**TECHNICAL UNIVERSITY OF MOLDOVA**

**FACULTY OF COMPUTERS, INFORMATICS AND MICROELECTRONICS**

**DEPARTMENT OF SOFTWARE ENGINEERING AND AUTOMATICS**

**Report of laboratory work №2**

**Theme: Operating systems**

**Fulfilled: st.gr. FAF-191 Boico Alexandr**

**Controlled: univ. lecturer Moraru Dumitru**

**Chișinău 2022**

**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 number 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.

**The progress of the work**

1. **Task 1**

In the first task we had to do button that changes state of a LED

**Appendix 1**

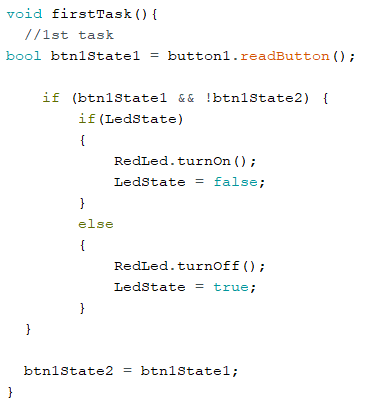
****

Fig 1 Implementation of first task

I have created 2 variables for the first button. “btn1State1” is a local one and saves the state of button in moment when function “firstTask()” is running. The second one is “btn1State2”. This variable is global and saves the last state of the button. It is used to avoid situations when the button is pressed permanently, so one pressing is equal to one state change.

If the first button is pressed and the previous state is *false* then the program checks the value of variable of type bool “LedState”. If it is *true* then the LED of color red is turning ON and “LedState” changes to *false,* else the LED of color red is turning OFF and “LedState” changes to *true.*

1. **Task 2**

In the second task we had to implement the second LED that flashes when the first is turning OFF.

To solve this task I need only to compare the variable “LedState” with true. If it’s *true* then the program turns ON the green LED else, when “LedState” is *false*, the program turns OFF this.

**Appendix 2**

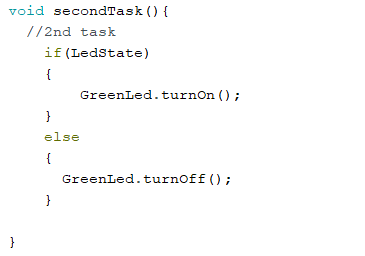
****

Fig 2 secondTask()

1. **Task 3**

In the 3rd task we had to implement an increment value of a variable by pressing the second button that represents the number of recurrences in which the LED from the second task will be in a state ON.

**Appendix 3**

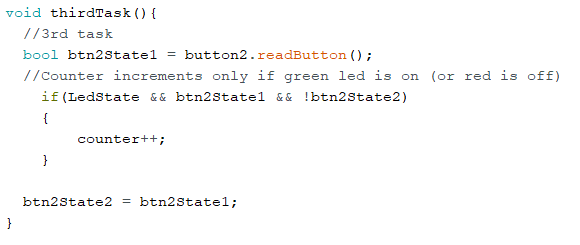
****

Fig 3 thirdTask()

I have created 2 variables for the second button. “btn2State1” is a local one and saves the state of the button in the moment when the function “thirdTask()” is running. The second one is “btn2State2”. This variable is global and saves the last state of the button. It is used to avoid situations when the button is pressed permanently, so one pressing is equal to changing the value of the “counter” by one.

If the second button is pressed and the previous state is *false* then the program checks the value of variable of type bool “LedState”. If it is *true* then the program increments the counter.

1. **Task 4**

In the 4th task we had to add displaying the LEDs status and counter.

**Appendix 4**

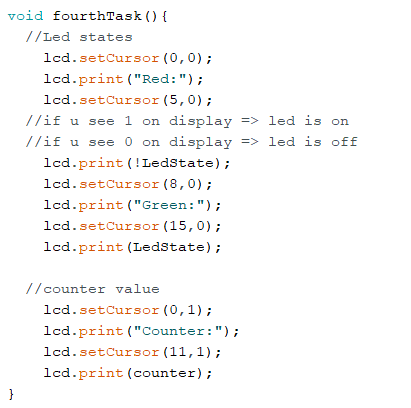
****

Fig 4 fourthTask()

**Sequential**

The sequential variant of implementation is the easiest to implement. In “setup()” we initialize buttons, LEDs and the LCD, but in “loop()” run all the tasks sequentially.

**Appendix 5**

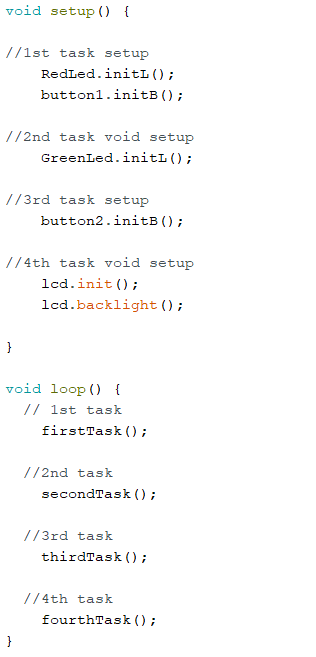


Fig 5 Sequential variant of implementation

**FreeRTOS**

The variant of implementation when we are using FreeRTOS is a bit more complicated. In this case all our code is situated in “setup()”. Besides the initialization of buttons, LEDs and the LCD, we have here creation of Tasks and starting the scheduler. But “loop()” we leave empty.

**Appendix 6**

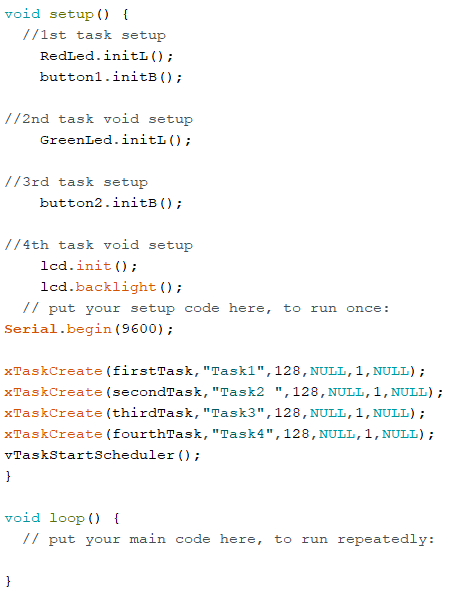


Fig 6 Variant of implementation by using FreeRTOS

**Bonus Task**

As a bonus task I decided to implement this using interruptions. I found this variant more complicated compared with the FreeRTOS variant and its realization consumed more time.

Interrupts are a set of priorities for certain processes executed by the controller. Interrupts allow certain important tasks to happen in the background and are enabled by default.

First that had to be done was to make the interruptions enabled (Fig 7) in “setup()”. The function “attachInterrupt” accepts 3 arguments where the first argument is for the button pin ( Arduino Uno board has 2 pins for interrupt: D2 and D3), the second one is ISR, and the third defines when the interrupt should be triggered. Interrupt Service Routines or ISRs are special kinds of functions that have some unique limitations most other functions do not have. An ISR cannot have any parameters, and they shouldn’t return anything.

**Appendix 7**



Fig 7 Interrupt setting up

Second that had to be done was to adopt laboratory work tasks to the ISR criterias. So now in the first task the program checks the “LedState” and depending on it turns on one LED, turns off another, changes the “LedState” value to the opposite and sets the value of text that will be shown on the display. The “thirdTask()” also checks the “LedState” value and in dependence on it increments the counter or no. All aforesaid you can see on Figure 8.

**Appendix 8**

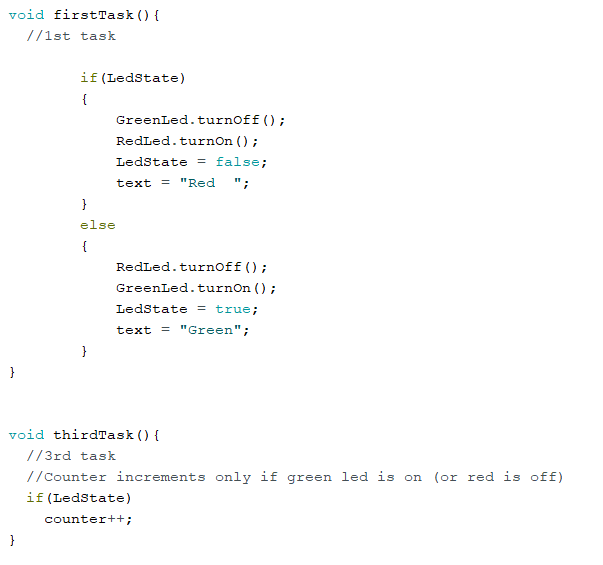


Fig 8 “firstTask()” and “thirdTask()” for bonus task

**Conclusions**

During this laboratory work I have studied work with FreeRTOS.h library and interruptions in Arduino.