CSSE3010 Stage 4

FreeRTOS Introduction

(Closed Shoes MUST BE Worn in the labs)

1 Assessment

- Git due 4pm Tuesday in week 9 .
- There is no demo. Do not attend a lab session. All lab sessions are cancelled
- No worksheet or workbook is due.
- Course Marks: 10%
- Electronic Course Profile Pass Hurdle: Must be submitted.

2 Resources

- Nucleo-F429ZI platform
- Joystick

3 Structure

Your final stage code must be titled `main.c` and saved in your `stage4` folder.
PATH: `csse3010/stage/stage4`

Listed below are the required mylib libraries, that you will need to develop for this stage. They should be saved in your `mylib/mylib_library_folder` folder.
PATH: `csse3010/mylib/mylib_library_folder`.

- lta1000g HAL and OS Library
- Joystick HAL and OS Library

4 Introduction

FreeRTOS is a commonly used realtime operating system for various embedded platforms. The stage will introduce you to the basics of using FreeRTOS Tasks, Queues and Semaphores. Refer to the lecture notes and www.freertos.org for more information.

4.1 FreeRTOS Tasks

A FreeRTOS Task is a process of execution, used to perform a function. Tasks must run continuously. Each Task has a priority level and a memory stack allocation. The priority of each task determines the sequence in which the task scheduler will execute the tasks.

4.2 FreeRTOS Binary Semaphores

A FreeRTOS Binary Semaphore is used for synchronisation of tasks. A task can be blocked or suspended until unblocked by a Binary Semaphore signal. Binary Semaphores are used by interrupts to alert tasks that an interrupt has occurred.

4.3 FreeRTOS Queue

A FreeRTOS Queue is a memory object that used to share information between tasks. The Queue allows a task to write and read variables with other tasks. A Queue can hold a number of fixed data items (integer, float, etc). Multiple tasks can write or read to a Queue simultaneously and a task can block or suspend on a Queue read or write.

4.4 FreeRTOS Memory Management

In FreeRTOS, memory can be assigned by the user for tasks, queues and other types of FreeRTOS elements. Typically this involves assigning a Task's stack. The minimal amount of memory required for a task, depends on the platform CPU (e.g. size of CPU register file). In FreeRTOS memory management is performed, using a Heap model. There are currently 5 different memory heap models that can be used.

- Heap 1: Does not allowing freeing of memory.
- Heap 2: Best Fit algorithm and allows memory to be freed.
- Heap 3: Implements Malloc and Free functions
- Heap 4/5: First Fit algorithm and allows memory to be freed.

4.5 FreeRTOS Configuration

FreeRTOS can be configured to enable or disable many of its features. The FreeRTOSConfig.h header file contains various definitions that can be used to control features such as preemtpion and task minimal stack size. Various task control functions are not included by default and usually have to be 'switched on' with appropriate definitions in the file. The makefile will also contain various sources (include other makefiles and compiler definitions that can configure FreeRTOS, e.g. memory heap type.

5 Timer with Freertos

Create a dual timer with the LED light bar, using tasks and queues. Control the start/stop of the timer with a the joystick Z signal.

5.1 Worksheet Tasks - Not Assessed

The worksheet contains helpful questions related to this stage. Worksheet is not assessed and is not compulsory for you to attempt. The worksheet task is designed to help you complete stage 4 .

5.2 Design Tasks - Do Not Attend Labs

The code required for the design tasks must be uploaded to your git repository by the specified due date in week 9 . The lab sessions are cancelled. Alternate online sessions will be organised on BlackBoard.

5.2.1 Git Requirements

You MUST have your stage and mylib code in git, in the correct folders and it must be able to be compiled by the markers, in order to be assessed for this stage. If your code only compiles on your computer and cannot be compiled by the markers, then it is not acceptable for marking. Please ensure that you follow the previously given git requirements.

5.2.2 mylib Setup

You MUST FOLLOW the Template Code given in the `sourcelib/examples/templates/mylib` folder. Your mylib code must meet the guidelines specified in the mylib and platform build guides, on Blackboard. You MUST create the right file structure in the `mylib` git folder.

You will create mylib OS library files for the lta1000g and joystick. Refer to the mylib OS and CLI peripheral guide, on Blackboard.

5.2.3 Design Task 1: lta1000g and Joystick OS mylib driver

Implement the lta1000g and joystick OS mylib drivers. Follow the mylib OS and CLI peripheral driver guide, given on BlackBoard. The lta1000g OS mylib driver must call the lta1000g HAL driver functions. The joystick OS mylib driver must call the joystick HAL driver functions, and should use a GPIO interrupt for the Z signal and provide a binary semaphore to signal when Z has been pressed. Reading of X and Y values, is not required. You should not use the ISS with the joystick OS mylib driver. Use previously specified pins for the lta1000g and joystick.

5.2.4 Design Task 2: Timer

Implement a dual timer with the lta1000g ledbar, using tasks and queues. The left timer increment every 1 seconds and the right timer increments every 0.1 seconds. Create three tasks, shown in Table 1 to control the timers – you may need to adjust the task priorities to ensure correct operation. `TaskTimerLeft` and `TaskTimerRight` tasks control the left and right timer values. Each timer value is 5 bits.

Table 1: Timer Tasks and Queue

Element	Description	lta1000g Segments
`TaskTimerLeft`	Left Timer – counts every 1s	
`TaskTimerRight`	Right Timer – counts every 0.1s	
`TaskTimerDisplay`	Show timer values: left right	 9-5 4-0
`queueTimerDisplay`	Used to transfer left and right timer values to `TaskTimerDisplay`	

The `TaskTimerDisplay` task should display the correct values on the lta1000g ledbar, using the lta1000g mylib OS driver. Use a queue (`queueTimerDisplay`) to send left and right timer values to `TaskTimerDisplay`. Use the following `struct` to send messages containing the timer type (left or right) and the timer value to be displayed. Note: you can modify the `struct` to include other fields.

```
struct dualTimerMsg { char type;  //type is either l or r
  unsigned char timerValue;
};
```

Refer to os/fr_led_flashing and os/fr_queue examples. Also refer to `www.freertos.org` *for the latest FreeRTOS API.*

5.2.5 Design Task 3: Timer Control

Use the joystick Z signal to control the stopping and starting of the dual timers. Use the joystick mylib OS Z interrupt and binary semaphores to signal stop or start commands to the `TaskTimerLeft` and `TaskTimerRight tasks`. Use previously assigned pin values for the joystic Z signal.

Refer to os/fr_led_flashing and os/fr_semaphore examples. Also refer to `www.freertos.org` *for the latest FreeRTOS API.*

6 Final Demo

Implement all design tasks as one demo.

7 Criterion

The stage is marked according to the criterion outlined in the table below. If you fail to demonstrate sufficient understanding and functionality you will not be allowed to repeat the stage. You must pass the pre-marking checks or your stage will not be marked. **All code assessed for the stage must be your own work.**

7.0.1 Pre Marking Checks

The following criteria **must** be met **before** you are allowed to be marked.

Check	**P/F**
Your latest stage and mylib code **must** be in git.	
Your git repository must be up to date with the latest version of sourcelib.	
Your git repository conforms to the git guidelines, provided.	
Your stage code must build without errors (including with the automated build process).	
Your mylib and top comments are correctly filled out.	

Failure to meet pre-marking checks will mean that your stage is not marked.

7.0.2 Marking Criterion

You must be able to combine all design tasks, into the same file and demo all design tasks, without reprogramming your Nucleo. Note: Your code must pass the Pre-marking checks or it will not be marked.

Design Task 1: lta1000g and Joystick OS mylib	
0	lta1000g and joystick OS mylib drivers are not implemented.
1	lta1000g and joystick OS mylib drivers are partially implemented or errors occur with both driver.
2	lta1000g and joystick OS mylib drivers are fully implemented but errors occur with one driver.
3	lta1000g and joystick OS mylib drivers are fully implemented and operate correctly.
Design Task 2: Counter	
0	None of the required timer or display tasks are not implemented.
1	Timer and display tasks are partially implemented or frequent errors occur).
2 or 3	Timer and display tasks are fully implemented but errors occur.
4	Timer and display tasks are fully implemented and operate correctly.
Design Task 3: Counter Control	
0	Timer control with the joystick Z interrupt signal is not implemented or a semaphore is not used to signal start or stop of the timers.
1	Timer control with the joystick Z interrupt signal semaphore is implemented but errors occur.
2	Timer control with the joystick Z interrupt signal semaphore is implemented and operates correctly.

Code Style	
0	One or more style errors found in one tutor selected stage file.
1	No style errors found in **one** tutor selected stage file.

Student Name:

Student Number	**Mark (/10)**	**Marker**	**Date**