Análisis de Rendimiento de dos Sistemas Operativos en Tiempo Real implementados en dos Arquitecturas de Hardware para la selección del Sistema Embebido que soportará el Software Command And Data Handling de la Misión Satelital Libertad 2

Diana Catalina Camargo Villamil
Universidad Sergio Arboleda, Bogotá, Bogotá D.C, Colombia, dianac.camargo@correo.usa.edu.co

Jonattan Andrés Andrade Mayorga
Universidad Sergio Arboleda, Bogotá, Bogotá D.C, Colombia, jonattan.andrade@correo.usa.edu.co

Tutor: Freddy Alexander Díaz González
Universidad Sergio Arboleda, Bogotá, Bogotá D.C, Colombia, freddy.diaz@correo.usa.edu.co

ABSTRACT

The Satellite Libertad 2 of the University Sergio Arboleda will be develop under the Cubesat standard, this standard contains physical constraints such as size, weight and shape so it is necessary to use Embedded Systems for development the control subsystem. Additionally required security and reliability features to ensure correct operation and mitigate the risks of the mission. The real-time operating systems (RTOS) provide these features, however, their performance varies with the hardware being used. With the offer of hardware and software on the market How must I select the Hardware and the RTOS that will support the control software of the satellite? This paper proposes the analysis of parameters for two RTOS implemented in two microcontrollers architectures, with the aim of providing tools to facilitate the selection of RTOS and the embedded system. In the results you can see that the performance of hardware, change depending of RTOS used. The satellite mission requirements such as power consumption, memory capacity, operating states among others, are decisive in the selection of the hardware architecture and the RTOS.

Keywords: Operating System, Embedded System, Hardware Architecture, System Resources, CubeSat

RESUMEN

El Satélite Libertad 2 de la Universidad Sergio Arboleda será desarrollado bajo el estándar CubeSat, este estándar contiene restricciones físicas como tamaño, peso y forma por lo cual es necesario el uso de Sistemas Embebidos para la implementación del subsistema de control, adicionalmente se requieren características de seguridad, robustez y fiabilidad que garanticen un correcto funcionamiento y mitiguen los riesgos de la misión. Los sistemas operativos en tiempo real (RTOS) proporcionan estas características, sin embargo, su rendimiento varía de acuerdo al Hardware que se utilice. Con la oferta de hardware y software en el mercado ¿Cómo se debe seleccionar el Hardware y el RTOS sobre el cual se implementará el software de control del satélite? Este artículo propone el análisis de parámetros para dos RTOS implementados en dos arquitecturas de microcontroladores, con el objetivo de brindar herramientas que faciliten la selección del Sistema Embebido. En los resultados analizados se observa que el rendimiento de un RTOS varía según el hardware utilizado. Los requerimientos de la misión satelital como consumo de energía, capacidad de memoria, estados de operación entre otros, son determinantes en la selección de la arquitectura de Hardware y en el RTOS que se implemente en ella.

Palabras claves: Sistema operativo, sistema embebido, arquitectura de Hardware, Recursos del sistema, CubeSat

1. INTRODUCCIÓN

La Universidad Sergio Arboleda dando continuidad a la carrera espacial iniciada el 27 de Abril de 2007 con el lanzamiento del primer pico satélite colombiano Libertad 1, ha puesto en marcha la misión satelital Libertad 2 que espera poner en una órbita LEO (Low Earth Orbit) un nano satélite tipo CubeSat para el año 2014.

Un satélite no debe requerir de ningún tipo de mantenimiento de hardware y muy poco de software, dado las condiciones que presenta su entorno de operación. El Nano Satélite a desarrollar requiere de características de seguridad, robustez y fiabilidad; estas características enmarcan a el satélite dentro del grupo de sistemas críticos de alta seguridad (Scholz, 2004). Debido a los requerimientos mencionados, el software de control del satélite debe implementarse sobre un sistema operativo en tiempo real (RTOS) en un sistema embebido, elementos que brindan robustez y seguridad a sistemas críticos (Laplante P. A., 2004).

El satélite Libertad 2 se encuentra enmarcado en el estándar CubeSat, este estándar reduce costos, tiempos de desarrollo de una misión Satelital y además facilita la puesta en órbita del satélite ya que es posible enviarlo como carga secundaria en el lanzamiento de satélites más grandes y complejos. Dentro de la comunidad de desarrolladores se comercializan gran parte de las unidades que componen el BUS principal del satélite, sin embargo uno de los principales objetivos del desarrollo de satélites académicos como los CubeSat, es el de adquirir todo el conocimiento relacionado con el desarrollo de las misiones satelitales (University, 2001).

Uno de los módulos que se desean desarrollar en la misión satelital Libertad 2 es el On Board Computer (OBC), este subsistema se debe implementar utilizando Sistemas Embebidos debido a las restricciones espaciales del estándar CubeSat y a las restricciones de energía propias del espacio. Sin embargo, no se han encontrado parámetros o herramientas que permitan la selección de los componentes de procesamiento necesarios en el desarrollo del OBC.

El conjunto del OBC y su respectivo Software de control, son el sub sistema Command and Data Handling (C&DH) de un satélite, este combina elementos software y hardware que deben soportar todas la funcionalidades de procesamiento, almacenamiento y control de otros subsistemas que componen el satélite. El C&DH es un sistema que debe garantizar el correcto funcionamiento y no debe poner en riesgo el desarrollo de la misión satelital.

Para el análisis e identificación de herramientas que faciliten la selección del RTOS y el Sistema Embebido (SE) que se implementará en el subsistema C&DH de la misión satelital Libertad 2 se propone tres protocolos de prueba enfocados a medir el desempeño y seguridad de la ejecución de dos RTOS implementados en dos diferentes SE. En esta investigación se evaluará el rendimiento de los RTOS Salvo y FreeRTOS combinados con las arquitecturas MSP430 y Cortex M3.

Este artículo se encuentra estructurado de la siguiente forma: la primera parte corresponde a la presente introducción, la segunda es una breve reseña del estándar cubesat, en la tercera parte se presentan los tipos de sistemas operativos disponibles para los microcontroladores, la cuarta expone los componentes de un RTOS, la quinta analiza los tipos de Scheduler, la sexta contiene las formas de optimización del código de un RTOS, la séptima presenta los RTOS seleccionados para el análisis, la octava parte presenta las arquitecturas de Microprocesadores estudiadas, la novena detalla el protocolo de pruebas, la décima presenta los resultados obtenidos en la implementación del protocolo de pruebas, el punto once contiene el análisis de resultados y finalmente se presentan las conclusiones y se propone el trabajo futuro.

2. ESTÁNDAR CUBESAT

El estándar Cubesat hace referencia a un tipo de pico satélites de órbita baja terrestre desarrollado por la Universidad de Calpoly que proporciona a universidades, instituciones y países la posibilidad de lanzar satélites al espacio, a un bajo costo (B, 2008). Los Cubesat son pequeños satélites con forma cúbica con determinadas características de tamaño (diametro de 10 cm) y peso (inferior a 1kg) entre otras, éstas características varían de acuerdo a la cantidad de unidades que tenga el pico satélite y al experimento que se lleve a cabo en la misión.

En la actualidad se ha incrementado el interés por el desarrollo de misiones satelitales y sus diferentes aplicaciones, la iniciativa CubeSat tiene como beneficio la reducción de costos de validación de prototipos y una amplia comunidad con desarrollos y contribuciones obtenidas de diferentes misiones, cuyo objetivo es viabilizar misiones espaciales desarrolladas por estudiantes y profesores (Diaz F. A., 2012).

En la Figura 1 y, Figura 2 se presentan algunas de las misiones espaciales regidas bajo estándar Cubesat, en donde se especifican los microprocesadores y el RTOS utilizados para la construcción de los pico satélite.

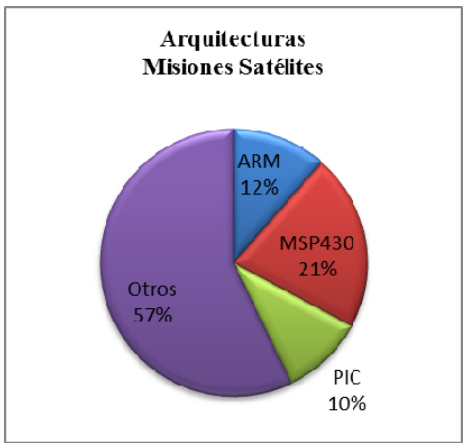

Figura 1: Arquitecturas en Misiones satelitales

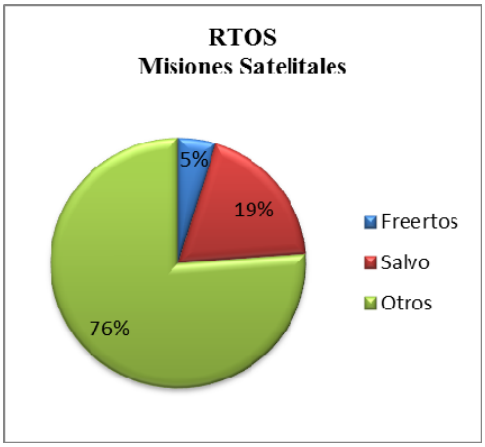

Figura 2: RTOS en Misiones satelitales

3. SISTEMAS OPERATIVOS EN TIEMPO REAL PARA MICROCONTROLADORES

Dadas las restricciones de tamaño y peso del estándar Cubesat para misiones Satelitales, se requiere de dispositivos pequeños que realicen el procesamiento de la información y satisfagan las necesidades del sistema sin emplear muchos recursos de memoria o energía. Estos dispositivos son conocidos como sistemas embebidos, entre ellos el Microcontrolador.

Con el avance de los microcontroladores de mayor capacidad de procesamiento se ha incrementado la complejidad del software que puede soportar (B, 2008), para la implementación de estas aplicaciones se usan herramientas como los RTOS para sistemas embebidos (Dick R. , 2003). Según William Stallings un sistema operativo es un programa que controla la ejecución de aplicaciones y procesos, y actúan como interfaz entre las aplicaciones y el hardware del sistema (Stallings, 2008).

Los sistemas operativos más comunes para sistemas embebidos se clasifican en dos tipos: abiertos y cerrados. Los primeros permiten examinar el código fuente de ejecución del sistema operativo, modificarlo y generar versiones alternas del mismo. Los sistemas operativos cerrados no permiten al desarrollador acceder el código del sistema operativo, sino que simplemente se usa de forma similar a una biblioteca (Victor Manuel).

Entre los sistemas operativos más implementados para sistemas embebidos se pueden encontrar FreeRtos, Salvo, TinyOS, UT Kernel y XMK OS (Tan, 2009).

4. COMPONENTES DE UN SISTEMA OPERATIVO EN TIEMPO REAL

Un sistema operativo en tiempo real (RTOS) responde a restricciones específicas de tiempo, asegura tiempos de respuesta para atención de eventos con un margen de error del orden de microsegundos y garantiza tiempos de ejecución para tareas y procesos. Los RTOS se encuentran conformados principalmente por dos componentes software: Dispatched y Scheduler. Estos métodos administran el orden de ejecución, los estados en que se encuentra cada tarea y de la asignación del procesador (CPU) a cada una. Las aplicaciones desarrolladas para ser implementadas sobre RTOS se dividen en métodos funcionales de procesamiento denominadas tareas, cada tarea

se ejecuta independientemente y la comunicación entre las tareas se realiza utilizando los servicios propios del sistema operativo (Barry, 2010).

Para la administración de las tareas y sus estados, un SO tiene dos tipos de planificador. El Dispatched es un planificador que se ejecuta en un periodo corto de tiempo y elije la siguiente tarea que se ejecutará. El scheduler o planificador a largo plazo, modifica el estado actual de una tarea dependiendo de eventos externos. En la Tabla 1 se presenta las políticas de planificación más comunes utilizadas en un RTOS

Tabla 1: Políticas de Planificación de un RTOS

Políticas de Planificación	
Round-Robin	SJF - Shortest Job First.
FIFO - First Input First Output	CFS - Completely Fair Scheduler
LIFO - Last Input First Output	SRT - Shortest Remaining Time
SPT - Shortest Process Time	

Un RTOS asigna estados a las tareas con el objetivo de planificar el orden de ejecución. Existen varios estados, dependiendo del RTOS que se esté utilizando. En la Figura 1 se muestran los estados más comunes asignados por un RTOS a una tarea.

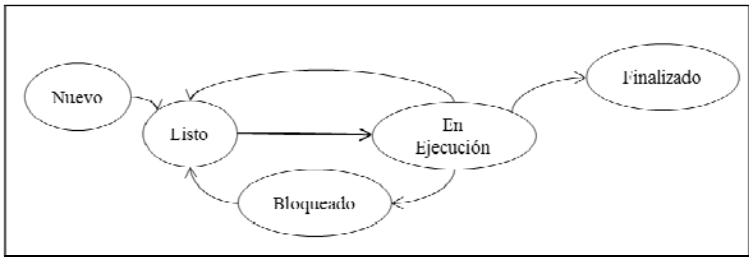

Figura 3: Estados de una Tarea

En el momento en el que se crea una tarea el Scheduler le asigna el estado Nuevo. El estado Listo corresponde a una tarea que está preparada para ser ejecutada por el procesador (CPU) . El estado en Ejecución corresponde a una tarea que tiene el control de la CPU. Una tarea entra al estado bloqueado cuando no puede ejecutarse hasta que ocurra un evento o llegue un dato. Por último, el estado finalizado corresponde a una tarea que ha terminado su ciclo en el sistema.

5. TIPOS DE SCHEDULER DE UN RTOS

Los RTOS se clasifican en dos grupos según el tipo de Scheduler o planificador implementado, Cooperativo o Preventivo. La utilización de cada planificador trae ventajas y desventajas notorias en el tiempo de ejecución, utilización de memoria RAM y ROM, entre otras.

Un RTOS cooperativo o con Scheduler cooperativo se caracteriza por ceder el control total del procesador (CPU) a la tarea que se encuentra en ejecución. El Scheduler solo podrá ejecutarse de nuevo si la tarea cede el control de la CPU. El riesgo de un fallo aumenta debido a que los errores de codificación pueden generar un bloqueo global en el sistema. Por otra parte, este Scheduler optimiza el uso de los recursos, como el tiempo de ejecución, ya que no se requiere almacenar ni restaurar las variables temporales utilizadas por la tarea.

Los sistemas operativos preventivos o con Scheduler preventivo hacen uso de interrupciones periódicas para ejecutar el planificador. En este caso, una tarea puede estar en estado de ejecución y sin necesidad que ésta termine su proceso o ceda el procesador, el planificador puede tomar el control de la CPU, pausar la tarea actual y elegir cuál es la siguiente tarea en estado Listo para ser ejecutada.

El uso del Scheduler preventivo minimiza los fallos por errores de codificación y garantiza el funcionamiento constante del sistema operativo. El uso de un sistema operativo preventivo es ideal para sistemas críticos con poca capacidad de mantenimiento. Sin embargo, este tipo de RTOS requiere una cantidad considerable de recursos, como memoria y tiempo de procesamiento, ya que al interrumpir una tarea de forma abrupta es necesario almacenar en memoria todos los datos temporales para garantizar una ejecución continua. En la Tabla 3 se presentan las principales características de los Scheduler cooperativo y preventivo.

Tabla 2: Características principales de los Scheduler

Cooperativo	Preventivo
Recibe la CPU de una tarea	Recibe la CPU de una interrupción
No requiere Stack	Requiere Stack para almacenar variables de la tarea interrumpida
Alto riesgo de bloqueo del sistema	Bajo Riesgo de bloqueo del sistema
Mayor predictibilidad	Menor predictibilidad
No utiliza memoria RAM para almacenar el contexto de una tarea	Mayor uso de memoria RAM para almacenar el contexto de una tarea
Los planificadores requieren poca memoria ROM y poco tiempo en ejecución porque sus tareas se simplifican en este tipo de RTOS	El Dispatched requiere más tiempo ya que necesita almacenar las variables temporales de la tarea que estaba en ejecución y debe recuperar el contexto de la nueva tarea a ejecutar

6. FORMAS DE OPTIMIZACIÓN

Además de las estrategias que pueden ser usadas en la codificación para reducir memoria, construir un código legible y optimizar recursos, esta investigación hará uso de compiladores optimizadores con el objetivo de mejorar la forma en que se utilizan los recursos, favoreciendo el rendimiento de los RTOS seleccionados para su análisis y comparación de acuerdo a los protocolos de prueba implementados.

Las estrategias de optimización identificadas que se pueden implementar en los compiladores son:

• Optimización de memoria: El resultado del código compilado es una ejecución que requiere menos tamaño en memoria pero consume más ciclos de máquina.

• Optimización de velocidad: El resultado del código compilado es una ejecución que consume más tamaño en memoria pero requiere menos ciclos de máquina

7. RTOS ANALIZADOS

Teniendo en cuenta la identificación de los RTOS utilizados en las misiones satelitales bajo el estándar CubeSat, el estudio del código interno, y la comparación de sistemas preventivos y cooperativos, fueron seleccionados los RTOS Salvo y FreeRTOS para el análisis e implementación de los casos de prueba.

Salvo es un sistema operativo en tiempo real desarrollado por la empresas Pumpkin. Este RTOS ha sido utilizado en misiones satelitales bajo el estándar CubeSat y se encuentra enfocado a sistemas embebidos de pocos recursos, para lo cual utiliza un planificador cooperativo

FreeRTOS es un sistema operativo en tiempo real con soporte para más de 33 arquitecturas de microcontroladores. Este RTOS utiliza un planificador preventivo, y es posible modificar o examinar su código fuente.

En la Tabla 4 se presentan las características principales de los sistemas operativos mencionados anteriormente.

Tabla 3: Características de los SO Salvo y FreeRTOS

Salvo	FreeRTOS
Planificador Cooperativo	Planificador Preventivo
Código Cerrado	Código Abierto
Multitarea	Multitarea
16 niveles de prioridades	Prioridades ilimitadas
Lenguaje ANSI C	Lenguaje ANSI C
Licenciado	Open Source
Tiempos del planificador irregulares	Planificador periódico
Almacenamiento de datos en la sección de Datos de la memoria RAM	Almacenamiento de datos en la sección de Heap de la memoria RAM
Pequeño footprint	6K-10K ROM footprint
Servicio de semáforos	Servicio de semáforos
Servicio de mutex	Servicio de mutex
Servicio de colas	Servicio de colas

8. ARQUITECTURAS DE MICROCONTROLADORES

Adicionalmente a la comparación de los RTOS, esta investigación evalúa las ventajas y desventajas del uso de una arquitectura específica. Para la elección de las dos arquitecturas a comparar se tuvo en cuenta la compatibilidad con cada uno de los RTOS seleccionados, el uso en misiones satelitales bajo el estándar CubeSat, consumo de energía, memoria RAM y ROM, y precio de los Microcontroladores.

La primera arquitectura elegida fue MSP430 de Texas Instruments, esta arquitectura ha sido utilizada en misiones satelitales tipo CubeSat junto con el RTOS Salvo, se caracteriza por ser de 16 Bits, bajo consumo de energía y utilizar una arquitectura Von Neumann.

La arquitectura Von Neumann contiene un único bus de datos para comunicar la memoria ROM y RAM, lo cual permite modificar el código fuente en tiempo de ejecución pero disminuye la eficiencia, ya que no es posible acceder a las dos memorias simultáneamente. En la Figura 2 se presenta la arquitectura Von Neumann.

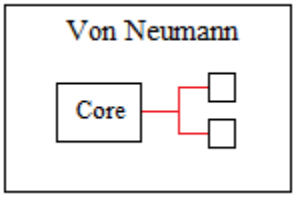

Figura 4: Arquitectura Von Neumann

La segunda arquitectura elegida fue la ARM CORTEX M3, esta arquitectura se caracteriza por ser de 32 bits, tener dos Stack Pointer, un Systick dedicado y utilizar una arquitectura Harvard.

La arquitectura Harvard se caracteriza por utilizar dos buses de datos para comunicarse paralelamente con la memoria RAM y ROM, esta arquitectura reduce tiempo en ejecución porque puede acceder simultáneamente a los dos tipos de memorias, sin embargo no puede modificar el código del programa. En la Figura 3 se observa la arquitectura Harvard.

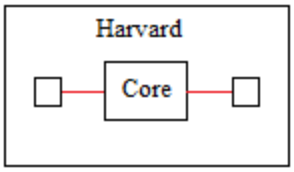

Figura 5: Arquitectura Harvard

9. PROTOCOLO DE PRUEBAS

Dada la implementación de tres recursos relevantes de un sistema, se realiza el siguiente protocolo de pruebas con el objetivo de analizar el rendimiento de las arquitecturas ARM-Cortex M3 y TI-MSP430. Para implementar el protocolo de pruebas se hace uso de los compiladores Keil Microvision y Rowley Crossworks, los cuales son compatibles con los RTOS Salvo y FreeRTOS. A continuación se presenta cada uno de los protocolos de prueba.

9.1 CASO DE PRUEBA 1: COLAS

Las colas son un recurso del sistema que permite el envío y/o recepción de datos entre tareas. Al crear una cola se debe definir la cantidad de elementos y el tamaño de los mismos. En este caso de prueba la cola fue definida con un tamaño de 10 datos enteros.

Este caso de prueba propone la implementación de dos tareas y una cola. La Tarea 1 guarda un dato en la cola y la Tarea 2 realiza la lectura del dato guardado y se establece los Breakpoints en puntos determinados que permiten realizar la medición de los parámetros identificados. En la Figura 5 se puede observar la estructura de este protocolo.

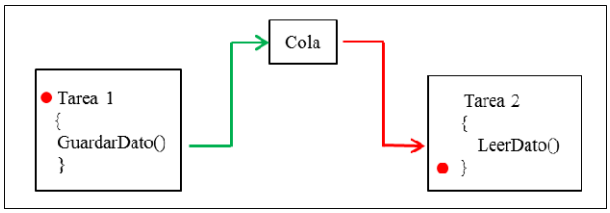

Figura 6: Protocolo de Pruebas Colas

9.2 CASO DE PRUEBA 2: MUTEX

El Mutex es utilizado para proteger un recurso del sistema. Cuando una tarea desea tener acceso un recurso protegido, esta debe tomar el Mutex para garantizar que ninguna otra tarea pueda tener acceso al mismo. Cuando la primera tarea ya no requiera el recurso, debe entregar en Mutex para permitir que otras tareas lo utilicen.

Se propone la implementación de dos tareas y un Mutex. La Tarea 1 accede a un recurso protegido tomando el Mutex y finaliza su proceso entregándolo, La tarea 2 realiza el mismo procedimiento de la tarea 1 después que esta entregue el Mutex. Se establece los Breakpoints en puntos determinados que permiten realizar la medición de los parámetros identificados. En la Figura 5 se puede observar la estructura de este protocolo.

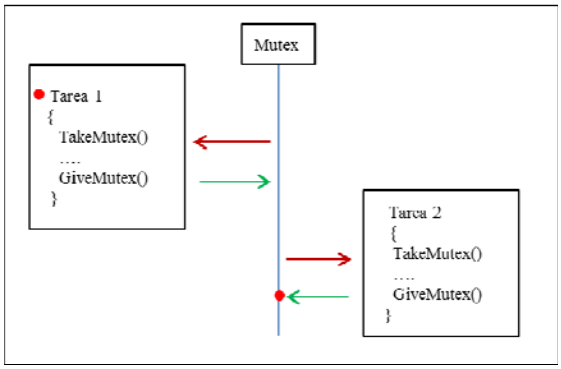

Figura 7: Protocolo de Pruebas Mutex

9.3 CASO DE PRUEBA 3: DELAY

Este recurso del sistema bloquea a la tarea durante el tiempo establecido, cediendo el control del procesador al sistema operativo. La tarea vuelve a estar en estado Listo al terminar este tiempo de espera.

Se propone la implementación de dos tareas, cada una con un delay interno y el establecimiento de los Breakpoints indicados en la Figura 4, que permiten realizar la medición de los ciclos de máquina y la memoria utilizada.

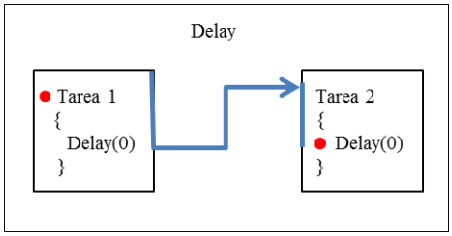

Figura 8: Protocolo de Pruebas Delay

10. RESULTADOS

A continuación se presentan los resultados obtenidos a través de la implementación de los casos de pruebas, en donde se especifica el RTOS, la arquitectura y la estrategia de optimización utilizada.

Como resultado de los casos de prueba se muestran en las siguientes tablas los Ciclos de Máquina y la cantidad de memoria ROM y RAM utilizadas por cada RTOS, arquitectura y estrategia de optimización implementada.

Tabla 4: Resultados de Caso de Prueba Colas

Colas					
RTOS	**Arquitectura**	**Estrategia de optimización**	**Ciclos de Máquina**	**Rom (KB)**	**Ram (KB)**
Salvo	Msp430	Tamaño	727,000	2,300	0,100
Salvo	Msp430	Velocidad	727,000	2,300	0,100
Salvo	ARM	Tamaño	5791,750	2,648	0,736
Salvo	ARM	Velocidad	5791,750	2,648	0,736
FreeRTOS	Msp430	Tamaño	1655,000	1,800	4,600
FreeRTOS	Msp430	Velocidad	1655,000	1,800	4,600
FreeRTOS	ARM	Tamaño	802,500	4,664	9,480
FreeRTOS	ARM	Velocidad	795,500	4,664	9,480

Tabla 5: Resultados de caso de prueba Mutex

Mutex					
RTOS	**Arquitectura**	**Estrategia de optimización**	**Ciclos de Máquina**	**Rom (KB)**	**Ram (KB)**
Salvo	Msp430	Tamaño	618,000	2,000	0,010
Salvo	Msp430	Velocidad	618,000	2,000	0,010
Salvo	ARM	Tamaño	2316,800	2,912	0,736
Salvo	ARM	Velocidad	2316,750	2,912	0,736
FreeRTOS	Msp430	Tamaño	887,000	1,800	4,600
FreeRTOS	Msp430	Velocidad	887,000	1,800	4,700
FreeRTOS	ARM	Tamaño	454,500	4,712	9,480
FreeRTOS	ARM	Velocidad	449,500	4,816	9,480

Tabla 6: Resultados de caso de prueba Delay

Delay					
RTOS	**Arquitectura**	**Estrategia de optimización**	**Ciclos de Máquina**	**Rom (KB)**	**Ram (KB)**
Salvo	Msp430	Tamaño	386	1,5	0,010
Salvo	Msp430	Velocidad	386	1,5	0,010
Salvo	ARM	Tamaño	1250	2,784	0,736
Salvo	ARM	Velocidad	1250	2,784	0,736
FreeRTOS	Msp430	Tamaño	251	1,8	3,2
FreeRTOS	Msp430	Velocidad	251	1,8	3,2
FreeRTOS	ARM	Tamaño	144	3,184	9,48
FreeRTOS	ARM	Velocidad	142	3,272	9,48

11. ANÁLISIS DE RESULTADOS

A continuación se presenta el análisis de resultados de los casos de prueba presentados previamente

11.1 CASO DE PRUEBA 1: COLAS

- Para ejecutar el caso de Prueba 1, la arquitectura MSP430 en el RTOS FreeRTOS requiere 127% más de ciclos de máquina que en el RTOS Salvo.
- Para ejecutar el caso de Prueba 1, la arquitectura MSP430 en el RTOS Salvo requiere 27% más de memoria ROM que el RTOS FreeRTOS.
- Para ejecutar el caso de Prueba 1, la arquitectura MSP430 en el RTOS FreeRTOS requiere 4500% más de memoria RAM que el RTOS Salvo.
- Para ejecutar el caso de Prueba 1, la arquitectura ARM Cortex M3 en el RTOS Salvo requiere 621% más de ciclos de máquina que FreeRTOS.
- Para ejecutar el caso de Prueba 1, la arquitectura ARM Cortex M3 en el RTOS FreeRTOS requiere 76.13% más memoria ROM que el RTOS Salvo.
- Para ejecutar el caso de Prueba 1, la arquitectura ARM Cortex M3 en el RTOS FreeRTOS requiere 1188% más memoria RAM que el RTOS Salvo.
- Para ejecutar el caso de Prueba 1, la arquitectura MSP430 en el RTOS Salvo requiere menos ciclos de máquina.
- Para ejecutar el caso de Prueba 1, la arquitectura MSP430 en el RTOS FreeRTOS requiere menos memoria ROM.

11.2 CASO DE PRUEBA 2: MUTEX

- Para ejecutar el caso de Prueba 2, la arquitectura MSP430 en el RTOS FreeRTOS requiere 43.52% más de ciclos de máquina que en el sistema operativo Salvo.
- Para ejecutar el caso de Prueba 2, la arquitectura MSP430 en el RTOS Salvo requiere 11% más de memoria ROM que el RTOS FreeRTOS.
- Para ejecutar el caso de Prueba 2, la arquitectura MSP430 en el RTOS FreeRTOS requiere 45900% más de memoria RAM que el RTOS Salvo.
- Para ejecutar el caso de Prueba 2, la arquitectura ARM Cortex M3 en el RTOS Salvo requiere 409% más de ciclos de máquina que el RTOS FreeRTOS
- Para ejecutar el caso de Prueba 2, la arquitectura ARM Cortex M3 en el RTOS FreeRTOS requiere 61.81% más ROM que el RTOS Salvo.
- Para ejecutar el caso de Prueba 2, la arquitectura ARM Cortex M3 en el RTOS FreeRTOS requiere 1188% más de memoria RAM que el RTOS Salvo.
- Para ejecutar el caso de Prueba 2, la arquitectura ARM Cortex M3 en el RTOS FreeRTOS requiere menos ciclos de máquina.
- Para ejecutar el caso de Prueba 2, la arquitectura MSP430 en el RTOS FreeRTOS requiere menos memoria ROM.
- Para ejecutar el caso de Prueba 2, la arquitectura MSP430 en el RTOS Salvo requiere menos memoria RAM.

11.3 CASO DE PRUEBA 3: DELAY

- Para ejecutar el caso de Prueba 3, la arquitectura MSP430 en el RTOS Salvo requiere 53.78% más ciclos de máquina que el RTOS FreeRTOS.
- Para ejecutar el caso de Prueba 3, la arquitectura MSP430 en el RTOS FreeRTOS requiere 20% más de memoria ROM que el RTOS Salvo.
- Para ejecutar el caso de Prueba 3, la arquitectura MSP430 en el RTOS FreeRTOS requiere 31900% más de memoria RAM que el RTOS Salvo.
- Para ejecutar el caso de Prueba 3, la arquitectura ARM Cortex M3 en el RTOS Salvo requiere 768% más ciclos de máquina que el RTOS FreeRTOS.
- Para ejecutar el caso de Prueba 3, la arquitectura ARM Cortex M3 en el RTOS FreeRTOS requiere 14.36% más memoria ROM que el RTOS Salvo.
- Para ejecutar el caso de Prueba 3, la arquitectura ARM Cortex M3 en el RTOS FreeRTOS requiere 1188% más memoria RAM que el RTOS Salvo.
- Para ejecutar el caso de Prueba 3, la arquitectura ARM Cortex M3 en el RTOS FreeRTOS requiere menos ciclos de máquina.
- Para ejecutar el caso de Prueba 3, la arquitectura MSP430 en el RTOS Salvo requiere menos memoria ROM.
- Para ejecutar el caso de Prueba 3, la arquitectura MSP430 en el RTOS Salvo requiere menos memoria RAM.

12. CONCLUSIONES

- EL RTOS Salvo se encuentra optimizado para la arquitectura de microprocesadores MSP430, pero es menos eficiente al implementarlo en la arquitectura ARM Cortex M3. Se atribuye este comportamiento a la posible falta de utilización de herramientas claves de ARM para el rendimiento como el doble Stack Pointer, sin embargo, no es posible tener certeza ya que salvo es un RTOS con código cerrado.
- Los resultados obtenidos revelan que los RTOS analizados presentan mayor eficiencia con determinadas arquitecturas, por lo cual, el criterio de selección no debe estar basado únicamente en la portabilidad de la arquitectura con el RTOS.
- El uso de un RTOS cooperativo, aunque disminuye notablemente el consumo de recursos del sistema embebido, aumenta la posibilidad de un bloqueo general del sistema por lo cual, requiere de un protocolo de pruebas más riguroso.
- El RTOS Salvo optimiza el uso de la memoria ROM y RAM, pero requiere mayor cantidad de ciclos de máquina que el RTOS FreeRTOS.
- La arquitectura del microprocesador ARM Cortex M3 consume menos ciclos de máquina que el microprocesador MSP430, sin embargo utiliza una cantidad considerable de memoria RAM y ROM.
- El RTOS Salvo presenta mayor eficiencia en la utilización de la memoria, sin embargo, FreeRTOS permite la administración dinámica de los datos a través de la utilización del Heap de la memoria RAM, factor que es importante en la ejecución de misiones satelitales de larga duración y con poca capacidad de mantenimiento.
- El compilador Keil uVisión presenta mayor resolución para el análisis de ciclos de máquina y consumo de memoria de pequeños códigos .

REFERENCIAS

Alan Burns, A. W. (s.f.). *Sistemas de tiempo real y lenguajes de programación* (Vol. 3ra edición). Addison Wesley.

Arboleda, U. S. (s.f.). *Qué es la Ingeniería de Sistemas y Telecomunicaciones*. Obtenido de Escuela de Ingenierías: http://ingenierias.usergioarboleda.edu.co/index.php?option=com_k2&view=item&layout=item&id=89&Itemid=195

B, S. (2008). ARM and Intel Battle over the Mobile Chip's Future. *IEEE Computer Society ISNN 0018-9162*.

Barry, R. (2010). Using the FreeRTOS Real Time Kernel ARM Cortez M3 Edition. En R. Barry, *Using the FreeRTOS Real Time Kernel ARM Cortez M3 Edition.* © Real Time Engineers Ltd.

Caspi, j. (2005). *http://sigbed.seas.upenn.edu/archives/2005-10/01-wese2005 (Jackson-Caspi).pdf*.

Chasqui_1. (s.f.). *Universidad Nacional de Ingeniería, Lima Perú*. Obtenido de http://www.chasqui.uni.edu.pe/

CubeSat.org. (s.f.). Obtenido de http://cubesat.org/index.php/about-us

Diaz, F. (2012). *MDR C&DH Libertad 2.*

Diaz, F. A. (2012). *PDR C&DH Libertad 2.* Bogotá.

Díaz, F. A. (s.f.). Iniciativa CubeSat. En *Subsistemas de Satélites.*

Dick, R. (2003). Analysis of power dissipation in embedded systems using real- time operating systems. *IEEE*.

Dick, R. P. (2000). Power Analysis of embedded operating systems. *IEEE*.

ECSS. (2002). *Space Engineering - Testing ECSS-E-10-03A.* Holanda.

ECSS. (2008). *Space Engineering, Space Segment Operability ECSS-E-70-11C.* Holanda.

ECSS. (2008). *Space Engineering; Space Wire-Links, nodes, routers and networks ECSS-E-ST-50-12C.* Holanda: ESA-ESTEC.

ECSS. (2009). *Space Engineering- Technical Requirements Specification, ECSS-E-ST-06C.* Holanda.

ECSS. (2009). *Space Engineering, Failure Modes , Effects and Criticality Analysis (FMECA) ECSS-Q-30-02A.* Holanda.

ECSS. (2009). *Space Engineering, Software ECSS-E-ST-40C.*

ECSS. (2009). *Space Engineering, System engineering general requirements, ECSS-E-ST-10C.*

ECSS. (2010). *Space Engineering, Spacecraft on board control Procedures ECSS-E-ST-70-01C.*

Gonzáles, J. (2012). *PDR EPS Libertad 2.* Bogotá, Colombia.

Hernández, R. (2010). *Metodología de la Investigación 5ta Edición.* McGraw Hill.

Herrera, J. L. (s.f.). *Programaci'on en tiempo real y bases de datos: un enfoque practico.* Barcelona: Universirar politecnica de Catalunya- BarcelonaTech.

Ibarra, A. (s.f.). *Rational Unified Process.*

IEEE_Std1471. (2000). *IEEE Recomended Practice For Architechtural Description of Software Intensive Systems.*

IEEE_Std610.12. (1990). IEEE Standard Glosary of Software Engineering Terminology.

Jacobson, B. R. (s.f.). *El Proceso Unificado de Desarrollo de Software.*

Jong, S. d. (2008). Improved Command and Data Handling System For The Delfi N3XT Nanosatellite.

Jose H Canós, P. L. (2007). Metodologías Ágiles en el Desarrollo de Software.

Laplante, P. A. (s.f.). *Real-Time Systems design and analysis* (Vol. Third edition). IEEE PRESS and WILEY-INTERSCIENCE.

Latif, L. (5 de Febrero de 2013). ARM announces a 19 percent increase in revenues. *ARM*.

Lu, R. A. (1995). *Building "Smaller, Cheaper, Faster" Satellites Within the Constraints of an Academic Environment.* Stanford, California.

MCSE Methodology, O. (s.f.). *Cofluent Design. The Methodoloy.*

MIL_STD_1533. (2002). *AIM GmbH Avionics Databus Solutions.*

Milenkovic, M. (1994). *Sistemas Operativos: Concepto y Diseño.* McGraw- Hill.

Misión Libertad 1, U. (s.f.). *Universidad Sergio Arboleda*. Obtenido de http://www.usergioarboleda.edu.co/proyecto_espacial/

NASA, S. F. (2012). *NASA Procedural Requirements.*

Nguyen, Q. (2008). A High Performance Command and Data Handling System For NASA's Lunar Recconaisance Orbiter.

Palacio, J. (s.f.). *Gestión Ágil de Proyectos, SCRUM.* Obtenido de www.navegapolis.net/files/presentaciones/scrum.pp

pc104.org. (2008). *PC104 Specification V2.6.* Embebed Consorcium.

Pérez, D. A. (2009). *Sistemas embebidos y sistemas operativos embebidos.* Obtenido de https://docs.google.com/viewer?a=v&q=cache:mHd-7hwRRqUJ:www.ciens.ucv.ve/escueladecomputacion/documentos/archivo/88+sistema+operativo+en+tiempo+real+ejemplo+embebido&hl=es&pid=bl&srcid=ADGEESgcO1_8DQnCq_Ln9Gh5YviLH0fNNdDMf1DROWVfABjrPpWrqaRz9Mr9lYOmhDqoMP

Prieto, A. (1995). *Introducción a la informática.* McGraw-Hill.

Rafael V. Aroca, G. C. (2009). A real time operating systems (Rtos) Comparison. *IEEE*.

Scholz, A. (2004). Command And Data Handling Design For The Compass-1 Picosatellite.

Sequoia Space. (s.f.). Obtenido de http://www.sequoiaspace.com/ss_cubesat_index.html

Tan, S.-L. L. (2009). Real-time operating system (RTOS) for small (16-bit) microcontroller. *IEEE*.

Tanenbaum. (1988). *Sistemas Operativos: Diseño e Implementación.* Prentice.

Tecnología, R. (2012). Servicios de Telefónica son ahora Movistar. *Portafolio*, http://www.portafolio.co/economia/servicios-telefonica-son-ahora-movistar.

TeleManagement Forum. (s.f.). Obtenido de TAM: http://www.tmforum.org/ApplicationFramework/2322/home.html

The Cube Sat Program, C. (s.f.). CubeSat Design Specification .

Ubbels, W. (2004). Delfi-C3 a Student Nanosatellite as a Test-bed for Thin Film Solar Cells .

University, C. P. (2001). Development of the Standard CubeSat Deployer and a CubeSat Class PicoSatellite. *IEEE*.

Vaartjes, B. (2007). Integration And Verification Of A Command And Data Handling Subsystem For Nano-Satellite Projects With Critical Time Constraints: Delfi-C3.

Vapnik, V. (1998). *Statistical learning theory.* Ed Wiley.

Victor Manuel, A. J. (s.f.). *Sistemas Operativos.* Alhambra.

Wiley J, L. (2005). *Space Mission Analysis And Design SMAD.* California Southern University: Space Technology Library.

Wolf, W. (s.f.). *Overheads for computers as components, 2nd ed.* Obtenido de http://www.waynewolf.us/embedded-book-2e/Overheads/ch1-1.ppt,

Yiu, J. (2007). The definitive Guide to the ARM Cortex-M3. *ELSEVIER ISBN 978-1-85617-963-4*.

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.