Cerca i Implementació d'un RTOS per a un system on chip PSoC

Jordi Bellés Solà

Resum—En aquest document es presenten un seguit de conceptes relacionats amb els sistemes operatius en temps real per entendre el context en què s'utilitzen i perquè s'utilitzen, enfocats al desenvolupament d'aplicacions en la plataforma PSoC i el seu idle PSoC Creator. A més es pot trobar un petit estudi teòric que a partir de la documentació oficial, compara les característiques i funcionalitats que aporten els diferents sistemes operatius compatibles amb els processadors ARM Cortex-M3, FreeRTOS, Zephyr, ChibiOS, ContikiOS i RiotOS. Analitzant les diferents característiques s'ha decidit que RiotOS podria ser un possible sistema operatiu a implementar gràcies al poc consum de memòria, les característiques i funcionalitats que aporta. A partir d'aquest resultat i al treballar amb una placa de desenvolupament PSoC 5 s'ha centrat l'interès en sistemes operatius compatibles amb aquesta plataforma, FreeRTOS, Micrium i embOS. S'han analitzat tant funcionalitats com característiques i s'han implementat diferents aplicacions per veure consums de memòria i energia que han permès concloure que, a partir dels resultats obtinguts, FreeRTOS és el més adient per implementar amb restriccions de memòria i energia.

Paraules clau—Sistema operatiu en temps real, multitasca, Tasques, System on Chip, PSoC 5, IoT, Programació Seqüencial, FreeRTOS, ARM, Cortex-M3, Sistema operatiu de propòsit general.

Resumen— En este documento se presentan una serie de conceptos relacionados con los sistemas operativos en tiempo real para entender el contexto en que se utilizan y para que se utilizan, enfocados al desarrollo de aplicaciones en la plataforma PSoC y PSoC Creator. Además, se ha desarrollado un pequeño estudio teórico qué, a partir de la documentación oficial, compara las características y funcionalidades que aportan los diferentes sistemas operativos compatibles con los procesadores ARM Cortex-M3, FreeRTOS, Zephyr, ChibiOS, ContikiOS y RiotOS. Analizando las diferentes características se ha decidido que RiotOS podría ser un posible sistema operativo implementar gracias al poco consumo de memoria, las características y funcionalidades que aporta. A partir de este resultado y al trabajar con una placa de desarrollo PSoC 5 se ha centrado el interés en sistemas operativos compatibles con esta plataforma, FreeRTOS, Micrium y embOS. Se han analizado tanto funcionalidades como características y se han implementado diferentes aplicaciones para ver consumos de memoria y energía que han permitido concluir que, a partir de los resultados obtenidos, FreeRTOS es el más adecuado para implementar con restricciones de memoria y energía.

Paraules clau—Sistema operativo en tiempo real, multitarea, Tareas, System on Chip, PSoC 5, IoT, Programación Secuencial, FreeRTOS, ARM, Cortex-M3, Sistema operativo de propósito general.

Abstract— This document presents a series of concepts related to real-time operating systems to understand the context in which they are used and for which they are used, focused on the development of applications on the PSoC and PSoC Creator platform. In addition, a small theoretical study has been developed which, from the official documentation, compares the characteristics and functionalities provided by the different operating systems compatible with the ARM Cortex-M3, FreeRTOS, Zephyr, ChibiOS, ContikiOS and RiotOS processors. Analysing the different characteristics, it has been decided that RiotOS could be a possible operating system to implement thanks to the low memory consumption, the features and functionalities it provides. From this result and working with a PSoC 5 development board, interest has been focused on operating systems compatible with this platform, FreeRTOS, Micrium and embOS. Both functionalities and features have been analysed and different applications have been implemented to see memory and energy consumption that have allowed us to conclude that, based on the results obtained, FreeRTOS is the most suitable to implement with memory and energy restrictions

Index Terms—Real time operating system, multitasking, Tasks, System on Chip, PSoC 5, Sequential Programming, FreeRTOS, ARM, Cortex-M3, General-purpose operating system.

1 Introducción

El objetivo principal de este proyecto es el de realizar una búsqueda de los RTOS del mercado para un sistema PSoC basado en arquitectura ARM Cortex-M e implementar sobre una placa de desarrollo para conocer el funcionamiento. PSoC es el acrónimo de Programmable System on Chip. Se trata de un sistema complejo que incorpora en un solo chip de silicio uno o varios procesadores de 32 bits CORTEX, memoria flash y Ram, UARTS de comunicaciones de diferentes protocolos, y varios módulos digitales y analógicos de alta complejidad.

A su vez, el documento está dividido en diferentes apartados.

El primer punto, planificación, se explica cómo se ha planificado este proyecto y que se ha desarrollado en cada fase. A partir de la planificación, hay tres apartados enfocados a dar a conocer conceptos relacionados con los RTOS y el contexto del porqué de su uso. El segundo apartado se define que es un sistema operativo em tiempo real y cuáles son sus funcionalidades. En el tercer apartado se nuestras los beneficios que aporta la programación multitarea que se aplica en los RTOS frente la programación secuencial. En el cuarto apartado se explican cuáles son las ventajas de implementar aplicaciones con el paradigma de programación de los sistemas ope-

rativos de tiempo real frente el diseño de aplicaciones en la plataforma PSoC Creator. A partir de este punto el siguiente apartado se hace un análisis de diferentes RTOS del mercado compatibles con los procesadores ARM Cortex-M3 y el siguiente punto un análisis de tres RTOS compatibles con PSoC 5, donde se ha implementado una aplicación de prueba para ver los consumos de memoria y energía de cada uno, aparte de comparar funcionalidades. Para finalizar un último apartado de conclusiones sobre el trabajo y el sistema operativo ideal para desarrollar aplicaciones en PSoC cumpliendo las necesidades de energía y memoria.

2 Planificación

El proyecto se ha realizado en diferentes fases siguiendo la planificación inicial. La primera fase ha permitido conocer los conceptos básicos necesarios sobre los sistemas operativos en tiempo real y que beneficios aportan respecto al paradigma de programación secuencial. Aparte, al tener que implementar el sistema operativo en un chip PSoC se analiza las ventajas e inconvenientes de porqué utilizar un RTOS frente al desarrollo de la aplicación directamente en PSoC Creator.

La segunda fase trata de hacer un análisis de diferentes RTOS en el mercado, compatibles con los procesadores Cortex-M3/4; comparando características para decidir cuáles podrían ser utilizados para ser implementados en PSoC.

Después de realizar este análisis, guiado por los tutores, me instaron a centrarme en la família PSoC 5. Por lo que en esta fase hice una nueva búsqueda de sistemas operativos ya compatibles y con proyectos de prueba ya realizados para PSoC 5 y analizar su funcionamiento. He tenido algunos problemas y retrasos para hacer funcionar algunos de los sistemas operativos e implementar la aplicación de prueba para testear consumos tanto de memoria o energía. A pesar del retraso he podido implementar la aplicación de prueba en los tres sistemas operativos compatibles con PSoC 5 y realizar el análisis del consumo de memoria de cada uno en diferentes situaciones.

Una vez implementados los sistemas operativos y configurados los proyectos para ejecutar las aplicaciones se ha analizado el consumo de energía para cada uno en diferentes situaciones, una aplicación que envía datos por una UART, una aplicación que utiliza un sensor para medir distancias y una aplicación que calcula la Transformada Rápida de Foruier.

Para acabar, se han extraído las conclusiones a partir del análisis de los resultados y las características y funcionalidades que aportan cada uno de los sistemas operativos para decidir qué sistema operativo sería el ideal utilizar para desarrollar aplicaciones.

3 Definición RTOS

La función principal de un sistema operativo de propósito general es la de la gestionar los recursos hardware del sistema y asignar estos recursos según las necesidades de las aplicaciones instaladas para asegurar una ejecución correcta y eficiente.

Un RTOS realiza las funciones de un sistema operativo de propósito general, con unos requerimientos diferentes para ejecutar aplicaciones de tiempo real.
En estos sistemas hay operaciones, procesamiento de datos, cálculos, llamadas al sistema, manejo de interrupciones que deben realizarse de forma correcta y además en un intervalo de tiempo específico. Un retraso o fallo en estas operaciones críticas puede conllevar un fallo del sistema. Se utilizan, como norma, planificadores basados en prioridades y se implementan en dispositivos aislados con gran conectividad y restricciones de memoria y consumo de energía.
Se pueden clasificar en hard si al retrasarse un resultado conlleva un fallo del sistema; Firm si se tolera el retraso de algún resultado, pero la calidad de servicio del sistema disminuye y el resultado pierdo todo su valor y Soft en que la utilidad del resultado va disminuyendo conforme se aleja de la meta de tiempo y a la vez se va degradando la calidad de servicio del sistema.[1][2]

4 Beneficios Programación Multitarea Frente Programación Secuencial

La programación multitarea se basa en la ejecución de más de una tarea al mismo tiempo. Las nuevas tareas pueden interrumpir la ejecución de otras tareas antes de que estas finalicen, en vez de esperar que estas acaben. Como resultado, se van ejecutando segmentos de múltiples tareas en intervalos compartiendo recursos como la CPU o la memoria principal. Las tareas son interrumpidas de forma automática, almacenando su estado y otorgando el uso de los recursos a otra tarea, normalmente de mayor prioridad. Este cambio de ejecución de tareas, nombrado cambio de contexto se puede realizar en intervalos fijos, pre-empitve multitasking, o la tarea puede avisar al sistema cuando puede ser interrumpida, cooperative multitasking.

Gracias a cómo funcionan los sistemas operativos multitarea permite a tareas de mayor prioridad ejecutarse antes que otras de baja prioridad o en momentos en que una tarea se queda en estado bloqueado. Es en ese momento cuando entra en juego el 'context switch', que

- *E-mail de contacte: jordi.belles@e-campus.uab.cat*
- *Menció realitzada: Enginyeria de Computadors.*
- *Treball tutoritzat per: Raul Aragonés i Roger Malet*
- *Curs 2019/20*

asigna tiempo de ejecución a una nueva tarea. De esta forma se hace un uso más eficiente de los recursos del sistema.

5 VENTAJAS DE UTILIZAR UN RTOS

En este apartado explicaré cuales son las ventajas de utilizar un RTOS frente al diseño de software en PSOC Creator y en que aplicaciones se suelen utilizar estos diferentes diseños.

El diseño de software en PSOC Creator se implementa mediante un único bucle infinito en el main del programa, Normalmente este concepto se utiliza en sistemas y aplicaciones pequeñas, no muy complejas. El código se ejecuta en un orden fijo que permite entender el flujo de la aplicación y estimar el tiempo de ejecución de la aplicación. Se utilizan ISR para ejecutar código crítico en tiempo, código que se ejecuta como respuesta a eventos o interrupciones de hardware.
Este diseño se utiliza en aplicaciones más simples, pero tienen limitaciones en aplicaciones más complejas.
Estas limitaciones incluyen algunas desventajas y sus efectos:

Al aumentar la complejidad de la aplicación con nuevas funciones hace más difícil al sistema cumplir con las restricciones de tiempo de la aplicación y predecir el tiempo de ejecución.

La información entre el thread principal y las interrupciones se comparte mediante variables compartidas, conlleva implementar métodos para asegurar la consistencia de los datos.

ISR son complejas y utilizan gran cantidad de tiempo de ejecución.

La ejecución de interrupciones encadenadas puede conllevar a un tiempo de ejecución de la aplicación impredecible.[3]

La característica principal para trabajar con un RTOS es la complejidad de la aplicación. Como aparece en [4], se utilizan RTOS cuando las aplicaciones tienen un gran número de funciones o fuentes de interrupciones y demandan interfaces estándar de comunicación. Trabajar con un RTOS permite relevar parte de la complejidad de la aplicación al sistema ya que separa las funciones del programa en tareas independientes y es el propio planificador, según la prioridad de las tareas, quien asigna el tiempo de ejecución.

Algunas de las ventajas y beneficios de un RTOS son el concepto de multitarea, tiempo de ejecución de las interrupciones reducido y dentro de un margen de tiempo, implementan estructuras y servicios para la comunicación entre tareas para evitar estructuras de datos compartidos, al definir el tamaño de la pila de las tareas se puede predecir el uso de memoria además de otras estructuras de datos usadas en la aplicación. Algunas de las características de los RTOS y sus beneficios son ampliados en el siguiente párrafo.

Multitarea y planificación apropiativa: este paradigma de diseño permite responder a cada uno de los eventos en tiempo real de forma independiente; esto permite añadir nuevas funcionalidades o tareas sin que afecten al tiempo de respuesta de las tareas existentes. A diferencia del diseño con un bucle secuencial donde cada evento se comprueba mediante un mecanismo de sonde y el añadir una nueva función afecta al tiempo de respuesta de todos los eventos de la aplicación. Aparte de que el multitasking separa la ejecución de las funcionalidades de una aplicación em diferentes tareas los RTOS implementan un planificador de tareas, este se encarga de definir el orden de ejecución las tareas según la prioridad. No es necesario tener control sobre el tiempo de ejecución del bucle para cumplir con los deadlines de las tareas.

Eficiencia: un beneficio de los RTOS y que deriva del concepto de multitarea permite utilizar la CPU en todo momento. El planificador se encarga de decidir qué tareas se ejecutan y en el caso de no haber ninguna puede llevar al procesador al estado de bajo consumo. Esto se diferencia de las aplicaciones con un único bucle donde la CPU puede quedar asignada a una tarea bloqueada donde espera a datos o se gastan los recursos ejecutando el bucle esperando a que acurran eventos.

Portabilidad: mediante librerías y APIs ofrecen una abstracción del hardware que permite centrarse en el desarrollo del software y hacer aplicaciones más portables. Las aplicaciones serán compatibles con las plataformas para los que se han realizado ports, no es necesaria ninguna configuración.

Servicios: los RTOS aportan una serie de servicios para la gestión de recursos, gestión de memoria, mensajes entre tareas e interrupciones, manejo de interrupciones entre otros que permiten al desarrollador centrarse en la implementación de la aplicación.

Herramientas de análisis: conforme las aplicaciones son más complejas se hace más difícil entender el funcionamiento y el uso de recursos que hace la aplicación del sistema. La mayoría de RTOS implementan una serie de herramientas para el análisis y el diagnóstico de las aplicaciones. Se utilizan para observar el flujo de ejecución de la aplicación y el uso de recursos.[5]

Para concluir, aplicaciones simples con pocas funciones se pueden implementar mediante un bucle infinito; ya que se pueden gestionar de forma fácil y a la vez cumplir con los requisitos de restricción de tiempo de respuesta. Cuando se desarrollan aplicaciones más complejas con un gran número de tareas o que necesitan ser ampliadas en el tiempo, implementarlas en un RTOS permite aprovechar las ventajas nombradas anteriormente, a la vez que se usan los recursos de forma eficiente y permite ahorrar tiempo de desarrollo.

6 ANÁLISIS Y COMPARATIVA DE RTOS COMPATIBLES CON PROCESADORES CORTEX-M3/4

6.1 ChibiOS [7]

Sistema modular, dividido internamente en diferentes componentes independientes:

- Kernel
- Port Layer
- Hardware Abstraction Layer
- Platform Layer
- Board Initalization
- Various, librería para diferentes utilidades.

6.1.1 Caracteristicas y funcionalidades:

- Virtual Timers. One-shot timers con la misma resolución que el system tick.
- Modulo Threading: incluye diferentes servicios relacionados con la ejecución, manejo y ejecución de threads
- Semáforos: Counting Semaphores y Binary Semaphores
- Mutexes y Variables de condición
- Synchronous Messages, funcionalidad única que permite crear arquitecturas cliente/servidor en un sistema embedido.
- Mailboxes
- Events
- Streams (Data Streams, I/O Channels, I/O Queues)
- Memory Management (Core Allocator, Memory Heaps, Memory Pools, Dynamic Threads)
- Subsistema para debug

6.1.2 Requerimientos del sistema:

- Compatible con procesadores de 8, 16 y 32-bits.
- Consume como mínimo 2 kB de RAM.
- De 8KB de Flash/ROM para el kernel con todas sus funcionalidades hasta un mínimo de 1,2KB para configuraciones reducidas. Se recomienda un mínimo de 16KB.
- Stack pointer real
- Soporte para el compilador C99.

Soporta de forma oficial la arquitectura de ARM Cortex-M.

6.1.3 Gestión de Energía:

El Kernel no incluye de forma directa ningún modo de gestión de energía, pero si proporciona un conjunto funciones para poder implementar dicho modo como respuesta a ciertas acciones del Kernel.
El kernel de ChibiOS incluye el Tickless Mode que permite desactivar las interrupciones generadas por el timer del sistema y reducir el consumo.

6.1.4 Interrupciones:

Implementa funciones de manejo de interrupciones en la API.

6.2 ContikiOS [8][9]

Modelo híbrido basado en un kernel controlado por eventos donde se implementa multi-threading apropiativo mediante una librería siempre que la aplicación lo requiera.

- Kernel
- Librerias
- Program Loader
- Conjunto de procesos, pueden ser aplicaciones o servicios

6.2.1 Características y funcionalidades:

- Incluye uIP TCP/IP stack. uIP proporciona los protocolos TCP, UDP, IP y ARP.
- Contiki file system interface define una API para leer directorios y leer/escribir ficheros.
- Clock library, interfície entre Contiki y la funcionalidad de reloj específica de la plataforma.
- Callback Timer
- Event timers
- Multi-threading library
- Protothreads
- Planficiación de tares en tiempo real
- Timer y stimer library, para crear, parar y configurar timers.
- Interrupt vector numbers

6.2.2 Requerimientos del sistema:

- Consumo normal de 2 kB de RAM.
- Consume como máximo con todas las funciones, incluyendo GUI, un total de 40 kB de ROM.

Compatible con arquitectura ARM Cortex-M (CMSIS)

6.2.3 Gestión de Energía

No dispone de un mecanismo estándar para controlar el consumo de los dispositivos. Si dispone de mecanismos para reducir el consumo en plataformas orientadas a sensores y transmisión de datos por red, ContikiMAC radio duty cycling protocol.

6.2.4 Interrupciones:

No tiene implementado IRQ Handler, utiliza ISR Vector del microcontrolador.

6.3 Riot [10]

El sistema se estructura de la siguiente forma:

- Boards: definiciones e implementaciones específicas para placas
- CPU: implementaciones para CPU específicas
- Lista de configuraciones en tiempo de compilación
- Drivers: drivers para dispositivos externos como radios, sensores, memorias, etc
- Kernel, micro-kernel que contiene la funcionalidad central
- Networking, librerías para dar soporte a las diferentes pilas de red, buses y servicios relacionados.
- Packages, librerías externas y aplicaciones.
- System, librería que contiene las herramientas y utilidades que hacen de RIOT un sistema operativo.

6.3.1 Características y funcionalidades:

- Semáforos
- Timers
- Manejo de memoria
- Mailboxes
- Threading, multi-threading
- API para comunicación entre procesos
- Librerías para diferentes sistemas de ficheros
- Colas de eventos

6.3.2 Requerimientos del sistema:

- Consume como mínimo 1.5 kB de RAM
- La configuración del kernel ocupa 5 kB de ROM. Compatible con la arquitectura ARM Cortex-M.

6.3.3 Gestión de energía:

Interfaz que se debe implementar para cada microcontrolador. Implementa funciones para reiniciar el MCU, apagarlo o ponerlo en modo de mínimo consumo.

6.3.4 Interrupciones:

Implementa funciones de manejo de interrupciones en la API.

6.4 FreeRTOS [11]

6.4.1 Características y funcionalidades:

- Colas
- Binary y Counting semáforos
- Mutex y mutex recursivos
- Software timers
- Stack overflow checking

6.4.2 Requerimientos del sistema:

- El consumo de RAM depende de la aplicación, ya que hay que tener en cuenta la configuración y sus necesidades. El scheduler tiene un tamaño de 256 bytes más el tamaño de cada cola,76 bytes cada una y el área de almacenamiento de la cola y además el tamaño de cada task, 64 bytes.
- Se necesita un tamaño entre 5-10 kB ROM, dependiendo del compilador que se usa, la arquitectura i la configuración del kernel.

Compatible con la arquitectura ARM Cortex-M.

6.4.3 Gestión de energía:

Implementa el modo Tickles Idle Mode, que para las interrupciones generadas por el timer tick durante los periodos en que la CPU está en idle, no está trabajando. Permite al microcontrolador mantenerse en el estado de bajo consumo hasta que se genera otra interrupción o el kernel deba cambiar el estado de una tarea a preparada.

6.4.4 Interrupciones:

Implementa el manejo de interrupciones mediante funciones en la API, pero como hay algunas funciones de la API que no se pueden realizar dentro de una interrupción ya que llevarían la tare al estado bloqueado; se han implementado dos versiones de algunas funciones de la API.
Unas funciones son para usarlas desde una tarea y otras desde ISR.

6.5 Zephyr [12][13]

La arquitectura del sistema es la siguiente:

- Kernel
- OS Services
- Application Services

6.5.1 Características y funcionalidades:

- Servicios del Kernel:
- Multi-threading
- Polling API
- Semáforos
- Mutexes
- Multiprocesamiento simétrico
- Data Passing, kernel objects que se pueden utilizar para pasar datos entre threads e interrupciones. (FIFO, LIFO, Stack, Message queue, Mailbox, Pipe)
- Memory Management (Memory Slabs, Memory Pools, Heap Memory Pool)

6.5.2 Requerimientos del sistema:

Se puede ejecutar Zephyr en 2 kB de RAM, con una configuración normal de 8 kB.
Compatible con la arquitectura ARM Cortex-M.

6.5.3 Gestión de energía:

Las funciones del gestor de energía se clasifican en las siguientes categorías: Tickless Idel, System Power Management y Device Power Management.
La función Tickless Idle es la de programar el thread idle, que para las interrupciones que genera el timer y a la vez invoca el scheduler. Para reducir el consumo de estas acciones, se deja el timer
Con el System Power Management, permite definir el estado del sistema.
El Device Power Management son un conjunto de funciones en la API que permiten enviar comandos de control al controlador de los periféricos para modificar su estado u obtenerlo.

6.5.4 Interrupciones:

Implementa funciones de manejo de interrupciones en la API.

6.6 Conclusión

Después de ver las diferentes características de los RTOS que implementar en un system on chip PSoC, se ha decidido realizar el desarrollo con el sistema operativo RiotOS. En el caso que no fuera posible desarrollar una implementación para el sistema PSoC se trabajaría en la implementación de Zpehyr o ChibiOS; ya que son parecidos a Riot-OS tanto en consumo de memoria cómo en características y funcionalidades. Aparte que no tienen un port ya realizado como FreeRTOS por lo que queda descartado, aunque cumple los requisitos para desarrollar aplicaciones para sistemas PSoC.

Se ha decidió que Riot-OS sería una buena opción para implementar en PSoC por ser el sistema operativo de menor tamaño entre los escogidos y que a la vez incluye todas las funcionalidades de un RTOS. También por su sistema modular que se diferencia en módulos dependientes del hardware y otros módulos independientes del hardware. Un punto importante es que al implementar un microKernel se consigue reducir el tamaño de este y ahorrar en memoria. También incluye un modo de gestión de energía, aunque no tan completo como por ejemplo el de Zephyr o FreeRTOS pero que nos permite controlar el estado del microcontrolador.

	Zephyr	ChibiOS	FreeRTOS	Riot	Contiki
RAM	8kB	2kB	Depende de la aplicación	1.5kB	2kB
ROM/FLASH	-	1.2-8-16kB	6-12kB	5kB	40kB
IRQ	IRQ handler	IRQ handler (API)	IRQ handler (API)	IRQ Handler (API)	ISR Vector, no IRQ handler
Power Management	Si	Tickless mode	Si, diferentes modos	Si	Si, solo networking
Compatibilidad Threads	Threads	Paradimga completo	Task (Threads)	Multi-Threading	Protothreads, multithreading con una libreria
Scheduler	Basado en prioridades	Basado en prioridades, apropiativo	Fixed priority, Pre-emptive y Time slicing	Tickless, preemptive, priority based	Event-based
Kernel	Monolithic	uKernel	RTOS Kernel	uKernel	-

Fig. 1. Tabla comparativa características RTOS.

7 COMPARATIVA DE RTOS COMPATIBLES CON PSOC 5

7.1 Micrium [14]

Características y funcionalidades:
- Multitasking apropiativo
- Planificación Round Robin
- Event flags
- Semáforos y Exclusión Mutua
- Mailboxes

7.1.1 Requerimientos del sistema:

ROM mínima necesaria entre 6kB y24kB. Uso mínimo de RAM aproximadamente de 1kB.

7.1.2 Gestión de energía:

No aparece información en la documentación oficial del producto, tanto en la pàgina web como en la wiki o manual de uso.

7.1.3 Interrupciones:

No incluye gestor de interrupciones, se tiene que implementar en código ensamblador.

7.2 embOS [15]

7.2.1 Características y funcionalidades:

- Tasks
- Software Timers
- Semáforos
- Mailboxes
- Queues
- Watchdog
- Regiones Críticas

7.2.2 Requerimientos del sistema:

Uso mínimo de ROM (Kernel Size) aproximadamente 1.7 kB y el consumo de RAM en función de las necesidades de la aplicación:
- Kernel: 71 bytes
- Task control block: 36 bytes de RAM
- Semaphore: 16 bytes de RAM
- Ccounting semaphore: 8 bytes de RAM
- Mailbox: 24 bytes de RAM
- Software Timer: 20 bytes de RAM
- Número máximo de tareas, mailboxes, semaphores, software timers ilimitado, depende de la RAM disponible.

7.2.3 Gestión de energía:

Proporciona tres modos para controlar el consumo de energía:
- La posibilidad para activar el modo de ahorro de energía con la función de embOS OS_Idle().
- Soporte para el modo tickless, permite al microcontrolador mantenerse en modo ahorro de energía durante largos períodos de tiempo
- Módulo para el control de consumo de periféricos, permite controlar el consumo de algunos periféricos específicos.

7.2.4 Interrupciones:

Implementa funciones de manejo de interrupciones mediante API.

Después de ver las características principales de los sistemas operativos vamos a comparar el uso de memoria, tanto en RAM como Flash que consume cada sistema operativo. Pero antes mostraré las características de la placa de desarrollo sobre la que he probado los sistemas operativos y la aplicación de prueba.

7.3 Características placa desarrollo

Kit de desarrollo CY8CKIT-059 PSoC 5LP que lleva el microcontrolador CY8C5888LTI-LP097. Incluye periféricos analógicos y digitales de alta precisión y programables con un procesador ARM Cortex-M3 en un solo chip. Puede trabajar a una frecuencia máxima de 80Mhz, con una memoria Flash de 256kB, 64kB de RAM, 2kB EEPROM y 48 IO.

Periféricos digitales como:
- 4 timers de 16-bits, contadores y bloques PWM,
- Bus I2C
- USB
- CAN 2.0b, 16 Rx y 8 Tx buffers
- 24 UDB

Periféricos analógicos:
- Un conversor analógico-digital delta-sigma configurable de 8-20-bits de resolución
- Hasta 2 ADC SAR de 12-bit
- 4 DAC 8-bit

Más información sobre el kit en [16].

7.4 Consumo Memoria

En este apartado se muestra el consumo de los diferentes RTOS, FreeRTOS, Micrium y embOS, en términos de memoria en diferentes casos. La placa, como se ha comentado anteriormente contiene 256kB de memoria Flash y 64kB de RAM.
Estos son las diferentes situaciones en que se ha medido el consumo de memoria:
- Empty: se ejecuta el sistema operativo sin ninguna tarea creada.
- Una tarea: se ejecuta el sistema operativo con una

sola tarea. Parpadea un led.
- Acc: se ejecuta la aplicación de prueba explicada anteriormente que simula el funcionamiento de un acelerómetro.

7.4.1 Aplicación de prueba:

La aplicación simula el funcionamiento de un acelerómetro que envía datos a la placa, los procesa, mediante leds indica la posición que se mueve el sensor y envía los datos procesados por UART para verlos por un terminal.

Está formada por 4 tareas.
- **vAccTask**: tarea que lee los datos del sensor, se generan los datos con una función generadora de números aleatorio y los envía a la tarea que los procesa.
- **vProcessData:** tarea que recibe los datos sin procesar, los procesa y envía los datos a la tarea de la UART para mostrarlos por un terminal y la señal a la tarea Toggle para encender los leds.
- **vToggle:** recibe la señal de la tarea que procesa los datos para encender los leds.
- **vUART:** recibe los datos procesados y los escribe por la UART. Se pueden ver en un terminal serie.
-

Para enviar los datos entre tareas se utilizan colas, Queues en FreeRTOS y Micrium y Mailboxes en EmbOS. Se utilizan un total de 3 para transmitir los datos entre tareas. Se utilizan dos estructuras de datps, sensor para almacenar los datos del sensor y cord para indicar que leds se deben encender e indicar el movimiento del acelerómetro

7.4.2 embOS

En embOS la pila de las tareas se define como un array de enteros, con un tamaño de 1024 bytes para que se puedan ejecutar las diferentes aplicaciones de prueba.
Los mailboxes, tiene el mismo funcionamiento de las colas en FreeRTOS, utilizan cada una un array del tamaño de la estructura que se utilizara para enviar entre tareas. Cada cola almacenará un máximo de 8 mensajes cada una.

7.4.3 FreeRTOS

La aplicación funciona con la pila de las tareas de 512 bytes. Las tareas y colas utilizan la RAM reservada del Heap de FreeRTOS, la cantidad de memoria se reserva en el archivo de configuración FreeRTOSConfig.h como se indica en la documentación oficial.[17][18]
Para la ejecución del sistema operativo sin ninguna tarea es suficiente reservar 512 bytes de memoria, para ejecutar una tarea que enciende y apaga un led se reservan 2048 bytes y para la ejecución de la aplicación de prueba se reservan 4096 bytes

7.4.4 Micrium uC/OS-III

La pila de las tareas es de 512 bytes para que se ejecuten las aplicaciones de prueba.

	embOS	Micrium	FreeRTOS
Empty	4837 bytes (7,4 %)	7137 bytes (10,9 %)	3305 bytes (5,0 %)
Una tarea	5437 bytes (8,3 %)	7841 bytes (12,0 %)	4841 bytes (7,4 %)
Acelerómetro	10485 bytes (16,0 %)	10121 bytes (15,4 %)	6921 bytes (10,6 %)

Fig. 2. Tabla comparativa uso RAM.

EmbOS como Micrium son sistemas operativos privados por lo que se debe adquirir una licencia para la producción de productos o si se quiere tener acceso al código fuente del sistema; lo que dificulta el desarrollo de las aplicaciones y además no se recomienda la implementación de aplicaciones desde cero sino a partir de proyectos.

En cambio, FreeRTOS destaca por ser un sistema operativo de código abierto, el acceso al código fuente es inmediato y sencillo crear proyectos incluyendo solo los archivos base necesarios para ejecutar el sistema y la aplicación.

Los resultados se han realizado implementado las mismas aplicaciones en los diferentes sistemas, asegurando el mismo funcionamiento. Como se puede observar, FreeRTOS es el sistema que utiliza menos memoria de los tres rtos compatibles con PSoC 5 en las situaciones y configuraciones anteriormente nombradas.

Si analizamos el uso de memoria, se puede ver que FreeRTOS es el que consume menos en todos los casos, 3305 bytes solo el sistema operativo, 4847 bytes con una sola tarea en ejecución y 6921 bytes con la aplicación del acelerómetro.

Aun así, la diferencia de uso de memoria RAM no es muy grande, entre los sistemas nunca hay una diferencia mayor de 6 kB en las diferentes implementaciones. Además, hay que añadir, que tanto embOS como Micrium reservan la memoria a partir del tamaño de las variables y, en cambio, FreeRTOS aparte de utilizar esta forma es necesario definir el tamaño del Heap que utilizaran las tareas y estructuras del sistema para reservar su parte de RAM para su ejecución.
Entonces se puede concluir que, si se trabaja en una plataforma con limitaciones de memoria, FreeRTOS es el sistema operativo que menos uso de memoria RAM hace, pero hay que tener en cuenta las necesidades de la aplicación para configurarlo correctamente.

7.5 Consumo Energía

En todos los escenarios se utiliza una frecuencia de 24Mhz y un voltaje de 3.3V para la aplicación de la UART como de la Transformada de Fourier y una alimentación de 5.0V para la aplicación del sensor.

7.5.1 Aplicación Transmisión UART

Se ha implementado una aplicación para ver el consumo

de los sistemas operativos al enviar datos por una UART. La aplicación genera dos números enteros aleatorios y los transmite por la UART. Después de enviar los datos el sistema pasa a modo de bajo consumo, sleep, para ver la transición de estados.

7.5.2 Aplicación Sensor Ultrasonido

La aplicación se basa en un proyecto publicado en el foro oficial de Cypress donde un usuario ha compartido un proyecto [19] donde están implementadas las funciones y el diseño hardware que se utiliza para controlar el sensor de ultrasonido, HC-SR04.
La aplicación está formada por una tarea que se encarga de reiniciar el contador que servirá para calcular la distancia, enviar el pulso del sensor y activar el contador. La tarea se queda en espera hasta recibir el pulso de vuelta.
Una vez se detecta el pulso de vuelta se genera una interrupción que almacena en una variable el valor del contador y desbloquea la tarea con la sincronización de un semáforo. Cuando la tarea vuelve a ejecutarse se calcula la distancia entre el sensor y el objeto con el valor del contador y se envía por la UART. Al acabar este proceso el sistema pasa a bajo consumo durante 12 segundos aproximadamente.

7.5.3 Aplicación Transformada Rápida de Fourier

Se ha implementado una tarea que calcula la Transformada rápida de Fourier y envía el resultado por UART. Una vez realiza el cálculo y se han enviado los datos el sistema pasa al modo de bajo consumo durante 16 segundos. Se ha utilizado el código de RosettaCode.org que utiliza el algoritmo recursivo Cooley-Turkey FFT. [20]

	FreeRTOS	Micrium-III	embOS
UART	Máxim: 6,933 mA Promedio: 5,562 mA	Máximo: 14,038 mA Promedio: 13,885 mA	Máximo: 13,299 mA Promedio: 13,169 mA
Sensor	Máximo: 17,203 mA Promedio: 16,486 mA	Máximo: 16,676 mA Promedio: 15,884 mA	Máximo: 16,255 mA Promedio: 15,371 mA
FFT	Máximo: 13,099 mA Promedio: 8,887 mA	Máximo: 12,780 mA Promedio: 12,293 mA	Máximo: 12,654 mA Promedio: 12,239 mA

Fig.3. Tabla comparativa consumos energía.

En el escenario de la aplicación UART, se observa FreeRTOS es el que consume menos y Micrium-RTOS es el que consume más, muy parecido al de embOS. FreeRTOS sí que hace la transición de enviar los datos y quedar a la espera los 500 uS hasta entrar en modo de bajo consumo, por lo que se ve reducido su promedio, en cambio Micrium y embOS se mantiene el consumo hasta entrar en el modo sleep.

En el escenario de la aplicación del sensor de ultrasonido los consumos son muy parecidos, hay una diferencia máxima aproximada de 1mA entre el que menos consume, embOS y el que más, FreeRTOS. Hay que tener en cuenta que el sensor tiene un consumo de 15mA y que se está utilizando un reloj aparte y un contador que aumentan el consumo del sistema.

En el escenario del cálculo de la FFT, se puede observar que el sistema operativo FreeRTOS hace un uso más eficiente de la energía y que tanto Micrium como embOS tienen un consumo parecido y superior FreeRTOS.

En general se puede observar que, de base, embOS y uC/OS-III tienen un consumo más alto que FreeRTOS, pero a la vez se mantiene bastante constante en las diferentes aplicaciones. En cambio, FreeRTOS tiene el consumo base más bajo, pero se puede observar como en el caso de la aplicación del sensor o de la Transformada de Fourier cuando se hace uso de los recursos o se realizan los cálculos el consumo aumenta a diferencia de los otros sistemas operativos.

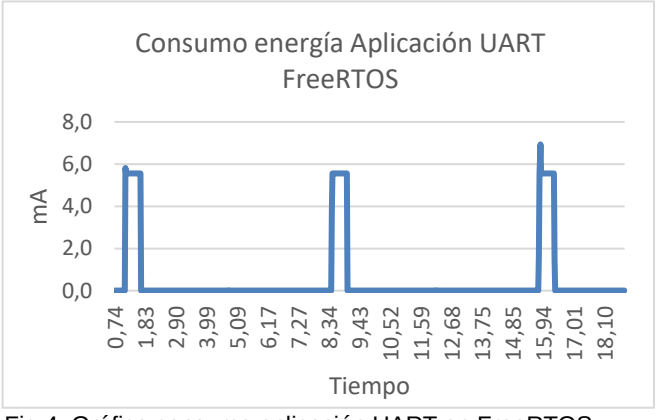

Fig.4. Gráfico consumo aplicación UART en FreeRTOS.

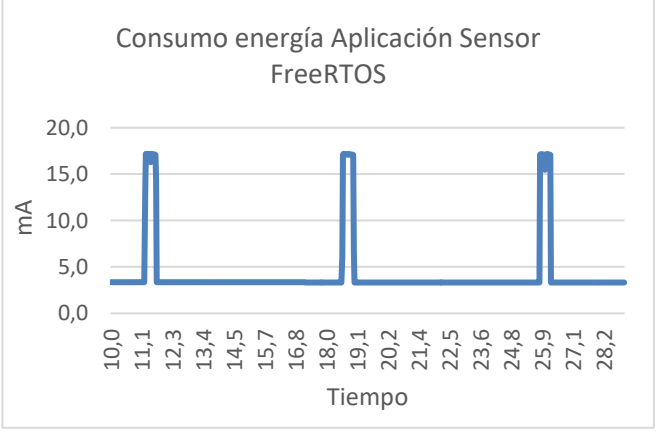

Fig.5. Gráfico consumo aplicación sensor FreeRTOS.

8 Conclusión

Para terminar el proyecto, hay que decir que el objetivo de búsqueda e implementación ha cambiado a lo largo del mismo, donde se han analizado diferentes sistemas operativos en el mercado. Es importante destacar que

hay un gran número de sistemas compatibles con el primer objetivo pensado, los procesadores ARM Cortex-M3, la mayoría comparten funcionalidades y a la vez se diferencian en implementar otras nuevas como pilas de conectividad compatibles con diferentes protocolos y redes, consumo de memoria y modos de gestión de energía. Como se ha comentado en apartados anteriores, el sistema operativo RiotOS es uno de los sistemas objetivos para poder desarrollar una implementación, ya que gracias a su microkernel y arquitectura modular permiten reducir el consumo de memoria y energía y a la vez implementar todas las funcionalidades de un sistema operativo de tiempo real

Adicionalmente, se han analizado diferentes sistemas operativos compatibles con PSoC 5 porque se ha utilizado la placa de desarrollo CY8CKIT-059 que forma parte de la familia PSoC 5 LP. Después de analizar tanto uso de memoria como consumo de energía implementando diferentes aplicaciones para analizar el consumo real; se puede concluir que el sistema operativo FreeRTOS es el sistema operativo más eficiente basado en los resultados obtenidos en las pruebas.
Aparte de los resultados también destaca por ser un proyecto de código abierto, con una gran comunidad y usuarios.

Posibles mejoras u objetivos futuros: trabajar más y profundizar en la implementación de las aplicaciones para obtener una mejor eficiencia tanto en la ejecución como en el uso de los recursos del sistema. También sería interesante implementar aplicaciones ya desarrolladas para sistemas operativos en tiempo real que utilicen sensores profesionales o más complejos y a la vez implementen conectividad entre el punto donde se encuentre el sensor y un servidor donde se almacenan estos datos para observar tanto el consumo de energía y memoria para obtener mejores referencias y precisión en los resultados.

AGRADECIMIENTOS

Agradecer en primer lugar a Raul Aragonés Ortiz y Roger Malet Munté por todo el soporte, apoyo y todos los consejos que me han proporcionado durante la realización del proyecto.

BIBLIOGRAFIA

[1] V. Nandana, A. Jithendran, R. Shreelekshmi. "Survey on RTOS: Evolution, Types and Current Research". [Online]. Disponible en: https://pdfs.semanticscholar.org/9944/32344e60ea719c63ff239015698460cae02f.pdf. [Accedido: 19-Enero.-2020].

[2] National Instruments. "What is a Real-Time Operating System (RTOS)?". [Online]. Disponible en: https://www.ni.com/es-es/innovations/white-papers/07/what-is-a-real-time-operating-system--rtos--.html. [Accedido: 19-Enero.-2020].

[3] Douglass Locke, Software Architecture for Hard Real-Time Applications Cyclic Executives vs. Fixed Priority Executives". [Online]. Disponible en: https://www.douglocke.com /Downloads/CylicvPriorityExecs.pdf. [Accedido: 19-Enero-2020].

[4] Advantages of using an RTOS - Keil. [Online]. Disponible en: http://www.keil.com/rl-arm/rtx_rtosadv.asp. [Accedido: 19-Enero-2020].

[5] N. Lethaby. OS Product Manager Texas Instruments Incorporated. "Why Use a Real-Time Operating System in MCU Applications". [Online]. Disponible en: http://www.ti.com/lit/spry238. [Accedido: 19-Enero-2020].

[6] D. Moore. Director of Engineering. "Why Use an RTOS?". [Online]. Disponible en: http://www.smxrtos.com/articles/whyrtos.htm. [Accedido: 19-Enero.-2020].

[7] ChibiOS Technical Wiki. [Online]. Disponible en: http://wiki.chibios.org/dokuwiki/doku.php?id=chibios:documents. [Accedido: 19-Enero-2020].

[8] ContikiOS. [Online]. Disponible en: https://en.wikipedia.org/wiki/Contiki.

[9] ContikiOS Wiki: Internals. [Online]. Disponible en: https://github.com/contiki-os/contiki/wiki. [Accedido: 19-Enero-2020].

[10] RIOT Documentation. [Online]. Disponible en: https://doc.riot-os.org/. [Accedido: 19-Enero-2020].

[11] R. Barry. "Mastering the FreeRTOS Real Time Kernel: A Hands-On Tutorial Guide". [pdf]. Disponible en: https://www.freertos.org/wp-content/uploads/2018/07/161204_Mastering_the_FreeRTOS_Real_Time_Kernel-A_Hands-On_Tutorial_Guide.pdf. [Accedido: 25-Enero-2020].

[12] Zephyr Project Documentation. [Online]. Disponible en: https://docs.zephyrproject.org/latest/index.html. [Accedido: 25-Enero-2020].

[13] Nathan Willis. "Why Zephyr? [LWN.net]". [Online]. Disponible en: https://lwn.net/Articles/682723/. [Accedido: 25-Enero-2020].

[14] "µC/OS-III Users Guide - Micrium Documentation". [pdf]. Disponible en: https://doc.micrium.com/download/attachments/10753180/600-uCOS-III-UsersGuide-004.pdf?version=1&modificationDate=1380828338000&api=v2. [Accedido: 25-Enero-202].

[15] "[Ebook]. embOS & embOS-MPU Real-Time Operating System User Guide & Reference Manual". [pdf]. Disponible en: https://www.segger.com/downloads/embos/UM01001. [Accedido: 25-Enero-2020].

[16] PSoC 5LP: CY8C58LP Family Datasheet Programmable System-on-Chip. [pdf]. Disponible en: https://www.cypress.com/file/45906/download. [Accedido: 19-Enero-2020].

[17] Documentación Oficial de FreeRTOS. "Developer Docs: FreeRTOSConfig.h". [Online]. Disponible en: https://www.freertos.org/a00110.html#configAPPLICATION_ALLOCATED_HEAP. [Accedido: 25-Enero-2020].

[18] Documentación Oficial de FreeRTOS. "Developer Docs: Heap Memory Management". [Online]. Disponible en: https://www.freertos.org/a00111.html. [Accedido: 25-Enero-2020].

[19] "HC-SR04 test program for CY8CKIT-059". [Online]. Disponible en: https://community.cypress.com/message/201051#201051. [Accedido: 26-Enero-2020].

[20] "Fast Fourier trasnform". [Online]. Disponible en: https://rosettacode.org/wiki/Fast_Fourier_transform#C. [Accedido: 25-Enero-2020].

APÉNDICE

A1. ÍNDICE FIGURAS

A2. DIAGRAMA DE GANTT DE LA PLANIFICACIÓN