ECEN-361 Lab-04:FreeRTOS & Multi-tasking

# NAME: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# Introduction and Objective of the Lab

In Lab-02, we saw how individual counter blocks could initiate tasks and work like a multi-tasking system. Each timer block would produce an interrupt, launch the task, then re-start its count. In “parallel” we had

* 3 different LEDs blinking
* A timer cycling thru, displaying each of the Seven-Segment display digits.
* Random Reaction Timer counting
* A response-timer keeping track of how long till a button was pressed.
* A serial port timer sending UART data to the USB-COM: port.

While this operated like a multi-tasking system, the reality is that there were very strict limitations and flexibility to this system. Our Nucleo was running out of timer blocks, there was no controlled/shared memory, interrupts had to planned such they were never “nested,” etc. This brute force approach is not scalable.

In Lab-03 we examined a simple approach to looking at how to launch multiple jobs per a scheduler.

Our next step is to implement a true, commercial-grade RTOS, which gives us all the infrastructure needed to implement multi-tasking. Instead of using multiple counters/timers/interrupts, we will now let the RTOS manage task swapping, memory management, and all else, based on a single timer: SYSTICK.

FreeRTOS will be the RTOS of choice for this class. The benefits and reasons for this system are reviewed in class, and it is supported directly with the STM32CubeIDE that we use. This lab will be the first use of FreeRTOS in our labs and has the following objectives:

Part 1: Introduction of FreeRTOS with a process-based ‘blinky’ project.

Part 2: Creation of tasks to do the same things we did in Lab-02, but with processes controlled by FreeRTOS instead of setting-up and controlling all the timers.

For each of the parts, follow the instructions, then fill in answers to the questions. Expected answers are indicated in the boxes with red text/spaces to fill in answers.

# Lab Instructions

## Part 1: Starting with the YT-based, add the MultiBoard into the project

Before the lab, you should’ve followed the instructions for the Pre-lab-4 Exam, and built a ‘blinky’ that runs from FreeRTOS. Steps:

Steps:

1. With that project working, power-down the Nucleo, add on the multi-function shield, and start your FreeRTOS blinky again.

2

## 2 Pts.

1. Which light blinks on the multiBoard?

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. Do the lights (MultiBoard & main Nucleo User light (LD2) blink the same?

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Finally, locate the process in the code where the on-board light is toggled. Look for:

**HAL\_GPIO\_TogglePin(GPIOA,GPIO\_PIN\_5);**

**osDelay(2000);**

With the MultiFunction Board in place, change that line to toggle the LED D4 instead:

**HAL\_GPIO\_TogglePin(LED\_D4\_GPIO\_Port,LED\_D4);**

## Part 2: Using the Multi-Board and Launching other FreeRTOS tasks

By now, you should’ve completed the assignment ([HERE](https://byui.instructure.com/courses/251502/files/113360890?module_item_id=31336740)) about getting on and using GitHub.

Start by accepting the assignment in GitHub Classroom for ECEN-361-Lab-04. Clone the repo.

### Add the repo project to your workspace.

Like previous labs, the submission for this lab is simply the repository that you’ll modify. Your modifications get pushed back to github.com. Your responses, as recorded in this file, will be checked along with your running project. When your repository is finished and pushed, then copy the URL and submit it in the Lab-04 Assignment on iLearn.

After you’ve cloned the repo, Import with File/Import and point to the directory of the newly cloned project.

Clean Project then Build Project:

There should be no errors or warnings.

### Run the project.

The project should simply blink the D2\_LED once per second.

*Note that the D1\_LED is not being used because it is tied to the built-in user LED on the STM32 board. These two are in conflict with one board treating it as active-high, and the other as active-low. So, D1\_LED is unused.*

No seven-segment display.

### Create 3 more blinking events with tasks (no interrupts or timer blocks this time)

Note that to add a new task in FreeRTOS, three things have to be coded. These are labelled with comments in “main.c” as “Task-Part-A,” “Task-Part-B,” and “Task-Part-C”. As they are discussed below – find these comments in the code for reference.

1. /\*\*\*\*\*\*\* Task-Creation-Part-A \*\*\*\*\*\*\*\*\*/  
   Declare a prototype for the function (this is a requirement for the C-compiler to link)
2. /\*\*\*\*\*\*\* Task-Creation-Part-b \*\*\*\*\*\*\*\*\*/  
   Write the task process itself
3. /\*\*\*\*\*\*\* Task-Creation-Part-c \*\*\*\*\*\*\*\*\*/  
   Launch the task by putting it in the scheduling queue

Note that the “StartDefaultTask “ is required when the system is built. That task currently blinks the D1\_LED at 1000mS. Using the single task in the code as a prototype (“StartDefaultTask”), create three more tasks that blink:

D2\_LED: Once every 500 mS.

D3\_LED: Once every 250 mS.

D4\_LED: Once every 125 mS.

When completed answer the following questions:

## 3 Pts.

True/False: I got all 4-LEDs are blinking – looks like binary counter! \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## Part 2-2 Questions (5pts)

Now add one final task that display a counter on the Seve-Segment LED bars. Count up from 0, and increment the count once per … 1.5Sec

## 5 Pts.

True/False: Got the counter running in parallel with everything else: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## Extra Credit Ideas (5 pts) for any. Only 1 counted (max: 5 pts)

## 2 Pts.

1. Stop one of the LED processes when the digit count gets to 20, -- quit running the D3-LED Toggle task. And explain how you did it? Global variable? Or read about and use the oSSuspend task???

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. Explore any difference between the two “wait” calls: HAL\_Delay and OsDelay.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. Eliminate the SevenSegment refresh routine, currently based off timer17, so that it refreshs like any other process to give the appearance of all 4 digits being turned on at the same time.

Explain what you did:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. Use one of the push buttons from an earlier lab to set up an interrupt such that it doubles the count frequency of the 7-Segment LED counter to go faster and faster. Explain how you did it.

When completed with everything, commit/push the repo, and submit the URL of the checked-in repo to the iLearn Lab submission assignment.