Modelo de tarea para recuperación de Slack para Aplicaciones en Sistemas Embebidos con DVS

José M. Urriza, Ricardo Cayssials y Javier D. Orozco
Universidad Nacional del Sur / CONICET
Departamento de Ingeniería Eléctrica y Computadoras
8000 Bahía Blanca, Argentina
{jurriza, iecayss, ieorozco}@criba.edu.ar

y

J.C.B. Leite
Universidade Federal Fluminense
Instituto de Computação
Niterói, Brasil
julius@ic.uff.br

Abstract

In this paper we present a task model that makes an efficient utilization of the slack available in hard real time systems that support *Dynamic Voltage Scheduling*. Slack-Stealing methods proposed in hard real-time literature consider that execution times of the tasks are the worst-case execution times when the schedulability of the system is analyzed. However, the worst-case execution time rarely happens during runtime. In such a case, the remaining time can be used as slack available to execute sporadic tasks, optional tasks, aperiodic tasks or to reduce the frequency-voltage of the system in order to save energy. Because the power consumption of the processors in embedded systems such as cellular phones, Palms and notebooks is a square function of voltage supplied, the reduction of the voltage supplied improves dramatically the power consumption of the system. In this way, the model proposed in this paper allows us to do an efficient utilization of the slack available towards a reduction of the power consumption of the processor.

Keywords: Slack Stealing, DVS, Power saving

Resumen

En este trabajo se presenta un modelo de tareas para uso eficiente del slack en sistemas de tiempo real duro donde se puede modificar el voltaje-frecuencia del procesador (*Dynamic Voltage Scheduling*). Los métodos de *Slack Stealing* propuestos en la literatura de sistemas de tiempo real duro consideran que los tiempos de ejecución de las tareas son los máximos posibles con el objeto de garantizar la planificabilidad del sistema. En la práctica, existe una baja probabilidad de que la tarea requiera ejecutarse su peor tiempo de ejecución. En tal caso, el tiempo disponible puede ser considerado tiempo ocioso adicional y ser utilizado para la ejecución de tareas esporádicas, opcionales, aperiódicas o para disminuir el nivel de voltaje-frecuencia con el objetivo de economizar energía. Debido a que el consumo de energía en los procesadores de sistemas embebidos, como los celulares, Palms y notebooks, es función del voltaje al cuadrado, pequeñas disminuciones de voltaje producen un ahorro de energía sustancial. De esta manera, el modelo propuesto en este trabajo permitirá un aprovechamiento del slack disponible que permitirá la reducción del consumo de energía del sistema.

Palabras Claves: Slack Stealing, DVS, Economía de Energía

1 INTRODUCCIÓN A LOS SISTEMAS DE TIEMPO REAL

Los *Sistemas de Tiempo Real* (*STR*) se pueden encontrar en casi todas las ramas de la ingeniería abarcando inclusive desde hace algunos años, áreas de sistemas electrónicos de consumo de mediana y baja complejidad.

Se puede definir a un *STR* como aquel en el que los resultados de una tarea son aceptables si se obtienen antes de un determinado tiempo denominado vencimiento. Es aceptada entonces la definición de Stankovic [1] que dice que: *en los STR los resultados no sólo deben ser lógicamente correctos si no que además su aceptabilidad depende del tiempo en el cual son producidos.*

Los *STR* se pueden clasificar en tres tipos dependiendo de la aceptabilidad de los resultados respecto de sus vencimientos: (1) Si no se admiten pérdidas en los vencimientos, se los denominan *duros* o *críticos* (*hard*), (2) si es admisible la pérdida de algunos vencimientos según alguna ley de aceptabilidad se los denominan *blandos* (*soft*). Estos últimos utilizan criterios basados en la calidad de servicio u otros de optimalidad global para caracterizar la tolerancia del sistema a las pérdidas de vencimientos. Es por ello que dentro de los *STR* blandos podemos encontrar una amplio espectro de definiciones que van desde sistemas muy próximos a los de *STR* duro a sistemas con muy bajo nivel de predictabilidad. A partir de estos extremos es posible encontrar algunas tipificaciones como las de (3) los sistemas *firmes* (*firm*) que admiten una determinada cantidad de pérdidas distribuidas según una ley prefijada por las características de la aplicación.

La definición de los *STR duros* parte de la premisa de que, la pérdida de un vencimiento en una tarea, puede tener consecuencias adversas. En los sistemas en los cuales la vida humana es participe, por ejemplo la avionica, se le debe garantizar que todas las tareas terminen antes de su vencimiento mediante un *test de diagramabilidad* o *planificabilidad*. Sí el test es exitoso, se dice que el sistema es *factible* o *planificable*.

En 1973, C. L. Liu y James W. Layland [2] realizan el primer aporte significativo para determinar la planificabilidad de un *STR duro*. Este trabajo se ha convertido en un trabajo liminar de la disciplina, debido a que propone un marco formal para el análisis de factibilidad de un sistema multitarea - monoprocesador con tareas independientes.

Existen diferentes mecanismos para lograr planificar un sistema multitarea – monoprocesador que van desde los sistemas de asignación de recursos estáticos (ejecutivos cíclicos) a sistemas completamente dinámicos tanto en la asignación de recursos como en la constitución del sistemas. Para estos últimos y todas las variantes intermedias, resulta conveniente la utilización de criterios basados en la aplicación de *disciplinas de prioridades* que establezcan una relación lineal de orden sobre el conjunto de tareas de manera que el *planificador* pueda determinar qué tarea tiene derecho de utilización del recurso en cada instante de activación.

En lo que sigue se utilizará un modelo de tareas donde las mismas se consideran periódicas, independientes y apropiables. Una tarea periódica es aquella en la que su arribo se produce cada T_i unidades de tiempo; independiente, cuando su ejecución no depende temporal ni lógicamente de la ejecución de otra tarea del sistema, y se dice que una tarea es apropiable, cuando el *planificador* puede suspender su ejecución y desalojar el recurso en cualquier momento.

Luego, una tarea τ_i se encontrará completamente caracterizada con la terna (C_i, T_i, D_i) donde C_i representa su tiempo de ejecución; T_i su periodo y D_i su vencimiento. Por lo tanto, un conjunto $\boldsymbol{S}(n)$ de n tareas se encuentra especificado por $S(n)=\{(C_1, T_1, D_1), (C_2, T_2, D_2),..., (C_n, T_n, D_n)\}$.

Existen en la actualidad tres *disciplinas de prioridades* básicas para la diagramación de tareas: Rueda Cíclica (Round Robin), Prioridades Fijas (Fixed Priorities) en sus variaciones Rate Monotonic [2] (RM) y Deadline Monotonic [3] (DM) y disciplinas de prioridades dinámicas como Earliest Deadline First [2] (EDF). Además de ellas, existen combinaciones de las mismas como el método Dual Priority [4].

El análisis de planificabilidad de sistemas de tiempo real se basa en el peor caso de carga del sistema. Si el sistema cumple con las restricciones temporales bajo el peor caso de carga, entonces se puede garantizar que el sistema las cumplirá bajo cualquier otro caso de carga arbitrario. Diseñar bajo esta hipótesis provoca diseños pesimistas. El procesador es dimensionado para un estado de carga que raramente ocurre y en consecuencia permanece ocioso la mayor parte del tiempo. La recuperación de esta diferencia entre el peor tiempo de ejecución y el tiempo que ocurrió, se la denomina *tiempo ganado*.

La planificación utilizando regulación de voltaje (*DVS*) permite maximizar la potencia de procesamiento, que depende de las exigencias del momento, tratando de adoptar un nivel de voltaje frecuencia que minimice el consumo de energía. Está probado que la minimización que este tipo de problema es no polinomial (NP). De esta manera, el objetivo de planificación con reducción de voltaje es cumplir con las restricciones temporales del sistema de tiempo real con el menor consumo de energía posible.

El trabajo plantea un modelo de tareas adecuado para la implementación de algoritmos de planificación con reducción de voltaje-frecuencia, basados en la utilización de algún método de *Slack Stealing* o aprovechamiento del *tiempo ganado*. El modelo utiliza el slack disponible con el fin de adecuar el voltaje-frecuencia del procesador para reducir el

consumo de energía del sistema al tiempo de garantizar su planificabilidad, permitiendo al algoritmo de planificador predecir que cantidad exacta posee de slack.

El presente trabajo se organiza de la siguiente manera: en la sección 2 se presenta los trabajos previos, en la sección 3 se introduce el modelo de tarea-planificador, en la sección 4 se presenta como es posible recuperar Slack y se ejemplifica y finalmente en la sección 5 se realizan las conclusiones.

2 TRABAJOS PREVIOS

El primer trabajo para reducir el consumo de energía en procesadores mediante la utilización de relaciones voltage-frecuencia variables, fue propuesto por Weiser [5] en 1994 y extendido en [6] por Govil. En los últimos años, ha cobrado especial importancia la reducción del consumo debido a la expansión y diversificación de las aplicaciones y dispositivos móviles (PDAs, Teléfonos móviles, computadoras portátiles, redes de sensores, etc). Tanto los investigadores como los fabricantes de microprocesadores (Intel, AMD, Transmeta, ARM) están buscando obtener el menor consumo posible conjuntamente con un desempeño acorde a las demandas del mercado.

Este objetivo, integra requerimientos que van desde la planificación de tareas, tecnología de baterías, arquitecturas de los microprocesadores, optimización de código en el proceso de compilación con el objeto de minimizar los requerimientos de memoria y por ultimo, optimizar las protocolos de comunicaciones para conservar una adecuada relación de seguridad, velocidad y consumo. Algunos otros motivos para ahorrar energía son originados en la necesidad de disminuir el tamaño del dispositivo, minimizar la disipación y ofrecer nuevas características tales como envió de imágenes y sonido que generalmente necesitan de un microprocesador con una gran capacidad de cálculo que, si son utilizados permanentemente a máxima frecuencia, poseen un alto consumo.

Los métodos de diagramación por reducción de voltaje implementan diversas disciplinas de prioridades para diagramar sus tareas. En trabajos como [7-9] utilizan prioridades fijas (*RM* y *DM*), en trabajos como [9-12] utilizan prioridades dinámicas (EDF) y por último, prioridades mixtas en trabajos como en [13]. Algunos otros trabajos utilizan un ejecutivo cíclico como en [10] o realizan su diagramación fuera de línea, es decir de forma estática como en [14-16].

Existe una división en la forma en cómo utilizan el slack y el *tiempo ganado*. Ciertos trabajos buscan bajar de niveles de frecuencia de manera determinística y otros trabajos utilizan una manera probabilística [17]. Todos aprovechan el *tiempo ganado* para lograr una reducción de la energía consumida.

En este trabajo se presenta un modelo de tareas para trabajar de manera determinística, bajo la *disciplina de prioridades RM* o *DM* con un método de *Slack Stealing*. Se demostrará que de esta manera, se puede recuperar todo el slack que no se utiliza cuando las tareas finalizan antes de su peor tiempo de ejecución.

3 INTRODUCCIÓN AL MODELO

Para definir este modelo se tienen en cuenta las siguientes consideraciones:

1. El modelo de procesador es de m niveles de voltaje-frecuencia con $m \geq 2$. El nivel 1 se considera el de mayor frecuencia y mayor voltaje.
2. El conjunto de n tareas de tiempo real $\mathbf{S}(n)$ se considera duro y debe ser planificable, por lo menos, para las mayores frecuencias del procesador adoptado.
3. El método propuesto cuenta con un mecanismo de cálculo de slack en línea (que puede ser exacto o aproximado) como los desarrollados en los trabajos [4, 18-22], en particular se utilizará el de [21] por ser diseñado para sistemas embebidos.

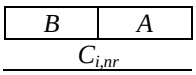
Bajo estas condiciones, el conjunto de tareas $\mathbf{S}(n)$ puede ser planificable en varias o en todas las frecuencias de un procesador, pero se considerara que por lo menos no lo es en varias de las menores frecuencias. Se parte de la base que, para ahorrar energía, el conjunto de tareas $\mathbf{S}(n)$ se ejecutará en la menor frecuencia en la cual es planificable. A esta frecuencia la definimos como f_{n1} (con $1 \leq n1 < m$) y en ésta se calcula el slack disponible (K_i) de cada tarea, por el método que está definido en el trabajo [21]. Por lo tanto se obtienen el slack disponible para cada tarea τ_i en el instante t, $K_i(t)$ y consecuentemente se obtiene el slack del sistema en dicho tiempo, $K(t)$.

3.1 Modelo de ejecución de tareas en sistemas embebidos con diagramación con reducción de voltaje

El *STR* sólo puede ser retrazado $K(t)$ unidades de tiempo y se considera que dicho retrazo es suficiente para poder reducir la frecuencia de operación hasta una cierta frecuencia que denominamos f_{nr} (con $n1 < nr \leq m$). Debido a esto, su tiempo de ejecución se amplía una proporción que depende de estas frecuencias:

$$C_{i,nr} = \frac{f_{n1}}{f_{nr}}.C_{i,n1}$$

Para poder tratar este aumento en el tiempo de ejecución sin peder planificabilidad, se divide a la tarea en dos partes. La parte *A* es el tiempo de ejecución de la tarea en la frecuencia f_{n1} y la parte *B* es la porción de aumento de la tarea a la nueva frecuencia f_{nr}.



El siguiente lema es condición necesaria para la ejecución de la tarea a frecuencia f_n:

Lema 1:

Se podrá ejecutar una tarea a una frecuencia f_{nr} si y solo si la parte *B* es igual o menor que el slack del sistema en ese instante.

***Prueba*:**

La ecuación que determina *B* es:

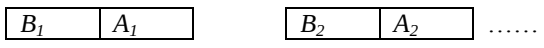
$$B = \frac{f_{n1}}{f_{nr}} A - A$$

Si la parte *B* fuese mayor al slack disponible, se perderían las constricciones de tiempo debido a que se retrazaría al sistema de tiempo real, más de lo que se puede permitir.□

Como la parte *A*, está contemplada en el cálculo del slack, ésta puede ser desalojada si una tarea de prioridad superior tiene que ser ejecutada. La parte *B*, por el contrario a la parte *A*, es de ejecución obligatoria y no puede ser desalojada hasta su finalización. Esto se debe a que el sistema debe decrementar los contadores $K_i(t)$, es decir, robarle al *STR*, *B* unidades de tiempo, sustrayendo de manera precisa el tiempo necesario para su ejecución. De no ser así, la llegada de una segunda tarea de prioridad mayor, desalojaría a la primera y consecuentemente le robaría el slack que se tenía en cuenta para cambiar la frecuencia de operación de la primera tarea. Sí esto ocurriese, se pierde el control de qué tarea puede cambiar de nivel, o no, y si existe suficiente slack disponible para que se realice. Por lo tanto, el orden de ejecución de las partes es: primero la parte *B*, de ejecución obligatoria, y luego la parte *A*, hasta que concluya o sea desalojada.

3.2 Ejecución de una Tarea en distintas frecuencias

Debido a que una tarea puede ser interrumpida por tareas de mayor prioridad, la parte *A* se puede fragmentar varias veces hasta completar su ejecución. Por esto, una tarea puede ser ejecutada en distintas frecuencias, siempre y cuando exista el slack necesario que lo permita. El tratamiento para cuando ocurren este tipo de casos es del mismo modo que si fuese una tarea nueva con su tiempo de ejecución en la última frecuencia a la cual se ejecuto, es decir se divide en una parte *A* y una parte *B*.



$$B_1 = A\left(\frac{f_1}{f_2} - 1\right),\ B_2 = \left(A\frac{f_1}{f_2} - A_1 - B_1\right)\left(\frac{f_2}{f_3} - 1\right) \ldots\ldots..$$

4 RECUPERACIÓN DE SLACK

Hasta ahora se a ha supuesto que el tiempo de ejecución (C_i) de la tarea, corresponde al peor tiempo de ejecución, pero puede que esto no ocurra. De hecho sólo muy pocas veces podrá suceder, pero al considerar al *STR* como duro, ésta es una hipótesis que se debe si o si realizar.

Sí no ocurre el peor tiempo de ejecución entonces, se debe recuperar el *tiempo ganado* como slack, para que sea aprovechado por las demás tareas. Existen dos tipos de slack que tienen que ser recuperados, una vez que la tarea τ_i termina.

El primero, es el *tiempo ganado* como está definido en los trabajos [4, 18-22] y es directamente la diferencia entre el tiempo de ejecución del peor caso y el tiempo de ejecución que aconteció. A esta diferencia la denominaremos δ. Todas las tareas de prioridad menor contemplan en su cálculo de slack el peor tiempo de ejecución de la tarea τ_i, y como la tarea acaba de terminar con un tiempo menor al peor tiempo de ejecución se puede recuperar esta diferencia. La forma de recobrarlo es simplemente incrementar los contadores de slack de cada tarea de menor prioridad a *i* con δ.

$$\overset{n}{\underset{j=i+1}{K_j(t)}} = K_j(t) + \delta \tag{1}$$

El segundo tipo de slack que hay que recuperar, es el que cada tarea gasto de más, en cada nivel, previendo que ocurriría el peor tiempo de ejecución. Hay que tener en cuenta que cada vez que se cambia la frecuencia de operación, la tarea se divide en dos partes y el cálculo de cuánto slack se debe robar al sistema se realiza con el peor tiempo de ejecución. Pero si el peor tiempo de ejecución no ocurre, existe una sobreestimación del slack que se precisaba para ejecutar dicha tarea, es decir se le robo al *STR* más slack del necesario y por lo tanto se debe recuperar y restituirlo al sistema.

Lema 2:

Dada una tarea que su $K_j(t)$ está calculado a la frecuencia f_{n1} y que su última parte se ejecuta a la frecuencia f_{nr}, la cantidad de slack que se recupera para todas las tareas de prioridad menor es:

$$\overset{n}{\underset{j=i+1}{K_j(t)}} = K_j(t) + \delta\left(\frac{f_{n1}}{f_{nr}} - 1\right) \tag{2}$$

Prueba:

Se demostrará para una tarea que se divide en *n* partes. Para simplificar la notación de la demostración, se toma al nivel *n*1 como 1 y al nivel final como *n*+1. Se hace notar que si una tarea se divide en *n* partes, con la frecuencia original hay *n*+1 frecuencias.

A_n = Tiempo ejecutado en la frecuencia *n*
Ar_n = Tiempo restante para ser ejecutado en la frecuencia *n*

$$A = C_i$$

$$A^* = A - \delta$$

B_1	A_1		B_2	A_2	……	B_n	A_n

$$Ar_1 = A,\ \ Ar_2 = Ar_1\frac{f_1}{f_2} - A_1 - B_1,\ \ Ar_3 = Ar_2\frac{f_2}{f_3} - A_2 - B_2 \ldots\ldots\ Ar_n = Ar_{n-1}\frac{f_{n-1}}{f_n} - A_{n-1} - B_{n-1}$$

$$Ar_1^* = A - \delta,\ \ Ar_2^* = Ar_2 - \delta\frac{f_1}{f_2},\ \ Ar_3^* = Ar_3 - \delta\frac{f_1}{f_3},\ldots\ldots\ Ar_n^* = Ar_n - \delta\frac{f_1}{f_n}$$

El slack gastado en cada nivel y el que tendría que haber sido es

$$B_1 = Ar_1\left(\frac{f_1}{f_2} - 1\right),\ B_2 = Ar_2\left(\frac{f_2}{f_3} - 1\right),\ldots\ldots\ B_n = Ar_n\left(\frac{f_n}{f_{n+1}} - 1\right)$$

$$B_1^* = Ar_1^*\left(\frac{f_1}{f_2}-1\right),\ B_2^* = Ar_2^*\left(\frac{f_2}{f_3}-1\right),\ldots\ldots\ldots B_n^* = Ar_n^*\left(\frac{f_n}{f_{n+1}}-1\right)$$

Se sabe que la cantidad que se ejecuta por cada parte es la misma.

$$B_1 + A_1 = B_1^* + A_1^*,\ B_2 + A_2 = B_2^* + A_2^*,\ldots\ldots\ldots\ B_n + A_n = B_n^* + A_n^*$$

Entonces

$$A_1^* = B_1 + A_1 - B_1^* = Ar_1\left(\frac{f_1}{f_2}-1\right) + A_1 - \left(A-\delta\right)\left(\frac{f_1}{f_2}-1\right) = A_1 + \delta\left(\frac{f_1}{f_2}-1\right)$$

$$A_2^* = B_2 + A_2 - B_2^* = Ar_2\left(\frac{f_2}{f_3}-1\right) + A_2 - Ar_2^*\left(\frac{f_2}{f_3}-1\right)$$

$$A_2^* = Ar_2\left(\frac{f_2}{f_3}-1\right) + A_2 - \left(Ar_2 - \delta\frac{f_1}{f_2}\right)\left(\frac{f_2}{f_3}-1\right)$$

$$A_2^* = Ar_2\frac{f_2}{f_3} - Ar_2 + A_2 - Ar_2\frac{f_2}{f_3} + Ar_2 + \delta\frac{f_1}{f_3} - \delta\frac{f_1}{f_2}$$

$$A_2^* = A_2 + \delta\frac{f_1}{f_3} - \delta\frac{f_1}{f_2} = A_2 + \delta\left(\frac{f_1}{f_3} - \frac{f_1}{f_2}\right)$$

…….

$$A_n^* = B_n + A_n - B_n^* = Ar_n\left(\frac{f_n}{f_{n+1}}-1\right) + A_n - Ar_n^*\left(\frac{f_n}{f_{n+1}}-1\right)$$

$$A_n^* = Ar_n\left(\frac{f_n}{f_{n+1}}-1\right) + A_n - \left(Ar_n - \delta\frac{f_1}{f_n}\right)\left(\frac{f_n}{f_{n+1}}-1\right)$$

$$A_n^* = Ar_n\frac{f_n}{f_{n+1}} - Ar_n + A_n - Ar_n\frac{f_n}{f_{n+1}} + Ar_n + \delta\frac{f_1}{f_{n+1}} - \delta\frac{f_1}{f_n}$$

$$A_n^* = A_n + \delta\frac{f_1}{f_{n+1}} - \delta\frac{f_1}{f_n} = A_n + \delta\left(\frac{f_1}{f_{n+1}} - \frac{f_1}{f_n}\right)$$

Despejando los B^* en función de B y δ

$$B_1^* = B_1 - \delta\left(\frac{f_1}{f_2}-1\right),\ B_2^* = B_2 - \delta\left(\frac{f_1}{f_3} - \frac{f_1}{f_2}\right),\ \ldots\ldots\ B_n^* = B_n - \delta\left(\frac{f_1}{f_{n+1}} - \frac{f_1}{f_n}\right)$$

El slack disponible que se debe recuperar será la diferencia entre el slack gastado y el slack que se tendría que haber gastado

$$K_r = \left(B_1 + B_2 + \ldots\ldots + B_n\right) - \left(B_1^* + B_2^* + \ldots\ldots + B_n^*\right) = \delta\left(\frac{f_1}{f_2}-1\right) + \delta\left(\frac{f_1}{f_3} - \frac{f_1}{f_2}\right) + \ldots\ldots + \delta\left(\frac{f_1}{f_{n+1}} - \frac{f_1}{f_n}\right)$$

$$K_r = \delta\frac{f_1}{f_2} - \delta + \delta\frac{f_1}{f_3} - \delta\frac{f_1}{f_2} + \ldots\ldots + \delta\frac{f_1}{f_{n+1}} - \delta\frac{f_1}{f_n} = \delta\frac{f_1}{f_{n+1}} - \delta$$

$$K_r = \delta\left(\frac{f_1}{f_{n+1}} - 1\right) \square$$

4.1 Ejemplo

Para demostrar la ventaja se realiza el siguiente ejemplo. Una tarea con tiempo de ejecución de 10 (peor tiempo de ejecución) en una frecuencia de 1000, se divide en tres partes que son ejecutadas en distintos niveles de frecuencia (800, 600, 400). El peor tiempo de ejecución no ocurre y sólo necesita un tiempo de ejecución de 8 para ser ejecutada, desde el punto de vista de la frecuencia 1000. Pero esto sólo se sabe cuando la tarea realiza su exit. Por lo tanto, para cada nivel, el cálculo de cuánto es el slack que se necesita para bajar de frecuencia se realiza con el peor tiempo de ejecución restante. Entonces el slack gastado con la distribución que se muestra en las Tabla 1 es de 8,1666. Pero el que se tendría que haber gastado, como muestra la Tabla 2, es de 5.1666, dado que la tarea tuvo un tiempo de ejecución de solo 8. Con la ecuación (2) obtenemos cuánto es el slack que se puede devolver a todas las tareas de prioridad menor que la tarea τ_i, sin necesidad de recalcular el slack de dichas tareas. Para el ejemplo de este caso, la diferencia entre el slack gastado y el necesario es de 3, por lo tanto se había gastado más del 58%. La carga computacional es sólo el cálculo de la formula de recuperación, que es de orden constante y la recarga de los contadores que es de orden lineal.

f	A_n	Ar_n	B_n
1000/800	2	10	2.50
800/600	2	8	2.6667
600/400	1(6)	6(1)	3

Tabla 1

f	A_n	Ar_n	B_n
1000/800	2.5	8	2
800/600	2.8334	5.5	1.8334
600/400	2.6667	2.6667	1.3334

Tabla 2

5 CONCLUSIONES

En el ejemplo presentado en la sección 4 se demuestra claramente que la utilización de este modelo beneficia sustancialmente al ahorro de energía, al utilizar sólo la cantidad de slack precisa y así poder disponer de este slack recuperado para la ejecución de otras tareas en frecuencias menores. Hay que tener en cuenta que los saltos de frecuencias son discretos, por lo cual pequeñas cantidades de slack pueden tener una relevancia sustancial al determinar sobre qué nivel se podrá ejecutar una tarea.

El método propuesto en este trabajo permite una utilización eficiente de la energía del sistema debido a su baja complejidad de implementación en tiempo de corrida.

Referencias

[1] J. A. Stankovic, "Misconceptions About Real-Time Computing: A Serius Problem for Next-Generations Systems," *IEEE Computer*, vol. Octubre, pp. 10-19, 1988.

[2] C. L. Liu and J. W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment," *Journal of the ACM*, vol. 20, pp. 46-61, 1973.

[3] J. Y. T. Leung and J. Whitehead, "On the Complexity of Fixed-Priority Scheduling of Periodic, Real Time Tasks.," *Perf. Eval. (Netherlands)*, vol. 2, pp. 237-250, 1982.

[4] R. I. Davis, "Dual Priority Scheduling: A Means of Providing Flexibility in Hard Real-Time Systems," Department of Computer Science, University of York, York, England 1995.

[5] M. Weiser, B. Welch, A. Demers, and S. Shenker, "Scheduling for Reduced CPU Energy," presented at 1st Symposium on Operating Systems Design and Implementation, Monterey, California, EUA, 1994.

[6] N. C. Audsley, A. Burns, M. F. Richarson, and A. J. Wellings, "Hard Real-Time Scheduling: The Deadline Monotonic Approach," presented at Proceedings 8th IEEE Workshop on Real-Time Operating Systems and Software, Atlanta, GA, USA, 1991.

[7] S. Saewong and R. Rajkumar, "Practical Voltage-Scaling for Fixed-Priority RT-Systems," presented at 9th IEEE Real-Time and Embedded Technology and Applications Symposium, Toronto, Canada, 2003.
[8] Y. Shin and K. Choi, "Power conscious fixed priority scheduling for hard real-time systems," presented at 36th Design Automation Conference, 1999.
[9] Y. Liu and A. K. Mok, "An integrated approach for applying dynamic voltage scaling to hard real-time systems," presented at Real-Time and Embedded Technology and Applications Symposium, The 9th IEEE, 27-30 May, 2003.
[10] C. M. Krishna and Y. a.-H. Lee, "Voltage-Clock Adaptive Scheduling Techniques for Low Power in Hard Real-Time Systems," presented at Real-Time Technology and Applications Symposium, 2000.
[11] Ala´Qadi, S. Goddard, and S. Farritor, "A dynamic voltage scaling algorithm for sporadic tasks," presented at Real-Time Systems Symposium (RTSS), 24th IEEE, 3-5 Dec, 2003.
[12] W. Kim, J. Kim, and S. L. Min, "A Dynamic Voltage Scaling Algorithm for Dynamic-Priority Hard Real-Time Systems Using Slack Time Analysis," presented at Conference on Design, Automation and Test in Europe, Washington, DC, EUA, 2002.
[13] M. A. Moncusí, A. Arenas, and J. Labarta, "Improving Energy Saving in Hard Real Time Systems via a Modified Dual Priority Scheduling," vol. 29, pp. 19-24, 2001.
[14] F. Yao, A. Demers, and S. Shenker, "A Scheduling Model for Reduced CPU Energy," presented at Foundations of Computer Science, 36th Annual Symposium, 1998.
[15] I. Hong, D. Kirovski, G. Qu, M. Potkonjak, and M. B. Srivastava, "Power Optimazation of Variable-Voltage Core-Based Systems," *IEEE Transactions On Computer- AIDED Design of Integrated Circuits and Systems*, vol. 18, pp. 1702-1714, 1999.
[16] T. Ishihara and H. Yasuura, "Voltage Scheduling Problem for Dynamically Variable Voltage Processors," presented at International Symposium on Low-Power Electronics and Desing, Monterey, California, EUA, 1998.
[17] W. Yuan and K. Nahrstedt, "Energy-Efficient Soft Real-Time CPU Scheduling for Mobile Multimedia Systems," presented at 20th Symposium on Operating Systems Principles, Bolton Landing, Nova York, EUA, 2003.
[18] R. I. Davis, K. W. Tindell, and A. Burn, "Scheduling Slack Time in Fixed-Priority Preemptive Systems," *Proceedings of the Real Time System Symposium*, pp. 222-231, 1993.
[19] R. I. Davis, "Approximate Slack Stealing Algorithms for Fixed Priority Pre-Emptive Systems," Real-Time Systems Research Group, University of York, York, England 1994.
[20] S. Ramos-Thuel and J. P. Lehoczky, "On-Line Scheduling of Hard Deadline Aperiodic Tasks in Fixed-Priority Systems," presented at Real-Time Systems Symposium, 1993.
[21] J. M. Urriza and J. D. Orozco, "Métodos Rápidos para el Cálculo del Slack Stealing Exacto y Aproximado para Aplicaciones en Sistemas Embebidos," Dep. de Ing. Eléctrica y Computadoras, Universidad Nacional del Sur, Argentina., Bahía Blanca 2004.
[22] J. D. Orozco, R. M. Santos, J. Santos, and R. Cayssials, "Taking advantage of priority inversions to improve the processing of non-hard real-time tasks in mixed systems," presented at WIP 21st IEEE Real-Time Systems Symposium, 2000.