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 **Pogorevici Daniel**

Controlled: univ. lecturer **Moraru Dumitru**

Chișinău 2023

1. The Task of the Laboratory Work

Creating an MCU application that will run at least 3 tasks in two versions - Sequential and FreeRTOS.

The application will run at least 3 tasks including:

Task 1: Led Button - LED status change when a button is pressed.

Task 2: A second Flashing LED when the LED on the first Task is off

Task 3: Increment / decrement value of a variable by pressing second button that represents the amount of recurrences / time in which the LED from the second task will be in a state ON.

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

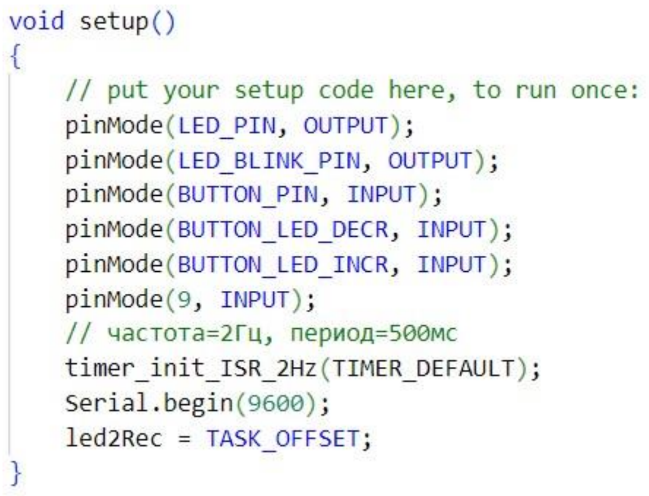
1. Implementation

1. Sequential Execution

Description:

For the sequential execution I used first of all the “time-api.h” library in order to set up a timer for tasks execution.

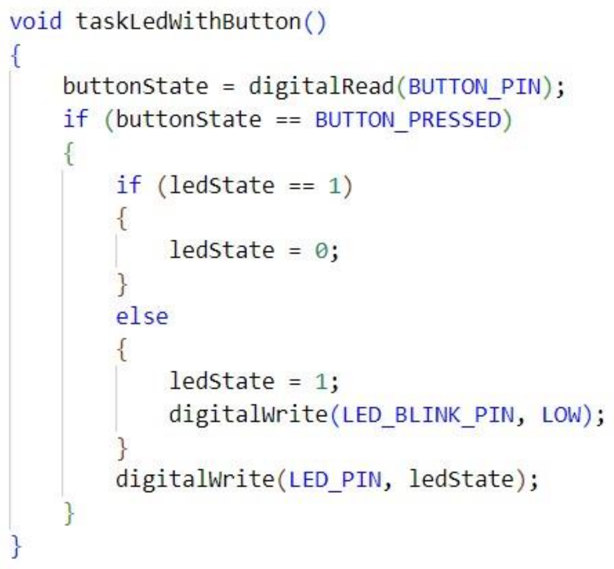
First, I use setup function (fig. 1.1) which initializes the pin modes and starts the timer interrupt at a frequency of 2 Hz. It also sets the LED2 Recur parameter and starts the serial communication at 9600 baud rates.



```
void setup()
{
    // put your setup code here, to run once:
    pinMode(LED_PIN, OUTPUT);
    pinMode(LED_BLINK_PIN, OUTPUT);
    pinMode(BUTTON_PIN, INPUT);
    pinMode(BUTTON_LED_DECR, INPUT);
    pinMode(BUTTON_LED_INCR, INPUT);
    pinMode(9, INPUT);
    // частота=2Гц, период=500мс
    timer_init_ISR_2Hz(TIMER_DEFAULT);
    Serial.begin(9600);
    led2Rec = TASK_OFFSET;
}
```

Fig. 1.1 – setup()

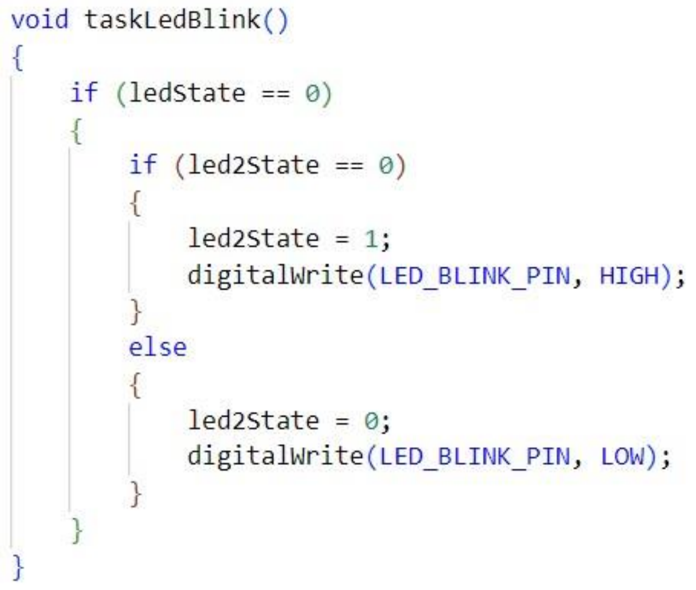
Next one is the function (fig. 1.2) to control the LED1 state with a button. If the button is pressed, the LED1 state is toggled and LED2 is turned off.



```
void taskLedWithButton()
{
    buttonState = digitalRead(BUTTON_PIN);
    if (buttonState == BUTTON_PRESSED)
    {
        if (ledState == 1)
        {
            ledState = 0;
        }
        else
        {
            ledState = 1;
            digitalWrite(LED_BLINK_PIN, LOW);
        }
        digitalWrite(LED_PIN, ledState);
    }
}
```

Fig. 1.2 – taskLedWithButton()

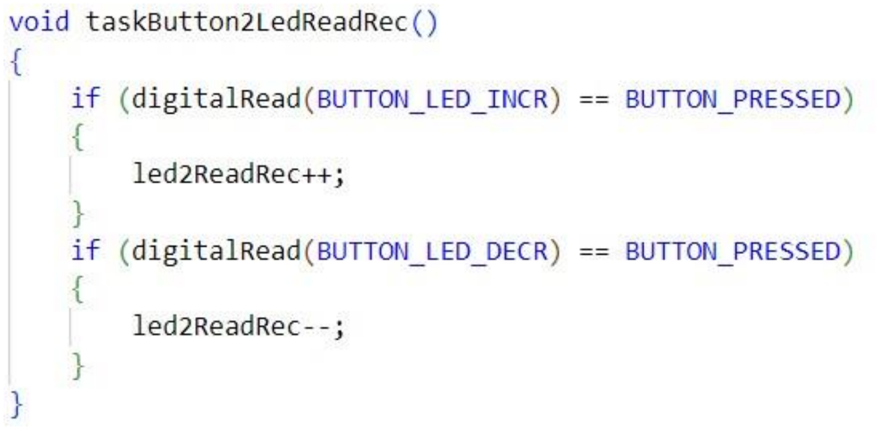
Next function (fig. 1.3) is the one for LED2. It only blinks if LED1 is turned off. So, it checks what is the state for the first led, if it’s 0 then the LED starts blinking.



```
void taskLedBlink()
{
    if (ledState == 0)
    {
        if (led2State == 0)
        {
            led2State = 1;
            digitalWrite(LED_BLINK_PIN, HIGH);
        }
        else
        {
            led2State = 0;
            digitalWrite(LED_BLINK_PIN, LOW);
        }
    }
}
```

Fig. 1.3 – taskLedBlink()

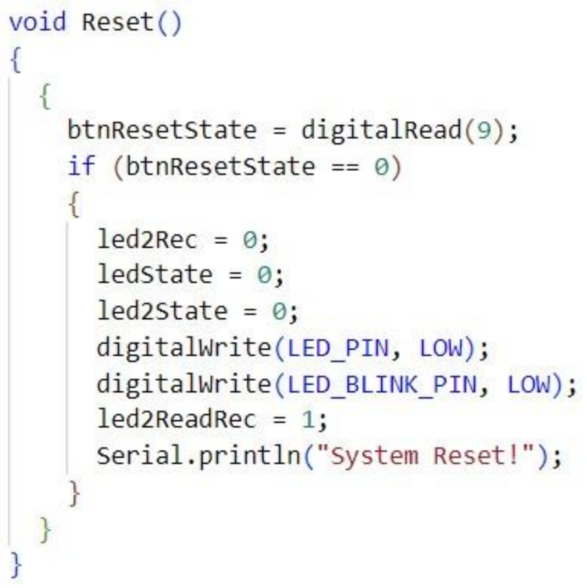
Now, the function (fig. 1.4) to read the LED2 blink frequency from buttons. If the button for increasing the frequency is pressed, LED2 blink frequency is incremented. If the button for decreasing the frequency is pressed, LED2 blink frequency is decremented.



```
void taskButton2LedReadRec()
{
    if (digitalRead(BUTTON_LED_INCR) == BUTTON_PRESSED)
    {
        led2ReadRec++;
    }
    if (digitalRead(BUTTON_LED_DECR) == BUTTON_PRESSED)
    {
        led2ReadRec--;
    }
}
```

Fig. 1.4 – taskButton2LedReadRec()

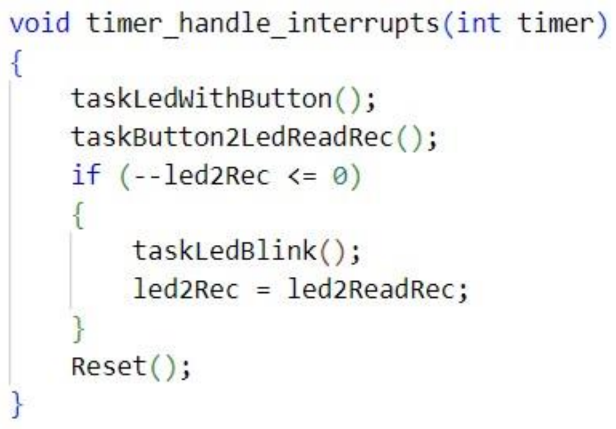
Also, I have a reset function (fig.1.5) which reads input from button and if it's pressed, the frequency and led states are reset.



```
void Reset()
{
  {
    btnResetState = digitalRead(9);
    if (btnResetState == 0)
    {
      led2Rec = 0;
      ledState = 0;
      led2State = 0;
      digitalWrite(LED_PIN, LOW);
      digitalWrite(LED_BLINK_PIN, LOW);
      led2ReadRec = 1;
      Serial.println("System Reset!");
    }
  }
}
```

Fig. 1.5 – Reset()

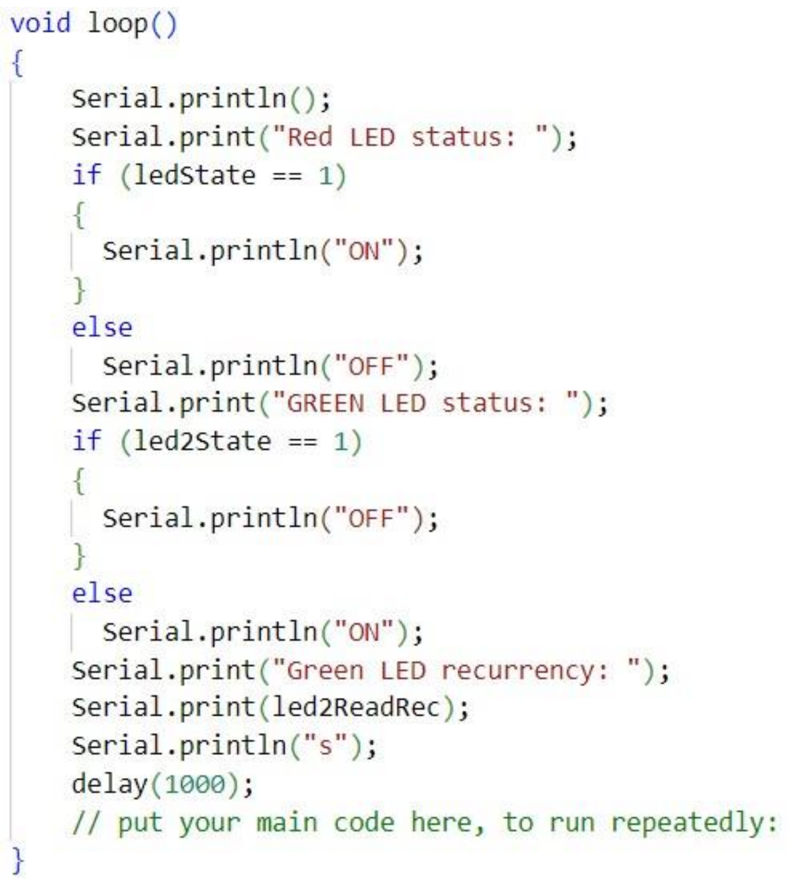
Afterwards, I have the function (fig. 1.6) which handles timer interrupts. Which basically is called by the timer interrupt service routine at a specified frequency. Also I subtract 1 from led2Rec and checks if it is less than or equal to zero, which indicates that it is time to blink the second LED.



```
void timer_handle_interrupts(int timer)
{
    taskLedWithButton();
    taskButton2LedReadRec();
    if (--led2Rec <= 0)
    {
        taskLedBlink();
        led2Rec = led2ReadRec;
    }
    Reset();
}
```

Fig. 1.6 – timer_handle_interrupts()

Finally, I have loop (fig. 1.7) function which prints the status of the LEDs and the blinking frequency of the second LED to the Serial Monitor every second in the loop() function.



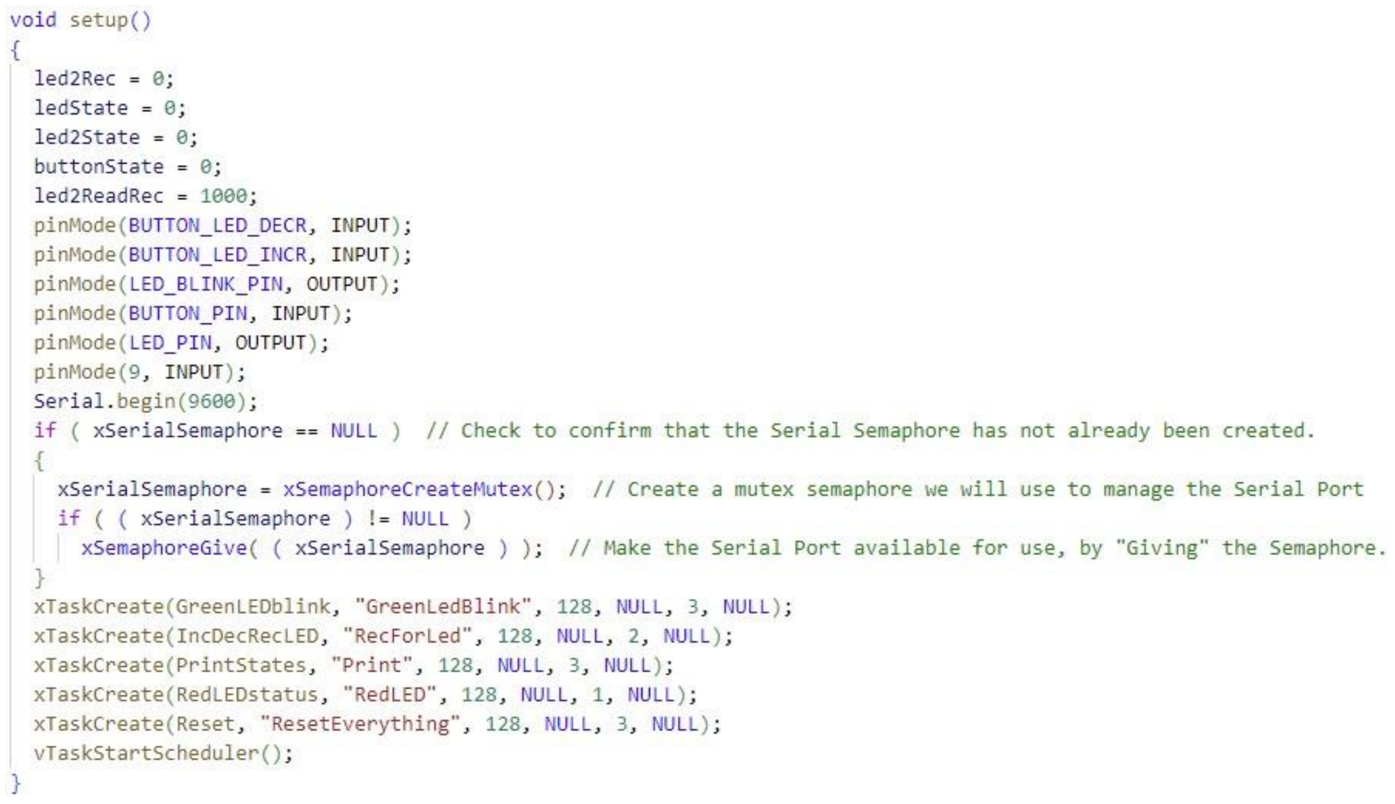
```
void loop()
{
    Serial.println();
    Serial.print("Red LED status: ");
    if (ledState == 1)
    {
      Serial.println("ON");
    }
    else
      Serial.println("OFF");
    Serial.print("GREEN LED status: ");
    if (led2State == 1)
    {
      Serial.println("OFF");
    }
    else
      Serial.println("ON");
    Serial.print("Green LED recurrency: ");
    Serial.print(led2ReadRec);
    Serial.println("s");
    delay(1000);
    // put your main code here, to run repeatedly:
}
```

Fig. 1.7 – loop()

2. FreeRTOS Execution

Description:

For the FreeRTOS tasks execution I use the “Arduino_FreeRTOS.h” library. In the setup (fig. 2.1) function I initialize the variables, pin modes, and start the FreeRTOS scheduler. It sets the variables to their initial values, sets the pin modes, starts the Serial communication, creates the tasks using the xTaskCreate function, creates a new serial semaphore and makes the serial port available, and finally starts the scheduler using the vTaskStartScheduler function.

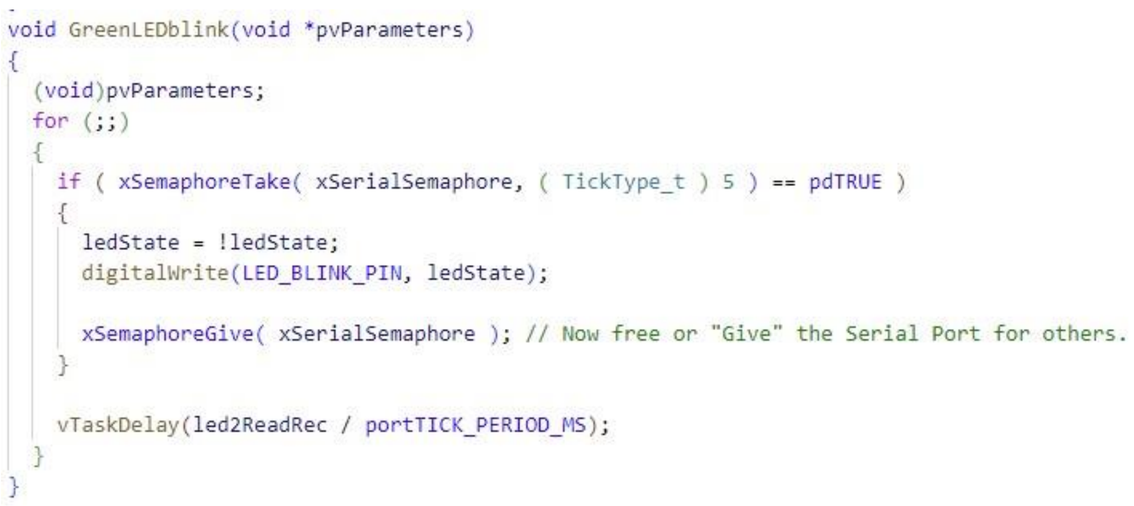


```
void setup()
{
  led2Rec = 0;
  ledState = 0;
  led2State = 0;
  buttonState = 0;
  led2ReadRec = 1000;
  pinMode(BUTTON_LED_DECR, INPUT);
  pinMode(BUTTON_LED_INCR, INPUT);
  pinMode(LED_BLINK_PIN, OUTPUT);
  pinMode(BUTTON_PIN, INPUT);
  pinMode(LED_PIN, OUTPUT);
  pinMode(9, INPUT);
  Serial.begin(9600);
  if ( xSerialSemaphore == NULL )  // Check to confirm that the Serial Semaphore has not already been created.
  {
    xSerialSemaphore = xSemaphoreCreateMutex();  // Create a mutex semaphore we will use to manage the Serial Port
    if ( ( xSerialSemaphore ) != NULL )
      xSemaphoreGive( ( xSerialSemaphore ) );  // Make the Serial Port available for use, by "Giving" the Semaphore.
  }
  xTaskCreate(GreenLEDblink, "GreenLedBlink", 128, NULL, 3, NULL);
  xTaskCreate(IncDecRecLED, "RecForLed", 128, NULL, 2, NULL);
  xTaskCreate(PrintStates, "Print", 128, NULL, 3, NULL);
  xTaskCreate(RedLEDstatus, "RedLED", 128, NULL, 1, NULL);
  xTaskCreate(Reset, "ResetEverything", 128, NULL, 3, NULL);
  vTaskStartScheduler();
}
```

Fig. 2.1 – setup()

For the FreeRTOS execution loop function is left empty because FreeRTOS uses a cooperative multitasking approach, where the tasks are executed in a round-robin fashion by the scheduler. So, there is no need for a traditional loop function.

Next, the GreenLEDblink (fig. 2.2) function blinks an LED connected to a pin on the board. The LED turns on and off in a repeating pattern with a delay that is controlled by the variable led2ReadRec. Also it uses the semaphore, it takes it with periodicity and frees after finishing. This function runs continuously in a loop.



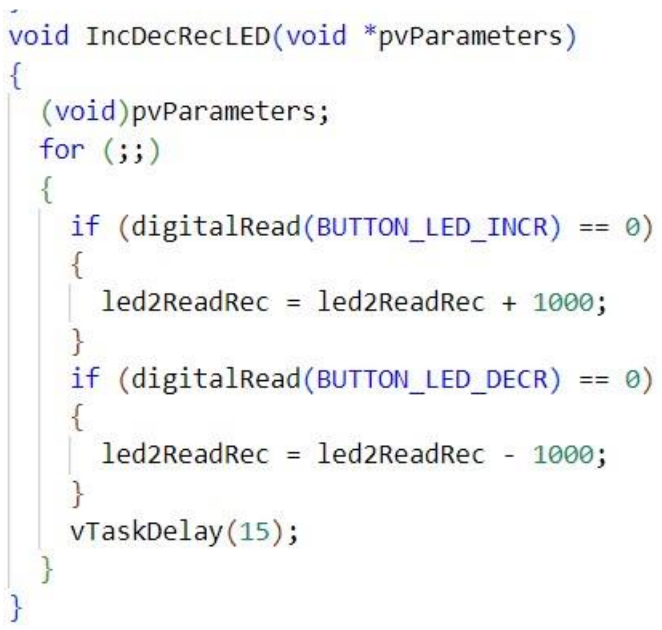
```
void GreenLEDblink(void *pvParameters)
{
  (void)pvParameters;
  for (;;)
  {
    if ( xSemaphoreTake( xSerialSemaphore, ( TickType_t ) 5 ) == pdTRUE )
    {
      ledState = !ledState;
      digitalWrite(LED_BLINK_PIN, ledState);

      xSemaphoreGive( xSerialSemaphore ); // Now free or "Give" the Serial Port for others.
    }

    vTaskDelay(led2ReadRec / portTICK_PERIOD_MS);
  }
}
```

Fig. 2.2 – GreenLEDblink()

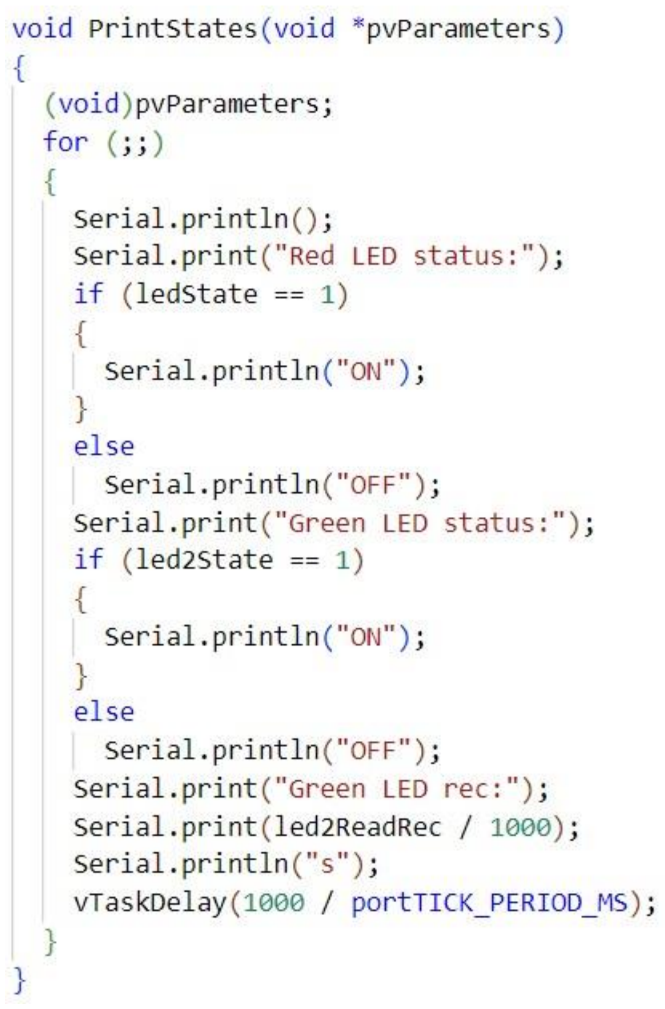
The IncDecRecLED (fig. 2.3) function monitors two buttons connected to pins on the board. If the button connected to the pin BUTTON_LED_INCR is pressed, led2ReadRec is incremented by 1000. If the button connected to the pin BUTTON_LED_DECR is pressed, led2ReadRec is decremented by 1000. This function also runs continuously in a loop.



```
void IncDecRecLED(void *pvParameters)
{
  (void)pvParameters;
  for (;;)
  {
    if (digitalRead(BUTTON_LED_INCR) == 0)
    {
      led2ReadRec = led2ReadRec + 1000;
    }
    if (digitalRead(BUTTON_LED_DECR) == 0)
    {
      led2ReadRec = led2ReadRec - 1000;
    }
    vTaskDelay(15);
  }
}
```

Fig. 2.3 – IncDecRecLED()

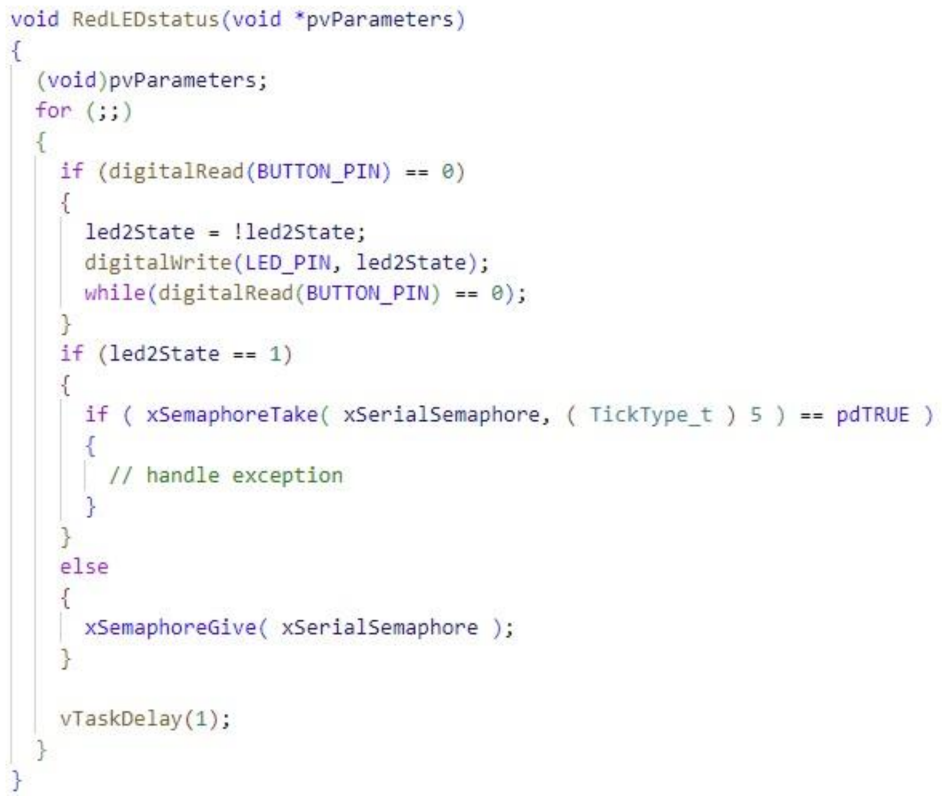
The PrintStates (fig. 2.4) function prints the current status of the two LEDs and the value of led2ReadRec to the serial port, which can be viewed on a computer using a serial terminal program. This function runs continuously in a loop.



```
void PrintStates(void *pvParameters)
{
  (void)pvParameters;
  for (;;)
  {
    Serial.println();
    Serial.print("Red LED status:");
    if (ledState == 1)
    {
      Serial.println("ON");
    }
    else
      Serial.println("OFF");
    Serial.print("Green LED status:");
    if (led2State == 1)
    {
      Serial.println("ON");
    }
    else
      Serial.println("OFF");
    Serial.print("Green LED rec:");
    Serial.print(led2ReadRec / 1000);
    Serial.println("s");
    vTaskDelay(1000 / portTICK_PERIOD_MS);
  }
}
```

Fig. 2.4 – PrintStates()

The RedLEDstatus (fig. 2.5) function monitors a button connected to a pin on the board. If the button connected to the pin BUTTON_PIN is pressed, the LED connected to the pin LED_PIN turns on or off depending on its current state. If the LED is currently on, it turns off, and if it is currently off, it turns on. Also it uses the semaphore, taking it when led2state == 1 and giving it after finishing. This function also runs continuously in a loop.

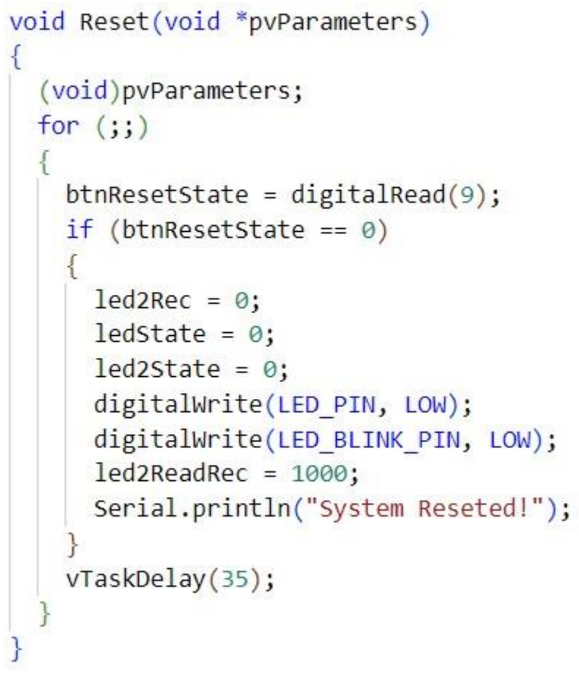


```
void RedLEDstatus(void *pvParameters)
{
  (void)pvParameters;
  for (;;)
  {
    if (digitalRead(BUTTON_PIN) == 0)
    {
      led2State = !led2State;
      digitalWrite(LED_PIN, led2State);
      while(digitalRead(BUTTON_PIN) == 0);
    }
    if (led2State == 1)
    {
      if ( xSemaphoreTake( xSerialSemaphore, ( TickType_t ) 5 ) == pdTRUE )
      {
        // handle exception
      }
    }
    else
    {
      xSemaphoreGive( xSerialSemaphore );
    }

    vTaskDelay(1);
  }
}
```

Fig. 2.5 – RedLEDstatus()

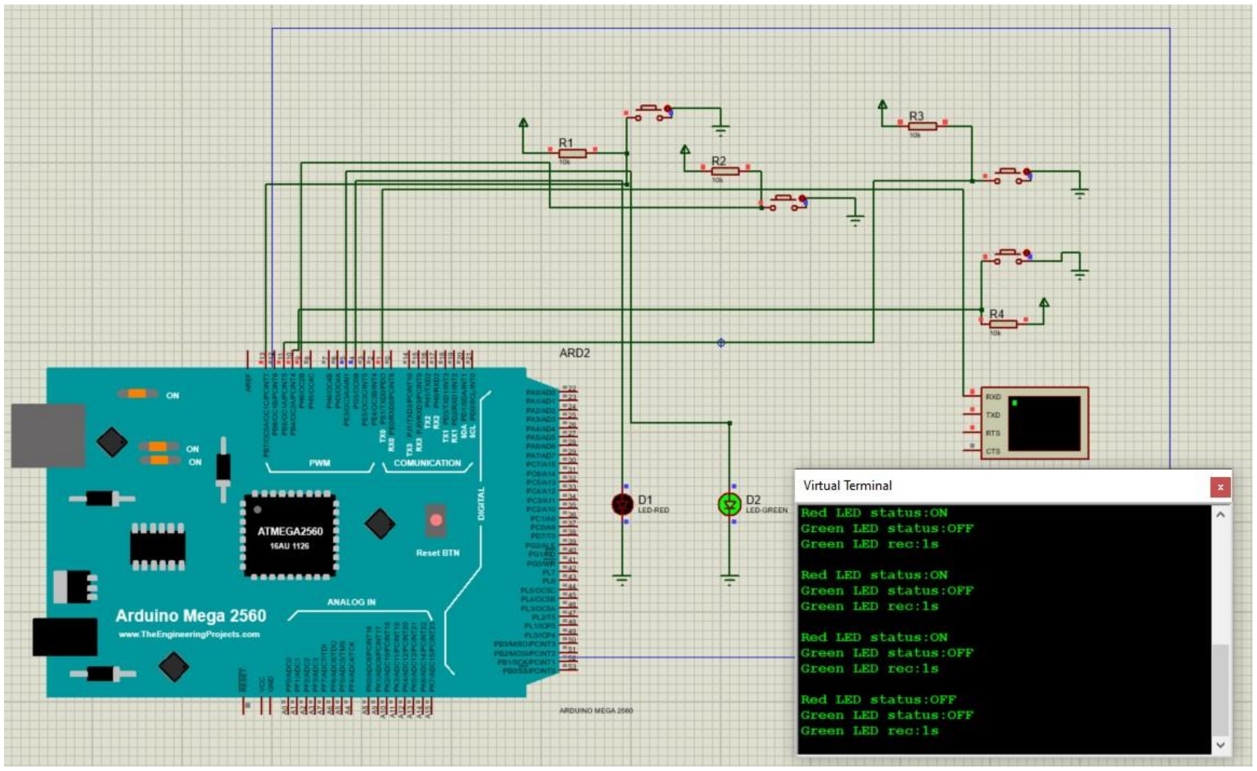
The Reset (fig. 2.6) function monitors a button connected to the pin 9 on the board. If the button is pressed, all variables are reset to their initial values, and both LEDs turn off. The value of led2ReadRec is set to 1000. This function also runs continuously in a loop.



```
void Reset(void *pvParameters)
{
  (void)pvParameters;
  for (;;)
  {
    btnResetState = digitalRead(9);
    if (btnResetState == 0)
    {
      led2Rec = 0;
      ledState = 0;
      led2State = 0;
      digitalWrite(LED_PIN, LOW);
      digitalWrite(LED_BLINK_PIN, LOW);
      led2ReadRec = 1000;
      Serial.println("System Reseted!");
    }
    vTaskDelay(35);
  }
}
```

Fig. 2.6 – Reset()

3. Electrical Scheme



```
Red LED status:ON
Green LED status:OFF
Green LED rec:1s

Red LED status:ON
Green LED status:OFF
Green LED rec:1s

Red LED status:ON
Green LED status:OFF
Green LED rec:1s

Red LED status:OFF
Green LED status:OFF
Green LED rec:1s
```

Fig. 3 – Electrical Scheme

4. Conclusions

After completing the laboratory work, it can be concluded that creating an MCU application that runs multiple tasks is a challenging yet rewarding experience.

By implementing the tasks using both the Sequential and FreeRTOS versions, it became clear that FreeRTOS provides a more efficient and organized way of managing tasks in comparison to the sequential version. It allows for easier task prioritization, synchronization, and communication between tasks.

The tasks implemented in the application demonstrated the ability to respond to user input and modify the program's behavior.

Overall, the laboratory work demonstrated the importance of task management and the benefits of using an RTOS like FreeRTOS. It also showcased the ability to create a responsive and dynamic application that can respond to user input and modify its behavior based on external events.