2 Placa Objetivo y Entorno de Desarrollo. Banco de Pruebas.

Este capítulo se divide en tres partes, que se encuentran relacionadas entre sí. La primera de ellas nos muestra la placa de evaluación, que realiza la función de soporte físico de este proyecto. La segunda se centra en el estudio de los diferentes entornos de desarrollo entre los que se podía optar, pero haciendo un mayor énfasis en la aplicación que, finalmente, se utilizó. Y la tercera tiene como objetivo obtener las limitaciones de tiempo y de código que soporta este sistema.

2.1 Placa del Sistema Objetivo.

2.1.1 Introducción.

En este apartado de la memoria se describirá la placa del sistema objetivo desde el punto de vista de su funcionamiento y de la relación con los elementos que a ella están conectados, como son, la fuente de alimentación y el compilador cruzado. La placa dispone de unos conectores de configuración ("*jumpers"*) que definen el modo de funcionamiento, así como de unos diodos luminiscentes (*"LED's"*) que muestran su estado.

Por otro lado, se comentará la documentación que proporciona el fabricante de la placa a través de las herramientas de desarrollo (*"developer kit"*), que consta de un conjunto de discos compactos que contienen diferentes compiladores cruzados y de una serie de archivos de ayuda [B1] en formato *hipertexto*.

Esta documentación incluye un programa ejemplo que muestra a los *LED's* brillando alternativamente, partiendo de un mapa de memoria, y que se ejecutará en el sistema objetivo.

2.1.2 Material de laboratorio.

Para llevar a cabo esta parte se ha hecho uso del siguiente material:

- Estación *PC* bajo *WindowsXP*.
- Entorno de desarrollo *Advanced Developer Suite v1.2*.
- Compilador cruzado *Metrowerks CodeWarrior* y el depurador *AXD* de *MetroWerks*.
- Placa de Pruebas *AT91EB40A* de *ATMEL* que incluye el microprocesador *ARM7TDMI*.
- Documentación en soporte informático sobre la placa de pruebas.
- Fuente de Alimentación *TRQ TC 1305* cuyo nivel de tensión continua de salida es programable a 4.5, 6, 7.5, 9 y 12 V con 1300 mA de corriente máxima de salida.
- Multímetro: *Digital Multimeter DT-830B*.

- Cables de conexionado.

2.1.3 Documentación y Compiladores.

En la documentación que se incluye en el *kit* de desarrollo [B1] aparecen varios compiladores, aquí los describiremos atendiendo a la interfaz con el usuario centrándonos en la facilidad de manejo, siempre respecto a la generación y edición del código fuente para su posterior compilado y descarga en el sistema objetivo.

2.1.3.1 Compiladores.

Los compiladores cruzados proporcionados son:

- *High C/C++ ™ ARM. Embedded Development Toolset* de *MetaWare Incorporated.*
- *Multi2000 Integrated Development Environment for Atmel Arm AT91* de *Green Hills Software Inc*.
- *ARM Developer Suite (Evaluation Version 1.1)* de *ARM Limited*.

Todos ellos son compatibles con el sistema operativo *WindowsXP*, presentando interfaces gráficas con diferente grado de facilidad de manejo e intuición para su aprendizaje.

El primero de los tres compiladores mostrados ofrece un formato menos amigable que el resto puesto que la interacción con el mismo se lleva a cabo mediante un intérprete de comandos; por otro lado, la ayuda que se facilita no hace mención a este compilador. Por estas cuestiones nos obliga a situarlo en el peor puesto de la comparativa, desechando su posterior utilización.

Los otros dos compiladores presentan una interfaz de usuario muy similar al de resto de paquetes informáticos desarrollados para *Windows*, con lo que resulta mucho más familiar e intuitivo. Además, la ayuda del *kit* de evaluación de *ATMEL* sí incluye, de forma detallada, los pasos a seguir para configurar cualquiera de estos dos compiladores cruzados.

Por problemas críticos en la gestión de la licencia para el compilador de *Green Hills Software Inc.*, se elige la opción de *ARM Limited*.

2.1.3.1.1 Instalación de los Compiladores.

El objetivo de este apartado es presentar las pautas seguidas en la instalación de cada uno de los compiladores que se mostraron en el punto anterior, haciendo hincapié en la problemática que éstas suscitaron y en las soluciones que se plantearon.

2.1.3.1.2 ARM Developer Suite (Evaluation Version 1.1) de ARM Limited.

Este paquete de software se instaló siguiendo esta ruta de directorio: *"D:\SW_FCS\ARM\ADSv1_1*. Al principio del proceso de instalación el gestor del fichero de licencia nos indicó que éste estaba corrupto, solicitando que se descargase la licencia operativa desde el proveedor del software, siendo los sitios de Internet los siguientes:

- http://www.c-dilla.com/support/lms.html
- ftp://ftp.macrovisioneruope.com/outgoing/cdalms.exe

El archivo de licencia se denomina *"cdalms.exe"*, pero, cuando intentamos ejecutarlo para que se gestionase la licencia correctamente, *WindowsXP* nos muestra un cuadro de error con el texto *"(1615:6): usuarios de Windows NT deben comprobar los derechos de acceso con el Administrador. Si el problema persiste, reinicie su PC"*. El mensaje resulta extraño puesto que la cuenta de usuario dentro de *Windows XP* posee permisos de *Administrador*; aún así, reinicio el *PC* tal y como lo solicita el mensaje.

Reinstalamos el software de *"ARM Limited"*, pero el problema con el gestor del fichero de licencia persiste, lo que nos impide trabajar con este compilador cruzado, al menos, de momento.

La solución definitiva y satisfactoria se presenta gracias a la distribución de la versión 1.2 de *"ARM Developer Suite"* (*ADS*). Lo descomprimimos en la carpeta *"D:\SW_FCS\temporal"* generándose dentro de ésta el directorio *"ARM.developer.suite v1.2"*. En él aparece el fichero *"setup.exe"* que, tras su ejecución, instala satisfactoriamente todo el paquete de software en la carpeta indicada: *"D:\SW_FCS\ARM\ADSv1_2"*.

A partir de entonces sí se podía abrir el compilador, y trabajar con él, sin que apareciera un cuadro de error indicando que no se hubiera gestionado correctamente la licencia, bajo *WindowsXP*, con el consiguiente cierre del programa.

2.1.3.1.3 Multi2000 Integrated Development Environment for Atmel Arm AT91 de Green Hills Software Inc.

Mediante el disco compacto que proporciona este fabricante se inicia el proceso de instalación del paquete de software en la ruta de directorio que se le especifica, sea: *"D:\SW_FCS\GHS"*. Durante dicho proceso se nos muestra un cuadro de diálogo que indica que se ejecute *"License Generator"* para que solicitemos del fabricante el fichero de licencia correspondiente.

Al ejecutarse, el programa descarga un archivo de licencia del sitio en Internet del fabricante. Al tratarlo se nos informa que se trata de una licencia defectuosa y, por lo tanto, no operativa. Esto resulta sorprendente, considerando que se está hablando de la propia licencia que reside en la base de datos del fabricante. Se nos impide trabajar con este paquete de software.

2.1.3.1.4 High C/C++ ™ ARM. Embedded Development Toolset de MetaWare Incorporated.

Aunque la interfaz con el usuario de este paquete de desarrollo nos resultó ardua e incómoda a la hora de trabajar con ella, se ha decidido exponer brevemente los pasos que nos llevaron a la instalación de la misma.

Introduciendo el *CD* que *"MetaWare Toolset"* nos proporciona instalamos el software en la siguiente ruta de directorio: *"D:\SW_FCS\hcarm_4.5a"*, cumplimentando los campos nombre de usuario como *"Sergio Latorre"* y nombre de compañía como *"EsCuela de InGenieros – Proyecto AERO1 FCS"*. Tras finalizar el proceso de instalación, el programa se abre sin problemas aparentes.

2.1.3.1.5 Sistemas Operativos y otros Compiladores.

- Sistemas Operativos: uCLinux, eCos y ucOS II.

Estos sistemas operativos poseen la licencia de código *GNU*. Al tratarse de programas relativamente más pequeños y que no se encuentran sometidos a las leyes del mercado de la oferta y la demanda, permiten generar códigos máquina más compactos y optimizados. Por otro lado, no ofrecen garantías de un correcto funcionamiento y presentan una interfaz de usuario terriblemente incómoda basado en el modo comando.

- uCLinux.

Se trata de un sistema operativo imbuido –*"embedded"*, usando terminología anglosajona– basado en el núcleo del conocido *Linux*, pero simplificado para su uso en sistemas basados en microcontroladores.

Permite la generación de código *C++* desde la consola (monitor y teclado) mediante el editor *"vi"* de *Linux* para la máquina de *ARM*, pero resulta incompatible con *Windows*. Esto exige instalar en una partición nueva del disco duro una distribución de *Linux*. Esto se realizó en un PC, basando en la tecnología *Intel x86* de un *Pentium* a 133 Mhz; pero, por cuestiones que exceden mi responsabilidad, los discos duros de dicha estación de trabajo no estaban disponibles posteriormente. Debido a esto, esta opción no se siguió estudiando.

La distribución de este *"micro-linux"* residía en Internet dentro de un paquete compatible con *"Red Hat Linux"* cuyo nombre era: *"uClinux-1.0.0-1.i386.rpm"*.

- eCos.

El nombre de este sistema operativo consiste en un acronismo cuyas siglas representan *"embedded Configurable operating system"*, o lo que es equivalente: *"sistema operativo imbuido configurable"*. Se trata de un sistema operativo de código abierto, que admite multitud de configuraciones, siendo específico para aplicaciones que trabajan en sistemas imbuidos.

El sistema operativo *"eCos"* es compatible tanto con *Linux* como con *Windows*, pero exige que la versión mínima para *Linux* sea la de la distribución *"Red Hat 6.1"* o que sea "*Windows NT 4.0"*, respectivamente.

La documentación de este sistema se encuentra en su página de Internet [B3], siendo "http://sources.redhat.com/ecos*"*. En ella, encontramos toda la información correspondiente en el archivo *"eCosDoc131.zip"* y la distribución asociada cuyo nombre es *"ecos-1.3.1-1.i386.rpm"* para compilar bajo *"Red Hat Linux"*.

- ucOS II.

Se trata de un sistema operativo imbuido que permite compilar desde la estación PC. También comparte la licencia *GNU* de código abierto siendo distribuido desde [B4] "http://embedded.n3.net*"* a través de un fichero comprimido de nombre *"arm7_ucos_1.0.zip"*.

- Otros Compiladores: CodeWright IDE.

Se trata de un entorno de desarrollo integrado en un único paquete de software, tal y como lo indican las siglas *IDE* que significan *"Integrated Development Environment"*, bajo terminología inglesa.

Seguimos los pasos del proceso de instalación gracias al sistema de autoarranque del *CD* que proporciona el fabricante. Seleccionamos como carpetas de destino las que poseen las siguientes rutas de directorio: *"D:\SW_FCS\hcarm_4.5a\CodeWrightInstall"* y *"D:\SW_FCS\Starbase\CodeWright"*; la primera da cabida a los archivos relativos a la instalación del entorno de desarrollo; mientras que la segunda recoge la base de datos de los controladores de los dispositivos, en donde se instalan las librerías *"Code Sense"*.

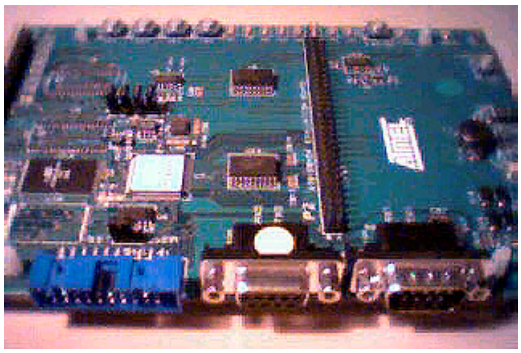
El nombre de usuario y el de la compañía se definen con los mismos datos que en el caso del compilador de *"MetaWare Incorporated"*.

2.1.4 Configuración, Indicadores y Conexionado de la Placa.

El objetivo de este apartado se centra en la descripción, de cara al usuario, del *kit* de desarrollo que nos proporciona el fabricante *ATMEL*. Para ello se indicará cuáles son los puertos de comunicación que permiten el trasiego de información entre la estación en la que reside el entorno de desarrollo, a la que llamaremos *"host"*, y la placa que posee el sistema objetivo, denominémosla *"target"*; cuáles son los indicadores hacia el usuario que ésta posee explicando su significado y, cuáles son los conectores o *"jumpers"* que permiten programar el modo de funcionamiento de la placa.

2.1.4.1 Conexionado.

La placa posee dos puertos serie *RS–232*, uno con conector hembra (*"Serial A"*) y otro con conector macho (*"Serial B"*), con interfaces *USART*[1], es decir, *"Transmisor y Receptor Síncrono y Asíncrono Universal"*. Asimismo, se facilita la conexión con otras tarjetas, mediante acceso directo a posiciones del mapa de memoria del microprocesador *ARM7TDMI* de *ARM*, gracias a un conector con el formato de la interfaz *JTAG*.



El *kit* de desarrollo también proporciona un cable serie de 9 patillas, con terminaciones hembra y macho, para conectar la placa al puerto serie de la estación *PC* o los dos puertos serie de la placa entre sí.

Un conector imprescindible, que suele obviarse, es el de alimentación; se trata de un conector normalizado hembra del tipo *J-1*.

2.1.4.2 Indicadores.

La placa presenta una serie de indicadores *"LED's"* (*"Light Emitter Diode"*) que muestran su estado y su correcto funcionamiento. Los denotaremos con una *'D'* seguida de un subíndice que los enumera.

[1] 'USART': acronismo de: *"Universal Synchronous and Asynchronous Receiver Transmitter*".

De acuerdo con los manuales de referencia dados por el fabricante, el significado de los diferentes *"LED's"* se asocia a las partes que forman la placa tal y como sigue:

- *'D_1'*: A la memoria *SRAM* interna.
- *'D_2'*: A la memoria *SRAM* externa, en el caso de que exista.
- *'D_3'*: A la memoria *Flash* externa.
- *'D_4'*: A la memoria *EEPROM* con la interfaz de acceso *"two-wire"*.
- *'D_5'*: A la *USART*.
- *'D_6'*, *'D_7'* y *'D_8'*: Están reservados, es decir, pueden programarse.

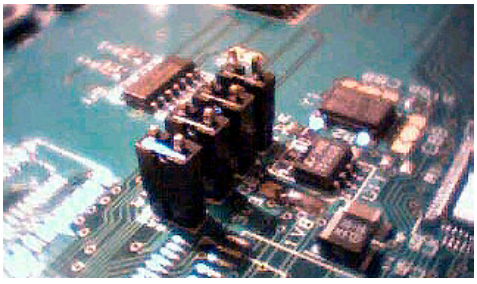
Estos *"LED's"* se activan indicando el correcto funcionamiento de cada elemento de la placa durante la prueba tras el arranque de la misma.

- *'D_{10}'* o *"RST"*: Está asociado al circuito que gestiona la señal de inicio o de *"reset"*, luego se enciende cuando la señal *"reset"* está activada.
- *'D_{11}'* o *"PWR"*: Se enciende cuando se alimenta adecuadamente la placa a través del conector *J-1*.

2.1.4.3 "Jumpers".

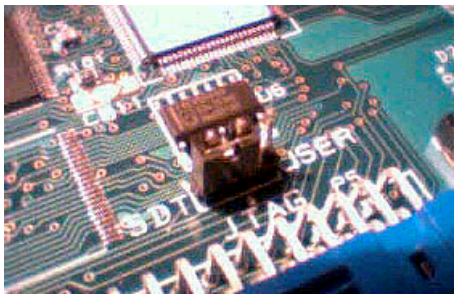
Éstos no son más que conectores metálicos entre distintas patillas, con nombre propio, perpendiculares a la placa. Según las patillas que queden unidas eléctricamente, el modo de funcionamiento de la placa de evaluación cambiará entre unos ya predefinidos. La notación de estos conectores constará de las siglas *"JP"* seguidas de un subíndice. Existen cinco de estos *"jumpers"* sobre la placa, de los cuales uno de ellos es crucial; sean:

- *'JP_1'*: Tiene dos posiciones, según las patillas que conecte. La primera de ellas se define como *"USER"*, mientras que la segunda se denomina *"STD"*.
- *'JP_{5A}'* y *'JP_{5B}'*: Si los quitamos y colocamos adecuadamente un multímetro podemos medir el consumo de corriente consumida por la placa (señal *"V_{DDIO}"*).
- *'JP_{7A}'* y *'JP_{7B}'*: De forma análoga a los conectores anteriores, a través de ellos se puede medir el consumo del núcleo del microcontrolador (señal *"V_{DDCORE}"*).



El conector *'JP_1'* define los dos modos de funcionamiento de la placa de evaluación. Describámoslos brevemente:

- *"STD"*: Se trata del modo *standard* en el que la placa inicia en una posición de memoria tal que gestiona un sistema de chequeo y facilita que un sistema operativo rudimentario ejecute el programa previamente descargado.
- *"USER"*: En este caso el modo de funcionamiento tras el arranque de la placa se denomina de *usuario*, empezándose a trabajar en la posición de memoria de usuario. Ésta es la zona en la que se descargan los programas desde la estación *"host"*.



La memoria que almacena, tanto el programa de chequeo como el de usuario, posee las mismas características, siendo *EEPROM*; luego puede ser programable eléctricamente. A la hora de descargar programas, resulta de vital importancia que *'JP_1'* esté en la posición *"USER"* para que la copia en la memoria de la placa se realice sobre la memoria de usuario, evitándose que se inicie sobre la memoria del fabricante o *"firmware"*, lo que eliminaría el programa de chequeo.

2.1.4.4 Pulsadores.

Los pulsadores o *"switch buttons"* permiten actuar sobre la placa durante el proceso de funcionamiento. La placa posee 6 de estos pulsadores; algunos, por su importancia, tienen nombre propio; otros, están enumerados con los símbolos *'SW_1'*, *'SW_2'*, *'SW_3'* y *'SW_4'*. Siendo sus significados:

- *"CLR_RST"*: Al pulsarlo cesa la señal de inicio o *"reset"* del circuito de gestión de arranque.
- *"RESET"*: Al pulsarlo se solicita a la rutina de gestión de arranque que se procese, provocando que el sistema objetivo se inicie.
- *'SW_1'*: Activa el software de chequeo funcional o *FTS* (*"Functional Test Software"*).
- *'SW_2'*: Activa el descargador de *SRAM* (*"SRAM downloader"*).
- *'SW_3'*: Activa el cargador de *Flash* (*"Flash uploader"*).
- *'SW_4'*: Se trata de un pulsador reservado, por lo tanto, programable por el usuario.

La activación de cada uno de los subsistemas requiere que el pulsador se mantenga accionado –encendiéndose todos los *"LED's"*– y que, posteriormente, se suelte quedándose únicamente encendido el indicador correspondiente.

2.1.5 Fuente de Alimentación y Calibrado.

De acuerdo con los manuales de usuario [B1] que proporciona el fabricante se define una serie de requisitos previos y de consideraciones de uso para poder alimentar correctamente al sistema.

Debido a la sensibilidad electrostática de estos sistemas, la placa se provee con una bolsa aislante que la protege durante su almacenaje; asimismo, mientras se trabaje con ella hay que ser escrupuloso en que no se produzcan contactos metálicos con las patillas al usar el material de laboratorio, tal y como pueden ser destornilladores, cables, etc.

La placa posee un conector del tipo *J-1*. Se exige que el nivel de tensión de alimentación quede acotado entre 7 y 9 V en continua, así como que la corriente de alimentación sea de 1000 mA aproximadamente. Cuando se suministra la potencia necesaria a través de la red de distribución eléctrica, no tenemos problemas de alimentación, sin más que atender a que el rectificador sea el adecuado en sus características de salida. En cambio, para un sistema portátil como es una aeronave, se exigen criterios de diseño más complejos y robustos que incluyen la disyuntiva entre peso y potencia.

La fuente de alimentación hace las funciones de rectificación y transformación de la señal proveniente de la red de distribución eléctrica. Como ya se ha indicado, esta fuente se puede programar en tensión de salida entre varios posibles valores discretos. Usando el destornillador, retiramos la carcasa protectora de la fuente, se puede observar que el sistema de programación se basa en un conjunto de contactos metálicos, el cual selecciona un número de vueltas determinado de la bobina del transformador según su posición. Debido a su simplicidad quisimos asegurar que la tensión de salida era la indicada en el reverso del rectificador; para ello lo cargamos con la placa en diferentes estados de funcionamiento, realizando las medidas también descargado.

Los resultados son:

- Con la fuente en abierto (descargada):

Valor Seleccionado de Tensión (V)	Valor medido (V)
4.5	6.40
6	7.81
7.5	9.52
9	11.30
12	14.85

Se observa que existe una clara diferencia entre el valor programado, según sus etiquetas, y el valor real ofrecido.

- Con la fuente cargada y el programa *FTS –"Function Test Software"* – ejecutándose:

Valor Seleccionado de Tensión (V)	Valor medido (V)	Notas
6	7.05	Chequeo *FTS* OK
7.5	8.61	Chequeo *FTS* OK

Destacamos que, debido al consumo de potencia que realiza la placa, el valor de tensión continua de salida disminuye ligeramente frente al caso anterior. No se ha medido para otros valores de tensión programados porque la tensión medida queda fuera del intervalo de alimentación adecuada que nos indica el fabricante.

- Con la fuente cargada y el programa de prueba *"LED's swinging"* ejecutándose:

Valor programado de Tensión (V)	Valor medido (V)	Notas
6	7.05	Chequeo *FTS* OK: *LED's* bailando
7.5	8.75	Chequeo *FTS* OK: *LED's* bailando

Se ha creído adecuado realizar esta tercera prueba puesto que pretende emular el comportamiento de la fuente cuando la placa de evaluación ejecuta un código de usuario. Nótese que no existe, prácticamente, diferencia en los valores medidos entre las dos últimas

configuraciones; esto nos indica el correcto funcionamiento de regulación del circuito de alimentación que discurre por la placa.

Atendiendo a los datos medidos con el multímetro, en los distintos casos, se decide programar la fuente en la posición indicada con la etiqueta *'6'*, para que ofrezca la tensión de salida mínima dentro del intervalo de 7 a 9 V –siendo el indicado por el fabricante–.

2.1.6 Puesta en funcionamiento.

Para comenzar a trabajar con el entorno de desarrollo *ADS* se hace imprescindible comprobar que la placa *AT91EB40A* funcione correctamente. Para ello *"ATMEL Limited"*, el fabricante de la misma, nos proporciona una ayuda muy útil y fácil de seguir en el disco compacto *ARM THUMB* [B1].

Al insertar ese *CD* se abre automáticamente la ayuda en formato hipertexto. En la primera página que se nos muestra aparece una distribución de las distintas partes principales en las que se divide la ayuda, pinchemos sobre la que tiene la etiqueta *"getting started"*, que en español se expresa como *"cómo comenzar"*. Seleccionando, del menú que se despliega, la opción *"EB40A"* se abre la página cuyo título dice *"GETTING STARTED FOR THE AT91EB40A: TOOL CHAIN"*, es decir, se trata de la cadena que hay que seguir para iniciar, por primera vez, la placa de forma correcta.

Se nos presentan cuatro opciones de trabajo, según el entorno de desarrollo que vayamos a usar o el sistema operativo *"firmware"* con el que ya esté configurado el sistema. Respecto al software de trabajo podemos optar entre *"ARM ADS v1.1"* o *"Green Hills v3.01"*; con el primer caso podremos elegir entre *"Angel"* o *"Multi-Ice"* como sistema operativo del fabricante; mientras que en el segundo disponemos de las opciones *"Angel"* y *"Raven"*.

Debido a que no disponemos de una licencia válida, ni la proporciona el fabricante desde su sitio en Internet, la posibilidad de trabajar con el paquete de *"Green Hills"* queda descartada. Dentro de la casuística que presenta el software de *"ARM"* elegimos el sistema operativo *"Angel"*, puesto que es éste el que está instalado en la placa de trabajo, según la documentación que el fabricante proporciona desde su sitio en Internet.

El proceso a seguir antes de desarrollar algoritmos para el sistema objetivo se divide en seis fases; detallaremos cada una de sus partes en las que se añadirá una serie de consideraciones de carácter práctico.

2.1.6.1 Arranque y Chequeo de la Placa de Evaluación.

Nos aseguramos que el conector programable *'JP_1'* de la placa configure el modo de funcionamiento denominado *"STD"* y que la fuente de alimentación quede fijada en la posición indicada por un *'6'*. Conectamos la fuente, mediante su conector *J-1*, al homólogo en la placa y su enchufe a la red eléctrica. Nótese que el mismo rectificador posee un interruptor, que será el último en accionarse dando paso a la potencia de alimentación.

Tras esto, el software de arranque –o *"boot"*, en terminología anglosajona– se inicia, mostrando su funcionamiento a través de los indicadores *"LED's"*, haciéndolos parpadear a todos una vez al mismo tiempo. Cuando el *"boot"* ha terminado sólo quedan encendidos *'D_1'*, *'D_{10}'* y *'D_{11}'*. Esto nos indica que el arranque se ha realizado correctamente.

En esta fase aparece, por primera vez, la sensibilidad del conector *J-1* puesto que ante el más leve movimiento o roce dejaba de hacer su función correctamente. Se manifestó al apagarse la placa sin motivo aparente, es decir, *'D_{11}'* o *"PWR"* no brillaba. Usando el multímetro medimos la caída de tensión desde el positivo del conector *J-1* de la placa al nivel de referencia o *"tierra"*; como era de esperar resultó ser nulo. Esta medida sirvió para eliminar, directamente, otra posible causa para esta anomalía en el funcionamiento.

Entre los dos cables de los que disponía la fuente de alimentación uno de ellos era extraíble; éste permitía la conexión con la toma de corriente de la red eléctrica. El fabricante lo provee, dentro de la caja que contiene el rectificador, no conectado; lo enchufamos suavemente en el lugar adecuado.

La primera vez que probamos a activar la placa, ésta no respondía; pero el transformador se calentó alarmantemente: Se desconectó de inmediato la fuente gracias al interruptor que incorpora. Posteriormente, desenchufamos la toma de la red y el conector *J-1* de la placa; con ayuda de un destornillador de estrella retiramos la carcasa protectora, con lo que pudimos ver fácilmente que los contactos con el cable extraíble se encontraban en cortocircuito. Es decir, la salida del rectificador sufría un puente, subsanándose el problema con un destornillador plano. Se observó que, al contactar eléctricamente ambos contactos, saltó un arco voltaico por el extremo del destornillador; esto demuestra que la bobina del transformador y sus condensadores estaban cargados.

Colocando la carcasa y volviendo a conectar los enchufes necesarios pudimos completar esta parte de forma satisfactoria.

La segunda parte de esta fase consiste en la ejecución del *"FTS"* o programa del chequeo funcional. Para ello se requiere que los puertos *"Serial A"* y *"Serial B"* estén conectados entre sí, de esta forma se comprueba el funcionamiento de éstos de forma independiente a otro sistema. Además, esta parte considera que la placa ya ha sido iniciada correctamente y que está operativa.

Para iniciar *"FTS"* se debe pulsar, al unísono, los botones *"RESET"* y *'SW_1'*, generándose la secuencia de arranque; mantenemos apretado *'SW_1'* que, al soltarlo, da comienzo al test funcional. Esto se manifiesta puesto que los *"LED's"*, desde *'D_1'* a *'D_5'*, parpadean una vez consecutivamente. Si un diodo hubiera parpadeado dos veces entonces el sistema asociado no ha pasado satisfactoriamente el chequeo; lo cual nunca nos ocurrió durante todo el proceso de trabajo y depuración con el sistema.

En la primera vez que pretendimos llevar a cabo esta parte se nos presentó un problema de carácter mecánico, es decir, los conectores de los puertos *RS-232* de la placa estaban demasiado próximos entre sí como para que la conexión, con el cable adjunto, pudiera efectuarse. La solución no podría ser otra que, utilizando el destornillador plano a modo de cizalla, recortar el borde de plástico de los conectores de dicho cable hasta que la conexión mecánica fuera factible. Forzar la conexión pudiera haber producido la rotura de las soldaduras de los conectores serie de la placa de evaluación, con el consiguiente retraso en tiempo y el gasto de material.

2.1.6.2 Instalar el sistema de desarrollo *"ARM ADS v1.2"*

En un principio se instaló la versión 1.2, sin más que extrayéndola del *CD* que proporciona el *kit* de evaluación [B1]. Todo el proceso de instalación se describe detalladamente en el apartado correspondiente.

2.1.6.3 Conectar el PC *"host"* a la Placa de Evaluación.

Apagamos la placa accionando el interruptor del rectificador para conectar el cable serie entre los puertos *COM1* del *"host"* y *"Serial A"* del *"target"*.

2.1.6.4 Instalar las Librerías Adecuadas.

Esto permite que desde el *"host"* se conozcan los controladores correspondientes a las partes funcionales de la placa *AT91EB40A*, junto con un grupo de programas de prueba para cada una de ellas. De esta forma, el entorno de desarrollo podrá acceder a los detalles y características propias del sistema objetivo concreto.

Para instalarlas se siguen los pasos indicados, claros y precisos, en la ayuda; éstos nos conducen a la extracción de los archivos que las contienen dentro de la ruta *"C:\"*. Finalmente, puede comprobarse que éstas han sido extraídas en la ruta de directorio *"C:\AT91\software\"*; esta carpeta estaba incluida dentro del mismo paquete *"ZIP"*. Para la extracción de las librerías no se podía elegir la carpeta de destino de las mismas, exigiéndose que el directorio fuese fijo.

2.1.6.5 Procesado de las Librerías.

Para tratar las librerías necesitamos trabajar con el entorno de desarrollo *"ARM ADS"*. Distinguiremos cuatro tipos de procesado: En el primero reconstruiremos una librería de dispositivo; en el segundo, una librería de controladores; en el tercero, un proyecto completo de la librería *AT91* y, en el último, la librería *"Tools"* o de herramientas.

2.1.6.5.1 Reconstrucción de una librería de dispositivo.

Mediante *"ADS v1.2"* abrimos el proyecto de nombre *"r40008_lib16.mcp"* que se encuentra en la ruta de directorio *"C:\AT91\Software\parts\r40008\"*. Debido a que las librerías están definidas para la versión *v1.1*, que es la que se proporciona con el *kit* de evaluación, el programa que ha sido instalado –versión *v1.2*– detecta que se trata de una versión anterior dando paso al proceso de actualización de las mismas.

Dentro del cuadro de diálogo que contiene todos los archivos que componen la librería seleccionamos la lengüeta denominada *"Targets"*; marcamos todas las opciones que nos

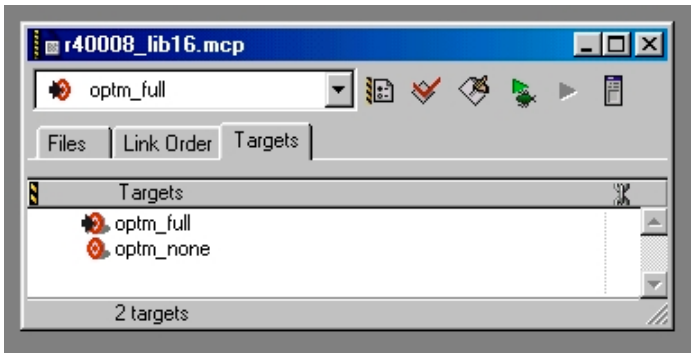


Figura: Ventana de Proyecto

brinda (*Ctrl+A*). Pulsando el botón *"Make"* o la opción homónima dentro del menú *"Project"* se inicia el proceso de compilado de la librería. A su término se muestra una ventana de mensajes con los errores y avisos, que en este caso se han dado cero errores y un aviso, el cual ignoramos tal y como nos señala la ayuda de referencia por tratarse de una directiva obsoleta que no se necesita en *"ARM ADS"*.

Cerramos la ventana de mensajes y el proyecto, es decir, a través de la opción *"Close"* del menú *"Project"*.

2.1.6.5.2 Reconstrucción de una librería de controladores.

Se procede de forma análoga con la librería *"lib_drv_16.mcp"* del directorio *"c:\at91\software\drivers\lib_drv\"* que en el caso anterior. También se actualiza la versión de esta librería y se compila sin errores ni avisos.

2.1.6.5.3 Reconstrucción de un proyecto completo.

El proyecto contiene un grupo de ficheros, entre ellos algunos se definen en lenguaje *C++*, que describen el comportamiento deseado que, en este caso, consiste en que los diodos luminiscentes brillen provocando un efecto de vaivén.

Para compilar estos archivos se abre el fichero *"led_swing.mcp"* que está contenido en *"c:\at91\software\projects\led_swing_eb40a\"*. Tras la actualización a la versión *v1.2* se procede a compilar todas las opciones de la pestaña *"Targets"*, pero se produce un error porque el programa no puede encontrar la librería de nombre *"r40008_lib16.a"*.

Con la intención de solventar este imprevisto buscamos el archivo mediante el sistema de búsqueda que proporciona *WindowsXP*, el cual nos indica que se encuentra en la ruta de directorio *"c:\at91\software\parts\r40008\r40008_llib16_Data\optm_full\"*. Sin más que incluir explícitamente la librería buscada, a través de la opción *"Add Files..."* del menú *"Projects"*, ya podemos compilar sin que aparezca ningún mensaje de error o de aviso.

2.1.6.5.4 Reconstrucción de la librería *"Tools"*.

Actuando de forma similar a los casos anteriores se abre el archivo *"flash_16x4.mcp"* situado en *"c:\at91\software\tools\flash_downloader\flash_16x4\"*. Seleccionando sólo la opción *"int_sram"* de la lengüeta *"Targets"*, la compilamos sin errores ni avisos.

2.1.6.6 Descargar un Programa de Prueba a la Placa y Ejecutarlo.

A partir de este punto se trabajará, además de con el equipo *"host"*, con el *"target"*; para ello se requiere la conexión vía *RS-232* entre ambos sistemas, con la consiguiente configuración previa del módulo de comunicaciones del depurador *AXD*.

En el entorno de desarrollo *"ARM ADS v1.2"*, mediante la opción *"Open"* del menú *"File"* abrimos el archivo *"led_swing.mcp"* de la ruta *"c:\at91\software\projects\led_swing_eb40a\"*. Dentro de la pestaña *"Targets"* hacemos uso de la opción *"Select All"* del menú *"Edit"* para indicar que queremos actuar sobre todas las variantes de dicha pestaña; posteriormente elegimos *"Make"* del menú *"Project"* para iniciar el proceso de compilado de las mismas.

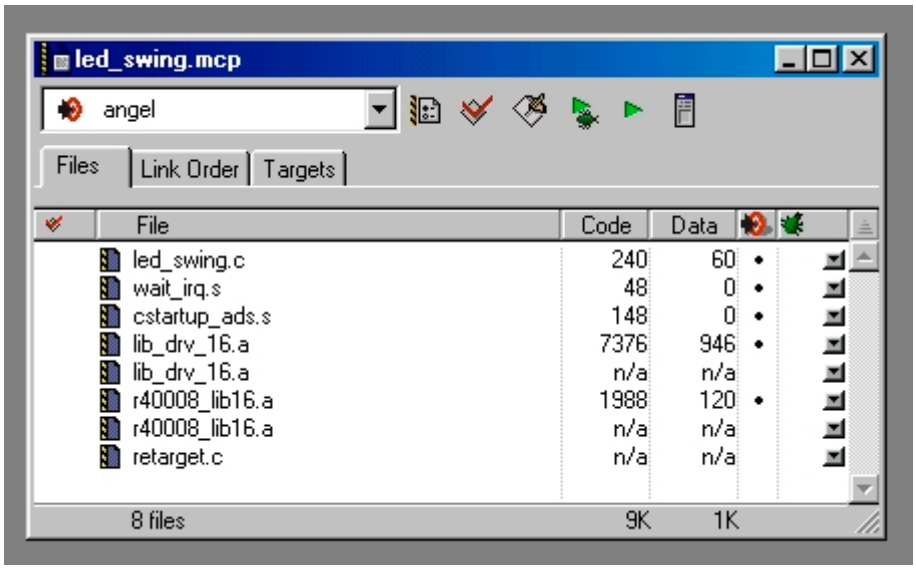


Figura: Archivos del Proyecto.

Lamentablemente, en la ventana de mensajes, se nos muestra que se ha incurrido en 17 errores debido a la existencia de símbolos indefinidos. Con la intención de determinar la fuente de estos errores compilamos cada una de las variantes independientemente, llegando a la conclusión que los 17 errores proceden únicamente de la variante denominada *"flash"*.

Gracias a que se trata de una librería proporcionada por el fabricante, confiamos en su buen hacer; luego se decide continuar con el proceso arrastrando los errores. Si no se pudiera proceder entonces deberíamos incluir explícitamente la librería que dé forma a los símbolos indefinidos, de manera análoga a como ya se realizó anteriormente con el fichero *"r40008_lib16.a"*.

Configuramos la placa en el modo *standard* y la alimentamos correctamente, asimismo conectamos el PC *"host"* con el sistema objetivo mediante el cable serie. Mediante la opción *"Debug"* del menú *"Project"* del compilador *"ARM ADS v1.2"* se inicia la aplicación del depurador *"ARM AXD"* , la cual se abre en una ventana independiente.

El depurador nos presenta su configuración por defecto, es decir, nos muestra el mensaje *"ARMulator ADS v1.2 (Build 805)"* a través de su ventana de registro. Se hace necesario indicarle a esta aplicación la configuración específica de su interfaz de comunicaciones con el sistema operativo *"Angel"*, así lograremos un depurado coherente que funcione correctamente. Esto se describirá en el capítulo correspondiente.

Dentro del depurador seleccionamos la opción *"Go"*, dos veces, del menú *"Execute"* para iniciar el proceso de seguimiento de la ejecución del código. Los indicadores *"LED's"* se encienden logrando el efecto de una onda; a su vez se da manifiesta actividad a través del puerto *"Serial A"*, gracias a que se iluminan los diodos asociados al envío (*"TX_D"*) y a la recepción (*"RX_D"*) de datos.

La primera vez que se pulsa *"Go"* se da paso a la preparación de la ejecución y a que se sitúe el punto de trabajo al inicio de la función principal o *"main"*; mientras que la segunda vez que se selecciona se comienza con su ejecución. Marcado la opción *"Stop"* del menú *"Execute"* se detiene la ejecución del programa.

Cerramos el depurador y el proyecto en el compilador y, posteriormente, apagamos la placa.

2.1.6.7 Grabar un Programa en la Memoria *"Flash"* y Ejecutarlo.

En el caso anterior el programa se almacenaba en soporte *"SRAM"*, al que sólo se le tiene acceso como memoria de usuario. Esta es una gran diferencia frente al caso que ahora nos ocupa, puesto que el soporte *"Flash"* es accesible tanto por el usuario como por la rutina *"boot"*. Resulta crítico indicar el tipo de acceso para evitar grabar sobre la rutina de arranque el código de usuario; para ello conectamos *'JP_1'* a la patilla *"USER"* antes de descargar la imagen sobre *"Flash"* desde el depurador.

Conectamos *'JP_1'* a *"STD"* tras alimentar la placa para ejecutar el gestor de arranque definido en el *"firmware"*, inmediatamente después se conmuta a *"USER"*. Desde el propio depurador se selecciona la opción *"Load Image"* del menú *"File"* para cargar en el sistema objetivo un fichero con la secuencia binaria adecuada o imagen para la máquina de *ARM*. Este

fichero imagen se encuentra dentro de la ruta de directorio denominada *"c:\at91\software\tools\flash_downloader\flash_16x4\flash_16x4_Data\int_sram\"*, siendo su nombre de archivo *"flash_16x4.axf"*.

Al aceptar se muestra un cuadro de diálogo de aviso que nos pregunta si queremos modificar la imagen cargada. Este es el momento último de configurar el modo de funcionamiento de la placa como *"USER"*, si no lo hemos hecho previamente. Al pulsar el botón *"Aceptar"* comienza el proceso de descarga, el cual se observa porque los diodos luminiscentes del puerto *"Serial A"* están encendidos y porque, a su vez, nos lo indica el programa de depuración.

Seleccionando dos veces la opción *"Go"* del menú *"Execute"* se inicia la ejecución. Ahora, existe comunicación entre los equipos sólo cuando se requiere interacción con el usuario a través de la consola que proporciona el equipo *"host"* al *"target"*. A través de la consola se solicita la dirección de comienzo de la carga, indicando *"0x1000000"* en hexadecimal, y la ruta de directorio del fichero, introduciendo *"c: \at91 \software \projects \led_swing_eb40a \led_swing_data \flash \led_swing.bin"*, tal y como nos lo indica la ayuda del fabricante.

Lamentablemente se incurre en un error puesto que se nos muestra el mensaje *"Cannot open file image"*, es decir, que no puede abrir el archivo imagen. Como solución propuesta buscamos la ruta de directorio el archivo *"led_swing.bin"* mediante el sistema de búsqueda de *WindowsXP*, pero como destino sólo nos muestra la carpeta ya indicada.

En nuestro afán por resolver el error analizamos el código en *C++* que se nos facilita en una ventana del depurador. Las funciones que esta imagen usa para comunicarse mediante la consola se llaman *"printf"* y *"sscanf"*, ambas relacionadas con los mensajes de texto. Uno de los parámetros de *"sscanf"* se denominaba *"image"*; el cual se definía a partir de la llamada *"image=fopen(name,'rb')"*. Gracias al observador de los valores de las variables que proporciona el depurador para facilitar el seguimiento, se constata que la función *"fopen"* devuelve *NULL*.

Para conocer el prototipo de esta función, así como su cometido, nos documentamos en el sitio de Internet [B2] http://c.conclase.net/ficheros/ficheros002.html. En él se muestra el prototipo de *"fopen"*, siendo: *"FILE *fopen(char *nombre, char *modo)"*. Dentro de la cadena que define al *"modo"* , la 'r' implica que se abre el fichero como lectura, exigiendo que exista el archivo y, la 'b' quiere decir que se tratará como binario. Esto tampoco nos solventa el error porque sí que existe el archivo –*WindowsXP* así lo atestigua–, pero *"fopen"* devuelve *NULL*.

Salimos del depurador e iniciamos la placa en el modo usuario, para que ejecute el programa descargado a modo de prueba y, efectivamente, los indicadores luminiscentes se encienden dando lugar a un efecto de vaivén. Luego el código se había grabado en la memoria *"Flash"* de usuario satisfactoriamente.

2.2 Entorno de Desarrollo: CodeWarrior y AXD Debugger.

2.2.1 Introducción.

Este capítulo de la memoria se centra en la descripción del entorno de desarrollo [B5] mediante el que se ha elaborado el banco de pruebas, el cual obtiene medidas sobre la capacidad de procesamiento del microcontrolador *ARM7TDMI* de *ARM* trabajando en la placa de evaluación *AT91EB40A* de *ATMEL*.

La funcionalidad principal del paquete de software de *MetroWerks* es la de compilador cruzado para la máquina de *ARM*. Se encuentra formado por dos entidades básicas y fundamentales, las cuales se pueden ejecutar independientemente, aunque se obtiene una mayor capacidad cuando trabajan conjuntamente. Estas dos entidades son el compilador cruzado y el depurador. La primera construye, a partir de un código fuente descrito en lenguaje *C++*, una secuencia de código máquina para el microcontrolador específico. La segunda permite analizar el funcionamiento de dicho código sobre la máquina gracias a la ejecución paso a paso y la definición de observadores; estos facilitan el seguimiento de las variables que forman parte del código original en *C++* y de los registros del microcontrolador del código máquina.

Para cumplimentar nuestro objetivo se planteará el método de trabajo que se utilizó con el entorno de diseño integrado o *IDE –"Integrated Design Environment"–* de *MetroWerks*, tanto con el módulo de desarrollo de algoritmos como con el módulo de seguimiento y depurado.

Por último, y a modo de ejemplo, se mostrarán los pasos que se siguieron para el desarrollo y la depuración del algoritmo de generación de la secuencia de *Fibonacci*.

2.2.2 Método de Trabajo con MetroWerks CodeWarrior.

La configuración *hardware* del sistema a instalar en la aeronave, por fuertes restricciones de peso y de tamaño, nos fuerza a dirigir el proceso de diseño hacia sistema imbuidos. Esto obliga a trabajar con una estación *"host"*, en donde se ejecutará el entorno de desarrollo integrado, y con una placa *"target"*, en donde se computará el algoritmo que actúa directamente sobre el sistema físico a controlar.

Por lo tanto, debido a las características intrínsecas de este trabajo, nos vemos impulsados a elegir un tipo determinado de proyecto del conjunto de configuraciones de diseño que facilita el entorno *"CodeWarrior for ARM Developer Suite"*, es decir, se elegirá la opción *"ARM*

Executable Image" de la pestaña *"Project"* del cuadro de diálogo que surge al pinchar sobre la opción *"New"* del menú *"File"*.

Hemos de definir el nombre de nuestro nuevo proyecto y su ubicación en el árbol de directorio. El fichero que contiene la información sobre nuestro proyecto poseerá la extensión *"mcp"*.

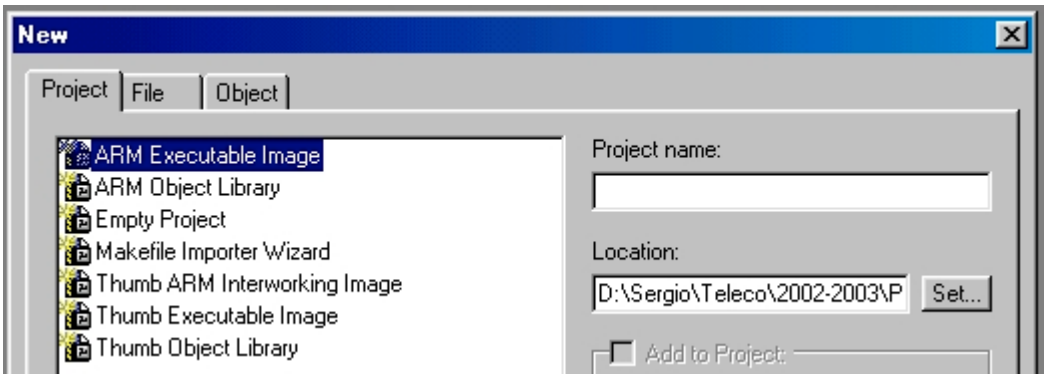


Figura: Cuadro de archivo nuevo.

Al solicitar la generación de un nuevo proyecto se realiza una serie de operaciones, como asociar el compilador *"Angel"* y el depurador *"DebugRel"* al proyecto; tras ellas se muestra, en un cuadro de diálogo, la lista de ficheros de código (archivos *.c*) y de librerías (archivos *.h*) asociada al proyecto nuevo, así como el orden de compilado dentro de la misma. Dentro de los archivos *.c* debe solamente existir uno de ellos en el que se defina la función principal del lenguaje *C++*, también llamada *"main"*.

Ahora necesitamos asociar al proyecto los archivos correspondientes, bien ya estén definidos o bien haya que construirlos. Para incluir ficheros preexistentes hay que indicar su ruta de directorio tras seleccionar la opción *"Add Files..."* del menú *"Project"*; para definir archivos nuevos tan sólo se pincha en la función *"New"* del Menú *"File"* para, posteriormente, seleccionar *"Text File"* de la lengüeta *"File"* del cuadro de diálogo que aparece, en el que se especifica su nombre y ruta. Para la redacción de ficheros nuevos se abre un editor de texto integrado en el paquete de desarrollo en una ventana contenida en la principal del entorno.

Dentro de la ventana asociada a nuestro proyecto se puede modificar el orden de compilado de los archivos que forman el conjunto de nuestro trabajo, así como compilar cada uno de ellos independientemente; esto facilita considerablemente la tarea de detección del origen de los errores. Nótese que para que podamos iniciar el proceso de depuración y seguimiento es de necesidad que no exista ningún error ni en el compilado ni en el enlazado.

El orden de compilado se indica por la posición con la que los archivos se colocan en la lista de ficheros. El primero a transcribir a código máquina se sitúa en la cima de la lista, mientras que el último queda en su base; el resto de ficheros se ordena siguiendo esta misma regla. Resulta crítico este orden en lo referente a la corrección de errores del enlazador –o *"linker"*, bajo terminología anglosajona–. Todas las constantes, funciones, macros y definiciones de tipos que en un archivo se usan deben estar descritas, y correctamente definidas, en alguno de los ficheros que lo preceden en el orden de compilado. Esto ocurre siempre que estemos tratando con definiciones globales o con funciones calificadas como externas.

Teniendo, junto a estas consideraciones previas, instaladas y reconstruidas con anterioridad las librerías para la placa *AT91EB40A* no tiene por qué surgir ningún error en el proceso de definición, compilado y enlazado de ficheros en una imagen ejecutable para *ARM*.

2.2.3 Método de Trabajo con AXD Debugger.

El depurador *"AXD Debugger for ARM Developer Suite v1.2"* puede trabajar con independencia del compilador de *ADS*. Pero cuando es llamado desde éste hay que considerar que únicamente puede existir una única sesión abierta del depurador, así se evitará que se produzcan conflictos en la comunicación entre procesos abortándose la depuración.

Puesto que se está trabajando con un sistema externo a la máquina donde reside el depurador, utilizar la configuración por defecto denominada *"ARMulator"* no resulta productivo. Este modo de funcionamiento, de acuerdo con la documentación suministrada por el fabricante, simula el comportamiento de los registros internos del microcontrolador real de *ARM* y su funcionamiento general sobre la arquitectura hardware del sistema *"host"*. Luego no se comunica con el sistema real.

Consiguientemente, lo primero que se debe lograr es que se trabaje con el sistema *"target"* de *ATMEL*, gracias a la configuración del módulo de comunicaciones que incorpora el depurador. Para ello se ha de seguir los siguientes pasos:

- Seleccionamos la opción *"Configure Target..."* del menú *"Options"* de *AXD*.
- Tras esto debemos elegir el objetivo a configurar del cuadro de diálogo que se ha abierto cuyo nombre es *"Choose Target"*: Seleccionamos "*APD"* y pulsamos el botón *"Configure"*.
- Con ello se muestra un nuevo cuadro denominado *"Remote_A Connection"*.

- Pinchando sobre *"Select..."* aparece una lista con los controladores de comunicación disponibles, seleccionamos la conexión serie, es decir, *"ARM Serial Driver"*.
- Pinchando sobre *"Configure..."* podemos indicar la velocidad, en bits por segundo o *bps*, y el nombre del puerto serie; en nuestro caso trabajaremos con el puerto *COM1* a 38400 bps.

- Para finalizar aceptamos en todos los cuadros de diálogo que se han ido abriendo sin más que pinchando en el botón *"Ok"* de cada uno de ellos.

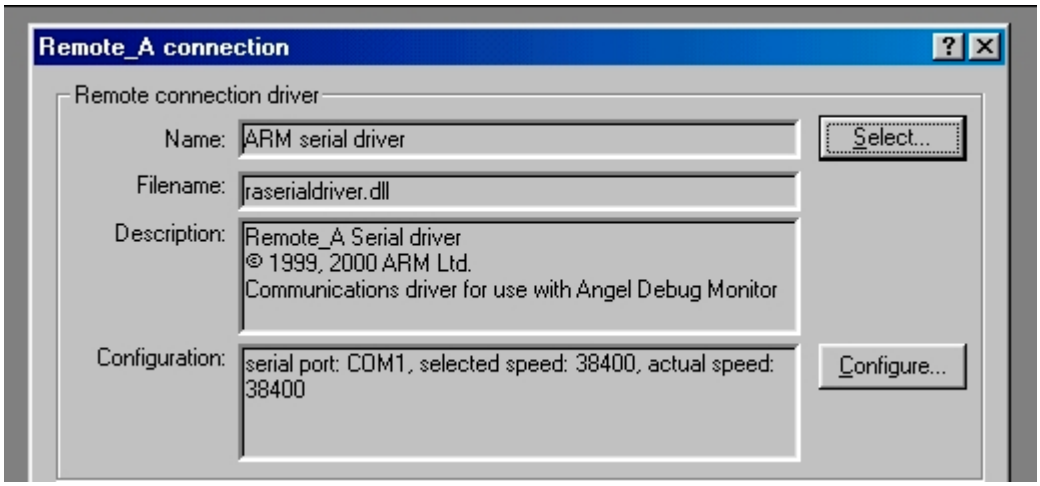


Figura: Vista final de la configuración.

Si la placa de *ATMEL* está conectada vía puerto serie al equipo *"host"*, se actualiza el contenido de la ventana de registro o *"RDI Log"* con el mensaje *"Angel Debug Monitor (serial) for at91eb40a"*, permitiéndose, a partir de este momento, la depuración paso a paso y el seguimiento de los valores de las variables.

En caso contrario, se produce un error fatal porque resulta imposible establecer la comunicación con el equipo *"target"*, lo que obliga a cerrar la aplicación del depurador o a volver a definir el puerto de comunicaciones, habiendo anteriormente conectado la placa de evaluación correctamente.

Para ejecutar paso a paso o a través de la definición de punto de ruptura o *"break-points"*, *AXD* proporciona herramientas comunes a la gran mayoría de los entornos de desarrollo integrados, como son la ejecución instrucción por instrucción, el salto al interior de la función llamada o la ejecución hasta un punto de ruptura. Resulta muy útil el hecho de que facilite estos métodos de trabajo tanto sobre el código descrito en *C++* como sobre el que está descrito mediante los mnemónicos de ensamblador; permitiéndose la observación simultánea de la

ejecución con dos ventanas, una para cada lenguaje. Esto, a su vez, proporciona una herramienta excelente para poder asociar, en tiempo real, secuencias de instrucciones en código *C++* con sus homólogas en ensamblador.

El ensamblador para *ARM* permite diferentes niveles de optimización a la hora de transcribir el código *C++* en su ensamblado equivalente. Gracias al uso de las dos ventanas podíamos saber si realmente se ejecutaban instrucciones de *C++* en la máquina *"target"* y, en el caso de que fuese necesario que se computasen, forzar a que la optimización fuese menos restrictiva en tiempo o en tamaño de la imagen. Por ejemplo, la optimización provoca que no se ensamblen las operaciones cuyo resultado no se asigne a ninguna variable.

2.2.4 Ejemplo de Desarrollo: Serie de Fibonacci.

La utilidad de este ejemplo posee una doble cara. Por un lado, tiene valor ilustrativo para los puntos anteriores y, por otro, sirvió para afianzar un método de trabajo con el *IDE* de *MetroWerks* para *ARM*, el cual es necesario para la elaboración del banco de pruebas y la coordinación con *"Real Time Workshop Embedded Coder"*.

La serie del matemático italiano *Fibonacci* se define de forma recursiva como sigue mediante una sucesión de elementos reales:

$$a_n = \begin{cases} 0; & n = 0; \\ 1; & n = 1; \\ a_{n-1} + a_{n-2}; & \forall n > 1; \ni n \in N \end{cases}$$

Luego, se logra la secuencia: $\{a_n\}_{n \in N} = \{0, 1, 2, 3, 5, 8, 13, 21, ...\}$.

El algoritmo que se ha escogido no es recursivo, siendo su código fuente, expresado bajo el lenguaje *C++*, el siguiente:

```
#include <stdio.h>
#define N 10                    //nº de interacciones máximo

int main()
{
  int fib0;             // a(n)
  int fib1=1;           // a(n-1)
  int fib2=0;           // a(n-2)
  int n=0;
  while(n<N)
  {                     //Serie Fibonacci: a(n)=a(n-1)+a(n-2);
        fib0=fib1+fib2;
        printf("\n Valor %i:%i",n,fib0);
        n++;
                      //Actualizamos valores
       fib2=fib1;// a(n-2) = a(n-1)
```

```
            fib1=fib0;          // a(n-1) = a(n)
        }
        return 0;
}
```

Puesto que esta serie numérica es monótona creciente, se ha limitado el número de elementos que se generarán mediante la constante *'N'*.

En lugar de usar un vector de variables, lo que hubiera constituido un código más legible, se ha decidido definir una variable para cada término de la ecuación, lo que desemboca a que los requisitos hardware sean menores; siendo, para un sistema imbuido, una limitación importante.

Tras la compilación con el entorno de desarrollo *MetroWerks CodeWarrior*, se genera un informe que muestra la cantidad de memoria requerida por el código ensamblador. Éste aparece en la ventana *"Errors & Warnings" –"Errores y Avisos"*, en español–.

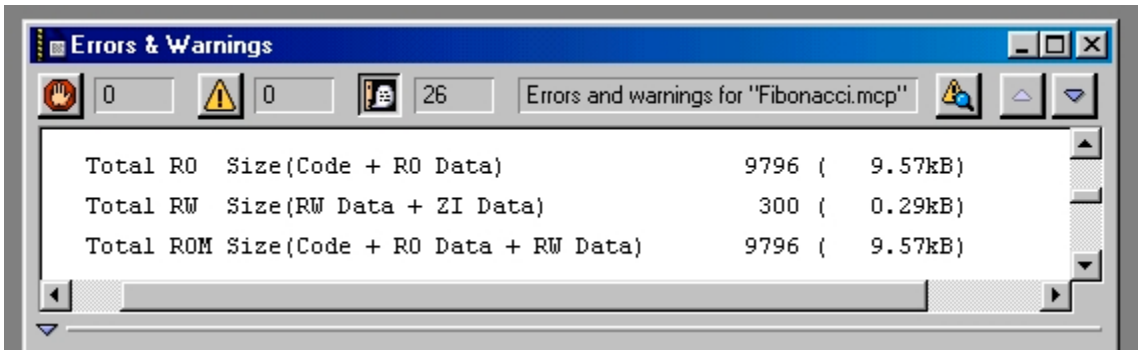


Figura: Ventana de Errores y Avisos.

Durante el depurado, con los observadores, se permite conocer la evolución de las variables del código fácilmente, comprobándose que es la esperada.

Resulta interesante, gracias al juego que nos ofrece la disposición de la ventana del código fuente *C++* y la de ensamblador, poder comparar ambos códigos. La ventana de ensamblador nos muestra lo siguiente:

```
main            [0xe92d4070] *   stmfd   r13!,{r4-r6,r14}
000080ac        [0xe3a05001]     mov     r5,#1
000080b0        [0xe3a00000]     mov     r0,#0
000080b4        [0xe3a04000]     mov     r4,#0
000080b8        [0xea000007]     b       0x80dc  ; (main + 0x34)
000080bc        [0xe0856000]     add     r6,r5,r0
000080c0        [0xe1a02006]     mov     r2,r6
000080c4        [0xe1a01004]     mov     r1,r4
000080c8        [0xe28f001c]     add     r0,pc,#0x1c ; #0x80ec
000080cc        [0xeb00000c]     bl      _printf
000080d0        [0xe2844001]     add     r4,r4,#1
000080d4        [0xe1a00005]     mov     r0,r5
000080d8        [0xe1a05006]     mov     r5,r6
000080dc        [0xe354000a]     cmp     r4,#0xa
000080e0        [0xbafffff5]     blt     0x80bc  ; (main + 0x14)
```

```
000080e4    [0xe3a00000]    mov     r0,#0
000080e8    [0xe8bd8070]    ldmfd   r13!,{r4-r6,pc}
```

Se ha seleccionado la parte del código ensamblador relativa a la ejecución de la función *"main"*, excluyéndose la inclusión de librerías con la definición de funciones y las instrucciones que permiten la salida de programa de forma coherente.

Cada una de las 4 columnas que representan las instrucciones tiene su significado propio. La primera de ellas indica el valor de la dirección, dentro del mapa de memoria del microprocesador *ARM7TDMI*, que ocupa cada instrucción expresada bajo notación hexadecimal. La segunda columna nos muestra el código de la instrucción en lenguaje máquina, es decir, en una secuencia binaria. Las otras dos columnas no son más que la traducción cada instrucción mediante el uso de mnemónicos; la tercera nos muestra su nombre y la cuarta, sus argumentos.

Los mnemónicos son propios de cada fabricante, aunque sí es cierto que existe cierta estructuración similar entre los diferentes esquemas propietarios, puesto que todos ellos buscan la legibilidad del código. Por ejemplo: *"mov r4,r5"* provoca que el valor almacenado en el registro número cuatro adquiera el que tenía el quinto.

De acuerdo con los manuales [B5] que proporciona *ARM Limited*, cada uno de los códigos expresa:

- *"stmfd"*: Se trata de la instrucción *"stm"* o *"store multiple"* con los bits opcionales *'F'* y *'D'* activados. Se utiliza cuando se va a producir una excepción, sea por una interrupción o por una llamada a una función, de forma que el estado actual de los registros del sistema se guarda en su pila.
- *"mov"*: El nombre completo de la instrucción es *"move"*, se ha diseñado para mover o trasladar el valor del parámetro segundo al primero.
- *"b"*: De nombre *"branch"* provoca un salto condicional a la dirección de memoria especificada siempre que los bits de estado indiquen un resultado cero. Estos bits consideran si la última operación ha dado como resultado cero, positiva, negativa, división por cero, etc.
- *"add"*: La suma entre los contenidos entre el segundo y el tercer operando se vuelca al primero.
- "bl": Provoca un salto condicional en el flujo del programa a un enlace especificado por una etiqueta, siendo el nombre completo de la instrucción *"branch and link"*.

- "cmp": Compara –"compare" según la notación original– el valor del registro indicado con el valor constante del segundo operando, esto provoca que los bits de estado se actualicen.
- *"blt"*: Es análoga a la instrucción *"bl"*, pero con el bit opcional *'T'* activado.
- *"ldmfd"*: Se trata de la instrucción *"ldm"* o *"load multiple"* con los bits opcionales *'F'* y *'D'* activados. Es la instrucción complementaria a "store multiple", puesto que se utiliza cuando se regresa de una rutina de excepción o de la ejecución de una función.

Gracias a la proximidad del lenguaje *C++* al código ensamblador, la asociación entre ambos juegos de instrucciones resulta, relativamente, sencilla. Veamos algunos ejemplos:

- Inicialización:

```
int fib1=1;          // a(n-1)
int fib2=0;          // a(n-2)
int n=0;
```

```
000080ac[0xe3a05001]    mov     r5,#1
000080b0 [0xe3a00000]   mov     r0,#0
000080b4 [0xe3a04000]   mov     r4,#0
```

Se observa que el registro cinco queda asociado a la variable *"fib1"*. Análogamente, el registro cero, a la variable *"fib2"*, y el cuarto, a la variable *"n"*. El registro *"r5"* se adquiere un valor inicial de uno; mientras que el *"r0"* y el *"r4"*, un valor cero.

- Bucle *"mientras"*:

```
while(n<N){                  //Serie Fibonacci: a(n)=a(n-1)+a(n-2);
   fib0=fib1+fib2;
   printf("\n Valor %i:%i",n,fib0);
```

```
000080b8            [0xea000007]    b       0x80dc  ; (main + 0x34)
000080bc            [0xe0856000]    add     r6,r5,r0
000080c0            [0xe1a02006]    mov     r2,r6
000080c4            [0xe1a01004]    mov     r1,r4
000080c8            [0xe28f001c]    add     r0,pc,#0x1c ; #0x80ec
000080cc[0xeb00000c]    bl      _printf
```

La primera instrucción provoca el salto al final de la función *"main"* si los bits de estado provienen de un resultado nulo. La siguiente da lugar a la actualización de la variable *"fib0"*, la cual se construye con el registro *"r6"*. Las siguientes instrucciones preparan y ejecutan una llamada a la función de librería *"_printf"*, la cual muestra por la salida estándar las variables *"n"* y *"fib0"*.

```
   n++;
        //Actualizamos valores
   fib2=fib1;        // a(n-2) = a(n-1)
   fib1=fib0;        // a(n-1) = a(n)
}
```

```
000080d0    [0xe2844001]    add     r4,r4,#1
000080d4    [0xe1a00005]    mov     r0,r5
000080d8    [0xe1a05006]    mov     r5,r6
000080dc    [0xe354000a]    cmp     r4,#0xa
000080e0    [0xbafffff5]    blt     0x80bc  ; (main + 0x14)
```

La primera instrucción facilita la cuenta de cada una de las iteraciones del bucle *"mientras"*. Las dos siguientes construyen la actualización de variables, la cual es vital para este algoritmo. La instrucción *"cmp"* compara el contenido de *'r4'* –variable *'n'* – con el número hexadecimal *'a'*, es decir, con 10; esta comparación implica una operación de resta que sólo altera los bits de estado, dejando de lado el resultado, para que pueda trabajar coherentemente la instrucción de salto condicional al inicio del bucle *"mientras"*, el cual queda representado en *C++* con el carácter *'}'*.

- Regreso de la función *"main"*:

```
    return 0;
}
```

```
000080e4    [0xe3a00000]    mov     r0,#0
000080e8    [0xe8bd8070]    ldmfd   r13!,{r4-r6,pc}
```

La primera instrucción tiene como cometido forzar los bits de estado al caso en el que un cero ha sido la última operación; nótese que se coloca un 0 en un registro para devolverlo mediante la instrucción *"return"*. De esta forma el vuelco de la pila a los registros adecuados se realiza correctamente.

2.3 Banco de Pruebas.

2.3.1 Objetivo.

Se pretende desarrollar un banco de pruebas para evaluar la potencia del microcontrolador *ARM7TDMI* de *ARM.* Con los resultados de esta evaluación se busca orientar el desarrollo de futuros algoritmos para este microprocesador que presenten restricciones en tiempo.

2.3.2 Material de Laboratorio.

Para la realización de este banco se ha hecho uso del siguiente material:

- Estación *PC* bajo *WindowsXP*.
- Entorno de desarrollo *Advanced Developer Suite v1.2*.
- Compilador cruzado *Metrowerks CodeWarrior* y el depurador *AXD* de *Metrowerks.*
- Entorno de desarrollo *Borland C++ Builder*.
- Placa de Pruebas *ARM7eb40a* de *ATMEL* que incluye el microprocesador *ARM7TDMI.*
- Conexionado y alimentación.

2.3.3 Pruebas: Elección y Planteamiento.

Las medidas de las pruebas que forman el banco de pruebas poseen naturaleza temporal. Para obtenerlas se itera repetidas veces una misma operación unitaria, sea una suma o una multiplicación. Aumentando el número de iteraciones se logra una mayor precisión en la medida, puesto que el efecto de las variables aleatorias tiende a su valor medio, es decir, a hacerse cero.

Con la intención de obtener las medidas para las instrucciones tipo se ha realizar el experimento dos veces: La primera con el código completo, que comprende la instrucción base y el código asociado al control de flujo, y la segunda, que excluye la instrucción base. De su diferencia se obtiene la medida temporal que se busca: La relativa a la instrucción base.

Con el fin de acercar más la medida al comportamiento interno del microprocesador se estudia la traducción en su lenguaje ensamblador [B8] del código fuente expresado en *C++*.

Asimismo, se plantea un conjunto de algoritmos que se componen de múltiples operaciones unitarias, a las que hay que añadir el control de flujo. Debido a su interés respecto a

su aplicación al control, se ha decidido evaluar el consumo de recursos temporales para estos algoritmos en el microprocesador de trabajo.

2.3.3.1 Planteamiento y Algoritmos.

Los algoritmos que se usaron sufrieron un proceso de adecuación y depuración hasta lograr la funcionalidad deseada. Aquí se expondrán los códigos y métodos definitivos siendo en el pliego correspondiente donde se aúnan todas sus fases, con sus errores y soluciones.

2.3.3.1.1 Algoritmo para las instrucciones base.

El método queda recogido en el archivo *"bancoDePruebas3_7.cpp"* para aritmética flotante y en el archivo *"bancoDePruebasInt.cpp"* para aritmética entera. Los códigos llevan a cabo la medida tanto para la instrucción de suma como para la de multiplicación.

- Funciones en *C++* y funcionalidad:
  - *"main"* o programa principal: Define la interfaz en modo consola con el usuario y realiza las llamadas adecuadas al resto de funciones para obtener la medida. El usuario indica el número de veces que se repetirá la instrucción base.
  - *"VA"*: Genera una muestra de una variable aleatoria uniforme comprendida entre 0 y 1. Se usa como valor sobre el cual se efectuará la instrucción base.
  - *"tiempoSuma"*: Devuelve el tiempo que transcurre para que se aplique la instrucción base de suma un determinado número de veces.
  - *"tiempoProducto"*: Idéntica a la función anterior, salvo que se aplica la instrucción producto.

2.3.3.1.2 Algoritmos de interés en control.

Los algoritmos que se han planteado forman las herramientas básicas que son usadas por numerosas estrategias de control automático. Estos son:

- Las ecuaciones en diferencias; las cuales se utilizan para la implementación de filtros, lo que ayuda a estimar los parámetros de los sistemas mediante estrategias de aprendizaje para construir leyes de autosintonía.
- Los cálculos estadísticos se usan para analizar el comportamiento histórico de los parámetros del sistema a controlar. Esto facilita que se elaboren métodos de control de alto nivel, en los que se definen, por ejemplo, qué estrategias de control seguir, según se estime el modelo del sistema.

- La inversión matricial constituye un pilar básico en el cálculo de las señales de control cuando la descripción del comportamiento del sistema se lleva a cabo mediante el espacio de estados.
- El producto matricial posee un peso análogo al caso de la inversión.

2.3.4 Instrucciones Base.

2.3.4.1 Aritmética Flotante.

Para este caso se obtiene:

Número de Iteraciones	Tiempo de Suma (s)	Tiempo de Producto (s)
10×10^6	7 – 1 = 6	8 – 1 = 7
20×10^6	15 – 2 = 13	15 – 2 = 13
50×10^6	38 – 4 = 34	38 – 4 = 34
100×10^6	76 – 9 = 67	77 – 9 = 68
200×10^6	151 – 18 = 133	155 – 19 = 136

En cada campo de esta tabla se muestra como resultado el tiempo relativo a la operación base, siendo la diferencia entre el tiempo completo y el que está asociado únicamente al control de flujo.

Se observa que:

- Para la suma se usa el 88.08 % del tiempo en los cálculos; realizándose 1,5035 MFLOPS.
- Para el producto, el 87.74 %; realizándose 1,4705 MFLOPS.

Puesto que se obtiene una gran similitud entre ambas operaciones, se desprende que en la arquitectura del microprocesador existe una unidad aritmético-lógica propia para las multiplicaciones. Este hecho es coherente con la existencia de una instrucción en ensamblador dedicada exclusivamente para el producto de números flotantes cuya etiqueta es *"fmul"*. **Esto será especialmente útil en el desarrollo de algoritmos que implementan ecuaciones en diferencias para los métodos de control.**

2.3.4.2 Aritmética Entera.

Obteniéndose para este caso la tabla de resultados:

Número de Operaciones	Tiempo de Suma (s)	Tiempo de Producto (s)
10 x 10^6	2 – 1 = 1	2 – 1 = 1
20 x 10^6	4 – 2 = 2	4 – 2 = 2
50 x 10^6	10 – 4 = 6	10 – 5 = 5
100 x 10^6	20 – 9 = 11	21 – 9 = 12
200 x 10^6	40 – 18 = 22	42 – 19 = 23

Cada campo presenta el mismo formato que la tabla asociada a los resultados con flotantes.

Se observa que:

- Para la suma se usa el 55% del tiempo para los cálculos; realizándose 9,090 MINTS.
- Para el producto, el 54.7%; realizándose 8,704 MINTS.

De igual forma que en el caso de operar con flotantes, tampoco existe una gran diferencia entre el tiempo dedicado a los productos y el de las sumas.

2.3.5 Control.

La aplicación de estrategias de control exige que, previamente, se conozcan las exigencias de tiempo de herramientas básicas. Así pues, se han realizado algoritmos que midan los recursos que éstas consumen.

Como herramientas a estudio cabe señalar las ecuaciones en diferencias, el cálculo de estadísticos, la inversión matricial y los productos de matrices. Las ecuaciones se usan para implementar leyes de control básicas como un *PID* con una tabla de ganancia (*"gain–scheduling"*); los estadísticos poseen utilidad en cuanto se analiza la evolución de las variables de un sistema a partir de registros y, la inversión y el producto matricial son básicas para las herramientas avanzadas de control discreto en el dominio del espacio de estados, como son los filtros de *"Kalman"* o predictores generalizados y los esquemas basados en el método *LQR – "Linear Quadratic Regulator" –*.

2.3.5.1 Ecuaciones en Diferencias: Filtros.

Una ecuación en diferencias expresa la relación dinámica que existe entre una secuencia de entrada a la ecuación y la secuencia de salida de la misma. Para un caso genérico, puede expresarse dicha relación formalmente mediante la ecuación:

$$y(k)=a_1y(k-1)+a_2y(k-2)+\cdots+a_my(k-m)+b_1u(k-1)+b_2u(k-2)+\cdots+b_nu(k-n); \quad [1]$$

En donde:

- *'y(k)'*: La secuencia de salida de la ecuación.
- *'u(k)'*: La secuencia de entrada de la ecuación.
- *'{a_1, a_2, ..., a_m, b_1, b_2, ...,b_n}'*: Conjunto de coeficientes que definen la ecuación.
- $n < m$: Para forzar la causalidad de la ecuación.
- *'k'*: Índice de iteración de la ecuación.

Tras ejecutar el código que implementa esta ecuación en la placa de *ATMEL* obtenemos los siguientes resultados:

Tiempo (s)	**Número de Operaciones**		
Orden (n, m)	**100**	**1000**	**10000**
(1,1)	1	14	210
(2,2)	2	26	-
(3,3)	2	25	-
(4,4)	2	20	-
(5,5)	2	23	-
(10,10)	4	25	-
(20,20)	2	29	-
(50,50)	3	31	227

Se observa, para 10000 operaciones, que:

- Usando un orden de (1, 1) se invierten 21 ms por iteración.
- En el caso de orden (50, 50) se necesitan 22.7 ms por iteración.

Estos resultados recogen tanto el código de control de flujo como el que propiamente realiza los cálculos de la ecuación en diferencias. La medida se hace conjuntamente puesto que para poder construir la ecuación se requiere de un control de flujo inexorablemente y, éste,

consume tiempo y recursos. De aquí que no se presente una dependencia marcada entre el orden del filtro y el tiempo medido.

2.3.5.2 Cálculos Estadísticos.

Para elaborar un banco de medidas de tiempo de estos parámetros se ha considerado, como elementos de interés, el orden del estadístico y el número de muestras del espacio muestral con el que se trabaja.

La importancia del análisis y del estudio del comportamiento y evolución de las variables asociadas a una planta real obliga a la medida de tiempos de parámetros estadísticos, puesto que estos forman una herramienta muy útil en la predicción de esos comportamientos.

Considerando un conjunto de muestras de una variable aleatoria uniforme, a priori definida, se han obtenido los valores de los estadísticos siguientes:

$$x \triangleq U(x_1, x_2); \quad x_1 = \hat{x} - \Delta/2; \quad x_2 = \hat{x} + \Delta/2;$$

$$x_k \in x; \quad \Rightarrow \quad \begin{cases} \langle x \rangle = \dfrac{1}{N}\sum_{k=1}^{k=N} x_k; \quad \langle x \rangle_k = \langle x \rangle_{k-1} + x_k / N; \quad \lim_{k \to N} \langle x \rangle_k = \langle x \rangle; \\ \xi_m = \dfrac{1}{N}\sum_{k=1}^{k=N} \left(x_k - \langle x \rangle\right)^m; \quad \langle \xi_m \rangle_k = \langle \xi_m \rangle_{k-1} + \left(x_k - \langle x \rangle_k\right)^m / N; \quad \lim_{k \to N} \langle \xi_m \rangle_k = \xi_m; \end{cases} \qquad [2]$$

En donde:

- *'x'*: Es la variable aleatoria uniforme cuya amplitud es *'D'* y su media es *'x'* circunfleja.
- *'x_k'*: Es cada una de las muestras de la variable aleatoria *'x'*. Con ellas se definen sus estadísticos.
- *'<x>'*: Es el valor medio del conjunto formado por todas las *'x_k'* consideradas.
- *'<x>$_k$'*: Es el valor acumulado de *'<x>'* para la muestra k–ésima *'x_k'*.
- *'ξ_m'*: Es el valor del estadístico de orden m para las muestras *'x_k'* respecto al valor promedio.
- *'<ξ_m>$_k$'*: Es el valor acumulado de *'ξ_m'* para la muestra k–ésima *'x_k'*.
- *'N'*: El número de muestras extraídas de la variable aleatoria *'x'*.

Tanto el valor medio como el estadístico de orden superior se calculan conforme se van obteniendo muestras, por lo tanto, se obtienen como el límite de las variables acumuladas respectivas. Estas operaciones se han efectuado tanto para muestras expresadas en flotante como

para muestras de naturaleza entera. Para este último caso, existe un error de cuantificación apreciable en los resultados obtenidos, pero carece de importancia puesto que nuestro objetivo consiste en la medida de los tiempos de cómputo.

Tratándose muestras flotantes, éstas se encuentran comprendidas entre 0.99 y 1.01, se logra la siguiente tabla de resultados:

N	t (s)	$\langle x \rangle_N$	$\langle x_2 \rangle_N$	t (s)	$\langle x \rangle_N$	$\langle x_3 \rangle_N$	t (s)	$\langle x \rangle_N$	$\langle x_5 \rangle_N$	t (s)	$\langle x \rangle_N$	$\langle x_{10} \rangle_N$
100	4	1.001	0.329	3	1.000	0.245	2	0.999	0.162	2	1.000	0.086
1000	21	1.000	0.333	16	1.000	0.250	12	1.000	0.166	20	1.000	0.090
2000	37	1.000	0.333	40	1.000	0.250	42	1.000	0.166	41	1.000	0.091
5000	92	1.000	0.333	102	1.000	0.250	101	1.000	0.167	86	1.000	0.091
10000	204	1.000	0.333	199	1.000	0.250	207	1.000	0.167	214	1.000	0.091

En donde se observa que:

- *'*$\langle x \rangle_N$*'* tiende al valor definido de la media de la variable aleatoria; esto se puede interpretar a partir de que ligera diferencia, que existe cuando se trabaja con 100 muestras, desaparece completamente al tratar con más.
- De forma análoga ocurre para *'*$\langle x_2 \rangle_N$*'*.

La primera columna de cada bloque representa el tiempo invertido en segundos. Considerando 10000 muestras resulta que:

- Para orden 2 se invierten 20.4 ms por muestra para calcular el *'*$\langle x_2 \rangle_k$*'*.
- Para orden 3 se precisan 19.9 ms por muestra para obtener *'*$\langle x_3 \rangle_k$*'*.
- Para orden 5 se requieren 20.7 ms por muestra para deducir *'*$\langle x_5 \rangle_k$*'*.
- Y para orden 10 se miden 21.4 ms por muestra para hallar *'*$\langle x_{10} \rangle_k$*'*.

No se presenta una dependencia entre el orden y el tiempo invertido, al menos, para órdenes menores de 10. Es posible que para órdenes mucho mayores sí exista esa relación, ya que el mayor tiempo medido se debe al caso de mayor orden.

2.3.5.3 Inversión de Matrices.

Esta operación con matrices está muy extendida en la realización práctica de algoritmos de control que trabajan en el dominio discreto desde la perspectiva del espacio de estados.

Los cálculos de tiempo toman como parámetro de referencia la dimensión de la matriz a invertir; puesto que ésta ha de ser cuadrada, por definición, se considera su dimensión como el número de filas o de columnas.

Las entradas de las matrices se expresan sólo como números flotantes, ya que tratar con números enteros no tiene aplicación práctica en las estrategias de control. Nótese que las matrices que en éstas suelen aparecer representan coeficientes o parámetros de similitud que no son números enteros por estar normalizados, entre otras cuestiones.

A la hora de aplicar un algoritmo de inversión de matrices sobre un sistema imbuido como el que nos ocupa hay que tener muy en consideración la cantidad de memoria que se necesitará. Esto es crítico porque estos sistemas no poseen memoria de usuario, en la mayoría de los casos actuales, más allá del orden de los *kilobytes*. Así pues, para una dimensión *'n'*, se tiene:

- Las propias entradas de la matriz: n^2 entradas.
- Las entradas de una matriz adjunta de orden *n-1*: $(n-1)^2$ entradas.
- Una llamada a la función que calcula el determinante para un orden *n-2* genera una matriz adjunta del mismo: $(n-2)^2$ entradas.
- Y así sucesivamente hasta que se da una llamada con orden *2*: 2^2 entradas.

Luego, aunando todas las entradas e imponiendo la condición de que no se supere la memoria de usuario, se tiene que:

$$n_0 = min\left\{ \forall n \in N; \quad \middle| \quad M = \left\lceil \sum_{k=1}^{n} \frac{k^2}{4} \right\rceil \right\};$$

En donde se ha definido:

- *'n_0'*: Es la dimensión máxima de la matriz que admite la memoria de la placa *ARM7eb40a*.
- *'M'*: Es la cantidad de bytes que forma la memoria de usuario, que para esta placa es de 256 Kb.
- 4: Es el número de bytes que se necesita para codificar una entrada mediante aritmética flotante.

En esta expresión se ha usado la función suelo, la cual devuelve el entero contiguo menor al valor dado como parámetro. Por lo tanto, para nuestro caso concreto, resulta que $n_0 = 59$. Esto quiere decir que el límite que será decisivo en la aplicación de una inversión matricial será el tiempo límite de iteración de los bucles de control impuesto por la dinámica del sistema a controlar y no, el tamaño de la memoria de usuario porque no se usará, para casos prácticos, modelos de dimensión 59 para describir el comportamiento del sistema.

De acuerdo con las pruebas realizadas sobre el sistema de desarrollo de *ATMEL* la dimensión máxima que se admite es *7x7*. Esto es de este modo porque para matrices mayores el algoritmo se bloquea sin mostrar ningún mensaje por la salida estándar en un tiempo prudencial. Esta situación no se alcanza cuando se prueba el mismo código sobre el equipo de sobremesa, aunque la limitación se da en el orden la matriz por motivos de tiempo de cálculo.

Veamos una tabla con los resultados de tiempo para diferentes órdenes:

Orden (nxn)	Tiempo (ms)	Notas
4x4	0±0	10 pruebas con tiempos 0.000 s y entradas acotadas por ±10
5x5	3±1	10 pruebas con tiempos: 2, 3 y 4 ms
6x6	388±170	10 pruebas con media 387.9 ms y desviación típica 170.47 ms
7x7	1234±97	10 pruebas con media 1234.2 ms y desviación típica 97.17 ms
8x8	7371±98	3 pruebas con media 7371 ms y desviación típica 97.5 ms

El equipo utilizado para obtener estos resultados presenta la configuración *PentiumII233* bajo *Windows98 2nd Edition*.

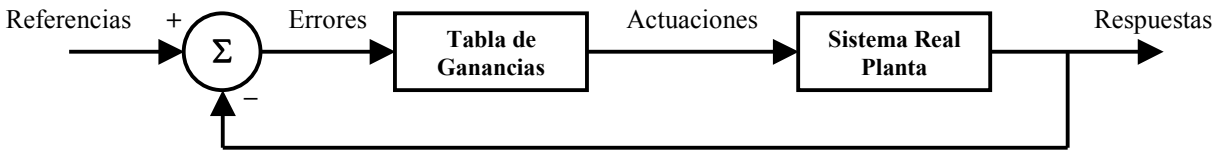
Debido a los tiempos medidos, aplicar algoritmos de control que necesiten la inversión matricial no es viable considerando tiempos de control del orden del milisegundo.

2.3.5.4 Producto de Matrices.

Al igual que la inversión matricial, el producto de matrices se encuentra muy extendido en los algoritmos de control basados en una descripción del sistema mediante el espacio de estados. De acuerdo con las medidas de tiempo y de potencia de cálculo, se limita el número de leyes de control potencialmente aplicables.

Estudiando leyes de control aplicadas sobre sistemas reales en aviónica se observa una predominancia importante del algoritmo de nombre *"gain-scheduling"* –bajo terminología anglosajona– o de *"tabla de ganancia"*.

Describamos, mediante el lenguaje formal de diagrama de bloques, el funcionamiento y el marco de aplicación de esta técnica:



Para poder aplicar estrategias de control suponemos que el sistema real se encuentra descrito mediante un modelo linealizado, es decir, que procede de un modelo más complejo cuyas expresiones han sido plasmadas en función de un punto de trabajo y de una desviación, respecto de éste, lo suficientemente pequeña. La no-linealidad del comportamiento dinámico queda recogida, por tanto, en la dependencia del punto de trabajo y de los coeficientes de la desviación frente a las condiciones externas e internas del sistema. Esta relación, por construcción, no posee ninguna característica dinámica. Ilustremos estos conceptos mediante un ejemplo sencillo.

- *Ejemplo: Linealización de las ecuaciones del sistema: Llenado de un depósito.*

El nivel de llenado del depósito se constatará en la altura del líquido que contenga y se regulará con una válvula de salida; la excitación que se aporta al sistema queda representada por su caudal de entrada. Por simplicidad, se supone que la sección transversal del depósito, denotada por *'A'*, es independiente de la altura del líquido.

Se usará la siguiente notación: Cada variable se representará por una letra mayúscula, para los valores en el punto de trabajo (o régimen permanente) se añadirá a la variable respectiva el subíndice '$_o$' y para la desviación asociada a cada variable se usará la minúscula correspondiente.

Se definen las variables:

- *'Q_i (m^3/s)'*: Caudal de entrada, cuyo valor en el punto de trabajo se denota como: *'Q_{io}'*.
- *'Q (m^3/s)'*: Caudal de salida, cuyo valor en el punto de trabajo se denota como: *'Q_o'*.
- *'H (m)'*: Altura de líquido del depósito, cuyo valor en el punto de trabajo se denota como: *'H_o'*.
- *'$V(m^3)$'*: Volumen de líquido del depósito.

Por dinámica de fluidos se conoce que el caudal de salida es proporcional a la raíz cuadrada de la altura de llenado del depósito. Este sistema presenta una clara dependencia no lineal, hemos de linealizarlo para poder aplicar la descripción basada en la función de transferencia puesto que, desde un punto de vista formal, se exige que el sistema a modelar sea lineal en las variables. Esto es:

- Planteemos la ecuación de balance de masa para hallar la primera ecuación del sistema:

$$\partial V / \partial t = Q_i - Q; \quad [4]$$

- Considerando estas expresiones se alcanza la segunda ecuación del sistema:

$$\left.\begin{array}{l} V = H \times A; \\ Q \propto \sqrt{H}; \end{array}\right\} \Rightarrow A \times \frac{\partial H}{\partial t} = Q_i - k\sqrt{H}; \quad [5]$$

- Mediante la linealización de variables se tiene que:

$$\begin{array}{l} Q = Q_o + q = Q_o + \left.\dfrac{\partial Q}{\partial H}\right|_{Q=Q_o} \times (Q - Q_o); \\ H = H_o + h = H_o + \left.\dfrac{\partial H}{\partial t}\right|_{H=H_o} \times (H - H_o); \end{array} \quad [6]$$

- Aplicando estas expresiones a las ecuaciones del sistema se obtiene que una relación lineal entre las variables *'h'* y *'q_i'*:

$$\left.\begin{array}{l} Q = k\sqrt{H} = k\sqrt{H_o} + \dfrac{k}{2\sqrt{H_o}}(H - H_o) \equiv Q_o + q; \Rightarrow q = \dfrac{k}{2\sqrt{H_o}} \times h; \\ A\dfrac{\partial H}{\partial t} = Q_i - Q; \equiv A\dfrac{\partial}{\partial t}(H_o + h) = Q_{io} + q_i - Q_o - q; \\ \qquad A\left.\dfrac{\partial H}{\partial t}\right|_{H=H_o} = Q_{io} - Q = 0; \end{array}\right\} \Rightarrow A\frac{\partial h}{\partial t} = q_i - q \Bigg\} \Rightarrow A\frac{\partial h}{\partial t} + \frac{k}{2\sqrt{H_o}} h = q_i; \quad [7]$$

Nótese que la dinámica lineal queda expresada entre las variaciones de las variables originales a través de unos coeficientes, que son constantes dado el punto de trabajo *'H_o'*.

- Aplicándole la transformada de Laplace queda la función de transferencia de la siguiente forma:

$$G(s) = \frac{H(s)}{Q_i(s)} = \frac{1}{A \cdot s + \frac{k}{2\sqrt{H_o}}} \Rightarrow G(s) = \frac{K_o}{1 + \tau s};\ K_0 = \frac{2\sqrt{H_o}}{k};\ \tau = AK_o;$$

Siendo:

- *'K_o'*: La ganancia estática.
- *'τ'*: La constante de tiempo.

Para el sistema dinámico de estudio de este proyecto las ecuaciones presentan un formato más complejo, pero se le puede aplicar el mismo método de trabajo para obtener su equivalente linealizado. Así pues, a modo de ejemplo, se podría decir que las condiciones externas o puntos de trabajo que afectasen a los coeficientes serían la altura, la orientación de la aeronave, su tendencia, su masa, la velocidad del viento, etc.

Con la intención de evaluar el tiempo invertido en una iteración del cálculo de la señal de control mediante un algoritmo de *"gain-scheduling"* se ha desarrollado el código *C++* residente en el archivo *"ControlProp2.cpp"*. De su ejecución en el sistema de desarrollo de *ATMEL* se desprenden los siguientes resultados, que resumidos a modo de tabla, quedan:

Nº repeticiones	**Tiempo (cs)**	**Tiempo (μs) por iteración**
10^5	300	30
$5 \cdot 10^5$	1600	32
10^6	3300	33

De lo que se desprende que:

- Debido a la limitación en la precisión impuesta por la macro *"CLK_TCK"* de 100 *ticks* por segundo, a mayor número de repeticiones se afina la medida del tiempo por iteración.
- El tiempo invertido para una iteración es, aproximadamente, de 33 μs; es decir, 2 órdenes de magnitud menor que el máximo exigido por diseño para el controlador, que era de 1 ms.

2.3.6 Conclusiones.

En este apartado se pretende organizar, a modo de resumen, las conclusiones que se desprenden de cada uno de los resultados obtenidos anteriormente; para ellos recurriremos a las medidas.

- Aritmética Flotante.
  - Para la suma se usa el 88.08 % del tiempo en los cálculos; realizándose 1,5035 MFLOPS (*"million of floating point operations per second"*).
  - Para el producto, el 87.74 %; realizándose 1,4705 MFLOPS.

Gracias a la similitud de los resultados se deduce que en la arquitectura *hardware* del microprocesador de *ARM* aparece una unidad aritmético – lógica especializada en productos. Puesto que estamos hablando de, aproximadamente, 1,5 MFLOPS se puede afirmar que este sistema ejecuta una instrucción bajo aritmética flotante en menos de 1 μs. Estos resultados son coherentes con los proporcionados por los manuales del fabricante del *chip*.

- Aritmética Entera.
  - Para la suma se usa el 88.08 % del tiempo en los cálculos; realizándose 1,5035 MINTS (*"million of instructions per second"*).
  - Para el producto, el 87.74 %; realizándose 1,4705 MINTS.

  Los comentarios al efecto son análogos a los que se han escrito para el caso flotante.

- Ecuaciones en diferencias: Filtros.
  - Usando un orden de (1, 1) se invierten 21 ms por iteración.
  - En el caso de orden (50, 50) se necesitan 22.7 ms por iteración.

De acuerdo con los resultados, el orden no posee una influencia de primera magnitud sobre el tiempo invertido en el cálculo de la ecuación en diferencias del filtro. De todas formas 21 ms excede, con creces, el límite impuesto por diseño de 1 ms para el bucle de control de bajo nivel. Por otra parte, sí podría ser usado este algoritmo para bucles de mayor nivel, puesto que requieren constantes de tiempo más largas.

- Cálculos Estadísticos.
  - Se invierten 20.4 ms por muestra para calcular el estadístico de segundo orden.
  - Se precisan 19.9 ms por muestra para obtener el estadístico de tercer orden.
  - Se requieren 20.7 ms por muestra para deducir el estadístico de quinto orden.
  - Y, se miden 21.4 ms por muestra para hallar el estadístico de décimo orden.

Por la similitud de estos resultados con los del caso anterior, se desprenden conclusiones similares. Un ejemplo interesante de bucle de nivel medio sería el que sintoniza o ajusta automáticamente el modelo del sistema que se pretende controlar.

- Inversión de Matrices.
  - Para orden sexto el tiempo requerido queda acotado por 0.565 s.

Con los tiempos medidos resulta prohibitivo implementar los algoritmos de inversión matricial considerando tiempos de 1 ms para el bucle de control de bajo nivel.

- Producto de Matrices.
  - El tiempo invertido para una iteración es, aproximadamente, de 33 μs; es decir, 2 órdenes de magnitud menor que el máximo exigido por diseño para el controlador, que era de 1 ms.

Estos resultados son gratamente interesantes porque abren la posibilidad de utilizar algoritmos de control basados en la técnica denominada *"gain-scheduling"* para el bucle cuyo nivel de trabajo lo sitúa más cercano al sistema real.

2.3.7 Cuestiones Prácticas.

El depurador *AXD* permite la visualización simultánea, en tiempo de ejecución, de la evolución del algoritmo tanto en lenguaje *C++* como en el ensamblador propio de *ARM7TDMI*. Para forzar que existiera una instrucción en ensamblador relativa a la instrucción base, es decir, que realmente el microprocesador la ejecutara, se hizo necesario definir las opciones del compilador optándose por el modo en el cual no se aplicaba ningún tipo de optimización. Por defecto el compilador no convertía las instrucciones de *C++* cuyas variables no se usaban en otra parte del código o que no tenían asignación.

Para modificar las opciones del compilador se requiere que exista un proyecto abierto; entonces de puede seleccionar la opción *"angel Settings..."* del menú *"Edit"* o pulsando las teclas *Alt-F7*, abriéndose el siguiente cuadro:

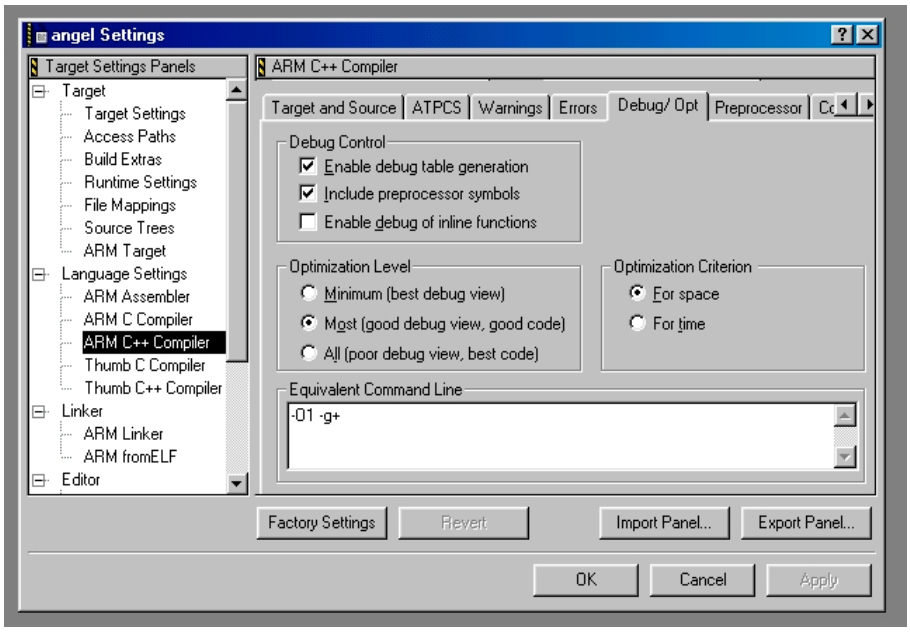


Figura: Opciones por defecto de Angel.

Por otro lado, el valor de la macro *"CLK_TCK"* define el nivel de resolución de las medidas de tiempo que permite la máquina. Esto implica que con la configuración de *ATMEL* resulta imposible hacer medidas de tiempo más precisas que una centésima de segundo, ya que toma un valor de 100 toques de reloj por segundo.

2.4 Códigos fuente del Banco de Pruebas.

En este apartado se expondrán los códigos fuente que hicieron posible el banco de pruebas, para los que se comentarán los resultados obtenidos y el algoritmo utilizado en aquellos casos cuyo interés sea el suficiente.

2.4.1 Desarrollo del Banco de Pruebas para Flotantes.

Debido a que el desarrollo del banco lleva asociado la edición y puesta en marcha de algoritmos de nueva construcción, se hace necesario la elaboración de varias fases de diseño hasta lograr el objetivo deseado, considerando parámetros como el tiempo de cómputo y el tamaño del código.

2.4.1.1 Fase 1: "BancoDePruebas.cpp".

El planteamiento de esta fase parte de la elaboración de un conjunto de muestras de la variable aleatoria uniforme comprendida entre 0 y 1. Sobre cada una de estas muestras se aplica la operación base, sea suma o producto.

Al ejecutar este código en la estación *PC* se obtiene la siguiente tabla de datos:

Nº de muestras	Tiempo de Suma (s)	Tiempo de Producto (s)
10000	0.06	0.05
30000	0.17	0.16
50000	0.28	0.27
100000	0.55	0.55
500000	2.69	2.69
1000000	5.71	5.49

Notas sobre resultados:

- Se invierte gran parte del tiempo de procesado en la reserva dinámica de la tabla de muestras aleatorias. La reserva se hace de una sola vez, siendo no practicable en el sistema microcontrolador; por lo tanto se realizará la generación de las sucesivas muestras aleatorias en línea, es decir, a la par que se opera con ellas.
- Gran parte del tiempo computado se debe a operaciones de lectura/escritura, tal y como demuestra el código ensamblador asociado. Una solución a este problema consistiría en no tratar con muestras situadas en una tabla, sino ir generándolas conforme se necesiten.

- Código fuente.

```
#include <stdio.h>
#include <stdlib.h>
#include <dos.h>
#include <conio.h>
```

```
#define NCS 5                   //NCS: Número de Cifras Significativas
struct TABLA{
        int n;                  //número de elementos de la tabla
   float *data;                 //vector de datos de la tabla
};
struct TABLA inicializaTabla(struct TABLA);
float VA(int);                  //genera variable aleatoria U(0,1)
float tiempoSuma(struct TABLA);
float tiempoProducto(struct TABLA);
float tiempoSistema(void);
void main()
{
   int n;                       //nº de muestras
   struct TABLA tab;
   printf(" Número de muestras? ");
   scanf("%d",&n);
   tab.n=n;
   tab.data = (float *)calloc(n,sizeof(float));
   tab = inicializaTabla(tab);
   printf("\n Tiempo de suma: %.2f s\n",tiempoSuma(tab));
   printf("\n Tiempo de producto: %.2f s\n",tiempoProducto(tab));
   printf("\n Pulsa cualquier tecla para terminar...\n");
   while(!kbhit());    //mientras no se pulse una tecla
   free(tab.data);
   return;
}
//---------------------------------------------------------------------------
struct TABLA inicializaTabla(struct TABLA tabla)
{
        //la variable aleatoria (VA) tendrá NCS cifras significativas
        // la VA será U(0,1)
   int k;
   for(k=0;k<(tabla.n);k++)
        tabla.data[k]=VA(NCS);
   return tabla;
}

float VA(int N)         //N: número de cifras significativas de la VA
{
   float varAleatoria;
   int rango;
   rango=(int)10^N;
   varAleatoria=(float)random(rango)/(float)rango;
   return varAleatoria;
}

float tiempoSuma(struct TABLA tab)
{
   float time;          //tiempo invertido en la suma
   int k;
   int n;
   float *datos;
   datos=tab.data;                      //usamos variable auxiliar para limitar el número
                                        // de operaciones de lectura/escritura
   n=tab.n;
   time = tiempoSistema();
   for(k=0;k<n;k++)
        datos[k]=datos[k]+VA(NCS);
   time = tiempoSistema()-time;
   return time;
}

float tiempoProducto(struct TABLA tab)
{
   float time;                          //tiempo invertido en la suma
   int k;
   int n;
   float *datos;
   datos=tab.data;                      //usamos variable auxiliar para limitar el número
   n=tab.n;                     // de operaciones de lectura/escritura

   time = tiempoSistema();
   for(k=0;k<n;k++)
        datos[k]=datos[k]*VA(NCS);
```

```
   time = tiempoSistema()-time;
   return time;
}

float tiempoSistema(void)
{
   struct time t;
   float tiempo;
   gettime(&t);
   tiempo=(float)t.ti_hour*3600.0+(float)t.ti_min*60.0+(float)t.ti_sec+(float)t.ti_hund*0.01;
   return tiempo;
}
```

- Librerías ".h".

En este código se hace uso de la librería *"dos.h"* compatible con *DOS*. En ella se define la estructura *"time"* y una serie de funciones para el tratamiento de la misma a partir de la información que suministra el reloj del sistema. Se trata de un generador de eventos de tiempo periódicos que depende del reloj de cuarzo del equipo. La estructura *"time"* se define como sigue:

```
struct time{
        unsigned char ti_min;       //minutes
        unsigned char ti_hour;      //hours
        unsigned char ti_hund;      //hundredth of second
        unsigned char ti_sec; //seconds
};
```

Las instancias asociadas a esta estructura deben la gestión de sus valores a las funciones *"gettime"* y *"settime"* que pertenecen a la librería, no compatible con la normativa de *ANSI C*, de nombre *"dos.h"*. La primera de ellas, partiendo del tiempo del sistema, rellena los campos de la instancia dada como parámetro por referencia; en cambio, la segunda de ellas, actualiza el tiempo del sistema tomando como dato la instancia concreta.

2.4.1.2 Fase 2: "BancoDePruebas2.cpp".

En esta fase la generación de muestras de la variable aleatoria se lleva a cabo dentro del bucle que repite la operación base, obteniéndose los resultados:

Nº de muestras	Tiempo de Suma (s)	Tiempo de Producto (s)
10^6	11.21	10.76

La generación interna de las muestras ralentiza en demasía el tiempo de cómputo.

- Código fuente.

Tan sólo incluiremos las diferencias significativas frente al código de la fase anterior. Sea:

```
float tiempoSuma(int noper)
{
   float time;          //tiempo invertido en las sumas
   int k;
```

```
   time = tiempoSistema();
   for(k=0;k<noper;k++)          // se realiza un número de operaciones 'noper'
      VA(NCS)+VA(NCS);
   time = tiempoSistema()-time;
   return time;
}

float tiempoProducto(int noper)
{
   float time;          //tiempo invertido en los productos
   int k;
   time = tiempoSistema();
   for(k=0;k<noper;k++)
        VA(NCS)*VA(NCS);
   time = tiempoSistema()-time;
  return time;
}
```

2.4.1.3 Fase 3: "bancoDePruebas3.cpp".

En esta fase se genera una única muestra de la variable aleatoria fuera de línea con la que se opera repetidas veces. Bajo esta idea se obtiene la siguiente tabla:

Nº de operaciones	Tiempo de Suma (s)	Tiempo de Producto (s)
10^5	0	0
$5x10^5$	0.49	0
10^6	0.27	0
$5x10^6$	0.71	0.17
10^7	0.77	0.33
$5x10^7$	2.17	1.48
10^8	3.52	3.02
10^9	30.98	30.32

Hasta 5 x 10^7 operaciones el efecto multitarea del sistema operativo *Windows XP* es verdaderamente apreciable.

- Código fuente.

Únicamente se incluirán las líneas de código distintivas de esta fase, es decir:

```
float tiempoSuma(int noper)
{
   float time;          //tiempo invertido en las sumas
   int k;
   float va;
   va=VA(NCS);
   time = tiempoSistema();
   for(k=0;k<noper;k++);          // se realiza un número de operaciones 'noper'
            va+va;
   time = tiempoSistema()-time;
   return time;
}

float tiempoProducto(int noper)
{
   float time;          //tiempo invertido en los productos
   int k;
   float va;
   va=VA(NCS);
   time = tiempoSistema();
   for(k=0;k<noper;k++);
            va*va;
   time = tiempoSistema()-time;
   return time;
```

}

2.4.1.4 Fase 4: "BancoDePruebas3_1.cpp".

Respecto a la fase 3 se ha omitido la instrucción asociada a la operación básica, obteniéndose la tabla de medidas siguiente:

Nº de operaciones	Tiempo de Suma (s)	Tiempo de Producto (s)
10^7	0.77	0.16
10^8	2.36	1.81
$5x10^8$	9.88	9.28
10^9	19.33	18.56

Hasta ahora se ha usado un conjunto de librerías que son incompatibles con *ANSI C*, aunque válidas en un entorno *DOS*. Para poder probar los códigos en la plataforma de *ATMEL* hemos de forzar que las librerías sean compatibles con *ANSI C*, según la documentación proporcionada por este fabricante. Para ello se hizo uso de la bibliografía [B7] y [B8], obteniéndose los siguientes prototipos de funciones con su funcionalidad básica:

- *clock_t clock(void);*
  - Compatible con *ANSI C* e incluido en *"time.h"*.
  - Devuelve el número de ciclos de reloj del sistema desde el comienzo de ejecución de la aplicación que la llama. Para transformar el segundos se divide el resultado por la macro *CLK_TCK*.
  - El tipo *clock_t* se constituye como un tipo definido a partir de *long unsigned int*.
- *time_t time(time_t *hora);*
  - Compatible con *ANSI C* e incluido en *"time.h"*.
  - Devuelve el tiempo del sistema en el formato "hora del calendario".
  - El tipo *time_t* se define como *long unsigned int*.
  - Si la variable *"hora"* es igual que la macro *NULL*: Devuelve el tiempo del sistema.
  - Si dicha variable es distinta que *NULL*: Además, inicia el puntero dado con el tiempo del sistema.
- *double difftime(time_t hora2, time_t hora1);*
  - Compatible con *ANSI C* e incluido en *"time.h"*.
  - Devuelve la diferencia entre la variable *"hora2"* y la *"hora1"* en segundos.
- *long biostime(int orden, long nuevaHora);*
  - Pertenece a *"bios.h"*, no compatible con *ANSI C*.
  - Lee o escribe el reloj del sistema; siendo la hora dentro de cada día a partir de las 00:00.

  - Si la variable *"orden"* vale 0: Devuelve el valor de la hora del sistema en *ticks* o toques del reloj del sistema.
  - Si vale 1: Devuelve el valor de la hora dada por *"nuevaHora"* y la escribe en el reloj del sistema.

- *void delay(unsigned time);*
  - Pertenece a *"dos.h"*, no compatible con *ANSI C*.
  - Detiene la ejecución de un programa durante el tiempo especificado en *"time"* (ms).
  - Función relacionada: *sleep()*.

- *void ftime(struct timeb *hora);*
  - Pertenece a *"sys\time.h"*; no compatible con *ANSI C*.

- *int kbhit(void);*
  - Pertenece a *"conio.h"*; no compatible con *ANSI C*.

- *void sleep(unsigned time);*
  - Pertenece a *"dos.h"*; no compatible con *ANSI C*.
  - Suspende la ejecución del programa durante el tiempo indicado por *"time"* (segundos).

- *void exit(int estado);*
  - Pertenece a *"stdlib.h"*; siendo compatible con *ANSI C*.
  - Devuelve la ejecución al sistema operativo.
  - Si el parámetro *"estado"* vale cero: Terminación normal *(EXIT_SUCCESS)*.
  - Si *"estado"* toma el valor uno: Terminación con error *(EXIT_FAILURE)*.

2.4.1.5 Fase 5: "BancoDePruebas3_5.cpp".

La adaptación del código de la fase 4 a la normativa *ANSI C* da lugar al código de esta fase.

En las funciones de esta fase se hace uso de la función *"clock()"* que devuelve el tiempo del reloj del sistema en toques de reloj (o *ticks*). Resulta interesante resaltar el significado de la macro *"CLK_TCK"* porque en ella reside el número de *ticks* por segundo que marca el reloj del sistema; el valor de esta macro depende del sistema físico concreto, así pues, para la estación PC vale 1000, mientras que para la placa de *ATMEL* toma un valor de 100.

Los resultados de las medidas de tiempo que se ofrecen a través de la consola del depurador *AXD* son múltiplos de 100; lo cual es coherente con el valor de la macro de tiempo,

pero nos obliga a incluir cambios en el código para ganar en precisión en la medida de los tiempos.

- Código fuente.

Incluiremos aquí sólo la parte del código que presenta los cambios de adaptación:

```
float tiempoSuma(int noper)
{
   clock_t nticks;                          //número de ciclos de reloj (ticks) invertido en las sumas
   float time;                              //número de segundos invertido en las sumas
   int k;
   float va;
   va=VA();
   nticks = clock();
   for(k=0;k<noper;k++)                     // se realiza un número de operaciones 'noper'
            va+va;
   nticks = clock()-nticks;
   time = nticks/CLK_TCK;
   return time;
}
```

No se ha expuesto el cuerpo de la función *"tiempoProducto"* puesto que su funcionalidad es idéntica a la de la función *"tiempoSuma"*, que sí se ha incluido, salvo que la instrucción *"va+va;"* sea sustituida por *"va*va;"*.

De acuerdo con los resultados del proceso de compilación, *MetroWerks* nos indica que se requieren 24282 bytes para albergar este código en la memoria de la placa de evaluación *AT91EB40A*.

2.4.1.6 Fase 6: "BancoDePruebas3_8.cpp".

Con el objetivo de adquirir mayor precisión en la medida de tiempos se aumenta el tiempo total del experimento sin mas que considerar más iteraciones. Obtenemos, tras ejecutar el código en el microprocesador de *ATMEL*, la siguiente tabla de resultados que se adjunta en la memoria correspondiente en el apartado *"1.4.1.1 Aritmética Flotante"*.

Gracias a la configuración del compilador de *Metrowerks CodeWarrior* sin optimización se permite que toda instrucción de C se traduzca a ensamblador; así, se consigue asociar código a la instrucción *"aux = va + va;"*.

Tras el proceso de compilación, se nos indica, a modo de tabla resumen, que la cantidad de memoria que se necesita asciende a 16914 bytes.

2.4.2 Desarrollo del Banco de Pruebas para Enteros.

Este banco de pruebas se lleva a cabo mediante la adaptación del código de *"BancoDePruebas3_8.cpp"* desde el tipo flotante al entero. En este caso no partíamos desde el principio pudiendo utilizar lo que ya se había comprobado y verificado.

2.4.2.1 Fase 1: "BancoDePruebasInt.cpp".

Se incluirá el código fuente relevante:

```
clock_t tiempoSuma(int noper)
{
   clock_t nticks;                              //número de ticks invertido en las sumas
   int k;
   int va;
   int aux;
   va=(int)VA();
   nticks = clock();
   for(k=0;k<noper;k++)            // se realiza un número de operaciones 'noper'
                        aux=va+va;
   nticks = clock()-nticks;
   return nticks;
}
```

La variable relativa a la operación se ha declarado como entera, esto fuerza a que el compilador genere la función *"add"* en lugar de *"fadd"*, que usa con números flotantes. La función asociada a la toma de medidas del producto es análoga.

Según el informe de la compilación, el código ensamblador correspondiente se extiende por 16914 bytes, es decir, por 16.52 Kbytes.

2.4.3 Realización de Ecuaciones en Diferencias: Filtros.

Para lograr un programa que lleve a cabo la ejecución de una ecuación en diferencias causal de orden genérico se ha pasado por 2 fases. Primero se construyó un código para un filtro de orden uno que, finalmente, se extendió a orden genérico considerando. En este apartado tan sólo se tendrá en consideración la aritmética flotante porque los coeficientes de un sistema real no suelen ser, en la gran mayoría de los casos, números enteros.

La generación de los coeficientes se hace gracias a muestras aleatorias para evitar la dependencia de la medida de tiempo con valores concretos de los mismos. De igual modo, la secuencia de partida también se genera a partir de un patrón de muestras uniforme, cuya media y rango son dadas por el usuario.

Ejecutando el código fuente en el sistema de *ATMEL* se logra la siguiente tabla:

Flotantes		Enteros	
Nº operaciones	**Tiempo (s)**	**Nº operaciones**	**Tiempo (s)**
1000	21	**1000**	19

De las medidas se desprende que:

- Para flotantes: Cada iteración de la ecuación consume 21 ms.
- Para enteros: Consume 19 ms.

2.4.3.1 Fase 1: "filtro_orden1.cpp".

- Código fuente.

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define b 2
#define a 0.8
int main(void)
{
        unsigned int n,N;
        float y,y1,x,x1;        //variables para la ecuación en diferencias
        float mx, vx;           //valor medio (mx) y varianza (vx) de la entrada
        clock_t t0,Dt;          //valores de ticks
        x=0.0; x1=0.0; y=0.0; y1=0.0; t0=0; Dt=0;  // Valores iniciales
        printf("\n\tValor medio de la excitación:");
        scanf("%f",&mx);
        printf("\n\tVariación de la excitación:");
        scanf("%f",&vx);
        printf("\n\n\tNúmero de iteraciones:");
        scanf("%u",&N);
        for(n=0;n<N;n++)
        {
                x=mx+vx*((float)rand()/(float)RAND_MAX-0.5);
                t0=clock();
                y=-a*y1+b*x1;       //ecuación en diferencias de orden (1,1)
                Dt+=clock()-t0;     //acumula la diferencia de tiempos para cada iteración
                x1=x;
                y1=y;
        }
        printf("\n\tTiempo invertido en %d iteraciones: %u s",N,Dt/CLK_TCK);
        return 0;
}
```

En donde:

- *'x'*: Muestra actual de la secuencia de entrada.
- *'x1'*: Muestra de la secuencia de entrada de la iteración anterior.
- *'y'*: Muestra actual de la secuencia de salida.
- *'y1'*: Análogo a 'x1', pero respecto a la secuencia de salida.
- *'Dt'*: Acumulador de medidas de tiempo para cada iteración.

Resulta interesante señalar que tal y como se ha escrito el código sólo se computa el tiempo invertido en la ecuación en diferencias.

El resultado para flotantes lo da este código, en cambio, el de enteros se logra mediante *"filtro_orden1_int.cpp"*. Éste no es más que cambiar las variables *'x'*, *'x1'*, *'y'* e *'y1'* de flotantes a enteros del código *"filtro_orden1.cpp"*.

De acuerdo con el informe emitido al término de la compilación, este algoritmo ocupa 26378 bytes, siendo el 10,06% del total de la memoria *RAM* disponible, que son 256 Kbytes.

2.4.3.2 Fase 2: "filtroNM.cpp".

La extensión de la fase anterior desemboca en una ecuación en diferencias causal de orden genérico. En este caso, los coeficientes poseen dos modos de generación: Una permite que sea el usuario quién indique sus valores; otra obtiene los valores de muestras de una variable aleatoria uniforme que define el mismo usuario.

Para este caso la tabla que se obtiene con el microprocesador de *ARM* se presenta en el apartado de la memoria *"2.3.5.1 Ecuaciones en Diferencias: Filtros"*.

- Código fuente.

```
//filtro (n,m): ecuación en diferencias.
//Estudio de tiempo de cómputo.
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
typedef float muestra;//tipo de datos de cada muestra
typedef unsigned int indice;
#define ESPACIO 32
int main(void)
{
   clock_t t0, Dt;                  //t0: tiempo inicial; Dt: tiempo invertido
   muestra *a;                      //tablas que contienen los coeficientes del filtro
   muestra *b;
   muestra max,min;  //valores máximo y mínimo de los coeficientes
   muestra *x;                      //tabla que contiene los m últimos valores de la excitación
   muestra *y;                      //tabla que contiene los n últimos valores de la respuesta
   muestra mx,vx;                   //valor medio (mx) de la excitación; su variación (vx)
   indice i;                        //índice para las iteraciones
   indice k;                        //índice para las muestras
   indice n;                        //orden del denominador del filtro
   indice m;                        //orden del numerador del filtro
   indice N;                        //número de iteraciones del filtro
   indice modo = 0;                 //Modo de generación del filtro

   //entrada de los órdenes del filtro
   printf("\n\tDefina los órdenes del filtro:");
   printf("\n\tOrden del numerador (b[m]): ");
   scanf("%u",&m);
   printf("\n\tOrden del denominador (a[n]): ");
   scanf("%u",&n);
   a=(muestra *)calloc(n,sizeof(muestra));
   b=(muestra *)calloc(m,sizeof(muestra));
   x=(muestra *)calloc(m+1,sizeof(muestra));
   y=(muestra *)calloc(n+1,sizeof(muestra));
   printf("\n\n\tGeneración de los coeficientes del filtro:\n\t- manual(1);\n\t- automático(no 1)\n");
   scanf("%u",&modo);
   if(modo==1)                      //definición manual del filtro
   {
                              //entrada de los coeficientes del filtro
         printf("\n\tIntroduzca los coeficientes de a[n]:\n");
```

```
            for(k=0;k<n;k++)
            {
                        printf("\n\ta[%u]=",k+1);
                        //definir el formato de acuerdo con 'muestra'
                        scanf("%f",&a[k]);
            }
            printf("\n\tIntroduzca los coeficientes de b[m]:\n");
            for(k=0;k<m;k++)
            {
                        printf("\n\tb[%u]=",k+1);
                        //definir el formato de acuerdo con 'muestra'
                         scanf("%f",&b[k]);
            }
   }
   else                 //generación aleatoria del filtro
   {
            printf("\n\n\tValor máximo de los coeficientes: ");
            scanf("%u",&max);
            printf("\n\n\tValor mínimo de los coeficientes: ");
            scanf("%u",&min);
            for(k=0;k<n;k++)
                        b[k]=(muestra)rand()/(muestra)RAND_MAX*(max-min)+2*min-max;
            for(k=0;k<m;k++)
                        a[k]=(muestra)rand()/(muestra)RAND_MAX*(max-min)+2*min-max;
   }
   printf("\n\n\tIndique el número de iteraciones del filtro: ");
   scanf("%u",&N);
                        //definir el formato de acuerdo con 'muestra'
   printf("\n\tValor medio de la excitación: ");
   scanf("%f",&mx);
   printf("\n\tVariación de la excitacion: ");
   scanf("%f",&vx);
   //valores iniciales   (según el formato de 'muestra')
   for(k=0;k<n;k++)
              y[k+1]=0.0;
   for(k=0;k<m;k++)
            x[k+1]=0.0;
   Dt=0;
   //Comienzo de las iteraciones
   for(i=0;i<N;i++)
   {
            x[0]=mx+vx*((float)rand()/(float)RAND_MAX-0.5);
            t0=clock();
            for(k=1;k<=n;k++)
                        y[0]+=-a[k-1]*y[k];
            for(k=1;k<=m;k++)
                        y[0]+=b[k-1]*x[k];
            Dt+=clock()-t0;

            //actualizacion de variables
            for(k=n;k>0;k--)
                        y[k]=y[k-1];
            for(k=m;k>0;k--)
                        x[k]=x[k-1];
   }

   printf("\n\n\tPara %u iteraciones se ha invertido %.3f s.",N,Dt/CLK_TCK);
   printf("\n\n\tPulse espacio intro para salir...");
   while(getchar()!=ESPACIO);
   return 0;
}
```

Debido a que el orden del filtro es genérico, se precisa de una reserva dinámica de memoria para los vectores que almacenan las últimas m muestras para la secuencia de entrada y las últimas *n* para la de salida. De igual forma de ha de actuar para los vectores de los coeficientes, los cuales son generados aleatoriamente en el modo *"automático"*.

El cómputo del tiempo sólo recoge el código de control de flujo que implementa la ecuación en diferencias. A diferencia de la ecuación de orden 1, que sí se conoce el número de términos que la forman al editar el código, con un filtro genérico se requiere de control de flujo para la propia ecuación. Esto hace que el tiempo efectivo sea menor, proporcionalmente, que para un orden conocido.

El montante de bytes que ocupa este código en la memoria *RAM* de la placa de evaluación asciende, de acuerdo con los informes pertinentes, a 31026 bytes, alcanzándose el 11,83% del máximo.

2.4.4 Cálculos Estadísticos.

Se ha decidido que la generación de las muestras se realice en línea respecto al cálculo de los estadísticos. Esto se debe a que se intenta imitar el estimador que construye una ley de adaptación sobre un controlador ajustable de forma automática; siendo esta estrategia muy utilizada en entornos variables o dependientes de las condiciones externas.

Se ha elaborado el código adecuado que obtenga las medidas de tiempo para los diferentes estadísticos, siendo aplicado tanto para aritmética flotante como para aritmética entera. La tabla de resultado se expone en el apartado *"2.3.5.2 Cálculos Estadísticos"* de la memoria mostrándose sólo el caso flotante, puesto que el caso entero no añade información relevante.

2.4.4.1 Fase 1: "calculosEstadisticos2.cpp".

Este código se desarrolla para su aplicación sobre números expresados bajo aritmética flotante. Comentaremos su código fuente en *C++*.

- Código fuente.

```
//inclusion de librerias
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>

//definicion de tipos
typedef unsigned int contador;
typedef float muestra;

//constantes
#define ESPACIO 32
#define NNtab 5                 //diferentes nº de muestras
#define nntab 4                 //diferentes nº de ordenes

int main(void)
{
        //variables locales
```

```
	contador k; contador i,j;
	contador N;		//número total de muestras (iteraciones)
	contador n;		//orden del estadístico
	contador Ntab[NNtab];
	contador ntab[nntab];

	muestra x;
	muestra mx;		//x estimada, en valor promedio
	muestra chi;		//chi = <x-mx>^n; estadístico de x de orden n respecto a su valor medio
	muestra media;
	muestra rango;

	clock_t t0; //t0: tiempo inicial;
	clock_t Dt;		//Dt: incremento acumulativo de tiempo

	//Diversos números de operaciones
	Ntab[0]=100; Ntab[1]=1000; Ntab[2]=2000; Ntab[3]=5000; Ntab[4]=10000;
	//Diversos órdenes
	ntab[0]=2; ntab[1]=3; ntab[2]=5; ntab[3]=10;

	//ALGORITMO
	printf("\n\nCalculos estadisticos:");
	printf("\nValor medio de las muestras? ");
	scanf("%f",&media);
	printf("\nValor del rango de las muestras? "),
	scanf("%f",&rango);
	for(i=3;i<NNtab;i++)
	{
		N=Ntab[i];		//definimos el número de muestras de esta iteración
		for(j=0;j<nntab;j++)
		{
			printf("\nOperando...");
			n=ntab[j];  //definimos el órden del estadístico en esta iteración
			//Valores iniciales
			mx=0.0; chi=0.0; Dt=0;
			for(k=0;k<N;k++)
			{
				// media-rango/2 <= x <= media+rango/2
				x=media+rango*((float)rand()/(float)RAND_MAX-0.5);
				t0=clock();
				mx+=x/N;		//media acumulativa
				chi+=(muestra)pow((double)(x-mx),(double)n)/N;   //estadístico acumulativo
				Dt+=clock()-t0;			//acumulador de diferencias de tiempo
			}
			//Resultados
			printf("\nNúmero de muestras: %u",N);
			printf("\nOrden del estadístico: %u",n);
			printf("\n<x>[%u]=%.3f; chi[%u]=%.3f",k,mx,k,chi);
			printf("\nTiempo invertido: %u s",Dt/CLK_TCK);
			while(getchar()!=ESPACIO);
		}
	}
	return 0;
}
```

Se calculan estadísticos de diferentes órdenes –los indicados en la tabla *"nntab"* – para diferentes espacios muestrales –los especificados en la tabla *"NNtab"*– de una variable aleatoria uniforme de media –variable *"media"*– y amplitud –variable *"rango"* – definidas por el usuario.

La medida acumulada de los tiempos de cómputo se realiza en la variable *"Dt"*. Por la forma en la que se lleva a cabo el proceso de medida, tan sólo se considera en tiempo invertido en los cálculos de los estadísticos y en el control de flujo a él asociado, quedando excluido el resto del código.

Para evaluar el estadístico de orden superior se usa la función *"pow"* recogida por la normativa *ANSI C*; esta inclusión provoca que se produzca un gran número de instrucciones en código ensamblador que no está directamente relacionado con el cálculo del propio estadístico, luego, la importancia del código de control de flujo gana peso en la medida del tiempo. De aquí que no se aprecie dependencia entre el tiempo consumido y el orden del estadístico, al menos, dentro del intervalo de órdenes que se ha considerado.

El número total de bytes que se requiere para cubrir este algoritmo llega a ser 37330 bytes; considerando que la cantidad de memoria *RAM* disponible es de 256 Kbytes, este programa copa el 14,24%. Esto, en determinadas situaciones, puede ser condicionante al diseño de algoritmos de control.

2.4.4.2 Fase 2: "calculosEstadisticos3.cpp".

En este código se define el tipo *"muestra"* como entero. Esto afecta ligeramente a los tiempos de cálculo, puesto que sólo modifica la naturaleza de las muestras, de su media y la de los estadísticos de orden superior, manteniendo el uso de la función *"pow"*.

- Código fuente.

Este código es idéntico al expuesto en el apartado "*2.4.4.1.1 Código fuente."*, salvo que el tipo *"muestra"* se construye a partir del tipo predefinido *"int"* o entero. Incluiremos aquí el código que marca esta diferencia:

```
//definicion de tipos
typedef unsigned int contador;
typedef int muestra;
```

Según el informe de la compilación, el código de este apartado ocupa 27982 bytes, es decir, el 10,68%. La manera en la que se planteen las soluciones ante un mismo problema modifica, notablemente, la cantidad de memoria que se necesita para albergarlo.

2.4.5 Inversión de Matrices.

En este apartado se describirá el funcionamiento de los algoritmos que realizan la inversión matricial con detenimiento puesto que la simple lectura de los códigos fuente no es, por sí misma, suficiente para una correcta compresión.

Por otra parte, se distinguen tres fases dentro del desarrollo de los algoritmos. Aquí sólo se expondrá el código responsable de la última fase puesto que en ella se recoge el resultado de las demás. En la primera fase se diseña el algoritmo de inversión y se formulan las estructuras

de datos, en la segunda se añade el cómputo del tiempo invertido y, en la tercera se engloban las funciones propias de la operación en un fichero independiente que será enlazado con el fichero principal.

Primeramente se expondrá la definición de la matriz inversa a una matriz dada:

- Dada una matriz A cuadrada de dimensión n:

$$A = \begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nn} \end{bmatrix}; \quad [9]$$

- Se define su matriz inversa A^{-1}, a partir de su adjunta Adj(A), como:

$$A^{-1} = \frac{1}{|A|} \cdot \left(Adj(A)\right)^T; \quad [10]$$

- Siendo cada elemento (i, j) de la matriz adjunta:

$$B^{ij} = \begin{bmatrix} a_{11} & \cdots & a_{1,j-1} & a_{1,j+1} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots & \vdots & & \vdots \\ a_{i-1,1} & \cdots & a_{i-1,j-1} & a_{i-1,j+1} & \cdots & a_{i-1,n} \\ a_{i+1,1} & \cdots & a_{i+1,j-1} & a_{i+1,j+1} & \cdots & a_{i+1,n} \\ \vdots & & \vdots & \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{n,j-1} & a_{n,j+1} & \cdots & a_{nn} \end{bmatrix}; \quad [11]$$

- Definiéndose B^{ij} como la siguiente submatriz de *'A'* de dimensión *n-1*:

$$Adj(A)_{ij} = (-1)^{i+j} \cdot \left|B^{ij}\right|; \quad \forall i, j = 1,2,\ldots,n; \quad [12]$$

Para la construcción de la matriz adjunta hace falta calcular n^2 determinantes de orden *n-1*, siendo la definición por filas del operador determinante la siguiente:

Esta definición, tal y como queda expresada, no es cerrada. Es decir, que el determinante de una matriz de dimensión *n* se construye a partir de determinantes de varias matrices, extraídas de la anterior, cuya dimensión es *n-1*. Para cerrar la definición se requiere definir qué valor toma el determinante de una matriz de dimensión 1, es decir, de un escalar; esto es: El determinante de un escalar es el propio escalar.

La definición adoptada para el operador determinante obliga a que las funciones en *C++* que realizan la inversión matricial se relacionen recursivamente.

$$\left|A\right| = \sum_{k=1}^{n} \left\{ (-1)^{i+k} \cdot a_{ik} \cdot \left|B^{ik}\right| \right\}, i = 1,2,\cdots,n; \quad [13]$$

Las pruebas realizadas hacen uso de diferentes matrices cuyo orden comienza en 3 y termina en 8. Gracias a la función de generación automática de matrices se ahorra un tiempo valioso en la realización de los sucesivos experimentos; ésta evita, por un lado, que el usuario teclee, una a una, las numerosas entradas de una matriz –n^2 entradas para una matriz de orden n– y, por otro, que este proceso se repita con las diferentes matrices que forman el experimento.

Para que sirva de ejemplo ilustrativo se muestra una matriz de orden 4, que se generó automáticamente, y su correspondiente matriz inversa, que se calculó mediante el algoritmo propuesto.

Esto es:

$$A = \begin{bmatrix} 0.021 & 0.008 & 0.670 & 0.067 \\ 0.711 & 0.434 & 1.074 & 0.392 \\ 1.401 & 1.900 & 0.550 & 0.889 \\ 0.219 & 1.396 & 1.129 & 0.083 \end{bmatrix}; \quad B = \begin{bmatrix} -6.596 & 4.153 & -1.396 & 0.645 \\ -0.132 & -0.661 & 0.246 & 0.587 \\ 0.683 & 0.412 & -.252 & 0.211 \\ 10.257 & -5.388 & 2.956 & -2.404 \end{bmatrix}; \quad [14]$$

En donde se verifica que A x B = I, luego se comprueba que $B \equiv A^{-1}$.

Las pruebas llevadas a cabo sobre el sistema de desarrollo de *ATMEL* devuelven un valor de cero con independencia del orden. Esto se debe a la realización particular de la macro *CLK_TCK*, que sólo se modifica cuando el número de *ticks* alcanzados es múltiplo de 100, es decir, la precisión que ofrece no es menor del segundo.

Para aumentar la precisión artificialmente se debe repetir el mismo experimento, o sea, invertir múltiples veces la misma matriz y contabilizar todo el tiempo invertido; tan sólo restaría dividir el tiempo total entre el número de inversiones. Pero debido a que la inversión reiterada bloqueaba el sistema de *ATMEL*, no pudimos realizar estas pruebas; aunque se aislasen y delimitasen las llamadas a los destructores y constructores de forma que únicamente una *"struct *matriz"*, para la matriz original, estuviera en memoria al mismo tiempo no se lograba el bloqueo del sistema. Éste impedía cualquier trabajo mediante el depurador *AXD Metrowerks CodeWarrior*.

La inversión de una sola matriz en el microprocesador *ARM7TDMI* se realizaba aparentemente mucho más rápido que en el equipo de sobremesa, incluso para órdenes mayores que 8. Pero, debido a que no fue posible obtener medidas fiables con este sistema, se decidió presentar la tabla de resultados que ofrecía la estación *PC* cuyos tiempos eran más restrictivos que los de *ATMEL*.

Partiendo del informe que nos ofrece el proceso de compilación, podemos afirmar que el algoritmo necesita de 32298 bytes de la memoria *RAM*. En estos bytes se albergan los códigos asociados a los dos ficheros de código fuente: *"matrices.cpp"* y *"InvertirMatriz5.cpp"*.

2.4.5.1 Funciones Diseñadas para la Inversión Matricial: "matrices.cpp".

Se ha elaborado un conjunto de funciones que llevan a cabo la construcción de cada uno de los pasos fundamentales que requiere una inversión matricial. Para ello se ha diseñado una función que extrae la submatriz B^{ij}, otra que calcula el valor del determinante y otra que calcula la matriz inversa a partir de los resultados parciales de las otras dos.

Asimismo, para el correcto tratamiento de la información residente en cada una de las matrices que aparece en el proceso se ha conformado una estructura de datos *ex profeso*. La cual, expresada en código *C++*, queda de la forma:

```
struct matriz{
   entrada ** data;          // puntero a las entradas de la matriz
   int nfilas;               // número de filas
   int ncols;                // número de columnas
};
```

Por otro lado, debido a que los algoritmos necesitan la asignación dinámica de memoria, con el fin de minimizar la cantidad de recursos en uso para un instante dado, se decidió conveniente definir funciones de construcción y destrucción de matrices.

- Código fuente.

```
#include <stdio.h>
#include <stdlib.h>

typedef float entrada; //tipo de entrada de la matriz

extern struct matriz *mat;
extern struct matriz *matrix;
extern struct matriz *adj;
extern int modo;          //Generación automática de entradas o dadas por el usuario
extern int n;             //Dimensión de la matriz a invertir
extern int nfil;          //Número de filas
extern int ncol;          //Número de columnas
extern int i;             //Índice para las filas
extern int j;             //Índice para las columnas
```

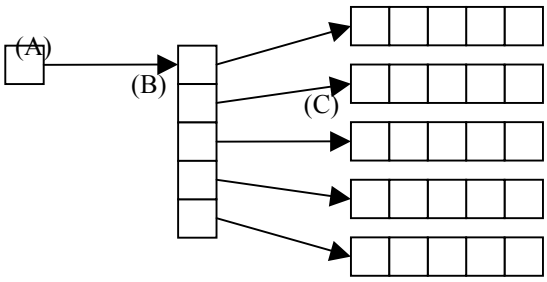
Se han usado variables denominadas como externas –*"extern"*– puesto que serán procesadas e iniciadas por otras funciones situadas en diferentes módulos definidos en otros ficheros *C++* y que, posteriormente, se usarán por las funciones de *"matrices.cpp"*.

```
//Definición de la estructura matriz
struct matriz{
   entrada ** data;     // puntero a las entradas de la matriz
   int nfilas;                  // número de filas
   int ncols;                   // número de columnas
};
```

Esta función muestra por la consola la matriz dada como parámetro; nótese que sólo es compatible con la estructura de datos que define a una matriz. El formato de representación aquí expuesto resulta el apropiado cuando se trabaja con entradas de naturaleza flotante.

```
void presentaMatriz(struct matriz *mat)
{
   int fil;                     //índice usado para recorrer la matriz por filas
   int col;                     //ídem, pero para las columnas
   for(fil=0;fil<(mat->nfilas);fil++)
   {
            printf("\n[");
            for(col=0;col<(mat->ncols);col++)
                     //según el tipo de entrada hay que modificar
                     //el formato de presentación
            printf(" %.3f",mat->data[fil][col]);
      printf("]");
   }
   printf("\n");
   return;
}
```

En esta función se reserva dinámicamente la memoria necesaria para albergar a una estructura matriz y a todas sus entradas. El resultado lo devuelve bajo un puntero a *"struct *matriz"*. El campo puntero señala a un vector de punteros a vectores de entradas, gráficamente queda de la forma:



- En donde:
  - (A): Es el campo puntero de la estructura matriz.
  - (B): Es el vector de punteros.
  - (C): Es el conjunto de vectores de entradas.

```
struct matriz *constructorMatriz(int nfil,int ncol)
{
   struct matriz *mat; //Puntero que apuntará a la matriz reservada
   int fil;                     //índice para las filas
   int col;                     //índice para las columnas
            //Construcción de los campos de la struct matriz
```

```
   mat = (struct matriz*)calloc(1,sizeof(struct matriz));
   mat->nfilas=nfil;
   mat->ncols=ncol;
            //Construcción de la estructura de punteros de la matriz
   mat->data = (entrada **)calloc(mat->nfilas,sizeof(entrada *));          //Puntero a las filas
   for(fil=0;fil<(mat->nfilas);fil++)
   {
      mat->data[fil]=(entrada *)calloc(mat->ncols,sizeof(entrada));        //Puntero a las entradas de cada fila
      for(col=0;col<(mat->ncols);col++)
            //Para inicializar: según el tipo de dato
            mat->data[fil][col]=0.0;
   }
   return mat;
}
```

El propósito de esta función es complementario al de la función anterior. Se necesita debido a que los diferentes grupos de datos que constituyen una estructura matriz no están definidos explícitamente en la propia *"struct matriz"*. Así pues, con una llamada a *"free(matriz)"* no libera la memoria asociada a los vectores de entradas ni al vector de punteros; hemos de hacerlo explícitamente.

```
void destructorMatriz(struct matriz *mat)
{
   int fil;
   for(fil=0;fil<(mat->nfilas);fil++)
            free(mat->data[fil]);                                  //Liberamos cada fila
   free(mat);                                                      //Liberamos la estructura
   return;
}
```

Debido a que la definición del operador determinante puede aplicarse a cualquier fila, se pretende elegir la fila que más términos nulos posea con el fin de disminuir el tiempo de cálculo del operador. Puesto que la definición del determinante de una matriz cuadrada es recursiva, este planteamiento minimiza el número de llamadas a funciones de forma considerable.

```
int filaMaxNoCeros(struct matriz *mat)
{
   int fil,col;                                                    //Indices para recorrer la matriz
   int nCeros;                                                     //Nº de Ceros que hay en cada fila
   int nCerosMax=0;                                        //1º fila si no hay ningún 0 como entrada
   int filaMax=0;                                                  //Fila que contiene el máximo número de ceros
   for(fil=0;fil<(mat->nfilas);fil++)
   {
      nCeros=0;
      for(col=0;col<(mat->ncols);col++)                    //Contamos el número de 0's de cada fila
      {
            if(mat->data[fil][col]==0.0)
            nCeros++;
      }
      if(nCeros>nCerosMax)
      {
         nCerosMax=nCeros;
         filaMax=fil;                                          //Apuntamos la fila que tiene más ceros
      }
   }
   return filaMax;
}
```

Esta función extrae la submatriz definida como B^{ij} de una matriz dada en el parámetro *"mat"*. Para ello se va copiando cada elemento, salvo los de la fila i–ésima y los de la columna j–ésima, en la estructura de datos reservada para la matriz extraída que es apuntada por la variable *"adj"*.

```
struct matriz *extraerMatrizAdjunta(int i,int j,struct matriz *mat,struct matriz *adj)
{
                        // 'i' y 'j' marcan la fila y la columna críticas, respectivamente
   int fil;
   int col;  //índices de la matriz adjunta (adj)
   for(fil=0;fil<(mat->nfilas)-1;fil++)
   {
      if(fil<i)                     //Si no hemos llegado a la fila: copiamos tal cual
      {
         for(col=0;col<(mat->ncols)-1;col++)
         {
            if(col<j)    //Si no hemos llegado a la columna: copiamos
              adj->data[fil][col]=mat->data[fil][col];
            else                    //Si ya hemos llegado: nos la saltamos
              adj->data[fil][col]=mat->data[fil][col+1];
         }
      }
      else                          //Nos saltamos la fila si la hemos alcanzado
      {
         for(col=0;col<(mat->ncols)-1;col++)
         {
            if(col<j)               //Si no hemos llegado a la columna: copiamos
              adj->data[fil][col]=mat->data[fil+1][col];
            else                    //Si ya hemos llegado: nos la saltamos
              adj->data[fil][col]=mat->data[fil+1][col+1];
         }
      }
   }
   return adj;
}
```

Esta función se utiliza para la operación $(-1)^n$, siendo n un número entero. Puesto que esta expresión únicamente puede tomar los valores 1 ó –1, se plasmó la idea de sumar o restar elementos en el cálculo del determinante a partir de que los índices de fila y columna en cuestión sumasen un número par o no. Es decir:

```
int par(int n)
{
   int flag;
   int nant = n;
   n=n*0.5;                     //Gracias a la aritmética entera se aplicará "parte entera"
   n=2*n;                       // o "función techo" a la hora de redondear si 'n' es impar
   if (nant==n)                 //Si se mantiene: era par
            flag = 1;
   else                         //Si no: impar
            flag = -1;
   return flag;
}
```

La función *"Det"* devuelve el valor del determinante de la matriz apuntada por la variable *"matrix"*. Ésta, para realizar la operación, efectúa la llamada a las funciones *"filaMaxNoCeros"*, *"extraerMatrizAdjunta"* y *"Det"*. El cálculo del determinante, como se define a través de un sumatorio, se hace progresivamente sobre la variable *"det"*.

```
float Det(struct matriz *matrix)
{
   int col = 0;                 //índice del sumatorio (columnas)
   int filaCeros;               //Fila con más ceros
   float det = 0.0;
   struct matriz *adjunta;
   if ((matrix->nfilas)>1 && (matrix->ncols)>1)                 //Si no es un escalar
   {
      filaCeros=filaMaxNoCeros(matrix);
      adjunta=constructorMatriz((matrix->nfilas)-1,(matrix->ncols)-1);

      while(col<(matrix->ncols))                //Para todas las entradas de la fila seleccionada
      {
```

```
        if(matrix->data[filaCeros][col]!=0.0)                    //Sólo para los elementos no nulos
        {
            //Cálculo del determinante
            adjunta=extraerMatrizAdjunta(filaCeros,col,matrix,adjunta);
           if(par(filaCeros+col)==1)
              det+=(float)(matrix->data[filaCeros][col])*Det(adjunta);        //(-1)^(filaCeros+col) == 1
           else
               det-=(float)(matrix->data[filaCeros][col])*Det(adjunta);                //(-1)^(filaCeros+col) == -1
        }
        col++;
     }
     destructorMatriz(adjunta);
   }
   else                                                  // Si la matriz es un escalar, el determinante es el propio escalar
     det=(float)(matrix->data[0][0]);
   return det;
}
```

Esta función coordina todos los esfuerzos de las funciones anteriores de forma que devuelve la matriz invertida de la dada en *"matrix"*.

```
struct matriz *invierteMatriz(struct matriz *matrix)
{
   //Definición de matriz inversa:
   //                    [a11 a12 ... a1n]
   //         Sea A =    [a21 a22 ... a2n]
   //                    [... ... ... ...]
   //         [an1 an2 ... ann]
   //Su matriz inversa se define como:
   //         A^(-1) = |A|^(-1)*adj(A)^t
   //Siendo adj(A)(i,j) = (-1)^(i+j)*|[alm]; todo l!=i; todo m!=j|

   int fil;                                  //índice de filas
   int col;                                  //índice de columnas
   int dim;                          //dimensión de la matriz
   float det;                                //valor del determinante de matrix
   float detAdjunta;                         //valor del determinante de adjunta
   struct matriz *adjunta;                   //puntero a la matriz adjunta
   struct matriz *matinversa;                //puntero a la matriz inversa

   if((det=Det(matrix))!=0 && (dim=matrix->nfilas)==matrix->ncols)
                                     //si el determinante es no nulo y la matriz es cuadrada
   {
      matinversa=constructorMatriz(dim,dim);
      adjunta=constructorMatriz(dim-1,dim-1);
      det=1.0/det;                           //Al aplicar la inversa se ha de dividir por el determinante;
                                             //multiplicar es menos costoso en tiempo que dividir
      for(fil=0;fil<dim;fil++)
      {
         for(col=0;col<dim;col++)
         {
             adjunta=extraerMatrizAdjunta(fil,col,matrix,adjunta);
            detAdjunta=Det(adjunta);
            if(par(col+fil)==1)                                          //(-1)^(col+fil)=1;
                matinversa->data[col][fil]=(entrada)(det*detAdjunta);
            else                                                         //(-1)^(col+fil)=-1;
                matinversa->data[col][fil]=(entrada)(-det*detAdjunta);
         }
      }
      destructorMatriz(adjunta);
      destructorMatriz(matrix);
   }
   else
      matinversa=NULL;
   return matinversa;
}
```

Esta función define la interfaz con el usuario a la hora de definir los valores de las entradas para las matrices que se deseen invertir. Se desarrolló con el fin de facilitar matrices de prueba con las que depurar, probar el código y obtener medidas de tiempo.

```
struct matriz *asignaValoresMatriz(struct matriz *mat,int modo)
{
   int fil;                          //Índice para las filas de la matriz
   int col;                          //Índice para las columnas de la matriz
   entrada min;                      //Valor mínimo para las entradas
   entrada max;                      //Valor máximo para las entradas

   if(modo==1)                       //Si inicialización por el usuario
   {
      for(fil=0;fil<(mat->nfilas);fil++)
      {
         for(col=0;col<(mat->ncols);col++)
         {
             printf("\n Elemento (%i,%i):",fil,col);        //Según el tipo de dato hay que especificar
                                                            // formato adecuado de acuerdo con "entrada"
             scanf("%f",&(mat->data[fil][col]));
         }
         printf("\n");
      }
   }
   else                              //Si generación automática
   {
      int error = 1;
      while (error)
      {
         printf("\n Valor máximo de las entradas: ");
         scanf("%f",&max);                  //Formato adecuado a "entrada"
         printf("\n Valor mínimo de las entradas: ");
         scanf("%f",&min);                  //Formato adecuado a "entrada"
         if(max<=min)
         {
             error=1;
            printf("\n El máximo ha de ser mayor que el mínimo.");
            printf("\n Vuelva a definir el intervalo.");
         }
         else
             error=0;
      }
      for(fil=0;fil<(mat->nfilas);fil++)
         for(col=0;col<(mat->ncols);col++)
             mat->data[fil][col]=(entrada)rand()/(entrada)RAND_MAX*(max-min)+2*min-max;
            //RAND_MAX es una variable aleatoria U(0,100) definida en una macro
            //Las entradas de la matriz son U(min,max)
   }
   return mat;
}
```

Para que el enlazador funcionase correctamente se hacía necesario que el orden de inclusión, dentro del archivo *"matrices.cpp"*, de las diferentes funciones respetara la llamada de funciones, es decir, que las funciones que eran llamadas por otra debían estar previamente incluidas en el archivo. De forma análoga ocurría con las variables declaradas *"extern"*; sólo que, por legibilidad, se incluyeron al inicio del documento.

2.4.5.2 Función Principal: "InvertirMatriz5.cpp".

La función principal o *"main"* se ha definido en el fichero *"InvertirMatriz5.cpp"* y constituye el hilo conductor para que el usuario, fácilmente, elabore la serie de pruebas conveniente.

- Código fuente.

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define ESPACIO 32

//prototipos de las funciones
extern void presentaMatriz(struct matriz *);
extern struct matriz *asignaValoresMatriz(struct matriz *,int);
extern struct matriz *invierteMatriz(struct matriz *);
extern float Det(struct matriz *);
extern struct matriz *extraerMatrizAdjunta(int,int,struct matriz *,struct matriz *);
extern int filaMaxNoCeros(struct matriz *);
extern int par(int);
extern struct matriz *constructorMatriz(int,int);
extern void destructorMatriz(struct matriz *);
```

Los prototipos que se declaran pertenecer a otro archivo *C++*, luego se definen como *"extern"* para indicárselo al enlazador –*"linker"* –.

```
int main()
{
   int modo;                              //modo de generación de matrices
   int dim;                    //nº de filas y columnas de la matriz cuadrada (dimensión)
   struct matriz * matriz;
   clock_t t0;                            //instante inicial del proceso (ticks)
   clock_t Dt;                            //incremento de tiempo (ticks) tras la inversión
   printf("\n Inversión de matrices...\n");
   printf("\n Generación de Matriz:");
   printf("\n  Modo Manual (1).\n   Modo Automático(otro).\n");
   scanf("\n %i",&modo);
   printf("\n Dimensión de la matriz:");
   scanf("%i",&dim);           //el usuario elige el modo de inicialización de la matriz de prueba
   matriz=constructorMatriz(dim,dim);
   matriz=asignaValoresMatriz(matriz,modo);
   presentaMatriz(matriz);

   t0=clock();                            //se guarda el instante de tiempo antes de realizar la inversión
   matriz=invierteMatriz(matriz);
   Dt=clock()-t0;                         //se calcula de diferencia entre instantes de tiempo antes y después

   presentaMatriz(matriz);
   destructorMatriz(matriz);
   printf("\n\n\tTiempo invertido en la inversión: %u s",Dt/CLK_TCK);
   printf("\n\tOrden de la matriz invertida: %d",dim);
   printf("\n\tPulse espacio - enter para continuar...");
   while(getchar()!=ESPACIO);             //se espera a recibir un espacio por la entrada stardard
   return 0;
}
```

2.4.6 Producto de Matrices.

El algoritmo que aquí se presenta tiene como objetivo limitar el tiempo que requiere una ley de control *"gain-scheduling"* en ser aplicada. Para ello se ha analizado cada una de las partes de la aplicación de dicha ley para plasmar, en *C++*, un código fuente que emule su funcionamiento. Así, se han distinguido las siguientes partes:

- Lectura de las entradas de las variables a través del bus de datos.
- Cálculo de la señal de control mediante la aplicación de las ganancias.
- Escritura de las actuaciones en el bus de datos.

Como ejemplo de sistema a estudio, se plantea un total de 4 señales de actuación que parten de 6 señales de referencia o de respuesta. Si usamos notación matricial para la descripción de la ley de control por ganancia en el bucle de realimentación del espacio de estados, tenemos:

$$u = -K \times x; \quad [15]$$

En donde:

$$u = \begin{pmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{pmatrix}; K = \begin{pmatrix} k_{11} & k_{12} & \cdots & k_{16} \\ k_{21} & k_{22} & \cdots & k_{26} \\ \vdots & \vdots & \ddots & \vdots \\ k_{41} & k_{42} & \cdots & k_{46} \end{pmatrix}; x = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_6 \end{pmatrix}; \quad [16]$$

Denotando:

- *'u'*: La secuencia de control a aplicar cuya componente i-ésima es u_i, todo i = 1, 2, 3, 4.
- *'K'*: La matriz de realimentación dada por la tabla de ganancias; siendo cada una de sus entradas k_{ij}; todo i = 1,2, 3, 4, todo j = 1, 2, ..., 6; dependiente de las condiciones internas y externas del sistema.
- *'x'*: El vector de salida del sistema cuyas componente i-ésima es x_i, todo i = 1, 2, 3, 4.

Veamos las dos fases que se desarrollaron para lograr el objetivo de este apartado: La medida de tiempo del producto matricial.

2.4.6.1 Algoritmo: “ControlProp.cpp”.

El planteamiento de esta versión calcula el tiempo de cada multiplicación mediante un promedio, es decir, realiza la operación un número entero de veces lo suficientemente grande como para generar un espacio muestral representativo del proceso, de esta forma resulta coherente otorgarle carácter de valor medido a la media de dicho espacio.

El algoritmo aquí diseñado hace uso de un bucle de selección múltiple para obtener cada una de las componentes de la secuencia de control de forma separada. Este bucle está inmerso dentro del que genera el espacio muestral representativo.

Veamos su descripción bajo el lenguaje *C++*.

- Código fuente.

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define ESPACIO 32
#define N 6                         //tamaño de la matriz de realimentación del vector de estados
#define RANGE 0.8
#define OFFSET 0.1
int main(void)
{
   float aux1,aux2,aux3,aux4,aux5,aux6;     //simulan las instrucciones R/W
   float x1,x2,x3,x4,x5,x6;              //vector de estados
   float u1,u2,u3,u4,u5,u6;                                          //vector de control
   float k[N][N];
   clock_t t0;
   int i,j,l;                 //índices
   int nrep;                  //nrepeticiones del proceso

   aux1=aux2=aux3=aux4=aux5=aux6=0.0;//inicialización
   for(i=0;i<N;i++)
            for(j=0;j<N;j++)
                     k[i][j]=(float)rand()/(float)RAND_MAX*RANGE+OFFSET;       //U(offset,offset+range);

   printf("\nNúmero de repeticiones? ");
   scanf("%d",&nrep);

   t0=clock();
   for(l=0;l<nrep;l++)
   {
            x1=aux1;                  //lectura de las variables de estado
            x2=aux2;
            x3=aux3;
            x4=aux4;
            x5=aux5;
            x6=aux6;
           for(i=0;i<N;i++)                     //cálculo de la señal de control
                     switch(i)
            {
                                case 0: u1=-(k[0][0]*x1+k[0][1]*x2+k[0][2]*x3+k[0][3]*x4+k[0][4]*x5+k[0][5]*x6);
                                         break;
                                case 1: u2=-(k[1][0]*x1+k[1][1]*x2+k[1][2]*x3+k[1][3]*x4+k[1][4]*x5+k[1][5]*x6);
                                         break;
                               case 2: u3=-(k[2][0]*x1+k[2][1]*x2+k[2][2]*x3+k[2][3]*x4+k[2][4]*x5+k[2][5]*x6);
                                         break;
                                case 3: u4=-(k[3][0]*x1+k[3][1]*x2+k[3][2]*x3+k[3][3]*x4+k[3][4]*x5+k[3][5]*x6);
                                         break;
                               case 4: u5=-(k[4][0]*x1+k[4][1]*x2+k[4][2]*x3+k[4][3]*x4+k[4][4]*x5+k[4][5]*x6);
```

```
                                        break;
                                case 5: u6=-(k[5][0]*x1+k[5][1]*x2+k[5][2]*x3+k[5][3]*x4+k[5][4]*x5+k[5][5]*x6);
                                        break;
                                default: break;
                }
                 aux1=u1;               //escritura de las variables de control
                 aux2=u2;
                aux3=u3;
                aux4=u4;
                aux5=u5;
                aux6=u6;
        }
        t0=clock()-t0;
        printf("\n\nTiempo invertido en %d repeticiones: %u ms",nrep,t0);
        printf("\nTiempo promedio por repetición: %.3f ns",(float)t0/(float)nrep*1000);
        while(getchar()!=ESPACIO);
        return 0;
}
```

En lugar de usar índices para recorrer la matriz de ganancias *'K'*, mediante el acceso a memoria indexado, se ha preferido hacer uso de constantes numéricas para referirnos a los campos de la matriz; esta opción acelera las operaciones puesto que no hace uso de variables innecesarias disminuyendo el número de accesos a memoria. De igual modo se ha preferido construir tanto el vector de la secuencia de control *'u'* como el de salida *'x'* de forma expandida en múltiples variables, es decir, desde *'u1'* hasta *'u6'* y desde *'x1'* hasta *'x6'*, respectivamente.

La generación de las constantes que forman la matriz de ganancias se realiza fuera del marco que define la función *"clock()"*. Para el caso en el que se aplicara el algoritmo de control proporcional con *"gain-scheduling"* la adaptación de las entradas de la matriz de realimentación debe hacerse en línea con el proceso, pero debido a que las constantes de tiempo del bucle de control (1ms) son mucho menores que los cambios externos e internos del sistema se puede afirmar que la adaptación se llevará a cabo en un número pequeño de iteraciones, con lo que, su consumo temporal no será apreciable frente al total.

Con la intención de contemplar el efecto en la medida del cálculo de las constantes de realimentación se ha planteado una matriz mayor que la que exige el modelo del sistema de trabajo de este proyecto; así pues, se ha pasado de una matriz 4x6 a otra cuadrada de tamaño 6x6.

El tamaño que este código ocupa en la memoria *RAM* de nuestro sistema objetivo es de 26234 bytes, siendo el 10,01% de total. Considerando que el algoritmo de control, aquí expuesto, queda situado dentro de los más sencillos, la cantidad de memoria requerida es apreciable comparándola con el total que se ella se dispone.

2.4.6.2 Algoritmo: “ControlProp2.cpp”.

El algoritmo que se plantea pretende disminuir, aún más, los tiempos requeridos para la realización del cálculo de la secuencia de control; para ello sustituye el bucle interno de selección múltiple de la primera versión por la declaración directa de todas las componentes de la secuencia.

Añadiremos a continuación el código *C++* que cambia frente al de la versión anterior.

- Código fuente.

```
t0=clock();
for(i=0;i<nrep;i++)
{
      x1=aux1;                    //lectura de las variables de estado
      x2=aux2;
      x3=aux3;
      x4=aux4;
      x5=aux5;
      x6=aux6;

     //cálculo de las señales de control
     u1=-(k[0][0]*x1+k[0][1]*x2+k[0][2]*x3+k[0][3]*x4+k[0][4]*x5+k[0][5]*x6);
     u2=-(k[1][0]*x1+k[1][1]*x2+k[1][2]*x3+k[1][3]*x4+k[1][4]*x5+k[1][5]*x6);
     u3=-(k[2][0]*x1+k[2][1]*x2+k[2][2]*x3+k[2][3]*x4+k[2][4]*x5+k[2][5]*x6);
     u4=-(k[3][0]*x1+k[3][1]*x2+k[3][2]*x3+k[3][3]*x4+k[3][4]*x5+k[3][5]*x6);
     u5=-(k[4][0]*x1+k[4][1]*x2+k[4][2]*x3+k[4][3]*x4+k[4][4]*x5+k[4][5]*x6);
     u6=-(k[5][0]*x1+k[5][1]*x2+k[5][2]*x3+k[5][3]*x4+k[5][4]*x5+k[5][5]*x6);

      aux1=u1;                    //escritura de las variables de control
      aux2=u2;
      aux3=u3;
      aux4=u4;
      aux5=u5;
      aux6=u6;
  }
  t0=clock()-t0;
```

Podemos observar la realización compacta de la multiplicación de la matriz de ganancias con el vector de salida como sustitución del bucle de selección múltiple; esto favorece tanto la lectura y compresión del código como la velocidad de cálculo.

La cantidad de memoria que se necesita, para este caso, asciende a los 26182 bytes.