Evaluación de Sistemas Operativos de Tiempo Real sobre Microcontroladores

Medina Santiago[1], Pi Puig Martín[1], Paniego Juan Manuel[1], Dell'Oso Matías[1], Romero Fernando[1], De Giusti Armando[1,2], Tinetti Fernando G.[1,3]

[1]Instituto de Investigación en Informática LIDI (III-LIDI), Facultad de Informática, Universidad Nacional de La Plata, 50 y 120 2do piso, La Plata, Argentina.
[2]CONICET – Consejo Nacional de Investigaciones Científicas y Técnicas
[3]CIC – Comisión de Investigaciones de la Pcia. de Buenos Aires

{smedina, mpipuig, jmpaniego, mdelloso, fromero, degiusti, fernando}@lidi.info.unlp.edu.ar

Abstract. En el presente trabajo se presentan mediciones sobre diferentes sistemas operativos que soportan las características de tiempo real, instalados sobre un sistema de microcontrolador. Dichas evaluaciones caracterizan los posibles tiempos de respuesta, es decir, un límite que determina el tipo de aplicación en la cual pueden ser utilizados, dentro del marco del desarrollo de aplicaciones de Sistemas Embebidos de Tiempo Real. Una de las principales métricas es el tiempo de latencia, el cual representa el tiempo que transcurre desde que se produce una entrada hasta que se emite la salida de respuesta. Luego, no solo se evalúa este tiempo, sino la variabilidad del mismo, ya que en tiempo real es fundamental el determinismo. Si bien no es la única condición que debe cumplir un sistema operativo para ser considerado de tiempo real, es un requisito necesario. Los tiempos de latencia y su variabilidad, determinan las cotas de requerimientos temporales que brinda el sistema.

Keywords: Tiempo Real, Microcontroladores, Sistemas Operativos, Sistemas Embebidos, Latencia.

1 Introducción

Los Sistemas de Tiempo Real (STR) [1] [2] [3] [4] son aquellos que permiten proveer en forma garantizada un servicio dentro de un intervalo de tiempo de respuesta limitado. Esta restricción exige un cuidadoso diseño tanto del hardware como del software.

Los STR están conformados por un conjunto de dispositivos electrónicos acompañados de un Sistema Operativo de Tiempo Real (SOTR). En un enfoque ascendente, estos sistemas pueden estar implementados mediante:

- Sistema electrónico, o sea solo hardware, diseñado especialmente para dar solución a los requerimientos de control.
- Sistemas electrónicos programables. En general, el hardware está diseñado para un conjunto generalizado de requerimientos, y se personaliza a partir del software. Solo se utiliza un programa con instrucciones para lograr la funcionalidad deseada, sobre un hardware que no cuenta con sistema operativo. También son conocidos como entorno de metal desnudo (Bare-metal environment). Son sistemas en los que una máquina virtual se instala directamente en el hardware en lugar de sobre un sistema operativo host [6].
- Sistema electrónico con un Sistema Operativo de Tiempo Real (SOTR) con un programa que logre la funcionalidad deseada. Dentro de estos sistemas los hay de complejidad variada, desde microcontroladores con un SOTR elemental (FreeRTOS, MQX), hasta complejas computadoras corriendo SOTR mayormente derivado de UNIX (QNX) o Linux (RTLinux, RTAI, Linux RT-Preempt).

Las ventajas en los sistemas más complejos, con SOTR, son:

- Tiempo de desarrollo mínimo: debido a que se utiliza el soporte proporcionado por el SOTR.
- Flexibilidad: ya que puede cambiarse fácilmente el software.
- Consecuencia del punto anterior es que estos sistemas son de fácil mantenimiento.
- Cumplir con compromisos temporales estrictos: brinda al programador herramientas para que, una vez ocurrido un evento, la respuesta ocurra dentro de un tiempo acotado. Esto implica que una aplicación mal diseñada puede fallar en la atención de eventos aún cuando se use un RTOS.
- Manejo del tiempo: el RTOS proporciona manejo de temporizadores y esperas.
- Tarea Idle: cuando ninguna de las tareas requiere del procesador, el sistema ejecuta una tarea llamada idle u ociosa, que permite contabilizar el nivel de ocupación del CPU, poner al mismo en modo de bajo consumo o correr cualquier tarea que pudiera ser de utilidad para el sistema cuando no debe atender ninguno de sus eventos.
- Multitarea: simplifica la programación de sistemas con varias tareas.
- Escalabilidad: al tener ejecución concurrente de tareas se pueden agregar más, teniendo la precaución de insertarlas correctamente en el esquema de ejecución del sistema.
- Mayor reutilización de código: si las tareas se diseñan bien, con nula o mínima dependencia, es más fácil incorporarlas a otras aplicaciones.

Entre las desventajas se pueden nombrar:

- Mayor consumo: entre las diferentes aplicaciones de STR se encuentran sistemas móviles como también desconectados de la red eléctrica, que utilizan baterías para el suministro de energía. Su autonomía dependerá de su consumo.
- Mayor probabilidad de fallas: por contener mayor cantidad de componentes, electrónicos o de software, un STR requiere de funcionamiento continuo y sin desperfectos, por lo que muchas veces se prefiere sistemas lo más simples posibles.
- Mayores latencias: todo componente agregado, principalmente de software, implica demoras debido a su ejecución (ejemplo: planificador de tareas).

En este trabajo se aborda la medición de latencia, es decir, el tiempo que se demora entre la ocurrencia de un evento y la ejecución de la primera instrucción en el código del programa que da servicio a la interrupción.

Conforme se asciende en la complejidad del procesamiento, los elementos intervinientes suelen incrementar esta métrica. En este trabajo se analizan los tiempos de latencia de un sistema de desarrollo TWR-K70 que forma parte de la familia Kinetis creada por Freescale. Las diferentes muestras obtenidas sin SOTR se comparan con las evaluaciones sobre el mismo sistema pero utilizando los SOTR FreeRTOS y MQX.

Un Sistema de Tiempo Real puede requerir tiempos de respuesta desde el orden de los segundos, al orden de los milisegundos o microsegundos. La corrección de un sistema de tiempo real, a diferencia de los sistemas convencionales, involucra tanto a las salidas que produce como al tiempo que demora en producirlas. Si no se satisfacen estas restricciones temporales en las respuestas, se arriesga a la falla completa del sistema. De allí la necesidad de conocer las latencias que tengan las plataformas de hardware y SOTR sobre las cuales se piensa desarrollar una determinada solución.

2 Objetivos y metodología

Por tratarse de sistemas embebidos interactuando con el medio físico, los eventos que se produzcan en este medio deben generar respuestas del sistema. Estas respuestas deben ser correctas y emitidas dentro de los límites temporales. La demora en producirlas, también llamada latencia, debe ser conocida. Es objetivo de este trabajo obtener una medida de dicha latencia, tanto utilizando un SOTR como sin él (Bare-metal). La metodología consiste en generar un evento, tomar la medida de tiempo en el que se produjo, y medir nuevamente cuando se elabora la respuesta en la interrupción generada. La diferencia entre estas dos medidas determina el tiempo de latencia. El proceso que se ejecuta como respuesta al evento de una interrupción se lo conoce como Rutina de Servicio de Interrupción (en inglés, Interrupt Service Routine, ISR).

Como pueden seguir produciéndose eventos de interrupción en forma periódica, situación típica en los SOTR, es deseable que dicha ISR se ejecute en el menor tiempo posible. Por tanto, en general, se implementa la respuesta a la interrupción en dos partes:

una dentro de la ISR, lo más pequeña posible, y el resto en una rutina diferida. En este trabajo se analiza la metodología para implementar esta técnica.

2.1 Experimentación

Para llevar a cabo las pruebas, se trabajó con una placa de desarrollo propietaria de Freescale utilizando el IDE Kinetis Design Studio. Se realizaron experimentos sin SOTR y con SOTR, empleándose dos de ellos, uno de código libre y el otro propietario.

El método de medición consistió en tomar el instante de inicio antes de interrumpir al sistema y el tiempo final al captar el evento y atender la interrupción. Para ello, se generó un pulso por un determinado puerto del sistema, el cual se configuró para desencadenar una interrupción al momento de recibir el evento. Esta situación se repitió varias veces para lograr mejores aproximaciones del tiempo final.

De igual manera, los tiempos obtenidos se enviaron por puerta serie, mediante la placa anexa TWR-SER, hacia un host local para realizar el posterior tratamiento de los datos.

Por otra parte, se realizaron diferentes pruebas sobre casos dónde parte de la tarea de elaborar la respuesta se realiza en forma diferida, con lo que el planificador del SOTR interviene de manera directa. Dichas pruebas fueron ejecutadas sobre el SOTR FreeRTOS.

Las tres situaciones contempladas fueron las siguientes:

- Caso A: La tarea principal genera el pulso de salida, el sistema capta el evento generado y llama al manejador de la interrupción, donde se señaliza mediante un semáforo a la tarea principal que, posteriormente, detiene el timer.
- Caso B: La tarea principal genera el pulso de salida, el sistema capta el evento generado y llama al manejador de la interrupción, donde se señaliza mediante un semáforo a una tarea secundaria que, posteriormente, detiene el timer. La prioridad de la tarea principal es igual a la prioridad de la tarea secundaria.
- Caso C: La tarea principal genera el pulso de salida, el sistema capta el evento generado y llama al manejador de la interrupción, donde se señaliza mediante un semáforo a una tarea secundaria que, posteriormente, detiene el timer. La prioridad de la tarea principal es menor a la prioridad de la tarea secundaria.

2.2 Hardware utilizado

Se utilizó un kit de desarrollo TWR-K70 que forma parte de la familia Kinetis creada por Freescale [12]. Está compuesto por un microcontrolador ARM-CortexM4 de 32bits, con una frecuencia de Reloj de 120Mhz. Cuenta con memoria SRAM de 1GB, memoria Flash de 2GB, Acelerómetro, cuatro Led's de uso general, cuatro sensores touch pad, potenciómetro, slot para tarjeta Micro SD, Puerto USB para programación y debugging.

La placa permite la programación de una gran diversidad de aplicaciones ya que tiene un hardware muy variado y permite agregar placas de complemento conformando una estructura de torre. Dentro del hardware adicional, existen módulos de comunicación, módulos de sensado y módulos multimedia. Cabe aclarar que, para el desarrollo del proyecto, se utilizó además el modulo de comunicación TWR-SER, que permite la conexión a una interfaz Ethernet y una Serie.

2.3 Sistemas Operativos Evaluados

Para llevar a cabo las pruebas se emplearon los SOTR FreeRTOS y MQX. Ambos son denominados Sistemas Operativos Embebidos, en los cuales hay una fusión entre el Sistema Operativo y la Aplicación.

2.3.1 FreeRTOS

FreeRTOS [7] [8] [9] [10] [11] es un kernel de sistema operativo de tiempo real para sistemas embebidos que se ha vuelto muy popular, ha sido portado a 35 arquitecturas de microcontroladores y plataformas de hardware, incluso a computadoras personales. Se distribuye bajo licencia GPL con una excepción: permite que el código propietario de los usuarios siga siendo código cerrado, manteniendo el núcleo en sí como código abierto, lo que facilita el uso de FreeRTOS en aplicaciones propietarias.

FreeRTOS está diseñado para ser simple, el kernel está formado por un conjunto de archivos en lenguaje C. A su vez, en algunas arquitecturas, incluye algunas líneas de código ensamblador, mayormente en las rutinas del planificador.

FreeRTOS provee:

- Poco uso de memoria
- Bajo nivel de sobrecarga
- Ejecución rápida
- Métodos para múltiples hilos o tareas
- Mutexes y semáforos
- Timers de software
- Reducción del tick para aplicaciones de bajo consumo.
- Prioridades para las tareas
- Cuatro esquemas de asignación de memoria
- Es de código abierto: está bajo continuo desarrollo, y no hay costo de implementación.
- Amplia comunidad de usuarios: ayuda a resolver problemas que surgen.

El método de múltiples hilos cambia tareas en función de la prioridad y un esquema de planificación round-robin. Luego, el planificador puede ser configurado para operar en

forma apropiativa o colaborativa. Las tareas tienen una prioridad de ejecución, donde 0 es la menor prioridad (se recomienda usar referencias a tskIDLE_PRIORITY +1,+2, etc).

2.3.2 MQX

Es un sistema operativo de tiempo real propietario pero de distribución gratuita. Tiene dos partes:

- PSP: Plataform Support Package, es el subsistema de MQX que está relacionado al core sobre el que va a correr, ej: ARM, ColdFire, PowerPC,etc.
- BSP: Board Support Package, es el subsistema de MQX que contiene el código para soportar una placa determinada de un procesador determinado. Este código tiene los drivers de los periféricos disponibles en una placa dada.

Este sistema operativo provee:

- Comportamiento de tiempo real.
- Tareas.
- Múltiples hilos.
- Drivers para dispositivos.
- Comunicaciones y sus correspondientes pilas de protocolo.
- Kernel multitarea con programación preventiva y respuesta de interrupción rápida.
- Comunicación inter-tarea y sincronización.
- Sistema de archivos.
- Se puede configurar tan sólo 6 KB de ROM, incluyendo núcleo, interrupciones, semáforos, colas y administrador de memoria.
- Incluye pila TCP/IP (RTC).
- Integrado sistema de archivos MS-DOS (MFS).
- Pila de host/dispositivo USB.
- Soporte disponibles para varias plataformas, incluyendo Kinetis, ColdFire, Vybrid, i.MX y Power Architecture.

2.4 Sistema de Pruebas

Para obtener los tiempos de latencia se utilizaron dos herramientas diferentes:

- Reloj del sistema: Se utiliza el reloj del core ARM Cortex M4, denominado Systick. Es un contador decreciente de 24bits que oscila a una frecuencia de entrada de 24MHz. Luego, su configuración se realiza a través de cuatro registros del procesador.
- Componente “TimerUnit” de Processor Expert (PE): En dicho método, se utiliza la componente TimerUnit_LDD propia del sistema procesador experto que provee Freescale. La misma hace uso del módulo FlexTimer de la placa de desarrollo. El FlexTimer es un contador de 16bits, el cual permite configurar el sentido y la

frecuencia de la cuenta. En este caso, se personalizó el módulo para utilizar una frecuencia de 20.9MHz. Su control se realiza a través de funciones de alto nivel de PE.

La ventaja de usar el reloj del core Systick es que se puede unificar dicho método sobre los tres escenarios planteados (BareMetal, FreeRTOS, MQX).

Por otra parte, el método que utiliza el componente de PE, es solo aplicable sobre BareMetal y sobre FreeRTOS, pero no así en MQX. Esto se debe a que Freescale MQX no provee soporte para la creación de proyectos de Processor Expert sobre la placa TWR-K70.

De todas maneras, el tiempo medido utilizando el método de PE corroboró el resultado obtenido en las pruebas con Systick.

3 Resultados

En la Tabla 1 se pueden observar los resultados obtenidos para cada uno de los escenarios.

Tabla 1. Tiempos de latencia

Tiempo Bare-metal	6.9 μs
Tiempo FreeRTOS	6.9 μs
Tiempo MQX	9.0 μs

Como se puede observar, el tiempo de latencia correspondiente a la placa de desarrollo en un entorno Bare-metal es realmente pequeño. Este valor es la referencia principal para las posteriores comparaciones con los SOTR. Además, el tiempo obtenido decreta un límite impuesto por la placa para el desarrollo de aplicaciones que requieran respuestas en intervalos menores al obtenido.

En el caso de FreeRTOS, no se percibe ningún tiempo extra añadido por la utilización del mismo. Esto se debe a que el planificador del SOTR no se ejecuta, ya que en la rutina de interrupción propia del evento no se realiza ningún llamado a alguna API de FreeRTOS. En conclusión, el manejador de la interrupción generada por el pulso, no introduce overhead alguno en el tiempo final de latencia, resultando el mismo idéntico al obtenido en el escenario sin SOTR.

Sin embargo, al evaluar el sistema de microcontrolador sobre el SOTR de Freescale, se advierte un mínimo incremento del tiempo de latencia. Al utilizar las interrupciones estándares que provee MQX, se genera un tiempo extra de aproximadamente 2 μs [5]. Asimismo, al igual que en el caso anterior, el código correspondiente al manejador de la interrupción no realiza llamado alguno a una determinada API del SOTR. En caso de requerirse tiempos de latencia menores al impuesto por MQX, se debe recurrir a interrupciones en modo kernel, las cuales son soportadas completamente por el SOTR de Freescale. Para ello, solo basta con instalar la interrupción al evento requerido con

funciones exclusivas para el kernel que provee MQX. De todas maneras, si ese fuera el caso, la aplicación no dispondría de ciertos beneficios que otorgan las interrupciones clásicas de MQX, como por ejemplo, utilizar los flags y eventos del SOTR como una API para otros procesos MQX.

Como se analizó anteriormente, el manejador de la interrupción de ambos SOTR no ejecuta ningún llamado a una determinada API del mismo. En consecuencia, el tiempo de latencia varía mínimamente o no lo hace, respecto de la utilización de un escenario de placa desnuda.

Luego, se realizaron diferentes pruebas donde se hace un llamado explícito a ciertas funciones del SOTR para, de esta manera, observar el posible overhead introducido por el mismo.

Por tanto, en la Tabla 2 se detallan los resultados obtenidos para el escenario donde la placa TWR-K70 incorpora el SOTR FreeRTOS, con la modificación de que se utiliza una sección diferida en la gestión del evento de entrada.

Tabla 2: Resultados obtenidos utilizando la sección diferida sobre FreeRTOS

Caso	T. mín.	T. max.
A	22 µs	32 µs
B	55 µs	75 µs
C	40 µs	50 µs

Como consecuencia del llamado explícito a una determinada API del SOTR, los tiempos de respuesta se incrementan sustancialmente.

En el primer caso, el manejador del evento desbloquea el semáforo que espera la tarea principal, generando así un overhead propio del llamado a la API de semáforos de FreeRTOS. Como se puede observar, este procesamiento diferido agrega un tiempo extra de entre 15 y 25 µs al tiempo obtenido en la Tabla 1.

Luego, en el caso B, la interrupción desbloquea el semáforo que espera una tarea secundaria, diferente a la tarea principal que generó el pulso de entrada. No obstante, las dos tareas presentan la misma prioridad. En esta situación, el tiempo obtenido se incrementa en mayor medida al igual que su variabilidad. Esto se debe a que el SOTR genera diferentes situaciones al momento de decidir cuál de las dos tareas se ejecuta, luego de terminada la gestión de la interrupción.

Por último, el caso C es idéntico al anterior, con la diferencia de que la tarea secundaria posee una prioridad mayor a la de la tarea principal. Por tanto, el tiempo medido disminuye en comparación a la situación anterior pero se incrementa respecto al caso A. Esto se debe a que, al finalizar la ejecución de la rutina de interrupción, FreeRTOS evalúa si la tarea desbloqueada por el semáforo tiene mayor prioridad que la tarea anteriormente interrumpida. Si esta condición se cumple, se ejecuta el planificador de tareas introduciendo un overhead producido por la utilización de una API del SOTR. Por otra

parte, la variabilidad es idéntica que en el primer escenario, ya que no existen tareas con la misma prioridad.

4 Conclusión

El propósito principal de este trabajo consistió en verificar, mediante el tiempo de latencia, el desempeño de un STR, empleando dos SOTR diferentes y comparando los resultados obtenidos en dichos sistemas con los obtenidos ejecutando aplicaciones directamente sobre el hardware en la modalidad Bare-metal. En las diferentes pruebas se constató que los tiempos de latencia sobre Bare-metal y FreeRTOS son iguales si el procesamiento de la respuesta se realiza dentro de la ISR, donde el SOTR no introduce overhead alguno. En el caso de ejecutarse una rutina diferida con llamadas a API's del SOTR, aparecen tiempos extras con variabilidades mayores, debido a las tareas que lleva a cabo el SOTR, principalmente por el planificador del mismo.

En otro orden, el SOTR propietario MQX, introduce un overhead mínimo en el tiempo de latencia del sistema, siempre y cuándo se haga uso de las, generalmente utilizadas, interrupciones clásicas.

5 Bibliografía

1. N.C. Audsley , A. Burns , M. F. Richardson , A. J. Wellings: Hard Real-Time Scheduling: The Deadline-Monotonic Approach. Proc. IEEE Workshop on Real-Time Operating Systems and Software (1991).
2. L. Buhr. "An Introduction to Real Time Systems". Prentice Hall (1999).
3. A. Burns, A. J. Wellings: Designing hard real-time systems. Proceedings of the 11th Ada-Europe international conference on Ada. ISBN:0-387-55585-4 (1992).
4. A. Burns & A. Wellings: Real-Time Systems and Programming Languages. Addison Wesley, ISBN 90-201-40365-x.
5. L. Prokop, Freescale Semiconductor Application Note. Motor Control Under the Freescale MQX Operating System.
6. A. Silberschatz, Peter Galvin, Greg GAGNE. Operating System Concepts.
7. R. Barry. Using the FreeRTOS Real Time Kernel.
8. C. Becker. RTOS en sistemas embebidos, disponible en http://ieee.eie.fceia.unr.edu.ar/PDF_RTOS.pdf
9. A. Celery. Sistemas Operativos de Tiempo Real, disponible en http://www.sase.com.ar/2011/files/2010/11/SASE2011Introduccion_RTOS.pdf
10. FreeRTOS, disponible en http://en.wikipedia.org/wiki/FreeRTOS
11. FreeRTOS, disponible en http://www.freertos.org/
12. TWR-K70F120M: Kinetis Tower System Module, disponible en http://www.freescale.com/webapp/sps/site/prod_summary.jsp?code=TWR-K70F120M