# Lab 2: Tasks, Priorities, Synchronization, Queues, Interrupts

Farhang Nemati

In this lab you will practice working with real-time tasks using FreeRTOS tasks. You will practice the timing issues, interference among real-time tasks and synchronizing them on mutually exclusive accessed resources.

**Notice 1:** Before you start, to avoid overflow in pdMS_TO_TICKS (defined in projdefs.h) either in *FreeRTOSConfig.h* located in FreeRTOS libraries, e.g., in "C:\Program Files (x86)\Arduino\libraries\FreeRTOS-10.2.0-3\src", change *configUSE_16_BIT_TICKS* to 0 or override *pdMS_TO_TICKS* in *FreeRTOSConfig.h* as following:

```
#define pdMS_TO_TICKS(xTimeInMs)((TickType_t) (((unsigned long) (xTimeInMs)*(TickType_t) configTICK_RATE_HZ)/(TickType_t) 1000))
```

Where *TickType_t* is replaced by *unsigned long*

## A. Two tasks with priorities

### Delaying a task without sleeping

Usually for delaying a task you would use a FreeRTOS delaying functions, vTaskDelay(). Calling this function suspends the caller task (the task goes to sleep). However, you can delay a task without making it sleep (it should perform busy-wait for a given time interval).

### A.1.

Create a task that turns on a LED for 2s (2 seconds). The task will have a period of 3s. This means that every 3s the task turns on the LED for 2s. The task has to use busy-waiting for delays and should not suspend. Delay functions like *delay(ms)* in Arduino delay a task like busy-waiting. This means that the task doesn't leave the processor but is just waiting for some amount of time.

### A.2.

In this part you need to create and run two tasks; *task1* and *task2*. The tasks will be the same as the *task1* as you created in Section A.1. Each task will control its own LED. Assign the same priority to both of the tasks.

**What kind of scheduling is used for scheduling the tasks?**

### A.3.

Assign different priorities to the tasks from A2 and run the program.

**Do you see any difference?**

### A.4.

Change the tasks so that they perform their delays with suspending, i.e., use *vTaskDelay()* for delaying tasks. Assign different priorities to tasks.

**Do you see any change in the behavior of the tasks?**

## B. Three tasks with synchronization

### B.1.

In this part you will create 3 tasks (*task1, task2,* and *task3*). Set periods of all 3 tasks to 100ms. Assign priorities to all 3 tasks. It's recommended that *task3* has a lower priority than other 2 tasks. *task1* and *task2* will control the LEDs using the PWM that you implemented in Lab 1. The periodic task *task3* will read a command from the serial port. Depending on the command that is entered by the user from the serial should be sent to either *task1* or *task2* using shared variables. Each of the global variables are accessed by 2 tasks and thus MUST be protected using synchronization. Each *task1* and *task2* reads its command, use it and clear it (set it to -1). If there is no command for a task (meaning the command is -1), it should ignore the command. The program can hold only one command and if another command is entered before the previous one is consumed the new command should replace the old one. *task3* accepts a float number as the command according to the following protocol:

***n.m***

where *n* indicates the task number (1 for *task1* and 2 for *task2*) and *m* indicates the brightness of the LED controlled by task *n*. For example, if the command is 1.25 it means that the LED controlled by *task1* will be shining for 25% of its full brightness. If the command is 2.5 it means that the LED controlled by *task2* will be shining for 50% of its full brightness. Depending on *n*, *task3* delivers the command to either *task1* or *task2*. The LEDs have to continue with the same state until a command changes it. If the command is not a valid one, task3 has to ignore it.

### B.2.

Change the program so that received commands can be queued by *task1* and *task2*. The commands have to be queued in a FreeRTOS Queue; there should be 2 queues; one queue to queue commands for *task1* and one that queues commands for *task2*. Depending on *n* part of the received command, *task3* puts it into one of the queues. The length of the queues has to be limited to 5 commands. If a new command arrives while the corresponding queue is full the command has to be thrown away. If a task (*task1* or *task2*) tries to read a command from its corresponding command queue while the queue is empty, the task has to block until a command is in the queue. Change the periods of *task1* and *task2* to 5 seconds so that the user can manage to put multiple commands into queues.

### B.3.

Use a button to reset the command queues, i.e., clear all the existing commands in the queues. The button has to trigger an external interrupt. On Arduino Uno pins 2 and 3 are external interrupt pins. Implement an Interrupt Service Routine (ISR) for the interrupt where the contents of queues are cleared.

## Examination

To pass the lab hand in a short report (where you answer the questions asked in this document) and your code. You must hand in only in Blackboard! You also need to demonstrate the programs for the lab assistant in one of the lab sessions. The lab assistant may ask you questions related to lab task.