"Desarrollo de un driver para la tarjeta de adquisición de datos PCM-3718H en el S.O.T.R. QNX 6.2 Neutrino"

Proyecto Fin de Carrera por

Raúl Jiménez Ruiz,

para la obtención del Título de

Ingeniero en Automática y Electrónica Industrial

por la Escuela Técnica Superior de Ingenieros de Sevilla

INDICE

keywords: qnx, driver, pcm-3718h, advantech, adquisición de datos, resource manager, c, librería compartida, appbuilder, ide.

gracias a Pepa, por su paciencia y amor

gracias a José y Maria Luisa, por sus enseñanzas y cariño

1 Introducción

1.1 Objetivo

Precedentes

Las tarjetas de adquisición de datos son tarjetas electrónicas de expansión que proporcionan a un computador la capacidad de interactuar con el ambiente que le rodea, de manera que pueden leer los valores proporcionados por sensores o comandar elementos que actúen en el entorno.

Para que un software cualquiera pueda comunicarse con una tarjeta de adquisición de datos, es necesario disponer del "driver" correspondiente para el Sistema Operativo que esté instalado en la computadora que hace de anfitrión para la tarjeta. Este "driver" es un software que se integra con el Sistema Operativo anfitrión para proveer un interfaz estándar de comunicación con el hardware de la tarjeta.

La tarjeta de adquisición PCM-3718H de Advantech dispone de un driver para varias versiones del Sistema Operativo Windows™ de Microsoft. También pueden encontrarse drivers para sistemas basados en GNU/Linux. Pero no se disponía de un driver para el Sistema Operativo de Tiempo Real QNX.

Objetivo

Este Proyecto Fin de Carrera tiene como propósito fundamental el desarrollo del driver de la tarjeta de adquisición de datos Advantech PCM-3718H para el Sistema Operativo de Tiempo Real QNX 6.2 Neutrino.

El driver de la tarjeta de adquisición de datos consiste en un paquete de software formado por tres componentes: *Resource Manager*, Librería compartida y Aplicación GUI[1] de ejemplo, que se describirán con detalle en los siguientes apartados. Los dos primeros implementan el *driver* de la tarjeta de adquisición de datos y el tercero es una aplicación gráfica para mostrar un ejemplo del uso de la tarjeta y de los componentes anteriores. Un esquema básico se puede observar en la figura 1.

[1] *Graphics User Interface* (Interfaz Gráfica de Usuario)

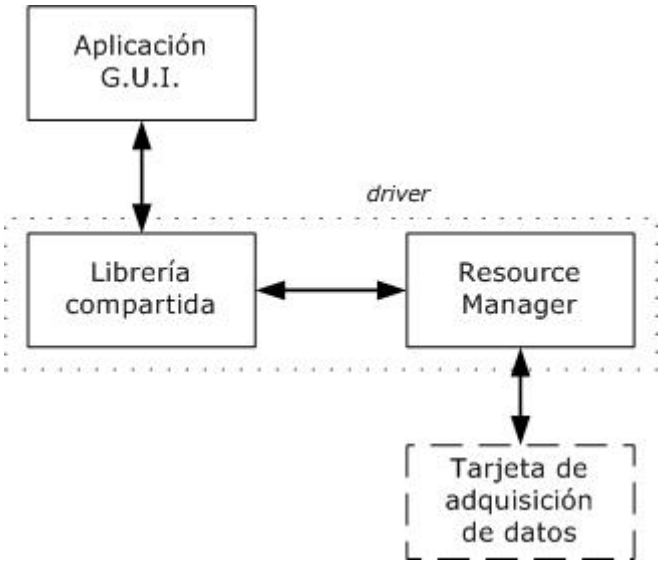


Fig. 1

La tarjeta mencionada en el párrafo anterior es la tarjeta PC/104 de adquisición de datos PCM-3718H, de Advantech, que ofrece conversión A/D de 12 bits de ganancia programable, dos canales de 8 bits de entradas/salidas digitales y dos contadores programables.

La idea principal de este Proyecto es desarrollar el *driver* de manera que se simplifique la realización de soluciones hardware/software basadas en la tarjeta antes mencionada. En caso contrario, se debería escribir cada vez el código fuente de acceso a los registros internos de la tarjeta y sería necesario conocer en detalle su funcionamiento interno. Con el uso del driver, sólo es necesario conocer ciertos datos básicos de la tarjeta así como sus modos de operación.

1.2 S.O.T.R. QNX 6.2 Neutrino

El Sistema Operativo en Tiempo Real que se empleará en este proyecto, tanto en el desarrollo del código como en las pruebas operativas del mismo, es el QNX 6.2 Neutrino. A continuación se recuerda parte de su historia y se describen sus características más importantes.

Algo de Historia

QNX Software Systems Ltd. fue fundada en 1980 por Gordon Bell y Dan Dodge para desarrollar, mantener y poner en el mercado el sistema operativo de tiempo real QNX que corre bajo procesadores INTEL: 386, 486, Pentiums y sus clones como AMD, Nat Semiconductor, Cyrix,...

QNX fue creado por dos fundadores y co-presidentes Gordon Bell y Dan Dodge. Existe la leyenda de que las primeras ideas surgieron en la época en que se encontraban en la universidad de Waterloo. A principio de los 80 comenzaron a trabajar en un pequeño SO basado en un kernel con paso de mensajes cuya interfase de usuario se parecía a la de UNIX. Los prototipos eran dos procesadores un 6809 construido por Dan y un 8088 construido por Gordon. Al principio de los 80 IBM lanzó su IBM PC basada en el 8088 y ahí comenzó la carrera. Cerca de 6 meses después de que IBM hiciera su lanzamiento Dan y Gordon hicieron conocido su producto con un aviso en una revista sobre ordenadores PC. Ellos denominaron a su producto QUNIX, dado que era un "Quick UNIX". Llamaron a su compañía Quantum Software Systems Limited. El nombre QUNIX duró un año o dos hasta que recibieron una notificación de AT&T de que debían cambiarlo. Siempre listos para un desafío, cambiaron la forma de escribirlo a QNX pero no cambiaron su pronunciación (se supone que QNX se

debe pronunciar "quiunix").

Las primeras betas que se lanzaron no tenían multitasking pero corrían en una IBM PC con solo 64k de RAM y un floppy de 180k. Después de unos pocos meses se lanzó la versión 1.0 que si soportaba multitasking. Aunque parezca increíble los 64 k de memoria eran suficientes para correr el SO, una shell y aún compilar programas. Es más, es posible que incluso se pudieran hacer trabajos en background, como imprimir un archivo. Un relativamente pequeño grupo de usuarios ambiciosos trabajó con los desarrolladores para reportar bugs y agregar mejoras. Este grupo de personas aún continúa usando QNX o trabajando en la compañía y dando convenciones sobre el producto.

El siguiente gran paso para QNX fueron las maquinas IBM AT. Las primeras modificaciones permitieron el uso de un procesador mucho más rápido, el 80286 que corría a 6MHz. Posteriormente, sólo 6 semanas más tarde, ya soportaba el modo protegido. Debido a una impresionante visión de futuro algunos programas soportaban el modo protegido sin necesidad de ser modificados y de hecho ni siquiera se necesitó recompilarlos, en contra de lo que ocurría con otros SOs. Siguiendo algunas pequeñas reglas, incluso los drivers podían correr en cualquiera de ambos modos. De esta manera las aplicaciones se hicieron mucho más estables, impidiendo que alguna tarea haga fallar a todo el sistema.

Distintos tipos de RTOS

En el mercado actual, los sistemas operativos de tiempo real caen bajo tres categorías principales - ejecutivo de tiempo real, monolítico y modelo de proceso universal. A continuación se explican los tres tipos de configuraciones:

- Ejecutivo de tiempo real:

Referido a veces por como "kernel" estos sistemas operativos siguen un modelo de sistema operativo de espacio de direccionamiento simple. Todos los módulos de software se alojan en el mismo espacio de direccionamiento que el núcleo del SO, sin protección de la memoria entre las tareas, ya sean tareas de la aplicación o tareas del sistema. Consecuentemente, un solo error de puntero en una aplicación puede corromper las estructuras fundamentales del código o de los datos, dando como resultado un fallo general del sistema. Cuando ocurren este tipo de fallos de funcionamiento, puede tomar semanas o meses localizar la fuente, particularmente en sistemas complejos con centenares de módulos. Puede también llevar mucho tiempo agregar o modificar características, puesto que cada cambio del código significa que el núcleo (kernel) tiene que ser reconstruido y reexaminado.

- Arquitectura monolítica:

Para corregir algunos de los problemas de la configuración "ejecutivo en tiempo real", algunos vendedores de RTOS han adoptado una arquitectura monolítica. En este modelo, el sistema operativo y todos los servicios fundamentales, tales como file system, residen en un monitor monolítico que se accede a través de un mecanismo de llamada al núcleo. Las llamadas al núcleo hacen la transición del modo aplicación (o usuario) al modo supervisor. Se proporciona protección de modo que solo se puede acceder a recursos del núcleo en modo supervisor; las aplicaciones se ponen en ejecución generalmente como procesos que tienen espacios de direccionamiento separados con protección entre ellos.

Con un monitor monolítico, los servicios fundamentales que requieran acceso a los recursos del núcleo deben residir en este. De esta forma, la complejidad del núcleo aumenta, aumentando la probabilidad de encontrar errores. Asimismo, el acceso a entrada-salida, al vector de interrupciones y a la memoria física se puede restringir al núcleo por razones de seguridad, lo que significa que la mayoría de los drivers de dispositivos deben residir en el núcleo. Cruzar la barrera kernel/aplicación es con frecuencia costoso. Así pues, en algunos casos, los drivers que, de otro modo, se podrían poner en ejecución modo usuario, como drivers de tarjetas gráficas, se incorpora en el núcleo por razones de rendimiento. Mientras que las aplicaciones no pueden corromper el núcleo, cualquiera

de estos drivers de dispositivos sí pueden, aumentando la probabilidad de que ocurran errores fatales.

- Modelo de proceso universal:

La configuración "modelo de proceso universal" describe un conjunto de características que se han asociado tradicionalmente a los sistemas operativos de microkernel. Bajo la configuración UPM, un conjunto pequeño de servicios de base reside en el núcleo, tal como mecanismos de comunicación entre procesos y de sincronización. Todo el código restante en el sistema se ejecuta en procesos separados con espacios de direccionamiento de memoria propios y protegidos entre sí. El paso de mensajes proporciona un mecanismo de comunicación al software mediante el cual un proceso puede solicitar un servicio de otro. Los servicios convencionales del sistema operativo, incluyendo los de file system, networking y drivers de dispositivos, se ponen en ejecución como procesos separados que se ejecutan en su propio espacio de direccionamiento de memoria, protegidos entre sí.

En el caso de los drivers de dispositivos, las rutinas de servicio de interrupciones se limitan a realizar las operaciones de ubicación necesarias, entonces se envía un mensaje asíncrono para activar el proceso que pone el driver de dispositivo apropiado en ejecución. Ya que el costo del paso de mensajes es mínimo, éste método puede competir con el monitor monolítico, evitando la necesidad de realizar demasiadas rutinas de servicio de interrupciones o de poner drivers de dispositivos en el núcleo para mejorar el rendimiento. Dado que los drivers de dispositivos se ejecutan como procesos individuales, pueden también ser comenzados y detenidos sobre la marcha, o ser reiniciados en caso de que ocurra un fallo.

Con la configuración de UPM, un fallo en una aplicación no puede dar lugar a un fallo general del sistema (a diferencia de los "ejecutivos de tiempo real") y la falla en un driver de dispositivo tampoco (a diferencia de los "sistemas operativos monolíticos").

Arquitectura Microkernel

Actualmente, para los desarrolladores de sistemas empotrados hay tres opciones clave a la hora de seleccionar la arquitectura del Sistema Operativo: Tiempo real ejecutivo, Monolítico y Microkernel. La arquitectura microkernel, núcleo principal del sistema operativo QNX, ofrece dos mecanismos principales que la hacen la mejor opción frente a otras arquitecturas en relación a confiabilidad, rendimiento, ciclos de desarrollo más cortos y escalabilidad:

- Sólo las primitivas del Sistema Operativo más fundamentales (señales, temporizadores, planificación) se manejan en el propio núcleo. Todos los demás componentes (*drivers*, sistemas de archivo, pilas de protocolo, aplicaciones de usuario) corren fuera del núcleo como procesos separados con memoria protegida (ver figura 2). La tolerancia a fallos es inherente al propio sistema.

- Todos los componentes se comunican a través de un único bus virtual de paso de mensajes que permite conectar o desconectar cualquier componente en cualquier momento. Adicionalmente, los mensajes pueden fluir transparentemente de un computador a otro, proporcionando acceso a cualquier recurso desde cualquier lugar de la red.

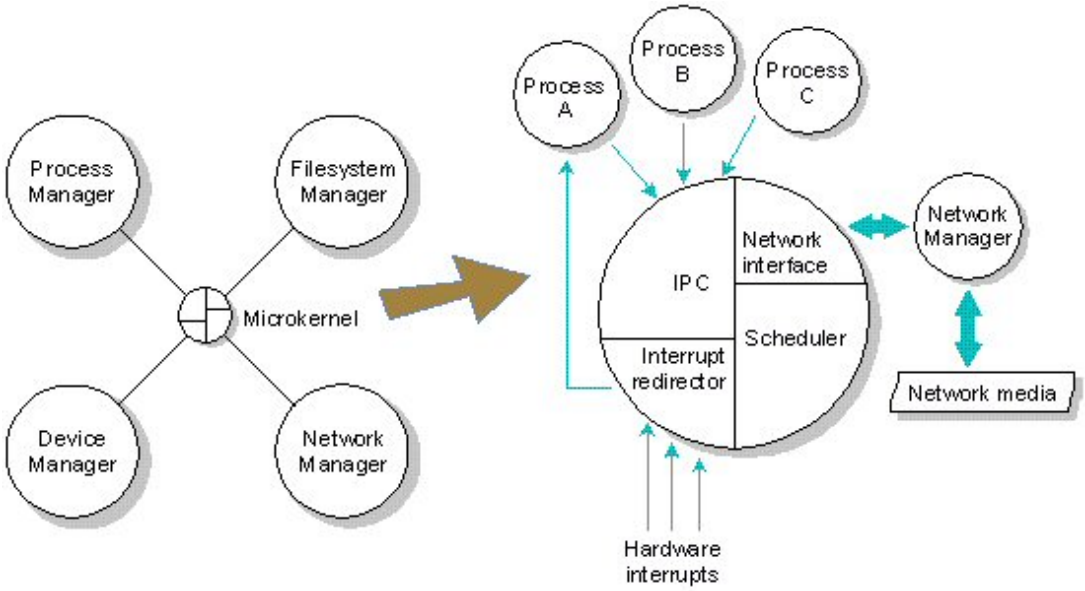


Fig. 2

Como resultado, capacidades que son muy difíciles de implementar en otros modelos de arquitectura, como en los sistemas Monolíticos, aparecen sin coste añadido en un sistema basado en arquitectura microkernel.

Resource Managers

En QNX Neutrino un *Resource Manager* es simplemente un programa a nivel de usuario que provee servicios a otros programas y, opcionalmente, se comunica con el hardware. En otros sistemas operativos, la mayor parte de esta funcionalidad es implementada mediante los *drivers* de los dispositivos. Sin embargo, al contrario que los *drivers* convencionales, un *Resource Manager* se ejecuta en un espacio de usuario con memoria protegida, permitiendo arrancarlo o pararlo de manera dinámica y depurarlo como si fuera una aplicación de usuario cualquiera.

2 Componentes del Sistema

A continuación se detallan los componentes, tanto hardware como software, que se han usado para la elaboración del Proyecto así como el propio software desarrollado.

2.1 Hardware empleado

El hardware empleado, que se detallará a continuación, consta de la propia tarjeta de adquisición de datos para la que se desarrolla el software, el ordenador PC anfitrión, donde esta instalada dicha tarjeta y el sistema operativo QNX 6.2, y el ordenador PC que se ha empleado para el desarrollo de los componentes software que son objeto de este Proyecto.

2.1.1 Tarjeta de adquisición de datos PCM-3718H de Advantech

La PCM-3718H es una tarjeta de adquisición de datos de alto rendimiento, desarrollada por Advantech, que se conecta a un conector PC/104 de la placa base de un PC o de otro modulo PC/104. A continuación se muestra una imagen de la misma.

La circuitería de escaneo automático de canales y la SRAM presente en la misma tarjeta permiten realizar conversiones A/D de múltiples canales con ganancia individual por cada canal.

Esta tarjeta es una versión avanzada de otros modelos de Advantech, los de la serie PCL-818, siendo compatible a nivel software con las PCL-818H y PCL-818HG.

Características:
- 8 entradas analógicas diferenciales o 16 en modo común, seleccionable por *jumper*.
- Conversión A/D de 12 bits de hasta 100kHz de frecuencia de muestreo con transferencia DMA.
- Valor de ganancia de cada canal de entrada analógico programable por software.
- Rango para cada canal de entrada analógico seleccionable por software.
- Dos canales digitales de entrada/salida de 8 bits, con niveles compatibles TTL.
- Flexibilidad en las opciones de lanzamiento de la conversión A/D: por software, mediante el contador programable o por pulso externo.
- Transferencia de datos controlada por programa, por rutina de interrupción o por DMA.

Especificaciones:

- *Entradas analógicas*:
  - Canales: 8 diferenciales o 16 en modo común, seleccionable por *jumper*.
  - Resolución: 12 bits.
  - Rango de entrada (programable por software, V_{DC}):
    - Bipolar: ±10, ±5, ±2.5, ±1.25, ±0.625.
    - Unipolar: 0~10, 0~5, 0~2.5, 0~1.25.
  - Frecuencia máxima de captura: 100kHz.
  - Precisión (depende de los valores de ganancia):

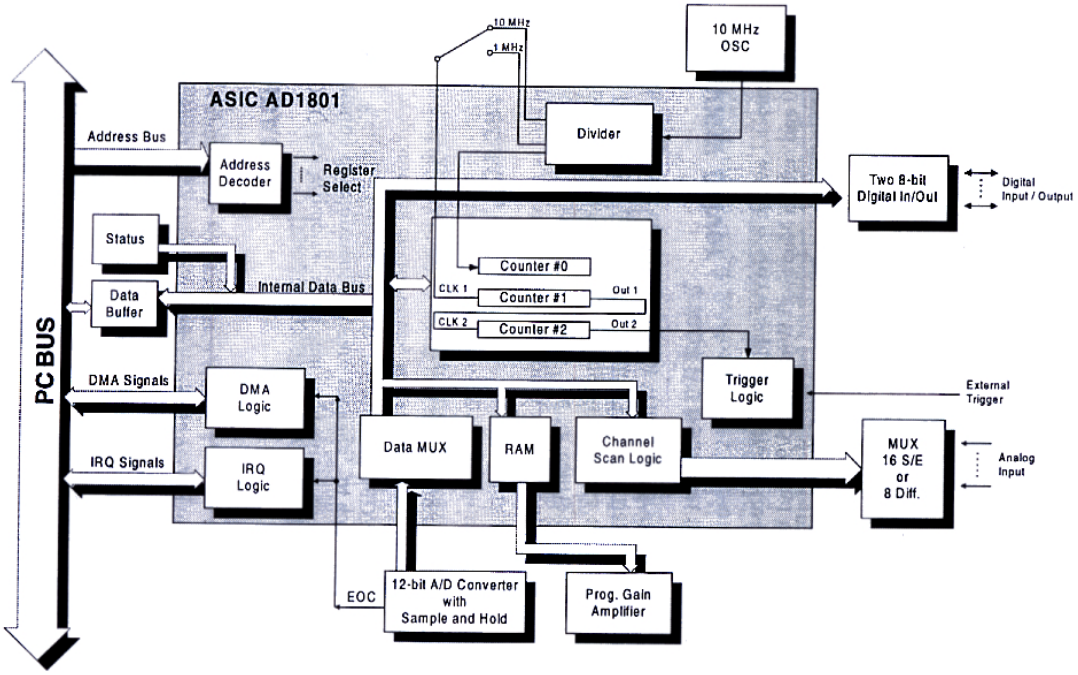
Ganancia	Precisión[2]
0.5, 1	0.01% de FSR ±1 LSB
2,4	0.02% de FSR ±1 LSB
8	0.04% de FSR ±1 LSB

  - Error de no-linealidad diferencial: ±1 LSB.
  - Impedancia de entrada: 1GΩ.
  - Modo de lanzamiento de la conversión: por software, por contador o externo.
  - Modo externo: compatible TTL.
- *Entradas/Salidas digitales*:
  - Canales: dos de 8 bits cada uno.
  - Niveles: TTL.
  - Voltaje de entrada;
    - Lógica 0: 0.8 V máximos.
    - Lógica 1: 2.0 V mínimos.
  - Voltaje de salida:
    - Lógica 0: 0.33 V máximos a 6 mA (*sink*).
    - Lógica 1: 3.84 V mínimos a 6 mA (*source*).
- *Contador programable:*
  - Dispositivo: Intel 8254 o equivalente.
  - Contadores: 3 canales de 16 bits. Los contadores 1 y 2 están permanentemente configurados en cascada, para proporcionar, si se configura de esa manera, una conversión A/D temporizada. El contador 0 esta reservado para uso futuro.
  - Base temporal (para la entrada de reloj del contador 1): 10MHz o 1MHz, seleccionable por *jumper*.
- *General:*
  - Consumo: típico de $5V_{DC}$ a 180mA y máximo de $5V_{DC}$ a 400mA.
  - Conector digital E/S: de 20 pin.
  - Conector de entradas analógicas: de 20 pin.
  - Temperatura de operación: 0 ~ 60 ºC
  - Temperatura de almacenamiento: -20 ~ 70 ºC
  - Humedad de operación: 5 ~ 95% sin condensación.
  - MTBF[3]: sobre 235,35 h a 25 ºC

Diagrama de bloques:

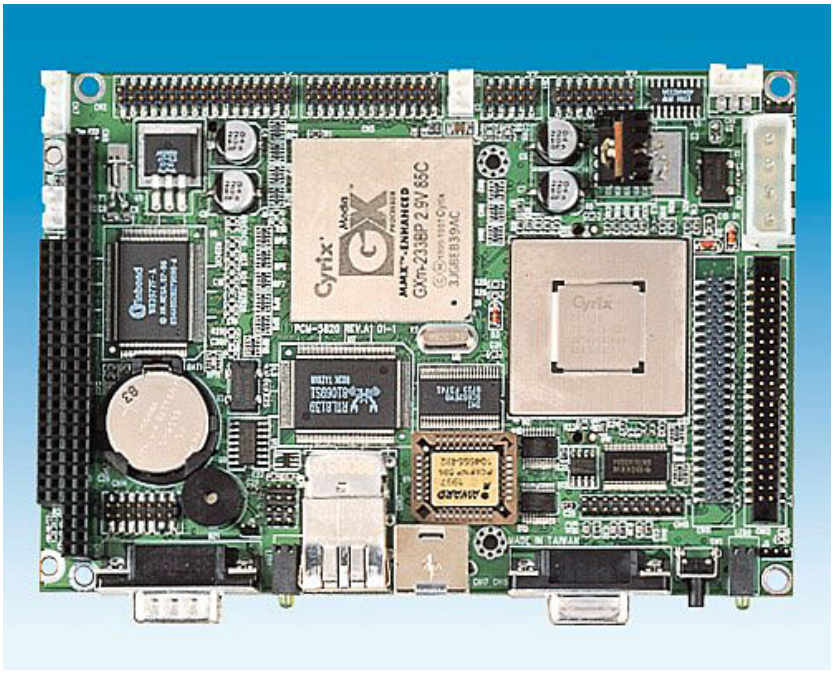
[2] LSB : *Least Significant Bit* (Bit Menos Significativo) ; FSR : Full-Scale Resolution (Resolución de la escala completa)

[3] *Mean Time Before Failure* (Tiempo Medio Entre Fallos)



2.1.2 Ordenador PC anfitrión

El ordenador PC anfitrión, donde se instala la tarjeta PCM3718-H así como el S.O. QNX y el software desarrollado en este Proyecto Fin de Carrera es el S.B.C.[4] PCM-5820 de ADVANTECH. A continuación se muestra una imagen del mismo.



Como características fundamentales, cabe destacar las siguientes:

[4] *Single Board Computer* (Computador integrado en una Unica Tarjeta)

- CPU AMD Geode a 233/300 MHz (según modelo).
- Consumo menor de 8W
- 1 slot SDRAM SODIMM, max 256Mb
- AWARD 256kb Flash BIOS
- Soporte para tarjetas CompactFlash™
- Interface de expansión PC-104
- E/S:
    - 1 x EIDE (Ultra DMA33)
    - 1 x FDD
    - 1 x Keyboard + 1 x Mouse
    - 1 x IrDA (1.0),
    - 1 x RS232/422/485
    - 1 x RS232
    - 1 x LPT
    - Audio: AMD CX 5530, compatible AC'97 2.0
    - 2 x USB 1.0
    - Ethernet 100Base-T IEEE 802.3u (chipset RealTek 8139C)
- Display:
    - Chipset AMD CX 5530
    - Resoluciones hasta 1024x768 @ 24bpp no entrelazado en display CRT y hasta 1024x768 @ 18bpp en display TFT

2.1.3 Ordenador PC de desarrollo

El ordenador PC de desarrollo es un Pentium IV a 1.7GHz, 256 Mb de RAM y sistema operativo Windows 2000 Professional. El sistema operativo QNX se instaló como un archivo único sobre una partición FAT en la que estaba instalado Windows 98 S.E. Posteriormente se instalaron, en QNX, los paquetes necesarios, incluido el I.D.E.[5], para el desarrollo del Proyecto.

Para la redacción del Proyecto se ha usado el programa Microsoft Word 2000 SR-1.

2.2 Software empleado

El software empleado esta directamente relacionado con los requerimientos iniciales de este Proyecto, como es el hecho que el driver software debe ser desarrollado para el sistema operativo QNX 6.2.

2.2.1 Herramientas de desarrollo

Las herramientas imprescindibles para el desarrollo de este proyecto han sido el I.D.E. (ver figura 3), que permite desarrollar aplicaciones y librerías en C/C++, así como aplicaciones en Java. Este entorno facilita el desarrollo de aplicaciones al integrar editor con resaltado de sintaxis, compilador y depurador, además de utilidades varias, como buscador de expresiones o *wizard* (literalmente "hechicero") para generar esqueletos de aplicaciones (y los *makefile* correspondientes).

[5] *Integrated Development Environment* (Entorno Integrado de Desarrollo)

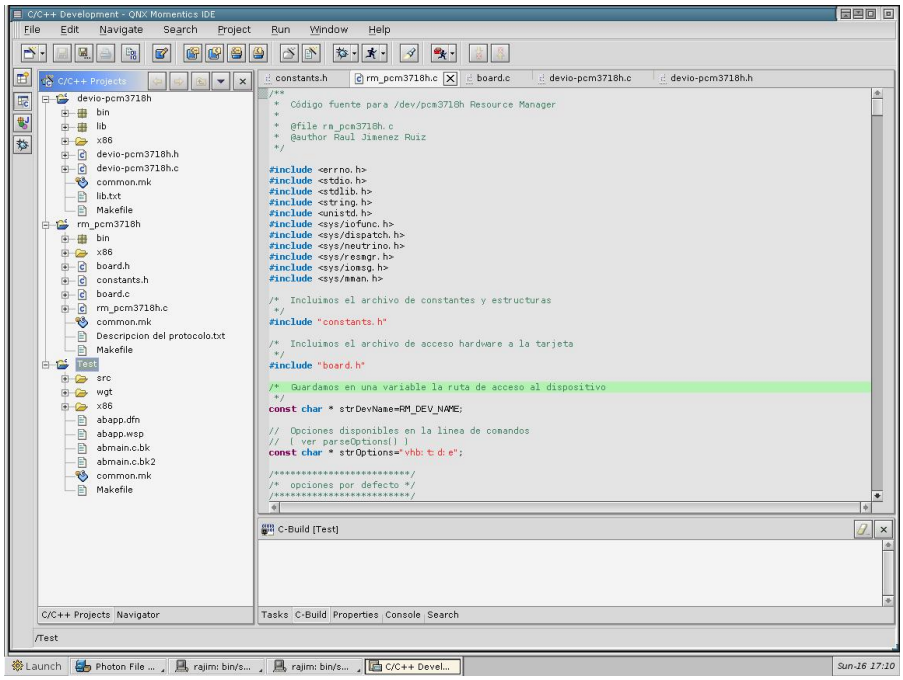


Fig. 3

Otra herramienta indispensable ha sido la aplicación denominada *AppBuilder* (ver figura 4), integrada con el I.D.E., que permite facilitar el desarrollo de aplicaciones gráficas para Photon. Con esta herramienta, se puede diseñar el aspecto de la aplicación gráfica, arrastrando controles a la ventana de la aplicación que se esta diseñando, dándoles nombre y especificando nombres de funciones que responderán a los eventos generados por el control. El AppBuilder es luego responsable de crear los archivos necesarios (de fuente en C, makefiles, de recursos,...) para que, al compilarlos, obtengamos como resultado la aplicación gráfica que se ha diseñado previamente.

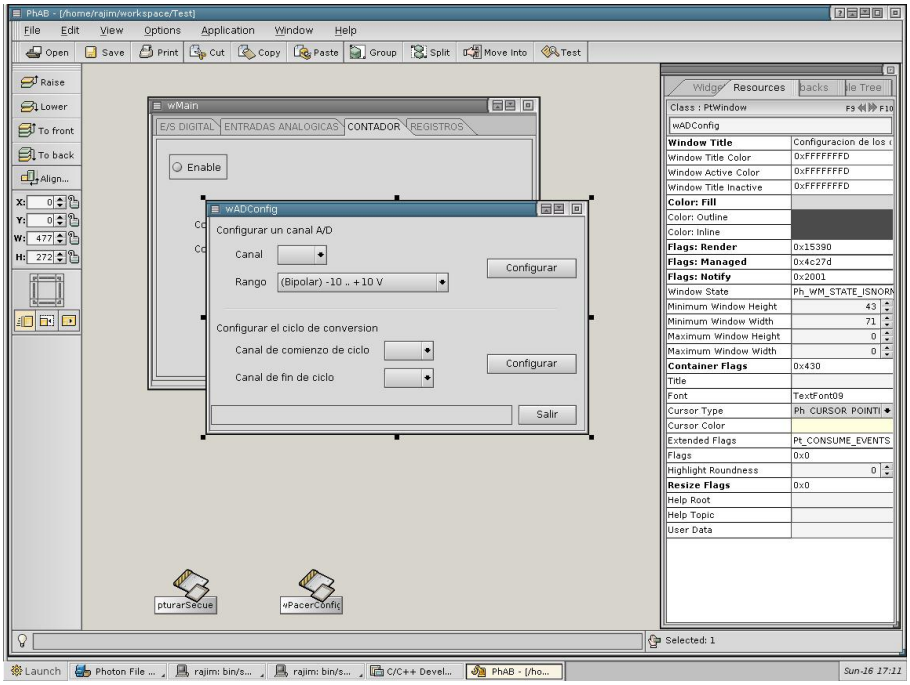


Fig. 4

2.3 Software desarrollado

En los siguientes apartados se da una breve descripción de los componentes software desarrollados, tanto el *driver* de la tarjeta (*Resource Manager* + Librería compartida) como la aplicación GUI de ejemplo. Esta breve descripción tiene el objeto de que el lector pueda hacerse una idea global del alcance y función de los distintos componentes, así como la relación que existe entre ellos. En la figura siguiente se puede ver dicha relación:

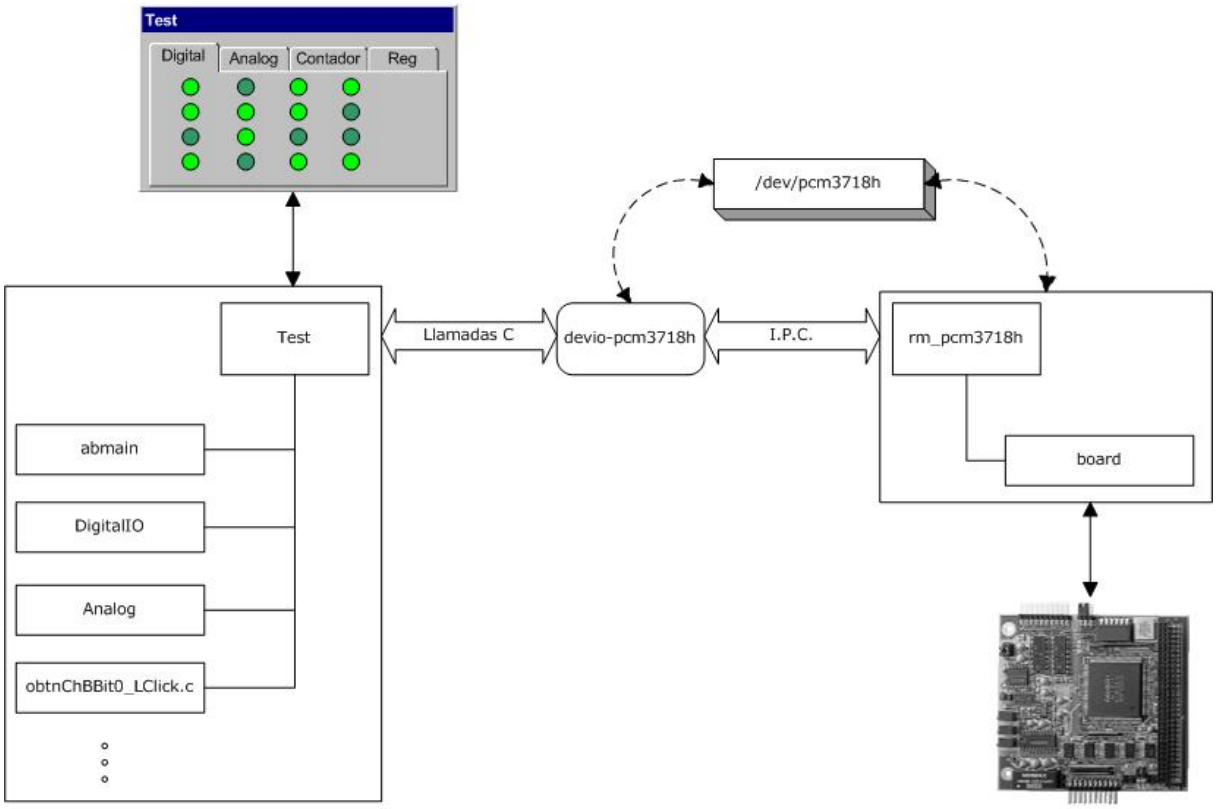


Fig. 5

2.3.1 Resource Manager

En QNX, un *Resource Manager* es un programa que provee un interface a través del espacio de nombres usando las funciones estándar POSIX de E/S: `read()`, `write()`, `select()`, ...

Se puede describir también como un programa servidor, ejecutándose a nivel de usuario (no a nivel de núcleo, como le ocurre a los *device drivers* en otros sistemas operativos), que acepta mensajes de otros programas y, opcionalmente, se comunica con el hardware. Este programa servidor es un proceso que registra un nombre en el espacio de nombres (*pathname space*), por ejemplo `/dev/ser1`, y, tras haberlo registrado, otros procesos pueden abrir ese nombre usando la función de la librería estándar de C `open()` y usar `read()` o `write()` con el descriptor de archivo devuelto. Esto hace que el *resource manager* reciba una petición de `open`, seguidas de peticiones `read` o `write`.

Sin embargo, un *resource manager* no esta restringido a manejar sólo llamadas a `open()`,`read()` y `write()`, puede manejar cualquier función que haga uso de descriptores de archivo o punteros

a archivo, así como otras formas de IPC[6], como `MsgSend()`, `MsgReply()`, ...

Al ejecutarse a nivel de usuario, el programa no esta restringido en lo que puede hacer (como lo están los *device drivers* creados en otros sistemas operativos), aunque para poder llamar con éxito a `attach` (función que permite registrar el nombre en el espacio de nombres) debe ser ejecutado como usuario `root`.

Uno de los valores añadidos sobre los *resource managers* es que pueden comunicarse con las utilidades de línea de comandos (aquellas que son de manejo de archivos), por ejemplo, si tuviéramos un *resource managers* manejando la comunicación con un robot, se podría hacer algo como: `echo 90 > /dev/robot/arm1/angle`.

El nombre del archivo ejecutable que lanza el *resource manager* es, en este caso, `rm_pcm3718h`.

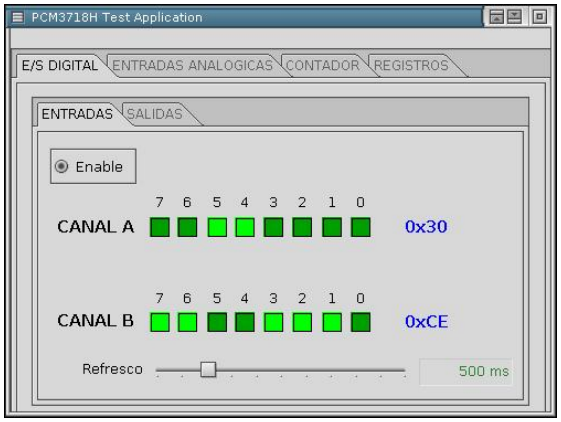
2.3.2 *Librería compartida*

Solo con el *Resource Manager* ya es posible comunicarse con la tarjeta, tanto para configurarla como para leer/escribir datos. Sin embargo, el desarrollo de una nueva aplicación que haga uso de la misma requeriría del conocimiento de la cantidad y significado de los registros así como los modos de configuración y operación. Para simplificar este proceso se ha desarrollado una librería, de tipo compartida, que hace las veces de A.P.I.[7] de programación de la tarjeta, facilitando y estandarizando el uso de la tarjeta de adquisición de datos.

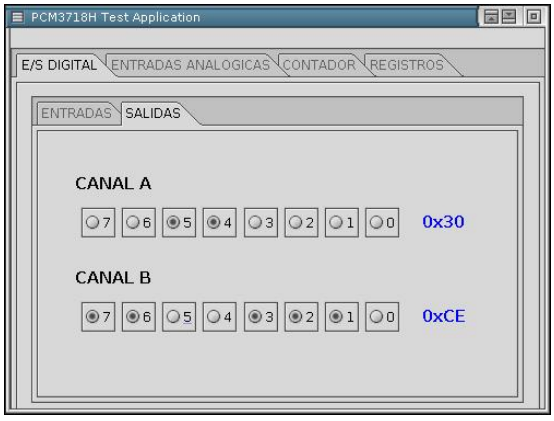
El nombre de la librería compartida es, en este caso, `devio-pcm3718h`.

2.3.3 *Aplicación GUI*

Como aplicación de ejemplo de las capacidades de la tarjeta y, al mismo tiempo, como herramienta de configuración de la misma, se ha desarrollado una aplicación gráfica para Photon. Las siguientes figuras dan una idea del aspecto de la aplicación en ejecución.



Entradas digitales

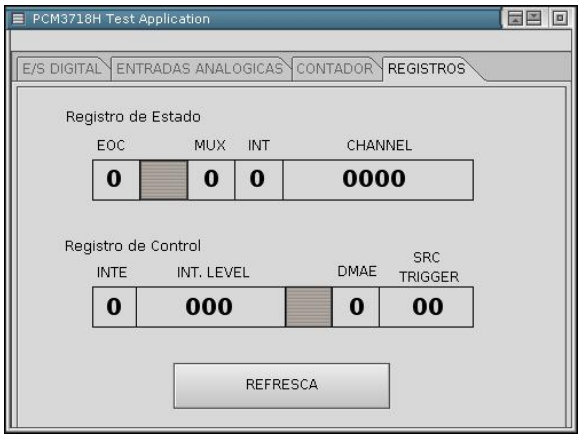


Salidas digitales

[6] *Inter-Process Communication* (Comunicación Entre Procesos)
[7] *Application Programming Interface* (Interfase de Programación de Aplicaciones)

Entradas analógicas

Contadores



Registros

La idea fundamental que subyace en el desarrollo de esta aplicación es la posibilidad de experimentar con las distintas funciones de que dispone la tarjeta. Al mismo tiempo, también sirve como medio para asegurar la correcta instalación y configuración de la misma.

En su desarrollo se ha incidido más en acumular el mayor número posible de características de la tarjeta que en la usabilidad de la aplicación, pues no es su finalidad el convertirse en una aplicación de uso comercial, sino el de ser una herramienta de pruebas.

El nombre del archivo ejecutable que lanza la aplicación de ejemplo es, en este caso, `Test`.

2.3.4 Aplicación GUI IRQ

La aplicación GUI comentada en el apartado anterior permite configurar la captura A/D y probar el método "poling" de muestreo, es decir, a petición del usuario. Para ello, la aplicación es capaz de lanzar un hilo en paralelo que, periódicamente, pide a la tarjeta que realice una conversión A/D. Este método es muy lento, en general, para aplicaciones de control que requieran tiempos de respuesta corto, pero es útil en sistemas de supervisión o para probar la correcta configuración del sistema.

Otro modo de funcionamiento disponible en la tarjeta consiste en capturar datos mediante interrupciones del sistema. De esta manera, y usando el *pacer* para temporizar las capturas, la tarjeta informa al sistema de que tiene un dato disponible activando una interrupción (con la que previamente ha sido parametrizada). El *Resource Manager* es capaz de capturar esa interrupción,

procesar los datos provenientes de la tarjeta y servirlos a la aplicación cliente que este esperando por ellos.

Esta aplicación permite seleccionar la IRQ del sistema con la que parametrizar la tarjeta y muestra los datos capturados a razón de 4 por segundo (para lo que se configura previamente el *pacer* para que lance capturas A/D a 4 Hz).

En este caso, esa aplicación cliente es la que se ha denominado "Aplicación GUI IRQ" y el archivo ejecutable que la lanza se llama `TestIRQ`.

3 Descripción detallada

A continuación se procede a describir con más detalle los componentes que forman parte de este proyecto, así como el sistema operativo en el que se implantan.

3.1 Resource Manager.

El código fuente que da lugar al *Resource Manager* de este proyecto se reparte entre los archivos `rm_pcm3718h.c`, `constants.h`, `board.h`, `board.c`, `isr.h` y `isr.c`; brevemente, se pueden describir del siguiente modo:

- `rm_pcm3718h.c`: es el archivo que contiene la función `main` y, en ella, el bucle de recepción y procesado de mensajes, además de las funciones que "construyen" las capas que forman el *resource manager*.
- `constants.h`: contiene las constantes y macros que sirven para la comunicación con el *resource manager*.
- `board.h`: contiene los prototipos de las funciones que responden a los mensajes que le llegan al *resource manager*.
- `board.c`: contiene la implementación de las funciones que responden a los mensajes que le llegan al *resource manager*.
- `isr.h`: contiene los prototipos de las funciones que se encargan de la gestión de la conversión A/D mediante interrupciones.
- `isr.c`: contiene la implementación de las funciones que se encargan de la gestión de la conversión A/D mediante interrupciones.

3.1.1 Función main

El diagrama de flujo de la función main se presenta en la figura 3. En dicho diagrama se distinguen tres bloques principales: un primer bloque de análisis de la línea de comandos (en busca de opciones seleccionadas por el usuario), luego el bloque de inicialización (tanto de la estructura del *resource manager* como de los parámetros iniciales de la tarjeta) y, por último, el bloque que implementa el bucle de recepción de mensajes.

Para el tratamiento de los mensajes se podía haber optado por una sentencia `switch` en la función `msg_callback`, que es la que primera se llama tras recibir cualquier mensaje. Sin embargo, esa estructura sería poco práctica para añadir nuevo código que responda a nuevos mensajes, o para modificar el existente.

En lugar de eso, se ha optado por una estructura basada en un array de punteros a función, como se puede ver en la figura 4. Mediante la llamada a la función `message_attach` registramos la función que se encarga de recepcionar los mensajes (en este caso `msg_callback`), la cual, a su vez, cuando recibe un mensaje, llama a `RMProcessMsg`, que se encarga de decodificar el mensaje y llamar a la función manejadora correspondiente.

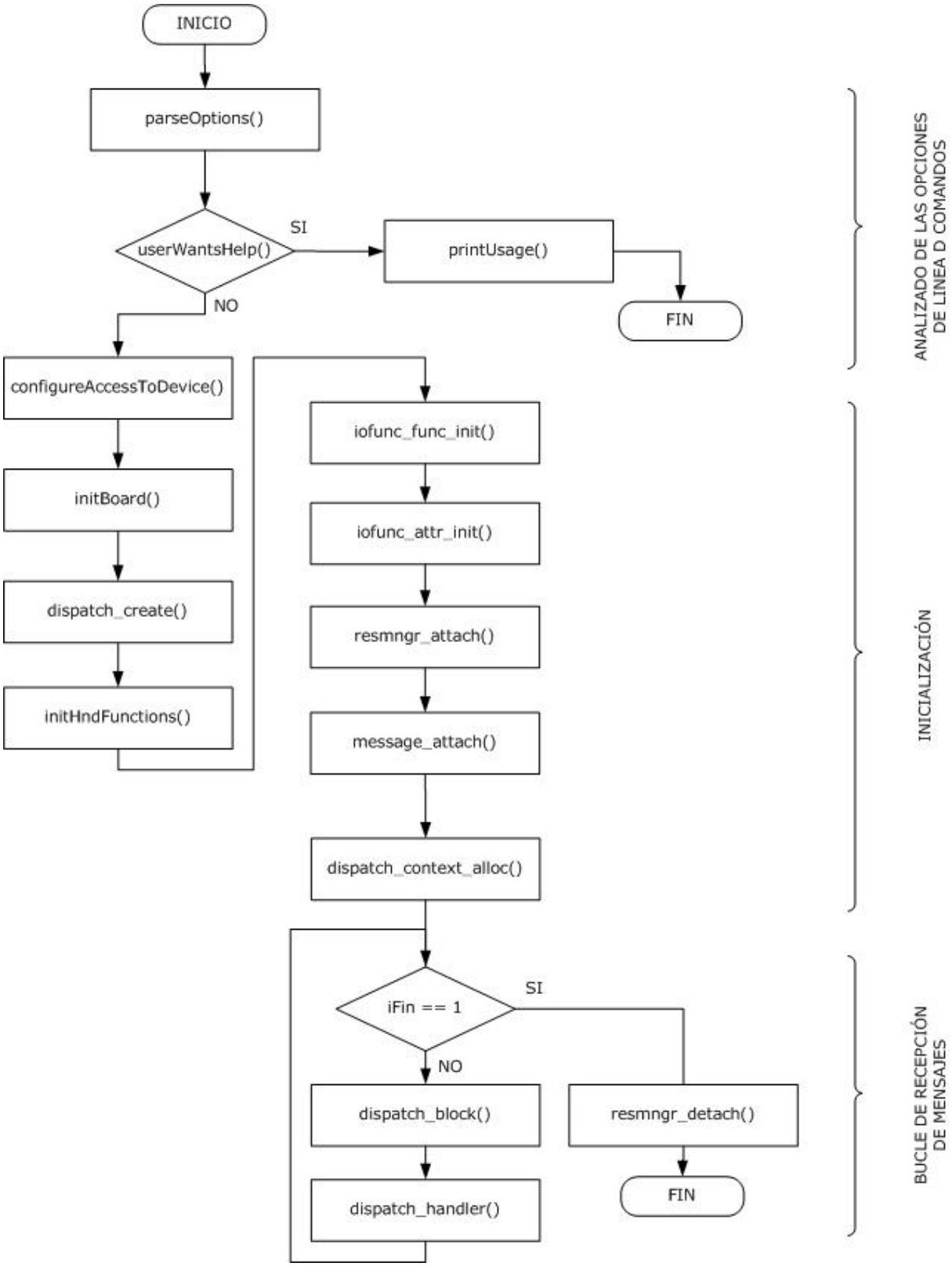

INICIO
parseOptions()
userWantsHelp()
SI
printUsage()
FIN
NO
configureAccessToDevice()
initBoard()
dispatch_create()
initHndFunctions()
iofunc_func_init()
iofunc_attr_init()
resmngr_attach()
message_attach()
dispatch_context_alloc()
iFin == 1
SI
NO
dispatch_block()
dispatch_handler()
resmngr_detach()
FIN
ANALIZADO DE LAS OPCIONES DE LÍNEA D COMANDOS
INICIALIZACIÓN
BUCLE DE RECEPCIÓN DE MENSAJES

Fig. 6

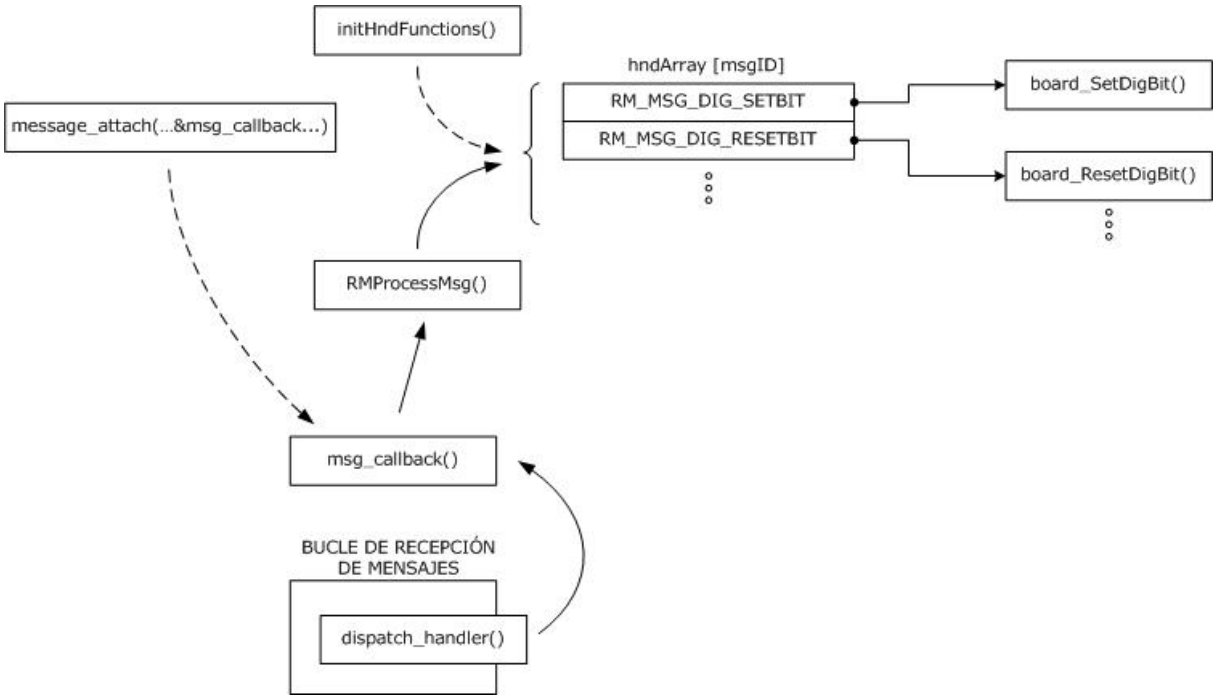


Fig. 7

Se comentará a continuación el código que compone la función `main`, al menos, los bloques de código y las llamadas a función que son más significativas.

Antes de nada, reseñar que, para que las llamadas a las funciones `printf` y `fprintf` sean útiles, el resource manager debe lanzarse desde una consola.

Primero se analiza la línea de comandos para comprobar las opciones indicadas por el usuario. En caso de que haya algún error en el analizado, se presenta por consola al usuario un listado de las opciones disponibles, así como los parámetros de cada opción y luego acaba la ejecución.

```
if(parseOptions(argc, argv)!=RM_NO_ERROR)
{
      fprintf(stderr,"%s: (!) Error en los argumentos de la línea
      de comandos.\n",strDevName);
      printUsage();
      return -1;
}
```

Si el usuario especifica como opción "-h", indicando una petición de ayuda, la función `userWantsHelp()` devolverá 1 (="true"), entonces se presentara un listado de las opciones disponibles, así como los parámetros de cada opción y luego acaba la ejecución.

```
if(userWantsHelp())
{
      printUsage();
      return 0;
}
```

A partir de aquí, si el usuario ha especificado en la línea de comandos la opción "-v", cada vez que se ejecute una función de cierta importancia, se escribirá en la consola un mensaje descriptivo. Como ejemplo valga el siguiente extracto del código:

```
if(isVerbose())
```

```
      printf ("%s: Iniciando...\n", strDevName);
```

El siguiente paso es configurar el acceso al dispositivo, que consiste en hacer que el hilo principal adquiera privilegios de E/S y mapear las direcciones de E/S del dispositivo. Esto se consigue con la llamada a la función `configureAccessToDevice`.

```
if(configureAccessToDevice()==0)
{
      fprintf(stderr,"%s: (!) Error configurando el acceso al
            dispositivo.\n",strDevName);
      return -1;
}
```

Luego inicializamos los valores por defecto de la tarjeta con la llamada a la función `initBoard`. Esos valores, usados dentro de dicha función, son los siguientes:

- Rango de conversión de los canales analógicos — Bipolar, -10 ... 10 V
- Canales de inicio y fin del ciclo de conversión A/D — Inicio = canal 0
Fin = canal 0
- Tipo de conversión A/D — "sin interrupciones" +
"sin DMA" +
"triggering por software"
- Modo de los contadores 1 y 2 — "transferencia LSBMSB" +
"modo 3" +
"contaje binario"

```
if(!initBoard())
{
      fprintf(stderr,"%s: (!) Error inicializando los valores por
            defecto.\n",strDevName);
      return -1;
}
```

Se crea ahora la estructura de dispatch que utilizará el bucle de recepción de mensajes.

```
dpp = dispatch_create();
if (dpp == NULL)
{
      fprintf (stderr, "%s: (!) Error en dispatch_create: %s\n",
             strDevName, strerror (errno));
      exit (1);
}
```

Luego inicializamos el array de manejadores de mensajes.

```
initHndFunctions();
```

A continuación inicializamos la estructura de atributos del resource manager con los valores por defecto. Esa estructura es la que luego usaremos como parámetro en la llamada a la función `resmgr_attach`.

```
memset (&rattr, 0, sizeof (rattr));
rattr.nparts_max=1;
```

```
rattr.msg_max_size=2048;
```

Y luego inicializamos las estructuras de funciones *connect* y de E/S con sus parámetros por defecto, sobrescribiendo a continuación las funciones para las que añadimos funcionalidad, que son las funciones que responderán a las llamadas POSIX *open*, *read* y *write*. Inicializamos también los atributos de las funciones de E/S.

```
iofunc_func_init (_RESMGR_CONNECT_NFUNCS, &connect_funcs,
                  _RESMGR_IO_NFUNCS, &io_funcs);
connect_funcs.open = io_open;
io_funcs.read = io_read;
io_funcs.write = io_write;

iofunc_attr_init(&ioattr, S_IFCHR | 0666, NULL, NULL);
```

Se llama ahora a la función que registra este proceso (el que representa esta función *main*) en el espacio de nombres como un resource manager.

```
pathID = resmgr_attach(dpp, &rattr, strDevName, _FTYPE_ANY, 0,
                       &connect_funcs, &io_funcs, &ioattr);
```

Mediante la llamada a la función `message_attach` registramos, para la estructura de *dispatch* anteriormente creada, la función que se encarga de recepcionar los mensajes (en este caso, la función `msg_callback`) así como el rango de mensajes (rango de enteros) aceptables.

```
memset(&msgAttr,0,sizeof(msgAttr));
msgAttr.nparts_max=1;
msgAttr.msg_max_size=4096;
iRes=message_attach(dpp,&msgAttr,
                  _IO_MAX+RM_MSG_FIRST,_IO_MAX+RM_MSG_LAST,
                  msg_callback,NULL);
```

El bucle de recepción de mensajes se implementa con un bucle `while` cuya condición depende de una variable (`iFin`) cuyo valor esta siempre a cero, hasta que alguien escriba (mediante la llamada POSIX `write`) en el Resource Manager la palabra "terminate" (por ejemplo, haciendo desde consola "`echo terminate > /dev/pcm3718h`"), momento en el que el Resource Manager termina su ejecución cíclica saliendo del bucle.

```
ctp = dispatch_context_alloc(dpp);
while (!iFin)
{
      if ((ctp = dispatch_block(ctp)) == NULL)
      {
            fprintf (stderr, "%s:(!) Error en dispatch_block: %s\n",
                        strDevName, strerror (errno));
            exit (1);
      }
      dispatch_handler(ctp);
}
```

Después de salir del bucle, se deshace la asociación de este proceso al nombre reservado anteriormente en el espacio de nombres en `/dev`, de manera que otro proceso pueda hacer uso de ese nombre. En caso contrario, el nombre quedaría reservado pero ningún proceso respondería a los mensajes (incluidas las llamadas POSIX) dirigidos a dicho nombre.

```
iRes=resmgr_detach(dpp,pathID,_RESMGR_DETACH_ALL);
```

```
if(iRes==-1)
{
      fprintf (stderr, "%s:(!) Error en resmg_detach: %s\n",
                        strDevName, strerror (errno));
      exit (1);
}
return 0;
```

— Procesado de los mensajes

Como se vio en párrafos anteriores, se ha optado por una estructura basada en un array de punteros a función. Mediante la llamada a la función `message_attach` registramos la función que se encarga de recepcionar los mensajes (en este caso `msg_callback`), la cual, a su vez, cuando recibe un mensaje, llama a `RMProcessMsg`, que se encarga de decodificar el mensaje y llamar a la función manejadora correspondiente. Todo este proceso se verá a continuación con algo más de detalle.

El esquema de la función `msg_callback` se puede observar en la figura 5, donde se comprueba que la tarea principal es redirigir el procesado del mensaje hacia la función `RMProcessMsg` y, tras eso, responder al proceso llamante.

Fig. 8

La siguiente figura (fig. 6) representa, como diagrama de bloques, el funcionamiento de `RMProcessMsg`. La llamada al manejador del mensaje se realiza indexando en el array `hndArray` con el `ID` del mensaje, pero no con el `ID` del `message_context_t` entregado a `msg_callback` si no con `(msg->id - _IO_MAX)`, ya que al registrar el manejador de mensaje y el rango de mensajes válidos con la llamada a `message_attach`, dicho rango se establece en `[_IO_MAX + RM_MSG_FIRST,_IO_MAX + RM_MSG_LAST]` y el rango válido de índices para el array de manejadores de mensaje es `[RM_MSG_FIRST,RM_MSG_LAST]`.

Fig. 9

Como ejemplo de esta arquitectura de respuesta a mensajes, veamos como se maneja la pulsación, en la aplicación GUI, del botón que representa el bit 0 del canal A de las salidas digitales:

1. Dicha pulsación hace que la aplicación GUI llame a la función `obtnChABit0_LClick`, que se encuentra el archivo C del mismo nombre y cuyo código fuente se reproduce a continuación:

```
int obtnChABit0_LClick( PtWidget_t *widget, ApInfo_t *apinfo,
PtCallbackInfo_t *cbinfo )
{
/* eliminate 'unreferenced' warnings */
widget = widget, apinfo = apinfo, cbinfo = cbinfo;

RefreshOutput(ABW_obtnChABit0,ABW_lblHexChA,RM_DIG_CHANNEL_A,0);

return( Pt_CONTINUE );
}
```

   donde la función `RefreshOutput` se encuentra en el archivo `DigitalIO.c` y su código fuente es el que sigue:

```
void RefreshOutput(PtWidget_t * button,PtWidget_t * label,int
channel,int bit)
{
char hex[15],*hexptr=hex;
unsigned short *state;
int status=-1;

PtGetResource(button,Pt_ARG_ONOFF_STATE,&state,0);

if(*state==0)
{
      if(channel==RM_DIG_CHANNEL_A)
            status=ResetDigBitChannelA(bit);
      else if(channel==RM_DIG_CHANNEL_B)
            status=ResetDigBitChannelB(bit);
}
else
{
      if(channel==RM_DIG_CHANNEL_A)
            status=SetDigBitChannelA(bit);
      else if(channel==RM_DIG_CHANNEL_B)
            status=SetDigBitChannelB(bit);
}

Bin2Hex(status,hex);
PtSetResource(label,Pt_ARG_TEXT_STRING,hexptr,0);

}
```

Las funciones `SetDigBitChannelA`, `SetDigBitChannelB`, `ResetDigBitChannelA` y `ResetDigBitChannelB` son funciones "helper" que encapsulan llamadas a las funciones de la librería que realizan las funciones correspondientes a través de la librería y, al mismo tiempo, recuperan el estado actual de las entradas/salidas digitales, para refrescar el estado visual de los botones que representan las salidas digitales. Como ejemplo véase a continuación el código fuente de `SetDigBitChannelA`:

```
int SetDigBitChannelA(int iBit)
{
int status=0,mask=0;

if(iBit<0 || iBit>7)
      return -1;

BYTE_SET_BIT(mask,byte_mask[iBit]);
if(pcm3718h_SetDigBit(RM_DIG_CHANNEL_A,mask)!=SO_NO_ERROR)
      return -1;

if(pcm3718h_ReadDigStatus(&status,NULL)!=SO_NO_ERROR)
      return -1;
else
      return status;
}
```

2. Las funciones de librería que comunican con el *Resource Manager* son, en este caso, `pcm3718h_SetDigBit` y `pcm3718h_ReadDigStatus`. Se encuentran en el archivo devio-`pcm3718h.c` y su código fuente es el que sigue:

```
int pcm3718h_SetDigBit(int iChannel,int iBitMask)
```

```
{
	pcm3718h_msg_t reply;
	unsigned char data[2];
	int ret;

	if(iChannel!=RM_DIG_CHANNEL_A && iChannel!=RM_DIG_CHANNEL_B)
		return SO_ERROR_BAD_ARGUMENT;

	//	como solo hay dos canales de 8 bits cada uno ...
	data[0]=LOW_BYTE(iChannel);
	data[1]=LOW_BYTE(iBitMask);

	ret=sendMsg(RM_MSG_DIG_SETBIT,data,2,&reply);

	RETURN_ERR(ret,reply);
}

int pcm3718h_ReadDigStatus(int *piChannelA,int *piChannelB)
{
	pcm3718h_msg_t reply;
	unsigned char data[2];
	int ret;

	if(piChannelA==NULL && piChannelB==NULL)
		return SO_ERROR_BAD_ARGUMENT;

	if(piChannelA != NULL) data[0]=1; else data[0]=0;
	if(piChannelB != NULL) data[1]=1; else data[1]=0;

	ret=sendMsg(RM_MSG_DIG_READSTATUS,data,2,&reply);

	if(piChannelA != NULL) *piChannelA=RM_NULLPARAM;
	if(piChannelB != NULL) *piChannelB=RM_NULLPARAM;

	if(piChannelA != NULL) *piChannelA=(int)(reply.data[0]);
	if(piChannelB != NULL) *piChannelB=(int)(reply.data[1]);

	RETURN_ERR(ret,reply);
}
```

En ellas es de destacar la llamada a la función `sendMsg`, que es la que realmente manda el mensaje al Resource Manager, que se encuentra también en el archivo y su código fuente es el que sigue:

```
int sendMsg(unsigned int msgN,unsigned char *pData, unsigned int
dataLen,pcm3718h_msg_t *msgReply)
{
int fd,ret;
pcm3718h_msg_t *send;
pcm3718h_msg_t *reply;

if(pData==NULL || dataLen>=RM_MSG_MAX_LEN || msgN<RM_MSG_FIRST ||
msgN>RM_MSG_LAST)
	return RM_ERROR_BAD_ARGUMENT;

send=(pcm3718h_msg_t *)malloc(sizeof(pcm3718h_msg_t));
reply=(pcm3718h_msg_t *)malloc(sizeof(pcm3718h_msg_t));

memset(send,0,sizeof(pcm3718h_msg_t));
memset(reply,0,sizeof(pcm3718h_msg_t));
```

```
// ¡¡¡ ATENCION:
// msgN no puede ser 0, ya que el id del mensaje debe ser > _IO_MAX
send->id=msgN+_IO_MAX;
copyData(send->data,pData,(dataLen<RM_MSG_MAX_LEN    ?    dataLen    :
RM_MSG_MAX_LEN));

fd=open(RM_DEV_NAME,O_RDWR);
if(fd==-1)
{
      close(fd);
      free(send);
      free(reply);

      return SO_ERROR_OPN_DEV;
}

ret=MsgSend(fd,send,sizeof(pcm3718h_msg_t),reply,sizeof(pcm3718h_msg_
t));
if(ret==-1)
{
      printf("<error %d: %s>\n",errno,strerror(errno));

      close(fd);
      free(send);
      free(reply);

      return SO_ERROR_MSG_SEND;
}

if(msgReply!=NULL)
{
      msgReply->id=reply->id;
      msgReply->error=reply->error;
      copyData(msgReply->data,reply->data,RM_MSG_MAX_LEN);
}

close(fd);
free(send);
free(reply);

return SO_NO_ERROR;
}
```

En este código se ve, como punto final de la parte del cliente, la llamada a la función de la API de QNX `MsgSend`, que es quien realmente manda el mensaje.

3. Al *Resource Manager* le llega un mensaje `RM_MSG_DIG_SETBIT` seguido del mensaje `RM_MSG_DIG_READSTATUS`. Veamos como se procesa el primero.

   Después de salir del bloqueo de la función `dispatch_block`, que esta dentro del bucle principal de escucha/respuesta de mensajes en la función `main`, se llama a la función `dispatch_handler`, como puede verse en bloque de código siguiente, comentado cuando se analizó en detalle la función main:

```
while (!iFin)
{
      if ((ctp = dispatch_block(ctp)) == NULL)
      {
```

```
            fprintf (stderr, "%s:(!) Error en dispatch_block: %s\n",
                        strDevName, strerror (errno));
            exit (1);
      }
      dispatch_handler(ctp);
}
```

La función `dispatch_handler` llama a la función `msg_callback`, que fue previamente registrada como función de respuesta a los mensajes en la llamada a `message_attach`.

Dicha función, `msg_callback`, tiene el siguiente código:

```
int  msg_callback(message_context_t  *ctp,  int  type,unsigned  flags,
void *handle)
{
      pcm3718h_msg_t *msg;
      int ret;

      if(isVerbose())
            printf("%s: Entrando en msg_callback\n",strDevName);

      msg=(pcm3718h_msg_t *)ctp->msg;

      if(isVerbose())
            printf("%s: Procesado del mensaje\n",strDevName);

      //    procesamos el mensaje
      ret=RMProcessMsg(msg);

      if(isVerbose())
            printf("%s: Fin de procesado del mensaje\n",strDevName);

      //    y lo devolvemos como respuesta
      MsgReply(ctp->rcvid,EOK,msg,sizeof(pcm3718h_msg_t));

      if(isVerbose())
            printf("%s: Saliendo de msg_callback\n",strDevName);

      return 0;
}
```

En el que puede observarse como el manejo del mensaje, su procesado y la construcción de la respuesta lo cede a la función `RMProcessMsg`, cuyo código puede verse a continuación:

```
int RMProcessMsg(pcm3718h_msg_t *msg)
{
      int ret,id;

      if(msg==NULL)
            return RM_ERROR_BAD_ARGUMENT;

      id=msg->id - _IO_MAX; // ver message_attach() en main()

      if(id<RM_MSG_FIRST || id>RM_MSG_LAST)
      {
            //    No se conoce el mensaje, y asi lo indicamos
            msg->error=RM_ERROR_UNK_MSG;

            if(isVerbose())
```

```
                  printf("%s:     [error:     mensaje     desconocido
(%d)]\n",strDevName,id);

      }
      else
      {
            // Llamamos al handler
            if(hndArray[id]!=NULL)
            {
                  if(isVerbose())
                        printf("%s: Llamando al manejador del mensaje
msg_id=%d\n",strDevName,id);

                  ret=(*hndArray[id])(msg->data);

                  if(isVerbose())
                        printf("%s: El manejador del mensaje msg_id=%d
devuelve %d\n",strDevName,id,ret);
            }
            else
            {
                  if(isVerbose())
                        printf("%s:  [error:  no  hay  manejador  para
msg_id=%d]\n",strDevName,id);

                  ret=RM_ERROR_NO_HANDLER;
            }

            msg->error=ret;
      }

      //    En todo caso, siempre hay una respuesta
      msg->id=RM_MSG_REPLY;

      return RM_NO_ERROR;
}
```

Tras comprobar que es un mensaje válido, llama a su manejador (si es que tiene asignado alguno) mediante el código `ret=(*hndArray[id])(msg->data)`. Para el mensaje `RM_MSG_DIG_SETBIT` resulta que el vector de manejadores (`hndArray`) contiene un puntero a la función `board_SetDigBit`, que se encuentra en el archivo `board.c` y tiene el siguiente código:

```
int board_SetDigBit(unsigned char * ucParams)
{
      int iChannel, iBitMask;

      if(isVerbose())
            printf("%s: > en board_SetDigBit\n",RM_DEV_NAME);

      if(ucParams==NULL)
            return RM_ERROR_BAD_ARGUMENT;

      iChannel=ucParams[0];
      iBitMask=ucParams[1];

      // Si DI0 es 'external trigger', forzamos ese bit a cero
      // (el bit menos significativo del canal A)
      if(iChannel==RM_DIG_CHANNEL_A)
```

```
            CHECK_DI0(iBitMask);

      if(iChannel!=RM_DIG_CHANNEL_A && iChannel!=RM_DIG_CHANNEL_B)
            return RM_ERROR_BAD_ARGUMENT;
      else
      {
            if(iChannel==RM_DIG_CHANNEL_A)
            {
                  BYTE_SET_BIT(iChannelStatus[0],iBitMask);
                  WRITE(3,iChannelStatus[0]);
            }
            else
            {
                  BYTE_SET_BIT(iChannelStatus[1],iBitMask);
                  WRITE(11,iChannelStatus[1]);
            }

            return RM_NO_ERROR;
      }
}
```

Y es esta función la que, finalmente, pone a 1 el bit especificado.

El procesado del otro mensaje es similar al del primero, solo cambia la función que esta al final de la cadena, la que realiza la tarea solicitada. En el primer mensaje era poner a 1 un bit determinado y con el segundo mensaje se devuelve el estado de las salidas digitales.

3.1.2 Descripción del protocolo

A continuación se describe el protocolo que sigue la comunicación entre el *Resource Manager* y la librería compartida (y, en general, con cualquier otro software que se quiera comunicar con el *Resource Manager*).

Antes, se muestra la estructura C, definida en el archivo `constants.h`, que usan el *Resource Manager* y la librería compartida, se muestra en el siguiente bloque de código.

```
/*****************************************************
 *    MENSAJE
 */

/*    Tamaño maximo de los datos del mensaje */
#define RM_MSG_MAX_LEN  16

/*
 *    Estructura que define la forma del mensaje
 */
typedef struct
{
      unsigned int      id;                   /**< ID del mensaje*/
      int               error;                /**< Codigo de error */
      unsigned char     data[RM_MSG_MAX_LEN];   /**< Cuerpo del mensaje*/
}pcm3718h_msg_t;
```

En la descricpción del protocolo, que sigue a continuación, donde dice "Enviar:" se refiere al envio del mensaje desde la librería hacia el *Resource Manager* y donde dice "Se devuelve:" se refiere a lo que devuelve el *Resource Manager* a la librería.

```
//============================
//    OPCIONES DE CONFIGURACION
//============================

//
//    Mostrar por consola las opciones de
//  la linea de comandos
//-------------------------------------
Enviar:
      - id  = RM_MSG_SHOWOPTIONS
      - error     = <nada>
      - data      : <nada>

Se devuelve:
      - id  = RM_MSG_REPLY
      - error     = codigo de error
      - data      : <nada>

//
//    Obtener las opciones de configuracion
//-------------------------------------
Enviar:
      - id  = RM_MSG_GETOPTIONS
      - error     = <nada>
      - data      : <nada>

Se devuelve:
      - id  = RM_MSG_REPLY
      - error     = codigo de error
      - data      : data[0] = LSB de dirBase
                    data[1] = MSB de dirBase
                    data[2] = dmaChannel
                    data[3] = timeBase
                    data[4] = extTrigger
                    data[5] = analogChannelMode


//============================
//    ESTADO / CONTROL de A/D
//============================

//-----------------------------------------
//    Registro de Estado
//-----------------------------------------

//
//    Leer el registro
//-------------------------------------
Enviar:
      - id  = RM_MSG_AD_READSTATUS
      - error     = <nada>
      - data      : <nada>

Se devuelve:
      - id  = RM_MSG_REPLY
      - error     = codigo de error
      - data      : data[0] = registro de estado

//
//    Escribir el registro -> ClearInterruptRequest
```

```
//---------------------------------------
Enviar:
      - id  = RM_MSG_AD_CLEARINTERRUPTREQUEST
      - error     = <nada>
      - data      : <nada>

Se devuelve:
      - id  = RM_MSG_REPLY
      - error     = codigo de error
      - data      : <nada>

//-----------------------------------------
//    Registro de Control
//-----------------------------------------

//
//    Leer el registro
//---------------------------------------
Enviar:
      - id  = RM_MSG_AD_READCONTROL
      - error     = <nada>
      - data      : <nada>

Se devuelve:
      - id  = RM_MSG_REPLY
      - error     = codigo de error
      - data      : data[0] = registro de control

//
//    Escribir el registro
//---------------------------------------
Enviar:
      - id  = RM_MSG_AD_WRITECONTROL
      - error     = <nada>
      - data      : data[0] = datos a escribir
                    data[1] = la mascara de bits sobre los datos a escribir

Se devuelve:
      - id  = RM_MSG_REPLY
      - error     = codigo de error
      - data      : <nada>



//============================
//    CONVERSION A/D
//============================

//
//    Leer datos del canal actual
//---------------------------------------
Enviar:
      - id  = RM_MSG_AD_READDATA
      - error     = <nada>
      - data      : <nada>

Se devuelve:
      - id  = RM_MSG_REPLY
      - error     = codigo de error
      - data      : data[0] = canal al que pertenecen los datos convertidos
```

```
                    data[1] = LSB del dato convertido
                    data[2] = MSB del dato convertido

//
//    Lanzar una conversion A/D (por software)
//-----------------------------------------
Enviar:
      - id  = RM_MSG_AD_TRIGGERCONV
      - error     = <nada>
      - data      : <nada>

Se devuelve:
      - id  = RM_MSG_REPLY
      - error     = codigo de error
      - data      : <nada>

//
//    Establece el rango de medida de un canal
//-----------------------------------------
Enviar:
      - id  = RM_MSG_AD_SETRANGE
      - error     = <nada>
      - data      : data[0] = canal a configurar
                    data[1] = rango a aplicar

Se devuelve:
      - id  = RM_MSG_REPLY
      - error     = codigo de error
      - data      : <nada>

//
//    Lee el registro de multiplexado
//-----------------------------------------
Enviar:
      - id  = RM_MSG_AD_READMUXSCAN
      - error     = <nada>
      - data      : <nada>

Se devuelve:
      - id  = RM_MSG_REPLY
      - error     = codigo de error
      - data      : data[0] = startChannel,
                              canal de comienzo de la conversion
                    data[1] = stopChannel,
                              canal de finalizacion de la conversion

//
//    Escribe en el registro de multiplexado
//-----------------------------------------
Enviar:
      - id  = RM_MSG_AD_WRITEMUXSCAN
      - error     = <nada>
      - data      : data[0] = startChannel,
                              canal de comienzo de la conversion
                    data[1] = stopChannel,
                              canal de finalizacion de la conversion

Se devuelve:
      - id  = RM_MSG_REPLY
      - error     = codigo de error
```

```
      - data      : <nada>


//============================
//    E/S DIGITAL
//============================

//-----------------------------------------
//    Lectura/Escritura de bits
//-----------------------------------------

//
//    Escribir a 1 una mascara de bits
//--------------------------------------
Enviar:
      - id  = RM_MSG_DIG_SETBIT
      - error     = <nada>
      - data      : data[0] = canal (DIGITAL_CHANNEL_A o DIGITAL_CHANNEL_B)
                    data[1] = la mascara de bits

Se devuelve:
      - id  = RM_MSG_REPLY
      - error     = codigo de error
      - data      : <nada>


//
//    Escribir a 0 una mascara de bits
//--------------------------------------
Enviar:
      - id  = RM_MSG_DIG_RESETBIT
      - error     = <nada>
      - data      : data[0] = canal (DIGITAL_CHANNEL_A o DIGITAL_CHANNEL_B)
                    data[1] = la mascara de bits

Se devuelve:
      - id  = RM_MSG_REPLY
      - error     = codigo de error
      - data      : <nada>

//
//    Leer el estado de los canales digitales
//-----------------------------------------------
Enviar:
      - id  = RM_MSG_DIG_READSTATUS
      - error     = <nada>
      - data      : data[0] = 0/1 si queremos leer o no el canal A
                    data[1] = 0/1 si queremos leer o no el canal B

Se devuelve:
      - id  = RM_MSG_REPLY
      - error     = codigo de error
      - data      : Si no ha habido error:
                         data[0] = estado del canal A
                         data[1] = estado del canal B

//
//    Leer el estado de los canales digitales
//-----------------------------------------------
Enviar:
```

```
      - id  = RM_MSG_DIG_WRITESTATUS
      - error     = <nada>
      - data      : data[0] = 0/1 si se escribe o no el canal A
                    data[1] = 0/1 si se escribe o no el canal B
                    data[2] = estado del canal A
                    data[3] = estado del canal B

Se devuelve:
      - id  = RM_MSG_REPLY
      - error     = codigo de error
      - data      : <nada>

//-----------------------------------------
//    Lectura/Escritura de entero
//-----------------------------------------

//
//    Leer los canales digitales como un entero
//--------------------------------------
Enviar:
      - id  = RM_MSG_DIG_READINT
      - error     = <nada>
      - data      : <nada>

Se devuelve:
      - id  = RM_MSG_REPLY
      - error     = codigo de error
      - data      : data[0] = LSB del entero leido
                    data[1] = MSB del entero leido
//
//    Escribir los canales digitales como un entero
//--------------------------------------
Enviar:
      - id  = RM_MSG_DIG_WRITEINT
      - error     = <nada>
      - data      : data[0] = LSB del entero a escribir
                    data[1] = MSB del entero a escribir

Se devuelve:
      - id  = RM_MSG_REPLY
      - error     = codigo de error
      - data      : <nada>


//============================
//    CONTADOR
//============================

//
//    Enable/Disable
//---------------------------------------------
Enviar:
      - id  = RM_MSG_PACER_ENABLEDISABLE
      - error     = <nada>
      - data      : data[0] = 0/1 para enable/disable el contador

Se devuelve:
      - id  = RM_MSG_REPLY
      - error     = codigo de error
      - data      : <nada>
```

```
//
//    Get Enable/Disable status
//--------------------------------------------
Enviar:
      - id  = RM_MSG_PACER_GETENABLED
      - error     = <nada>
      - data      : <nada>

Se devuelve:
      - id  = RM_MSG_REPLY
      - error     = codigo de error
      - data      : data[0] = 0/1 si esta enable/disable el contador


//
//    Lee el valor de un contador
//--------------------------------------------
Enviar:
      - id  = RM_MSG_PACER_READCOUNTER
      - error     = <nada>
      - data      : data[0] = counterID, contador cuyo valor se quiere leer

Se devuelve:
      - id  = RM_MSG_REPLY
      - error     = codigo de error
      - data      : data[0] = valor LSB del contador seleccionado
                    data[1] = valor MSB del contador seleccionado

      (nota: los valores LSB y MSB tendran sentido dependiendo de como este
      comfigurado el modo R/W del pacer. si esta configurado como MSB, el
      dato ira en su lugar, en el data[1])

//
//    Escribe el valor de un contador
//--------------------------------------------
Enviar:
      - id  = RM_MSG_PACER_WRITECOUNTER
      - error     = <nada>
      - data      : data[0] = counterID, contador cuyo valor se
                              quiere escribir
                    data[1] = valor LSB del dato a escribir
                    data[2] = valor MSB del dato a escribir

Se devuelve:
      - id  = RM_MSG_REPLY
      - error     = codigo de error
      - data      : <nada>

      (nota: los valores LSB y MSB tendran sentido dependiendo de como este
      comfigurado el modo R/W del pacer. si esta configurado como MSB, el
      dato debe ir en su lugar, en el data[2])

//
//    Configura un contador
//--------------------------------------------
Enviar:
      - id  = RM_MSG_PACER_CONFIGURECOUNTER
      - error     = <nada>
      - data      : data[0] = counterID, contador cuyo valor se
```

```
                              quiere escribir
                    data[1] = modo de R/W
                    data[2] = modo de operacion
                    data[3] = contaje BCD / BINARIO

Se devuelve:
      - id  = RM_MSG_REPLY
      - error     = codigo de error
      - data      : <nada>

//
//    Lee el estado de un contador
//  ("readback command")
//-----------------------------------------
Enviar:
      - id  = RM_MSG_PACER_READCOUNTERSTATUS
      - error     = <nada>
      - data      : data[0] = counterID, contador cuyo valor se
                              quiere escribir

Se devuelve:
      - id  = RM_MSG_REPLY
      - error     = codigo de error
      - data      : data[0] = estado del contador seleccionado

//
//    Lee el registro de control del Pacer
//-----------------------------------------
Enviar:
      - id  = RM_MSG_PACER_READCONTROL
      - error     = <nada>
      - data      : <nada>

Se devuelve:
      - id  = RM_MSG_REPLY
      - error     = codigo de error
      - data      : data[0] = registro de control

//    Escribe el registro de control del Pacer
//-----------------------------------------
Enviar:
      - id  = RM_MSG_PACER_WRITECONTROL
      - error     = <nada>
      - data      : data[0] = registro de control

Se devuelve:
      - id  = RM_MSG_REPLY
      - error     = codigo de error
      - data      : <nada>
```

3.1.3 Opciones en línea de comandos

El *Resource Manager* puede leer directamente de la tarjeta ciertas opciones de configuración, mientras que otras deben ser especificadas por el usuario cuando lo lanza en línea de comandos. Para conocer las opciones disponibles así como los parámetros necesarios para cada una, se puede escribir en la línea de comandos lo siguiente "rm_pcm3718h –h", que nos presentará la siguiente lista de opciones:

```
Uso:
  rm_pcm3718h [opt]

donde [opt] puede ser:
  -h                  = help, muestra esta pantalla
  -v                  = verbose, muestra informacion detallada durante
                        la ejecucion
  -b:<direccion hex>  = dirBase, establece la direccion base de la PCM3718H
                        (entre 0x000 y 0x3F0) (*)
  -d:<1 o 3>          = dmaChannel, establece el canal DMA que para la
                         transferencia de datos de la conversion A/D
  -t:<1 o 10>         = timeBase, establece la base de tiempo del contador
                        interno (1 : 1MHz , 10 :10MHz)
  -e                  = extTrigger, para indicar que DI0 sera trigger
                        externo y no entrada/salida digital

(*) la direccion se debe dar en hexadecimal (p.e. si es 0x1A2, la opcion de
linea de comandos seria '-b 1A2')
```

3.1.4 Captura por IRQ

Para capturar conversiones A/D mediante interrupciones se sigue el siguiente método: el *Resource Manager* instala un manejador de interrupción (*ISR*) asociada a la interrupción elegida por el usuario. Este manejador se encarga de recuperar de la tarjeta el dato capturado. Luego lanza un hilo en paralelo que espera a que se dispare la interrupción, lee el dato capturado y encola una señal SIGRTMIN, que contiene el dato capturado junto con el canal A/D del que procede, al proceso de usuario identificado por su PID.

Todos los pasos anteriormente citados se llevan a cabo mediante los siguientes fragmentos de código del *Resource Manager*:

- Instalación del manejador de interrupción / lanzamiento del hilo paralelo

```
int isrInstall(int iIrq)
{
      irqID=iIrq;

      //    lanzamos el thread
      if(pthread_create(&isrThreadID,NULL,isrSignalThread,NULL)!=EOK)
            return -1;
      else
            return 1;
}
```

- Manejador de la interrupcion

```
const struct sigevent * isrIRQHandler(void * pvArea,int iId)
{
      int base0,base1;  // que se leeran resp de BASE+0 y BASE+1

      //    leemos del hardware
      base0 = READ(0);
      base1 = READ(1);

      //    construimos la informacion
      isrCanal=base0 & 0x0F;
```

```
	isrData=((int)base1)<< 4 | ((int)base0)>>4;

	//	bloqueamos la irq (para que el PIC no vuelva a
	//	interrumpir inmediatamente al kernel)
//	if(ThreadCtl(_NTO_TCTL_IO,0)==-1)
//		return NULL;
	if(InterruptMask(irqID,irqFuncID)==-1)
		return NULL;

	//	como aqui no se puede llamar a 'sigqueue' (que se llama
	//	desde 'signalToUser')
	//	, ponemos un flag a 1 para que un hilo en paralelo sea el
	//	que envie la seÃ±al
	isrFlagSignal=1;

	return (&event);
}
```

- Hilo paralelo

```
void * isrSignalThread(void *pvParams)
{
	//	parametrizamos el evento que devolvera el ISR
	SIGEV_INTR_INIT(&event);

	//	instalamos el ISR
	if(ThreadCtl(_NTO_TCTL_IO,0)==-1)
		return NULL;
	if((irqFuncID=InterruptAttach (irqID,isrIRQHandler,NULL,0,0))
		==-1)
		return NULL;

	//	bucle principal de captura
	while(isrFlagToEnd==0)
	{
		//	esperamos a que se dispare una interrupcion
		if(InterruptWait(0,NULL)==-1)
			return NULL;

		//	mandamos los datos leidos usando una señal RT
		if(signalToUser(isrCanal,isrData)==1)
		// OK
		else
		{
		// FALLO
			break;
		}

		//	limpiamos el bit INT de la tarjeta
		board_ClearInterruptRequest(NULL);

		//	desbloqueamos la irq
		if(ThreadCtl(_NTO_TCTL_IO,0)==-1)
			return NULL;
		if(InterruptUnmask(irqID,irqFuncID)==-1)
			return NULL;
	}

	//	desinstalamos el ISR
	if(ThreadCtl(_NTO_TCTL_IO,0)==-1)
```

```
            return NULL;
      if(InterruptDetach(irqID)==-1)
            return NULL;

      return NULL;
}
```

- Lanzamiento del mensaje al proceso de usuario

```
int signalToUser(int iChannel, int iData)
{
      union sigval valor;
      int res;

      if(iUserProcessPID!=-1)
      {
            valor.sival_int= ((channel & 0xF) << 12) |
                  (data & 0x0FFF);
            res=sigqueue(iUserProcessPID,SIGRTMIN,valor);
            if(res==-1)
                  if(errno==EAGAIN)
                  {
                        //printf(" (EAGAIN) ");
                        return 1; // para que lo reintente luego
                  }
                  else
                        return 0;
            else
                  return 1;
      }

      return 0;
}
```

3.2 Librería compartida.

El procesado de mensajes se ha visto con detalle en el apartado anterior, dedicado al *Resource Manager*, por lo que en esta sección solo se describirá la estructura y funcionalidad de la librería compartida.

En primer lugar, las contantes, macro y funciones relacionadas con la librería compartida se encuentran en el archivo `devio-pcm3718h.h`.

Las constantes y la macro están relacionadas con los errores y su gestión. Dichos errores se generan en la propia librería o en el *Resource Manager*, siendo entonces la librería un mero puente entre éste y el usuario de la misma. Para distinguirse de otros errores, los de la librería comienzan por "`SO_`" (mientras que los del *Resource Manager* comienzan por "`RM_`") y se listan a continuación.

```
/*
 *    Lista de errores de la libreria
 */
#define SO_NO_ERROR                              ((int)1000)
      /**> No ha ocurrido ningun error                        */

#define SO_ERROR_OPN_DEV                         ((int)1001)
      /**> Error abriendo el dispositivo en /dev/pcm3718h     */
```

```
#define SO_ERROR_MSG_SEND                       ((int)1002)
      /**> Error enviando mensaje al Resource Manager         */

#define SO_ERROR_BAD_ARGUMENT                   ((int)1003)
      /**> Error en el argumento de funcion                   */

#define SO_ERROR_EXECUTING_COMMAND              ((int)1004)
      /**> Error ejecutando el comando en el Resource Manager */

#define SO_ERROR_SYST   EM                      ((int)1005)
      /**> Error de sistema                                   */

#define SO_ERROR_TIME_EXCEEDED                  ((int)1006)
      /**> Error, tiempo excedido en la espera                */

#define SO_ERROR_CONFIGURATION                  ((int)1007)
      /**> Error de configuracion del dispositivo             */
```

La macro a la que se hacía mención en el párrafo anterior sirve para difrenciar un error devuelto por cualquier función de la librería entre un error de la propia librería y uno generado en el Resource Manager.

```
#define IS_SO_ERROR(e) ((e)>=SO_NO_ERROR)
```

Hay tres funciones que estan relacionadas con la gestión de errores y son:

1. `const char * errorToString(int err);`
2. `const char * RMErrorToString(int err);`
3. `const char * SOErrorToString(int err);`

La primera devuelve un texto describiendo el error pasado en el parámetro `err`. Las otras dos hacen lo mismo pero sólo con los errores procedentes del *Resource Manager* la primera y de la librería la segunda.

El resto de funciones sirven para comunicar con el *Resource Manager*, implementando el protocolo descrito anteriormente. Estan divididas en cinco bloques:

- E/S digital

```
int pcm3718h_SetDigBit(int iChannel,int iBitMask);
int pcm3718h_ResetDigBit(int iChannel,int iBitMask);
int pcm3718h_ReadDigStatus(int *piChannelA,int *piChannelB);
int pcm3718h_WriteDigStatus(int iChannelA, int iChannelB);
int pcm3718h_ReadDigInt(int *piDigI);
int pcm3718h_WriteDigInt(int iDigO);
```

- Conversión A/D

```
int pcm3718h_ReadADData(int *piChannel, int *piData);
int pcm3718h_TriggerConversion();
int pcm3718h_SetADRange(int iChannel, int iRange);
int pcm3718h_ReadADMuxScan(int *piStartChannel, int *piStopChannel);
int pcm3718h_WriteADMuxScan(int iStartChannel, int iStopChannel);
int pcm3718h_WaitToEndOfConversion(unsigned long ulMs);
int pcm3718h_SetUserProcessPID(int iPid);
int pcm3718h_ClearUserProcessPID();
```

- Estado y Control

```
int pcm3718h_ReadADStatus(int *piADStatusRegister);
int pcm3718h_ClearInterruptRequest(void);
int pcm3718h_ReadADControl(int *piADControlRegister);
int pcm3718h_WriteADControl(int iControlBits,int iMask);
int pcm3718h_GetADTransferMode(int *piMode, int *piIRQ);
int pcm3718h_SetADTransferMode(int iMode, int iIRQ);
int pcm3718h_GetADTriggerSource(int *piTrgSrc);
int pcm3718h_SetADTriggerSource(int iTrgSrc);
```

- Contador

```
int pcm3718h_CalcPacerRate(unsigned int uiC1, unsigned int uiC2,float
                           *pfRate);
int pcm3718h_EnablePacer(void);
int pcm3718h_DisablePacer(void);
int pcm3718h_GetPacerEnabled(int *piPacerEnabled);
int pcm3718h_ReadCounter(int iCounterID,int *piCount);
int pcm3718h_WriteCounter(int iCounterID, int iCount);
int pcm3718h_ConfigureCounter(int iCounterID,int iRWMode, int
                              iOpMode, int iBCDMode);
int pcm3718h_ReadCounterStatus(int iCounterID, int *piStatus);
int pcm3718h_ReadPacerControl(int *piControlRegister);
int pcm3718h_WritePacerControl(int iControlRegister);
```

- Configuración

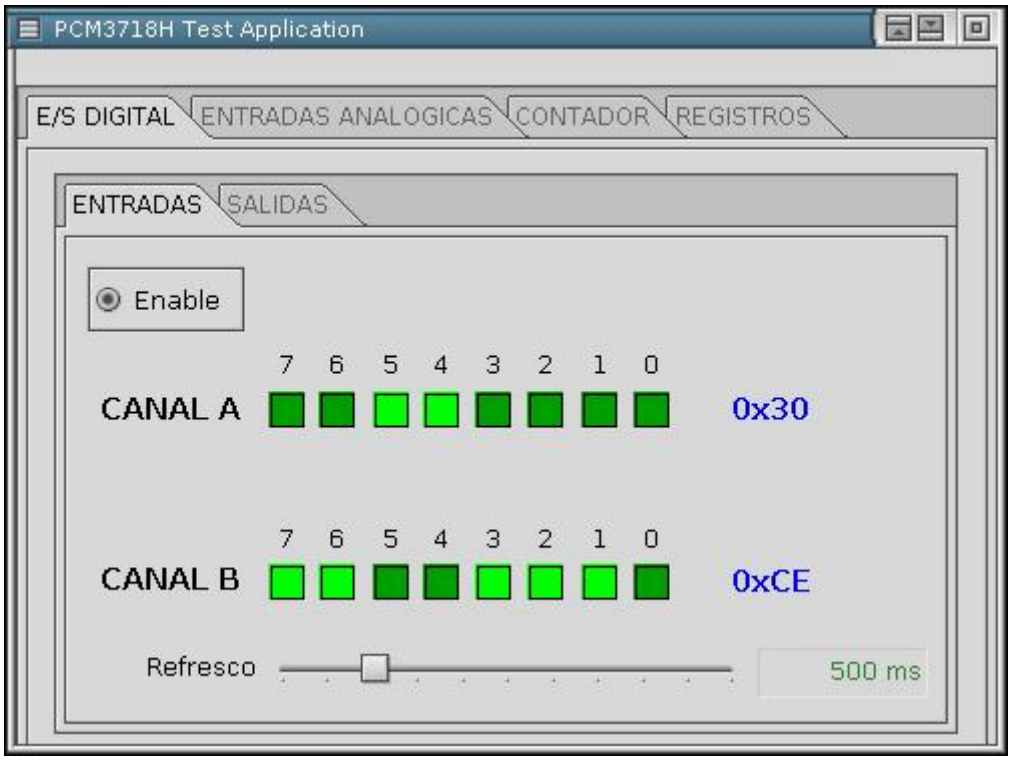
```
int pcm3718h_ShowOptions(void);
int pcm3718h_GetDirBase(int * piDirBase);
int pcm3718h_GetDMAChannel(int * piDMAChannel);
int pcm3718h_GetTimeBase(int * piTimeBase);
int pcm3718h_GetExtTrigger(int * piExtTrigger);
int pcm3718h_GetAnalogChannelMode(int * piAnalogChannelMode);
```

Todas estas funciones están ampliamente documentadas en el propio código fuente, que se ha usado con la herramienta Doxygen para generar, en formato HTML, una ayuda para el usuario de la librería.

3.3 Aplicación GUI.

La aplicación gráfica de usuario, que se lanza colocándonos en el directorio que proceda y escribiendo en la línea de comandos `Test` (que es el nombre del archivo ejecutable), esta compuesto de una ventana principal, con varias pestañas que contienen la funcionalidad básica de la tarjeta, y algunas ventanas secundarias que permiten cambiar los parámetros de operación de los distintos elementos de la tarjeta.

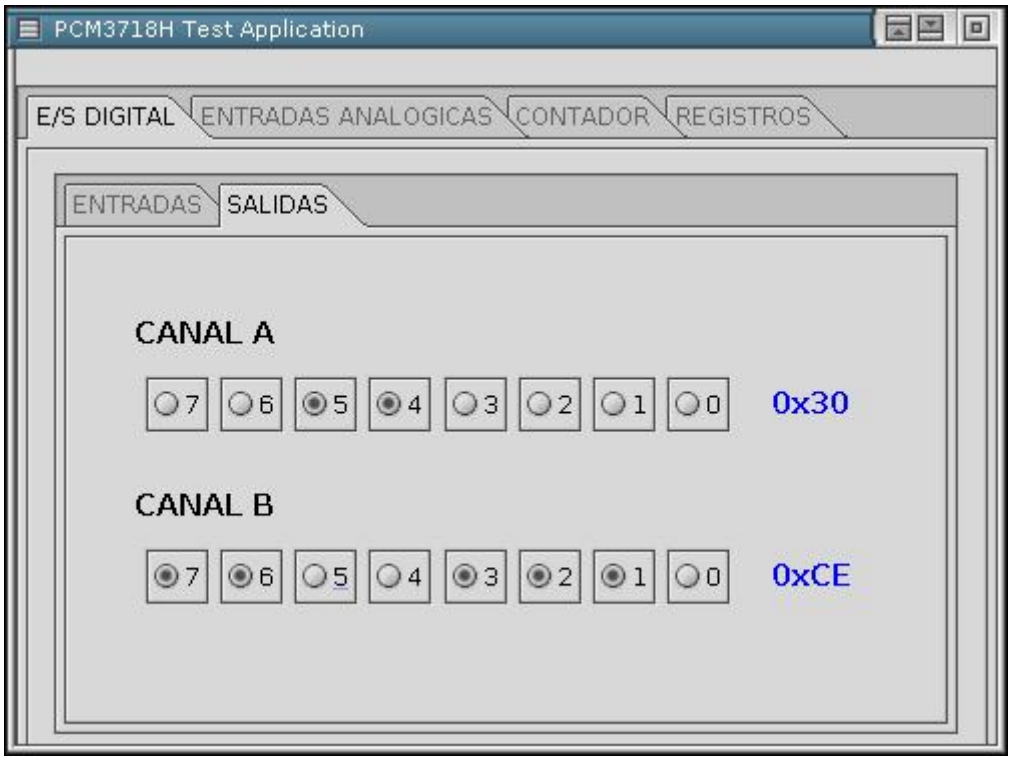
La ventana principal, con la pestaña "E/S Digital" en primer plano, se puede observar en la siguiente figura:



Puede observarse que, dentro de la pestaña en cuestión, hay otras dos pestañas: una para las entradas digitales y otra para las salidas digitales. En la de las entradas digitales, un botón etiquetado "Enable" permite habilitar o deshabilitar el hilo que, en segundo plano, lee el contenido de las entradas digitales y refresca en consecuencia los elementos gráficos asociados a dichas entradas. Con el control desplazable situado en la parte inferior de la ventana seleccionamos la tasa de refresco deseada para las entradas digitales, es decir, cada cuanto tiempo debe el hilo en segundo plano leer las entradas digitales.

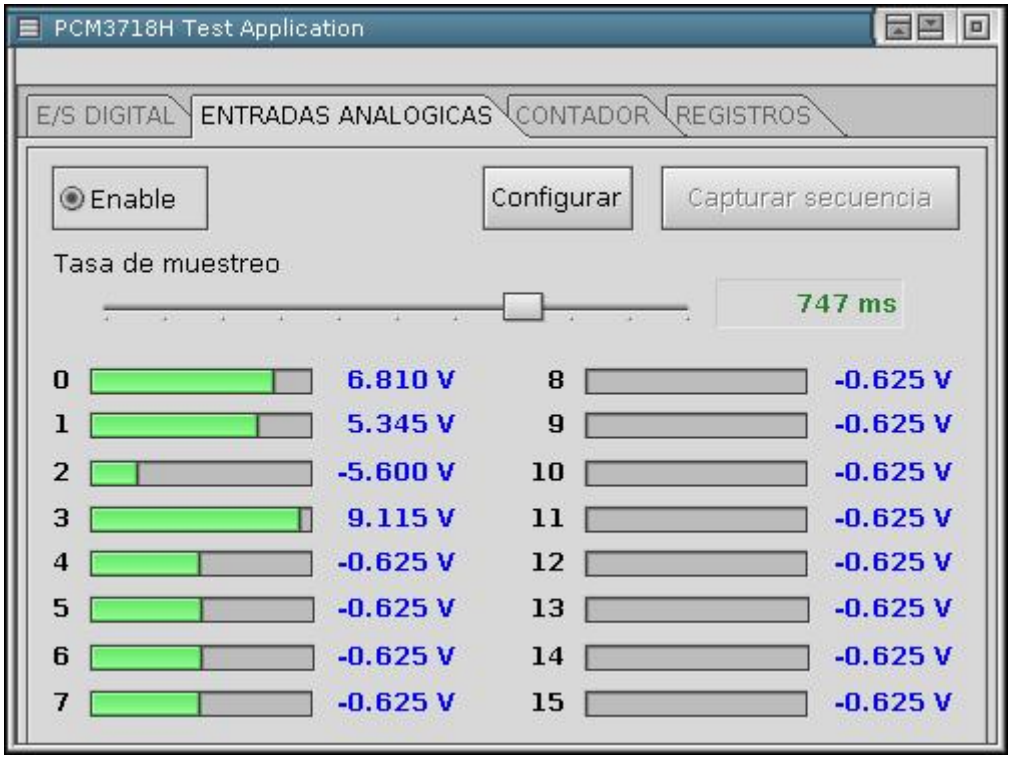
A la izquierda de cada fila correspondiente a cada canal, se muestra el contenido de dicho canal expresado como un número en notación hexadecimal.

El contenido de la pestaña correspondiente a las salidas digitales se puede observar en la siguiente figura:



Para activar/desactivar las salidas digitales se dispone de botones con estado que, numerados de 0 a 7, representan los bits de cada canal digital. A la izquierda de cada fila de botones se muestra el contenido de cada canal representado como un número en notación hexadecimal.

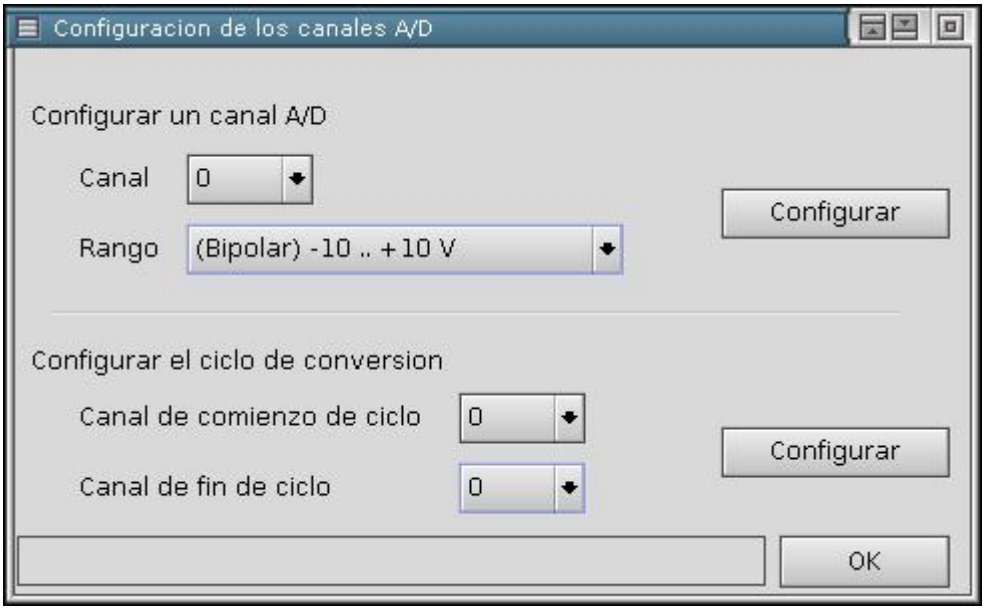
En la siguiente figura puede verse el contenido de la pestaña asociada a las entradas analógicas:



Al igual que sucede con las entradas digitales, en este caso también un hilo en segundo plano es el

encargado de leer, con la periodicidad especificada mediante el control deslizante, las entradas analógicas. Aunque se muestran los 15 canales disponibles como máximo, los que sean actualizados dependerá de la configuración de la tarjeta y del ciclo de conversión.

Pulsando en el botón titulado "Configurar" accedemos a la siguiente ventana secundaria:



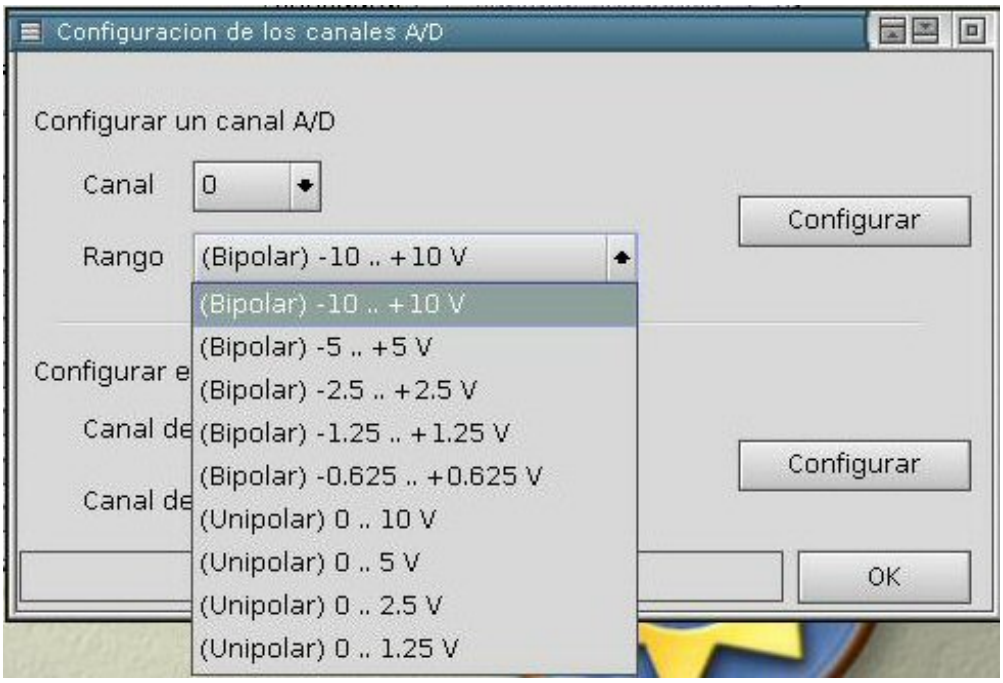
Dicha ventana se encuentra dividida verticalmente en dos secciones:

- Configuración de un canal A/D
- Configuración del ciclo de conversión analógica

Para configurar un canal se deben seguir tres pasos:

1. Seleccionar el canal que se pretende configurar, como se indica en la siguiente figura:

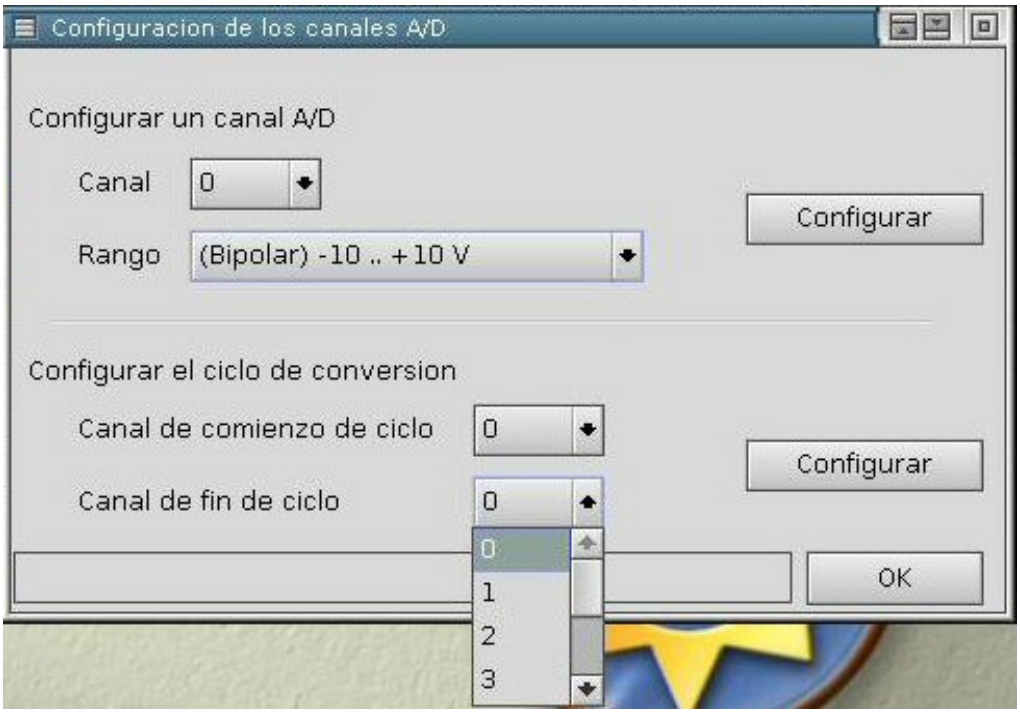
2. Seleccionar el rango al que se desea configurar el canal seleccionado previamente:



3. Pulsar el botón "Configurar" que esta en la parte superior. Si todo va bien, un texto indicando que el canal ha sido configurado correctamente aparecerá en la barra de estado (situada en la parte inferior de la ventana, a la izquierda del botón "Ok")

Para configurar el ciclo de conversión analógica se deben seguir tres pasos:

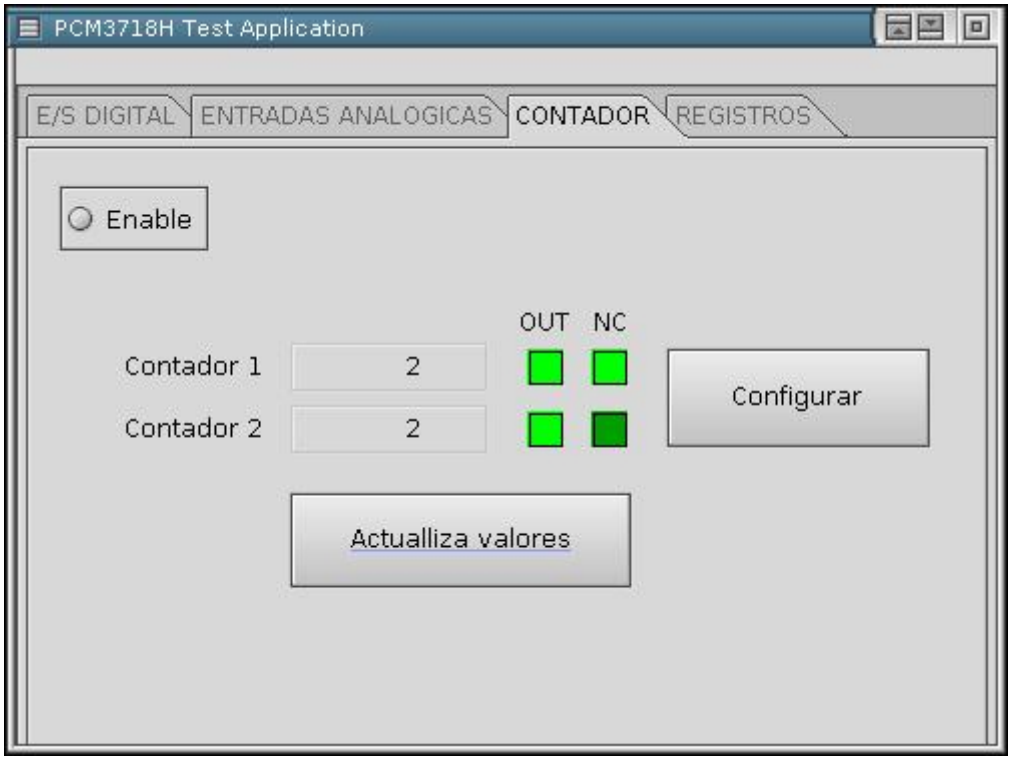
1. Seleccionar el canal de comienzo del ciclo
2. Seleccionar el canal de fin de ciclo



3. Pulsar el botón "Configurar" que esta en la parte inferior. Si todo va bien, un texto

indicando que el ciclo ha sido configurado correctamente aparecerá en la barra de estado (situada en la parte inferior de la ventana, a la izquierda del botón "Ok")

La siguiente pestaña en la ventana principal es la correspondiente al manejo de los contadores:



El primer elemento que aparece es un botón "Enable", que permite activar/desactivar los contadores (no hay ningún hilo en paralelo, como en el caso de las entradas digitales, por ejemplo, si no que la activación/desactivación tiene lugar en la tarjeta de adquisición de datos).

Luego pueden verse los últimos valores leídos de los contadores 1 y 2, valores que pueden ser actualizados pulsando en el botón "Actualiza valores". Junto a estos valores, puede verse el estado de dos bits asociados a los contadores: el bit "OUT" (que indica el valor actual de la salida del contador) y el bit "NC" ("Null Count", que indica, cuando esta a 1, que el último valor de contaje escrito en el contador ha sido efectivamente cargado en el elemento de contaje)[8].

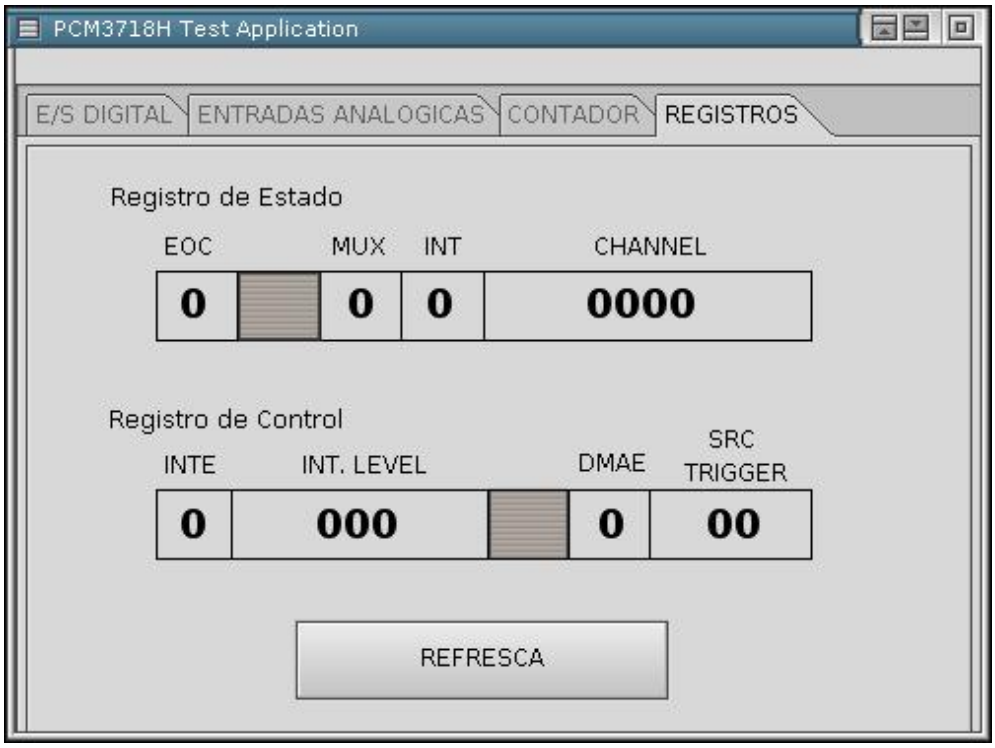
Ambos contadores son configurables, accediéndose a la ventana de configuración pulsando el botón "Configurar".

[8] Para más información diríjase al manual de usuario de la tarjeta

En esta ventana, además del modo de operación del contador (recuadros oscuros), podemos seleccionar el valor que queremos establecer para cada uno de los contadores. Dichos valores modifican la frecuencia de salida del *Pacer*, que también depende de la frecuencia base seleccionada en la tarjeta. Para poder ver la frecuencia efectiva de salida del *Pacer*, basta pulsar el botón "Recalcula", que calcula el valor de dicha frecuencia de salida y lo muestra. Esto puede resultar de ayuda a la hora de seleccionar los valores de los contadores para una frecuencia de salida deseada.

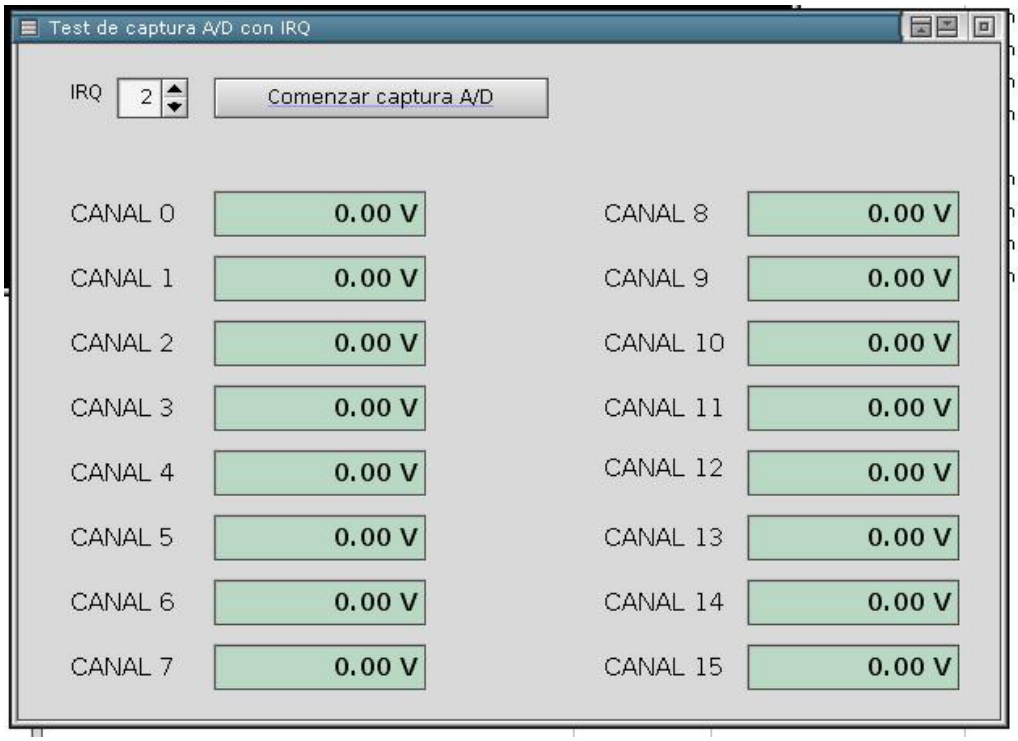
La última pestaña es la correspondiente a los registros de estado y de control de la tarjeta. En ella se muestran los bits más importantes de ambos registros, debajo de los nombres que les da el manual de usuario de la tarjeta (para más información sobre el significado de cada bit, diríjase al manual de usuario de la tarjeta).

El botón "Refresca" hace una llamada a la tarjeta para recuperar los valores actuales de ambos registros y luego muestra en pantalla el estado de los bits de cada uno.



3.4 Aplicación GUI IRQ.

La aplicación GUI para probar la captura A/D por IRQ, llamada `TestIRQ`, se compone de una única ventana en que se muestran los valores capturados. Para iniciar la captura se debe seleccionar la IRQ a usar y pulsar el botón etiquetado como "Comenzar captura A/D".



Al pulsar el botón "Comenzar captura A/D" se ejecuta el código siguiente:

```
int
btnStart_OnClick( PtWidget_t *widget, ApInfo_t *apinfo, PtCallbackInfo_t
*cbinfo )

      {
      int res;

      /* eliminate 'unreferenced' warnings */
      widget = widget, apinfo = apinfo, cbinfo = cbinfo;

      //    instalamos el manejador de señal
      printf("<btnStart> installSignalHandler\n");
      if(!installSignalHandler())
      {
            printf("-- Error instalando el manejador de señal\n");
            return( Pt_CONTINUE );
      }

      //    Para hacerlo "mejor", habria que leer todos los parametros que
      //    se van a cambiar y volver a dejarlo
      //    todo como estaba si se produce algun error.

      //    configuramos el pacer a 4Hz, para tener 4 muestras por segundo
      //    (suponemos que FCLK = 1Mhz)
      //    para ello, escribimos en los contadores: c1=250 y c2=1000  -->
      //    f = FCLK / ( c1 * c2) = 4 Hz

      //    primero, lo paramos
      printf("<btnStart> pcm3718h_DisablePacer\n");
      if((res=pcm3718h_DisablePacer())!=SO_NO_ERROR)
      {
            printf("-- Error disabling pacer
                  (OnStart)[%s]\n",errorToString(res));
            return( Pt_CONTINUE );
      }

      //    configuramos el contador 1
      printf("<btnStart> pcm3718h_WriteCounter 1\n");
      if((res=pcm3718h_WriteCounter(RM_PACER_SC_COUNTER1,250))!=
            SO_NO_ERROR)
      {
            printf("-- Error configurando contador 1
                  [%s]\n",errorToString(res));
            return( Pt_CONTINUE );
      }

      //    configuramos el contador 2
      printf("<btnStart> pcm3718h_WriteCounter 1\n");
      if((res=pcm3718h_WriteCounter(RM_PACER_SC_COUNTER2,1000))!=
            SO_NO_ERROR)
      {
            printf("-- Error configurando contador 2
                  [%s]\n",errorToString(res));
            return( Pt_CONTINUE );
      }

      //    especificamos que el ciclo de conversion comprenda los 16
      //    canales
```

```
	printf("<btnStart> pcm3718h_WriteADMuxScan\n");
	if((res=pcm3718h_WriteADMuxScan(RM_AD_CHANNEL0,RM_AD_CHANNEL15))!=
		SO_NO_ERROR)
	{
		printf("-- Error estableciendo el ciclo de conversion
			[%s]\n",errorToString(res));
		return( Pt_CONTINUE );
	}

	//	especificamos que el disparo de la conversion A/D lo realice
	//	el pacer
	printf("<btnStart> pcm3718h_SetADTriggerSource\n");
	if((res=pcm3718h_SetADTriggerSource(RM_AD_CONTROL_TRIGSRC_PACER))!=
		SO_NO_ERROR)
	{
		printf("-- Error estableciendo el trigger source por pacer
			[%s]\n",errorToString(res));
		return( Pt_CONTINUE );
	}

	//	al final, habilitamos el pacer
	printf("<btnStart> pcm3718h_EnablePacer\n");
	if((res=pcm3718h_EnablePacer())!=SO_NO_ERROR)
	{
		printf("-- Error enabling pacer [%s]\n",errorToString(res));
		return( Pt_CONTINUE );
	}

	printf("<btnStart> OK\n");

	return( Pt_CONTINUE );

	}
```

Paso a paso, el código realiza las siguientes acciones (líneas resaltadas en negrita):

- Instala el manejador de señal.
- Deshabilita el *pacer*.
- Configura el contador 1 con un preset de 250
- Configura el contador 2 con un preset de 1000
- Configura el ciclo de conversión A/D para capturar los 16 canales
- Especifica que el disparo de la conversión A/D lo lance el *pacer*.
- Habilita el *pacer*.

4 Código fuente

A continuación se muestra el código fuente de los archivos más importantes del Proyecto.

4.1 Resource Manager.

4.1.1 rm_pcm3718h.c

```
/**
 *    Codigo fuente para /dev/pcm3718h Resource Manager
 *
 *    @file rm_pcm3718h.c
 *    @author Raul Jimenez Ruiz
 */

#include <errno.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/iofunc.h>
#include <sys/dispatch.h>
#include <sys/neutrino.h>
#include <sys/resmgr.h>
#include <sys/iomsg.h>
#include <sys/mman.h>

#include <signal.h>

/*    Incluimos el archivo de constantes y estructuras
 */
#include "constants.h"

/*    Incluimos el archivo de acceso hardware a la tarjeta
 */
#include "board.h"

/*    Guardamos en una variable la ruta de acceso al dispositivo
 */
const char * strDevName=RM_DEV_NAME;

//    Opciones disponibles en la linea de comandos
//    ( ver parseOptions() )
const char * strOptions="vhb:t:d:e";

//    PID del proceso de usuario al que mandar los datos A/D
//    cuando la conversion A/D es por interrupciones
int iUserProcessPID=-1;

/*************************/
/*  opciones por defecto */
/*************************/
int printHelpScreen=0;  /* -h  -> printHelpScreen=1*/
int verbose=0;          /* -v  -> verbose=1 */
int dirBase=0x300;      /* -b: -> 0x000 .. 0x3F0 */
```

```
int dmaChannel=1;       /* -d: -> 1 o 3 */
int timeBase=1;         /* -t: -> 1 MHz o 10 MHz */
int extTrigger=0;       /* -e  -> exTrigger=1 ; significado-> 0: DI0 es
                              entrada digital, 1: DI0 es trigger externo */

uintptr_t dirBasePtr;   /* puntero devuelto por mmap_device_io y usado en
                        in8, out8 */

/**
 *    Array de punteros a funciones que manejan los mensajes.
 *    Dichas funciones tienen la signatura siguiente:
 *    <tt>int fname(unsigned char *params)</tt>
 */
msg_hnd_t hndArray[RM_MSG_LAST+1];

/*
 *    Funciones connect e I/O
 */

resmgr_connect_funcs_t  connect_funcs;
resmgr_io_funcs_t       io_funcs;

/*
 *    Variables de dispatch, atributos y contexto del resource manager
 *    y
 *    estructura de atrributos de las iofunc
 */

dispatch_t              *dpp;
resmgr_attr_t           rattr;
dispatch_context_t      *ctp;
iofunc_attr_t           ioattr;
message_attr_t          msgAttr;

/** Condicion de terminacion del bucle principal */
int iFin=0;

//    Guarda el modo R/W (para no tener que leerlo cada vez de la tarjeta)
int iRWMode=RM_PACER_RW_LSBMSB;

/**
 *    Gestiona la linea de comandos.
 *    La opcion -h muestra informacion detallada de las opciones
 *    disponibles
 *
 *    @param argc Se corresponde con el parametro 'argc' de main
 *    @param argv Se corresponde con el parametro 'argv' de main
 *
 *    @return Devuelve si ha habido algun error al analizar la linea de
 *          comandos. Los codigos de error en constants.h
 *
 *    @see constants.h
 */
int parseOptions (int argc, char **argv)
{
      int opt;
      long aux;

      while ((opt = getopt (argc, argv, strOptions)) != -1)
      {
```

```
            switch (opt)
            {
                  case 'h':
                        printHelpScreen=1;
                        return RM_NO_ERROR;

                  case 'v':
                        verbose=1;
                        break;

                  case 'b':
                        aux=strtol(optarg,0,16);
                        if(aux>=0 && aux<=0x3F0)
                              dirBase=aux;
                        else
                        {
                              printf("\n [error] El parametro 'direccion
                               base' debe estar comprendido entre 0x000 y
                               0x3F0\n");
                              return RM_ERROR_CMDLINE_PARSING;
                        }
                        break;

                  case 'd':
                        aux=atoi(optarg);
                        if(aux==1 || aux==3)
                              dmaChannel=aux;
                        else
                        {
                              printf("\n[error] El parametro 'canal dma'
                               debe ser 1 o 3\n");
                              return RM_ERROR_CMDLINE_PARSING;
                        }
                        break;

                  case 't':
                        aux=atoi(optarg);
                        if(aux==1 || aux==10)
                              timeBase=aux;
                        else
                        {
                              printf("\n[error] El parametro 'base temporal'
                               debe ser 1 (1MHz) o 10 (10MHz)\n");
                              return RM_ERROR_CMDLINE_PARSING;
                        }
                        break;

                  case 'e':
                        extTrigger=1;
                        break;

                  case '?': // error
                        return RM_ERROR_CMDLINE_PARSING;
            }
      }

      return RM_NO_ERROR;
}

/**
```

```
 *    Comprueba si el usuario ha solicitado ayuda
 *
 *    @return Devuelve 1 si se ha seleccionado la opcion '-h'
 *    @see parseOptions
 */
 int userWantsHelp(void)
 {
      return (printHelpScreen==1);
 }

/**
 *    Comprueba si se ha selecionado el modo en que se muestra informacion
 *    detallada durante la ejecucion
 *
 *    @return Devuelve 1 si se ha seleccionado la opcion '-v'
 *    @see parseOptions
 */
int isVerbose(void)
{
      return (verbose==1);
}

/**
 *    Obtiene la direccion base del mapa de memoria en la que se encuentra
 *    direccionado dispositivo
 *
 *    @return Devuelve la direccion base seleccionada por el usuario (o su
 *    valor por defecto)
 *    @see parseOptions
 */
int getDirBase(void)
{
      return dirBase;
}

/**
 *    Obtiene el canal DMA seleccionado
 *
 *    @return Devuelve el canal DMA seleccionado por el usuario (o su valor
 *    por defecto)
 *    @see parseOptions
 */
int getDMAChannel(void)
{
      return dmaChannel;
}

/**
 *    Obtiene la base temporal para la captura A/D y las temporizaciones
 *
 *    @return Devuelve la base temporal seleccionada por el usuario (o su
 *    valor por defecto)
 *    @see parseOptions
 */
int getTimeBase(void)
{
      return timeBase;
}

/**
```

```
 *    Obtiene el modo del bit 0 del canal digital A
 *
 *    @return Devuelve el modo de DI0/Ext trigger
 *    @see parseOptions
 */
int getExtTrigger(void)
{
      return extTrigger;
}

/**
 *    Comprueba si el bit 0 del canal digital A se usa como trigger externo
 *    o no
 *
 *    @return Devuelve 1 si se ha seleccionado DI0 como trigger externo
 *    @see parseOptions
 */
int isExtTrigger(void)
{
      return (extTrigger==1);
}

/**
 *    Obtiene el modo de los canales analogicos
 *
 *    @return Devuelve el modo de los canales analogicos (0 : 8 Diff ; 1 :
 *    16 S.E.)
 */
int getAnalogChannelMode(void)
{
      unsigned char status[1];

      board_GetADStatusRegister(status);

      return (((int)status[0]) & RM_AD_STATUS_MUX_BIT)!=0;
}

/**
 *    Comprueba si los canales analogicos estan en modo '8 differential'
 *
 *    @return Devuelve 1 si los canales analogicos estan en modo '8
 *    differential'
 */
int isAnalogChannel8diff(void)
{
      return (getAnalogChannelMode()==0);
}

/**
 *    Comprueba si los canales analogicos estan en modo '16 single-ended'
 *
 *    @return Devuelve 1 si los canales analogicos estan en modo '16
 *    single-ended'
 */
int isAnalogChannel16se(void)
{
      return (getAnalogChannelMode()==1);
}

/**
```

```
 *    Muestra el estado actual de las opciones seleccionadas
 *    (tiene esa signatura porque es respuesta a un mensaje)
 *
 *    @param ucParams No se usa
 *    @return Devuelve siempre RM_NO_ERROR
 */
int showOptions(unsigned char *ucParams)
{

      if(isVerbose())
            printf("%s: > en showOptions\n",RM_DEV_NAME);

      printf("\n");
      printf("- verbose           = %d\n",isVerbose());
      printf("- dirBase           = %#x\n",getDirBase());
      printf("- dmaChannel        = %d\n",getDMAChannel());
      printf("- timeBase          = %d\n",getTimeBase());
      printf("- extTrigger        = %d\n",getExtTrigger());
      printf("- analogChannelMode = %d\n",getAnalogChannelMode());
      printf("\n");

      return RM_NO_ERROR;
}

/**
 *    Establece el PID del proceso cliente al que mandar, usando
 *    SIGRTMIN, el [canal,valor] de la captura A/D
 *
 *    @param ucParams Es un array de unsigned char's con el
 *     siguiente significado:
 *          - ucParams[0] = LSB del PID
 *          - ucParams[1] = ...
 *          - ucParams[2] = ...
 *          - ucParams[3] = MSB del PID
 *    @return Devuelve el codigo de error
 */
int setUserProcessPID(unsigned char *ucParams)
{
      unsigned int pid;

      if(ucParams==NULL)
            return RM_ERROR_BAD_ARGUMENT;

      pid  = (unsigned int)(ucParams[0]);
      pid |= (unsigned int)(ucParams[1])<< 8;
      pid |= (unsigned int)(ucParams[2])<< 16;
      pid |= (unsigned int)(ucParams[3])<< 24;

      iUserProcessPID=(int)pid;

      if(isVerbose())
            printf("%s: > en setUserProcessPID(pid=%d)\n",
                  RM_DEV_NAME,iUserProcessPID);

      return RM_NO_ERROR;
}

/**
 *    Pone a -1 el PID del proceso cliente
 *
```

```
 *    @param ucParams No se usa
 *    @return Devuelve siempre RM_NO_ERROR;
 *
 *    @see setUserProcessPID
 */
int clearUserProcessPID(unsigned char *ucParams)
{
      iUserProcessPID=-1;

      return RM_NO_ERROR;
}

/**
 *    Manda una señal SIGRTMIN al proceso de usuario con [canal,dato]
 *    de la conversion A/D (canal es de 4 bits y dato es de 12 bits,
 *    por lo que ambos se pueden compactar en 16 bits)
 *
 *    @param iChannel Canal al que pertenece el dato procedente de la
 *    conversion
 *    @param iData Dato A/D convertido
 *
 *    @return Devuelve 1 si todo ha ido bien y 0 si se ha producido
 *     algun error
 */
int signalToUser(int iChannel, int iData)
{
      union sigval valor;
      int res;

      if(iUserProcessPID!=-1)
      {
            valor.sival_int= ((channel & 0xF) << 12) | (data & 0x0FFF);
            res=sigqueue(iUserProcessPID,SIGRTMIN,valor);
            if(res==-1)
                  if(errno==EAGAIN)
                  {
                        //printf(" (EAGAIN) ");
                        return 1; // para que lo reintente luego
                  }
                  else
                        return 0;
            else
                  return 1;
      }

      return 0;
}

/**
 *    Obtiene las opciones de linea de comando que han sido invocadas al
 *    ejecutar el R.M.
 *
 *    @param ucParams Es un array de unsigned char's con el
 *    siguiente significado:
 *          - ucParams[0] = LSB de la direccion base donde esta mapeado el
```

```
 *           dispositivo
 *          - ucParams[1] = MSB de la direccion base donde esta mapeado el
 *           dispositivo
 *          - ucParams[2] = canal DMA
 *          - ucParams[3] = base temporal
 *          - ucParams[2] = modo de DI0/ExtTrigger
 *          - ucParams[3] = modo de los canales analogicos
 *    @return Devuelve el codigo de error
 */
int getOptions(unsigned char *ucParams)
{
      if(ucParams==NULL)
            return RM_ERROR_BAD_ARGUMENT;

      ucParams[0]=(unsigned char)(LOW_BYTE(getDirBase()));
      ucParams[1]=(unsigned char)(HIGH_BYTE(getDirBase()));
      ucParams[2]=(unsigned char)getDMAChannel();
      ucParams[3]=(unsigned char)getTimeBase();
      ucParams[4]=(unsigned char)getExtTrigger();
      ucParams[5]=(unsigned char)getAnalogChannelMode();

      return RM_NO_ERROR;
}

/**
 *    Presenta por pantalla las opciones disponibles para lanzar el R.M.
 */
void printUsage(void)
{
      printf("\n");
      printf("Uso:\n");
      printf("  rm_pcm3718h [opt]\n\n");
      printf("donde [opt] puede ser:\n");
      printf("  -h                  = help, muestra esta pantalla\n");
      printf("  -v                  = verbose, muestra informacion
            detallada durante\n");
      printf("                        la ejecucion\n");
      printf("  -b:<direccion hex>  = dirBase, establece la direccion base
            de la PCM3718H\n");
      printf("                        (entre 0x000 y 0x3F0) (*)\n");
      printf("  -d:<1 o 3>          = dmaChannel, establece el canal DMA
            que para la\n");
      printf("                         transferencia de datos de la
            conversion A/D\n");
      printf("  -t:<1 o 10>         = timeBase, establece la base de tiempo
            del contador\n");
      printf("                        interno (1 : 1MHz , 10 :10MHz)\n");
      printf("  -e                    = extTrigger, para indicar que DI0
            sera trigger externo\n");
      printf("                        y no entrada/salida digital\n");
      printf("\n");
      printf("(*) la direccion se debe dar en hexadecimal (p.e. si es
            0x1A2, la opcion de linea \n");
      printf("    de comandos seria '-b 1A2'\n");
      printf("\n");
}

/**
 *    Obtiene la direccion base para las operaciones de I/O
 *
```

```
 *    @return Devuelve el puntero al comienzo del mapeado I/O del
 *    dispositivo
 */
uintptr_t getDirBasePtr(void)
{
      return dirBasePtr;
}

/**
 *    Inicializa los valores por defecto en la tarjeta
 *
 *    @return Devuelve 1 si todo ha ido bien y 0 si se ha producido algun
 *    error
 */
int initBoard(void)
{
      unsigned char params[4];
      int i,channel[16]={
         RM_AD_CHANNEL0, RM_AD_CHANNEL1, RM_AD_CHANNEL2, RM_AD_CHANNEL3,
         RM_AD_CHANNEL4, RM_AD_CHANNEL5, RM_AD_CHANNEL6, RM_AD_CHANNEL7,
         RM_AD_CHANNEL8, RM_AD_CHANNEL9, RM_AD_CHANNEL10, RM_AD_CHANNEL11,
         RM_AD_CHANNEL12, RM_AD_CHANNEL13, RM_AD_CHANNEL14, RM_AD_CHANNEL15
      };

      // Configura los canales analogicos disponibles como bipolar -10..10
      if(isVerbose())
            printf("%s: > [init] Configurando canales analogicos
              (0..7)\n",RM_DEV_NAME);

      for(i=0;i<8;i++)
      {
            params[0]=channel[i];
            params[1]=RM_AD_RANGE_BIPOLAR_10_0;
            if(board_SetADRange(params)!=RM_NO_ERROR)
                  return 0;
      }

      if(isAnalogChannel16se())
      {
            if(isVerbose())
                  printf("%s: > [init] Configurando canales analogicos
                    (8..15)\n",RM_DEV_NAME);

            for(i=8;i<16;i++)
            {
                  params[0]=channel[i];
                  params[1]=RM_AD_RANGE_BIPOLAR_10_0;
                  if(board_SetADRange(params)!=RM_NO_ERROR)
                        return 0;
            }
      }

      // Configura el ciclo de conversion como start=0 y stop=0
      if(isVerbose())
            printf("%s: > [init] Configurando el ciclo de conversion
             A/D\n",RM_DEV_NAME);

      params[0]=0;
      params[1]=0;
      if(board_WriteADMuxScan(params)!=RM_NO_ERROR)
```

```
            return 0;

      // Configura la conversion A/D como 'sin interrupciones', 'sin DMA'
      // y el triggering por software
      if(isVerbose())
            printf("%s: > [init] Configurando el modo de conversion
             A/D\n",RM_DEV_NAME);

      params[0]=RM_AD_CONTROL_TRIGSRC_SOFTW;
      params[1]=RM_AD_CONTROL_TRIGSRC_BIT;
      if(board_SetADControlRegister(params)!=RM_NO_ERROR)
            return 0;

      // Configura los contadores 1 y 2 como LSBMSB, modo 3 y no BCD
      if(isVerbose())
            printf("%s: > [init] Configurando contador 1\n",RM_DEV_NAME);

      params[0]=RM_PACER_SC_COUNTER1;
      params[1]=RM_PACER_RW_LSBMSB;
      params[2]=RM_PACER_M_MODE3;
      params[3]=RM_PACER_BCD_BINARYCOUNT;
      if(board_PacerConfigureCounter(params)!=RM_NO_ERROR)
            return 0;

      if(isVerbose())
            printf("%s: > [init] Configurando contador 2\n",RM_DEV_NAME);

      params[0]=RM_PACER_SC_COUNTER2;
      params[1]=RM_PACER_RW_LSBMSB;
      params[2]=RM_PACER_M_MODE3;
      params[3]=RM_PACER_BCD_BINARYCOUNT;
      if(board_PacerConfigureCounter(params)!=RM_NO_ERROR)
            return 0;

      // Para indicar el modo R/W (y no tener que leerlo cada vez de la
      // tarjeta)
      iRWMode=RM_PACER_RW_LSBMSB;

      // Escribimos los valores iniciales de los contadores
      params[0]=RM_PACER_SC_COUNTER1;
      params[1]=LSB(65535);
      params[2]=MSB(65535);
      if(board_PacerWriteCounter(params)!=RM_NO_ERROR)
            return 0;

      params[0]=RM_PACER_SC_COUNTER2;
      params[1]=LSB(65535);
      params[2]=MSB(65535);
      if(board_PacerWriteCounter(params)!=RM_NO_ERROR)
            return 0;

      return 1;
}

/**
 *    Inicializa los punteros a los manejadores de los distintos mensajes
 */
void initHndFunctions(void)
{
      if(isVerbose())
```

```
            printf("%s: Inicializando manejadores de
             mensaje\n",strDevName);

      // casi todas las funciones en board.h ,las demas en este archivo

      hndArray[RM_MSG_SHOWOPTIONS]= showOptions;
      hndArray[RM_MSG_GETOPTIONS]= getOptions;

      hndArray[RM_MSG_DIG_SETBIT]= board_SetDigBit;
      hndArray[RM_MSG_DIG_RESETBIT]= board_ResetDigBit;
      hndArray[RM_MSG_DIG_READSTATUS]= board_ReadDigStatus;
      hndArray[RM_MSG_DIG_WRITESTATUS]= board_WriteDigStatus;

      hndArray[RM_MSG_AD_TRIGGERCONV]= board_TriggerConversion;
      hndArray[RM_MSG_AD_READDATA]= board_ReadADData;
      hndArray[RM_MSG_AD_SETRANGE]= board_SetADRange;
      hndArray[RM_MSG_AD_READMUXSCAN]= board_ReadADMuxScan;
      hndArray[RM_MSG_AD_WRITEMUXSCAN]= board_WriteADMuxScan;
      hndArray[RM_MSG_AD_READSTATUS]= board_GetADStatusRegister;
      hndArray[RM_MSG_AD_CLEARINTERRUPTREQUEST] =
            board_ClearInterruptRequest;
      hndArray[RM_MSG_AD_READCONTROL]= board_GetADControlRegister;
      hndArray[RM_MSG_AD_WRITECONTROL]= board_SetADControlRegister;

      hndArray[RM_MSG_PACER_ENABLEDISABLE] = board_EnableDisablePacer;
      hndArray[RM_MSG_PACER_GETENABLED]= board_GetPacerEnabled;
      hndArray[RM_MSG_PACER_READCOUNTER]= board_PacerReadCounter;
      hndArray[RM_MSG_PACER_WRITECOUNTER]= board_PacerWriteCounter;
      hndArray[RM_MSG_PACER_CONFIGURECOUNTER]= board_PacerConfigureCounter;
      hndArray[RM_MSG_PACER_READCOUNTERSTATUS]=
            board_PacerReadCounterStatus;
      hndArray[RM_MSG_PACER_READCONTROL]= board_PacerReadControl;
      hndArray[RM_MSG_PACER_WRITECONTROL]= board_PacerWriteControl;

      hndArray[RM_MSG_AD_SET_PID]= setUserProcessPID;
      hndArray[RM_MSG_AD_CLEAR_PID]= clearUserProcessPID;
}

/**
 *    Procesa el mensaje que le llega como parametro y lo
 *    sobreescribe con el mensaje de respuesta.
 *
 *    @param msg Mensaje (de tipo pcm3718h_msg_t *, definido
 *          en constants.h)
 *
 *    @return Devuelve RM_NO_ERROR si todo va bien
 *    @see constants.h
 */
int RMProcessMsg(pcm3718h_msg_t *msg)
{
      int ret,id;

      if(msg==NULL)
            return RM_ERROR_BAD_ARGUMENT;

      id=msg->id - _IO_MAX; // ver message_attach() en main()

      if(id<RM_MSG_FIRST || id>RM_MSG_LAST)
      {
            //    No se conoce el mensaje, y asi lo indicamos
```

```
            msg->error=RM_ERROR_UNK_MSG;

            if(isVerbose())
                  printf("%s: [error: mensaje desconocido
                   (%d)]\n",strDevName,id);

      }
      else
      {
            // Llamamos al handler
            if(hndArray[id]!=NULL)
            {
                  if(isVerbose())
                        printf("%s: Llamando al manejador del mensaje
                         msg_id=%d\n",strDevName,id);

                  ret=(*hndArray[id])(msg->data);

                  if(isVerbose())
                        printf("%s: El manejador del mensaje msg_id=%d
                         devuelve %d\n",strDevName,id,ret);
            }
            else
            {
                  if(isVerbose())
                        printf("%s: [error: no hay manejador para
                         msg_id=%d]\n",strDevName,id);

                  ret=RM_ERROR_NO_HANDLER;
            }

            msg->error=ret;
      }

      //    En todo caso, siempre hay una respuesta
      msg->id=RM_MSG_REPLY;

      return RM_NO_ERROR;
}

/**
 *    Funcion de manejo de los mensajes
 *
 *    @param ctp contexto de mensaje
 *    @param type tipo del mensaje
 *    @param flags opciones en forma de patron de bits
 *
 *    @return devuelve el codigo de error
 */
int msg_callback(message_context_t *ctp, int type,unsigned flags, void
*handle)
{
      pcm3718h_msg_t *msg;
      int ret;

      if(isVerbose())
            printf("%s: Entrando en msg_callback\n",strDevName);

      msg=(pcm3718h_msg_t *)ctp->msg;
```

```
      if(isVerbose())
            printf("%s: Procesado del mensaje\n",strDevName);

      //    procesamos el mensaje, ¿y ret? (p.e. se podria usar para llevar
      //    estadisticas de msgs fallidos)
      ret=RMProcessMsg(msg);

      if(isVerbose())
            printf("%s: Fin de procesado del mensaje\n",strDevName);

      //    y lo devolvemos como respuesta
      MsgReply(ctp->rcvid,EOK,msg,sizeof(pcm3718h_msg_t));

      if(isVerbose())
            printf("%s: Saliendo de msg_callback\n",strDevName);

      return 0;
}

/**
 *    Manejador para la llamada 'open'
 *    Llegamos a esta funciÃ³n cuando el cliente realiza una llamada open.
 *    Depende de nosotros si queremos establecer un contexto o no.
 *    Simplemente devolvemos un codigo de estado.
 *
 *    @param ctp contexto del Resource Manager
 *    @param msg contiene el mensaje asociado a 'open'
 *    @param handle manejador del Resource Manager
 *    @param extra datos extras
 *
 *    @return devuelve el codigo de error
 */
int io_open (resmgr_context_t *ctp, io_open_t *msg, RESMGR_HANDLE_T
*handle, void *extra)
{
      if (isVerbose()) {
            printf ("%s: En io_open\n", strDevName);
      }

    return (iofunc_open_default (ctp, msg, handle, extra));
}

/**
 *    Manejador para la llamada 'read'
 *    El cliente ha realizado un llamada a "read"
 *    y espera recibir 0 o mÃ¡s bytes. Debido a que es el
 *    resource manager /dev/Null, devolvemos 0 bytes para
 *    indicar EOF -- no mas bytes.
 *
 *    @param ctp contexto del Resource Manager
 *    @param msg contiene el mensaje asociado a 'read'
 *    @param handle puntero al OCB del Resource Manager
 *
 *    @return devuelve el codigo de error
*/
int io_read (resmgr_context_t *ctp, io_read_t *msg, RESMGR_OCB_T *ocb)
{
      int status;

      if (isVerbose()) printf ("%s: En io_read\n", strDevName);
```

```
      if ((status = iofunc_read_verify(ctp, msg, ocb, NULL)) != EOK)
            return (status);

      // Ningun tipo especial xtype
      if ( (msg->i.xtype & _IO_XTYPE_MASK) != _IO_XTYPE_NONE) {
            return(ENOSYS);
      }

      _IO_SET_READ_NBYTES (ctp, 0);

      if (msg->i.nbytes > 0) ocb->attr->flags |= IOFUNC_ATTR_ATIME;

      return(_RESMGR_NPARTS (0));
}

/**
 *    Manejador para la llamada 'write'
 *    El client ha llamado a la funcion "write"
 *    y espera que el resource manager escriba el numero
 *    de bytes especificados a algun dispositivo.
 *    Como es /dev/Null, todas las escrituras siempre
 *    funcionan.
 *
 *    @param ctp contexto del Resource Manager
 *    @param msg contiene el mensaje asociado a 'write'
 *    @param handle puntero al OCB del Resource Manager
 *
 *    @return devuelve el codigo de error
*/

int io_write (resmgr_context_t *ctp, io_write_t *msg, RESMGR_OCB_T *ocb)
{
      int   status,iRes=1;
      char *pszBuff;

      if (isVerbose()) printf ("%s: En io_write\n", strDevName);

      if ((status = iofunc_write_verify(ctp, msg, ocb, NULL)) != EOK)
            return (status);

      // No special xtypes
      if ( (msg->i.xtype & _IO_XTYPE_MASK) != _IO_XTYPE_NONE) {
            return(ENOSYS);
      }

      _IO_SET_WRITE_NBYTES (ctp, msg -> i.nbytes);

      /*
       *    Comprobamos si nos piden que terminemos la ejecucion
       */
      pszBuff = (char *) malloc(msg->i.nbytes + 1);
      if (pszBuff == NULL)
            return(ENOMEM);

      resmgr_msgread(ctp, pszBuff, msg->i.nbytes, sizeof(msg->i));

      //    forzamos a que sea un 'string' poniendo al final el '\0'
      if(strchr(pszBuff,'\0')==NULL)
          pszBuff[msg->i.nbytes] = '\0';
```

```
      iRes=strcmp(pszBuff,"terminate\n");
      if(iRes==0)
      {
            iFin=1;
            if(isVerbose())
                  printf("%s: < terminating!! >\n",strDevName);
      }
      else
            if(isVerbose())
                  printf("%s: received: %s.\n",strDevName,pszBuff);

      if (msg->i.nbytes > 0)
            ocb->attr->flags |= IOFUNC_ATTR_MTIME | IOFUNC_ATTR_CTIME;

      return (_RESMGR_NPARTS (0));
}

/*
 *    Configura el proceso actual para tener acceso al dispositivo a
 *    traves de in8 y out8. Devuelve 0 si ha habido algun problema
 */
int attachToDevice(void)
{
      if(ThreadCtl_r(_NTO_TCTL_IO,NULL)!=EOK)
      {
            fprintf(stderr,"%s: (!) Error adquiriendo privilegios de I/O
             (ThreadCtrl)\n",strDevName);
            return 0;
      }

      if((dirBasePtr=mmap_device_io(16,getDirBase())) ==
       (uintptr_t)MAP_FAILED)
      {
            fprintf(stderr,"%s: (!) Error mapeando las direcciones I/O del
             dispositivo (mmap_device_io)\n",strDevName);
            return 0;
      }

      return 1;
}

/*
 *    Desconfigura el proceso actual para tener acceso al dispositivo a
 *    traves de in8 y out8. Devuelve 0 si ha habido algun problema
 */
int dettachToDevice(void)
{
      if(munmap_device_io(dirBasePtr,16)==-1)
      {
            fprintf(stderr,"%s: (!) Error desmapeando las direcciones I/O
             del dispositivo (munmap_device_io)\n",strDevName);
            return 0;
      }
      else
            return 1;
}

/* ========================================
 *    M A I N
```

```
 * ========================================
 */

int main (int argc, char **argv)
{
      int pathID,iRes;

      /*
       *    Analizamos la linea de comandos para buscar opciones
       *    seleccionadas por el usuario
       */
      if(parseOptions(argc, argv)!=RM_NO_ERROR)
      {
            fprintf(stderr,"%s: (!) Error en los argumentos de la linea de
             comandos.\n",strDevName);
            printUsage();
            return -1;
      }

      if(userWantsHelp())
      {
            printUsage();
            return 0;
      }

      if(isVerbose())
            printf ("%s: Iniciando...\n", strDevName);

      if(attachToDevice()==0)
      {
            fprintf(stderr,"%s: (!) Error configurando el acceso al
             dispositivo.\n",strDevName);
            return -1;
      }

      //    Si ya tenemos acceso al dispositivo, inicializamos los valores
      //    por defecto
      if(!initBoard())
      {
            fprintf(stderr,"%s: (!) Error inicializando los valores por
             defecto.\n",strDevName);
            return -1;
      }

      /*
       *    Creamos la estructura de dispatch que utilizara nuestro bucle
       *    de de recepcion de mensajes
       */
      if(isVerbose())
            printf("%s: dispatch_create...\n", strDevName);

      dpp = dispatch_create();
      if (dpp == NULL)
      {
            fprintf (stderr, "%s: (!) Error en dispatch_create: %s\n",
                                       strDevName, strerror (errno));
            exit (1);
      }
```

```
	/*
	 *    Inicializamos el array de handlers de mensaje
	 */
	initHndFunctions();

	/*
	 *    Inicializamos las estructuras de atributos del resource manager
	 *    que pasaremos a la llamada resmgr_attach().
	 *    Aqui se tiene el comportamiento por defecto.
	 */
	memset (&rattr, 0, sizeof (rattr));
	rattr.nparts_max=1;
	rattr.msg_max_size=2048;

	/*
	 *    Inicializamos las estructruas de funciones connect e I/O
	 *    a sus parametros por defecto y sobreescribimos las funciones
	 *    para las que aÃ±adimos funcionalidad.
	 */
	if(isVerbose())
		printf("%s: iofunc_func_init...\n", strDevName);

	iofunc_func_init (_RESMGR_CONNECT_NFUNCS, &connect_funcs,
	                  _RESMGR_IO_NFUNCS, &io_funcs);
	connect_funcs.open = io_open;
	io_funcs.read = io_read;
	io_funcs.write = io_write;

	/*
	 *    Llamamos resmgr_attach para registrarnos en el espacio de
	 *    nombres. Si se produce un eror, devuelve -1 y se fija errno.
	 */
	if(isVerbose())
		printf("%s: iofunc_attr_init...\n", strDevName);

	iofunc_attr_init(&ioattr, S_IFCHR | 0666, NULL, NULL);

	if(isVerbose())
		printf("%s: resmgr_attach...\n", strDevName);

	pathID = resmgr_attach(dpp, &rattr, strDevName, _FTYPE_ANY, 0,
	                       &connect_funcs, &io_funcs, &ioattr);
	if (pathID == -1)
	{
		fprintf (stderr, "%s:(!) Error en resmg_attach: %s\n",
		                 strDevName, strerror (errno));
		exit (1);
	}
	else
		printf("%s: LinkID = %d\n",strDevName,pathID);

	/*
	 *    Indicamos la funcion que queremos que se encargue
	 *    de responder a los mensajes
	 */

	if(isVerbose())
		printf("%s: message_attach...\n", strDevName);

	memset(&msgAttr,0,sizeof(msgAttr));
```

```
	msgAttr.nparts_max=1;
	msgAttr.msg_max_size=4096;
	iRes=message_attach(dpp,&msgAttr,
		_IO_MAX+RM_MSG_FIRST,_IO_MAX+RM_MSG_LAST,
		msg_callback,NULL);
	if(iRes==-1)
	{
		fprintf (stderr, "%s:(!) Error en resmg_attach: %s\n",
				strDevName, strerror (errno));
		exit (1);
	}

	if(isVerbose())
		printf("%s: dispatch_context_alloc...\n", strDevName);
	ctp = dispatch_context_alloc(dpp);

	/*
	 *	Entramos en el bucle principal de escucha/respuesta de mensajes
	 */

	if(isVerbose()) printf("%s: main loop...\n", strDevName);
	while (!iFin)
	{
		if ((ctp = dispatch_block(ctp)) == NULL)
		{
			fprintf (stderr, "%s:(!) Error en dispatch_block: %s\n",
					strDevName, strerror (errno));
			exit (1);
		}
		dispatch_handler(ctp);
	}

	/*
	 *	Salimos, quitamos la asociacion de este proceso al nombre
	 *	reservado antes en el espacio de nombres de /dev
	 */

	if(isVerbose())
		printf("%s: resmgr_detach...\n", strDevName);

	iRes=resmgr_detach(dpp,pathID,_RESMGR_DETACH_ALL);
	if(iRes==-1)
	{
		fprintf (stderr, "%s:(!) Error en resmg_detach: %s\n",
				strDevName, strerror (errno));
		exit (1);
	}

	if(isVerbose())
		printf("%s: detachToDevice...\n", strDevName);
	dettachToDevice();

	return 0;
}
```

4.1.2 constants.h

```
/**
 *	Contiene constantes, macros y estructuras para el
```

```
 *    uso de la PCM3718H
 *
 *    @file constants.h
 *    @author Raul Jimenez Ruiz
 */

#ifndef _CONSTANTS_H_
#define _CONSTANTS_H_

typedef unsigned char BYTE;

/*================================================================
 *    MANEJO DE BITS / BYTES
 *  (en relacion con enteros de 2 bytes)
 */
#define BYTE_MASK                                ((int)0x00FF)
#define BYTE_BIT0                                ((int)0x0001)
#define BYTE_BIT1                                ((int)0x0002)
#define BYTE_BIT2                                ((int)0x0004)
#define BYTE_BIT3                                ((int)0x0008)
#define BYTE_BIT4                                ((int)0x0010)
#define BYTE_BIT5                                ((int)0x0020)
#define BYTE_BIT6                                ((int)0x0040)
#define BYTE_BIT7                                ((int)0x0080)

#define BYTE_SET_BIT(x,bit)         ((int)((int)(x) |= (int)(bit)))
#define BYTE_RESET_BIT(x,bit)       ((int)((int)(x) &= ~(int)(bit)))
#define BYTE_READ_BIT(x,bit)        ((int)((int)(x) & (int)(bit)))

#define APPLY_MASK(var,value,mask)  ((int)((int)(mask) & (int)(value)) | \
                                    (~(int)(mask) & (int)(var)))

#define MAKE_INT(hi,lo)             ((int)(((int)hi)<<8 | ((int)(lo))))

#define HIGH_BYTE(i)                      ((int)(((i)>>8) & BYTE_MASK))
#define LOW_BYTE(i)                       ((int)((i) & BYTE_MASK))
// quitar los dos anteriores
#define MSB(i)                            ((int)(((i)>>8) & BYTE_MASK))
#define LSB(i)                            ((int)((i) & BYTE_MASK))

#define HIGH_NIBBLE                       ((unsigned char)0xF0)
#define LOW_NIBBLE                        ((unsigned char)0x0F)

/*================================================================
 *    CONSTANTES PARA EL USO DE LA PCM3718H
 */

/*
 *    Para seleccionar los canales digitales
 */
#define RM_DIG_CHANNEL_A                         ((int)0)
#define RM_DIG_CHANNEL_B                         ((int)1)

/*
 *    Para indicar que un parametro, de tipo int, es 'nulo' a la hora de
 *    llamar ciertas funciones de la libreria. Uso del usuario de la
 *    libreria.
 */
#define SO_NULLPARAM                             ((int)0xFFFF)
```

```
/*
 *    Para indicar que un parametro, de tipo int, es 'nulo' a la hora de
 *    llamar ciertas funciones del Resource Manager. De uso interno (lo usa
la
 *  libreria en sus mensajes al Resource Manager).
 */
#define RM_NULLPARAM                            ((unsigned char)0xFF)

/*
 *    Nombre del Resource Manager en el espacio de nombres.
 */
#define RM_DEV_NAME "/dev/pcm3718h"

/*
 *    Datos sobre la conversion A/D
 */
#define RM_AD_MAX_CHANNELS                      ((int)16)
#define RM_AD_DATA_BITS                         ((int)12)
#define RM_AD_DATA_BITMASK                      ((int)0x0FFF)
#define RM_AD_DATA_MAX_INT_VALUE                ((int)4095)

/*
 *    Manejo del registro de estado A/D
 */
#define RM_AD_STATUS_EOC_BIT                    ((int)0x80)
#define RM_AD_STATUS_MUX_BIT                    ((int)0x20)
#define RM_AD_STATUS_INT_BIT                    ((int)0x10)
#define RM_AD_STATUS_CHANNEL_BIT                ((int)0x0F)

#define RM_AD_STATUS_MUX_8DIFF                  ((int)0x00)
#define RM_AD_STATUS_MUX_16SE                   ((int)0x20)

/*
 *    Manejo del registro de control A/D
 */

#define RM_AD_CONTROL_INTE_BIT                  ((int)0x80)
#define RM_AD_CONTROL_INTLEVEL_BIT              ((int)0x70)
#define RM_AD_CONTROL_DMAE_BIT                  ((int)0x04)
#define RM_AD_CONTROL_TRIGSRC_BIT               ((int)0x03)

#define RM_AD_CONTROL_IRQ2                      ((int)0x20)
#define RM_AD_CONTROL_IRQ3                      ((int)0x30)
#define RM_AD_CONTROL_IRQ4                      ((int)0x40)
#define RM_AD_CONTROL_IRQ5                      ((int)0x50)
#define RM_AD_CONTROL_IRQ6                      ((int)0x60)
#define RM_AD_CONTROL_IRQ7                      ((int)0x70)

#define RM_AD_CONTROL_TRIGSRC_SOFTW             ((int)1)
#define RM_AD_CONTROL_TRIGSRC_EXT               ((int)2)
#define RM_AD_CONTROL_TRIGSRC_PACER             ((int)3)

/*
 *    Para seleccionar los canales analogicos
 */
#define RM_AD_CHANNEL_MASK                      ((int)0x0F)
#define RM_AD_CHANNEL0                          ((int)0)
#define RM_AD_CHANNEL1                          ((int)1)
#define RM_AD_CHANNEL2                          ((int)2)
#define RM_AD_CHANNEL3                          ((int)3)
```

```
#define RM_AD_CHANNEL4                          ((int)4)
#define RM_AD_CHANNEL5                          ((int)5)
#define RM_AD_CHANNEL6                          ((int)6)
#define RM_AD_CHANNEL7                          ((int)7)
#define RM_AD_CHANNEL8                          ((int)8)
#define RM_AD_CHANNEL9                          ((int)9)
#define RM_AD_CHANNEL10                         ((int)10)
#define RM_AD_CHANNEL11                         ((int)11)
#define RM_AD_CHANNEL12                         ((int)12)
#define RM_AD_CHANNEL13                         ((int)13)
#define RM_AD_CHANNEL14                         ((int)14)
#define RM_AD_CHANNEL15                         ((int)15)

/*
 *    Para seleccionar el rango de los canales analogicos
 */
#define RM_AD_RANGE_MASK                        ((int)0x0F)
#define RM_AD_RANGE_BIPOLAR_10_0                ((int)8)
#define RM_AD_RANGE_BIPOLAR_5_0                 ((int)0)
#define RM_AD_RANGE_BIPOLAR_2_5                 ((int)1)
#define RM_AD_RANGE_BIPOLAR_1_25                ((int)2)
#define RM_AD_RANGE_BIPOLAR_0_625               ((int)3)
#define RM_AD_RANGE_UNIPOLAR_10_0               ((int)4)
#define RM_AD_RANGE_UNIPOLAR_5_0                ((int)5)
#define RM_AD_RANGE_UNIPOLAR_2_5                ((int)6)
#define RM_AD_RANGE_UNIPOLAR_1_25               ((int)7)

#define RM_AD_RANGE_MIN
      RM_AD_RANGE_BIPOLAR_5_0
#define RM_AD_RANGE_MAX
      RM_AD_RANGE_BIPOLAR_10_0

/*
 *    Para seleccionar el modo de transferencia de los datos
 *    de una conversion A/D
 */
#define RM_AD_TRANSFERMODE_PROGRAM              ((int)1)
#define RM_AD_TRANSFERMODE_IRQ                  ((int)2)
#define RM_AD_TRANSFERMODE_DMA                  ((int)3)

/*
 *    Opciones del 'Pacer'
 */
#define RM_PACER_ENABLE                         ((int)0)
#define RM_PACER_DISABLE                        ((int)1)

#define RM_PACER_SC_MASK                        ((int)0xC0)
#define RM_PACER_SC_COUNTER0                    ((int)0x00)
#define RM_PACER_SC_COUNTER1                    ((int)0x40)
#define RM_PACER_SC_COUNTER2                    ((int)0x80)
#define RM_PACER_SC_READBACK_COMMAND            ((int)0xC0)

#define RM_PACER_RW_MASK                        ((int)0x30)
#define RM_PACER_RW_COUNTERLATCH                ((int)0x00)
#define RM_PACER_RW_LSB                         ((int)0x10)
#define RM_PACER_RW_MSB                         ((int)0x20)
#define RM_PACER_RW_LSBMSB                      ((int)0x30)

#define RM_PACER_M_MASK                         ((int)0x0E)
#define RM_PACER_M_MODE0                        ((int)0x00)
```

```
#define RM_PACER_M_MODE1                        ((int)0x02)
#define RM_PACER_M_MODE2                        ((int)0x04)
#define RM_PACER_M_MODE3                        ((int)0x06)
#define RM_PACER_M_MODE4                        ((int)0x08)
#define RM_PACER_M_MODE5                        ((int)0x0A)

#define RM_PACER_BCD_MASK                       ((int)0x01)
#define RM_PACER_BCD_BINARYCOUNT                ((int)0)
#define RM_PACER_BCD_BCDCOUNT                   ((int)1)
#define RM_PACER_BCD_BINARYCOUNT_MAX            ((int)65535)
#define RM_PACER_BCD_BCDCOUNT_MAX               ((int)9999)

#define RM_PACER_STATUS_OUT_BIT                 ((int)0x80)
#define RM_PACER_STATUS_NC_BIT                  ((int)0x40)
#define RM_PACER_STATUS_RW_BIT                  ((int)0x30)
#define RM_PACER_STATUS_M_BIT                   ((int)0x0E)
#define RM_PACER_STATUS_BCD_BIT                 ((int)0x01)

#define RM_PACER_CONTROL_CNT_BIT                ((int)0x20)
#define RM_PACER_CONTROL_STA_BIT                ((int)0x10)
#define RM_PACER_CONTROL_COUNTER2_BIT           ((int)0x08)
#define RM_PACER_CONTROL_COUNTER1_BIT           ((int)0x04)
#define RM_PACER_CONTROL_COUNTER0_BIT           ((int)0x02)

/*================================================================
 *    Opciones de la linea de comandos (se va a hacer de otro modo)
 */
#define RM_OPT_NONE                             ((int)0)
#define RM_OPT_VERBOSE                          ((int)1)

/*================================================================
 *    MENSAJE
 */
/*    Tamaño maximo de los datos del mensaje */
#define RM_MSG_MAX_LEN  16

/*
 *    Estructura que define la forma del mensaje
 */
typedef struct
{
      unsigned int      id;                     /**< ID del mensaje*/
      int               error;                  /**< Codigo de error */
      unsigned char     data[RM_MSG_MAX_LEN];   /**< Cuerpo del mensaje*/
}pcm3718h_msg_t;

/*
 *    Lista de mensajes
 */
#define RM_MSG_FIRST    RM_MSG_DIG_SETBIT

//    DIGITAL
#define RM_MSG_DIG_SETBIT                       ((int)1)
#define RM_MSG_DIG_RESETBIT                     ((int)2)
#define RM_MSG_DIG_READSTATUS                   ((int)3)
#define RM_MSG_DIG_WRITESTATUS                  ((int)4)
#define RM_MSG_DIG_READINT                      ((int)5)
#define RM_MSG_DIG_WRITEINT                     ((int)6)

//    OPTIONS
```

```
#define RM_MSG_SHOWOPTIONS                      ((int)7)
#define RM_MSG_GETOPTIONS                       ((int)8)

//    A/D
#define RM_MSG_AD_READSTATUS                    ((int)9)
#define RM_MSG_AD_CLEARINTERRUPTREQUEST         ((int)10)
#define RM_MSG_AD_READCONTROL                   ((int)11)
#define RM_MSG_AD_WRITECONTROL                  ((int)12)
#define RM_MSG_AD_READDATA                      ((int)13)
#define RM_MSG_AD_TRIGGERCONV                   ((int)14)
#define RM_MSG_AD_SETRANGE                      ((int)15)
#define RM_MSG_AD_READMUXSCAN                   ((int)16)
#define RM_MSG_AD_WRITEMUXSCAN                  ((int)17)

//    PACER
#define RM_MSG_PACER_ENABLEDISABLE              ((int)18)
#define RM_MSG_PACER_GETENABLED                 ((int)19)
#define RM_MSG_PACER_READCOUNTER                ((int)20)
#define RM_MSG_PACER_WRITECOUNTER               ((int)21)
#define RM_MSG_PACER_CONFIGURECOUNTER           ((int)22)
#define RM_MSG_PACER_READCOUNTERSTATUS          ((int)23)
#define RM_MSG_PACER_READCONTROL                ((int)24)
#define RM_MSG_PACER_WRITECONTROL               ((int)25)

//    A/D (2), para la captura por interrupciones
#define RM_MSG_AD_SET_PID                       ((int)26)
#define RM_MSG_AD_CLEAR_PID                     ((int)27)

#define RM_MSG_LAST           RM_MSG_AD_CLEAR_PID

#define RM_MSG_UNK                              ((int)1000)
#define RM_MSG_REPLY                            ((int)1001)

/*
 *    Tipo de los manejadores de mensaje
 */
typedef int (*msg_hnd_t)(unsigned char *ucParams);

/*
 *    Lista de errores posibles del Rsesource Manager
 */
#define RM_NO_ERROR                             ((int)0)
#define RM_ERROR_OPN_DEV                        ((int)1)
#define RM_ERROR_UNK_MSG                        ((int)2)
#define RM_ERROR_MSG_SEND                       ((int)3)
#define RM_ERROR_BAD_ARGUMENT                   ((int)4)
#define RM_ERROR_OUT_OF_RANGE                   ((int)5)
#define RM_ERROR_NO_HANDLER                     ((int)6)
#define RM_ERROR_CMDLINE_PARSING                ((int)7)
#define RM_ERROR_EXECUTING_COMMAND              ((int)8)
#define RM_ERROR_NOT_IMPLEMENTED                ((int)9)
#define RM_ERROR_CONFIGURATION                  ((int)10)
#define RM_ERROR_IO                             ((int)11)
#define RM_ERROR_THREADCTL                      ((int)12)
#define RM_ERROR_INSTALL_ISR                    ((int)13)
#define RM_ERROR_UNINSTALL_ISR                  ((int)14)

#endif
```

4.1.3 board.h

```
/**
 *    Contiene constantes y funciones para el uso de la PCM3718H
 *    a nivel hardware.
 *
 *    @file board.h
 *    @author Raul Jimenez Ruiz
 */

#ifndef _BOARD_H_
#define _BOARD_H_

#include <sys/mman.h>   //    para uintptr_t
#include <hw/inout.h>   //    para in8 y out8

//    De rm_pcm3718h.c
_Uintptrt getDirBasePtr(void);

#define READ(dir)       ((unsigned char)(in8(getDirBasePtr()+(dir))))
#define WRITE(dir,val)  out8(getDirBasePtr()+(dir),(unsigned char)(val))

/*******************************************************
 *
 *    E/S DIGITAL
 *
 *******************************************************/

/**
 *    Pone a 1 los bis del canal seleccionado que esten
 *    a 1 en la mascara.
 *
 *    @param ucParams Es un array de unsigned char's con el
 *          siguiente significado:
 *          - ucParams[0] = indica el canal que se modifica,
 *               puede ser RM_DIG_CHANNEL_A o RM_DIG_CHANNEL_B.
 *          - ucParams[1] = mascara de bits a poner a 1
 *
 *    @return Devuelve el codigo de error
 */
int board_SetDigBit(unsigned char * ucParams);

/**
 *    Pone a 0 los bis del canal seleccionado que esten
 *    a 1 en la mascara.
 *
 *    @param ucParams Es un array de unsigned char's con el
 *          siguiente significado:
 *          - ucParams[0] = indica el canal que se modifica,
 *               puede ser RM_DIG_CHANNEL_A o RM_DIG_CHANNEL_B.
 *          - ucParams[1] = mascara de bits a poner a 0
 *
 *    @return Devuelve el codigo de error
 */
int board_ResetDigBit(unsigned char * ucParams);

/**
 *    Lee el estado de las seÃ±ales digitales.
 *
```

```
 *    @param ucParams Es un array de unsigned char's con el
 *          siguiente significado:
 *          - ucParams[0] = 0/1 si queremos leer el canal A
 *          - ucParams[1] = 0/1 si queremos leer el canal B
 *          (devuelve en estas mismas posiciones los estados
 *          que se querian leer)
 *
 *    @return Devuelve el codigo de error
 *
 */
int board_ReadDigStatus(unsigned char * ucParams);

/**
 *    Pone a 1 los bis del canal seleccionado que esten
 *    a 1 en la mascara.
 *
 *    @param ucParams Es un array de unsigned char's con el
 *          siguiente significado:
 *          - ucParams[0] = 0/1 si queremos escribir el canal A
 *          - ucParams[1] = 0/1 si queremos escribir el canal B
 *          - ucParams[2] = estado del canal A
 *          - ucParams[3] = estado del canal B
 *
 *    @return Devuelve el codigo de error
 */
int board_WriteDigStatus(unsigned char * ucParams);

/**
 *    Lee las entradas digitales como si representaran un entero
 *    de 2 bytes, siendo el canal A el LSB y el canal B el MSB de
 *    dicho entero.
 *
 *    @param ucParams Es un array de unsigned char's con el
 *          siguiente significado:
 *          - ucParams[0] = LSB del entero
 *          - ucParams[1] = MSB del entero
 *
 *    @return Devuelve el codigo de error
 */
int board_ReadDigInt(unsigned char * ucParams);

/**
 *    Escribe las salidas digitales como si representaran un entero
 *    de 2 bytes, siendo el canal A el LSB y el canal B el MSB de
 *    dicho entero.
 *
 *    @param ucParams Es un array de unsigned char's con el
 *          siguiente significado:
 *          - ucParams[0] = LSB del entero
 *          - ucParams[1] = MSB del entero
 *
 *    @return Devuelve el codigo de error
 */
int board_WriteDigInt(unsigned char * ucParams);

/*****************************************************
 *
 *    REGISTROS DE ESTADO Y CONTROL
 *
 *****************************************************/
```

```
/**
 *    Lee el registro de estado de la tarjeta
 *
 *    @param ucParams Es usado para devolver (en ucParams[0]) el registro
 *    de estado
 *
 *    @return Devuelve el codigo de error
 */
int board_GetADStatusRegister(unsigned char *ucParams);

/**
 *    Borra el bit INT del registro de estado
 *
 *    @param ucParams Es ignorado (se mantiene por compatibilidad)
 *
 *    @return Devuelve el codigo de error
 */
int board_ClearInterruptRequest(unsigned char *ucParams);

/**
 *    Lee el registro de control de la tarjeta
 *
 *    @param ucParams Es usado para devolver (en ucParams[0]) el registro
 *    de control
 *
 *    @return Devuelve el codigo de error
 */
int board_GetADControlRegister(unsigned char *ucParams);

/**
 *    Escribe en el registro de control de la tarjeta
 *
 *    @param ucParams Es un array de unsigned char's con el
 *          siguiente significado:
 *          - ucParams[0] = bits a modificar en el registro de control
 *          - ucParams[1] = mascara con los bits de ucParams[0] a usar
 *
 *    @return Devuelve el codigo de error
 */
int board_SetADControlRegister(unsigned char *ucParams);

/*******************************************************
 *
 *    CONVERSION A/D
 *
 *******************************************************/

/**
 *    Devuelve el dato A/D convertido, asi como el canal del
 *    que procede.
 *
 *    @param ucParams Es un array de unsigned char's en el que
 *          se devuelve la informacion solicitada y tiene el siguiente
 *          significado:
 *          - ucParams[0] = canal A/D del que procede el dato
 *          - ucParams[1] = LSB del dato convertido
 *          - ucParams[2] = MSB del dato convertido
 *
 *    @return Devuelve el codigo de error
```

```
 */
int board_ReadADData(unsigned char * ucParams);

/**
 *    Lanza una conversion A/D.
 *
 *    @param ucParams Pueder ser NULL. No se usa.
 *
 *    @return Devuelve el codigo de error
 */
int board_TriggerConversion(unsigned char * ucParams);

/**
 *    Establece un rango de conversion para un canal especificado
 *
 *    @param ucParams Es un array de unsigned char's con el
 *          siguiente significado:
 *          - ucParams[0] = canal a configurar, puede ser un valor
 *           entre RM_AD_CHANNEL0 y RM_AD_CHANNEL15, dependiendo
 *           de los canales que tenga configurada la tarjeta.
 *          - ucParams[1] = rango de conversion a establecer. Puede
 *           ser uno de entre los valores siguientes:
 *            - RM_AD_RANGE_BIPOLAR_10_0:  -10 .. +10 V
 *            - RM_AD_RANGE_BIPOLAR_5_0:   -5 .. +5 V
 *            - RM_AD_RANGE_BIPOLAR_2_5:   -2.5 .. +2.5 V
 *            - RM_AD_RANGE_BIPOLAR_1_25:  -1.25 .. +1.25 V
 *            - RM_AD_RANGE_BIPOLAR_0_625: -0.625 .. +0.625 V
 *            - RM_AD_RANGE_UNIPOLAR_10_0: 0 .. 10 V
 *            - RM_AD_RANGE_UNIPOLAR_5_0:  0 .. 5 V
 *            - RM_AD_RANGE_UNIPOLAR_2_5:  0 .. 2.5 V
 *            - RM_AD_RANGE_UNIPOLAR_1_25: 0 .. 1.25 V
 *
 *    @return Devuelve el codigo de error
 *    @see constants.h
 */
int board_SetADRange(unsigned char * ucParams);

/**
 *    Lee el registro scan de multiplexado de canales, en el que
 *    se indica el canal de comienzo y el de fin del ciclo de
 *    conversion.
 *
 *    @param ucParams Es un array de unsigned char's con el
 *          siguiente significado:
 *          - ucParams[0] = canal de comienzo del ciclo
 *          - ucParams[1] = canal de fin del ciclo
 *          Dichos canales pueden ser un valor entre
 *          RM_AD_CHANNEL0 y RM_AD_CHANNEL15.
 *
 *    @return Devuelve el codigo de error
 */
int board_ReadADMuxScan(unsigned char * ucParams);

/**
 *    Escribe el registro scan de multiplexado de canales con
 *    el canal de comienzo y el canal de fin de ciclo de
 *    conversion.
 *
 *    @param ucParams Es un array de unsigned char's con el
 *          siguiente significado:
```

```
 *          - ucParams[0] = canal de comienzo del ciclo
 *          - ucParams[1] = canal de fin del ciclo
 *          Dichos canales pueden ser un valor entre
 *          RM_AD_CHANNEL0 y RM_AD_CHANNEL15.
 *
 *    @return Devuelve el codigo de error
 */
int board_WriteADMuxScan(unsigned char * ucParams);


/*****************************************************
 *
 *    CONTADOR
 *
 *****************************************************/

/**
 *    Habilita o deshabilita el Pacer
 *
 *    @param ucParams Es un array de unsigned char's con el
 *          siguiente significado:
 *          - ucParams[0] = 0/1 si queremos deshabilitar/habilitar en
 *           contador
 *
 *    @return Devuelve el codigo de error
 */
int board_EnableDisablePacer(unsigned char *ucParams);

/**
 *    Lee el estado de habilitacion del Pacer
 *
 *    @param ucParams En ucParams[0] se devuelve 0/1 segun el estado del
 *     pacer
 *
 *    @return Devuelve el codigo de error
 */
int board_GetPacerEnabled(unsigned char *ucParams);

/**
 *    Lee el valor de uno de los contadores del Pacer
 *
 *    @param ucParams en unParams[0] debe ir el ID del contador
 *          del que queremos leer su valor. Puede ser uno de entre
 *          RM_PACER_SC_COUNTER0 a RM_PACER_SC_COUNTER2.
 *          Cuando la funcion vuelve, en ucParams[0] tenemos el LSB
 *          del valor leido y en ucParams[1] el MSB.
 *
 *    @return Devuelve el codigo de error
 */
int board_PacerReadCounter(unsigned char *ucParams);

/**
 *    Escribe un valor en el contador seleccionado.
 *
 *    @param ucParams Es un array de unsigned char's con el
 *          siguiente significado:
 *          - ucParams[0] = ID del contador a modificar. Puede
 *               ser uno de entre RM_PACER_SC_COUNTER0 a
 *               RM_PACER_SC_COUNTER2.
 *          - ucParams[1] = LSB del valor a escribir
```

```
 *          - ucParams[2] = MSB del valor a escribir
 *
 *    @return Devuelve el codigo de error
 */
int board_PacerWriteCounter(unsigned char *ucParams);

/**
 *    Configura uno de los contadores del Pacer.
 *
 *    @param ucParams Es un array de unsigned char's con el
 *          siguiente significado:
 *          - ucParams[0] = ID del contador a modificar. Puede
 *                ser uno de entre RM_PACER_SC_COUNTER0 a
 *                RM_PACER_SC_COUNTER2.
 *          - ucParams[1] = modo de lectura/escritura del valor, puede
 *                ser:
 *                - RM_PACER_RW_COUNTERLATCH
 *                - RM_PACER_RW_LSB
 *                - RM_PACER_RW_MSB
 *                - RM_PACER_RW_LSBMSB
 *          - ucParams[2] = modo de operacion del contador, puede ser
 *                un valor de entre RM_PACER_M_MODE0 a RM_PACER_M_MODE5.
 *          - ucParams[3] = modo de contaje en BCD (RM_PACER_BCD_BCDCOUNT)
 *                o en binario (RM_PACER_BCD_BINARYCOUNT)
 *
 *    @return Devuelve el codigo de error
 */
int board_PacerConfigureCounter(unsigned char *ucParams);

/**
 *    Lee el estado del contador seleccionado.
 *
 *    @param ucParams En ucParams[0] debe ir el ID del contador
 *          del que queremos leer su estado. Puede ser uno de entre
 *          RM_PACER_SC_COUNTER0 a RM_PACER_SC_COUNTER2.
 *          Cuando la funcion vuelve, en ucParams[0] tenemos el estado
 *          del contador que se haya especificado.
 *
 *    @return Devuelve el codigo de error
 */
int board_PacerReadCounterStatus(unsigned char *ucParams);

/**
 *    Lee el registro de control del Pacer.
 *
 *    @param ucParams En ucParams[0] se devuelve el valor que
 *          tiene el registro de control del Pacer.
 *
 *    @return Devuelve el codigo de error
 *    @see Para mas detalle,consultar con el manual de la tarjeta PCM3718-H
 */
int board_PacerReadControl(unsigned char *ucParams);

/**
 *    Escribe el registro de control del Pacer.
 *
 *    @param ucParams En ucParams[0] debe ir el valor que
 *          se quiere escribir en el regsitro de control del Pacer.
 *
 *    @return Devuelve el codigo de error
```

```
 *     @see Para mas detalle, consultar con el manual de la tarjeta
 *      PCM3718-H
 */
int board_PacerWriteControl(unsigned char *ucParams);

#endif
```

4.1.4 board.c

```
/**
 *     Implementacion de las funciones de acceso a bajo nivel
 *     a la PCM3718h
 *
 *     @file board.c
 *     @author Raul Jimenez
 *
 */

//#define _DEBUG_

/*
 *     Para printf
 */
#include <stdio.h>
#include <stdlib.h>

//#ifndef _DEBUG_
//    #define NULL ((long)0)
//#endif

#include <sys/neutrino.h>

/*
 *     Constantes para el uso de la PCM3718H y la comunicacion
 *     con el Resource Manager
 */
#include "../rm_pcm3718h/constants.h"

/* declaracion forward, incluirla en constants.h daria problemas de
declaracion ciclica */
int isVerbose();
int isExtTrigger();

/*
 *     Inclusion del archivo de cabecera propio
 */
#include "board.h"

/*
 *     Para el ISR
 */
#include "isr.h"

// Si DI0 es 'external trigger', forzamos ese bit a cero
#define CHECK_DI0(data) data = isExtTrigger() ? data & 0xFE : data;

/**
 * Ultimo estado leido desde la tarjeta
 *     [0]: DIGITAL_CHANNEL_A
```

```
 *    [1]: DIGITAL_CHANNEL_B
 */
int iChannelStatus[2];

int iStatus;      /**> ultimo valor leido del registro de estado */
int iControl;     /**> ultimo valor leido del registro de control */

int iEnablePacer; /**> es 0 si el contador esta inhabilitado; 1 para lo
                       contrario (ultimo valor leido) */
int iPacerControl; /**> ultimo valor leido del registro de control del
                       pacer */

int channelStart=0,channelStop=0; /* los canales de comienzo y fin del
                                     ciclo de conversion */

int pacerC1Status=RM_PACER_RW_LSBMSB | RM_PACER_M_MODE3 |
                  RM_PACER_BCD_BINARYCOUNT;
int pacerC2Status=RM_PACER_RW_LSBMSB | RM_PACER_M_MODE3 |
                  RM_PACER_BCD_BINARYCOUNT;

int pacerC1Val=2;
int pacerC2Val=2;

extern int iRWMode;     /* modo R/W del pacer */

#ifdef _DEBUG_
      int adChannel=0;
      int adValue=0;
#endif

//    guarda el ID de funcion ISR devuelto por InterruptAttach y necesario
//    para InterruptDetach
int irqID=-1;

////////////////////////////////////////////////////////
// Para los 'unsigned char * ucParams' se supone
// que apunta al sitio correcto !!!
////////////////////////////////////////////////////////

/*****************************************************
 *
 *    E/S DIGITAL
 *
 *****************************************************/

int board_SetDigBit(unsigned char * ucParams)
{
      int iChannel, iBitMask;

      if(isVerbose())
            printf("%s: > en board_SetDigBit\n",RM_DEV_NAME);

      if(ucParams==NULL)
            return RM_ERROR_BAD_ARGUMENT;

      iChannel=ucParams[0];
      iBitMask=ucParams[1];

      // Si DI0 es 'external trigger', forzamos ese bit a cero
      // (el bit menos significativo del canal A)
```

```
	if(iChannel==RM_DIG_CHANNEL_A)
		CHECK_DI0(iBitMask);

	if(iChannel!=RM_DIG_CHANNEL_A && iChannel!=RM_DIG_CHANNEL_B)
		return RM_ERROR_BAD_ARGUMENT;
	else
	{
		if(iChannel==RM_DIG_CHANNEL_A)
		{
			BYTE_SET_BIT(iChannelStatus[0],iBitMask);
			WRITE(3,iChannelStatus[0]);
		}
		else
		{
			BYTE_SET_BIT(iChannelStatus[1],iBitMask);
			WRITE(11,iChannelStatus[1]);
		}
		return RM_NO_ERROR;
	}
}

int board_ResetDigBit(unsigned char * ucParams)
{
	int iChannel, iBitMask;

	if(isVerbose())
		printf("%s: > en board_ResetDigBit\n",RM_DEV_NAME);

	if(ucParams==NULL)
		return RM_ERROR_BAD_ARGUMENT;

	iChannel=ucParams[0];
	iBitMask=ucParams[1];

	// Si DI0 es 'external trigger', forzamos ese bit a cero
	// (el bit menos significativo del canal A)
	if(iChannel==RM_DIG_CHANNEL_A)
		CHECK_DI0(iBitMask);

	if(iChannel!=RM_DIG_CHANNEL_A && iChannel!=RM_DIG_CHANNEL_B)
		return RM_ERROR_BAD_ARGUMENT;
	else
	{
		if(iChannel==RM_DIG_CHANNEL_A)
		{
			BYTE_RESET_BIT(iChannelStatus[0],iBitMask);
			WRITE(3,iChannelStatus[0]);
		}
		else
		{
			BYTE_RESET_BIT(iChannelStatus[1],iBitMask);
			WRITE(11,iChannelStatus[1]);
		}
		return RM_NO_ERROR;
	}
}

int board_ReadDigStatus(unsigned char * ucParams)
{
	if(isVerbose())
```

```
            printf("%s: > en board_ReadDigStatus\n",RM_DEV_NAME);

      if(ucParams==NULL)
            return RM_ERROR_BAD_ARGUMENT;

      iChannelStatus[RM_DIG_CHANNEL_A]=READ(3);
      iChannelStatus[RM_DIG_CHANNEL_B]=READ(11);

      if(ucParams[0]==1) ucParams[0]=iChannelStatus[RM_DIG_CHANNEL_A];
      if(ucParams[1]==1) ucParams[1]=iChannelStatus[RM_DIG_CHANNEL_B];

      return RM_NO_ERROR;
}

int board_WriteDigStatus(unsigned char *ucParams)
{
      unsigned char digA,digB;

      if(isVerbose())
            printf("%s: > en board_WriteDigStatus\n",RM_DEV_NAME);

      if(ucParams==NULL)
            return RM_ERROR_BAD_ARGUMENT;

      digA=ucParams[2];
      digB=ucParams[3];

      // Si DI0 es 'external trigger', forzamos ese bit a cero
      // (en este caso, el bit menos signif. del canal A)
      CHECK_DI0(digA);

      if(ucParams[0]==1)
      {
            iChannelStatus[RM_DIG_CHANNEL_A]=(int)digA;
            WRITE(3,digA);
      }

      if(ucParams[1]==1)
      {
            iChannelStatus[RM_DIG_CHANNEL_B]=(int)digB;
            WRITE(11,digB);
      }

      return RM_NO_ERROR;
}

int board_ReadDigInt(unsigned char * ucParams)
{
      int res;
      char params[2];

      if(isVerbose())
            printf("%s: > en board_ReadDigInt\n",RM_DEV_NAME);

      if(ucParams==NULL)
            return RM_ERROR_BAD_ARGUMENT;

      if((res=board_ReadDigStatus(params))!=RM_NO_ERROR)
            return res;
```

```
	// el canal A es el LSB del entero y el canal B es el MSB del entero
	ucParams[0]=params[0]; // LSB
	ucParams[1]=params[1]; // MSB

	return RM_NO_ERROR;
}

int board_WriteDigInt(unsigned char * ucParams)
{
	int res;
	char params[4];

	if(isVerbose())
		printf("%s: > en board_WriteDigInt\n",RM_DEV_NAME);

	if(ucParams==NULL)
		return RM_ERROR_BAD_ARGUMENT;

	// el canal A es el LSB del entero y el canal B es el MSB del entero
	params[0]=1;	// escribimos el canal A
	params[1]=1;	// escribimos el canal B
	params[2]=ucParams[0]; // LSB
	params[3]=ucParams[1]; // MSB

	// Si DI0 es 'external trigger', forzamos ese bit a cero
	// (el bit menos significativo del canal A)
	CHECK_DI0(params[2]);

	if((res=board_WriteDigStatus(params))!=RM_NO_ERROR)
		return res;

	return RM_NO_ERROR;
}

/*****************************************************
 *
 *	REGISTROS DE ESTADO Y CONTROL
 *
 *****************************************************/

int board_GetADStatusRegister(unsigned char *ucParams)
{
	if(isVerbose())
		printf("%s: > en board_GetADStatusRegister\n",RM_DEV_NAME);

//#ifdef _DEBUG_
//	iStatus=rand();
//	iStatus=RM_AD_STATUS_EOC_BIT; // para que se pueda simular la
//	                              // conversion A/D
//#endif

	if(ucParams==NULL)
		return RM_ERROR_BAD_ARGUMENT;

	iStatus=READ(8);

	ucParams[0]=iStatus;

	return RM_NO_ERROR;
}
```

```
int board_ClearInterruptRequest(unsigned char *ucParams)
{
      /* ucParams puede ser NULL ya que no se usa */

      if(isVerbose())
            printf("%s: > en board_ClearInterruptRequest\n",RM_DEV_NAME);

      /* escribimos cualquier valor en BASE + 8 */
      WRITE(8,(unsigned char)1);

      return RM_NO_ERROR;
}

int board_GetADControlRegister(unsigned char *ucParams)
{
      if(isVerbose())
            printf("%s: > en board_GetControlRegister\n",RM_DEV_NAME);

      iControl=(int)READ(9);

      if(ucParams==NULL)
            return RM_ERROR_BAD_ARGUMENT;

      ucParams[0]=iControl;

      return RM_NO_ERROR;
}

int board_SetADControlRegister(unsigned char *ucParams)
{
      int irq=-1;
      static int irqInstalled=0; // para recordar quitar el ISR cuando se
                                 // cambie de modo

      if(isVerbose())
            printf("%s: > en board_SetControlRegister\n",RM_DEV_NAME);

      if(ucParams==NULL)
            return RM_ERROR_BAD_ARGUMENT;

      iControl = APPLY_MASK(iControl,ucParams[0],(int)ucParams[1]);

      //    Antes de escribir comprobamos si se ha pedido el control
      //    de la conversion A/D por IRQ y, en ese caso, instalamos
      //    el ISR correspondiente
      if(iControl & RM_AD_CONTROL_INTE_BIT)
      {
            irq= (iControl & RM_AD_CONTROL_INTLEVEL_BIT) >> 4;

            if((irqID=isrInstall(irq))==-1)
                  return RM_ERROR_INSTALL_ISR;

            irqInstalled=1;
      }
      else if(irqInstalled==1) // hay que quitar el ISR
      {
            if(isrUninstall()==-1)
                  return RM_ERROR_UNINSTALL_ISR;
```

```
            irqInstalled=0;
      }

      //    escribimos el registro de control
      WRITE(9,(unsigned char)iControl);

      return RM_NO_ERROR;
}

/*******************************************************
 *
 *    CONVERSION A/D
 *
 *******************************************************/

//    Comprueba que se puede accerder al canal iChannel
//    verificando que queda dentro del rango especificado
//    en el registro de control
//    (supongo que iOK != NULL )
int checkChannelConfig(int iChannel,int *iOK)
{
      int res;
      char param[1];

      if((res=board_GetADStatusRegister(param))!=RM_NO_ERROR)
      {
            *iOK=0;
            return res;
      }

      // si son 8-diff hay que comprobar que 0<= iChannel <= 7
      if((param[0] & RM_AD_STATUS_MUX_BIT) == RM_AD_STATUS_MUX_8DIFF)
      {
            if(iChannel<0 || iChannel>7)
            {
                  *iOK=0;
                  return RM_ERROR_CONFIGURATION;
            }
            else
            {
                  *iOK=1;
                  return RM_NO_ERROR;
            }
      }
      else // si son 16-se hay que comprobar que 0<= iChannel <= 15
      {
            if(iChannel<0 || iChannel>15)
            {
                  *iOK=0;
                  return RM_ERROR_CONFIGURATION;
            }
            else
            {
                  *iOK=1;
                  return RM_NO_ERROR;
            }
      }
}

int board_ReadADData(unsigned char * ucParams)
```

```
{
	int data;
	int base0,base1;  // que se leeran resp de BASE+0 y BASE+1

	if(isVerbose())
		printf("%s: > en board_ReadADData\n",RM_DEV_NAME);

	// leemos de BASE+0 y BASE+1
	base0 = READ(0);
	base1 = READ(1);

	// construimos el dato
	data=((int)base1)<< 4 | ((int)base0)>>4;

	ucParams[0]=base0 & LOW_NIBBLE;  // canal al que corresponde el dato
	ucParams[1]=LSB(data);
	ucParams[2]=MSB(data);

	return RM_NO_ERROR;
}

int board_TriggerConversion(unsigned char * ucParams)
{
	// ucParams puede ser NULL ya que no se usa

	// hay que comprobar primero que esta configurado
	// el triggering por software

	char params[1];
	int res;

	if(isVerbose())
		printf("%s: > en board_TriggerConversion\n",RM_DEV_NAME);

	if((res=board_GetADControlRegister(params))!=RM_NO_ERROR)
		return res;

	if((params[0] & RM_AD_CONTROL_TRIGSRC_BIT) ==
	   RM_AD_CONTROL_TRIGSRC_SOFTW)
	{
		// si esta seleccionado el triggering por software, escribiendo
		// cualquier valor en BASE+0 lanzamos una conversion
		WRITE(0,(unsigned char)1);
	}
	else
		return RM_ERROR_CONFIGURATION;

	return RM_NO_ERROR;
}

int board_SetADRange(unsigned char * ucParams)
{
	int channel,range,ok,res;
	char paramsR[2], paramsW[2];

	if(isVerbose())
		printf("%s: > en board_SetADRange\n",RM_DEV_NAME);

	if(ucParams==NULL)
		return RM_ERROR_BAD_ARGUMENT;
```

```
	//	primero chequeamos que la tarjeta este configurada
	//	para el canal seleccionado

	channel=((int)ucParams[0]) & RM_AD_CHANNEL_MASK;
	if((res=checkChannelConfig(channel,&ok))!=RM_NO_ERROR)
		return RM_ERROR_BAD_ARGUMENT;

	if(!ok)
		return RM_ERROR_CONFIGURATION;

	range=((int)ucParams[1]) & RM_AD_RANGE_MASK;
	if(range<RM_AD_RANGE_MIN || range>RM_AD_RANGE_MAX)
		return RM_ERROR_BAD_ARGUMENT;

	//	Primero leemos el MUX scan para 'backup' del start channel
	if((res=board_ReadADMuxScan(paramsR))!=RM_NO_ERROR)
		return res;

	//	Ahora hacemos que channel sea el start channel
	paramsW[0]=channel;
	paramsW[1]=paramsR[1];
	if((res=board_WriteADMuxScan(paramsR))!=RM_NO_ERROR)
		return res;

	//	y establecemos el rango escribiendo en BASE+1
	WRITE(1,(unsigned char)range);

	//	Y volvemos a dejar el start channel como estaba
	paramsW[0]=paramsR[0];
	paramsW[1]=paramsR[1];
	if((res=board_WriteADMuxScan(paramsR))!=RM_NO_ERROR)
		return res;

	if(isVerbose())
		printf("%s: > board_SetADRange > ch:%d
		 r:%d\n",RM_DEV_NAME,channel,range);

	return RM_NO_ERROR;
}

int board_ReadADMuxScan(unsigned char * ucParams)
{
	/*	ucParams puede ser NULL */

	unsigned char base2;

	if(isVerbose())
		printf("%s: > en board_ReadADMuxScan\n",RM_DEV_NAME);

	//	Leemos de la tarjeta (BASE+2)
	base2=READ(2);

	ucParams[0]=base2 & LOW_NIBBLE;		/* start channel */
	ucParams[1]=(base2 & HIGH_NIBBLE)>>4;	/* stop channel */

	return RM_NO_ERROR;
}

int board_WriteADMuxScan(unsigned char * ucParams)
```

```
{
      int res,ok;
//    int channelStart,channelStop;

      if(isVerbose())
            printf("%s: > en board_WriteADMuxScan\n",RM_DEV_NAME);

      if(ucParams==NULL)
            return RM_ERROR_BAD_ARGUMENT;

      // primero chequeamos que la tarjeta este configurada
      // para los canales seleccionados

      channelStart=((int)ucParams[0]) & RM_AD_CHANNEL_MASK;
      if((res=checkChannelConfig(channelStart,&ok))!=RM_NO_ERROR)
            return res;

      if(!ok)
            return RM_ERROR_CONFIGURATION;

      channelStop=((int)ucParams[1]) & RM_AD_CHANNEL_MASK;
      if((res=checkChannelConfig(channelStop,&ok))!=RM_NO_ERROR)
            return res;

      if(!ok)
            return RM_ERROR_CONFIGURATION;

      // ahora escribimos en BASE+2
      WRITE(2,((channelStop & LOW_NIBBLE)<<4) | (channelStart &
            LOW_NIBBLE));

      if(isVerbose())
            printf("%s: > board_WriteADMuxScan > chStart:%d
             chStop:%d\n",RM_DEV_NAME,channelStart,channelStop);

      return RM_NO_ERROR;
}


/*****************************************************
 *
 *    CONTADOR
 *
 *****************************************************/

int board_EnableDisablePacer(unsigned char *ucParams)
{
      if(isVerbose())
            printf("%s: > en board_EnableDisablePacer\n",RM_DEV_NAME);

      if(ucParams==NULL)
            return RM_ERROR_BAD_ARGUMENT;

      iEnablePacer= ucParams[0]==1 ? RM_PACER_ENABLE : RM_PACER_DISABLE;

      // escribimos en BASE+10
      WRITE(10,(unsigned char)iEnablePacer);

      return RM_NO_ERROR;
}
```

```
int board_GetPacerEnabled(unsigned char *ucParams)
{
//    unsigned char en=0;

      if(isVerbose())
            printf("%s: > en board_GetPacerEnabled\n",RM_DEV_NAME);

      if(ucParams==NULL)
            return RM_ERROR_BAD_ARGUMENT;

      // no podemos leer de BASE+10, usamos la variable global
      // iEnablePacer
//    en=READ(10);

      ucParams[0]=LOW_BYTE(iEnablePacer);

      return RM_NO_ERROR;
}

int board_PacerReadCounter(unsigned char *ucParams)
{
      int counterID,control,controlOld,res,desp;
      int auxLSB=0,auxMSB=0;
      char paramsR[1],paramsW[1];

      if(isVerbose())
            printf("%s: > en board_PacerReadCounter\n",RM_DEV_NAME);

      if(ucParams==NULL)
            return RM_ERROR_BAD_ARGUMENT;

      counterID=(int)ucParams[0] & RM_PACER_SC_MASK;

      // primero hacemos una 'copia de seguridad' del registro de control
      if((res=board_PacerReadControl(paramsR))!=RM_NO_ERROR)
            return res;
      controlOld=(int)paramsR[0];

      // ahora hacemos un read-back command para latchear el valor del
      // contador
      control=0;
      control=RM_PACER_SC_READBACK_COMMAND | RM_PACER_CONTROL_STA_BIT;
      switch(counterID)
      {
            case RM_PACER_SC_COUNTER0:
                  control |= RM_PACER_CONTROL_COUNTER0_BIT;
                  desp=12;
                  break;
            case RM_PACER_SC_COUNTER1:
                  control |= RM_PACER_CONTROL_COUNTER1_BIT;
                  desp=13;
                  break;
            case RM_PACER_SC_COUNTER2:
                  control |= RM_PACER_CONTROL_COUNTER2_BIT;
                  desp=14;
                  break;

            default:
                  return RM_ERROR_BAD_ARGUMENT;
```

```
	}

	paramsW[0]=(char)control;
	if((res=board_PacerWriteControl(paramsW))!=RM_NO_ERROR)
		return res;

	// ahora leemos el valor del contador
	auxLSB=0;
	auxMSB=0;
	if(iRWMode==RM_PACER_RW_LSBMSB)
	{
		auxLSB=READ(desp);
		auxMSB=READ(desp);
	}
	else if(iRWMode==RM_PACER_RW_MSB)
		auxMSB=READ(desp);
	else /* en caso contrario es el modo LSB */
		auxLSB=READ(desp);

	ucParams[0]=auxLSB;
	ucParams[1]=auxMSB;

	// ... y vovemos a dejar el registro de control como estaba
	paramsW[0]=(char)controlOld;
	if((res=board_PacerWriteControl(paramsW))!=RM_NO_ERROR)
		return res;

	return RM_NO_ERROR;
}

int board_PacerWriteCounter(unsigned char *ucParams)
{
	int counterID,control,res,desp;
	int auxLSB,auxMSB;
	char paramsR[1];

	if(isVerbose())
		printf("%s: > en board_PacerWriteCounter\n",RM_DEV_NAME);

	if(ucParams==NULL)
		return RM_ERROR_BAD_ARGUMENT;

	counterID   = (int)ucParams[0] & RM_PACER_SC_MASK;
	auxLSB      = (int)ucParams[1];
	auxMSB      = (int)ucParams[2];

	if(isVerbose())
		printf("%s: write counter > counterID:%d lsb:%d
		 msb:%d\n",RM_DEV_NAME,counterID,auxLSB,auxMSB);

	// antes de escribir comprobamos que no estamos en read-back command
	if((res=board_PacerReadControl(paramsR))!=RM_NO_ERROR)
		return res;
	control=(int)paramsR[0];

	if(control & RM_PACER_SC_READBACK_COMMAND)
		return RM_ERROR_EXECUTING_COMMAND;

	//    comprobamos el contador seleccionado
	switch(counterID)
```

```
	{
		case RM_PACER_SC_COUNTER0: desp=12; break;
		case RM_PACER_SC_COUNTER1: desp=13; break;
		case RM_PACER_SC_COUNTER2: desp=14; break;

		default:
			return RM_ERROR_BAD_ARGUMENT;
	}

	//	ahora escribimos el valor del contador
	if(iRWMode==RM_PACER_RW_LSBMSB)
	{
		WRITE(desp,auxLSB);
		WRITE(desp,auxMSB);
	}
	else if(iRWMode==RM_PACER_RW_MSB)
		WRITE(desp,auxMSB);
	else /* en caso contrario es el modo LSB */
		WRITE(desp,auxLSB);

	return RM_NO_ERROR;
}

int board_PacerConfigureCounter(unsigned char *ucParams)
{
	int counterID,control,controlOld,res;
	int rwMode,opMode,bcdMode;
	char paramsR[1],paramsW[1];

	if(isVerbose())
		printf("%s: > en board_PacerConfigureCounter\n",RM_DEV_NAME);

	if(ucParams==NULL)
		return RM_ERROR_BAD_ARGUMENT;

	counterID   = (int)ucParams[0] & RM_PACER_SC_MASK;
	rwMode      = (int)ucParams[1] & RM_PACER_RW_MASK;
	opMode      = (int)ucParams[2] & RM_PACER_M_MASK;
	bcdMode     = (int)ucParams[3] & RM_PACER_BCD_MASK;

	// comprobamos que el contador es el 1 o el 2 (unicos libres para el
	// usuario)
	if(counterID!=RM_PACER_SC_COUNTER1 &&
	 counterID!= RM_PACER_SC_COUNTER2)
		return RM_ERROR_BAD_ARGUMENT;

	//	lo guardamos para recordarlo luego
	iRWMode=rwMode;

	// antes de configurar comprobamos que no estamos en read-back
	// command (de paso, hacemos una 'copia de seguridad' del registro de
	// control)
	if((res=board_PacerReadControl(paramsR))!=RM_NO_ERROR)
		return res;
	controlOld=(int)paramsR[0];

	if(controlOld & RM_PACER_SC_READBACK_COMMAND)
		return RM_ERROR_EXECUTING_COMMAND;

	// ahora configuramos el canal seleccionado
```

```
	control=0;
	control=counterID | rwMode | opMode | bcdMode;
	paramsW[0]=(char)control;
	if((res=board_PacerWriteControl(paramsW))!=RM_NO_ERROR)
		return res;

	// ... y vovemos a dejar el registro de control como estaba
	paramsW[0]=(char)controlOld;
	if((res=board_PacerWriteControl(paramsW))!=RM_NO_ERROR)
		return res;

	return RM_NO_ERROR;
}

int board_PacerReadCounterStatus(unsigned char *ucParams)
{
	int counterID,control,controlOld,res,desp;
	int status=0;
	char paramsR[1],paramsW[1];

	if(isVerbose())
		printf("%s: > en board_PacerReadCounterStatus\n",RM_DEV_NAME);

	if(ucParams==NULL)
		return RM_ERROR_BAD_ARGUMENT;

	counterID=(int)ucParams[0] & RM_PACER_SC_MASK;

	// primero hacemos una 'copia de seguridad' del registro de control
	if((res=board_PacerReadControl(paramsR))!=RM_NO_ERROR)
		return res;
	controlOld=(int)paramsR[0];

	// ahora hacemos un read-back command para latchear el status del
	// contador
	control=0;
	control=RM_PACER_SC_READBACK_COMMAND | RM_PACER_CONTROL_CNT_BIT;
	switch(counterID)
	{
		case RM_PACER_SC_COUNTER0:
			control |= RM_PACER_CONTROL_COUNTER0_BIT;
			desp=12;
			break;
		case RM_PACER_SC_COUNTER1:
			control |= RM_PACER_CONTROL_COUNTER1_BIT;
			desp=13;
			break;
		case RM_PACER_SC_COUNTER2:
			control |= RM_PACER_CONTROL_COUNTER2_BIT;
			desp=14;
			break;

		default:
			return RM_ERROR_BAD_ARGUMENT;
	}

	paramsW[0]=(char)control;
	if((res=board_PacerWriteControl(paramsW))!=RM_NO_ERROR)
		return res;
```

```
	// ahora leemos el valor del estado del contador
	status=READ(desp);

	ucParams[0]=status;

	// ... y vovemos a dejar el registro de control como estaba
	paramsW[0]=(char)controlOld;
	if((res=board_PacerWriteControl(paramsW))!=RM_NO_ERROR)
		return res;

	return RM_NO_ERROR;
}

int board_PacerReadControl(unsigned char *ucParams)
{
	if(isVerbose())
		printf("%s: > en board_PacerReadControl\n",RM_DEV_NAME);

	if(ucParams==NULL)
		return RM_ERROR_BAD_ARGUMENT;

	iPacerControl=(int)READ(15);
	ucParams[0]=iPacerControl;

	return RM_NO_ERROR;
}

int board_PacerWriteControl(unsigned char *ucParams)
{
	if(isVerbose())
		printf("%s: > en board_PacerWriteControl\n",RM_DEV_NAME);

	if(ucParams==NULL)
		return RM_ERROR_BAD_ARGUMENT;

	iPacerControl=ucParams[0];
	WRITE(15,(unsigned char)iPacerControl);

	return RM_NO_ERROR;
}
```

4.1.5 isr.h

```
/**
 *    Funciones asociadas a la conversion A/D por interrupciones
 *
 *    @file isr.h
 *    @author Raul Jimenez Ruiz
 */

#include <signal.h>
#include <sys/siginfo.h>
#include <sys/neutrino.h>

/**
 *    Manejador de la interrupcion que se dispara tras terminar
 *    una captura A/D.
 *
```

```
 *    @param pvArea No se usa
 *    @param iId No se usa
 *
 *    @return Devuelve una estructura sigevent inicializada en
 *     el hilo de captura A/D por interrupciones mediante la
 *     macro SIGEV_INTR_INIT(&event)
 *
 *    @see isrInstall, isrUninstall
 */
const struct sigevent * irqHandler(void * pvArea,int iId);

/**
 *    Lanza el hilo de captura A/D por interrupciones, el cual,
 *    a su vez, instala el ISR.
 *
 *    @param iIrq Es el numero de interrupcion a usar
 *
 *    @return Devuelve 1 si todo va bien y -1 si se produce algun error
 */
int isrInstall(int iIrq);

/**
 *    Desinstala el ISR (marca la terminacion del hilo de captura A/D por
 *    interrupciones y luego hace un pthread_join)
 *
 *    @return Devuelve 1 si todo va bien y -1 si se produce algun error
 */
int isrUninstall(void);
```

4.1.6 isr.c

```
/**
 *    Implementa las funciones asociadas a la conversion A/D por
 *    interrupciones
 *
 *    @file isr.c
 *    @author Raul Jimenez Ruiz
 */

#include <pthread.h>
#include <sys/neutrino.h>
#include <errno.h>

#include <stdio.h>

#include "isr.h"
#include "board.h"

//    de rm_pcm3718h.c
int signalToUser(int iChannel, int iData);

/**
 * Flag para notificar al thread que debe lanzar una señal al usuario
 */
volatile int isrFlagSignal=0;

/**
 * Flag para notificar al thread que debe finalizar
 */
```

```
volatile int isrFlagToEnd=0;

/**
 * Datos de la ultima conversion A/D, que escribe el ISR y posteriormente
 * lee el thread de envio de señales.
 */
volatile int isrCanal,isrData;

/**
 * ID del thread
 */
pthread_t isrThreadID;

/**
 * ID de la IRQ
 */
volatile int irqID;

/**
 * ID de la funcion asociado a la interrupcion
 */
volatile int irqFuncID;

/**
 * Evento que devolvera el ISR
 */
struct sigevent event;

/*
 *    Codigo del handler de la IRQ
 */
const struct sigevent * isrIRQHandler(void * pvArea,int iId)
{
      int base0,base1;  // que se leeran resp de BASE+0 y BASE+1

      //    leemos del hardware
      base0 = READ(0);
      base1 = READ(1);

      //    construimos la informacion
      isrCanal=base0 & 0x0F;
      isrData=((int)base1)<< 4 | ((int)base0)>>4;

      //    bloqueamos la irq (para que el PIC no vuelva a interrumpir
      //    inmediatamente al kernel)
      if(InterruptMask(irqID,irqFuncID)==-1)
            return NULL;

      //    como aqui no se puede llamar a 'sigqueue' (que se llama desde
      //    'signalToUser'), ponemos un flag a 1 para que un hilo en
      //    paralelo sea el que envie la señal
      isrFlagSignal=1;

      return (&event);
}

//
//    Thread para lanzar señales al usuario
//
void * isrSignalThread(void *pvParams)
```

```
{
	//	parametrizamos el evento que devolvera el ISR
	SIGEV_INTR_INIT(&event);

	//	instalamos el ISR
	if(ThreadCtl(_NTO_TCTL_IO,0)==-1)
		return NULL;
	if((irqFuncID=InterruptAttach (irqID,isrIRQHandler,NULL,0,0))==-1)
		return NULL;

	//	bucle principal de captura
	while(isrFlagToEnd==0)
	{
		//	esperamos a que se dispare una interrupcion
		if(InterruptWait(0,NULL)==-1)
			return NULL;

		//	mandamos los datos leidos usando una señal RT
		if(signalToUser(isrCanal,isrData)==1)
			// OK
		else
		{
			// FALLO
			break;
		}

		//	limpiamos el bit INT de la tarjeta
		board_ClearInterruptRequest(NULL);

		//	desbloqueamos la irq
		if(ThreadCtl(_NTO_TCTL_IO,0)==-1)
			return NULL;
		if(InterruptUnmask(irqID,irqFuncID)==-1)
			return NULL;
	}

	//	desinstalamos el ISR
	if(ThreadCtl(_NTO_TCTL_IO,0)==-1)
		return NULL;
	if(InterruptDetach(irqID)==-1)
		return NULL;

	return NULL;
}

/*
 *	Lanza el hilo de captura (e instala el ISR)
 */
int isrInstall(int iIrq)
{
	irqID=iIrq;

	//	lanzamos el thread
	if(pthread_create(&isrThreadID,NULL,isrSignalThread,NULL)!=EOK)
		return -1;
	else
		return 1;
}

/*
```

```
 *    Para el hilo de captura (y desinstala el ISR )
 */
int isrUninstall(void)
{
      //    paramos el thread
      isrFlagToEnd=1;
      if(pthread_join(isrThreadID,NULL)!=EOK)
            return -1;
      else
            return 1;
}
```

4.2 Librería compartida.

4.2.1 devio-pcm3718h.h

```
/**
 *    Declara prototipos de las funciones publicas de la libreria
 *    compartida.
 *
 *    @file devio-pcm3718h.h
 *    @author Raul Jimenez Ruiz
 */
#ifndef _DEVIO_PCM3718H_
#define _DEVIO_PCM3718H_

//#ifdef _SHARED_LIB_CODE_
//    #define EXTERN
//#else
//    #define EXTERN extern
//    #define EXTERN
//#endif

/*******************************************************
 *    UTILIDADES
 *
 *******************************************************/

/*
 *    Lista de errores de la libreria
 */
#define SO_NO_ERROR                        ((int)1000)
      /**> No ha ocurrido ningun error  */

#define SO_ERROR_OPN_DEV                   ((int)1001)
      /**> Error abriendo el dispositivo en /dev/pcm3718h     */

#define SO_ERROR_MSG_SEND                  ((int)1002)
      /**> Error enviando mensaje al Resource Manager         */

#define SO_ERROR_BAD_ARGUMENT      ((int)1003)
      /**> Error en el argumento de funcion                   */

#define SO_ERROR_EXECUTING_COMMAND  ((int)1004)
      /**> Error ejecutando el comando en el Resource Manager */

#define SO_ERROR_SYSTEM                    ((int)1005)
```

```
      /**> Error de sistema                                        */

#define SO_ERROR_TIME_EXCEEDED           ((int)1006)
      /**> Error, tiempo excedido en la espera                     */

#define SO_ERROR_CONFIGURATION           ((int)1007)
      /**> Error de configuracion del dispositivo                  */

/**
 *    Macro para determinar si un error devuelto por una funcion dada
 *    es debido a la libreria compartida (SO_ERROR_...) o al Resource
 *    Manager (RM_ERROR_...).
 */
#define IS_SO_ERROR(e) ((e)>=SO_NO_ERROR)

/**
 *    Convierte un codigo de error en un mensaje descriptivo.
 *
 *    @param err Codigo de error.
 *
 *    @return Devuelve una cadena de texto (const char *) con el
 *          mensaje que describe el error.
 */
const char * errorToString(int err);

/**
 *    Convierte un codigo de error generado por el Resource Manager
 *    en un mensaje descriptivo.
 *
 *    @param err Codigo de error.
 *
 *    @return Devuelve una cadena de texto (const char *) con el
 *          mensaje que describe el error.
 */
const char * RMErrorToString(int err);

/**
 *    Convierte un codigo de error generado por la libreria en
 *    un mensaje descriptivo.
 *
 *    @param err Codigo de error.
 *
 *    @return Devuelve una cadena de texto (const char *) con el
 *          mensaje que describe el error.
 */
const char * SOErrorToString(int err);

/*******************************************************
 *
 *    E/S DIGITAL
 *
 *******************************************************/

/**
 *    Pone a 1 los bits indicados en el canal seleccionado.
 *
 *    @param iChannel Canal seleccionado.
 *    @param iBitMask Mascara de bits que indica aquellos que
 *          seran puestos a 1.
 *
```

```
 *    @return Devuelve SO_NO_ERROR si todo va bien, si no, devuelve
 *          un codigo de error.
 *
 *    @see constants.h (contiene codigos de error y parametros).
 */
 int pcm3718h_SetDigBit(int iChannel,int iBitMask);

/**
 *  Pone a 0 los bits indicados en el canal seleccionado.
 *
 *    @param iChannel Canal seleccionado.
 *    @param iBitMask Mascara de bits que indica aquellos que
 *          seran puestos a 0.
 *
 *    @return Devuelve SO_NO_ERROR si todo va bien, si no, devuelve
 *          un codigo de error.
 *
 *    @see constants.h (contiene codigos de error y parametros).
 */
int pcm3718h_ResetDigBit(int iChannel,int iBitMask);

/**
 *    Lee el estado de ambos canales de entradas digitales.
 *
 *    @param piChannelA Puntero a entero donde se guardara, en el
 *          byte bajo, el estado del canal A. Puede ser NULL.
 *    @param piChannelB Puntero a entero donde se guardara, en el
 *          byte bajo, el estado del canal B. Puede ser NULL.
 *
 *    @return Devuelve SO_NO_ERROR si todo va bien, si no, devuelve
 *          un codigo de error.
 *
 *    @see constants.h (contiene codigos de error y parametros).
 */
int pcm3718h_ReadDigStatus(int *piChannelA,int *piChannelB);

/**
 *    Escribe todas las entradas de ambos canales al mismo tiempo.
 *
 *    @param iChannelA Estado del canal A (en el byte bajo). Puede
 *          ser RM_NULLPARAM si no se quiere escribir nada.
 *    @param iChannelB Estado del canal B (en el byte bajo). Puede
 *          ser RM_NULLPARAM si no se quiere escribir nada.
 *
 *    @return Devuelve SO_NO_ERROR si todo va bien, si no, devuelve
 *          un codigo de error.
 *
 *    @see constants.h (contiene codigos de error y parametros).
 */
int pcm3718h_WriteDigStatus(int iChannelA, int iChannelB);

/**
 *    Lee los dos canales de 8 bits de entradas digitales como
 *    si fueran un entero de 16 bits.
 *
 *    @param piDigI Puntero a entero donde se guardara el entero de
 *          16 bits leido de las entradas digitales.
 *
 *    @return Devuelve SO_NO_ERROR si todo va bien, si no, devuelve
 *          un codigo de error.
```

```
 *
 *    @see constants.h (contiene codigos de error y parametros).
 */
int pcm3718h_ReadDigInt(int *piDigI);

/**
 *    Escribe los dos canales de 8 bits de salidas digitales como
 *    si fueran un entero de 16 bits.
 *
 *    @return Devuelve SO_NO_ERROR si todo va bien, si no, devuelve
 *          un codigo de error.
 *
 *    @see constants.h (contiene codigos de error y parametros).
 */
int pcm3718h_WriteDigInt(int iDigO);

/*****************************************************
 *
 *    CONVERSION A/D
 *
 *****************************************************/

/**
 *    Lee el dato proveniente de una conversion A/D.
 *
 *    @param piChannel Puntero a entero donde se guarda el canal
 *          en el que se ha realizado la conversion A/D.
 *    @param piData Puntero a entero donde se guarda el valor del
 *          dato proveniente de la conversion A/D.
 *
 *    @return Devuelve SO_NO_ERROR si todo va bien, si no, devuelve
 *          un codigo de error.
 *
 *    @see constants.h (contiene codigos de error y parametros).
 */
int pcm3718h_ReadADData(int *piChannel, int *piData);

/**
 *    Lanza una conversion A/D.
 *
 *    @param void
 *
 *    @return Devuelve SO_NO_ERROR si todo va bien, si no, devuelve
 *          un codigo de error.
 *
 *    @see constants.h (contiene codigos de error y parametros).
 */
int pcm3718h_TriggerConversion();

/**
 *    Establece el rango de entrada de uno de los canales analogicos.
 *
 *    @param iChannel ID del canal del que se quiere configurar el rango.
 *          El canal debera quedar dentro del rango configurado en la
 *          tarjeta (RM_AD_CHANNEL0..RM_AD_CHANNEL7 para '8 differential' o
 *          RM_AD_CHANNEL0..RM_AD_CHANNEL15 para '16 single-ended').
 *    @param iRange Rango a establecer en el canal, que se corresponde con
 *          la siguiente tabla:
 *                Rango                              Valores
 *                RM_AD_RANGE_BIPOLAR_10_0      -10     .. +10 V
```

```
 *                RM_AD_RANGE_BIPOLAR_5_0        -5      ..  +5 V
 *                RM_AD_RANGE_BIPOLAR_2_5        -2.5    ..  +2.5 V
 *                RM_AD_RANGE_BIPOLAR_1_25       -1.25   ..  +1.25 V
 *                RM_AD_RANGE_BIPOLAR_0_625      -0.625  ..  +0.625 V
 *                RM_AD_RANGE_UNIPOLAR_10_0       0      .. +10 V
 *                RM_AD_RANGE_UNIPOLAR_5_0        0      ..  +5 V
 *                RM_AD_RANGE_UNIPOLAR_2_5        0      ..  +2.5V
 *                RM_AD_RANGE_UNIPOLAR_1_25       0      ..  +1.25 V
 *
 *    @return Devuelve SO_NO_ERROR si todo va bien, si no, devuelve
 *          un codigo de error.
 *
 *    @see constants.h (contiene codigos de error y parametros).
 */
int pcm3718h_SetADRange(int iChannel, int iRange);

/**
 *    Lee el registro de multiplexado del bloque de conversion A/D, que
 *    contiene el canal de comienzo del ciclo de la conversion y el canal
 *    de finalizacion.
 *
 *    @param piStartChannel Puntero a entero donde se guardara el ID del
 *          canal de comienzo del ciclo de conversion A/D.
 *    @param piStopChannel Puentero a entero donde se guardara el ID del
 *          canal de finalizacion del ciclo de conversion a/D.
 *
 *    @return Devuelve SO_NO_ERROR si todo va bien, si no, devuelve
 *          un codigo de error.
 *
 *    @see constants.h (contiene codigos de error y parametros).
 */
int pcm3718h_ReadADMuxScan(int *piStartChannel, int *piStopChannel);

/**
 *    Escribe el registro de multiplexado del bloque de conversion A/D
 *    con el canal de comienzo del ciclo de conversion y el canal de
 *    finalizacion.Los canales deberan quedar dentro del rango configurado
 *    en la tarjeta (RM_AD_CHANNEL0..RM_AD_CHANNEL7 para '8 differential' o
 *    RM_AD_CHANNEL0..RM_AD_CHANNEL15 para '16 single-ended').
 *
 *    @param iStartChannel ID del canal de comienzo del ciclo de conversion
 *    @param iStopChannel ID del canal de finalizacion del ciclo de
 *             conversion
 *
 *    @return Devuelve SO_NO_ERROR si todo va bien, si no, devuelve
 *    un codigo de error.
 *
 *    @see constants.h (contiene codigos de error y parametros).
 */
int pcm3718h_WriteADMuxScan(int iStartChannel, int iStopChannel);

/**
 *    Hace que el hilo que llama a esta funcion espere a la finalizacion
 *    de una conversion A/D.
 *
 *    @param ulMs Especifica los milisegundos que debe esperar como maximo
 *          el fin de la conversion. Si ulMS=0, entonces espera
 *         indefinidamente.
 *
 *    @return Devuelve SO_NO_ERROR si todo va bien, si no, devuelve
```

```
 *          un codigo de error o SO_ERROR_TIME_EXCEEDED si se ha excedido
 *          el tiempo de espera.
 *
 *    @see constants.h (contiene codigos de error y parametros).
 */
int pcm3718h_WaitToEndOfConversion(unsigned long ulMs);

/*****************************************************
 *
 *    ESTADO / CONTROL
 *
 *****************************************************/

/**
 *    Lee el registro de estado de la conversion A/D, que es un
 *    byte que tiene el siguiente significado:
 *
 *          -MSB-                                -LSB-
 *           7    6    5    4    3    2    1    0
 *          EOC   X   MUX  INT  CN3  CN2  CN1  CN0
 *
 *    Para mas informacion acudir al manual de usuario de la tarjeta.
 *
 *    @params piADStatusRegister Puntero al entero donde se guardara,
 *               en el byte bajo, el registro de estado de la conversion
 *               A/D.
 *
 *    @return Devuelve SO_NO_ERROR si todo va bien, si no, devuelve
 *          un codigo de error.
 *
 *    @see constants.h (contiene codigos de error y parametros).
 */
int pcm3718h_ReadADStatus(int *piADStatusRegister);

/**
 *    Borra el bit INT del registro de estado.
 *    Para mas informacion acudir al manual de usuario de la tarjeta.
 *
 *    @params void
 *
 *    @return Devuelve SO_NO_ERROR si todo va bien, si no, devuelve
 *          un codigo de error.
 *
 *    @see constants.h (contiene codigos de error y parametros).
 */
int pcm3718h_ClearInterruptRequest(void);

/**
 *    Lee el registro de control de la conversion A/D, que es un
 *    byte que tiene el siguiente significado:
 *
 *          -MSB-                                -LSB-
 *           7    6    5    4    3    2    1    0
 *          INTE  I2   I1   I0   X  DMAE  ST1  ST0
 *
 *    Para mas informacion acudir al manual de usuario de la tarjeta.
 *
 *    @params piADControlRegister Puntero al entero donde se guardara,
 *               en el byte bajo, el registro de control de la conversion
A/D.
```

```
 *
 *    @return Devuelve SO_NO_ERROR si todo va bien, si no, devuelve
 *          un codigo de error.
 *
 *    @see constants.h (contiene codigos de error y parametros).
 */
int pcm3718h_ReadADControl(int *piADControlRegister);

/**
 *    Escribe bits en el registro de control de la conversion A/D,que es un
 *    byte que tiene el siguiente significado:
 *
 *          -MSB-                                -LSB-
 *           7    6    5    4    3    2    1    0
 *          INTE  I2   I1   I0   X  DMAE  ST1  ST0
 *
 *    Para mas informacion acudir al manual de usuario de la tarjeta.
 *
 *    @params iControlBits Contiene los bits (usando el operador '|' con
 *          los posibles parametros) a escribir en el resgistro de
 *          control. Si en iMask se especifica:
 *          - RM_AD_CONTROL_INTE_BIT | RM_AD_CONTROL_INTLEVEL_BIT, entonces
 *            puede contener  uno de RM_AD_CONTROL_IRQ2 a
 *            M_AD_CONTROL_IRQ7.
 *          - RM_AD_CONTROL_TRIGSRC_BIT, entonces puede contener uno de los
 *            siguientes:
 *                - RM_AD_CONTROL_TRIGSRC_SOFTW indica que el 'triggering'
 *                  es manual
 *                - RM_AD_CONTROL_TRIGSRC_EXT indica que el
 *                  'triggering' es mediante pulso externo
 *                - RM_AD_CONTROL_TRIGSRC_PACER indica que el 'triggering'
 *                  lo provee el contador integrado
 *
 *    @params iMask Contiene una mascara que indica los bits a cambiar.
 *            Debe ser uno o varios concatenados con el operador '|' de
 *            entre los siguientes:
 *          - RM_AD_CONTROL_INTE_BIT para indicar que se van a
 *            usar interrupciones.
 *          - RM_AD_CONTROL_INTLEVEL_BIT  para indicar que en iControlBits
 *            esta el numero  de la interrupcion que se va a usar.
 *          - RM_AD_CONTROL_DMAE_BIT para indicar que se va a usar DMA.
 *          - RM_AD_CONTROL_TRIGSRC_BIT para indicar que en iControlBits
 *            esta el parametro que especifica la fuente de lanzamiento
 *            ('triggering')de la conversion A/D.
 *
 *    @return Devuelve SO_NO_ERROR si todo va bien, si no, devuelve
 *          un codigo de error.
 *
 *    @see constants.h (contiene codigos de error y parametros).
 */
int pcm3718h_WriteADControl(int iControlBits,int iMask);

/**
 *    Lee el modo de transferencia de los datos de una convrsion A/D.
 *
 *    @param *piMode Puntero a entero donde se guarda el modo seleccionado
 *          en la tarjeta. Puede ser:
 *          - RM_AD_TRANSFERMODE_PROGRAM: transferencia manual
 *          - RM_AD_TRANSFERMODE_IRQ: mediante manejador de interrupcion
 *          - RM_AD_TRANSFERMODE_DMA: mediante DMA (que tambien necesita de
```

```
 *            un manejador de interrupciones)
 *    @param piIRQ Puntero a entero donde se guarda el numero de la IRQ
 *           usado en los modos IRQ y DMA. Sera uno de RM_AD_CONTROL_IRQ2 a
 *           RM_AD_CONTROL_IRQ7.
 *
 *    @return Devuelve SO_NO_ERROR si todo va bien, si no, devuelve
 *          un codigo de error.
 *
 *    @see constants.h (contiene codigos de error y parametros).
 */
int pcm3718h_GetADTransferMode(int *piMode, int *piIRQ);

/**
 *    Establece el modo de transferencia de los datos de una conversion
A/D.
 *
 *    @param iMode Modo de transferencia a establecer. Puede ser:
 *          - RM_AD_TRANSFERMODE_PROGRAM: transferencia manual
 *          - RM_AD_TRANSFERMODE_IRQ: mediante manejador de interrupcion
 *          - RM_AD_TRANSFERMODE_DMA: mediante DMA (que tambien necesita de
 *            un manejador de interrupciones)
 *    @param iIRQ Numero de la IRQ usado en los modos IRQ y DMA.Puede ser
 *           uno de RM_AD_CONTROL_IRQ2 a RM_AD_CONTROL_IRQ7.
 *
 *    @return Devuelve SO_NO_ERROR si todo va bien, si no, devuelve
 *          un codigo de error.
 *
 *    @see constants.h (contiene codigos de error y parametros).
 */
int pcm3718h_SetADTransferMode(int iMode, int iIRQ);

/**
 *    Lee el modo de lanzamiento ('triggering') establecido para la
 *    conversion A/D.
 *
 *    @param piTrgSrc Puntero a entero donde se guarda el modo de
 *           lanzamiento de la conversion A/D o 'trigger source'. Puede
 *           ser:
 *           - RM_AD_CONTROL_TRIGSRC_SOFTW : lanzamiento por software.
 *           - RM_AD_CONTROL_TRIGSRC_EXT : lanzamiento por pulso externo.
 *           - RM_AD_CONTROL_TRIGSRC_PACER : lanzamiento por pulso
 *             proporcionado por el contador integrado en la tarjeta.
 *
 *    @return Devuelve SO_NO_ERROR si todo va bien, si no, devuelve
 *          un codigo de error.
 *
 *    @see constants.h (contiene codigos de error y parametros).
 */
int pcm3718h_GetADTriggerSource(int *piTrgSrc);

/**
 *    Establece el modo de lanzamiento ('triggering') para la conversion
 *    A/D.
 *
 *    @param iTrgSrc Modo de lanzamiento de la conversion A/D o 'trigger
 *           source'.
 *           Puede ser:
 *           - RM_AD_CONTROL_TRIGSRC_SOFTW : lanzamiento por software.
 *           - RM_AD_CONTROL_TRIGSRC_EXT : lanzamiento por pulso externo.
 *           - RM_AD_CONTROL_TRIGSRC_PACER : lanzamiento por pulso
```

```
 *             proporcionado por el contador integrado en la tarjeta.
 *
 *    @return Devuelve SO_NO_ERROR si todo va bien, si no, devuelve
 *          un codigo de error.
 *
 *    @see constants.h (contiene codigos de error y parametros).
 */
int pcm3718h_SetADTriggerSource(int iTrgSrc);

/*******************************************************
 *
 *    CONTADOR
 *
 *******************************************************/

/**
 *    Calcula la salida del contador interno ('pacer') a partir del valor
 *    de tiempo base establecido y los valores que se preestablecerian en
 *    los contadores 1 y 2. Esta funcion sirve de ayuda para encontrar
 *    dichos valores para los contadores de manera que se obtenga a la
 *    salida la frecuencia deseada.
 *
 *    @param uiC1 Entero sin signo que contiene el valor que se
 *           preestableceria en el contador 1.
 *
 *    @param uiC2 Entero sin signo que contiene el valor que se
 *           preestableceria en el contador 2.
 *
 *    @param pfRate Puntero a float donde se guarda el valor de frecuencia
 *           de salida si los contadores 1 y 2 tuvieran preestablecidos los
 *           valores uiC1 y uiC2 respectivamente. La  frecuencia devuelta
 *           esta calculada en MHz.
 *
 *    @return Devuelve SO_NO_ERROR si todo va bien, si no, devuelve
 *          un codigo de error.
 *
 *    @see constants.h (contiene codigos de error y parametros).
 */
int  pcm3718h_CalcPacerRate(unsigned  int  uiC1,  unsigned  int  uiC2,float
*pfRate);

/**
 *    Habilita el contador interno.
 *
 *    @return Devuelve SO_NO_ERROR si todo va bien, si no, devuelve
 *          un codigo de error.
 *
 *    @see constants.h (contiene codigos de error y parametros).
 */
int pcm3718h_EnablePacer(void);

/**
 *    Inhabilita el contador interno.
 *
 *    @return Devuelve SO_NO_ERROR si todo va bien, si no, devuelve
 *          un codigo de error.
 *
 *    @see constants.h (contiene codigos de error y parametros).
 */
int pcm3718h_DisablePacer(void);
```

```
/**
 *    Lee el estado de habilitacion del contador.
 *
 *    @param piPacerEnabled Puntero a entero donde se guarda un 1 si
 *          el contador esta habilitado o 0 si esta inhabilitado.
 *
 *    @return Devuelve SO_NO_ERROR si todo va bien, si no, devuelve
 *          un codigo de error.
 *
 *    @see constants.h (contiene codigos de error y parametros).
 */
int pcm3718h_GetPacerEnabled(int *piPacerEnabled);

/**
 *    Lee el valor de uno de los contadores.
 *
 *    @param iCounterID Debe contener el ID del contador que se pretende
 *          leer. Puede ser:
 *          - RM_PACER_SC_COUNTER0: ID para el contador 0 (reservado)
 *          - RM_PACER_SC_COUNTER1: ID para el contador 1
 *          - RM_PACER_SC_COUNTER2: ID para el contador 2
 *
 *    @param piCount Puntero a entero donde se guarda el valor del contador
 *          seleccionado.
 *
 *    @return Devuelve SO_NO_ERROR si todo va bien, si no, devuelve
 *          un codigo de error.
 *
 *    @see constants.h (contiene codigos de error y parametros).
 */
int pcm3718h_ReadCounter(int iCounterID,int *piCount);

/**
 *    Escribe un valor dado en el contador seleccionado.
 *
 *    @param iCounterID Debe contener el ID del contador que se pretende
 *          escribir. Puede ser:
 *          - RM_PACER_SC_COUNTER1: ID para el contador 1
 *          - RM_PACER_SC_COUNTER2: ID para el contador 2
 *          El contador 0 (RM_PACER_SC_COUNTER0) esta reservado para uso
 *          futuro.
 *
 *    @param iCount Entero que contiene el valor del contador
 *
 *    @return Devuelve SO_NO_ERROR si todo va bien, si no, devuelve
 *          un codigo de error.
 *
 *    @see constants.h (contiene codigos de error y parametros).
 */
int pcm3718h_WriteCounter(int iCounterID, int iCount);

/**
 *    Configura el modo de funcionamiento del contador seleccionado.
 *
 *    @param iCounterID Debe contener el ID del contador que se pretende
 *           escribir. Puede ser:
 *          - RM_PACER_SC_COUNTER1: ID para el contador 1
 *          - RM_PACER_SC_COUNTER2: ID para el contador 2
 *          El contador 0 (RM_PACER_SC_COUNTER0) esta reservado para uso
```

```
 *          futuro.
 *
 *    @param iRWMode Modo de lectura/escritura. Puede ser:
 *          - RM_PACER_RW_COUNTERLATCH: latchea el valor del contador.
 *          - RM_PACER_RW_LSB: lee/escribe solo el LSB.
 *          - RM_PACER_RW_MSB: lee/escribe solo el MSB
 *          - RM_PACER_RW_LSBMSB: lee/escribe primero el LSB y luego el
 *            MSB.
 *
 *    @param iOpMode Modo de operacion del contador. Puede ser:
 *          - RM_PACER_M_MODE0: 'stop on terminal count'
 *          - RM_PACER_M_MODE1: 'programmable one shot'
 *          - RM_PACER_M_MODE2: 'rate generator'
 *          - RM_PACER_M_MODE3: 'square wave generator'
 *          - RM_PACER_M_MODE4: 'software triggered strobe'
 *          - RM_PACER_M_MODE5: 'hardware triggered strobe'
 *          Para mas informacion acudir al manual de usuario de la tarjeta.
 *
 *    @return Devuelve SO_NO_ERROR si todo va bien, si no, devuelve
 *          un codigo de error.
 *
 *    @see constants.h (contiene codigos de error y parametros).
 */
int pcm3718h_ConfigureCounter(int iCounterID,int iRWMode, int iOpMode, int
iBCDMode);

/**
 *    Lee el byte de estado del contador especificado.
 *
 *    @param iCounterID Debe contener el ID del contador que se pretende
 *          escribir. Puede ser:
 *          - RM_PACER_SC_COUNTER1: ID para el contador 1
 *          - RM_PACER_SC_COUNTER2: ID para el contador 2
 *          El contador 0 (RM_PACER_SC_COUNTER0) esta reservado para uso
 *          futuro.
 *
 *    @param piStatus Puntero a entero donde se guarda, en el byte bajo, el
 *          registro de estado del contador especificado, que tiene el
 *          siguiente significado:
 *
 *          -MSB-                              -LSB-
 *           7    6    5    4    3    2    1    0
 *          OUT   NC  RW1  RW0   M2   M1   M0  BCD
 *
 *    Para mas informacion acudir al manual de usuario de la tarjeta.
 *
 *    @return Devuelve SO_NO_ERROR si todo va bien, si no, devuelve
 *          un codigo de error.
 *
 *    @see constants.h (contiene codigos de error y parametros).
 */
int pcm3718h_ReadCounterStatus(int iCounterID, int *piStatus);

/**
 *    Lee el registro de control del contador interno ('pacer').
 *
 *    @param piControlRegister Puntero a entero donde se guarda, en el byte
 *          bajo, el registro de control del contador interno ('pacer').
 *          Tiene el siguiente significado:
 *
```

```
 *          -MSB-                                -LSB-
 *           7    6    5    4    3    2    1    0
 *          SC1  SC0  RW1  RW0   M2   M1   M0  BCD
 *
 *    Para mas informacion acudir al manual de usuario de la tarjeta.
 *
 *    @return Devuelve SO_NO_ERROR si todo va bien, si no, devuelve
 *          un codigo de error.
 *
 *    @see constants.h (contiene codigos de error y parametros).
 */
int pcm3718h_ReadPacerControl(int *piControlRegister);

/**
 *    Escribe el registro de control del contador interno ('pacer').
 *
 *    @param iControlRegister Entero donde que especifica (solo se tendra
 *           en cuenta el byte bajo) el registro de control del
 *           contador interno ('pacer'). Tiene el siguiente significado:
 *
 *          -MSB-                                -LSB-
 *           7    6    5    4    3    2    1    0
 *          SC1  SC0  RW1  RW0   M2   M1   M0  BCD
 *
 *    Para mas informacion acudir al manual de usuario de la tarjeta.
 *
 *    @return Devuelve SO_NO_ERROR si todo va bien, si no, devuelve
 *          un codigo de error.
 *
 *    @see constants.h (contiene codigos de error y parametros).
 */
int pcm3718h_WritePacerControl(int iControlRegister);

/*******************************************************
 *
 *    OPCIONES DE CONFIGURACION
 *
 *******************************************************/

/**
 *    Indica al Resource Manager que presente por consola (si
 *    es que se lanzo desde una) las opciones de configuracion
 *    disponibles en la tarjeta.
 *
 *    @params void
 *
 *    @return Devuelve SO_NO_ERROR si todo va bien, si no, devuelve
 *    un codigo de error.
 *
 *    @see constants.h (contiene codigos de error y parametros).
 */
int pcm3718h_ShowOptions(void);

/**
 *    Pregunta al Resource Manager por la direccion base especificada
 *    por el usuario.
 *
 *    @param piDirBase Puntero a entero donde se almacenara la direccion
 *           base de la tarjeta.
 *
```

```
 *    @return Devuelve SO_NO_ERROR si todo va bien, si no, devuelve
 *    un codigo de error.
 *
 *    @see constants.h (contiene codigos de error y parametros).
 */
int pcm3718h_GetDirBase(int * piDirBase);

/**
 *    Pregunta al Resorce Manager por el canal DMA especificado
 *    por el usuario
 *
 *    @param piDMAChannel Puntero a entero donde se almacenara el canal
 *           DMA.
 *
 *    @return Devuelve SO_NO_ERROR si todo va bien, si no, devuelve
 *    un codigo de error.
 *
 *    @see constants.h (contiene codigos de error y parametros).
 */
int pcm3718h_GetDMAChannel(int * piDMAChannel);

/**
 *    Pregunta al Resorce Manager por la base temporal especificada
 *    por el usuario (para el contador interno).
 *
 *    @param piTimeBase Puntero a entero donde se almacenara la base
 *           temporal.
 *
 *    @return Devuelve SO_NO_ERROR si todo va bien, si no, devuelve
 *    un codigo de error.
 *
 *    @see constants.h (contiene codigos de error y parametros).
 */
int pcm3718h_GetTimeBase(int * piTimeBase);

/**
 *    Pregunta al Resorce Manager si el usuario ha designado la entrada
 *    digital 0 como trigger externo o no.
 *
 *    @param piExtTrigger Puntero a entero donde se almacenara la seleccion
 *          del usuario (0 -> DI0 es entrada digital ; 1 -> DI0 es trigger
 *          externo).
 *
 *    @return Devuelve SO_NO_ERROR si todo va bien, si no, devuelve
 *    un codigo de error.
 *
 *    @see constants.h (contiene codigos de error y parametros).
 */
int pcm3718h_GetExtTrigger(int * piExtTrigger);

/**
 *    Pregunta al Resorce Manager por el modo de los canales analogicos
 *    que ha especificado el usuario.
 *
 *    @param piAnalogChannelMode Puntero a entero donde se almacenara
 *          la seleccion del usuario (0 -> 8 entradas diferenciales ;
 *          1 -> 16 entradas 'single-ended')  .
 *
 *    @return Devuelve SO_NO_ERROR si todo va bien, si no, devuelve
 *    un codigo de error.
```

```
 *
 *    @see constants.h (contiene codigos de error y parametros).
 */
int pcm3718h_GetAnalogChannelMode(int * piAnalogChannelMode);

#endif
```

4.2.2 devio-pcm3718h.c

```
/**
 *    Implementa las funciones publicas de la libreria
 *    compartida.
 *
 *    @file devio-pcm3718h.c
 *    @author Raul Jimenez Ruiz
 */

/*
 *    INCLUDES genericos
 */
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/neutrino.h>
#include <sys/iomsg.h>
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <fcntl.h>

#include <errno.h>
#include <string.h>

//#include <photon/realtime/RtTimer.h> /* para RtTimer... */

/*
 *    Constantes para el uso de la PCM3718H y la comunicacion
 *    con el Resource Manager
 */
#include "../rm_pcm3718h/constants.h"

/*
 *    Inclusion del .h que usaran los clientes de este API
 */
#define _SHARED_LIB_CODE_
#include "devio-pcm3718h.h"

/**
 *    Esta macro es el codigo de terminacion de todas las funciones de la
 *    libreria. El funcionamiento es el siguiente: si 'ret', que es la
 *    variable que almacena el valor devuelto por la funcion 'sendMsg', es
 *    distinta de SO_NO_ERROR es que se ha producido un error en el nivel
 *    de la libreria, por lo que devuelve ese error. En caso de que sea
 *    SO_NO_ERROR, se verifica la respuesta del Resource Manager en la
 *    variable reply.error ('reply' es un 'pcm3718h_msg_t' devuelto por
 *    'sendMsg', conteniendo el mensaje de respuesta del Resource Manager),
 *    si este indica que no ha habido error (==RM_NO_ERROR) se devuelve
 *    SO_NO_ERROR, en caso contrario se devuelve el error indicado por
 *    el Resource Manager.
 */
```

```
#define RETURN_ERR(ret,reply) if(ret==SO_NO_ERROR)          \
                              {                             \
                              if(reply.error==RM_NO_ERROR)  \
                                    return SO_NO_ERROR;     \
                              else                          \
                                    return reply.error;     \
                              }                             \
                              else                          \
                                    return ret;


/*******************************************************
 *
 *    UTILIDADES
 *
 *******************************************************/

const char * RMErrorToString(int err)
{
      switch(err)
      {
            case RM_NO_ERROR:
                  return "<rm> No error";

            case RM_ERROR_OPN_DEV:
                  return "<rm> Error abriendo el dispositivo";

            case RM_ERROR_UNK_MSG:
                  return "<rm> Error, mensaje desconocido";

            case RM_ERROR_MSG_SEND:
                  return "<rm> Error enviando el mensaje";

            case RM_ERROR_BAD_ARGUMENT:
                  return "<rm> Error, argumento no valido";

            case RM_ERROR_OUT_OF_RANGE:
                  return "<rm> Error, fuera de rango";

            case RM_ERROR_NO_HANDLER:
                  return "<rm> Error, no hay manejador para ese mensaje";

            default:
                  return "<rm> ?";
      }
}

const char * SOErrorToString(int err)
{
      switch(err)
      {
            case SO_NO_ERROR:
                  return "<so> No error";

            case SO_ERROR_OPN_DEV:
                  return "<so> Error abriendo el dispositivo";

            case SO_ERROR_MSG_SEND:
                  return "<so> Error enviando el mensaje";
```

```
            case SO_ERROR_BAD_ARGUMENT:
                  return "<so> Error, argumento no valido";

            default:
                  return "<so> ?";
      }
}

const char * errorToString(int err)
{
      const char *str;

      str=RMErrorToString(err);
      if(strcmp("<rm> ?",str)!=0) return str;

      str=SOErrorToString(err);
      if(strcmp("<so> ?",str)!=0) return str;

      return "!?";
}

/*******************************************************
 *
 *    FUNCIONES PARA EL PASO DE MENSAJES
 *
 *******************************************************/

void copyData(unsigned char *dst, unsigned char *src,unsigned int len)
{
      //    ATENCION : dst y src deben ser al menos de tamaÃ±o len
      int i;

      if(dst==NULL || src==NULL)
            return;

      for(i=0;i<len;i++)
            dst[i]=src[i];
}

//
//    si no queremos respuesta ponemos al final un NULL
//
int   sendMsg(unsigned   int   msgN,unsigned   char   *pData,   unsigned   int
dataLen,pcm3718h_msg_t *msgReply)
{
      int fd,ret;
      pcm3718h_msg_t *send;   //    es necesario que sean punteros? !!!
      pcm3718h_msg_t *reply;  //    y tener que hacer malloc cada vez? !!!

      if(pData==NULL || dataLen>=RM_MSG_MAX_LEN || msgN<RM_MSG_FIRST ||
        msgN>RM_MSG_LAST)
            return RM_ERROR_BAD_ARGUMENT;

      send=(pcm3718h_msg_t *)malloc(sizeof(pcm3718h_msg_t));
      reply=(pcm3718h_msg_t *)malloc(sizeof(pcm3718h_msg_t));

      memset(send,0,sizeof(pcm3718h_msg_t));
      memset(reply,0,sizeof(pcm3718h_msg_t));

      // ¡¡¡ ATENCION:
```

```
	// msgN no puede ser 0, ya que el id del mensaje debe ser > _IO_MAX
	// send->id=msgN+_IO_MAX;
	copyData(send->data,pData,(dataLen<RM_MSG_MAX_LEN ? dataLen :
	RM_MSG_MAX_LEN));

	fd=open(RM_DEV_NAME,O_RDWR);
	if(fd==-1)
	{
		close(fd);
		free(send);
		free(reply);

		return SO_ERROR_OPN_DEV;
	}

	ret=MsgSend(fd,send,sizeof(pcm3718h_msg_t),reply,
	 sizeof(pcm3718h_msg_t));
	if(ret==-1)
	{
		printf("<error %d: %s>\n",errno,strerror(errno));

		close(fd);
		free(send);
		free(reply);

		return SO_ERROR_MSG_SEND;
	}

	if(msgReply!=NULL)
	{
		msgReply->id=reply->id;
		msgReply->error=reply->error;
		copyData(msgReply->data,reply->data,RM_MSG_MAX_LEN);
	}

	close(fd);
	free(send);
	free(reply);

	return SO_NO_ERROR;
}

/*******************************************************
 *
 *    FUNCIONES PARA LAS SEÃ‘ALES DIGITALES
 *
 *******************************************************/

int pcm3718h_SetDigBit(int iChannel,int iBitMask)
{
	pcm3718h_msg_t reply;
	unsigned char data[2];
	int ret;

	if(iChannel!=RM_DIG_CHANNEL_A && iChannel!=RM_DIG_CHANNEL_B)
		return SO_ERROR_BAD_ARGUMENT;

	//    como solo hay dos canales de 8 bits cada uno ...
	data[0]=LOW_BYTE(iChannel);
	data[1]=LOW_BYTE(iBitMask);
```

```
	ret=sendMsg(RM_MSG_DIG_SETBIT,data,2,&reply);

	RETURN_ERR(ret,reply);
}

int pcm3718h_ResetDigBit(int iChannel,int iBitMask)
{
	pcm3718h_msg_t reply;
	unsigned char data[2];
	int ret;

	if(iChannel!=RM_DIG_CHANNEL_A && iChannel!=RM_DIG_CHANNEL_B)
		return SO_ERROR_BAD_ARGUMENT;

	data[0]=LOW_BYTE(iChannel);
	data[1]=LOW_BYTE(iBitMask);

	ret=sendMsg(RM_MSG_DIG_RESETBIT,data,2,&reply);

	RETURN_ERR(ret,reply);
}

int pcm3718h_ReadDigStatus(int *piChannelA,int *piChannelB)
{
	pcm3718h_msg_t reply;
	unsigned char data[2];
	int ret;

	if(piChannelA==NULL && piChannelB==NULL)
		return SO_ERROR_BAD_ARGUMENT;

	if(piChannelA != NULL) data[0]=1; else data[0]=0;
	if(piChannelB != NULL) data[1]=1; else data[1]=0;

	ret=sendMsg(RM_MSG_DIG_READSTATUS,data,2,&reply);

	if(piChannelA != NULL) *piChannelA=RM_NULLPARAM;
	if(piChannelB != NULL) *piChannelB=RM_NULLPARAM;

	if(piChannelA != NULL) *piChannelA=(int)(reply.data[0]);
	if(piChannelB != NULL) *piChannelB=(int)(reply.data[1]);

	RETURN_ERR(ret,reply);
}

int pcm3718h_WriteDigStatus(int iChannelA, int iChannelB)
{
	pcm3718h_msg_t reply;
	unsigned char data[4]={0,0,0,0};
	int ret;

	if(iChannelA==RM_NULLPARAM && iChannelB==RM_NULLPARAM)
		return SO_ERROR_BAD_ARGUMENT;

	//	nos quedamos con el byte bajo de ambos

	if(iChannelA != RM_NULLPARAM)
	{
		data[0]=1;
```

```
            data[2]=LOW_BYTE(iChannelA);
      }

      if(iChannelB != RM_NULLPARAM)
      {
            data[1]=1;
            data[3]=LOW_BYTE(iChannelB);
      }

      ret=sendMsg(RM_MSG_DIG_WRITESTATUS,data,4,&reply);

      RETURN_ERR(ret,reply);
}

int pcm3718h_ReadDigInt(int *piDigI)
{
      pcm3718h_msg_t reply;
      unsigned char data[1]={0};
      int ret;

      if(piDigI==NULL)
            return SO_ERROR_BAD_ARGUMENT;

      ret=sendMsg(RM_MSG_DIG_WRITESTATUS,data,1,&reply);

      *piDigI=MAKE_INT(reply.data[1],reply.data[0]);

      RETURN_ERR(ret,reply);
}

int pcm3718h_WriteDigInt(int iDigO)
{
      pcm3718h_msg_t reply;
      unsigned char data[2]={0,0};
      int ret;

      data[0]=LSB(iDigO);
      data[1]=MSB(iDigO);

      ret=sendMsg(RM_MSG_DIG_WRITESTATUS,data,2,&reply);

      RETURN_ERR(ret,reply);
}

/*******************************************************
 *
 *    CONVERSION A/D
 *
 *******************************************************/

int pcm3718h_ReadADData(int *piChannel, int *piData)
{
      pcm3718h_msg_t reply;
      unsigned char data[1];
      int ret;

      if(piChannel==NULL || piData==NULL)
            return SO_ERROR_BAD_ARGUMENT;

      ret=sendMsg(RM_MSG_AD_READDATA,data,2,&reply);
```

```
	*piChannel=reply.data[0];
	*piData=MAKE_INT(reply.data[2],reply.data[1]);

	RETURN_ERR(ret,reply);
}

int pcm3718h_TriggerConversion()
{
	pcm3718h_msg_t reply;
	unsigned char data[1];
	int ret;

	ret=sendMsg(RM_MSG_AD_TRIGGERCONV,data,1,&reply);

	RETURN_ERR(ret,reply);
}

int pcm3718h_SetADRange(int iChannel, int iRange)
{
	pcm3718h_msg_t reply;
	unsigned char data[2];
	int ret;

	//    aqui se hace una breve comprobacion, se deja para el RM
	//    una comprobacion mas exhaustiva
	if(iChannel<0) iChannel=-iChannel;
	if(iChannel>15)
		return SO_ERROR_BAD_ARGUMENT;

	iRange=iRange & RM_AD_RANGE_MASK;

	data[0]=iChannel;
	data[1]=iRange;

	ret=sendMsg(RM_MSG_AD_SETRANGE,data,2,&reply);

	RETURN_ERR(ret,reply);
}

int pcm3718h_ReadADMuxScan(int *piStartChannel, int *piStopChannel)
{
	pcm3718h_msg_t reply;
	unsigned char data[2];
	int ret;

	if(piStartChannel==NULL || piStopChannel==NULL)
		return SO_ERROR_BAD_ARGUMENT;

	ret=sendMsg(RM_MSG_AD_READMUXSCAN,data,2,&reply);

	*piStartChannel=reply.data[0];
	*piStopChannel=reply.data[1];

	RETURN_ERR(ret,reply);
}

int pcm3718h_WriteADMuxScan(int iStartChannel, int iStopChannel)
{
	pcm3718h_msg_t reply;
```

```
	unsigned char data[2];
	int ret;

	data[0]=iStartChannel;
	data[1]=iStopChannel;

	ret=sendMsg(RM_MSG_AD_WRITEMUXSCAN,data,2,&reply);

	RETURN_ERR(ret,reply);
}

int pcm3718h_GetADTransferMode(int *piMode, int *piIRQ)
{
	int ret,control;

	if(piMode==NULL || piIRQ==NULL)
		return SO_ERROR_BAD_ARGUMENT;

	ret=pcm3718h_ReadADControl(&control);
	if(ret!=SO_NO_ERROR)
		return ret;

	if(control & RM_AD_CONTROL_INTE_BIT)
	{
		if(control & RM_AD_CONTROL_DMAE_BIT)
			*piMode=RM_AD_TRANSFERMODE_DMA;
		else
			*piMode=RM_AD_TRANSFERMODE_IRQ;

		*piIRQ=control & RM_AD_CONTROL_INTLEVEL_BIT;
	}
	else
	{
		*piIRQ=0;
		*piMode=RM_AD_TRANSFERMODE_PROGRAM;
	}

	return SO_NO_ERROR;
}

int pcm3718h_SetADTransferMode(int iMode, int iIRQ)
{
	int ret,control,mask;

	mask= RM_AD_CONTROL_INTE_BIT |
		RM_AD_CONTROL_INTLEVEL_BIT |
		RM_AD_CONTROL_DMAE_BIT;

	if(iMode!=RM_AD_TRANSFERMODE_PROGRAM &&
	   iMode!=RM_AD_TRANSFERMODE_IRQ &&
	   iMode!=RM_AD_TRANSFERMODE_DMA)
		return SO_ERROR_BAD_ARGUMENT;

	if(iMode==RM_AD_TRANSFERMODE_PROGRAM)
		control=0;
	else if(iMode==RM_AD_TRANSFERMODE_IRQ ||
		iMode==RM_AD_TRANSFERMODE_DMA)
	{
		control=RM_AD_CONTROL_INTE_BIT;
		switch(iIRQ)
```

```
            {
                  case RM_AD_CONTROL_IRQ2:
                        control |= RM_AD_CONTROL_IRQ2; break;
                  case RM_AD_CONTROL_IRQ3:
                        control |= RM_AD_CONTROL_IRQ3; break;
                  case RM_AD_CONTROL_IRQ4:
                        control |= RM_AD_CONTROL_IRQ4; break;
                  case RM_AD_CONTROL_IRQ5:
                        control |= RM_AD_CONTROL_IRQ5; break;
                  case RM_AD_CONTROL_IRQ6:
                        control |= RM_AD_CONTROL_IRQ6; break;
                  case RM_AD_CONTROL_IRQ7:
                        control |= RM_AD_CONTROL_IRQ7; break;
                  default:
                        return SO_ERROR_BAD_ARGUMENT;
            }

            // para el DMA hace falta tambien seleccionar una IRQ
            if(iMode==RM_AD_TRANSFERMODE_DMA)
                  control |= RM_AD_CONTROL_DMAE_BIT;
      }

      ret=pcm3718h_WriteADControl(control,mask);

      return ret;
}

int pcm3718h_GetADTriggerSource(int *piTrgSrc)
{
      int ret,control;

      if(piTrgSrc==NULL)
            return SO_ERROR_BAD_ARGUMENT;

      ret=pcm3718h_ReadADControl(&control);
      if(ret!=SO_NO_ERROR)
            return ret;

      *piTrgSrc=control & RM_AD_CONTROL_TRIGSRC_BIT;

      return SO_NO_ERROR;
}

int pcm3718h_SetADTriggerSource(int iTrgSrc)
{
      int ret,control,mask,tr;

      if(iTrgSrc!=RM_AD_CONTROL_TRIGSRC_SOFTW ||
         iTrgSrc!=RM_AD_CONTROL_TRIGSRC_EXT ||
         iTrgSrc!=RM_AD_CONTROL_TRIGSRC_PACER)
            return SO_ERROR_BAD_ARGUMENT;

      //    si se elige RM_AD_CONTROL_TRIGSRC_EXT, hay que comprobar que
      //    este seleccionado
      //    DI0 como 'ExtTrigger'
      ret=pcm3718h_GetExtTrigger(&tr);
      if(ret!=SO_NO_ERROR)
            return ret;

      if(iTrgSrc==RM_AD_CONTROL_TRIGSRC_EXT && tr!=1)
```

```
            return RM_ERROR_CONFIGURATION;

      control=iTrgSrc;
      mask=RM_AD_CONTROL_TRIGSRC_BIT;

      ret=pcm3718h_WriteADControl(control,mask);

      return ret;
}

int pcm3718h_WaitToEndOfConversion(unsigned long ulMs)
{
      struct timespec ms0,dms;
      int ret,status,condicion=1;

      if(clock_gettime(CLOCK_REALTIME,&ms0)==-1)
            return SO_ERROR_SYSTEM;

      dms.tv_sec=0;
      dms.tv_nsec=0;
      while(condicion==1)
      {
            //    leemos el bit EOC=0 del registro de estado
            ret=pcm3718h_ReadADStatus(&status);
            if(ret!=SO_NO_ERROR)
                  return ret;

            if(status & RM_AD_STATUS_EOC_BIT)
                  return SO_NO_ERROR;

            if(clock_gettime(CLOCK_REALTIME,&dms)==-1)
                  return SO_ERROR_SYSTEM;

            if(ms!=0)
                  condicion = (dms.tv_sec*1000+dms.tv_nsec/1000) < ms;
      }

      return SO_ERROR_TIME_EXCEEDED;
}

/*******************************************************
 *
 *    ESTADO / CONTROL
 *
 *******************************************************/

// --- ESTADO

int pcm3718h_ReadADStatus(int *piADStatusRegister)
{
      pcm3718h_msg_t reply;
      int ret;
      unsigned char data[1];

      if(piADStatusRegister==NULL)
            return SO_ERROR_BAD_ARGUMENT;

      ret=sendMsg(RM_MSG_AD_READSTATUS,data,1,&reply);

      *piADStatusRegister=(int)(reply.data[0]);
```

```
	RETURN_ERR(ret,reply);
}

int pcm3718h_ClearInterruptRequest(void)
{
	pcm3718h_msg_t reply;
	int ret;
	unsigned char data[1];

	ret=sendMsg(RM_MSG_AD_CLEARINTERRUPTREQUEST,data,1,&reply);

	RETURN_ERR(ret,reply);
}

// --- CONTROL

int pcm3718h_ReadADControl(int *piADControlRegister)
{
	pcm3718h_msg_t reply;
	int ret;
	unsigned char data[1];

	if(piADControlRegister==NULL)
		return SO_ERROR_BAD_ARGUMENT;

	ret=sendMsg(RM_MSG_AD_READCONTROL,data,1,&reply);

	*piADControlRegister=(int)(reply.data[0]);

	RETURN_ERR(ret,reply);
}

int pcm3718h_WriteADControl(int iControlBits,int iMask)
{
	pcm3718h_msg_t reply;
	int ret;
	unsigned char data[2];

	data[0]=LOW_BYTE(iControlBits);
	data[1]=LOW_BYTE(iMask);

	ret=sendMsg(RM_MSG_AD_WRITECONTROL,data,2,&reply);

	RETURN_ERR(ret,reply);
}

/*******************************************************
 *
 *    CONTADOR
 *
 *******************************************************/

int pcm3718h_CalcPacerRate(unsigned int C1, unsigned int C2,float *pfRate)
{
	int ret,timeBase;

	if(C1==0 || C2==0 || pfRate==NULL)
		return SO_ERROR_BAD_ARGUMENT;
```

```
	ret=pcm3718h_GetTimeBase(&timeBase);
	if(ret!=SO_NO_ERROR)
		return ret;

	*pfRate=(float)timeBase/((float)C1*(float)C2);

	return SO_NO_ERROR;
}

int pcm3718h_EnablePacer(void)
{
	pcm3718h_msg_t reply;
	int ret;
	unsigned char data[1];

	data[0]=1;

	ret=sendMsg(RM_MSG_PACER_ENABLEDISABLE,data,1,&reply);

	RETURN_ERR(ret,reply);
}

int pcm3718h_DisablePacer(void)
{
	pcm3718h_msg_t reply;
	int ret;
	unsigned char data[2];

	data[0]=0;

	ret=sendMsg(RM_MSG_PACER_ENABLEDISABLE,data,1,&reply);

	RETURN_ERR(ret,reply);
}

int pcm3718h_GetPacerEnabled(int *piPacerEnabled)
{
	pcm3718h_msg_t reply;
	int ret;
	unsigned char data[1];

	if(piPacerEnabled==NULL)
		return SO_ERROR_BAD_ARGUMENT;

	ret=sendMsg(RM_MSG_PACER_GETENABLED,data,1,&reply);

	*piPacerEnabled=(int)(reply.data[0]);

	RETURN_ERR(ret,reply);
}

int pcm3718h_ReadCounter(int iCounterID,int *piCount)
{
	pcm3718h_msg_t reply;
	unsigned char data[1];
	int ret;

	if(piCount==NULL)
		return SO_ERROR_BAD_ARGUMENT;
```

```
	if(iCounterID!=RM_PACER_SC_COUNTER0 &&
	iCounterID!=RM_PACER_SC_COUNTER1 && iCounterID!=RM_PACER_SC_COUNTER2)
		return SO_ERROR_BAD_ARGUMENT;

	data[0]=(unsigned char)iCounterID;

	ret=sendMsg(RM_MSG_PACER_READCOUNTER,data,1,&reply);

	*piCount=MAKE_INT(reply.data[1],reply.data[0]);

	RETURN_ERR(ret,reply);
}

int pcm3718h_WriteCounter(int iCounterID, int iCount)
{
	pcm3718h_msg_t reply;
	unsigned char data[3];
	int ret;

	if(iCounterID!=RM_PACER_SC_COUNTER0 &&
	   iCounterID!=RM_PACER_SC_COUNTER1 &&
	   iCounterID!=RM_PACER_SC_COUNTER2)
		return SO_ERROR_BAD_ARGUMENT;

	data[0]=(unsigned char)iCounterID;
	data[1]=LSB(iCount);
	data[2]=MSB(iCount);

	ret=sendMsg(RM_MSG_PACER_WRITECOUNTER,data,3,&reply);

	RETURN_ERR(ret,reply);
}

int pcm3718h_ConfigureCounter(int iCounterID,int iRWMode, int iOpMode, int
iBCDMode)
{
	pcm3718h_msg_t reply;
	unsigned char data[4];
	int ret;

	if(iCounterID!=RM_PACER_SC_COUNTER0 &&
	   iCounterID!=RM_PACER_SC_COUNTER1 &&
	   iCounterID!=RM_PACER_SC_COUNTER2)
		return SO_ERROR_BAD_ARGUMENT;

	if(iRWMode!=RM_PACER_RW_LSBMSB && iRWMode!=RM_PACER_RW_COUNTERLATCH
	   && iRWMode!=RM_PACER_RW_LSB && iRWMode!=RM_PACER_RW_MSB)
		return SO_ERROR_BAD_ARGUMENT;

	data[0]=(unsigned char)iCounterID;
	data[1]=(unsigned char)iRWMode;
	data[2]=(unsigned char)iOpMode;
	data[3]=(unsigned char)iBCDMode;

	ret=sendMsg(RM_MSG_PACER_CONFIGURECOUNTER,data,4,&reply);

	RETURN_ERR(ret,reply);
}

int pcm3718h_ReadCounterStatus(int iCounterID, int *piStatus)
```

```
{
      pcm3718h_msg_t reply;
      unsigned char data[1];
      int ret;

      if(piStatus==NULL)
            return SO_ERROR_BAD_ARGUMENT;

      if(iCounterID!=RM_PACER_SC_COUNTER0 &&
         iCounterID!=RM_PACER_SC_COUNTER1 &&
         iCounterID!=RM_PACER_SC_COUNTER2)
            return SO_ERROR_BAD_ARGUMENT;

      data[0]=(unsigned char)iCounterID;

      ret=sendMsg(RM_MSG_PACER_READCOUNTERSTATUS,data,1,&reply);

      *piStatus=(int)reply.data[0];

      RETURN_ERR(ret,reply);
}

int pcm3718h_ReadPacerControl(int *piControlRegister)
{
      pcm3718h_msg_t reply;
      unsigned char data[1];
      int ret;

      if(piControlRegister==NULL)
            return SO_ERROR_BAD_ARGUMENT;

      ret=sendMsg(RM_MSG_PACER_READCONTROL,data,1,&reply);

      *piControlRegister=(int)reply.data[0];

      RETURN_ERR(ret,reply);
}

int pcm3718h_WritePacerControl(int iControlRegister)
{
      pcm3718h_msg_t reply;
      unsigned char data[1];
      int ret;

      data[0]=(unsigned char)iControlRegister;

      ret=sendMsg(RM_MSG_PACER_WRITECONTROL,data,1,&reply);

      RETURN_ERR(ret,reply);
}


/*******************************************************
 *
 *    OPCIONES DE CONFIGURACION
 *
 *******************************************************/

int pcm3718h_ShowOptions(void)
{
```

```
	pcm3718h_msg_t reply;
	int ret;
	unsigned char data[1];

	ret=sendMsg(RM_MSG_SHOWOPTIONS,data,1,&reply);

	RETURN_ERR(ret,reply);
}

int pcm3718h_GetDirBase(int * piDirBase)
{
	pcm3718h_msg_t reply;
	int ret;
	unsigned char data[6];

	if(piDirBase==NULL)
		return SO_ERROR_BAD_ARGUMENT;

	ret=sendMsg(RM_MSG_GETOPTIONS,data,6,&reply);

	*piDirBase=MAKE_INT(reply.data[1],reply.data[0]);

	RETURN_ERR(ret,reply);
}

int pcm3718h_GetDMAChannel(int * piDMAChannel)
{
	pcm3718h_msg_t reply;
	int ret;
	unsigned char data[6];

	if(piDMAChannel==NULL)
		return SO_ERROR_BAD_ARGUMENT;

	ret=sendMsg(RM_MSG_GETOPTIONS,data,6,&reply);

	*piDMAChannel=(int)(reply.data[2]);

	RETURN_ERR(ret,reply);
}

int pcm3718h_GetTimeBase(int * piTimeBase)
{
	pcm3718h_msg_t reply;
	int ret;
	unsigned char data[6];

	if(piTimeBase==NULL)
		return SO_ERROR_BAD_ARGUMENT;

	ret=sendMsg(RM_MSG_GETOPTIONS,data,6,&reply);

	*piTimeBase=(int)(reply.data[3]);

	RETURN_ERR(ret,reply);
}

int pcm3718h_GetExtTrigger(int * piExtTrigger)
{
	pcm3718h_msg_t reply;
```

```
	int ret;
	unsigned char data[6];

	if(piExtTrigger==NULL)
		return SO_ERROR_BAD_ARGUMENT;

	ret=sendMsg(RM_MSG_GETOPTIONS,data,6,&reply);

	*piExtTrigger=(int)(reply.data[4]);

	RETURN_ERR(ret,reply);
}

int pcm3718h_GetAnalogChannelMode(int *piAnalogChannelMode)
{
	pcm3718h_msg_t reply;
	int ret;
	unsigned char data[6];

	if(piAnalogChannelMode==NULL)
		return SO_ERROR_BAD_ARGUMENT;

	ret=sendMsg(RM_MSG_GETOPTIONS,data,6,&reply);

	*piAnalogChannelMode=(int)(reply.data[5]);

	RETURN_ERR(ret,reply);
}
```

4.3 Aplicación GUI.

Se verá a continuación el código funete de algunos archivos que forman parte de la aplicación GUI, entre ellos, el que contiene la función de respuesta al evento "abrir ventana" de la ventana principal de la aplicación y los que contienen los prototipos de las funciones para el manejo de las entradas/salidas digitales, el de la etapa analógica y el de los contadores.

4.3.1 wMain_Open.c

```
/* Y o u r   D e s c r i p t i o n                       */
/*                           AppBuilder Photon Code Lib */
/*                                         Version 2.01  */

/* Standard headers */
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>

#include <errno.h>

/* Local headers */
#include "ablibs.h"
#include "abimport.h"
#include "proto.h"

#include "libproto.h"
```

```
extern int waitDigThread;
extern int waitADThread;

int
wMain_Open( PtWidget_t *widget, ApInfo_t *apinfo, PtCallbackInfo_t *cbinfo
)

      {
      int status,i,max;

      /* eliminate 'unreferenced' warnings */
      widget = widget, apinfo = apinfo, cbinfo = cbinfo;

      //    Inicializamos los arrays de objetos graficos
      initArrays();

      //    Antes del bucle principal lanzamos el hilo de refresco de I/O
      //    digital
      waitDigThread=0;
      if(LaunchDigInputsThread()!=EOK)
            printf("\n- ERROR: No se ha podido crear el hilo de refresco de
                  entradas digitales.\n");
      else
            waitDigThread=1;

      //    Antes del bucle principal lanzamos tambien el hilo de refresco
      //    de A/D
      waitADThread=0;
      if(LaunchADThread()!=EOK)
            printf("\n- ERROR: No se ha podido crear el hilo de refresco de
                  conversion A/D.\n");
      else
            waitADThread=1;

      //    Ahora comfiguramos por defecto los canales A/D disponibles
      //    (se deberian deshabilitar los progress de los no disponibles)
      if(pcm3718h_ReadADStatus(&status)==SO_NO_ERROR)
      {
            if((status & RM_AD_STATUS_MUX_BIT) == RM_AD_STATUS_MUX_16SE)
                  max=16;
            else
                  max=8;

            for(i=0;i<max;i++)
            {

      if(pcm3718h_SetADRange(i,RM_AD_RANGE_BIPOLAR_10_0)!=SO_NO_ERROR)
                        break;
                  configureProgress(i,-10000,10000);
            }
      }

      return( Pt_CONTINUE );

      }
```

4.3.2 DigitalIO.h

```
/**
 *    Declaracion de las funciones helper para el manejo de
 *    las E/S digitales desde el GUI.
 *
 *    @file DigitalIO.h
 *    @author Raul Jimenez Ruiz
 */

#ifndef _DIGITALIO_H_
#define _DIGITALIO_H_

#include "libproto.h"

//#ifndef _DENTRO_DE_DIGITALIO_C_

/*******************************************************
 *    FUNCIONES DE UTILLAJE
 */

/**
 *    Bin2Hex. Convierte un entero en un string con formato
 *    hexadecimal.
 *
 *    @param iBin Entero a convertir
 *    @param pcHex Puntero a char donde guardar el string
 *    @return Devuelve 0 si no ocurre ningun error
 */
int Bin2Hex(int iBin, char * pcHex);

void initArrays();

/*******************************************************
 *    FUNCIONES DE I/O DIGITAL
 */

/**
 *    SetDigBitChannelA. Pone a uno el bit indicado para
 *    el canal A.
 *
 *    @param iBit Bit a cambiar
 *    @return Devuelve -1 si ocurre ningun error o el estado actual
 *          del canal A si todo va bien
 */
int SetDigBitChannelA(int iBit);

/**
 *    SetDigBitChannelB. Pone a uno el bit indicado para
 *    el canal B.
 *
 *    @param iBit Bit a cambiar
 *    @return Devuelve -1 si ocurre ningun error o el estado actual
 *          del canal B si todo va bien
 */
int SetDigBitChannelB(int iBit);

/**
 *    ResetDigBitChannelA. Pone a cero el bits indicados para
```

```
 *    el canal A.
 *
 *    @param iBit Bit a cambiar
 *    @return Devuelve -1 si ocurre ningun error o el estado actual
 *          del canal A si todo va bien
 */
int ResetDigBitChannelA(int iBit);

/**
 *    ResetDigBitChannelB. Pone a cero el bits indicados para
 *    el canal B.
 *
 *    @param iBit Bit a cambiar
 *    @return Devuelve -1 si ocurre ningun error o el estado actual
 *          del canal B si todo va bien
 */
int ResetDigBitChannelB(int iBit);

/**
 *    ReadDigChannelA. Lee el estado del canal diital A.
 *
 *    @param piChannelStatus Variable donde se devuelve el
 *          estado del canal A
 *    @return Devuelve NO_ERROR (definido en constants.h) si todo
 *          va bien
 */
int ReadDigChannelA(int *piChannelStatus);

/**
 *    ReadDigChannelB. Lee el estado del canal diital B.
 *
 *    @param piChannelStatus Variable donde se devuelve el
 *          estado del canal B
 *    @return Devuelve NO_ERROR (definido en constants.h) si todo
 *          va bien
 */
int ReadDigChannelB(int *piChannelStatus);

/*******************************************************
 *    FUNCIONES DE REFRESCO DE GUI
 */

/**
 *    RefreshOutput. Actualiza el estado de una salida digital en
 *    respuesta a la pulsacion de un boton de salida digital,
 *    actualizando el texto de una etiqueta dada.
 *
 *    @param button Boton que origina el evento
 *    @param label Etiqueta donde se escribira (en hexadecimal)
 *          el estado del canal seleccionado
 *    @param iChannel Canal afectado por la actualizacion
 *    @param bit Bit del canal seleccionado cuyo valor va a cambiar
 *
 *    @return void
 */
void RefreshOutput(PtWidget_t *button, PtWidget_t *label, int iChannel, int
iBit);

/**
 *    RefreshAllOutputs. Actualiza el estado de las salidas digitales en
```

```
 *    respuesta a la pulsacion de unos botones de salida digital,
 *    actualizando el texto de una etiqueta dada.
 *
 *    @param abutton Botones asociados a las salidas
 *    @param label Etiqueta donde se escribira (en hexadecimal)
 *          el estado del canal seleccionado
 *    @param iChannelStatus Estado del canal afectado por la actualizacion
 *
 *    @return void
 */
void  RefreshAllOutputs(PtWidget_t  **abutton,  PtWidget_t  *label,  int
iChannelStatus);


/**
 *    RefreshAllInputs. Actualiza el estado del GUI en funcion del estado
 *    del canal especificado.
 *
 *    @param rect Array de widgets de tipo Rectangulo (que representan
 *          las entradas) a actualizar
 *    @param label etiqueta a actualizar
 *    @param iChannelStatus Estado del canal de entradas digitales
 *
 *    @return void
 */
void   RefreshAllInputs(PtWidget_t   **rect,   PtWidget_t   *label,   int
iChannelStatus);

/*******************************************************
 *    HILO DE REFRESCO DE ENTRADAS DIGITALES
 */

/**
 *    SetDigitalRefresh. Establece la tasa de refresco (en ms)
 *    de las entradas digitales.
 *
 *    @param iRefr Nueva tasa de refresco (en ms)
 *
 *    @return void
 */
void SetDigitalRefresh(unsigned int iRefr);

/**
 *    LaunchDigInputsThread. Lanza el hilo de refresco de las
 *    entradas digitales.
 *
 *    @return Devuelve EOK si todo va bien
 */
int LaunchDigInputsThread();

/**
 *    PauseDigInputsThread(). Pausa el hilo de refresco de las
 *    entradas digitales.
 *
 *    @return Devuelve EOK si todo va bien
 */
void PauseDigInputsThread();

/**
 *    ContinueDigInputsThread. Hace que el hilo de refresco de
```

```
 *    las entradas digitales continue su ejecucion.
 *
 *    @return Devuelve EOK si todo va bien
 */
void ContinueDigInputsThread();

/**
 *    isDigInputsThreadPaused. Permite conocer si el hilo de
 *    refresco de entradas digitales esta ejecutandose o no.
 *
 *    @return Devuelve EOK si todo va bien
 */
int isDigInputsThreadPaused();

/**
 *    SignalDigInputsThreadToEnd. Avisa al hilo de refresco de las
 *    entradas digitales que debe terminar su ejecucion.
 *
 *    @return void
 */
void SignalDigInputsThreadToEnd();

/**
 *    WaitDigInputsThreadTermination. Espera a que el hilo de
 *    refresco de las entradas digitales termine su ejecucion.
 *
 *    @return Devuelve EOk si todo va bien
 */
int WaitDigInputsThreadTermination();

#endif
```

4.3.3 Analog.h

```
/**
 *    Declaracion de las funciones helper para el manejo de
 *    la conversion A/D desde el GUI.
 *
 *    @file Analog.h
 *    @author Raul Jimenez Ruiz
 */

#ifndef _ANALOG_H_
#define _ANALOG_H_

#include "ablibs.h"
#include "abimport.h"
#include "proto.h"

#include "libproto.h"

//    en ms
#define MAX_ANALOG_SAMPLING_FREC    1000
#define MIN_ANALOG_SAMPLING_FREC    100

/*****************************************************
 *    FUNCIONES ASOCIADAS AL GUI
```

```
 */

/**
 *    min y max se expresan en mV
 */
void configureProgress(unsigned int channel,int min, int max);

/**
 *    Mapea value de 0..RM_AD_DATA_MAX_INT_VALUE a min..max del progress
 */
float mapADValue(unsigned int value, PtWidget_t * prog);

/**
 *    value se expresa en mV
 */
void refreshADOutput(unsigned int channel,unsigned int value);

/*******************************************************
 *    HILO DE REFRESCO DE LA CONVERSION A/D
 */

/**
 *    rate en ms (para el thread de refresco de los canales A/D)
 */
int setAnalogRefresh(int rate);

/**
 *    Lanza el hilo de refresco de las entradas analogicas
 */
int LaunchADThread();

/**
 *    Avisa al hilo de refresco que debe terminar su ejecucion
 */
void SignalADThreadToEnd();

/**
 *    Espera a que el hilo de refresco termine
 */
int WaitADThreadTermination();

/**
 *  Pausa el hilo A/D
 */
void PauseADThread();

/**
 *  Continua la ejecucion del hilo A/D
 */
void ContinueADThread();

/**
 *  Comprueba si el hilo A/D esta parado
 */
int isADThreadPaused();

#endif
```

4.3.4 Pacer.h

```
/**
 *    Declaracion de las funciones helper para el manejo de
 *    los contadores desde el GUI.
 *
 *    @file Pacer.h
 *    @author Raul Jimenez Ruiz
 */

#ifndef _PACER_H_
#define _PACER_H_

#include <stdlib.h>
#include <stdio.h>

#include "ablibs.h"
#include "abimport.h"
#include "proto.h"

#include "libproto.h"

/**
 *    lee el modo de operacion seleccionado por el usuario, para el
 *    contador indicado
 */
int getMode(int counter);

/**
 *    establece el modo de operacion para el contador especificado
 */
void setMode(int counter,int mode);

/**
 *    Activa el toggle button correspondiente a un contador y un modo
 */
void selectToggleButton(int counter, int mode);

/**
 *    Devuelve el valor del contador especificado en el widget 'numeric
 *    integer' del contador seleccionado
 */
int getNIValue(int counter);

/**
 *    Establece el valor del widget 'numeric integer' asociado al contador
 *    seleccionado
 */
void setNIValue(int counter, int val);

/**
 *    Recalcula el valor de la frecuencia de salida y lo escribe en el
 *    label corespondiente
 *
 */
void refreshOutputPacerFrec(int C1, int C2);

/**
 *    Escribe el valor de la base temporal
```

```
 */
void setTimeBase(int tB);

/**
 *    Si en=1 -> habilita los controles de la pestaÃ±a del pacer, si es =0
 *    los deshabilita
 */
void enablePacerWidgets(int en);

#endif
```

4.4 Aplicación GUI IRQ.

Se verá a continuación el código funete de algunos archivos que forman parte de la aplicación GUI de prueba de la captura mediante interrupciones, entre ellos, el que contiene la función de respuesta al evento "click" del boton "START" de la ventana principal de la aplicación y los que contienen los prototipos de las funciones para el manejo de las interrupciones.

4.4.1 btnStart_OnClick.c

```
/* Standard headers */
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>

/* Local headers */
#include "ablibs.h"
#include "abimport.h"
#include "proto.h"

#include "libproto.h"
#include "util.h"

int
btnStart_OnClick( PtWidget_t *widget, ApInfo_t *apinfo, PtCallbackInfo_t
*cbinfo )

      {
      int res;

      /* eliminate 'unreferenced' warnings */
      widget = widget, apinfo = apinfo, cbinfo = cbinfo;

      //    instalamos el manejador de señal
      printf("<btnStart> installSignalHandler\n");
      if(!installSignalHandler())
      {
            printf("-- Error instalando el manejador de señal\n");
            return( Pt_CONTINUE );
      }

      //    Para hacerlo "mejor", habria que leer todos los parametros que
      //    se van a cambiar y volver a dejarlo todo como estaba si se
      //    produce algun error.
```

```
//    configuramos el pacer a 4Hz, para tener 4 muestras por segundo
//    (suponemos que FCLK = 1Mhz) para ello, escribimos en los
//    contadores: c1=250 y c2=1000  --> f = FCLK / ( c1 * c2) = 4 Hz

//          primero, lo paramos
printf("<btnStart> pcm3718h_DisablePacer\n");
if((res=pcm3718h_DisablePacer())!=SO_NO_ERROR)
{
      printf("-- Error disabling pacer
            (OnStart)[%s]\n",errorToString(res));
      return( Pt_CONTINUE );
}

//          configuramos el contador 1
printf("<btnStart> pcm3718h_WriteCounter 1\n");
if((res=pcm3718h_WriteCounter(RM_PACER_SC_COUNTER1,250))!=
  SO_NO_ERROR)
{
      printf("-- Error configurando contador 1
            [%s]\n",errorToString(res));
      return( Pt_CONTINUE );
}

//          configuramos el contador 2
printf("<btnStart> pcm3718h_WriteCounter 1\n");
if((res=pcm3718h_WriteCounter(RM_PACER_SC_COUNTER2,1000))!=
  SO_NO_ERROR)
{
      printf("-- Error configurando contador 2
            [%s]\n",errorToString(res));
      return( Pt_CONTINUE );
}

//    especificamos que el ciclo de conversion comprenda los 16
//    canales
printf("<btnStart> pcm3718h_WriteADMuxScan\n");
if((res=pcm3718h_WriteADMuxScan(RM_AD_CHANNEL0,RM_AD_CHANNEL15))!=
  SO_NO_ERROR)
{
      printf("-- Error estableciendo el ciclo de conversion
            [%s]\n",errorToString(res));
      return( Pt_CONTINUE );
}

//    especificamos que el disparo de la conversion A/D sea el pacer
printf("<btnStart> pcm3718h_SetADTriggerSource\n");
if((res=pcm3718h_SetADTriggerSource(RM_AD_CONTROL_TRIGSRC_PACER))!=
  SO_NO_ERROR)
{
      printf("-- Error estableciendo el trigger source por pacer
            [%s]\n",errorToString(res));
      return( Pt_CONTINUE );
}

//    al final, habilitamos el pacer
printf("<btnStart> pcm3718h_EnablePacer\n");
if((res=pcm3718h_EnablePacer())!=SO_NO_ERROR)
{
      printf("-- Error enabling pacer [%s]\n",errorToString(res));
      return( Pt_CONTINUE );
```

```
      }

      printf("<btnStart> OK\n");

      return( Pt_CONTINUE );
      }
```

4.4.2 util.h

```
//
//    Utilidades
//

////////////////////////////
#ifndef _UTIL_H_
#define _UTIL_H_
////////////////////////////

/**
 * Escribe el valor de uno de los 16 widgets de tipo PtNumericFloat
 */
void nfSetValue(int ID, double val);

/**
 * Instala el manejador de señal
 */
int installSignalHandler(void);

/**
 * Desinstala el manejador de señal
 */
void uninstallsignalHandler(void);

////////////////////////////
#endif
////////////////////////////
```

4.4.3 util.c

```
/**
 *    Implementa las funciones de util.h
 *
 */

////////////////////////////
#include <signal.h>
#include <process.h>

#include "ablibs.h"
#include "abimport.h"
#include "proto.h"

#include "libproto.h"

#include "util.h"
```

```
////////////////////////////

sigset_t setIRQ;
struct sigaction saIRQ;

////////////////////////////
void nfSetValue(int ID, double val)
{
      double aux=val;

      switch(ID)
      {
            case 0:
                  PtSetResource(ABW_nfCanal0,Pt_ARG_NUMERIC_VALUE,&aux,0);
                  break;
            case 1:
                  PtSetResource(ABW_nfCanal1,Pt_ARG_NUMERIC_VALUE,&aux,0);
                  break;
            case 2:
                  PtSetResource(ABW_nfCanal2,Pt_ARG_NUMERIC_VALUE,&aux,0);
                  break;
            case 3:
                  PtSetResource(ABW_nfCanal3,Pt_ARG_NUMERIC_VALUE,&aux,0);
                  break;
            case 4:
                  PtSetResource(ABW_nfCanal4,Pt_ARG_NUMERIC_VALUE,&aux,0);
                  break;
            case 5:
                  PtSetResource(ABW_nfCanal5,Pt_ARG_NUMERIC_VALUE,&aux,0);
                  break;
            case 6:
                  PtSetResource(ABW_nfCanal6,Pt_ARG_NUMERIC_VALUE,&aux,0);
                  break;
            case 7:
                  PtSetResource(ABW_nfCanal7,Pt_ARG_NUMERIC_VALUE,&aux,0);
                  break;
            case 8:
                  PtSetResource(ABW_nfCanal8,Pt_ARG_NUMERIC_VALUE,&aux,0);
                  break;
            case 9:
                  PtSetResource(ABW_nfCanal9,Pt_ARG_NUMERIC_VALUE,&aux,0);
                  break;
            case 10:
                  PtSetResource(ABW_nfCanal10,Pt_ARG_NUMERIC_VALUE,&aux,0);
                  break;
            case 11:
                  PtSetResource(ABW_nfCanal11,Pt_ARG_NUMERIC_VALUE,&aux,0);
                  break;
            case 12:
                  PtSetResource(ABW_nfCanal12,Pt_ARG_NUMERIC_VALUE,&aux,0);
                  break;
            case 13:
                  PtSetResource(ABW_nfCanal13,Pt_ARG_NUMERIC_VALUE,&aux,0);
                  break;
            case 14:
                  PtSetResource(ABW_nfCanal14,Pt_ARG_NUMERIC_VALUE,&aux,0);
                  break;
            case 15:
                  PtSetResource(ABW_nfCanal15,Pt_ARG_NUMERIC_VALUE,&aux,0);
                  break;
```

```
	}
}

////////////////////////////
void signalHandlerIRQ(int signo, siginfo_t *datos, void *extra)
{
	//	suponemos que:
	//	- la señal empleada es SIGRTMIN
	//	- los canales estan configurados como -10..10 V

	int aux,canal,val;
	if(signo==SIGRTMIN)
	{
			aux=datos->si_value.sival_int;
			canal=(aux & 0xFFFF) >>12;
			// el canal esta en los 4 bits mas signif

			val=aux & 0x0FFF;
			// el valor esta en los 12 bits menos signif

			nfSetValue(canal, -10.0 + 20.0 * (double)val/4095.0);
	}
}

////////////////////////////
int installSignalHandler(void)
{
	int *irq;
	int iIRQ;

	//	primero se instala el manejador
	saIRQ.sa_sigaction=signalHandlerIRQ;
	sigemptyset(&saIRQ.sa_mask);
	saIRQ.sa_flags=SA_SIGINFO;
	printf("<util> llamada a sigaction\n");
	if(sigaction(SIGRTMIN,&saIRQ,NULL)==-1)
		return 0;

	sigemptyset(&setIRQ);
	sigaddset(&setIRQ,SIGRTMIN);
	printf("<util> llamada a sigprocmask\n");
	if(sigprocmask(SIG_UNBLOCK,&setIRQ,NULL)==-1)
		return 0;

	//	luego se configura la tarjeta
	printf("<util> llamada a pcm3718h_SetUserProcessPID\n");
	if(pcm3718h_SetUserProcessPID(getpid())!=SO_NO_ERROR)
		return 0;

	PtGetResource(ABW_niIRQ,Pt_ARG_NUMERIC_VALUE,&irq,NULL);
	if(irq==NULL)
		return 0;

	switch(*irq)
	{
		case 2: iIRQ=RM_AD_CONTROL_IRQ2; break;
		case 3: iIRQ=RM_AD_CONTROL_IRQ3; break;
		case 4: iIRQ=RM_AD_CONTROL_IRQ4; break;
		case 5: iIRQ=RM_AD_CONTROL_IRQ5; break;
		case 6: iIRQ=RM_AD_CONTROL_IRQ6; break;
```

```
            case 7: iIRQ=RM_AD_CONTROL_IRQ7; break;

            default: iIRQ=-1;
      }
      printf("<util> llamando a pcm3718h_SetADTransferMode con irq=%d
            (==%d)\n",iIRQ,*irq);
      if(iIRQ==-1 ||
        pcm3718h_SetADTransferMode(RM_AD_TRANSFERMODE_IRQ,iIRQ)!=
        SO_NO_ERROR)
            return 0;

      return 1;
}

////////////////////////////
void uninstallSignalHandler(void)
{
      //    se quita el manejador
      sigemptyset(&setIRQ);
      sigaddset(&setIRQ,SIGRTMIN);
      sigprocmask(SIG_BLOCK,&setIRQ,NULL);

      //    luego informamos a la tarjeta
      pcm3718h_SetADTransferMode(RM_AD_TRANSFERMODE_PROGRAM,0);
      pcm3718h_ClearUserProcessPID();

      return;
}
```

4.4.4 wMain_Close.c

```
/* Standard headers */
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>

/* Local headers */
#include "ablibs.h"
#include "abimport.h"
#include "proto.h"

#include "libproto.h"
#include "util.h"

int
wMain_Close( PtWidget_t *widget, ApInfo_t *apinfo, PtCallbackInfo_t *cbinfo
)

      {

      /* eliminate 'unreferenced' warnings */
      widget = widget, apinfo = apinfo, cbinfo = cbinfo;

      //    primero, paramos el pacer (= la conversion A/D)
      if(pcm3718h_DisablePacer()!=SO_NO_ERROR)
      {
```

```
            printf("-- Error disabling pacer\n");
            return( Pt_CONTINUE );
      }

      //    desinstalamos el handler de la seÃ±al
      uninstallSignalHandler();

      //    especificamos que el disparo de la conversion A/D sea por
      //    software
      if(pcm3718h_SetADTriggerSource(RM_AD_CONTROL_TRIGSRC_PACER)!=
        SO_NO_ERROR)
      {
            printf("-- Error estableciendo el trigger source por
                  software\n");
            return( Pt_CONTINUE );
      }

      return( Pt_CONTINUE );

      }
```

5 Contenido del CD-ROM

Además del texto de la memoria del Proyecto en formatos Word y PDF, se incluye en el CD-ROM adjunto el código fuente desarrollado, así como las API de programación en formato HTML.