AIN SHAMS UNIVERSITY
I-Credit Hours Engineering Programs
(i.CHEP)

University of East London

MCTA – Senior 1

Course Code:	CSE411	Course Name:	Real-Time and Embedded Systems Design	Assignment No.	1	Date:	**28/03/2023**
Student Name:	**Ahmed Hisham Fathy Hassabou**			Student ID:	19P4007		

	A (89-100)				B (76-88)				C (67-75)				D (60-66)				F (0-59)			
	100	96	92	89	88	84	80	76	75	72	69	67	66	64	62	60	59	40	20	0
Answering Questions &Organization (70%)	• Answers are correct and complete				• Most answers are correct and complete				• Some answers are correct and complete				• Few answers are correct and complete.				• Most answers are incorrect and complete			
Presentation and Language (30%)	• Excellent language and technical vocabulary.				• Good language and technical vocabulary.				• Normal language and technical vocabulary.				• Low language and technical vocabulary.				• Difficult language and technical vocabulary.			

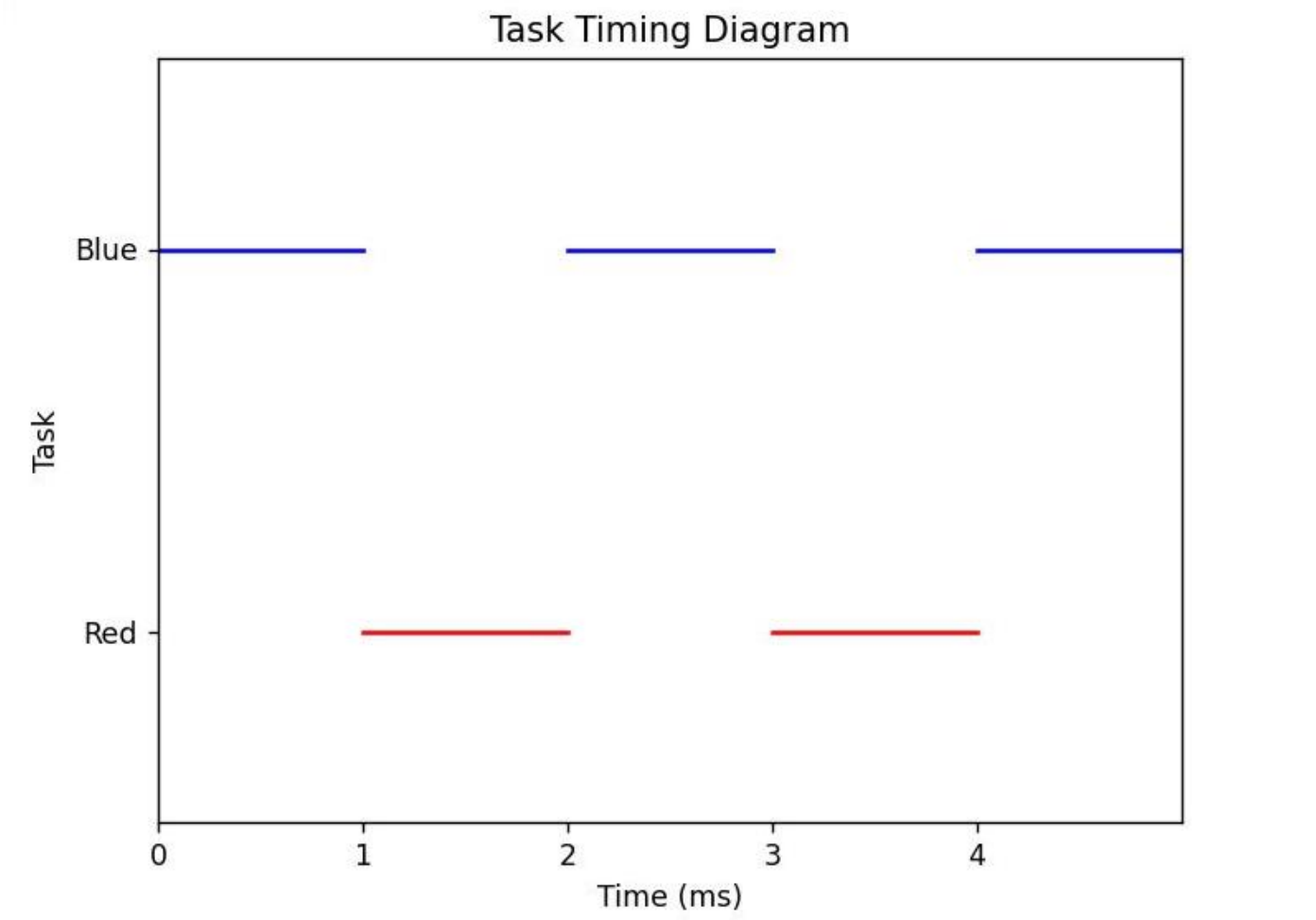
1st marker Total		1st marker Signature		Agreed Mark	
2nd Marker Total		2nd marker Signature			

General Comments:

UEL Grading System	Agreed Mark	ASU Grading Scale	
% equivalent at UEL	**Range**	**% at ASU**	**Grade**
95% and higher		97% and higher	A+
82% to less than 95%		93% to less than 97%	A
70% to less than 82%		89% to less than 93%	A-
66% to less than 70%		84% to less than 89%	B+
63% to less than 66%		80% to less than 84%	B
60% to less than 63%		76% to less than 80%	B-
56% to less than 60%		73% to less than 76%	C+
53% to less than 56%		70% to less than 73%	C
50% to less than 53%		67% to less than 70%	C-
45% to less than 50%		64% to less than 67%	D+
40% to less than 45%		60% to less than 64%	D
Less than 40%		Less than 60%	F

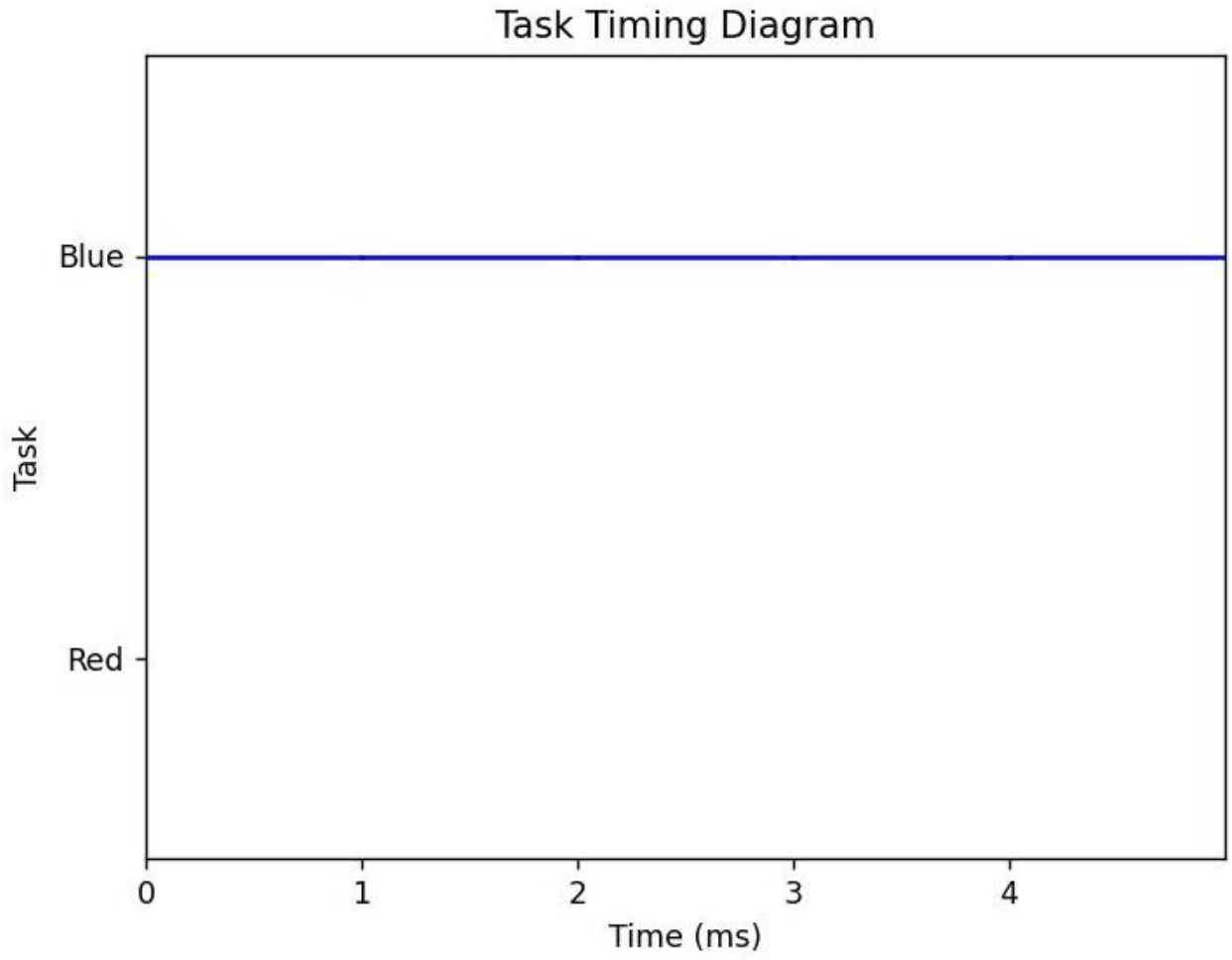
1. Question 1 (2 Tasks from 1 Task Function)

1-a. Non-Blocking Tasks with same priority



For timestamp of 1ms the tasks shifting will happen only when the SYSTICK fires and OS switch between tasks of same priority. The task that was working last timestamp will leave the CPU and the other will acquire the CPU.

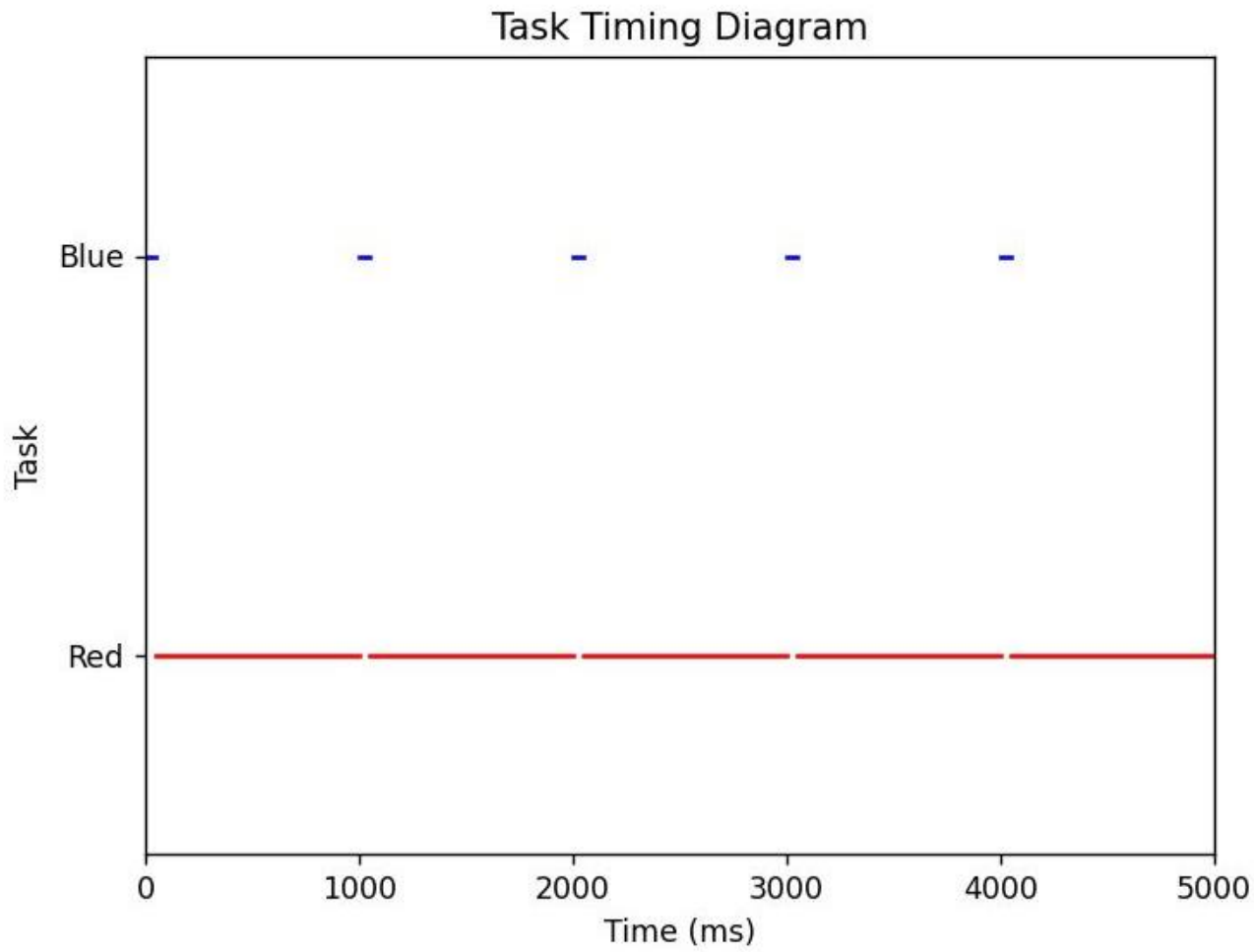
1-b. Non-Blocking Tasks, task 2 higher priority



Task 1 that blinks the Red Led will starve the CPU, each time the SYSTICK fires and OS have the option to choose which task will acquire the CPU it will choose Task 2 that blinks the Blue Led due to its higher priority.

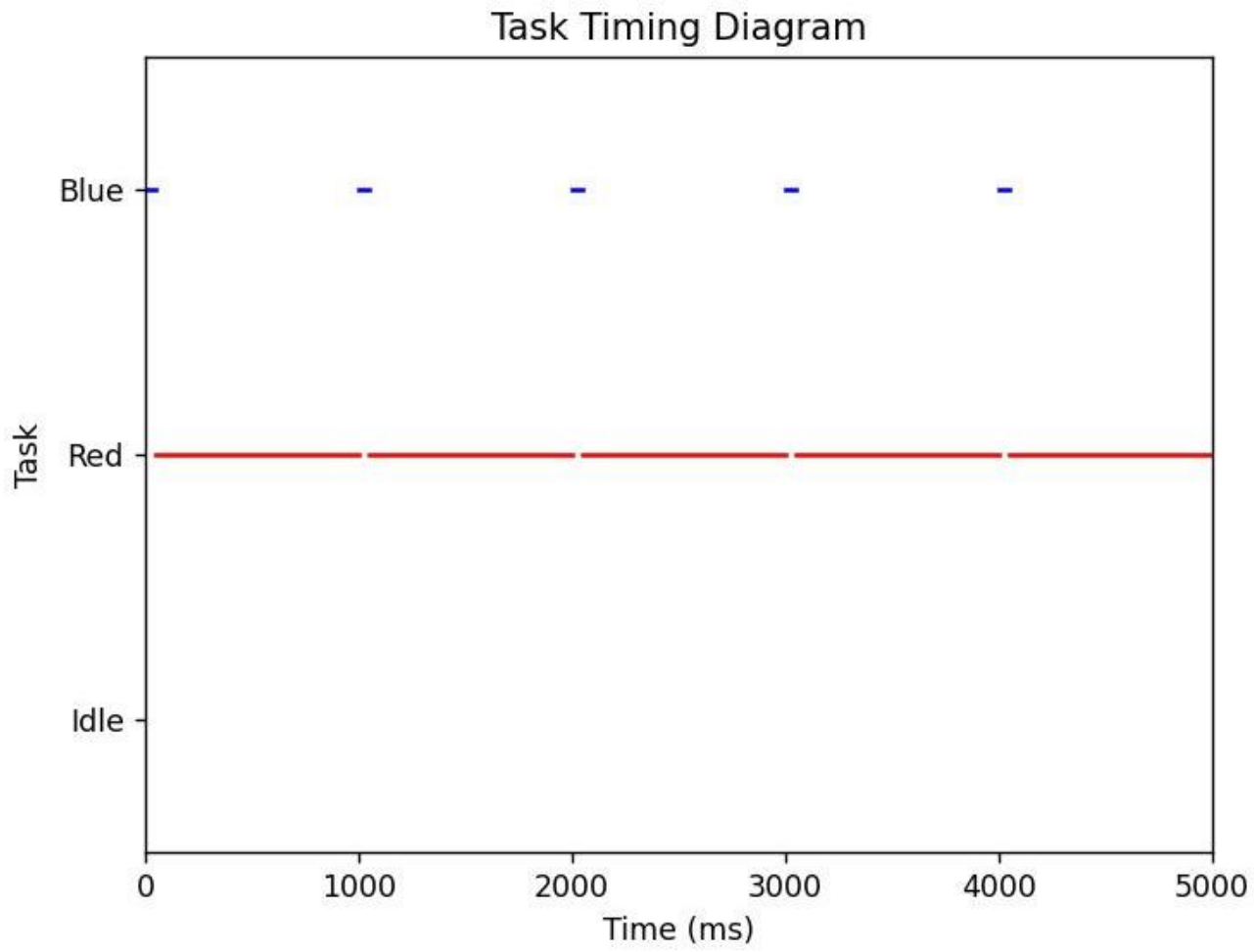
2. Question 2 (2 Separate Tasks)

2-a. Blue Task Have Higher Priority and blocking while Red Task is continuous.



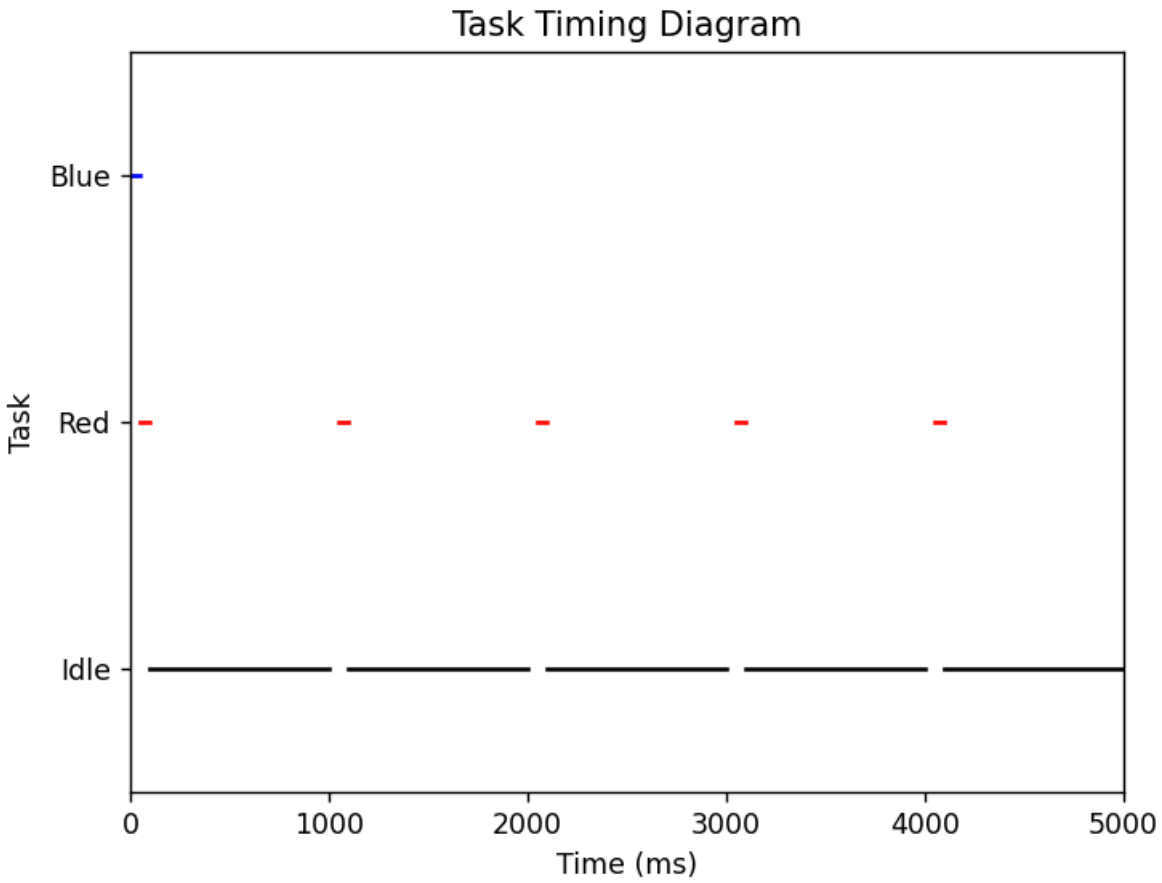
At the beginning the higher priority task acquire the CPU but then get blocked for 1 second, so the lower priority task which is continuous takes the chance of acquiring the CPU, during the period of blockage of the higher priority task, each time the SYSTICK fires the OS find nothing ready but the lower priority task so it takes the CPU again. Until the delay period finishes, the higher priority task will preempt the lower one and this cycle keeps going on.

2-b. Blue Task Have Higher Priority and blocking while Red Task is continuous, and the idle task is the lowest priority.



In this case the idle task will never take the CPU. When the higher priority task is blocked there will be 2 tasks available 1 higher priority continuous task (red) and 1 lower priority task (idle) so the idle task will be starving, and the previous example algorithm will occur again.

2-c. Blue Task Have Higher Priority and blocking, Red Task is lower priority and also blocking, and the idle task is the lowest priority.



Only now the idle task will have a chance to acquire the CPU. When the blue task get to the blockage state, the red task will acquire the CPU but then it also get to the blockage state, so the idle task that have the lowest priority will acquire the CPU and deletes the blue task so when the delay period ends the red task will be the only one available to run so it will preempt the idle and the loop will continue between the red and idle task.

3. Question 3

Both vTaskDelay() and vTaskDelayUntil() are delay functions provided by the FreeRTOS real-time operating system, but they differ in how they are used to delay the execution of a task.

vTaskDelay() is a blocking delay function that suspends the calling task for a specified amount of time. The delay time is specified in ticks of the FreeRTOS tick count. The tick count represents the number of elapsed ticks since the FreeRTOS scheduler was started. For example, if the tick rate is set to 1 kHz and vTaskDelay(100) is called, the calling task will be suspended for 100 ticks.

vTaskDelayUntil() is also a blocking delay function that suspends the calling task, but it delays the task until a specified absolute time has been reached. The delay time is also specified in ticks, but instead of specifying a duration, the function takes an absolute time value as its second argument. The absolute time is calculated by adding the delay time to the current tick count. For example, if the current tick count is 1000 and vTaskDelayUntil(&xLastWakeTime, 200) is called, the calling task will be suspended until the tick count reaches 1200 (1000 + 200).

In summary, vTaskDelay() delays the task for a specified duration, while vTaskDelayUntil() delays the task until a specified absolute time has been reached.

4. Question 4

4-a. uxTaskPriorityGet API:

The uxTaskPriorityGet API is used to get the priority of a task. It takes one argument:

- A handle to the task to get the priority of.

The function returns an UBaseType_t value, which is the current priority of the specified task.

4-b. vTaskPrioritySet API:

The vTaskPrioritySet API is used to set the priority of a task. It takes two arguments:

- A handle to the task to set the priority of.
- The new priority to set for the task. The priority is specified as a UBaseType_t value, where 0 is the lowest priority and the maximum priority is dependent on the specific configuration of the FreeRTOS kernel.

The function does not return a value.