Practica 2 RTOS

Jose Luis Rocabado Rocha

1. Ejercicio 1

*El stack size es de nº de palabras, para obtener el tamaño en bits se necesita multiplicar por 4. El reloj es de 1000 ticks/s. Tener en cuenta que la tarea 2 tienen más peso en el bucle pero que el hecho de utilizar el JTAG genera que se compense dándonos un uso de la CPU de ~50%.

Se generan las 4 tareas propuestas en el ejercicio además de la tarea que nos permite visualizar las estadísticas y la tarea inicializadora con las siguientes características.

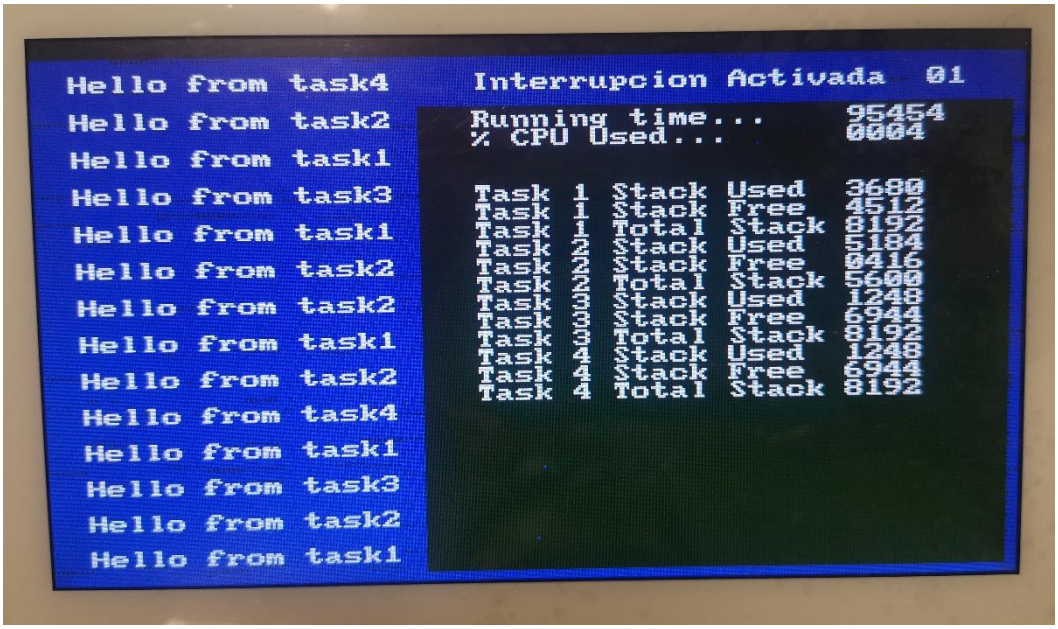
```
/* Definition of Task Stacks */
#define   TASK_STACKSIZE  2048
#define   TASK2_STACKSIZE 1400
OS_STK    task1_stk[TASK_STACKSIZE];
OS_STK    task2_stk[TASK2_STACKSIZE];
OS_STK    task3_stk[TASK_STACKSIZE];
OS_STK    task4_stk[TASK_STACKSIZE];
OS_STK    taskstart_stk[TASK_STACKSIZE];
OS_STK    showstat_stk[TASK_STACKSIZE];


/* Definition of Struct for Stats*/
OS_STK_DATA data;
TASK_USER_DATA  TaskUserData[10];

/* Definition of Task Priorities */
#define TASKSTART_PRIORITY  0
#define TASK1_PRIORITY      1
#define TASK2_PRIORITY      2
#define TASK3_PRIORITY      3
#define TASK4_PRIORITY      4
#define TASK_SHOWSTATS_PRIO 5

/* Definition of Task ID, realmente no valen para nada, esperemos que tengan sentido en uCOSII
#define TASKSTART_ID  2
#define TASK1_ID      3
#define TASK2_ID      4
#define TASK3_ID      5
#define TASK4_ID      6
#define TASK_SHOWSTATS_ID 10
```

Sin embargo, las tareas 3 y 4 tienen una periodicidad de 0.5 y 2 segundos respectivamente.

Tras construir el proyecto, se obtiene el siguiente resultado, donde se puede apreciar claramente los recursos de cada tarea, el espacio ocupado y el tamaño de *stack* libre.

Nótese que el tamaño del *stack* especificado en la creación de la tarea consiste en un tamaño de palabras de 32 bits, pero el que se muestra en la MTL consiste en el tamaño en bytes.

Por ejemplo, el *stack* de la tarea 2 tiene un tamaño de 1400 palabras que corresponden con 5600 bytes.



Se observa también el uso del servicio ***OSTaskDel(OS_PRIO_SELF)*** para terminar la ejecución de la tarea creadora puesto que una vez ha cumplido su propósito (inicializar el resto de tareas y habilitar las estadísticas) no es necesario que siga ejecutándose.

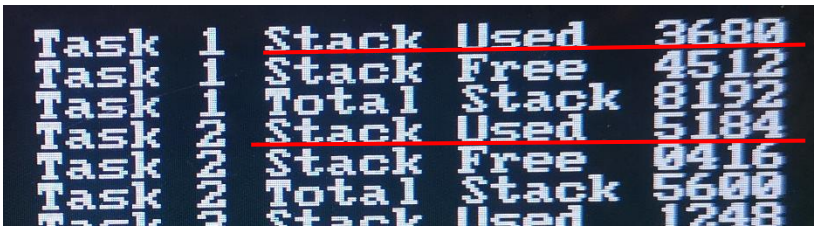
2. Ejercicio 2

Se configuran las tareas 1 y 2 para obtener el tiempo en *ticks* que requiere cada tarea para su ejecución. Si observamos los resultados:

```
El mensaje desde task2 es: Hola Don José
El mensaje desde task2 es: Hola Don José
El mensaje desde task2 es: Hola Don José
Hello from task2, I need 2682 - 256 = 2426 ticks from RTOS

El mensaje es: Hola Don Pepito
El mensaje es: Hola Don Pepito
El mensaje es: Hola Don PepitoHello from task1

Hello from task1, I need 3355 - 3221 = 134 ticks from RTOS
```

Se observa que la tarea 2 requiere de mucho más tiempo para ejecutarse. Esto es así dado que la tarea 2 tiene una variable *dummy* de mayor tamaño que la tarea 1 i por lo tanto el bucle *for* no requiere tanto coste computacional. Para hacer una comparación en segundos es necesario saber que el reloj del sistema está configurado con una frecuencia de 1000 *ticks*/s por lo que se puede estimar que la tarea 2 tarda aproximadamente 2s en ejecutarse mientras que la tarea 1 solo 100ms.

Esto también podemos visualizarlo en el consumo de la pila de cada tarea (siguiente figura) donde la tarea 2 consume alrededor de 5184 bytes, prácticamente toda su pila, la tarea 1 consume 3680 bytes.



Aunque la tarea 2 consume prácticamente todo su recurso, se comprueba que uCOSII no se queda colgado y sigue funcionando correctamente dado que no se llega a consumir la pila.

3. Ejercicio 3

a)

Nos permite saber el tiempo de ejecución en el que se encuentra todo el sistema operativo

b)

Tal y como se explica en el ejercicio anterior, el tamaño de pila y su uso son coherentes con lo especificado en el código y también los tiempos obtenidos.

4. Ejercicio 4

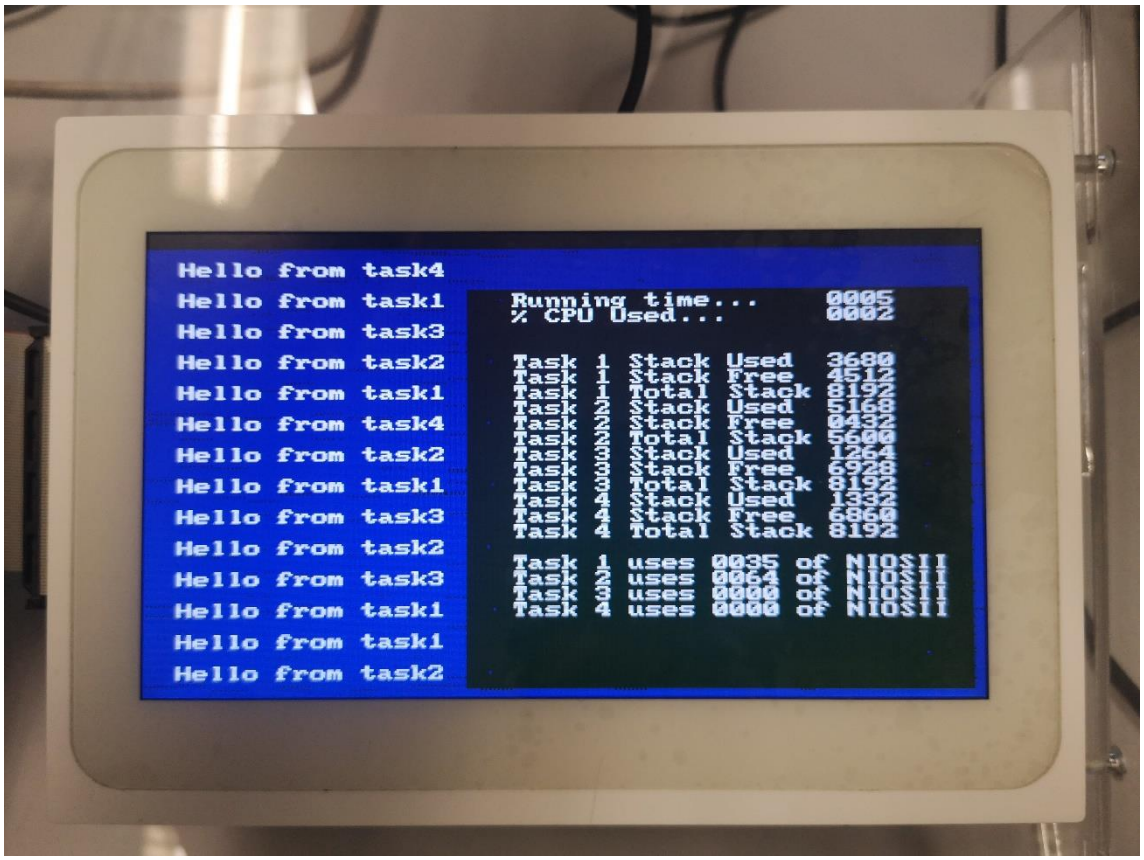
a)

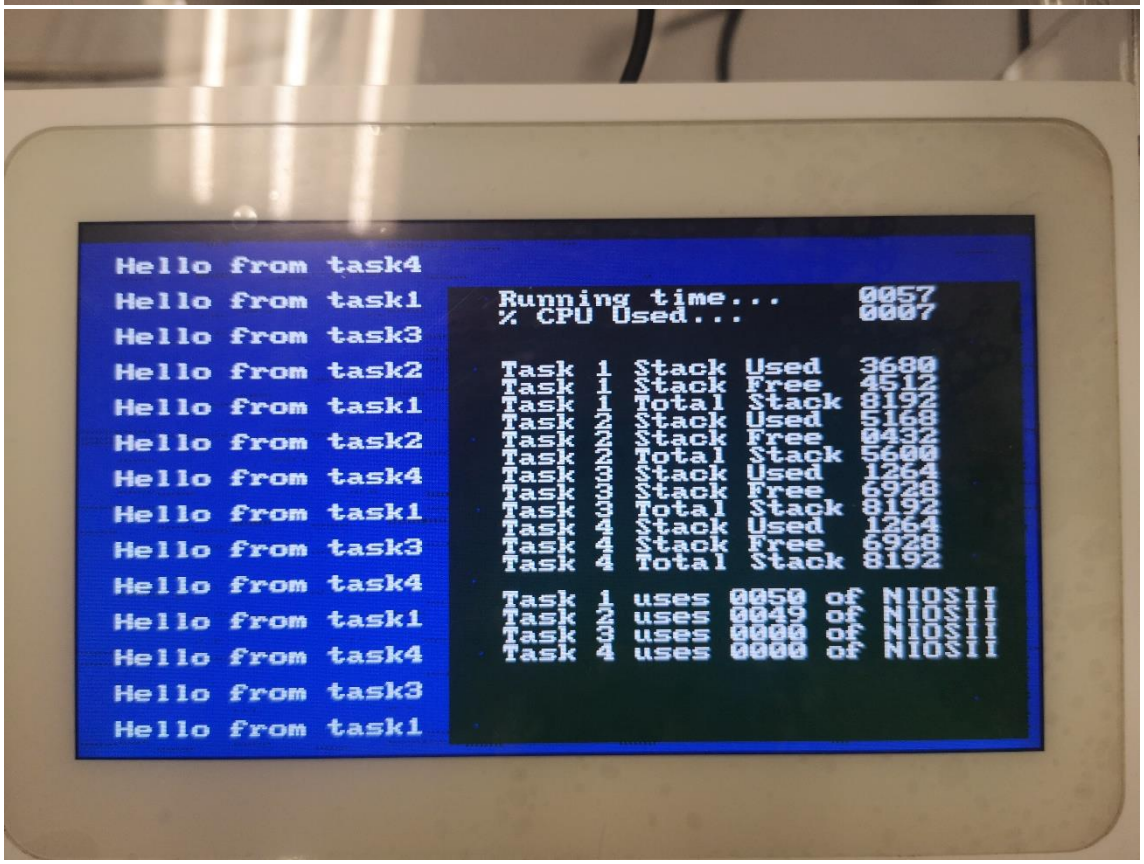
```
Hello from task1, I need 1 - 0 = 1 ticks from RTOS

TaskExecTime      Task 1   is   0
TaskTotalExecTime Task 1   is 342
TaskCtr   of      Task 1   is 600

TaskExecTime      Task 2   is   1
TaskTotalExecTime Task 2   is 338
TaskCtr   of      Task 2   is 1243

TaskExecTime      Task 3   is   0
TaskTotalExecTime Task 3   is   0
TaskCtr   of      Task 3   is  12

TaskExecTime      Task 4   is   0
TaskTotalExecTime Task 4   is   1
TaskCtr   of      Task 4   is  15
```

Sin embargo, utilizando las funciones *hook* para completar las estadísticas temporales de cada tarea. Se observa que los tiempos no corresponden con lo obtenido dentro de cada tarea y esto puede deberse a que la tarea que muestra las estadísticas se ejecuta cada segundo dándonos un *frame* del estado en ese momento. Por otro lado, la cuenta de llamada de cada tarea parece funcionar correctamente.

c)

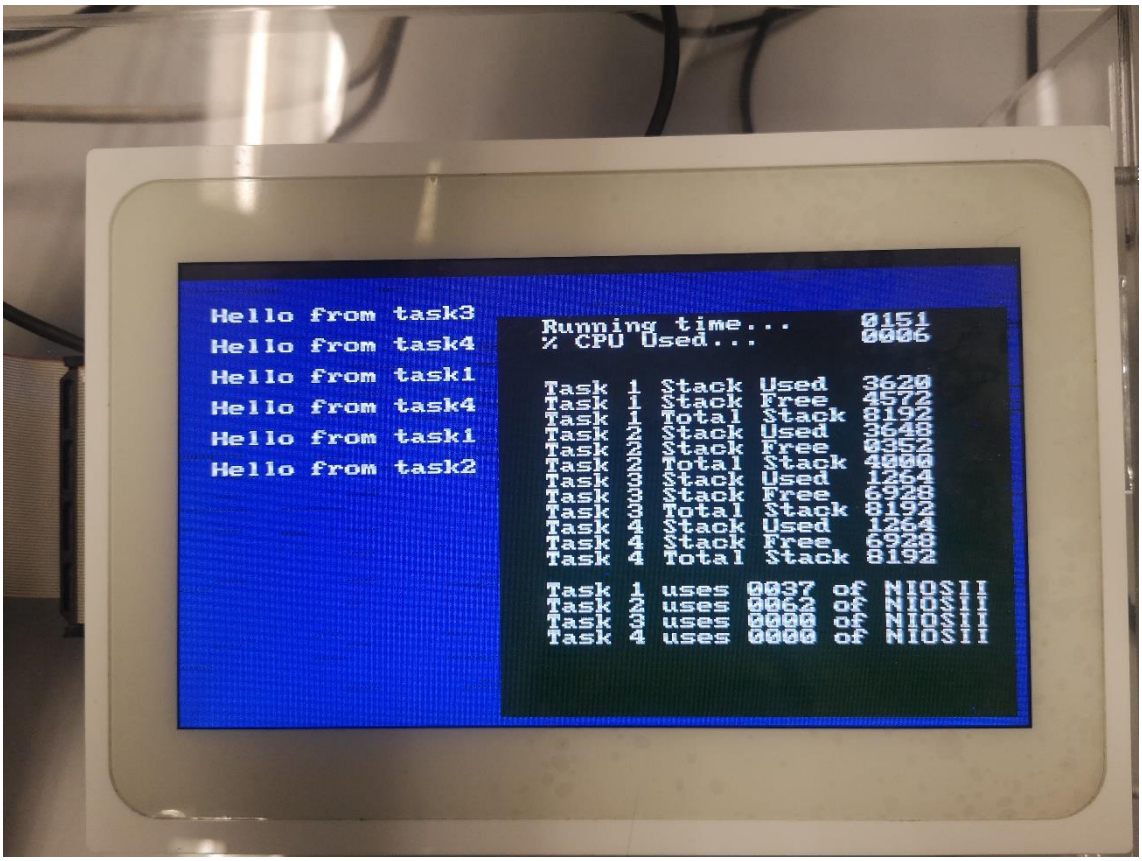
Se comprueba que no es necesario llamar a la función puesto que el scheduler lo hace periódicamente cada segundo.





Se observa por lo tanto como que a medida que pasa el tiempo, los porcentajes de uso de la CPU convergen a un valor de aproximadamente un 50% entre las dos tareas ya que la tarea 1 tiene una periodicidad de 1s y la tarea 2, de 1’3 segundos.

5. Desbordamientos



En cuanto a los desbordamientos, se comenta como durante la práctica se observa como el sistema se ralentiza, y empieza a funcionar incorrectamente, puesto que se intenta escribir en direcciones de memoria fuera del stack instanciado.