Escuela Superior de Ingenieros
Universidad de Sevilla

PROYECTO FIN DE CARRERA

"Diseño de un Sistema de Control para un Motor de Continua basado en DSP"

Volumen I

Realizado Por:
Juan Carlos del Pino López
Tutor del Proyecto:
Pedro L. Cruz Romero

Quiero dedicar este proyecto a mi familia, en particular a mis padres por la paciencia desarrollada a lo largo de tantos años de carrera. También se lo dedico especialmente a Susana, sin cuyo amor y apoyo nunca habría llegado este momento. Recuerdos a tantos compañeros que quedaron atrás..... y sobretodo por delante, por su ayuda y compañía. Agradecimientos a mi tutor, Pedro L. Cruz Romero, por las facilidades prestadas para la elaboración de este proyecto.

A todos, gracias.

Sevilla, 10 de Mayo de 2004.

Índice general de contenidos – **VOLUMEN I**

VOLUMEN II

Índice de Figuras y Tablas

VOLUMEN I

CAPÍTULO 1: INTRODUCCIÓN A LOS DSP´S

CAPÍTULO 2: KIT DE INICIACIÓN DSK F243

CAPÍTULO 3: SOFTWARE DE DESARROLLO DE APLICACIONES

VOLUMEN II

CAPÍTULO 4: REGISTROS Y PERIFÉRICOS DEL TMS320F243

CAPÍTULO 5: APLICACIÓN DE LOS DSP´S AL CONTROL DE MOTORES

OBJETIVOS DEL PROYECTO

Debido al gran auge que ha sufrido la electrónica en los últimos años, gracias en parte al desarrollo de las Telecomunicaciones, han ido apareciendo en el mercado nuevos modelos de procesadores específicamente diseñados para el tratamiento digital de señales. La gran versatilidad de estos procesadores, junto con su potencia de cálculo, los hacen especialmente beneficiosos para la implementación de un control digital de una aplicación, ya que disponen de numerosos periféricos empleados en aplicaciones de control y que se encuentran integrados en el mismo circuito del procesador. De esta manera, el uso de este tipo de procesadores permite realizar aplicaciones de control digital fácilmente actualizables y que requieren de un menor número de componentes para su realización.

Gracias a las ventajas que introduce el DSP, en la actualidad el control digital de motores ha sufrido un gran avance, como se comprueba diariamente en cualquiera de los equipos informáticos y electrodomésticos de uso cotidiano, al igual que en multitud de aplicaciones industriales. Por ello, el presente proyecto intenta adentrarse en esta materia para edificar los cimientos de futuras investigaciones basadas en estos procesadores, de modo que una persona no familiarizada con los mismos pueda comenzar desde cero a desarrollar aplicaciones basadas en DSP´s. Así, el proyecto desarrollado analiza en los distintos capítulos los principales aspectos que presentan estos procesadores, como son origen, estructura, funcionamiento, lenguajes de programación y herramientas de desarrollo. Cada uno de estos aspectos serán detallados para su fácil comprensión por parte de un inexperto en la materia, completando las explicaciones con ejemplos prácticos para su aplicación. También se introducirán algunos conceptos sobre el control de motores, en especial de corriente continua, como son las distintas técnicas de modulación de señales PWM. Ello permitirá desarrollar el objeto final de este proyecto: el diseño de una placa de control de un motor de continua basado en DSP. Esta placa será diseñada para el control de la velocidad de los motores de continua de 24V disponibles en los laboratorios del departamento de Ingeniería Eléctrica, permitiendo poner en práctica todos los conceptos y conocimientos analizados anteriormente. Esto hace que este documento no solo analice la parte teórica de estos procesadores, sino que permite realizar físicamente una aplicación de control de velocidad que permita continuar con esta línea de investigación en futuros proyectos del departamento. En concreto, este proyecto es una pequeña aportación a uno más ambicioso cuyo objetivo final es realizar un banco de pruebas en los laboratorios del departamento de Ingeniería Eléctrica, el cual sirva para la simulación de la variación de carga solicitada a un vehículo de tracción eléctrica dependiendo de las condiciones del terreno por el que circula. Este proyecto se construirá a partir de los estudios realizados por distintos proyectos como el que nos ocupa, y que tendrá como finalidad el desarrollo de una nueva práctica de laboratorio para los alumnos que realicen esta carrera.

CAPÍTULO 1: INTRODUCCIÓN A LOS DSP´S

1.1. INTRODUCCIÓN:

Aunque muchas veces no son visibles, y a menudo tampoco accesibles para el programador, los procesadores de señal digital (DSP) se hallan en un gran número de aplicaciones. En un principio se usaron de forma académica y para aplicaciones militares, pero cada vez se extiende mas su uso comercial. En los últimos años han evolucionado considerablemente facilitando su uso, gracias también a la aparición de nuevos y avanzados programas de desarrollo, haciendo que se disparen las ventas. Estos circuitos se han incluido, desde su aparición a principios de los ochenta, en sistemas de comunicaciones móviles, controladoras de unidades de disco duro y módems, así como en el proceso de audio y ejerciendo funciones de control. A todo ello hay que añadir su creciente presencia en nuevos segmentos, como son la transmisión de voz por medio de redes IP (Internet Protocol), en conexiones del tipo ADSL (Assymetric Digital Subscriber Line), en TV digital o en soluciones que combinan telefonía e informática (CTI, Computer Telephony Integration). Son todas estas nuevas aplicaciones en telecomunicaciones y control (Tabla 1.1) las que están ampliando el abanico de posibilidades de los DSP´s, provocando una mayor inversión en investigación y dando lugar a mejores dispositivos y software de diseño más sencillos que agilizan los procesos de diseño, desarrollo y puesta en el mercado de una aplicación concreta. Esto ha llevado a un gran incremento de la demanda en estos dispositivos y de su mercado (figura 1.1).

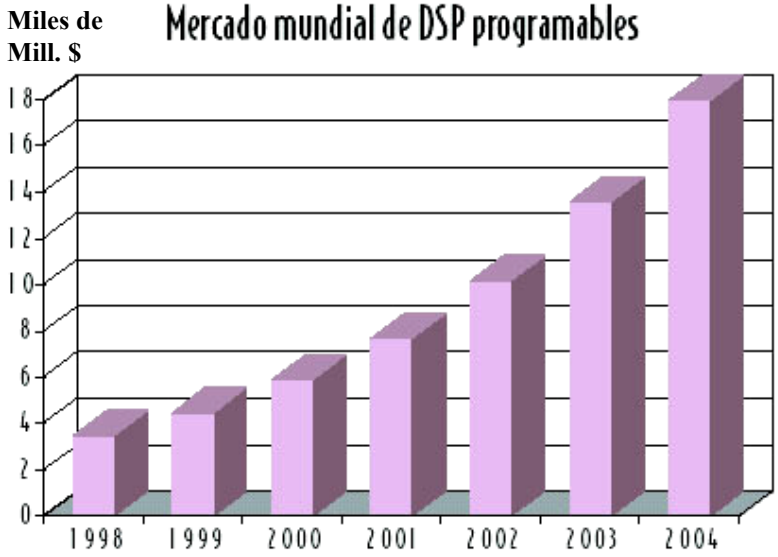


Figura 1.1. Evolución prevista del mercado de DSP.

Telecomunicaiones y campos relacionados	
Telecomunicaciones	**Análisis voz y conversación.**
Transmisión de voz: Teléfonos móviles	Compresión de voz.
(GSM), celulares e inalámbricos.	Codificación de conversación.
Cancelación de ecos.	Reconocimiento de voz.
ADPCM Transcoders.	Enriquecimiento de voz
PBXs digitales.	Conversión de texto en voz y viceversa.
ATM.	Encriptado de voz.
Repartidores de línea.	**Procesamiento de Imágenes.**
Multiplexado de canales.	Animación.
Modems de alta velocidad.	Estaciones de trabajo.
Ecualizadores adaptativos.	Rotación tridimensional.
Fax.	Reconocimiento de patrones.
Interpolación digital del habla.	Proceso homomórfico.
Conmutación de paquetes.	Compresión y transmisión de imágenes.
Transmisión de voz.	**Aplicaciones digitales.**
Adaptadores de terminal.	Ventanas de adquisición.
Controladores HDLC.	Convolución.
Instrumentación.	Correlación.
Analizadores de espectro.	Transformación de Hilbert.
Generadores de función.	Transformada rápida de Fourier.
Osciloscopios digitales: Procesado de	Filtrado digital.
datos.	Generación de formas de onda.
Procesado de datos sísmicos.	
Otros campos	
Aplicaciones de Control.	**Automóvil.**
Robótica.	Cancelación de ruidos.
Regulación en velocidad de motores.	Suspensión activa.
Servocontrol.	ABS.
Impresoras.	Máquinas.
	Navegación.
	Control de entrada de voz.
	Posicionamiento.
	Análisis de vibraciones.
Consumo.	**Militar.**
Sistemas de respuesta.	Navegación.
Autorradios.	Sónar.
Juguetes.	Procesado de imágenes.
Herramientas.	Radar.
TV y música digital.	Guía de misiles.
Sintetizadores musicales.	Seguridad en las comunicaciones.
Medicina.	**Industria.**
Ayuda a deficientes auditivos.	Medida y control.
Monitorización de pacientes.	Robots.
Equipos de ultrasonido.	Control numérico.
Monitor fetal.	Seguridad de acceso.
	Medida de la red de alimentación.

Tabla 1.1. Principales aplicaciones de los DSP.

En muchos casos los DSP resultan inaccesibles, porque se trata de dispositivos que llevan a cabo funciones dedicadas (embedded), es decir, no programables. Pueden clasificarse en dos grandes grupos, programables y no programables, y son estos últimos los que representan la mayor parte de la demanda: alrededor de un 60% del total. El suministro de DSP programables se concentra en cuatro grandes fabricantes (Texas Instruments, Lucent Technologies, Motorola y Analog Devices), como se muestra en la figura 1.2, mientras que es muy superior el número de fabricantes de DSP no programables.

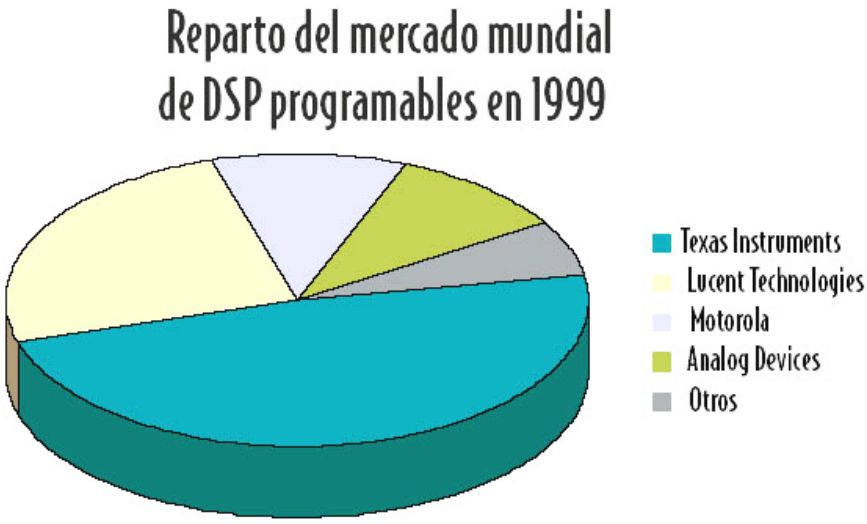


Figura 1.2. Principales fabricantes.

1.2. GENERALIDADES:

Un *DSP* es un microprocesador que posee una CPU de gran potencia y que está preparado para el tratamiento digital de señales en tiempo real y para la realización del mayor número de operaciones aritméticas en el menor tiempo posible. Su función consiste en tomar en tiempo real señales a gran velocidad, como radio, sonido o video, y manipularlas para una gran variedad de usos.

En el mundo real las señales son analógicas, es decir, que son continuas y varían de forma suave. En el mundo digital las señales son series discretas de números binarios lo que permite una gran exactitud en su medida y control a diferencia de los sistemas analógicos. La meta del DSP es aplicar esta capacidad de procesamiento para gestionar y modificar señales de datos. De esta forma, el primer paso en muchos sistemas DSP es la traducción de las señales analógicas reales en una aproximación digital. Este proceso lo llevan a cabo los convertidores analógicos-digitales, los cuales toman medidas de la señal de entrada cada cierto tiempo discretizándola. Esto hace que esta tarea sea llevada a cabo con gran rapidez con el fin de que no lleguen datos falseados al DSP. Una vez digitalizada la señal y filtrada de señales extrañas, esta pasa al DSP en donde será procesada. En algunos casos, después de ser tratada la señal se suele transformar de nuevo en analógica (figura 1.3).

Figura 1.3. Funcionamiento básico del DSP.

Así los DSP pueden filtrar ruidos de una señal, quitar interferencias indeseadas, amplificar ciertas frecuencias y suprimir otras, encriptar información, descomponer ondas complejas en sus componentes, etc. Esto hace que se usen para limpiar ruidos de viejas grabaciones de música, eliminar el eco en líneas de comunicación, mejorar la definición en scanner médicos, proteger la privacidad en conversaciones con teléfonos móviles, borrar el ruido en largas llamadas telefónicas o permitir a los satélites detectar pequeños objetos en la superficie terrestre. En automóviles se usan DSP´s para crear sonido digital y ser los responsables de los sistemas de suspensión activa que se ajustan automáticamente a las condiciones de la carretera. En los teléfonos móviles, ayudan a mantener una conversación fluida sin interrupciones ni esperas y sin interferencias entre la multitud de ondas que originan los millones de móviles en el mundo. En ordenadores permiten nuevas formas de comunicación al transmitir video y sonido en tiempo real. También existen aplicaciones en las que no es necesaria la conversión a analógica, con lo que la señal puede ser directamente procesada por el DSP, como por ejemplo el tratamiento de imágenes por ordenador, simulaciones, modelos meteorológicos, etc.

El proceso del DSP es muy numérico; son esencialmente rápidos operadores numéricos. A medida que van llegando datos de la señal de entrada, estos deben ser multiplicados, sumados y transformados según complejas formulas. Lo que caracteriza a los DSP es su velocidad. Estos deben trabajar en tiempo real, capturando y procesando gran cantidad de información al tiempo que ésta se está produciendo, con lo que debe haber una perfecta coordinación entre este y los convertidores analógico-digitales del sistema, ya que si uno de ellos falla el resultado será erróneo o impreciso. Por todo esto son los principales candidatos para aplicaciones que requieran el manejo y procesamiento de grandes cantidades de datos numéricos en tiempo real.

Estos microprocesadores se caracterizan por tener arquitecturas especiales, orientadas a la realización hardware de los cálculos que otro tipo de microprocesadores implementan vía software, mediante la ejecución secuencial de varias instrucciones. Por ello, el hardware de la CPU de este tipo de sistemas digitales es generalmente mucho más complejo que el de otros microprocesadores o microcontroladores. El área de silicio es mucho mayor y, por tanto, el coste del producto aumenta respecto a los microprocesadores y microcontroladores. La principal diferencia de los DSP´s y otros procesadores modernos, es que los primeros se diseñan para ser escalables, es decir, se diseñan para poder operar en paralelo con otros dispositivos similares. Para ello, se le añaden periféricos de control y bloqueo del programa (como líneas de entrada-salida que pueden bloquear la ejecución de ciertas instrucciones si se encuentran a un

determinado valor) y periféricos de entrada-salida de alta velocidad (como puertos serie síncronos) que permiten la conexión sencilla de varios DSP´s para aplicaciones con multiprocesadores. Así, el grado de paralelismo es directamente proporcional al número de operaciones que el DSP será capaz de realizar en un ciclo de reloj. Además de éstos periféricos, en los DSP´s se incluyen otros especializados en la toma de medidas analógicas (convertidores analógico-digitales), en la captura de señales procedentes de sensores, en la generación de señales de control, temporizadores, etc, además de incluir distintos tipos de memoria, y todo ello incluido en el mismo encapsulado del chip. Ésto los diferencian aún más de los microcontroladores que no suelen incluir dichos periféricos los cuales son necesarios adquirirlos a parte con el consiguiente aumento del gasto. Otra ventaja que tienen los DSP´s al igual que los microcontroladores es la flexibilidad que proporcionan al sistema al que pertenecen, ya que pueden adaptarse fácilmente a cualquier modificación o mejora que requiriera el mismo sólo con reprogramarlos para su nueva tarea.

Las principales alternativas que aparecen al uso de DSP´s, son:

• **ASIC´s de función fija ó FPGA´s**.
A diferencia de cualquier microprocesador, su labor no se realiza mediante la ejecución secuencial de instrucciones, sino que se programa en la circuitería que lleva dentro. Cuando aparecieron estos dispositivos en el mercado, sus principales inconvenientes eran la falta de versatilidad (no valían para otra cosa distinta de aquella para la que fueron diseñados) y el coste de desarrollo que tenían asociado (el diseño de una aplicación basada en FPGA´s ó ASIC´s era mucho más caro en tiempo que el desarrollo de un programa en C o ensamblador para un DSP) y su utilidad principal estaba en el empleo como coprocesadores de los DSP´s. En la actualidad, y gracias al desarrollo de las herramientas de programación y de las tecnologías de fabricación y técnicas de integración asociada a estos dispositivos, los inconvenientes relacionados con la falta de versatilidad se han solucionado si bien el coste de desarrollo sigue siendo superior al ligado a los DSP´s.

• **Ordenador personal o estación de trabajo.**
Está alcanzando popularidad por el, cada vez, menor coste de los sistemas PC multimedia. El sistema operativo se ejecuta en paralelo con la aplicación de procesado de la señal, lo que resta velocidad de procesado a la aplicación e impide su empleo en tiempo real, limitando su uso al procesado de la señal fuera de línea. Son más caros que las tarjetas basadas en DSP´s y aunque la potencia de los microprocesadores es elevada, no pueden competir con el DSP en tiempo de ejecución de las operaciones aritméticas. El manejo de datos que realizan equivale en rapidez con el de los DSP´s pero las operaciones aritméticas suelen realizarse en tiempos más elevados.

• **Microcontroladores**.
Se diseñan principalmente para el control de procesos en tiempo real. Los microcontroladores se clasifican en función del tamaño del bus de datos. Los microcontroladores de 8 y 16 bits de bus de datos, no alcanzan en ningún caso las velocidades de los DSP´s. Los de 32 bits de bus de dato son tan rápidos como los DSP´s y pueden emplearse en aplicaciones con pequeñas constantes de tiempo, del orden de los microsegundos, o frecuencias de trabajo elevadas. Las estructuras internas de estos microcontroladores se asemejan a la de los DSP´s

pero no están preparados para el trabajo en paralelo mediante arquitecturas multiprocesadores, sino que están pensados, más bien, para el funcionamiento en solitario y con el menor número de dispositivos externos (a ser posibles, el producto final dispondrá de un único integrado, el propio microcontrolador).

En la tabla 1.2 se muestran los tiempos empleados por varios DSP´s y microprocesadores en realizar algunas operaciones aritméticas básicas.

Procesador	*Bus datos*	*Reloj*	±	×	÷	√
		Mhz	ns	ns	µs	µs
DSPs						
TMS32010	16 bits	5	200	200	<12.8	<59
TMS32014	16 bits	6.25	160	160	<10.3	<47
TMS32020	16 bits	5	200	200	<3.4	<59
TMS32025	16 bits	12.5	80	80	<1.32	
TMS320C30	32 bits	33	60	60	<2	<2
TMS320C40	32 bits	40	50	50	<1	<1
MC56000	32 bits	10	100	100	<3.7	
MC96002	32 bits	16.5	60	60		
Microcontroladores						
M68HC11	8 bits	8	1000	5000	20.5	
MCS51	8 bits	12	1000	4000	8	
MCS96	16 bits	16	800	1750	2.5	
MC68020	32 bits	16	240	1620	3.36	
MC68030	32 bits	20	200	1400	2.8	
IAPX80286	32 bits	8	375	3000	3.12	>2
NOVIX4000	32 bits	8	125	3130	5.62	10.62

Tabla 1.2. Comparación de tiempos de proceso.

1.3. CARACTERÍSTICAS PRINCIPALES:

La selección de un determinado DSP se basa en numerosos parámetros que se pueden resumir en estos aspectos fundamentales:

- potencia de cálculo,
- capacidad de almacenamiento,
- conectividad,
- consumo,
- disponibilidad de bibliotecas de soporte por parte de terceros y
- herramientas de desarrollo.

Sin embargo, toda consideración queda supeditada a la aplicación a la cual se vaya a destinar el dispositivo.

1.3.1. Potencia de cálculo:

Los fabricantes ofrecen una cifra que hace referencia a la potencia máxima, indicada generalmente en millones de instrucciones (**MIPS**), millones de operaciones (**MOPS**) o millones de operaciones en coma flotante por segundo (**MFLOPS**). Este dato debe ser matizado ya que en función del tipo de aplicación el resultado será bien distinto, así como por la importancia de conocer la potencia de cálculo de forma sostenida. Entre las especificaciones debe primar la indicación de si los cálculos aritméticos son con **coma fija** (se reserva un número fijo de bits para representar los decimales) **o flotante** (usa un número variable de bits para los decimales), ya que cada modelo ha sido diseñado para efectuar las operaciones de uno u otro modo. Este dato influye directamente sobre otra característica principal de los DSP, que es su **rango dinámico**: el rango numérico que es capaz de soportar en sus operaciones. A mayor rango, mayor precisión tendrá en su tarea. En él tiene que tenerse en cuenta el soportar los resultados de operaciones como la multiplicación que generan grandes números, de forma que no se dé overflow. Este rango viene dado por el numero de bits que maneja el DSP. Un DSP de 32 bits tiene mayor rango que uno de 24 bits y éste más que uno de 16 bits. Por ejemplo, si trabajamos con un DSP de 16 bits se tiene un rango dinámico que varía entre $-2^{15}=-32768$ y $2^{15}-1=32767$. El trabajar en coma flotante da lugar a un mayor rango que en coma fija, ya que puede tomarse mayor número de bits para representar la parte entera si es necesario. Además, los DSP de coma flotante son más rápidos que los de coma fija, aunque algo más caros. Esto diferencia los distintos tipos de DSP según las tareas que puedan realizar. De esta forma en telefonía, al trabajarse con una rango relativamente bajo de frecuencias de sonido, suelen utilizarse DSP que trabajan con 16 bits en coma fija, pero en alta fidelidad, al trabajar con mayor rango de frecuencias, se opta por uno de 24 bits. Para trabajos de gráficos en 3-D que requieren un rango mayor y más dinámico se usan de 32 bits en coma flotante.

1.3.2. Capacidad de almacenamiento:

Con ello se hace referencia fundamentalmente al número de **registros** y al bloque de **memoria**. Como es lógico, cuanto mayor sea el número de registros, más rápidos serán los cálculos ya que se evitan los accesos a memoria. Sin embargo, estos accesos son necesarios en la gran mayoría de los casos, por lo cual resulta muy importante que el tiempo de lectura/escritura sea mínimo. Es útil asimismo disponer de la opción de **memoria externa**, ya que a menudo no es suficiente con la memoria integrada en el propio procesador. El ancho de banda propio del DSP se debe precisar, por tanto, en función de si se trata del envío de datos entre registro y memoria interna, o bien si se trata de la comunicación desde / hacia la memoria externa. Por otra parte, destacar la importancia de contar con la posibilidad de acceder directamente a memoria (**DMA**), lo cual agiliza el funcionamiento. También resulta interesante disponer de un bloque de **memoria cache**. En función de todo ello, la estructura de memoria (es decir, conocer todos los tipos de memoria disponibles, su capacidad y el ancho) deberá ser la adecuada para la aplicación. La estructura de memoria de algunos DSP avanzados es realmente compleja, e incluye no sólo sendos bloques de memoria interna y externa, además de cache (especializada a su vez para el almacenamiento de partes del programa o bien de datos), sino también un gran número de registros organizados por ficheros y de uno o varios controladores DMA. Algunos de estos procesadores comparten el espacio total de direccionamiento en memoria entre programa y datos.

La capacidad de almacenamiento se indica a menudo en **palabras** (words), ya que el ancho varía en función de cada modelo. Así, a modo de ejemplo, en la gama ADSP-2106x de Analog Devices hay 16 Mpalabras para programa (ancho: 48 bit) y 4 Gpalabras para datos (ancho: 40 bit). Este ancho es variable en procesadores como el C55x de Texas Instruments, en cuyo caso puede ser de 8, 16, 24, 32, 40 u 80 bit.

1.3.3. Conectividad:

Es esencial tanto desde un punto de vista interno (tal como hemos visto para las comunicaciones entre registros y memoria) como en lo relativo a puertos de comunicaciones con el exterior, ya sea con otro DSP o con un PC. La disponibilidad de **puertos serie** es habitual en numerosos procesadores, así como de un puerto de interfaz al host (PC). En algunos modelos también existe la posibilidad de comunicación a través de una interfaz **JTAG** externa o mediante el **puerto paralelo** si ésta está integrada en la placa. Esta interfaz es muy empleada para la programación, depuración y emulación de programas. La **velocidad de transmisión** que sean capaces de alcanzar los puertos de comunicación es un parámetro fundamental ya que es uno de los principales requisitos de todo DSP, independientemente de la aplicación. Por ejemplo, el TigerSharc de Analog Devices dispone de cuatro puertos capaces de llegar hasta 600 MB/s; a ellos hay que añadir la posibilidad de conectar un puerto externo con otros 600 MB/s.

1.3.4. Consumo:

Es uno de los factores a tener muy en cuenta a la hora de decidirse por un DSP u otro, debido al uso cada vez más extendido de éstos en aplicaciones portátiles como la telefonía celular. Por ello, los fabricantes los diseñan para tensiones bajas de trabajo (3.3 V – 3 V) que incorporan prestaciones para gestión de energía. De este modo se puede inhibir el reloj del DSP a todas o algunas partes del mismo, realizar determinadas tareas a velocidad inferior o desactivar algún periférico si no se prevé su aplicación, disminuyendo así el consumo del dispositivo.

1.3.5. Herramientas de desarrollo:

No sólo son relevantes, sino que determinan poderosamente la validez del diseño. El DSP que finalmente se elija deberá disponer de un amplio conjunto de herramientas de desarrollo. Algunos requerimientos básicos son:

- Documentación de diseño detallada.
- Herramientas de desarrollo de código en ensamblador y/o en lenguaje de alto nivel.
- Herramientas para el test de la funcionalidad del diseño.
- Notas de aplicación u otro tipo de ayuda al diseño.
- Disponibilidad de bibliotecas de soporte por parte de terceros.

El objetivo será seleccionar el DSP que permita terminar el proyecto en el tiempo previsto y que la solución alcanzada sea la que presente la mejor relación coste-eficiencia. En aplicaciones de gran volumen de producción, esto probablemente signifique que el DSP escogido será el más barato que pueda realizar la aplicación. Para aplicaciones con un volumen bajo-medio existirá el compromiso entre el coste de las herramientas de desarrollo y el coste y eficiencia del DSP. En cambio, para aplicaciones

con un volumen bajo de producción tendrá más sentido utilizar un DSP que facilite el diseño o que tenga las herramientas de desarrollo más baratas.

Históricamente, las herramientas de desarrollo para estos procesadores han sido:

- generadores de código (compiladores, ensambladores y encadenadores),
- depuradores (debuggers) y simuladores,
- Programas de análisis (profilers).

Pero, en muchos casos, eran herramientas diferentes, sin posibilidad de comunicaciones de datos entre estas aplicaciones y teniendo que realizar continuos cambios entre ellas. La depuración en tiempo real también ha sido una tarea compleja, ya que para poder controlar una aplicación en condiciones de tiempo real es necesario realizar un seguimiento de ésta sin detener el procesador. Sin embargo, los típicos sistemas de depuración sólo permiten realizarlo de una forma estática, es decir, mediante un análisis del código paso a paso con puntos de ruptura (breakpoints), que detienen al procesador. Estos depuradores no son capaces de dar información de lo que ha sucedido en un tramo de la aplicación que ya se ha ejecutado, excepto alterando la ejecución normal del programa y, por lo tanto, modifican los resultados. Para poder realizar estas tareas ha sido necesario utilizar costosos analizadores lógicos. Otro gran problema ha surgido cuando se intenta integrar diferentes partes del código de una aplicación en tiempo real. Esto es debido a las diferentes interacciones entre estas secciones del código.

Todo esto lleva a que, si los diseñadores no disponen de herramientas que les puedan indicar cuándo una aplicación cumple los requisitos de tiempo real, pueden ir apareciendo muchos problemas muy difíciles de analizar y resolver, con el consiguiente retraso en la finalización del proyecto. La popularidad de los DSP continúa creciendo, y un reto es tener unas herramientas que los diseñadores quieran usar. Los entornos de desarrollo para este tipo de procesadores deben dar las mismas capacidades que los ya existentes para los microprocesadores. Además, el conjunto de herramientas no sólo debe ir acompañado de generadores de código y programas de depuración integrados, sino que también deben disponer de interfaces gráficas que permitan analizar los datos de una forma más intuitiva, deben tener la posibilidad de seguir la ejecución del programa sin parar el procesador y realizar un análisis de los problemas que podrían surgir al integrar ciertas aplicaciones que requieran cumplir algunas restricciones temporales para poder ser ejecutadas en tiempo real. Los diseñadores deben tener la posibilidad de añadir nuevas prestaciones a sus productos para estar por delante de sus competidores. Esta integración debe hacerse de una manera fácil y para ello se debería disponer de una arquitectura abierta.

Un ejemplo de todo esto lo da Texas Instruments, que ha creado un entorno completo de desarrollo de software con la introducción de la tecnología de software en tiempo real **eXpressDSP**. Esta tecnología de software para DSP ofrece:

- un entorno de desarrollo integrado (**Code Composer Studio**), en el que se incluyen generadores de código, simuladores, emuladores y depuradores (figura 1.4),
- un software para realizar aplicaciones de tiempo real (**DSP/BIOS** para series C5000 y C6000),
- un estándar para la interoperatividad de las aplicaciones (DSP Algorithm Standard) y
- la posibilidad de tener un software reutilizable modularmente.

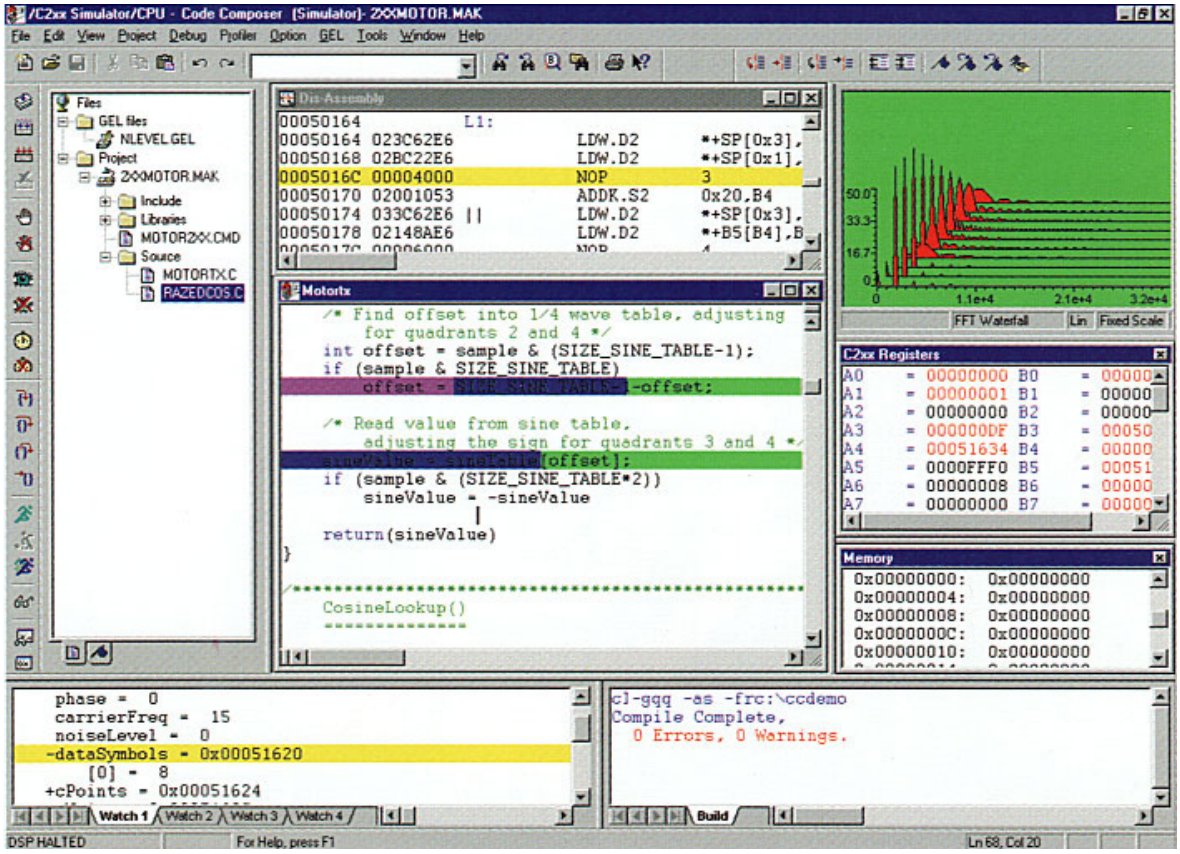


Figura 1.4. Ejemplo de Code Composer Studio.

Otro ejemplo es el programa de simulación y el modelado de sistemas dinámicos complejos, **VisSim** de Visual Solutions (figura 1.5). Permite la modelización dinámica y la simulación de sistemas dinámicos complejos. Todo ello trabajando en un entorno gráfico agradable y con una programación sencilla mediante bloques. El motor de simulación se ha optimizado para proporcionar soluciones rápidas y precisas para sistemas lineales, no lineales, continuos o discretos. Además, dispone de un módulo en tiempo real que le permite conectarse a procesos industriales para su control o validación. VisSim combina una interfaz de diagrama de bloques tipo "arrastrar y soltar" con un potente motor de simulación. La interfaz visual ofrece un método simple para construir, modificar y mantener modelos de sistemas complejos. En él se incluyen módulos enfocados al diseño (**VisSim/Motion**) e implementación (**VisSim - Embedded Controls Developer for TI C2000**) de sistemas de control de movimiento sobre DSP´s de Texas Instruments, los cuales pueden intercambiar información con el Code Composer Studio y Matlab. De esta forma se pueden desarrollar los algoritmos deseados sin necesidad de tener grandes conocimientos de programación ni de la estructura interna de los DSP´s. Basta sólo con conocer la finalidad y el funcionamiento de los distintos bloques y periféricos que componen el DSP.

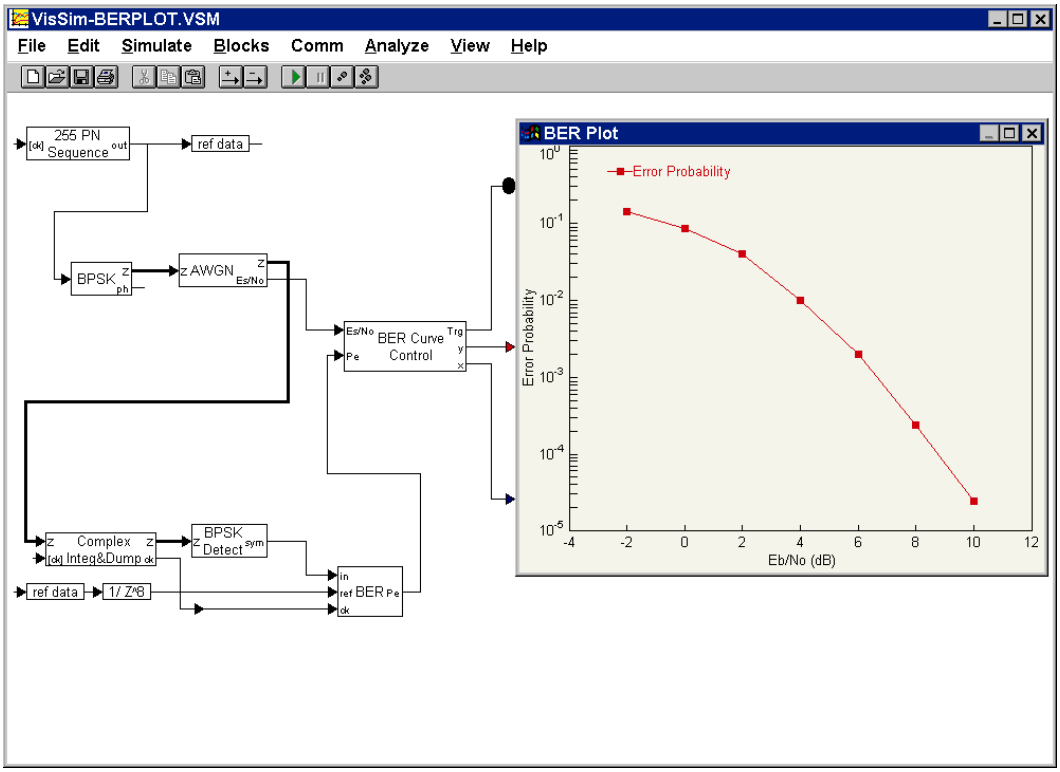


Figura 1.5. Programa de diseño y simulación VisSim.

1.4. MODELOS DE DSP´S:

A la hora de elegir un DSP, en primer lugar hay que tener claro para qué aplicación se va a utilizar y las prestaciones que necesitamos de él. En el mercado hay varios fabricantes de DSP´s, pero en general estructuran los modelos basándose en un mismo criterio:

I. Modelos optimizados para altas prestaciones y bajo consumo,
II. Modelos específicamente diseñados para dispositivos portátiles por su eficiente y bajo consumo,
III. Modelos destinados al control digital en industria, de reducido coste y gran capacidad computacional,
IV. Modelos para otras aplicaciones.

En el caso del fabricante Texas Instruments, esas categorías corresponden con las series: C6000, C5000, C2000 y C3000 (figura 1.6).

De igual forma, otros fabricantes también estructuran sus modelos del mismo modo, como por ejemplo Motorola (figura 1.7). De esta manera siempre se encuentra en el mercado el dispositivo adecuado para cada aplicación. En la actualidad, con el gran auge en las telecomunicaciones, estos fabricantes están haciendo grandes inversiones en el desarrollo de sus productos, debido a su gran demanda para dispositivos de comunicación y portátiles.

Figura 1.6. Series de DSP´s de Texas Instruments.

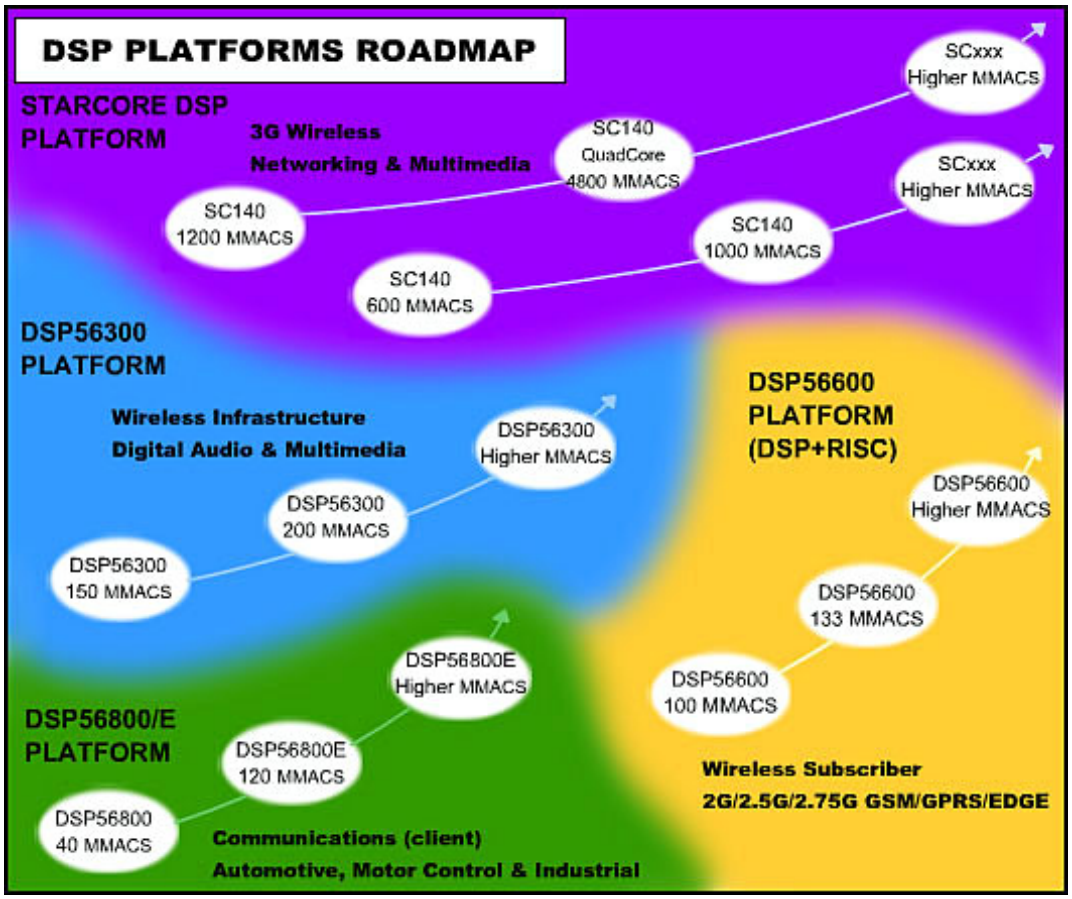


Figura 1.7. Modelos de DSP´s de Motorola.

En relación con el control de motores todos los fabricantes tienen modelos en el mercado con similares características y aplicaciones (figura 1.8). Éstos están optimizados para procesar en tiempo real una gran cantidad de información proveniente del sistema a controlar para la estimación de distintos parámetros, tomando decisiones y actuando de forma casi instantánea sobre el sistema. Para ello están dotados de entradas y salidas por las que reciben datos del sistema y con las que actúan sobre el mismo, y todo ello implantado sobre una misma base, reduciéndose de forma significativa el número de componentes en dicho proceso con el ahorro en gasto y espacio que ello supone.

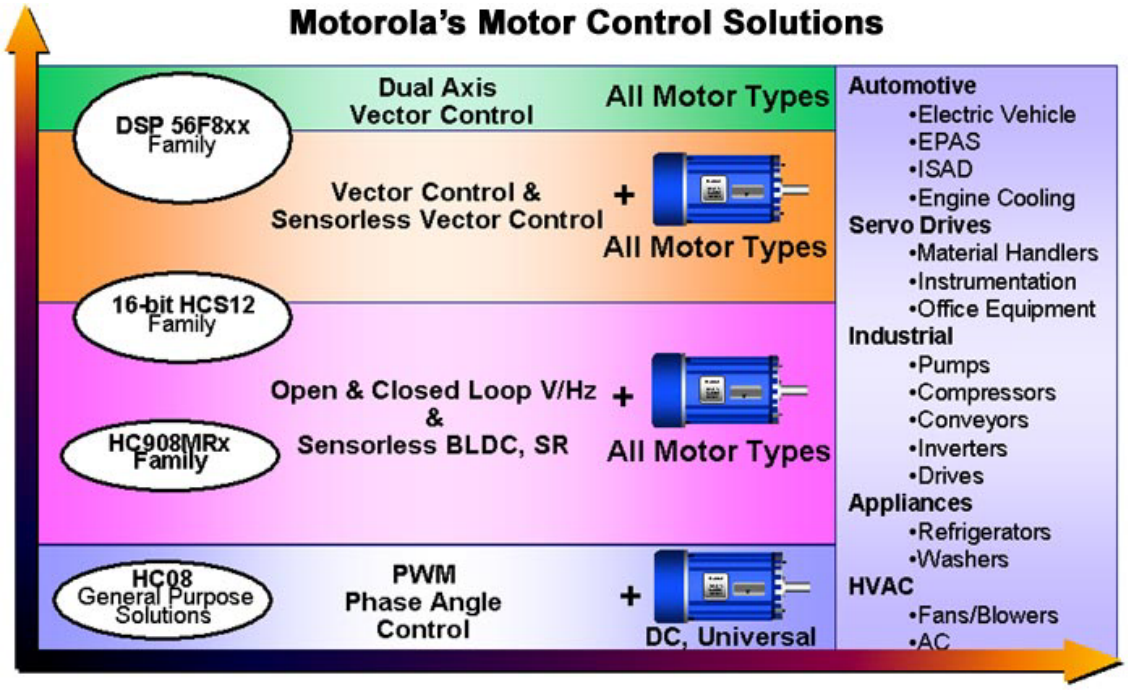


Figura 1.8. DSP´s de Motorola destinados al control de motores.

El principal fabricante de DSP´s en el mercado, Texas Instruments, dispone de una gran variedad de modelos destinados al control de motores en su serie TMS320C2000 (figura 1.9). Dicha serie se divide a su vez en dos grandes grupos según el número de bits con el que operan (tabla 1.3): la serie C24x de 16 bits y la serie C28x de 32 bits, ambas en coma fija.

DSP Generation	DSP Type	Features	Price*
C24x	16-bit data Fixed-Point	SCI, SPI, CAN, A/D, event manager, watchdog timers, on-chip Flash memory, 20-40 MIPS	$215
C28x	32-bit data Fixed-Point	SCI, SPI, CAN, 12-bit A/D, McBSP, watchdog timers, on-chip Flash memory, up to 400 MIPS	$1863

Tabla 1.3. DSP´s de Texas Instruments dedicados al control.

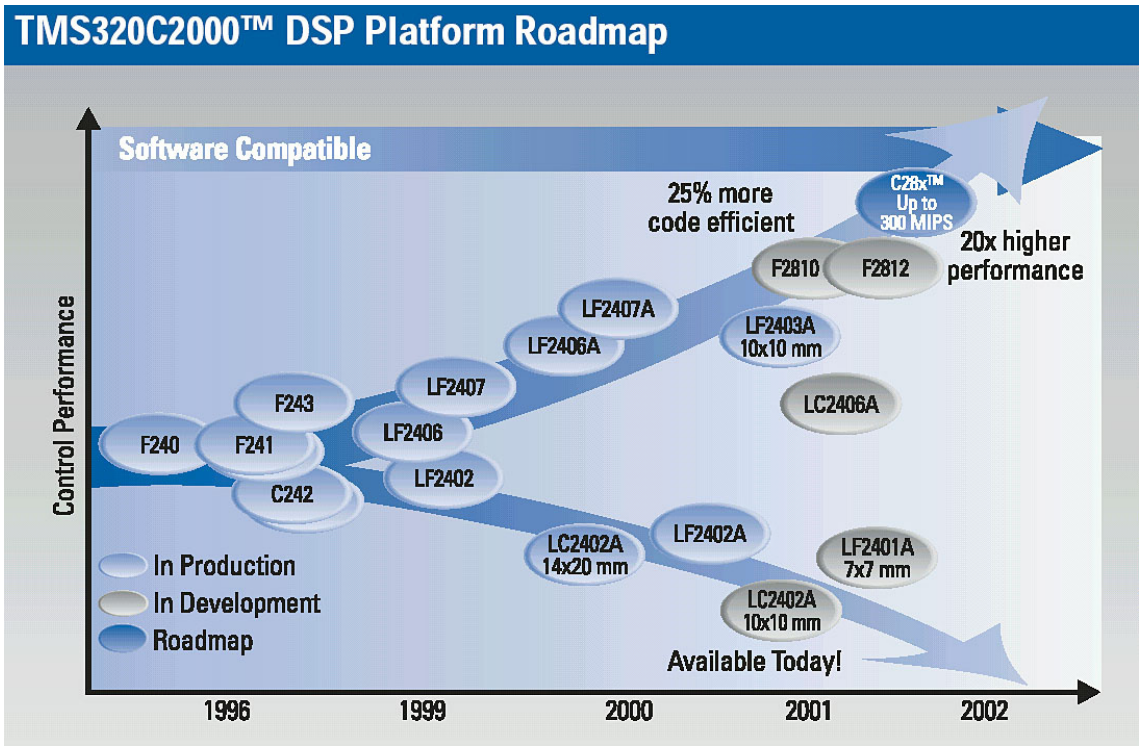


Figura 1.9. Serie C2000 de Texas Instruments.

En el caso que nos ocupa vamos a trabajar con la serie C24x, de la que se presentan las características de algunos modelos en la tabla 1.4 y el rango de aplicaciones en que se emplean (figura 1.10).

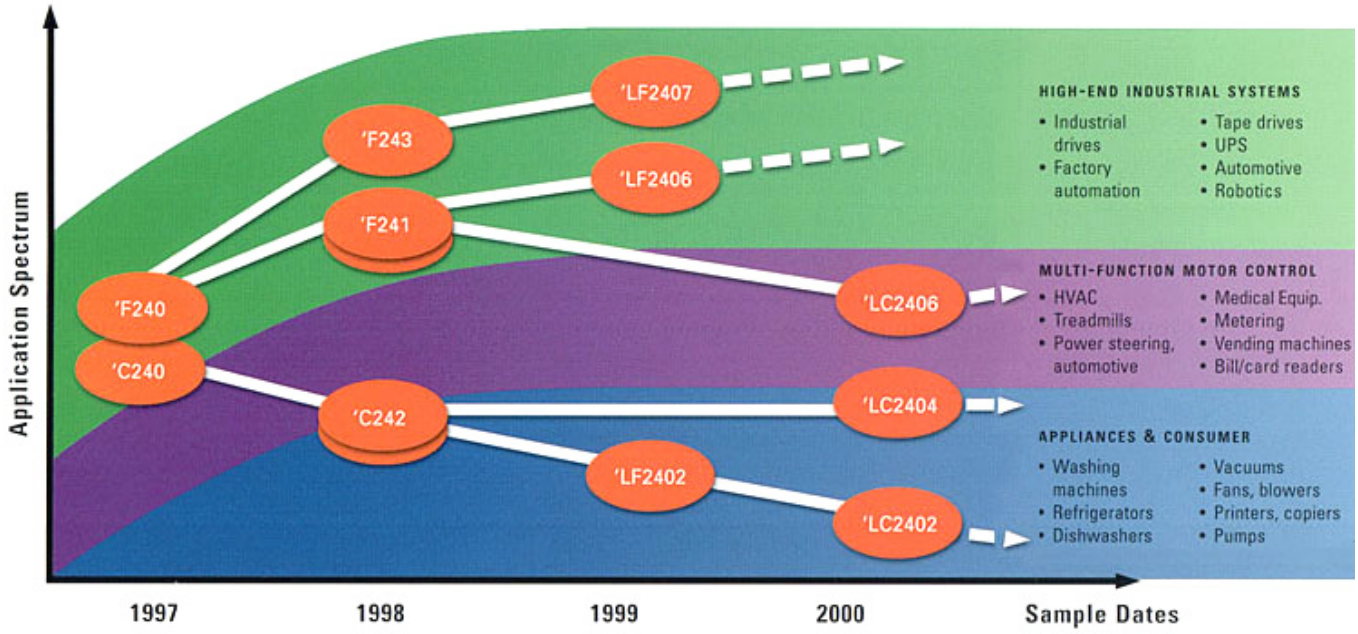


Figura 1.10. Aplicaciones de diversos modelos de DSP´s TMS320C24x.

TMS320C24x™ DSP Generation Product Specification Guide

Device	RAM (16-bit words)	ROM (16-bit words)	Flash (16-bit words)	Boot ROM (words)	EMIF	General-Purpose Timers	Watchdog Timer	PWM Channels	SPI	SCI	CAN	A/D Channels** Conversion Time (µs)	I/O Pins	Voltage (V)	MIPS	Packaging	$U.S./1KU+	$U.S./10KU+
TMS320LF2407A	2.5K	–	32K	256	Yes	4	Y	16	Y	Y	Y	16 ch 0.375	41	3.3	40	144 LQFP	10.56	9.53
TMS320LF2406A	2.5K	–	32K	256	–	4	Y	16	Y	Y	Y	16 ch 0.375	41	3.3	40	100 LQFP	11.03	9.96
TMS320LF2403A	1K	–	16K	256	–	2	Y	8	Y	Y	Y	8 ch 0.375	21	3.3	40	64 TQFP	9.38	8.40
TMS320LF2402A	1K	–	8K	256	–	2	Y	8	–	Y	–	8 ch 0.375	21	3.3	40	64 PQFP	8.21	7.41
TMS320LF2401A	1K	–	8K	256	–	2	Y	7	–	Y	–	5 ch 0.5	13	3.3	40	32 LQFP	7.24	6.54
TMS320LC2406A	2.5K	32K	–	–	–	4	Y	16	Y	Y	Y	16 ch 0.375	41	3.3	40	100 LQFP	–	5.70
TMS320LC2404A	1.5K	16K	–	–	–	4	Y	16	Y	Y	–	16 ch 0.375	41	3.3	40	100 LQFP	–	5.80
TMS320LC2402A	544	6K	–	–	–	2	Y	8	–	Y	–	8 ch 0.425	21	3.3	40	64 PQFP 64 TQFP	–	2.95
TMS320LC2401A	1K	8K	–	–	–	2	Y	7	–	Y	–	5 ch 500 ns	32	3.3	40	32 LQFP	–	3.83
TMS320F243	544	–	8K	–	Yes	2	Y	8	Y	Y	Y	8 ch 0.9	26	5	20	144 LQFP	13.99	12.62
TMS320F241	544	–	8K	–	–	2	Y	8	Y	Y	Y	8 ch 0.9	26	5	20	64 PQFP	12.37	11.16
TMS320C242*	544	4K	–	–	–	2	Y	8	–	Y	–	8 ch 0.9	26	5	20	64 PQFP	–	3.69
TMS320F240	544	–	16K	–	Yes	3	Y	12	Y	Y	–	16 ch 6.1	28	5	20	132 PQFP	16.21	14.62

Tabla 1.4. Modelos de la serie TMS320C24x de Texas Instruments.

1.5. PRODUCTOS Y HERRAMIENTAS DE DESARROLLO

A la hora de diseñar una nueva aplicación basada en el uso de DSP´s, los fabricantes aportan a los diseñadores una amplia gama de productos cuya función consiste en agilizar los procesos de diseño, desarrollo y puesta en el mercado del producto. Además ofrecen una serie de herramientas software que permiten la depuración, simulación, emulación y actualización de los programas desarrollados para DSP´s, facilitando la posibilidad de ampliar y mejorar las aplicaciones desarrolladas sin necesidad de partir desde cero a la hora de actualizar el sistema. Todo esto esta permitiendo una rápida evolución en lo que a aplicaciones con DSP´s se refiere.

1.5.1. Tarjetas de desarrollo:

Para empezar a desarrollar una nueva aplicación, el primer paso consiste en elegir el tipo de procesador que se requiera para la misma. Esta elección se verá agilizada por la clasificación que hacen los fabricantes de sus productos: para control digital (C2000), alto rendimiento (C6000) o bajo consumo (C5000). Una vez elegido el tipo de procesador, los fabricantes (como por ejemplo Texas Instruments, Analog Devices, etc.) ponen a disposición de los diseñadores una serie de kits basados en DSP´s que les permiten evaluar las prestaciones del procesador elegido para comprobar si son adecuados para la aplicación deseada y desarrollarla. Los más típicos son:

- **DSP Starter Kit (DSK)**:
  Para ayudar a la decisión del tipo de procesador a elegir y su evaluación, se disponen de estos kits de iniciación con los cuales se puede empezar a conocer la arquitectura, periféricos e instrucciones de los DSP´s. Son de fácil uso y bajo coste y son suficientemente potentes para el tratamiento digital de señales en tiempo real. Además disponen de conectores de expansión con los que se pueden conectar dispositivos y circuitos externos para el desarrollo de aplicaciones. Para su programación y depuración de programas llevan incluido un puerto serie estándar RS-232 de comunicación con un PC además de una interfaz JTAG. En el kit también se incluyen un ensamblador, linkador, depurador y simulador para el desarrollo de programas.
- **Módulos de evaluación (EVM)**:
  Para empezar a desarrollar una aplicación se dispone de una serie de kits más avanzados para facilitar el proceso diseño del dispositivo y el software necesario. Estas tarjetas basadas en DSP van dotadas de mejores prestaciones que los kits de iniciación, como son: más memoria para programas, múltiples conectores de expansión, periféricos más avanzados, interfaz de comunicación más rápida (emulador JTAG), etc. Además van acompañados de un paquete de software de desarrollo más avanzado en el que se incluyen todo lo necesario para la creación de código de programa (ensamblador, linkador, etc.) y su depuración en tiempo real (debugger, emulador, etc.), como por ejemplo el Code Composer Studio.
- **Motion Starter Kit (MCK/MSK)**:
  Existen también módulos específicos de evaluación para el control de motores, los cuales incluyen ejemplos funcionales de algoritmos de control de motores preparados para su uso inmediato. Destacar por su especial interés las herramientas hardware y software desarrolladas por

Technosoft para la resolución de aplicaciones de control digital de motores. Se incluyen tarjetas de evaluación con el procesador TMS320x24x, módulos de potencia para la alimentación de diferentes tipos de motores, y abundante software de desarrollo. Este software incorpora, entre otras cosas, algoritmos para el control vectorial de motores de inducción, sin escobillas, de reluctancia variable, etc.

1.5.2. Software de desarrollo:

Las dificultades inherentes al desarrollo de aplicaciones en tiempo real con procesadores DSP, obliga al empleo de herramientas específicas no siempre fáciles de utilizar. La complejidad de esas aplicaciones, que habitualmente requieren tiempos de desarrollo importantes, hace aconsejable el empleo de lenguajes de programación de muy alto nivel (p. e. MATLAB/SIMULINK)) que permitan un rápido diseño y validación de los algoritmos empleados. Una vez superada esta primera fase del desarrollo puede pasarse a la generación de código para el procesador específico, empleando lenguajes de alto nivel (generalmente C) y ensambladores. La ejecución y depuración de este código permitirá validar temporalmente la solución.

En cualquier caso, la alternativa que da resultados más rápidos es la correspondiente al empleo de paquetes de software para el desarrollo específico del tipo de aplicaciones que nos ocupa. Los paquetes más avanzados permiten, sin la necesidad de programar código para el DSP, desarrollar aplicaciones empleando un lenguaje gráfico de bloques y facilitando la puesta a punto del sistema al posibilitar la obtención gráfica de los valores de las variables seleccionadas, lo que reduce considerablemente el tiempo de puesta en el mercado del producto, como por ejemplo el **VisSim - Embedded Controls Developer for TI C2000** (Figura 1.5).

Uno de los paquetes de software más utilizados en el diseño y programación de aplicaciones basadas en DSP´s es el **Code Composer Studio** (Figura 1.4). Es un entorno integrado para implementar y depurar aplicaciones de tiempo real basadas en los DSP's. Dispone de un simulador, que junto con el controlador correspondiente permite realizar simulaciones de aplicaciones que en un futuro se pretende que corran sobre la plataforma a la que pertenece el controlador.

Las características más importantes de este entorno de desarrollo son:

- Permite trabajar con programas escritos tanto en ensamblador como en lenguaje C, lo cual hace bastante flexible el procedimiento de programación, limitando a los procedimientos de más bajo nivel el ensamblador, acelerando así el desarrollo del software.
- Permite de modo muy sencillo la compilación, ensamblado y linkado de todos los elementos necesarios para la implementación de una aplicación completa.
- Una vez realizado el procedimiento se puede cargar el programa y simularlo bajo el controlador adecuado. Además el proceso de depuración se puede hacer tanto sobre el código en ensamblador como sobre el código en C, de modo que la depuración se simplifica.
- Durante la depuración de programas se pueden visualizar y modificar los contenidos de zonas de memoria, registros de la CPU.

- Permite realizar representaciones gráfica de muy diversa índole de variables que maneje la aplicación, lo cual elimina la necesidad de tener que exportar datos a otra aplicación para estudiar su comportamiento en caso de que fuese necesario.
- Otra característica muy importante es que permite enlazar archivos a variables, tanto como para entrada como para salida, permitiendo controlar el momento en el que se produce la asignación.

1.5.3. Emulación JTAG:

El siguiente paso en el desarrollo del producto consiste en la depuración del software y hardware diseñado hasta el momento. Para ello es necesario buscar un método rápido de comunicación entre el software de desarrollo y el Hardware empleado que nos permita comprobar el buen funcionamiento del sistema. Esto se consigue a través de los denominados emuladores, los cuales nos permiten el acceso a todos los dispositivos internos del chip mientras se está ejecutando el programa. Además posibilitan la ejecución del programa paso a paso, en tiempo real o cuasi-real, monitorizar variables o zonas de memoria, etc.

Existen distintos tipos de emuladores dependiendo de su arquitectura:

- **Emulación basada en adaptadores:**
  Consisten en dispositivos que sustituyen al DSP en la tarjeta de desarrollo (pod-based emulation). Constan de un procesador de la misma familia más una lógica accesoria con la que se conecta a un host que permite la depuración de la aplicación. Tienen el inconveniente de que no siempre permiten trabajar a la máxima velocidad del procesador.
- **Emulación basada en monitor:**
  Este tipo de emulación se basa en la ejecución de un programa llamado monitor, cargado en el propio procesador, y que se ejecuta sobre éste proporcionando facilidades de edición de memoria, registros, etc. Su desventaja frente a otros métodos de emulación es su imposibilidad de trabajar en tiempo real.

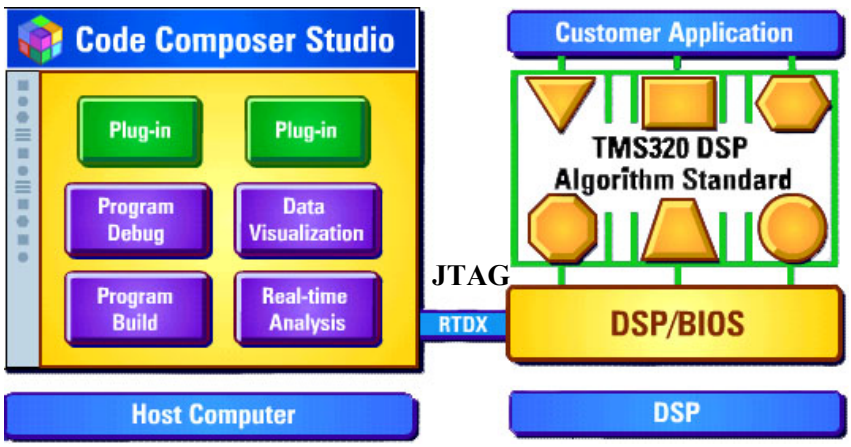


Figura 1.11. Depuración con Code Composer Studio.

- **Emulación basada en rastreo:**
  Este tipo de emulación (también denominada scan-based emulation) se basa en la inclusión de lógica de depuración en el propio DSP por el fabricante, y que monitoriza el estado interno del DSP a través de una conexión **JTAG** (Joint Test Action Group) estándar según la normativa IEEE 1149.1, a través de la cual se tiene acceso directo a las señales internas más importantes del procesador para permitir la lectura o escritura de los datos transferidos en serie. Para no tener que cablear todas las señales del DSP, el procesador lleva incluido una lógica (controlador JTAG) que se encarga de administrar los comandos que recibe, localizar los datos requeridos y enviarlos. De esta forma se puede monitorizar, casi en tiempo real, los cambios en los registros internos del procesador mientras se ejecuta la aplicación, visualizando si es correcto el funcionamiento de la misma. El uso de un emulador externo (como por ejemplo el XDS510 de Spectrum Digital) agiliza el proceso haciéndolo no-intrusivo y permitiendo la depuración en tiempo real sin parar la aplicación. Con esta herramienta se comprueban los cambios que sufren los distintos registros de configuración de los periféricos, las variaciones que sufren algunas zonas de la memoria, se pueden tomar datos para la representación gráfica de algunas variables, etc. Este tipo de emulación es la empleada en los DSP´s fabricados por Texas Instruments que se describen a lo largo de este documento, empleada junto los software de desarrollo Code Composer y Vissim (figura 1.11).

CAPÍTULO 2: KIT DE INICIACIÓN DSK F243

2.1.INTRODUCCIÓN:

Para comenzar a trabajar con DSP´s, existen en el mercado distintas tarjetas basadas en dichos dispositivos, las cuales están preparadas para su uso como banco de pruebas para familiarizarse con la forma de trabajar de los DSP´s. Con ellas, cualquier persona que empiece en este mundo puede aprender lenguajes de programación, a programar dispositivos, ver como trabajan los distintos periféricos del DSP, desarrollar aplicaciones, etc. Además existen ayudas que provienen de distintas fuentes con las que empezar a trabajar, como son manuales, catálogos (data sheets), tutoriales o ejemplos de aplicaciones desarrollados por empresas o universidades. Estas ayudas son claves para el entendimiento tanto del DSP como de sus herramientas de desarrollo, ya que en ellas se explican multitud de conceptos que de otra forma serían difíciles de entender por parte de un principiante en la materia.

En lo que concierne al control de motores los DSP´s que destacan son la familia C2000 fabricados por Texas Instruments, como se comentó anteriormente. Entre ellos destacan el TMS320F243 y el TMS320LF2407 de 16 bits en coma fija. El primero de ellos fue uno de los primeros DSP´s en tener éxito en el mercado del control digital, debido sobretodo a la inclusión de periféricos muy empleados en el control en el mismo circuito integrado. Aunque ya es antiguo y está casi descatalogado, es una magnifica opción para comenzar a adentrarse en el manejo de estos dispositivos. Por ello Spectrum Digital fabricó tarjetas de desarrollo basadas en estos DSP´s, tales como Starter Kits o módulos de evaluación que se analizan en este documento. Con el avance de los DSP´s, Texas Instruments fue desarrollando modelos cada vez más potentes y con mejores prestaciones, el más extendido de los cuales es el TMS320LF2407, muy empleado para implementar distintos tipos de control de motores. Actualmente ha salido al mercado el modelo de 32 bits TMS320F2812.

Para el objeto de estudio de este proyecto se ha empleado la tarjeta de iniciación desarrollada por Spectrum Digital basada en el procesador TMS320F243 (figura 2.1). Las tarjetas de inciación permiten determinar si éste modelo de procesador satisface los requerimientos de la aplicación que se quiere desarrollar, de forma que es un buen soporte para empezar a familiarizarse con el uso y aplicaciones de los DSP´s. Estas placas contienen, además del DSP que se quiere analizar, los circuitos y conexiones necesarias que permiten el desarrollo de aplicaciones, ya sean para objetivos comerciales o académicos. De la placa DSK F243 caben destacar las siguientes características:

- Interfaz de comunicación serie RS-232 para la conexión a un PC.
- 32K words de memoria RAM en placa para datos y programas.
- 8K words de memoria Flash del TMS320F243.
- Cristal oscilador a 5 Mhz.
- 3 conectores de expansión para la conexión de circuitos externos.
- Conector IEEE 1149.1 JTAG para emulación.
- Alimentador de tensión a 5 voltios.
- Cable serie de 9 pines.

- Ensamblador sd24xasm en MS-DOS.
- Go DSP Code Explorer Debugger para depuración.
- Compatible con Code Composer.
- Compatible con C Compiler/assembler/linker de Texas Instruments

Figura 2.1. Starter Kit DSK F243.

Al estar dotada de una memoria de programas y datos de 32K words, tiene capacidad suficiente para múltiples aplicaciones de procesamiento digital de señales. Además dispone de 3 conectores de expansión para su conexión a cualquier circuito externo necesario para la evaluación del kit. Para simplificar el desarrollo de códigos de programa, se incluyen un ensamblador y un depurador, además de un conector JTAG en la placa para su posible emulación a través de programas de desarrollo más avanzados. Todo esto hace que esta placa tenga multitud de posibilidades a la hora de diseñar cualquier tipo de aplicaciones. A lo largo de este documento se irán describiendo las características de los distintos componentes que la constituyen, así como de sus posibles aplicaciones, todo ello a través de ejemplos que ayuden a comprender su funcionamiento y manejo, tanto del hardware como del software de desarrollo.

2.2.CARACTERÍSTICAS DEL PROCESADOR TMS320F243:

Los modelos TMS320x243 de Texas Instruments son actualmente los más extendidos en aplicaciones de control digital. Disponen de un núcleo DSP segmentado de coma fija de 16 bits. Están preparados para la ejecución de una instrucción en cada pulso de reloj, lo cual se consigue a través del procesamiento paralelo o pipeline. Ello consiste en poder buscar y decodificar una nueva instrucción mientras se está ejecutando la instrucción actual. En la ejecución de una instrucción se distinguen las siguientes fases: búsqueda de la instrucción, decodificación de la misma, búsqueda del operando y ejecución de la instrucción. Al ser estas cuatro fases independientes, dichas operaciones pueden solaparse en el tiempo, por lo que en un mismo ciclo pueden estar activas hasta cuatro instrucciones distintas, cada una de ellas en una de las cuatro fases. Por otro lado, poseen una arquitectura Harvard modificada (figura 2.2) con estructura de buses de memoria separados: uno para el espacio de programa y otro para el de datos, que permite múltiples accesos a memoria para leer instrucciones de programa y datos de forma simultánea, lo cual optimiza la potencia de procesamiento.

Figura 2.2. Arquitectura Harvard.

Existe un tercer espacio, el de Entrada/Salida, que es accesible a través de la interfaz de bus externo. Estos dispositivos combinan además la arquitectura de la CPU con diversos periféricos avanzados para conseguir configuraciones óptimas en aplicaciones de control de motores, para lo cual poseen un bus de periféricos que permite la conexión de un gran número de periféricos (figura 2.3). Entre estos periféricos se incluye un gestor de eventos, que está provisto de temporizadores de propósito general y registros de comparación para la generación de hasta 12 salidas PWM. Otro de los periféricos importantes es el convertidor analógico-digital: Disponen de dos convertidores de 10 bit que pueden realizar dos conversiones simultáneas en 6,6 µs. Por otro lado, existen modelos cuya memoria es ROM (C243), mientras que para otros es FLASH EEPROM (F243). Destacar que todo lo que a continuación se describe está integrado en el mismo chip, lo que hace que este tipo de DSP´s tengan tanto éxito frente a los microprocesadores tradicionales.

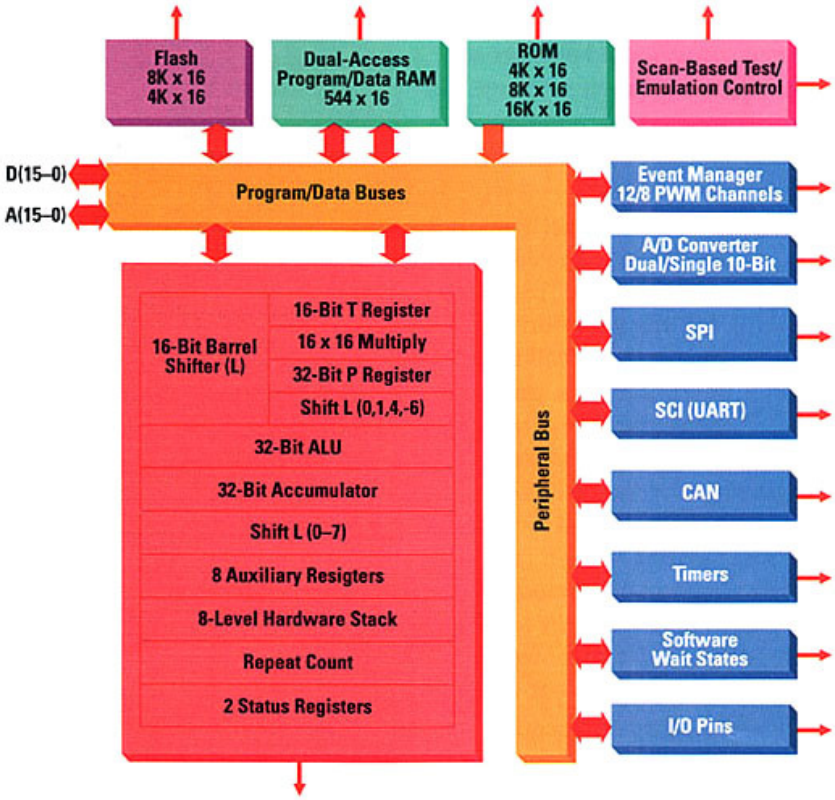


Figura 2.3. Arquitectura de los procesadores x24x.

A continuación se detallan las características principales y prestaciones del procesador TMS320F243:

- Basado en la CPU del TMS320C24x.
  - Código compatible con F241/C242/LF2407.
  - Juego de instrucciones compatibles con el TMS320F24x.

- 50 nsg de tiempo del ciclo de instrucción.

- 20 millones de instrucciones de punto fijo por segundo (MIPS).

- Memoria interna:
  - 8K Words x 16 bits de memoria Flash EEPROM.
  - 544 Words x 16 bits de memoria RAM de acceso dual en chip (DARAM):
    - 288 Words x 16 bits para memoria de datos en dos bloques denominados B1 (256 Words) y B2 (32 Words).
    - 256 Words x 16 bits de memoria configurable para datos o programas en un bloque llamado B0.
  - Capaz de direccionar hasta 224K Words x 16 bits de memoria total.

- Interfaz de memoria externa de 16 bits.

- Mapa de memoria dividido en tres espacios (192K Words en total):
  - 64K Words x 16 bits de memoria de Programa.
  - 64K Words x 16 bits de memoria de Datos.
  - 64K Words x 16 bits de memoria de I/O.

- Gestor de eventos o Event Manager (EV) optimizado para el control de motores:
  - 8 canales comparadores/PWM.
  - 2 Timers de propósito general de 16 bits.
  - 3 unidades de comparación de 16 bits con programación de tiempo muerto (Dead-Band).
  - 3 unidades de captura (2 con capacidad de capturar pulsos de encoder de cuadratura).

- Convertidor Analógico-Digital (A/D) de 10 bits con:
  - 8 entradas multiplexadas.
  - 600 ns de tiempo mínimo de conversión.
  - Tensiones de referencia programables.
  - Inicio de conversión por eventos en la unidad EV, pin externo o por programación.

- Módulos de comunicaciones:
  - Controller Area Network (CAN).
  - Iterfaz serie (SCI).
  - Interfaz de 16 bits de periféricos Serie (SPI).

- 32 pines de Entrada/Salida programables (GPIO).

- 5 Interrupciones externas:
  - Protección del driver de potencia (PDPINTA/B).

- Reset.
- NMI (interrupción no enmascarables).
- Interrupciones externas enmascarables (XINTA/B).

- Modos de reducción de bajo consumo:
  - 3 modos de reducción del consumo (power-down).
  - Reactivación del funcionamiento por diversos periféricos.

- Temporizados Watchdog (WD).

- Interfaz JTAG para emulación scan-based.

- Tecnología CMOS de altas prestaciones.

2.2.1. Estructura interna del TMS320F243:

Como puede observarse en la figura 2.3, el corazón del dispositivo, la CPU, contiene básicamente los siguientes bloques:

- **Sección aritmético-lógica (ALU):**
  Utiliza operandos de 32 bits a los que aplica operaciones aritméticas (suma, resta, suma con acarreo, división, etc) y lógicas (AND, OR, negación, etc), produciendo un resultado de 32 bits que se almacena en el registro acumulador (ACC). Una entrada siempre es el acumulador y la otra puede venir del multiplicador o de los registros de desplazamiento.

- **Acumulador (ACC):**
  Registro de 32 bits que contiene datos a los que se les va a aplicar alguna operación. Puede emplearse en dos partes de 16 bits: la parte alta del acumulador (ACCH) o la parte baja (ACCL). Soporta operaciones de desplazamiento y rotación.

- **Desplazadores:**
  Registros de 32 bits que permiten el escalado, la extracción de bits, aritmética extendida y operaciones de prevención de overflow.

- **Multiplicador:**
  El multiplicador hardware es un elemento esencial en la arquitectura de los DSP´s, ya que es una de las operaciones básicas en el procesado digital de señales. Aplica productos entre dos números de coma fija de 16 bits sin signo y en complemento a dos, dando un resultado de 32 bits en el mismo formato, con o sin signo. Todo ello lo realiza en un solo ciclo máquina. La combinación en el hardware de un sumador y un multiplicador (MAC o multiplicador-acumulador) es muy deseable puesto que permite implementar funciones (suma de productos) muy comunes en el tratamiento de señales digitales y control de procesos, que pueden expresarse de la forma siguiente:

$$F = \sum X \cdot Y$$

Este tipo de funciones son muy utilizadas en filtro digitales, transformadas de Fourier, reguladores PID, etc. Las características principales de un MAC son las siguientes:

- Multiplicación y suma en un solo ciclo máquina.
- Detección previa de overflow.
- Líneas de realimentación internas que permiten optimizar el control de bucles.
- Modo paralelo (pipeline).

- **Registro de instrucciones:**
  Registro en el que se almacena la siguiente instrucción a ejecutar por la CPU.

- **Unidad aritmética de registros auxiliares (ARAU):**
  Unidad que contiene 8 registros (denominados ARn) en los que se pueden realizar operaciones aritméticas de forma independiente a la sección principal de la ALU. Estos registros auxiliares son muy empleados en el direccionamiento indirecto de cualquier lugar de memoria cuando se programa en lenguaje ensamblador.

- **Registros de estado:**
  Los registros de estado ST0 y ST1 contienen los estados de diversas condiciones y modos de funcionamiento y bits de control. Pueden almacenarse o cargarse desde la memoria de datos, lo que permite almacenar y restaurar el estado de la CPU cuando se trabaja con llamadas a subrutinas. De entre los datos que contienen destacan bits o secciones:
  - ARP: Varios bits que indican el registro auxiliar ARx que está siendo empleado.
  - OV: Bit que indica si ha sucedido un desbordamiento en la operación realizada.
  - INTM: Bit que activa o desactiva globalmente las interrupciones enmascarables del DSP.
  - CNF: Bit que configura la memoria RAM interna del DSP como memoria de datos o de programa.
  - SXM: Bit de configuración del uso de signo en los datos.
  - XF: Determina el estado del pin de salida XF de uso general del DSP.

2.2.2. Memoria del TMS320F243:

Como se mencionó anteriormente, los procesadores de la familia TMS320F423 utilizan buses de direcciones de 16 bits que pueden acceder a tres espacios de memoria individualmente seleccionables, por lo que pueden direccionar un total de 192K words (figura 2.4). Esto tres espacios se dividían en:

- **64K para memoria de programas:** En este espacio se almacenan las instrucciones que se van a ejecutar durante la ejecución del programa. Suele almacenar también operandos inmediatos e información sobre tablas.
- **64K para memoria de datos:** Este espacio contiene datos utilizados por las instrucciones que componen el programa.
- **64K de espacio E/S:** Este espacio sirve de interfaz con periféricos externos y puede contener algunos registros en el propio chip.

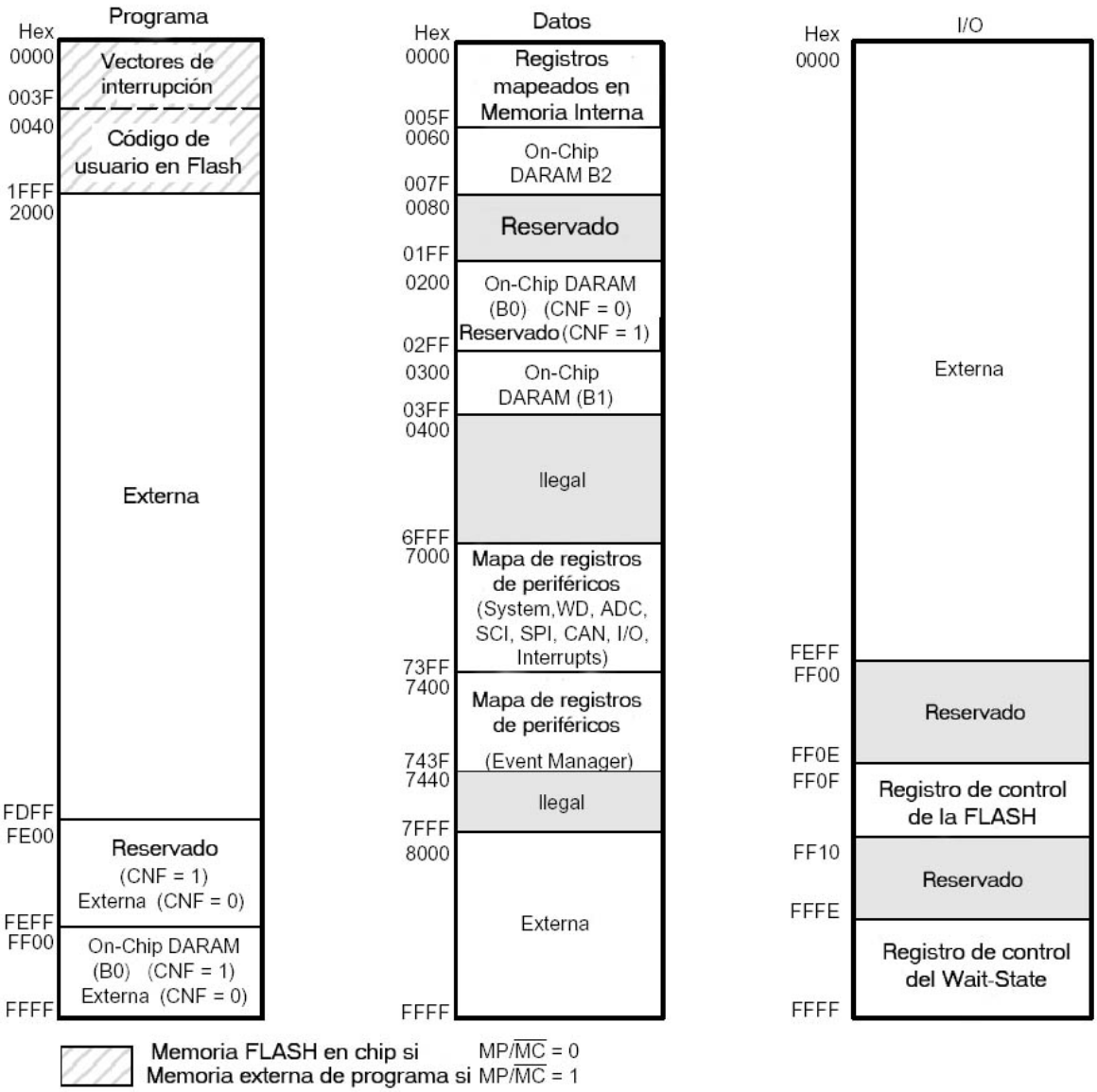


Figura 2.4. Mapa de memoria del TMS320F243.

Estos espacios están localizados en los distintos tipos de memoria interna de que disponen el TMS320F243 junto con la memoria externa que puede conectarse, pudiéndose configurar algunos de los distintos bloques de memoria que lo forman. Los dos tipos de memoria interna en los que se distribuyen son:

- **FLASH EEPROM:**
  Este tipo de memoria no volátil se puede programar y borrar eléctricamente, de forma que puede reprogramarse directamente sin tener que sacar el chip de la placa. La inclusión de este tipo de memoria es una mejora respecto de los primeros modelos de DSP´s de la familia C2000, los cuales incluían memoria ROM más complicada de programar. Además la memoria FLASH es una memoria rápida que permite accesos en un solo ciclo de reloj, por lo que es muy empleada como memoria interna de programas para mejorar las prestaciones del dispositivo.

- **DARAM:** Como apoyo a la memoria FLASH, el TMS320F243 dispone de memoria interna RAM de acceso dual o DARAM. Ésta memoria puede ser accedida por dos puertos diferentes en un solo ciclo de reloj, lo que implica una velocidad de acceso máxima y sin estados de espera. Se divide en tres bloques: B0, B1 y B2, los cuales tienen como función primaria el mantenimiento de datos, aunque el bloque B0 puede programarse para memoria de programas o de datos.

Ambos tipos de memoria pueden configurarse para alojar datos o programas dependiendo de la configuración de algunos bits de los registros de control (bit CNF contenido en el registro de estado ST1), y el pin MP/MC que configura el DSP para funcionar en modo Microprocesador o Microcontrolador (figura 2.4):

- Si el pin MP/MC = 1, la zona de memoria de programas alojada entre las direcciones 0000h-2000h se sitúa en la memoria externa de programas, no en la FLASH interna del DSP.
- Si el pin MP/MC = 0, dicho rango de direcciones sí se sitúa en la memoria FLASH del DSP.
- El bloque de memoria B0 puede configurarse como memoria de datos si el valor del bit CNF = 0, entre las direcciones 0200h-02FFh. En este caso la zona que ocuparía en la memoria de programas pasa a ser mapeada como memoria externa.
- Si el bit CNF = 1, el bloque de memoria B0 se configura como memoria de programas entre las direcciones FF00h-FFFFh. De esta forma la zona que ocuparía en la memoria de datos pasa a ser mapeada como zona reservada.
- Los bloques B1 y B2 se encuentran siempre localizados en la memoria de datos.

De esta forma se obtienen diversas combinaciones en función de estas opciones. De todas formas, sea como sea la configuración de los distintos bloques de memoria, se observa en el mapa de memoria, que en las posiciones bajas de la memoria de programas (0000h-003Fh) están reservadas siempre a los vectores de interrupción, los cuales son los siguientes:

Nombre	Dirección	Nombre	Dirección
Reset	0000h-0001h	Reservado	000Eh-000Fh
Interrupción de nivel 1	0002h-0003h	Interrup. Software	0010h-0021h
Interrupción de nivel 2	0004h-0005h	TRAP	0022h-0023h
Interrupción de nivel 3	0006h-0007h	NMI	0024h-0025h
Interrupción de nivel 4	0008h-0009h	Reservado	0026h-0027h
Interrupción de nivel 5	000Ah-000Bh	Interrup. Software	0028h-003Fh
Interrupción de nivel 6	000Ch-000Dh		

Tabla 2.1. Tabla de vectores de interrupción.

Por otro lado, el mapa de memoria I/O tiene mapeados el registro de control de la FLASH y el del generador de estados de espera (WSGR), siendo la mayoría de las direcciones dedicadas a memoria externa para mapear periféricos externos que puedan conectarse al DSP.

También hay que destacar que de los 64K disponibles para memoria de datos hay 32K de memoria interna que incluyen los registros internos del DSP mapeados en memoria, la memoria DARAM y los registros de periféricos mapeados en memoria (figura 2.5). El resto forman parte de la memoria de datos externa.

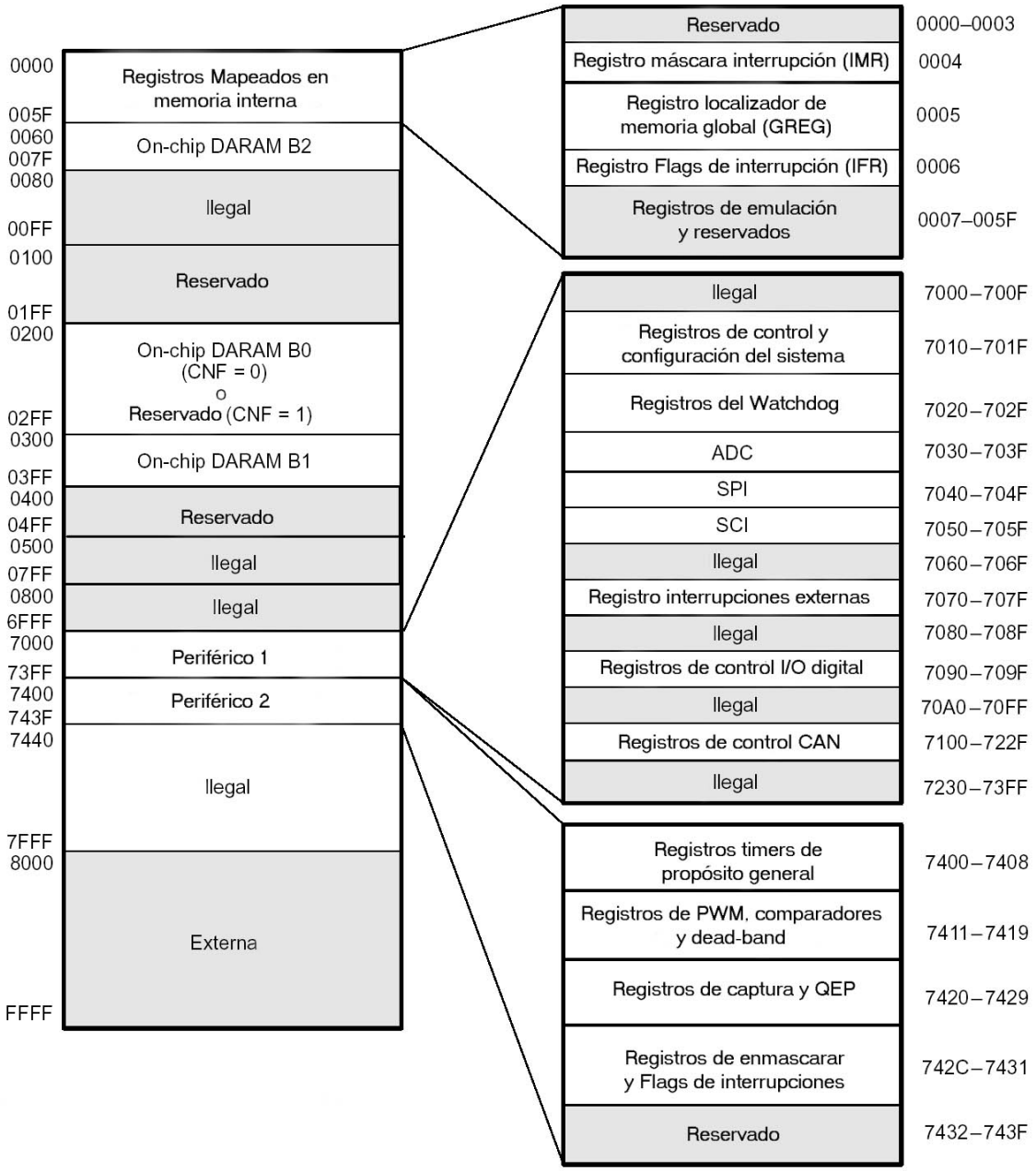


Figura 2.5. Mapa de memoria de datos.

La memoria de datos admite direccionamientos directos e indirectos. En direccionamiento directo la memoria de datos se divide en bloques de 128 palabras que se denominan páginas, de forma que los 64K de memoria de datos se dividen en 512 páginas desde la 0 (0000h-007Fh) hasta la 511 (FF80h-FFFFh). Esto se emplea mucho cuando se programa en lenguaje ensamblador, en el que para acceder a una determinada dirección hay que cargar primero el número de página en el que se encuentra ésta, lo cual se hace mediante la instrucción "LDP nº de página". Para mayor información referirse a los documentos que desarrollan los tipos de direccionamiento.

En el mapa de memoria (figuras 2.4 y 2.5) aparecen zonas marcadas como ilegales y reservadas. Todas estas posiciones son utilizadas por el procesador y no debe accederse a ellas. Las nombradas como ilegales provocan una interrupción no enmascarable (NMI) si se accede a ellas. Las zonas reservadas son para test y si se usan pueden provocar cambios en el modo de operación.

2.2.3. Periféricos del TMS320F243:

El TMS320x24x incorpora funcionalidades propias de los microcontroladores de gama alta, dirigidas a resolver aplicaciones de control digital de movimiento. Esta circunstancia permite hoy en día la resolución de aplicaciones de control de motores con niveles de integración y coste muy económicos. Ello se debe en gran medida a la incorporación de múltiples periférico empleados en el control en el mismo chip que el procesador. A continuación se hará una breve descripción de los principales periféricos que incorpora el TMS320F243 y que en capítulos posteriores se entrará en detalles sobre las aplicaciones y empleo de cada uno ellos.

Los principales periféricos que incorpora el TMS320x24x para la resolución de aplicaciones de control digital de motores son:

- **Gestor de eventos**:
  Este módulo de control especializado en control de motores minimiza la intervención de la CPU a la hora de controlar conversores de potencia y permite al dispositivo generar salidas que son capaces de controlar cualquier tipo de motor. Para ello dispone de temporizadores de propósito general y registros de comparación para la generación de hasta 8 canales PWM: 2 señales independientes generadas con 2 temporizadores de propósito general y otras 6 generadas por 3 unidades comparadoras, cada una de las cuales genera dos señales iguales pero opuestas. Éstas últimas señales están diseñadas para el control de inversores trifásicos, por lo que además se pueden configurar con tiempos muertos para evitar un cortocircuito en un semipuente del inversor mediante la unidad Dead-band. Los temporizadores de propósito general pueden emplearse también para la inicación de distintos procesos, como por ejemplo el comienzo de una conversión analógico-digital. Por otro lado, viene dotado de 3 entradas de captura de señales, dos de ellas QEP (Quadrature Encoder Pulse), las cuales proporcionan al sistema una herramienta excepcional para obtener información sobre la posición y la velocidad de una máquina rotativa. La mayoría de estos periféricos comparten un mismo pin como interfaz al exterior con otros dispositivos como son las 32 entradas y salidas digitales de propósito general de que dispones el DSP.
- **Convertidor A/D**:
  La integración en el mismo chip de un convertidor analógico-digital de 10 bits de precisión, con 8 entradas multiplexadas y alta velocidad de conversión, ha posibilitado la reducción de espacio y componentes de los sistemas de control, además de mejorar su rendimiento, ya que minimizan la intervención de la CPU y logran una alta velocidad de muestreos y conversión. Con ellos, por ejemplo, se puede obtener la velocidad del motor mediante la lectura de la señal de una dinamo tacométrica a través del convertidor A/D. En la documentación del TMS320F243 se indica que éste

modelo dispone de dos unidades de convertidores A/D, lo cual no es cierto, ya que físicamente sólo se dispones de un convertidor. Ello se ha hecho para hacer compatibles las nomenclaturas de los distintos procesadores de la serie C2000, pero a la hora de trabajar con el convertidor se hace como si hubiese dos de ellos como se detallará más adelante.

- **Puertos de comunicación serie**:
  Con los que el DSP puede comunicarse con otros dispositivos externos. Estos pueden ser síncronos (Serial Peripheral Interface o SPI) o asíncronos (Serial Communication Interface o SCI-UART).
- **Módulo CAN (Controller Area Network)**:
  Actualmente se está extendiendo el uso de un nuevo protocolo de comunicación serie entre dispositivos (CAN), ya que permite establecer un algoritmo de control distribuido robusto y bastante más sencillo desde el punto de vista de la instalación y mantenimiento. Consiste en la instalación alrededor de un bus central de distintos equipos (sensores, actuadores, controladores, etc.), cada uno de ellos con un objetivo concreto, funcionando simultáneamente y de forma independiente los unos de los otros, con lo que se dividen las responsabilidades, aunque será necesario un trasvase de información entre ellos. Ahí es donde se aprovecha la enorme versatilidad del bus CAN, debido a su alta velocidad de comunicación y gran inmunidad al ruido, lo que lo hace muy útil en entornos de trabajo ruidosos. Los nodos conectados al bus no presentan direcciones específicas. En lugar de eso la información se compone de mensajes etiquetados con un identificador que determinará la prioridad del mensaje y el contenido. Así, cada equipo conectado al bus recibe el mensaje y, en función de su contenido, realiza las acciones pertinentes. La integración en el DSP de un controlador CAN ahorra espacio en la placa y la necesidad de un controlador externo más costoso. Además permite el control de varios motores por un solo DSP en cualquier aplicación industrial.
- **Generador de estados de espera:**
  Los estados de espera son necesarios cuando el procesador se comunica con memoria o periféricos externos que son más lentos que él. Éste dispositivo añade estados de espera que hacen que la CPU espere ciclos extra de reloj durante un tiempo para que al periférico externo le dé tiempo a acceder a los datos u otros eventos. Ello se consigue configurando el registro de control del generador de estados de espera o **WSGR**, que se encuentra en la dirección **FFFFh del espacio I/O**:

15–11	10–9	8–6	5–3	2–0
Reservado	BVIS	ISWS	DSWS	PSWS

Las secciones ISWS, DSWS y PSWS constan de tres bits para configurar hasta 7 (111) estados de espera en cada uno de los distintos espacios en que se divide la memoria:

- ISWS: para configurar estados de espera en el espacio I/O.
- DSWS: para el espacio de datos.
- PSWS: para el espacio de programas.

Después de un reset del DSP todas ellas quedan configuradas para generar 7 estados de espera en cada espacio.

2.2.4. Pines del TMS320F243:

Para tener acceso a todas las señales que son necesarias para los distintos dispositivos que incorpora el TMS320F243, el encapsulado del DSP puede ser de distintas formas (PGE, FN y PG), teniendo mayor o menor número de pines externos. En el caso que nos ocupa, el starter kit DSK F243, el encapsulado llevado a cabo en el DSP es PGE (figura 2.5) que proporciona un total de 144 pines, a través de los cuales se tiene acceso a las señales de los distintos periféricos.

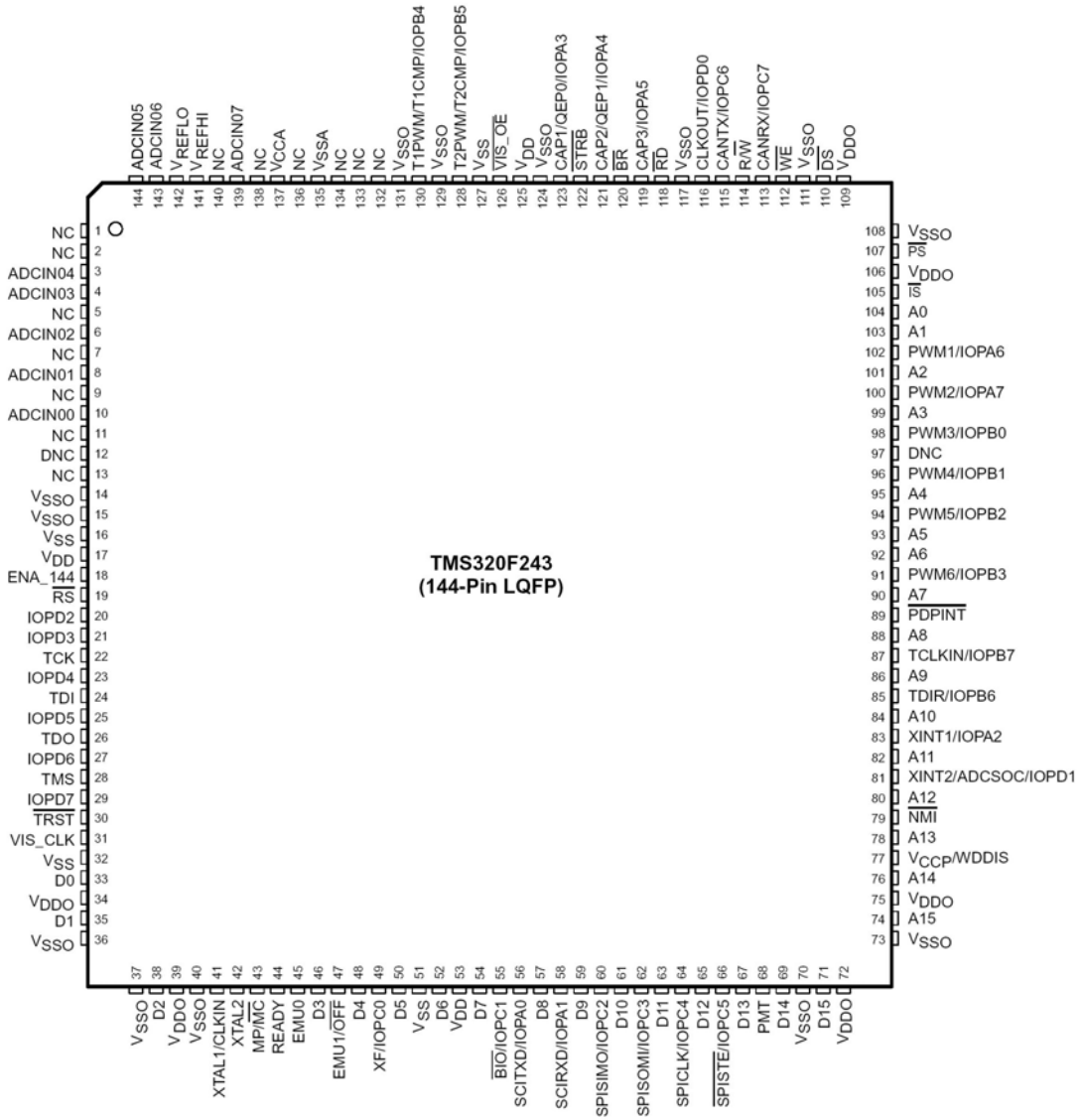


Figura 2.6. Encapsulado PGE del TMS320F243.

Como se aprecia en la figura, muchos de estos pines están compartidos por varias señales pertenecientes a dispositivos distintos. A continuación se describirán las distintas funciones de cada uno y su nombre, así como su situación en el encapsulado, de forma ordenada por dispositivos. Con letra negrita e itálica se indican las funciones que se encuentran activas después de un reset en el DSP, en todos aquellos pines que son compartidos por más de una función.

Nombre del pin	Nº de pin	Descripción
ADCIN00	10	Entrada del Canal 0 del ADC
ADCIN01	8	Entrada del Canal 1 del ADC
ADCIN02	6	Entrada del Canal 2 del ADC
ADCIN03	4	Entrada del Canal 3 del ADC
ADCIN04	3	Entrada del Canal 4 del ADC
ADCIN05	144	Entrada del Canal 5 del ADC
ADCIN06	143	Entrada del Canal 6 del ADC
ADCIN07	139	Entrada del Canal 7 del ADC
V_{CCA}	137	Alimentación positiva analógica del ADC (max. 5V)
V_{SSA}	135	Alimentación negativa analógica del ADC
V_{REFHI}	141	Referencia de la tensión alta del ADC
V_{REFLO}	142	Referencia de la tensión baja del ADC

Tabla 2.2. Pines del convertidor A/D.

Nombre del pin	Nº de pin	Descripción
T1PWM/T1CMP/***IOPB4***	130	Salida de la comparación del Timer 1 / Pin 4 de E/S del puerto B
T2PWM/T2CMP/***IOPB5***	128	Salida de la comparación del Timer 2 / Pin 5 de E/S del puerto B
TDIR / ***IOPB6***	85	Dirección de conteo de los Timers. Si TDIR=1, cuenta ascendente / Pin 6 d E/S del puerto B
TCLKIN / ***IOPB7***	87	Entrada externa del reloj para el Timer / Pin 7 de E/S del puerto B
CAP1/QEP0/ ***IOPA3***	123	Entrada de captura 1 / Cuadratura del encoder 0/ Pin 3 de E/S del puerto A
CAP2/QEP1/ ***IOPA4***	121	Entrada de captura 2 / Cuadratura del encoder 1/ Pin 4 de E/S del puerto A
CAP3 / ***IOPA5***	119	Entrada de captura 3 / Pin 5 de E/S del puerto A
PWM1 / ***IOPA6***	102	Salida PWM1 / Pin 6 de E/S del puerto A
PWM2 / ***IOPA7***	100	Salida PWM2 / Pin 7 de E/S del puerto A
PWM3 / ***IOPB0***	98	Salida PWM3 / Pin 0 de E/S del puerto B
PWM4 / ***IOPB1***	96	Salida PWM4 / Pin 1 de E/S del puerto B
PWM5 / ***IOPB2***	94	Salida PWM5 / Pin 2 de E/S del puerto B
PWM6 / ***IOPB3***	91	Salida PWM6 / Pin 3 de E/S del puerto B
PDPINT	89	Entrada de la interrupción de protección de las señales de control de potencia

Tabla 2.3. Pines del Event Manager.

Nombre del pin	Nº de pin	Descripción
CANRX / ***IOPC7***	113	Recepción de datos del CAN / Pin 7 E/S del puerto C
CANTX / ***IOPC6***	115	Transmisión de datos del CAN/Pin 6 E/S de puerto C
SCITXD / ***IOPA0***	56	Transmisión de datos del SCI / Pin 0 E/S de puerto A
SCIRX / ***IOPA1***	58	Recepción de datos del SCI / Pin 1 E/S del puerto A
SPICLK / ***IOPC4***	64	Reloj del SPI / Pin 4 de E/S del puerto C
SPISIMO / ***IOPC2***	60	Entrada del Esclavo o Salida del Maestro del SPI / Pin 2 de E/S del puerto C
SPISOMI / ***IOPC3***	62	Salida del Esclavo o Entrada del Maestro del SPI / Pin 3 de E/S del puerto C
SPISTE / ***IOPC5***	66	Habilitación del Esclavo del SPI / Pin 5 de E/S del puerto C

Tabla 2.4. Pines para las comunicaciones.

Nombre del pin	Nº de pin	Descripción
IOPD2	21	GPIO dedicado - Pin 2 del puerto D
IOPD3	21	GPIO dedicado - Pin 3 del puerto D
IOPD4	23	GPIO dedicado - Pin 4 del puerto D
IOPD5	25	GPIO dedicado - Pin 5 del puerto D
IOPD6	27	GPIO dedicado - Pin 6 del puerto D
IOPD7	29	GPIO dedicado - Pin 7 del puerto D

Tabla 2.5. Pines dedicados E/S.

Nombre del pin	Nº de pin	Descripción
EMUO	45	Pin empleado por el JTAG
EMU1 / OFF	47	Pin empleado por el JTAG
TCK	22	Testea el reloj del JTAG
TDI	24	Testeo de la entrada de datos del JTAG
TDO	26	Testeo de la salida de datos del JTAG
TMS	28	Selecciona el modo de testeo del JTAG
TRST	30	Reset interno del JTAG
DS	110	Activación del espacio de datos. Activa a nivel bajo
IS	105	Activación del espacio I/O. Activa a nivel bajo
PS	107	Activación del espacio de programas. Activa nivel bajo
R / W	114	Indica dirección de transferencia en comunicación con otros dispositivos
XF / IOPC0	49	Pin de salida de uso general XF / Pin 0 de E/S del puerto C
RD	118	Habilitación de lectura de memoria. Activa a nivel bajo
WE	112	Habilitación de escritura en memoria. Activa a nivel bajo
STRB	122	Habilitación de acceso a memoria externa. Act. nivel bajo
READY	44	Se pone a nivel bajo para añadir tiempos de espera

Tabla 2.6. Pines para la emulación y test.

Nombre del pin	Nº de pin	Descripción
MP / MC	43	Selección del modo de trabajo como Microprocesador o como Microcontrolador
ENA_144	18	Activa a nivel alto habilita las señales del interfaz externo
VIS_OE	126	Habilita la visibilidad de la salida del bus de datos. Activo a nivel bajo
A0	104	Bit 0 del bus de direcciones (16 bits)
A1	103	Bit 1 del bus de direcciones (16 bits)
A2	101	Bit 2 del bus de direcciones (16 bits)
A3	99	Bit 3 del bus de direcciones (16 bits)
A4	95	Bit 4 del bus de direcciones (16 bits)
A5	93	Bit 5 del bus de direcciones (16 bits)
A6	92	Bit 6 del bus de direcciones (16 bits)
A7	90	Bit 7 del bus de direcciones (16 bits)
A8	88	Bit 8 del bus de direcciones (16 bits)
A9	86	Bit 9 del bus de direcciones (16 bits)
A10	84	Bit 10 del bus de direcciones (16 bits)
A11	82	Bit 11 del bus de direcciones (16 bits)
A12	80	Bit 12 del bus de direcciones (16 bits)
A13	78	Bit 13 del bus de direcciones (16 bits)
A14	76	Bit 14 del bus de direcciones (16 bits)
A15	74	Bit 15 del bus de direcciones (16 bits)
D0	33	Bit 0 del bus de datos (16 bits)
D1	35	Bit 1 del bus de datos (16 bits)
D2	38	Bit 2 del bus de datos (16 bits)
D3	46	Bit 3 del bus de datos (16 bits)
D4	48	Bit 4 del bus de datos (16 bits)
D5	50	Bit 5 del bus de datos (16 bits)
D6	52	Bit 6del bus de datos (16 bits)
D7	54	Bit 7 del bus de datos (16 bits)
D8	57	Bit 8 del bus de datos (16 bits)
D9	59	Bit 9 del bus de datos (16 bits)
D10	61	Bit 10 del bus de datos (16 bits)
D11	63	Bit 11 del bus de datos (16 bits)
D12	65	Bit 12 del bus de datos (16 bits)
D13	67	Bit 13 del bus de datos (16 bits)
D14	69	Bit 14 del bus de datos (16 bits)
D15	71	Bit 15 del bus de datos (16 bits)

Tabla 2.7. Pines para el acceso a memoria.

Nombre del pin	Nº de pin	Descripción
XTAL1 / CLKIN	41	Pin de entrada del oscilador PLL
XTAL2	42	Salida del amplificador inversor del oscilador
RS	19	Reset del dispositivo a nivel bajo
XINT1 / ***IOPA2***	83	Entrada de la interrupción externa 1 / Pin 2 de E/S del puerto A
XINT2 /ADCSOC / ***IOPD1***	81	Entrada de la interrupción externa 2 / programada produce el arranque del conversor A/D / Pin 1 de E/S del puerto D
V_{CCP}	77	Voltaje de 5V de programación de la memoria FLASH. Sino debe estar a masa
BIO / IOPC1	55	Entrada de control de salto
V_{DD0}	34, 39, 72, 75, 106, 109	Alimentación digital de los buffers de I/O de 5V
V_{SS}	16, 32, 51, 127	Masa de alimentación digital del núcleo del sistema
V_{SSO}	14, 15, 36, 37, 40, 70, 73, 108, 111, 117, 124, 129, 131	Masa de alimentación digital de los buffers de I/O
V_{DD}	17, 53, 125	Alimentación digital del núcleo del sistema de 5V

Tabla 2.8. Otros pines de usos varios.

2.3.CARACTERÍSTICAS DEL STARTER KIT DSK F243:

A continuación, y una vez analizado el procesador TMS320F243 y sus dispositivos, se van a describir las particularidades que presenta la tarjeta de iniciación DSK F243 basada en dicho procesador. Dicha tarjeta (figura 2.1) es una plataforma de pruebas del procesador en cuestión, de forma que sea fácil la interconexión con otros dispositivos para la realización de aplicaciones de uso comercial o didáctico. Para ello, el fabricante Spectrum Digital, ha dispuesto la circuitería necesaria para acceder de forma sencilla a las principales señales de los periféricos que incorpora el procesador, a las cuales se accede a través de una serie de conectores de expansión. En la figura 2.7, se puede apreciar la interconexión existente entre los principales bloques que constituyen la placa de inciación: Procesador, memoria y conectores de expansión y comunicaciones. Este es el objetivo principal de la placa de inciación: dotar a los diseñadores de aplicaciones de una herramienta lo suficientemente potente como para desarrollar y probar una aplicación con un determinado modelo de DSP antes de llevarla a cabo de manera definitiva y comercial. A través de este proyecto se presentarán ejemplos de cómo conectar ésta placa de iniciación a distintos equipos: desde un PC a distintos circuitos para desarrollar aplicaciones de control de motores.

Cabe destacar que, aunque la placa disponga de un procesador TMS320F243, al ser una placa de inciación trae algunas de sus prestaciones limitadas de fábrica, principalmente la memoria. Esto se desarrollará en apartados posteriores.

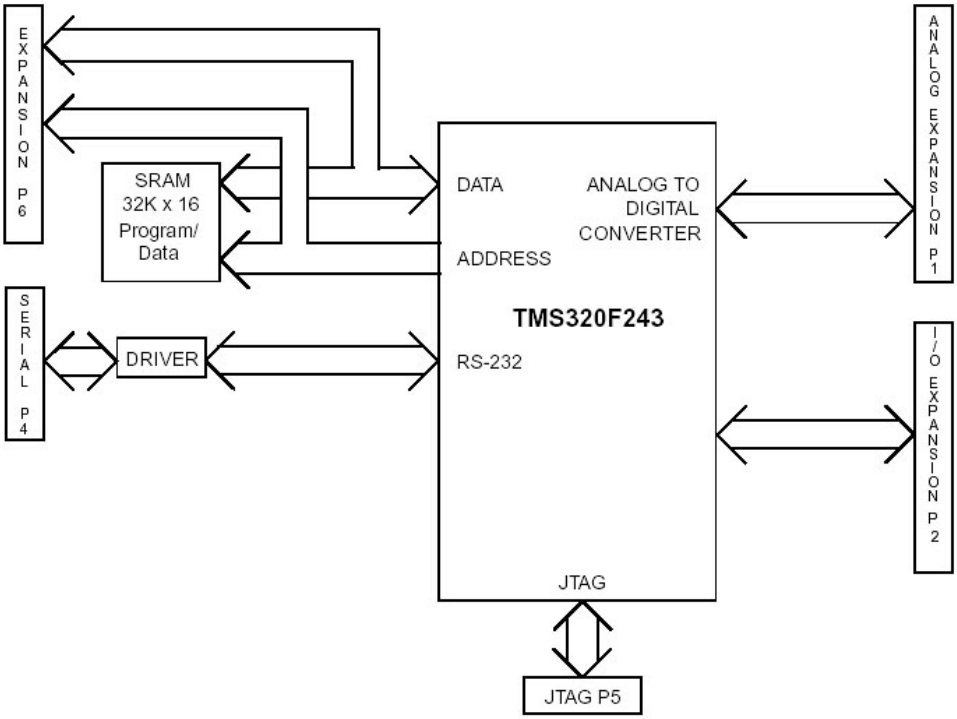


Figura 2.7. Estructura general del Starter Kit DSK F243.

2.3.1. Interfaz externa:

Para la comunicación de la tarjeta con otros dispositivos o, simplemente, para tener acceso a determinadas señales, ésta placa dispone de diversas conexiones que le sirven de interfaz al exterior (figura 2.8). Con ellas se puede conseguir la programación del procesador o la conexión de un circuito externo adaptador de señales con el que se maneja algún dispositivo exterior.

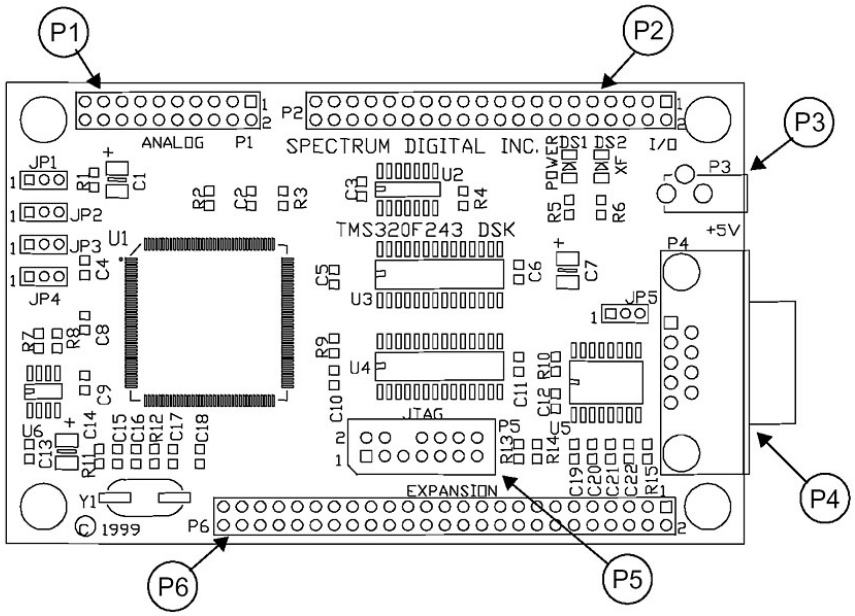


Figura 2.8. Conectores de expansión del DSK F243.

Sus principales características son:

- ***Conector de expansión analógico(P1)*:**
  A través de este conector de 20 pines, se tiene acceso directo a las 8 entradas del conversor analógico-digital que contiene el DSP. De esta forma la tarjeta puede obtener medidas del exterior una vez adaptadas los niveles de tensión de dichas señales.

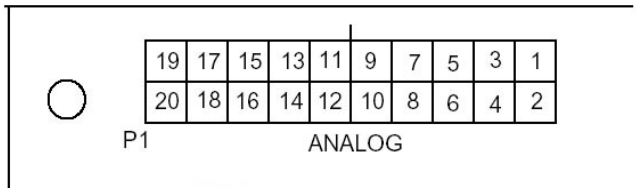


Figura 2.9. Conector de expansión analógico.

La disposición de las distintas señales del convertidor analógico en dicho conector se describen a continuación:

Pin Nº	Señal	Pin Nº	Señal
1	GND	2	ADC0
3	GND	4	ADC1
5	GND	6	ADC2
7	GND	8	ADC3
9	GND	10	ADC4
11	GND	12	ADC5
13	GND	14	ADC6
15	GND	16	ADC7
17	GND	18	VREFLO
19	GND	20	VREFHI

Tabla 2.9. Pines del conector de expansión analógico.

- ***Conector de expansión de Entrada/salida (P2):***
  Conector de 40 pines por el cual se tiene acceso a las señales digitales que genera y recibe el DSP. Entre ellas cabe destacar las señales del protocolo CAN y de los puertos serie SPI y SCI, las señales del generador de estados de espera, señales QEP y otras controladas por el gestor de eventos (señales PWM, timers de uso general), y sobretodo las señales digitales de Entrada/Salida de los distintos puertos de comunicación de que dispone el procesador.

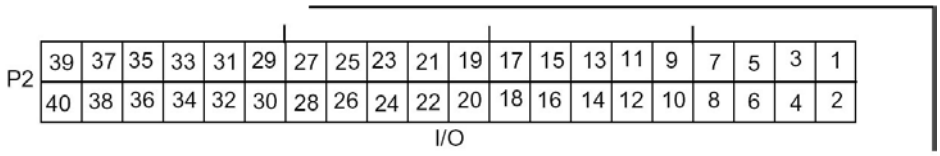


Figura 2.10. Conector de expansión de Entrada/Salida.

A través de éstos 40 pines se tienen acceso a las siguientes señales del procesador:

Pin Nº	Señal	Pin Nº	Señal
1	V_{CC}	2	V_{CC}
3	SCITXD/IOPA0	4	SCIRXD/IOPA1
5	XINT-/IOPA2	6	CAP1/QEP0/IOPA3
7	CAP2/QEP1/IOPA4	8	CAP3/IOPA5
9	CMP1/IOPA6	10	CMP2/IOPA7
11	CMP3/IOPB0	12	CMP4/IOPB1
13	CMP5/IOPB2	14	CMP6/IOPB3
15	T1PWM/T1CMP/IOPB4	16	T2PWM/T2CMP/IOPB5
17	IDIR/IOPB6	18	TCLKIN/IOPB7
19	GND	20	GND
21	XF/IOPC0	22	BIO-/IOPC1
23	SPISIMO/IOPC2	24	DPISOMI/IOPC3
25	SPICLK/IOPC4	26	SPISTE/IOPC5
27	CANTX/IOPC6	28	CANRX/IOPC7
29	CLKOUT/IOPD0	30	XINT2-/ADCSOC/IOPD1
31	IOPD2	32	IOPD3
33	IOPD4	34	IOPD5
35	IOPD6	36	IOPD7
37	PDPINT-	38	RESERVADO
39	GND	40	GND

Tabla 2.10. Pines Del conector de Entrada/Salida.

- ***Interfaz serie RS-232 (P4)*:**
  Conector de 9 pines para la conexión de la tarjeta a un PC para su programación y depuración de aplicaciones. También puede servir para la comunicación con otros DSP´s o dispositivos externos.

- ***Conector JTAG (P5)*:**
  Interfaz muy utilizada para la comunicación entre el TMS320F243 y el PC en aplicaciones de depuración, emulación y programación a través de un emulador JTAG externo (no incluido en el kit).

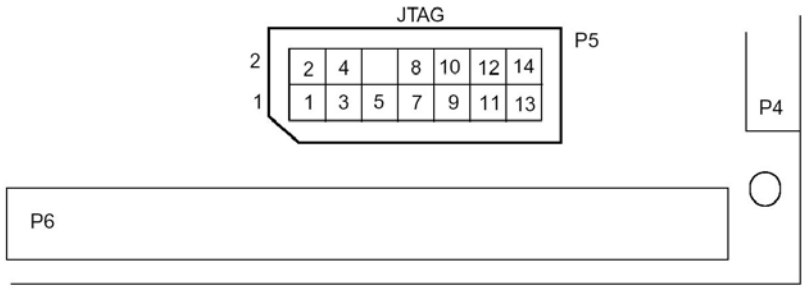


Figura 2.11. Interfaz JTAG.

A través de esta interfaz de conexión se tiene acceso a las siguientes señales:

Pin Nº	Señal	Pin Nº	Señal
1	TMS	2	TRST-
3	TDI	4	GND
5	PD (+5V)	6	NO PIN
7	TDO	8	GND
9	TCK-RET	10	GND
11	TCK	12	GND
13	EMU0	14	EMU1

Tabla 2.11. Pines de la interfaz JTAG.

- ***Conector de expansión (P6)*:**

Con este conector de 50 pines se deja accesible tanto el bus de datos como el de direcciones del DSP, junto con las señales de control del mismo. De esta forma se puede añadir más memoria o periféricos externos al DSP.

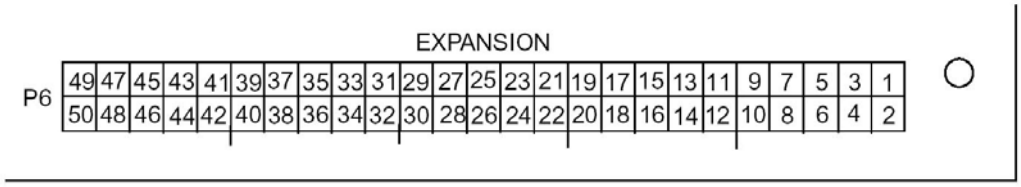


Figura 2.12. Conector de expansión P6.

Pin Nº	Señal	Pin Nº	Señal	Pin Nº	Señal
1	VCC	19	A0	37	PS-
2	VCC	20	A1	38	DS-
3	D0	21	A2	39	READY
4	D1	22	A3	40	IS-
5	D2	23	A4	41	R/W-
6	D3	24	A5	42	STRB-
7	D4	25	A6	43	WE-
8	D5	26	A7	44	RD-
9	D6	27	A8	45	BR-
10	D7	28	A9	46	NMI-
11	D8	29	A10	47	RS-
12	D9	30	A11	48	RESERVADO
13	D10	31	A12	49	GND
14	D11	32	A13	50	GND
15	D12	33	A14		
16	D13	34	A15		
17	D14	35	GND		
18	D15	36	GND		

Tabla 2.12. Pines del conector de expansión P6.

2.3.2. Mapa de memoria del Starter Kit:

Al ser una placa de pruebas y evaluación, el Starter Kit DSK F243 dispone de un mapa de memoria limitado en comparación con el anteriormente mostrado del procesador TMS320F243. Ello se debe en parte a que en un principio esta placa estaba diseñada para emplear aplicaciones de depuración de programas a través del puerto serie que requerían de la implementación en fábrica de un código monitor de depuración en la memoria FLASH del DSP. Esto hace que haya zonas de memoria a las que no se pueda acceder, ya sea por estar localizado ese programa monitor, o por ser zonas reservadas por el mismo para sus operaciones. De esta forma, la capacidad de memoria capaz de direccionar el DSP queda bastante limitada. Si se comparan los mapas de memoria de las figuras 2.4 y 2.9 se comprueban las siguientes variaciones:

- **Memoria de programas:**
    - Las direcciones dedicadas a memoria externa en el TMS320F243 eran 55K words entre las direcciones 2000h y FDFFh. Ahora quedan reducidas al sector entre las direcciones 8000h y BFFFh (16K).
    - El bloque de memoria B0 aparece como duplicado como B0'. Esto no tiene ningún efecto, ya que en el mapa original del procesador (figura 2.4) también lo estaba aunque se indicaba como reservado si CNF=1.

- **Memoria de datos:**
    - En esta sección la duplicidad del bloque B0 no tiene consecuencias al entrar dentro de lo que antes era zona reservada. Tampoco produce problemas la duplicidad del bloque B1 por el mismo motivo.
    - Por el contrario las direcciones de memoria dedicadas a memoria externa se ven limitadas en la zona alta de memoria, en donde se disponían 32K en el TMS320F243 y ahora quedan reducidas a 16K entre las direcciones 8000h y BFFFh, bloque duplicado a partir de la dirección C000h.

El significado de la duplicidad de los bloques no es que se tengan los datos repetidos en zonas distintas de memoria, sino que acceder a dichas zonas tiene físicamente el mismo efecto que acceder a la zona a la que se duplica. Es decir, acceder a una dirección de memoria del bloque B0' (el duplicado) tiene el mismo efecto que si accediésemos directamente a la misma zona del bloque B0 (original). Ésta es una forma de reservar zonas de memoria que se quieren proteger por ser usadas por el programa monitor para depuración, pero que limitan bastante la capacidad de memoria de la tarjeta de iniciación.

En cuanto a la zona de memoria de I/O, ésta no sufre cambios respecto de lo que disponía el procesador. Ésta se encuentra totalmente disponible para el usuario para el desarrollo de aplicaciones con periféricos externos a la placa.

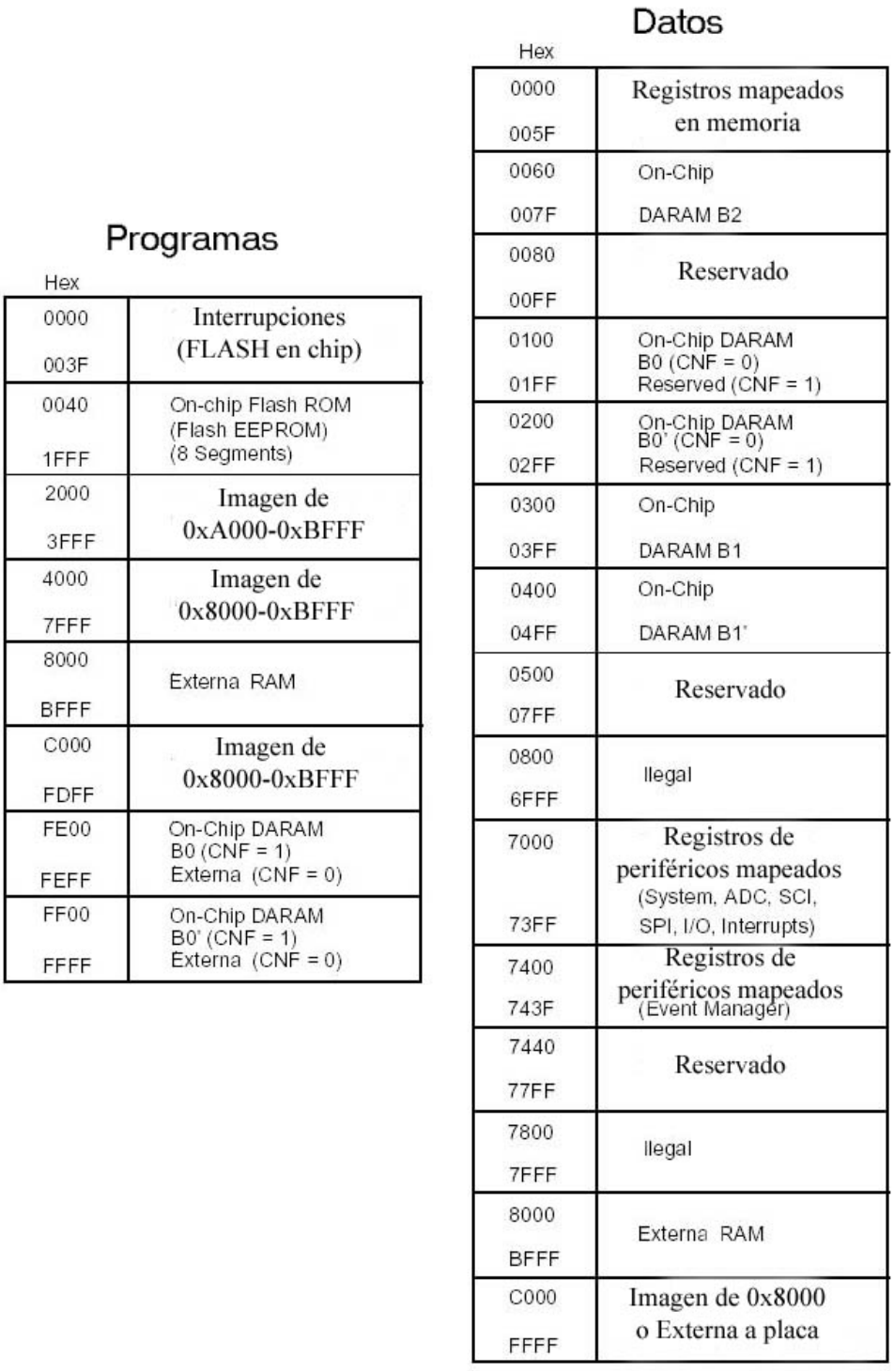


Figura 2.13. Mapa de memoria del Starter Kit.

2.4.INSTALACIÓN Y USO DEL SOFTWARE INCLUIDO EN EL STARTER KIT DSK F243:

En el kit de iniciación de Spectrum Digital se incluyen una serie de documentos y programas con los que elaborar aplicaciones para el DSP en un PC. Para empezar a trabajar con ellos, primero es necesario proceder a la conexión y configuración de la tarjeta y a la instalación del software necesario, procesos que se detallan a continuación.

2.4.1. Conexión al PC:

Para su instalación es necesario un PC al que conectarlo y con el que se realizará la programación y depuración del software. Para realizar dicha conexión, el DSK viene provisto de un cable serie de 9 pines para realizar la comunicación a través del puerto serie del PC (figura 2.14). Ésta forma de conexión tiene sus límites, debido a que la comunicación serie se realiza enviando la información a través de un único cable de los 9 pines que dispone el conector, siendo los restantes usados para alimentación y protocolo de la comunicación. Esto reduce la velocidad en la conexión limitando las prestaciones del sistema. Para evitar esto, el DSK trae un conector JTAG para la conexión al PC mediante un emulador JTAG opcional no incluido en el kit. Dicha conexión se realizaría con el PC a través su puerto paralelo (figura 2.15). Esta conexión es mucho más rápida, ya que los datos se envían de forma paralela por la mayoría de los 25 pines de los que consta dicho puerto, posibilitando el emulador JTAG la comunicación y depuración a través del PC casi en tiempo real.

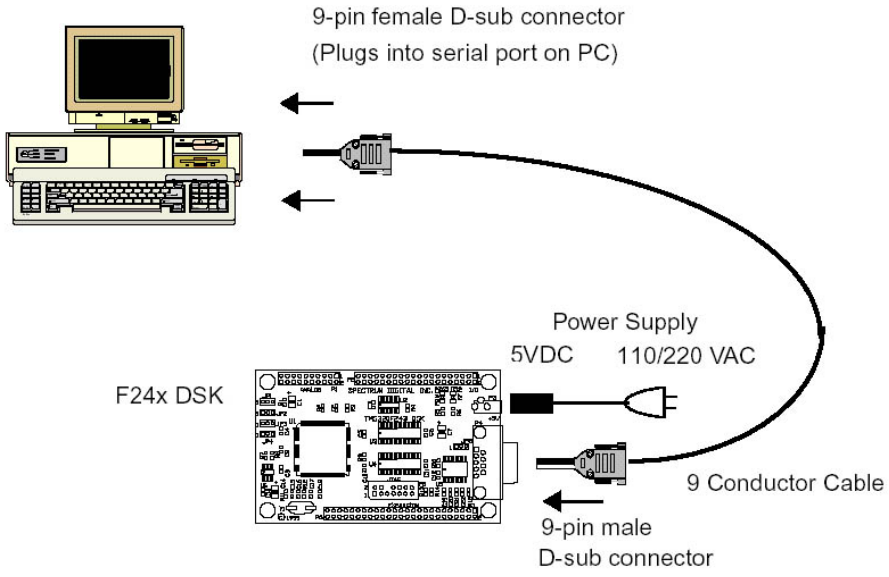


Figura 2.14. Conexión a través de puerto serie de PC.

Ambas opciones de conexión son perfectamente válidas para empezar el desarrollo de programas, pero el uso de una u otra determinará el software de desarrollo que se deberá usar. Así pues, si es usada la conexión a través del puerto serie, habrá de utilizarse el software Code Explorer, el cual viene incluido en el DSK, ya que únicamente funciona a través de dicho puerto de comunicación. Sin embargo, si se utiliza la conexión a través de un emulador JTAG, deberá usarse Code Composer, los cuales no van incluidos en el DSK.

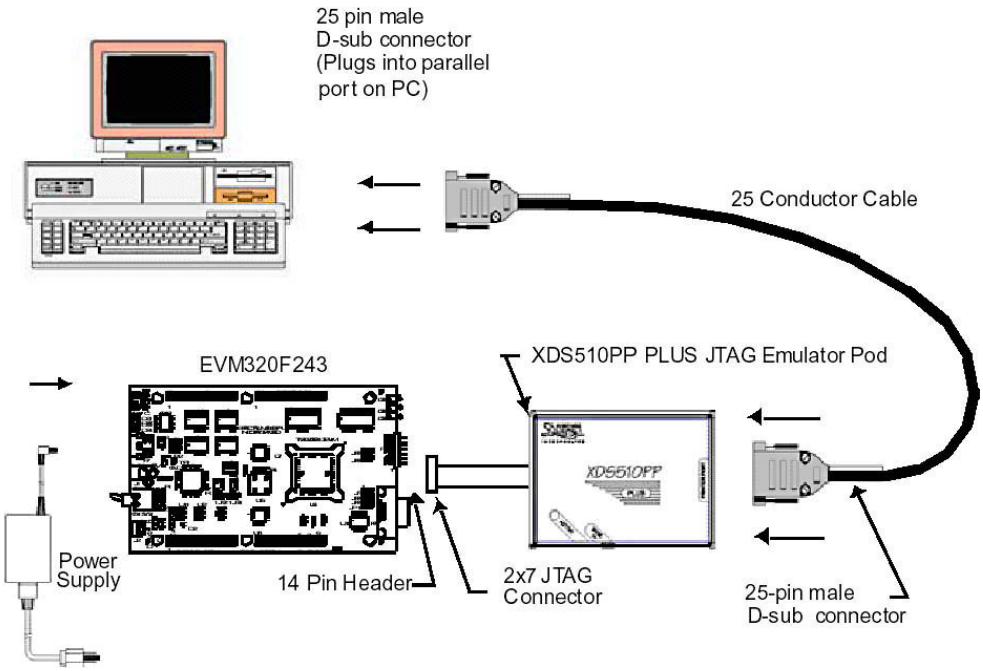


Figura 2.15. Conexión a PC mediante emulador JTAG.

2.4.2. Configuración de Jumpers:

Una vez decido el tipo de conexión a utilizar, deben configurarse los jumpers de que dispone la placa DSK para su correcto funcionamiento. Parte de dicha configuración también vendrá obligada por el tipo de conexión elegida, como se verá más adelante. Son 5 jumpers y su función se describe en la tabla 2.13 y su situación se da en la figura 2.16.

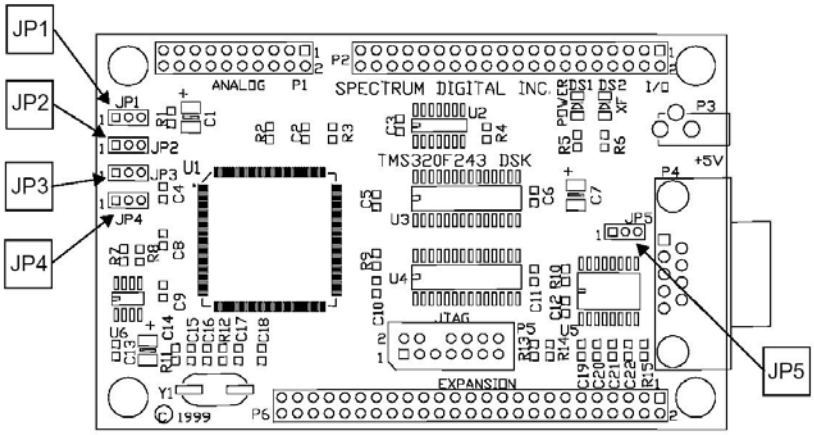


Figura 2.16. Localización de los Jumpers.

El jumper JP1 es el más importante, ya que elige el modo de funcionamiento del DSP. Esto es, que funcione en modo microcontrolador o modo microprocesador. La diferencia estriba en el uso que va ha hacer de la memoria de que dispone el chip. Si se elige modo microprocesador, la memoria FLASH del DSP F243 no será utilizada, y en su lugar se empleará la memoria RAM externa que incluye la placa. Mientras que si se usa el modo microcontrolador, sí se usará dicha memoria FLASH. El porqué de esto de debe a que para usar el DSK mediante el puerto serie del PC y utilizando el software

Code Explorer como depurador, es necesario que esté activa la memoria FLASH del DSP, ya que en ella reside un programa monitor para la depuración y que viene implementado de fábrica. Además, contiene la tabla de vectores de interrupción del DSP en la parte baja de la memoria, quedando parte de la FLASH disponible para guardar el programa desarrollado. Dicho programa monitor se encarga de gestionar la comunicación con el PC para la programación y depuración de la aplicación a desarrollar, y no es posible su eliminación, al igual que los vectores de interrupción. Por ello, para el desarrollo de aplicaciones más complejas que requieran del manejo de dichos vectores por el usuario según sean necesarios, existe la posibilidad de conexión a través del emulador JTAG junto con el software Code Composer, para lo cual se configura el DSP en modo microprocesador, ya que así se desactiva la FLASH, de forma que la zona de vectores de interrupción pasa a estar en la memoria RAM externa de la placa donde pueden ser tratados por el programador sin restricciones.

Jumper	Posición	Función
JP1	1-2	Modo Microprocesador
	2-3	Modo Microcontrolador
JP2	1-2	Vpp habilitado/Watchdog deshabilitado
	2-3	Vpp deshabilitado/Watchdog habilitado
JP3	1-2	VREFLO en placa
	2-3	VREFLO externa
JP4	1-2	VREFHI en placa
	2-3	VREFHI externa
JP5	1-2	SCIRXD en pin 4 de conector expansión
	2-3	SCIRXD en conector puerto serie

Tabla 2.13. Configuración de los Jumpers.

EL jumper JP2 sirve para habilitar y deshabilitar tanto la programación de la memoria FLASH, como el funcionamiento del temporizador de seguridad "watchdog" del DSP. Por otro lado la función de los jumpers JP3 y JP4 es indicar al DSP la procedencia del nivel bajo y del nivel alto de la tensión de referencia usada por el convertidor analógico-digital del chip. Estas tensiones pueden ser tomadas de la misma placa (usualmente 0 y 5 voltios) o introducidas a través de los conectores de expansión de la misma. Por último, para definir la procedencia de la señal SCIRXD de la comunicación serie se emplea el jumper JP5. Esta señal puede venir desde el puerto serie de 9 pines o desde el pin 4 del conector de expansión de Entrada/Salida de la tarjeta.

De todo esto se deduce que para el empleo de la conexión serie con Code Explorer la configuración de los jumpers debe ser con JP1 en la posición 2-3 y JP2 en la posición 1-2. De esta forma se habilita el uso de la memoria FLASH y se habilita su programación. Por otro lado, para el empleo de Code composer con el emulador JTAG debe configurarse JP1 en la posición 1-2 y deshabilitar así la memoria FLASH para usar la RAM externa. Es conveniente poner JP2 en 1-2 para deshabilitar el Watchdog. En ambos casos, los jumpers restantes se configurarán según se crea conveniente (normalmente en la posición 1-2, excepto JP5 que suele estar en 2-3).

Una vez configurados los jumpers según las necesidades, el proceso de instalación consiste en:

- apagar el ordenador
- conectar la tarjeta al puerto serie/puerto paralelo a través del JTAG
- conectar la placa a la alimentación de 5 voltios incluida en el kit
- encender el ordenador y ejecutar el programa adecuado.

2.4.3. Code explorer:

El starter kit dispone de tres discos de instalación en los que se contiene el software Code Explorer, ensamblador "sd24xasm" y manuales. Para su instalación basta con ejecutar el archivo "setup.exe" del primer disco e ir introduciendo el resto a medida que sean requeridos por la instalación. Se creará en el menú de inicio de Windows una carpeta llamada "Spectrum Digital DSK24x" la cual contiene el acceso directo a Code Explorer además de manuales de instalación y uso del software. Al ejecutar Code Explorer el usuario es requerido para indicar cual es el puerto serie que se está utilizando para comunicarse con el DSK (figura 2.17). Una vez elegido, el programa termina de arrancar mostrando la ventana "Dis-Assembly" en la que aparece el contenido actual de la memoria del DSP (figura 2.18).

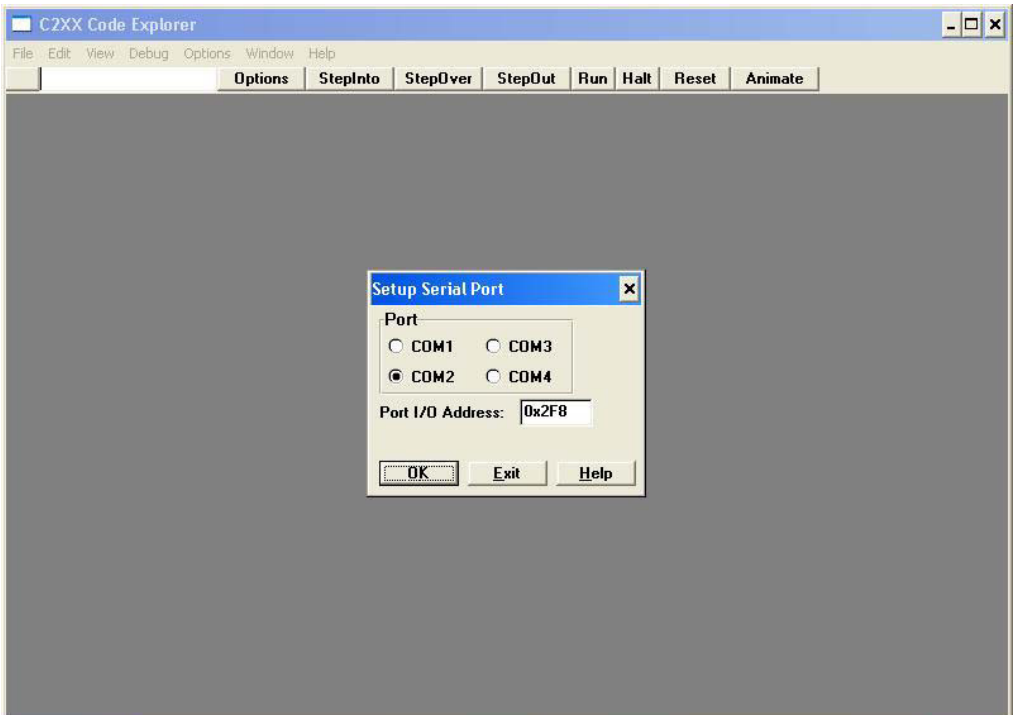


Figura 2.17. Inicio del Code Explorer.

Este software es sencillo debido a que únicamente necesita de un archivo fuente con el programa principal en lenguaje ensamblador para hacer una aplicación, aunque esta sencillez trae consigo uno de sus principales inconvenientes: la imposibilidad de trabajar con archivos en lenguaje C, mucho más sencillo que el ensamblador. Para programar el DSP es necesario pasar dicho archivo en lenguaje ensamblador

"archivo.asm" a código máquina que es el que entiende el chip. Para ello la instalación anterior ha guardado en la carpeta "c:\specdig\dsk24x\sd24xasm\" un programa ensamblador en formato MS-DOS, con el que basta poner en la línea de comandos "sd24xasm –l archivo.asm" para obtener el archivo en código máquina "archivo.dsk". Este archivo se introduce en el DSP accediendo al menú "File" de la barra superior del programa y seleccionado la opción "Load Program". Esto abre una ventana en la que seleccionamos el "archivo.dsk" que es cargado automáticamente en el DSP. En la ventana de código "Dis-Assembly" se muestra ahora el contenido de la memoria con el programa cargado y marcada con una línea amarilla la línea de código por la que va el programa. Para correr el programa cargado basta con pulsar el botón "Run" de la barra superior de Code Explorer y para pararlo pulsando "Halt".

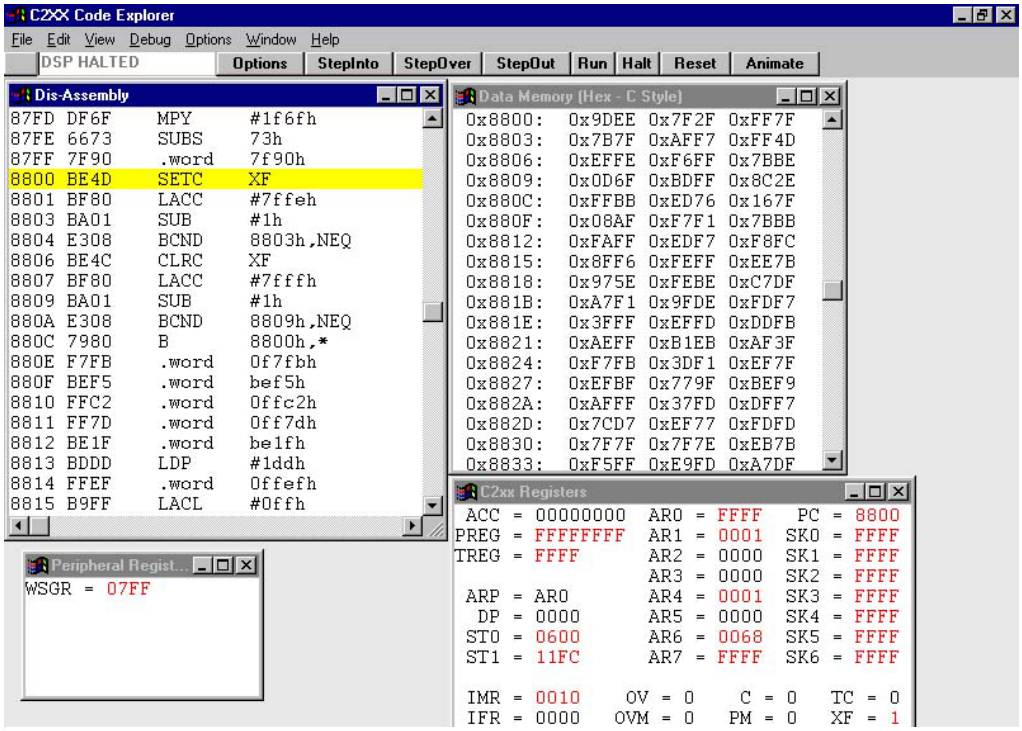


Figura 2.18. Visualización en Code Explorer.

Code Explorer dispone de una serie de herramientas y opciones muy útiles para la depuración de un código (figura 2.18). Las principales son:

- Ventana del código cargado en memoria "Dis-Assembly": En dicha ventana se muestra el código programado tanto en formato ensamblador como en código máquina, y realzada en amarillo la línea actual de código.
- Posibilidad de ejecución del código paso a paso: Para ello en la ventana superior del programa están los botones "StepInto", "StepOver" y "StepOut", siendo la primera utilizada para ejecutar el programa paso a paso, mientras que las dos últimas se usan para saltarse rutinas del código o salir de ellas.
- Introducción de "Breakpoints" en el código: Disponible en el menú "Debug" de la barra superior. Se colocan en la línea deseada, la cual queda resaltada por una barra rosa, de forma que al correr el programa, cada vez que se pase por ese punto, se detiene la ejecución del mismo.

- Ventanas de visualización de la memoria, registros de la CPU y registros de los periféricos: Estas opciones están disponibles en el menú "view" de la barra superior del Code Explorer.
- Ayuda sobre registros e instrucciones en el menú "Help".

Este programa dispone de algunas herramientas más, pero se dejarán para posteriores explicaciones en Code Composer, ya que existen también en éste y son más útiles en él debido a la mejora asociada a la emulación JTAG, al igual que ocurre con las opciones aquí descritas.

2.5.OTROS DE KITS DE DESARROLLO:

A continuación se describen someramente dos más modelos de kits de desarrollo para DSP´s disponibles en el mercado a modo de comparativa con el Stater Kit, destacando sus características más importantes a la hora de su aplicación en sistemas de control digital de motores. Uno de ellos está basado en el mismo procesador que el kit de inciación y otro en uno más avanzado: el TMS320LF2407.

2.5.1. TMS320F243 Módulo de Evaluación (EVM):

Al igual que la tarjeta DSK, éste módulo de evaluación también esta basado en el DSP TMS320F243 de Texas Instruments y es comercializado por Spectrum Digital. Tiene como finalidad aportar a los diseñadores de nuevas aplicaciones un hardware sólido sobre el que comenzar el desarrollo de dichas ideas. Para ello viene provisto de mejores prestaciones que el caso anterior, entre las que caben destacar el mayor tamaño de memoria externa para datos y programas (128K words) además de incluir 4 conectores de expansión como interfaz externa a la hora de evaluar circuitos desarrollados. También dispone en la placa de un convertidor digital-analógico de 4 canales y de un conector de interfaz CAN. Además se incluye en este kit (Figura 2.15):

- Emulador JTAG XDS510PP PLUS para puerto paralelo.
- Cable para puerto paralelo de 25 pines.
- Alimentador de tensión a 5 voltios.
- CD-ROM con Code Composer para la serie C2xxx.

Los principales bloques que componen esta tarjeta de evaluación son los siguientes (figura 2.19):

- Memoria externa.
- Interfaz digital-analógica.
- Puerto serie (P6).
- Interfaz CAN (P7).
- Conectores de expansión (P1, P2, P3, P4).
- Interfaz JTAG (P5).
- Interruptores de configuración de acción de los leds.

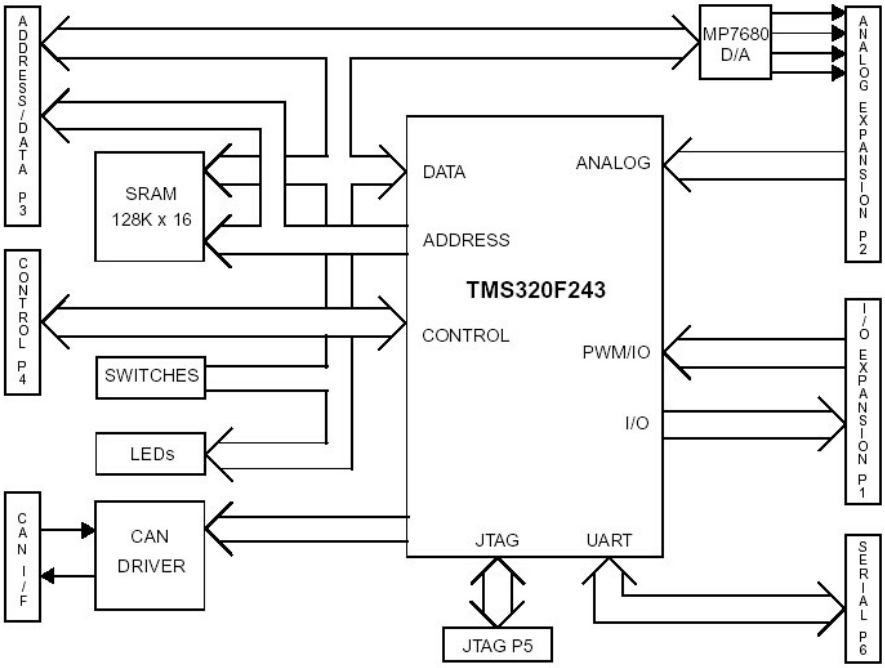


Figura 2.19. Bloques principales del módulo de evaluación.

2.5.1.1.Interfaz externa:

A través de los conectores de expansión se puede conectar la tarjeta a un circuito externo diseñado para una aplicación específica, evitando así la necesidad de tener que diseñar una tarjeta microprocesadora desde cero (figura 2.20).

- ***Conector de expansión de Entrada / salida (P1)*:**
  Este conector de 34 pines da las señales que recibe y genera el DSP. Entre ellas quedan accesibles las señales PWM, las cuales pueden ser usadas para control digital de motores actuando sobre inversores, rectificadores, etc. Al igual que antes, se tiene acceso a las señales de los puertos serie SCI y SPI además de otras señales generadas por el gestor de eventos (CAP/QEP). En este caso las señales de la interfaz CAN no están disponibles en este conector, ya que la placa lleva un conector específico a parte para dicho módulo. La disposición de las señales en este conector difiere de la distribución existente en la placa DSK, ya que se han ordenado siguiendo otro criterio de funcionalidad de las señales.

- ***Conector de expansión analógico (P2)*:**
  Con él tenemos acceso a las entradas para la conversión analógico-digital al igual que en la placa DSK, pero además se accede a las salidas de la conversión digital-analógica que contiene la placa.

- ***Conector de expansión de direcciones y datos (P3)*:**
  Aquí se tienen accesibles tanto el bus de datos como el de direcciones. No se accede a las señales de control.

- ***Conector de expansión de control (P4)*:**
  A diferencia del modelo DSK, las señales de control del bus de datos y direcciones, además de las del módulo CAN, son accesibles en este conector adicional de expansión específico para ellas.

- ***Interfaz JTAG (P5)*:**
  Conector JTAG estándar de 14 pines para interfaces emuladoras utilizadas en los procesos de depuración y diseño de aplicaciones para DSP´s.

- ***Conector serie RS-232 (P6)*:**
  Interfaz para la comunicación del dispositivo con un PC a través del puerto serie asíncrono de 9 pines.

- ***Interfaz CAN (P7)*:**
  Conector específico para el protocolo CAN. Con el se dota a la placa de una interfaz serie adicional de alta velocidad para su conexión a un bus CAN en un entorno de control distribuido.

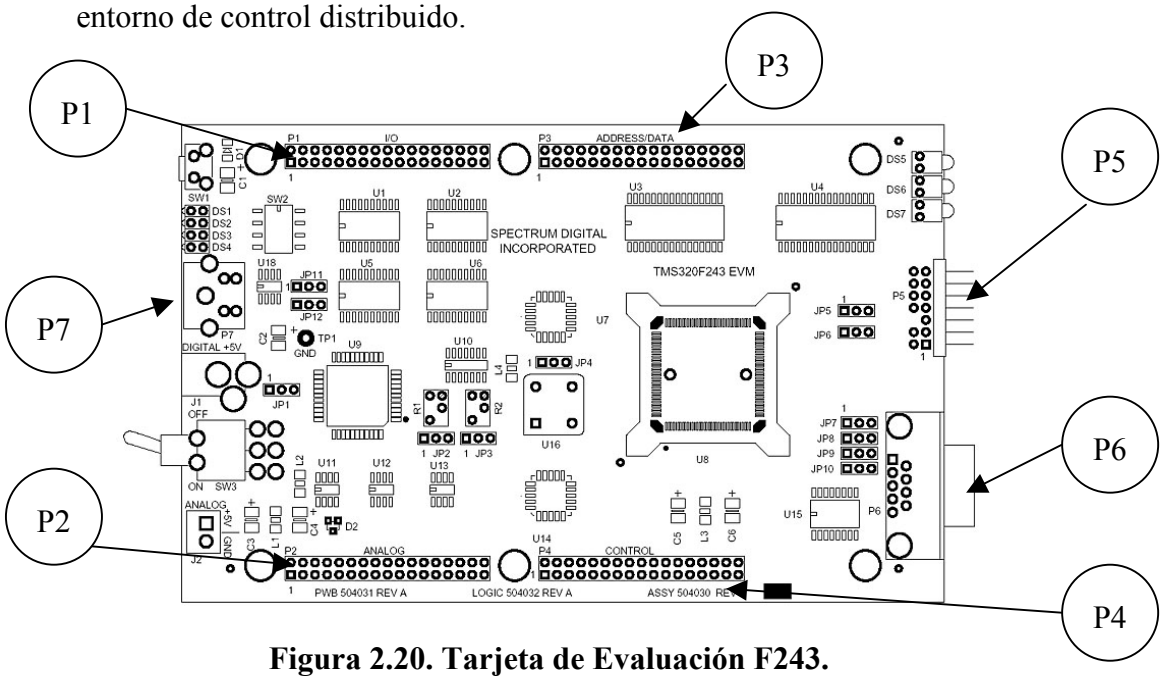


Figura 2.20. Tarjeta de Evaluación F243.

2.5.2. eZdsp LF2407:

Esta tarjeta de desarrollo, es un sistema basado en el DSP TMS320LF2407 que permite al microprocesador comunicarse con el exterior a través de cuatro conectores (JTAG y 3 conectores de expansión) y aumentar su memoria disponible para los datos y programas. La tarjeta puede ser programada a través del puerto paralelo ya que trae integrado un controlador JTAG en la placa, o mediante un JTAG externo. Se suministra con el kit (figura 2.21) un software de desarrollo y depuración de programas ejecutable desde un PC, permitiendo la programación en lenguaje ensamblador. A diferencia de las placas basadas en el F243, este kit emplea un procesador de mayor velocidad y potencia, además de disponer los mismos periféricos que las anteriores pero con mayores y mejores prestaciones. En general su funcionamiento es similar a las otras dos, exceptuando el conversor analógico-digital que funciona de manera distinta al disponer de múltiples opciones de configuración.

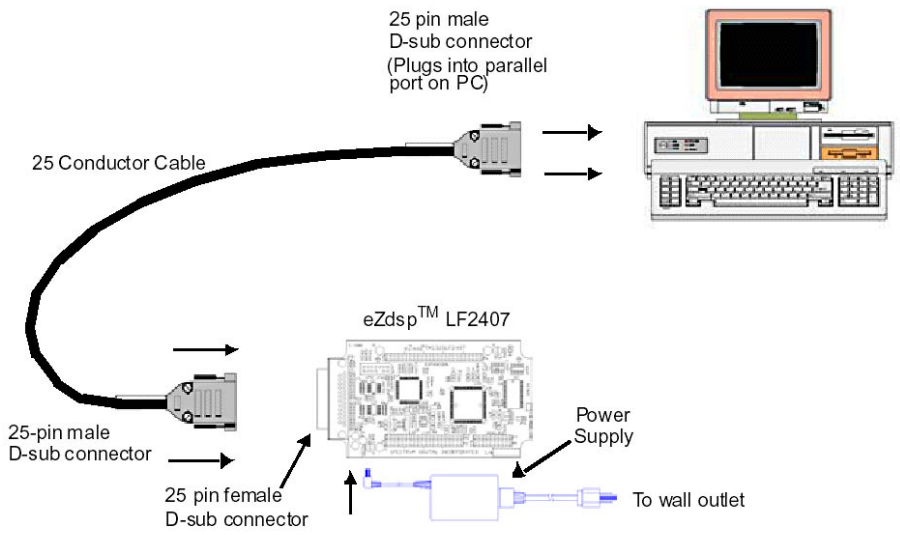


Figura 2.21. Instalación de la tarjeta eZdsp LF2407.

Las características principales de esta tarjeta son:

- Microprocesador DSP TMS320LF2407.
- Frecuencia de instrucción 30MHz.
- 64k words de memoria RAM en tarjeta para datos (32k) y programas (32k).
- Reloj externo de 7.3728MHz.
- Tres conectores de expansión: Analógico, E/S digital, Expansión.
- Controlador JTAG para depuración IEEE 1149.1.
- Conector para emulación a través de una Interfaz JTAG
- Alimentación con adaptador de red a 5V.
- Cable de comunicación paralelo de 25 pines.
- CD-ROM con Software Code Composer para serie C2xxx, códigos de ejemplos y manuales.
- Compatible con etapas de potencia para el control digital de motores (DMC1500).

A estas característica se le suman las disponibles por el microprocesador empleado TMS320LF2407:

- 30 MIPS (Millones de instrucciones en punto fijo por segundo).
- 2.5K words de memoria RAM.
- 32k words de memoria flash EEPROM.
- Interfaz de memoria externa de 16bits.
- 16 canales de convertidores A/D de 10 bits,
- 500 ns de tiempo de conversión.
- Módulo CAN (Controller Area Network).
- 4 Timers.
- 2 puertos serie.
- 2 Gestores de Eventos optimizados para el control de motores.
- 16 canales PWM.

La tarjeta eZdsp LF2407 contiene 32k words de memoria FLASH incluida en el chip y 32k words de memoria RAM, de los 64 K words que dispone la placa, para la implementación de programas. El resto de los 64K words (32K words) se emplean como memoria para datos. La tarjeta está diseñada con el fin de cargar programas en memoria RAM para su depuración.

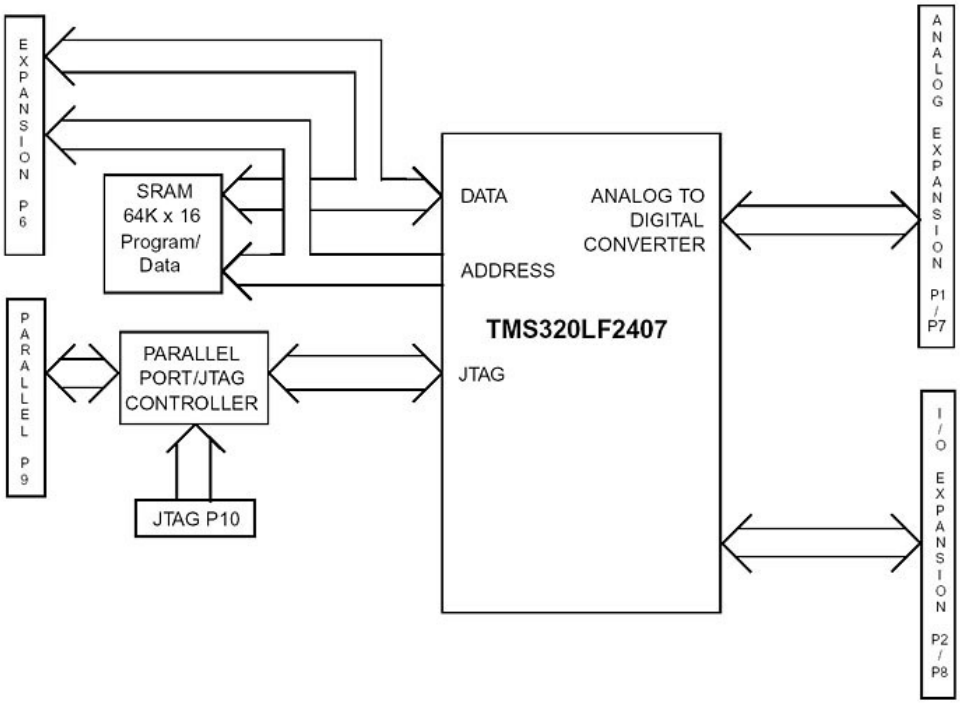


Figura 2.22. Diagrama de bloques de la tarjeta eZdsp LF2407.

Las principales vías de comunicación de la tarjeta con el exterior para el desarrollo de sistemas son las siguientes (figura 2.22):

- 2 conectores de expansión analógica (P1, P7).
- 2 conectores de expansión para entrada / salida (P2, P8).
- Conector de expansión externa (P6).
- Puerto paralelo con controlador de interfaz JTAG (P9).
- Interfaz JTAG para emulación externa (P10).

2.5.2.1.Interfaz externa:

La tarjeta eZdsp LF2407 dispone de diversos tipos de conectores para la implementación y depuración de código, además de posibilitar su conexión a circuitos externos desarrollados para la aplicación que se esté diseñando (figura 2.23). La mayoría de estos conectores son exactamente iguales en distribución de señales con los de la placa DSK, por lo que son totalmente compatibles en interfaz externa.

- ***Conectores de expansión analógicos(P1 y P7)*:**
  Esta tarjeta dispone de un conector de 20 pines (P1 equivalente al del DSK. Contiene las 8 primeras señales del convertidor A/D) y otro de 10 (P7) a través de los cuales quedan accesibles las 16 entradas para la conversión analógico-digital de las que dispone el DSP.

- ***Conectores de expansión de Entrada/salida (P2 y P8)*:**
  Mediante un conector de 40 pines (P2 muy parecido en distribución de señales del DSK) y otro de 20 (P8) se tiene acceso a las señales que genera y recibe el DSP, como son las señales de los puertos serie SPI y SCI, entradas QEP, comunicación por el bus CAN, las 16 señales PWM, etc.

- ***Conector de expansión el DSP (P6)*:**
  Con este conector de 50 pines (exactamente el mismo que el del DSK) se tiene la posibilidad de aumentar el tamaño de memoria disponible en el sistema al dejar accesibles las señales del bus de datos y direcciones, además de sus señales de control.

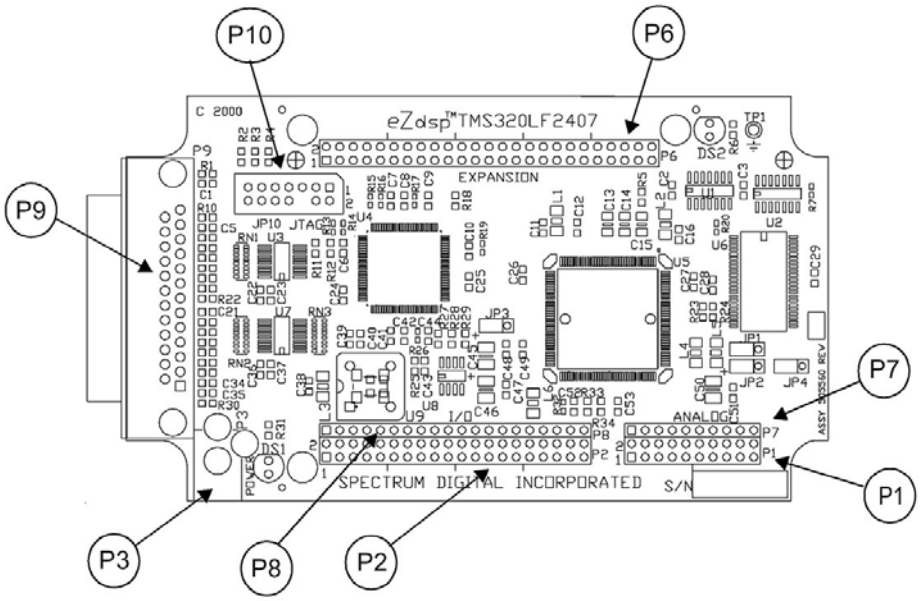


Figura 2.23. eZdsp LF2407: Conectores de expansión, JTAG y puerto paralelo.

- ***Puerto paralelo/interfaz JTAG(P9)*:**
  Este dispositivo dispone de un puerto paralelo para la comunicación con un PC, a través el cual se tiene acceso directo a la interfaz JTAG integrada en la placa para procesos de depuración.

- ***Interfaz JTAG (P10)*:**
  También trae incluida en la placa un conector JTAG estándar de 14 pines para la conexión de emuladores externos en procesos de depuración de código.

CAPÍTULO 3: SOFTWARE DE DESARROLLO DE APLICACIONES

3.1. INTRODUCCIÓN

Los DSP´s son dispositivos bastante complejos de entender, ya que en su funcionamiento hay multitud de variables a tener en cuenta. Es por ello que los fabricantes han puesto a disposición de los desarrolladores una serie de herramientas de trabajo que facilitaran su uso. Dichas herramientas de desarrollo han ido evolucionando a media que se extendía el uso de DSP´s en el mercado, llegando al punto de apenas necesitar grandes conocimientos para poder desarrollar pequeñas aplicaciones. Ello es debido a que han evolucionado a interfaces gráficas fácilmente entendibles por el usuario habitual de un ordenador, además de integrar en un solo programa todas las aplicaciones que eran necesarias de forma individual anteriormente. De esta forma, algo que antes era complicado de visualizar, ahora se hace de forma sencilla sólo añadiendo ventanas o con cliks de ratón. Así, los desarrolladores disponen de nuevas herramientas de programación, simulación, depuración, etc. que les permiten, en un relativo corto periodo de tiempo, poner a disposición de los usuarios aplicaciones más avanzadas y de forma más rápida. Ejemplos de estos programas son: Code Explorer, Code Composer y VisSim Embedded Controls Developer for TI C2000. Los dos primeros son facilitados por Texas Instruments para el desarrollo de aplicaciones en sus DSP´s TMS320. Básicamente, el primero de ellos es un sencillo programador y depurador de aplicaciones, que ha evolucionado y se ha convertido en un entorno integrado de desarrollo (IDE) conocido como Code Composer en el cual se incluyen, entre otras mejoras, el compilador de lenguaje C, el ensamblador y el enlazador (linker) necesarios para crear el código para el DSP, los cuales eran aplicaciones de MS-DOS individuales que no incluía Code Explorer, además de incluir la posibilidad de depuración en tiempo real. Actualmente, y debido al desarrollo de DSP´s más avanzados, Code Composer ha sido mejorado por Texas Instruments añadiéndole mayores prestaciones dando lugar al llamado Code Composer Studio. Por otra parte, existen en el mercado herramientas gráficas de simulación de sistemas como el desarrollado por Visual Solutions llamado VisSim Embedded Controls Developer for TI C2000, con el cual se pueden simular aplicaciones en DSP´s a través de diagramas de bloques de forma sencilla. Dicho programa viene dotado de herramientas para la simulación de sistemas de control de motores que es la finalidad de este proyecto. En los apartados siguientes serán analizados dichos programas, aunque no se entrará en detenimiento con Code Explorer debido a su antigüedad y limitaciones. Para ello se ha hecho uso de la tarjeta de iniciación (Starter Kit) F243 DSK de Spectrum Digital, la cual fue analizada anteriormente. Esta tarjeta es la más sencilla del mercado para el aprendizaje en DSP´s, lo cual no impide que disponga de todos los periféricos habituales en los DSP´s de su familia, siendo además compatible con los tres programas de desarrollo que serán analizados en este documento. Así, éste capítulo servirá de base para las descripciones de los distintos periféricos que componen el TMS320F243 que se darán capítulos posteriores, en los que se propondrán ejemplos de programas para el entendimiento de cada uno de ellos, por lo que serán necesarios unos mínimos conocimientos de las herramientas de programación y depuración que aquí se detallan.

3.2. SOFTWARE DE DESARROLLO CODE COMPOSER:

Con el avance que ha sufrido el mercado de los DSP´s los fabricantes han tenido que actualizar sus productos para el desarrollo de aplicaciones, ya que la competencia exige de una rápida fase de diseño y desarrollo con el fin de poner lo antes posible un producto en el mercado. Esto ha derivado en productos para el desarrollo más intuitivos, además de englobar en un mismo software todas las aplicaciones individuales que eran necesarias para la creación de códigos y su depuración. Por otro lado, el perfeccionamiento de la emulación y depuración a través de la interfaz JTAG ha posibilitado que dichos procesos puedan ser monitoreados casi en tiempo real, pudiendo así comprobar el correcto funcionamiento del programa desarrollado sin apenas interferir en dicho proceso. De esta forma, el programa Code Explorer ha evolucionado al entorno integrado de desarrollo llamado Code Composer de Texas Instruments, debido a la adición de nuevas y mejores prestaciones para el desarrollo de aplicaciones.

Éste software no viene incluido en el starter kit F243 DSK, por lo que se ha utilizado la versión de Code Composer para la serie C2000 incluido en el modulo de evaluación F243 EVM de Spectrum Digital, además del emulador JTAG XDS510 PP PLUS de dicho kit. Basándonos en estos componentes, se van a desarrollar en los próximos apartados las principales características y funcionalidades de dicho software. Para ello serán necesarias algunas explicaciones sobre la estructura y funcionamiento de algunos de los componentes de la tarjeta de desarrollo, además de introducir terminologías basadas en lenguajes de programación en ensamblador y C.

3.2.1. Instalación y configuración de Code Composer:

El software de desarrollo Code Composer se suministra en un CD con el módulo de evaluación F243 EVM de Spectrum Digital. Esta versión únicamente es válida para los DSP´s de la serie C2000. Para su instalación basta con introducir el CD y ejecutar la opción de instalar del menú de inicio y seguir las indicaciones del programa de instalación. Una vez finalizada la instalación, se procede a su configuración para indicarle las características de nuestro sistema DSP. Como se vio anteriormente, este software sólo funciona a través de la interfaz JTAG, por lo que se ha utilizado el emulador XDS510 PP PLUS incluido en el módulo de evaluación. Por tanto, hay que configurar el programa para dicha conexión. Esto se hace a través del icono creado en la instalación llamado “Setup CC C2000”, el cual abre una ventana en la que se muestran las configuraciones disponibles (figura 3.1). En esta ventana se selecciona la configuración “C2xx PP Emulator” y se pulsa sobre “Add to system configuration” y a continuación sobre “Close”. De esta forma se ha añadido a nuestro sistema dicha configuración, y así lo indica la ventana “Code Composer Setup” en la sección “System Configuration” en la que aparece el sistema “sdgo2xx (Spectrum Digital)” (figura 3.2). Antes de salir del Setup hay que guardar esta configuración con la opción “Save” del menú “File”.

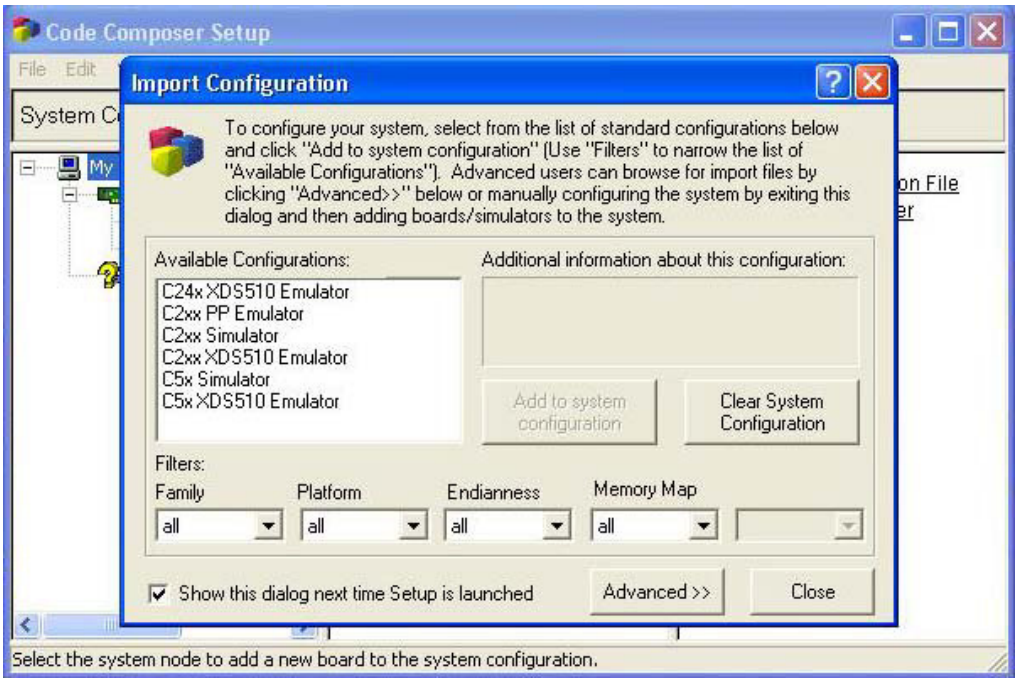


Figura 3.1. Configuración del Code Composer.

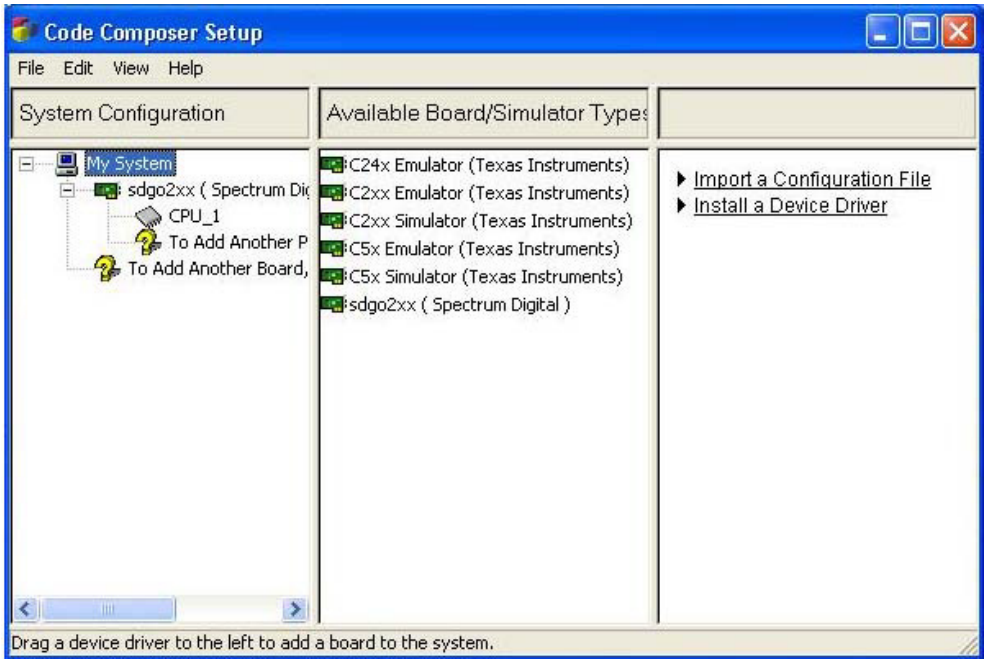


Figura 3.2. Configuración del sistema.

Puede darse el caso de que la configuración "C2xx PP Emulator" no aparezca entre las opciones. Ello se debe a que Code Composer no dispone de los drivers del emulador XDS510 PP PLUS de Spectrum Digital, con lo que se hace necesaria su instalación: "XDS510PP/SPI510/SPI515/XDS510PP_PLUS for TMS320C2xx, TMS320C24xx and TMS320C5x, Win95/WinNT Drivers for GoDsp Code Composer version 4.1" cuyo archivo se denomina "SetupCC2400_v193". Al ejecutarlo crea en la carpeta de instalación de Code Composer una carpeta llamada "specdig" en la que instala un programa de configuración del puerto JTAG denominado "sdconfig" (figura 3.3).

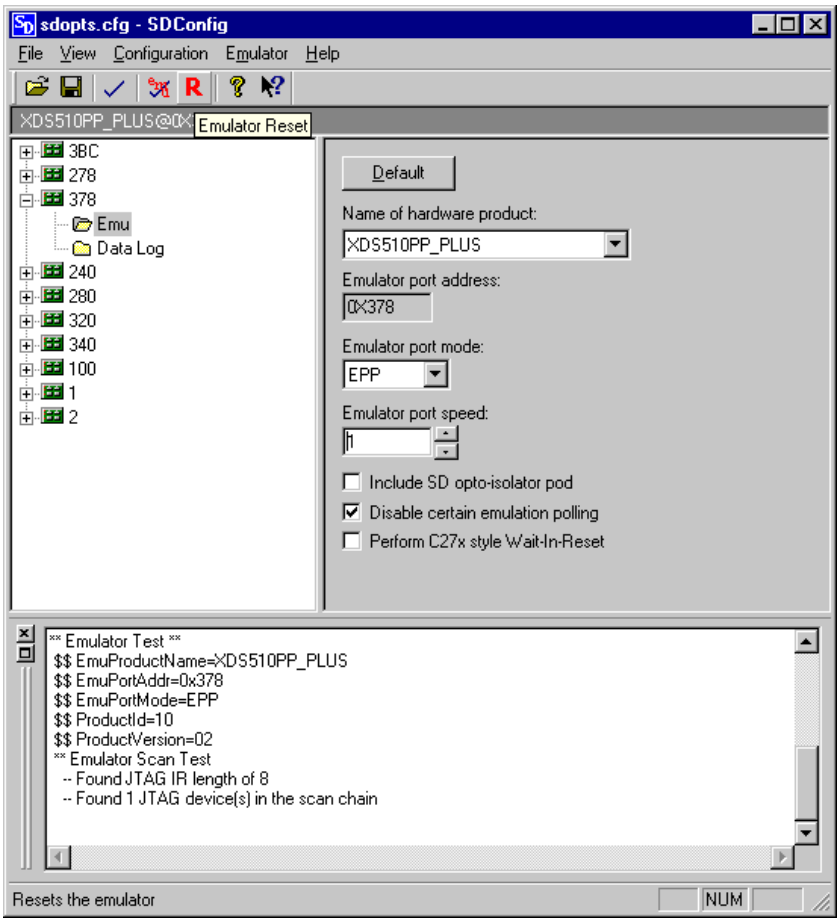


Figura 3.3. Configuración del puerto JTAG.

Este programa crea al configuración "C2xx PP Emulator" que se necesita con solo decir el modelo de JTAG, el puerto paralelo que se quiere usar y su modo EPP/ECP. Además permite hacer un test de comunicación y un reset del dispositivo (opciones disponibles en el menú "Emulator" de la barra superior). Para todo ello el sistema debe estar conectado al Pc mediante el JTAG. Antes de cerrar el programa hay que grabar esta configuración para que aparezca en el Setup del Code Composer.

Una vez configurado el sistema, se ejecuta Code Composer a través de su icono en el escritorio de Windows. Si todo es correcto, el programa arranca sin dar ningún error y abriendo la ventana "Dis-Assembly" con el contenido de la memoria de la placa en ese momento.

3.2.2. Comenzando con Code Composer:

Para trabajar con Code Composer hay que tener conectado todo el sistema DSP al Pc mediante el emulador JTAG y enchufado a la alimentación de 5 voltios. De otra forma el programa dará errores al arrancar y no nos permitirá usar algunas opciones. Una vez conectado el sistema se ejecuta el programa (figura 3.4) y aparecerá la ventana "Dis-Assembly", la cual muestra el contenido del DSP en ese momento que serán restos del último programa con el que se trabajó.

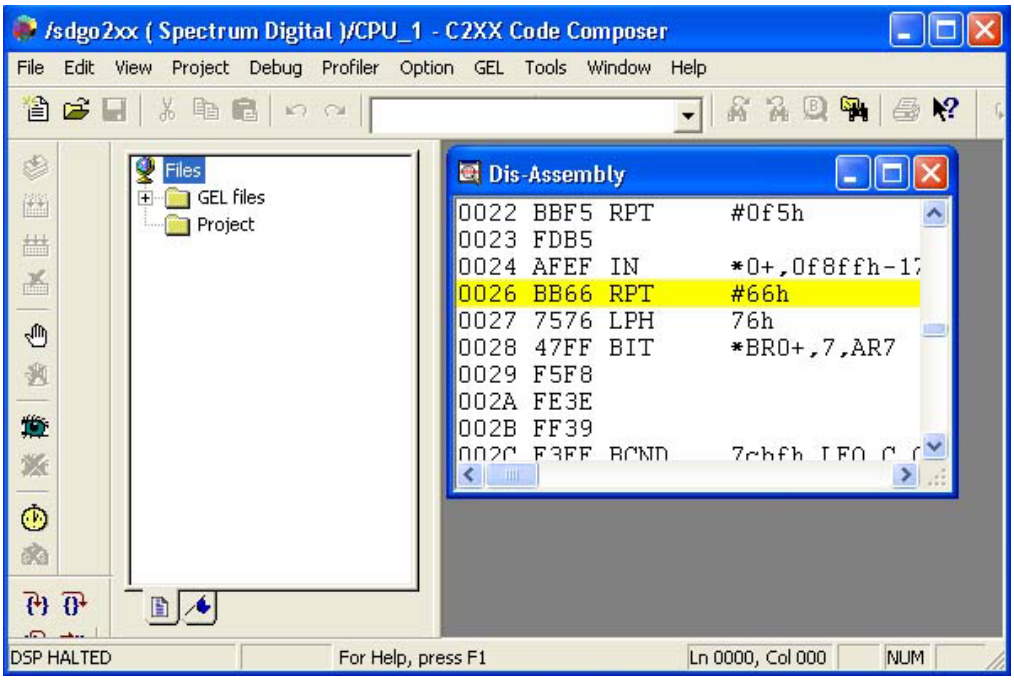


Figura 3.4. Code Composer.

El entorno del programa es el habitual de los programas que trabajan bajo Windows. En la parte superior se dispone la barra de menús a través de la cual se puede acceder a las distintas funciones del programa. Justo debajo de ella y en el lateral izquierdo de la ventana se encuentran algunos de los iconos de acceso rápido a dichas funciones. Por otro lado, la zona de trabajo se divide en dos zonas:

- **La ventana de proyectos:** Esta ventana se encuentra en la parte izquierda del programa, y en ella se visualiza bajo una estructura en árbol los distintos archivos que componen el proyecto que se esté llevando a cabo.

- **Zona de visualización:** Abarca el resto de la zona de trabajo y sirve para abrir las distintas ventanas de visualización de registros, memoria, archivos de programa, Dis-Assembly, etc. de las que dispone Code Composer.

Por último, en la parte inferior del programa se encuentra la barra de estado, en la cual se muestra el estado de funcionamiento del DSP. Por ejemplo, si el DSP está corriendo un programa se visualiza el texto “DSP RUNNING”, o si está inactivo se muestra “DSP HALTED”.

La mejor forma de empezar a familiarizarse con este entorno es abriendo un proyecto ya desarrollado. Para ello basta con pulsar sobre “Project” en la barra superior de menús y elegir la opción “Open” (figura 3.5). Esto abrirá una ventana de navegación para localizar el archivo maestro del proyecto, cuyo nombre es del tipo proyecto.mak”. Como ejemplo abriremos un proyecto que contiene los archivos típicos necesarios para la creación de un proyecto. Code Composer está preparado para trabajar con archivos escritos en lenguaje C y en ensamblador. Por ello los proyectos pueden realizarse con archivos escritos en distintos lenguajes de programación según sea necesario.

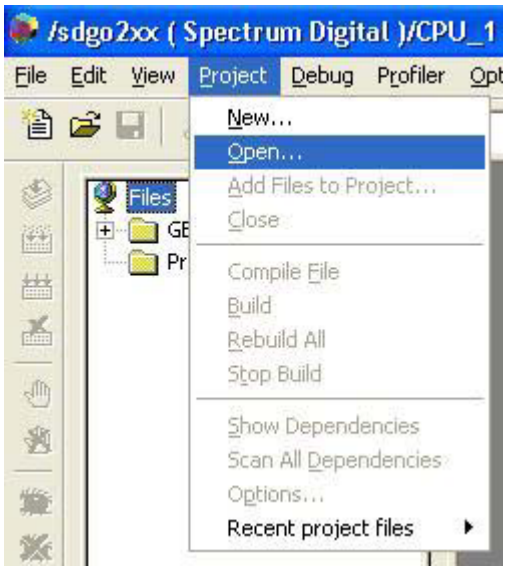


Figura 3.5. Abriendo proyecto.

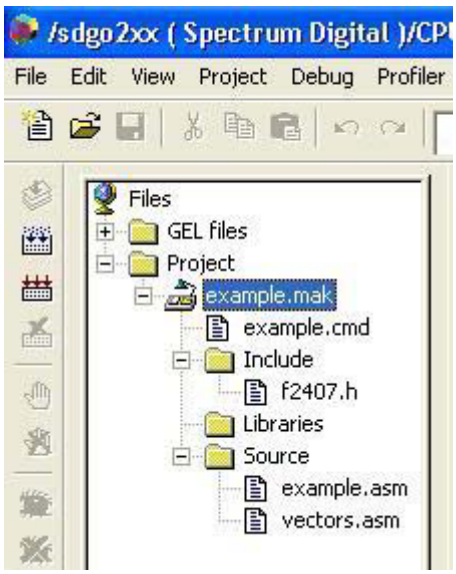


Figura 3.6. Proyecto abierto.

Una vez abierto el proyecto, éste se muestra en la ventana de proyectos como una estructura de carpetas en árbol, cuyo contenido puede expandirse pulsando sobre el signo + adjunto (figura 3.6). En dicha ventana se muestran los archivos incluidos en el proyecto distribuidos en carpetas según su función. Dichas funciones se refieren al contenido de los archivos. Esta división facilita el trabajo a la hora de realizar algún cambio en el proyecto, ya que sólo sería necesario modificar la parte correspondiente de uno o varios archivos en lugar de trastocar todo el proyecto. Por ejemplo, para cambiar la definición de un registro sólo sería necesario modificar el archivo correspondiente sin tener que modificar el programa principal. Por ello se distribuyen en 4 tipos:

- ***Include:*** Son archivos de extensión ".h" y en ellos se definen constantes, funciones, registros y parámetros internos del DSP para que sean incluidos en el programa principal. Pueden estar escritos en lenguaje C o ensamblador.

- ***Librerías:*** Aquí se incluyen archivos librerías en los que se definen distintas funciones (incluidas matemáticas) optimizadas para el lenguaje de los DSP´s. Se diferencian por su extensión ".lib".

- ***Source:*** Son archivos fuentes de programas en los cuales está escrito el programa principal o auxiliares a éste, como son la gestión de los vectores de interrupciones o la inicialización del DSP. Por ello suelen escribirse en lenguaje C (archivo.c) o ensamblador (archivo.asm).

- ***Archivo de comandos para el Linker:*** Es un único archivo de extensión ".cmd", en el cual se da información al enlazador de programas para la reserva de las distintas zonas de memoria que necesitará la aplicación. Opcionalmente puede incluirse información para configurar las distintas opciones del Linker a la hora de enlazar un proyecto, como son: el nombre del archivo de salida, los archivos a enlazar, etc.

Para ver el contenido o modificar cualquiera de estos archivos sólo es necesario hacer dos pulsaciones de ratón sobre el nombre del archivo, abriéndose una ventana en la zona de trabajo en la que se verá el archivo, y en distintos colores los comentarios, las constantes definidas y los comandos de programación del lenguaje utilizado, facilitando así su entendimiento (figura 3.7). Si es un archivo fuente en lenguaje C, pulsando sobre ella con el botón derecho o a través del menú "View" se puede visualizar en modo mixto tanto el código C como el ensamblador.

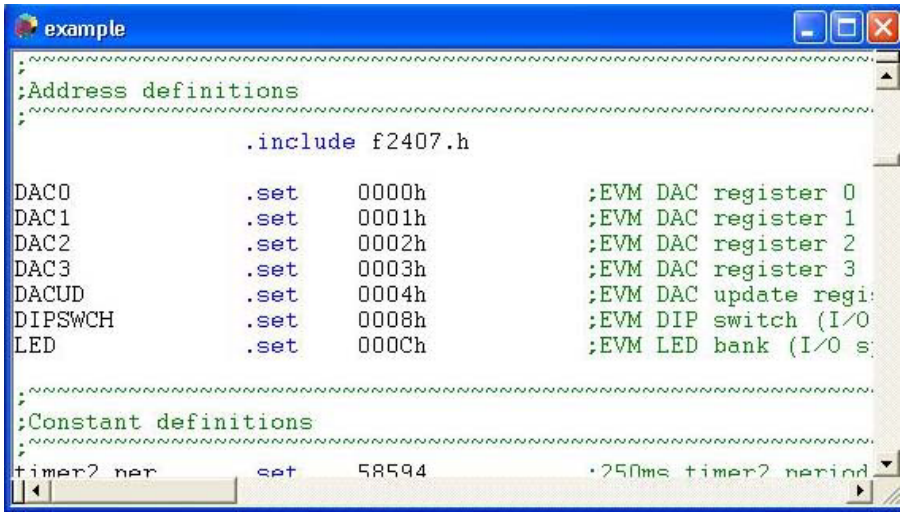


Figura 3.7. Contenido archivo example.asm.

Una vez abierto el proyecto y revisado, el siguiente paso es generar el archivo que será cargado en la memoria del DSP para ejecutarlo. Para ello hay que configurar las distintas opciones de las que disponen las herramientas generadoras de código de Code Composer. Estas herramientas son únicamente el compilador, ensamblador y enlazador de código que están integrados en este paquete software. Se accede a sus opciones a través del menú "Project" y eligiendo "options..." (figura 3.5), lo cual abre una ventana con una pestaña por cada uno de los componentes (figura 3.8). En cada una de ellas se pueden configurar sus opciones de una manera fácil y sencilla, ya que dispone de una ventana superior en la que se recogen las distintas opciones que están configuradas para ser utilizadas en la línea de comandos. Algunas de estas opciones serán comentadas más adelante, aunque las que tienen por defecto son validas.

Hecho esto, el archivo a cargar en el DSP se genera pulsando la opción "Build" del menú "Project" (figura 3.5). De inmediato se abre en la parte inferior del programa una ventana (figura 3.9) en la que se muestran las distintas etapas de la generación del código, como son: el compilado de archivos en lenguaje C, el ensamblado de archivos en ensamblador y, finalmente, el enlazado de todo el proyecto. Es aquí donde se utilizan las opciones configuradas anteriormente para cada herramienta, y que equivalen a la ejecución de cada una de ellas mediante una línea de comandos en MS-DOS, como sería por ejemplo: "dspa example.asm example.obj -gs -v2xx" utilizado para ensamblar el archivo "example.asm" (para que no existan problemas, destacar que Code Composer no admite que los proyectos se encuentren en directorios con nombres que incluyan espacios (por ejemplo "mi proyecto") o con una ruta de directorios excesivamente larga, ya que ello provoca errores en el proceso de enlazado al no ser capaz de encontrar los archivos ".obj" generados).

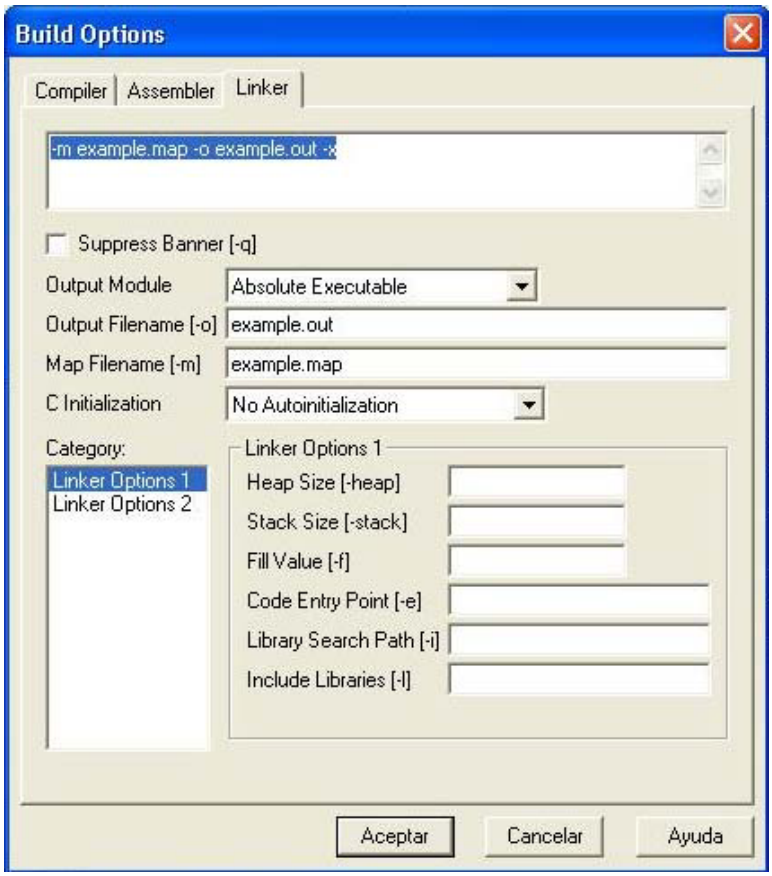


Figura 3.8. Opciones de proyecto.

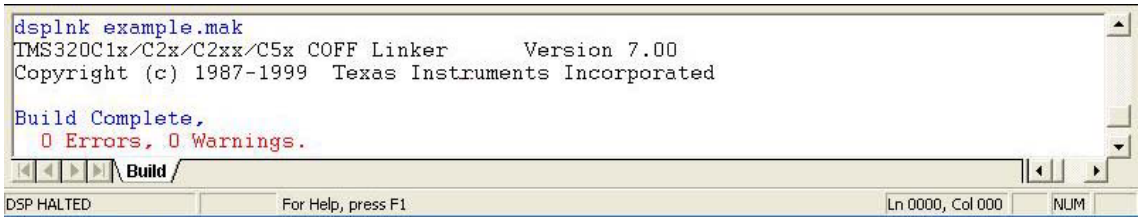
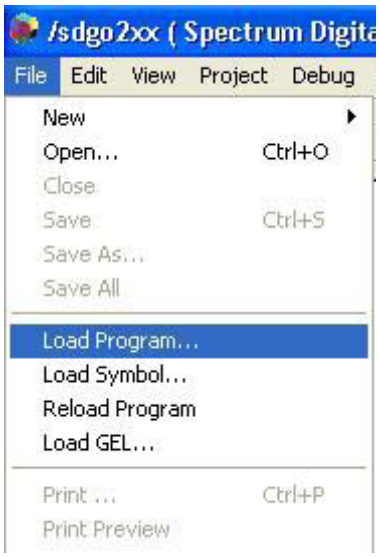


Figura 3.9. Estado del proceso.

De esta forma se tiene un seguimiento del proceso, dando además información sobre los errores encontrados en el mismo. Básicamente éste consiste en pasar los archivos fuentes a código objeto ".obj", que son los entendibles por la CPU del procesador, pero de manera que no se asignan valores ni direcciones absolutas a parámetros que no han sido definidos en dichos archivos, pero sí lo está en alguno de los restantes que forman el proyecto. Estos valores quedan a la espera del siguiente proceso que es el enlazado de todos los archivos, y en el cual se completa la comprobación y asociación de dichos parámetros en todo el proyecto, dando lugar al archivo ".out" que se cargará en el DSP. Si se produjese algún error de tipo gramatical en el código, aparecería en la ventana de estado un mensaje en color rojo indicando la línea donde está el error y la posible causa. Si se pulsa dos veces sobre esa información, automáticamente se abriría en una ventana el archivo afectado y con el cursor en la línea a corregir, facilitando así la corrección. Otro tipo de información que puede aparecer es sobre las opciones utilizadas u omitidas en el proceso de construcción del archivo, indicando la opción necesaria para la correcta finalización del proyecto.



Tras este proceso, se genera en la carpeta donde están guardados los archivos del proyecto la aplicación bajo la extensión ".out". Para cargar este archivo en el DSP se utiliza la opción "Load Program..." del menú "File" del Code Composer (figura 3.10), la cual abre una ventana de navegación para localizar dicho archivo. Este proceso dura pocos segundos, finalizado el cual se abre una ventana (figura 3.11) con el archivo que comienza la ejecución del proceso. Dicho archivo suele ser uno de los archivos fuentes o la ventana "Dis-Assembly". En cualquier caso, queda resaltada mediante una franja amarilla la primera línea de código que se ejecutaría si se diese la orden de correr el programa.

Figura 3.10. Carga del programa.

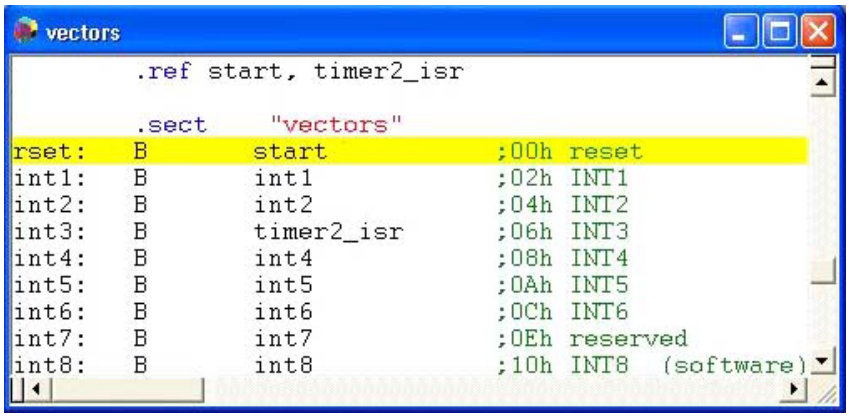
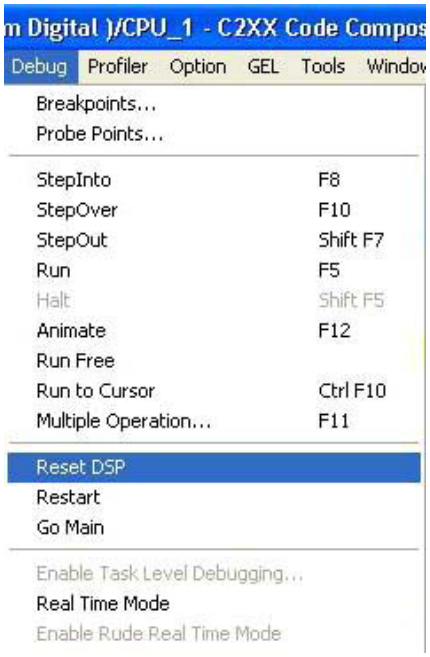


Figura 3.11. Comienzo de ejecución.



En éste punto se tiene ya el DSP cargado con la aplicación desarrollada. Por tanto el siguiente paso consiste en hacer correr la aplicación, pero antes es conveniente resetear el dispositivo, ya que es desconocido el estado en el que se encuentran los distintos registros del DSP, y así son llevados al estado por defecto. Esto se consigue mediante la opción "Reset DSP" del menú "Debug" (figura 3.12). Una vez hecho esto se mostrará en la ventana "Dis-Assembly" el código existente en el DSP con la primera línea del programa "0000" resaltada en amarillo. De esta forma el programa empezará con la iniciación de los registros del DSP. Para correr el programa se pulsa F5 o la opción "Run" del menú "Debug" (figura 3.12), con lo que se mostrará en la barra inferior de estado DSP RUNNING, indicando que el proceso está en marcha.

Figura 3. 12. Menú Debug.

Para parar la ejecución del proceso se utiliza la opción "Halt" del menú "Debug", siendo la información del estado DSP HALTED.

Existe la posibilidad de ejecutar la aplicación paso a paso, esto es, ejecutando una a una las líneas de código que forman el programa. Esta opción es muy útil para comprobar y seguir la secuencia con la que se ejecuta la aplicación de forma visible para el usuario. Ello se consigue pulsando la tecla F8, que ejecuta la línea actual de código y pasa a la siguiente que queda en espera de una nueva pulsación, o a través de la opción "StepInto" del menú "Debug" (figura 3.12). En este mismo sentido están las opciones "StepOver" y "StepOut", cuya función es saltarse o salirse de algún tipo de rutina del programa, como sería un bucle del que pulsando F8 se tardaría mucho en salir. Éstas funciones también permiten que el usuario pueda ver la evolución de los registros o la memoria del sistema mediante las ventanas de visualización. Estas ventanas se activan mediante las opciones "Memory", "CPU Register" y "Status Register" del menú "View" de la barra superior (figura 3.13).

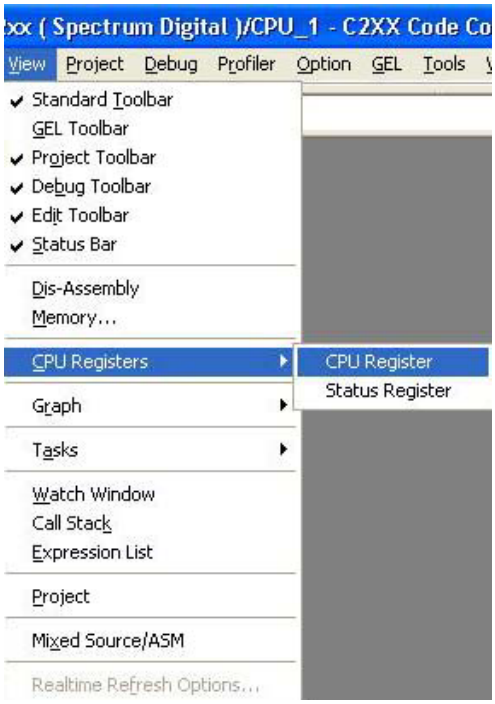


Figura 3.13. Menú View.

Al elegir la opción de visualizar la memoria, aparece una ventana en la que habrá de indicarse si queremos ver la memoria de datos, programa o I/O, y en su caso a partir de qué dirección se desea visualizar. Estas opciones (figura 3.14) son muy adecuadas para la depuración del programa, ya que, junto con la ejecución paso a paso, se pueden ver la evolución de los registros y la memoria durante el proceso de ejecución de la aplicación.

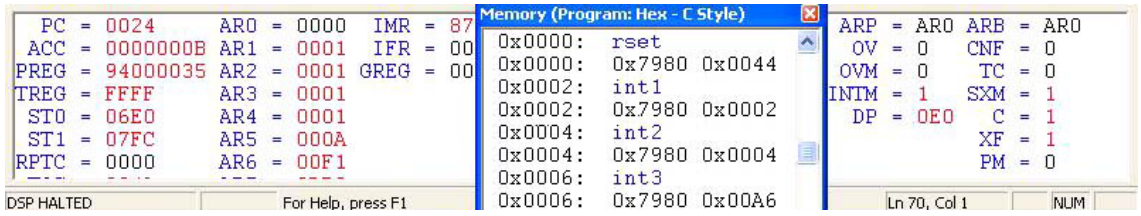


Figura 3.14. Registros y memoria.

Otra herramienta de visualización importante es la ventana "Watch Window", la cual se activa mediante la correspondiente opción en el menú "View" (figura 3.13), apareciendo en la parte inferior del programa junto con los registros. En ella se pueden introducir variables usadas en la programación del código fuente o registros de los distintos periféricos del sistema DSP para ser monitorizados. Posteriormente será explicada de forma más extensa.

Todos los parámetros visualizados en estas ventanas tienen la posibilidad de ser editados para cambiar su valor, simplemente con una doble pulsación sobre el elemento correspondiente.

3.2.3. Creando un proyecto con Code Composer:

Para crear un proyecto en Code Composer se accede al menú "Project" (figura 3.5) y se selecciona la opción "New...". Esto abrirá una ventana de navegación para indicarle el nombre del proyecto y el directorio donde guardarlo, el cual es conveniente que sea el mismo donde están los archivos que vamos a utilizar. Lo siguiente es añadir los archivos necesarios para el proyecto. Esto se consigue expandiendo en la ventana proyectos el árbol para que aparezca nuestro archivo ".mak", y pulsando con el botón derecho del ratón para elegir la opción "Add Files..." del menú emergente (figura 3.15). También es posible eligiendo la misma opción por el menú "Project" de la barra superior. Esta acción abre una ventana de navegación en la que se debe elegir el tipo de archivos que se quiere añadir (figura 3.16).

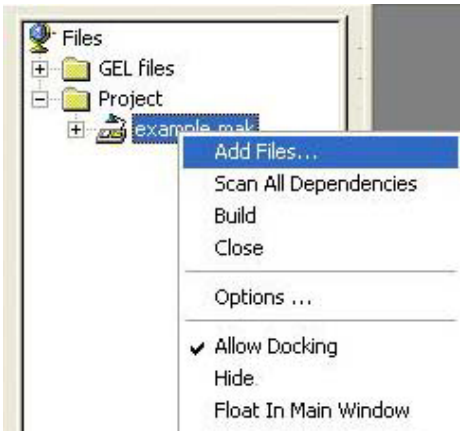


Figura 3.15. Añadir archivos.

El tipo de archivos a elegir se corresponde con las divisiones existentes en la ventana "Proyectos" explicados anteriormente (excepto los "Include"). Para incluirlos en el proyecto se van seleccionando por dichos tipos de forma sistemática mediante la opción "Add Files...". Los archivos de la carpeta "Include" se añaden de forma automática al pulsar con el botón derecho del ratón sobre el nombre del proyecto y eligiendo la opción "Scan All Dependencies" del menú emergente (figura 3.15). Estos archivos deben estar situados en la carpeta del proyecto para poder emplearlos. Una vez añadidos todos los archivos, el proceso de edición y construcción de la aplicación es el mismo que el descrito anteriormente. Si se quiere eliminar un archivo basta con señalarlo y pulsar la tecla "Supr".

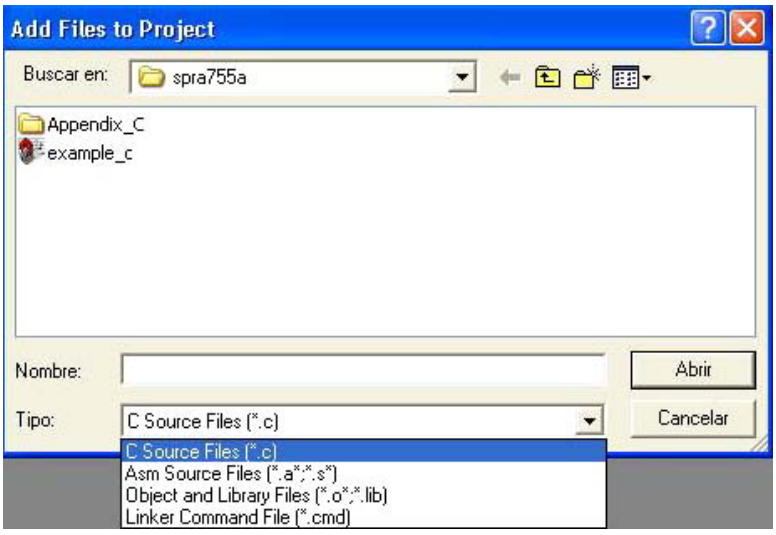


Figura 3.16. Tipo de archivos.

3.2.3.1. Archivos del proyecto:

El número de archivos necesarios para un proyecto depende del tipo de lenguaje de programación que se haya utilizado, además de la descomposición en archivos que crea necesario el programador para facilitar posibles actualizaciones. La influencia del lenguaje se debe a la necesidad de incluir, o no, librerías de definición de funciones matemáticas, secuencia de inicialización del DSP, etc. Esto influye en el caso de que el programa fuente principal esté escrito en lenguaje C, ya que en este caso es necesario inicializar el entorno de ejecución del programa bajo lenguaje C. Esto consiste en la inicialización de la pila de datos y de las variables globales definidas en el programa, para a continuación comenzar con la ejecución del programa principal. De todo ello se encarga el módulo "boot.asm" que está definido en la librería "rts2xx.lib", incluida en Code Composer en el directorio "C:\tic2xx\c2000\cgtools\lib", y que deberá ser incluida en el proyecto en desarrollo siempre que se emplee el lenguaje C. El punto de comienzo de dicho módulo se define como la función "c_int0". Esta función debe ser asociada en la construcción del proyecto al vector de interrupción 0 del DSP, de forma que sea el punto de inicio en la ejecución de la aplicación después de resetear el sistema, lo cual se define en un archivo normalmente llamado "vectors.asm" que será descrito más adelante. Una vez inicializado el entorno de ejecución para lenguaje C, éste mismo módulo se encarga de llamar al programa principal "main" de la aplicación para empezar con su ejecución. Éste módulo es incluido en el archivo final ".out" de forma automática por el enlazador cuando se utiliza la opción "-c" en su línea de comandos. Esta opción se configura en la pestaña "Linker" de la ventana "Build Options" (figura 3.8) a través del campo "C Initialization", en el que se elige la opción "ROM Autoinitialization", quedando así definido el punto de inicio de la aplicación en la función "c_int0". Bajo entorno C la documentación consultada recomienda emplear el módulo "Boot.asm" ya extraído de la librería e incluido en algunos ejemplos del fabricante , ya que se ha modificado para un mejor funcionamiento de la aplicación.

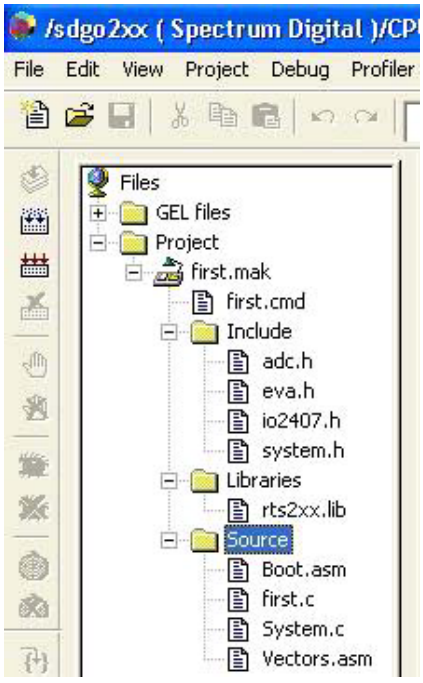


Figura 3.17. Proyecto.

Por otro lado, el proyecto puede estar dividido en multitud de archivos, cada uno de los cuales puede ser un subprograma, archivos de definición de registros, definición de funciones usadas por el programa principal, etc. Esto puede facilitar la tarea de modificar parámetros, ya que sólo se verían afectados pequeñas porciones de código. Por ello, en el proyecto pueden utilizarse varios archivos fuentes (Source ".asm" o ".c") o Includes (".h"), como se puede observar en la figura 3.17. En dicho ejemplo, el proyecto consta de varios archivos fuentes, cada uno de los cuales se encarga de una parte de la aplicación:

- Boot.asm: módulo extraído de la librería para arranque del DSP (recomendable).
- System.c: Define funciones de inicialización de los periféricos del sistema.
- Vectors.asm: Definición de la tabla de los vectores de interrupción.
- First.c: Programa principal de la aplicación.

De esta forma, puede hacerse en un archivo el programa principal "main" que haga uso de una serie de funciones, las cuales pueden estar definidas en archivos a parte. Así, en caso de necesitar alguna modificación, ésta sólo se haría en el archivo en el que se encuentre la función a modificar. Esto es de gran ayuda a la hora de depurar y mejorar la aplicación. Después, a la hora de construir el archivo final ".out", el enlazador se encargará de sustituir las llamadas a funciones desde el programa principal por el código de ejecución de dichas funciones si es que se encuentran definidas en otro archivo fuente.

- **Archivos de definición ".h":**

Por otro lado, se incluyen varios archivos ".h" en los cuales se definen funciones, constantes y registros del sistema. De esta forma, puede utilizarse un archivo por cada aplicación o periférico utilizado para definir sus registros. Para el ejemplo de la figura 3.17, se usan archivos para el convertidor Analógico-Digital (adc.h), los puertos de entrada/salida (io2407.h), el gestor de eventos (eva.h) y definición de funciones de iniciación de los periféricos del sistema (system.h). En los dedicados a periféricos, se asignan nombres o etiquetas a cada una de las direcciones asociadas a sus registros, de forma que cada vez que se quiera hacer uso de ellos basta con utilizar ese nombre en lugar de la dirección, facilitándose así la tarea del programador. Un ejemplo sería:

```
#define ADCCTRL1      *(volatile unsigned int *) 0x70A0
#define ADCCTRL2      *(volatile unsigned int *) 0x70A1
#define MAX_CONV      *(volatile unsigned int *) 0x70A2
```

De esta manera se definen punteros a las direcciones de cada registro del convertidor Analógico-Digital del DSP, de forma que se acceda a ellos más fácilmente utilizando un nombre simbólico, como sería ADCCTRL1 para el registro de control 1 de dicho convertidor, en vez de tener que recordar o buscar su dirección de memoria 0x70A0. Como puede apreciarse en el ejemplo estos archivos pueden estar escritos en lenguaje C, aunque también pueden estarlo en ensamblador:

```
ADCTRL1     .set 70A0h
ADCTRL2     .set 70A1h
MAX_CONV    .set 70A2h
```

El lenguaje a utilizar depende del archivo en el que se vayan a incluir, esto es, deben estar escritos en el mismo lenguaje que el utilizado en el archivo donde serán incluidos. Esto se debe a que a la hora de enlazar el proyecto dichos archivos serán insertados como código en los archivos que utilicen sus definiciones. Esta operación se indica utilizando la directiva "include" en el programa afectado. Por ejemplo, si el programa principal está escrito en lenguaje C y va a utilizar los registros del convertidor Analógico-Digital del DSP, al comienzo del código se incluiría la línea:

```
#include "adc.h"
```

Si este mismo programa está escrito en ensamblador la línea a insertar sería:

```
.include adc.h
```

Para que estos archivos aparezcan en la estructura en árbol del proyecto en la ventana "Proyecto" de Code Composer, únicamente hay que pulsar con el botón derecho del ratón sobre el nombre del archivo "proyecto.mak" en dicha ventana para elegir la opción "Scan All Dependencies" del menú emergente (figura 3.18). Esta opción analiza todos los archivos fuente incluidos en el proyecto en busca de directivas "include", presentando en el árbol del proyecto el resultado de la búsqueda en su categoría. Para que puedan ser editados y utilizados en la elaboración del proyecto, estos archivos deben estar en el mismo directorio que el resto de componentes del proyecto. De esta forma son fácilmente localizados por las herramientas de generación de código a la hora de completar el archivo final ".out". De dichas herramientas, el enlazador se encargará de cambiar en todo el programa las etiquetas y funciones definidas en los archivos ".h" por las direcciones a las que representan, que es lo que necesita realmente la CPU para ejecutar el código.

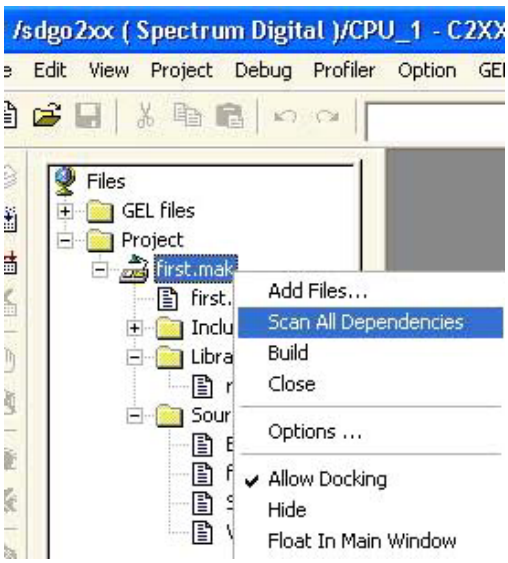


Figura 3.18. Añadir Include.

- **Tabla de vectores de interrupción:**

Un archivo a incluir obligatoriamente siempre es el denominado "vectors.asm", el cual define la tabla de vectores de las distintas interrupciones del DSP. Este archivo está escrito en ensamblador, indicando los nombres de las subrutinas asociadas a las distintas interrupciones. Un ejemplo sería:

```
    .ref    _bad_trap   ; illegal trap
      .ref        _c_int0     ; entry point to the code

      .sect "vectors"

RSVECT    B    _c_int0   ; PM 0  Reset Vector          1
INT1        B    _bad_trap    ; PM 2      Int level 1       4
INT2        B    _bad_trap    ; PM 4      Int level 2       5
INT3        B    _bad_trap    ; PM 6      Int level 3       6
INT4        B    _bad_trap    ; PM 8      Int level 4       7
INT5        B    _bad_trap    ; PM A      Int level 5       8
INT6        B    _bad_trap    ; PM C      Int level 6       9
RESERVED  B    _bad_trap  ; PM E  (Analysis Int)       10
SW_INT8     B    _bad_trap    ; PM 10     User S/W int      -
SW_INT9     B    _bad_trap    ; PM 12     User S/W int      -
.
.
.
```

Debe estar escrito en ensamblador, por lo que la primera columna de código es únicamente para situar etiquetas a las cuales hacer referencia en la aplicación. Dichas etiquetas pueden ir acompañadas opcionalmente del símbolo ":". A continuación se indica la directiva a seguir, en este caso "B" que significa "Branch", es decir, salto incondicional a la función indicada por el siguiente operando. Por ello, la línea:

```
RSVECT      B     _c_int0
```

indica que tras un reset del DSP se salte a la función "c_int0" descrita anteriormente. El símbolo "_" que le precede es para indicar que dicha función proviene de un archivo escrito en lenguaje C. En el caso de que el programa esté totalmente en ensamblador, RSVECT debe saltar al punto de comienzo de la aplicación, usualmente denominada "Start". De esta forma, las demás interrupciones del ejemplo saltarían a una función llamada "bad_trap", también definida en otro archivo en C. Para que todas las variables del archivo queden totalmente definidas se hace uso de la directiva ".ref":

```
.ref    _bad_trap   ; illegal trap
.ref        _c_int0     ; entry point to the code
```

indicando que las funciones "bad_trap" y "c_int0" ya han sido definidas completamente en otros archivos del proyecto.

Esta tabla de vectores será almacenada en la zona de memoria que se reserve para ello, lo cual se indica mediante la directiva ".sect". En el ejemplo, la línea ".sect "vectors" " indica que esta tabla debe guardarse en la sección "vectors" de la memoria. Esta sección junto con otras necesarias para la aplicación se definen mediante el archivo de comandos para el enlazador ".cmd".

- **Archivo de comandos del enlazador:**

Este archivo está formado por dos partes fundamentales:

- Definición de los bloques de memoria disponibles en el DSP (MEMORY).
- Localización de las distintas secciones en dichos bloques de memoria (SECTIONS).

aunque opcionalmente se puede incluir en su cabecera los distintos parámetros que el programador crea necesario utilizar con el enlazador.

Un ejemplo de archivo de comandos ".cmd" sería:

```
MEMORY
{
    PAGE 0:                                 /* Program Memory*/
      VECS:       origin = 00000h, length = 00040h
      FLASH:      origin = 00040h, length = 03FC0h
      EXTRAM:     origin = 04000h, length = 03FFFh
      B0:         origin = 0FF00h, length = 00100h   /*CNF=1*/

    PAGE 1:                                 /* Data Memory*/
      B0:          origin = 00200h, length = 00100h  /*CNF=0*/
      B1:          origin = 00300h, length = 00100h
      B2:          origin = 00060h, length = 00020h
      EXTRAM:      origin = 08000h, length = 03FFFh
}

SECTIONS
{
     vectors       :    > VECS      PAGE 0
      .text        :      > FLASH     PAGE 0
      .cinit       :      > FLASH     PAGE 0
      .switch      :      > FLASH     PAGE 0
      .const       :      > B1        PAGE 1
      .data        :      > EXTRAM    PAGE 1
      .bss         :      > B1        PAGE 1
      .stack       :      > EXTRAM    PAGE 1
      .sysmem      :      > B1        PAGE 1
}
```

Para definir los distintos bloques de los que consta la memoria de la tarjeta DSP, es necesario seguir las indicaciones que se hacen en la referencia técnica que acompaña al producto. En ella se muestra gráficamente las distintas zonas en las que se descompone la memoria del dispositivo, como se detalló en el capítulo anterior (apartado 2.3.2, figura 2.13), de las cuales dos son las principales: Memoria de programas y Memoria de Datos. A partir de esos datos se configura la sección de memoria en el archivo de comandos ".cmd", indicando la dirección de comienzo de cada bloque y la longitud que quiere reservarse para el mismo. Para diferenciar las dos zonas principales de memoria, se designa con "PAGE 0" a la memoria de programas y con "PAGE 1" la de datos. Por otro lado, cada bloque de memoria es designado por una etiqueta que suele corresponderse con la nomenclatura utilizada en la figura 2.13, en las que "FLASH" y "DARAM" (la cual se divide en tres bloques: B0, B1, B2) son los dos tipos de memoria interna que dispone el chip DSP, y "EXTRAM" se corresponde con la RAM externa incorporada en la placa DSK.

Hay que recordar, como se comentó en el apartado de instalación y configuración del DSK, que el jumper JP1 era necesario que estuviese en la posición 1-2 para poder operar con Code Composer, y que esto desactivaba la memoria FLASH interna del DSP pasando a ser mapeada esa zona de memoria a la memoria externa en la placa. Por ello, la zona baja de la memoria de programas se encuentra realmente en la memoria externa, aunque se seguirá denominando FLASH porque es obligatorio utilizarla para almacenar la tabla de vectores y el código ejecutable de la aplicación como se hace a la hora de programar una aplicación definitiva.

En la figura 2.13 se puede observar que en ambos bloques existen zonas de memoria ilegales o reservadas, además de bloques de memoria duplicados en distintas direcciones. Una particularidad sobre esto es el bloque de memoria RAM interna "B0", el cual puede ser memoria de programas o datos según el estado del bit "CNF" del registro de estado de la CPU. Si este bit es 1, el bloque B0 será usado como memoria de programas, quedando el bloque correspondiente B0 de la memoria de datos como zona reservada. Si el bit es 0, el bloque B0 se utilizará como memoria de datos y la zona que deja libre en la memoria de programas se asignará a memoria externa. Otra característica importante es que en la memoria de datos están mapeados los registros de configuración de los distintos periféricos del sistema a partir de la dirección 0x7000, y a partir de la 0x0000 los registros de configuración de las interrupciones.

La otra parte fundamental del archivo de comandos del enlazador ".cmd" es la de localización de las distintas secciones de la memoria. En ella se indica en cuál de los bloques definidos se va a reservar espacio para cada sección. Por ejemplo, en la sección "vectors" se incluirá la tabla de vectores definida en el archivo "vectors.asm", y esto será en la zona de memoria de programa reservada para tal efecto llamada "VECS", que comienza en la dirección 0000h y acaba en la 0040h. De igual forma se reserva espacio para las demás secciones. El uso de estas secciones es un método para la utilización más eficiente de la memoria. La función de algunas de las más comunes son:

- .text: Sección para guardar el código ejecutable del programa.
- .stack: Sección reservada a la pila (obligatorio para lenguaje C).
- .bss: Reserva espacio para variables no inicializadas.
- .data: Contendrá datos inicializados.
- .cinit, .const: Secciones necesarias bajo lenguaje C para el almacenamiento de tablas de datos iniciados, declaración e iniciación de variables y constantes estáticas o globales respectivamente.

Estas secciones se emplazan en la memoria mediante la sintaxis:

```
.text       :       > FLASH     PAGE 0
```

en la que se indica que la sección ".text" sea cargada en el bloque de memoria FLASH. Opcionalmente se indica la página (PAGE 0) para evitar errores si hay etiquetas de bloques con idéntico nombre. Algunas de estas secciones se utilizan para almacenar código o datos ya inicializados, como son: .text, .data, pero otras son simplemente para reservar espacio para inicializar nuevas variables a medida que sean necesarias, como son: .bss, .stack. En este caso la sintaxis anterior sólo reserva espacio en memoria para ello. Esta diferenciación en las secciones impone el tipo de localización en memoria que deben emplear algunas de ellas:

- En memoria de programas (PAGE 0): .text, .cinit, .switch (pueden localizarse tanto en memoria ROM/FLASH como en RAM).
- En memoria de datos (PAGE 1): .const (en ROM/FLASH o RAM), .bss, .stack, .sysmen (únicamente en RAM).

Por otro lado, de forma opcional, se puede incluir al principio del archivo ".cmd" las opciones que el programador crea necesarias para incluir en la línea de comandos a la hora de invocar al enlazador. Estas opciones también podían ser configuradas a través del menú opciones del proyecto en Code Composer. Un ejemplo podría ser:

```
-c                          /* ROM autoinitialization */
-x                         /* force rereading libraries */
-o example.out                    /* output file */
-m example.map                     /* map file */

/* files to be linked */

example.obj
vectors.obj
-l rts2xx.lib          /* Run Time Support  */
```

En él se muestran las opciones más comunes para enlazar un proyecto. De ellas, la opción "-c" es obligatoria siempre que se tenga el programa principal escrito en lenguaje C, ya que indica al enlazador que el punto de arranque del programa debe ser la función "c_int0" descrita anteriormente, además de usar convenciones especiales definidas para dicho lenguaje, como es inicializar las secciones .cinit, .sysmen, etc. Por su lado, la opción "-x" obliga a releer el código para comprobar que no hay referencias y asociaciones indefinidas. También puede indicarse el nombre de salida del fichero final ".out" mediante la opción "-o nombre.out". Para programadores avanzados puede ser interesante la obtención de un archivo mapa, en el cual se especifica dónde se han localizado las distintas funciones y variables en las secciones definidas en la memoria y las direcciones y valores finales que les han sido asignados. Ello se consigue mediante la opción "-m nombre.map". Por último pueden indicase los nombres de los archivos que deben ser enlazados. Dichos archivos son códigos objeto (.obj) resultados de la compilación de archivos fuente (Source), además de las librerías necesarias indicadas mediante la opción "-l". Esto último no suele incluirse en el archivo, ya que el enlazador lo comprueba automáticamente cuando es invocado.

3.2.3.2. Ejemplo de proyecto:

A continuación se describe un ejemplo de proyecto el cual puede ser tomado como base para el diseño de otras aplicaciones en el starter kit F243. Este ejemplo contiene los archivos mínimos necesarios y recomendables para la elaboración de una aplicación sencilla, la cual puede ser ampliada mediante la adición de código y de los archivos que el programador crea necesario. Se ha tomado esta estructura como base debido a que es más sencilla la programación de una aplicación mediante el lenguaje C, por lo que se han incluido todos los archivos y opciones necesarias para trabajar en dicho entorno descritas anteriormente. El código completo se detalla en el Anexo 1.

- **Descripción de la aplicación:**

Este ejemplo consiste simplemente en hacer parpadear uno de los leds de la placa DSK, la cual dispone de dos leds denominados DS1 Y DS2. El primero se enciende cuando el DSK es conectado a la alimentación de 5 voltios (POWER), y el segundo está conectado a una de las patillas de salida del DSP F243 la cual indica el estado del bit XF del registro de estados de la CPU. Éste es el led que se va a utilizar, el cual se encenderá cuando XF valga 1 y se apagará cuando XF valga 0. Para actuar sobre el valor de dicho bit será necesaria la utilización de directivas en ensamblador, ya que no existen funciones en lenguaje C que actúen directamente sobre un bit de un registro de estado. Por ello, para poner dicho bit a 1 (set) se utiliza la directiva "SETC XF" y para llevarlo a 0 (clear) será "CLRC XF". Como ejecutar cada instrucción tarda del orden de los 400 ns, será necesario emplear algún retardo para que sea posible apreciar el parpadeo del led. Ello se consigue a través de bucles. Ésta aplicación se realiza de forma sencilla mediante instrucciones en lenguaje ensamblador, el cual reserva la primera columna de datos para situar etiquetas. El proceso es simple:

- Primero se enciende el led mediante la instrucción:

```
SETC    XF
```

- Luego se carga en el acumulador del DSP un número suficientemente grande, como por ejemplo FFFFh (el máximo soportado por el acumulador sin signo, equivalente a 65535 en decimal) mediante la instrucción:

```
LACC    #0FFFFh
```

- A dicho número se le resta uno y el resultado sigue almacenado en el acumulador:

```
LOOP: SUB    #1
```

- Se comprueba si el acumulador se ha anulado y si no es así salta a la etiqueta loop:

```
BCND    LOOP, NEQ
```

- Cuando se anule el acumulador se ejecuta la siguiente instrucción, apagar el led:

```
CLRC    XF
```

Esto produce un retraso que consigue diferenciar los estados de encendido y apagado del led. Con éste mismo método se realiza también el proceso de volver a encender el led. Como el programa principal está escrito en ensamblador y queremos trabajar bajo

un entorno en lenguaje C, es necesario poner las instrucciones a través de la sintaxis "asm("SETC XF");", para indicar al archivo en C que lo escrito está en ensamblador. Por ello, el contenido del archivo "luz.c" será:

```
void main(void)
{
asm("     CLRC     SXM");    /* desactiva bit de signo*/
asm("START:  SETC     XF");
asm("        LACC     #0FFFFh");
asm("LOOP1:  SUB      #1");
asm("        BCND     LOOP1, NEQ");
asm("        CLRC     XF");

asm("        LACC     #0FFFFh");
asm("LOOP2:  SUB      #1");
asm("        BCND     LOOP2, NEQ");
asm("        B        START");

}
```

El valor cargado FFFFh es el mayor que soporta el acumulador sin utilizar signo, por lo que para aumentar la frecuencia de parpadeo del led basta con disminuir este número. Si se aumenta la frecuencia demasiado, el parpadeo llega a ser imperceptible por el ojo humano.

- **Proyecto en Code Composer:**

Para la correcta funcionalidad de la aplicación bajo un entorno en lenguaje C es necesaria la inclusión de una serie de archivos en el proyecto desarrollado en Code Composer como se explicó anteriormente. Por ello, el proyecto "luz.mak" queda como se muestra en la figura 3.19.

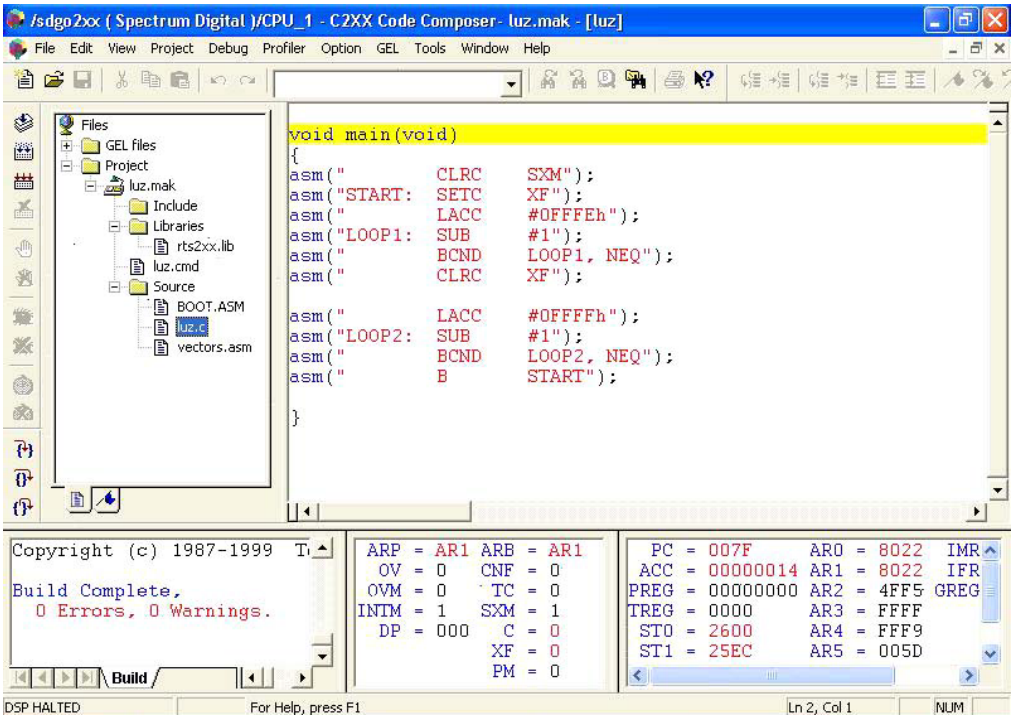


Figura 3.19. Proyecto luz.mak.

En el apartado anterior se explicó el programa principal de la aplicación "luz.c" para conseguir el parpadeo del led DS2. En lo que sigue a continuación se desarrolla el contenido y funcionalidad del resto de los archivos que forman este proyecto, cuya estructura puede servir como base de inicio para la construcción de una nueva aplicación con sólo añadir el código y archivos que sean necesarios para el funcionamiento de la misma.

Como se puede apreciar en la figura 3.19, el proyecto no incluye archivos de definición de registros de periféricos o de funciones ".h". Esto es porque no se va a utilizar ningún periférico ni funciones adicionales debido a la sencillez de la aplicación. Por el contrario son necesarios los siguientes archivos:

- luz.c (programa principal)
- luz.cmd (archivo de comandos del enlazador)
- vectors.asm (tabla de vectores de interrupción)
- boot.asm (módulo de inicialización del entorno de ejecución)
- rts2xx.lib (librería de entorno bajo lenguaje C)

Como se analizó anteriormente, el archivo de comandos "luz.cmd" define el mapa de memoria del dispositivo y la localización de las distintas secciones en la memoria. Su contenido es el siguiente:

```
MEMORY
{
    PAGE 0: VECS:  origin = 00000h, length = 00040h
            FLASH: origin = 00040h, length = 03FC0h
            SARAM: origin = 04000h, length = 00800h
            B0:    origin = 0FF00h, length = 00100h

    PAGE 1: B0:    origin = 00200h, length = 00100h
            B1:    origin = 00300h, length = 00100h
            B2:    origin = 00060h, length = 00020h
            SARAM: origin = 08000h, length = 00800h
}


SECTIONS
{
     vectors    : { } > VECS    PAGE 0
    .text       : { } > FLASH   PAGE 0
    .cinit      : { } > FLASH   PAGE 0
    .switch     : { } > FLASH   PAGE 0
    .const      : { } > SARAM   PAGE 1
    .data       : { } > SARAM   PAGE 1
    .bss        : { } > SARAM   PAGE 1
    .stack      : { } > SARAM   PAGE 1
    .sysmem     : { } > SARAM   PAGE 1
}
```

En él se definen los módulos de memoria para la tabla de vectores de interrupción VECS, las zonas de memoria interna FLASH y SARAM para programas y la zona de SARAM externa e interna (B0, B1, B2) para memoria de datos. También se definen la localización de las distintas secciones, entre las que destacan aquellas necesarias para trabajar en lenguaje C, como son ".cinit", ".const", ".switch" y ".stack".

Por otro lado, es necesario incluir la tabla de vectores de interrupción definida en el archivo "vectors.asm", ya que en ella se indica cual es el punto de comienzo de la aplicación. Su estructura es la siguiente:

```
         .ref _c_int0

        .sect    "vectors"
rset:   B       _c_int0         ;00h reset
int1:   B       int1            ;02h INT1
int2:   B       int2            ;04h INT2
int3:   B       int3           ;06h INT3
int4:   B       int4            ;08h INT4
int5:   B       int5            ;0Ah INT5
int6:   B       int6            ;0Ch INT6
int7:   B       int7            ;0Eh reserved
int8:   B       int8            ;10h INT8  (software)
int9:   B       int9            ;12h INT9  (software)
int10:  B       int10           ;14h INT10 (software)
int11:  B       int11           ;16h INT11 (software)
int12:  B       int12           ;18h INT12 (software)
int13:  B       int13           ;1Ah INT13 (software)
int14:  B       int14           ;1Ch INT14 (software)
int15:  B       int15           ;1Eh INT15 (software)
int16:  B       int16           ;20h INT16 (software)
int17:  B       int17           ;22h TRAP
int18:  B       int18           ;24h NMI
int19:  B       int19           ;26h reserved
int20:  B       int20           ;28h INT20 (software)
int21:  B       int21           ;2Ah INT21 (software)
int22:  B       int22           ;2Ch INT22 (software)
int23:  B       int23           ;2Eh INT23 (software)
int24:  B       int24           ;30h INT24 (software)
int25:  B       int25           ;32h INT25 (software)
int26:  B       int26           ;34h INT26 (software)
int27:  B       int27           ;36h INT27 (software)
int28:  B       int28           ;38h INT28 (software)
int29:  B       int29           ;3Ah INT29 (software)
int30:  B       int30           ;3Ch INT30 (software)
int31:  B       int31           ;3Eh INT31 (software)
```

De esta forma se indica al procesador a que rutina saltar cuando se da alguna de las interrupciones. En cada línea se especifica el nombre de la interrupción y el lugar a donde saltar (tras el ";" viene un comentario en el que se indica la dirección de cada uno de los vectores de interrupción). Como se puede apreciar casi todas saltan a ellas mismas. Esta es una manera de aislar interrupciones no deseadas durante el proceso cuando no son utilizadas, aunque lo normal es definir una función de error a la que saltarían dichas interrupciones. En nuestro caso sólo es empleada la interrupción 0 que marca el punto de comienzo de la aplicación en la función "c_int0", la cual se encuentra en el módulo "Boot.asm" de la librería "rts2xx.lib", y que se encargará de llamar a la función "main" posteriormente. Dicho módulo "Boot.asm" puede ser extraído de la librería para utilizarse en el proyecto y modificarlo si es necesario. Por ello se encuentra dicho archivo en la carpeta "Source" del proyecto (figura 3.19), el cual está específicamente modificado para su correcto funcionamiento en la serie C2xx, por lo que es recomendado su empleo en la documentación consultada. Esto no implica que se pueda prescindir de la librería "rts2xx.lib", ya que en ella se definen multitud de funciones necesarias para configurar el entorno de ejecución y utilizadas por el módulo "Boot.asm". Es por todo esto que las opciones que se toman para invocar al enlazador son "-c -o luz.out –x", imponiendo el punto de comienzo de la aplicación en la función

"c_int0", indicando el nombre del archivo de salida y obligando a la relectura del código. Dichas opciones se configuran a través del apartado "Options" del menú "Project" de la barra superior de Code Composer (figura 3.8). El resto de opciones para el compilador y ensamblador se utilizan las definidas por defecto por el programa.

Una vez creado el proyecto sólo queda construir el archivo "luz.out" mediante la opción "Build" del menú "Project" (figura 3.5), cargarlo mediante la opción "Load Program..." (figura 3.10) y ejecutarlo pulsando F5 tras hacer un reset del dispositivo (figura 3.11), como se detalló con anterioridad. Esto provocará el parpadeo del led DS2 de la tarjeta DSK. Si se modifica parte de un archivo, la opción "Build" compila y enlaza sólo los archivos modificados (Incremental Build). Si quieren compilarse todos los archivos del proyecto se utiliza la opción "Rebuild All" del menú "Project" (figura 3.5).

En el Anexo 1 de este documento se incluyen otras formas de programar la misma aplicación con menor cantidad de código.

3.2.4. Herramientas de depuración en Code Composer:

El paquete de software Code Composer está ampliamente orientado a al generación y depuración de código para sistemas basados en DSP. Por ello dispone de multitud de opciones y herramientas de depuración, las cuales ayudan al programador a corregir errores y perfeccionar el código de la aplicación. Dichas herramientas permiten seguir visualmente la evolución del proceso, facilitando así la tarea de seguimiento del mismo. Una de estas opciones es la posibilidad de trabajar con programas en lenguaje C y en ensamblador. Debido a la complejidad del lenguaje ensamblador, éste suele quedar reservado para su utilización en aplicaciones sencillas que no requieran un gran cocimiento de su código. Por ello Code Composer está optimizado para desarrollar proyectos en los que el archivo con la aplicación principal esté escrito en lenguaje C, más sencillo de emplear y entender en aplicaciones complejas. Debido a que este lenguaje es algo más extenso en cuanto a código máquina del DSP se refiere, para reducir tamaño y optimizar el código resultante existe la opción de visualizar en la misma ventana el código escrito en lenguaje C junto con su traducción en lenguaje ensamblador (figura 3.20). Esto es útil para programadores con experiencia en ensamblador, ya que pueden optimizar el código rescribiendo parte de él directamente en ensamblador para reducir su tamaño. Es decir, si una acción escrita en lenguaje C se traduce en 7 líneas en código ensamblador, el programador puede ser capaz de implementar esa misma acción directamente en ensamblador a través de un menor número de líneas, reduciendo así su tamaño y consiguiendo más memoria para almacenarlo. Para poder utilizar esta opción, se debe haber generado y cargado en el DSP el archivo final del proyecto ".out". Con el proyecto cargado en el dispositivo se puede visualizar el archivo con el programa principal ".c" en ambos lenguajes sólo con elegir la opción "Mixed Source/ASM" del menú "View" de la barra superior del programa (figura 3.20). De esta forma se visualiza cada línea de código C seguida de su traducción en código máquina y ensamblador en un tono más claro. Para desactivar esta opción se accede a la misma opción del menú "View" para desmarcarla. Todo esto también es posible a través del menú emergente que resulta de pulsar con el botón derecho del ratón sobre el interior de la ventana del archivo ".c", eligiendo "Source Mode" o "Mixed Mode".

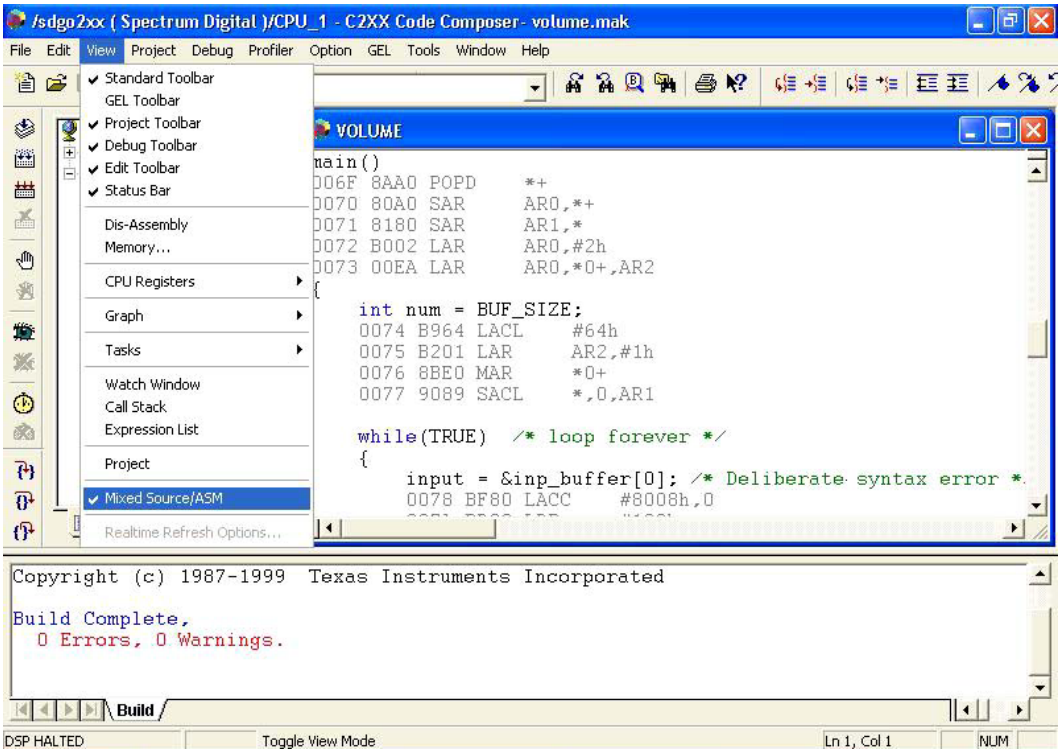


Figura 3.20. Código en modo mixto.

A la hora de ejecutar la aplicación se tiene la posibilidad de hacerlo paso a paso, es decir, línea a línea de código, como se comentó con anterioridad. Esto se consigue a través de las opciones disponibles "StepInto" (paso a paso), "StepOver" (saltar rutina) y "StepOut" (salir de rutina) en el menú "Debug" de la barra superior (figura 3.21). En dicho menú se encuentran otras opciones para la ejecución del código después de haber hecho un reset del dispositivo. Entre ellas está "Go Main", la cual lleva directamente a la función "main" del programa principal tras la ejecución del código que le precede en la aplicación y quedando la ejecución detenida en dicho punto.

Otra opción interesante es "Run to Cursor", muy útil para salir de bucles entre otras aplicaciones. Su función consiste en ejecutar el código hasta llegar al lugar donde esté posicionado el cursor, en donde se detiene la ejecución. También caben destacar las opciones "Run", "Run Free", "Animate" y "Halt". Las tres primeras tienen como función ejecutar el código cargado en el DSP, pero lo hacen de distinta manera: "Run" y "Animate" lo ejecutan hasta que encuentren un "Breakpoint" o punto de parada en el código donde se detiene la ejecución, aunque "Animate" actualiza las variables mostradas en las ventanas de visualización continuando después con la ejecución del código de forma automática, dando así la sensación de animación de los valores visualizados.

Figura 3.21. Menú Debug.

A diferencia de estas dos formas de ejecución está la opción “Run Free”, la cual ejecuta el código sin limitaciones, saltándose cualquier punto de parada y quedando el sistema DSP totalmente libre de comunicación con el Pc a través del JTAG que puede ser desconectado. Los tres tipos de ejecución pueden ser detenidos mediante la opción “Halt”. Al emplear cualquiera de estas opciones, la barra de estado situada en la parte inferior del programa Code Composer muestra en la parte izquierda el estado en que se encuentra el DSP (Running, Animate, Halted).

3.2.4.1.Ventanas de visualización:

Una de las herramientas fundamentales de Code Composer son las ventanas de visualización de memoria, registros y variables. Estas ventanas son activadas a través de las opciones “Memory” y “CPU Registers” del menú “View” de la barra superior (figuras 3.13 y 3.22).

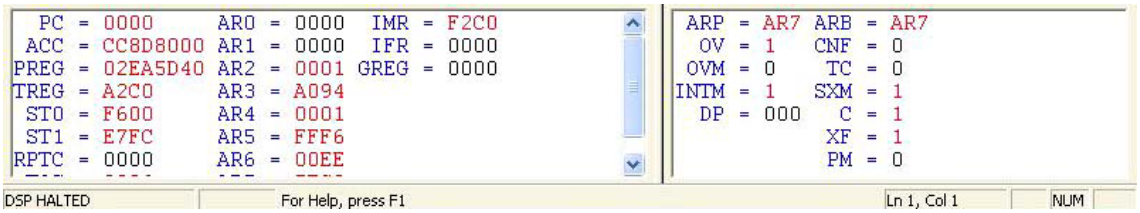


Figura 3.22. Registros de la CPU.

Esta herramienta permite visualizar la evolución de dichos parámetros además de editar su valor para lo que crea oportuno el programador. Para ello basta con hacer doble clic sobre el parámetro elegido y editar su valor en la ventana de dialogo que se abre (figura 3.23). En dicha ventana se puede desplegar una lista con los distintos parámetros incluidos en la ventana para poder modificarlos.

Figura 3.23. Edición de registro.

Las modificaciones que se lleven a cabo no serán efectivas hasta que se ejecute la siguiente línea de código. Esta herramienta es muy útil combinada con la ejecución paso a paso de la aplicación como se comentó con anterioridad, ya que permite visualizar la variación de los parámetros con la ejecución de cada línea de código, mostrando en color rojo los valores que han cambiado. Al contrario de la ejecución paso a paso, la ejecución libre del código mediante “Run” o “Run Free” no actualiza automáticamente los valores de los parámetros mostrados en dichas ventanas. Únicamente al detener la ejecución del programa dichos valores serán actualizados. Más adelante se mostrará cómo la opción “Animate” si actualiza los valores en conjunción con puntos de parada.

En la opción "CPU Registers" se muestran los registros de configuración de la CPU y sus registros de estado (figura 3.22), entre los que caben destacar:

- **Registros de la CPU:**
  - Program Counter (*PC*): Indica la línea actual de código a ejecutar (resaltada con una franja amarilla en el archivo).
  - *ACC*: Registro acumulador de la CPU para realizar distintos tipos de operaciones (con datos o direcciones).
  - *ARx*: 8 registros auxiliares para direccionamiento indirecto de datos en memoria.
  - *ST0, ST1*: Registros de estado de la CPU.
  - *IMR*: Registro de habilitación de las interrupciones enmascarables de la CPU.
  - *IFR*: Registro en el que se almacenan y eliminan las interrupciones que están en espera.
- **Registros de estado:** En realidad estos son los subregistros y bits en que se dividen los registros de estado ST0 y ST1.
  - *ARP*: Varios bits que indican el registro auxiliar ARx que está siendo empleado.
  - *OV*: Bit que indica si ha sucedido un desbordamiento en la operación realizada.
  - *CNF*: Bit que configura la memoria RAM interna del DSP como memoria de datos o de programa.
  - *SXM*: Bit de configuración del uso de signo en los datos.
  - *XF*: Determina el estado del pin de salida XF de uso general del DSP.

Si por ejemplo se quiere modificar el estado del bit XF conectado al led DS2, basta con pulsar dos veces sobre él y se abre la ventana de edición (figura 3.23), en la que poniendo 1 o 0 encenderemos o apagaremos dicho led. El cambio efectuado sólo tendrá validez cuando se reanude la ejecución del programa y se detenga (por ejemplo paso a paso).

Un tipo especial de ventana de visualización es la denominada "Watch Window", activada también a través del menú "View". En ella se pueden visualizar las variables empleadas en los archivos fuente ".c" para la programación de la aplicación. Para ello se pulsa con el botón derecho del ratón sobre la ventana "Watch Window" y se elige la opción "Insert New Expression..." (figura 3.24). A continuación se abre un cuadro de texto donde se escribe el nombre de la variable, la cual queda mostrada en la ventana. Al ejecutar paso a paso la aplicación se verá como varía el valor de la variable en el proceso.

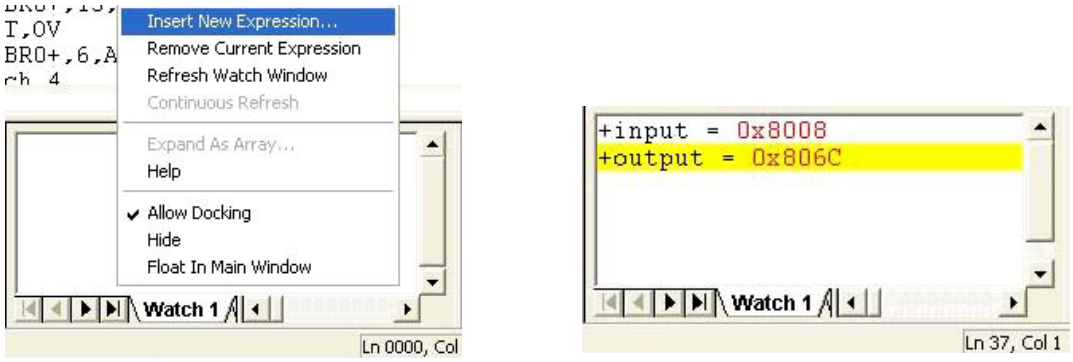


Figura 3.24. Watch Window.

Esto mismo se puede hacer seleccionando con el cursor la expresión a monitorizar en la ventana del archivo correspondiente, pulsando con el botón derecho del ratón sobre ella y eligiendo del menú emergente la opción "Quick Watch". Para actualizar el contenido de la pantalla se dispone de la opción "Refresh Watch Window" del menú emergente de la ventana "Watch Window". Del mismo menú se puede eliminar una variable seleccionándola y usando la opción "Remove Current Expression" (figura 3.24). La actualización de los valores de las variables mostradas sólo se produce al finalizar la ejecución de la aplicación, mostrando en color rojo los valores que han cambiado en el proceso ejecutado. Para automatizar esta actualización de valores, Code Composer dispone de dos tipos de herramientas: "Breakpoints" y "Probe Points". Ambas se utilizan en conjunción con todos los tipos de ventanas de visualización comentadas anteriormente, las cuales pueden desactivarse mediante la opción "Hide" del menú emergente (figura 3.24).

Este tipo de ventana de visualización permite mostrar la información de una variable de varias formas. Por ejemplo, si se quieren ver los valores que toma una variable llamada "i" utilizada por un bucle "for", bastaría con poner ese nombre al usar la opción "Insert New Expresión..." (figura 3.24). De esta forma en "Watch Window" aparecen los valores de dicha variable, que suele ser en formato decimal. Si se quiere expresar en algún formato determinado se pondría "i,d" para el decimal e "i,x" para el hexadecimal. Si por el contrario, lo que interesa es ver la dirección de memoria donde se almacena esa variable, la expresión a insertar sería "&i", indicando con "&" que el dato es una dirección de una variable de lenguaje C. Esta convención también es válida en cualquier campo de Code Composer que requiera una dirección de una variable, y siempre que en las opciones del compilador se haya activado la opción "-g" indicando "Generate Symbolic Debug Information", con lo que se puede emplear sólo el nombre de la variable sin tener que averiguar su dirección. Esto hace que la variable aparezca en la ventana de visualización junto con un "+", lo que indica que se puede expandir (figura 3.24). Si se expande se comprueba que en dicha variable se especifican dos datos: su dirección en formato hexadecimal y su valor en ese momento. Para tener correctamente cualquiera de estos valores, siempre es recomendable hacer inicialmente un reset del DSP y ejecutar parte de la función principal para asegurarse que esa variable ya se ha inicializado y no obtener datos erróneos.

3.2.4.2. Puntos de parada o Breakpoints:

Los "Breakpoint" son marcas que se dejan en alguna línea del código para que la ejecución se detenga de forma automática cuando se llegue a ellas. Al llegar a este punto se actualizan las ventanas de visualización de registros, variables y memoria, mostrando en color rojo los nuevos valores que han tomado las variables. Para utilizar esta herramienta es necesario tener conectado el sistema DSP al PC y con el proyecto cargado en el dispositivo para poder ejecutarlo. Una vez cargado, situar un "Breakpoint" es simplemente colocar el cursor en la línea del programa donde se quiera detener la ejecución y, pulsando con el botón derecho del ratón, elegir la opción "Toggle Breakpoint" del menú emergente (figura 3.25). También es posible pulsando el icono de la barra izquierda del programa. Esto marcará de un color rosa la línea elegida. Si a continuación se ejecuta la aplicación mediante la opción "Run", ésta se detendrá de forma automática en la línea marcada actualizando los valores de las ventanas de visualización activadas. En ese instante la línea marcada se mostrará en dos colores,

amarillo y rosa, indicando así que esa línea es un "Breakpoint" y que es la línea actual a ejecutar del código cuando se reanude la ejecución. En cambio, si se ejecuta el código mediante la opción "Animate", cada vez que se pase por esa línea Code composer detiene la ejecución, actualiza las ventanas de visualización y continúa con la ejecución de manera automática, dando así la apariencia de animación en las variables, aunque no es en tiempo real.

Para eliminar un "Breakpoint" se procede exactamente de la misma forma en la que se situaron (figura 3.25). Por el contrario, si se quieren eliminar todos los que se hayan empleado, basta con pulsar el icono de la barra izquierda del programa. Cuando se cambia de proyecto sin reiniciar Code Composer, es una buena práctica eliminar antes todos los "Breakpoints", pues éstos no se eliminan automáticamente al cambiar de proyecto.

Con esta herramienta puede ejecutarse el código "por bloques", analizando así el funcionamiento de distintos tramos de código.

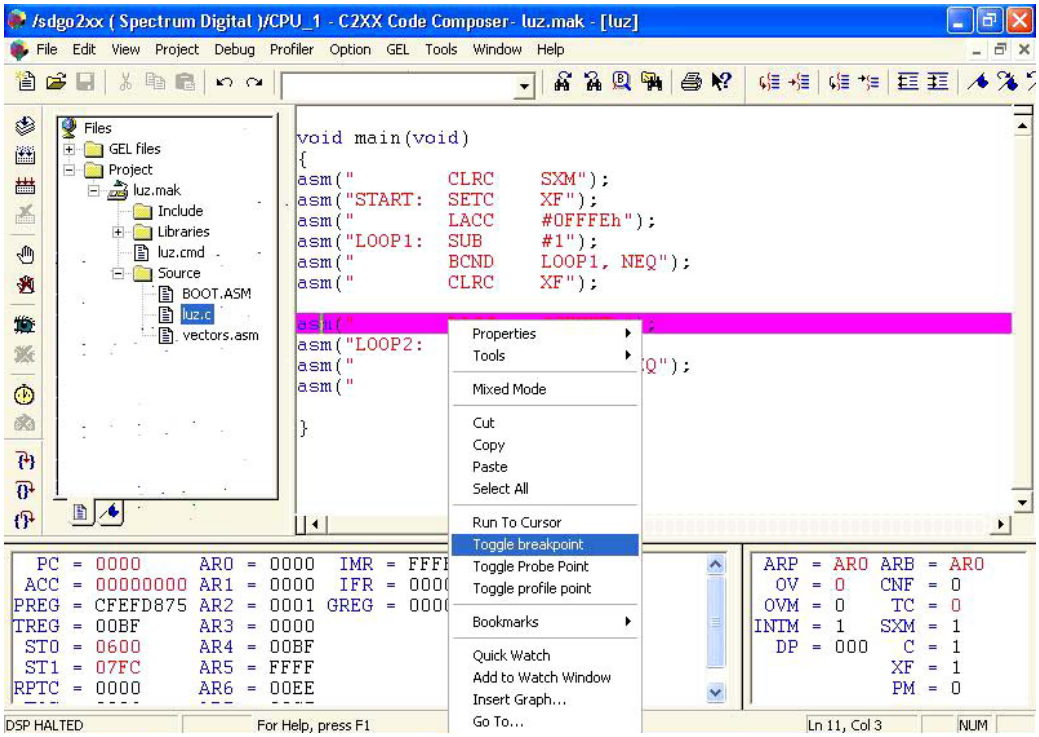


Figura 3.25. Situar Breakpoint.

3.2.4.3. Puntos de prueba o Probe Points:

Los "Probe Points" o puntos de prueba son más específicos para la monitorización continua de variables en las ventanas de visualización y representación gráfica, además de ser utilizados para simular le entrada de datos por alguno de los periféricos de entrada al dispositivo. Su establecimiento en el código se hace de manera similar al de los "Breakpoints" mediante la opción "Toggle Probe Point" (figura 3.26), o mediante el icono de la barra izquierda del programa. Esto resaltará en color azul la línea elegida, la cual será tomada como un punto de actualización de algún parámetro o aplicación al cual esté conectado. Es este punto el que define las posibilidades de los "Probe Points" dependiendo del objeto al que se conecte. Por

ejemplo puede conectarse a una de las ventanas de visualización, como puede ser la de registros de estado de la CPU. De esta forma, al ejecutar la aplicación mediante la opción "Run" o "Animate", las variables contenidas en dicha ventana serán actualizadas automáticamente cada vez que el código ejecute la línea marcada como "Probe Point".

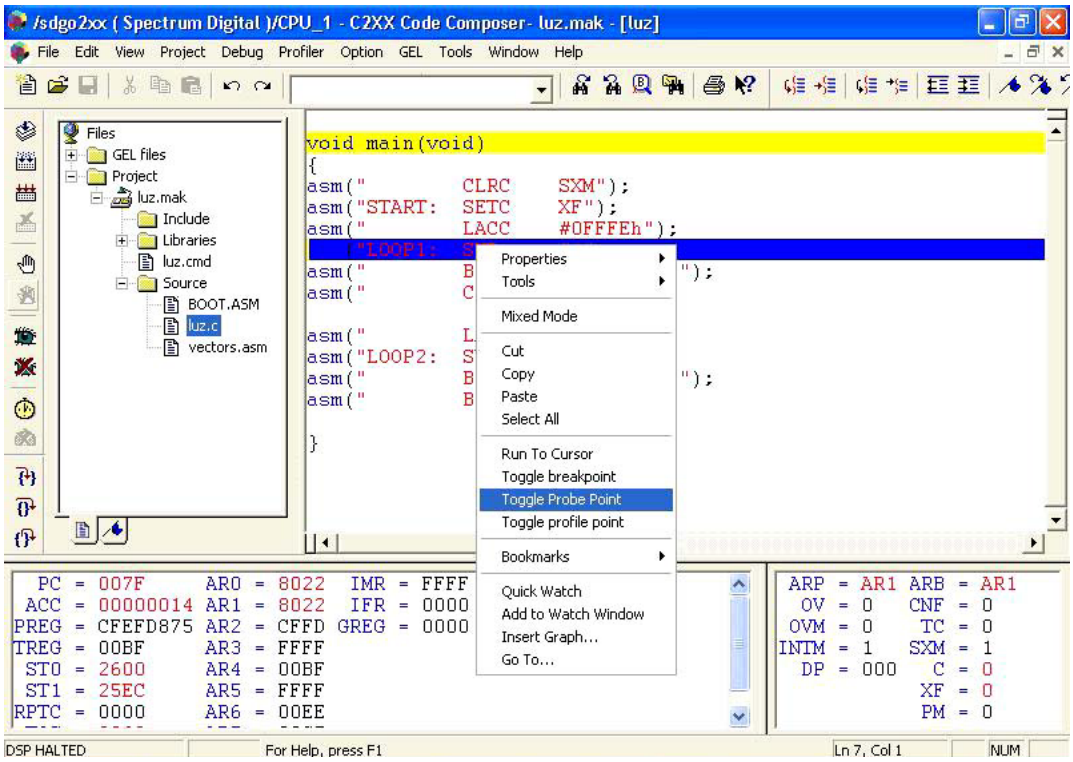


Figura 3.26. Situar Probe Point.

La ejecución del código en estas condiciones se realizará de manera muy lenta, ya que en cada paso por el punto de prueba, Code Composer detiene la ejecución momentáneamente para actualizar el objeto conectado al mismo, volviendo después a reanudar la ejecución haciéndose así mucho más lenta. Esta conexión puede realizarse con cualquiera de las ventanas de visualización de que dispone Code Composer, en especial con "Watch Window", ya que en ella se muestran las variables de programación deseadas por el programador para ser vigiladas. Ello se consigue a través de la opción "Probe Points..." del menú "Debug" de la barra superior (figura 3.21), que abre un cuadro de texto también utilizado por los "Breakpoints" y "Profile Points" según la pestaña superior que se elija (figura 3.27). En él se muestran los puntos de prueba que se han situado en el código indicando el elemento al que se han conectado. Al situar uno nuevo, éste no estará conectado por lo que se mostrará "No Connection". Para conectarlo se dispone de un submenú desplegable en la opción "Connect To" que contiene la lista de elementos disponibles a los cuales conectarlo (C2XX Registers Type 0 equivale a CPU Registers y Type 1 a Status Registers). Una vez elegido el adecuado se pulsa sobre el botón "Replace", quedando así conectado el punto de prueba. De esta forma, el elemento conectado será actualizado automáticamente en la ejecución de la aplicación. En ésta misma ventana se pueden activar y desactivar "Breakpoints" y "Probe Points" disponibles en el código, únicamente marcando o desmarcando la casilla que le precede en el listado. Esto sirve para prescindir momentáneamente de algún punto de prueba sin eliminarlo para posteriores usos.

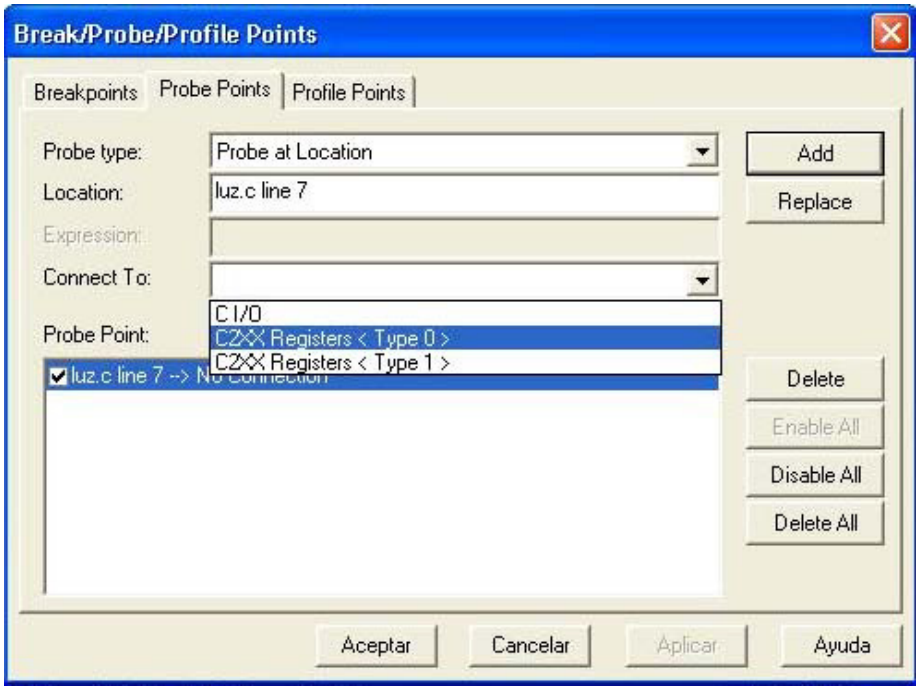


Figura 3.27. Conexión de Probe Points.

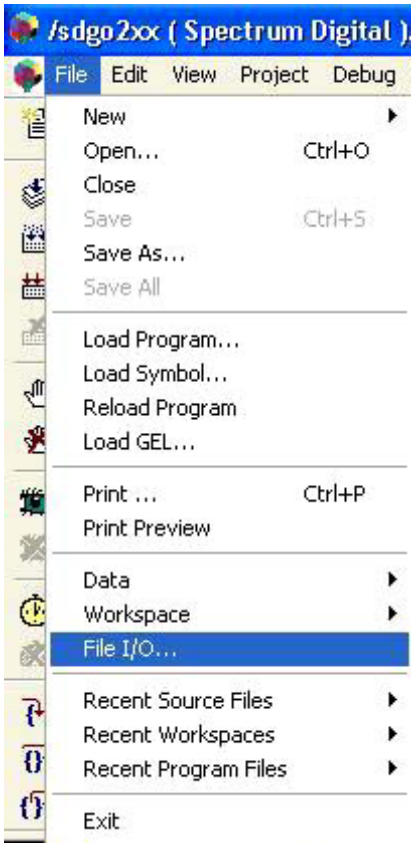


Figura 3.28. Datos de entrada.

Otra de sus aplicaciones es para la simulación de la entrada y salida de datos a través de los puertos de comunicación del sistema DSP. Para ello se conecta el "Probe Point" a un archivo que contenga o vaya a contener los datos a utilizar. Este archivo se selecciona mediante la opción "File I/O..." del menú "File" de la barra superior (figura 3.28). Esto abre una ventana (figura 3.29) en la que se selecciona el archivo en cuestión mediante el botón "Add File", como por ejemplo el archivo "sine2.dat" que contiene los valores de una onda senoidal para simular una entrada al sistema. En dicha ventana (figura 3.29) se debe seleccionar el archivo añadido para poder indicar la dirección a partir de la cual se quieren almacenar los datos leídos (Address) y el número de medidas que se harán en cada lectura (Length). El primer campo puede ser el nombre de la variable en la cual se guardará le lectura realizada, como por ejemplo "inp_buffer", evitando así utilizar valores de direcciones. En este caso, para que la entrada de datos de la onda senoidal sea de manera continua, se selecciona la casilla "Wrap Around" para que cuando se acaben los datos del archivo ".dat" se vuelva a su inicio de forma automática.

Por último, es necesario conectar el archivo a un punto de prueba o "Probe Point" en el programa para que sean leídos los datos de entrada, por lo que éste punto debe estar situado en la línea que llame a la función de adquisición de datos definida en el programa. La conexión se realiza pulsando el botón "Add Probe Point" (figura 3.29) que abre la ventana de conexión de los puntos de prueba (figura 3.27), en la cual se elegirá en el campo "Connect To" el archivo de entrada "sine2.dat", tras lo que se pulsará el botón "Replace" para confirmar la conexión. De esta forma, y ejecutando el código mediante la opción "Animate", cada vez que se pase por la función de adquisición que contiene el "Probe Point" se tomarán datos de la entrada senoidal, y todo ello de forma continua y sin detener la ejecución de la aplicación de manera permanente. Por el contrario, si se utiliza la opción "Run", la ejecución se detendrá en el primer "Breakpoint" que contenga el programa.

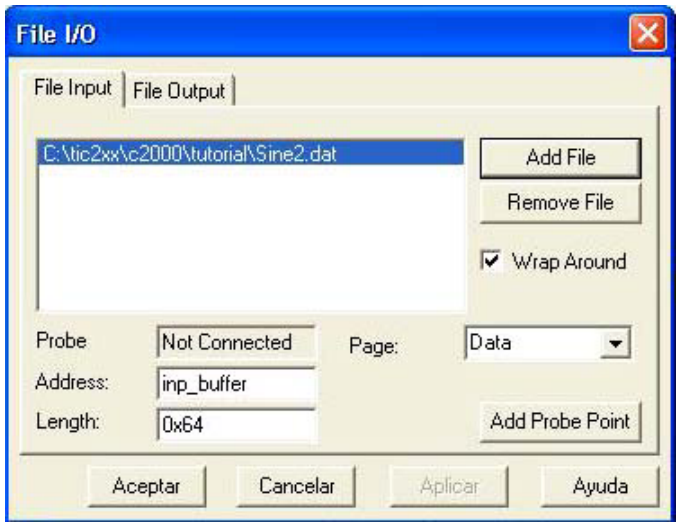


Figura 3.29. Conexión de datos.

Por otro lado, se pueden emplear los puntos de prueba para almacenar en un archivo los valores que ha tomado una variable, lo cual se realiza de igual forma que para simular una entrada excepto que en la ventana "File I/O" (figura 3.29) se emplea la pestaña "File Output" para indicar el nombre del archivo destino. El resto de parámetros son los mismos, salvo que el campo "Address" es ahora la variable que queremos almacenar.

En caso de querer eliminar totalmente un solo punto de prueba se procede de igual forma que para situarlo (figura 3.26). Si por el contrario se quieren eliminar todos los empleados en el programa, se utiliza el icono de la barra izquierda del programa.

3.2.4.4. Ejemplos de aplicaciones:

A continuación se muestran algunos ejemplos de aplicaciones de los "Probe Points" y otras herramientas explicadas anteriormente utilizando como base el proyecto "cuadrado.mak" (figura 3.30), cuyo código se detalla en el Anexo 1 de este documento. Para abrir este proyecto se procede como se comentó anteriormente a partir de la opción "Open" del menú "Project" de la barra superior. Lo siguiente es generar el archivo para el DSP "cuadrado.out" con la opción "Rebuild All" y cargarlo en la memoria del DSP mediante "Load Program" en el menú "File". Una vez hecho esto siempre es recomendable hacer un reset al DSP mediante la opción correspondiente del menú "Debug" y posteriormente "Go Main" para visualizar el programa principal.

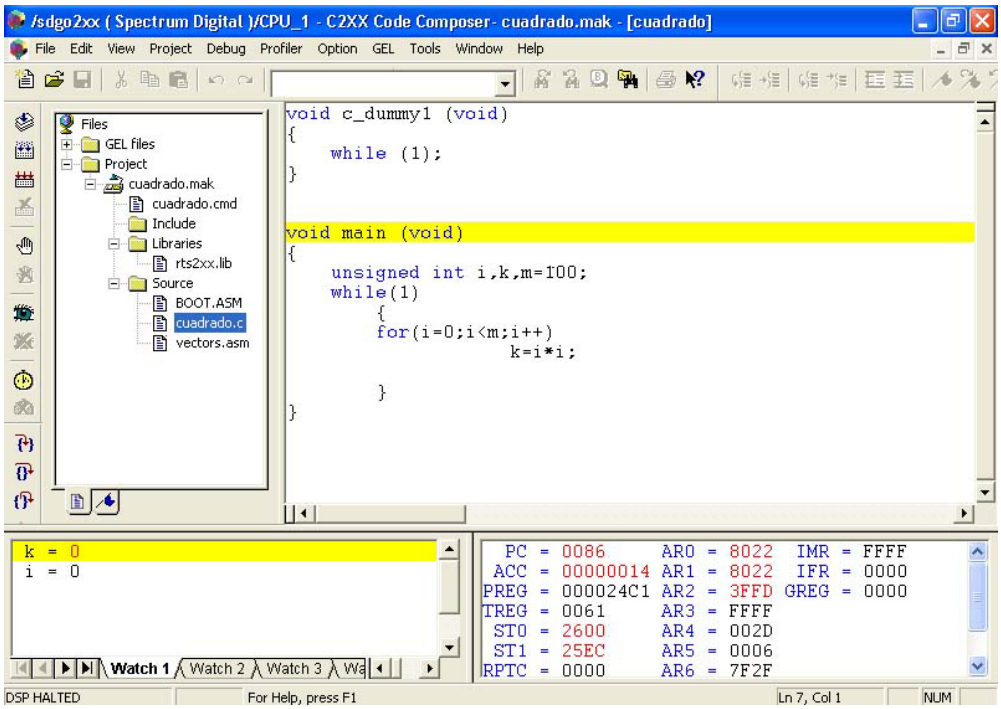


Figura 3.30. Proyecto Cuadrado.mak.

Con el programa ya cargado activamos las ventanas de visualización "Watch Window" y "CPU Registers", introduciendo en la primera las variables "i" y "k" (figura 3.30). Para ver el funcionamiento del programa situaremos un "Probe Point" en la línea "k=i*i" como se explicó anteriormente (figura 3.26) al cual asociaremos la ventana "Watch Window" mediante la ventana de asociación de la opción "Probe Points" del menú "Debug" (figura 3.27). En al campo "Connect To" se elige del menú desplegable la primera "Watch Window" de las que aparecen, ya que dicha ventana de visualización tiene 4 pestañas y se está utilizando sólo la primera. Hecho esto se pulsa "Replace" y ya está hecha la conexión. Si se ejecuta el código mediante "Run" o F5, se puede apreciar como varían los valores de "i" y "k" en la ventana de visualización (figura 3.31).

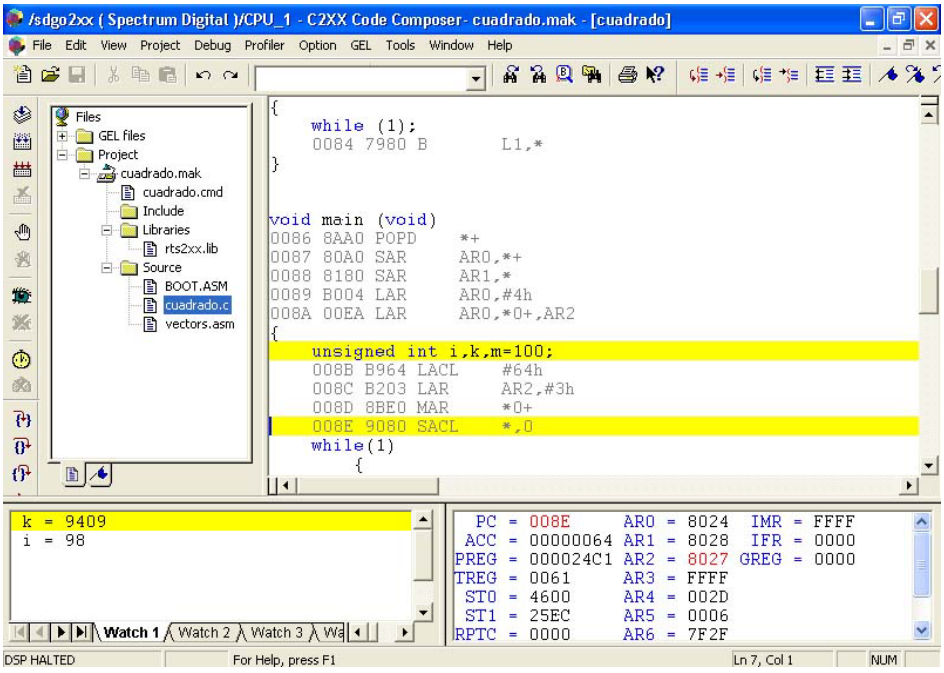


Figura 3.31. Control de variables.

Si por ejemplo necesitamos conocer las direcciones de memoria donde se encuentran las variables "i" y "k" se puede proceder de la siguiente forma. En primer lugar haremos uso de la ejecución paso a paso (F8) y del código mostrado en modo mixto (Mixed Source/ASM del menú View). Visualizando el texto en modo mixto y pulsando repetidas veces F8, se puede comprobar que la línea de definición de la variables "i" y "k" hace uso del registro auxiliar de direccionamiento indirecto a memoria AR2. Ello se ve en el código ensamblador que sigue a esa línea, además de ver que el valor de dicho registro varía en la ventana de visualización de los registros de la CPU. Se comprueba que al final de dicha ejecución AR2 apunta a la dirección 0x8027 y AR0 a la 0x8024 (figura 3.31), con lo que las variables estarán localizadas en torno a dichas direcciones. Su localización correcta se puede determinar haciendo uso de la ventana de visualización de memoria (View->Memory). Esto abre un cuadro para determinar la fracción de memoria que se desea visualizar (figura 3.32), en la que se indica la dirección de memoria (0x8024) y la zona (DATA). De esta forma se abre una ventana en la que se muestra la memoria de datos a partir de la dirección facilitada (figura 3.33).

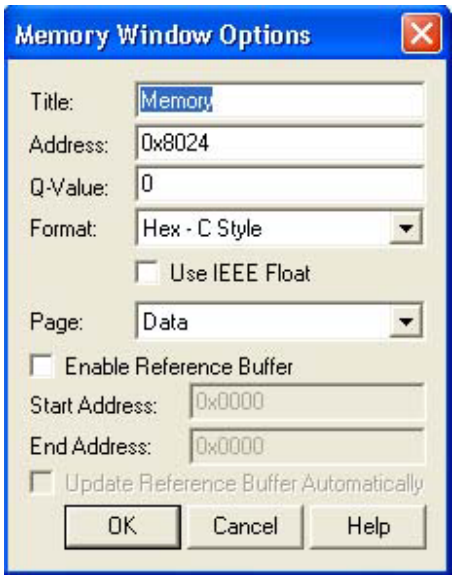


Figura 3.32. Memory Options.

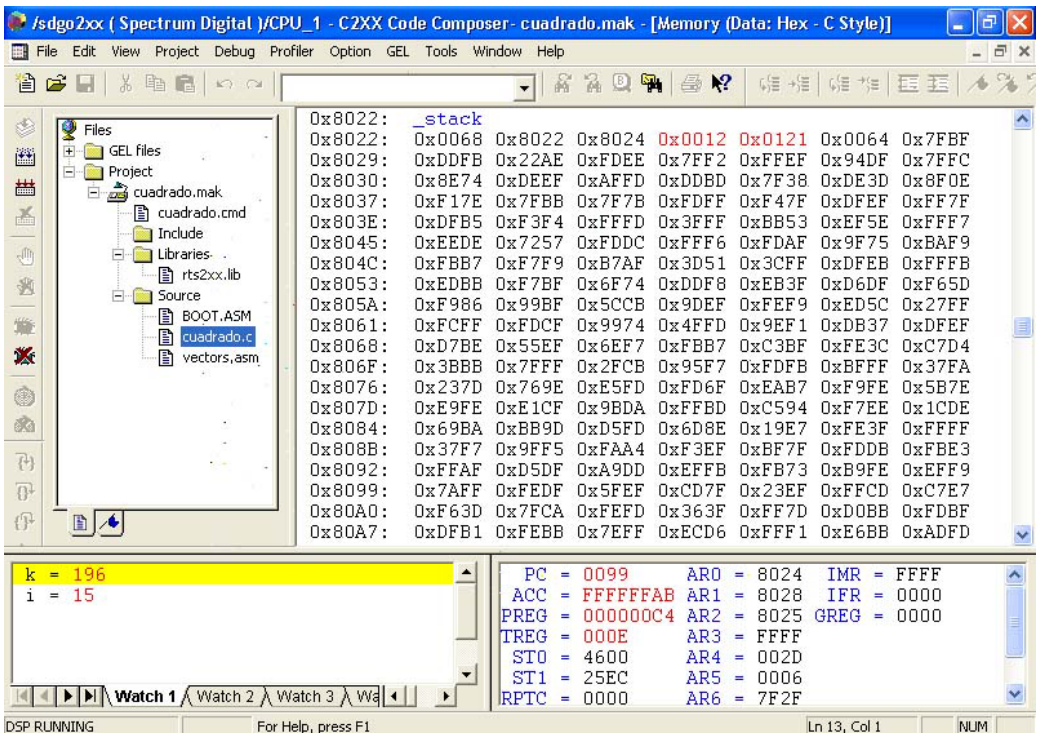
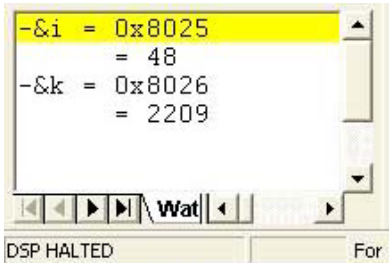


Figura 3.33. Memoria de datos con i y k.

Para averiguar dónde se ubican las variables asociaremos esta ventana al punto de prueba ya situado. Para ello se vuelve a la ventana de conexión (figura 3.27) y se conecta dicho punto con la ventana "Memory (DATA: HEX – C Style)". Si ahora se ejecuta el código con "Run" o F5 se comprueba como varían dos direcciones de memoria (figura 3.33), siendo la primera la variable "i" y la siguiente "k", lo cual se puede comprobar transformando dichos valores hexadecimales a decimales y comparándolos con los mostrados en "Watch Window". De esto se desprende que las direcciones empleadas son 0x8025 para "i" y 0x8026" para "k".



Otro método mucho más rápido para hallar esas direcciones sería utilizando la ventana "Watch Window" en la que se introducen las expresiones "&i" y "&k". Al desplegarlas pulsando el signo + se mostrarán sus direcciones y valores en ese instante (figura 3.34). Si se ejecuta parcialmente el código se asegura la iniciación de esas variables evitando tomar valores erróneos.

Figura 3.34. Dirección y valor.

3.2.5. Representación Gráfica:

Otra de las herramientas para la depuración de código que incluye Code Composer es la representación gráfica de señales y variables. Dispone de varios tipos de representación:

- Tiempo/frecuencia
- Constelación
- Eye Diagram
- Image

aunque en este documento sólo se comentarán los aspectos fundamentales del diagrama de tiempos, el cual es uno de los más utilizados habitualmente. Para explicar sus particularidades nos basaremos en un proyecto sencillo llamado "cuadrado.mak" (figura 3.30), el cual contiene dos únicas variables: "i" que varía de 0 a 100 y "k" que es el cuadrado de "i". Para representarlas gráficamente son necesarias sus direcciones, las cuales fueron obtenidas anteriormente. Dichas direcciones son 0x8025 para "i" y 0x8026" para "k". Su representación se efectúa mediante la opción "Graph" del menú "View" del menú superior (figura 3.35) eligiendo la representación "Time/Frecuency". Esto despliega una ventana de configuración del gráfico (figura 3.35) en la que se indican las características que tendrá el mismo, así como de la fuente de los datos a representar, entre las que destacan:

- *Display Type:* Dedicado a indicar el tipo de gráfico que se requiere.
- *Graph Title:* Campo para indicar un título para el gráfico.
- *Start Address*: Para indicar la dirección de comienzo de la variable o los datos a representar.
- *Page:* Zona de memoria en la que se encuentra la dirección dada en el campo anterior, y por tanto la variable.
- *Acquisition Buffer Size:* Número de datos que se tomarán en cada momento para su representación.
- *Display Buffer Size*: Rango de datos que se representarán en pantalla.
- *DSP Data Type:* Indica el tipo de dato que se emplea (con o sin signo, 16 o 32 bits).
- *Autoscale:* Para activar el ajuste automático de la escala de representación.

Con todo esto, si se quiere representar gráficamente la evolución de "i", se selecciona "Single Time" en el tipo de gráfico, ya que sólo se representa una variable, dirección de comienzo 0x8025 (o escribir &i), Page DATA, buffer de adquisición 1 (que coja los datos de uno en uno), tamaño del display 200 (por ejemplo), tipo de datos de 16 bits sin signo y Auto escala activada (figura 3.35). Para ver con movimiento la evolución del gráfico, conectaremos un punto de prueba que situaremos en la línea "k=i*i" con dicho gráfico a través de la ventana de conexión (figura 3.27), seleccionando en ella dicho punto y tomando la opción "Graphical Display" del campo "Connect To".

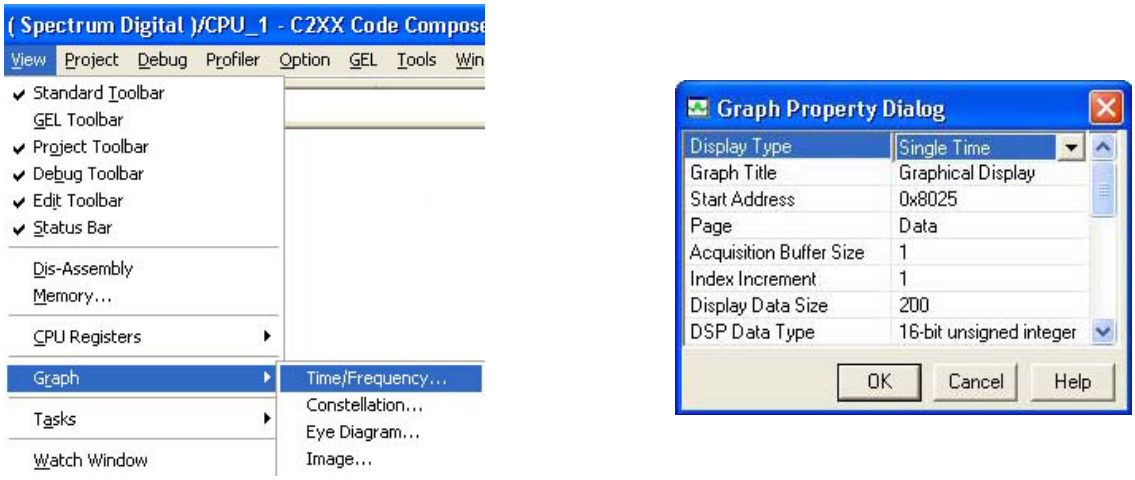


Figura 3.35. Tipos de gráficas y configuración.

Para volver a ver el gráfico en pantalla, se selecciona éste en el menú "Windows" de la barra superior de Code Composer, en el que se incluyen todas las ventanas abiertas en el proyecto. Si se ejecuta la aplicación con "Run" o F5, se comprueba cómo se dibuja la gráfica adaptándose automáticamente la escala (figura 3.36). Si se hubiese querido representar la variable "k" habría de introducirse su dirección de memoria 0x8026 en la ventana de configuración (o haber puesto &k como dirección). Existe la posibilidad de representar a la vez las dos variables (figura 3.36). Ello se consigue eligiendo la opción "Dual Time" en el campo "Display Type" de la ventana de configuración (figura 3.35) e introduciendo las direcciones de las dos variables.

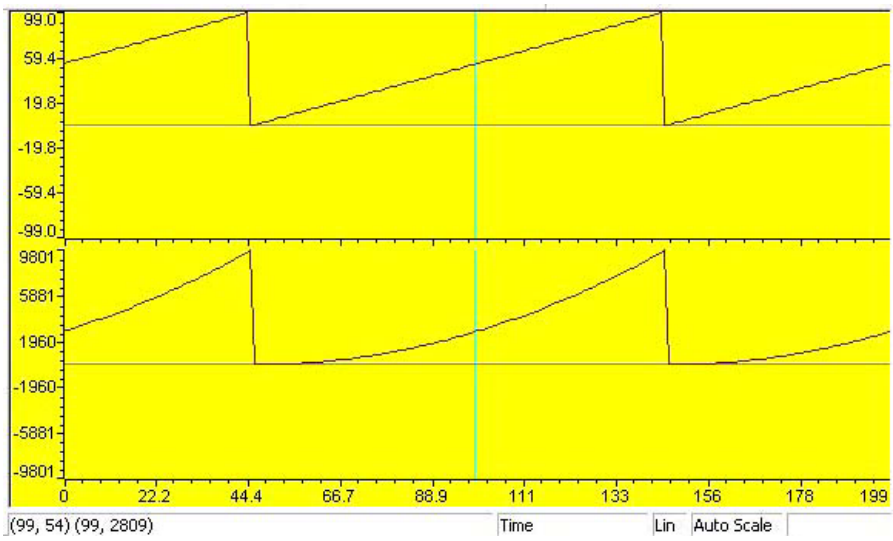


Figura 3.36. Doble gráfico con variables i y k.

Una vez desplegado el gráfico, se puede acceder a la ventana de configuración de sus propiedades (figura 3.35) pulsando con el botón derecho del ratón sobre el mismo y tomando la opción "Properties..." del menú emergente. Para quitar la ventana de representación se utiliza la opción "Hide" de dicho menú.

Puede darse el caso de que los datos a representar vengan dados en un archivo y ser representados más de un punto a la vez. Éste es el caso del ejemplo que trae Code Composer en su Tutorial "Volume", en el cual se simula la entrada de una onda senoidal a partir de un archivo. En dicho tutorial se utiliza como dirección de comienzo el nombre de la variable "inp_buffer" y el tamaño de adquisición del buffer en 100 datos por medida. No es necesario utilizar el símbolo "&" al ser dicha variable una matriz de datos, con lo que su nombre ya indica la dirección de comienzo de la misma.

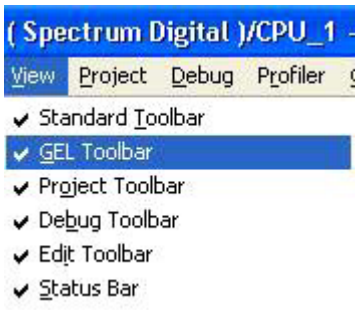
3.2.6. Lenguaje de extensión general GEL:

Es un lenguaje auxiliar de Code Composer muy parecido al lenguaje C, y que sirve para crear funciones que amplían las funcionalidades del programa. Este lenguaje es muy útil para personalizar el entorno de trabajo y permitir el acceso automático a funciones de Code Composer, facilitando la tarea del programador. Los comandos en este lenguaje pueden ser introducidos en cualquier campo del programa que admita expresiones, incluida "Watch Window", de forma que se ejecuten dichas expresiones en cada actualización, ya sea por "Breakpoints" o "Probe Points". De esta forma pueden automatizarse tareas típicas en Code Composer como la compilación del proyecto, carga y ejecución de la aplicación, abrir ventanas de visualización con unas variables determinadas, etc., además de personalizar el entorno de trabajo con funciones preprogramadas para visualizar, por ejemplo, los registros de algún periférico del DSP sólo con seleccionarlo de una lista. Por ello esta herramienta tiene multitud de aplicaciones, de las cuales se comentará como introducción a este lenguaje una de las principales: la personalización del entorno en Code Composer.

Este lenguaje permite definir funciones para ser llamadas desde cualquier punto de Code Composer, pero también dispone de funciones ya definidas y que son de gran ayuda. Estas funciones son del tipo "GEL_función(parámetros)". Existen una gran variedad de ellas, entre las que destacan:

- **GEL_MapAdd():** Se emplea para definir zonas de memoria de igual forma que lo hace el archivo de comandos del enlazador ".cmd".
- **GEL_WatchAdd():** Empleada para añadir de forma automática alguna variable o registro en la ventana "Watch Window".
- **GEL_ProjectBuild():** Función que compila el proyecto de forma automática.
- **GEL_ProjectLoad():** Carga el proyecto en el DSP.
- **GEL_Reset():** Hace un reset al dispositivo.
- **GEL_Run():** Ejecuta el código del DSP.

Los parámetros que deben incluirse en estas funciones, junto con el resto de funciones de este tipo de que dispone el programa, están disponibles en la ayuda y manuales de Code Composer. En este documento se pondrán algunos ejemplos de dichas funciones para explicar su funcionamiento.



Para emplear las funciones generadas en lenguaje GEL, Code Composer dispone de una barra especial en la parte superior izquierda del programa a través de la cual se introducen las llamadas a dichas funciones (figura 3.38), y que se activa mediante la opción "GEL Toolbar" del menú "View" de la barra superior (figura 3.37).

Figura 3.37. Activa GEL.

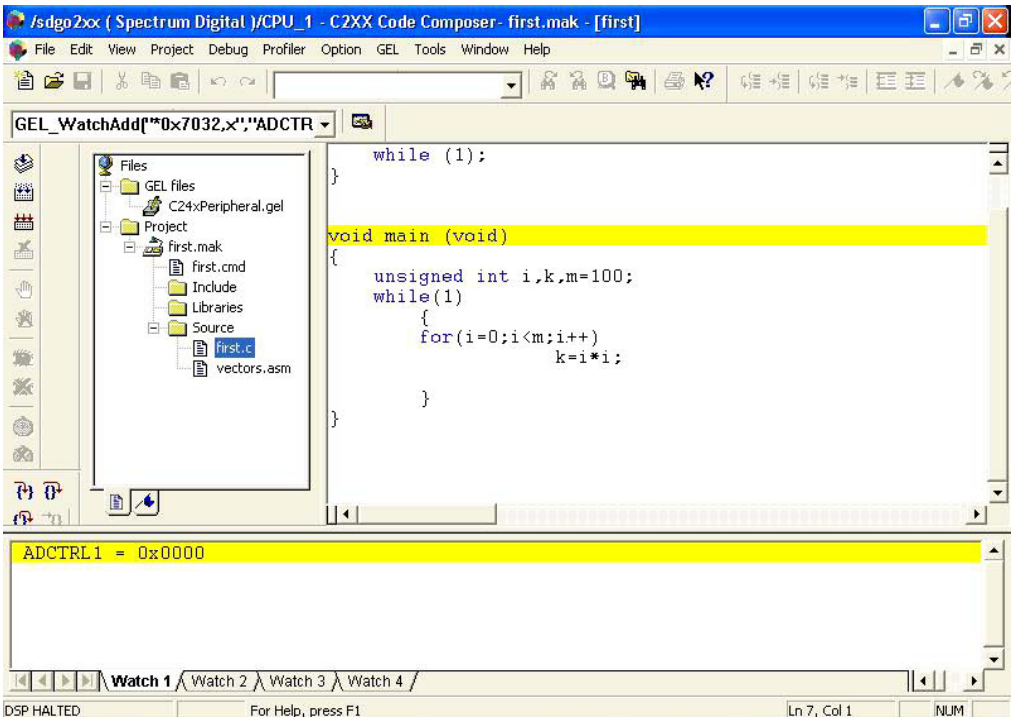


Figura 3.38. Barra GEL y registro ADCTRL1.

Si por ejemplo se quisiera ver en la ventana "Watch Window" el registro de control 1 del convertidor analógico-digital, en dicha barra se introduciría el comando: "GEL_WatchAdd("*0x7032,x","ADCTRL1");". De esta forma se abre automáticamente la ventana de visualización mostrando el valor del registro, que por simplicidad se llama ADCTRL1 para evitar usar su dirección de memoria 0x7032 (figura 3.38). El resto de funciones GEL se emplean de la misma forma, pero puede darse el caso de que se deseen aplicar varias funciones de forma consecutiva para personalizar el entorno antes de comenzar a trabajar. Ello puede simplificarse a través de la creación de un archivo en el que se incluyan todas las funciones a ejecutar. Este archivo tendría extensión ".gel", y permite aplicar todas las funciones que contiene sólo con elegirlas de una lista. Eso se consigue cargándolo en Code Composer mediante la opción "Load GEL" del menú superior "File" (figura 3.39).

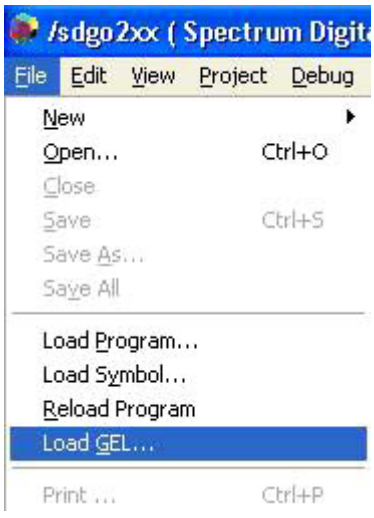


Figura 3.39. Load GEL.

Un ejemplo de archivo sería éste, que podría llamarse “registros.gel”:

```
menuitem "Watch General Purpose Timer Registers";

hotmenu All_GP_Regs()
{
    GEL_WatchAdd("*0x7400,x","GPTCON");
    GEL_WatchAdd("*0x7401,x","T1CNT");
    GEL_WatchAdd("*0x7402,x","T1CMPR");
    GEL_WatchAdd("*0x7403,x","T1PR");
    GEL_WatchAdd("*0x7404,x","T1CON");
    GEL_WatchAdd("*0x740c,x","T3CON");
}
```

En este archivo la directiva “menuitem” genera un submenú llamado “Watch general purpose timer regiters” en el menú “GEL” de la barra superior del programa (figura 3.40). En dicho submenú se genera una opción llamada “All_GP_Regs” debido a la directiva “hotmenu”, que al elegirla ejecuta todas las funciones que se encuentran en su interior, las cuales incluyen los registros indicados en “Watch Window” (figura 3.40) para su visualización. De esta forma se aplican de una vez todas las funciones, siendo esto muy útil cuando se trabaja frecuentemente con un mismo grupo de funciones.

Para un mismo proyecto pueden emplearse distintos archivos “.gel”, y cada uno de ellos puede incluir una amplia lista de menús y submenús en la opción GEL de la barra superior, personalizándose así el entorno de trabajo en Code Composer.

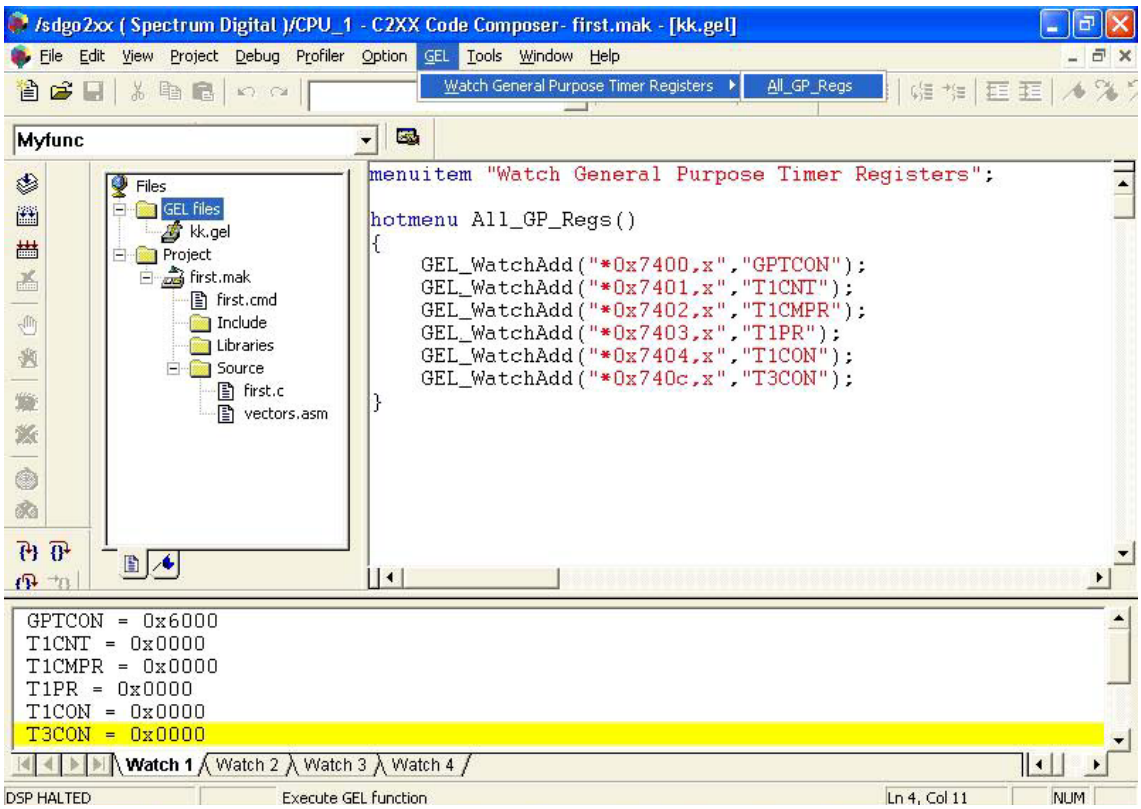


Figura 3.40. Submenú GEL.

Pero existe una forma más rápida de iniciar y personalizar el entorno. Esto se consigue haciendo un único archivo que contenga todo, pero que además se cargue automáticamente cuando se arranca Code Composer. Para ello el archivo debe contener la función "Startup()", en la cual se incluyen los comandos a ejecutar en el arranque, seguido de los submenús que se quieran añadir al menú GEL. Un ejemplo sería:

```
StartUp()

{
GEL_MapOn();
GEL_MapReset();
GEL_MapAdd(0x0000,1,0x005F,1,1);/* MMRs  */
GEL_MapAdd(0x0060,1,0x0020,1,1);/* On-Chip RAM B2 */
GEL_MapAdd(0x0200,1,0x0100,1,1);/*On-Chip RAM B0 if CNF=0 */
GEL_MapAdd(0x7010,1,0x000F,1,1);/*Peripheral Config */
GEL_MapAdd(0x7020,1,0x0010,1,1);/* Peripheral-WDT / RTI */
GEL_MapAdd(0x7030,1,0x0010,1,1);/* Peripheral - ADC */

GEL_WatchAdd("*(int *)0x7029@data,x", "WDCR");
GEL_WatchAdd("*(int *)0x7403@data" , "T1PR");
GEL_WatchAdd("*(int *)0x7417@data" , "CMPR1");

GEL_ProjectBuild();
GEL_Load("E:\\Borapn3\\F243\\F243led2\\F243led2.out");
GEL_Reset();                          /* reset the DSP       */
GEL_Go(main);                /* run the code until main    */
}
menuitem "Watchdog";/*Add the submenu "Watchdog" to "GEL" */

hotmenu Disable_Watchdog()
{
    GEL_MemoryFill(0x7029,1,1,0x40);/*  A  function  to  disable  the
watchdog    */
}
```

Este ejemplo se encargaría de iniciar las zonas de memoria que se van a emplear, las direcciones de los registros de los periféricos, incluir registros en la ventana de visualización, generar y cargar el proyecto, resetear el DSP y llevarlo a la función "main" y por último generar un menú "Watchdog" en la opción GEL de la barra superior de menús. Este archivo debe ser cargado a la vez que arranca Code Composer. Ello se consigue pulsando con el botón derecho del ratón en el acceso directo al programa que existe en el escritorio de Windows, para elegir "Propiedades" del menú emergente. En la pestaña "Acceso directo" existe un campo llamado "Destino", el cual indica la ruta hacia el ejecutable de Code Composer seguido de la ruta al archivo ".gel" que se cargará en el arranque. Este archivo es por defecto "init.gel" y debe ser sustituido por el que se quiera emplear, como por ejemplo "migel.gel" (figura 3.42) incluido en Anexo 2. Aplicando estos cambios, la próxima ejecución de Code Composer realizará todas estas operaciones en su inicio, ahorrando tiempo al programador y facilitando su trabajo. Todos los archivos ".gel" que se carguen serán mostrados en la ventana de proyectos en la sección GEL (figura 3.41). De esta forma podrán ser recargados en el programa si han sufrido alguna modificación o eliminarlos del proyecto. Para ello se pulsa con el botón derecho del ratón sobre el archivo afectado y eligiendo la opción necesaria del menú emergente (figura 3.41).

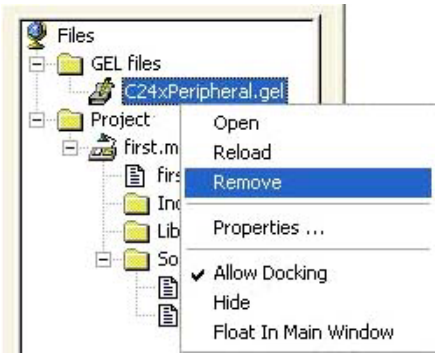


Figura 3.41. Opciones GEL.

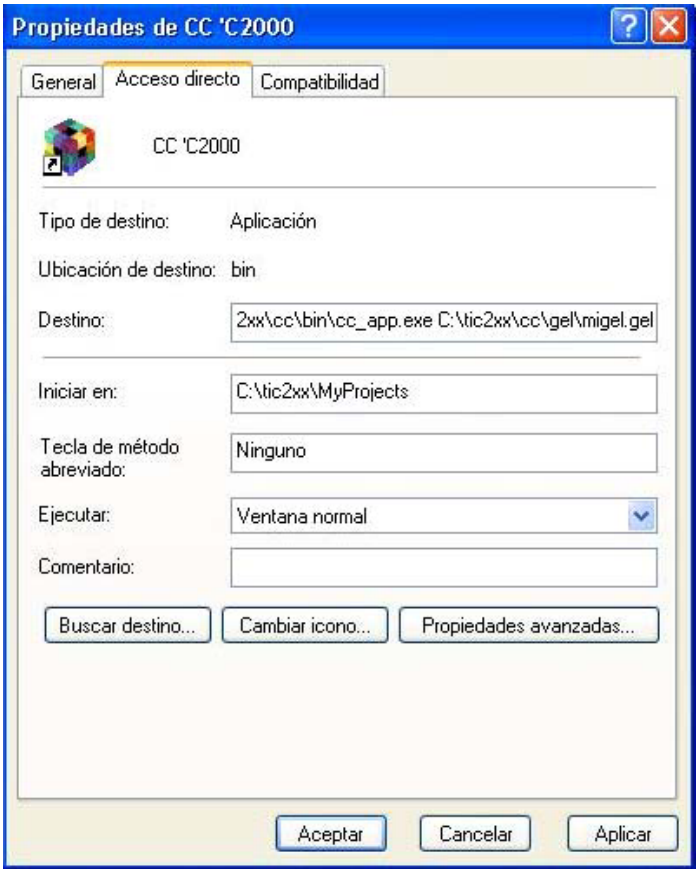


Figura 3.42. Configuración de inicio con GEL.

3.2.7. Monitorización en tiempo real:

Una de las principales ventajas de Code Composer es que permite la depuración y monitorización de la aplicación en tiempo real. Esto es de gran ayuda a los programadores pues les permite comprobar de una forma real el funcionamiento del código implementado, sin apenas interferir en él gracias a la emulación JTAG. De todas formas, aunque esto es la teoría, en la práctica se aprecian pérdidas de sincronismo y ralentización en la ejecución del código, de lo que se deduce que la emulación interfiere en ella más de lo deseado, haciéndose notar su efecto en menor o mayor medida dependiendo de las ventanas de visualización que se empleen. De todas formas, como se verá a continuación, los archivos que son necesarios incluir para tener emulación en tiempo real son configurables, con lo que posiblemente se puedan mejorar las prestaciones de esta herramienta de depuración. Esta configuración se encuentra ampliamente desarrollada en la ayuda y, sobretodo, en el tutorial sobre tiempo real que dispone Code Composer.

Para explicar los pasos a seguir a la hora de obtener una aplicación que pueda ser emulada en tiempo real, se va a hacer uso de la aplicación desarrollada anteriormente para el parpadeo del led DS2 del DSK. Para facilitar esta tarea el programa se ha simplificado, de modo que incluya lo estrictamente necesario para su correcto funcionamiento. Por otro lado se ha pasado a lenguaje ensamblador, ya que la mayoría de las líneas y archivos a añadir están en este lenguaje. De esta forma se evita también el proceso de iniciado necesario bajo un entorno en C, con lo que el número de archivos disminuye. Por todo esto, el proyecto del que partimos contiene los siguientes archivos:

- *led.asm:* programa principal de la aplicación.
- *luz.cmd:* Archivo del enlazador.
- *vectors.asm:* Tabla de vectores de interrupción.

El proyecto es básicamente igual, salvo que al estar en ensamblador no se necesita la librería para C "rts2xx.lib". Además, se ha eliminado el archivo "Boot.asm" al incluir dentro del programa principal lo necesario para que la aplicación funcione correctamente (ver punto 3.2.8. Notas Adicionales), ya que la aplicación es muy sencilla y no necesita todas sus funciones. De esta forma se ha reducido el número de archivos, siendo "luz.cmd" y "vectors.asm" los mismos que se emplearon en el proyecto original. Por tanto, el archivo "led.asm" contiene el siguiente código:

```
      .global  _c_int0      ;Global declaration

WD_CNTL    .set  07029h           ;WD Control register

    .text                    ;Code in text section


_c_int0:                    ;Entry point from reset vector

       CLRC    SXM       ;Clear Sign Extension Mode

       LDP     #0E0h           ; Load Data Page (WD)
       SPLK    #00EFh,WD_CNTL  ;Disable watchdog


start:                                  ;Aplicación original
                SETC XF

                LACC #0FFFFh
loop1:          SUB #1
                BCND loop1, NEQ
                CLRC XF

                LACC #0FFFFh
loop2:          SUB #1
                BCND loop2, NEQ

                B start
```

Para que esta aplicación pueda ser utilizada en tiempo real hay que modificarla añadiendo una serie de líneas y archivos que incluyan en su código el monitor de tiempo real que dispone Code Composer. Este monitor emplea las interrupciones 7 y 19 del DSP para comunicarse con el ordenador a través del emulador JTAG, por lo que la tabla de vectores de interrupción debe ser modificada en dichas interrupciones para asociarlas a las rutinas que emplea el monitor. Por tanto, el archivo "vectors.asm" queda de esta manera:

```
         .include    "c200mnrt.i" ; Include conditional
                                    ;assembly options.
         .mmregs             ; Include standard register
                                 ;mnemonics.

         .global _c_int0, PHANTOM


           .sect "vectors"
```

```
RESET     B   _c_int0   ; 00
INT1      B   PHANTOM   ; 02
INT2      B   PHANTOM   ; 04
INT3      B   PHANTOM   ; 06
INT4      B   PHANTOM   ; 08
INT5      B   PHANTOM   ; 0A
INT6      B   PHANTOM   ; 0C
              .if ( 1 ) ; macro occupies fourteen words
                                  ; in the vector table.
MON_EINTR     mon_eintr_vecs       ; 0E
                                   ; 10
                                   ; 12
                                   ; 14
                                   ; 16
                                   ; 18
                                   ; 1A
              .else ; macro not in vector table.
MON_EINTR_B   B     MON_EINTR      ; 0E
HUNG10        B     HUNG10         ; 10
HUNG12        B     HUNG12         ; 12
HUNG14        B     HUNG14         ; 14
HUNG16        B     HUNG16         ; 16
HUNG18        B     HUNG18         ; 18
HUNG1A        B     HUNG1A         ; 1A
              .endif
HUNG1C        B     HUNG1C         ; 1C
HUNG1E        B     HUNG1E         ; 1E
HUNG20        B     HUNG20         ; 20
TRAP          B     TRAP           ; 22
NMI           B     PHANTOM          ; 24
              .if ( 1 ) ; macro occupies eight words in
                                  ;the vector table.
MON_ETRAP     mon_etrap_vecs       ; 26
                                   ; 28
                                   ; 2A
                                   ; 2C
              .else ; macro not in vector table.
MON_ETRAP_B   B     MON_ETRAP      ; 26
HUNG28        B     HUNG28         ; 28
HUNG2A        B     HUNG2A         ; 2A
HUNG2C        B     HUNG2C         ; 2C
              .endif
HUNG2E        B     HUNG2E         ; 2E
HUNG30        B     HUNG30         ; 30
HUNG32        B     HUNG32         ; 32
HUNG34        B     HUNG34         ; 34
HUNG36        B     HUNG36         ; 36
HUNG38        B     HUNG38         ; 38
HUNG3A        B     HUNG3A         ; 3A
HUNG3C        B     HUNG3C         ; 3C
HUNG3E        B     HUNG3E         ; 3E

              .end
```

En este archivo se introducen unas macros de llamadas a rutinas de interrupción asociadas al monitor de tiempo real en las dos rutinas afectadas (7 y 19). Estas modificaciones están basadas en los archivos de ejemplo que trae Code Composer. Además se hace necesaria la inclusión del archivo "c200mnrt.i" que contiene las opciones de configuración de dicho monitor. También se ha incluido para las primeras interrupciones una rutina "PHANTOM" que será definida más adelante en el programa principal, aunque su función es únicamente volver a dicho programa. Por lo demás el archivo es idéntico al original, salvo que las interrupciones han sido etiquetadas con

otro nombre (HUNG) a partir de la interrupción 8. Esto no influye en nada al programa, ya que dichas interrupciones saltan a ellas mismas si se diera alguna de ellas, quedando de esta forma aisladas de igual forma que en el "vector.asm" original.

Todos estos cambios necesitan de una nueva estructuración de las secciones de memoria en el archivo "luz.cmd". Para ello se incluyen unas nuevas secciones exclusivas para el monitor de tiempo real mediante la inserción de las siguientes líneas:

```
mon_main    : { } >    FLASH      PAGE 0
mom_pge0    : { } >    B2         PAGE 1
mom_rgst    : { } >    B2         PAGE 1
```

De esta forma el programa principal del monitor queda localizado en la memoria de programas en la sección FLASH, además de reservar espacio para sus operaciones en otras dos secciones situadas en el bloque de memoria RAM interna del DSP llamado B2, el cual debe quedar empleado de forma exclusiva por el monitor. Por ello ha de evitarse el uso de B2 por cualquier parte del programa que no sea el monitor de tiempo real. El resto del archivo "luz.md" queda de la misma forma que en el proyecto original.

A continuación deben añadirse una serie de líneas en el programa principal que posibiliten la ejecución del programa monitor. Esto se ha hecho siguiendo los pasos (step) que indica el tutorial de Code Composer sobre tiempo real, por lo que se han dejado explícitamente dichos comentarios. De todas formas, alguno de ellos se han suprimido debido a que no son necesarios para nuestra aplicación. La función de los pasos empleados es:

- ***Step 1:*** Declaración global de las variables PHANTOM y MON_RT_CNFG (rutina de configuración del monitor).
- ***Step 3:*** Llamada a la rutina de configuración del monitor antes de habilitar interrupciones y comenzar la aplicación.
- ***Step 4:*** Habilitar la interrupción enmascarable 7 para su uso por el monitor de tiempo real.

En el código también se definen la rutina de interrupción "PHANTOM" y una variable auxiliar "test" que se empleará para almacenar el estado del led. Por tanto el archivo "led.asm" resulta ser:

```
* Step 1: Global Declaration:

     .global  _c_int0, PHANTOM
     .global MON_RT_CNFG

* Variables declaration:

WD_CNTL    .set 07029h          ;WD Control register
IMR        .set 0004h       ;Interrupt Mask Register
IFR        .set 0006h       ;Interrupt Flag Register

    .bss test,1             ;Declara señal test en bss

* Main Program:

    .text             ;Código en sección text

_c_int0:                ;Entry point from reset vector
```

```
      CLRC    SXM          ;Clear Sign Extension Mode

      LDP     #0E0h          ; Load Data Page (WD)
      SPLK  #006Fh, WD_CNTL  ; Disable WatchDog

      LDP     #04h
      SPLK #01h,test    ; Inicializa variable test=1

* Step 3: Initialize real-time monitor

       CALL MON_RT_CNFG; Call real time monitor routine

* Step 4: Unmask interrupt

       LDP    #0h
       SPLK        #01000000b,IMR   ;Unmask interrupt 7
                                    ;for real time interrupt

* Aplicación:

start:
           SETC XF          ;Enciende led

           LDP     #04h
           SPLK #01h,test   ;test=1

           LACC #0FFFFh          ;Acumulador=FFFFh
loop1:     SUB #1                ;Resta 1
           BCND loop1, NEQ       ;Ve si resultado=0
           CLRC XF               ;Apaga led

           LDP     #04h
           SPLK #0h,test    ;test=0

           LACC #0FFFFh           ;Acumulador=FFFFh
loop2:     SUB #1                 ;Resta 1
           BCND loop2, NEQ        ;Ve si resultado=0

           B start           ;Vuelve al comienzo

* Rutina de interrupción PHANTOM:

PHANTOM:
           RET         ;Vuelve al programa
```

Una vez hechas todas las modificaciones sólo resta hacer el proyecto en Code Composer. Dicho proyecto debe contener los siguientes archivos:

- *luz.cmd:* Archivo de comandos para el enlazador modificado para incluir el monitor en tiempo real.
- *led.asm:* Programa principal modificado para tiempo real.
- *vectors.asm:* Tabla de vectores de interrupción modificada para el monitor de tiempo real.
- *c200mnrt.asm:* Programa monitor en tiempo real.
- *c200mnrt.i:* Opciones de configuración del monitor.

Para hacer un nuevo proyecto empleamos la opción Project->New (figura 3.5), en la que se indica la carpeta y nombre del proyecto. A continuación se emplea "Add Files.." (figura 3.15) para añadir todos los archivos mencionados anteriormente en el proyecto, a excepción de "c200mnrt.i" que se incluye mediante la opción "Scan All Dependencies" (figura 3.15). Los archivos "c200mnrt.asm" y "c200mnrt.i" se incluyen en Code Composer en la carpeta "C:\tic2xx\c2000\MONITOR". Con todo ello el proyecto queda como se muestra en la figura 3.43.

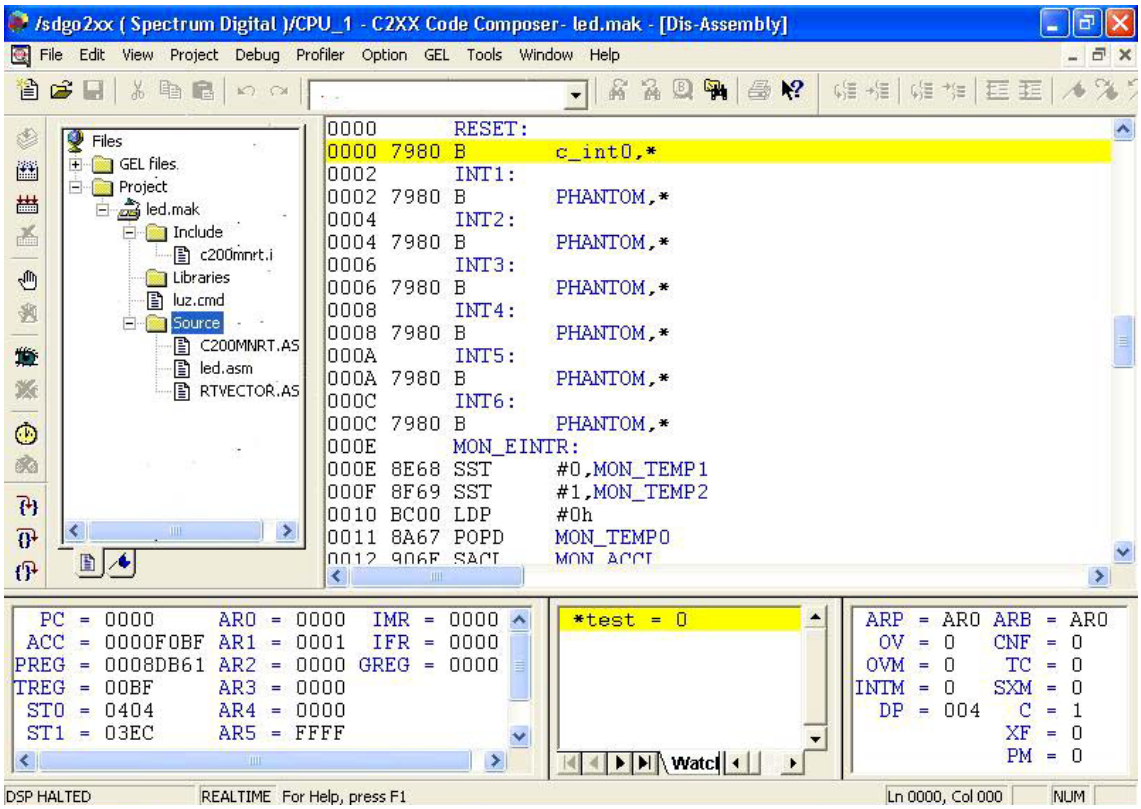
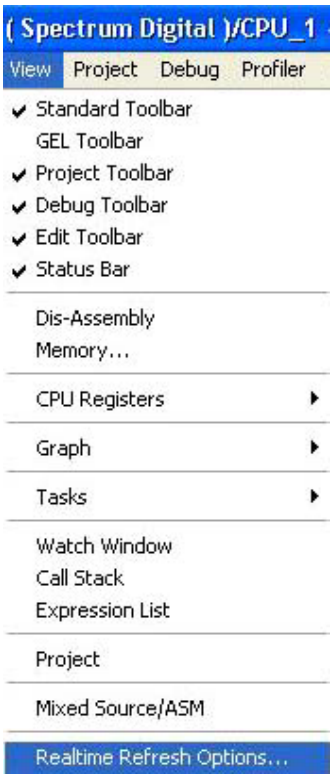


Figura 3.43. Proyecto en tiempo real.

Para generar el archivo final "led.out" se dejan las opciones por defecto en la ventana de configuración del proyecto (figura 3.8), asegurándose de que el enlazador tiene la opción "-c" (ROM Autoinitialization), ya que así se impone que el punto de entrada del programa sea la etiqueta "c_int0". Esta etiqueta no es obligatoria en un proyecto en ensamblador, pero se ha utilizado como analogía del lenguaje C para simplificar la comprensión. Si es necesario se puede sustituir en todo el programa (incluido "vectors.asm") por otra etiqueta (usualmente START), con lo que la opción "-c" ya no sería necesaria. Una vez configurado se construye el archivo mediante Project->Rebuild All (figura 3.5) y, si no ha habido errores, se carga en el DSP mediante File->Load Program (figura 3.10). Antes de comenzar siempre es recomendable hacer un reset al DSP mediante Debug->Reset DSP (figura 3.12) para inicializar los registros y evitar errores. Para comprobar el funcionamiento en tiempo real haremos uso de las ventanas de visualización "Watch Window, "CPU Registers" y "Status Registers" del menú "View" de la ventana superior (figura 3.13), quedando el entorno de trabajo como se muestra en la figura 3.43. En "Watch Window" se monitorizará la variable auxiliar "test", la cual se inserta mediante la opción "Insert New Expresión.." (figura 3.24) del menú emergente, introduciendo en el cuadro de texto "*test" para indicar que es un número entero sin signo.

Para que el programa sea monitorizado en tiempo real debe activarse la opción correspondiente mediante Debug->Real Time Mode (figura 3.44). Esto puede provocar que la línea actual a ejecutar (resaltada en amarillo) deje de ser la 0000, con lo que siempre es conveniente hacer otro reset del DSP. De esta forma ya está activo el modo en tiempo real, y así se muestra en la barra de estado en la zona inferior izquierda `REALTIME`. Para ejecutar la aplicación basta con pulsar F5 o mediante "Run".

Figura 3.44. Real Time Mode.



En un principio, si no se ha usado el tiempo real anteriormente, no se apreciará ningún cambio respecto al modo habitual de ejecución, ya que debe configurarse el tiempo de muestreo del monitor. Esto se consigue mediante la opción View->Real Time Refresh Options (figura 3.45), que abre una ventana (figura 3.46) en la que se introduce el tiempo de muestreo. Éste debe ser mayor de 100ns, con lo que la monitorización no será estrictamente en tiempo real. Para que automáticamente todas las ventanas de visualización se actualicen, debe activarse la casilla "Global Continuous Refresh". Al aceptar las opciones, de forma instantánea comienzan a variar los valores de algunas de las variables mostradas en las ventanas de visualización, entre las que destacan el acumulador "ACC" utilizado para el retardo en el proceso de parpadeo del led y la señal auxiliar test, que oscila entre 0 y 1 mostrando el estado del led. Pueden hacerse pruebas poniendo distintas frecuencias de muestreo: 100ns, 200ns, 400ns, 1000ns, etc.

Figura 3.45. Refresh options.

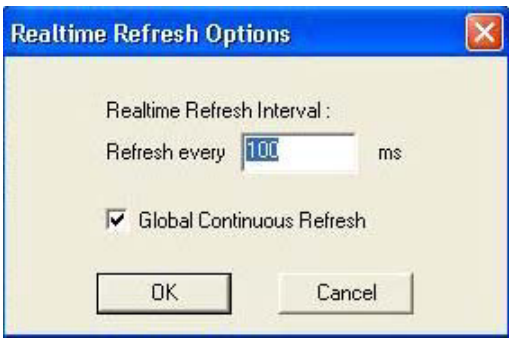


Figura 3.46. Tiempo de muestreo.

Con mayores tiempos de muestreo las variables cambian más lentamente, lo cual influye en un aspecto importante. Se puede apreciar cómo el número de ventanas y variables actualizándose junto con un tiempo de refresco corto influyen en la ejecución de la aplicación, produciendo retrasos y alteraciones indeseadas, de modo que a menor número de ventanas o variables, menor es su influencia. Es necesario destacar que el tener activa la ventana de visualización de memoria el programa sufre un retraso muy apreciable en su ejecución, con lo que no parece viable la monitorización a través de ella. Por el contrario, puede emplearse en su defecto la ventana "Watch Window", la cual tiene menor influencia en el proceso. Para evitar esta posible sobrecarga por ventanas activas, puede desactivarse la opción "Global Continuous Refresh" de las opciones de refresco del monitor (figura 3.46), de forma que pueda emplearse una única ventana para la monitorización. Ello se consigue eligiendo la opción "Continuous Refresh" del menú emergente al pulsar con el botón derecho del ratón sobre la ventana elegida para monitorizar (figura 3.47). Esto es válido para todas las ventanas visualización (Watch Window, Memory, CPU Registers, Graph). De esta forma únicamente la ventana elegida se actualizará de forma continua, mientras las demás no lo hacen.

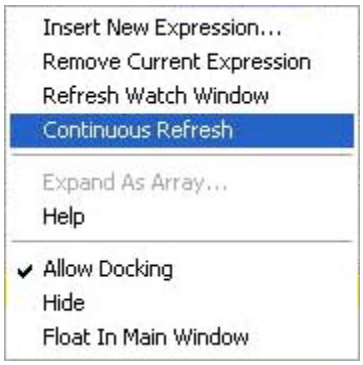
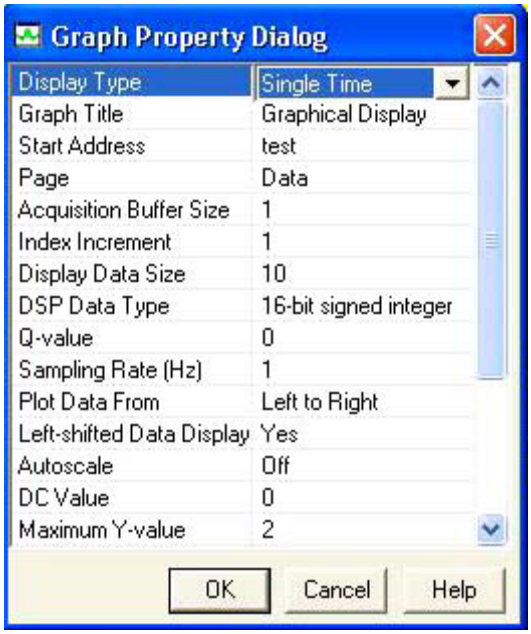


Figura 3.47. Refresco continuo.



El uso de la monitorización en tiempo real permite representar gráficamente las variables necesarias en movimiento continuo según vayan evolucionando, como si se tratase de un osciloscopio. En este ejemplo se puede representar la señal de prueba "test" para ver la evolución de los estados del led DS2. Para ello se emplea la opción View -> Graph -> Time Frecuency (figura 3.35) y se rellenan las opciones como se muestra en la figura 3.48.

Figura 3.48. Opciones de gráfico.

Después de aceptar estos datos se abre la ventana gráfica, la cual habrá de configurarse para refresco continuo (figura 3.47) si no está marcada la opción "Global Continuous Refresh" de las opciones de tiempo real (figura 3.46). De esta manera, si se ejecuta la aplicación, se mostrará de forma continua en la gráfica la evolución de la señal "test" (figura 3.49).

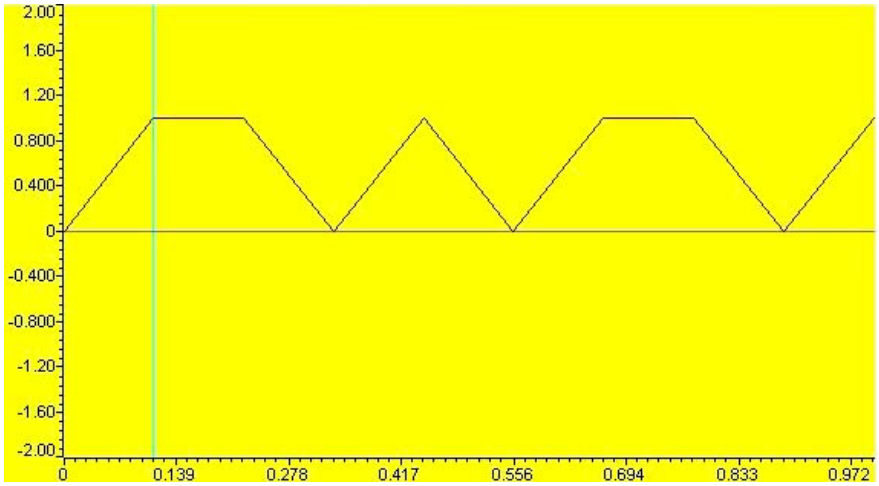


Figura 3.49. Representación de señal test.

Una de las ventajas del modo en tiempo real es la posibilidad de cambiar el valor de las variables de las ventanas de visualización sin necesidad de detener la ejecución del programa. Para ello basta con pulsar dos veces sobre el parámetro requerido y cambiar su valor en la ventana de edición, de igual forma que se hacía en modo normal, salvo que en ese caso la ejecución debía estar detenida.

A la hora de detener la ejecución de la aplicación se procede de igual forma que en el modo normal mediante Debug->Halt (figura 3.44), aunque en este modo existe una particularidad, que el programa principal se detiene pero las interrupciones siguen siendo procesadas. De esta forma, las variables empleadas en el programa principal dejan de cambiar su valor en las ventanas de visualización, pero, si existe alguna asociada a una rutina de interrupción, ésta seguirá cambiando de forma continua.

En modo tiempo real se puede hacer uso de los "breakpoints" de la misma forma que se empleaban en el modo normal (Stopmode), a través de la opción "Toggle Breakpoint" del menú emergente (figura 3.25), aunque estos deben estar situado dentro del programa principal de la aplicación. Para emplearlos en las rutinas asociadas a las interrupciones es necesario configurar el monitor de tiempo real según se describe en los manuales y tutoriales de Code Composer, ya que todavía están siendo procesadas aunque el programa principal esté detenido. El uso de "Breakpoints" permite el empleo de la ejecución paso a paso dentro del programa principal de la manera habitual (StepInto, StepOut, StepOver).

Por último, puede pasarse a modo normal (Stopmode) desactivando la opción "Real Time Mode" del menú "Debug" (figura 3.44) cuando la aplicación ha sido detenida mediante "Halt". Debe aclararse que los cambios de un modo a otro deben ser siempre con el DSP detenido. En el caso de que se quiera emplear la ejecución paso a paso en modo normal (Stopmode) en programas preparados para tiempo real, debe evitarse la ejecución de la línea de llamada a la rutina de configuración del monitor "CALL MON_RT_CNFG", ya que no se podría salir de ella mediante ejecución paso a paso. Por ello, si se quiere emplear paso a paso, es recomendable ejecutar inicialmente el programa en tiempo real para que al detenerse se haya saltado ya esa línea. Si, por el contrario, se entrase en dicha rutina, se puede salir de ella mediante la opción "Run Free".

3.2.8. Notas adicionales sobre Code Composer:

A lo largo de este documento se han detallado una serie de datos y explicaciones sobre cómo afrontar el desarrollo de una aplicación utilizando Code Composer. Los métodos descritos están encaminados a la iniciación y comprensión de una persona que se adentre en el mundo de los DSP´s sin grandes conocimientos de electrónica digital y programación, por lo que muchos de los detalles descritos pueden ser susceptibles de mejoras y simplificaciones. En el documento se han hecho mención a algunos comandos sobre lenguajes de programación para un mayor entendimiento de lo que estaba desarrollando, pero no se ha entrado con detenimiento en ninguno de los lenguajes utilizados debido a su complejidad y extensión. Por ello se recomienda la consulta de la bibliografía facilitada para un mayor entendimiento de ambos lenguajes, en especial los tutoriales desarrollados a tal efecto por Texas Instruments. En lo detallado sobre este software se ha optado por mostrar métodos sencillos para la comprensión de sus funciones por el lector, pero a la hora de consultar información y ejemplos adicionales y ponerlos en práctica se han observado una serie de procesos que son interesantes de comentar para afianzar ideas sobre lo desarrollado en este documento, los cuales se enumeran a continuación:

- Code Composer dispone gran cantidad de herramientas, muchas de las cuales disponen de ventanas para configuración de sus parámetros. Estos datos suelen quedar invariables en el programa aunque se cierre el proyecto empleado y se abra otro nuevo, por lo que es recomendable reiniciar Code Composer siempre que se cambie o comience un proyecto distinto. De esta forma se evitan posibles errores en el desarrollo del nuevo proyecto.

- Un detalle no comentado en el documento y que puede ser útil para el desarrollo de aplicaciones complejas es el uso de la opción de copia del entorno de trabajo File->Work Space. Con esta opción se puede grabar o cargar una configuración de dicho entorno. Esto permite al programador guardar la configuración de Code Composer en un momento determinado: ventanas de visualización activas, proyecto desarrollado, configuraciones varias, etc., evitando así comenzar nuevamente el proyecto desde cero. Esta opción únicamente afecta a Code Composer, por lo que el DSP debe ser nuevamente cargado con el archivo ".out" resultante de dicho proyecto para poder emplear alguna de las herramientas del software.

- A la hora de comprobar el funcionamiento de proyectos de ejemplo, como el del led DS2, se comprobó que dicha aplicación entraba en los bucles de retardo y no salía de ellos. Se hicieron distintas comprobaciones y se determinó que un simple bucle FOR nunca llegaba a terminarse, ya que en algún punto intermedio de su ejecución las variables empleadas volvían a su valor inicial, con lo que se entraba en un bucle infinito. Debido a este mal funcionamiento se incluyó en los proyectos de ejemplo el archivo extraído "Boot.asm", con el que el problema desaparecía. Después de analizarlo y hacer distintas pruebas se determinó que la porción de código que solucionaba el problema era:

```
LDP     #0E0h          ; Load Data Page (WD)
SPLK  #006Fh, WD_CNTL   ; Disable WatchDog
```

la cual configura el funcionamiento el Watchdog y lo deshabilita mediante su registro de control WD_CNTL. Por ello, después de muchas pruebas, se concluyó que el causante de este funcionamiento anómalo era el temporizador de seguridad del DSP Watchdog. Éste dispositivo del DSP se encarga de reiniciar el DSP si no detecta un funcionamiento normal del mismo. Por ello, si está habilitado, debe pasarse información a dicho dispositivo de forma periódica para evitar un reset del DSP. Teóricamente, el watchdog de nuestra tarjeta estaba deshabilitado mediante el jumper de configuración JP2, el cual estaba situado en la posición 1-2 (watchdog disable) según la información facilitada para el DSK. Se analizó la información acerca de su registro de control y se comprobó que su bit número 6 configura la habilitación y deshabilitación del mismo, y que esto sólo es posible cuando el jumper JP2 esté en la posición de deshabilitado. De esto y de las pruebas realizadas se deduce que el jumper JP2 tiene dos funciones, según su posición, no muy bien definidas en los catálogos, y son:

- *JP2 en posición 1-2:* Permite la habilitación y deshabilitación del watchdog por software según sea necesario. No lo deshabilita el jumper, sino la aplicación.
- *JP2 en la posición 2-3:* El mismo jumper habilita el watchdog sin posibilidad de deshabilitarlo por software.

Por todo ello se introdujo la línea de comandos en ensamblador descrita anteriormente para una posición de JP2 en 1-2. De esta forma se deshabilita el watchdog y se puede prescindir de emplear el archivo "Boot.asm" para realizar aplicaciones sencillas, como se muestra en el Anexo 1. Esto no implica que no se emplee la librería "rts2xx.lib", ya que ésta es obligatoria para un entorno en C e incluye su propio módulo "Boot.asm", el cual no incluye la configuración del watchdog, por lo que se hace necesaria la inclusión de dicha línea en ensamblador.

- La división del proyecto en archivos es únicamente para simplificar las posibles modificaciones y actualizaciones. Por ello, hay archivos que pueden agruparse en uno solo. En el caso de lenguaje ensamblador, todos los archivos (boot, vectors, .h) pueden incluirse dentro del mismo archivo fuente con la aplicación principal si fuese necesario.

- En algunos proyectos, debido a su simplicidad, puede omitirse el uso del archivo "vectors.asm" si no se emplean las interrupciones del DSP. En el caso del parpadeo del led la única interrupción usada era la 0, por lo que podía haberse omitido dicho archivo. Esto no influye en el funcionamiento de la aplicación, pero es recomendable su uso. Además, al no haber especificación alguna sobre donde debe ir Code Composer para iniciar la aplicación (interrupción 0), después de hacer un reset al DSP la primera línea de código que queda marcada para ejecutarse no suele coincidir con el principio del programa. Por ello, para llevar Code Composer al punto de entrada de la aplicación (usualmente llamado c_int0) debe emplearse la opción Debug->Restart.

- En la creación de un proyecto, un paso importante es la configuración de las herramientas generadoras de código (compilador, ensamblador y enlazador). Por defecto, las opciones del enlazador siempre son "-x", "-c", "-o". La opción "-c" puede emplearse en cualquier proyecto, sea cual sea el lenguaje empleado (C o ensamblador), ya que indica que el punto de entrada al comienzo del programa se denomina "c_int0". También indica que posiblemente el proyecto contenga

archivos en C y se necesite configurar el entorno para ello, pero si no encuentra la librería "rts2xx.lib" en dicho proyecto esto último no tiene efecto. Por ello se suele emplear también esta opción en proyectos en ensamblador para emplear el mismo nombre para el punto de entrada del programa (c_int0), aunque en este tipo de proyectos se puede omitir "-c" y llamar a dicho punto "start" por ejemplo.
Por otro lado, una de las opciones que puede emplearse tanto en el compilador como en el ensamblador es "-g", la cual permite depurar y seguir la aplicación a través del código escrito (en C y ensamblador) en lugar de emplear la ventana "Dis-Assembly". Esta opción se utiliza por defecto en el compilador de lenguaje C, ya que en este lenguaje es muy útil visualizar el código escrito que se está ejecutando al ser más sencillo que el ensamblador que muestra la ventana "Dis-Assembly". Esta opción no suele emplearse en proyectos en ensamblador, ya que la ventana "Dis-Assembly" muestra exactamente la misma información que el archivo. Por todo esto, para poder depurar empleando "Breakpoints" o "Pobre Points" en los archivos fuente utilizados (en C o ensamblador) debe configurarse la opción "-g" en las herramientas generadoras de código. Si ésta opción no se emplea, los "Breakpoints" o "Pobre Points" sólo podrán situarse en la ventana "Dis-Assembly".

- A la hora de depurar un código puede ser interesante emplear la opción de visualizar el código en modo mixto (C y ensamblador). Esta opción tiene la limitación de no permitir la edición del archivo mostrado. Por ello, para poder modificar el archivo debe volverse a la visualización en modo "Source".

- Es recomendable a veces ejecutar inicialmente la aplicación principal para que las variables empleadas se inicien. De esta forma, si se están monitorizando o representando gráficamente se evitan lecturas erróneas de datos.

- En la depuración de un programa se suele dar la necesidad de tener que cambiar el valor de un parámetro del mismo. Una manera de hacerlo es variar el cogido y rehacer el archivo ".out" para cargarlo en el DSP, pero esto es relativamente largo. Para simplificar estos casos Code Composer tiene la opción "Patch Assembly". Esta opción permite cambiar una línea del programa en la ventana "Dis-Assembly", y por tanto modifica el programa introducido en el DSP directamente sin tener que rehacer todo el proceso. Para emplearla es necesario tener algunos conocimientos de lenguaje ensamblador. Se accede a ella pulsando con el botón derecho del ratón sobre la ventana "Dis-Assembly" y eligiendo "Patch Assembly" del menú emergente. Esto abre una ventana en la que debe indicarse el número de línea a modificar y escribir completa la nueva línea que se quiera introducir.

- En la ventana "Watch Window" se pueden visualizar distintas variables y zonas de memoria empleadas en la aplicación, como se mostró en apartados anteriores. El formato de introducción de dichos objetos en la ventana de visualización depende del tipo de variable que sea. En el caso de variables de ensamblador como la señal "test", la expresión a introducir puede ser de tres tipos:

    - test: Al introducir este dato "Watch Window" muestra la dirección en memoria de esa variable en hexadecimal.

- *test: Con este formato se muestra el valor de dicha variable como un entero sin signo (de 0 a 65535).
- *(int*)test: De esta forma se muestra el valor de la variable como un número entero con signo (de +32768 a –32767).

- La monitorización en tiempo real es bastante relativa, ya que existe una velocidad límite de muestreo. Además influye en el tiempo de ejecución de la aplicación provocándole retardos y desincronizaciones, en mayor medida cuantas más ventanas y variables se estén visualizando en tiempo real. Este efecto puede reducirse aumentando la frecuencia de refresco de las ventanas de visualización. Por otro lado, los archivos incluidos en el proyecto que contienen el programa monitor pueden ser configurados, con lo que posiblemente pueda paliarse estas deficiencias.

3.3.SOFTWARE DE DESARROLLO VISSIM EMBEDDED CONTROLS DEVELOPER FOR TI C2000:

3.3.1. Introducción:

VisSim es una aplicación especialmente orientada a la simulación y el modelado de sistemas dinámicos complejos. El programa combina un entorno de diseño de modelos orientado a bloques para la definición del sistema del proceso a simular con un potente motor de simulación. La interfaz visual de VisSim ofrece al usuario un método muy simple para desarrollar y construir, modificar y mantener modelos complejos. El motor de simulación proporciona rápidas y precisas soluciones para diseños de sistemas lineales, no lineales, sistemas continuos y discretos e híbridos. Si los requerimientos del diseño se extienden más allá de lo contemplado en la librería de bloques de VisSim, se pueden incluir bloques de usuario escritos en C, Fortran o Pascal.

Para el diseño de sistemas de control, VisSim provee de un entorno totalmente integrado que permitirá realizar todas las tareas de simulación y modelado sin escribir una sola línea de código. La versión profesional de VisSim (Professional VisSim), incluye el programa VisSim Viewer; se trata de una versión ejecutable de VisSim que permitirá distribuir todos los modelos de VisSim a usuarios finales que no dispongan de la versión de desarrollo. Mediante el Viewer podrán ejecutarse las simulaciones, cambiar los parámetros de los bloques de la simulación y analizar escenarios del tipo ‘que pasaría si...’ totalmente interactivos.

La familia de productos VisSim incluye el programa principal de simulación de sistemas dinámicos -Professional VisSim- además de varios módulos optativos que permiten conectarse con hardware externo y realizar "hardware-in-the-loop" (VisSim/Real Time), diseñar prototipos DSP y sistemas empotrados (VisSim/DSP) y generar automáticamente código ANSI C (VisSim/C-Code). Además el fabricante Visual Solutions ha lanzado también varios productos basados en el producto base VisSim que incorporan librerías de bloques específicos para diferentes campos de la simulación. Se trata de VisSim/Comm (Communication Systems Design Software) y VisSim/DPCS (Dynamic Production Control Simulator). El primero de ellos está orientado al prototipaje rápido de sistemas de comunicaciones analógicos, digitales y sistemas híbridos (incluye bloques para diseño de filtros analógicos y discretos, filtros adaptativos, matemática compleja y bloques para simulación de sistemas de

comunicaciones completos como modulción/demodulación, codificación/decodificación, canales, etc); el segundo está pensado para ingenieros implicados en el control de procesos, permitiendo monitorizar y conocer condiciones actuales de procesos en planta, así como la simulación de procesos dinámicos.

Existen otra serie de productos complementarios que aumentan las funcionalidades de las herramientas de simulación. Entre ellos está VisSim/Analyze para realizar análisis frecuencial y linealización de sistemas, VisSim/Neural-Net permite realizar identificación de sistemas no lineales y reconocimiento de patrones mediante redes neuronales y VisSim/Optimize aumenta las posibilidades de optimización de parámetros ya disponibles con la herramienta básica.

Destacar los últimos productos de VSI enfocados al diseño (VisSim/Motion) e implementación (VisSim - Embedded Controls Developer v5.0 for TI C2000) de sistemas de control de movimiento sobre DSPs de Texas Instruments, los cuales se detallan a continuación.

3.3.2. Instalación de VisSim - Embedded Controls Developer for TI C2000:

Ésta versión especial del programa VisSim incluye los módulos "Vissim/DSP" y "Vissim Motion". Al igual que los demás versiones, puede ser comprada al fabricante o descargada de la página oficial de Visual Solutions. La versión descargada es para 60 días de prueba, pero tiene algunos bloques limitados. Su instalación se realiza del mismo modo que cualquier aplicación que trabaje en un entorno Windows, por lo que sólo hay que seguir las instrucciones del programa de instalación. Entre ellas se pide al usuario que seleccione los modelos de procesadores DSP que va a emplear de entre los que soporta el programa: C24x, C2407 y F28xx. También pide al usuario el número de serie para la versión comprada. Si es de prueba el número de serie es "TRIAL". Hay que destacar que en las múltiples pruebas que se han realizado, para la correcta instalación del programa bajo Windows 9x, es necesario instalar el programa y reinstalarlo encima posteriormente, ya que en la primera instalación únicamente se instala el programa VisSim, y en la reinstalación se añaden los módulos de DSP y Motion. Por el contrario, si se instala bajo Windows XP, el programa se instala correctamente en una sola vez.

Si el programa instalado es una versión comprada al fabricante, el número de serie introducido en la instalación no es suficiente para el completo uso del programa sin restricciones. Los programas basados en VisSim están diseñados para que cada módulo específico que se le añada (Motion, DSP, etc) deba ser habilitado mediante la compra de su licencia. Por ello, en la carpeta "VisSim – Software" añadida en el menú de inicio de windows, existe un programa denominado "VisSim License Manager", el cual se encarga de la habilitación de las licencias de los distintos módulos que incorpora VisSim (figura 3.50). En este programa se indican los bloques limitados y módulo habilitados en la presente instalación además de información sobre el sistema en el que está instalado. Para habilitar las licencias es necesario mandar por correo electrónico unos códigos que proporciona VisSim al pulsar el botón "Enable License...". Esto abre una ventana (figura 3.51) en la cual se muestran unos códigos resultados de la información de la instalación y del sistema, por los que son únicos para el PC en el que se ha instalado. Estos códigos deben ser enviados a Visual Solutions, quienes responden facilitando otros tres códigos que deben introducirse en esta misma ventana y que habitan finalmente las licencias de los distintos módulos instalados.

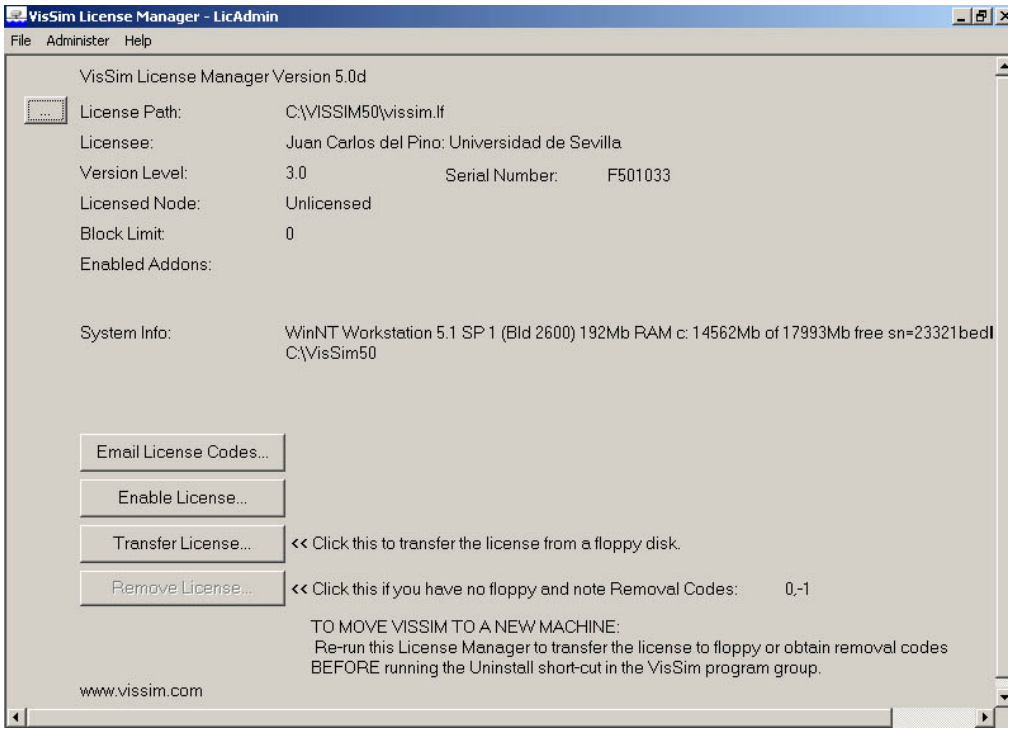


Figura 3.50. License Manager de VisSim.

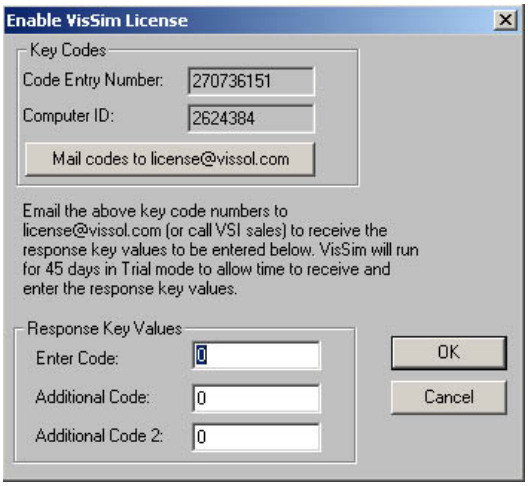


Figura 3.51. Códigos para habilitar licencias de VisSim.

Hay que destacar que, aunque este programa facilita mucho la tarea de diseño de un algoritmo para su implementación en un DSP, presenta una serie de problemas que impiden que su funcionamiento sea todo lo bueno que se espera de él. Por ello, a lo largo de este capítulo, se encontrarán comentarios respecto a diversos problemas que interfieren en el uso del programa. Uno de estos problemas es el de la instalación comentada anteriormente, la cual se desarrolla de distinta manera según el entorno Windows que se emplee, sin tener explicación razonable por parte del proveedor .

Conviene recordar que, aunque el programa VisSim - Embedded Controls Developer v5.0 for TI C2000 está enfocado para el diseño de algoritmos para DSP´s, contiene todas las funcionalidades del entorno de trabajo VisSim, por lo que es posible realizar y simular otros tipos de aplicaciones que no estén destinadas a estos procesadores.

3.3.3. Entorno de trabajo VisSim:

El entorno de trabajo de VisSim tiene la misma estructura que cualquier programa que trabaje bajo un entorno Windows. Entre ellas destacan (figura 3.52):

- ***Barra de menús e iconos:*** En ella se sitúan las principales funcionalidades del programa. A través de los menús se tiene acceso a todos las opciones y aplicaciones de VisSim, como son los distintos bloques de funciones, configuración, simulación, generación de código, etc. También se dispone de barras de iconos de acceso rápido a la mayoría de las funciones incluidas en los menús anteriores. Los principales iconos se corresponden con herramientas para la simulación (comentarios, gráficas, displays, etiquetas, etc.) y otras son de generación de señales o aplicaciones matemáticas.

- ***Zona principal de bloques:*** Esta es la zona de trabajo en la que se sitúan y conectan los diversos bloques para la implementación de un algoritmo y su simulación. En ella se muestran todos los bloques que componen la aplicación además de las gráficas, displays, comentarios, etc que sean necesarios para la validación del algoritmo.

- ***Zona de listado de bloques:*** En esta zona se muestra en forma de estructura en árbol la asociación y composición de los distintos bloques que conforman el algoritmo, ya que VisSim permite crear un único bloque a partir de un conjunto de ellos.

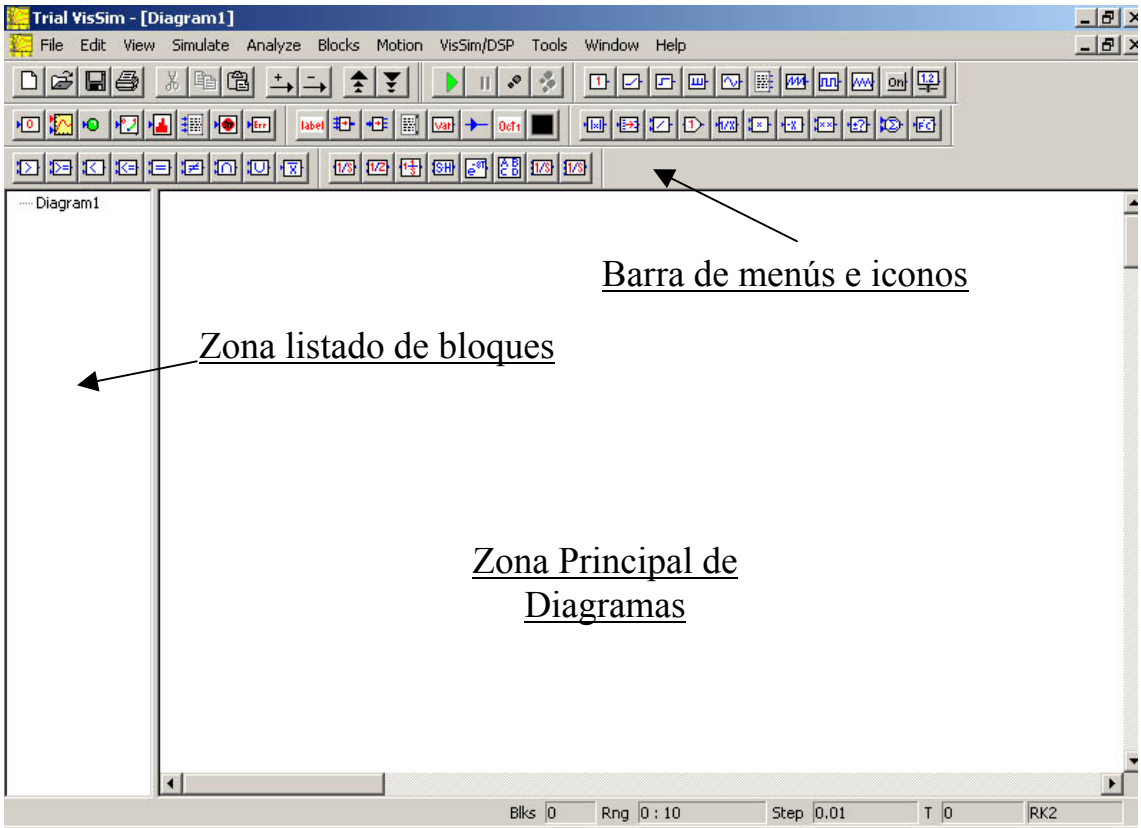


Figura 3.52. Entorno de trabajo en VisSim.

3.3.4. Trabajando con VisSim:

Como se ha comentado anteriormente, el entorno de trabajo de VisSim permite realizar algoritmos y aplicaciones mediante la interconexión de distintos bloques funcionales. Para ello, VisSim dispone de una serie de elementos que permiten realizar todo tipo de aplicaciones, desde matemáticas hasta de interfaz con un elemento exterior. Todos estos elementos se encuentran en los distintos menús de la barra superior del programa, incluso algunos tienen su propio icono de acceso rápido (figura 3.52). Los menús principales de los que consta VisSim son los siguientes:

- **Menú File:** En él se incluyen las opciones básicas de cualquier programa bajo entorno Windows, como es la apertura y almacenamiento de diagramas, opciones de impresión, etc.
- **Menú Edit:** que incluye las opciones típicas de edición en windows, preferencias de aspecto del programa, propiedades de los bloques, etc.
- **Menú Simulate:** Aquí se accede a las distintas opciones de configuración de la simulación de diagramas, además del inicio de la simulación y su detención.

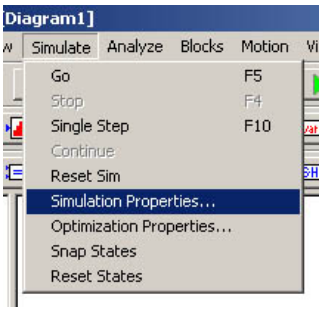


Figura 3.53. Menú Simulate de VisSim.

- **Menú Blocks:** Aquí se encuentran la mayoría de bloques que se emplean para la realización de diagramas, como son: bloques de operaciones aritméticas, booleanas, de punto fijo, sistemas lineales y no lineales, de matrices, etc.

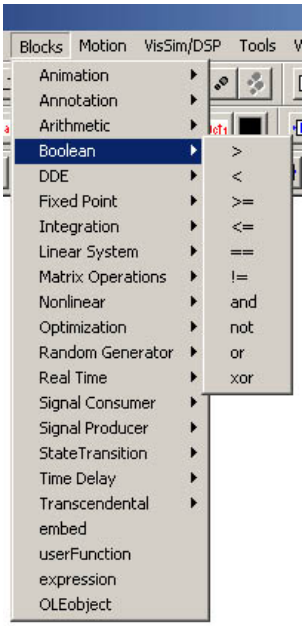


Figura 3.54. Menú Blocks de VisSim.

- **Menú View:** que permite la visualización de las distintas barras de herramientas de iconos rápidos.
- **Menú Analyze:** en el que se incluyen distintas herramientas para el análisis de sistemas de control como son raices, Nyquist, etc.
- **Menú Motion:** Este es uno de los módulos añadidos al programa base VisSim. En el se encuentran bloques funcionales que simulan distintos elementos empleados en el movimiento de máquinas rotativas, como son: Motores, filtros, cargas, sensores, etc.

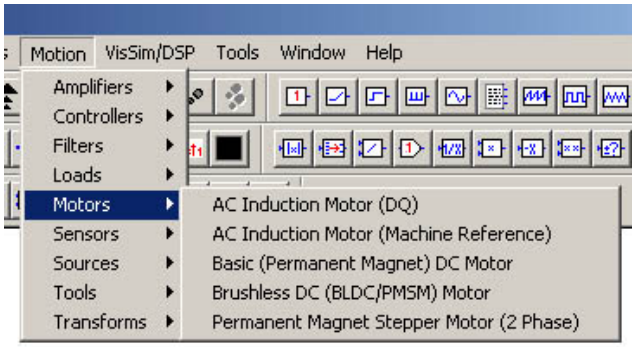


Figura 3.55. Menú Motion de VisSim.

- **Menú VisSim/DSP:** Es otro módulo añadido a VisSim, el cual contiene los bloques que hacen de interfaz externa con los distintos periféricos de los tres tipos de DSP´s que soporta el programa, como son: entradas/salidas analógicas y digitales, generación de señales PWM, captura de señales procedentes de encoders, etc.

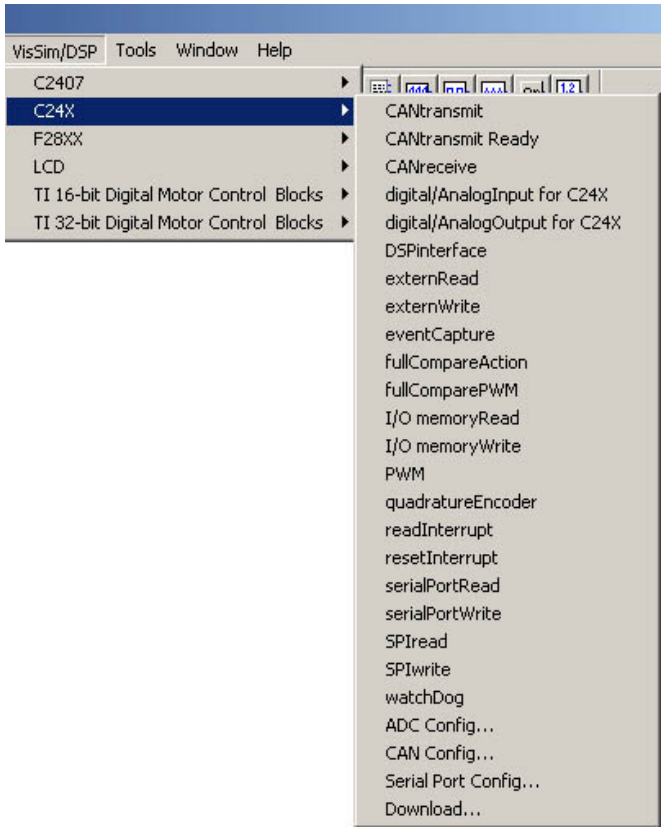


Figura 3.56. Menú VisSim/DSP de VisSim.

- **Menú Tools:** el cual contiene las herramientas de generación de código en lenguaje C y su compilación. Además permite la importación de elementos de otras aplicaciones.

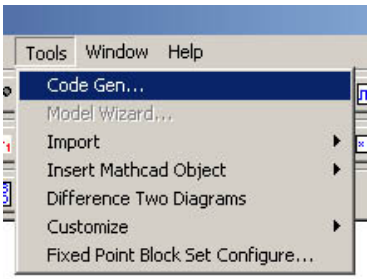


Figura 3.57. Menú Tools de VisSim.

- **Menú Windows:** para la selección entra las distintos diagramas que se encuentren abiertos.
- **Menú Help:** que contiene loas ayudas necesarias para el empleo de este programa.

Para desarrollar el diagrama de una aplicación únicamente es necesario situar e interconectar los distintos bloques que el usuario crea conveniente. Esto se realiza fácilmente seleccionando el bloque necesario de los iconos rápidos o de los distintos menús y situándolo en la zona del diagrama conveniente. La interconexión de los bloques se realiza mediante el cableado de las salidas de los bloques a las entradas de otros elementos. Para ello basta con aproximar el cursor del ratón a la salida del bloque a conectar, momento en el que el cursor se transforma en una flecha negra vertical. Pulsando en ese momento el botón izquierdo del ratón y arrastrando hasta la salida que se quiere conectar se realiza la conexión entre ambos bloques (figura 3.58).

Figura 3.58. Interconexión de bloques.

La desconexión de bloques se realiza de manera semejante, únicamente aproximando el cursor a la entrada del bloque a desconectar, pulsando y arrastrando hacia una zona vacía del diagrama.

Figura 3.59. Selección de bloques en VisSim.

Una vez situados y conectados los distintos bloques que conforman el diagrama, a veces resulta cómodo la agrupación de uno o varios bloques en un único bloque compuesto. Para ello es necesario seleccionar los bloques afectados, lo que se consigue pulsando con el ratón en un espacio vacío del diagrama y arrastrar el cursor hasta seleccionar los bloques necesarios (figura 3.59). Una vez seleccionados los bloques se pulsa el botón derecho del ratón, lo que abre un menú desplegable donde se selecciona la opción "Create Compound" para generar el bloque compuesto (figura 3.60).

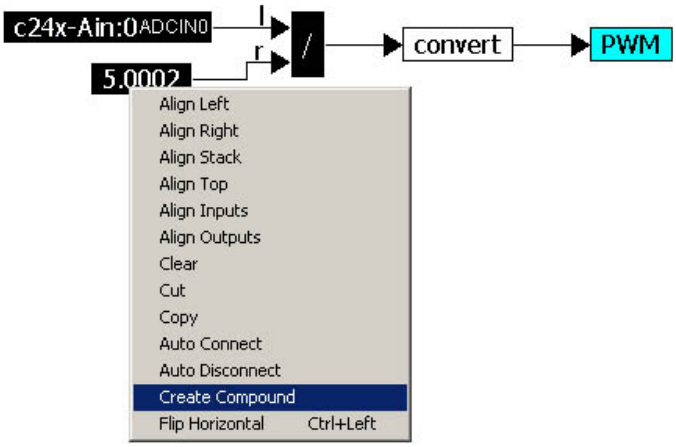


Figura 3.60. Creación de bloques compuestos.

Esto abre una ventana en la que se configuran las opciones del bloque a crear, entre ellas el nombre que se quiere dar al bloque creado (figura 3.61). De esta manera se crea un bloque único con el mismo numero de señales de entrada y salida que tenían el conjunto de bloques que lo forman (figura 3.62).

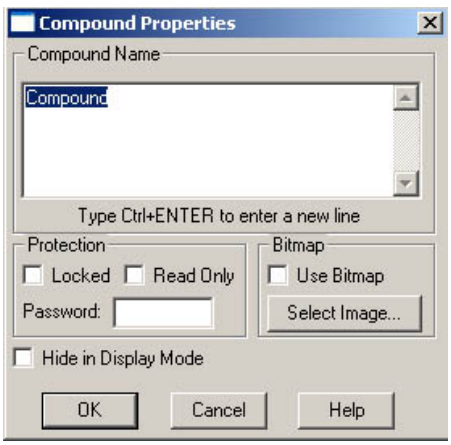


Figura 3.61. Configuración de bloque compuesto.

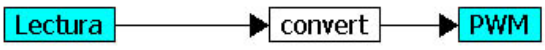


Figura 3.62. Bloques compuestos.

Para ayudar a la creación de diagramas en los que pueda repetirse el uso de un determinado valor o variable, existe en VisSim la posibilidad de emplear etiquetas con las que identificarlas. De esta forma se asocia una etiqueta a un valor o variable y únicamente se emplea su etiqueta para el diseño del diagrama, de forma que si debe hacerse una modificación de su valor, haya un solo lugar en el algoritmo donde se realice la modificación. Esto se emplea mucho para definir constantes empleadas en los algoritmos que sean susceptibles de ser modificadas para optimizar la ejecución del mismo. Su empleado es muy sencillo, bastando con situar una etiqueta conectada a la variable deseada mediante el icono .Dándole un nombre a esta etiqueta se asocia con dicho valor, de forma que en los lugares donde sea necesario se utilice esa etiqueta en vez del valor. Por ejemplo, para definir constantes empleadas en un diagrama se puede realizar lo que muestra la figura 3.63.

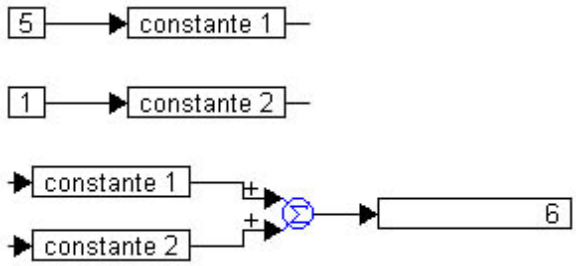


Figura 3.63. Empleo de etiquetas en VisSim.

en el que se definen dos constantes que vana ser sumadas. Las constantes se sitúan en el diagrama mediante el icono (o la opción "const" en menú blocks), con el que pulsando dos veces sobre el bloque situado se configura el valor de dicha variable. En caso de realizar modificaciones, sólo es necesario realizarlas en estos últimos bloques. También es habitual emplear deslizadores como entradas a determinados bloques, con los cuales se modifica rápidamente el valor de la constante. Estos se sitúan mediante el icono de la barra superior, situándose el bloque de la figura 3.64. El rango de valores que puede tomar el deslizador se indica en su ventana de configuración (figura 3.64), en la que también se indica el incremento que emplea al desplazar el deslizador.

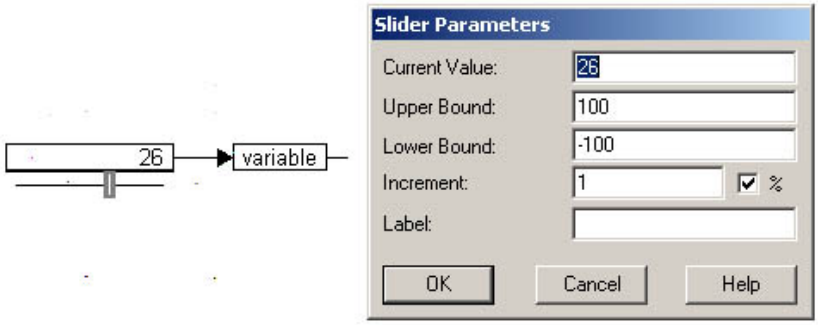


Figura 3.64. Deslizador y configuración en VisSim.

De esta manera se pueden modificar fácilmente valores de constantes empleadas en el algoritmo para la comprobación de la integridad y estabilidad del mismo, modificar parámetros de control de un sistema, etc.

También se disponen otra serie de elementos que suelen emplearse en programación de algoritmos, como son:

- ***Generadores de señales:*** incluidos en el menú "Blocks" o en la barra superior de iconos de acceso rápido, en las que se dispone de funciones cuadradas, triangulares, seno, etc.

- ***Operaciones Booleanas:*** disponibles en el menú "Blocks" y en la barra superior de iconos. En ellas se incluyen operaciones de comparación, negación, etc. Las de comparación generan un '1' cuando la comparación es verdadera y '0' cuando es falsa.

- ***Transformación de formatos y funciones matemáticas:*** Se encuentran en el menú "Blocks" o en la barra de iconos superior. Dispone de bloques para sumas, multiplicaciones, divisiones, etc así como de cambio de formato de la variable empleada.

- ***Elementos en transformada de Laplace:*** Incluidos en el menú "Blocks" o en la barra de iconos superior. Incluye bloques de integración, transformada, delays, etc.

Para una mayor comprensión del algoritmo, suele ser necesaria el etiquetado de variables para su identificación, así como de diversos comentarios que clarifiquen el funcionamiento del mismo. Esto puede realizarse mediante los iconos "label"  y "Coment"  . El primero sirve para identificar bloques o variables, mientras que el segundo genera una ventana donde desarrollar un texto explicatorio (figura 3.65).

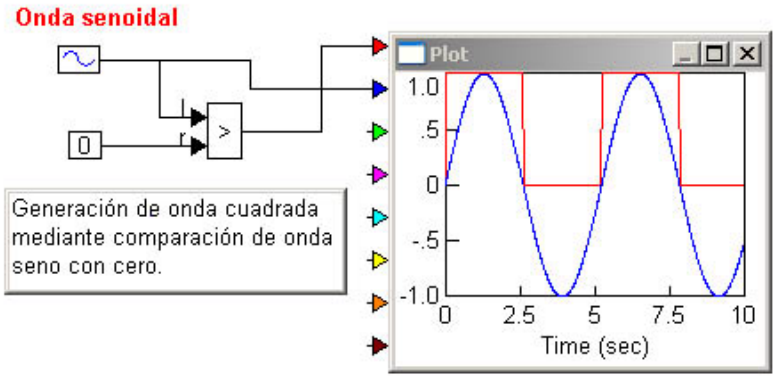


Figura 3.65. Comentarios y representación gráfica.

Cuando el diseño del diagrama ha finalizado, es el momento de comprobar su funcionamiento mediante su simulación. Para ello VisSim dispone de múltiples herramientas que permiten comprobar el correcto funcionamiento de un algoritmo: gráficas, displays, histogramas, etc. Por ejemplo, para situar una gráfica en el diagrama en el que representar la variación de una variable, se emplea el icono "Plot" de la barra superior de iconos de acceso rápido. Ello sitúa un gráfico en el diagrama como el de la figura 3.65. Éste gráfico puede configurarse mediante su ventana de configuración, la cual aparece al pulsar dos veces sobre el gráfico (figura 3.66).

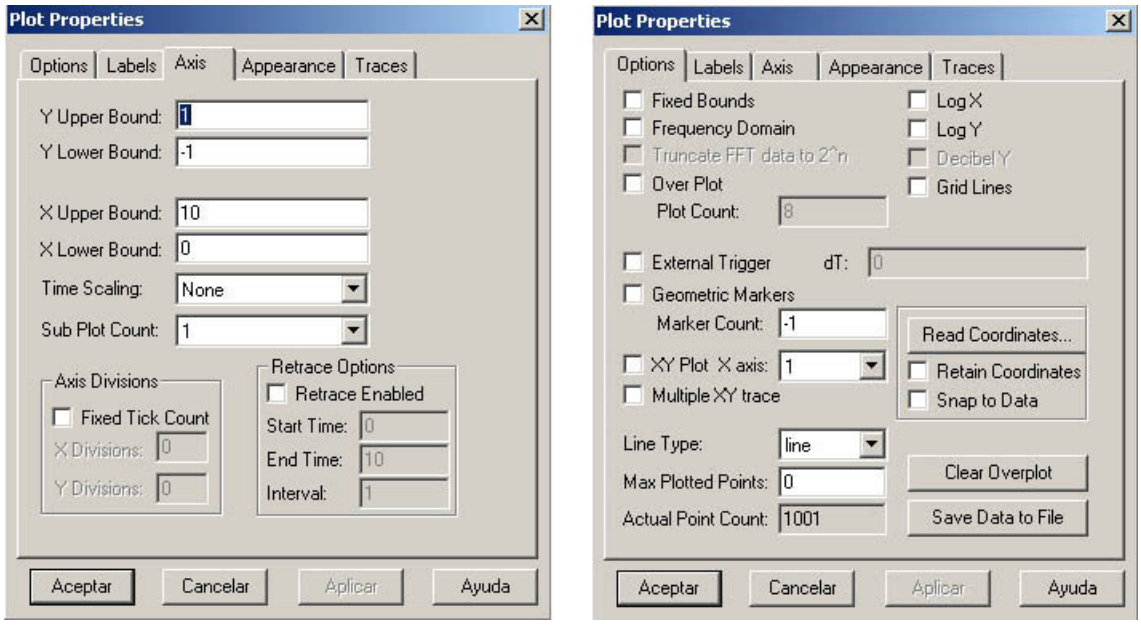


Figura 3.66. Configuración de gráficas en VisSim.

Por defecto está configurado para emplear valores automáticos en los ejes de coordenadas, pero pueden hacerse fijos mediante la casilla "Fixed Bounds", indicando posteriormente los valores de ambas coordenadas en la pestaña "Axis". El rango de valores que toma la variable del eje X debe ser configurada en las opciones de simulación, ya que ahí se indica el tiempo que se ejecutará la aplicación.

Para mostrar el valor de una variable o el resultado de una operación se suelen emplear displays, los cuales se sitúan mediante el icono , como el mostrado en la figura 3.63 que muestra el resultado de una suma. De esta forma se monitoriza también la evolución de una determinada señal del algoritmo.

Para validar el modelo desarrollado VisSim dispone de un potente simulador, que permite comprobar el correcto funcionamiento del diagrama antes de su implementación. Las opciones de simulación se disponen mediante la opción "Simulation Properties" del menú "Simulate" de la barra superior (figura 3.53). Esto abre la ventana de configuración de la simulación (figura 3.67), en la que se configuran el tiempo de simulación, si ésta se realiza en tiempo real (casilla marcada) o si se reinicia automáticamente a simulación al llegar al tiempo establecido.

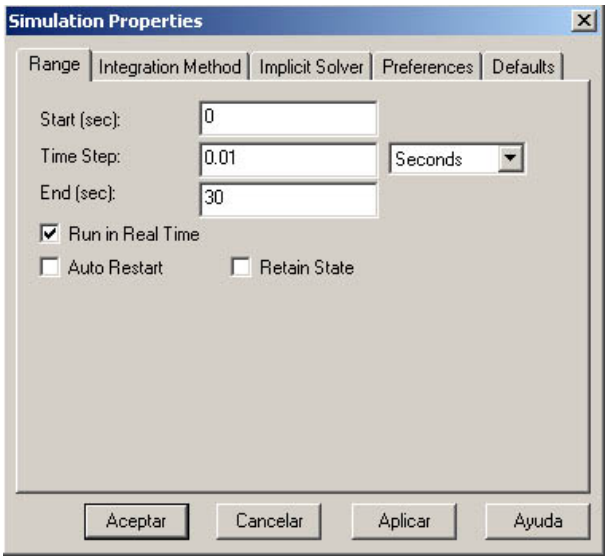


Figura 3.67. Opciones de simulación.

Una vez configuradas las opciones de simulación, se procede a la ejecución de la misma mediante la opción "Go" del menú "Simulate" o pulsando el icono de la barra superior de iconos. Para detener la simulación basta con pulsar el icono de la barra superior .Durante la simulación se pueden ver las variaciones que sufren las distintas señales y variables en los displays del diagrama, así como su evolución en las gráficas añadidas. La simulación en VisSim puede realizarse de dos modos distintos: en modo interactivo, en el cual la mayoría de la simulación la realiza el propio PC excepto las acciones que producen dispositivos exteriores, y en modo compilado, las cuales se realizan descargando el diagrama generado a un dispositivo externo para la simulación en el destino final de la aplicación. Las diferencias entre ambos modos se detallan en el siguiente apartado.

3.3.5. VisSim/DSP y VisSim Motion:

Para el diseño de aplicaciones de control de motores y su implementación en procesadores DSP´s, VisSim dispone de dos módulos específicos que incluyen variedad de bloques funcionales con herramientas empleadas en estas aplicaciones. Por un lado VisSim/DSP contiene todos los periféricos incluidos en un DSP, los cuales hacen de interfaz externa con el procesador a través de bloques (figura 3.56). Cada uno de estos bloques se corresponden con las entradas analógicas, digitales, captura de encoders, etc incluidos en el DSP. Este módulo contiene los periféricos de tres modelos de DSP´s: para la series C24x, C2407 y F28xx. Cada uno de elles tiene su propio submenú en el menú "VisSim/DSP" del programa (figura 3.56). Por otro lado se encuentra el módulo "VisSim Motion", el cual incluye bloques que simulan el funcionamiento de diversos tipos de motores, cargas, sensores, etc (figura 3.55). Unificando ambos bloques, en el menú "VisSim/DSP" se encuentra un submenú denominado "TI 16 bits Digital Motor Control Blocks" (figura 3.68). En él se incluyen bloques que se han construido a partir del código en lenguaje C de funciones prototipos empleadas en el control digital de motores con DSP´s y desarrolladas por Texas Instruments para sus dispositivos. En ellos se incluyen modelos de distintos tipos de motores, reguladores PID, bloques de cálculo de velocidad con señales en cuadratura o QEP, generadores de señales PWM con vectores espaciales, etc.

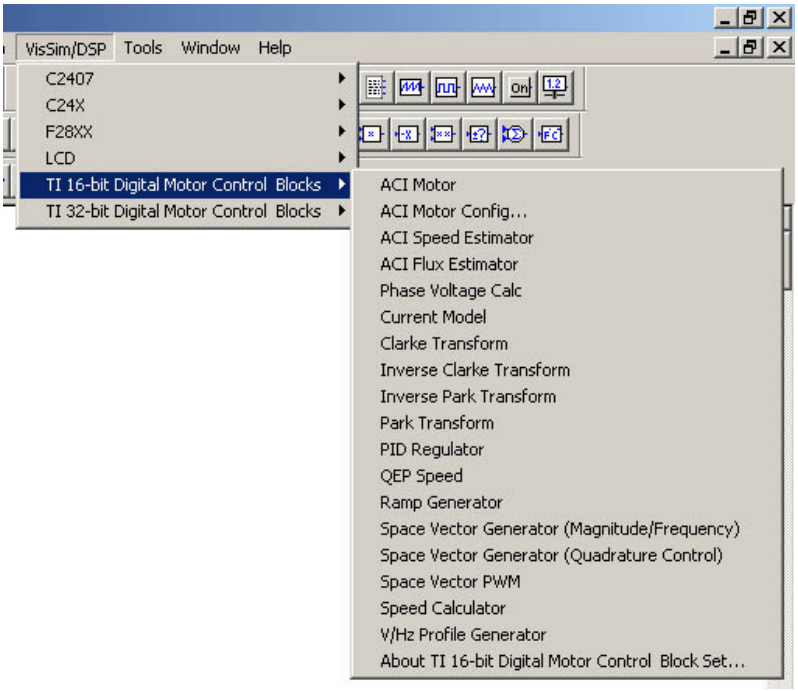


Figura 3.68. Librería de control de motores en VisSim.

Para trabajar con DSP´s en VisSim, el proceso de construcción del diagrama se realiza de la misma forma que en cualquier otro caso, salvo que algunos bloques son tomados del menú "VisSim/DSP" y que hacen de interfaz externa con el DSP empleado. Cada bloque añadido al diagrama es cableado con el resto como se indicó anteriormente, y todos ellos tienen una ventana de configuración a la que se accede pulsando dos veces sobre el bloque en cuestión. Por ejemplo, un diagrama como el de la figura 3.69, se ha realizado con el fin de generar señales PWM con el DSP, cuyo ancho de pulso es controlado por el valor de entrada del canal analógico ADCIN0 del DSP.

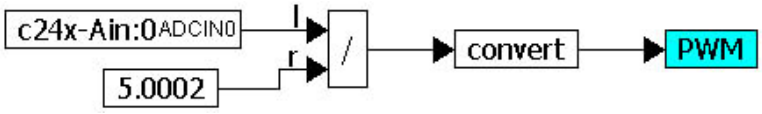


Figura 3.69. Diagrama para uso en DSP.

Dicho diagrama puede ser simulado tal cual está pulsando el botón "Go" de simulación. Esta simulación se denomina en "modo interactivo", puesto que todos los cálculos del diagrama los realiza el PC excepto la medida del canal analógico y la generación de señales PWM, las cuales la realiza el DSP. Este modo de simulación se realiza pues de forma mixta entre el ordenador y el DSP, con lo que los resultados pueden no ser los adecuados si la aplicación será implementada en el DSP. Además, existen problemas para el uso de algunos bloques específicos del DSP en modo interactivo, los cuales no funcionan correctamente en este modo de simulación (por ejemplo, el diagrama anterior puede no generar físicamente las señales PWM en las salidas del DSP). Según informaciones del fabricante de VisSim, esto se debe a problemas en el programa, el cual no ha sido suficientemente testeado, por lo que periódicamente sacan versiones revisadas en su página web.

Existe otro modo de simulación denominado “modo compilado” que es muy utilizado cuando se diseñan aplicaciones para DSP´s. Este modo consiste en generar un archivo en código C del diagrama completo o de parte de él, el cual es compilado y cargado en el DSP por el propio VisSim. De esta manera la simulación se realiza toda o en parte en el DSP, con lo que los resultados obtenidos son más fiables si éste es el destino de la aplicación. Éste modo suele producir menos errores con los bloques específicos del DSP, por lo que el fabricante recomienda el uso de este método. Como se ha comentado, esto se puede realizar con todo el diagrama o con solo una parte de él, pero debe tenerse en cuenta que si se emplea este método, no pueden emplearse bloques del DSP que no estén dentro de la parte compilada, ya que al estar fuera se ejecutarían en modo interactivo a la vez que se ejecuta un código descargado en el DSP, y así el DSP no puede trabajar.

Para realizar la simulación en modo compilado, previamente los boques que vana ser compilados deben ser agrupados en un único bloque, lo cual se realiza como se comentó con anterioridad (figura 3.60). Una vez generado el bloque compuesto de todo o parte del diagrama debe seleccionarse dicho bloque, para posteriormente ir a la opción “Code Gen...” del menú “Tools” de la barra superior (figura 3.57). Esta opción abre la ventana de configuración de la herramienta generador de códigos (figura 3.70), en la que debe indicarse el modelo de DSP en la opción “Target”, el nombre del archivo y se ruta en la opción “Result File” y seleccionar las casillas “Include Block Nesting as Comment” y “Include VisSim Communication interface”. Éstas dos últimas opciones deben estar siempre marcadas, al contario que la opción “Make Callable from User App”, la cual puede dar problemas en algunos ejemplos.

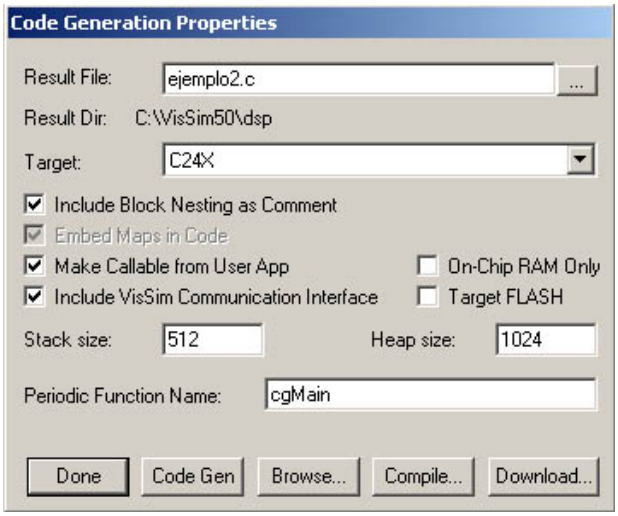


Figura 3.70. Configuración de generación de código.

Una vez configurada la herramienta, mediante el botón “Code Gen” se hace el archivo en código C del bloque seleccionado. Mediante la opción “Compile” dicho archivo es compilado, generando un archivo “.out”. En este proceso se abre una ventana de MS-DOS en la que se visualiza el proceso y se muestran los posibles errores producidos en el mismo (figura 3.71). Este archivo generado puede ser descargado al DSP mediante el botón “Download” de la ventana de configuración. Una vez terminado el proceso dicha ventana se cierra mediante la opción “Done”.

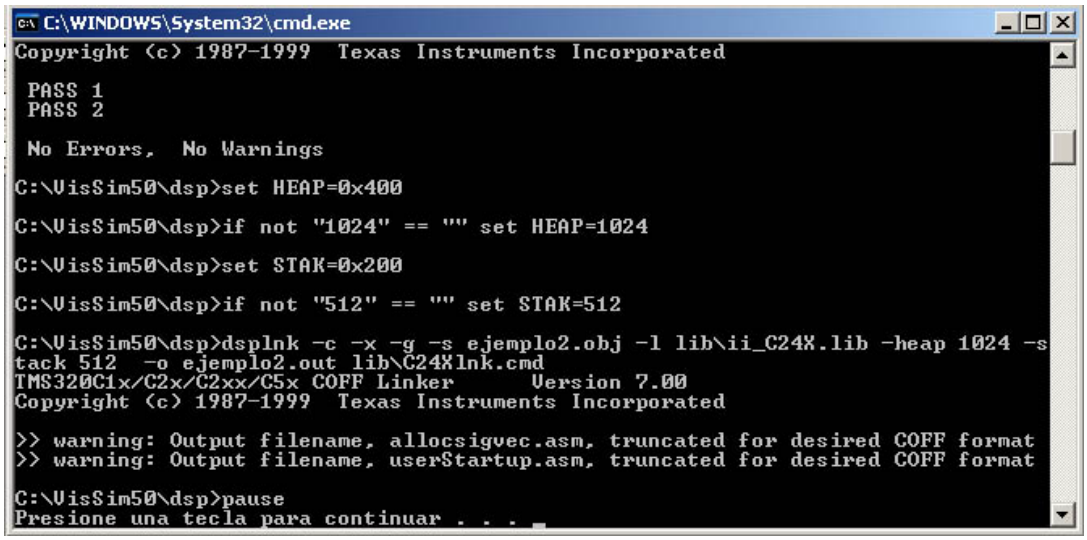


Figura 3.71. Ventana de proceso de la compilación.

Terminada la generación del archivo compilado ".out", se elimina del diagrama el bloque compuesto con el que se ha generado. Este bloque es sustituido por un nuevo bloque incluido en el menú "VisSim/DSP" denominado "DSP interface" (figura 3.72), el cual corresponde con un bloque que hace de interfaz con el DSP en el diagrama, realizando las operaciones programadas en un archivo ".out" ya compilado.

Figura 3.72. Bloque de interfaz DSP.

Para indicar el archivo compilado que se quiere ejecutar, se accede a la ventana de configuración pulsando dos veces sobre dicho bloque, abriéndose la ventana de la figura 3.73. En ella se indica el archivo ".out" a cargar en el DSP mediante el campo "DSP Execution File". También puede indicarse el número de señales de entradas y salidas que debe tener el bloque mediante el campo "Connectors".

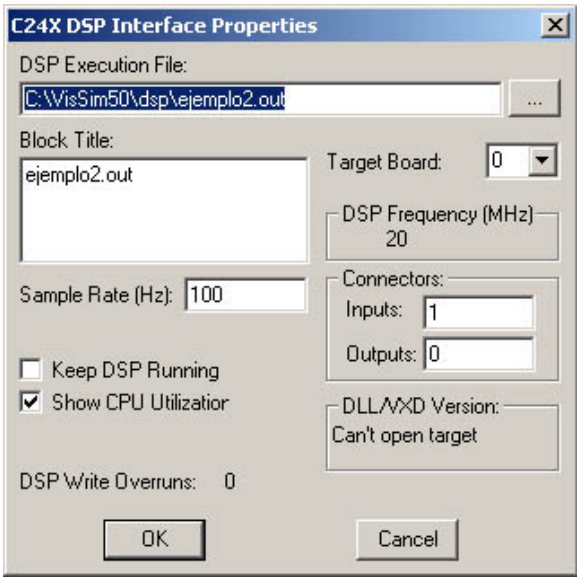


Figura 3.73. Configuración de interfaz DSP.

Si se marca la casilla “Keep DSP Running”, el DSP no detiene sus operaciones aunque se pulse el botón de finalización de la simulación. Por último, la casilla “Show DSP Utilizatior” permite dotar al bloque de una salida específica que, mediante un display, indica el porcentaje de CPU empleado por el programa (figura 3.74). De esta manera se comprueba la eficiencia del código generado.

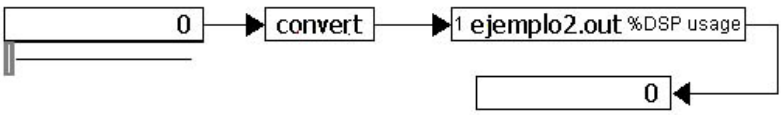


Figura 3.74. Diagrama con interfaz DSP.

A lo largo de los próximos capítulos se detallarán ejemplos de diagramas empleados con DSP´s. La mayoría pueden ser simulados en modo interactivo, el cual permite comprobar las variaciones que sufre la aplicación al variar algunos parámetro, pero puede que algunos deban ser simulados en modo compilado para su correcto funcionamiento. Este es el caso de la generación de señales PWM, las cuales se ven limitadas por el fabricante al emplear los mismos recursos para la comunicación con el DSP.

3.3.6. Problemas con VisSim:

A lo largo de las numerosas pruebas efectuadas, se han comprobado diversas incidencias en este programa. Algunas de ellas se deben a limitaciones puestas por el programador al emplear los mismos recursos que algunas de las aplicaciones para la comunicación con el DSP. Esto suele dar como resultado que el diagrama no pueda ser compilado con las opciones requeridas por el usuario, o que la simulación deba hacerse en modo compilado, con lo que algunos parámetros no podrán ser modificados durante la simulación.

Algunos de los problemas encontrados se han resuelto al obtener una versión corregida del programa en la página web de Visual Solutions, ya que resultaron ser defectos en la programación del mismo. Algunos de estos problemas fueron el mal funcionamiento de las señales de entrada y salida analógicas y digitales, así como de la generación de señales PWM. Esto último todavía sigue sin funcionar correctamente en modo interactivo, debido a que comparten recursos con las comunicaciones del VisSim. En capítulos posteriores se detallarán las limitaciones que esto conlleva.

Por todo esto, se recomienda emplear la versión más actualizada posible del programa, y en caso de duda o detectar problemas, ponerse en contacto con el fabricante para comprobar el origen del problema.