Implementación en FreeRTOS del software de control de un Electrocardiógrafo

C A Centeno[1] **, J A Voos**[1]**, F J Cagnolo**[1] **E A Gonzalez**[1] **e C E Olmos**[1]
Grupo de Ingeniería Clínica, Universidad Tecnológica Nacional, Regional Córdoba, Córdoba, Argentina

E-mail: ccenteno@gmail.com

Abstract. In this work we evaluate the feasibility to use a Real Time Operating System (FreeRTOS) on the CPU from electrocardiograph. This device is based on a microcontroller platform that uses a PIC 18F4620 where the medical device functions are performed by the microcontroller. The main objective for this research is check if the FREERTOS can be embedded on this microcontroller, including its capabilities and resources. This work is based on the results from other RTOS kernel running on the same platform. In this case we evaluated RAM and ROM necessary to execute all device and kernel functions like semaphore and queues for tasks synchronization. The processes or tasks to be evaluated were already checked in previous experiences on other RTOS kernel. The final result is an electrocardiograph based in a FreeRTOS Kernel.

1. Introducción

En el Grupo de Ingeniería Clínica (GIC), perteneciente al Departamento Ingeniería Electrónica de la Universidad Tecnológica Nacional Facultad Regional Córdoba, se diseño un Electrocardiógrafo (ECG) utilizando microcontroladores (uC) de la familia Microchip [1][2]. Sabemos que la integración actual de circuitos electrónicos en dispositivos de menor tamaño y mayores prestaciones, en configuraciones de 8 hasta 32 bits con periféricos integrados, permiten crear dispositivos o sistemas embebidos con distintos grados de integración y funcionalidades. En virtud de ello inicialmente se optó por el 16F877A y luego por la migración a la familia 18, en donde primero fue el 18F4550 y por último el 18F4620 [3][4]. Sobre este uC se diseñó el control digital de las etapas de acondicionamiento, adquisición y conversión, con el fin de presentar en un display de cristal líquido inteligente (LCD) la señal adquirida junto a los parámetros de configuración del equipo. Adicionalmente se incluyó el envío de los datos digitalizados a una computadora personal (PC) para su almacenamiento y posterior postprocesamiento. El cambio de familia permitió la posibilidad de evaluar la implementación de todas las funciones de control asociadas al uC mediante el uso de un sistema operativo de tiempo real (RTOS) [5].

En virtud de la existencia de una opción de software libre, se decidió evaluar el uso de FreeRTOS, con el propósito de utilizar un kernel en la capa de control para aprovechar los tiempos en los que el CPU no ejecuta operaciones útiles al sistema. Para poder hacer uso de FreeRTOS es necesario disponer de un uC capaz de manejar un STACK de memoria lo suficientemente grande; ya que la ejecución basada en tareas implica que al realizar un cambio de contexto es necesario almacenar los valores de los registros de estado del uC; los que se preservan en memoria STACK.

En forma complementaria, se encontró que en la actualidad las empresas de software que desarrollan sistemas RTOS trabajan en versiones aptas para el uso médico, las cuales están orientadas a satisfacer los requerimientos establecidos para cumplir con las normas FDA510K. Se pueden mencionar los productos SAFERTOS y uCOS-II como alternativas para cumplimentar con los requerimientos de los estándares citados anteriormente. Se sabe que en el mercado existen distintas implementaciones de RTOS, MQx de Freescale[6], AVIX-RT para Microchip[7], TI-RTOS de Texas Instruments[8], FreeRTOS[9] y Nucleus RTOS[10] de Mentor Graphics, entre otros.
En el caso de Nucleus RTOS, su Kernel se utiliza en dispositivos médicos; entre los cuales podemos mencionar Desfibrilador Zoll Medical, Máquina de Anestesia Datex Ohmeda y Monitor Multiparamétrico Mindray.

2. Objetivos

El objetivo general del trabajo fue evaluar el desempeño de FreeRTOS en el electrocardiógrafo diseñado en el GIC para ser usado como plataforma de desarrollo e investigación y también por parte de los estudiantes de la carrera de grado.

Los Objetivos Específicos planteados fueron:

- Determinar la factibilidad de transformación de la topología Súper Loop a multitareas basado en FreeRTOS.
- Determinar la cantidad de RAM y ROM necesarios.
- Evaluar la posibilidad de asociar nuevas funcionalidades al ECG.
- Determinar las diferencias entre el uso de software libre o licenciado.

3. Materiales y Métodos

3.1. Evaluación del sistema

El diseño inicial del ECG estaba orientado al uso de súper loop como metodología de desarrollo del software de control. En virtud de ello y para poder realizar la migración a una plataforma RTOS, fue necesario efectuar un análisis pormenorizado de la interacción de cada etapa con el CPU, ya sean analógicas o digitales, para definir cuáles serían los módulos de software resultantes en el nuevo diseño basado en tareas.
En la forma original cada bloque de código se ejecuta uno a continuación del otro y la administración del tiempo se realiza con uno de los timers integrado y su correspondiente interrupción. Durante un ciclo del Loop general se borra la pantalla, se filtra el dato adquirido, se envía vía canal serie, se escanea el teclado y por último se dibuja el punto correspondiente en la pantalla de cristal líquido. Lo más complejo en el diseño del súper loop fue el ajuste de los tiempos de cada bloque de código para poder respetar la frecuencia de muestreo seleccionada para el diseño.

3.2. Etapas Electrónicas

El hardware del sistema está compuesto de diversas etapas, las que se enumeran a continuación: teclado, multiplexor de entrada, etapas de acondicionamiento y filtrado, amplificador de ganancia programable vía protocolo serie sincrónico (SPI), tensión de offset programable SPI, conversor analógico digital SPI, decodificador selector de etapas, display de cristal liquido inteligente e interfaz de comunicaciones serie. Se presenta en la figura 1, un esquema de distribución de las distintas etapas del ECG indicadas sobre una imagen del circuito impreso definitivo (PCB).

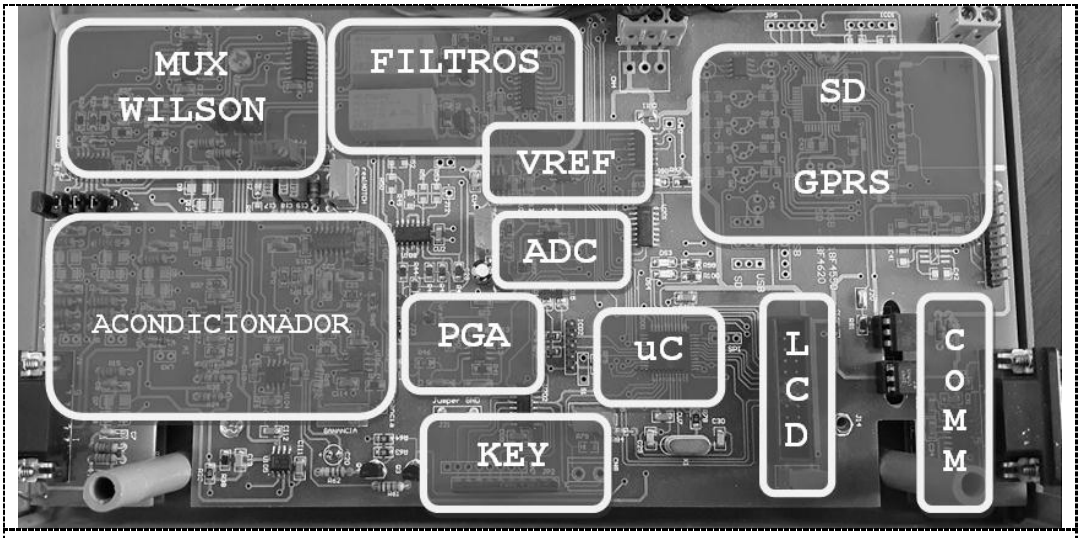

Figure 1. Distribución de las distintas etapas sobre el PCB definitivo**.**

Estas etapas interactúan con el uC y para ello emplean líneas de entrada salida, o periféricos específicos, tales como EUART o SPI. En la tabla 1 se presenta un resumen de etapas y sus respectivas conexiones al uC.

Tabla 1: Recursos de Hardware necesarios

Etapa	Conexión
Teclado Matricial	3 líneas I/O
Multiplexor de Entrada	4 líneas I/O
Filtrado	1 línea I/O
Amplificador de Ganancia Programable	SPI integrado
Tensión de Offset Programable	SPI integrado
Conversor Analógico Digital	SPI integrado
Decodificador selector de módulos	3 líneas de I/O
Display de Cristal Líquido	10 líneas de I/O
Interfaz de Comunicación	Interfaz EUART

Cuando la plataforma estaba basada en el 18F4550 el periférico SPI era emulado, ya que se deseaba utilizar el módulo USB como medio de conexión al PC, y éste compartía las mismas líneas de I/O con la interfaz antes mencionada. Con el cambio al 18F4620 se decidió utilizar el SPI integrado, y la comunicación a la PC mediante el uso de la UART y el protocolo RS232.
Cuando observamos la interacción entre las distintas etapas y/o periféricos vemos que el canal SPI debe ser utilizado en forma sincronizada ya que es compartido por distintos bloques de hardware. Es importante tener en cuenta este factor, ya que al efectuar el switch entre tareas, es posible que la interfaz SPI no esté libre, y debido a esto se pueda generar algún tipo de dato incongruente.
Se debe recordar que en el súper loop cada bloque de código se ejecuta uno a continuación del otro, con lo cual no existe la posibilidad de la superposición temporal en el uso de los periféricos compartidos por más de uno de ellos, ya que cuando el ADC usa el SPI, el módulo de tensión programable espera su turno.

Es necesario recordar que la conexión del ECG a una PC requiere de aislación galvánica, por ello se emplean optoacopladores en el canal UART. Por otro lado se debe indicar que la fuente de alimentación es del tipo switching apta para equipos médicos. Estas medidas se emplearon para poder satisfacer las condiciones de seguridad eléctrica basadas en las normas IEC60601.[11][12][13]

3.3. Multitarea

Cuando se usa ~~de~~ un RTOS se busca dividir el súper bucle en tareas independientes con el objetivo inicial de optimizar el uso del CPU. Se debe mencionar que al usar tareas, es prioritario transformar las rutinas de consumo de tiempo en servicios del kernel, recordando que en estas rutinas se utilizan bucles de repetición ***for*** o ***while*** con la sentencia NOP() en su bloque de operación o con estructuras como las que se visualizan en la figura 2.

```
#include    "delay.h"

void
DelayMs(unsigned char cnt)
{
#if XTAL_FREQ <= 2MHZ
    do {
        DelayUs(996);
    } while(--cnt);
#endif

#if    XTAL_FREQ > 2MHZ
    unsigned char   i;
    do {
        i = 4;
        do {
            DelayUs(250);
        } while(--i);
    } while(--cnt);
#endif
}
```

file length : 393 lines : 29 Ln : 24

```
#ifndef XTAL_FREQ
#define XTAL_FREQ   20MHZ       /* Crystal frequency in MHz */
#endif

#define MHZ *1000L          /* number of kHz in a MHz */
#define KHZ *1          /* number of kHz in a kHz */

#if XTAL_FREQ >= 12MHZ

#define DelayUs(x)  { unsigned char _dcnt; \
                      _dcnt = (x)*((XTAL_FREQ)/(12MHZ)); \
                      while(--_dcnt != 0) \
                          continue; }
#else

#define DelayUs(x)  { unsigned char _dcnt; \
                      _dcnt = (x)/((12MHZ)/(XTAL_FREQ))|1; \
                      while(--_dcnt != 0) \
                          continue; }
#endif

extern void DelayMs(unsigned char);
```

Figura 2. Delay en topología Súper Loop [14]

Se pueden emplear servicios del kernel para realizar el switch entre procesos, y transformar el tiempo de CPU ocioso en tiempo de procesamiento. Estos servicios son funciones propias del RTOS y provocan el cambio de una tarea del estado Activo al estado Suspendido.

A su vez es posible que una tarea se ejecute a continuación de otra y siempre en ese orden si se hace uso de los denominados eventos, que son elementos del kernel cuya finalidad es sincronizar.

Para poder efectuar el switch entre tareas en forma ordenada, se asigna a cada una de éstas una prioridad, y de esta manera el administrador de cambios, o ***scheduler***, sabrá en cada instante cuál debe ejecutarse. Con esta funcionalidad es posible que siempre la tarea de mayor prioridad se ejecute en el tiempo previsto al momento del diseño del software de control.

El uso de un RTOS puede provocar que dos tareas intenten usar el mismo periférico, lo que llevaría a una inestabilidad del sistema, con lo cual es deseable en primera instancia el análisis profundo de las secuencias que se deben completar y luego utilizar los eventos mencionados anteriormente, tales como semáforos, mailboxes, colas (queues) para la sincronización en el uso del periférico en cuestión.

3.4. FreeRTOS

Es un RTOS soportado por diversas arquitecturas y/o familias de microcontroladores y microprocesadores. Es robusto, y como característica sobresaliente se debe mencionar que no requiere de licencias para su uso comercial. Se puede descargar desde el sitio web del producto [9] y dispone de distintas opciones en función de la necesidad del sistema a desarrollar. Como recursos posee la opción de trabajar en modo cooperativo o en modo preemptivo, incluye semáforos mutuamente excluyentes, binarios y con contador, además de una API fácil de usar.

Utilizando el modo preemptivo, el scheduler pone en modo ejecución (Ready) la tarea de más alta prioridad cuando ésta lo requiere, mientras que en el modo cooperativo se espera que la tarea termine su ejecución, para que el scheduler pueda realizar el cambio de contexto. En este punto si por algún motivo la tarea activa queda en un bucle sin control, el sistema colapsará, mientras que cuando se usa el modo preemptivo, siempre se ejecutarían las tareas que posean mayor prioridad. Por lo tanto es requisito primordial del diseño del software de control, la correcta asignación de prioridades y el uso de servicios del Kernel en cada tarea para que puedan llevarse a cabo los correspondientes cambios de contexto. En la figura 3 se presentan los estados posibles que puede tomar cada proceso.

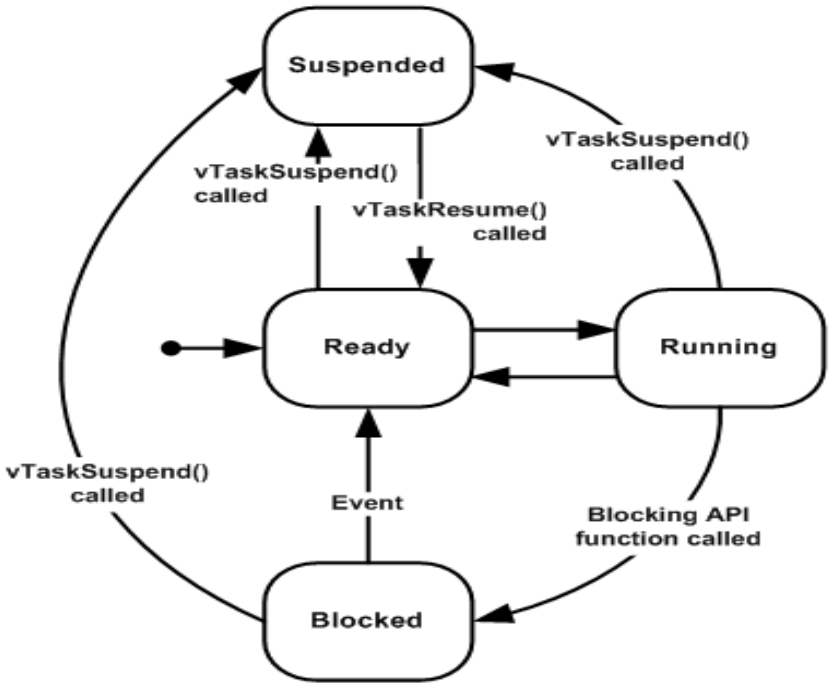

Figura 3. Estados definidos para el sistema. [15]

3.5. Uso de Recursos

Desde el punto de vista del RTOS, para la implementación del cambio entre tareas es necesario el uso de un scheduler y de estructuras de control específicas. Por cada tarea independiente, el sistema emplea un bloque de control de tarea (TCB) y para la sincronización se usan semáforos, mailboxes y/o colas, que requieren a su vez de bloques específicos de control (ECB).

Una capacidad muy útil es la posibilidad de agregar un timeout a estos elementos, con lo cual se puede determinar el fallo de la sincronización entre las tareas que hacen uso de un recurso compartido.

En la tabla 2 se presentan los recursos de memoria que FreeRTOS requiere, esta información resumida surge del proceso de debugging y es obtenida de la interfaz de desarrollo (IDE).

Tabla 2: Recursos de memoria necesarios

Funcionalidad	ROM	RAM
Tareas - Creación y Uso	37 bytes	0 bytes
Semáforos - Creación y uso	38 bytes	6 bytes
Queues - Creación y uso	30 bytes	6 bytes

Además cada tarea requiere memoria STACK para almacenar las condiciones de contexto al momento del cambio entre ellas. Para ello FreeRTOS dispone de una herramienta que permite la verificación de overflow en el STACK, con lo cual es posible minimizar los efectos adversos de esta instancia, es decir evitar que el sistema se torne inestable cuando se supera los límites de la misma asignada a una tarea determinada.

4. **Resultados**

Al igual que en la evaluación de factibilidad basado en la otra plataforma RTOS [5], se definieron cuatro tareas, las que se presentan interactuando entre sí en la figura 4.

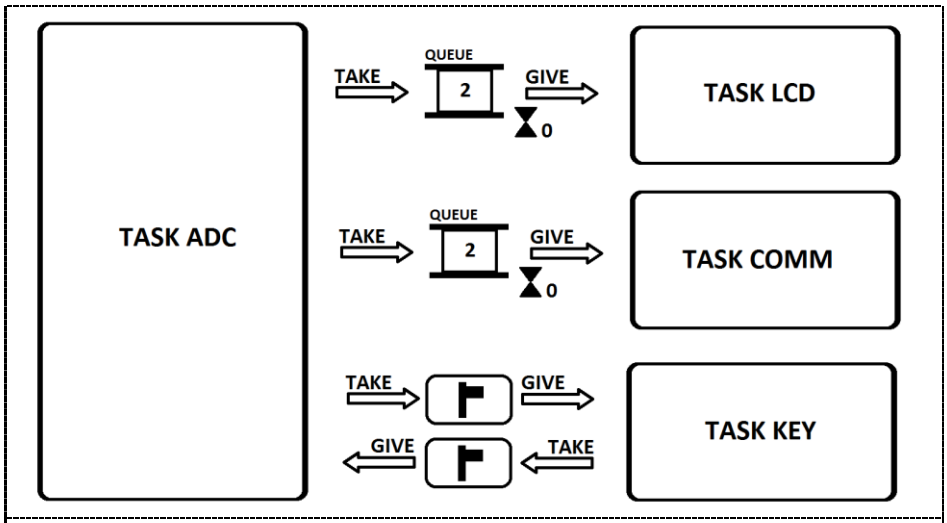

Figura 4. Interacción entre tareas

4.1. Tarea vTaskADC

En este proceso el conversor analógico digital (ADC) de 12 bits, ADS1286, interactúa con el uC empleando el protocolo SPI. Se ejecuta con una frecuencia de muestreo de 500Hz, y el resultado obtenido es enviado usando colas a los procesos *vTaskLCD* y *vTaskCOM*.

4.2. Tarea vTaskKEY

El teclado disponible para las funciones de configuración del ECG emplea un codificador discreto de 10 a 1 líneas. En virtud de que el usuario puede modificar la ganancia, la velocidad de barrido, la derivación y el uso o no de filtros analógicos; es necesario el uso del canal SPI para estos ajustes en las etapas de hardware correspondientes. Por tal motivo, es muy importante que exista una correcta sincronización con la tarea *vTaskADC,* donde la solución óptima es el empleo de dos semáforos binarios.

4.3. Tarea vTaskLCD

El biopotencial obtenido luego de la digitalización es presentado en un LCD gráfico. Esta tarea debe ejecutarse una vez que se obtienen los datos correspondientes, por lo que es necesario que esté sincronizada con la tarea *vTaskADC*. Se usa para ello una cola como elemento de transferencia de información y sincronización temporal.

4.4. Tarea vTaskCOM

Los datos digitalizados son enviados a una PC para su presentación, almacenamiento y/o posterior postprocesamiento. Por lo tanto se requiere una tarea que disponga del valor convertido y lo envíe usando la USART disponible. El dato convertido es compartido mediante el uso de una cola de datos.

En la figura 5 se muestra el diagrama temporal resultante, en donde se presentan los eventos o servicios del kernel utilizados y se indica cuando se produce el cambio de contexto (CS). Se debe considerar siempre al momento del diseño que cuando hay cambio de contexto existe consumo de tiempo de procesamiento razón por la cual estamos ante una de las condiciones que podrían limitar el uso de un RTOS.

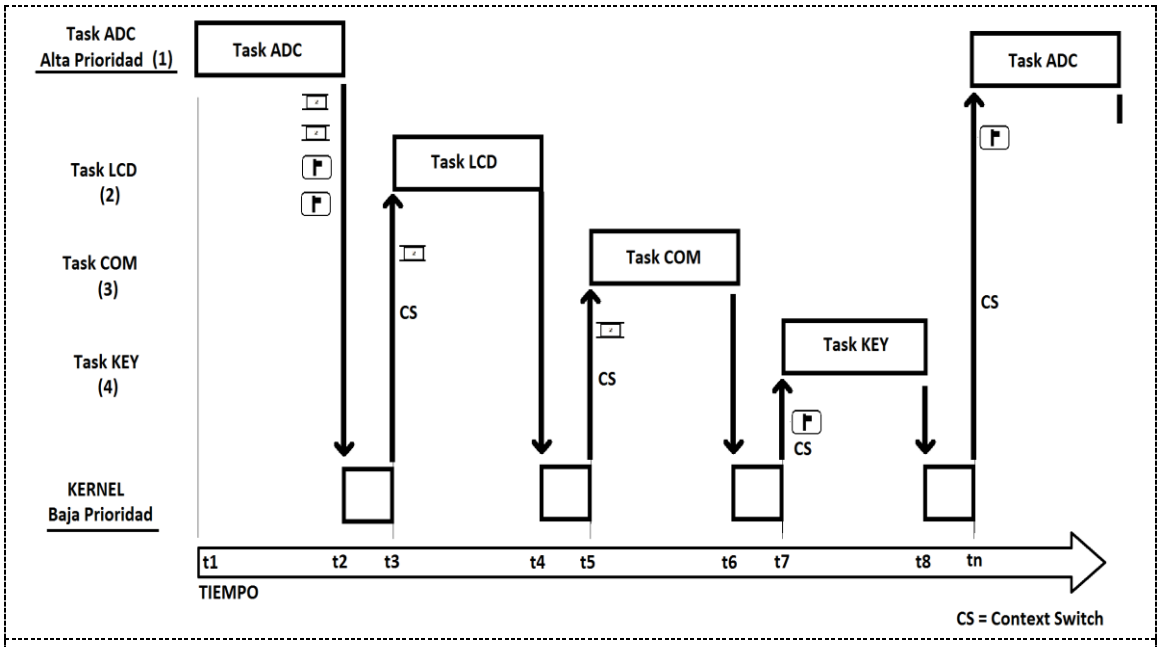

Figura 5. Diagrama temporal resultante

Debido a que la configuración electrónica del dispositivo hace uso de un recurso compartido como ser el periférico SPI entre diversos módulos, el ADC, la tensión de referencia, y el amplificador de ganancia programable, fue necesario la evaluación e implementación en forma precisa de la sincronización de los mismos para evitar la superposición sobre el recurso y las eventuales inestabilidades del sistema, para ello el uso de los semáforos y colas es muy importante.

En la figura 6 se presenta el código resultante de dos tareas en donde se indica cuando se efectúa la sincronización. Adicionalmente se presenta el concepto de tarea, definido como un bucle infinito, en el cual se ejecutan funciones o procedimientos, y en los que además se hace uso de los servicios (vTaskDelayUntil, vTaskDelay) y eventos del Kernel (xQueueSend, xSemaphoreGive o xSemaphoreTake).

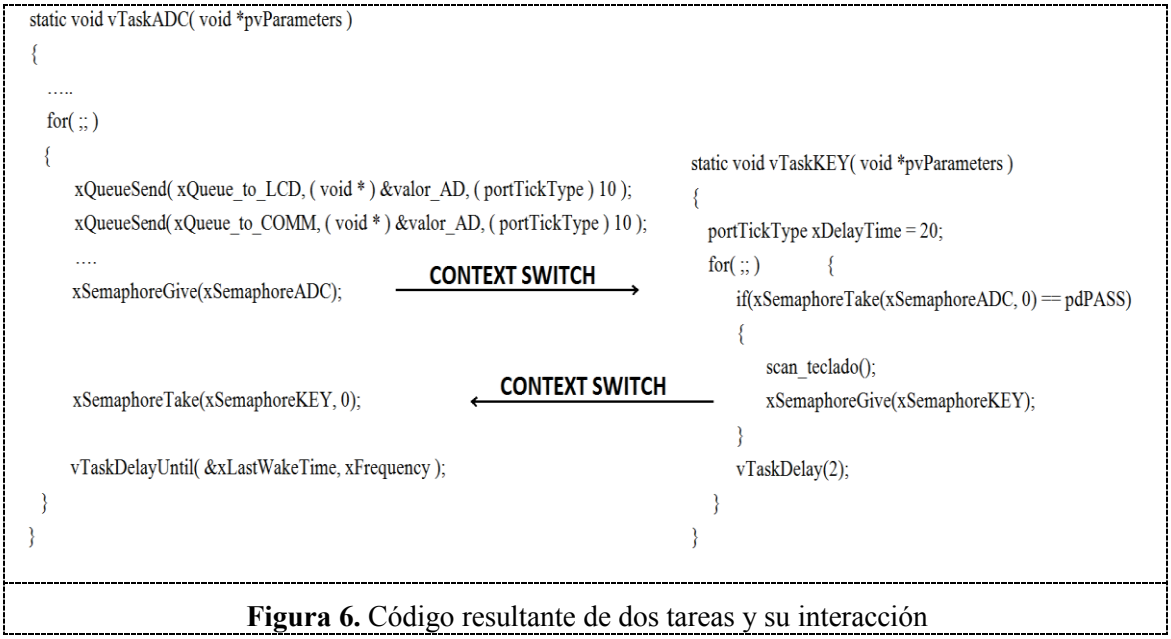

Figura 6. Código resultante de dos tareas y su interacción

Se aprecia el uso de las colas para sincronizar el ADC con el LCD y la tarea COMM, mientras que luego del envío de la información adquirida se utiliza un semáforo binario para la sincronización con el teclado, y la espera del semáforo activado luego de que el proceso de escaneo finaliza.

Ya que se utiliza el mismo canal electrónico en el hardware, SPI, es necesario que cuando una conversión se encuentre activa no se intente escanear el teclado. Si bien el tiempo empleado por el ADC durante la conversión es mínimo, esta modalidad de diseño nos garantiza poder utilizar el SPI las cuatro veces requeridas (uno por byte) para que el proceso de conversión sea completado. Es por ello que luego de recibir el semáforo del ADC y realizar el escaneo de las teclas, el proceso *vTaskKEY* le avisa que ha finalizado a *vTaskADC*.

Una de las limitaciones resultantes del software con esta distribución de tareas es que cuando se está ante la presencia de una presión de tecla con la intención de realizar un cambio de derivación o ajustar la ganancia; se utilizará la interfaz SPI. Debido a ello, no es posible realizar una conversión por parte del ADC.

Finalmente en la tabla 3 se muestra un detalle de los recursos necesarios para la implementación del software de control del ECG en el modo Súper Loop, FreeRTOS (libre) y uCOS-II (comercial).

Tabla 3: Recursos totales necesarios según sea el uC empleado

	RAM		ROM	
	Bytes Usados	**% de uso RAM total**	**Bytes Usados**	**% de uso ROM total**
Super Loop- 18F4620	614	15,47%	4268	13,02%
uCOS-II - 18F4620	1882	47,43%	12721	38,82%
FreeRTOS - 18F4620	1101	27,75%	11544	35,23%

Se puede resumir que la utilización de una plataforma RTOS requiere para esta configuración de hardware particular, es decir el ECG, el empleo de tareas independientes adicionales a los que necesita dicho sistema embebido para funcionar en la topología súper loop.
Esto recae en la necesidad de disponer de una cantidad de RAM considerable y que la misma pueda ser configurada como STACK.

Conclusiones

La implementación del software de control del electrocardiógrafo portátil empleando una plataforma FreeRTOS puede ser analizada desde distintos puntos de vista. Se puede considerar la factibilidad de implementación según los recursos de RAM y ROM del uC seleccionado, o desde el punto de vista del costo de las licencias del RTOS.

En caso de contar con un hardware existente, es preciso analizar el costo de la licencia comercial de RTOS (uCOS-II) en comparación con adquirir y/o rediseñar un nuevo hardware que nos permita utilizar FreeRTOS.

En este punto es deseable recalcar que los uC utilizados permitieron el cambio del CPU, sin ajustes del hardware gracias a su compatibilidad pin a pin. Por ello se puede completar el análisis del uso del mismo software de control basado en FreeRTOS en dos uC, 18F4550 y 18F4620.

Del análisis de requerimientos se observa que la implementación del software de control basado en RTOS requiere una importante cantidad de recursos RAM y ROM para dar soporte a las funciones del Kernel (Tabla 3). Respecto a la opción de Super Loop, se requiere aproximadamente 1,8 veces más de RAM y 2,7 veces más de ROM.

Si se usara un uC 18F4550 [5], vemos que la ROM alcanza un 70,46% del total, lo cual indicaría que la factibilidad de adición de nuevas funcionalidades al ECG se vería limitada. Como se expuso anteriormente, con el solo cambio de uC a 18F4620, este valor porcentual baja al 35,23% debido a que se dispone de mayor cantidad de ROM.

De los números obtenidos en cuanto a RAM y ROM se puede establecer que no existen diferencias marcadas, cuando se usa el 18F4620, por lo que al momento de elegir FreeRTOS o uCOS-II éste no sería un factor de decisión.

Sin embargo el costo de una licencia de software puede determinar el camino a seguir. En base a nuestra experiencia podemos decir que la facilidad de uso de las herramientas de software y luego la asistencia por parte de las empresas o grupos de desarrollo también pueden influir en optar por una u otra.

Los paquetes comerciales cuentan con áreas de soporte técnico que brindan asistencia específica en forma directa, mientras que en algunos casos de herramientas libres hay que dedicar tiempo en la búsqueda de las respuestas en foros dedicados, administrados por la comunidad que desarrolla y/o mantiene dicha herramienta.

El uso de FreeRTOS como Kernel de control de la plataforma electrocardiógrafo se emplea como ejemplo en la Materia Software en Tiempo Real, la cual integra la currícula de la carrera Ingeniería en Electrónica de la Universidad Tecnológica Nacional Regional Córdoba.

Como continuación de esta línea de investigación, se está evaluando la posibilidad de migrar el hardware de control a una plataforma basado en microcontroladores Freescale, como ser el

MK64FN1M0VLL12 que es parte de la placa FRDM-K64F, en donde además de disponer de mayores recursos de RAM, ROM y periféricos, es posible utilizar los RTOS ensayados, de los cuales ya se tiene la experiencia en la implementación.

Referencias

[1] Centeno C y Trento I 2003, *Monitor de Pulsos Cardíacos con Interfaz a PC,* En: Anales del XIV Congreso Argentina de Bioingeniería SABI 2003, Córdoba, 22-24 de Octubre.

[2] Centeno C 2007, *Procesamiento Digital sobre Monitor Cardíaco* CLAIB 2007 IFMBE Proceedings, Isla Margarita, 24-28 de Setiembre. p. 497-500.

[3] Centeno C 2010, *Sistema para el análisis de la Morfología de la señal de ECG de ratones de Laboratorio infectados con Mal de Chagas* Anales del XXII Congresso Brasileiro de Engenharia Biomédica, Tiradentes- Minas Gerais, 21-25 de Noviembre.

[4] Centeno C 2007, *Sistema de transmisión de datos médicos para centros de salud de zonas alejadas* Anales del XVI Congreso Argentino de Bioingeniería, San Juan, 26-28 de Setiembre.

[5] Centeno C, Voos J, Riva G, Zerbini C y Gonzalez E 2011, *Electrocardiógrafo portátil basado en RTOS* Anales del XVIII Congreso Argentino de Bioingeniería, Mar del Plata, 28-30 de Setiembre.

[6] MQX-Real Time Operating System(RTOS). http://www.freescale.com/

[7] AVIX-RT, http://www.avix-rt.com/

[8] TI-RTOS, http://www.ti.com/tool/ti-rtos

[9] FreeRTOS, http://www.freertos.org

[10] Nucleus RTOS, http://www.mentor.com/embedded-software/industries/medical-devices

[11] Rubio D, Murad C, Ponce S, Alvarez Abril A, Terrón A, Vicencio D y Fascioli E 2007, *Acreditación de Laboratorios de Ensayos de Equipamiento Electromédico* Anales del XVI Congreso Argentino de Bioingeniería, San Juan, 25-28 de Setiembre.

[12] Rodriguez C , Wevar Oller C, Bruni R, Vanella O y Taborda R 2009, *Sistema de evaluación de sobrecarga de transformadores de alimentación para aparatos electromédicos* Anales del XVII Congreso Argentino de Bioingeniería, Rosario, 14 – 16 de Octubre.

[13] IRAM 4220-2-17: 1992, *Aparatos para Electromedicina Electrocardiógrafos. Exigencias particulares de seguridad* Instituto Argentino de Racionalización de Materiales.

[14] HT-PIC, http://www.htsoft.com/

[15] FreeRTOS Task States, www.freertos.org. http://www.freertos.org/RTOS-task-states.html