CÁTEDRA DE SISTEMAS OPERATIVOS II

Departamento de Computación

FCEFyN - UNC

FreeRTOS

Esteban Tissot
<egtisso@gmailcom>
35276396

Córdoba, 2017

INDICE

1. Introducción

1.1. Propósito

Implementar, analizar y comprender el funcionamiento de un sistema operativo de tiempo real.

1.2. Ámbito del Sistema

Se utilizará una placa de desarrollo Freecale (FRDM-k64F), donde se implementará el sistema operativo de tiempo real FreeRTOS. El análisis se basará en capturas realizadas por el software TraceAlyzer.
El IDE que se utilizara se denomina Kinetis® Design Studio. KDS es un ambiente de desarrollo para microcontroladores Kinetis que permite diseñar, compilar y debugear proyectos.

1.3. Referencias

Tracealyzer

Información: https://percepio.com/docs/FreeRTOS/manual/
Descarga: https://percepio.com/download/ y http://www.mono-project.com/download/

FreeRTOS

Información: http://www.freertos.org/

KSDK 2.0

Descarga KSDK 2.0:
https://www.element14.com/community/community/designcenter/kinetis_kl2_freedom_board/blog/2016/03/17/getting-started-with-ksdk-20-how-to-use-the-sdk-with-kinetis-design-studio-using-frdm-k64f-freedom-board
Tutorial:
https://www.element14.com/community/community/designcenter/kinetis_kl2_freedom_board/blog/2016/04/15/getting-started-with-freertos-and-ksdk-20-creating-a-new-freertos-with-ksdk-v20-application-using-frdm-k64f-freedom-board

FRDM-K64F

Informacion:
http://www.nxp.com/products/developer-resources/hardware-development-tools/freedom-development-boards/freedom-development-platform-for-kinetis-k64-k63-and-k24-mcus:FRDM-K64F
User Guide: http://www.farnell.com/datasheets/2295012.pdf

2. Descripción General

Instalar el sistema operativo FreeRTOS en un la placa de desarrollo que posea un microcontrolador y ejecutar cuatro tareas simples que permitan realizar un análisis completo del comportamiento del Sistema operativo, identificando las tareas de mayor prioridad y como afectan sobre las de menor prioridad. También se realizará un informe sobre los recursos utilizados en tiempo de ejecución.

3. Requisitos Específicos

Se deberá implentar el sistema operativo FreeRTOS en un la placa FRDM-k64F y ejecutar cuatro tareas simples con distintas prioridades, para ello, se utilizará el IDE KDS para instalar, compilar y debuguear el proyecto.
Se tomará como base uno de los ejemplos que vienen con el KDS para mayor facilidad y por último se utilizará Tracealycer para analizar la implementación.
Se utilizará el led rojo (RGB) integrado en la placa FRDM-k64F y un pulsador además de una cola FIFO para almacenar los datos que luego van a ser transmitidos mediante UART, a un baudrate de 115200, e impresos por consola, se necesitará una estructura para diferenciar el tipo de dato que cada tarea pone en la cola. Del lado del usuario se utilizará el software “minicom” para poder recibir los datos provenientes de la placa.

4. Diseño de solución

Se implementarán cuatro tareas en total que comparten una cola FIFO donde se guardarán datos de tipo “interger” o “char” dependiendo de la tarea que se ejecute, y, para diferenciar estos datos se utilizará una estructura. Estos datos serán enviados mediante UART junto con el número correspondiente al log y el nombre de la tarea que escribió ese dato. La tarea de mayor prioridad verificará cada 10 milisegundos si se ha producido una interrupción por pulsador, en caso de que haya pasado debe sumar un contador y poner el valor del contador en la cola compartida por todas las tareas. La subrutina de interrupción solo seteará una bandera que será leída y reseteada por la tarea. Las dos tareas de siguiente prioridad se deben ejecutar cada 1 segundo. Estas tareas utilizan un bucle for para escribir distinta secuencia de números en la cola. Al tener la misma prioridad se debe esperar que las tareas se interrumpan entre ellas.Por último, la tarea de menor prioridad es la que tomará los datos de la cola, los discriminará y los enviará por UART junto con el número de log y el nombre de la tarea que lo escribió.
Por último se realizará un análisis completo del Sistema con Tracealycer donde se tendrá en cuenta los tiempos de ejecución y memoria.

5. Implementación

Instalación y configuración

- Instalar Kinetis Design Studio
- Instalar Percepio Trace Exporter desde KDS. Help -> Install New Software.
    - Agregar el repositorio: http://dl.percepio.com/exporter/
- Configurar Percepio
- Instalar Firmware JLink_OpenSDA_V2.bin

Prototipos, Estructuras y Prioridades

A continuación se muestra como se nombraron las funciones, configuraron las prioridades y se armó la estructura de datos, para mayor información dirigirse al título “Código” de la sección Apéndice.

Prototipo de Funciones

```
static void first_task(void *pvParameters);
static void write1(void *pvParameters);
static void write2(void *pvParameters);
static void print_console(void *pvParameters);
static void interrupt_task(void *pvParameters);
/* Cola FIFO */
static QueueHandle_t log_queue = NULL;
```

Estructura de datos

```
typedef enum INFO_TYPE{
	TipoValor,
	TipoString
} TipoDato;

typedef struct TASK_INFO{
	TipoDato eTipoDato;
	void* pDato;
} QueueEstruct;
```

Task priorities

```
#define first_task_PRIORITY	 (configMAX_PRIORITIES)
#define led_task_PRIORITY	(configMAX_PRIORITIES - 1)
#define write1_PRIORITY		(configMAX_PRIORITIES - 2)
#define write2_PRIORITY		(configMAX_PRIORITIES - 2)
#define print_console_PRIORITY	(configMAX_PRIORITIES - 3)
```

Funciones utilizadas

En este punto se está en condiciones de inicializar las correspondientes funciones en el main y de crear las tareas necesarias mediante las funciones **xTaskCreate()** donde se pasa como parámetro el nombre de la tarea, un nombre descriptivo, el tamaño del stack (en cantidad de palabras), un valor que va a ser pasado como parámetro, la prioridad y un puntero que también pasa como parámetro. Al finalizar la creación de las tareas se inicia el scheduler mediante la función **vTaskStartScheduler()**.

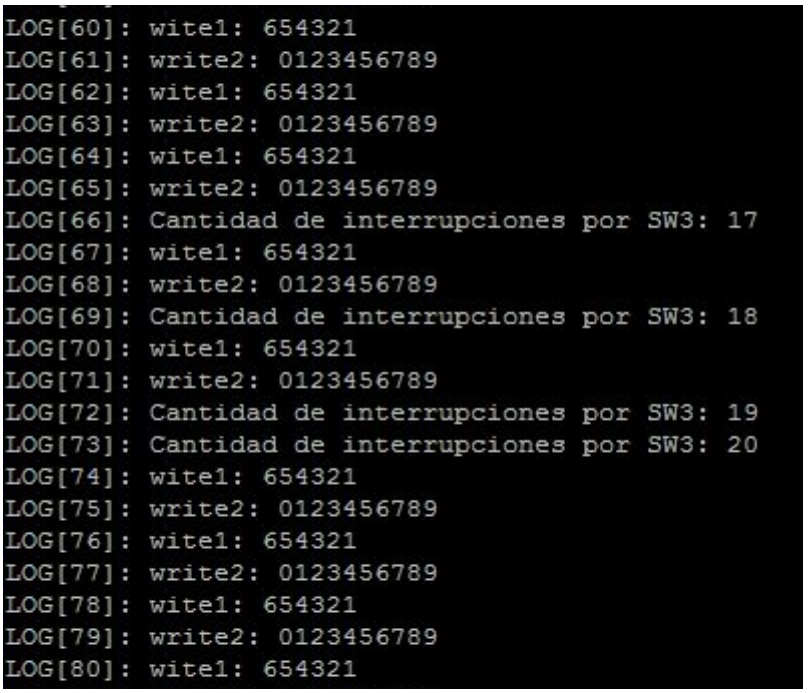
Para colocar datos en la cola se utiliza la funcion **xQueueSend()** y **xQueueReceive()** para sacar los datos de la misma, los parámetros de estas funciones son: el identificador de la cola, el puntero a los datos y el tiempo máximo que debe bloquearse la tarea en caso de estar llena la cola.

Dentro de las dos funciones que escriben en la cola se utilizó la función **taskYIELD()**, esta función se utiliza para solicitar un cambio de contexto a otra tarea. Sin embargo, si no hay otras tareas con mayor o igual prioridad, el scheduler simplemente ejecutará nuevamente la tarea. Debido a esta función es que en la sección “Resultados” se observará un entrelazado de las dos tareas que tienen igual prioridad.

La función **vTaskDelay()** se utiliza para bloquear la tarea por la cantidad de ticks que son enviados como parámetro. En este caso cada tick dura un milisegundo, por lo que la tarea *“interrupt_task”* se ejecuta cada 10ms y las tareas *“write1”* y *“write2”* cada 500ms. La tarea *“first_task”* implementa la función **vTaskSuspend()**, esta función hace que la tarea se suspenda hasta que se ejecute la función **vTaskResume()** en otra tarea y la despierte.

6. Resultados

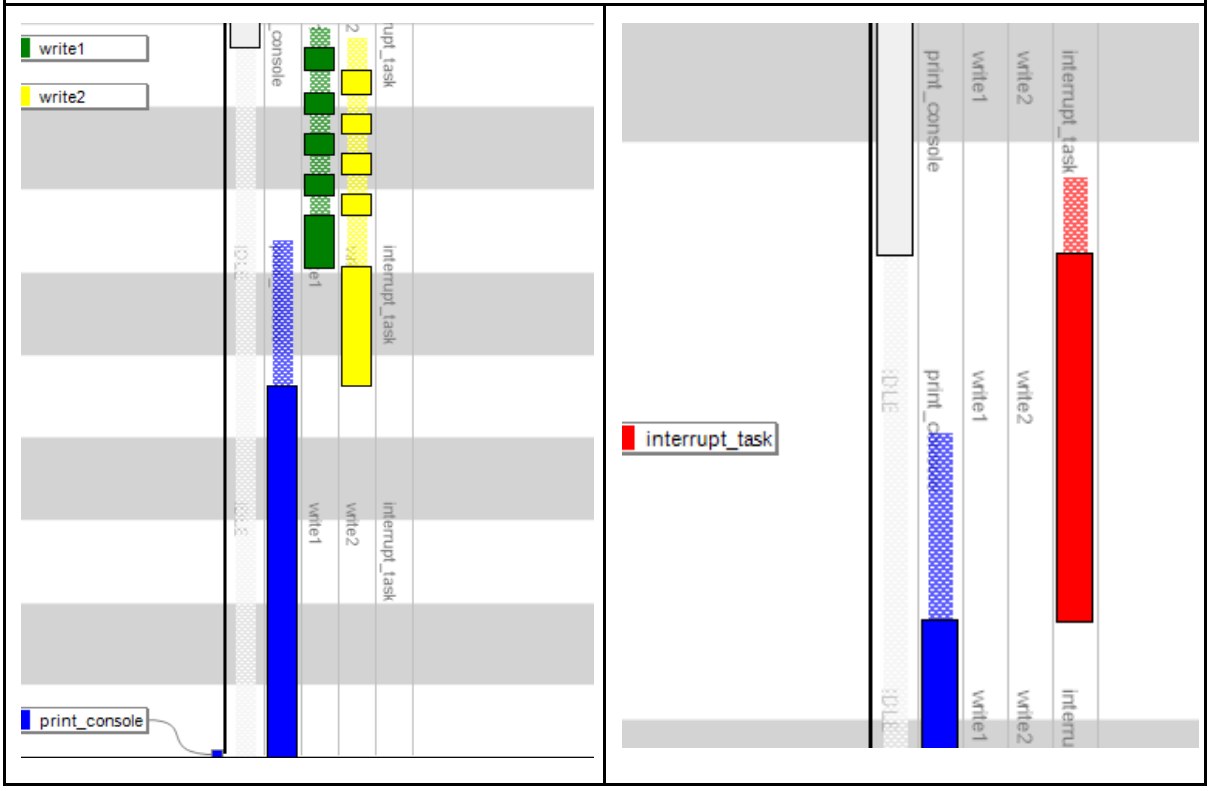
Utilizando el software PuTTY y configurandolo para que el baudrate sea de 115200 obtenemos la siguiente salida, la cual la provee la tarea “print_console” meidante UART. Se observa que las tareas write1 y write2 se intercalan para imprimir en pantalla, y que la impresión por parte de la interrupción no sigue un patrón determinado.



Impresión en Consola

En las imágenes provista por el software TraceAlycer se puede analizar con mayor precisión lo que está haciendo el sistema operativo.
En la primer imagen se ve que las tareas write1(verde) y write2(amarillo) se intercalan, pero esto no es para imprimir ya que no es su funcionalidad, lo que está pasando es que una escribe en la cola, es interrumpida por la otra que también escribe en la misma cola y así sucesivamente. Esto es así porque ambas tareas tienen la misma prioridad y tienen que compartir el mismo recurso, en cambio, la tarea azul que es la encargada de enviar los datos por UART tiene menor prioridad y este es el motivo por el cual tiene que esperar hasta que liberen la cola las otras dos tareas.
En la segunda imagen de TraceAlycer muestra la tarea “interrupt_task” (rojo), esta tarea se ejecuta periódicamente cada 10 milisegundos pero solo escribe en la cola cuando se presiona el pulsador, que también enciende y apaga el led rojo de la placa, esta es la razón por la cual en la consola no vemos un comportamiento patrón de esta tarea. También se observa que que tiene mayor prioridad que la tarea “print_console”, ya que esta última no la interrumpe.

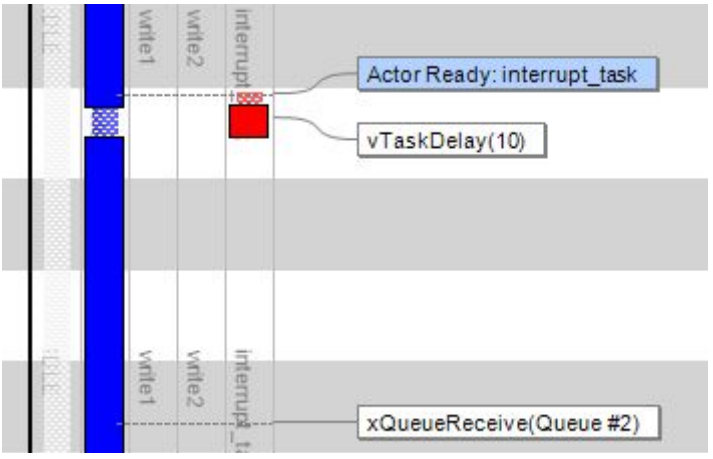
Ejecución de tareas

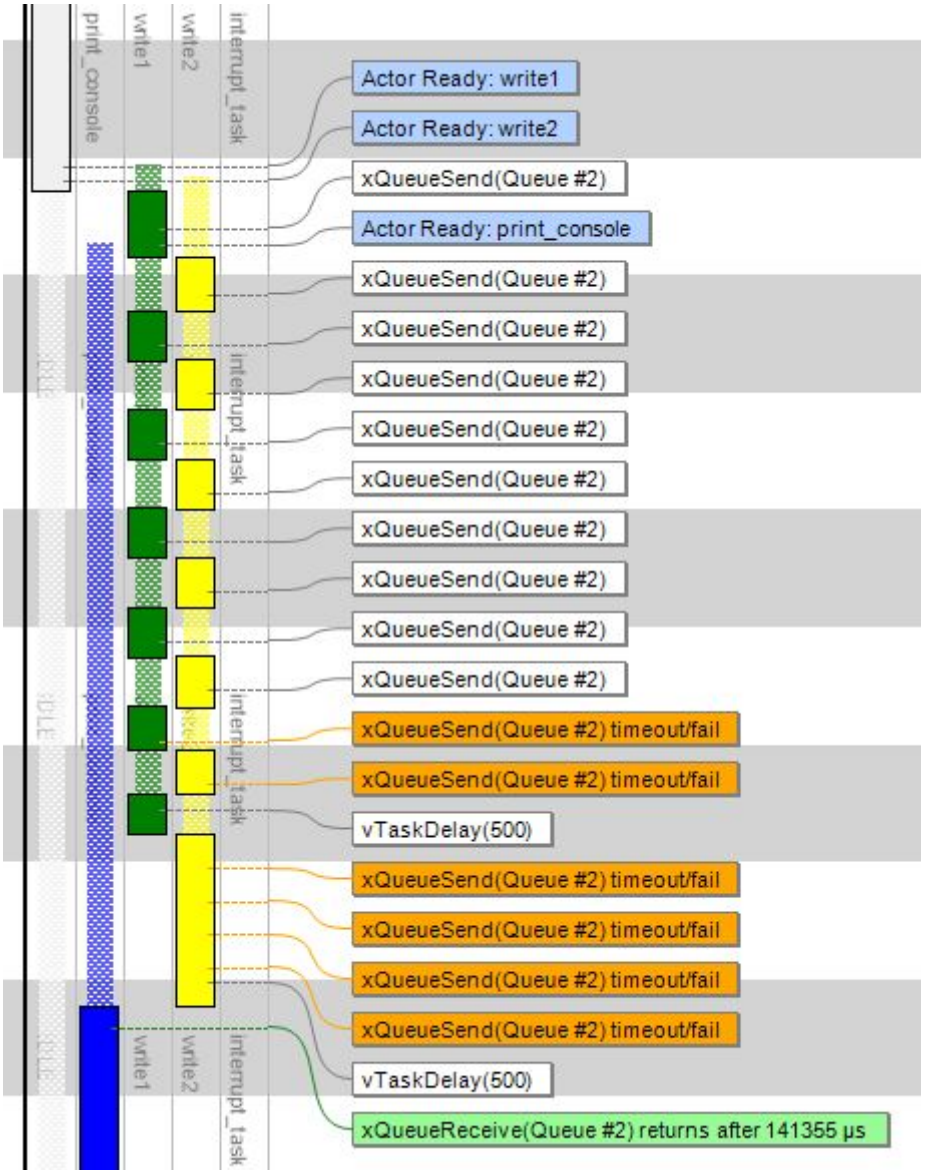


No se pudo capturar el momento en que se encuentren las cuatro tareas en “ready”. En este caso se debería observar que se ejecute primero la tarea “interrupt_task”, cuando finalice se deberían intercalar las tareas “write1” y “write2” hasta que finalicen y por último “print_console”. Este comportamiento está dado por las prioridades de las tareas.

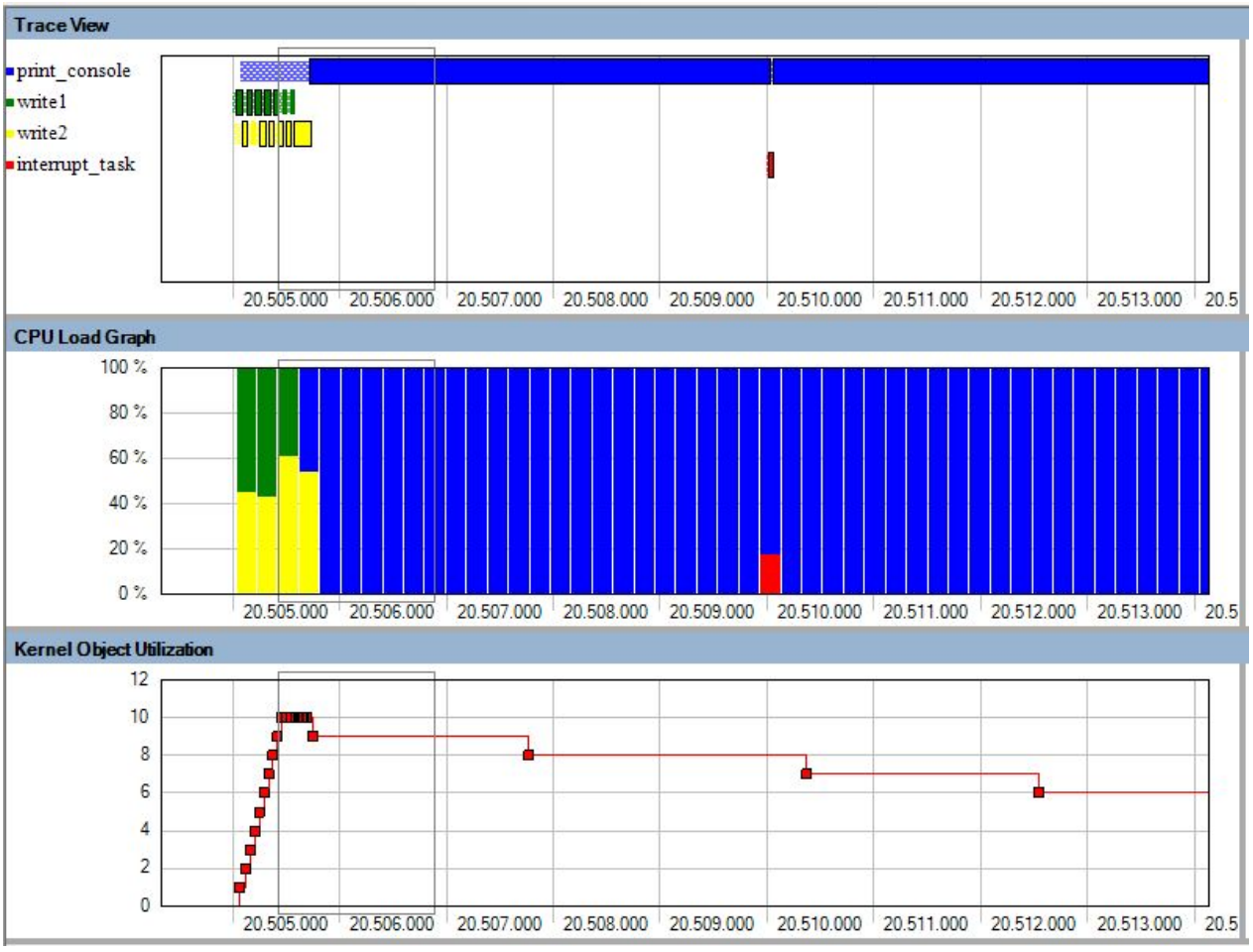
En ambas imágenes se aprecia una quinta tarea, “IDLE”. Esta tarea es creada por el sistema operativo con la mínima prioridad, con el sentido de que se ejecute solamente cuando el procesador esté en ciclos ociosos, y es responsable de liberar la memoria asignada por el RTOS a tareas que se han eliminado desde entonces.

A continuación se muestran imágenes (en tiempo de ejecución) que muestran el momento en que se realizan las llamas del sistema y el estado de ready de las tareas.





La imagen que se encuentra abajo está dividida por recuadros, en el primero se muestra la traza de ejecución, mientras que en el segundo observamos el porcentaje de uso del CPU y en el último vemos como se va llenando el recurso “cola” mientras se ejecutan las tareas “write1” y “write2” y luego se va vaciando cuando se ejecuta la tarea “print_console”.



Estadísticas

Función: *interrupt_task*

interrupt_task inst 51/141
- Start Time: 20.510.006 (s.ms.µs)
- End Time: 20.510.054 (s.ms.µs)
- Execution Time: 34 (µs)
- Response Time: 48 (µs)
- Wait Time: 13 (µs)
- Startup Time: 13 (µs)
- Response Interference: 38,92%
- Fragmentation: 1
- Priority: 4
- Previous Instance (50 / 141)
  - Start Time: 20.500.006 (s.ms.µs)
  - End Time: 20.500.054 (s.ms.µs)
  - Separation: 9.952 (ms.µs)
  - Periodicity: 10.000 (ms.µs)
- Next Instance (52 / 141)
  - Start Time: 20.520.006 (s.ms.µs)
  - End Time: 20.520.052 (s.ms.µs)
  - Separation: 9.952 (ms.µs)
  - Periodicity: 10.000 (ms.µs)

Task interrupt_task
- Instances: 141
- Total CPU time: 4.853 (0,35 %)
- Total Wait time: 0 (µs)

Funcion: *write1*

write1 inst 2/3
- Start Time: 20.505.006 (s.ms.µs)
- End Time: 20.505.577 (s.ms.µs)
- Execution Time: 296 (µs)
- Response Time: 571 (µs)
- Wait Time: 275 (µs)
- Startup Time: 23 (µs)
- Response Interference: 92,87%
- Fragmentation: 7
- Priority: 3
- Previous Instance (1 / 3)
  - Start Time: 20.005.131 (s.ms.µs)
  - End Time: 20.005.577 (s.ms.µs)
  - Separation: 499.429 (ms.µs)
  - Periodicity: 499.875 (ms.µs)
- Next Instance (3 / 3)
  - Start Time: 21.005.006 (s.ms.µs)
  - End Time: 21.005.577 (s.ms.µs)
  - Separation: 499.429 (ms.µs)
  - Periodicity: 500.000 (ms.µs)

Task write1
- Instances: 3
- Total CPU time: 832 (0,06 %)
- Total Wait time: 711 (µs)

Funcion: *write2*

write2 inst 2/3
- Start Time: 20.505.018 (s.ms.µs)
- End Time: 20.505.724 (s.ms.µs)
- Execution Time: 399 (µs)
- Response Time: 706 (µs)
- Wait Time: 307 (µs)
- Startup Time: 67 (µs)
- Response Interference: 77,03%
- Fragmentation: 7
- Priority: 3
- Previous Instance (1 / 3)
  - Start Time: 20.005.085 (s.ms.µs)
  - End Time: 20.005.724 (s.ms.µs)
  - Separation: 499.294 (ms.µs)
  - Periodicity: 499.933 (ms.µs)
- Next Instance (3 / 3)
  - Start Time: 21.005.018 (s.ms.µs)
  - End Time: 21.005.724 (s.ms.µs)
  - Separation: 499.294 (ms.µs)
  - Periodicity: 500.000 (ms.µs)

Task write2
- Instances: 3
- Total CPU time: 1.197 (0,09 %)
- Total Wait time: 720 (µs)

Función: *print_console*

print_console inst 2/3
- Start Time: 20.505.073 (s.ms.µs)
- End Time: 20.529.488 (s.ms.µs)
- Execution Time: 23.696 (ms.µs)
- Response Time: 24.415 (ms.µs)
- Wait Time: 719 (µs)
- Startup Time: 651 (µs)
- Response Interference: 3,04%
- Fragmentation: 3
- Priority: 2
- Previous Instance (1 / 3)
  - Start Time: 20.005.724 (s.ms.µs)
  - End Time: 20.029.490 (s.ms.µs)
  - Separation: 475.583 (ms.µs)
  - Periodicity: 499.350 (ms.µs)
- Next Instance (3 / 3)
  - Start Time: 21.005.074 (s.ms.µs)
  - End Time: 21.029.486 (s.ms.µs)
  - Separation: 475.585 (ms.µs)
  - Periodicity: 500.000 (ms.µs)

Task print_console
- Instances: 3
- Total CPU time: 71.087 (5,06 %)
- Total Wait time: 207 (µs)

7. Conclusiones

Este trabajo permite analizar como es un sistema de tiempo real y comprobar los beneficios que brinda. Se pudo comprobar como se interrumpen las tareas con distintas prioridades y que asegura la periodicidad de ejecución. Para todo este análisis se aprendió a utilizar la herramienta TraceAlycer la cual resultó de gran importancia para saber lo que realmente estaba pasando en tiempo de ejecución, medir tiempos y uso de recursos.

8. Apéndices

Tracealyzer

The trace recorder library is the target system component of Tracealyzer for FreeRTOS. The recorder integrates with FreeRTOS and records a trace of RTOS events and your own User Events. The recorder has two main operating modes, described below.

In snapshot mode, the trace data is kept in target RAM, allowing you to take a "snapshot" at any point by saving the contents of the RAM buffer. The snapshot mode is highly optimized for memory efficiency and the resulting data rate is often just 10-20 KB/s, depending on the system. So a trace buffer of a few KB is often sufficient to get a decent trace of the most recent events. This can be used in essentially any system, and even used as "black-box" during field testing or in deployed systems.

In streaming mode, the trace data is transfered continuously to the host PC and thereby allows for tracing over long durations. This supports streaming via SEGGER J-Link debug probes, but may also utilize other interfaces in your system, such as USB, TCP/IP, UARTs, or device file systems.

The recorder library is provided in C source code, and is found in the Tracealyzer installation directory (see "Trace Recorder Library" directory) as a zip file. Since version 3.1, the snapshot and streaming modes are integrated in the same recorder library with a common API.

FRDM-K64F board

The Flagship FRDM-K64F has been designed by NXP in collaboration with mbed for prototyping all sorts of devices, especially those requiring optimized size and price points. The board is well sized for connected applications, thanks to its power efficient Kinetis K64F MCU featuring an ARM® Cortex®-M4 core running up to 120MHz and embedding 1024KB Flash, 256KB RAM and lots of peripherals (16-bit ADCs, DAC, Timers) and interfaces (Ethernet, USB Device Crystal-less and Serial). The Kinetis K64 MCU family remains fully software, hardware and development tool compatibility with Kinetis MCU and Freedom board families. It is packaged as a development board including extension headers

compatible with Arduino R3 shields and includes a built-in USB Debug and Flash Programmer.

Features

- MK64FN1M0VLL12 MCU (120 MHz, 1 MB flash memory, 256 KB RAM, low-power, crystal-less USB, and 100 Low profile Quad Flat Package (LQFP))
- Dual role USB interface with micro-B USB connector
- RGB LED
- FXOS8700CQ accelerometer and magnetometer
- Two user push buttons
- Flexible power supply option – OpenSDAv2 USB, Kinetis K64 USB, and external source
- Easy access to MCU input/output through Arduino™ R3 compatible I/O connectors
- Programmable OpenSDAv2 debug circuit supporting the CMSIS-DAP Interface software that provides:
    - Mass storage device (MSD) flash programming interface
    - CMSIS-DAP debug interface over a driver-less USB HID connection providing run-control debugging and compatibility with IDE tools.
    - Virtual serial port interface
    - Open source CMSIS-DAP software project
- Ethernet
- SDHC
- Add-on Bluetooth module: JY-MCU BT board V1.05 B

Codigo

main.c

```
/*System includes.*/
#include <stdio.h>

/* FreeRTOS kernel includes. */
#include "FreeRTOS.h"
#include "task.h"
#include "queue.h"
#include "timers.h"
#include "semphr.h"

/* Freescale includes. */
#include "fsl_device_registers.h"
#include "fsl_debug_console.h"
#include "board.h"
#include "trcBase.h"
#include "pin_mux.h"
#include "clock_config.h"
#include "fsl_port.h"
#include "fsl_common.h"
#include "fsl_gpio.h"

/*******************************************************************************
 * Definitions
 ******************************************************************************/
#define APP_SYSVIEW_APP_NAME "FRDMK64F System Viewer Demo"
```

```
#define APP_SYSVIEW_DEVICE_NAME "Cortex-M4"
#define APP_SYSVIEW_RAM_BASE (0x1FFF0000)

#define BOARD_LED_GPIO BOARD_LED_RED_GPIO
#define BOARD_LED_GPIO_PIN BOARD_LED_RED_GPIO_PIN

#define BOARD_SW_GPIO BOARD_SW3_GPIO
#define BOARD_SW_PORT BOARD_SW3_PORT
#define BOARD_SW_GPIO_PIN BOARD_SW3_GPIO_PIN
#define BOARD_SW_IRQ BOARD_SW3_IRQ
#define BOARD_SW_IRQ_HANDLER BOARD_SW3_IRQ_HANDLER
#define BOARD_SW_NAME BOARD_SW3_NAME

/* Task priorities. */
#define first_task_PRIORITY          (configMAX_PRIORITIES)
#define led_task_PRIORITY     (configMAX_PRIORITIES - 1)
#define write1_PRIORITY              (configMAX_PRIORITIES - 2)
#define write2_PRIORITY           (configMAX_PRIORITIES - 2)
#define print_console_PRIORITY   (configMAX_PRIORITIES - 3)

/*******************************************************************************
* Globals
******************************************************************************/
/* Logger queue handle */
static QueueHandle_t log_queue = NULL;

/*******************************************************************************
 * Variables
 ******************************************************************************/
/* Whether the SW button is pressed */
volatile bool g_ButtonPress = false;

/*******************************************************************************
 * Prototypes
 ******************************************************************************/
static void first_task(void *pvParameters);
static void write1(void *pvParameters);
static void write2(void *pvParameters);
static void print_console(void *pvParameters);
static void interrupt_task(void *pvParameters);


/* Define the structure type that will be passed on the queue. */
typedef enum INFO_TYPE{
       TipoValor,
       TipoString
} TipoDato;

typedef struct TASK_INFO{
       TipoDato eTipoDato;
       void* pDato;
} QueueEstruct;


/* Define the init structure for the output LED pin*/
gpio_pin_config_t led_config = {
    kGPIO_DigitalOutput, 0,
};

/* Define the init structure for the input switch pin */
gpio_pin_config_t sw_config = {
    kGPIO_DigitalInput, 0,
};

void BOARD_SW_IRQ_HANDLER(void){
```

```
    /* Clear external interrupt flag. */
    GPIO_ClearPinsInterruptFlags(BOARD_SW_GPIO, 1U << BOARD_SW_GPIO_PIN);
    /* Change state of button. */
    g_ButtonPress = true;
}

/*******************************************************************************
 * Code
 ******************************************************************************/
/*!
 * @brief Main function
 */
int main(void)
{
    /* Init board hardware. */
    BOARD_InitPins();
    BOARD_BootClockRUN();
    BOARD_InitDebugConsole();
    vTraceInitTraceData();

    /* Init output LED GPIO. */
    GPIO_PinInit(BOARD_LED_GPIO, BOARD_LED_GPIO_PIN, &led_config);

    /* Init input switch GPIO. */
    PORT_SetPinInterruptConfig(BOARD_SW_PORT, BOARD_SW_GPIO_PIN,
kPORT_InterruptFallingEdge);
    EnableIRQ(BOARD_SW_IRQ);
    GPIO_PinInit(BOARD_SW_GPIO, BOARD_SW_GPIO_PIN, &sw_config);

    if (!uiTraceStart()){
        vTraceConsoleMessage("Could not start recorder!");
    }

    xTaskCreate(first_task, "first_task", configMINIMAL_STACK_SIZE, NULL,
first_task_PRIORITY, NULL);
    xTaskCreate(write1, "write1", configMINIMAL_STACK_SIZE + 166, NULL,
write1_PRIORITY, NULL);
    xTaskCreate(write2, "write2", configMINIMAL_STACK_SIZE + 166, NULL,
write2_PRIORITY, NULL);
    xTaskCreate(interrupt_task, "interrupt_task", configMINIMAL_STACK_SIZE + 166,
NULL, led_task_PRIORITY, NULL);
    vTaskStartScheduler();
    while(1);
}

/*!
 * @brief first_task function
 */
static void first_task(void *pvParameters){
    traceLabel consoleLabel;
    consoleLabel = xTraceOpenLabel("Messages");
    PRINTF("Start address: 0x%x \r\n", RecorderDataPtr);
    PRINTF("Bytes to read: 0x%x \r\n", sizeof(*RecorderDataPtr));

    /* Init Queue. */
    log_queue = xQueueCreate(10, sizeof(struct xTaskInfo *));
    xTaskCreate(print_console, "print_console", configMINIMAL_STACK_SIZE + 166,
NULL, print_console_PRIORITY, NULL);
    vTaskSuspend(NULL);
}

/*!
 * @brief interrupt_task function
 */
static void interrupt_task(void *pvParameters){
```

```
       static unsigned short i;
       static QueueEstruct Dato;
       static QueueEstruct* p = &Dato;
       traceLabel consoleLabel;
       consoleLabel = xTraceOpenLabel("Messages");
       i=0;
    for(;;){
        if (g_ButtonPress){
               i = i+1;
               p -> eTipoDato = TipoValor;
               p ->pDato = &i;
               xQueueSend(log_queue, (void *)&p, 0);
              GPIO_TogglePinsOutput(BOARD_LED_GPIO, 1U << BOARD_LED_GPIO_PIN); //
Toggle LED.
              g_ButtonPress = false; // Reset state of button.
         }
        vTaskDelay(10); //delay de 10ms
    }
}

/*!
 * @brief write1 function
 */
static void write1(void *pvParameters){
    traceLabel consoleLabel;
    char log[20];
    char aux[4];
       uint8_t i;
       static QueueEstruct Dato;
       static QueueEstruct* p = &Dato;
       consoleLabel = xTraceOpenLabel("Messages");
       for(;;){
              sprintf(log, "wite1: ");
              for (i = 6; i > 0; i--){
                     sprintf(aux, "%d",(int)i);
                     strcat(log, aux);
                     taskYIELD();
              }
              p -> eTipoDato = TipoString;
              p ->pDato = &log;
              xQueueSend(log_queue, (void *)&p, 0);
              vTaskDelay(500); //delay 500ms
       }
}

/*!
 * @brief write2 function
 */
static void write2(void *pvParameters){
    traceLabel consoleLabel;
    consoleLabel = xTraceOpenLabel("Messages");
    char log[20];
    char aux[4];
    uint8_t i;
    static unsigned short aEnviar;
    static QueueEstruct Dato;
       static QueueEstruct* p = &Dato;

       for(;;){
              sprintf(log, "write2: ");
              for (i = 0; i < 10; i++){
                     sprintf(aux, "%d",(int)i);
                     strcat(log, aux);
                     taskYIELD();
              }
```

```
            p -> eTipoDato = TipoString;
            p ->pDato = &log;
            xQueueSend(log_queue, (void *)&p, 0);
            vTaskDelay(500); //Delay de 500ms
        }
}

/*!
 * @brief print_console function
 */
static void print_console(void *pvParameters){
    traceLabel consoleLabel;
    consoleLabel = xTraceOpenLabel("Messages");
    uint32_t counter = 0;
    static unsigned short recibidoSensor;
    static char* recibidoUsuario;
    static QueueEstruct* log;

    for(;;){
        xQueueReceive(log_queue, &log, portMAX_DELAY);
        if (log ->eTipoDato == TipoValor){
            recibidoSensor = *((unsigned short*) log -> pDato);
            PRINTF("LOG[%d]: Cantidad de interrupciones por %s: %u \r\n",
counter, BOARD_SW_NAME, recibidoSensor);
        }
        else if (log -> eTipoDato == TipoString){
            recibidoUsuario = ((char*) log ->pDato);
            PRINTF("LOG[%d]: %s\r\n",counter, recibidoUsuario);
        }
        counter++;
    }
}
```

pin_mux.c

```
#include "fsl_common.h"
#include "fsl_port.h"
#include "pin_mux.h"

#define PIN4_IDX                         4u   /*!< Pin number for pin 4 in a port */
#define PIN16_IDX                       16u   /*!< Pin number for pin 16 in a port */
#define PIN17_IDX                       17u   /*!< Pin number for pin 17 in a port */
#define PIN22_IDX                       22u   /*!< Pin number for pin 22 in a port */

#define SOPT5_UART0TXSRC_UART_TX      0x00u   /*!< UART 0 transmit data source select:
UART0_TX pin */

void BOARD_InitPins(void) {
  CLOCK_EnableClock(kCLOCK_PortA);  /* Port A Clock Gate Control: Clock enabled */
  CLOCK_EnableClock(kCLOCK_PortB);  /* Port B Clock Gate Control: Clock enabled */

  const port_pin_config_t porta4_pin38_config = {
    kPORT_PullUp,                   /* Internal pull-up resistor is enabled */
    kPORT_FastSlewRate,             /* Fast slew rate is configured */
    kPORT_PassiveFilterDisable,     /* Passive filter is disabled */
    kPORT_OpenDrainDisable,         /* Open drain is disabled */
    kPORT_HighDriveStrength,        /* High drive strength is configured */
    kPORT_MuxAsGpio,                /* Pin is configured as PTA4 */
    kPORT_UnlockRegister            /* Pin Control Register fields [15:0] are not locked
*/
  };
  PORT_SetPinConfig(PORTA, PIN4_IDX, &porta4_pin38_config);  /* PORTA4 (pin 38) is
configured as PTA4 */
  PORT_SetPinMux(PORTB, PIN16_IDX, kPORT_MuxAlt3);   /* PORTB16 (pin 62) is configured
as UART0_RX */
```

```
  PORT_SetPinMux(PORTB, PIN17_IDX, kPORT_MuxAlt3);   /* PORTB17 (pin 63) is configured
as UART0_TX */
  PORT_SetPinMux(PORTB, PIN22_IDX, kPORT_MuxAsGpio);  /* PORTB22 (pin 68) is configured
as PTB22 */
  SIM->SOPT5 = ((SIM->SOPT5 &
    (~(SIM_SOPT5_UART0TXSRC_MASK)))          /* Mask bits to zero which are setting */
      | SIM_SOPT5_UART0TXSRC(SOPT5_UART0TXSRC_UART_TX)  /* UART 0 transmit data source
select: UART0_TX pin */
    );
}
```

FreeRTOSConfig.h

```
#ifndef FREERTOS_CONFIG_H
#define FREERTOS_CONFIG_H

#define configUSE_PREEMPTION                    1
#define configUSE_TICKLESS_IDLE                 0
#define configCPU_CLOCK_HZ                      (SystemCoreClock)
#define configTICK_RATE_HZ                      ((TickType_t)1000)
#define configMAX_PRIORITIES                    5
#define configMINIMAL_STACK_SIZE                ((unsigned short)90)
#define configMAX_TASK_NAME_LEN                 20
#define configUSE_16_BIT_TICKS                  0
#define configIDLE_SHOULD_YIELD                 1
#define configUSE_TASK_NOTIFICATIONS            1
#define configUSE_MUTEXES                       1
#define configUSE_RECURSIVE_MUTEXES             1
#define configUSE_COUNTING_SEMAPHORES           1
#define configUSE_ALTERNATIVE_API               0 /* Deprecated! */
#define configQUEUE_REGISTRY_SIZE               8
#define configUSE_QUEUE_SETS                    0
#define configUSE_TIME_SLICING                  0
#define configUSE_NEWLIB_REENTRANT              0
#define configENABLE_BACKWARD_COMPATIBILITY     0
#define configNUM_THREAD_LOCAL_STORAGE_POINTERS 5
#define configUSE_APPLICATION_TASK_TAG          0

/* Used memory allocation (heap_x.c) */
#define configFRTOS_MEMORY_SCHEME               4
/* Tasks.c additions (e.g. Thread Aware Debug capability) */
#define configINCLUDE_FREERTOS_TASK_C_ADDITIONS_H 1

/* Memory allocation related definitions. */
#define configSUPPORT_STATIC_ALLOCATION         0
#define configSUPPORT_DYNAMIC_ALLOCATION        1
#define configTOTAL_HEAP_SIZE                   ((size_t)(10 * 1024))
#define configAPPLICATION_ALLOCATED_HEAP        0

/* Hook function related definitions. */
#define configUSE_IDLE_HOOK                     0
#define configUSE_TICK_HOOK                     0
#define configCHECK_FOR_STACK_OVERFLOW          0
#define configUSE_MALLOC_FAILED_HOOK            0
#define configUSE_DAEMON_TASK_STARTUP_HOOK      0

/* Run time and task stats gathering related definitions. */
#define configGENERATE_RUN_TIME_STATS           0
#define configUSE_TRACE_FACILITY                1
#define configUSE_STATS_FORMATTING_FUNCTIONS    0

/* Co-routine related definitions. */
```

```
#define configUSE_CO_ROUTINES                   0
#define configMAX_CO_ROUTINE_PRIORITIES         2

/* Software timer related definitions. */
#define configUSE_TIMERS                        1
#define configTIMER_TASK_PRIORITY               2
#define configTIMER_QUEUE_LENGTH                10
#define configTIMER_TASK_STACK_DEPTH            (configMINIMAL_STACK_SIZE * 2)

/* Define to trap errors during development. */
#define configASSERT(x) if((x) == 0) {taskDISABLE_INTERRUPTS(); for (;;);}

/* Optional functions - most linkers will remove unused functions anyway. */
#define INCLUDE_vTaskPrioritySet                1
#define INCLUDE_uxTaskPriorityGet               1
#define INCLUDE_vTaskDelete                     1
#define INCLUDE_vTaskSuspend                    1
#define INCLUDE_xResumeFromISR                  1
#define INCLUDE_vTaskDelayUntil                 1
#define INCLUDE_vTaskDelay                      1
#define INCLUDE_xTaskGetSchedulerState          1
#define INCLUDE_xTaskGetCurrentTaskHandle       1
#define INCLUDE_uxTaskGetStackHighWaterMark     0
#define INCLUDE_xTaskGetIdleTaskHandle          0
#define INCLUDE_eTaskGetState                   0
#define INCLUDE_xEventGroupSetBitFromISR        1
#define INCLUDE_xTimerPendFunctionCall          1
#define INCLUDE_xTaskAbortDelay                 0
#define INCLUDE_xTaskGetHandle                  0
#define INCLUDE_xTaskResumeFromISR              1


/* Interrupt nesting behaviour configuration. Cortex-M specific. */
#ifdef __NVIC_PRIO_BITS
/* __BVIC_PRIO_BITS will be specified when CMSIS is being used. */
#define configPRIO_BITS __NVIC_PRIO_BITS
#else
#define configPRIO_BITS 4 /* 15 priority levels */
#endif

/* The lowest interrupt priority that can be used in a call to a "set priority"
function. */
#define configLIBRARY_LOWEST_INTERRUPT_PRIORITY ((1U << (configPRIO_BITS)) - 1)

#define configLIBRARY_MAX_SYSCALL_INTERRUPT_PRIORITY 2

#define configKERNEL_INTERRUPT_PRIORITY (configLIBRARY_LOWEST_INTERRUPT_PRIORITY
<< (8 - configPRIO_BITS))

#define configMAX_SYSCALL_INTERRUPT_PRIORITY
(configLIBRARY_MAX_SYSCALL_INTERRUPT_PRIORITY << (8 - configPRIO_BITS))

/* Definitions that map the FreeRTOS port interrupt handlers to their CMSIS
standard names. */
#define vPortSVCHandler SVC_Handler
#define xPortPendSVHandler PendSV_Handler
#define xPortSysTickHandler SysTick_Handler

/* Do not include if processing assembly file */
#if (!defined(__IAR_SYSTEMS_ASM__) && !defined(__ASSEMBLER__))
#   include "fsl_device_registers.h"
#   include "trcKernelPort.h"
#endif

#endif /* FREERTOS_CONFIG_H */
```