Arquitectura de software ejecutivo en tiempo real multitarea para sistemas embebidos basada en máquinas de estados finitos

Gabriel Rivas
Universidad Tecnológica de Panamá, Panamá, Panamá, gabriel.rivas@utp.ac.pa

RESUMEN

En la actualidad los avances en la electrónica digital han hecho que sea factible colocar sistemas embebidos en una amplia gama de dispositivos y máquinas. Por lo general cuando hablamos de sistemas embebidos nos referimos a sistemas computacionales que deben realizar sus tareas y funciones en tiempo real. Comúnmente los sistemas embebidos son plataformas de hardware fijas que permiten poco o ningún cambio en su arquitectura, es por eso que a través de software diseñado eficientemente se puede lograr un buen desempeño del sistema sin realizar cambios en la plataforma de hardware. Esto representa un gran reto para los diseñadores de software de sistemas embebidos quienes tienen que hacer uso de los recursos de hardware de la manera más eficiente para cumplir con los requerimientos establecidos. En éste trabajo se presenta una estructura de diseño que facilita la implementación de éste tipo de software.

Palabras Claves: Sistemas embebidos, software en tiempo real, máquinas de estados finitos, software multitarea.

1. INTRODUCCIÓN

En el campo de los sistemas embebidos muchas veces es imposible o poco eficiente utilizar sistemas operativos complejos, aun sistemas operativos en tiempo real, debido a que no cumplen con los requerimientos del sistema y se requiere de la utilización de arquitecturas de software más sencillas y eficientes para la administración de tareas.

La metodología de multitarea por medio de máquinas de estados finitos ya es muy conocida y se ha utilizado para desarrollar sistemas de control digital desde los primeros proyectos de computación masivos (un ejemplo es el sistema de navegación inercial del Apollo 11) sin embargo en la actualidad ésta práctica se ha confinado exclusivamente a plataformas de sistemas embebidos y solo los ingenieros de la vieja guardia tienen la experiencia práctica para llevar a cabo éste tipo de sistemas lo que hace más difícil encontrar personal para realizar este tipo de diseño.

El objetivo de éste trabajo es plantear una arquitectura de software reutilizable con todos los componentes y características básicas que faciliten el acceso y control de los recursos de hardware para producir sistemas en tiempo real, multitarea y de gran confiabilidad.

2. SISTEMAS EMBEBIDOS

2.1 DISEÑO DE SISTEMAS EMBEBIDOS

Al desarrollar un proyecto con sistemas embebidos, básicamente se ejecutan las siguientes tareas:

1. Toma de requerimientos
2. Co-diseño de arquitecturas de hardware y software
3. Implementación del hardware y software
4. Depuración y pruebas

5. Entrega

Basándonos en los requerimientos seleccionamos la plataforma de hardware a utilizar, y los componentes asociados al sistema (interfaces de comunicación, pantallas LCD, conectores, etc) y se diseña el circuito en base a los niveles de voltaje e interfaces entre estos componentes.

Al igual que cuando diseñamos hardware al final obtenemos un circuito esquemático y circuito impreso para poder entonces implementar el mismo físicamente, cuando diseñamos software también debemos tener algún tipo de diagrama esquemático que nos indique el flujo y operación del sistema, y de esta manera poder implementar el firmware en el lenguaje de programación de una manera más limpia y consistente. En la practica el desarrollo de firmware puede tomar gran parte del tiempo de desarrollo de un proyecto (20% - 50%) y si no se realiza de manera ordenada el periodo de depuración y pruebas puede sobrepasar el tiempo total del proyecto estimado inicialmente, es por eso que se requiere de herramientas que ayuden a agilizar el desarrollo del software permitiendo a los programadores concentrarse en el desarrollo de la aplicación basándose en una plantilla ya existente que asegure un nivel de orden y calidad mínimo.

3. EJECUTIVO EN TIEMPO REAL MULTITAREA COOPERATIVO

Los ejecutivos en tiempo real multitarea son el conjunto de módulos, librerías, interfaces, estructuras de datos de software que en conjunto permiten el acceso y control de los recursos de hardware entre varias tareas. El ejecutivo en tiempo real toma toda la responsabilidad de administrar las tareas, haciendo el trabajo del programador de aplicaciones mucho más simple (Wiliams, R. 2006). En términos de software, una tarea es una función que ejecuta una serie de acciones específicas.

En este tipo de ejecutivo las tareas se colocan en un orden pre-establecido, cada una se ejecuta en un periodo de tiempo y cede el control al administrador de tareas de modo cooperativo (Stoiner-Gibson,D. 2010).

Las tareas se ejecutan en una secuencia definida al momento de diseño dependiendo de las prioridades de cada una a un intervalo de tiempo controlado por una interrupción de un temporizador de hardware.

Si el tiempo que toma ejecutar todas las tareas de la lista no cubre el tiempo total del intervalo de cuenta del temporizador, se coloca una tarea que espere a que se cumpla el periodo del temporizador, una vez que llega la interrupción del temporizador el administrador de tareas vuelve al inicio de la lista de tareas y ejecute la secuencia hasta que se cumpla el periodo del temporizador y así sucesivamente. Esta es la forma más sencilla de asegurar que una tarea se ejecute a un periodo pre-establecido con precisión.

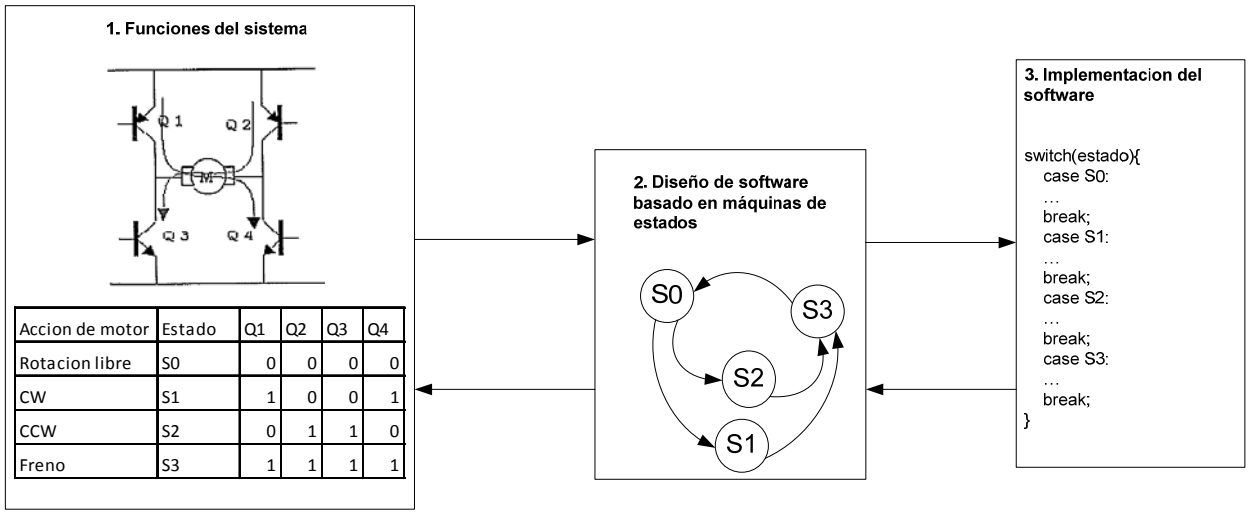

Figura 1. Diseño de software basado en máquinas de estados

Una ventaja muy importante de realizar el diseño del software para sistemas embebidos en base a máquinas de estados es que es muy fácil lograr una trazabilidad de los requerimientos hacia etapas avanzadas de diseño y vice-versa como se muestra en la figura 1. ya que los procesos mecánicos o eléctricos que se desean controlar también pueden modelarse como máquinas de estados.

Los administradores cooperativos pueden utilizar máquinas de estados, y de hecho es muy ventajoso hacer el diseño de un sistema operativo en tiempo real multitarea utilizando esta metodología ya que cada tarea se diseña como una máquina de estados independiente (Berndt, D. 2008).

Las máquinas de estados finitos nos permiten dividir las secuencias de control de los sistemas en una serie de pasos o estados que se van ejecutando en un orden pre-establecido (Curtis, K. 2006) y se puede variar el flujo de ejecución dependiendo del estado de las entradas del sistema.

Las máquinas de estados pueden definirse como la 5-tupla (Σ, S, S_0, F, δ) (Cheng, A. 2002).

- Σ, alfabeto finito de valores que pueden tomar las entradas
- S, conjunto finito de estados
- $S_0 \in S$, estado inicial
- F, conjunto de estados finales
- δ, Es la función de transición de estados $S \times \sum \rightarrow S$

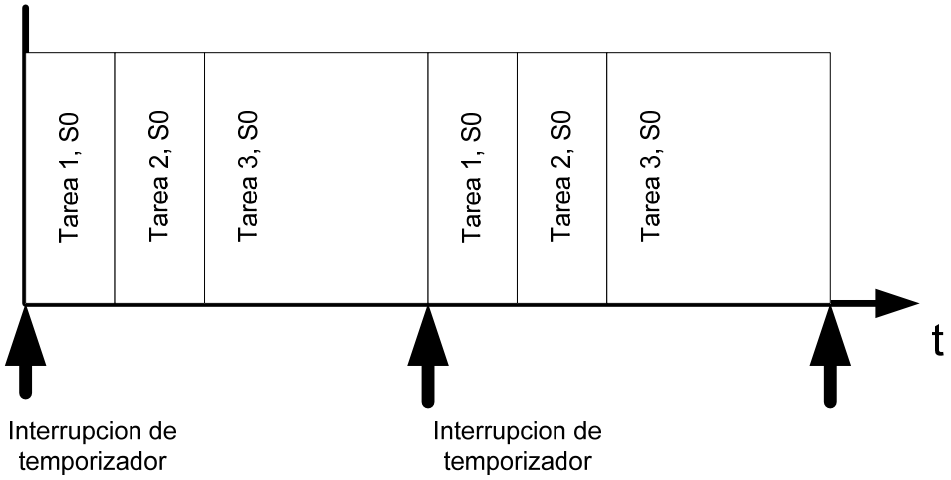

Figura 2. Administración de tareas cooperativamente, por diseño las tareas se colocan en el orden de ejecución requerido

Es importante mencionar que esta forma de multitarea es inherentemente determinística ya que el orden en que se ejecutan las tareas se define en la etapa de diseño y son codificadas en ese orden (ver figura 2), lo que hace que los sistemas con administradores de tareas cooperativos sean muy fáciles de depurar. Este tipo de administración de tareas es el que ofrece menor latencia cuando se cambia entre tarea y tarea, ya que el estado actual de la tarea depende solamente de su variable de estado y cada máquina de estado posee sus propias variables en la memoria.

En sistemas de aplicaciones criticas, inclusive en sistemas de los cuales depende la vida humana, se prefiere la utilización de administradores de tareas de tipo cooperativo por su determinismo.

Sin embargo a pesar de su simplicidad tiene algunas desventajas:

- El funcionamiento correcto del sistema depende de que todas las tareas estén libres de funciones o condiciones que secuestren los recursos de hardware por más del tiempo permitido. Aunque, este efecto puede ser mitigado por medio de la utilización de temporizadores que vigilen la operación del sistema y que puedan tomar alguna acción para desbloquear el mismo.
- Su diseño puede llegar a requerir un esfuerzo adicional en comparación a otras arquitecturas.

3.1 ARQUITECTURA DE SOFTWARE

El esquema de la arquitectura de software propuesta es sencillo y puede ser adaptado a múltiples plataformas de hardware (ver figura 3). Está compuesto principalmente por dos bloques que son el administrador de tareas cuya función es realizar la ejecución de las tareas por medio de un algoritmo dedicado y el módulo de administración de tiempo el cual requiere de un temporizador de hardware para generar el tiempo de ejecución base del sistema.

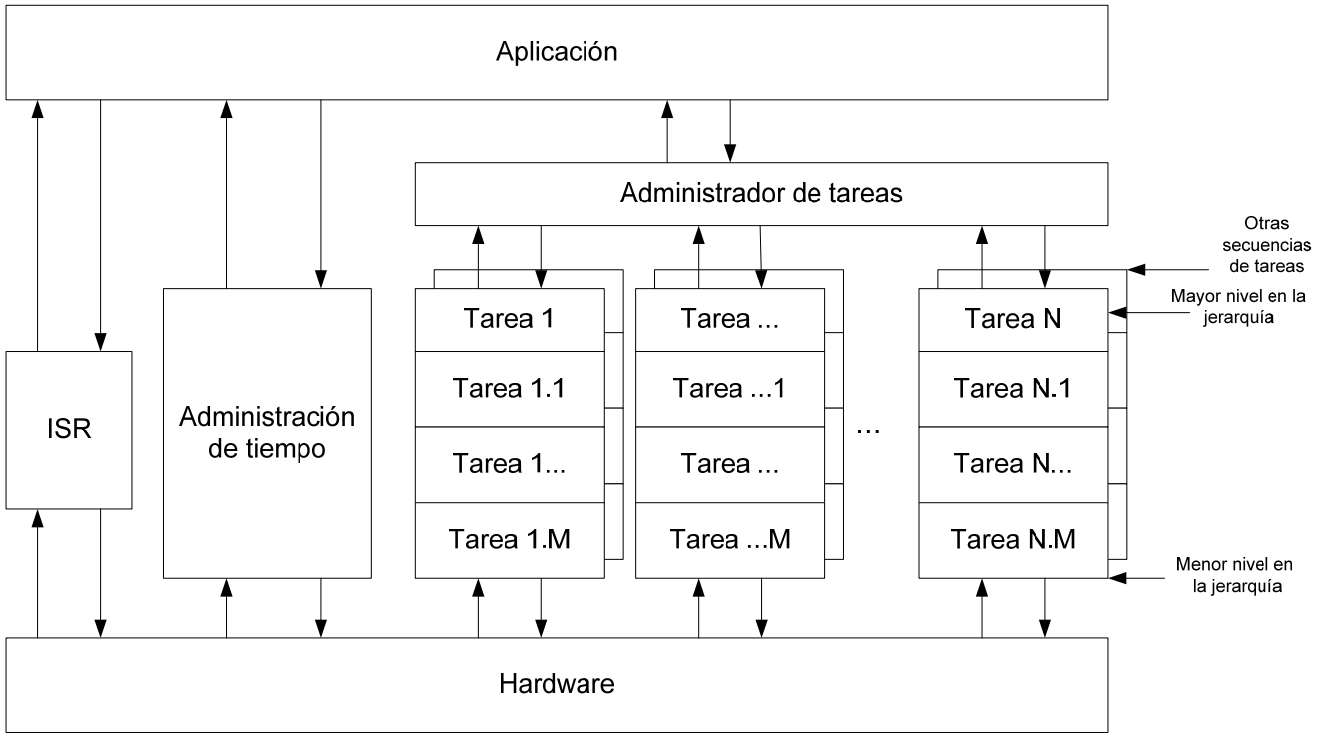

Figura 3. Esquema de ejecutivo en tiempo real propuesto

Como se mencionó anteriormente las tareas son implementadas como máquinas de estados finitos, las cuales se ejecutan en una secuencia según las prioridades asignadas. Pueden definirse varias secuencias de ejecución de tareas dependiendo del modo de operación del sistema, por ejemplo secuencias diferentes para modo normal, modo de error, modo de depuración, etc.

En la figura también se muestra que las tareas pueden ser jerárquicas, es decir que son máquinas de estados finitos compuestas de otras máquinas de estados finitos de niveles inferiores.

3.2 EJECUCIÓN DE TAREAS

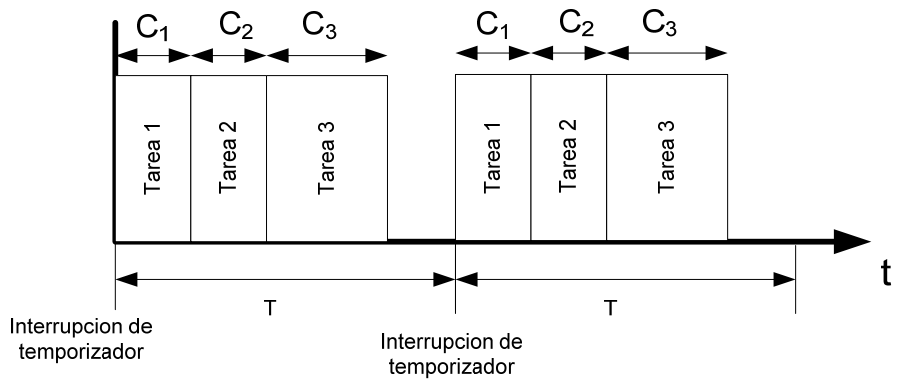

Figura 4. Duración de las tareas en relación con el tiempo base del sistema

En la figura 4. el tiempo T representa el tiempo base del sistema producido por el temporizador de hardware y C_1, C_2, C_3 representan los tiempos de ejecución de cada tarea en el peor de los casos, entonces:

$$U = \frac{\sum_{i=1}^{N} C_i}{T}$$

Donde U es el factor de utilización del procesador. Es claro que debe cumplirse que $U \leq 1$.

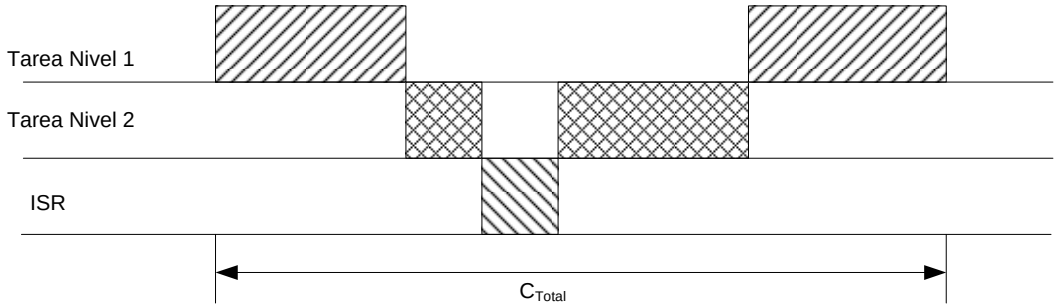

Figura 5. Composición de la duración de una tarea jerárquica

En la figura 5 se muestra el caso en el que una tarea esté compuesta de otra de jerarquía inferior y además se puede dar el caso de que se ejecute una rutina de interrupción entre ellas. Se recomienda diseñar las rutinas de interrupción de manera que su tiempo de ejecución sea lo más corto posible para evitar que tengan un impacto severo sobre la duración total de la tarea de mayor jerarquía.

Entonces la duración total percibida de la tarea de mayor jerarquía es:

$$C_{Total} = C_N + \sum_{i=1}^{M} C_{(N,i)} + C_{ISR}$$

En donde :

C_N : Tiempo de ejecución en el peor de los casos de la tarea de mayor jerarquía

$\sum_{i=1}^{M} C_{(N,i)}$: Se refiere a la sumatoria del tiempo de ejecución en el peor de los casos todas las tareas de menor jerarquía bajo la tarea C_N

C_{ISR} : Es el tiempo de ejecución de la rutina de interrupción

3.3 ESTATUS DE LAS TAREAS

El estatus de las tareas del sistema se refiere a un control de mayor nivel relacionado al administrador de tareas y no se debe confundir con la variable de estados. El administrador de tareas debe verificar el estatus de cada tarea y determinar si ésta debe ejecutarse o no.

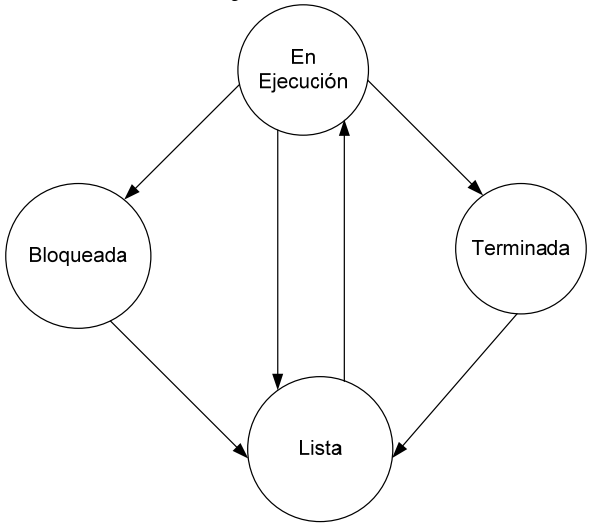

Figura 6. Estatus de tareas

En la figura 6. se muestra un diagrama genérico de los posibles estatus que pueden tomar las tareas y a continuación se da una breve descripción de cada uno de ellos:

En ejecución: Es el estado normal operación en que se ejecuta el programa o funciones específicos para los cuales fue diseñada la tarea.

Lista: Indica que la tarea esta lista para entrar en estado de ejecución cuando le llegue su turno.

Terminada: Es el estado en el que se detiene completamente la tarea. Para volver a ejecutar la tarea esta debe ser puesta en modo de lista.

Bloqueada: Este estado indica que la tarea está en espera de ser activada o de algún evento para poder entrar en estado de lista.

4. IMPLEMENTACIÓN

En la sección anterior se explicó el diseño conceptual y la arquitectura del ejecutivo en tiempo real multitarea. A continuación se detallan las consideraciones de implementación.

Para los ejemplos de programación se utilizará el lenguaje C ya que es el más común entre las plataformas de sistemas embebidos en la actualidad.

4.1 DEFINICIÓN DE ESTRUCTURA DE TAREA

La única variable que se necesita para implementar una tarea como una máquina de estados finitos en su forma más básica es la variable de estados, sin embargo se puede definir una estructura más completa con todas las variables requeridas por la tarea por ejemplo número de identificación de la tarea, contadores, variable de estado,

variable de estatus, variable de error, puntero a función de implementación de la tarea, etc, cada una con su tipo de datos correspondiente.

```
struct sm_task {
   uint8_t  id;              /*Número de identificación de la tarea*/
   uint8_t  state;           /*Variable de estados*/
   uint8_t  result;          /*Variable de resultado de ejecución de la tarea*/
   uint8_t  event;           /*Variable de evento*/
   uint32_t counter;         /*Contador de periodos de tiempo base*/
   uint8_t  status;          /*Estatus de la máquina de la tarea*/
   uint8_t  offset;          /*Desplazamiento de periodos de tiempo base*/
   uint32_t period;          /*Cantidad de periodos a contar*/
   void  (* fPtr) (void);  /*Puntero a función de implementación de la máquina de estados*/
};
```

4.2 CREACIÓN DE TAREAS

Las tareas pueden crearse al inicio de la aplicación principal por medio de un arreglo del tipo correspondiente a la estructura de tarea definida anteriormente, como por ejemplo:

```
struct sm_task SM_TASKS[] = {
                    {TAREA0,    0, 0, 0,  0,    TASK_READY, 0, 0,    tarea0},
                    { TAREA1,   0, 0, 0, 10,    TASK_READY, 0, 10,   tarea1},
                    { TAREA2,   0, 0, 0, 10,    TASK_READY, 0, 10,   tarea2},
                    {0xFF,       0, 0, 0,  0,    0,                  0, 1,    0}
                  };
```

Las tareas del sistema pueden ser implementadas por medio de estructuras switch/case muy similar a como aparece en la figura 1, en las cuales cada case representa una partición funcional de la tarea. Para hacer esta partición se requiere un análisis de la utilización de los recursos del sistema.

4.3 ALGORITMO DE EJECUCIÓN DE TAREAS

El algoritmo de ejecución de tareas se ejecuta por el módulo de administración de tareas antes de cada tarea en la lista, produciendo una pequeña latencia como se muestra en la figura 7 y ocupando tiempo del microprocesador.

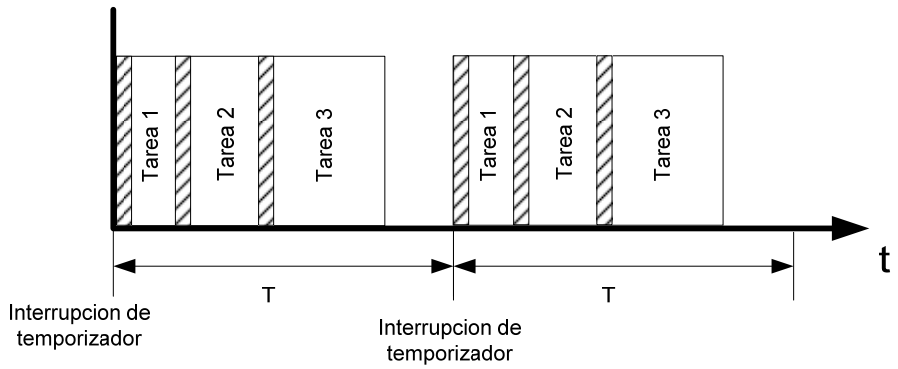

Figura 7. Las porciones sombreadas representan el tiempo de ejecución del administrador de tareas

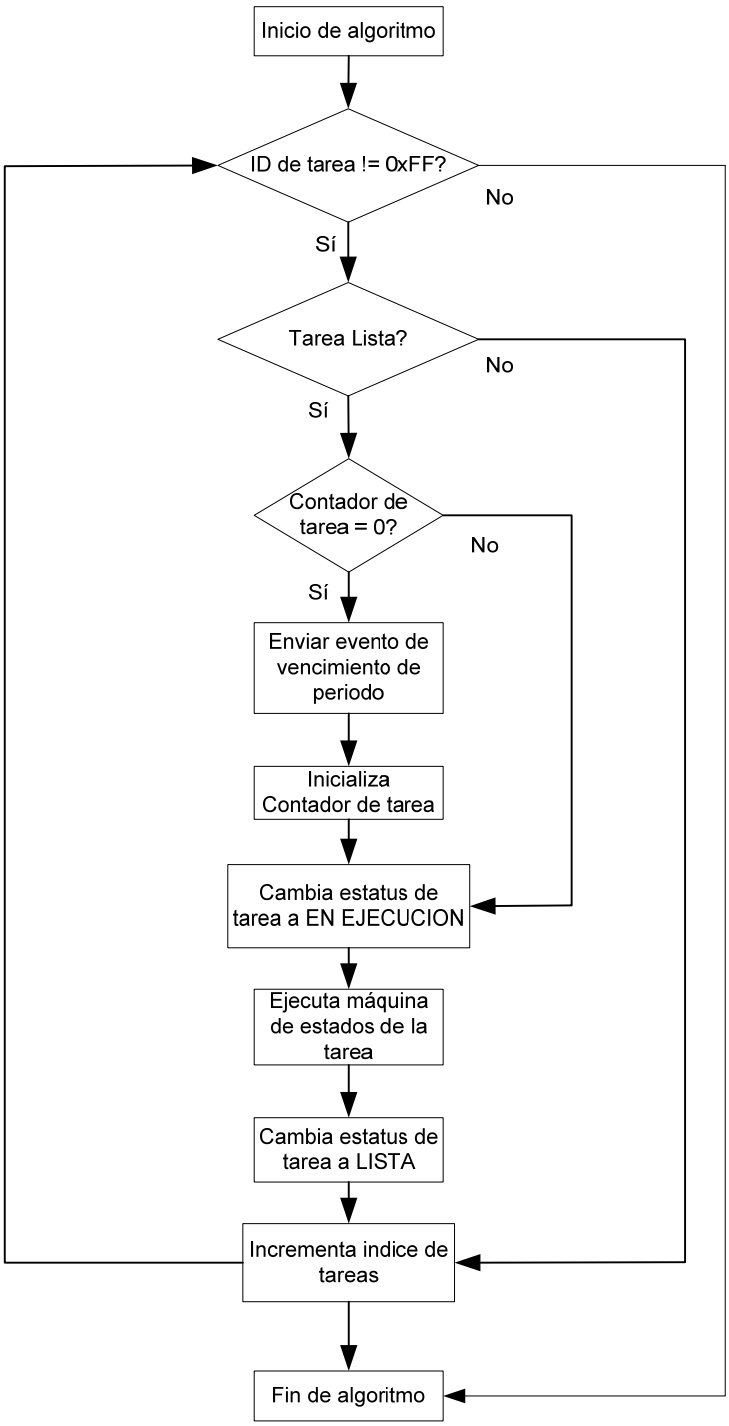

Figura 8. Diagrama de flujo del administrador de tareas

La rutina del administrador de tareas es muy eficiente (ver figura 8) ya que requiere de pocos ciclos de instrucción para cambiar entre tareas en comparación con otros esquemas de administración, lo cual reduce el tiempo sombreado en la figura 7, es decir mejora el factor de utilización.

Esta arquitectura de software puede ser implementada en microcontroladores o DSPs de fabricantes que ofrecen librerías, sistemas de archivos y controladores de software listos pero sin administrador de tareas.

5. PRUEBAS DE DESEMPEÑO

5.1 CONFIGURACIÓN DE LA PRUEBA

Se probó el desempeño de la implementación de la arquitectura propuesta en comparación con otros sistemas operativos de tiempo real o RTOS (por sus siglas en inglés) comerciales y no-comerciales disponibles en el mercado.

Para las pruebas se utilizó la misma arquitectura de hardware con las siguientes características:

Procesador: dsPIC33F
Arquitectura: 16 bit , Harvard Modificada
Frecuencia de bus: 40 MHz (40 MIPS)
Memoria de datos: 16.384 KB
Memoria de programa: 256 KB

Los parámetros considerados en las pruebas son los más comunes para determinar el desempeño del sistema en función del tiempo y utilización de recursos (Xu, T. 2008):

- **Tiempo de intercambio de contexto:** Tiempo que toma el sistema operativo para cambiar de una tarea a otra, sin ejecutar ninguna otra tarea o interrupción durante el cambio.
- **Tiempo de servicio de interrupción:** Tiempo que toma el sistema operativo en servir una interrupción y retornar el control de los recursos a la tarea interrumpida.
- **Utilización de memoria de programa:** Cantidad de memoria ROM del programa compilado, que incluye el código referente al sistema operativo y adicional el código de la aplicación.
- **Utilización de memoria de datos:** Cantidad de memoria RAM asignada a estructuras del sistema operativo y memoria reservada para las tareas.

Los sistemas operativos en tiempo real evaluados en esta prueba son el DSPNano que es un RTOS comercial y de código cerrado, FreeRTOS que es gratuito y de código abierto y el ejecutivo en tiempo real propuesto en este trabajo que llamaremos RTXFSM.

5.2 METODOLOGÍA DE MEDICIÓN

Por medio de diferentes fuentes (Xu, T. 2008) y (Aroca y Caurin, 2007) pudimos confirmar que la mejor manera de medir los parametros relacionados al tiempo es a través de hardware externo. Las mediciones se hacen a una salida digital del sistema que se activa al inicio del evento que se desea medir y se desactiva cuando se ha realizado el procesamiento en observación, utilizando un osciloscopio digital.

5.3 RESULTADOS

Tabla 1: Resultados de pruebas de desempeño

	FreeRTOS	**DSPNano**	**RTXFSM**
Tiempo de intercambio de contexto	32 μs	4.4 μs	0.96 μs
Tiempo de procesamiento de interrupción	32 μs	5.7 μs	5 μs
Memoria de Programa (ROM)	2523 B	2523 B	646 B
Memoria de Datos (RAM)	5378 B	2919 B	32 B

En la tabla 1 se muestran los resultados de la prueba de desempeño. En ella podemos observar los valores obtenidos para los parámetros de interés mencionados anteriormente. Con ésta información podemos estimar el desempeño del uso de alguno de éstos en una aplicación. Podemos observar que la arquitectura que proponemos

es la que consume menos tiempo en funciones administrativas de tarea permitiendo entonces factores de utilización mayores y también hace una utilización de los recursos de memoria mucho más eficiente.

REFERENCIAS

Aroca, R., Caurin, G.(2007). *A Real Time Operating Systems (RTOS) Comparison*. Universidade de São Paulo

Berndt, D. (2008). Mission-critical software architecture: FSMs versus RTOS, http://mil-embedded.com/article-id/?3049, Military Embedded Systems, 09/06/2010.

Cheng, A. (2002). *Real-Time Systems-Scheduling Analysis and verification.*John Wiley & Sons, USA.

Curtis, K. (2006). *Embedded Multitasking with small microcontrollers*, Elsevier Inc. UK.

Stoiner-Gibson,D. (2010). Understanding embedded microcontroller multitasking RTOS alternatives, SPLat Controls, http://www.splatco.com/rtos_1.htm, 09/26/10.

Wiliams, R. (2006). *Real-Time systems development*, Elsevier Inc., UK.

Xu, T(2008). *Performance benchmarking of FreeRTOS and its Hardware Abstraction*, Technische Universitet Eindhoven

Authorization and Disclaimer

Authors authorize LACCEI to publish the paper in the conference proceedings. Neither LACCEI nor the editors are responsible either for the content or for the implications of what is expressed in the paper.