CG2271 Real Time Operating system AY 2017/18

Lab 4 – Introduction to FreeRTOS and Real-Time Scheduling (8% of your final grade)

Lab Lessons: Wednesday 25 Oct
IVLE Submission of answer book: Friday 27 Oct midnight
Demo: Wednesday 1 Nov

1. Introduction

In this lab, you will be using FreeRTOS for the first time and experiment with real-time scheduling. **This lab is worth 39 marks for the answer book and 9 marks for the demo for a total 48 marks.**

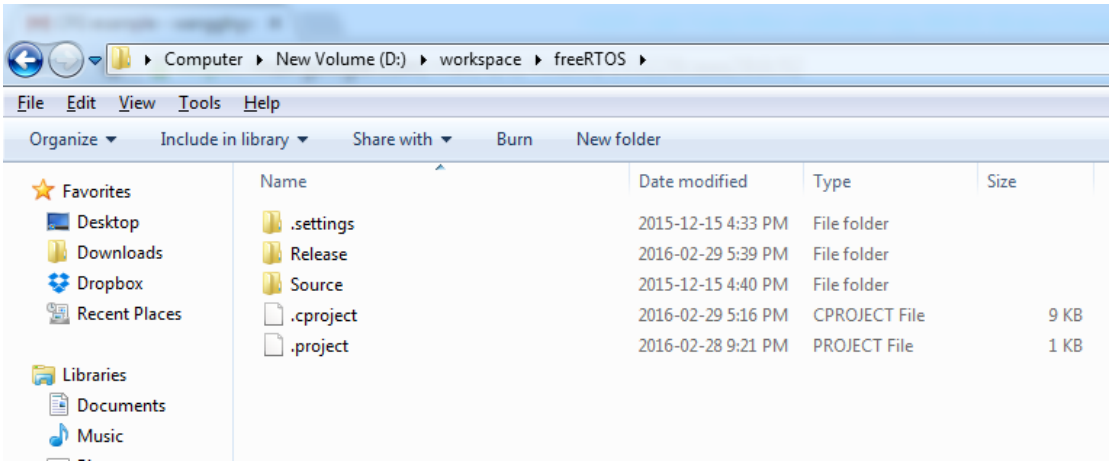
FreeRTOS is an open source, professionally developed and robust real-time operating system kernel. FreeRTOS is supported on multiple platforms. In particular, it is de-facto standard operating system for micro-controllers and small micro-processors.

In this lab, you are going to compile FreeRTOS as a static library and link this library to your applications. FreeRTOS port for ATmega328P in your Arduino board is not available. Therefore, we have taken the ATmega323 port and modified it for ATmega328P.

2. Download and Compile FreeRTOS

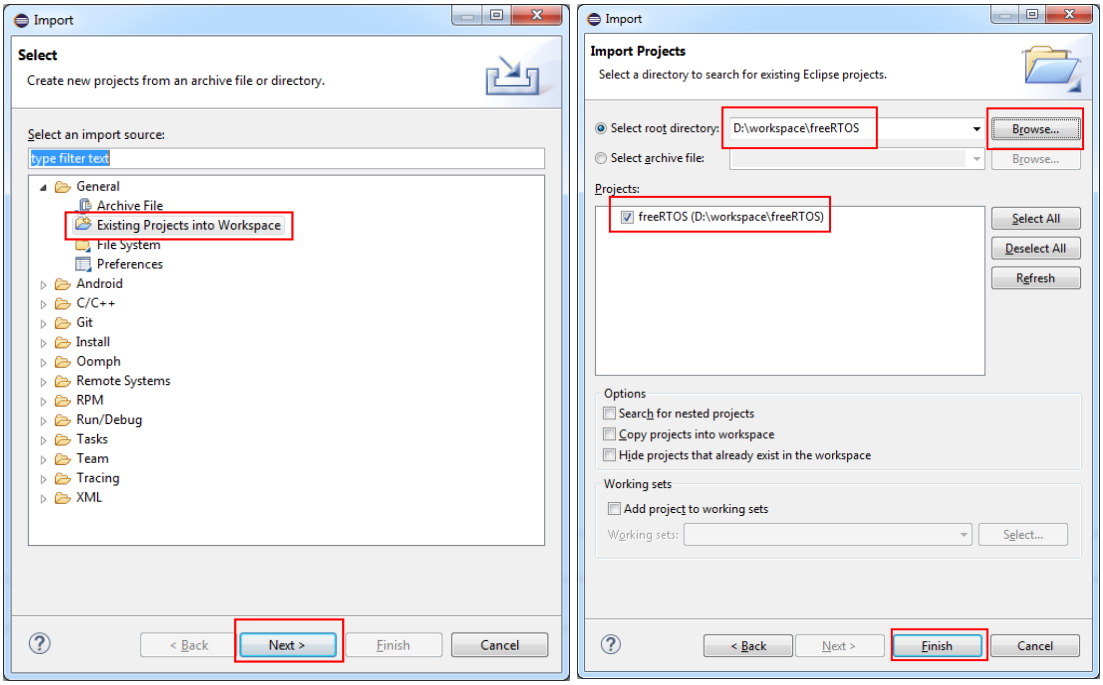
2.1 Download and decompress FreeRTOS

First, you need to download FreeRTOS from IVLE as a .rar file (FreeRTOS.rar) incorporating the entire source code and project settings. Decompress the downloaded file FreeRTOS.rar to your Eclipse workspace. After decompression, the directory structure should look like the following:

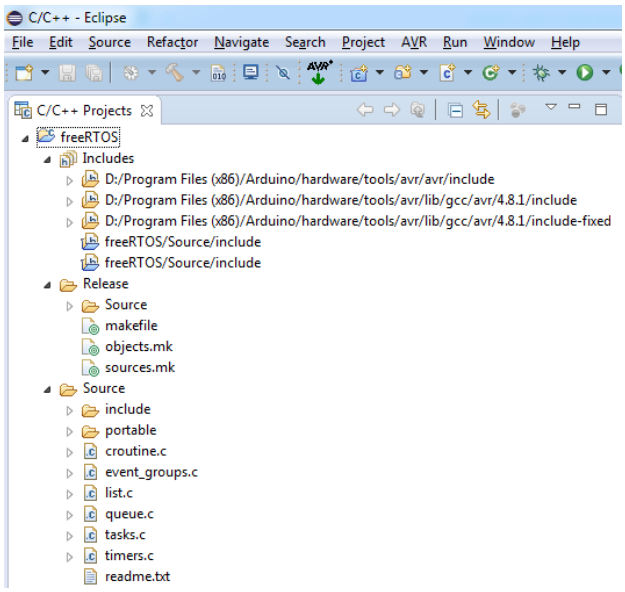


2.2 Import and compile FreeRTOS

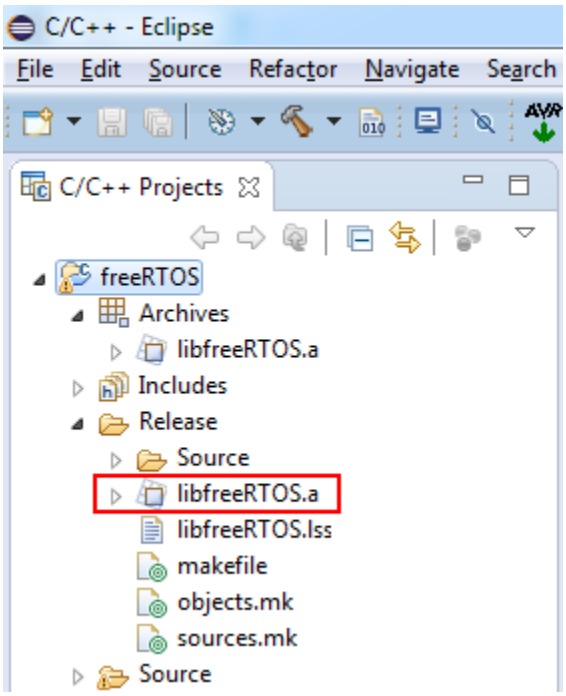
You are going to create a static library containing all FreeRTOS functions. Click eclipse.exe to get Eclipse running. From Eclipse go to "File→Import…". In the "Import" page, select "General→Existing Project into Workspace", click "Next" button to continue. In the "Import Projects" page, select "Select root directory" and browse the path to your FreeRTOS directory. Then your FreeRTOS project will be shown in list box; select this project and click "Finish".



This project can generate a static library contains all freeRTOS functions. If everything goes well, the project will appear at the left side of your Eclipse workspace. You will have a project structure like this:



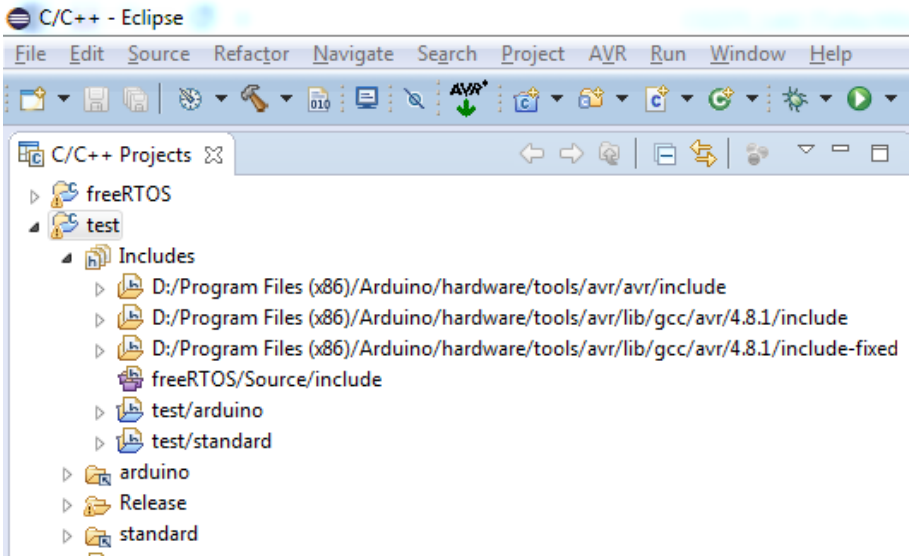
Please look through “Includes” list in the project structure to make sure everything is set properly. In the menu bar, click “Project → Build Project” to build this project. Finally, you will find a static library in directory “Release” named libfreeRTOS.a:



3. **Create a New Project with FreeRTOS**

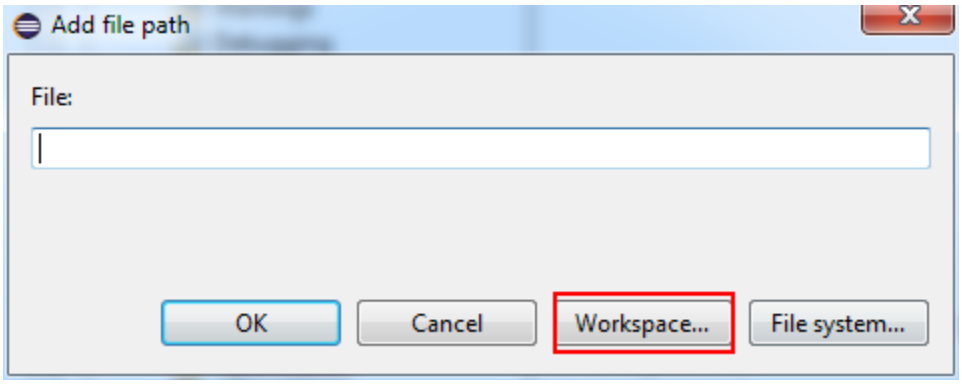
   In this step, you are going to create a simple application and link it to the static FreeRTOS library generated in the last step. Everything you are going to do now are quite similar to what you have done in previous Labs.

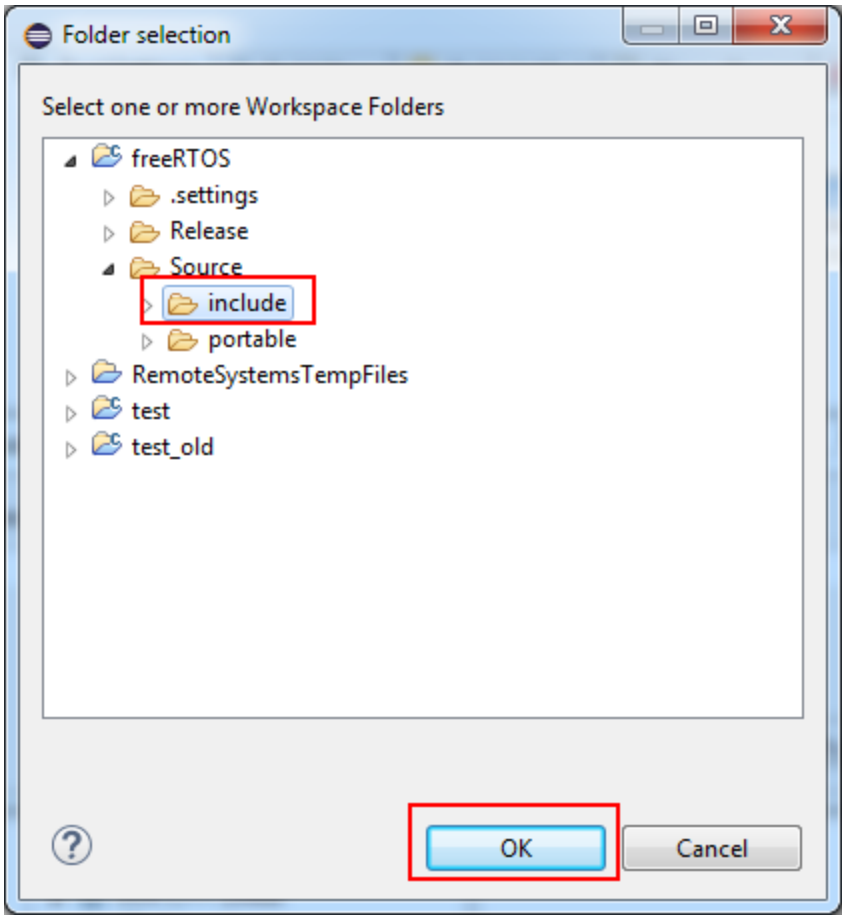
   As described in Lab 1, from “File→New→C++ Project”, create a new “Empty Project” with project type “AVR cross Target Application”. Remember to deselect the “Debug” configuration and select “ATmega328P” as MCU type with frequency 16000000Hz. After creating the new project, you should link Arduino Core files, “arduino” and “standard” directories to your project. You should add “arduino” and “standard” directories to your compiler include paths. If you have forgotten how to do this, please refer to the description of lab 1. After this step, the project should look like the following:



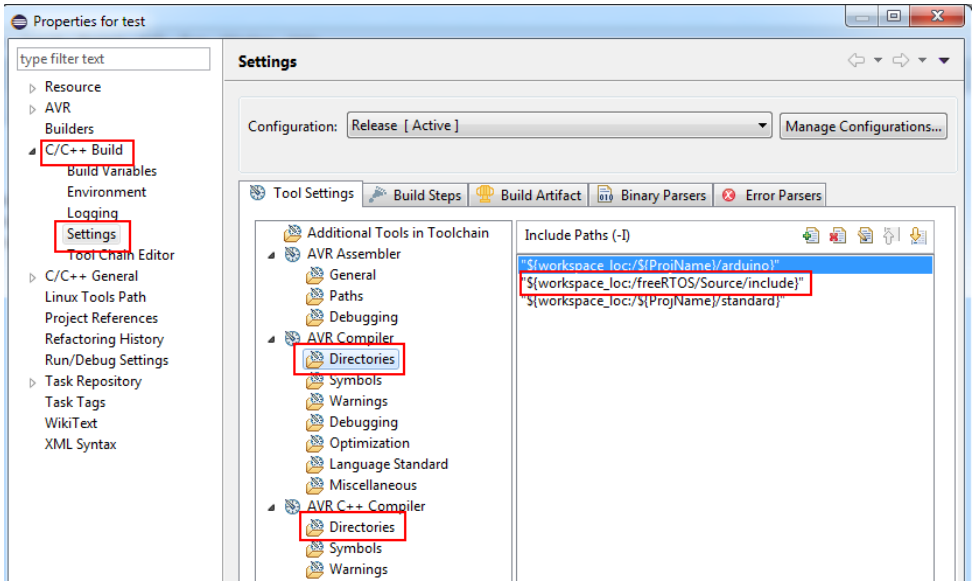
You need to add the folder “freeRTOS/source/include” from your FreeRTOS static library project to the include directory path of your application project. Right click your project and select “Properties”. In “C/C++ Build→Settings”, click “AVR Compiler→Directories”, click “plus”. In the “Add file path” dialog box, click “Workspace…”, browse the path to “freeRTOS/source/include”, click “OK” to return to the previous dialog box. Click “OK” again to return to “Properties” page. Perform the same steps for AVR C++ Compiler→Directories

Note that directories “standard” and “arduino” should also be in “Include Paths”; if not, please add them to the paths.

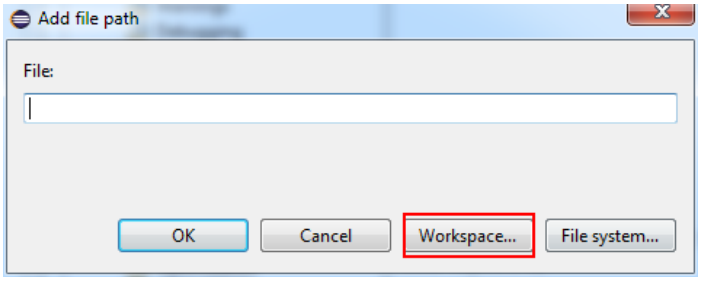



Folder selection
Select one or more Workspace Folders
freeRTOS
.settings
Release
Source
include
portable
RemoteSystemsTempFiles
test
test_old
OK
Cancel

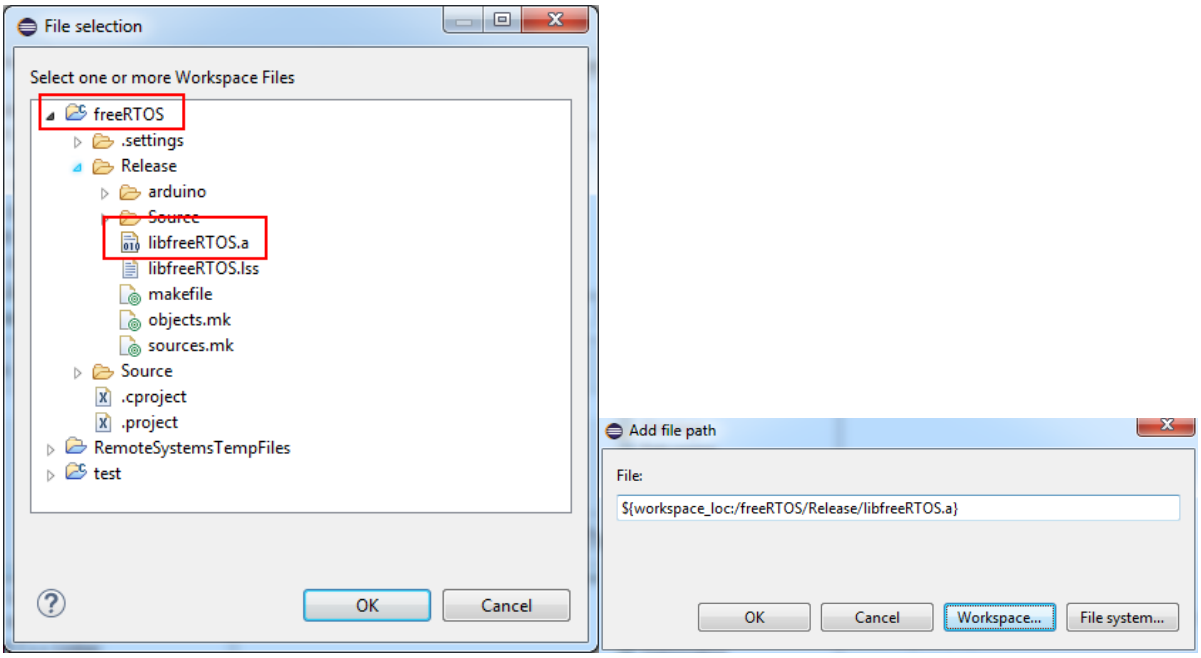
Now “Include Paths (-I)” should look like the following:



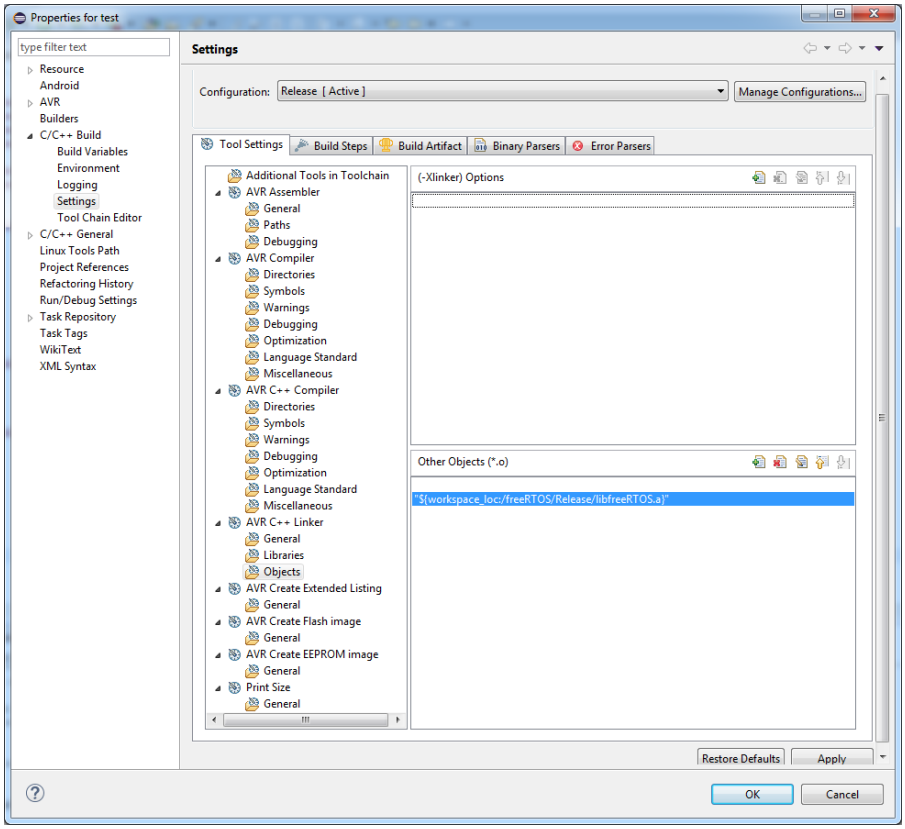
You also need to configure the linker so that it can find the symbols defined in FreeRTOS library. Select “AVR C++ Linker→Objects”, then click “plus” and the dialog box will appear:



Press button “Workspace…”. In the “File selection” page, browse the path to “freeRTOS/Release/”, select libfreeRTOS.a and click OK to return to “Add file path” page.



Click “OK” to return to “Properties” page. After all the steps, the libfreeRTOS.a would be added to “Other Objects (*.o)” list box as shown in the following



Finally, click “OK” to close “Properties” window.

4. **Add Code to Project and Compile the Project**

   Create a new file named “test.cpp” and add the code below.

```
#include <Arduino.h>
#include <avr/io.h>
#include <FreeRTOS.h>
#include <task.h>

#define STACK_SIZE   200
#define LED_PIN      6

void task1(void *p)
{
       for (;;) {
         digitalWrite(LED_PIN, HIGH);
         delay(1000);
         digitalWrite(LED_PIN, LOW);
         delay(1000);
       }
}

void setup()
{
       pinMode(LED_PIN, OUTPUT);
}

void loop() {
       xTaskCreate(task1,           // Pointer to the task entry function
                   "Task1",         // Task name
                   STACK_SIZE,      // Stack size
                   NULL,            // Pointer that will be used as parameter
                   1,               // Task priority
                  NULL);           // Used to pass back a handle by which the
created task can be referenced.
       vTaskStartScheduler();
}
```

Click “Build Project (Ctrl + B)” in “Project” menu to compile this project.

If you use MAC OS, you may encounter the compilation error of “TIMSK1 could not be resolved”. Please add a declaration of ***TIMSK1*** at the top “freeRTOS\Source\portable\GCC\ATMega328p\port.c” file:

```
#define TIMSK1 _SFR_MEM8(0x6F)
```

You are going to use a simple circuit as follows: connect positive leg of an LED to digital pin 6 while the negative leg is connected to a 330 ohm resistor (orange-orange-brown). The other leg of the resistor is connected to GND.

Finally connect the USB cable to board. Click “AVR” button to upload your project to device. You will see the LED is blinking.

5. **FreeRTOS Task Creation and priority**

Question 1 (2 marks)

Read FreeRTOS documentation (http://www.freertos.org/), lecture slides on “Task Management” (slides 37—52), and Chapter 8.1 Inter-process Communication Primitives of FreeRTOS in the book "Real-Time Embedded Systems: Open-Source Operating Systems Perspective" by Richard Zurawski. Taylor & Francis Group LLC 2012 (from IVLE Files → Reading → Ch8.pdf).

A) Which task has higher priority, a task with priority 1 or a task with priority 2? **[1 mark]**

B) Priority 0 is quite special. Explain what is special about priority 0 and why. Is it advisable to assign priority 0 to a task? **[1 mark]**

6. **Building the circuit for this lab**

The circuit for this lab is quite straightforward. Simply connect one LED each to digital pins 6, 7, 8, 9. Please make sure to add in resistors and connect to ground as necessary and described in Section 4.

7. **Periodic tasks and priority**

Question 2 (13 marks)

You will now learn how to create periodic tasks in FreeRTOS. You will create a simple periodic task that blinks the LED connected to digital pin 6. The task executes once every second, i.e., the task has a period of 1 sec. Every time the task executes, it switches the LED from ON state to OFF state and vice versa to create a blinking LED. In other words, the LED stays ON for 1 sec, OFF for 1 sec, and the pattern is repeated forever with a period of 2 sec. **Note that you are NOT allowed to use delay( ) or mydelay ( ) functions to solve this problem without creating a periodic task in FreeRTOS.**

Look up the following three functions from FreeRTOS API reference:

`xTaskGetTickCount( ), vTaskDelayUntil ( ), vTaskDelay ( )`

A) What does the function `xTaskGetTickCount( )` return? **Note that the default tick length for the platform and settings we have created is 1ms**. **[1 mark]**

B) What does the function `vTaskDelay( )` do? **[1 mark]**

C) What does the function `vTaskDelayUntil( )` do? How does it differ from `vTaskDelay( )`? **[2 marks]**

D) Should you use vTaskDelayUntil ( ) or vTaskDelay ( ) to create a periodic task? Why? Hint: Refer to the FreeRTOs API reference. **[1 marks]**

E) Create the periodic task to blink the LED on digital pin 6 where the task period is 1 sec as mentioned before. Your solution should use a subset of the above three functions: `xTaskGetTickCount( ), vTaskDelayUntil ( ), vTaskDelay ( ).` Hint: Considering `vTaskDelayUntil and vTaskDelay,` only one of them will produce the desired behavior. **[3 marks]**

F) Create three periodic tasks. Task 1 blinks the LED connected to Pin 6 and has a period of 1.5 sec. Task 2 blinks the LED connected to Pin 7 and has period of 2 sec. Task 3 blinks the LED connected to Pin 8 and has a period of 4 sec. Assign priorities to the tasks according to rate monotonic scheduling policy. **[5 marks]**

8. **Software PWM**

Question 3 (5 marks)

Previously in Lab 2, you have used the hardware PWM available in the micro-controller to control the brightness of the LED through `analogWrite( )` function. To explore task scheduling, in this lab you will implement a software-driven PWM. Please read up on PWM from Lecture slides on "Input/Output Interfacing" (slides 14-18).

Create a FreeRTOS task that gradually increases the brightness of the LED on digital Pin 6 from zero to the maximum brightness level in steps of 5% and then repeats the process. The task should alternatively switch on and off the LED using pulses of different widths (duty cycle) for different brightness levels. The blinking period for the task should be very fast (e.g., more than 30Hz; you can use 50Hz in this question) to create the right illusion. Otherwise, you will observe the blinking. If the blinking is fast enough, human eye will not be able to follow the blinking and will observe the average brightness of the LED.

You should use delay ( ) in your answer.

You are NOT allowed to use `analogWrite( )` function for this question.

9. **Rate Monotonic Scheduling**

Question 4 (11 marks)

In this question, we will explore RMS scheduling policy in FreeRTOS. You will create a periodic task set with four tasks and the following parameters

Task number	Digital Pin	Period	WCET
1	6	5000ms	1500ms
2	7	10000ms	2500ms
3	8	10000ms	1700ms
4	9	20000ms	700ms

We are going to emulate the computation done by a task by blinking LED as follows. Each iteration of the while loop below takes roughly 100ms. Hence, to emulate the WCET of 1500ms for Task1 for example, the loop has to iterate 15 times for each instance of execution of Task1. That is, the code below represents the computation corresponding to a task for one instance.

```
count = 0;
while (count < TASK_WCET) {
      digitalWrite(pin, HIGH);
      myDelay(50);
      digitalWrite(pin, LOW);
      myDelay(50);
      count +=100;
}
```

The code of myDelay function is as follows:

```
void myDelay(int ms) {
  for (int i = 0; i < ms; i++) {
       delayMicroseconds(1000);
  }
}
```

A) Execute the task set using RMS scheduling policy. Assign appropriate priorities. Break ties arbitrarily if two tasks have the same period. That is, do not assign the same priority to two tasks and use four unique priority values. The objective of the questions is to experience scheduling; so you should use `myDelay( )` function given here and not replace it with vTaskDelay. **[5 marks]**

B) Draw the schedule for the task set under RMS for the hyper-period and ensure that you observe the same behavior from the blinking of the LEDs. You may use the time granularity (1 sec or 100ms) that is easiest for you to draw the schedule clearly. **[2 marks]**

C) Now use the RMS scheduling policy again for the task set but assign the same priority to the two tasks with the same period, that is, use three unique priorities. Observe the schedule from the blinking of the LEDs.

Is the schedule different from the one you observed in Question 4(A) and 4(B)? If yes, explain the difference and draw the new schedule. **[4 marks]**

10. **Cyclic Executive**

Question 5 (8 marks)

In this question, you will use the same task set as in Question 4 and the same parameters.

Task number	Digital Pin	Period	WCET
1	6	5000ms	1500ms
2	7	10000ms	2500ms
3	8	10000ms	1700ms
4	9	20000ms	700ms

Read the lectures slides "Real-Time Scheduling part 2" on Cyclic executive.

A) Draw the cyclic executive schedule for the task set. **[2 marks]**

B) Implement the cyclic executive schedule in FreeRTOS using a sequence of functions and **not with tasks**. Although in the lecture we use `wait_for_interrupt ( )` function to synchronize at minor cycles, we will not rely on them as we have not discussed timer interrupts in FreeRTOS yet. Instead, you should create a `while (1)` loop for the entire schedule inside a single task, and use the combination of `xTaskGetTickCount( ), vTaskDelayUntil ( )` functions to synchronize at minor cycles. **[6 marks]**

11. Demo Session

Submit the Lab 4 answer book to IVLE Files → Lab4-Submissions → Your lab group. Please ensure that you submit to the correct Lab group. You should make one submission per team.

You will demonstrate Question 2(F), 3, 4(A), 4(C), 5 to your lab TA at the start of your lab session for Lab 5. Please be punctual for your demonstration. Your demonstration is worth 10 marks.