CG2271 Real Time Operating Systems
Lab 4 – FreeRTOS and Real-Time Scheduling
Answer Book

Name: Deepak Buddha	Matric Number: A0155454A
Name: Anton Chua	Matric Number: A0161811N

Submission Deadline: Friday 27 Oct midnight
Submit under IVLE → Files → Lab4-Submissions → Your Lab Group

Question 1A (1 mark)

A task with priority 2 has a higher priority than a task with priority 1.
In freeRTOS, higher number value signifies higher priority

Question 1B (1 mark)

Priority 0 is used for the idle task. The idle task is created automatically when the RTOS scheduler is started to ensure there always at least one task that is able to run. It is created at priority 0 (or lowest possible priority) to ensure it does not use any CPU time if there are higher priority application tasks in the ready task.

The idle task is responsible for freeing memory allocated by the RTOS to tasks that have been deleted during the execution of vTaskDelete. Hence it is not advisable to assign priority 0 to any other task as it might interfere with the the function of the idle task or otherwise the system may run out of memory not because there is not enough but because the idle task was unable to free it fast enough before reuse.

Question 2A (1 mark)

The function xTaskGetTickCount () returns the counts of ticks since vTaskStartScheduler was called, and in the platform and settings specified in our lab, since each tick length is 1ms, this function returns the amount of time passed since vTaskStartScheduler was called in milliseconds.

Question 2B (1 mark)

The function vTaskDelay () delays a task for a given number of ticks. In the context of our lab, with each tick length being 1ms, this function delays a task for a given amount of time in milliseconds relative to the time which vTaskDelay() is called.

Question 2C (2 marks)

The function vTaskDelayUntil () delays a task until an exact specified amount of time, which is the absolute time at which the task will unblock. It differs from vTaskDelay() in this aspect as vTaskDelay() is a relative time delay to when the function is called but vTaskDelayUntil() delays the task with an absolute(exact) time specified.

It was also different when the wake time specified to vTaskDelayUntil() has already past, in which case vTaskDelayUntil () will return immediately without blocking the task.

Question 2D (1 mark)

vTaskDelayUntil () should be used instead of vTaskDelay () to create a periodic task. This is primarily because of the the fact that vTaskDelay () will cause a task to block for the specified number of ticks from the time vTaskDelay () is called, making it difficult to use it to generate a fixed execution frequency as the time between the task unblocked following a call to vTaskDelay () and that task next calling vTaskDelay () may not be fixed.

This is not the case with vTaskDelayUntil () as the time specified is absolute.

Question 2E (3 marks)

Refer to function task2E in code segment at end of report.

Question 2F (5 marks)

Refer to functions task2f1, task2f2 and ask2f3 in code segment at end of report.

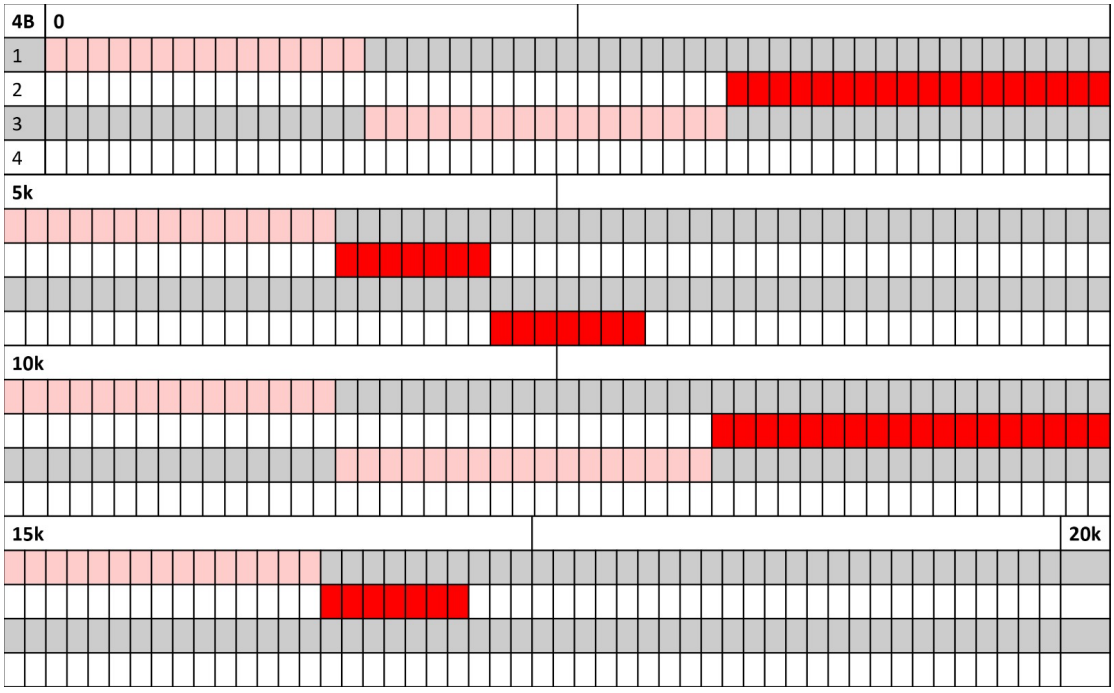
Question 3 (5 marks)

Refer to function task3 in code segment at end of report.

Question 4A (5 marks)

Refer to functions task4t1 to task4t4 and myDelay in code segment at end of report. Chosen priority order : Task 1: 4, Task 2: 2 , Task 3: 3, Task 4 : 1.

Question 4B (2 marks)



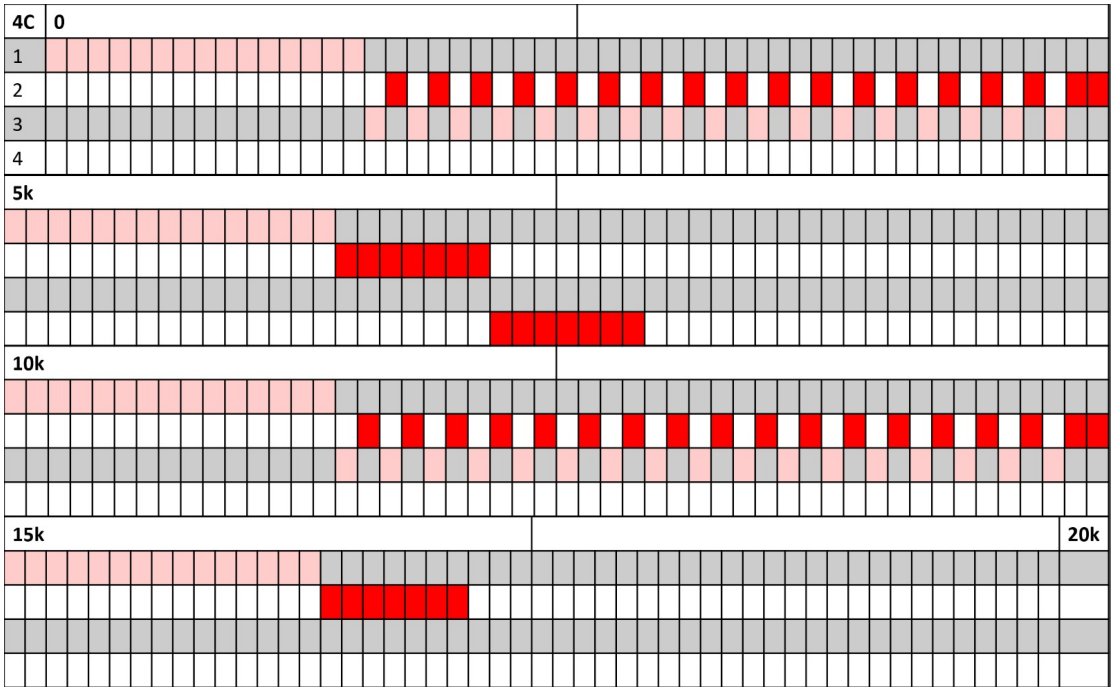
1-4 are Tasks 1 ,2, 3, and 4. Each small rectangle signifies 100ms units of time.
The picture is broken into 4 parts for practical and spacial purposes.

0, 5k, 10k , 15k and 20k are time markings of 0ms, 5000ms, 10000ms, 15000ms and 20000ms.

Pink/Red colours are WCETs and Grey/White is simply when the task is not executing. There is no difference between pink and red and no difference between grey and white, two colours each just used to improve visibility and readability.

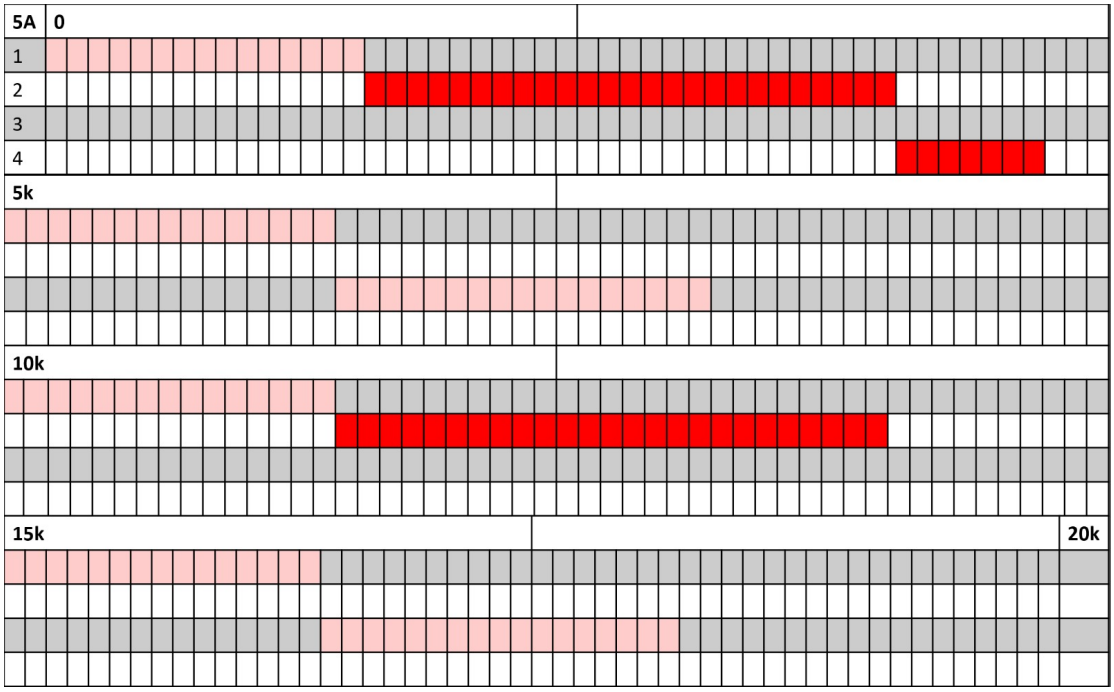
Question 4C (4 marks)

The schedule is different from 4A.



Since task 2 and task 3 are given equal priorities now, when both of them are in the ready state, freeRTOS will share the available processing time using a time sliced round robin scheduling scheme for equal priority tasks. Each 100ms (one iteration of the while loop in the code) task 2 and task 3 interrupt each other until the task execution is complete or interrupted by another higher priority task (like when task 1 interrupts task 2 at 5k). These switches happen every 100ms due to the myDelay(two times with 50ms) function being used in the while loop.

Question 5A (2 marks)



Minor 5k , Major 20k
Same format as diagram in question 4

Question 5B (6 marks)

Refer to functions 'function1' to 'function4' and task5 in the code segment below

```
////////////////////// LAB 4 G32 CODESEGMENT ////////////////////////////

#include <Arduino.h>
#include <avr/io.h>
#include <FreeRTOS.h>
#include <task.h>
#define STACK_SIZE 200
#define LED_PIN6 6
#define LED_PIN7 7
#define LED_PIN8 8
#define LED_PIN9 9
#define DUTY 20

//void task1given(void *p)
//{
//      for (;;) {
//              digitalWrite(LED_PIN6, HIGH);
//              delay(1000);
//              digitalWrite(LED_PIN6, LOW);
//              delay(1000);
//      }
//}

////////////////////// QUESTION 2E ////////////////////////////////
void task2E(void *p)
{
         TickType_t xLastWakeTime;
         const TickType_t xFrequency = 1000;
             xLastWakeTime = xTaskGetTickCount();

             for( ;; )
             {
                 vTaskDelayUntil( &xLastWakeTime, xFrequency );
                 digitalWrite(LED_PIN6, HIGH);

                 vTaskDelayUntil( &xLastWakeTime, xFrequency );
                        digitalWrite(LED_PIN6, LOW);
             }
}

////////////////////// QUESTION 2F ////////////////////////////////
void task2f1(void *p)
{
        TickType_t xLastWakeTime;
        const TickType_t xFrequency = 1500;
        xLastWakeTime = xTaskGetTickCount();

        for( ;; )
        {
                vTaskDelayUntil( &xLastWakeTime, xFrequency );
                digitalWrite(LED_PIN6, HIGH);

                vTaskDelayUntil( &xLastWakeTime, xFrequency );
                digitalWrite(LED_PIN6, LOW);
```

```
        }
}

void task2f2(void *p)
{
        TickType_t xLastWakeTime;
        const TickType_t xFrequency = 2000;
        xLastWakeTime = xTaskGetTickCount();

        for( ;; )
        {
                vTaskDelayUntil( &xLastWakeTime, xFrequency );
                digitalWrite(LED_PIN7, HIGH);

                vTaskDelayUntil( &xLastWakeTime, xFrequency );
                digitalWrite(LED_PIN7, LOW);
        }
}

void task2f3(void *p)
{
        TickType_t xLastWakeTime;
        const TickType_t xFrequency = 4000;
        xLastWakeTime = xTaskGetTickCount();

        for( ;; )
        {
                vTaskDelayUntil( &xLastWakeTime, xFrequency );
                digitalWrite(LED_PIN8, HIGH);

                vTaskDelayUntil( &xLastWakeTime, xFrequency );
                digitalWrite(LED_PIN8, LOW);
        }
}

////////////////////////// QUESTION 3 ////////////////////////////////
void task3(void *p)
{
        int pulseWidth;
        int period ;
        for ( ;; ) {
                pulseWidth = 0;
                while (DUTY >= pulseWidth) { // DUTY = 100% duty cycle, full ON-OFF
cycle (0.02s)
                        period = DUTY - pulseWidth;
                        digitalWrite(LED_PIN6, HIGH);
                        delay(pulseWidth);
                        digitalWrite(LED_PIN6, LOW);
                        delay(period);
                        pulseWidth++;  // +1ms ~ Approximately + 5% duty cycle @ 50Hz
each time (0.001s)
                }
        }
}

////////////////////////// QUESTION 4A ////////////////////////////////

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

```
void task4t1(void *p)
{
        TickType_t xLastWakeTime;
        const TickType_t xFrequency = 5000;
        xLastWakeTime = xTaskGetTickCount();
        int count = 0;
        int TASK_WCET1 = 1500;

        for( ;; )
        {
                count = 0;
                while (count < TASK_WCET1) {
                        digitalWrite(LED_PIN6, HIGH);
                        myDelay(50);
                        digitalWrite(LED_PIN6, LOW);
                        myDelay(50);
                        count +=100;
                }
                vTaskDelayUntil( &xLastWakeTime, xFrequency );
        }

}

void task4t2(void *p)
{
        TickType_t xLastWakeTime;
        const TickType_t xFrequency = 10000;
        xLastWakeTime = xTaskGetTickCount();
        int count = 0;
        int TASK_WCET2 = 2500;

        for( ;; )
        {
                count = 0;
                while (count < TASK_WCET2) {
                        digitalWrite(LED_PIN7, HIGH);
                        myDelay(50);
                        digitalWrite(LED_PIN7, LOW);
                        myDelay(50);
                        count +=100;
                }
                vTaskDelayUntil( &xLastWakeTime, xFrequency );
        }

}

void task4t3(void *p)
{
        TickType_t xLastWakeTime;
        const TickType_t xFrequency = 10000;
        xLastWakeTime = xTaskGetTickCount();
        int count;
        int TASK_WCET3 = 1700;

        for( ;; )
        {
                count = 0;
                while (count < TASK_WCET3) {
                        digitalWrite(LED_PIN8, HIGH);
                        myDelay(50);
                        digitalWrite(LED_PIN8, LOW);
                        myDelay(50);
                        count +=100;
```

```
                }
                vTaskDelayUntil( &xLastWakeTime, xFrequency );

        }
}

void task4t4(void *p)
{
        TickType_t xLastWakeTime;
        const TickType_t xFrequency = 20000;
        xLastWakeTime = xTaskGetTickCount();
        int count;
        int TASK_WCET4 = 700;

        for( ;; )
        {
         count = 0;
                while (count < TASK_WCET4) {
                        digitalWrite(LED_PIN9, HIGH);
                        myDelay(50);
                        digitalWrite(LED_PIN9, LOW);
                        myDelay(50);
                        count +=100;
                }
                vTaskDelayUntil( &xLastWakeTime, xFrequency);
        }
}

////////////////////////// QUESTION 5B ////////////////////////////////

void function1()
{
        int count = 0;
        while (count < 1500) {
                digitalWrite(LED_PIN6, HIGH);
                myDelay(50);
                digitalWrite(LED_PIN6, LOW);
                myDelay(50);
                count +=100;
        }
}

void function2()
{
        int count = 0;
        while (count < 2500) {
                digitalWrite(LED_PIN7, HIGH);
                myDelay(50);
                digitalWrite(LED_PIN7, LOW);
                myDelay(50);
                count +=100;
        }
}

void function3()
{
        int count = 0;
        while (count < 1700) {
                digitalWrite(LED_PIN8, HIGH);
                myDelay(50);
                digitalWrite(LED_PIN8, LOW);
                myDelay(50);
                count +=100;
```

```
        }
}

void function4()
{
        int count = 0;
        while (count < 700) {
                digitalWrite(LED_PIN9, HIGH);
                myDelay(50);
                digitalWrite(LED_PIN9, LOW);
                myDelay(50);
                count +=100;
        }
}
void task5 (void *p) {

        TickType_t xLastWakeTime;
        while(1) {
                xLastWakeTime = xTaskGetTickCount();
                function1();
                function2();
                function4();
                vTaskDelayUntil( &xLastWakeTime, 5000);

                xLastWakeTime = xTaskGetTickCount();
                function1();
                function3();
                vTaskDelayUntil( &xLastWakeTime, 5000);

                xLastWakeTime = xTaskGetTickCount();
                function1();
                function2();
                vTaskDelayUntil( &xLastWakeTime, 5000);

                xLastWakeTime = xTaskGetTickCount();
                function1();
                function3();
                vTaskDelayUntil( &xLastWakeTime, 5000);
        }


}


void setup()
{
        pinMode(LED_PIN6, OUTPUT);
        pinMode(LED_PIN7, OUTPUT);
        pinMode(LED_PIN8, OUTPUT);
       pinMode(LED_PIN9, OUTPUT);

}
void loop()
{
        //      xTaskCreate(taskE, // Pointer to the task entry function
                        //              "TaskE", // 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
                        //      xTaskCreate(task2f1, "Task2F1", STACK_SIZE, NULL, 3,
NULL);
```

```
//      xTaskCreate(task2f2, "Task2F2", STACK_SIZE, NULL, 2, NULL);
//      xTaskCreate(task2f3, "Task2F3", STACK_SIZE, NULL, 1, NULL);

//      xTaskCreate(task3, "Task3", STACK_SIZE, NULL, 1, NULL);

//      xTaskCreate(task4t1, "Task4T1", STACK_SIZE, NULL, 4, NULL); // part B : 3
//      xTaskCreate(task4t2, "Task4T2", STACK_SIZE, NULL, 2, NULL); // part B : 2
//      xTaskCreate(task4t3, "Task4T3", STACK_SIZE, NULL, 3, NULL); // part B : 2
//      xTaskCreate(task4t4, "Task4T4", STACK_SIZE, NULL, 1, NULL); // part B : 1
        xTaskCreate(task5, "Task5", STACK_SIZE, NULL, 1, NULL); //
        vTaskStartScheduler();
}
```