecond)
    {
      if (counter < blinksPerSecond)
      {
        Serial.print("Blinks per second: ");
        Serial.println(blinksPerSecond);
      }
      else if (counter > blinksPerSecond)
      {
        Serial.print("Blinks per second: ");
        Serial.println(blinksPerSecond);
      }
      counter = blinksPerSecond;
    }
    vTaskDelay(100 / portTICK_PERIOD_MS);
  }
}
```

Fig 2.4 task4() function

Semaphores are used to protect shared resources such as the state of the green LED and the blink rate. The xSemaphoreTake() and xSemaphoreGive() functions are used to acquire and release the mutex, respectively.

The setup() function (Fig 2.5) is called once when the program starts, and it is used to initialize variables, declare pin modes, and setup libraries or sensors. In the given code, the setup() function initializes and sets up the pins for three buttons, two LEDs, and initializes the Serial communication at a baud rate of 9600. It also creates four tasks using the FreeRTOS library.

```
void setup()
{
  button1.setPin(button1Pin);
  button1.setup();

  button2.setPin(button2Pin);
  button2.setup();

  button3.setPin(button3Pin);
  button3.setup();

  redLed.setPin(redLedPin);
  redLed.setup();

  greenLed.setPin(greenLedPin);
  greenLed.setup();

  Serial.begin(9600);
  Serial.println("This is laboratory work nr.2");

  greenLedStateMutex = xSemaphoreCreateMutex();

  xTaskCreate(task1, "Task 1", 128, NULL, 1, NULL);
  xTaskCreate(task2, "Task 2", 128, NULL, 2, NULL);
  xTaskCreate(task3, "Task 3", 128, NULL, 3, NULL);
  xTaskCreate(task4, "Task 4", 64, NULL, 4, NULL);
}
```

Fig 2.5 setup() function

On the other hand, the loop() function runs continuously after the setup() function is executed. It is the main part of the program that repeatedly checks the input values and updates the output. However, in the given code, the loop() function is empty as there are no instructions to be executed repeatedly. Instead, the tasks created in the setup() function run parallelly with the loop() function. Therefore, the program does not need to be stuck in a loop to continuously execute code; the FreeRTOS scheduler takes care of executing the four tasks in parallel.

Electrical scheme made in Proteus

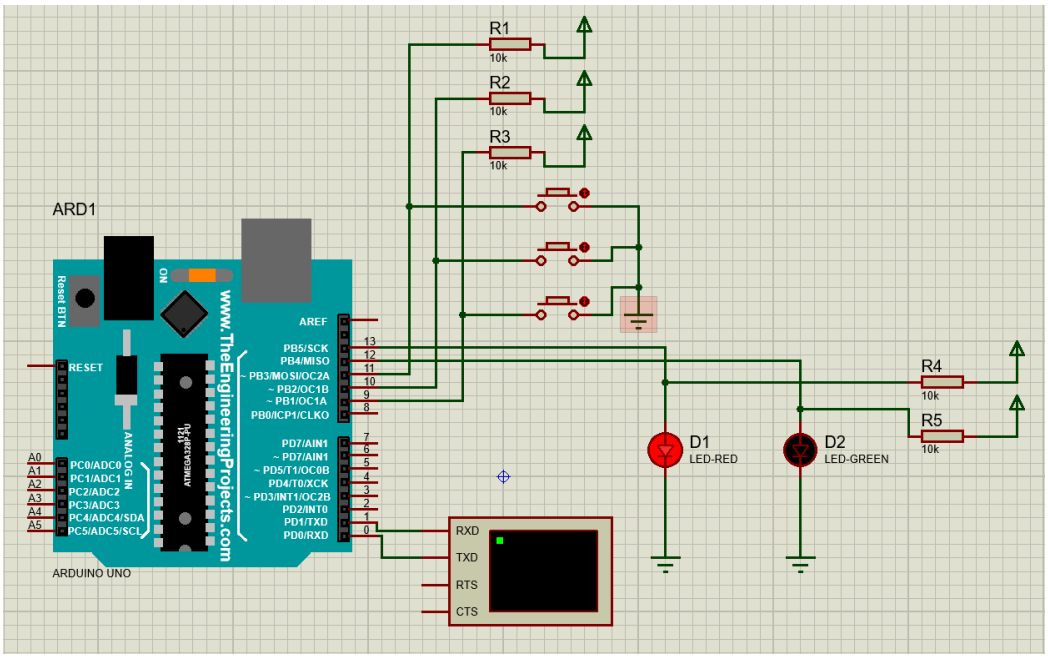

Fig 3 Electrical scheme

CONCLUSIONS

Completing the laboratory work showed that creating an MCU application that runs multiple tasks can be challenging but rewarding. The implementation of tasks using both the Sequential and FreeRTOS versions highlighted that FreeRTOS provides a more efficient and organized way of managing tasks, allowing for easier task prioritization, synchronization, and communication between tasks.

The application's tasks demonstrated their ability to respond to user input and modify the program's behavior. The laboratory work emphasized the importance of task management and the benefits of using an RTOS like FreeRTOS, showcasing the ability to create a responsive and dynamic application that can modify its behavior based on external events.