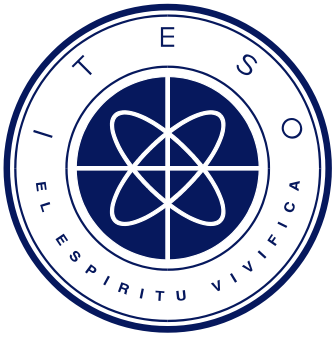
INSTITUTO TECNOLÓGICO Y DE ESTUDIOS SUPERIORES DE OCCIDENTE

Reconocimiento de validez oficial de estudios de nivel superior según acuerdo secretarial 15018, publicado en el Diario Oficial de la Federación el 29 de noviembre de 1976.

Departamento de Electrónica, Sistemas e Informática

ESPECIALIDAD EN SISTEMAS EMBEBIDOS



MECANISMO DE CAMBIO DE CONTEXTO DE TAREAS EN UNA ARQUITECTURA POWERPC

Tesina para obtener el grado de:

ESPECIALISTA EN SISTEMAS EMBEBIDOS

Presentan:
José Alonso Vallejo Ríos.
Miguel de Jesús Zamora Cortes.
Enrique Avalos Becerra.

Director: Luis Rizo Domínguez

San Pedro Tlaquepaque, Jalisco. junio de 2018.

Esta página es para la dedicatoria (opcional).

Alonso
A mi familia, CONACYT (CVU 861142) y a mi compañero de trabajo Agustín por su apoyo durante el desarrollo, a Goku y a No 17 porque salvo al universo 7 de ser destruido.

Enrique
Dedico este trabajo a CONACYT (CVU 862358) porque gracias a ellos y su soporte pude realizar los estudios de la Especialidad en Sistemas Embebidos. Agradezco a mis profesores por los conocimientos que me brindaron, a mi familia que sufrió junto conmigo el tiempo que no pude dedicarles mientras me encontraba trabajando en temas de la escuela, a mis amigos, a Continental por brindarme los medios para asistir y cumplir mis actividades y de la misma manera dedico esta obra la Norteña a quien le he dedicado varias desde que la conocí, porque no dedicarle también este trabajo.

Miguel
En primer lugar quisiera agradecer a mi tutor Francisco Martinez, por su constante orientación, valiosos comentarios y sugerencias que enriquecieron el desarrollo de la tesina.
Reconozco la ayuda recibida de mi querido coordi y amigo Sergio Pineda, quien aporto con conocimientos en la materia y su atención a brindarme todos los recursos que necesitara para lograr el éxito de esta tesina
Agradezco a CONACYT por la oportunidad otorgada al CVU 861150
A mis padres, Miguel y Pilar, por enseñarme que la perseverancia y la disciplina son los caminos para lograr el éxito. Les reitero mi gratitud por el apoyo incondicional que he recibido de ellos desde el día de mi nacimiento. A mis hermanas, Alondra y Andrea que comprendieron mis limitantes de tiempo y siempre me soportaron.
Tampoco sería junto olvidar a mis amigos, quienes aceptaron mis nuevas responsabilidades y a pesar de mis limitantes de tiempo siempre buscaron el espacio para verme y hacerme mejorar como persona, Nadia Ramírez, Luis Machuca, Agustín Díaz, Carlos Sandoval, Miguel Luna y Samantha Vázquez

Abstract

The study focuses on the implementation of task context switch mechanism in a preventive scheduler for the PowerPC architecture, because it is widely used in the automotive area, since a small actuator control module sensor monitoring module until development of a large processing module such as a BCM (Body Controller Module).

The PowerPC architecture was designed by the AIM [1] alliance formed by IBM, Motorola and Apple, which was originally developed for computer processors and later was introduced into automotive area. Nowadays it is one of the most used architectures and this work will be valid if this technology continues in use in the automotive area.

The development of this study shall contemplate the raison d'être of memory management selection techniques, performance and efficiency measurements considering the "worst case" scenario for different task context switch mechanisms and a list of advantages and disadvantages among them. In addition, a generic non-preemptive scheduler should be taken as a basis and adapted to be preemptive and compatible with the NXP TRK-MPC560XB development board.

Resumen

El estudio se centra en la implementación de un mecanismo de cambio de contexto rápido de tareas en un Scheduler Preemptive para la arquitectura PowerPC, ya que es utilizada ampliamente en el área automotriz, desde pequeños de control de un actuador o monitoreo de un sensor hasta el desarrollo de un módulo de gran procesamiento como lo es un BCM (Body Controller Module).

La arquitectura PowerPC se diseñó por la alianza AIM **[1]** *conformada por IBM, Motorola y Apple que, originalmente, fue desarrollada para procesadores de computadora y con el paso del tiempo se introdujo al área automotriz. Hoy en día es una de las arquitecturas más utilizadas y este trabajo se mantendrá vigente durante el tiempo en que esta tecnología se mantenga en uso en el área automotriz.*

El desarrollo de este estudio deberá contemplar la razón de ser de las técnicas de selección de manejo de memoria, mediciones de rendimiento y eficiencia considerando los escenarios de "peor caso" para diferentes mecanismos de cambio de contexto y una lista de ventajas y desventajas entre ellos. Además, se debe tomar como base un Scheduler non Preemptive genérico y adaptarlo para ser Preemptive y compatible con la tarjeta de desarrollo TRK-MPC560xB de NXP.

Contenido

Índice de Figuras

Índice de Tablas

Lista de acrónimos.

BCM - Modulo de control chasis (Body Control Module).

AIM - Apple-IBM-Motorola.

RTOS - Sistema operativo en tiempo real (Real time operative system).

RISC - Conjunto reducido de instrucciones (Reduced instruction set computing).

IEEE - Ingeniería eléctrica y electrónica internacional (Interntaional electrican and ||electronics engineering).

ISR - Rutina de servicio de interrupción. (Interrupt service rutine).

CPU - Unidad central de procesamiento (Central processing unit).

EABI - Interfaz Binaria de aplicación embebebida (Embedded application binary inteface).

IDE - Ambiente integrado de desarrollo (Integrated development enviorenment).

UART - Transmisor-Receptor asincrónico universal (Universal Asynchronous Receiver-Transmitter).

GPIO - Entrada-Salida de propósito general (General Purpose input-output).

PIT - Interrupción periódica por tiempo (Periodic timer interrupt).

IVOR - Vector de valores de compensaciones para interrupciones (Interrupt Vector Offset Values).

Introducción.

Un sistema operativo es el programa principal de un sistema embebido, que administra la ejecución de tareas, los recursos de Hardware, servicios de Software del sistema y la forma en que interactúan entre ellos. Puede ser o no un sistema operativo de tiempo real (RTOS por sus siglas en inglés), el cual debe de ejecutar una serie de tareas en un periodo definido de tiempo.

Los componentes principales de un sistema operativo son:

- **Kernel.** Provee y maneja los recursos del sistema a través de servicios.
- **Scheduler.** Implementa una política y un algoritmo de ejecución de tareas para seleccionar la siguiente tarea a realizar.
- **Dispatcher.** Llama la ejecución de la tarea seleccionada por el Scheduler.

Para la realización de esta tesina nos planteamos una serie de preguntas en las que centramos nuestro marco teórico, metodología y objetivos. La pregunta principal es: ¿Cuál es el método más eficiente para un cambio de contexto en los procesadores PowerPC?

Y las siguientes preguntas secundarias son:

- ¿Cuáles son los registros necesarios para un cambio de contexto en la arquitectura PowerPC?
- ¿Existe alguna forma de medir un cambio de contexto dentro de procesadores?
- ¿Cuáles serían los escenarios de prueba para responder la pregunta anterior?
- ¿Realmente existe un método más eficiente que los demás para un cambio de contexto en los procesadores PowerPC o éste depende de la aplicación que se busque?

La importancia de nuestra investigación radica en el hecho de que cada año los tiempos de desarrollo de nuevos proyectos disminuye y, a su vez, los costos se elevan; por lo tanto, es de vital importancia conocer desde la fase de diseño la tecnología a utilizar para evitar que se trabaje de nuevo en fases posteriores.

Las necesidades básicas que queremos satisfacer son:

- Mejorar la eficiencia del uso de los recursos de un sistema embebido.
- Otorgar un punto de comparación de diferentes mecanismos de cambio de contexto para elegir el que otorgue el mayor beneficio en la aplicación deseada.
- Reducir el tiempo de desarrollo de una arquitectura de una aplicación en un sistema embebido.
- Reducir los tiempos de cambio de contexto de tareas de un Scheduler Preemptive de una manera eficiente.

A partir de ello se busca:

- Mejorar la eficiencia en el uso de los recursos de un sistema embebido.
- Otorgar un punto de comparación de diferentes mecanismos de cambio de contexto para elegir el que otorgue el mayor beneficio en la aplicación deseada.
- Reducir el tiempo de desarrollo de una arquitectura de una aplicación en un sistema embebido.
- Reducir los tiempos de cambio de contexto de tareas de un Scheduler Preemptive de una manera eficiente.

Las inversiones de desarrollo e investigación en 2016 y 2017 de las principales empresas de desarrollo electrónico en el área automotriz que operan en el país son (todas ellas publicadas en sus reportes anuales):

- Bosch **[2, p. 60]** 7,264 Millones de Euros.
- Continental **[3, p. 49]** 3,103.7 Millones de Euros.
- Hella 636 **[4, p. 42]** Millones de Euros.
- Delphi **[5, p. 11]** 1,500 Millones de dólares.
- Valeo 1,895 **[6, p. 29]** Millones de Euros.

.

Production of passenger cars and light commercial vehicles

millions of units	2017	2016	2015	2014	2013
Europe[1]	22.1	21.4	20.8	19.9	19.2
North America	17.1	17.8	17.5	17.0	16.2
South America	3.3	2.7	3.1	3.8	4.5
Asia[2]	51.5	50.0	46.4	45.8	44.0
Other markets	1.1	1.1	1.0	0.9	0.8
Worldwide	**95.1**	**93.1**	**88.8**	**87.4**	**84.7**

Source: IHS Inc., preliminary figures and own estimates.
1 Western, Central and Eastern Europe, including Russia and Turkey.
2 Asia including Kazakhstan, Uzbekistan, Middle East and Oceania with Australia.

Fig. 0-1 Producción de Autos en 2017 de acuerdo con el reporte anual de Continental [3, p. 61].

.

TABLA I
PORCENTAJE DE INVERSIÓN EN FUNCIÓN DE GANANCIAS EN INVESTIGACIÓN Y DESARROLLO [7]

	R&d spending, % of sales	Profit, % of sales
Bosch	9.4	N.A.
Denso	8.4	7.1
Delphi	7.3	-1.5
Behr	6.4	4
Valeo	6.2	2.4
NGK	6.1	8.7
Autoliv	6	8.5
Aisin Seiki	5.5	4.8
Hella	5.5	3.1
ZF	5.3	3.8
Visteon	4.8	-2.2
INDUSTRY AVG.	4.3	5.3
Michelin	4.3	6.7
Continental	4.2	8.8
BorgWarner	3.5	9.7
GKN	2.6	-0.3
Federal-Mogul	2.2	-1.5

- **Productos o servicios en competencia.**
  - **Productos similares:** Se realizó una búsqueda en investigaciones de la IEEE, patentes y artículos técnicos y no se encontró un artículo similar enfocado al caso de estudio de cambio de contexto rápido en arquitecturas PowerPC.
  - **Productos alternativos:** A la par de esta investigación se encuentra en desarrollo una paralela en el caso de estudio de arquitecturas Arm **[7]**.

.

1. Antecedentes

La arquitectura PowerPC (por sus siglas de la frase en inglés "*Peformance Optimization With Enhanced RISC – Performance Computing*" [8]) es una arquitectura con un conjunto reducido de instrucciones (RISC por sus siglas en inglés, "*Reduced instruction set computing*"), creada por la alianza de fabricantes Apple, IBM y Motorola, llamada AIM en 1991 [8]. Fue desarrollada originalmente para computadoras personales y, con el paso del tiempo, se comenzó a utilizar en diferentes dispositivos como las consolas de video juegos y los sistemas embebidos.

La referencia más antigua relacionada a un cambio de contexto en un sistema operativo, dentro de la biblioteca virtual Xplore [9] de IEEE, es la publicación "*Problems of Programming for Shared Real-Time Systems*" de L.S. Tuomenoksa y W, Ulrich [10], donde define a un Sistema Operativo en Tiempo Real como un sistema en el que el ritmo de sus entradas y/o salidas no puede ser controlado por el sistema, sino que depende de factores externos, por lo que no tiene manera de definir en qué momento una tarea que se encuentra en ejecución podría ser interrumpida y sustituida por una nueva tarea. Con ello se genera la necesidad de que un sistema operativo tenga la habilidad de guardar y restaurar el contexto de una tarea.

De la misma manera se encontraron diversas aplicaciones que representan una referencia para basar y sustentar nuestra investigación, como son las patentes de Robert Alan Reid, "*Task Context switching RTOS*" [11]; Juraj Bystricky *et al*, "*System and method for performing an interface save/restore procedure*" [12]; Seungyoon Peter Song *et al*, "*Efficient context saving and restoring in a multi-tasking computing system environment*" [13]; y Howard Israel Nayberg, "*Method and System for Task Switching with Inline Execution*" [14]. De estas patentes se deduce que, a pesar de que las primeras referencias datan desde la década de los 60, aún se sigue trabajando arduamente para resolver de manera eficiente el mismo problema, ya que existen muchas variables en torno al manejo eficiente de un cambio de contexto, como lo son la arquitectura del microcontrolador, la aplicación, regulaciones internacionales como la OSEK/VDX (ISO 17356),

restricciones de diseño definidas en los requerimientos del sistema, el tipo de sistema operativo a utilizar, entre otras.

2. Sistema Operativo.

Un RTOS (Real Time Operating System) es aquel software que responde a eventos externos en un enfoque de tiempo oportuno. Estos sistemas siempre deben reaccionar a un evento dentro de un límite de tiempo, ya que al no hacerlo puede generar comportamientos incorrectos o decrementar el rendimiento de la aplicación llevándola hasta su falla. Estos sistemas se encuentran en aplicaciones de alto rendimiento donde la respuesta del sistema es muy sensible a retrasos, como, por ejemplo; plantas de control nuclear y químicas, sistemas de control aéreo, sistemas de telecomunicaciones, sistemas militares, robótica y en el ramo automotriz.

Los sistemas operativos en tiempo real se clasifican en dos tipos; sistemas Hard y sistemas Soft. Los sistemas Hard son aquellos que no tienen ninguna flexibilidad de tiempo. Los límites de tiempo siempre deben de ser alcanzados o fallas críticas podrían ocurrir. El costo de dichas fallas puede involucrar vidas humanas y grandes pérdidas materiales, como, por ejemplo; un marcapasos o un radar en pista de aterrizaje. Los sistemas Soft buscan alcanzar los límites de tiempo; sin embargo, contienen niveles de tolerancia y el no responder a tiempo no provoca una falla crítica, como, por ejemplo; una televisión, una computadora y un refrigerador.

Las características que definen a los sistemas operativos son:

- Tiempo de respuesta, se define como el tiempo en reaccionar a los eventos; para un sistema duro el tiempo de respuesta debe estar en milisegundos o menos y para un sistema suave puede estar en los segundos.
- Rendimiento de carga máxima, las cargas de trabajo se generan por la aplicación; para un sistema duro se tiene que calcular las cargas de trabajo y se debe garantizar que en el peor escenario (cuando más cargas haya) aún se cumplan los límites de tiempo y para un sistema suave se evitan estos cálculos ya que si se tiene una gran carga de trabajo se puede tolerar.
- Control de paz, un sistema duro debe estar alerta a cualquier cambio en el ambiente, para un sistema suave debe estar alerta a cambios programados.
- Seguridad, para sistemas suaves la seguridad consiste en tener detección de errores, recuperación autónoma y, de ser posible, un estado seguro ante fallas. Para un sistema suave se puede o no tener seguridad, queda a manos del diseñador.

- Tamaño de datos, en un sistema duro se tiene que tener tamaño de datos pequeños o medianos, pero deben ser moderados para no tener un compromiso con las cargas de trabajo, en un sistema suave se pueden tener datos grandes.
- Tipo de redundancia, para un sistema duro no existen tipos de redundancia, pero para un sistema suave sí y consiste en regresar a un punto seguro en que la falla no existía.

2.1. Kernel.

El Kernel forma parte de un sistema operativo. Es un supervisor de software, provee de lógica mínima para el procesador, maneja los recursos del sistema embebido y es parte de la selección del planificador. Algunos ejemplos de los recursos básicos que maneja el kernel son:

- Memoria.
- Procesador.
- Temporizadores y reloj.
- Las interrupciones.
- Las entradas y salidas de propósito general.
- Control de procesos.
- Buffer y colas.
- Política de programación.
- Maneja las exclusiones mutuas.
- Semáforos.

2.2. Dispatcher.

El dispatcher es la parte del sistema operativo que realiza el cambio de contexto y cambia el flujo de la ejecución. El pasador tiene como propósito general cambiar el flujo de ejecución y, usualmente, es llamado por el kernel, *ISR* y tareas.

2.3. Scheduler.

El Scheduler es un componente muy importante para sistemas operativos de procesos multitarea y multiprocesos. Es, además, una de las partes fundamentales de cada Kernel. El Scheduler provee los algoritmos necesarios para determinar cuál tarea se ejecuta y cuando.

El trabajo del Scheduler es administrar el tiempo disponible entre todos los procesos habilitados para su ejecución. Básicamente un Scheduler mantiene ocupado tanto como sea posible al microcontrolador ejecutando diferentes tareas.

2.3.1 Políticas de Scheduler.

La política de un Scheduler es el criterio que éste seguirá para la ejecución de las tareas. Las tareas en los RTOS consideran restricciones de tiempo, precedencia y recursos.

Las restricciones de tiempo dentro de las políticas de un Scheduler se determinan como:

- Tiempo de llegada: Es el tiempo en el cual una tarea ya está lista para su ejecución.
- Tiempo de ejecución: Es el tiempo requerido por una tarea desde el procesador para finalizar su ejecución.
- Tiempo Límite ("*deadline*" por sus siglas en inglés): Es tiempo máximo permitido para que una tarea finalice su ejecución. Si una tarea ha sobrepasado el tiempo límite se podrían obtener resultados desastrosos.
- Tiempo de inicio: Es el tiempo a partir de que una tarea comienza su ejecución.
- Tiempo de terminación: Es el tiempo en el que una tarea terminó su ejecución.
- Criticidad: El parámetro de criticidad está relacionado con la consecuencia de perder un tiempo límite o deadline.
- Tiempo de relajación: Es el tiempo máximo donde una tarea puede comenzar su ejecución para que una tarea pueda terminar en tiempo.
- Periodo: Es un intervalo de tiempo cuando la tarea es ejecutada otra vez. Este tiempo es necesario para una tarea periódica y esporádica.

Las restricciones de precedencia también son consideradas en la política de Scheduler. En algunas aplicaciones, las ejecuciones de las tareas no pueden ser activadas en un orden arbitrario.

La administración de recursos en las políticas del Scheduler también es importante. Se le conoce como restricciones de recursos.

2.3.2 Algoritmo de Scheduler.

El Scheduler determina cuál tarea se ejecutará siguiendo un algoritmo de schedulling. Los algoritmos de Scheduler conocidos son *Round-Robin, Rate-Monotonic* y *Earliest-Deadline*.

El algoritmo *Round-Rob*in es uno de los más simples para procesamiento en un sistema operativo. Asigna proporciones de tiempo a cada proceso en igualdad y en orden circular manejando todo el proceso sin prioridad.

En el algoritmo *Rate-Monotonic* las prioridades son asignadas con base al ciclo de duración de cada tarea: el ciclo de duración más corto es el que tiene la mayor prioridad y éste es generalmente de tipo Preemptive.

El algoritmo *Earliest-Deadline* asigna la mayor prioridad a la tarea con el tiempo de ejecución más pequeño. A una tarea se le asigna la mayor prioridad si el tiempo límite o si la actual petición es las más cercana.

2.3.3 Tarea.

Las tareas tienen un pequeño número de estados que incluyen estado listo (*ready*), corriendo (*running*) y suspendido (*suspend*). Una tarea cambia entre un estado a otro dependiendo de la lógica en una máquina de estados finita. La figura 2.1 muestra cómo es el cambio de los estados de una tarea.

Fig. 2-1 Estados de una tarea.

El Scheduler puede usar el estado Ready para mover tareas a un estado Running. Cuando una tarea está en estado Ready significa que solo espera la orden del Scheduler para ser ejecutada.

Cuando una tarea es cambiada a un estado Running, el procesador carga sus registros con el contexto de la tarea. Los valores de contienen los registros cuando se ejecuta una tarea se le denomina contexto. El procesador, entonces, puede ejecutar las instrucciones de la tarea y manipular la pila (*stack*) asociada. Una tarea puede regresar al estado Ready mientras está en estado Running. La figura 2.2 muestra mas detalle sobre el cambio de estados de las tareas.

Una tarea que se encuentra en estado Suspend ayuda a que otra de menor prioridad pueda pasar a estado Running. Una tarea solo puede cambiar a estado Suspend por una llamada

bloqueante. Una tarea permanece en estado Suspend hasta que una condición de desbloqueo es conocida.

2.3.4 Scheduler Non Preemptive.

Un Scheduler es Non Preemptive si una vez que una tarea comienza su ejecución no puede ser interrumpida hasta que finaliza. Cada tarea se ejecuta hasta que no tenga nada más que procesar. El Scheduler otorga el *CPU* a una tarea dependiendo de su prioridad. La tarea que se encuentre en estado Running solo puede ser interrumpida por un ISR ("*Interrupt Service Rutine*" por sus siglas en inglés). La figura 2.2 muestra el comportamiento de 2 tareas, A y B, para un Scheduler Non Preemptive.

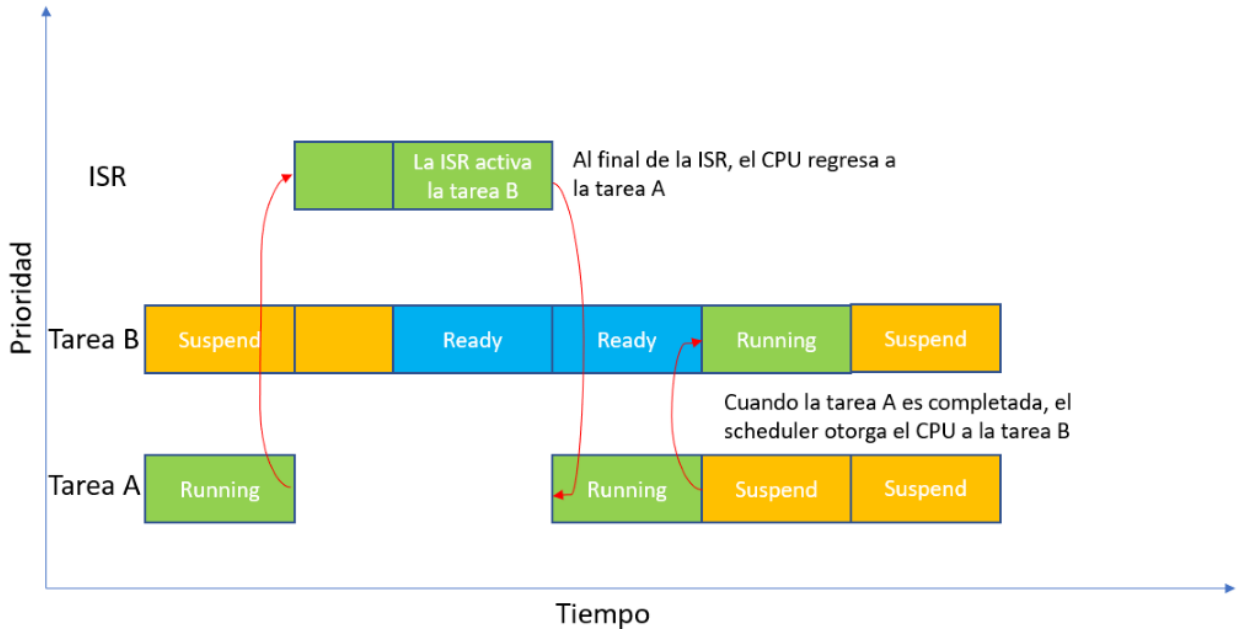


Fig. 2-2 Comportamiento de un Scheduler Non Preemptive.

2.3.5 Scheduler Preemptive.

Un Scheduler es Preemptive si permite que una tarea pueda pasar a estado Suspend en cualquier momento. Una tarea de mayor prioridad puede interrumpir a una de menor prioridad.

Cada tarea permanece en estado Running hasta que no tenga nada más que procesar o hasta que el Scheduler decida ejecutar una tarea de mayor prioridad.

La complicidad es que una tarea puede ser interrumpida en cualquier punto por otra tarea y a este proceso se le conoce como una tarea que fue *preempted* (por su concepto en inglés). Cuando una tarea es preempted, su contexto (variables e instrucciones) tiene que ser guardado. El Scheduler otorga el CPU a una tarea dependiendo de su prioridad. Una tarea puede ser interrumpida por una ISR o por otra tarea. La figura 2.3 muestra el comportamiento de 2 tareas A y B para un Scheduler Preemptive.

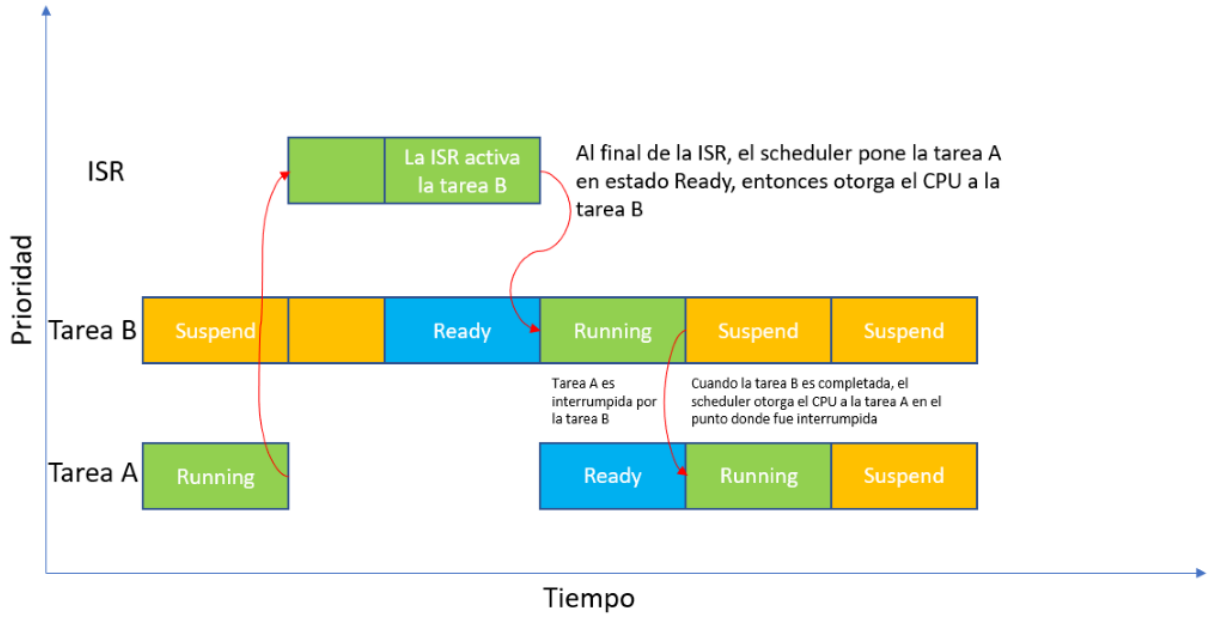


Fig. 2-3 Comportamiento de un Scheduler Preemptive.

3. Manejo de Memoria en un PowerPC.

Para llevar a cabo el desarrollo del caso de estudio de un cambio de contexto rápido en la arquitectura PowerPC es importante conocer detalladamente la arquitectura del microcontrolador utilizado, que es el MPC56XXB de la compañía Freescale (ahora NXP), en especial los registros de propósito general, manejo de interrupciones, su *stack frame*, así como el proceso de dicho cambio de contexto.

Las partes fundamentales de la arquitectura se detallan en los subcapítulos siguientes, así como las principales fuentes técnicas de información.

3.1. EABI.

El EABI [15] (por sus siglas en inglés "*Embedded Application Binary Interface*") define todas las interfaces y recursos para un Sistema Embebido basado en la arquitectura PowerPC, para microcontroladores de 32 bits de Freescale.

[15] es un complemento del SVR4 ABI [16], donde [16] define de manera general los microcontroladores PowerPC, mientras que [15] complementa y adapta la arquitectura a la implementación realizada en microcontroladores Freescale.

De [15] y [16] se extrajo la información necesaria para comprender el modo de funcionamiento del microcontrolador montado en la tarjeta de desarrollo utilizada en la realización de esta investigación.

3.2. Interrupciones.

Las interrupciones son eventos que detienen la ejecución normal de las tareas para otorgar el CPU a una rutina de mayor prioridad. Cada microcontrolador soporta una gama diferente de interrupciones, no importando que éstos sean de la misma arquitectura o, inclusive, de la misma familia, por lo que es importante siempre revisar la hoja de especificaciones del microcontrolador a utilizar.

Por su origen, las interrupciones se clasifican en 2 grupos:

- **Internas.** Este tipo de interrupciones son generadas por un evento interno del microcontrolador y son utilizadas para notificar al CPU del microcontrolador cuando una subtarea ha terminado, un contador llegó a una cuenta especificada, información ha sido procesada, etcétera. Este tipo de interrupciones pueden tener una periodicidad conocida, así como un comportamiento esperado, por lo que se son consideradas predecibles y son parte de la descripción del comportamiento dinámico del sistema.
- **Externas.** Este tipo de interrupciones son generadas por la interacción de un componente externo al microcontrolador por medio de un puerto, un pin de entrada, la recepción de un mensaje de comunicación, *watchdog* externo, etcétera. Estas interrupciones son impredecibles, ya que es imposible conocer cuándo se activarán; por ende, no hay manera de describir cómo afectan la ejecución normal del sistema más allá de generar un análisis del peor caso posible y ver si ello está dentro de los objetivos de comportamiento del sistema (especialmente de latencia, deadlines de tareas y la carga del CPU).

Las interrupciones del microcontrolador utilizado en este estudio son:

TABLA II
FUENTES DE INTERRUPCIÓN DISPONIBLES EN EL MICROCONTROLADOR MPC5606B [18].

Fuentes de Interrupción (204)	Número disponibles
Software	8
ECSM	1
eDMA	17
Software Watchdog (SWT)	1
STM	4
Flash/SRAM ECC (SEC-DED)	2
Real Time Counter (RTC/API)	2
System Integration Unit Lite (SIUL)	3
WKPU	4
MC_ME	4
MC_RGM	1
FXOSC	1
SXOSC	1
PIT	8
ADC_0	2
ADC_1	2
FlexCAN_0	8
FlexCAN_1	8
FlexCAN_2	8
FlexCAN_3	8
FlexCAN_4	8
FlexCAN_5	8
LINFlex_0	3
LINFlex_1	3
LINFlex_2	3
LINFlex_3	3
LINFlex_4	3
LINFlex_5	3
LINFlex_6	3
LINFlex_7	3

TABLA III
FUENTES DE INTERRUPCIÓN DISPONIBLES EN EL MICROCONTROLADOR MPC5606B [18] - Continuación.

Fuentes de Interrupción (204)	Número disponibles
DSPI_0	5
DSPI_1	5
DSPI_2	5
DSPI_3	5
DSPI_4	5
DSPI_5	5
I2C_0	1
Enhanced Modular I/O Subsystem 0 (eMIOS_0)	16
eMIOS_1	16

3.3. Registros.

Los registros son una sección reservada de memoria que almacena las configuraciones que modifican el funcionamiento del microcontrolador, datos de lectura de periféricos y dispositivos internos, datos de escritura de periféricos y dispositivos internos, memoria dedicada a tareas específicas y registros de propósito general.

Los más importantes para nuestra investigación son los registros de propósito general, ya que en ellos se almacena la información que contiene el contexto de una tarea, por lo tanto, el manejo de estos registros son el punto medular de la implementación de un cambio de contexto.

El SVR4 ABI [16] define para la arquitectura PowerPC 32 registros enteros de propósito general (r), cada uno de ellos de 32 bits. Además, contiene 32 registros de punto flotante (f), cada uno de ellos de 64 bits y registros de propósito especial. Todos los registros enteros, de punto flotante y de propósito especial, son accesibles para cualquier tarea en ejecución en el sistema, no importando su contexto.

.

TABLA IV
REGISTROS DEL PROCESADOR

Nombre de Registro	Uso
r0	Registro volátil el cuál puede ser modificado durante la conexión (linkage) de funciones
r1	Puntero al Stack frame
r2	Registro reservado del sistema
r3 - r4	Registros volátiles utilizado para el paso de parámetros y valores de retorno
r5 - r10	Registros volátiles usados para el paso de parámetros
r11 - r12	Registros volátiles los cuales pueden ser modificado durante la conexión (linkage) de funciones
r13	Punteros al área de Small data
r14 - r30	Registros usados por variables locales
r31	Usado para variables locales o punteros de ambiente, el cual apunta a la dirección donde se encuentran las instrucciones de la tarea o función en ejecución.
f0	Registro volátil
f1	Registro volátil utilizado para el paso de parámetros y valores de retorno
f2 - f8	Registros volátiles usados para el paso de parámetros
f9 - f13	Registros volátiles
f14 - f31	Registros usados por variables locales
CR0 - CR7	Registros de campos de condición, cada uno de 4 bits
LR	Registro de conexión (Link register)
CTR	Registro contador (Count Register)
XER	Registro de excepción de punto flotante
FPSCR	Estatus de punto flotante y Registro de Control

De acuerdo con [16], los registros r1, r14 hasta r31 y f4 hasta f31 no son volátiles, esto es, que pertenecen a la llamada de funciones. Una función que es llamada debe guardar la información de estos registros antes de cambiarlos, restaurando sus valores antes de regresar. Registros r0, r3 hasta r12, f0 hasta f13, y los registros especiales CTR y XER son volátiles, esto es, que no son preservados entre llamadas a funciones. Además, los valores en los registros r0, r11 y r12 pueden

ser alterados por llamadas cruzadas a funciones, por ello una función no puede confiar en que los valores de estos registros sean los mismos que tenían cuando la función fue llamada.

El registro r2 es reservado para el uso del sistema y no puede ser cambiado por el código de aplicación.

El registro r13 es el puntero al área de small data. El código del proceso de inicialización para ejecutables que referencian a áreas de small data con un offset de 16 bits direccionando de manera relativa a r13 deben cargar la dirección base del área de small data en r13. Objetos compartidos no deben alterar el valor de r13.

Lenguajes de programación que requieran "punteros de ambiente" deben usar r31 para ese propósito.

Los campos CR2, CR3 y CR4 de los registros de condición son no volátiles (el valor a la entrada debe ser preservado a la salida); el resto son volátiles (el valor en el campo no necesita ser preservado). Los bits VE, OE, UE, ZE, XE, NI y RN (modo de redondeo "rounding mode") del registro FPSCR pueden ser cambiados solo por una llamada a función (por ejemplo, *fpsetround()*) que tiene documentado el efecto secundario de cambiarlos. El resto del FPSCR es volátil.

Los siguientes registros tienen asignados roles en la secuencia estándar de llamado:

- **r1.** El stack pointer (guardado en r1) debe de mantener una alineación de 18 bytes. Éste debe de apuntar siempre al stack frame más bajo asignado y debe crecer hacia direcciones bajas. Los contenidos del "*Word*" de la dirección siempre debe apuntar al stack frame previamente asignado. Si es requerido, éste puede ser decrementado por una llamada a función.
- **r3 hasta r10 y f1 hasta f8.** Este grupo de registros volátiles puede ser modificado a través de la invocación de funciones y, por lo tanto, se supondrá que por la llamada a función será destruida. Son usados para el paso de parámetros en las llamadas a funciones. En adición, los registros r3, r4 y f1 son usados para los valores de retorno de las funciones llamadas.

- **Bit 6 del registro CR (CR1, Excepción de un Punto flotante invalido).** Este bit debe ser escrito por la lista de argumentos de variables de la función que realiza la llamada.
- **LR (Link Register).** Este registro debe contener la dirección a la cual una función llamada debe de regresar. *LR* es volátil a través de las llamadas a funciones.

3.4. Stack Frame.

En adición a los registros, cada función puede tener un stack frame en el *runtime stack* (stack en tiempo de ejecución). Este stack crece decreciendo desde la dirección más alta. La figura 3.1 muestra la organización del stack frame. *SP* en la figura denota el stack pointer (registro de propósito general r1) de la función llamada, después de que el código ejecutado establece su propio stack frame.

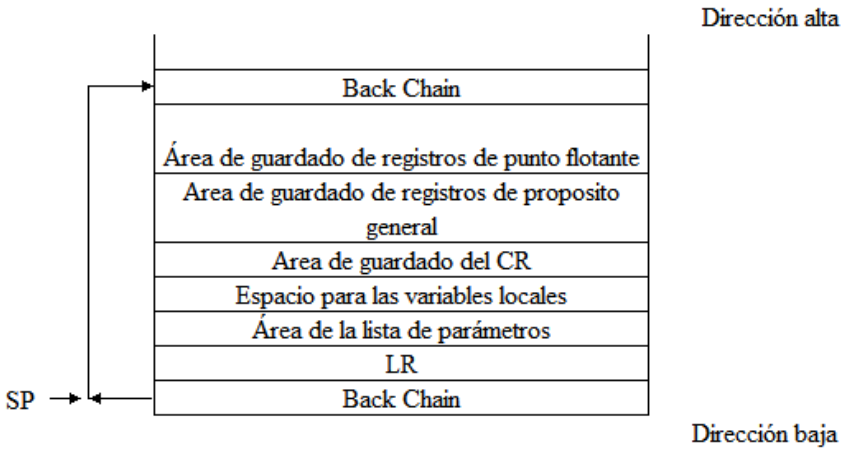


Fig. 3-1 Stack Frame estándar.

3.5. Excepciones IVOR.

Un *IVOR* (por sus siglas en inglés Interrupt Vector Offset Registers) es usado durante el proceso de interrupción para determinar la dirección inicial de un handler. El valor en el vector del IVOR asignado al tipo de interrupción se concatena con el valor en IVPR (por sus siglas en inglés Interrupt Vector Prefix Register) para formar una dirección de instrucción en qué ejecución debe comenzar. La tabla V muestra todas las excepciones IVOR disponibles para el MPC560XB.

TABLA V
IVOR disponibles

Tipo de interrupción	IVOR correspondiente	12 bit Hexa decimal offset
Critical Input	IVOR 0	0x000
Machine check	IVOR 1	0x010
Data Storage	IVOR 2	0x020
Instruction Storage	IVOR3	0x030
External Input	IVOR 4	0x040
Alignment	IVOR 5	0x050
Program	IVOR 6	0x060
Floating-point unavailable	IVOR 7	Reservada
System call	IVOR 8	0x080
AP unavailable	IVOR 9	Reservada
Decrementer	IVOR 10	Reservada
Fixed Interval Timer	IVOR 11	Reservada
Watchdog Timer	IVOR 12	Reservada
Data TLB Error	IVOR 13	Reservada
Instruction TLB Error	IVOR 14	Reservada
Debug	IVOR 15	0x0F0

4. Metodología.

4.1. Requerimientos del proyecto.

El objetivo principal del proyecto es la implementación de un algoritmo de cambio de cambio de contexto optimizado para los procesadores PowerPC. Para realizar las pruebas del cambio de contexto es necesario el diseño de un Scheduler Preemptive. El software del Scheduler debe implementarse sobre un microcontrolador que contenga un procesador PowerPC.

4.2. Herramientas de Hardware y Software.

Para la implementación de los requerimientos fueron seleccionadas diversas herramientas de hardware y software.

4.2.1 Plataforma de Hardware.

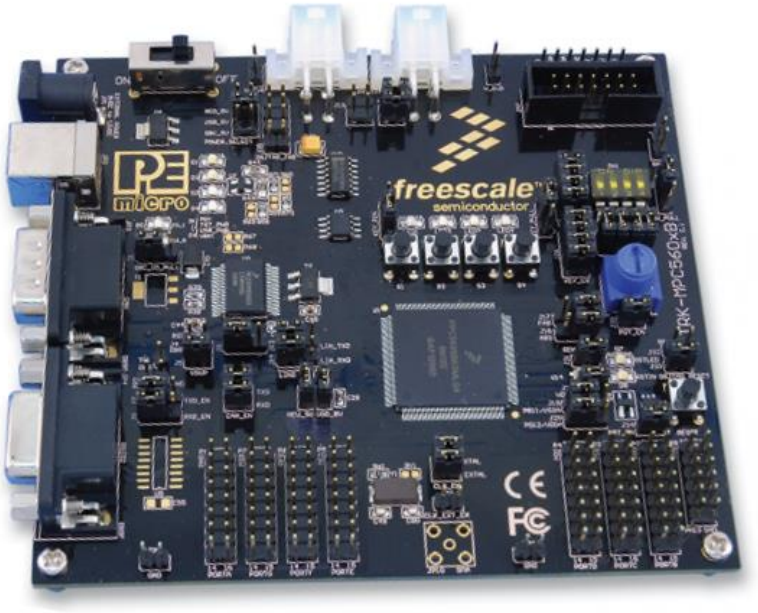


Fig. 4-1 Tarjeta de desarrollo TRK-MPC560XB de NXP.

El microcontrolador seleccionado para la implementación del software fue el MPC560XB. El microcontrolador MPC560XB tiene un procesador PowerPC. El microcontrolador seleccionado está montado sobre una base TRK-MPC560XB de NXP, como se muestra en la figura 4.1. No es necesario el uso de depuradores de código externos. La tarjeta de desarrollo TRK-MPC560XB contiene un programador integrado.

4.2.2 Plataforma de Software.

Para el desarrollo del software se utilizó un IDE (*Integrated Development Environment*) "*S32 Design Studio for Power Architecture*" basado en eclipse, como se muestra en la figura 4.2.

Fig. 4-2 S32 Design Studio IDE for Power Architecture based MCUs de NXP [19].

4.3. Desarrollo del Scheduler Preemptive.

El desarrollo del software del proyecto consta de 2 partes principales: La implementación de un Scheduler Preemptive y la implementación del algoritmo de cambio de contexto optimizado.

4.3.1 Tareas del Scheduler.

Cada tarea del Scheduler contienen código de aplicación diferente. El código de aplicación de las tareas consiste en el incremento de variables contadoras internas del software. Cada tarea incrementa sus contadores en pasos diferentes.

Los contadores de cada tarea son enviados desde el microcontrolador hasta una terminal serial a través de *UART*. Los tiempos en los que envían los valores de los contadores a la terminal

serial varían según la tarea. Los tiempos de ejecución y envíos de los contadores son controlados por las macros que muestran en la figura 4.3.

```
#define TASK1_TIMER_EXPIRED (1000u) /* 1000 * 1mS = 1s */
#define TASK2_TIMER_EXPIRED (3000u) /* 3000 * 1mS = 3s */
#define TASK3_TIMER_EXPIRED (5000u) /* 5000 * 1mS = 5s */
#define TASK4_TIMER_EXPIRED (8000u) /* 8000 * 1mS = 8s */

#define TASK1_TIMER_EXPIRED_SEND_DATA   (100u) /* 100  * 1mS = 100mS */
#define TASK2_TIMER_EXPIRED_SEND_DATA   (250u) /* 250  * 1mS = 250mS */
#define TASK3_TIMER_EXPIRED_SEND_DATA   (500u) /* 500  * 1mS = 500mS */
#define TASK4_TIMER_EXPIRED_SEND_DATA  (1000u) /* 1000 * 1mS = 1s    */
```

Fig. 4-3 Parámetros de control de tiempos de ejecución y envió de datos.

Dependiendo de la tarea que se encuentre en ejecución (estado Running) es activado algún *GPIO* ("*General Purpose Input Output*" por sus siglas en inglés) asignado. Cuando una tarea se encuentra desactivada (estado Suspend) su respectivo GPIO es desactivado. La figura 4.4 muestra el diagrama de flujo de las tareas del Scheduler.

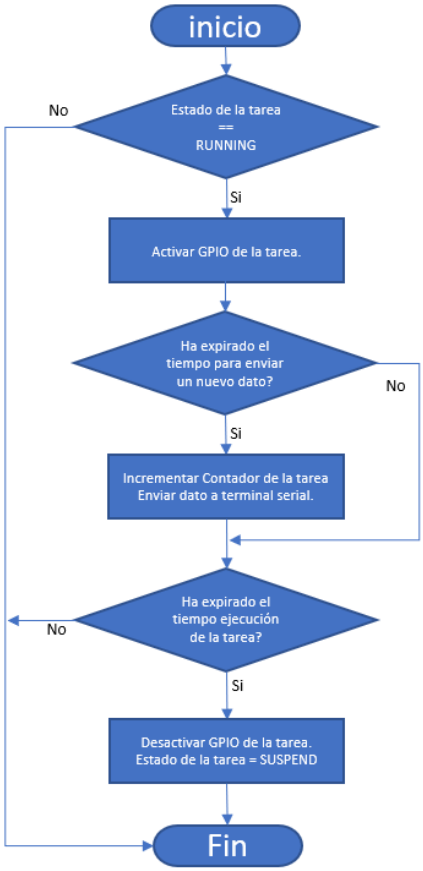


Fig. 4-4 Diseño de las tareas.

Las características de cada tarea del Scheduler se define a continuación.

- **Tarea 1:**
    - Tiempo de ejecución: 1 segundo.
    - Prioridad 4.
    - Cuando la tarea está en estado Running decrementa en uno su contador cada 100 ms y lo envía a la terminal serial.
    - Cuando la tarea está en estado Running activa su GPIO asignado.
    - Cuando la tarea está en estado Suspend o Ready desactiva su GPIO asignado.
- **Tarea 2:**
    - Tiempo de ejecución: 3 segundos.
    - Prioridad 3.
    - Cuando la tarea está en estado Running incrementa en uno su contador cada 250 ms y lo envía a la terminal serial.
    - Cuando la tarea está en estado Running activa su GPIO asignado.
    - Cuando la tarea está en estado Suspend o Ready desactiva su GPIO asignado.
- **Tarea 3:**
    - Tiempo de ejecución: 5 segundos.
    - Prioridad 2.
    - Cuando la tarea está en estado Running incrementa en uno su contador cada 500 ms y lo envía a la terminal serial.
    - Cuando la tarea está en estado Running activa su GPIO asignado.
    - Cuando la tarea está en estado Suspend o Ready desactiva su GPIO asignado.
- **Tarea 4:**
    - Tiempo de ejecución: 8 segundos.
    - Prioridad 1.
    - Cuando la tarea está en estado Running incrementa en uno su contador cada 1000 ms y lo envía a la terminal serial.
    - Cuando la tarea está en estado Running activa su GPIO asignado.

- Cuando la tarea está en estado Suspend o Ready desactiva su GPIO asignado.

La prioridad de las tareas fue asignada según su tiempo de ejecución. Cuanto menor es el tiempo de ejecución mayor es la prioridad de la tarea.

4.3.2 Implementación de las tareas.

El desarrollo del Scheduler Preemptive se basó en la implementación de 4 tareas principales y una tarea *IDLE*. Las 4 tareas principales tienen tiempos de ejecución y prioridades diferentes entre cada una, como se muestra en la figura 4.5. Cada tarea es ejecutada por eventos externos. Los eventos externos son comandos específicos de una terminal serial. La terminal serial se comunica al microcontrolador MPC560XB a través del puerto *UART*.

```
/*************************************************************
 *  Name                : os_init
 *  Description         : This functions inits all the scheduler´s task whit their
 *                        respective ID, stack, priority.
 *  Parameters          : void
 *  Return              : void
 *  Critical/explanation : No
 *************************************************************/
void os_init(void)
{
                /*Ptr to function*/ /*Task stack direction*/ /*Task stack size*/ /*Task Prior*/ /*Task ID*/
    os_task_init(   &TASKS_LIST_1S,         stack1,                 64,             4,          TASKS_1);
    os_task_init(   &TASKS_LIST_3S,         stack2,                 64,             3,          TASKS_2);
    os_task_init(   &TASKS_LIST_5S,         stack3,                 64,             2,          TASKS_3);
    os_task_init(   &TASKS_LIST_8S,         stack4,                 64,             1,          TASKS_4);
    os_task_init(   &TASK_LIST_NULL,        stack5,                 64,             0,          TASK_NULL);
}
```

Fig. 4-5 Inicialización de las tareas.

4.3.3 Tick del Scheduler.

El Scheduler requiere una implementación para generar el *Tick*. El *Tick* es utilizado por el Scheduler para controlar los tiempos de las tareas y la ejecución del dispatcher. Para generar el *Tick* fue utilizado el *PIT* (*Periodic Interrupt Timer* por sus siglas en inglés) del microcontrolador. La interrupción del *Tick* es generada cada 1 ms. En la interrupción del *Tick* se realiza la llamada mediante de la función *callback* del Scheduler, como se muestra en la figura 4.6. La función *callback* contiene el algoritmo Preemptive del Scheduler.

```
/*****************************************************************************/
/**
* \brief    Scheduler Start - Once time base is armed, start execution of   \n
           Multi-thread Round Robin scheduling scheme.                       \n
           This function requires prior execution of "vfnScheduler_Init"
* \author
* \param    void
* \return   void
* \
*/
void vfnScheduler_Start(void)
{
    vfn_init_PIT3_isr(vfnScheduler_Callback);
}
```

Fig. 4-6 Tick del Scheduler.

4.3.4 Activación de las tareas.

Cada tarea del Scheduler es ejecutada por eventos a través del puerto *UART*. Los caracteres enviados a través de la terminal serial seleccionarán cual tarea cambiará a estado Ready. La función es llamada *vfnRequest_TaskActivate*. Los caracteres permitidos son 1, 2, 3 y 4 para cambiar el estado de las tareas, respectivamente.

La función es colocada al principio del *callback* del Scheduler. La implementación de la función *vfnRequest_TaskActivate* se muestra en la figura 4.7.

```
/*****************************************************************
 *  Name                : vfnRequest_TaskActivate
 *  Description         : This function change the state to ready of each task
 *                        depending of the character received through UART.
 *  Parameters          : void
 *  Return              : void
 *  Critical/explanation : No
 *****************************************************************/
void vfnRequest_TaskActivate(void)
{
    /*Clear variable before read*/
    uint8_t getUARTchar = 0x00;

    /* Read UART and store the character*/
    /* TODO: UART is polling mode, change to ISR */
    ReadUARTPoll(&getUARTchar);

    /*Depending of the character that was received from UART, put in ready the respective task */
    switch(getUARTchar)
    {
        case '1': vfnScheduler_TaskActivate(&m_task_table.tasks[TASKS_1]); break;
        case '2': vfnScheduler_TaskActivate(&m_task_table.tasks[TASKS_2]); break;
        case '3': vfnScheduler_TaskActivate(&m_task_table.tasks[TASKS_3]); break;
        case '4': vfnScheduler_TaskActivate(&m_task_table.tasks[TASKS_4]); break;
        default:
            /* Do nothing */
        break;
    }
}
```

Fig. 4-7 Activación de tareas por eventos del UART.

4.3.5 Algoritmo del scheduler.

El algoritmo principal para el Scheduler Preemptive esta implementado en la función *vfnScheduler_Callback*. En la función *vfnScheduler_Callback* el Scheduler decide cual será la siguiente tarea por ejecutar y cuánto es el tiempo de ejecución restante de cada una. En la figura 4.8 se muestra la implementación de la función *vfnScheduler_Callback*.

```
/*****************************************************************************/
/**
* \brief    Periodic Interrupt Timer Service routine.
            This interrupt is the core of the task scheduler.
* \author
* \param    void
* \return   void
*****************************************************************************/
void vfnScheduler_Callback(void)
{
    /*To know which task will be executed*/
    vfnRequest_TaskActivate();

    os_curr_task = &m_task_table.tasks[m_task_table.current_task];

    vfnIncrements_task_counters();

    if(m_task_table.current_task != vfnScheduler_Dispacher())
    {
        os_curr_task = os_next_task;
        if(os_curr_task->enTaskState == RUNNING)
        {
            os_curr_task->enTaskState = READY;
        }
        m_task_table.current_task = vfnScheduler_Dispacher();
        os_next_task = &m_task_table.tasks[m_task_table.current_task];
        os_next_task->enTaskState = RUNNING;
    }
}
```

Fig. 4-8 Algoritmo del Scheduler.

4.4. Implementación del cambio de contexto.

El cambio de contexto esta divido en 2 partes: guardar contexto y restaurar contexto. Guardar el contexto de una tarea se refiere a guardar todos los registros importantes del procesador. Algunos registros son guardados de forma automática por el procesador, pero otros se deben guardar de forma manual por el programador. En el Scheduler Preemptive se guarda contexto antes de saltar a la siguiente tarea. Restaurar contexto se refiere a recuperar los valores de los registros del procesador que se guardaron previamente. Cuando se restaura contexto es porque se busca regresar a la tarea que fue interrumpida previamente. La figura 4.9 muestra cómo es el comportamiento cuando se cambia de tarea en un Scheduler Preemptive.

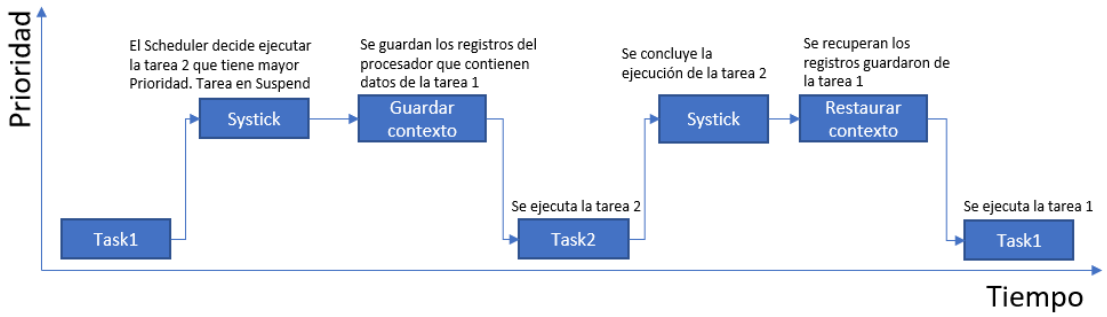


Fig. 4-9 Cambio de tareas.

Como se mencionó en el capítulo 3.5 el proceso para guardar y restaurar contexto de una tarea se realiza dentro de una excepción IVOR. La IVOR utilizada para el Scheduler es la IVOR8 *System call.* Un IVOR8 ocurre cuando una instrucción de llamada de sistema es ejecutada y no existe una excepción de mayor prioridad.

Dentro del IVOR8 existen dos partes principales: El prólogo y el epilogo. Dentro del prologo se realiza el guardado del contexto de la tarea y del epílogo la restauración del contexto de la tarea. La figura 4.10 muestra los pasos dentro del IVOR para el prólogo y el epílogo.

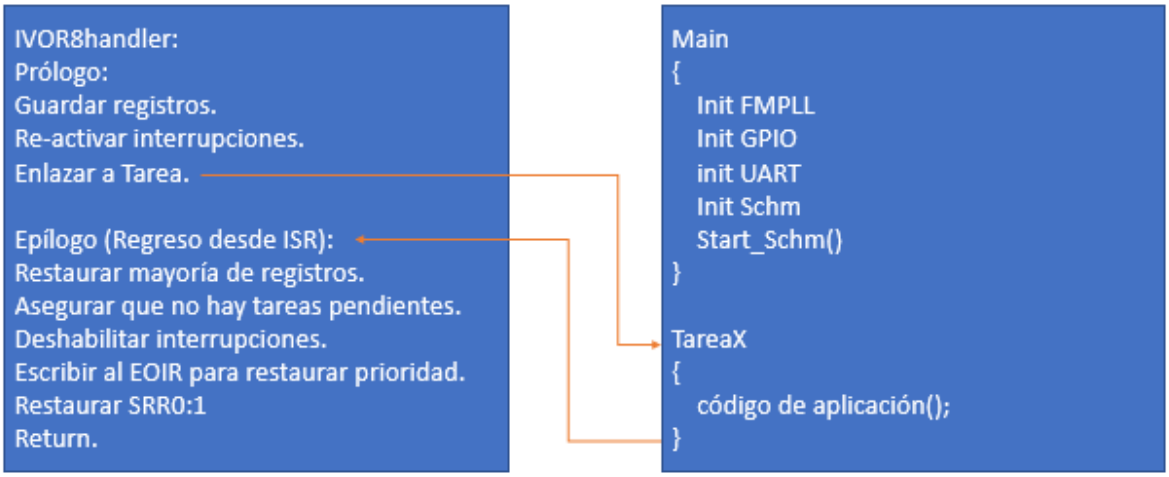


Fig. 4-10 Cambio de contexto en IVOR8

5. Resultados.

En este apartado se evalúan los resultados de la implementación del Scheduler Preemptive. Las tareas de aplicación del Scheduler deben ejecutarse de forma correcta. Los datos impresos en la terminal serial deben ser los esperados según las tareas ejecutadas. Los GPIO demuestran los tiempos de ejecución de las tareas.

La figura 5.1 muestra el comportamiento de las tareas del Scheduler. Se utilizó un analizador lógico *Saleae* conectado a los cuatro GPIOs asignados a cada tarea. Cada tarea es activada por los comandos a través de la terminal serial.

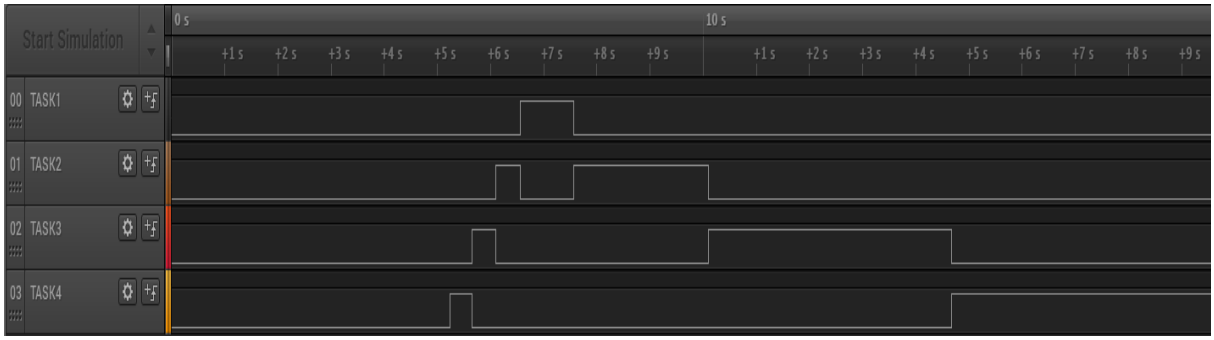


Fig. 5-1 Ejemplo del comportamiento de las tareas.

El comportamiento de las tareas se describe a continuación:

- La tarea 4 es la primera en comenzar su ejecución y pasa a estado Running.
- La tarea 4 es interrumpida por la tarea 3. Por lo que tarea 4 pasa a un estado Suspend.
- La tarea 3 pasa a estado Running y comienza su ejecución.
- La tarea 3 es interrumpida por la tarea 2. Por lo que tarea 3 pasa a un estado Suspend.
- La tarea 2 pasa a estado Running y comienza su ejecución.

- La tarea 2 es interrumpida por la tarea 1. Por lo que tarea 2 pasa a un estado Suspend.

La tarea 1 pasa a estado Running y comienza su ejecución. Dado que la tarea 1 es la de mayor prioridad, la tarea 1 termina su tiempo de ejecución de 1 segundo y pasa a estado Suspend como se muestra en la figura 5.2.

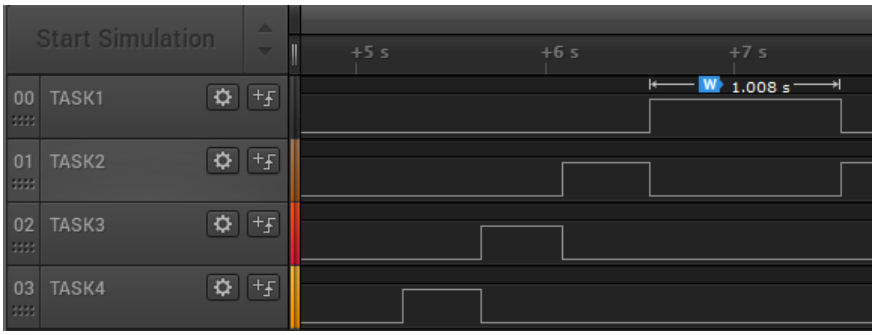


Fig. 5-2 Tiempo de ejecución de la tarea 1.

La tarea 2 reanuda su ejecución hasta completar su tiempo de 3 segundos y pasa a estado Suspend como se muestra en la figura 5.3.

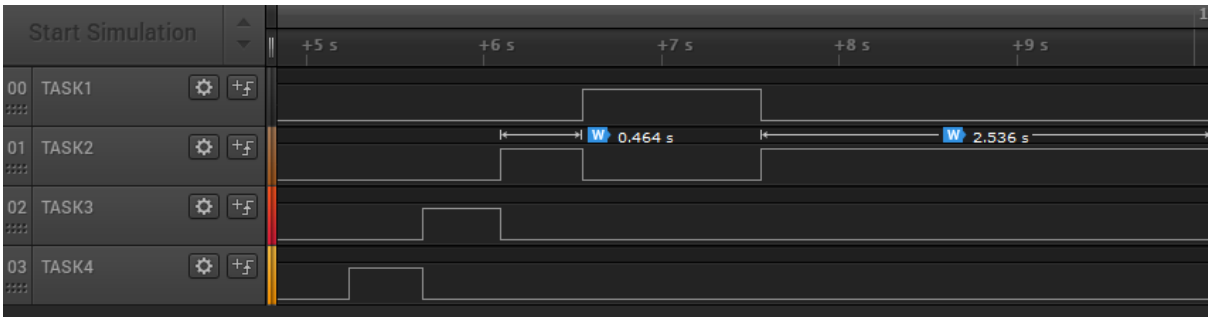


Fig. 5-3 Tiempo de ejecución de la tarea 2.

La tarea 3 reanuda su ejecución hasta completar su tiempo de 5 segundos y pasa a estado Suspend como se muestra en la figura 5.4.

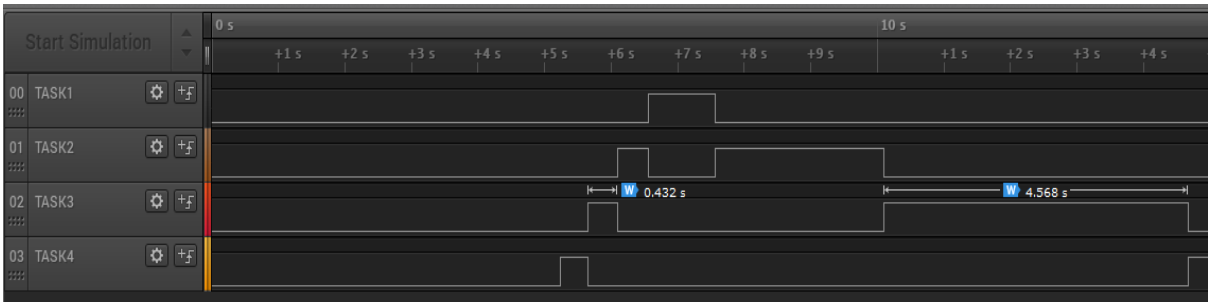


Fig. 5-4 Tiempo de ejecución de la tarea 3.

La tarea 4 reanuda su ejecución hasta completar su tiempo de 8 segundos y pasa a estado Suspend como se muestra en la figura 5.5.

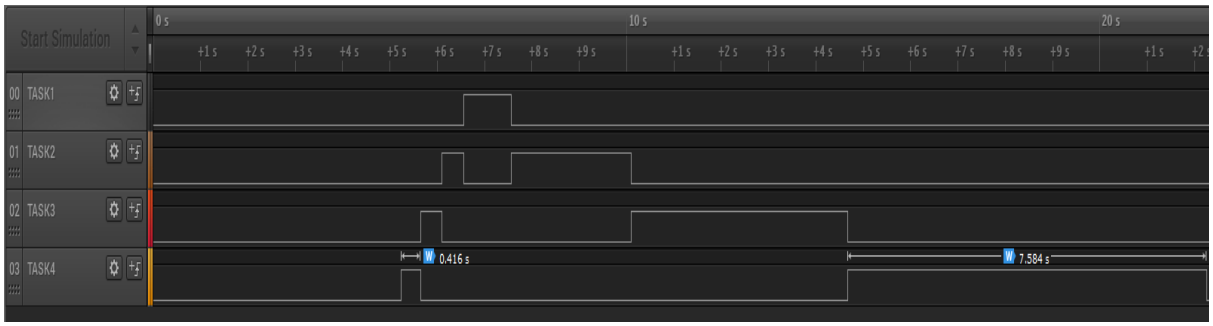


Fig. 5-5 Tiempo de ejecución de la tarea 4.

El tiempo de ejecución del Scheduler debe ser la suma de todas las tareas. La suma del tiempo de ejecución de todas las tareas es: 1 s + 3 s + 5 s + 8 s = 17 s. La figura 5.6 muestra el tiempo total de ejecución del Scheduler.

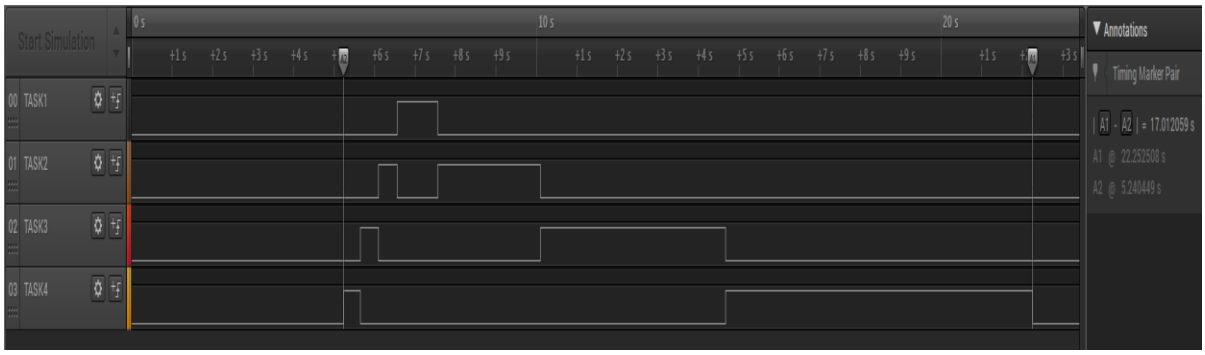


Fig. 5-6 Tiempo de ejecución del Scheduler.

Cuando una tarea termina su ejecución también cambia su estado. El cambio de estado de las tareas se realiza durante el SysTick de 1 ms. La separación de las tareas debe ser aproximadamente de 1 ms entre ellas, que en este caso coincide con el SysTick. La figura 5.7 muestra la separación de las tareas después de ejecutar el SysTick donde se encuentra el callback del Scheduler.

Fig. 5-7 Separación de las tareas entre ejecuciones.

Los valores de los contadores de cada tarea son desplegados en una terminal serial. La interfaz de usuario principal se muestra en la figura 5.8. El usuario puede indicar, a través de la terminal serial, la tarea a ejecutar.

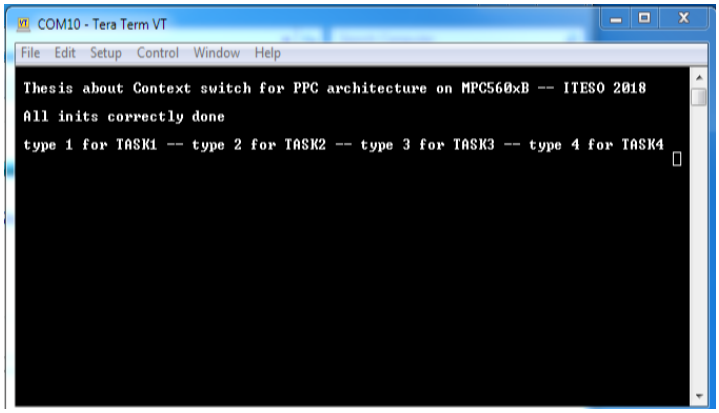


Fig. 5-8 Interfaz de usuario en terminal serial.

Los contadores de las tareas se muestran en la terminal serial siguiendo la misma secuencia de ejecución que se muestra en la figura 5.1. Los contadores siguen las reglas que se indican en el capítulo 4.3.1.

La figura 5.9 muestra cómo se imprimen los valores de los contadores en la terminal serial según el orden de ejecución de las tareas de la figura 5.1.

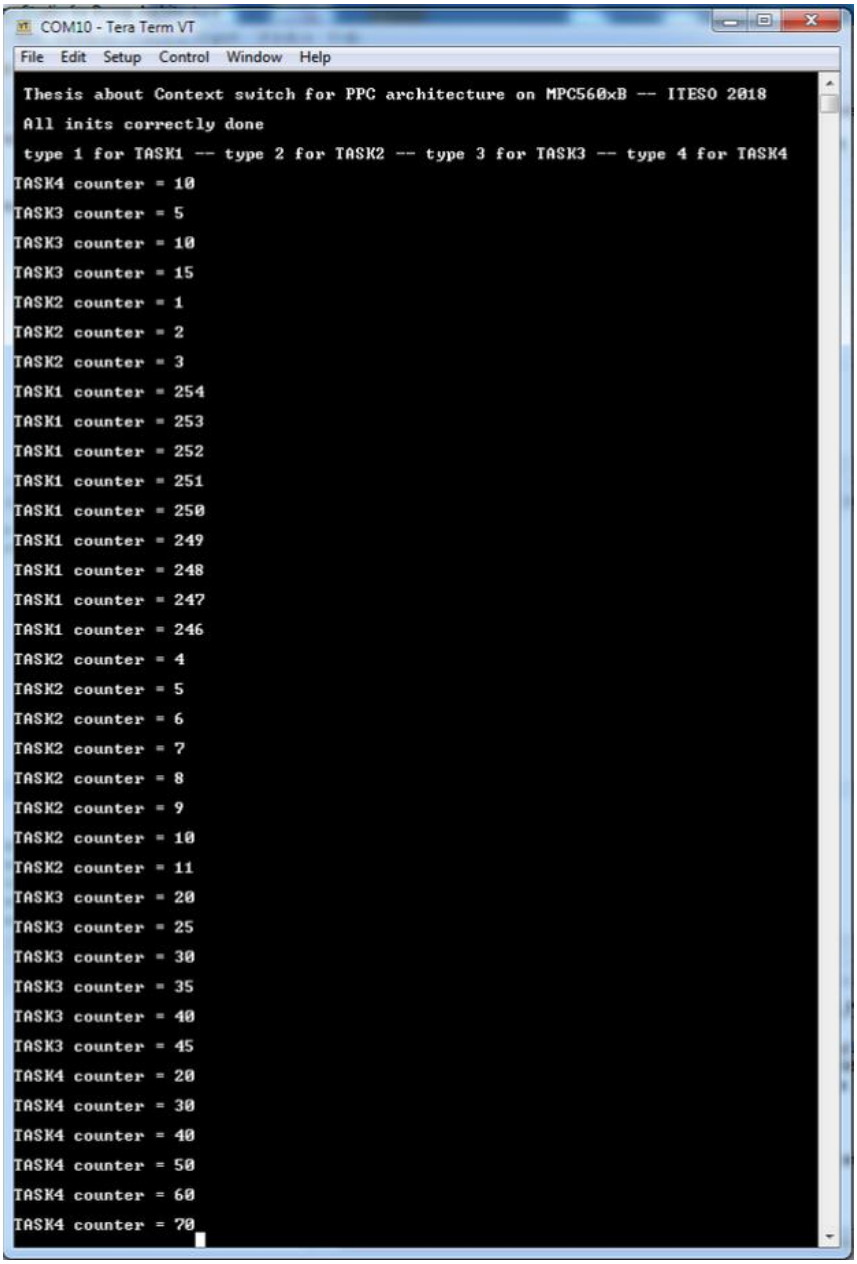


Fig. 5-9 Contadores de las tareas en monitor serial.

En el Scheduler que se presenta, las prioridades y los tiempos de ejecución de las tareas pueden ser modificados. Cambiar la prioridad y tiempo de ejecución de las tareas no provocará que el Scheduler falle. El Scheduler Preemptive que se presenta no puede ejecutar la misma tarea de forma consecutiva debido al diseño del dispatcher.

6. Conclusiones.

Con base en los resultados se tienen tres puntos importantes para la implementación del cambio de contexto. El primer punto es identificar los registros del procesador que se requieren guardar. El segundo, es definir el tamaño del stack para cada una de las tareas del Scheduler. El tercero, es identificar cuál es la IVOR (para el caso del PowerPC) que se utilizará en la implementación. Los puntos 2 y 3 dependen de la aplicación que tendrán las tareas del Scheduler.

A la implementación de cambio de contexto se pueden agregar los registros de punto flotante, pero se definió como requerimiento que no se respaldaran dichos registros.

La parte de la implementación más difícil fue definir el procedimiento completo para el cambio de contexto de las tareas, es decir, establecer el proceso del algoritmo Preemptive. Establecer las instrucciones en ensamblador para el cambio de contexto también fue difícil ya que no hay mucha información de apoyo sobre el PowerPC, la mayor parte de ella son documentos técnicos generales para cualquier microcontrolador de la familia, pero muy poco específico al que se encuentra en nuestra tarjeta de desarrollo o su núcleo.

Activar las tareas del Scheduler por eventos externos dificulta el proceso para guardar contexto de las tareas interrumpidas. La implementación del Scheduler puede ser cambiada de eventos externos a eventos internos del software y aprovechar sus interrupciones.

Como regla de diseño se debe tener cuidado con las tareas que se ejecutan durante el SysTick (tareas cíclicas), ya que estas están diseñadas para tener un tiempo máximo de ejecución definido desde la arquitectura del sistema operativo, que no debe exceder el tiempo de ejecución periódico al que está configurado el Tick del Scheduler para evitar retrasos en la ejecución de las tareas siguientes.

El tamaño del stack de las tareas puede ser definido en un Heap dentro del Linker. En el Scheduler que se presenta se define mediante el uso de RAM, inicializando variables globales.

La optimización del cambio de contexto depende de la aplicación que se tenga para el software, un ejemplo de esto es, como se menciona previamente, el hecho de que los a registros de punto flotante no fueron incluidos en este trabajo.

La arquitectura PowerPC no dispone de una implementación de hardware dedicado al manejo de cambio de contexto, por lo que ésta se requiere realizar mediante software, atendiendo esta necesidad durante la activación de una interrupción o desde el sistema operativo para cambiar la tarea en ejecución. Por este motivo fue un punto de ayuda en la realización de este trabajo basarnos en la implementación del cambio de contexto para las interrupciones externas (IVOR4), ya que sirvió como base para el diseño de nuestra estrategia.

7. Bibliografía.

[1] Wikimedia Foundation, Inc, «Wikipedia,» Wikimedia Foundation, Inc, 15 11 2017. [En línea]. Available: https://en.wikipedia.org/wiki/AIM_alliance. [Último acceso: 4 6 2018].

[2] Robert Bosch GmbH, «Annual Report,» Robert Bosch GmbH, Stuttgart, Germany, 2018.

[3] Continental Aktiengesellschaft, «Solutions for Clean Air.2017 Annual Report,» Continental Aktiengesellschaft, Hanover, Germany, 2018.

[4] HELLA KGaA Hueck & Co, «Annual Report,» HELLA KGaA Hueck & Co, Lippstadt, Germany, 2018.

[5] DELPHI AUTOMOTIVE PLC, «2016 Annual Report,» DELPHI AUTOMOTIVE PLC, Gillingham, United Kingdom, 2017.

[6] Valeo, «2017 CONSOLIDATED FINANCIAL STATEMENTS,» Valeo, Paris, Francia, 2018.

[7] Arm Limited, «arm,» 31 07 2017. [En línea]. Available: https://www.arm.com/. [Último acceso: 4 6 2018].

[8] J. Carlton, «Apple : the inside story of intrigue, egomania, and business blunders,» London, Random House Business Books, 1999.

[9] IEEE, «IEEE Xplore Digital Library,» 2018. [En línea]. Available: https://ieeexplore.ieee.org/Xplore/home.jsp. [Último acceso: 4 Junio 2018].

[10] U. L. S. Tuomenoksa and W, «Problems of Programming for Shared Real-Time Systems,» *IEEE Transactions on Communication Technology*, vol. 15, nº 1, pp. 5 - 10, Febrero 1967.

[11] R. A. Reid, «Task context switching RTOS». United States of America Patente US7434222B2, 20 Diciembre 2001.

[12] D. M. K. K. Juraj Bystricky, «System and method for performing an interface save/restore procedure». United States of America Patente US20060288299A1, 15 Junio 2005.

[13] M. A. M. H. P. L. T. N. J. R. V. A. A. F. a. A. R. R. Seungyoon Peter Song, «Efficient context saving and restoring in a multi-tasking computing system environment». United States of America Patente US6061711A, 19 Agosto 1996.

[14] H. I. Nayberg, «Method and System for Task Switching with Inline Execution». United States of America Patente US20090217290A1, 21 Febrero 2008.

[15] S. S. a. K. Burke, «Power PC Embedded Application Binary Interface (EABI): 32-Bit Implementation,» Freescale Semiconductor, Inc., 2004.

[16] S. Z. y. K. Karhi, «SYSTEM V APPLICATION BINARY INTERFACE. PowerPC Processor Supplement,» Sun Microsystems, Inc., California, U.S.A., 1995.

[17] T. Lewin, «Automotive News,» Crain Communications, Inc., 21 Noviembre 2005. [En línea]. Available: http://www.autonews.com/article/20051121/SUB/511210712/study:-higher-r&d-spending-can-mean-more-profits. [Último acceso: 23 Mayo 2018].

[18] Freescale Semiconductor, Inc., «MPC5606BK Microcontroller - Reference Manual,» Freescale Semiconductor, Inc., Japón, 2014.

[19] NXP Semiconductors, «S32DS-PA: S32 Design Studio IDE for Power Architecture based MCUs,» NXP Semiconductors, [En línea]. Available: https://www.nxp.com/support/developer-resources/run-time-software/s32-design-studio-ide/s32-design-studio-ide-for-power-architecture-based-mcus:S32DS-PA. [Último acceso: 23 6 18].