CENTRO POLITÉCNICO SUPERIOR

UNIVERSIDAD DE ZARAGOZA

PROYECTO FIN DE CARRERA

Migración del sistema operativo de tiempo real MaRTE OS al microprocesador ARM

Para acceder al título de

INGENIERO INFORMÁTICO

Departamento de Informática e Ingeniería de Sistemas

Autor: Luis Canales Mayo
Director: Jose Luis Villarroel Salcedo

Zaragoza, Septiembre de 2010

Quisiera agradecer a Jose Luis Villarroel la oportunidad que me ha dado al permitirme realizar este Proyecto Fin de Carrera. Su paciencia y su labor como director de este proyecto han sido cruciales.

Agradezco también a Mario Aldea Rivas todos los consejos y ayudas que me ha ido dando sobre MaRTE OS, sin los cuales el tiempo de dedicación de este Proyecto Fin de Carrera se habría sido enorme.

Por último, me gustaría agradecer a mi familia su apoyo en estos momentos de dificultad, gracias al cual mi moral y mi dedicación ante los sucesivos problemas surgidos en el proyecto no ha decaído.

RESUMEN

Palabras clave: MaRTE OS, sistema operativo de tiempo real, ARM, compilación cruzada, GNAT, GCC, Ada, driver, Embest, periféricos

MaRTE OS (*Minimal Real-Time Operating System for Embedded Applications*) es un sistema operativo de tiempo real que implementa los servicios definidos en el estándar POSIX.13. En general, el objetivo del estándar POSIX es proporcionar un API a las aplicaciones para abstraerlas del sistema operativo que hay debajo, con el fin de facilitar la migración entre los diferentes sistemas operativos que se ajusten a dicho estándar. En concreto, POSIX.13 está orientado a perfiles de entornos de aplicación de tiempo real, como es nuestro caso, y constituye una versión reducida de POSIX.

Actualmente, MaRTE OS ofrece soporte para aplicaciones escritas en C o Ada. El núcleo puede ser compilado junto con dichas aplicaciones para funcionar como proceso de Linux, o para ser ejecutado directamente sobre algunos procesadores de la familia Intel X86 (486, Pentium I y Pentium II). El objetivo final al que apunta este proyecto es a la migración total de MaRTE OS al microprocesador ARM7tdmi (dispuesto en el microcontrolador S3C44B0X), con el fin de ejecutar en él aplicaciones bajo este sistema operativo.

MaRTE OS está hecho para ser compilado desde Linux. Es por ello por lo que se ha elegido este último sistema operativo como base para la generación de todo el entorno de desarrollo. Dado que MaRTE OS está hecho en C y Ada ha sido necesario elaborar herramientas de compilación cruzada (GCC y Gnat), de forma que ha sido posible compilar desde Linux, alojado en un PC con procesador de la familia x86, aplicaciones que posteriormente iban a ser ejecutadas en un procesador distinto (el ARM7tdmi).

Una vez obtenido el entorno de desarrollo cruzado, se pudo proceder a la programación de la parte de más bajo nivel de MaRTE OS. Esta parte se denomina "interfaz abstracta con el hardware", y proporciona al resto del sistema operativo una visión abstracta de la plataforma sobre la que se está ejecutando. La tarea principal ha sido, pues, reprogramar las rutinas de esta interfaz (definida en un único fichero) satisfaciendo sus dependencias con el hardware. Usando los periféricos dispuestos en la placa utilizada (S3CEV40) se ha conseguido el comportamiento especificado para cada una de estas rutinas, las cuales tienen que ver con la carga/guardado de algunos registros del procesador, dehabilitación/habilitación de interrupciones, temporizaciones, etc...

Adicionalmente se han implementado las rutinas de inicialización de los periféricos asociados a la interfaz abstracta con el hardware de MaRTE OS. Además, se proporcionan algunas rutinas adicionales como el manejo de la interrupción de reseteo. Finalmente, se ha incluido una batería de pruebas consistente sobre la interfaz abstracta con el hardware, así como algunos *scripts* que permiten compilar todos los *tests*.

Tras compilar en Linux nuestras aplicaciones y generar los ejecutables correspondientes es necesario transferirlos a la placa utilizada. Para ello se proporciona un entorno, llamado Embest, que ofrece facilidades para cargar y depurar nuestras aplicaciones en el microcontrolador que se ha usado. El entorno Embest funciona bajo Windows, por lo que es necesario transferir los ejecutables desde Linux (donde hemos compilado nuestra aplicación) a este otro sistema operativo.

Indice de contenidos

Capítulo 1

Introducción

El objetivo de este primer capítulo es dar una idea general acerca del origen del proyecto, ponerlo en contexto y detallar la estructura general de esta memoria.

1.1.- Antecedentes.

La idea de este proyecto surge a partir de la asignatura "Sistemas de tiempo real". Es una asignatura optativa de los planes de estudios de Ingeniería Informática, Ingeniería en Telecomunicaciones e Ingeniería Industrial, impartidas actualmente en la Universidad de Zaragoza.

Actualmente en las prácticas [1] de esta asignatura cada alumno utiliza un PC con el entorno GPS (Gnat Programming Studio) bajo Windows XP. Debido a que Windows XP no es un sistema operativo de tiempo real, es necesario hacer uso de aplicaciones como "TQCRunas" para simular, en la medida lo posible, que las aplicaciones programadas en estas prácticas se ejecutan en tiempo real. "TQCRunas" proporciona la prioridad más alta posible (la de administrador) a los procesos que designemos, pero aún así Windows posee algunas tareas con prioridad incluso más alta que la del administrador, con lo que las aplicaciones nunca se van a poder ejecutar con carácter 100% tiempo real.

Existe un sistema operativo de tiempo real, MaRTE OS [2], el cual sirve actualmente de soporte para la programación de aplicaciones de tiempo real con el fin de ejecutarlas sobre PC (sin sistema operativo o bajo Linux). Afortunadamente el Centro Politécnico Superior dispone de una serie de placas de desarrollo (S3CEV40) que hacen uso de un microcontrolador (S3C44B0X) basado en núcleo ARM. En conjunto todo ello podría ser una buena alternativa al sistema actual de ejecución de aplicaciones bajo PC con Windows con requerimientos de tiempo real. Por ello se ha pensado que la generación de un entorno cruzado capaz de migrar MaRTE OS al microprocesador ARM podría ser una buena idea de cara al desarrollo de las prácticas de la asignatura “Sistemas de tiempo real”.

1.2.- Los sistemas operativos de tiempo real.

Los sistemas operativos más comunes, tales como Windows o Linux, tienen como función principal la de abstraer a las aplicaciones de usuario de cada uno de los dispositivos hardware internos al computador. Esto puede realizarse ofertando un API (interfaz con otros programas) o mediante una GUI más o menos atractiva (interfaz directa con el usuario). Este tipo de sistemas se denominan "sistemas interactivos".

Un sistema operativo de tiempo real no es más que un subconjunto de los sistemas operativos comunes, añadiendo a las características fundamentales de estos últimos (que las acciones sean correctas) una serie de restricciones temporales. Esto ha venido dado por la necesidad en algunos sistemas, como el de control del vuelo de un avión, de ser capaces de dar respuesta a los distintos eventos que van sucediendo en el entorno en un tiempo acotado

por una serie de plazos temporales.

Por lo general los sistemas de tiempo real reaccionan ante estímulos externos, realizando acciones en consecuencia a los mismos en tiempo acotado. Fuera de ese tiempo, el sistema no garantiza un correcto funcionamiento. Las características de cualquier sistema de tiempo real pueden resumirse en estas 5:

1. Determinismo temporal. El sistema debe responder correctamente ante cualquier situación posible en un intervalo de tiempo determinado. Es necesario para ello considerar todos los casos, incluido el de mayor tiempo de respuesta.
2. Fiabilidad y seguridad. Garantizar que si el sistema se encuentra ante un fallo realice las acciones oportunas para quedar en un estado seguro.
3. Concurrencia. Si el sistema recibe más de 2 o más estímulos externos a la vez debe ser capaz de controlarlos todos simultáneamente. Las acciones de control pueden generarse de manera concurrente usando varios procesadores o de manera secuencial usando tan solo 1 procesador (simular la concurrencia).
4. Interacción con dispositivos físicos. El sistema recibe mediante sus dispositivos de entrada estímulos externos, elabora una respuesta a los mismos y la emite por los dispositivos de salida.
5. Robustez. Las condiciones en las que el sistema trabaja no siempre son las óptimas y hay que tenerlo en cuenta.

No siempre las restricciones temporales impuestas a un sistema de tiempo real tienen por que seguirse con rigidez. En función de lo críticas que pueden llegar a ser las restricciones temporales podemos clasificar estos sistemas en 2 tipos:

- Sistemas de tiempo real críticos: las restricciones temporales deben respetarse siempre, con el fin de que el sistema siempre reaccione a los estímulos externos de forma correcta. Un ejemplo es el del sistema de control de un avión, donde el control es crítico.
- Sistemas de tiempo real acríticos: permiten un margen de tolerancia en cuanto a las restricciones temporales, debido a que no son tan críticos. Por ejemplo, en una videoconferencia el sistema puede permitirse el retraso en alguna de las tramas de audio/vídeo, incluso la pérdida de alguna de ellas.

1.2.1.- MaRTE OS.

MaRTE OS (*Minimal Real Time Operating System for Embedded Applications*) es el sistema operativo elegido como objeto fundamental de este proyecto. Su diseño e implementación, realizados por Mario Aldea Rivas y Michael Gonzalez Harbour (ambos pertenecientes a la Universidad de Cantabria), fueron liberados en el año 2000 bajo licencia GNU/GPL2 [3]. La primera versión de esta licencia fue creada en 1989 por la *Free Software Foundation* y aboga por la libre distribución, modificación y uso del *software* de manera libre, sin posibilidad de apropiación del mismo.

El sistema operativo MaRTE OS implementa todo lo referente al subconjunto más pequeño del estándar POSIX.13 (por ello se considera un sistema operativo de tiempo real mínimo), el cual corresponde a una parte del estándar POSIX [4]. Gracias a que el sistema operativo implementa el API especificado por POSIX.13, las aplicaciones programadas bajo él podrán ser portadas a otros sistemas que a su vez también lo implementen.

1.3.- Microprocesadores, microcontroladores y DSP's.

Un sistema empotrado de tiempo real es, a su vez, un subconjunto de los sistemas de tiempo real. En este caso también se presentan restricciones temporales, pero el sistema empotrado forma parte de un sistema aún mayor al cual ayuda de algún modo realizando una determinada funcionalidad. Por ejemplo, el *airbag* de un coche es un circuito que realiza una determinada función (proporciona seguridad), pero forma parte del sistema completo del vehículo. En caso de que el sistema no tenga restricciones temporales se denomina, simplemente, "Sistema empotrado". A las características de un sistema de tiempo real se añaden otras cuando hablamos de este tipo de sistemas, debido a su reducido tamaño:

1. Bajo consumo. Gracias a que estos sistemas consumen poco (están alimentados a veces con baterías o pilas) gozan de gran autonomía.
2. Bajo peso y pequeñas dimensiones. Normalmente son sistemas portátiles (p. ej. teléfonos móviles).
3. Bajo precio. Consecuencia de lo anterior.

A la hora de implementar un sistema empotrado (o de tiempo real), se necesita un soporte físico con unas determinadas características. Los microcontroladores y los DSP's son los tipos de soporte existentes que permiten asegurar los requerimientos de algunos sistemas de tiempo real y la mayoría de los sistemas empotrados.

Los microcontroladores son un tipo particular de microprocesador, es decir, un circuito integrado diseñado para aplicaciones de propósito general con una serie de buses y puertos E/S. Sin embargo, los microcontroladores disponen además de una pequeña memoria interna (ROM, RAM, *Flash*, etc.) y una serie de periféricos incorporados en el mismo *chip* que lo hacen más completo que un microprocesador.

Los DSP's son un tipo particular de microprocesador diseñado para realizar operaciones numéricas a muy alta velocidad. Es útil para el procesado digital de la señal y suele realizar un uso exhaustivo de conversores A/D. Suelen disponer de algunos temporizadores y líneas serie de alta velocidad.

1.3.1.- La placa de desarrollo S3CEV40.

La placa de desarrollo S3CEV40 [5] ha sido el soporte físico sobre el que se ha realizado la migración del sistema operativo MaRTE OS. Esta compuesta por un microcontrolador (el S3C44B0X de Samsung basado en ARM) y otros elementos adicionales: una pantalla LCD táctil, algunos pulsadores, interfaces Ethernet/USB para alimentación, una interfaz JTAG para carga de aplicaciones, 2 puertos serie para depurar programas, un visualizador de 8 segmentos e interfaces para E/S de sonido. Puede verse un esquema de dicha placa en la figura 1.3.1.1.

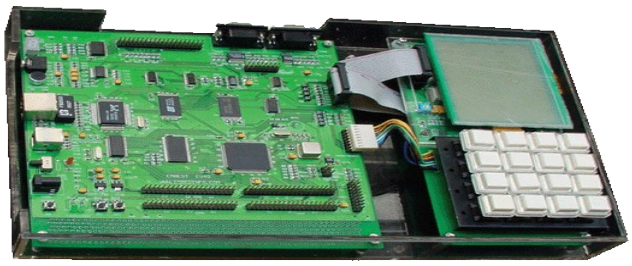

Figura 1.3.1.1.- Placa de desarrollo S3CEV40.

1.4.- El entorno cruzado.

En nuestro caso, MaRTE OS va a ser compilado desde el sistema operativo Linux (usando como arquitectura un Intel Pentium), dado que no puede compilarse directamente desde el sistema empotrado utilizado (el S3C44B0X). Debido a que la arquitectura destino (ARM) es diferente a la arquitectura desde la que se compila (PC con Linux) no basta con utilizar un compilador tradicional. Será necesario usar un compilador que genere código ejecutable en una arquitectura diferente de la arquitectura desde la que se está compilando, es decir, un compilador cruzado. En nuestro caso se ha obtenido una versión de GCC [6] que soporta aplicaciones escritas en C y Ada [7] y es capaz de traducir programas desde nuestro Linux sobre PC (también llamado Host) a código ejecutable en el ARM (denominado Target).

El fabricante de la placa S3CEV40 proporciona junto con esta el entorno Embest, con el que poder cargar ejecutables en el microcontrolador S3C44B0X. Debido a que este entorno requiere Windows, será necesario copiar el ejecutable obtenido con el compilador cruzado desde Linux a este otro sistema operativo y transferirlo con dicho entorno al microcontrolador. El entorno de desarrollo se analizará con mayor detalle en los siguientes capítulos.

1.5.- Objetivos del proyecto.

Los objetivos de este proyecto se enumeran a continuación:

1. Estudio de la documentación del microcontrolador S3C44B0X y la documentación del sistema operativo MaRTE OS, esta última disponible en la web de la Universidad de Cantabria.
2. Construcción del entorno de desarrollo cruzado, con el fin de compilar aplicaciones desde el sistema operativo usado como *host* (Linux), permitiendo su ejecución en el microprocesador ARM utilizado. El compilador generado debe soportar aplicaciones escritas en C y Ada.
3. Puesta a punto de las rutinas de arranque del microprocesador. Esto englobará tanto la inicialización de los periféricos mínimos requeridos por MaRTE OS como la programación de la rutina de reseteo del microcontrolador S3C44B0X utilizado.
4. Migración parcial del sistema operativo MaRTE OS. Esto requerirá la implementación correcta de las rutinas del sistema operativo dependientes del hardware relacionadas con la temporización y el control de interrupciones, adaptándolas al microcontrolador S3C44B0X.
5. Estudio de la posibilidad de implementar la parte de MaRTE OS dependiente del hardware referente a tareas.
6. Elaboración de un banco de pruebas consistente.
7. Programación de *scripts* que automaticen la compilación de los ficheros procedentes de la migración de MaRTE OS y las pruebas.

1.6.- Estructura de este documento.

Este documento está dividido en 6 capítulos:

- El capítulo 1 (el actual) ofrece una visión global del proyecto, detallando sus orígenes, sus objetivos y el contenido del resto de capítulos.
- El capítulo 2 destaca las secciones incluidas en el Plan de Gestión del Proyecto Software.
- El capítulo 3 recoge todo lo referido a la fase de análisis de este Proyecto Fin de

Carrera, haciendo hincapié en sus 3 pilares fundamentales: el sistema operativo MaRTE OS, el microcontrolador S3C44B0X y el entorno de desarrollo cruzado.

- El capítulo 4 detalla todo el proceso seguido en la fase de diseño de este Proyecto Fin de Carrera. Se incluye información acerca de la forma en la que ha sido realizada la migración del sistema operativo, como se utilizado el microcontrolador para conseguirla y como se ha abordado la generación del entorno de desarrollo cruzado.
- El capítulo 5 entra en detalle en la fase de implementación seguida en este Proyecto Fin de Carrera. Se recogerán los aspectos de más alto nivel del código fuente implementado, y detalles técnicos del desarrollo.
- El capítulo 6 hace una revisión del trabajo desarrollado y las dificultades afrontadas, estableciendo una serie de conclusiones finales y proponiendo una serie de alternativas de futuro al mismo.

Adicionalmente se incluyen algunos anexos. El anexo A describe la estructura general de los ficheros residentes en la máquina virtual entregada como resultado del proyecto. Su fin es entender el propósito de cada uno de estos ficheros y, en un futuro, poder realizar modificaciones y ampliaciones en ellos con relativa facilidad. El anexo B incluye un resumen del Plan de Gestión del Proyecto Software empleado en el proyecto. El resto de anexos incluyen partes más técnicas de los diferentes capítulos de la memoria y su propósito viene detallado en los apartados a los que hacen referencia.

Capítulo 2

El Plan de Gestión del Proyecto Software.

Este capítulo contiene un resumen de lo detallado en el Plan de Gestión del Proyecto Software, que ofrece una visión general del desarrollo de este Proyecto Fin de Carrera.

El Plan de Gestión del Proyecto Software recopila y documenta cualquier tipo de actividad seguida a lo largo de un proyecto informático y permite controlar el mismo. En este capítulo se incluyen los apartados que han sido desarrollados en el Plan de Gestión del Proyecto Software incluido en el anexo B de este documento:

1. Elementos entregados: incluye una descripción de los elementos entregados a lo largo de este Proyecto Fin de Carrera.
2. El modelo de proceso: contiene una descripción del ciclo de vida empleado en el proyecto y las horas dedicadas a cada una de las fases.
3. Asunciones, dependencias y restricciones: detalla los requerimientos técnicos necesarios para la prueba de los elementos entregados como resultado de este proyecto.
4. Gestión de riesgos: contiene información acerca de los diferentes riesgos identificados, así como sus estrategias de mitigación y si estas han resultado exitosas.
5. Herramientas: incluye una descripción de las herramientas software empleadas para en el desarrollo de este Proyecto Fin de Carrera.
6. Paquetes de trabajo: contiene un esquema con las distintas líneas de trabajo seguidas durante todo el ciclo de vida del proyecto Software.

Capítulo 3

Fase de análisis: elementos principales de este proyecto.

En este capítulo se describen los 3 pilares fundamentales sobre los que se apoya este Proyecto Fin de Carrera: el sistema operativo MaRTE OS, el microcontrolador S3C44B0X y el entorno de desarrollo cruzado.

3.1.- El sistema operativo MaRTE OS.

MaRTE OS es el sistema operativo objetivo de este proyecto. A continuación aparecen 4 apartados, que analizan sus características principales, su arquitectura general, el estado en el que se encuentra ahora y como extender su portabilidad para soportar nuevas plataformas.

3.1.1.- Características principales.

MaRTE OS (*Minimal Real Time Operating System for Embedded Applications*) es un sistema operativo de tiempo real que implementa la especificación dada en un pequeño subconjunto del estándar POSIX.13. Es bautizado como sistema operativo real mínimo debido a que no proporciona servicios como el de E/S o sistemas de ficheros, servicios proporcionados por otros SO de propósito general como Linux o Windows.

El sistema operativo incluye 2 API's POSIX para desarrollo de aplicaciones C y Ada, que se encargan de abstraer las llamadas al núcleo. Estas aplicaciones podrán ser portadas fácilmente a otros SO que también implementen el mismo subconjunto de POSIX.13 simplemente cambiando la implementación del kernel, tal y como se muestra en la figura 3.1.1.1. Esto en general puede ser extendido a cualquier conjunto de sistemas que implementen cualquier subconjunto de POSIX.

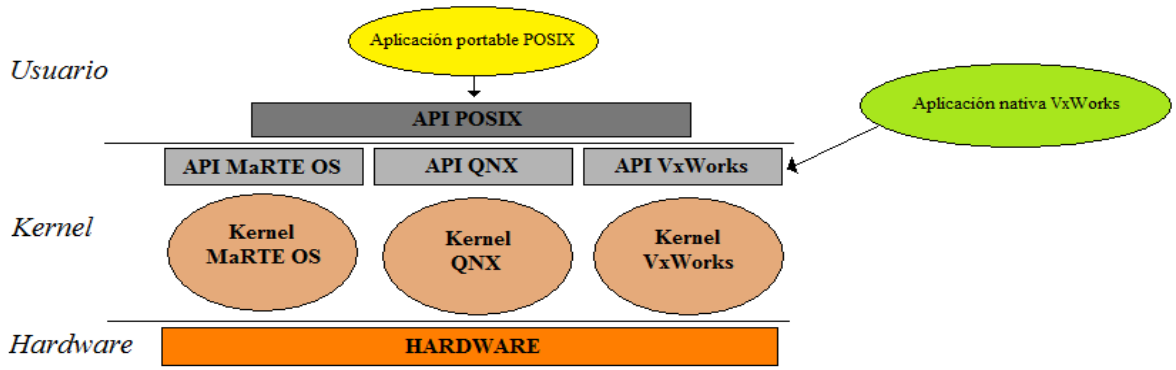

Figura 3.1.1.1. Portabilidad ofrecida por POSIX.

Tanto el núcleo principal como las diferentes capas de MaRTE OS están diseñados para ser enlazados (a modo de librerías) junto con las aplicaciones que hagan uso de este SO. A la hora de enlazar el código de este SO será posible acotar, mediante un fichero de configuración, diversos elementos a tener en cuenta en cualquier aplicación de tiempo real: la cantidad y tipo de recursos que utilizará el mismo en tiempo de ejecución (tales como memoria, número de *threads* y pila reservada a cada uno de ellos), los niveles de prioridad o el tamaño de la cola de señales pendientes. Salvo excepciones como las que se detallarán en el apartado 3.1.3, las diferentes implementaciones de MaRTE OS se traducirán directamente a código máquina ejecutable en máquinas desnudas y no a llamadas a otros SO. La tarea de compilación la realizará el entorno de desarrollo cruzado, el cual será analizado en el apartado 3.3.

El código fuente del sistema operativo MaRTE OS se liberó en el 2000 bajo licencia GNU/GPL2, permitiendo a los programadores la descarga y libre modificación del mismo sin posibilidad de apropiación. El código está casi totalmente escrito en Ada, aprovechando por supuesto toda la potencia de este lenguaje de programación. Existen algunas partes del sistema operativo escritas en C e incluso en código máquina, este último empleado para las secciones de código dependiente de las plataformas *hardware* para las que está implementado.

3.1.2.- Arquitectura.

Al igual que cualquier sistema operativo moderno, MaRTE OS está basado en un diseño de capas. De esta manera, el código fuente se estructura en distintas capas, cada una de las cuales proporciona servicios a la capa superior y emplea los servicios suministrados por la capa inferior. La figura 3.1.2.1 ilustra el modelo de capas de MaRTE OS para aplicaciones escritas en C y Ada.

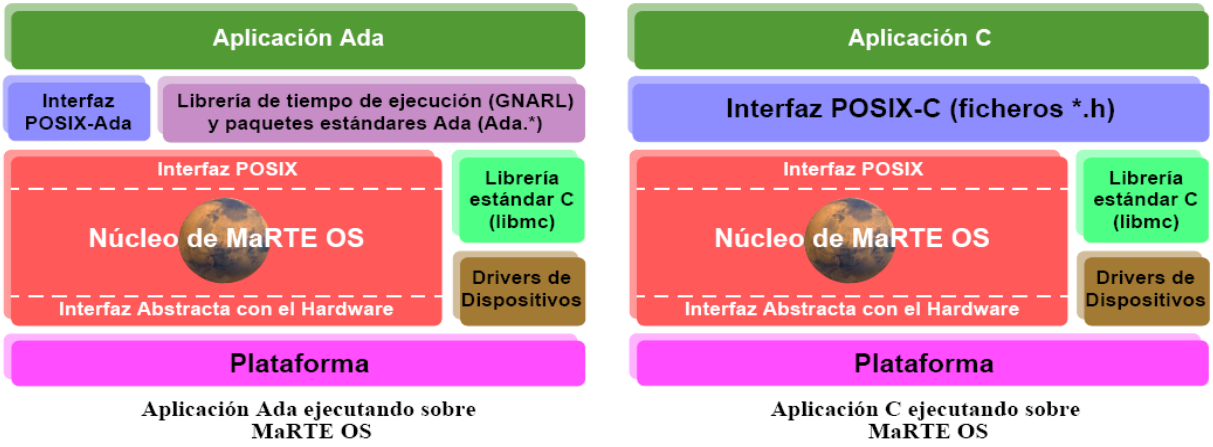

Figura 3.1.2.1. Modelo de MaRTE OS para aplicaciones escritas en C y Ada.

Los modelos para aplicaciones en C y Ada presentan numerosas semejanzas en cuanto al propósito de cada una de las capas. El centro de ambos modelos es el código fuente de MaRTE OS, el cual tiene como propósito comunicar y gestionar la relación entre las aplicaciones del programador y la plataforma *hardware* existente. Dentro del código fuente de MaRTE OS existen 3 capas bien diferenciadas:

- El núcleo o capa central. Contiene el código principal del *kernel* de MaRTE OS. Dentro de él existen bibliotecas que implementan el manejo de señales, temporizadores, *mutexes*, operaciones con tareas, variables de condición e interrupciones *hardware*.

- La interfaz POSIX o capa superior. Abstrae a las aplicaciones del programador del código interno de MaRTE OS, ofreciendo una interfaz universal que puede ser reimplementada por otros SO de tiempo real.
- La interfaz abstracta con el *hardware*. Proporciona al núcleo procedimientos y funciones que lo abstraen de la plataforma *hardware* empleada.

Paralelamente a estas 3 capas existen otras 2, cuyo propósito es servir a las plataformas *hardware* que dispongan de algún dispositivo de E/S (un monitor por ejemplo) de funciones para realizarla. Para este propósito el SO dispone de una serie de *drivers* de manejo de la E/S en los diferentes dispositivos para los que está implementado y una implementación de la biblioteca estándar de C que hace uso de dichos *drivers*.

La diferencia fundamental entre los modelos de las aplicaciones escritas en C y Ada es la capa inmediatamente superior a la interfaz POSIX de MaRTE OS y la biblioteca estándar de C suministrada. Esta capa es la que proporciona una interfaz (C o Ada) POSIX que hace uso directo de la interfaz POSIX suministrada por MaRTE OS, consiguiendo así soporte para ambos lenguajes. En el caso de C la interfaz la constituyen cabeceras .h (interfaz POSIX.1) y en el caso de Ada ficheros .ads (interfaz POSIX.5). Para el soporte de Ada es necesario incluir adicionalmente distintas implementaciones de la biblioteca de bajo nivel (GNULL) de la librería de tiempo de ejecución de GNAT, una por cada una de las plataformas *hardware* soportada, tal y como aparece en la tesis doctoral de Mario Aldea Rivas [8]. Esto se analiza con mayor detalle en el apartado 3.1.4.

3.1.3.- Estado actual del sistema operativo.

Como se ha detallado anteriormente, MaRTE OS es un sistema operativo diseñado para ser enlazado con aplicaciones escritas en C o Ada. Tal y como están programados todos los *scripts* de compilación/instalación adjuntos al SO, MaRTE OS debe ser compilado y enlazado con estas aplicaciones desde un sistema operativo Linux. Debe disponerse de algunos otros componentes *software/hardware* tal y como se indica en el apartado 3.3.

Actualmente, la última distribución (Agosto de 2009) de MaRTE OS implementa el soporte para las siguientes plataformas:

- Soporte para aplicaciones lanzadas como un proceso de Linux. En este caso, tanto las aplicaciones programadas en C/Ada como los fuentes del sistema operativo se compilan generando llamadas al sistema operativo Linux, el cual abstraerá a MaRTE OS de la plataforma *hardware* empleada. Como MaRTE OS está hecho para ser compilado y enlazado desde el propio Linux, esto implica que esta alternativa es la ideal para realizar las primeras pruebas con el SO. La ejecución de una aplicación compilada de esta manera se realizará invocando al ejecutable desde la línea de comandos.
- Soporte para aplicaciones lanzadas como un proceso de Linux con acceso a la biblioteca GLIBC del sistema operativo. Esto implica la posibilidad de acceder directamente al sistema de ficheros y los *drivers* de dispositivo, pero el resto de características son similares al caso anterior.
- Soporte para aplicaciones lanzadas directamente sobre una máquina desnuda X86, con sus 3 versiones para 386, Pentium I y Pentium II. En este caso tanto las aplicaciones como los fuentes del sistema operativo se compilan generando código máquina ejecutable directamente sobre la plataforma *hardware*. El proceso de lanzamiento de los ejecutables compilados será más complicado que en los otros casos, puesto que será necesario compilar previamente desde Linux la aplicación

C/Ada (y enlazarla con los fuentes del sistema operativo) y después lanzarla de alguna forma (con algún gestor de arranque como se especifica al final del anexo C o como se detalla la ayuda del proceso de arranque [9]) en la máquina X86 destino.

3.1.4.- Estudio de la portabilidad hardware.

Como se especificó en el apartado 3.1.2, existe una capa en la arquitectura del sistema operativo MaRTE OS que abstrae a todo el sistema de la plataforma *hardware* empleada. Esta capa se denomina interfaz abstracta con el *hardware*. A nivel de programación, esta capa lo constituye un solo fichero llamado "marte-hal.ads", que contiene la especificación de las funciones y procedimientos dependientes del *hardware* que deben ser implementadas para un correcto funcionamiento del *kernel* y demás elementos independientes de la plataforma.

La implementación del fichero "marte-hal.ads" viene dada, para cada una de las versiones actuales del sistema operativo, en el fichero "marte-hal.adb". La implementación de las funciones y procedimientos obviamente es distinta para cada una de las versiones, y adicionalmente se han implementado para cada una los *drivers* de los dispositivos utilizados desde la interfaz abstracta con el *hardware*.

El planteamiento inicial para conseguir la migración de MaRTE OS al microprocesador ARM es partir de la especificación de la interfaz abstracta con el *hardware* e ir implementando cada una de las funciones y procedimientos en un nuevo fichero "marte-hal.adb", respetando la estructura de directorios mantenida en los fuentes originales. En caso de tener que implementar *drivers* para algún periférico del microcontrolador utilizado se utilizarán ficheros separados, de la misma forma que se ha hecho para los distintos soportes implementados de MaRTE OS.

Mención aparte merecen los ficheros que contienen la implementación dependiente del *hardware* de los tipos *task* y *protected* de Ada. Estos ficheros, pertenecientes a la biblioteca de bajo nivel (GNULL) de la librería de tiempo de ejecución de GNAT, se almacenan en una carpeta aparte entre los fuentes del sistema operativo, y deben ser programados individualmente para cada uno de los soportes de MaRTE OS. Las rutinas contenidas en estos ficheros no han sido tenidas en cuenta en este proyecto debido al alcance del mismo, pero en un futuro deberán implementarse para completar la migración del sistema operativo. Para más información al respecto consultar la tesis de máster de Bartłomiej Horn [10], de la Universidad Técnica de Łódz, y la tesis doctoral de Mario Aldea Rivas [8].

3.2.- El microcontrolador S3C44B0X.

El S3C44B0X es el microcontrolador sobre el que se ha realizado la migración del sistema operativo MaRTE OS. El microcontrolador está dispuesto junto con otros elementos (ver apartado 1.3.1) de la placa de desarrollo S3CEV40.

3.2.1.- Características principales.

El S3C44B0X de Samsung es un microcontrolador basado en el núcleo ARM7tdmi, un procesador tipo RISC de la familia ARM (ver anexos D y E para más información). El microcontrolador dispone de los siguientes componentes:

- Un avanzado controlador de interrupciones, el cual soporta hasta 30 fuentes de interrupción distintas. Cabe la posibilidad de configurar las interrupciones de 3

maneras: IRQ, IRQ vectorizadas (la latencia es más reducida) e interrupciones FIQ (de procesado rápido, no compatible con IRQ vectorizadas).
- 6 temporizadores PWM de 16 bits, 5 de ellos con pin de salida.
- Un reloj de tiempo real, generador de la interrupción de tick y con posibilidad de configuración de alarmas en instantes absolutos.
- 79 pines de salida, de los cuales 71 pueden ser configurados como entrada.
- Una UART con 2 canales, con posibilidad de transmisión/recepción de 5 a 8 bits en serie. Dispone de una cola de 32 bits y tasa de envío/recepción configurable.
- Un completo controlador DMA.
- Un conversor analógico-digital de 8 canales y 10 bits de resolución.
- Pantalla LCD de 256 colores y 16 niveles de grises.
- Watchdog de 16 bits con reinicio del sistema y/o generación de interrupción.
- Buses I2C e I2S para comunicaciones serie de datos y sonido.

El microcontrolador ofrece, por tanto, soporte para implementar los servicios básicos de todo sistema operativo de tiempo real: gestión de interrupciones (mediante el controlador de interrupciones), *tick* de sistema (implementable con cualquier módulo que permita interrupciones periódicas, como el reloj de tiempo real o el temporizador PWM) y temporizaciones *hardware.*

3.3.- El entorno de desarrollo cruzado.

El entorno de desarrollo cruzado es el elemento que permite comunicar el sistema operativo MaRTE OS (y las aplicaciones desarrolladas con él) con el microcontrolador S3C44B0X y su procesador ARM.

La idea de por qué surge la necesidad de este entorno es muy clara: MaRTE OS y sus aplicaciones deben ser compilados desde un sistema operativo Linux y la aplicación resultante debe ejecutarse sobre una máquina desnuda ARM. Así pues, el entorno desde el que se va a compilar (entorno Host) es distinto al entorno sobre el que se va a ejecutar (entorno Target) el programa, por lo que se necesita un compilador especial que genere desde Linux ejecutables para un procesador ARM: un compilador cruzado.

Antes de comenzar a migrar MaRTE OS será necesario configurar este entorno de desarrollo que nos permitirá poder compilar nuestras aplicaciones. Para ello deberá partirse de los fuentes de GCC y GNAT (y algunos elementos que se detallan en el apartado de diseño y el anexo J) para generar un compilador con soporte para traducir C y Ada a código máquina ejecutable directamente sobre un procesador ARM. El esquema de los elementos que compondrán el entorno cruzado es el que aparece en la figura 3.3.1.

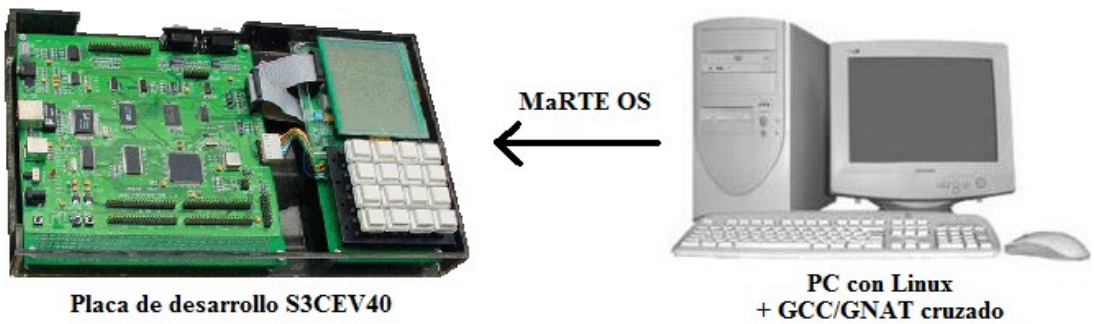

Figura 3.3.1. Elementos que intervienen en la compilación cruzada.

Capítulo 4

Fase de diseño.

En este capítulo se detallan las técnicas empleadas para satisfacer los objetivos planteados teniendo en cuenta las conclusiones extraídas de la fase de análisis. Pese a que los objetivos fueron definidos al inicio de este proyecto, las conclusiones de la fase de análisis permitieron refinarlos hasta dar con un hito alcanzable. Es en este momento cuando se redacta y entrega la propuesta del proyecto.

4.1.- Migración del sistema operativo MaRTE OS.

Como se especificó en la fase de análisis, la meta final para la migración de MaRTE OS será la implementación de las funciones/procedimientos del fichero "marte-hal.ads", que constituye la interfaz del sistema operativo con la plataforma *hardware* empleada. Se ha realizado una clasificación de las diferentes subrutinas de este fichero en categorías con el fin de estudiar como realizar la implementación de cada una de las mismas sobre el microcontrolador utilizado:

- Operaciones con interrupciones: esta categoría agrupa las subrutinas básicas que todo sistema debe ofrecer sobre las interrupciones *hardware*: habilitación/deshabilitación de una o todas las interrupciones, instalación de las mismas e inicialización del periférico adecuado. El controlador de interrupciones del S3C44B0X se ajusta perfectamente a estas necesidades. Será necesario definir también las constantes asociadas a cada una de las fuentes de interrupción disponibles.
- Operaciones con registros del procesador: se agrupan aquí todas las operaciones de bajo nivel sobre registros fundamentales del procesador, concretamente el registro de estado y el *stack pointer*. Por la simplicidad de estas operaciones, su implementación será realizada usando código máquina del ARM.
- Operaciones a nivel de bit: operaciones sobre campos de 32 bits. En caso de que las instrucciones de código máquina del ARM no permitan implementar estas operaciones con relativa eficiencia se utilizarán las sentencias del lenguaje Ada (o C).
- Operaciones con tiempo: el grupo más numeroso. Esto agrupa operaciones de trabajo con el tiempo del sistema (en *ticks*), operaciones de conversión de tiempos, programación de temporizadores y trabajo con instantes absolutos de tiempo. El reloj de tiempo real del microcontrolador permitirá implementar tanto las operaciones relacionadas con el tiempo del sistema como el trabajo con instantes absolutos, mientras que el temporizador PWM permitirá implementar *timers*. Las conversiones de tiempo son independientes del *hardware.*
- Operaciones de cambio de contexto: no implementadas, puesto que el compilador cruzado no está completo y se necesita soporte para tareas y objetos protegidos.

La implementación exacta de cada subrutina puede consultarse en el anexo I.

4.2.- Periféricos del S3C44B0X utilizados.

A continuación se realiza un pequeño análisis acerca de como se han utilizado los 3 periféricos del S3C44B0X elegidos para la migración de MaRTE OS: el controlador de interrupciones, el reloj de tiempo real y el temporizador PWM.

4.2.1.- El controlador de interrupciones.

El controlador de interrupciones del S3C44B0X es un periférico que soporta hasta 30 fuentes de interrupción diferentes y es el encargado de gestionar el arbitraje entre el código de programa y las distintas interrupciones que puedan ir apareciendo en su ejecución. Las 30 fuentes de interrupción son gestionadas a través de 26 líneas diferentes del modo indicado en la figura 4.2.1.1.

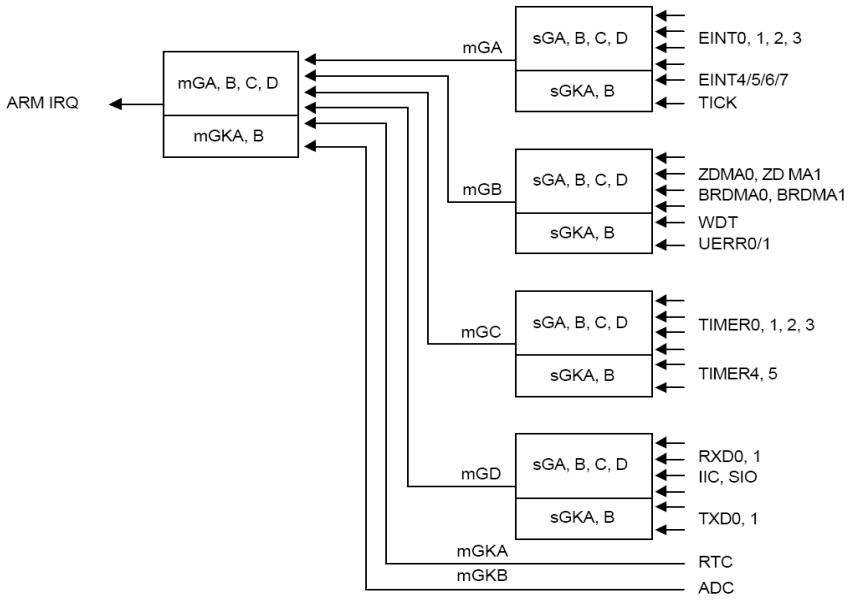

Figura 4.2.1.1. Fuentes de interrupción del controlador de interrupciones.

Como puede verse en la figura 4.2.1.1, algunas fuentes de interrupción (EINT4/5/6/7 y UERR0/1) comparten la misma línea. La estructura del controlador es muy sencilla. Un bloque maestro determina cual de sus 6 señales de entrada es la que debe generar la interrupción (teniendo en cuenta los niveles de prioridad establecidos). Existen 4 bloques esclavos que generan 4 de las 6 entradas del bloque maestro a partir, de nuevo, de 6 fuentes de interrupción, mientras que las otras 2 entradas provienen directamente de la línea de interrupción del reloj de tiempo real y el conversor analógico-digital. La forma de establecer prioridades es también muy sencilla: 2 de las 6 entradas de cada bloque tienen prioridad fija dentro del mismo, y las otras 4 entradas tienen prioridad configurable. Por defecto las prioridades comienzan establecidas siguiendo orden descendente en la figura.

Lo novedoso de este controlador respecto a otros es el tipo de posibilidades (ver figura 4.2.1.1) que ofrece a la hora de configurar el modo en el que van a ser tratadas las interrupciones. El controlador dispone de 2 líneas de activación de interrupciones: la línea IRQ (Interrupt ReQuest) y la línea FIQ (Fast Interrupt reQuest). Estas 2 líneas permiten establecer cierto tipo de prioridades entre las interrupciones, puesto que las interrupciones configuradas

para que envíen peticiones por la línea FIQ tendrán mayor prioridad que las peticiones por la línea IRQ. Además, el tratamiento de interrupciones puede ser configurado de 2 maneras:

- Tratamiento de interrupciones no vectorizado. Esto implica que cada vez que se solicita una interrupción el flujo de programa salta a una dirección concreta. En esa dirección se comprueba cual es la interrupción que se ha producido (consultando algún registro específico del controlador) y se genera un nuevo salto a la rutina de interrupción asociada.
- Por otro lado, puede configurarse el tratamiento de interrupciones como vectorizado. En caso de elegir esta alternativa, cada fuente de interrupción hará que el flujo de programa salte a una dirección específica diferente, predefinida (estática) dentro de lo que se denomina vector de interrupciones. Desde esta dirección se realizará un salto a la rutina de interrupción (este salto es el que deberá configurarse a la hora de instalar manejadores). Como la dirección en la que debe colocarse el salto a la rutina es fija para cada fuente de interrupción se consigue ahorrar el tiempo de calculo de dicho salto, sacrificando para ello un espacio de memoria de 32 bits por cada fuente de interrupción. Las interrupciones configuradas como FIQ tendrán siempre tratamiento no vectorizado, al margen de esta opción de configuración.

Debido a que el modo vectorizado es mucho más eficiente, fue el elegido en la fase de diseño. Como veremos en la fase de implementación (capítulo 5), surgieron algunos problemas derivados de esta decisión de diseño. En la figura 4.2.1.2 se muestra un esquema de los tipos de tratamiento de interrupciones no vectorizado y vectorizado, respectivamente.

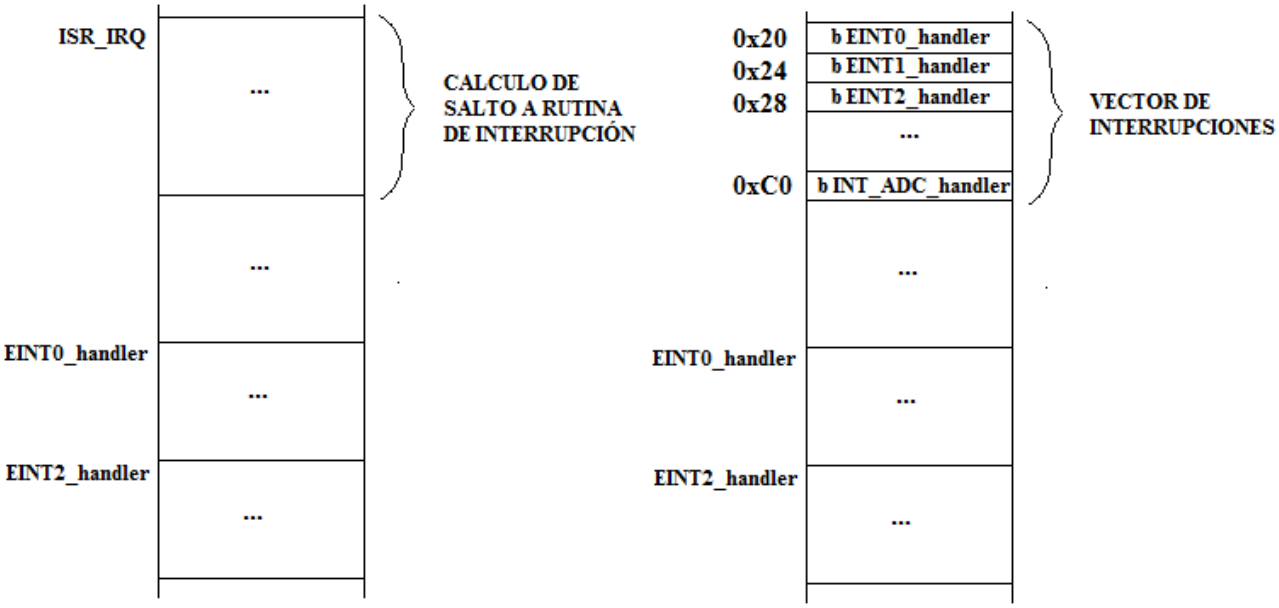

Figura 4.2.1.2. Tratamiento de interrupciones no vectorizado y vectorizado.

En el anexo F se resumen las características y registros principales de este periférico.

4.2.2.- El reloj de tiempo real.

El reloj de tiempo real del S3C44B0X ha sido el periférico empleado para implementar todo lo relacionado con el *tick* del SO y el trabajo con instantes absolutos de tiempo. La figura 4.2.2.1 muestra el diagrama de bloques de este periférico.

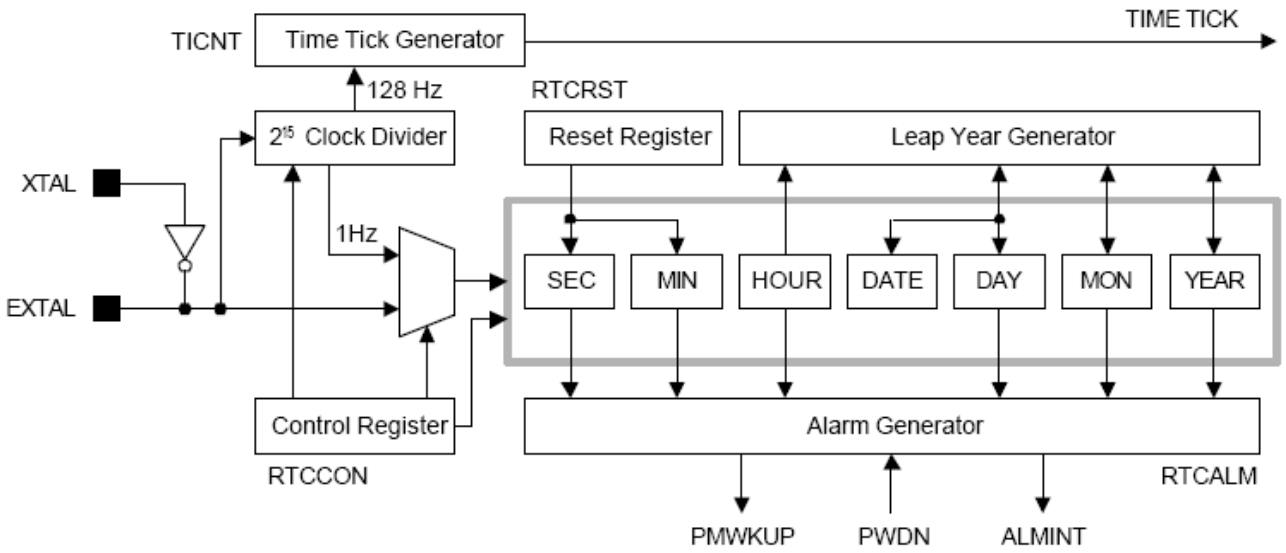

Figura 4.2.2.1. Diagrama de bloques del reloj de tiempo real.

La base del reloj de tiempo real son los 6 bloques que representan registros que almacenan la fecha actual: SEC, MIN, HOUR, DAY, MON y YEAR. Adicionalmente el bloque DATE representa el registro que almacena el día de la semana (de 1 a 7) actual. Aunque no se han utilizado, existen otros 6 registros (análogos a los 6 que representan la fecha actual) que permiten configurar (junto con el registro de control de alarmas) alarmas en instantes de tiempo absolutos. Cuando el valor de los registros de fecha actual coincida con los registros de alarma se enviará petición de interrupción RTCALM. Adicionalmente el periférico dispone de medios para detectar si el año actual es bisiesto y para resetear los registros de fecha actual que almacenan los minutos y los segundos.

Lo que si que se ha utilizado de este periférico es la capacidad de generar la interrupción TICK. Este periférico debe ser inicializado en el procedimiento de inicialización de la interfaz abstracta con el *hardware* de MaRTE OS de forma que cada vez que se produzca la interrupción TICK el contador interno que hay declarado se incremente. Esto deberá realizarse instalando la interrupción correspondiente en la inicialización.

Podría haberse utilizado el temporizador PWM (descrito en el apartado 4.2.3) para implementar el *tick* de sistema. Sin embargo el reloj de tiempo real es mucho más simple, sencillo de configurar, y está diseñado para generar únicamente interrupciones periódicas. Esta es la razón por la que se ha elegido este periférico como alternativa frente al temporizador PWM, este último con capacidad para otras funcionalidades adicionales como la generación de interrupciones *"one-shot"* (no periódicas, de un solo pulso). Podrá usarse en un futuro para la gestión de señales de control o la ampliación del número de temporizadores de MaRTE OS.

Pueden consultarse los registros específicos del reloj de tiempo real y detalles de más bajo nivel en el anexo G.

4.2.3.- El temporizador PWM.

El temporizador PWM ha sido el periférico empleado para implementar el soporte de programación de temporizadores. La figura 4.2.3.1 muestra el diagrama de bloques de este periférico.

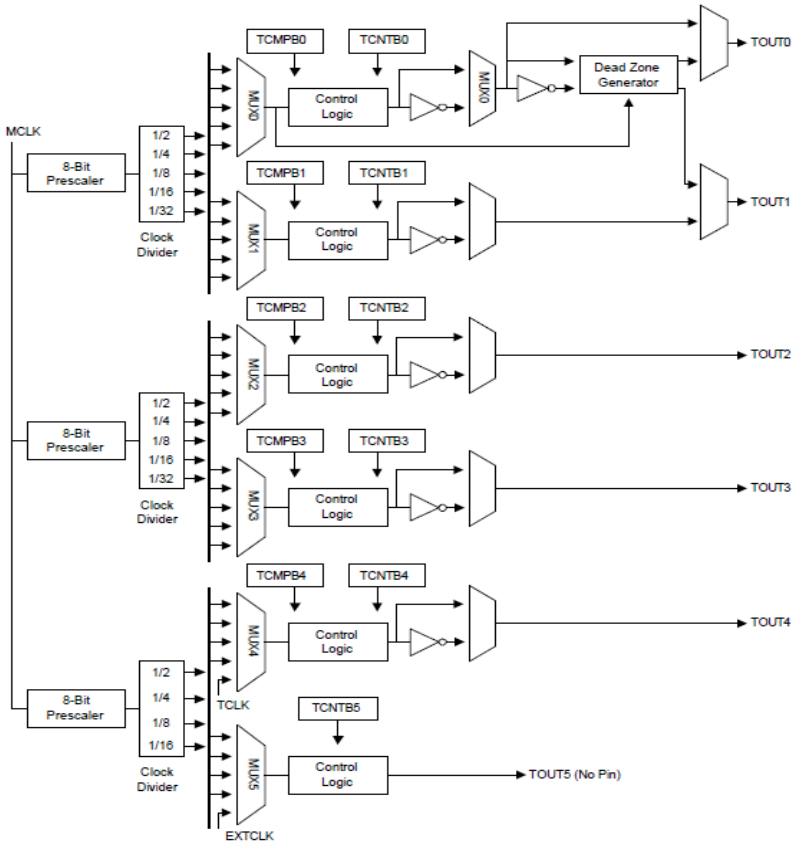

Figura 4.2.3.1. Diagrama de bloques del temporizador PWM.

Como se ve en la figura, el temporizador PWM dispone de 6 temporizadores individuales, agrupados de 2 en 2. Cada uno de los grupos toma como entrada la señal de reloj de la CPU a través de un *prescaler* y un divisor de frecuencia. El funcionamiento general de los temporizadores es muy sencillo: cada flanco de subida del reloj decrementa el valor del registro contador TCNTBi (para el temporizador i) y en caso de que el valor de ese registro sea igual al del registro de comparación TCMPBi se produce un pulso en la señal TOUTi. El temporizador 5 es ligeramente diferente al resto, puesto que no dispone de registro de comparación. En este caso, el pulso se producirá cuando el valor del registro contador sea 0. Debido a que este temporizador es el único que no dispone de pin de salida (TOUT5 es una señal interna) ha sido el empleado a la hora de implementar el soporte para la temporización.

El *watchdog* fue el periférico utilizado inicialmente para implementar la temporización. Sin embargo no ofrecía soporte para pulsos/interrupciones de tipo "one-shot" (no periodicos/as), por lo que fue desechado.

El temporizador PWM dispone de otras características, como generador de tiempos de incertidumbre al dispararse los temporizadores 0 y 1. Todas estas características se detallan más a fondo en el anexo H.

4.3.- Configuración del entorno de desarrollo.

Como se especificó en la fase de análisis, el proceso de compilación y ejecución de aplicaciones no es trivial: las aplicaciones (y MaRTE OS) deben ser compiladas desde Linux, para ser ejecutadas sobre una máquina desnuda basada en ARM. Se necesita, pues, configurar y definir las herramientas que se emplearán para llevar a cabo esta tarea.

Parte del entorno de desarrollo cruzado es proporcionado por el fabricante de la placa de desarrollo S3CEV40, utilizada en el desarrollo de este proyecto. La placa de desarrollo trae consigo unos CD's con documentación, ejemplos de programas, y un completo entorno de desarrollo/depuración de programas llamado EmbestIDE. Tras obtener la licencia necesaria para utilizar el entorno en el computador de trabajo, se pudieron realizar las primeras pruebas para verificar el correcto funcionamiento del *hardware* disponible. En el manual *on-line* del entorno [11] se muestra en detalle como compilar, cargar y depurar aplicaciones con EmbestIDE, junto con todas sus posibilidades.

Sin embargo, pronto se detectó que el entorno de desarrollo proporcionado por el fabricante resultaba insuficiente para realizar la migración de MaRTE OS, por 2 motivos:

- El entorno estaba basado en una versión del compilador GCC que únicamente soportaba lenguaje C. Para migrar MaRTE OS era necesario incluir adicionalmente soporte para lenguaje Ada.
- Uno de los requisitos era dispone de sistema operativo Windows. Debido a que MaRTE OS requiere Linux para ser compilado y enlazado con las aplicaciones, el proceso de compilación debía realizarse de otra forma que no fuera con este entorno de desarrollo.

Por estas 2 razones, se pensó que lo más sencillo era delegar el proceso de compilación de programas a un compilador construido de manera completamente independiente al entorno EmbestIDE. El compilador debía estar construido sobre Linux y su único requisito era tener soporte para los lenguajes C y Ada, por lo que debía estar basado en GCC y GNAT. El proceso de configuración (ver anexo J) del compilador cruzado fue el que más tiempo de dedicación al proyecto consumió, debido a la escasez y mala calidad de la documentación disponible en Internet. Finalmente, una vez configurado el compilador, el proceso de compilación, carga y depuración de aplicaciones quedó perfectamente definido:

- Inicialmente, debe tenerse una aplicación C o Ada programada y lista para ser compilada, en Linux.
- Desde Linux, es preciso ejecutar las utilidades proporcionadas por el compilador cruzado. Se proporcionan unos *scripts* que evitan tener que memorizar los comandos de compilación, cuyo funcionamiento se especifica en el anexo A.
- Una vez generado el ejecutable, será preciso transferirlo de algún modo a la máquina que dispone de sistema operativo Windows y tiene el entorno EmbestIDE instalado. Puede realizarse vía USB. Como en el caso de este proyecto se ha trabajado con una máquina virtual Ubuntu (Linux), basta con tener una carpeta compartida [12] entre dicha máquina virtual y el sistema operativo anfitrión Windows.
- Una vez en Windows, transferir el ejecutable a la memoria Flash/RAM según proceda, siguiendo lo detallado en el manual *on-line* de EmbestIde [11].

Este proceso es muy tedioso, por lo que en un futuro sería interesante poder prescindir del entorno EmbestIDE y cargar las aplicaciones desde Linux.

Capítulo 5

Fase de implementación.

En este capítulo se trata de manera resumida la fase de implementación.

En este capítulo no entraremos mucho en detalle acerca de como se han utilizado, a nivel de programación, los diferentes periféricos del S3C44B0X para la migración de MaRTE OS. El anexo I es el destinado a ese propósito. Únicamente trataremos las pautas principales que se han seguido a la hora de implementar lo definido en la fase de diseño. Lo relativo a las pruebas proporcionadas se trata en detalle en el anexo A.

Dado que el compilador cruzado construido a partir de los fuentes de GCC y GNAT ofrece soporte para los lenguajes C y Ada (así lo queríamos, dado que MaRTE OS está implementado usando esos 2 lenguajes) esos 2 lenguajes han sido los utilizados para la migración. Como norma general, se ha procurado no sobrecargar el fichero principal que contiene la implementación de la interfaz abstracta con el *hardware* ("marte-hal.adb"), derivando las operaciones de más bajo nivel a módulos independientes para cada uno de los *drivers* o categorías de subrutinas identificadas (de acuerdo al apartado 4.1). Cada uno de estos módulos dispondrá de su especificación Ada (fichero .ads) su implementación (fichero .adb) y un fichero en C con las operaciones del módulo de más bajo nivel. Todo esto junto con el correcto sangrado empleado y las cabeceras definidas para cada fichero hace que la legibilidad del código sea óptima. Se incluye un fichero adicional con código en ensamblador acerca de definiciones para el vector de interrupciones, inicializaciones y la rutina de reseteo, así como ficheros de ayuda al enlazado de programas. El total de ficheros, su estructura y su propósito exacto se detalla en el anexo A y los detalles de implementación en el anexo I.

La fase de implementación en este proyecto también podría haberse llamado "fase de problemas". Algunas de las decisiones de diseño tomadas inicialmente han hecho que en este punto haya que replantearse ciertas cuestiones:

- Inicialmente, en la fase de diseño se había optado por utilizar el *watchdog* del S3C44B0X para implementar lo referente a temporizadores, por su simplicidad de manejo frente al temporizador PWM. Sin embargo, se comprobó a la hora de programar que el *watchdog* era insuficiente para lo que el procedimiento de programación del temporizador de MaRTE OS debía realizar: la especificación del temporizador es que debe ser de tipo *one-shot*, y el *watchdog* solo ofrecía soporte para temporizadores periódicos. El módulo temporizador PWM ofrecía la opción de temporización *one-shot*, de modo que una vez expirado el temporizador debía ser relanzado.
- La elección de gestión de interrupciones de manera vectorizada hizo que la parte de los ficheros ejecutables generados relativa a vectores de interrupciones tuviera que ser estática (comenzando en la dirección 0x20 como se especificó en el apartado 4.2.1). Aunque se pretendió que todo el ejecutable se cargara en la memoria RAM (de acceso rápido pero volátil) del microcontrolador, la parte que conllevaba los vectores de interrupción e inicializaciones tuvo que configurarse para ser cargada en la memoria Flash (de acceso lento pero persistente) del microcontrolador.

Capítulo 6

Conclusiones y perspectiva.

En este capítulo se relatan las conclusiones extraídas como fruto de este proyecto así como una serie de perspectivas de futuro sobre el mismo.

6.1.- Resumen del trabajo realizado.

Se han completado todos los objetivos redactados en la propuesta de proyecto entregada. Como resultado de este Proyecto Fin de Carrera se han alcanzado los siguientes hitos:

- Se ha configurado un entorno de desarrollo cruzado en Ubuntu (Linux), basado en GCC/GNAT y por lo tanto con soporte para los lenguajes C y Ada. El entorno soporta prácticamente todas las sentencias de ambos lenguajes, a excepción de las referentes a tareas y objetivos protegidos de Ada.
- A partir del entorno de desarrollo de Linux y el entorno EmbestIDE (para Windows) proporcionado por el fabricante del S3C44B0X, se ha definido todo el proceso desde que se tiene el fuente C o Ada hasta que se carga en la placa de desarrollo S3CEV40. Este proceso implicará la compilación y generación del ejecutable desde Linux y su carga en la placa desde Windows. Hay que recordar que se generan 2 ficheros para cargar: uno con la definición estática de los vectores de interrupción y las rutinas de inicialización para cargar en Flash y otro con el programa para cargar en RAM.
- Se han puesto a punto las rutinas de arranque, reseteo y demás elementos de bajo nivel de la placa de desarrollo. Estas rutinas son almacenadas en un fichero ensamblador aparte del resto del código. Adicionalmente se proporcionan algunos *scripts* de ayuda al enlazado de programas.
- Siguiendo la estructura de directorios original de MaRTE OS, se ha generado una nueva carpeta que contiene la parte del SO dependiente del *hardware*, con un código implementado en C y en Ada, legible y ordenado. Respecto a la parte referente a cambios de contexto, se han investigado alternativas para implementarla en un futuro.
- Se proporciona una batería de pruebas consistente con los *drivers* implementados, verificando y validando que las llamadas sobre la interfaz abstracta con el *hardware* de MaRTE OS realizan lo detallado en las especificaciones.
- Adicionalmente, se han implementado una serie de Scripts de compilación automática de las pruebas y la interfaz abstracta con el *hardware*, cuyo único propósito es facilitar su *testeo*.

6.2.- Trabajo futuro.

Como se ha detallado a lo largo de esta memoria, la migración de MaRTE OS no está completa. La primera tarea que debería plantearse cuando este proyecto sea continuado debería ser finalizar el portado de este sistema operativo, con el fin de tener una versión

mínima sobre la que ya poder proponer prácticas para la asignatura "Sistemas de tiempo real", dado que esta fue la razón principal por la que surgió este Proyecto Fin de Carrera. Los 2 hitos principales que constituirán esta tarea serán la adaptación de la biblioteca de bajo nivel (GNULL) del compilador cruzado generado y la programación de las rutinas de la interfaz abstracta con el *hardware* referentes a cambios de contexto entre tareas.

Al margen de lo anterior, existen otros elementos que podrían ser interesantes desde el punto de vista docente. MaRTE OS dispone de un sistema de alto nivel con el que poder registrar *drivers* de dispositivo y relacionarlos con el periférico a nivel *hardware*. Sería interesante investigar al respecto y ver las posibilidades que ofrece.

En este proyecto nos hemos centrado más en la configuración del entorno de desarrollo (la tarea más costosa) y en la implementación de los *drivers* de periféricos necesarios. Sería interesante también integrar la compilación de todos los ficheros con los complejos *scripts* que actualmente dispone MaRTE OS. Debería investigarse más a fondo la interfaz que ofrece para la implementación de programas en C o Ada y ver como afecta a la interfaz abstracta con el *hardware*. Este es un punto muy importante también a nivel docente, al igual que la simplificación del método de compilación y carga de aplicaciones en la placa de desarrollo.

6.3.- Conclusiones.

Inicialmente, el planteamiento que se tuvo de este proyecto fue el de migrar por completo MaRTE OS al procesador ARM. La idea inicial que tuve en mente fue que el proyecto consistiría en implementar una serie de ficheros de código, teniendo el manual del procesador disponible en todo momento para consultar los registros específicos del procesador y el microcontrolador. En resumen, pensé que sería similar a las prácticas que recientemente había realizado en la asignatura "Sistemas empotrados", pero con una mayor complejidad y extensión.

Pronto me di cuenta de que en este proyecto, al igual que en muchos otros, las cosas no son lo que parecen. Apareció la necesidad de disponer de un método con el que conseguir los ejecutables a partir del código fuente y cargarlos en la placa, tarea que fue la que más tiempo consumió de todo el proyecto. Estimo que fueron unas ¾ partes del tiempo total. Esta tarea hizo necesario recortar objetivos, desembocando en una derivación parcial del SO.

El hecho de que la configuración de todo el entorno de desarrollo cruzado costara tanto tiempo derivaba de que todo el proyecto estaba basado en código libre. Inicialmente no conocía muy bien lo que era un entorno de desarrollo cruzado y vagamente sabía en que consistía, pero la cantidad de ambigüedades en los tutoriales existentes en Internet me hizo perder mucho tiempo. Cada tutorial detallaba la configuración del entorno cruzado de una manera diferente, dados por supuesto algunos de los pasos (que evidentemente los desconocía) y dejando de manifiesto algunas ambigüedades. Todo ello añadido a que no existía ningún tutorial específico para la obtención de un compilador cruzado para ARM hizo que el proyecto se demorara más de lo establecido. Finalmente, encontré la tesis de máster especificada en la bibliografía, la cual me sacó del apuro.

Mi conclusión respecto a esto es que, aunque la tarea principal sea compleja (como en este caso, una migración de un SO), la tarea que más tiempo cuesta es conseguir hacer funcionar un "Hola mundo", el programa más sencillo que probablemente solo se dedique a encender un Led. En comparación con esto, la migración de MaRTE OS fue bastante sencilla, puesto que únicamente fue necesario consultar el manual y hacer algunas pruebas para verificar el código fuente.