Laboratorio 1
Threads en FreeRTOS

Fecha: 24/02/2023 —— Deadline: 27/02/2023

Leonardo Achá Boiano
Departamento de Ingeniería Mecatrónica
Universidad Católica Boliviana Santa Cruz de la Sierra, Bolivia
leonardo.acha@ucb.edu.bo

Andrés Ayala
Departamento de Ingeniería Mecatrónica
Universidad Católica Boliviana Santa Cruz de la Sierra, Bolivia
andres.ayala@edu.ucb.bo

Bruno Ramiro Rejas Montero
Departamento de Ingeniería Mecatrónica
Universidad Católica Boliviana Santa Cruz de la Sierra, Bolivia
bruno.rejas@ucb.edu.bo

Ing. Cabrera José Jesús
Sistemas Embebidos II IMT-322
Departamento de Ingeniería Mecatrónica
Universidad Católica Boliviana Santa Cruz de la Sierra, Bolivia
jcabrera@edu.ucb.bo

***Abstract*—En el presente informe se detallan las bases de la programación de microcontroladores ARM Cortex M4 mediante el lenguaje ensamblador en el entorno de desarrolló Keil uVision5.**

Palabras Clave—Programación, microcontroladores, ARM, Keil uVision

I. INTRODUCCIÓN

FreeRTOS es un sistema operativo de tiempo real de código abierto que se utiliza en sistemas embebidos de bajo consumo energético. Bare metal programming implica programar directamente sobre el hardware sin utilizar un sistema operativo. Al combinar ambos, se puede crear un sistema eficiente y personalizado para controlar los LED en un microcontrolador de nucleo cortex M3.

II. OBJETIVOS

- Introducir a RTOS
- Aprender sobre tasks, threads
- Entender el proceso de creación de tasks
- Entender el rol de kernel 'FreeRTOS'
- Crear su primer programa en FreeRTOS

III. PROCEDIMIENTO

A. Ejercicio 1: Creación de proyecto

1) Crear un nuevo proyecto Keil.

2) Seleccione su plataforma y todas las dependencias adecuadas. En el apartado RTOS asegurese de seleccionar FreeRTOS y no RTOS-API.

3) Cree un prototipo de una funcion para la creacion de un Task.

```
// Task functions prototype
void vTask1( void *pvParameters );
void vTask2( void *pvParameters );
```

Figura 1. Prototipo de funciones para las tasks

4) Cree la implementacion de una funcion como se indica en el Codigo 2.

5) Dentro de la funcion main configure su primer Task utilizando su funcion creada previamente.

6) Una vez configurado el Task inicie el cronograma con la funcion vTaskStartScheduler()

7) Compile el codigo y asegurese de que no tenga ninguna error (Puede ignorar las Warnings)

8) Cargue su codigo a la plataforma. Por ahora, este codigo no tiene ninguna funcionalidad.

B. EJERCICIO 2:AGREGANDO FUNCIONALIDADES

1) Cree un nuevo prototipo de funcion

2) Implemente la funcion tal como se indica en el Codigo 2

3) Utilizando esta nueva funcion configure un nuevo Task. Por ahora, para ambos Tasks coloque la prioridad de 1. usStackDepth de 1000. pvParameters - NULL. No se olvide de crear un Handle para cada Task.

```
TaskHandle_t task2_Handle = NULL;
xTaskCreate(vTask2,          // Pointer to the function
"Task 2",                    // Text name for the task
1000,                        // Stack depth in words configMINIMAL_STACK_SIZE
(void*)pvTask2,              // Parameter passed into the task
1,       // Priority of the task
&task2_Handle );             // Handle to the task


TaskHandle_t task1_Handle = NULL;
xTaskCreate(vTask1,          // Pointer to the function
"Task 1",                    // Text name for the task
1000,                        // Stack depth in words
(void*)pvTask1,              // Parameter passed into the task
1,       // Priority of the task
&task1_Handle );             // Handle to the task
```

Figura 2. Inicialización de funciones para las tasks

4) Antes de la creacion del cronograma configure tres pins (a su eleccion) para prender y apagar tres LEDs.

```
RCC->APB2ENR |= RCC_APB2ENR_IOPAEN|RCC_APB2ENR_IOPBEN; //enable GPIO clock
// PA12 PB1 PB5 as outputs
GPIOA->CRH |= 0x00030000;
GPIOB->CRL |= 0x00300030;
// make PA12 high, PB1 high, low high
GPIOA->BSRR|= (1 << 12);
GPIOB->BSRR|= (1 << 1)|(1 << 5);
```

Figura 3.Configuración de GPIOs para la plataforma seleccionada

5) Una vez estos esten configurados como salida a un LED mande un voltaje alto y a los otros dos un voltaje bajo. Asegurese que esto se realice antes de la llamada a vTaskStartScheduler().

```
// Start the scheduler so our tasks start executing
vTaskStartScheduler();
/* If all is well we will never reach here as the scheduler will now be
running.  If we do reach here then it is likely that there was insufficient
heap available for the idle task to be created. */
while(1){
}
```

Figura 4. Llamada al cronograma de freertos.

6) Compile y cargue su codigo a la plataforma. Los Tasks no hacen nada aun. Asegurese de que su codigo se comporte como espera. Es decir, dos LEDs se encuentra apagado y uno prendido. Asegurese de configurar sus pines adecuadamente para tener este comportamiento.

7) Un Task se encargara de intercalar (Toggle) los valores el LED1, otro Task se encargara de intercalar los valores del LED2. Y el LED3 debera mantenerse prendido y ningun Task puede modificarlo.

```
void vTask1( void *pvParameters )
{
TickType_t xLastWakeTimeTask1;
    // Task is implemented in an infinite loop.
    while(1)
    {
        xLastWakeTimeTask1 = xTaskGetTickCount();
        GPIOB->ODR ^= (GPIO_ODR_ODR1);
        // GPIOB->ODR ^= (GPIO_ODR_ODR5);
        // GPIOA->ODR ^= (GPIO_ODR_ODR12);
        // Delay for a period.
        vTaskDelayUntil( &xLastWakeTimeTask1, pdMS_TO_TICKS( 1000 ) );
    }
}
/*-----------------------------------------------------------*/
void vTask2( void *pvParameters )
{
TickType_t xLastWakeTimeTask2;
    // Task is implemented in an infinite loop.
    while(1)
    {
        xLastWakeTimeTask2 = xTaskGetTickCount();
        // GPIOB->ODR ^= (GPIO_ODR_ODR1);
        // GPIOB->ODR ^= (GPIO_ODR_ODR5);
        GPIOA->ODR ^= (GPIO_ODR_ODR12);
        // Delay for a period.
        vTaskDelayUntil( &xLastWakeTimeTask2, pdMS_TO_TICKS( 1000 ) );
    }
}
```

Figura 5. Implementación de los task para analisis del comportamiento de freertos.

8) Compile y cargue su codigo. Que puede observar? El LED1 y el LED2 intercalan valores? Se puede apreciar el blinking de los LEDs?

Tras reiteradas ejecuciones y cambios se puede observar que cuando el comportamiento se programa en una sola task, el codigo se ejecuta como es esperado; Se evidenció que la unica task que se ejecuta es la que se crea primero. Nunca se llega a ejecutar la segunda.

9) Agregue un delay en cada Task de al menos 250 ms. Compile y cargue su codigo. Que puede observar?

Los delays se ejecutan como es esperado. Más aun se utilizaron las dos funciones disponibles para delay: `vTaskDelayUntil` y `vTaskDelay`. Ambas demostraron el mismo comportamiento.

C. Ejercicio 3: Prioridades

1) Busque en el documento FreeRTOSConfig.h cual es la prioridad maxima.

Se puede evidenciar en la documentación que el macro configMAX_PRIORITIES. En el caso de la STM32F103cx se trata de 56.

2) Cambie la prioridad del Task1 a la prioridad maxima.

3) Compile y cargue su codigo.

4) Que ocurre con el Task1? Que ocurre con el Task2?

La task que se crea en primer lugar es la que se ejecuta, mientras que a otra permanece bloqueada. Se comprobó prendiendo y apagando los diferentes LEDs en diferente orden, frecuencia y tiempo de delay. Devolviendo en repetidas ocasiones el mismo resultado, solo una de las task se ejecuta.

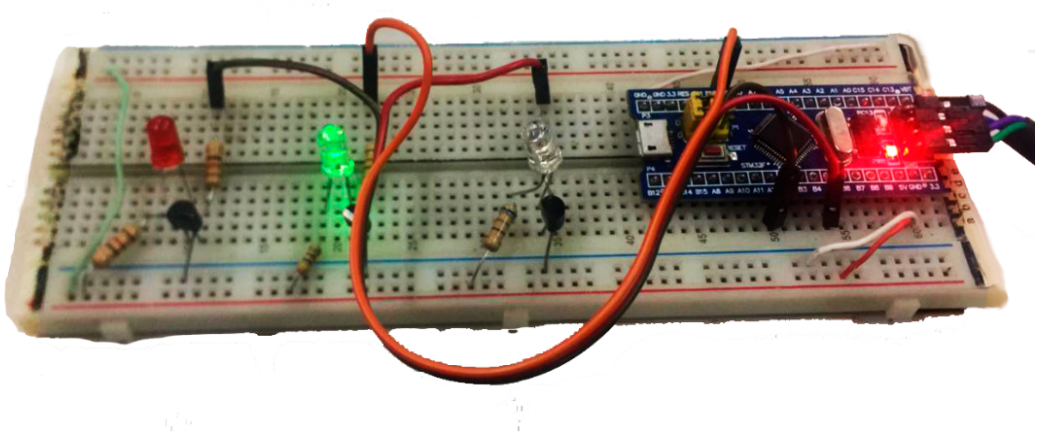

Figura 6.Hardware usado en el laboratorio, placa de desarrollo STM32F103C8: Blue Pill.

IV. CONCLUSIONES

Tras haber realizado el trabajo planteado, se logró apreciar cómo la ausencia de conocimientos sobre FreeRTOS puede causar problemas al momento de trabajar con sistemas embebidos multitarea y de tiempo real. Sin embargo, ejemplos diversos disponibles en la red llevan a la concluir que con la investigación y la práctica adecuadas, es posible utilizar FreeRTOS de manera efectiva para aprovechar al máximo las capacidades de los microcontroladores modernos. El uso de task en FreeRTOS permite una mayor flexibilidad y capacidad de respuesta para aplicaciones en tiempo real, al tiempo que hace un uso más eficiente de los recursos del microcontrolador. Esto es especialmente importante para aplicaciones con múltiples tareas y dependencias complejas. Con una formación adecuada, se pueden superar los obstáculos y aprovechar al máximo las capacidades de los microcontroladores modernos para crear sistemas embebidos complejos y altamente funcionales.