UNIVERSIDAD AUTÓNOMA DEL ESTADO DE HIDALGO

INSTITUTO DE CIENCIAS BÁSICAS E INGENIERÍAS

DESARROLLO DE APLICACIONES EN EL DSK TMS320C6711

T E S I S

QUE PARA OBTENER EL TÍTULO DE

INGENIERO EN ELECTRÓNICA Y TELECOMUNICACIONES

P R E S E N T A

PRIMITIVO EUTIMIO VÁZQUEZ MENDOZA

ASESORES DE TESIS

ING. MARÍA ANGÉLICA ESPEJEL RIVERA

DR. LUIS ENRIQUE RAMOS VELASCO

DR. JORGE CARLOS MEX PERERA

PACHUCA DE SOTO, HIDALGO. FEBRERO DE 2007

Agradecimiento

A:

Ma. Angélica y Luis Enrique.

Acrónimos y siglas

Notación	Significado	Traducción
AC97	Audio códec '97	Audio códec '97
ADC	Analog to Digital Converter	Convertidor analógico a digital
AIC	Analog Interface Circuit	Circuito de interfaz analógica
ALU	Arithmetic Logic Unit	Unidad aritmética lógica
AMR	Addressing Mode Register	Registro de modo de direccionamiento
API	Application Programming Interface	Interfaz programable para la aplicación
ASCII	American Standard Code for Information Interchange	Código estándar americano para intercambio de información
ATM	Atomic	Módulo de funciones atómicas
CCS	Code Composer Studio	Laboratorio de prácticas de código
CD-ROM	Compact Disc Read Only Memory	Memoria de sólo lectura en disco compacto
CLK	Clock	Módulo de sincronización
C-MOS	Complementary MOS	MOS complementario
CODEC	Coder-Decoder	Codificador-decodificador
COFF	Common Output File Format	Archivo de formato común de salida
COM	Component Object Model	Modelo de objetos componentes
CPU	Central Processing Unit	Unidad central de Proceso
CPUID	Identifies CPU	Identificador del CPU
CSL	Chip Support Library	Librería de soporte del chip
CSR	Control Status Register	Registro de control de estado
C62		Módulo de funciones del DSP

DAC	Digital to Analog Converter	Convertidor digital a analógico
DC	Direct Current	Corriente continua
DCC	Data Caché Control	Control de datos caché
DEV	Device	Módulo de dispositivos de control
DP	Double Precision	Doble precisión
DRR	Data Receive Register	Registro receptor de datos
DSK	DSP Starter Kid	Kid de prueba de DSP
DSP	Digital Signal Processing	Procesador de señales digitales
DSP/BIOS	DSP Basic Input/Output System	Sistema básico de entrada/salida del DSP
DXR	Data Transmit Register	Registro transmisor de datos
EDMA	Enhanced Direct Memory Access	Acceso directo mejorado a la memoria
EDMACC	EDMA Channel Controller	Controlador del canal del EDMA
EDMATC	EDMA Transfer Controller	Controlador de transferencia del EDMA
EMIF	External Memory Interface	Interfaz de memoria externa
EPROM	Erasable PROM	PROM borrable
E1		Formato europeo de transmisión digital
FADCR	Floating-Point Adder Configuration Register	Registro de configuración del sumador de punto flotante
FAUCR	Floating-Point Auxiliary Configuration Register	Registro de configuración auxiliar de punto flotante
FLOPS	Floating Point Operations Per Second	Operaciones de punto flotante por segundo
FMCR	Floating Point Multiplier Configuration Register	Registro de configuración del multiplicador de punto flotante
FTT	Fast Fourier Transform	Transformada rápida de Fourier
GBL	Global	Módulo de control global
GEL	General Extension Language	Lenguaje de extensión general
GIE	Global Interrupt Enable	Activación de interrupción global

GPS	Global Positioning System	Sistema de posicionamiento global
HST	Host	Módulo del Host
HPI	Host-port interface	Interfaz Host-puerto
HWI	Hardware Interrupt	Módulo de interrupciones de hardware
IACK	Interrupt Acknowledgment	Confirmación de interrupción
IDL	Idle	Modulo de funciones estáticas
IER	Interrupt Enable Register	Registro de activación de interrupción
IFR	Interrupt Flag Register	Registro de bandera de interrupción
IIS	Internet Information Services	Servicios de información de Internet
INUMx	Number of the Interrupt x	Número de interrupción x
I/O	Input/Output	Entrada/Salida
IRP	Interrupt Return Pointer	Puntero de retorno de interrupción
ISFP	Interrupt Service Fetch Packet	Paquete de servicio de interrupción
IST	Interrupt Service Table	Tabla de Servicio de interrupción
ISTP	Interrupt Service Table Pointer	Puntero de tabla con servicio de interrupción
JTAG	Joint Test Action Group	
LCK	Lock	Modulo de asignación
LED	Light-Emitting Diode	Diodo electroluminiscente
LOG	Log	Módulo de eventos
LPT	Printer Port	Puerto paralelo
LSB	Least Significant Bit	Bit menos significativo
MBX	Mailbox	Módulo de buzón
McBSP	Multichannel Buffered Serial Port	Puerto serial multicanal con buffer
MCR	Multichannel Control Register	Registro del multicanal
MEM	Memory	Módulo de memoria
MIPS	Millions of Instructions Per Second	Millones de instrucciones por segundo

MOS	Metal-Oxide-Semiconductor	Metal-óxido-semiconductor
MSB	Most Significant Bit	Bit más significativo
MVIP	Multi Vendor Integration Protocol	Protocolo de integración Multi Vendor
NaN	Not-a-Number	No número
NMI	Nonmaskable Interrupt	Interrupción no mascarable
NMIE	Nonmaskable Interrupt Enable	Activación de interrupción no mascarable
NRP	Non-maskable Interrupt Return Pointer	Puntero de retorno de interrupción no mascarable
OLE	Object Linking and Embeddling	Vinculación e incrustración de objetos
PaRAM	Parameter RAM	Parámetro RAM
PC	Personal Computer	Computadora personal
PC	Program Counter	Contador del programa
PCC	Program Caché Control	Control del programa caché
PGIE	Previous GIE Value	Valor GIE previo
PIP	Pipes	Módulo de canalizaciones
PRD	Periodics	Módulo de funciones periódicas
PROM	Programmable Read-Only Memory	Memoria programable de sólo lectura
QUE	Queue	Módulo de estructuras de colas
RAM	Random Access Memory	Memoria de acceso aleatorio
RBR	Receive Buffer register	Registro de recepción del buffer
RCER	Receive Channel Enable Register	Registro de la activación del canal de transmisión
RCR	Receive Control Register	Registro de control de recepción
Revid	Identifies CPU revision	Identificador de revisión del CPU
RISC	Reduced Instruction Set Computer	Computadora con conjunto de instrucciones reducido
RSR	Receive Shift Register	Registro receptor de desplazamiento
RTDX	Real-Time Data Exchange	Intercambio de datos en tiempo real
SCSA	Signal Computing System Architecture	Arquitectura para transmitir señales de voz y vídeo
SEM	Semaphore	Módulo de semáforos

SIO	Stream I/O	Módulo de transmisión de entrada/salida
SP	Single Precision	Simple precisión
SPCR	Serial Port Control Register	Registro de control del puerto serial
SRAM	Static Random Access Memory	RAM estática
SRGR	Sample Rate Generator Register	Registro del generador de rango de muestras
STS	Statistic	Módulo de estadísticas
SVGA	Super Video Graphics Array	Arreglo gráfico de super video
SWI	Software Interrupts	Módulo de interrupciones de software
SYS	System	Módulo de dispositivos del sistema
TI	Texas Instruments	
TRC	Trace	Módulo de trazo
TSK	Task	Módulo de tareas
T1		Sistema del T-Portador
VSLI	Very Large Scale Integrated	Circuito integrado de escala muy grande
VLIW	Very Long Instruction Word	Palabra de instrucción de gran longitud
XCR	Transmit Control Register	Registro de control de transmisión
XSR	Transmit Shift Register	Registro transmisor de desplazamiento

Índice general

Índice de tablas

Índice de figuras

Listings

Resumen

El avance tecnológico de nuestros días, ha proporcionado una serie de dispositivos aplicables a los sistemas de procesamiento de señales, cada vez más complejos. Este trabajo de tesis pretende contribuir, al manejo de un DSK, explicando su arquitectura, sus recursos, herramientas y su programación básica.

El uso de la tarjeta diseñada por Texas Instruments, para desarrollos de alto rendimiento, proporciona gran rapidez de ejecución y una programación accesible, dos ventajas lo suficientemente claras para mostrar la primacía de este dispositivo, induciendo al empleo del mismo en prototipos y aplicaciones.

La tesis se presenta como guía de consulta rápida para el uso del DSK TMS320C6711 en aplicaciones de procesamiento digital de audio y voz.

Capítulo 1

Introducción

Los procesadores digitales de señales ganaron popularidad en los años setentas con la introducción de la tecnología de estado sólido. Un DSP comprende los fundamentos matemáticos y algorítmicos que describen como procesar, en un ambiente de computo digital, información asociada a señales provenientes del mundo real. Entonces un DSP es un sistema electrónico que realiza un procesado digital de señales físicas, obtenidas, a menudo, con el empleo de transductores y convertidores analógico-digitales.

Estrictamente hablando, el término DSP se aplica a cualquier chip que trabaje con señales representadas de forma digital. En la práctica, el término se refiere a microprocesadores específicamente diseñados para realizar procesado digital de señales.

Actualmente el mercado se ha extendido enormemente en cuanto a la oferta de DSPs. Existen diversos fabricantes, cada uno con un tipo especial y particular de arquitectura, uso y/o aplicación. Entre los que destacan: Texas Instruments, con la serie TMS320C6000; Motorola, con las series DSP56000, DSP56100, DSP56300, DSP56600 y DSP96000; Lucent Technologies, con las series DSP1600 y DSP3200; y Analog Devices,con las series ADSP2100 y ADSP21000.

1.1. Objetivo general

Obtener una guía rápida para el desarrollo de aplicaciones en el DSK TMS320C6711 en el área de procesamiento de audio y voz.

1.2. Objetivos específicos

- Analizar la arquitectura del DSK TMS320C6711.
- Manejar las herramientas del DSK TMS320C6711.
- Analizar métodos y algoritmos de programación enfocados al DSK TMS320C6711.
- Utilizar los recursos que ofrece el DSK TMS320C6711 para la aplicación en áreas de procesamiento de audio y voz.

1.3. Justificación

El hombre desde las primeras etapas de la vida hace uso de señales, por ejemplo el llanto para demostrar alguna necesidad en la etapa infantil. Estas señales tienden a ser más complejas durante nuestra existencia, la voz puede cambiarse a otras señales como la escritura y después interpretarse. Los símbolos y los colores son señales visuales, que también provocan reacciones en nosotros. Es decir, estamos inmersos en un mundo de señales, las técnicas y métodos para transformar a éstas y usarlas como información ha evolucionado de manera impresionante y cada vez más compleja.

La globalización, ha sido un factor importante para el desarrollo de dispositivos usados en la ciencia y la ingeniería, pues este gremio usa señales como imágenes remotas de pruebas espaciales, voltajes generados por el corazón y el cerebro, ecos de radares y sonares, vibraciones sísmicas, y otras. Pero el hacer uso de estas señales involucra el análisis y procesamiento de señales, que son discretas en el tiempo. Aunque comúnmente las señales en la naturaleza nos llegan en forma analógica, también existen casos en que estas son por su naturaleza digitales, por ejemplo, las edades de un grupo de personas, el estado de una válvula en el tiempo (abierta/cerrada), etc.

El procesamiento digital de señales es una ciencia que usa computadoras para procesar este tipo de datos y que incluye filtrado, reconocimiento de voz, mejoramiento de imágenes, compresión de datos, redes neuronales y otros temas. El DSP es una de las tecnologías más modernas del siglo y su uso se ha vuelto necesario, por esto, el presente trabajo trata de aleccionar sobre sus recursos, arquitectura y herramientas de manera que permitan el diseño, simulación y programación en tiempo real, e interactuar con otros sistemas de tal forma que los aspectos teóricos aplicados sirvan de soporte a diferentes proyectos.

Este tipo de dispositivo se usa por las ventajas que presenta frente a otros sistemas en el procesado digital de señales, dado que; puede adaptarse fácilmente al cambio de la variables, presenta muy baja distorsión en las señales de entrada y procesamiento de las mismas en tiempo real, tiene un alto índice de precisión, control sobre el comportamiento del hardware, en la tarjeta del DSK, se encuentran ya incluidos los convertidores sin necesidad de agregar hardware adicional para las conversiones analógicas de entrada, cuenta con un robusto código de instrucciones que satisfacen cualquier aplicación de procesamiento digital de señales de audio y voz, además son programables y reprogramables (no requiere el cambio del sistema de hardware), y permite un consumo de energía pequeño ya que emplea tecnología de estado sólido.

La mayor parte de este trabajo proviene de los manuales de referencia para la familia TMS320C6000 de procesadores, sin embargo se extrae de cada uno la esencia, ya que el modelo que usa el DSK necesita restructurar, resumir, y modificar la información.

Aún cuando la literatura para DSPs es extensa, no es sencillo escudriñar en ella, pues es una tecnología multidisciplinaria y acrecentada en no mas de 10 años, además de seguirse innovando. La prioridad del desarrollo de esta tesis, es ofrecer un punto de apoyo para el comienzo de esta materia.

1.4. Aportaciones

Las aportaciones que se obtuvieron con el desarrollo de este trabajo son las siguientes:

- Guía para usuarios que recién empiezan en el ambiente de procesadores digitales de señales (DSPs)
- Notas de consulta para usuarios que ya tienen conocimientos en el área.
- Formato base para publicar el libro titulado "Procesamiento digital de señales basado en el DSK TMS320C6711"sometido a Thompson Learning.

1.5. Organización de la tesis

Esta tesis esta organizada en seis capítulos y tres apéndices, de tal manera que:

En los primeros capítulos se expone el software de instalación del DSP, el avance evolutivo, la arquitectura, las instrucciones básicas de su repertorio para programar en código ensamblador y en código C. Se hace uso de las herramientas básicas para ejecutar los programas, y de los recursos elementales de programación, también se nombran programas auxiliares para medición e instrumentación, que tienen la propiedad de compartir recursos con el DSP y las herramientas de software que hacen flexible la manipulación de datos.

El enfoque principal de la segunda mitad es la introducción a la programación del DSK, parte medular de este trabajo, usando los requerimientos necesarios, que permitan tener una referencia de partida al manejo de los recursos de esta tarjeta, de tal forma que la lógica de estas aplicaciones respondan a la inquietud.

A través del Capítulo 3, se mencionan conceptos para la programación del DSP, algunas funciones son exclusivas de este modelo, otras son compatibles, sin embargo, podemos valorar, una vez concluida su lectura, las ventajas de su programación.

El siguiente capítulo, expone los protocolos de comunicación básicos, para la trasferencia de información entre los programas de aplicación y la tarjeta del DSK TMS32C6711.

Se hace uso de la herramienta RTDX, cuya importancia radica en permitir la ejecución en tiempo real.

En el capítulo 5, se presenta una aplicación de audio donde se muestra la capacidad de transferencia de datos; para la cual fue desarrollada esta tarjeta y donde se muestra la amplia gama de recursos que usa para facilitar su empleo en tareas propias de estos sistemas.

Para el último capítulo, se editan las conclusiones obtenidas al trabajar con este dispositivo, las ventajas, desventajas y los problemas que se confrontaron.

Los apéndices reúnen información troncal, como las instrucciones para código ensamblador, las fases de la segmentación encauzada (pipeline), los registro, y el conjunto de instrucciones API.

Capítulo 2

Plataforma del sistema

- Uso del DSK
- Instalación de software y hardware
- Uso de herramientas del depurador (debugger)
- Ejemplos de programación en código TMS320C6711 para probar herramientas.

Este capítulo introduce algunas herramientas disponibles para el procesamiento digital de señales. Estas herramientas incluyen al DSK TMS320C6711 (DSP Starter Kid) con soporte de entrada y salida. Se incluyen también, algunos ejemplos para ilustrar estos desarrollos y en particular para probar al DSK.

2.1. Introducción

El DSK es una tarjeta compuesta principalmente por circuitos integrados. La tarjeta esta diseñada para procesamiento de voz y audio; e imágenes. Sus principales componentes son:

- Un procesador de punto flotante de 32 bits (DSP TMS320C6711)
- ROM Flash de 128 K Bytes
- SDRAM de 16 M Bytes
- Un codec de audio de 8 K Hertz (TLC320AD535)
- Tres switches programables
- Tres LEDs (Diodos electroluminiscentes) programables
- Un conector para la interfaz con el puerto paralelo de la PC.

Los procesadores digitales de señales, como el TMS320C6711, son microprocesadores rápidos con un conjunto de instrucciones especializadas y una arquitectura propia para el procesamiento de señales. La arquitectura de un procesador digital de señales esta optimizada para cálculos numéricamente intensos. Estos procesadores se usan en un rango amplio de las comunicaciones y en el procesamiento de voz e imágenes. Los procesadores digitales de señales los podemos encontrar en sintetizadores de música, teléfonos celulares, fax/modems, etc. Éstos han sido el producto del cambio para numerosas aplicaciones, por su costo efectivo. Las técnicas del DSP han sido muy productivas ya que permiten el desarrollo de software y hardware a bajo costo. Por ejemplo, las aplicaciones en modems y en reconocimiento de voz son baratas usando las técnicas del DSP. Además, los procesadores digitales de señales pueden tener diferentes tareas, ya que pueden rápidamente reprogramarse para diferentes aplicaciones. Mientras que los sistemas analógicos con componentes electrónicos discretos son sensibles a los cambios de temperatura, los sistemas basados en DSP no son tan afectados por las condiciones ambientales. [31, 19]

Una de las aplicaciones más comunes en un DSK TMS320C6711, es en audiofrecuencia en el rango de 0 a 20 KHz. La voz puede muestrearse a 10 KHz, lo que implica que cada muestra o valor se obtiene en un rango de 1/(10 KHz) o 0.1 ms. Por ejemplo, la velocidad de muestreo de un disco compacto es de 44.1 KHz, comúnmente.

El sistema básico consiste en un convertidor analógico a digital (ADC) para capturar una señal de entrada. La representación digital resultante de la señal capturada se procesa por un procesador digital de señales como el TMS320C6711 y la salida continua hacia un convertidor digital a analógico (DAC). También se incluye con el sistema básico, un filtro especial de entrada contra el efecto alias que elimina señales erróneas, y un filtro en la salida para reconstruir la señal procesada. [31, 29]

2.2. Requerimientos de soporte del DSK TMS320C6711

Para realizar los experimentos, son necesarias las siguientes herramientas:

1. Un DSP Starter Kit (DSK) de Texas Instruments, que incluye una placa con el procesador TMS320C6711 de punto flotante (Figura 2.1) y soporte de entrada-salida (I/O). La placa del DSK contiene un circuito de interfaz analógica (AIC) provista de un convertidor programable ADC y DAC, filtrado de entradas y salidas, todo en un chip. Las herramientas de software para ensamblar y compilar así como también algunos ejemplos de aplicaciones se incluyen en el paquete del DSK.

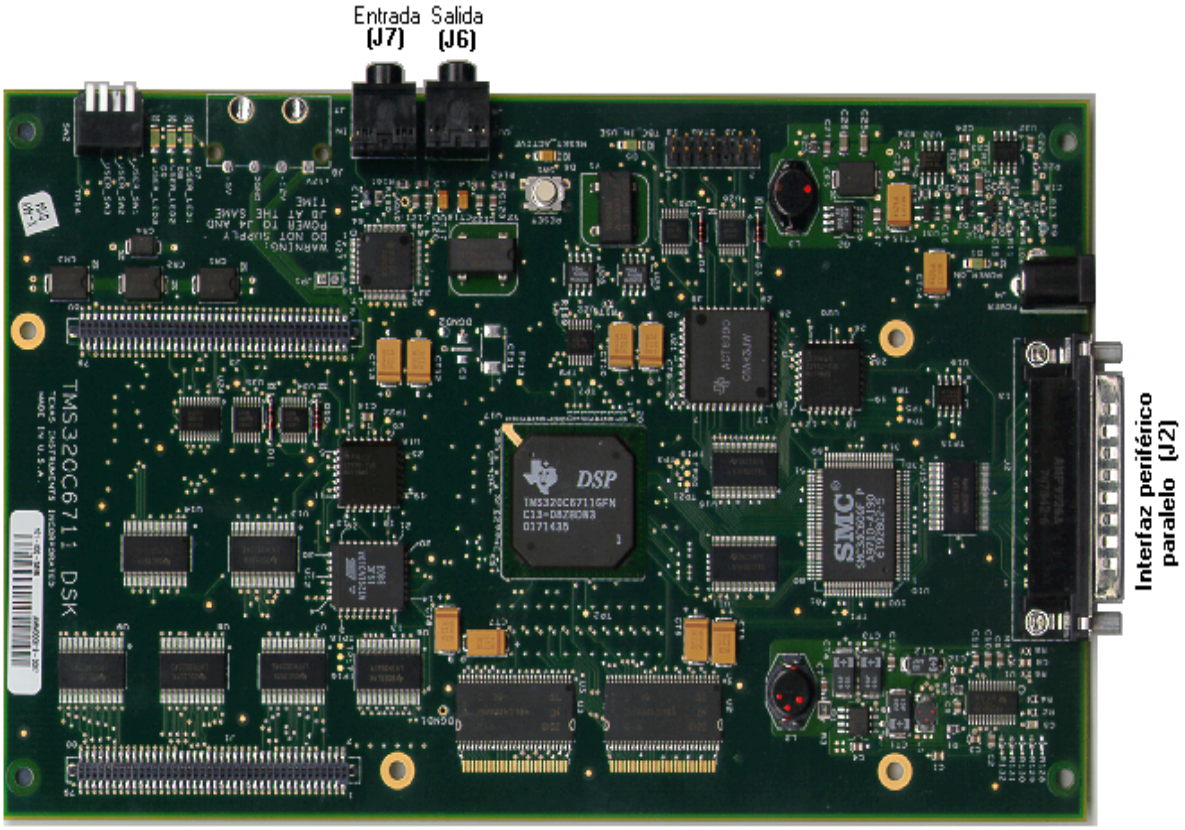


Figura 2.1: Tarjeta del DSK TSM320C6711.

2. Una PC, con los siguientes requerimientos [3]:

   - CPU Pentium compatible 233 MHz o más,
   - 600 MB de espacio libre en disco duro,
   - 64 MB de RAM,
   - Pantalla SVGA (800 × 600),
   - Internet Explorer (4.0 o posterior) o Netscape Navigator (4.0 o posterior),
   - Unidad de disco local para CD-ROM,
   - Sistema operativo Windows XP.

   La placa del DSK se conecta al puerto paralelo de la impresora, a través de un conector tipo DB25.

3. También se recomiendan otros instrumentos como son: osciloscopio, generador de señales, micrófono, analizador de espectro de señal y/o bocinas, depende de la aplicación.

4. El lenguaje ensamblador para el DSP TMS320C6711 de punto flotante soporta los programas en C. Esta herramienta incluye un compilador C, un ensamblador y un enlazador que crea un archivo ejecutable, este archivo se utiliza en el DSK.

El DSK basado en el TMS320C6711 es relativamente poderoso, desarrollado para procesamiento digital de señales en tiempo real. La placa del DSK contiene al procesador TMS320C6711 y al circuito de interfaz analógica TLC320AD535 (AIC) con canales independientes para voz y datos.

El ensamblador del DSK crea un archivo ejecutable que puede directamente descargarse del DSP C6711 en el DSK y ejecutarse. Esto no crea un archivo COFF (Common Output File Format), el que se obtiene usando el ensamblador DSP de punto flotante TMS320C6711. El ensamblador del DSK no incluye al enlazador. El código se ensambla en una dirección dentro de una sección específica de la memoria usando determinadas directivas del ensamblador. Estas directivas sirven como un enlazador y se usan para incluir o enlazar algunos archivos. El archivo ejecutable del ensamblador puede cargarse en el DSK C6711 usando el depurador [9].

2.3. Instalación de software y hardware

El equipo incluye una guía, un cable paralelo con conector tipo DB25 y un disco que contiene al ensamblador, al depurador, otras utilidades y ejemplos de aplicación. El DSK (placa) requiere de una fuente de 400 miliamperes. Los adaptadores con otras especificaciones no deben usarse. Cuando se conecta el DSK, los LEDs en la placa parpadean.

Para conectar el DSK a una computadora repita los siguientes pasos:

1. Apague la PC,
2. Conecte el cable del puerto paralelo a la tarjeta,
3. Conecte la otra terminal del cable al puerto paralelo de la PC. Si requiere instalar micrófono, bocinas y/u otra tarjeta, los debe conectar antes de encender el DSK,
4. Conecte el cable de alimentación a la tarjeta,
5. Conecte la otra terminal del cable de alimentación a la fuente de energía,
6. Los LEDs serán intermitentes para indicar que la tarjeta esta operando,
7. Encienda la computadora,
8. Instale el software.

El DSK C6711 debe conectarse al software integrado del dispositivo, la instalación del CCS (Code Composer Studio) debe configurar el puerto paralelo. Cuando el DSK no se encuentra conectado propiamente, suele ser por la selección del puerto paralelo. Por ejemplo, la dirección es 0x378 para el LPT1 (predeterminado). Si la dirección del puerto esta en uso, se selecciona otro puerto de comunicación (0x278 para LPT2 o 0x3BC para LPT3). Los conectores tipo RCA están disponibles en la placa del DSK para entrada y salida.

El procedimiento para instalación del software es:

1. Inserte el CD de instalación en la unidad de disco. Aparecerá una pantalla de instalación en pocos segundos (Figura 2.2); si no, ejecute setup.exe del CD-ROM,

Figura 2.2: Pantalla de instalación del CCS.

2. Escoja la opción para instalar CCS,

3. Responda a los cuadros de diálogo para que corra el programa de instalación,

4. Aparecerá un panel de bienvenida que describe los iconos de los programas que aparecerán en el escritorio de la computadora (Figura 2.3). El panel contiene vínculos para demostraciones y para el sitio TI DSPDevelopers.

Figura 2.3: Pantalla de bienvenida del CCS.

El CCS v2, automáticamente configura el sistema a menos que se detecte una versión previa.

Antes de correr el CCS, revise la conectividad del software y hardware, apoyado en los siguientes acciones:

- Ejecute **Reset**: del menu Inicio, elija **Programas** → **Texas Instruments** → **Code Composer Studio** → **Hardware Resets** → **Reset C6x11 DSK**.
- Para comenzar con el CCS, haga doble-clic sobre el icono CCS-DSK 2 ('C6000) del escritorio de la computadora.
- Aparecerá la pantalla principal de CCS (Figura 2.4).

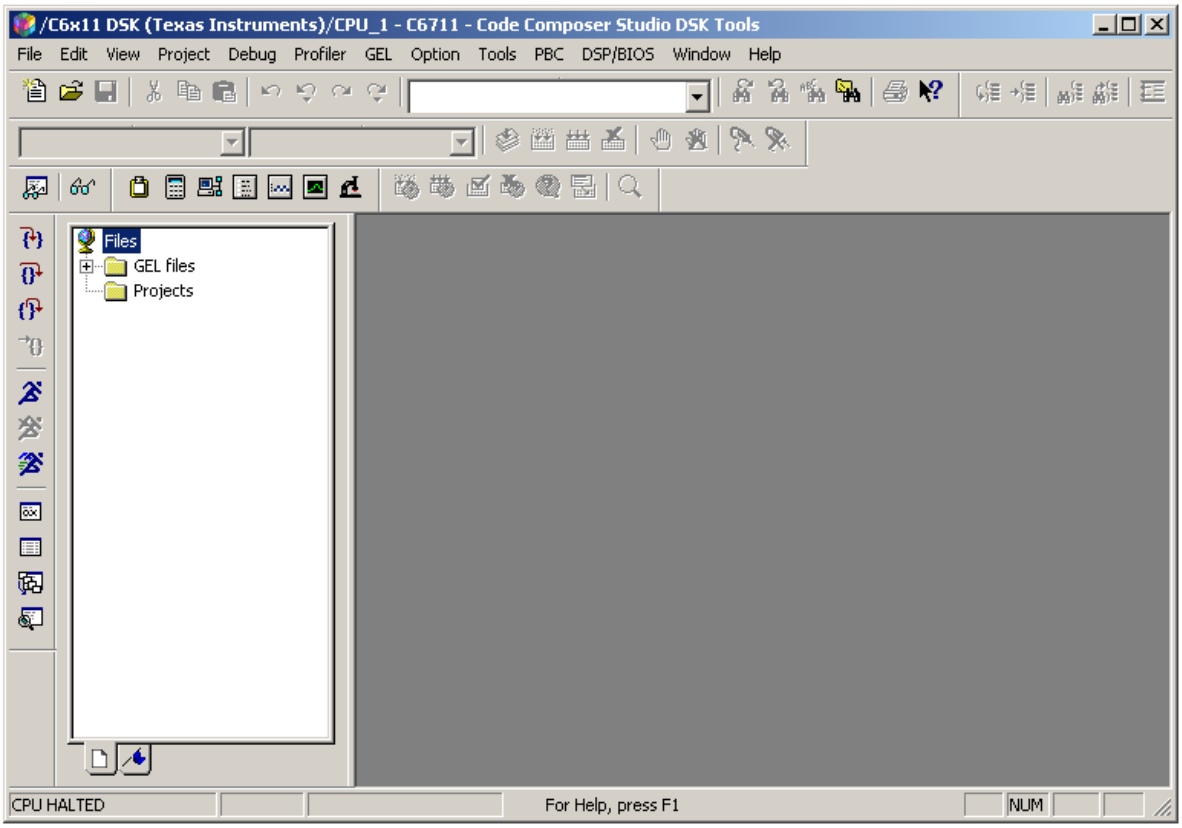


Figura 2.4: Pantalla de presentación del CCS.

- Luego, ejecute una revisión rápida, escoja **Gel → Check DSK → Quick Test**.

Estas pruebas confirmarán la correcta instalación del software del DSK, si detecta algún problema, quite los programas recientes y vuelva a instalar. Ahora puede trabajar en el CCS, y revisar el tutorial de ayuda.

2.4. Herramientas básicas

Se muestran algunos ejemplos para ilustrar las herramientas del DSK. Estos programas son sólo para probar las herramientas, en particular, al DSK. Los programas codificados en C o en lenguaje ensamblador se ensamblan usando el software del CCS v2.

Para crear un **Nuevo proyecto** realice los siguientes pasos:

1. Seleccione del menu **Project→ New**. Esto abre el cuadro de diálogo **Project Creation** (Figura 2.5). En el campo **Project Name** escriba `MATRICES` (o cualquier otro nombre).

2. En el campo **Location**, escriba el directorio donde desee que resida el proyecto o puede buscarlo. El directorio común donde se almacenan los programas es `c:\ti\myprojects`.

3. En el campo **Project Type**, elija Ejecutable (.out) o Librería (.lib), dependiendo del tipo de salida que desee crear.

4. En el campo **Target**, seleccione el código que tenga configurado para el CCS y haga clic en **Finalizar**.

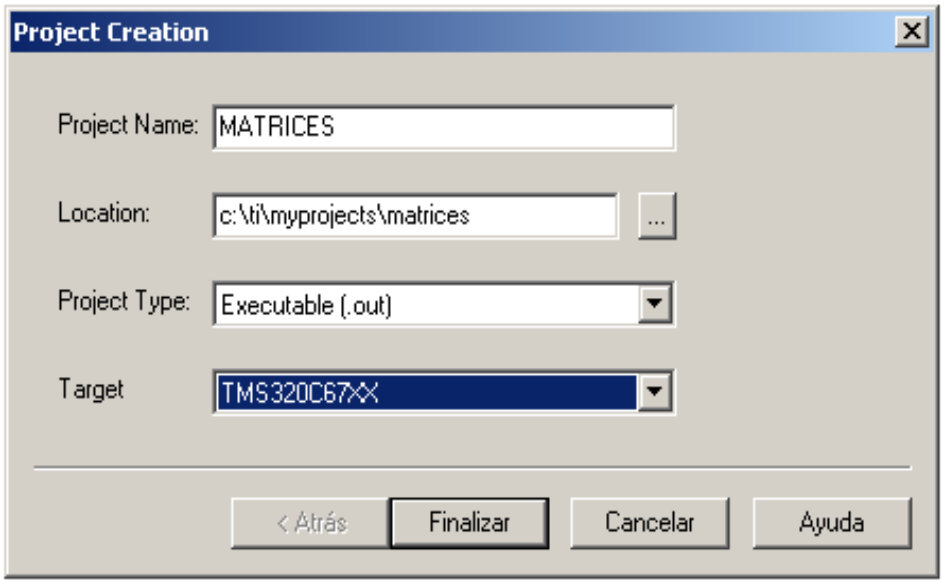


Figura 2.5: Ventana para creación de un proyecto.

Code Composer Studio ha creado un archivo llamado `MATRICES.pjt`. Este archivo almacena los programas necesarios para correr la aplicación.

Ahora, para crear un nuevo programa en código ensamblador del TMS320C6711, efectúe los siguientes pasos:

1. Seleccione **File → New → Source File**.

2. Escriba el Programa 2.1. Esto le hará familiarizarse con el lenguaje ensamblador.

Ejemplo 2.1. ***Multiplicación de una matriz por un vector usando código ensamblador TMS320C6711.***

Este ejemplo ilustra el uso de algunas de las herramientas. No se preocupe del código del programa. El Programa 2.1 muestra el listado para `MATRIX.ASM` *que multiplica una matriz A[3×3] por un vector B[3×1], y el resultado de la operación es el vector de elementos* $[14, 32, 50]^T$.

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} = \begin{bmatrix} 14 \\ 32 \\ 50 \end{bmatrix} \tag{2.1}$$

Programa 2.1: MATRIX.asm

```
;MATRIX.ASM - MULTIPLICACIÓN DE UNA MATRIZ POR UN VECTOR.
                        .data
dir_A                   .int        1,2,3,4,5,6,7,8,9
dir_B                   .int        1,2,3
dir_S                   .int        0,0,0
dir_flo                 .float      45
dir_dob                 .double     -4.6
x                       .set        3
                        .text
                        MVKL        dir_A,A2
                        MVKH        dir_A,A2
                        MVKL        dir_B,B2
                        MVKH        dir_B,B2
                        MVKL        dir_S,A10
                        MVKH        dir_S,A10
                        ZERO        A0
          ||            ZERO        A7
          ||            ZERO        B7
          ||            ZERO        A11
                        MVK         x,A6
                        MVK         9,A8
LOOPI
LOOPJ                   LDW         *+A2[A7],A3
                        LDW         *+B2[B7],B3
                        NOP         5
                        MPYI        A3,B3,A5
```

```
                    NOP     8
                    ADD     A0,A5,A0
||                  ADD     1,A7,A7
||                  ADD     1,B7,B7
                    SUB     A6,1,A6
                    MV      A6,A1
          [A1]      B       LOOPJ
                    NOP     5
                    MV      A0,A12
                    ZERO    A0
                    ZERO    B7
                    MVK     3,A6
                    SUB     A8,3,A8
                    MV      A8,A1
          [A1]      B       LOOPI
                    STW     A12,*A10[A11]
                    ADD     1,A11,A11
                    NOP     2
                    LDW     *+A10[0],A3
                    NOP     4
                    LDW     *+A10[1],A4
                    NOP     4
                    LDW     *+A10[2],A5
                    NOP     4
                    NOP     4
```

Una vez finalizado el proceso de captura, se debe incluir este programa en el proyecto anterior `MATRICES.pjt` y se procede de la siguiente manera:

1. Escoja **File** → **Save as**, en esta ventana introduzca el nombre que desee para su programa, en este caso escriba `MATRIX`, en el cuadro de diálogo **Nombre**. También debe elegir la extensión en el menú desplegable, para este programa seleccione **Assembly Source Files** (`*.asm`). Por último haga clic en **Save**.

2. Seleccione **Project** → **Add Files to Project**, en la ventanilla que aparece, busque el nombre del archivo `.pjt`. Elija del menú **Type** la extensión `asm`. Elija el programa y haga clic sobre el botón **Abrir**.

3. En la ventana Proyecto, ya se incluyó el programa.

Es necesario incluir en el proyecto, un programa para administrar memoria, este programa tiene la extensión `.cmd`.

Para cargar este programa debe realizar lo que se indica a continuación:

1. En el menú **Project** seleccione **Add Files to Project**, en el cuadro de diálogo **Buscar en:** encuentre la ruta:

   ```
   C:\ti\c6000\cgtools\lib\
   ```

2. En el menú **Type**: seleccione **Linker Command File** (*.cdm)

3. Haga doble clic sobre lnk.cmd

El proyecto requiere pasar a través de fases para poder ejecutarse, hasta aquí solo se han incluido en el proyecto un conjunto de programas que lo constituyen pero que aún son independientes, así que se deben tomar en cuenta las herramientas necesarias para que el proceso de enlace se lleve a cabo.

Herramientas de desarrollo para generación de código

La plataforma del TMS320C6711 esta soportada por un conjunto de herramientas, para el desarrollo de software que incluyen compilador C/C++ optimizado, ensamblador optimizado, enlazador y otras utilerías asociadas a ellos. Además, el TMS320C6711 soporta las siguientes herramientas para el desarrollo en lenguaje ensamblador [9, 24]:

- Ensamblador (Assembler).

- Archivador (Archiver).
- Enlazador (Linker).
- Listado Absoluto (Absolute lister).
- Listado de referencias cruzadas (Cross-reference lister).
- Utilerías de conversión hexadecimal (Hex coversion utility).

El optimizador del ensamblador, permite escribir código en lenguaje ensamblador lineal, sin importar la estructura de segmentación encauzada (pipeline) o la asignación de registros. Asigna registros y usa optimización de ciclos, para convertir el código de lenguaje ensamblador lineal a lenguaje ensamblador paralelo.

El compilador C/C++, acepta código fuente en lenguaje C/C++ y produce código fuente en lenguaje ensamblador para el TMS320C6711. Para llamar al *shell* del compilador la instrucción tiene la siguiente forma:

```
cl6x [opciones][nombredearchivos][-z[opciones de enlace][archivos
                                objeto]]
```

El ensamblador traduce los archivos de lenguaje ensamblador fuente en lenguaje máquina. Invoque el ensamblador de la siguiente manera:

```
asm6x [archivo entrada[archivo objeto[lista de archivos]]][opciones]
```

El enlazador combina los archivos objeto a un solo archivo ejecutable para el DSP. También acepta archivos de librerías y módulos de salidas, creados para la ejecución previa del mismo. La sintaxis general para llamar al enlazador es:

```
lnk6x [opciones]nombredearchivo1...nombredearchivo
```

El archivador une un grupo de archivos dentro de un solo archivo, llamado librería. Por ejemplo puede juntar varias macros, dentro de una librería de macros. Se invoca de la siguiente manera:

```
ar6x [-]comando[opción]nombredelib[nombrede archivo1..nombrede archivo]
```

También se puede usar la utilería de construcción de librerías para construir una librería de soporte en tiempo real.

El listado absoluto es una herramienta para depuración de programas. Acepta como entrada, archivos objeto enlazados y crea archivos con extensión .abs que son ensamblados para producir una lista que muestra las direcciones absolutas de código objeto. La sintaxis es:

```
abs6x [-opción]archivo de entrada
```

La utilería de conversión hexadecimal, convierte un archivo objeto de tipo COFF en un archivo cuyo formato objeto se puede seleccionar entre los siguientes: TI-Tagged, ASCII-Hex, Intel, Motorola-S, o Tektronix. El archivo convertido, puede ser transmitido a una memoria EPROM. Para llamar a esta utilería se escribe:

```
hex6x [opción]nombredelarchivo
```

El listado de referencias cruzadas usa archivos objeto para producir un listado de referencias cruzadas, mostrando símbolos, definiciones y sus referencias en el archivo fuente enlazado. Llame al listado de referencias cruzadas de la forma:

```
xref6x [opción][nombredelarchivo entrada[nombredelarchivo salida]]
```

Compilación, Ensamble y Enlace

Después de la fase de edición del programa de aplicación, éste se debe compilar, ensamblar y enlazar. Para esto se utilizan las herramientas descritas anteriormente, conjuntamente con las instrucciones o desde CCS en una elección.

Para compilar, haga uso del icono **Compilar Archivo** (Compiler File)  . Después de esto puede elegir entre dos opciones para ensamblar y enlazar.

Por ejemplo, si a un archivo en código ensamblador del TMS320C6711, tal como `MATRIX.asm`, lo quiere ensamblar usando las herramientas de generación de código, entonces, desde la trayectoria `C:\ti\myprojects\MATRICES` del *DOS* ejecute el siguiente comando:

```
asm6x MATRIX.asm -mv6700
```

Durante este proceso se crea el archivo `matrix.obj`, este se combina para formar un módulo objeto que puede localizarse en el sistema de memoria del dispositivo y es ejecutado por él.

El siguiente paso, es enlazar el programa. En la trayectoria anterior, ejecute el comando:

```
lnk6x matrix.obj -o matrix.out
```

El archivo compilado se compara con el archivo fuente, si el tiempo marcado es mayor que el correspondiente al archivo objeto, se vuelve a compilar. Para determinar si el archivo de salida debe ser re-enlazado, el tiempo marcado de cada archivo objeto se compara con ese archivo de salida. El archivo de salida se vuelve a re-enlazar sí el tiempo marcado del archivo objeto es mayor. Para reducir el tiempo, CCS mantiene un archivo de información (.paf) para cada archivo de proyecto. La información de este archivo se usa para determinar cuales archivos necesitan reconstruirse o sí los programas necesitan re-enlazarse durante una construcción incremental. La información de los archivos se actualiza después de cada construcción/reconstrucción.

En este punto se ha creado el archivo `.out` que puede ser cargado a al tarjeta del C6711. Si tiene algún problema en el reconocimiento de los comandos incluya la trayectoria `C:\ti\c6000\cgtools\bin` en **Variables del sistema**, de las siguiente manera:

1. Efectúe un clic derecho sobre el ícono de **Mi PC**, elija **Propiedades**.

2. En la ventana **Propiedades del sistema**, haga clic sobre el borde **Opciones avanzadas** y pulse el botón **Variables de entorno**.

3. Seleccione en la parte de **Variables del sistema**, la trayectoria (path) y pulse el botón **Modificar**.

4. Luego se abrirá la ventana **Modificar la variable del sistema**, en el campo **Valor de variable**, escriba la trayectoria:

   ```
   C:\ti\c6000\cgtools\bin
   ```

5. Por último pulse **Aceptar**.

La segunda opción para generar un archivo .out, es decir compilar, ensamblar y enlazar, es a través de CCS, pulsando el icono **Rebuild All**  ., de la barra de herramientas del proyecto, o del menu **Project** → **Rebuild All**.

El archivo ejecutable .out, es un enlace común al formato del archivo objeto (COFF), popular en sistemas Unix y adoptado por algunos creadores de procesadores digitales de señales. El formato COFF hace más fácil la programación modular y la administración de segmentos de código.

Carga de un archivo en el DSK para ejecutarlo

Ya que se concluyeron los pasos anteriores, el archivo .out esta listo para ejecutarse en el C6711. Para llevar a cabo este proceso, se utiliza el cargador (loader), que es un dispositivo que coloca un módulo ejecutable. Las subsecuentes instrucciones permiten realizarlo:

Haga clic sobre el menu **File** → **Load Program**. En la ventana **Load Program** haga clic en la carpeta Depurador (Debug), pulse el botón **Abrir**. Haga doble clic sobre el nombre del programa con extensión `.out`. Asegurese de que el cuadro **Tipo** tenga la extensión `.out`.

Cuando se carga un programa en la tarjeta, el depurador automáticamente abre la ventana **Disassembly**. Esta ventana despliega las instrucciones desensambladas e información simbólica necesaria para la depuración. El desensamble retorna el proceso de ensamble y permite observar el contenido como en código de lenguaje ensamblador. La información simbólica consta de símbolos y cadenas de caracteres alfanuméricos que representan direcciones o valores para la tarjeta. Para cada instrucción de lenguaje ensamblador, la ventana despliega una instrucción de desensamble, la dirección en la que se localiza la instrucción, y el código de operación correspondiente (código máquina que representa la instrucción). Para producir el listado de desensamble, el depurador lee el código de operación de la tarjeta, y añade la información que indica la dirección en la que comienza el contador del programa (PC). Identifique la flecha verde sobre el margen que reconoce la localización actual del registro PC (Figura 2.6).

Vea que el código del programa comienza en la seccion `.text`, como se muestra en la ventana de **Disassembly**. La primer columna de notación hexadecimal representa la sección del contador del programa (PC), la segunda columna representa el código de operación de una instrucción y la tercera contiene las sentencias del programa fuente original.

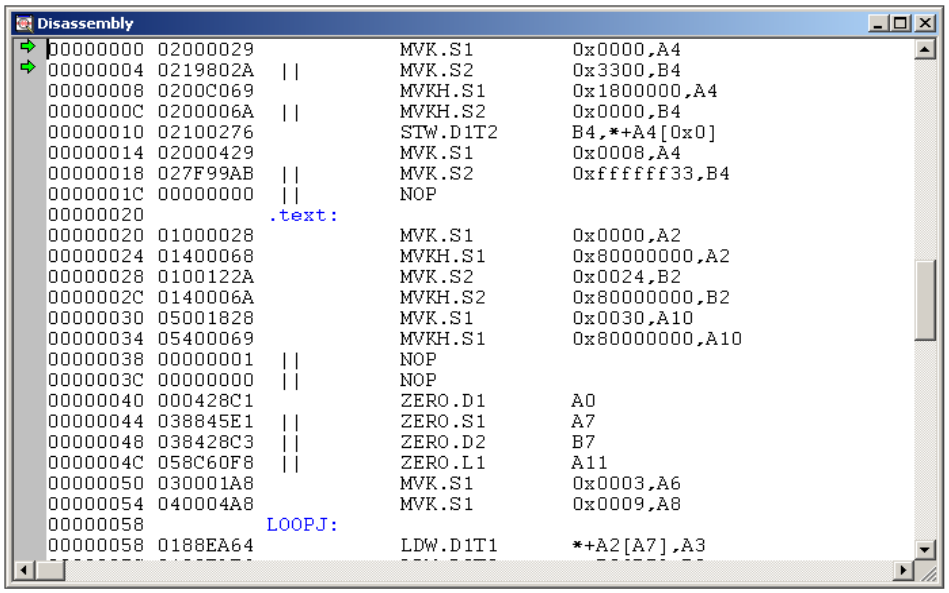


Figura 2.6: Ventana de desensamble.

Ejecución del programa

Ahora puede observar los resultados que genera la aplicación, pues el programa esta preparado para efectuar la tarea para el cual fue desarrollado. Continue con los siguientes pasos:

1. Presione F8 para ejecutar **Step Into** (Paso a paso) las primeras líneas del código del programa. Las instrucciones que se disponen para el C6711 se listan en el Apéndice A. Con este procedimiento el registro PC avanza a la instrucción apropiada. Una vez que haya alcanzado la sección `.text` la ventana actual cambia a la ventana de edición original, note la flecha amarilla, si desea observar varias ventanas en cascada, use el menu **Window** → **Cascade**.

   Abra la ventana de registros para observar los valores que toman durante la ejecución de este programa, esta ventana se activa eligiendo **View** → **CPU Registers** → **Core Registers**.

   En la ventana se muestra que `A2` y `B2` adquieren el valor de la dirección de memoria para el almacenamiento de los datos de la matriz A y del vector B, respectivamente; y `A10` contiene la dirección de memoria para los tres valores del resultado de salida. La matriz A multiplicada por el vector B da como resultado los valores E, 20, 32 en hex, equivalente a los valores 14, 32, 50 en decimal(Figura 2.7).

`A0-A15` y `B0-B15` son treinta y dos registros de propósito general de 32 bits, son frecuentemente usados para designar una dirección específica en la memoria que contiene una instrucción o un dato.

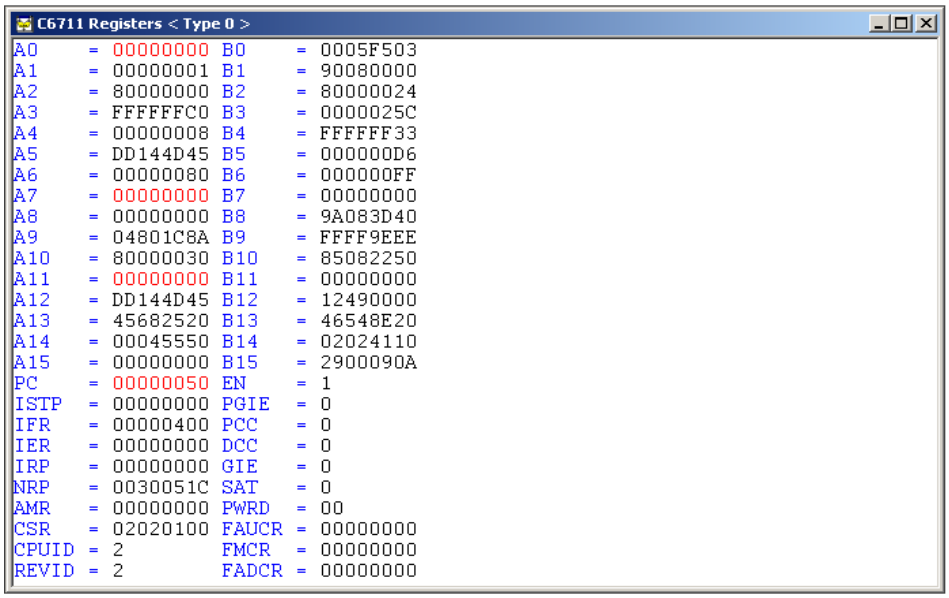


Figura 2.7: Ventana de registros.

2. La ventana **Memory** permite editar el contenido de la memoria en una dirección específica y las opciones permiten elegir el formato.

   Para ver los contenidos de la memoria seleccione **View** → **Memory**, o haga clic en el botón **View Memory** en la barra de herramientas del depurador.

   Antes de que abra la ventana Memoria, en las opciones especifique la localización que desee ver. Haga clic en **OK**.

   Para modificar cualquier característica de la ventana Memoria activa, oprima dos veces el botón derecho del mouse y seleccione **Propiedades**.

   Escriba la dirección de `A2` o de `B2` para desplegar el contenido de la memoria en formato hexadecimal de 32 bits. Observe los nueve valores de la matriz A, que están almacenados en la memoria, los tres valores del vector B que se almacenan en la dirección posterior a estos valores y los espacios reservados para el vector de salida (Figura 2.8).

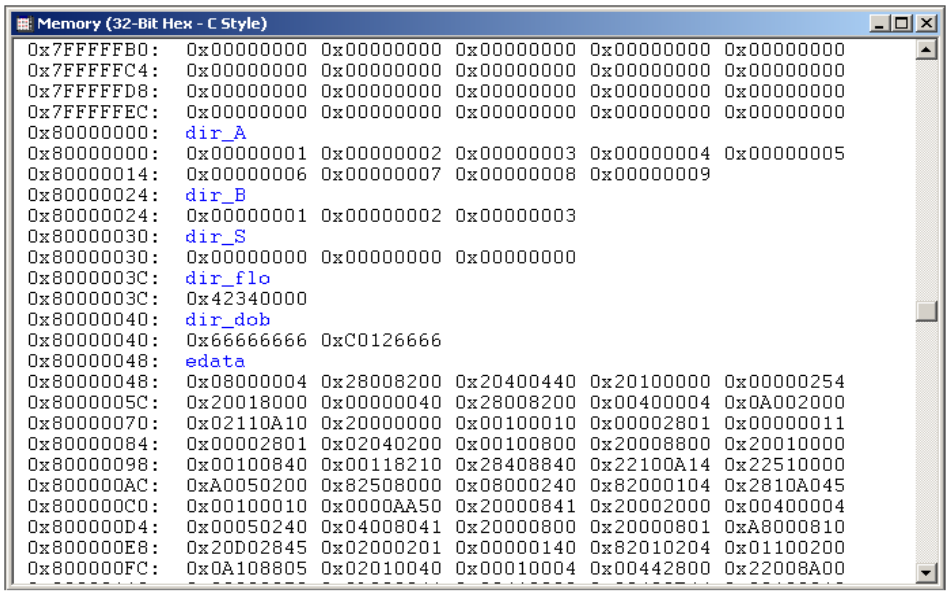


Figura 2.8: Ventana de direcciones de memoria.

3. Con **Paso a paso** recorra el programa y ejecute las instrucciones, observe los valores de los registros (parte inferior de la ventana), también despliega el contenido de los registros del CPU en formato hexadecimal. Al final del programa, verifique cada valor resultante E, 20, 32 (14, 32, 50), en la ventana Memoria, cambie el formato de representación. Los tres valores del resultado están en las localizaciones de memoria continuas, la notación 0x para hexadecimal es necesaria.

   Presione el botón derecho del mouse, sobre la ventana de los registros (parte inferior), puede cambiar el formato a Vista Larga (View Long), Vista Flotante (View Float), Vista Doble (View Double), dependiendo de el formato que requiera el registro.

4. Para correr el programa otra vez elija **File** → **Reload Program** .

   Algunas veces es necesario verificar los valores de una variable, durante la ejecución de un programa.

   Coloque el cursor en la última línea del programa original. Haga clic en el botón **Toggle Breakpoint**, o presione F9. El punto rojo, sobre el margen izquierdo, indica que este es el punto de interrupción o de paro. Ejecute un clic sobre el icono **Run** .

Ahora escoja **View** → **Watch Window**. Aparecerá una ventana en el área derecha baja de la ventana del CCS. Esta ventana puede mostrar los valores de las variables. Elija **Watch 1**, seleccione los registros `A3`, `A4`, `A5`, y peguelos en dicha área; al final de la corrida observará el valor de cada variable, el número de bits para su representación y su tipo (Figura 2.9).

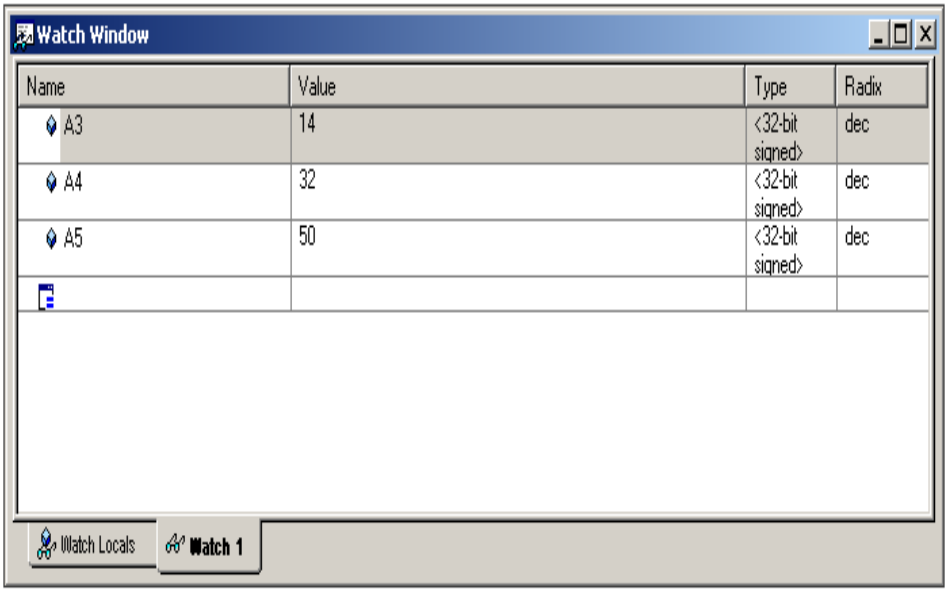


Figura 2.9: Ventana de variables.

En el Capítulo 3 veremos que la instrucción `STW A12,*A10[A11]` almacena cada resultado de `A12` en una localidad de memoria especificada por `A10`. El registro `A11` se incrementa para cada valor de salida y contiene la dirección en la memoria donde cada resultado es almacenado. De este modo, `A10` se usa como un puntero, que señala la dirección de la memoria.

2.5. Herramientas adicionales

Las siguientes herramientas pueden ser útiles junto con el DSK

1. *GOLDWAVE*. Es un instrumento virtual para las formas de onda (goldwave.zip), puede usarse como un osciloscopio o un analizador de espectro junto con la PC y una tarjeta de sonido como la Sound Blaster. Goldwave también se usa para generar funciones como una señal senoidal con una frecuencia específica o un ruido aleatorio. Este puede extraerse de la dirección www.goldwave.com.

2. *Virtual Bench* es un instrumento virtual disponible por la National Instruments (que produce LabView) en www.natinst.com. Con una tarjeta de adquisición de datos que configura la PC en una ranura y una placa I/O para entrada y salida, el Virtual Bench puede usarse como un generador de funciones, como un osciloscopio o como un analizador de espectro.

3. *SigLab*. Este es un laboratorio virtual con soporte de software, disponible en DSPTecnology en siglab@dspt.com. El SigLab tiene una interfaz hacia la PC vía un conector SCSI. Dispone de dos canales, 20-KHz de banda ancha y cuatro canales con un ancho de banda de 50 KHz. El SigLab incluye un TMS320C6711 para el procesamiento de señales en tiempo real y dos procesadores digitales de señales para dispositivos analógicos para soporte de filtrado. El SigLab, se conecta a la PC por medio de una interfase SCSI, puede tener acceso para generación de entrada en tiempo real y salida monitoreada mientras que el DSK esta corriendo. Por ejemplo, puede usarse como un osciloscopio o como un analizador de espectro a través de un canal del SigLab conectado a la salida del DSK, mientras se generan las señales como un tono senoidal o un ruido aleatorio por medio de otro canal del SigLab conectado a la entrada del DSK.

4. *RIDE40*. Exportable de Hyperception en info@hyperception.com es una herramienta de diseño virtual que se usa para completar algoritmos del DSP. Contiene un rango amplio de bloques funcionales para FFT, correlación, filtrado, etc., y se usa para simulación y tiempo real. En pocos minutos, podemos diseñar y probar un sistema DSP que incluye bloques funcionales como son generadores de funciones seno, filtros, y la FFT. Los resultados se presentan en el monitor de la PC o en un dispositivo como un osciloscopio. Los filtros digitales pueden fácilmente diseñarse con el paquete de filtros disponible en Hyperception.

5. *Actualización DSK y herramientas C67x*. El sitio web de Texas Instruments contiene la versión mas reciente de las herramientas del software para el DSK C6711. Estas herramientas incluyen el ensamblador y el depurador así como algunos soportes y ejemplos de aplicaciones. El sitio de Texas Instruments es *dspvillage.TI.com*. Selecciona herramientas C7xdsk para actualizar el software para el DSK C6711.

Capítulo 3

Arquitectura e instrucciones

- Arquitectura y conjunto de instrucciones del procesador TMS320C6711
- Modos de direccionamiento de memoria
- Directivas del ensamblador
- Ejemplos en código C y en código ensamblador

Algunos de los programas que se incluyen en este capítulo muestran la arquitectura, las directivas de ensamblado, y el conjunto de instrucciones del procesador TMS320C6711 con las herramientas asociadas.

3.1. Introducción

Texas Instruments produjo la primera generación de procesadores digitales de señales en 1982 con el TMS32010, la segunda generación en 1985 con el TMS32020, seguido de la versión C-MOS del TMS320C25 en 1986, y el TMS320C50 en 1991. La primera generación de procesadores tenía 144×16 bits de memoria interna (RAM), con un tiempo de ciclo de instrucción de 200 ns. Más de una instrucción podía ejecutarse en un ciclo. Los miembros de la primera generación estaban disponibles comúnmente en versiones C-MOS, con alta velocidad de ejecución [31].

De la segunda generación, el TMS320C25, con 544×16 bits de memoria RAM, compatible con la familia del procesador TMS320C10 (C1x) y tiempo de ciclo de instrucción de 100 ns, era capaz de ejecutar 10 millones de instrucciones por segundo. Otros miembros de la segunda generación, los procesadores de la familia C2x disponían de alta velocidad de ejecución. Algunas versiones de los procesadores de esta generación (C1x, C2x, y C5x) tenían diferentes características, como la velocidad de ejecución y la ROM.

Los procesadores C1x, C2x, y C5x eran de punto fijo basados en una modificación de la arquitectura Harvard con espacios de memoria separados para datos e instrucciones que permitían el acceso simultaneo.

El TMS320C32 es un procesador, de la tercera generación de punto flotante, pero con un cuarto de memoria del C31 (aunque tiene características especiales para acceso a la memoria externa). El TMS320C40 es un procesador de la cuarta generación de punto flotante, tiene código compatible con el procesador C3x. También tiene la misma cantidad de memoria que el C3x, y seis puertos seriales (la versión C44 tiene cuatro puertos seriales). Un procesador C40 puede conectarse directamente a uno de los seis puertos de otro procesador C40 sin espera lógica, haciendo al C40 adaptable para procesamiento paralelo.

El TMS320C62 (C62) es el procesador de punto fijo, expuesto en 1997, a diferencia de los procesadores de punto fijo previos, éste se basó en una arquitectura VLIW (very long instruction word) y no es compatible con los procesadores de punto fijo de generaciones anteriores. El procesador de punto fijo TMS320C80 fue anterior al C62 y contiene cuatro procesadores de punto fijo y un procesador con un conjunto de instrucciones reducidas (RISC). El C62 fue creado para aplicaciones de alto nivel como video y multimedia. El TMS320C67, de código compartido con el C62, también salió a la venta en 1997; este es otro miembro de la familia C6x basado en arquitectura VelociTI desarrollada por Texas Instruments, que consta de unidades de ejecución múltiple paralela que realizan varias instrucciones durante un ciclo de reloj.

El TMS32C6711, es un procesador digital de señales de alto rendimiento, miembro de la generación TMS320C6x, procesadores de punto flotante de la familia C6000. Con una ejecución superior a 2000 millones de instrucciones por segundo (MIPS) a 250 MHz y un amplio conjunto de herramientas. La plataforma TMS320C6000 es ideal para novedosas aplicaciones; por ejemplo: seguridad personalizada con reconocimiento facial y de huellas digitales; control marítimo con sistemas de posicionamiento global (GPS) y diagnósticos médicos remotos, entre otras aplicaciones [19].

Un procesador de punto fijo es mejor para dispositivos como los teléfonos celulares que usan baterías, ya que estos usan menos energía que un procesador equivalente de punto flotante. Los procesadores de punto fijo C1x, C2x y C5x tienen un rango dinámico limitado de precisión mientras que los procesadores de punto flotante C3x, C4x, C6x tiene mayor rango dinámico. En un procesador de punto fijo, es necesario medir el dato para reducir el sobreflujo, y esto debe hacerse con detenimiento. El sobreflujo ocurre cuando una operación como en la suma de dos números produce un resultado con mas bits que los que pueden disponer un registro del procesador.

El TMS320C6711 ejecuta arriba de 8 instrucciones de 32 bits cada ciclo. El CPU principal tiene 32 registros de propósito general de una longitud de palabra de 32 bits y ocho unidades funcionales: dos multiplicadores y seis ALUs. Con un tiempo de ciclo de instrucción de 6.7 ns, esto le da la capacidad de realizar:

- Un máximo de 1336 MIPS (millones de instrucciones por segundo), a 167 MHz.
- Un máximo de 1 G FLOPS (operaciones de punto flotante por segundo), a 167 MHz, para operaciones de precisión simple
- Un máximo de 250 M FLOPS a 167 MHz, para operaciones de doble precisión.
- Un máximo de 688 M FLOPS a 167 MHz, para operaciones de multiplicación y acumulación.

El tiempo de ciclo de instrucción o MIPS no provee la dimensión total de ejecución, es decir que se necesita considerar el uso eficiente de la memoria y el tipo adecuado de instrucción. El TMS320C6711 es un procesador de 32 bits capaz de ejecutar operaciones de punto flotante, enteras y operaciones lógicas. Tiene un bus de dirección de 32 bits, capacitándolo para direccionar 2^{32} o 4096 megabytes al espacio de memoria para un programa, un dato, o una entrada/salida. Con tales ventajas y en especial con los modos de direccionamiento, el C6711 es apropiado para aplicaciones en comunicaciones, control de instrumentación y procesamiento de voz e imágenes.

La estructura fundamental y las características de las diferentes generaciones de DSPs son muy parecidas, ya que todos disponen de bloques esenciales: memoria de datos, módulos controladores de periféricos, sin embargo cada arquitectura hace uso de recursos cada vez más propios para el usuario.

3.2. Arquitectura del TMS320C6711

En la arquitectura de Von Neumann, las instrucciones del programa y los datos están almacenados en un simple espacio de memoria. Un procesador con una arquitectura Von Neumann puede realizar una lectura o escribir a la memoria durante cada ciclo de instrucción. Las típicas aplicaciones del DSP requieren de varios accesos a la memoria en un ciclo de instrucción [23].

La arquitectura Harvard dispone de dos memorias independientes: una, que contiene sólo instrucciones y otra, sólo datos. Ambas disponen de sus respectivos sistemas de buses de acceso y es posible realizar operaciones de lectura o escritura simultáneamente en ambas memorias.

El TMS320C6711 esta basado en una modificación de la arquitectura Harvard, con espacios independientes de memoria, que permiten dos accesos de memoria en un ciclo de instrucción. Dos espacios independientes de memoria pueden ser accesados usando dos buses independientes. Un espacio de memoria puede tomar cada instrucción del programa (o programa y datos), mientras el otro espacio de memoria puede tomar únicamente el dato. Con buses separados para un programa, un dato, y un acceso directo mejorado a la memoria (EDMA, Enhanced Direct Memory Access), el TMS320C6711 puede ejecutar envíos de programas, leer y escribir un dato, y operaciones EDMA. Ya que los datos y las instrucciones radican en espacios de memoria separados, los accesos a la memoria son posibles. La arquitectura del C6711 admite cuatro niveles de acoplamiento, mientras una instrucción es ejecutada, tres instrucciones subsecuentes son leídas, decodificadas y enviadas [19].

La arquitectura VelociTI hace que el TMS320C6711, sea de los primeros que usan una mejora de la tradicional VLIW para alcanzar una alta ejecución a través del paralelismo. La arquitectura VLIW consta de múltiples unidades de ejecución trabajando en paralelo para realizar múltiples instrucciones durante un ciclo de reloj. El paralelismo es la clave de la extrema rapidez de ejecución. La arquitectura VelociTI es altamente determinista, con pocas restricciones en cómo y cuándo las instrucciones deben ser buscada, ejecutadas, o almacenadas. Esta flexibilidad en la arquitectura es un avance en la eficiencia de los niveles del compilador del C6711.

El procesador C6711 esta constituido por tres partes fundamentales el CPU, los periféricos y la memoria. Ocho unidades funcionales operan en paralelo, con dos conjuntos similares de cuatro unidades básicas. Las unidades se comunican usando una trayectoria de cruce entre los dos archivos de registros, los que tienen 16 registros de 32 bits. El programa paralelo se define en la etapa de compilación si no hay dependencia de datos, la revisión la hace el hardware durante el tiempo de ejecución. La memoria de programa realiza ocho instrucciones de 32 bits cada ciclo.

La Figura 3.1 es un diagrama de bloques del dispositivo TMS320C6711, donde se observa que tiene una memoria para el programa y otra para los datos, que puede configurarse como caché. Los periféricos incluyen un controlador de la memoria de acceso directo mejorado (EDMA), interrupciones lógicas, interfaz de memoria externa (EMIF), puertos seriales, y temporizadores (timers).

Los periféricos del procesador C6711, incluyen: la EMIF, que proporciona el tiempo necesario para accesar a la memoria externa. El EDMA, permite que los datos se muevan de un lugar en la memoria a otro, sin la interferencia del CPU. El McBSP, ocasiona un enlace multicanal serial de gran rapidez. El HPI, permite al host tener acceso a la memoria interna. Timer, que procura dos contadores de 32 bits. La unidad de lógica de bajo consumo de energía, usada para ahorrar energía cuando el CPU esta inactivo.

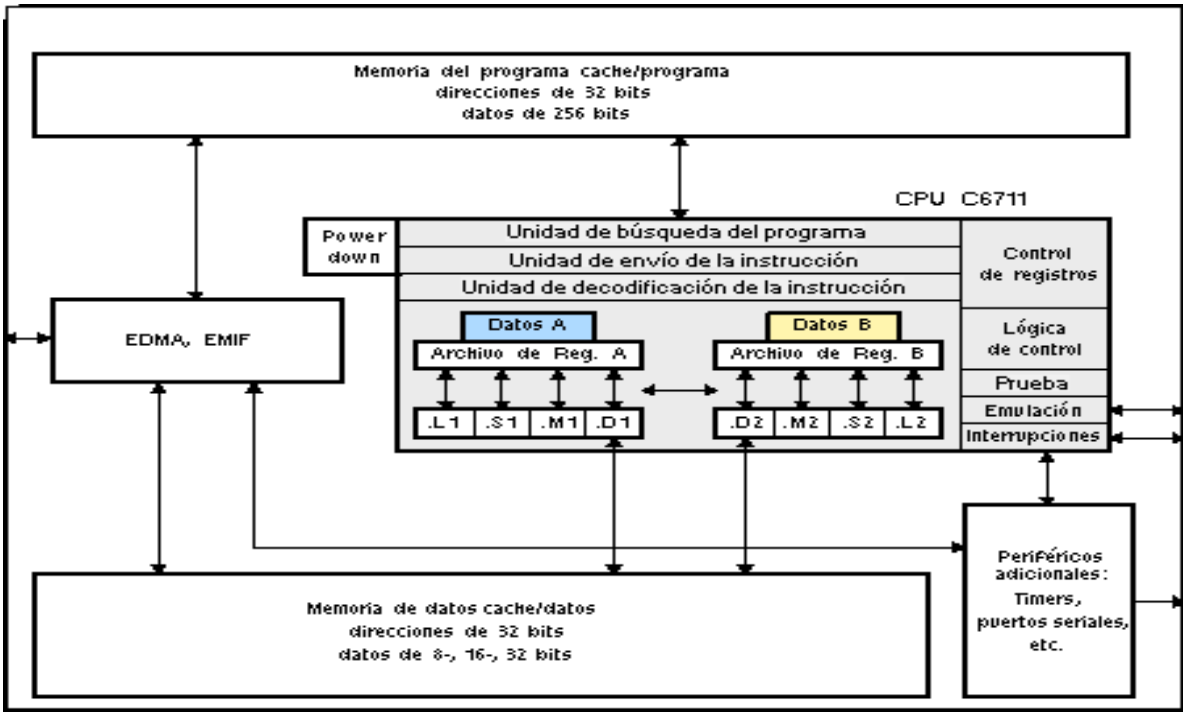


Figura 3.1: Diagrama de bloques del TSM320C6711.

3.2.1. Unidad Central de Procesamiento (CPU)

El CPU contiene:

- Unidad de búsqueda del programa.
- Unidad de envío de la instrucción.
- Unidad de decodificación de la instrucción.
- 32 registros de 32 bits.
- Dos trayectorias de acceso para datos, cada una con cuatro unidades funcionales.
- Control de registros.
- Lógica de control.
- Lógica de prueba (test), emulación, e interrupción.

El CPU tiene dos trayectorias para los datos (A y B) para que ocurra el proceso. Cada trayectoria tiene cuatro unidades funcionales (.L, .S, .M, y .D) y un archivo de registro que tiene 16 registros de 32 bits.

Las unidades funcionales ejecutan operaciones lógicas, de corrimiento, de multiplicación y operaciones de direccionamiento de datos. Todas las instrucciones aceptan operaciones de carga y de almacenamiento sobre los registros. Las dos unidades de direccionamiento de datos (.D1 y .D2) tienen exclusivamente la responsabilidad de la transferencia de datos entre los archivos de registros y la memoria.

La trayectoria de las cuatro unidades funcionales es un simple bus de datos conectado a los registros extremos del CPU así que las unidades pueden intercambiar los datos de un archivo de registro al lado contrario. Para un acceso cruzado el CPU soporta una operación de lectura y escritura por ciclo.

Los conjuntos de las unidades funcionales incluyen:

- Dos multiplicadores.
- Seis unidades aritméticas lógicas (ALUs).
- Dos archivos de registros, cada uno con 16 registros de 16 bits.

Cada unidad funcional esta controlada por una instrucción de 32 bits. Los bloques de búsqueda, envío, y decodificación pueden entregar instrucciones de 32 bits al programa de memoria de las unidades funcionales cada ciclo. El archivo registro de control tiene métodos para configurar y controlar varios aspectos de operación del procesador. El acceso a los registros de control esta sobre la ruta de datos B.

El procesamiento VLIW comienza cuando un instrucción de 256 bits de ancho se busca en la memoria interna. Las instrucciones ligadas de ejecuciones simultaneas (arriba de ocho instrucciones) forman un paquete ejecutado.

3.2.2. Archivo de Registros

Los registros del procesador se emplean para controlar instrucciones en ejecución, manejar direccionamiento de memoria y proporcionar capacidad aritmética. Los registros son espacios físicos dentro del procesador con capacidad de 32 bits hasta 64 bits dependiendo del tipo que se emplee. Los registros son direccionables por medio de una viñeta, que es una dirección de memoria. Los bits, por conveniencia, se numeran de derecha a izquierda (31, 30, 29, 28,... 3, 2, 1, 0), los registros están divididos en dos grupos los cuales tienen un fin específico [11].

Registros de propósito general

Hay dos archivos de registros de propósito general (A y B) en la trayectoria de datos del C6711. Cada uno de estos archivos contiene 16 registros de 32 bits (A0-A15 para el archivo A y B0-B15 para el archivo B). Los registros de propósito general pueden usarse para manejar datos o punteros de direccionamiento.

Los archivos de registros de propósito general soportan datos de 32 y 40 bits de punto fijo. Los datos de 32 bits, pueden almacenarse en cualquier registro de propósito general. Los datos de 40 bits se almacenan en dos registros; los 32 menos significativos del dato (LSB) se colocan en un registro par y los restantes ocho bits mas significativos (MSB) se colocan en los ocho bits menos significativos del registro próximo superior (que es siempre un registro impar). El C6711 también usa ese par de registros para colocar valores de punto flotante de doble precisión de 64 bits.

Registros propósito especial

El TMS320C6711 contiene los siguientes registros:

1. **A0-A15, B0-B15**, dieciséis registros de 32 bits que permiten manejar precisión extendida. Estos registros pueden almacenar números de 32, 40 y 64 bits.

2. **A1, A2, B0, B1 y B2**, cinco registros que pueden utilizarse como registros de condicionamiento.

3. **A4-A7 y B4-B7**, ocho registros auxiliares de propósito general que son comúnmente usados para direccionamiento circular de memoria.

4. **PC**, registro de contador de programa, contiene la dirección de la instrucción ejecutada.

5. **ISTP**, puntero de tabla con servicio de interrupción, señala el comienzo de la tabla con servicio de interrupción.

6. **IFR**, registro de bandera de interrupción, despliega el estado de las interrupciones.

7. **IER**, registro de activación de interrupción, habilita/deshabilita las interrupciones individuales.

8. **IRP**, puntero de retorno de interrupción, contiene la dirección para regresar de una interrupción mascarable.

9. **NRP**, puntero de retorno de interrupción no mascarable, contiene la dirección de retorno de una interrupción no mascarable.

10. **AMR**, registro de modo de direccionamiento, especifica sí utiliza direccionamiento lineal o circular para cada uno de los ocho registros; también contiene el tamaño para el direccionamiento circular.

11. **CSR**, registro de control de estado, contiene el bit de interrupción global, los bits de control de caché y otros bits de control de estado misceláneos.

12. **CPUID**, identificador del CPU. Registro-campo del CSR, únicamente de lectura.

13. **REVID**, identificador de revisión del CPU. Registro-campo del CSR, únicamente de lectura.

14. **EN**, modo actual del CPU.

15. **PGIE**, valor GIE (Global Interrupt Enable) previo. Solo de lectura.

16. **PCC**, control del programa caché. Inaccesible.

17. **DCC**, control de datos caché . Inaccesible.

18. **FADCR**, registro de configuración del sumador de punto flotante, especifica el modo de subdesbordamiento, modos de redondeo, NaN y otras excepciones para la unidad .L.

19. **FAUCR**, registro de configuración auxiliar de punto flotante, señala modos de subdesbordamiento, modos de redondeo, NaN y otras excepciones para la unidad .S.

20. **FMCR**, registro de configuración del multiplicador de punto flotante, especifica modos de subdesbordamiento, modos de redondeo, NaN y otras excepciones para la unidad .M.

Registros de control

La unidad (.S2) puede leer de y escribir hacia los registros de control. Se tiene acceso a cada registro con la instrucción `MVC`.

El C6711 posee tres registros de configuración, para soportar operaciones de punto flotante. Los registros especifican los modos de redondeo para punto flotante para las unidades .M y .L. También contienen campos de bit para advertir si src1 y src2 son NaN (no es un número) o números desnormalizados. Además si resulta con desbordamiento o con subdesbordamiento, es inexacto, infinito o inválido. Hay campos que advierten si una división por cero se realizó o si una comparación fue ejecutada con un NaN.

3.2.3. Unidades funcionales

Las ocho unidades funcionales del C6711 pueden dividirse en dos grupos de cuatro; cada unidad funcional en una trayectoria de datos, es casi idéntica a la unidad correspondiente en la otra trayectoria. Las unidades se describen en la Tabla 3.1.

La mayoría de los buses en el CPU, soportan operaciones de 32 bits, y algunos soportan operaciones de mayor longitud (40 bits). Cada unidad funcional tiene su propio puerto de escritura de 32 bits, en un archivo de registros de propósito general. Todas las unidades terminan en 1 cuando se refieren al archivo de registros A y en 2 cuando se refiere al archivo de registros B. Cada unidad funcional tiene dos puertos de lectura de 32 bits, para cada operando src1 y src2. Las cuatro unidades (.L1, .L2, .S1 y .S2) tienen un puerto extra de 8 bits para ejecutar operaciones de 40 bits. Debido a que cada unidad tiene su propio puerto de escritura de 32 bits, las ocho unidades pueden usarse en paralelo en cada ciclo.

3.2.4. Buses de datos del CPU

El CPU esta constituido principalmente por:

- Dos archivos de registros de propósito general (A y B)
- Ocho unidades funcionales (.L1, .L2, .S1, .S2, .M1, .M2, .D1, .D2,)
- Dos buses de lectura de memoria (LD1 y LD2)
- Dos buses de almacenamiento en memoria (ST1 y ST2)
- Dos buses cruzados entre los archivos de registros (1X y 2X)
- Dos buses de direccionamiento de datos (DA1 y DA2). La Figura 3.2 muestra los buses de datos del CPU C6711.

Unidad Funcional	**Operaciones en punto fijo**	**Operaciones en punto flotante**
Unidad .L (.L1, .L2)	Operaciones de comparación y aritméticas de 32 y 40 bits. Operaciones lógicas de 32 bits. Contador de 1 o 0 mas a la izquierda, para 32 bits. Normalización de 32 y 40 bits. .	Operaciones aritméticas. Operaciones de conversión: DP→SP, INT→DP, INT→SP
Unidad .S (.S1, .S2)	Operaciones aritméticas de 32 bits. Corrimientos de 32/40 bits y operaciones campos de bits en 32 bits. Operaciones lógicas de 32 bits. Saltos. Generación de constantes. Transferencia de registros de/hacia registros (solamente .S2)	Comparación recíproca. Operaciones de raíz cuadrada. Operaciones de valor absoluto. Operaciones de conversión SP a DP
Unidad .M (.M1, .M2)	Operaciones de multiplicación de 16×16 bits, Operaciones de corrimiento variable. Rotación, Multiplicación	Operaciones de multiplicación de 32×32 bits. Operaciones de multiplicación de punto flotante.
Unidad .D (.D1, .D2)	Sumas, restas y cálculos de direccionamiento lineal y circular de 32 bits. Carga y almacenamiento con offset constante de 5 bits. Carga y almacenamiento con offset constante de 15 bits (solo .D2)	Lectura de dobles palabras con offset constante de 5 bits.

Tabla 3.1: Unidades funcionales y operaciones que realizan.

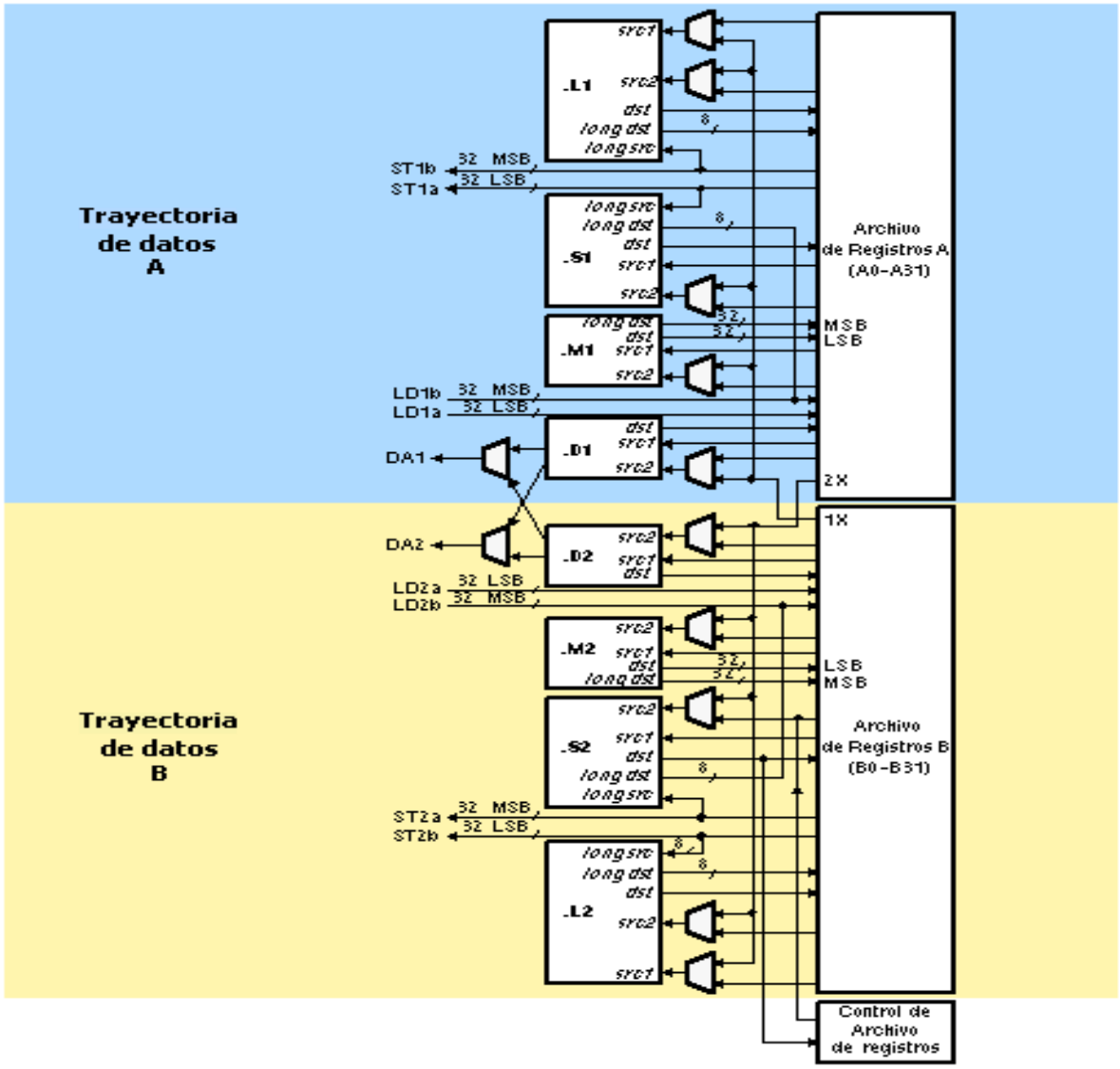


Figura 3.2: Buses de datos del CPU del TSM320C6711.

Buses entre archivos de registros

Cada unidad funcional lee directamente de y escribe directamente hacia el archivo de registros, dentro de su propia trayectoria de datos. Esto es, las unidades .L1, .S1, .D1 y M1 escriben en el archivo de registros A y las unidades .L2, .S2, .D2 y .M2 escriben en el archivo de registros B. Los archivos de registros se conectan a las unidades funcionales del archivo de registros opuestos, a través de los buses cruzados 1X y 2X. Estos buses cruzados permiten a las unidades funcionales, de una trayectoria de datos, tener acceso a operandos de 32 bits, del lado opuesto. El bus cruzado 1X permite a las unidades funcionales del camino de datos A, leer su operando fuente del archivo de registros B. El bus cruzado 2X permite a las unidades funcionales del camino de datos B, leer su operando fuente del archivo de registros A.

Buses de Memoria, Carga y Almacenamiento

Hay dos buses de 32 bits, para cargar los datos de memoria en los registros de almacenamiento: LD1 para el archivo de registros A y LD2 para el archivo de registros B. El C6711 tiene un segundo bus de carga de 32 bits para ambos archivos de registros A y B. Este segundo bus permite a la instrucción LDDW leer simultáneamente dos registros de 32 bits en los lados A y B. Existen además dos buses de 32 bits, ST1 y ST2, para almacenar valores de los registros a la memoria, para cada archivo de registros. Los buses a las unidades .L y .S se comparten con los buses de almacenamiento.

Buses de direccionamiento de datos

Los buses de direccionamiento de datos (DA1 y DA2) de las unidades .D, permiten generar direcciones de datos de un archivo de registros. Con eso se sostienen cargas y almacenamientos en memoria, desde el otro archivo de registros. Sin embargo, las cargas y almacenamientos ejecutados en paralelo, deben cargar al y del mismo archivo de registro. Aunque también existe la alternativa de que ambos usen un bus cruzado al registro opuesto.

3.3. Organización de la memoria

En los DSPs la memoria de instrucciones y datos está integrada en el propio chip. Una debe ser tipo ROM, destinada a contener el programa de instrucciones que gobierna la aplicación. La otra memoria será tipo RAM, y se destina a guardar variables y datos.

3.3.1. Mapa de memoria

La Figura 3.3 muestra la organización de la memoria del TMS320C6711.

Dirección inicial	Mapa de memoria 0 (Ejecución directa)	Bloque (Bytes)	Dirección inicial	Mapa de memoria 1 (Modo Boot)	Bloque (Bytes)
0000 0000h	Memoria externa CE0	16 M	0000 0000h	RAM interna de programa	64 K
0100 0000h	Memoria externa CE1	4 M	0100 0000h	Reservada	4 M-64 K
0140 0000h	RAM interna de programa	64 K	0140 0000h	Memoria externa CE0	16 M
0141 0000h	Reservada	4 M-64 K	0141 0000h	Memoria externa CE1	4 M
0180 0000h	Periféricos internos	8 M	0180 0000h	Periféricos internos	8 M
0200 0000h	Memoria externa CE2	16 M	0200 0000h	Memoria externa CE2	16 M
0300 0000h	Memoria externa CE3	16 M	0300 0000h	Memoria externa CE3	16 M
0400 0000h	Reservada	1 G-64 M	0400 0000h	Reservada	1 G-64 M
4000 0000h	Expansión del bus (para el C6202)	1 G	4000 0000h	Expansión del bus (para el C6202)	1 G
8000 0000h	RAM interna de datos	64 K	8000 0000h	RAM interna de datos	64 K
8001 0000h 8002 0000h	Reservada	2 G-64 K	8001 0000h 8002 0000h	Reservada	2 G-64 K

Figura 3.3: Mapa de Memoria del TSM320C6711.

El rango total de direcciones de memoria de los dispositivos C6000 es de 4 Gbytes (correspondiente a la representación de dirección interna de 32 bits). Cada mapa de memoria esta dividido en memoria interna de programa, memoria interna de datos, espacios de memoria externa, y espacios de periféricos internos.

Los dos tipos de operación de estos dispositivos son ejecución directa y modo de arranque. En ejecución directa, el programa comienza cargando de la dirección externa 0, mientras que en el modo de arranque, el programa es cargado con la memoria externa o de un host externo antes de que comience la ejecución en una dirección interna 0.

Cuando se permite el mapeo de la memoria, el depurador comprueba cada acceso a la memoria del mapa de memoria estipulado. Si se quiere tener acceso a una área indefinida o protegida, el depurador presenta el valor predeterminado en vez de tener acceso a la tarjeta. Esto se comprueba en el software y no en el hardware. El depurador no puede evitar que el programa intente entrar a una memoria inexistente. Existen dos maneras para definir el rango del mapa de memoria de la tarjeta:

- Definir el mapa de memoria de manera interactiva, mientras se usa el depurador. Esto puede ser inconveniente porque, en muchos casos, se configura un mapa de memoria antes de depurar y este mapa se usa en todas las sesiones siguientes.
- Definir el mapa de memoria usando funciones predefinidas GEL (General Extension Language). Gel proporciona un conjunto completo de funciones para el mapeo de la memoria. El método mas fácil para definir el mapa de memoria es usar las funciones en un archivo de texto GEL y ejecutarlo al inicio.

Un programa en lenguaje ensamblador requiere de directivas para reubicar los bloques de código y datos; cuando se enlaza este programa, se debe especificar donde se ubican estas secciones.

3.3.2. Secciones de un programa y memoria

La unidad mas pequeña de un archivo objeto se llama sección. Una sección es un bloque de código o de datos que ocupan espacios continuos en un mapa de memoria [9].

El Ensamblador y el Enlazador crean archivos COFF que pueden ejecutarse y también proporcionan directivas que permiten crear y manipular estas secciones. Cada sección de un archivo objeto esta separada y es distinta. Existen dos tipos básicos de secciones:

- Secciones inicializadas: contienen datos o código, se enlazan con la memoria ROM o RAM. Las secciones `.text` y `.data` son secciones inicializadas del programa `fuente.asm`.

  La sección `.text` contiene código ejecutable.

La sección `.data` contiene datos inicializados.

- Secciones no inicializadas: reserva espacio en el mapa de memoria para datos no inicializados. Estas secciones se enlazan con la memoria RAM.

La sección `.bss` es no inicializada.

Algunas directivas del ensamblador permiten asociar varias partes del código y datos con las secciones apropiadas. El Ensamblador construye estas secciones durante el proceso de ensamble, y crea un archivo objeto organizado como muestra la Figura 3.4 .

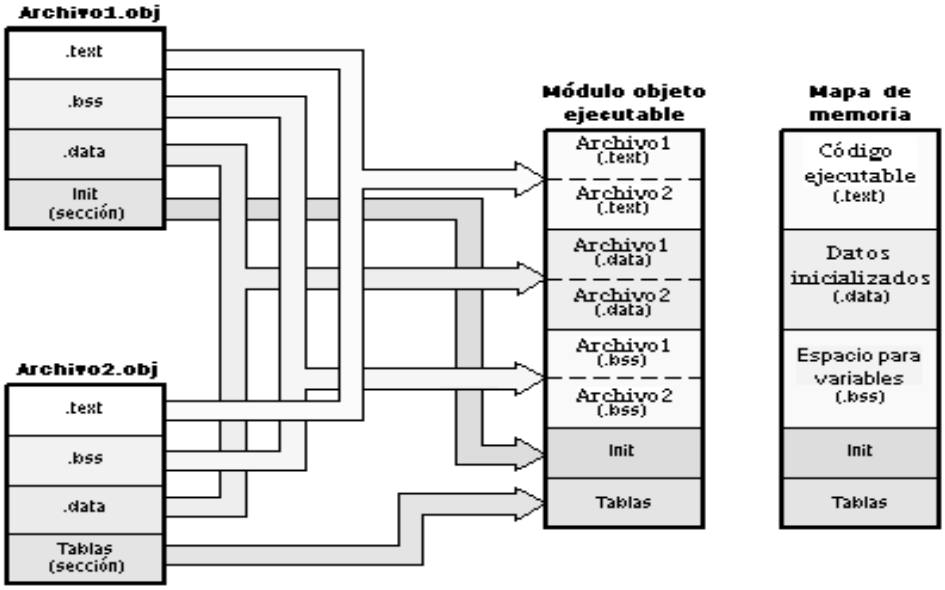


Figura 3.4: Combinación de secciones para formar un Módulo Objeto Ejecutable.

Una de las funciones del Enlazador es reubicar las secciones en el mapa de memoria de la tarjeta del sistema; esta función se llama asignación (allocation). Porque estos sistemas contienen distintos tipos de memoria y usando las secciones se usa la memoria de la tarjeta eficientemente. Todas las secciones pueden reubicarse independientemente, se puede colocar cualquier sección en cualquier bloque de asignación de memoria. Por ejemplo, se puede definir una sección que contenga una rutina de inicialización y después asignar la rutina en la parte del mapa de memoria que contiene la ROM.

La Figura 3.5 muestra la relación entre las secciones en un archivo objeto y una tarjeta de memoria hipotética.

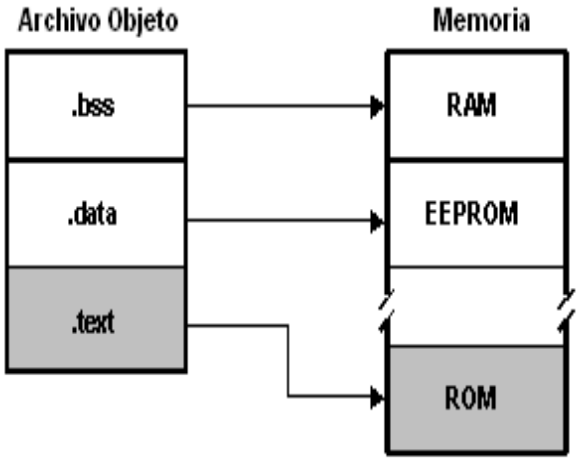


Figura 3.5: Partición de memoria en bloques lógicos.

El Enlazador tiene dos funciones principales relacionadas a las secciones.

1. Usa las secciones en los archivos objeto COFF como bloques; esto combina las secciones de entrada (cuando mas de un archivo se enlaza) para crear las secciones de salida en un módulo de salida ejecutable COFF.

2. Escoge las direcciones de memoria para las secciones de salida

Las directivas del Enlazador que soportan estas funciones son:

- La directiva *MEMORY*, que permite definir el mapa de memoria de una tarjeta. Se pueden nombrar las porciones de la memoria y especificar sus direcciones iniciales y sus longitudes.

- La directiva *SECTIONS* comunica al Enlazador como combinar las secciones de entrada con las secciones de salida y donde colocar las secciones de salida en la memoria.

Las subsecciones (Subsections) permiten manipular las secciones con mayor precisión.

En el Programa 3.1 `lnk.cmd`, del proyecto MATRICES.pjt. Cuando use la directiva MEMORY, asegurese de identificar todos los rangos de la memoria que se disponen para cargar el código. Esta directiva se especifica en el archivo, seguida por una lista de especificaciones para los rangos permitidos.

Programa 3.1: Distribución de direcciones de memoria.

```
MEMORY {
        PMEM:   o = 00000020h   l = 0000ffe0h
        EXT0:   o = 00400000h   l = 01000000h
        EXT1:   o = 01400000h   l = 00400000h
        EXT2:   o = 02000000h   l = 01000000h
        EXT3:   o = 03000000h   l = 01000000h
        BMEM:   o = 80000000h   l = 00010000h
}


SECTIONS {
    .text       >       PMEM
    .stack      >       BMEM
    .bss        >       BMEM
    .cinit      >       BMEM
    .cio        >       BMEM
    .const      >       BMEM
    .data       >       BMEM
    .switch     >       BMEM
    .sysmem     >       BMEM
    .far        >       EXT2
}
```

La sintaxis general para la directiva MEMORY es:

```
MEMORY
{
        nombre[(attr)] : origin = constante, length = constante[, fill = constante]
        :
        nombre[(attr)] : origin = constante, length = constante[, fill = constante]
}
```

- *Nombre*, nombra un rango de la memoria, puede tener de 1 a 64 caracteres; mayúsculas o minúsculas incluyendo otros caracteres. Los nombres no tienen significado para el Enlazador; simplemente identifican los rangos de la memoria. Los nombres de los rangos de la memoria son internos para el Enlazador y no se conservan en el archivo de salida o en la tabla de símbolos. Todos los rangos deben tener nombres únicos y no deben solaparse.

- *attr*, especifica uno de los cuatro atributos asociados con el nombre del rango. Son opcionales, cuando se usan deben estar encerrados en paréntesis. Los atributos

limitan la asignación de las secciones de salida en los rangos de memoria, si no se usan se asigna cualquier sección de salida en cualquier rango sin restricciones. Cualquier memoria sin especificación de atributos tiene cuatro. Los atributos validos incluyen:

- R, especifica que la memoria es de lectura.
- W, indica que la memoria puede ser de escritura.
- X, indica que la memoria contiene código ejecutable.
- I, especifica que la memoria puede ser inicializada.

- *origen*, indica la dirección inicial de un rango de la memoria; se escribe o, org, u origin. El valor en bytes, es una constante de 32 bits y puede estar en decimal, octal, o hexadecimal; length, especifica la longitud del rango de la memoria, puede usar length, len, o l. El valor también puede indicarse en cualquier formato mencionado.

- *fill*, indica un lleno de caracteres para el rango de la memoria, se escribe fill o solo f. El llenado es opcional. El valor es una constante entera de 16 bits y puede ser decimal, octal, o hexadecimal. El valor de llenado se usa para ocupar áreas en los rangos de memoria que no son asignadas a una sección.

La directiva MEMORY identifica los rangos de la memoria que estan físicamente presentes en la tarjeta y es usada por el programa.

Con la directiva SECTIONS, el Enlazador puede manipular una sección de salida que cubra el espacio dentro del rango de la memoria en la que se coloco. En vez de modificar el archivo de comandos del Enlazador, se puede mover la sección dentro de algunas áreas, el mapa de memoria comunica al depurador (debugger) a que áreas de la memoria puede o no puede tener acceso. Los mapas de memoria cambian dependiendo de la aplicación. Comúnmente, el mapa de memoria se acopla a la definición de MEMORY del archivo de comandos del Enlazador.

La directiva SECTIONS:

- Describe como se combinan las secciones de entrada y las secciones de salida.
- Define las secciones de salida del programa ejecutable.
- Especifica donde se colocan las secciones de salidas en la memoria.
- Permite renombrar las secciones de salida

La directiva SECTIONS se especifica en un archivo de comandos, seguida de una lista de parámetros. La Sintaxis general es:

```
SECTIONS
{
        nombre: [property [, property] [, property] . . . ]
        :
        nombre: [property [, property] [, property] . . . ]
}
```

Los modos de direccionamiento determinan la manera de acceso a la memoria. Estos especifican cómo un dato es acogido, cómo un operando es recuperado de un registro o de una localización de memoria.

3.4. Modos de direccionamiento

En el C6711 los modos de direccionamiento son lineales, aunque también existe el modo de direccionamiento circular. El modo de direccionamiento se especifica con el registro AMR (Modo de Direccionamiento). Con todos los registros se puede ejecutar el direccionamiento lineal. Solo en ocho de ellos se puede ejecutar el direccionamiento circular: del A4 al A7 (usados por la unidad .D1) y del B4 al B7 (usados por la unidad D2). Ninguna otra unidad puede ejecutar direccionamiento circular. Las instrucciones LDB/LDH/LDW, STB/STH/STW, ADDAB/ADDAH/ADDAW, y SUBAB/SUBAH/SUBAW se apoyan en el registro AMR, para determinar que tipo de cálculo de direccionamiento es ejecutado por esos registros.

El CPU del C6711 tiene arquitectura de carga/almacenamiento, lo que significa que la única manera de tener acceso a datos en memoria es con la instrucción de carga o almacenamiento.

Algunos modos de direccionamiento son los siguientes:

a) ***R** El direccionamiento indirecto de memoria se representa con el símbolo *. Por ejemplo, *A0 contiene la dirección de una localización de memoria donde el valor del dato esta almacenado; el contenido en memoria es el dato con dirección especificado o señalado por A0.

b) ***+R[d].** R especifica la dirección de memoria, que se incrementa a una dirección mas alta definida por d, así que la nueva dirección es la dirección R+d (d es un entero sin signo). Si se usa el signo menos (-) la dirección se decrementa.

c) ***++R[d].** El contenido de la memoria es una dirección preincrementada a R+d. La dirección se modifica cuando se usa este modo de direccionamiento. Con un doble signo menos (–) predecrementa la dirección de memoria a R-d.

d) ***R[d]++.** El contenido de la memoria es la dirección postincrementada R+d. R se modifica como en el caso anterior. Con un doble signo menos (–) postdecrementa la dirección de memoria a R-d.

El Programa 3.2 incluye algunos modos de direccionamiento.

Programa 3.2: Modos de direccionamiento.

```
                .data
dir_A           .int        9,7,6,5,4,1
                .text
                MVKL        dir_A,A0
                MVKH        dir_A,A0
                ZERO        A2
                LDW         *A0,A1
                NOP         5
                LDW         *+A0[3],A1
                NOP         5
                LDW         *++A0[2],A1
                NOP         5
                LDW         *A0++[1],A1
                NOP         5
                LDW         *A0,A1
                NOP         5
```

Analizando el Programa 3.2, las instrucciones MVKL y MVKH, fijan la dirección de dir_A (0x 80000000) en el registro A0, este registro puede ser manipulado con los modos de direccionamiento indirecto, mostrados anteriormente, el cambio de la dirección puede observarse en el registro `A0` y el contenido de los datos en el registro `A1`.

Las operaciones como la suma, la resta y la multiplicación, son fundamentales en un procesador digital de señales. Una operación muy importante es la multiplicación acumulada, que se usa para un número de aplicaciones que requieren filtrado, correlación, y análisis de espectro. Ya que la multiplicación es comúnmente ejecutada y es esencial para muchos algoritmos de procesamiento digital de señales, esta se ejecuta en un simple ciclo. Un procesador digital de señales típico contiene un multiplicador/acumulador interno para operaciones rápidas y eficientes.

3.5. Conjunto de instrucciones

El procesador TMS320C6711 tiene una arquitectura y un conjunto de instrucciones algo diferentes a las de los procesadores anteriores a él y a los de punto fijo. Aun cuando el TMS320C6711 contiene un rico y poderoso conjunto de instrucciones comparado con los procesadores de punto fijo. El Apéndice A contiene un resumen del conjunto de instrucciones.

Un programa de lenguaje ensamblador debe ser un archivo de texto en código ASCII. Cualquier línea de código ensamblador puede incluir cinco campos ordenados.

La sintaxis general es la siguiente:

[etiqueta[:]] [||] mnemónico [unidad] [operandos] [;comentarios]

- *Etiqueta*: una etiqueta identifica una línea de código, o una variable y representa una dirección de memoria, que contiene cualquier instrucción o dato. Los dos puntos posteriores a la etiquetas son opcionales. Las etiquetas deben reunir las siguientes condiciones:
  - El primer carácter de la etiqueta debe ser una letra o un guión bajo(_) seguido por una letra.
  - Las etiquetas deben estar en la primer columna del archivo de texto.
  - La etiqueta puede incluir hasta 32 caracteres diferentes.
- *Barras Paralelas* (||), para indicar que una instrucción se ejecuta en paralelo con la instrucción previa, se indica con las barras paralelas (||). Este campo debe ser un espacio en blanco, para una instrucción que no se ejecuta en paralelo, con la anterior.
- *Condición*, el C6711 tiene cinco registros disponibles para las condiciones A1, A2, B0, B1 y B2.
- *Instrucción*, las instrucciones en ensamblador son directivas o mnemónicos:
  - Directivas. Son comandos para el ensamblador `ASM6x` que controlan el proceso de ensamblado o que definen la estructura de los datos (constantes o variables), en el programa de lenguaje ensamblador. Todas las directivas del ensamblador comienzan con un punto.
  - Mnemónicos. Son las instrucciones verdaderas del microprocesador que se encuentran en rutinas y ejecutan la operación del programa. Los mnemónicos comienzan a partir de la segunda columna.

- *Unidades funcionales*, estas unidades ya explicadas anteriormente, son opcionales para especificarlas en el código. La especificación puede usarse para documentar, que recurso(s) utiliza cada instrucción.
- *Operandos*, las instrucciones tienen los siguientes requerimientos para manejar los operandos del código ensamblador

  - Todas las instrucciones requieren un operando destino.
  - La mayoría de las instrucciones requieren uno o dos operandos fuente.
  - El operando destino debe estar en la misma unidad de registro que el operando fuente.
  - Un operando fuente de cada archivo de registros por paquete de ejecución, puede llegar a un archivo de registro opuesto del otro operando fuente.

  Cuando un operando llega de otro archivo de registro, la unidad incluye una X, indicando que la instrucción utiliza uno de los buses cruzados. Las instrucciones usan tres tipos de operandos para el acceso a datos:

  - Operandos de registro. Señalan al registro que contiene el dato.
  - Operando constante. Especifica el dato dentro del código ensamblador
  - Operador puntero. Contiene la dirección del valor de datos

- Comentarios, los comentarios proporcionan la documentación del código. Las siguientes son directrices, para usar los comentarios en código ensamblador.

  - Un comentario puede comenzar en cualquier columna, cuando esta precedido de un punto y coma (;).
  - Un comentario debe comenzar en la primer columna, cuando esta precedido de un asterisco (*).
  - Se recomienda su uso, aunque no son indispensables.

Por ejemplo, la siguiente línea de código

```
LOOP SUB 1,A0 ; resta 1 de A0
```

se compone de una etiqueta (LOOP), que comienza en la primer columna, seguida de una instrucción de resta SUB; el operando 1,A0; y un comentario. Uno o más espacios en blanco deben separar cada uno de los campos.

Cuando se escribe un programa en lenguaje ensamblador, no es necesario indicar la unidad que se usa, esta especificación es opcional.

Si se especifica esta unidad se controla la trayectoria del registro usado en la instrucción lo que ayuda al optimizador de ensamblador y a la asignación de registros.

También en un programa en lenguaje ensamblador existen directivas para el control del proceso. Estas directivas se usan para que el código sea optimizado por el optimizador de ensamble.

Para que un programa en código ensamblador sea eficiente, el C6711 usa los siguientes recursos: buses multiples, unidades funcionales, segmentación encauzada al CPU y organización de la memoria. Esto incluye el uso de instrucciones paralelas, la eliminación de retardos o NOPs, bucles y el formato de los datos.

Tipos de Instrucciones.

1. **Instrucciones matemáticas para suma, resta y multiplicación** La instrucción

   ```
   ADD A1, B1, B2
   ```

   Suma los valores de A1 y B1 y almacena el valor del resultado en B2. Remplazando la instrucción ADD por SUB puede restar B1 de A1, con el resultado almacenado en B2. La instrucción

   ```
   MPYI A3,B3,A5
   ```

   Multiplica el contenido del registro A3 por el contenido del registro B3, y almacena el valor del resultado en A5. Esta es una instrucción de tres operandos enteros.

2. **Instrucciones para cargar y almacenar**

   Una palabra de 32 bits puede cargarse de la memoria a un registro o almacenarse en un registro. Observe el código del Programa 3.3.

   Las primeras instrucciones, `MVKL` y `MVKH`, cargan la dirección del arreglo dir_A en el registro `A1`. Después la instrucción LDW, carga el primer valor en `A2` dentro de la memoria, cuya dirección esta especificada por `*A1++[1]`. `A1` se incrementa después para apuntar a la próxima dirección de memoria (con un desplazamiento de uno).

Programa 3.3: Acceso de memoria.

```
                .data
RE:             .space      16
dir_A           .float      1.2,2.4,3.9
                .text
                MVKL        dir_A,A1
                MVKH        dir_A,A1
                MVKL        RE,A6
                MVKH        RE,A6
                LDW         *A1++[1],A2
                NOP         5
                MPYSP       A2,A2,A2
                NOP         3
                STW         A2,*A6++[1]
                NOP         1
                LDW         *A1++[1],A2
                NOP         5
                MPYSP       A2,A2,A2
                NOP         3
                STW         A2,*A6++[1]
                NOP         1
                LDW         *A1++[1],A2
                NOP         5
                MPYSP       A2,A2,A2
                NOP         3
                STW         A2,*A6++[1]
                NOP         1
```

Observe que la `W` en `LDW`, determina en la dirección un dato del tipo palabra (word). También se usan las siguientes instrucciones, dependiendo del tipo del dato, para bytes (8 bits) LDB, para media palabra (16 bits) LDH.

Para almacenar datos a una dirección se usan las instrucciones `STB`, `STH` o `STW`, dependiendo del tipo de dato.

El Programa 3.3 eleva al cuadrado los números indicados por el arreglo `dir_A` y los almacena en el espacio reservado por el vector RE cuya dirección es la indicada por el registro `A6`.

3. **Instrucciones de bifurcación**

   Una instrucción de bifurcación normal se ejecuta en cinco ciclos y posiblemente pueden evitarse. Una bifurcación atrasada, con o sin condición, también puede ejecutarse.

   La bifurcación condicional con la instrucción de retardo B es para proseguir a la siguiente instrucción o ir a la instrucción con una etiqueta.

4. **Instrucciones paralelas**

   El símbolo paralelo ||, escrito en la primer columna, define que la primer instrucción esta en paralelismo con la instrucción de anterior a esta; en consecuencia, si se localizan dentro de un ciclo repetitivo, también se ejecuta el número de veces, que se defina.

   Otros tipos de instrucciones de que se dispone, son las lógicas: AND, OR, NOT y XOR para manipulación de bits, estas se usan en un proceso de decisión. Un bit particular puede examinarse y tomar una decisión con base en el resultado. Un bit particular puede evaluarse junto con una instrucción.

3.6. Directivas

Las directivas del ensamblador empiezan con un punto como `.set`. Una directiva del ensamblador es un mensaje para el ensamblador y no es una instrucción. Esto se determina durante el proceso de ensamblado y no ocupa espacio de memoria como lo hace una instrucción. Por ejemplo, las direcciones de comienzo de diferentes secciones se especifican con directivas del ensamblador, de ese modo se elimina el enlazado.

Las siguientes son las directivas del ensamblador más usadas y algunas se mostrarán a través de los ejemplos.

A	.set	5	A se fija al valor 5
B	.word	k	B se inicia a un valor entero k de 32 bits
C	.float	k	C se inicia con un valor de punto flotante k de 32 bits
	.text		Ensambla una sección del programa en la memoria, comúnmente contiene el código ejecutable
	.data		Ensambla datos inicializados del programa en la memoria
	.sect	"mysect"	Ensambla una sección definida por el usuario mysect
	.def	símbolo	Identifica el símbolo definido en un módulo y que se usa en otros módulos.
	.global	símbolo	Identifica uno o mas símbolos globales (externo)
	.ref	símbolo	Identifica el símbolo usado en un módulo y que esta definido en otro módulo
	.loop	[expr]	Comienza un ciclo repetitivo de un bloque; el contador del ciclo se determina por expr.
	.endloop		Finaliza el código de .loop
	.end		Fin del programa
	.if	cond	Ensambla el bloque de código si cond es verdadera
	.endif	expr	Finaliza el bloque if
A	.space	n	Reserva n bytes en una sección con A como la dirección inicial del espacio reservado
label	.cproc	[variables]	Inicia un procedimiento llamado por C/C++. Debe usarse con `.endproc`

3.7. Programación

El modo combinado con funciones del ensamblador llamadas desde C, hace que este sea mas manejable y sostenible que el código ensamblador, un programa en C no logra la eficiencia y la rapidez de procesamiento de un programa en código ensamblador. Muchas aplicaciones son potentes y pueden necesitar funciones de tiempo crítico en código ensamblador. Este ejemplo da mas familiaridad con las instrucciones del TMS320C6711, las directivas del ensamblador y con las herramientas asociadas [29, 25].

El Programa 3.4 muestra el listado de `Vector.asm` para un vector de cuatro elementos 68, 11, 6 y 80.

La directiva del ensamblador `.data` especifica que la sección comienza en una localización de memoria, asignada en el proceso de enlace. La directiva del ensamblador `.int` (también hay instrucción .float) define los valores como constantes enteras de 32 bits y almacenados en localizaciones de memoria consecutivas comenzando en la dirección especificada por `dir_V`.

Programa 3.4: Vector.asm

```
                    .data
dir_V               .int            68,11,6,80
```

El Programa 3.5 extrae los valores del vector cuya dirección comienza en `dir_V`.

Programa 3.5: Valores.c

```
#include <stdio>

void main() {
            float a;

            int j, i = 4, *point;

            point = (int *) 0x80004054;

            for(j=0; j<i; j++)
            {
                a = point[j];
            }
}
```

El proceso para ejecutar este proyecto es el siguiente:

1. Abra un proyecto con CCS, **Project** → **New**.
2. En el campo **Project name**, escriba `P_capitulo3` y elija **Finalizar**
3. Copie ambos programas en archivos diferentes. Seleccione **File** → **New** → **Source File**.
4. Escoja **File** → **Save as**, en esta ventana introduzca el nombre para el primer programa, escriba `Vector`, y elija la extensión **Assembly Source Files** (`*.asm`). Por último haga clic en **Save**. Repita lo mismo para el segundo programa, pero ahora el nombre `Valores` y la extensión **C Source Files** (`*.c`).
5. Seleccione **Project** → **Add Files to Project**, en la ventanilla aparece el nombre del archivo, elija la extensión desplegando menú y haga doble clic sobre icono que representa al programa. En la ventana del Proyecto, se deben observar ambos programas.
6. El proyecto requiere un programa de administración de memoria, usaremos el programa lnk.cmd. En el menú **Project** seleccione **Add Files to Project**, en el cuadro de dialogo **Buscar en:** encuentre la ruta:

   `C:\ti\c6000\cgtools\lib\`

   En **Tipo** seleccione **All Files (*.*)** y haga doble clic sobre él.
7. Incluya en el proyecto, las librerías de C, que se localizan en:

   `C:\ti\c6000\cgtools\lib\rts6701`

8. Haga clic en el icono para compilar, ensamblar y enlazar el proyecto.
9. Ejecute un clic sobre el menu **File** → **Load Program**. Haga clic en la carpeta Depurador (Debug), pulse el botón **Abrir**. Haga doble clic sobre el nombre del programa con extensión `.out`. Confirme que el cuadro **Tipo** tenga la extensión `.out`.
10. Con el programa en la ventana Disassembly, vaya al menú **Debug** → **Go main** y abra la ventana de observar variables (Watch window)en el menú **View**
11. Con la tecla F8, ejecute las instrucciones y observe los resultados. Figura 3.7.

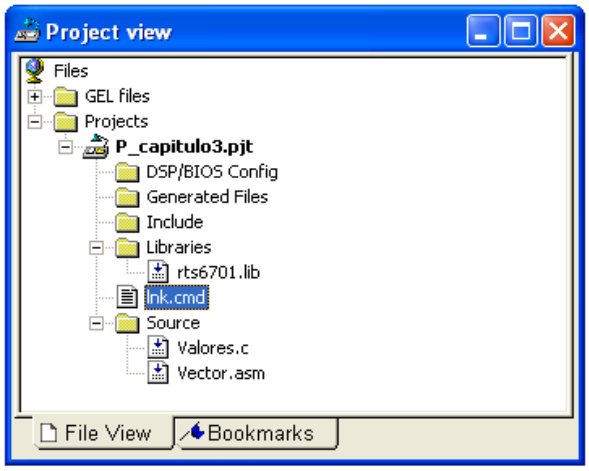


Figura 3.6: Integración de archivos en el proyecto.

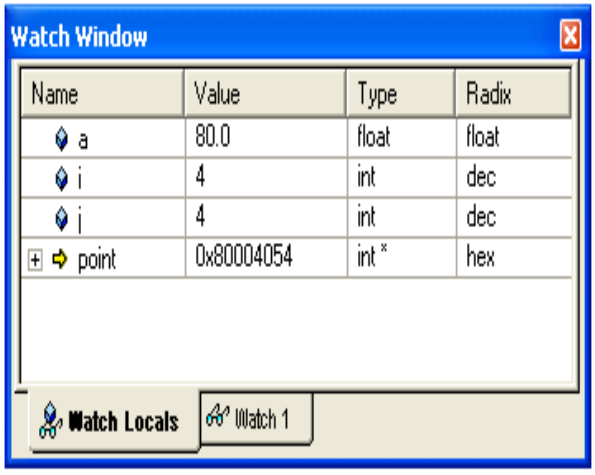


Figura 3.7: Ventana de variables.

Capítulo 4

Sistema básico de entrada/salida

- Circuito de Interfaz Analógico (AIC)
- Sistema básico de entrada/salida del DSP
- DSP/BIOS
- Interrupciones

Este capítulo expone las características generales del sistema básico de entrada/salida, incluyendo el esquema de interrupciones.

4.1. Introducción

Las aplicaciones típicas usando las técnicas del DSP requieren al menos del sistema básico mostrado en la Figura 4.1, el cual se compone de una entrada analógica y una salida analógica [32].

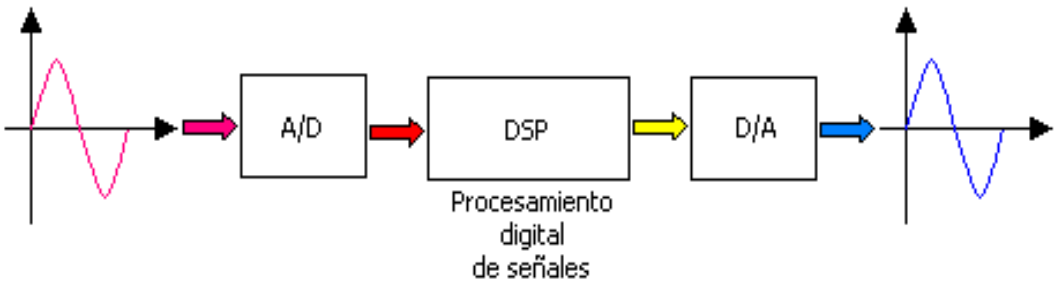


Figura 4.1: Sistema del DSP.

En la entrada existe un filtro contra el efecto alias para eliminar las frecuencias menores a la frecuencia de Nyquist, definida como la mitad de la frecuencia de muestreo. De otra forma, ocurre dicho efecto, en este caso una señal con una frecuencia mayor que la mitad de F_s se hace pasar como una señal con frecuencia baja. El teorema de muestreo dice que la frecuencia de muestreo debe ser al menos dos veces la frecuencia f de la componente más alta en una señal o

$$F_s > 2f$$

De aquí

$$1/T_s > 2(1/T)$$

donde T_s es el periodo de muestreo, o

$$(1/2)T > T_s$$

y

$$T_s < (1/2)T$$

El periodo de muestreo debe ser menor que un medio del periodo de la señal. Por ejemplo, si asumimos que el oído no puede detectar frecuencias sobre los 20 KHz., podríamos hacer un muestreo a una señal de música en $F_s > 40$ KHz. (comúnmente en 44.1 o 48 KHz.) para remover la frecuencia de las componentes mayores que 20 KHz. Entonces usamos un filtro pasa bajas en la entrada con un ancho de banda o frecuencia de corte en 20 KHz. para evitar el efecto alias [31, 21].

La Figura 4.2 muestra una señal con efecto alias. Se tiene una frecuencia de muestreo $F_s = 4$ KHz, o un periodo de muestreo de $T_s = 0.25$ ms. Es imposible determinar si la señal de 5 KHz. o la señal de 1 KHz., es la señal que se representa por la secuencia (0, 1, 0, -1). Una señal de 5 KHz. aparecerá como una señal de 1 KHz., de aquí que, la señal de 1 KHz. es una señal con efecto alias. De igual manera, una señal de 9 KHz. aparecería como una señal de 1 KHz. con efecto alias.

Los ADC convierten la señal de entrada analógica a una representación digital, tratada por un procesador digital de señales. El máximo nivel de la señal de entrada se determina por el convertidor analógico a digital (ADC). Los niveles discretos se usan para representar la salida de la señal. El número de niveles se basa en el rango del nivel de la señal de entrada y el número de bits del ADC. Después de que la señal capturada se procesa, el resultado necesita enviarse al exterior. A la salida se encuentra un convertidor digital a analógico (DAC) que realiza la operación inversa del ADC, con diferentes niveles de salida producidos por el DAC y basados en su entrada. Un filtro a la salida forma o reconstruye los niveles en una señal analógica equivalente.

El C6711 cuenta con un circuito de interfaz analógica, un circuito integrado que ejecuta conversiones seriales A/D y D/A.

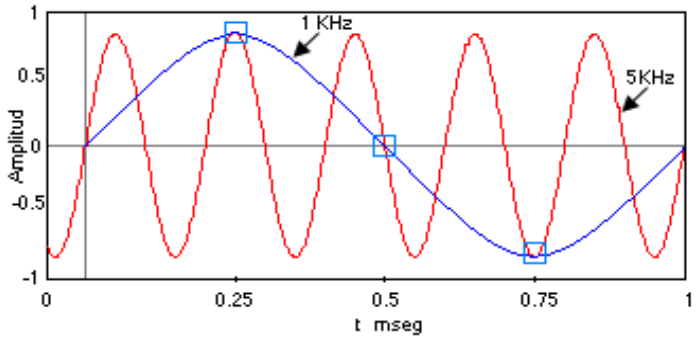


Figura 4.2: Onda senoidal con Alias.

4.2. Circuito de interfaz analógica (AIC)

La tablilla del DSK incluye un chip del circuito de interfaz analógica (AIC) que conecta al puerto serial multicanal con buffer (McBSP0) con el DSP del C6711. El AIC TLC320AD535 contiene doble canal de voz/datos, es un dispositivo de banda ancha de señales mixtas. El TLC320AD535 consta de un *códec* con doble canal y un circuito híbrido con dos puertos seriales independientes de 16 bits para la comunicación con el procesador del *host* y otras funciones lógicas. El *códec* del TLC320AD535 ejecuta las funciones de conversión de analógico a digital y de digital a analógico, filtrado pasa-bajas, control de ganancias analógicas de entrada y de salida, sobremuestreo interno con interpolación. El máximo rango de muestreo es de 11.025 KHz [7].

La Figura 4.3 muestra el diagrama de bloques del AIC TLC320AD535 con una entrada, una salida, un ADC y un DAC de 16 bits, y filtros de entrada y salida.

La cuantización del error o el nivel de ruido del ADC estaba inmiscuido con el procesador de punto fijo. Un ADC solo usa el mejor valor estimado digital para representar una entrada. Por ejemplo, considerando un ADC con una longitud de palabra de 8 bits y un rango entrada de ± 1.5 volts. Los pasos representados por el ADC son: (rango de entrada)/$(2^8) = 3/256 = 11.72$ mv. Esto produce errores que pueden elevarse a $\pm(11.72mv)/2 = \pm 5.86$ mv. Únicamente una mejor estimación usada por el ADC puede representar valores de entrada que no son múltiplos de 11.72 mv. Con el ADC de 8 bits, 2^8 o 256 niveles diferentes se puede representar la señal de entrada. En un ADC con una longitud más larga de palabra como por ejemplo, un ADC de 16 bits (generalmente el más común) puede reducir el error de cuantificación, produciendo una resolución más alta. Muchos bits en un ADC, tienen una mejor representación en una señal de entrada.

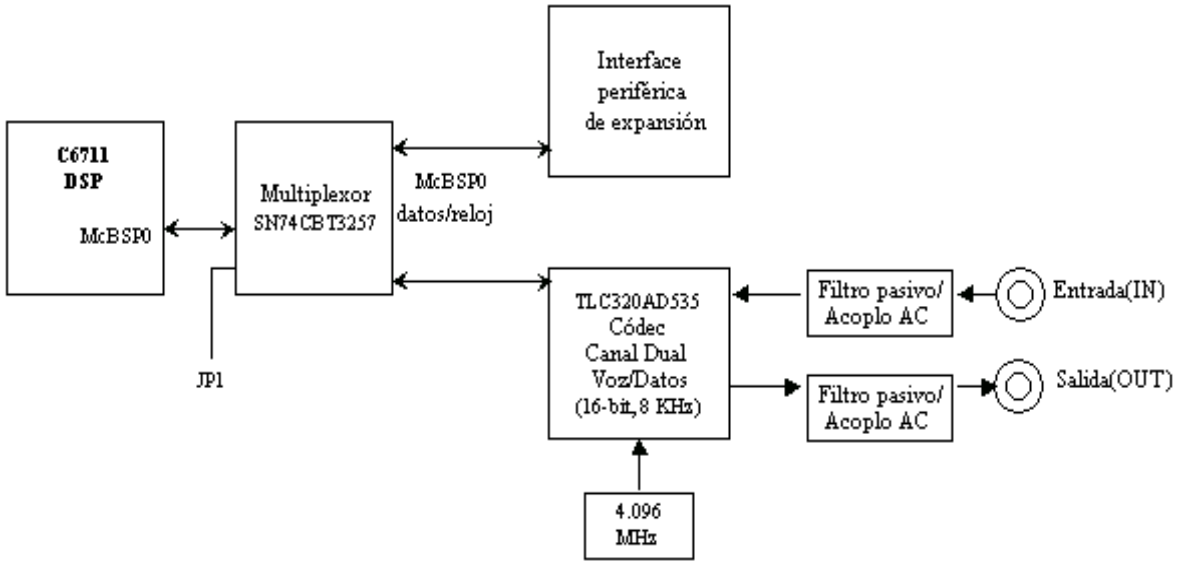


Figura 4.3: Diagrama de bloques de la interfaz analógica (AIC).

La entrada primaria del AIC (`IN`), puede tener acceso por un conector en la tablilla del DSK. El filtro pasabandas en la entrada es auxiliar y puede programarse para una frecuencia de corte deseada o un ancho de banda basado en la frecuencia de muestreo. El filtro pasa bajas de reconstrucción a la salida es fijo.

El componente DSP/BIOS tiene la capacidad de análisis en tiempo real mediante un kernel, ejecución de monitoreo y transmisión de archivos.

4.3. DSP/BIOS

DSP/BIOS es un kernel escalar de tiempo real. Esta diseñado para aplicaciones que requieren programación en tiempo real y sincronización, comunicación entre el host y la tarjeta, o instrumentación en tiempo real. El DSP/BIOS proporciona múltiples tareas con jerarquización, abstracción de hardware, análisis en tiempo real, y herramientas de configuración. [6, 8, 14].

El DSP/BIOS es un conjunto de módulos que se enlazan con una aplicación; esta aplicación incluye las funciones del DSP/BIOS. La configuración de las herramientas del DSP/BIOS permiten optimizar el tamaño del código y la rapidez de ejecución.

El DSP/BIOS se puede usar para instrumentar, probar, trazar y monitorear cualquier aplicación en tiempo real, los programas que usan la configuración del DSP/BIOS tienen la ventaja de instrumentarse implícitamente.

4.3.1. Herramientas para configuración del DSP/BIOS

Esta herramienta permite crear y configurar los objetos usados en los programas. Se puede usar esta herramienta para configurar la memoria, definir prioridades y controlar las interrupciones. Tiene multiples funciones:

- Permite fijar un rango de parámetros usados por las librerías del DSP/BIOS en el tiempo de ejecución.
- Sirve como un editor virtual para crear objetos. Estos objetos incluyen interrupciones de software, tareas, flujo de entradas y salidas, y registros de sucesos.
- Fija parámetros para la librería de soporte del chip (Chip Support Library, CSL) y a de los módulos.

Usando esta herramienta, los objetos del DSP/BIOS pueden preconfigurarse y acoplarse en un programa virtual ejecutable. Alternativamente, un programa del DSP/BIOS puede crear y borrar objetos en el tiempo de ejecución.

4.3.2. Herramientas de análisis en tiempo real del DSP/BIOS

Esta ventana permite observar la actividad de un programa en tiempo real. Por ejemplo, la ejecución de gráficos, muestra un diagrama de la actividad. La herramienta complementa al ambiente del CCS ya que activa el análisis del programa en tiempo real. Puede monitorear una aplicación del DSP.

Esta capacidad de análisis en tiempo real de un programa, permite:

- *Trazo del programa.* Despliega los eventos escritos en el registro, reflejando el control de flujo dinámico durante la ejecución del programa.
- *Monitoreo de la ejecución.* Estadísticas del seguimiento que reflejan el uso de los recursos de la tarjeta, así como la carga del procesador y la temporización.
- *Transmisión del archivo.* Acoplamiento entre los objetos de entrada/salida a los archivos del *host*.

4.3.3. APIs (Application Programming Interface) del DSP/BIOS

Es un conjunto de constantes, tipos, variables y funciones usadas para programación interactiva con software. Los programas escritos en C y en ensamblador pueden hacer uso de unas 150 funciones API del DSP/BIOS. Las APIs del DSP/BIOS se dividen en módulos. Todas las operaciones relacionadas con un módulo comienzan con la letra inicial del módulo. Los programas de aplicación usan a las APIs para llamar al DSP/BIOS.

Todos los módulos tienen interfaz con C y algunos módulos contienen macros en lenguaje ensamblador optimizado. Muchas interfaces de C también se pueden llamar con los programas en lenguaje ensamblador. Usando las APIs, se activa la tarjeta para capturar y cargar la información del *host* a través del CCS [13, 8, 14].

Cuando se abre un archivo de configuración del DSP/BIOS, el CCS muestra un editor visual, que permite crear y establecer propiedades para objetos en tiempo real. Estos objetos son usados en las llamadas a las APIs del DSP/BIOS. También incluyen interrupciones por software, vías de entrada/salida, mensajes de eventos (LOGs), etc. La Figura 4.4 muestra el editor visual DSP/BIOS.

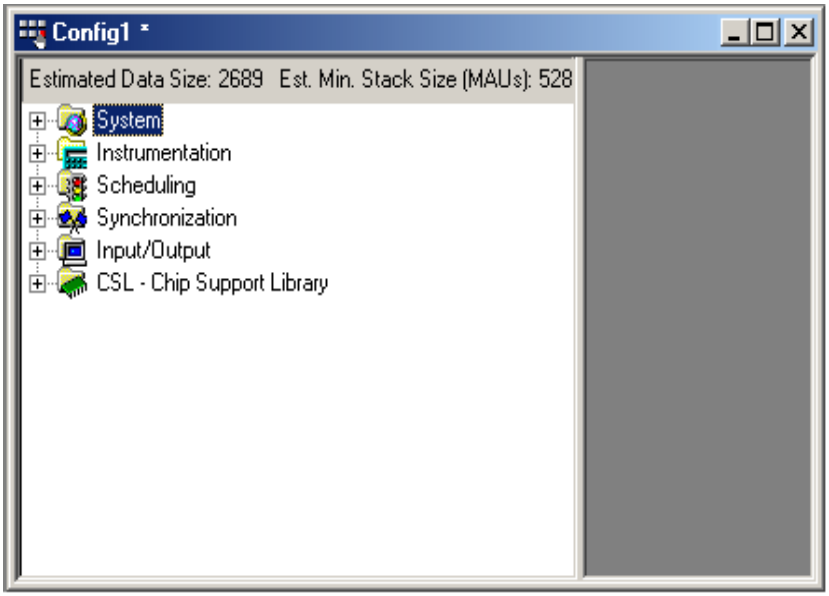


Figura 4.4: Configuración del DSP/BIOS.

Las APIs del DSP/BIOS están dividas en los siguientes módulos:

- ATM: Módulo de funciones atómicas que pueden usarse para manipular datos compartidos.
- C62: Este módulo proporciona funciones específicas del DSP, para manejo de interrupciones.
- CLK: Esté módulo controla el timer interno del DSP y proporciona un reloj lógico de 32 bits en tiempo real con alta resolución.
- DEV: Este módulo permite crear y usar sus propios de dispositivos de control.

- GBL: Este módulo no maneja ningún objeto individual, pero permite un control global o un sistema extenso usado por otros módulos
- HST: El módulo *host*, maneja este tipo de objetos de canal que permiten a una aplicación transmitir flujo de datos entre el DSP y un *host*. Los canales *host* son configurados estáticamente para entrada o salida.
- HWI: El módulo de interrupciones del hardware proporciona soporte para rutinas que atienden las interrupciones
- IDL: Este módulo maneja funciones estáticas (idle), que corren cuando no se ejecuta ninguna función de mayor prioridad.
- LCK: El módulo de asignación maneja recursos globales compartidos y es usado para controlar el acceso a estos recursos, entre varias tareas que los pretenden ocupar.
- LOG: Este módulo maneja objetos tipo LOG, que capturan eventos en tiempo real mientras el programa objeto se ejecuta. Se puede usar logs del sistema o se pueden definir logs propios. Con el CCS se pueden visualizar estos mensajes logs mientras se ejecuta el programa.
- MBX: El módulo de buzón maneja objetos que pasan mensajes de una tarea a otra.
- MEM: El módulo de memoria permite especificar los segmentos de memoria requeridos.
- PIP: Este módulo maneja canalizaciones de datos (pipe), que se usan para el flujo de datos de entrada y salida en los buffers. Estas canalizaciones de datos proporcionan una estructura de datos consistente, para manejar entradas/salidas entre el DSP u otros dispositivos periféricos, en tiempo real.
- PRD: El módulo de manejo de funciones periódicas administra objetos de este tipo. Permite activar ejecuciones cíclicas de una función. La velocidad de ejecución para estos objetos puede ser controlada, por la frecuencia de reloj mantenida por el módulo CLK o por llamadas regulares a la función PRD_tick.
- QUE: Este módulo maneja estructuras de colas de datos.
- RTDX: Permite el intercambio de datos en tiempo real entre la PC y el DSP, además puede analizar y desplegar los datos en la PC usando una automatización cliente OLE (esta puede programarse en Visual C++, Visual Basic, Excel, Matlab, Lab View, etc)
- SEM: El módulo de samáforos permite sincronizar tareas y realizar exclusión mutua.

- SIO: El módulo transmisión maneja objetos que proveen eficacia en tiempo real de dispositivos de entrada/salida.

- STS: Módulo de estadísticas, administra acumulación de estadísticas clave en tiempo real, mientras el programa se ejecuta.

- SWI: Este módulo administra las interrupciones por software. Estas interrupciones son procesos que tienen menor prioridad que las interrupciones por hardware y mayor prioridad que el módulo de tareas. Cuando una función anuncia a un objeto SWI, con una llamada API, el módulo SWI proyecta la función correspondiente para ejecutarla.

- SYS: Módulo de dispositivos del sistema, proporcionan funciones de propósito general.

- TRC: El módulo de trazo, envía mensajes a la ventana de depuración en tiempo real.

- TSK: Módulo de tareas (las tareas son procesos con menor prioridad que las interrupciones por software.)

Cada módulo del DSP/BIOS tiene un único nombre que se usa como prefijo para las operaciones (funciones), archivos cabecera, y para los objetos de los módulos. Este nombre consta de 3 o 4 iniciales mayúsculas. Los identificadores comienzan con letras mayúsculas seguidos de un guión bajo (XXX_*), estos deben tratarse como palabras reservadas.

Cada módulo tiene dos archivos cabecera que contienen las declaraciones de todas las constantes, los tipos y las funciones disponibles en la interfaz de los módulos.

- **xxx.h**. Archivos cabecera API del DSP/BIOS para programas en C. Los archivos fuente de C deben incluir `std.h` y los archivos cabecera para cualquier módulo de las funciones en uso.

- **xxx.h62**. Archivos cabecera API del DSP/BIOS para programas en ensamblador. Los archivos fuente de ensamblador deben incluir el archivo cabecera `xxx.62` para cualquier módulo de ensamblador que se use. Este archivo contiene definiciones macros específicas para este dispositivo.

El programa fuente debe incluir la correspondiente cabecera para cada módulo usado en una aplicación particular. Además los archivos fuente en C deben incluir después de `std.h`, los archivos cabecera en cualquier secuencia. Por ejemplo:

```
#include <std.h>
#include <tsk.h>
#include <sem.h>
#include <prd.h>
#include <swi.h>
```

El DSP/BIOS incluye los módulos que se usan internamente. Los archivos cabecera para estos módulos internos están distribuidos como parte del DSP/BIOS y deben presentarse en el sistema cuando se compilan y enlazan los programas. Usando la configuración del DSP/BIOS se reduce el tamaño del programa.

El formato para el nombre de una operación API del DSP/BIOS es MOD_action donde MOD son las letras del módulo que contiene la operación, y action es la acción ejecutada por la operación. Por ejemplo, la función SWI_post se define por el módulo SWI; este envía una interrupción de software.

La implementación de las APIs, también incluyen algunas funciones predefinidas que se ejecutan por varios objetos. Aquí algunos ejemplos:

CLK_F_isr. Ejecutada por un objeto HWI para proveer la resolución menor de un pulso de CLK.

PRD_F_tick. Ejecutada por el objeto CLK PRD_clock para administrar PRD_SWI y el pulso del sistema.

PRD_F_swi. Accionado por PRD_tick para ejecutar las funciones PRD.

_KNL_RUN. Ejecutada por la prioridad mas baja del objeto SWI, KNL_swi, para ejecutar la tarea del programa si esta activo. Esta es una función C llamada KNL_run. Se usa un guión bajo como prefijo porque la función es llamada del código ensamblador.

_IDL_F_loop. Ejecutada por la prioridad mas baja del objeto TSK, TSK_idle, para correr las funciones IDL.

IDL_F_BUSY. Ejecutada por IDL_cpuLoad del objeto IDL para calcular la carga actual del CPU.

RTA_F_dispatch. Ejecutada por el objeto IDL RTA_dispatcher para recopilar datos en tiempo real.

LNK_F_dataPump. Ejecutada por el objeto IDL LNK_dataPump para manejar la transferencia del análisis en tiempo real y los datos del canal HST al *host*.

HWI_unused. En realidad no es el nombre de una función. Esta cadena se usa en la herramienta de configuración para marcar los objetos HWI sin uso.

Tipo	Descripción
Arg	Tipo capaz de mantener los argumentos Ptr e Int
Bool	Valor booleano
Char	Valor carácter
Fxn	Puntero de una función
Int	Valor entero con signo
LgInt	Valor entero largo con signo
LgUns	Valor entero largo sin signo
Ptr	Valor genérico de puntero
String	Secuencia de caracteres con terminación cero (/0)
Uns	Valor entero sin signo
Void	Tipo vacío

Tabla 4.1: Tipos de datos para las APIs

El código del programa no puede llamar a ninguna función cuyos nombres empiecen con MOD_F_. Estas funciones pretenden ser llamadas únicamente como parámetros especificados con la herramienta de configuración.

Los símbolos que comienzan con MOD_ y MOD_F_ (donde MOD son las iniciales de cualquier módulo del DSP/BIOS) son reservados para uso interno.

Las APIs del DSP/BIOS no usan explícitamente los tipo básicos de C, como int o char. En vez de eso, para asegurar la portabilidad para otros procesadores que soportan las APIs, el DSP/BIOS define sus propios tipos de datos. En muchos casos, los tipos son versiones en mayúsculas de los tipos en C.

Los tipos de datos, mostrados en la Tabla 4.1 se definen en el archivo cabecera std.h. Los tipos de datos adicionales son definidos por std.h, pero no son usados por las APIs del DSP/BIOS. Además, la constante estándar NULL (0) usada por el DSP/BIOS indica un valor de puntero vacío. Las constantes TRUE (1) y FALSE(0) se usan para valores de tipo Booleano.

Las estructuras de un objeto usado por los módulos API del DSP/BIOS usan una sintaxis convencional, como MOD_Obj, donde MOD son las letras del módulo del objeto. Si un programa usa los objetos creados por la herramienta de configuración, se hará una declaración externa del objeto.

La herramienta de configuración automáticamente genera una cabecera C en el archivo que contiene las declaraciones propias de todos los objetos DSP/BIOS creados por esta herramienta (`<program>.cfg.h`).

El archivo (`<program>.cfg.h`), puede incluirse en los archivos fuente de la aplicación para acompañar las declaraciones de los objetos del DSP/BIOS.

La herramienta de configuración es un editor que permite inicializar datos, estructuras y un conjunto de varios parámetros usados por el DSP/BIOS. Cuando se guarda un archivo, la herramienta de configuración crea una fuente en ensamblador, los archivos cabecera y un archivo de comando de enlace para marcar la configuración. Cuando se genera la aplicación, estos archivos se enlazan con los programas de aplicación.

Para aplicaciones típicas del DSP, muchos objetos deben crearse con la herramienta de configuración porque se usan a través de toda la ejecución del programa. Algunos objetos predeterminados se definen automáticamente en la plantilla de configuración.

Ejemplo 4.1. ***Multiplicación de dos números, usando la herramienta de configuración***

En el siguiente ejemplo se usará la herramienta de configuración y se creará un objeto para la visualización de datos y mensajes. El Programa 4.1 muestra el código en el que se utiliza lenguaje C, librerías para los archivos cabecera, tipos definidos para las APIs. Siga los siguientes pasos para realizar la ejecución.

1. Seleccione del menú **Project** → **New**, para que se abra la ventana **Project Creation**. En el campo **Project Name** escriba *Multiplicacion* (u otro nombre). Revise los siguientes campos y verifique que sean correctos. Presione **Finalizar**. (Figura 4.5).

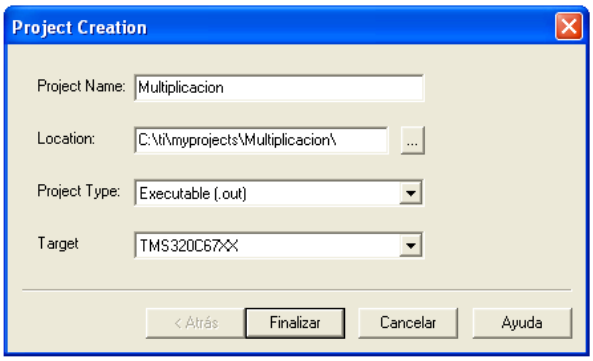


Figura 4.5: Creación de un proyecto nuevo

2. Escriba el código del Programa 4.1, para esto seleccione **File** →**New**)→**Source File**. Guarde este programa con el mismo nombre pero con extensión `*.c`

Programa 4.1: Multiplicacion.c

```
#include <std.h>
#include <log.h>
#include "multiplicacioncfg.h"
#define NUMERX 5
#define NUMERY 10

Void impri(int,int);

Void main() {

    int  num1 = NUMERX;
    int  num2 = NUMERY;
    Char *msg = "Multiplicación\n";

    LOG_printf(&trace, "%s", msg);

    impri(num1, num2);

    return;
}

Void impri(int numr1, int numr2) {

int i;

    for (i = 1; i < numr2+1; i++)
        LOG_printf(&trace,"x %d = %d", numr1 ,i * numr1);
}
```

3. Seleccione en el menú **Project →Add Files to Project**. En la ventana elija el archivo y oprima **Abrir** (Figura 4.6).

4. Escoja del menú **File→New→DSP/BIOS Configuration**.

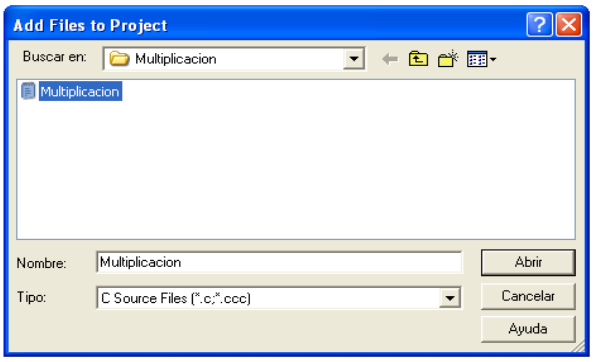


Figura 4.6: Agregando un archivo al Proyecto

5. De la ventana New, elija la plantilla **dsk6711.cdb** (Figura 4.7). Pulse OK.

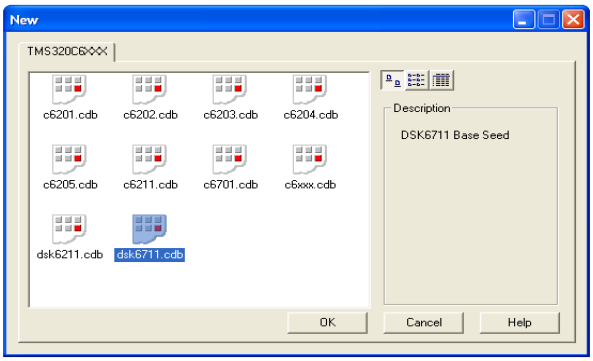


Figura 4.7: Creación de una nueva plantilla

6. La ventana de configuración del DSP/BIOS, contiene los objetos usados en las llamadas a las APIs. Realice la creación de un nuevo objeto LOG. Abra el folder **Instrumentation**. Sobre el administrador de eventos **LOG** seleccione **insert LOG**, renombre este objeto como **trace**. Cambie las características de este objeto, haciendo clic con el botón derecho del ratón y eligiendo **Properties** . Elija el tamaño del buffer, según lo requiera. Guarde el archivo de configuración como *Multiplicacion.cdb* (Figura 4.8).

7. El paso anterior genera otros archivos. Agregue los siguientes archivos al proyecto: (Ver paso 3)

   - Multiplicacion.cdb
   - Multiplicacioncfg.cmd

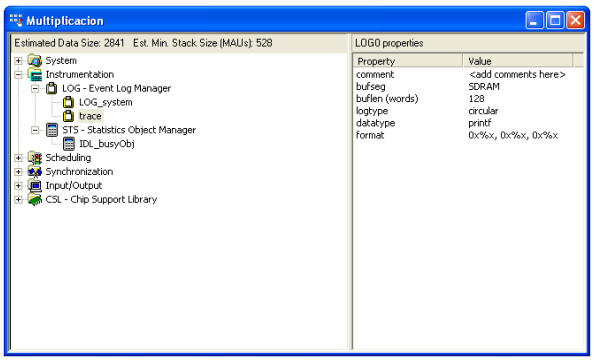


Figura 4.8: Ventana de configuración de objetos del DSP/BIOS

8. Para compilar, ensamblar y enlazar un proyecto de la barra de herramientas elija **Project → Rebuild All**.

   Cuando se finaliza esta rutina satisfactoriamente, el proyecto esta preparado para ejecutarse en el DSK.

9. Para el proceso de carga de un Archivo Ejecutable en el DSK, del menú **File → Load Program**. En la ventana **Load Program** haga clic en la carpeta Debug (Depurador) y pulse el botón **Abrir**. Haga doble clic sobre el nombre del programa con extensión `.out`. Asegure que el cuadro **Tipo** tenga la extensión `.out`.

10. Seleccione del menu **Debug→Go Main**.

11. Elija del menu **DSP/BIOS→ Message Log**.

12. Efectúe la ejecución del programa, haciendo cualquiera de las siguientes acciones:

    - Haga clic sobre el icono Correr (Run) .
    - Presione F5.
    - Del menú **Debug→Run**.

    Observe los resultados de la Figura 4.9

13. Por último, detenga la ejecución, con alguna de las siguientes opciones:

    - Haciendo clic sobre el icono Detener (Halt) .
    - Presionando MAYUSCULAS y F5.
    - Del menú **Debug→Halt**.

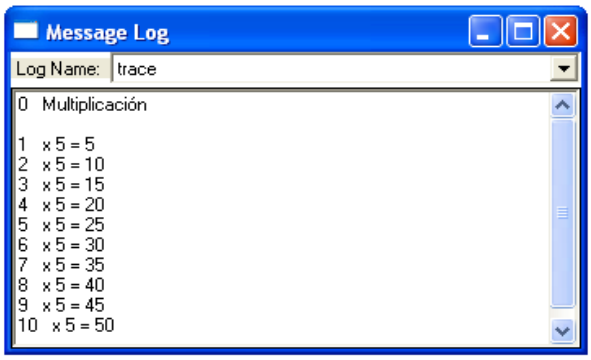


Figura 4.9: Resultados de la ejecución del proyecto

Cuando se especifican las funciones en C para ejecutarse en varios objetos, se aumenta un guión bajo antes del nombre de la función. Por ejemplo, escriba *_myfunc* para correr una función en C llamada *myfunc*. El guión bajo es un prefijo necesario porque la herramienta de configuración crea una fuente en ensamblador, y las llamadas de C requieren de un guión bajo antes de que las funciones en C se llamen del ensamblador.

Algunos objetos pueden únicamente crearse en la herramienta de configuración. Se pueden crear muchos, pero no todos los objetos DSP/BIOS con la llamada a la función XXX_create donde XXX es el nombre de un módulo especifico. Cada función XXX_create localiza la memoria para almacenar la información del estado interno de los objetos, y regresa un punto de arrastre usado para hacer referencia al nuevo objeto creado cuando se llame a otras funciones dadas por el módulo XXX.

Muchas funciones XXX_create aceptan como último de sus parámetros un puntero en la estructura de tipo XXX_Attrs que se usa para asignar los atributos al nuevo objeto creado. por convención, el objeto se asigna a un conjunto de valores predeterminados si este parámetro es NULL. Estos valores predeterminados están contenidos en la estructura de la constante XXX_ATTRS, listada en los archivos cabecera, activada para inicializar primero una variable de tipo XXX_Attrs y después actualizar sus campos con los valores del atributo de la aplicación dependiente antes de la llamada a XXX_create.

Cuando se guarda un archivo de configuración para el programa, se crean los siguientes archivos:

- program.cdb
- programcfg.h62
- programcfg.s62
- programcfg.cmd

- programcfg.h
- programcfg_c.c

La Figura 4.10 muestra los archivos que se usan para crear los programas del DSP/BIOS. Los archivos escritos por el usuario se representan en fondo blanco; los archivos generados se representan en fondo gris. La palabra *program* representa el nombre del proyecto o programa.

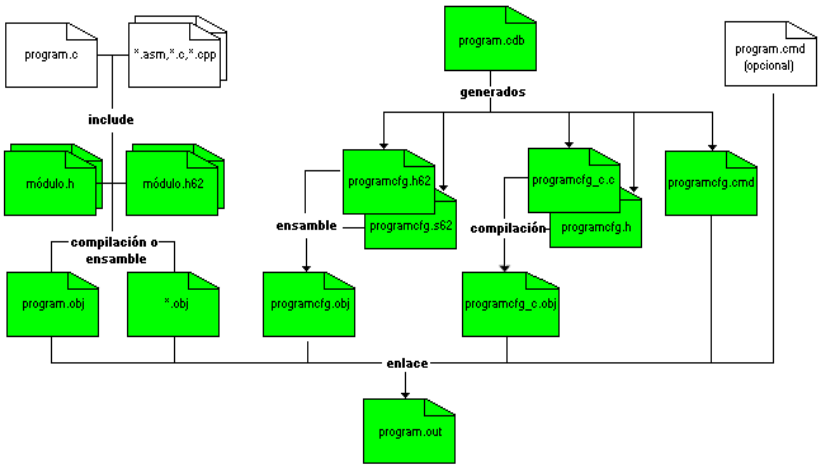


Figura 4.10: Programas generados en la configuración del DSP/BIOS.

- program.c. Archivo de programa fuente que contiene la función *main*. Se pueden tener archivos adicionales y archivos de programas .h.
- *.asm. Archivo de código fuente ensamblador opcional. Uno de estos archivos puede contener una función en lenguaje ensamblador llamada *_main* como alternativa para usar una función principal en C o C++
- módulo.h. Archivo cabecera API para programas en C o C++. Los archivos fuente pueden incluir std.h y los archivos cabecera para cualquier módulo que use el programa.
- módulo.h62. Archivos cabecera API para programas en lenguaje ensamblador. Los archivo fuente en ensamblador pueden incluir el archivo cabecera .h62 para cualquier módulo que use el programa
- program.obj Archivo objeto compilado o ensamblado del archivo fuente.
- *.obj. Archivo objeto para archivo fuente ensamblado.

- program.cdb. Archivo de configuración, que almacena la configuración. Este archivo se crea por la herramienta de configuración, usa la herramienta de configuración y la herramienta de análisis del DSP/BIOS.

- programcfg.h62 Archivo cabecera generado por la herramienta de configuración. Este archivo se incluye por el archivo programcfg.s62.

- programcfg.s62. Código fuente en lenguaje ensamblador generado por la herramienta de configuración.

- programcfg.cmd. Archivo de comando de enlace creado por la herramienta de configuración y usado cuando el archivo ejecutable.

  Este archivo define las opciones de enlace especificas para el DSP/BIOS, los nombres de los objetos, las secciones generales de datos para los programas del DSP (como son .text, .bss, .data, etc).

- programcfg.obj Archivo objeto creado por el archivo fuente generado por la herramienta de configuración.

- *.cmd. Archivo de comando de enlace opcional, que contiene las secciones adicionales para el programa no definido por la herramienta de configuración.

- program.out. Programa ejecutable para la tarjeta (completamente compilado, ensamblado y enlazado). Se puede cargar y ejecutar este programa con los comandos del CCS

- programcfg.h. Archivo que contiene las declaraciones de los objetos creados con la herramienta de configuración. Esta también contiene la definición de la macro CSL_xxxx, macro donde xxxx es el tipo de chip propiedad del módulo de Configuración Global (Global Settings).

- programcf_c.c Archivo que contiene el código para las configuraciones del CSL generadas automáticamente cuando un archivo de configuración se guarda. Este contiene el #include para el archivo programcfg.h.

4.3.4. Secuencia de inicio del DSP/BIOS

Cuando inicia una aplicación, las llamadas o instrucciones en los archivos determinan la secuencia de inicio. Las versiones compiladas de estos archivos se dan con el bios.ann y las librerías biosi.ann.

1. *Inicializa el DSP.*. Un programa de DSP/BIOS empieza en el ambiente C o C++ en el punto c_int00. El vector de interrupción de reinicialización (reset) se configura para saltar a c_int00 después de la reinicialización.

Para la plataforma C6711, el inicio c_int00, el puntero de la pila del sistema (B15) y el puntero de la página global (B14) se configuran para apuntar al final de la sección de la pila y al inicio de la sección .bss respectivamente. El control de los registros como AMR, IER, y CSR también se inicializan.

2. *Inicializa .bss del registro .cinit.* Una vez que la pila se configura, se llama a la rutina de inicialización para generar las variables de los registros .cinit.

3. *Llamada a BIOS_init para inicializar los módulos usados por la aplicación.* BIOS _init ejecuta la inicialización de los módulos básicos. BIOS_init invoca a la macro MOD_init para cada módulo del DSP/BIOS usado por la aplicación. El BIOS_init se genera por la herramienta de configuración y se localiza en el archivo program-cfg.snn.

   - HWI_init, inicia ISTP y el registro selector de interrupción, asigna el bit NMIE en IER, y borra a IFR de la plataforma.

     Cuando se configura una interrupción con la herramienta de configuración, el DSP/BIOS se conecta con el registro correspondiente ISR (Interrupt Service Register) en la localización propia de la tabla de servicio de interrupción. Sin embargo, el DSP/BIOS no activa el bit de la interrupción IER. este debe iniciarse en cualquier parte durante la ejecución de la aplicación.
   - HST_init, inicializa el canal I/O de interfaz del *host*. La especificación de esta rutina depende de la implementación usada por el enlace de la tarjeta al *host*. Por ejemplo, si RTDX se usa, HST_init activa el bit en IER que corresponde a la interrupción de hardware reservada para RTDX.
   - IDL_init, numera el contador de la instrucción del bucle inactivo, si se activa en el administrador de funciones. La instrucción se usa para evaluar la carga al CPU visualizada por la gráfica de carga del CPU.

4. *Proceso de la tabla .pinit.* La tabla .pinit se integra de punteros para inicializar funciones. Para programas de C++, la clase de constructores de objetos globales se ejecutan durante el proceso .pinit.

5. *Llamada a la rutina main del programa.* Después de que todos los módulos completan sus procedimientos de inicialización, se llama a la rutina *main*. Esta rutina puede escribirse en ensamblador, en C, C++ o una combinación. A causa de que el compilador de C añade un guión bajo como prefijo al nombre de la función, esta puede ser una función C o C++ llamada _main o una función en ensamblador llamada main.

   Ya que ninguna interrupción de hardware o software se activa, deben inicializarse los procedimientos para la aplicación (como llamadas a la inicialización de rutinas de hardware) de la rutina main.

La función *main* puede activar los bits de una interrupción individual mascarable, pero no debe llamar a HWI_enable para activar interrupciones globalmente.

6. *Llamada a BIOS*_start *para iniciar el DSP/BIOS.* igual que BIOS_init, BIOS_start también se genera por la herramienta de configuración y se localiza en el archivo programcfg.snn. BIOS_start llama a la rutina *main*. BIOS_start es responsable de activar los módulos del DSP/BIOS y de llamar a la macro MOD_startup para cada módulo de DSP/BIOS.

7. *Rutina estática*. Se puede entrar a la rutina estática de dos maneras. De la primer manera, el administrador de tareas (Task) se activa. El planificador de tareas ejecuta TSK_idle que llama a IDL_loop. De la segunda manera, el administrador de tareas se desactiva y de esta manera la llamada a BIOS_start retorna y llama a IDL_loop.

   Cuando se llama a IDL_loop, la rutina del boot falla dentro del rutina estática del DSP/BIOS. En este punto, las interrupciones del hardware y del software pueden ocurrir y previenen la ejecución inactiva. Ya que la rutina estática maneja la comunicación con el *host*, los datos se transfieren entre el *host* y la tarjeta dentro de esta rutina.

Las funciones definidas por el usuario, llamadas por los objetos del DSP/BIOS (IDL, TSK, SWI, PIP, PRD, y CLK) necesitan seguir las convenciones específicas para asegurarse de que los registros se usan propiamente y que esos valores son preservados a través de las llamadas a las funciones.

En la plataforma del C6711, todas las funciones definidas por el usuario llamadas por los objetos del DSP/BIOS necesitan ajustarse a las convenciones de compilación de C. Esto se aplica para funciones escritas en C y en ensamblador.

El compilador diferencia entre las funciones en C y en ensamblador ya que los nombres de las funciones en C están precedidas por un guión bajo (_), y los nombres de las funciones en ensamblador no.

La rutina *main* en una aplicación DSP/BIOS se usa para propósitos de inicialización, como la configuración de un periférico, o para la activación de una interrupción individual de hardware. Es importante reconocer que *main* no entra en ningún tipo de subproceso del DSP/BIOS (HWI, SWI, TSK, o IDL), y que cuando el programa se ejecuta localiza a *main*, pero la inicialización del DSP/BIOS aún no esta completa, esto es porque la inicialización del DSP/BIOS tiene dos fases: durante BIOS_init que se ejecuta antes de *main*, y durante BIOS_start que se ejecuta después de que el programa retorna a *main*.

Las llamadas API del DSP/BIOS no deben hacerse en la rutina *main*, porque la iniciación de BIOS_start no se ha ejecutado. BIOS_start es responsable de activar las interrupciones globales, configurar e iniciar al temporizador, y de activar los programadores (schedulers) para que los subprocesos del DSP/BIOS puedan empezar la ejecución. Por lo tanto, las llamadas al DSP/BIOS que no son apropiadas para *main* son APIs que requieren interrupciones de hardware y que el temporizador este activo, o APIs que hacen la planeación de llamadas y que pueden ejecutar un bloque. Por ejemplo, las funciones como CLK_gethtime y CLK_getltime no pueden llamarse de *main* porque el timer no esta en ejecución. HWI_disable y HWI_enable no pueden llamarse porque las interrupciones del hardware no se activan globalmente. Potencialmente los bloques de llamadas como SEM_pend o MBX_pend, no pueden ser llamados por *main* porque el planificador no se inicializó. Las llamadas programadas como TSK_disable, TSK_enable, SWI_disable, o SWI_enable tampoco son propias de *main*.

BIOS_init, es responsable de inicializar el módulo MEM. Por lo tanto, es correcto llamar a las funciones de localización de memoria dinámica, desde *main*. No sólo las funciones del módulo MEM (MEM_alloc, MEM_free, etc), también están permitidas, las APIs de creación dinámica y de borrado de objetos del DSP/BIOS, como TSK_create y TSK_delete.

Mientras las llamadas a bloques no están permitidas en *main*, las llamadas programadas que se realizan en los subprocesos, sí. Estas son llamadas como SEM_post o STW_post. Tales llamadas se hacen en *main*, los subprocesos leídos se programan para ejecutarse después de que el programa regrese a *main* y de que BIOS_start finalice la ejecución.

4.4. Interrupciones

Las interrupciones son técnicas que colocan al programa temporalmente en suspenso mientras el DSK ejecuta otro conjunto de instrucciones en respuesta a un suceso [11, 31, 29, 24].

Popularmente, los DSPs trabajan en un ambiente que contienen múltiples eventos asíncronos externos. Estos eventos requieren tareas ejecutadas por el DSP cuando aparecen. Una interrupción es un evento que suspende el proceso actual del CPU para que este pueda atender a la tarea necesaria para completar el proceso. Las interrupciones pueden ser provocadas por los temporizadores, convertidores analogicos/digitales, u otros periféricos.

El servicio de una interrupción implica guardar el contexto del proceso presente, completar la tarea de interrupción, restaurar los registros y el contexto del programa, y

reanudar el proceso original. Existen ocho registros para controlar los servicios de una interrupción.

Una transición apropiada en una terminal (pin) de interrupción define el estado de espera de la interrupción con el registro de la bandera de interrupción (IFR, interrupt flag register). Si la interrupción esta propiamente activada, el CPU comienza el proceso de la interrupción y vuelve a direccionar el flujo del programa para dar servicio a la rutina de interrupción.

El DSK TMS320C6711 tienen tres tipos de interrupciones: reset, la interrupción no mascarable (NMI) e interrupciones de la 4 a la 15. Estas interrupciones corresponden a las señales Reset, NMI e INT4-INT15 respectivamente. Estas señales pueden estar ligadas directamente a los pines del dispositivo, conectando periféricos al chip, o pueden ser desactivadas permanentemente. Las prioridades se muestran en la Tabla 4.2.

Los CPUs de los DSP C6711 tienen 12 interrupciones enmascaradas. Las siguientes condiciones deben cumplirse para procesar una interrupción enmascarada:

- El bit GIE (Activación de interrupción global) del registro CSR (registro de control de estado) debe fijarse a 1
- El bit NMIE (Activación de interrupción no enmascarada) se fija a 1
- Se activa la interrupción fijando el bit IE4-IE15 del registro IER a 1.
- La interrupción ocurre, cuando se asigna 1 a bit correspondiente en el registro IFR.

4.4.1. Confirmación de la interrupción (IACK y INUMx)

Las señales IACK y INUMx alertan al hardware externo del C6711 que ha ocurrido una interrupción y que esta en proceso. La señal IACK indica que el CPU ha comenzado el proceso de una interrupción. Las señales INUMx (INUM3-INUM0) indican el número de la interrupción (posición del bit del registro IFR) que esta en proceso. Por ejemplo

```
INUM3 = 0 (MSB)
INUM2 = 1
INUM1 = 1
INUM0 = 1 (LSB)
```

el valor 0111, indica que INT7 esta en proceso.

Interrupciones		
Prioridad	**Nombre**	**Descripción**
Mayor	Reset	Usada para detener al CPU y regresar a un estado conocido.
	NMI	Se usa generalmente como alerta para el CPU de un problema de hardware como una falla energía.
	INT4	External_Pin_4 (Pin externo 4)
	INT5	External_Pin_5 (Pin externo 5)
	INT6	External_Pin_6 (Pin externo 6)
	INT7	External_Pin_7 (Pin externo 7)
	INT8	EDMA_Controller (Controlador EDMA)
	INT9	MCSP_0_Transmit (Transmisor del MCBSP 0)
	INT10	EMIF_SDRAM_Timer
	INT11	MCSP_0_Receive (Reseptor del MCBSP 0)
	INT12	MCSP_1_Transmit (Transmisor del MCBSP 1)
	INT13	Host_Port_Host_to_DSP
	INT14	CLKF_isr
Mínima	INT15	Timer_1

Tabla 4.2: Interrupciones

4.4.2. Tabla de servicio de interrupción (IST, Interrupt Service Table)

Cuando el CPU empieza el proceso para una interrupción, este acude a la tabla de servicio de interrupciones (IST). Es una tabla de extracción de paquetes que contienen código para el servicio de las interrupciones. El IST consta de 16 paquetes consecutivos. Cada paquete de servicio de interrupción (ISFP) contienen ocho instrucciones. Una rutina de servicio para la interrupción puede caber en un paquete individual. Las direcciones y contenidos del IST se muestran en la Figura 4.11. Ya que cada paquete contiene ocho instrucciones, cada dirección se incrementa 32 Bytes (20h) de un localización a otra.

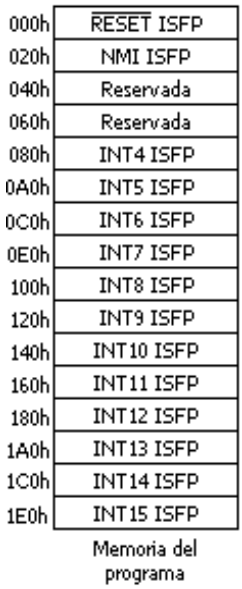


Figura 4.11: Tabla de servicio de interrupciones (IST).

En general, una ISR incluye tres partes. La primera y la última parte guardan y restablecen los registros, respectivamente. La rutina de la interrupción actual hace la segunda parte. Es necesario, guardar el estado del procesador al tiempo cuando se emitió la interrupción, con la finalidad de poder restaurarlo al término de la interrupción.

La interrupción puede activarse o desactivarse por fijación o borrado de los bits apropiados en el registro de activación de interrupción (IER). Hay un interruptor maestro, el bit de interrupción global activa el GIE como parte del registro de control de estado (CSR), el que puede usarse para activar o desactivar todas la interrupciones. El fragmento del Programa 4.2, en código ensamblador, indica como activar y desactivar la interrupción INT9. Aquí se usa la instrucción MVC para transferir un registro de

control a Registro del CPU por manipulación de bit. Otro registro llamado registro de bandera de interrupción (IFR) permite revisar si una interrupción ha ocurrido.

Programa 4.2: Activación de la interrupción INT9.

```
MVK     200h,B1     ; B1 = 200h
MVC     IER,B0      ; B0 = IER
OR      B1,B0,B0    ; B0 = B1 or B0
MVC     B0,IER      ; IER = BO
  :       :
  :       :
  :       :
MVK     FDFFh,B1    ; B1 =FDFFh
MVC     IER,B0      ; B0 = IER
AND     B1,B0,B0    ; B0 = B1 and B0
MVC     B0,IER      ; IER = BO
```

4.5. Subprocesos

Muchas aplicaciones del DSP en tiempo real deben ejecutar un número de funciones aparentemente no relacionadas al mismo tiempo, en respuesta a eventos externos como la disponibilidad de datos o la presencia de una señal de control. Ambas funciones son importantes en la ejecución.

Estas funciones son llamadas subprocesos. Los diferentes sistemas definen los subprocesos por su amplitud. Con el DSP/BIOS, el término se define para incluir cualquier conjunto independiente de instrucciones ejecutadas por el DSP. Un subproceso es un punto de control que contiene una subrutina, un servicio de interrupción (ISR) o una llamada a una función.

El DSP/BIOS activa las aplicaciones para estructurarlas como una colección de subprocesos, cada una de ellas lleva a cabo una función modularizada. Los programas con muchos subprocesos se ejecutan en el procesador tomando en cuenta la prioridad y permitiendo varios tipos de interacción entre ellos, incluyendo el bloqueo, la comunicación, y la sincronización.

En aplicaciones de tiempo real los programas se organizan de modo modular. El DSP/BIOS soporta varios tipos de subprocesos con diferentes prioridades. Cada tipo de subproceso se ejecuta con diferente prioridad.

4.5.1. Tipos de subprocesos

Los cuatro tipos importantes de subprocesos en un programa del DSP/BIOS (de mayor a menor prioridad) son:

- **Interrupciones de hardware (HWI), incluye funciones CLK**. Se activan en respuesta a un evento asíncrono que ocurre en el ambiente del DSP. Una función HWI (también llamada ISR) se ejecuta después de que una interrupción de hardware se activa para llevar a cabo una tarea crítica que esta sujeta a un límite. Las funciones HWI son los subprocesos con la mayor prioridad en una aplicación. Las HWIs deben usarse para aplicaciones que requieren ejecutarse en frecuencias aproximadas de 200 KHz, y que necesitan completarse en el límite de 2 a 100 microsegundos.

- **Interrupciones de software (SWI), incluye funciones PRD**. Se ejecuta después de las interrupciones de hardware (HWI). Mientras las HWI se activan por una interrupción de hardware, las interrupciones de software se activan por una llamada a las funciones SWI del programa. Las interrupciones de software tienen niveles adicionales de prioridad entre las interrupciones de hardware y las TSK. El gestor de subprocesos de SWIs esta sujeto a limitaciones de tiempo que las excluye de ejecutarse como tareas, pero esos límites no son tan severos como los de hardware de las ISRs. Igual que las HWIs, los subprocesos SWIs se ejecutan para concluir. Las interrupciones de software deben usarse para programar eventos con límites de 100 microsegundos o más. Las SWIs permiten que las HWIs pospongan el procesamiento crítico de un subproceso de baja prioridad, minimizando el tiempo en el que el CPU dedica a una rutina de servicio de interrupción, donde otras HWIs pueden desactivarse

- **Tareas (TSK)**, las tareas tienen una prioridad mayor a la de las subtareas y menor prioridad que la de las interrupciones de software. Las tareas se diferencian de las interrupciones de software ya que se pueden suspender durante la ejecución hasta que se disponga de los recursos necesarios. El DSP/BIOS proporciona un número de estructuras usadas en tareas de comunicación y sincronización. Estas estructuras incluyen colas, semáforos, y buzones.

- **Subtareas (IDL**), ejecutan la rutina estática (idle loop, IDL) en la prioridad más baja de una aplicación del DSP/BIOS. Después que regresa a *main*, una aplicación del DSP/BIOS llama a la rutina de inicio de cada módulo y posteriormente entra en la rutina estática. Esta rutina es un bucle continuo que llama a todas las funciones del objeto IDL. Cada función debe esperar que otras finalicen su ejecución, antes de ser llamada otra vez. La rutina estática corre continuamente excepto cuando existen subtareas de mayor prioridad. Únicamente las funciones que no tienen límites establecidos deben ejecutarse en las subtareas.

Hay otras clases de funciones que pueden ejecutarse en un programa del DSP/BIOS. Estas se ejecutan en el contexto de los tipos listados anteriormente.

- Funciones de reloj (CLK). Se activan en el rango de una interrupción temporizada. Por predeterminación, estas funciones se activan por una interrupción de hardware y se ejecutan como funciones HWI.
- Funciones periódicas (PRD). Se ejecutan paralelamente a una interrupción temporizada o en algún otro caso. Las funciones periódicas son un tipo especial de interrupciones de software.
- Funciones de notificación de datos. Se ejecutan cuando se usan canalizaciones (PIP) o canales del *host* (HST) para trasferir datos. Las funciones se activan cuando una trama de datos se lee o se escribe. Estas funciones se ejecutan como parte del contexto de la función con PIP_alloc, PIP_get, PIP_free, o PIP_put.

En la herramienta de configuración del DSP/BIOS, está el administrador de HWI, que contiene objetos HWI para cada interrupción de hardware del DSP.

El ISR de cada interrupción se puede configurar en la herramienta de configuración, únicamente se inserta el nombre del ISR que se llama en respuesta a una interrupción de hardware en la **Página de Propiedades** (Property Page) del objeto correspondiente. El DSP/bios fija la tabla de servicio de interrupción, ya que cada interrupción de hardware se maneja por un ISR propio.

Las interrupciones de software tiene menor prioridad que las interrupciones de hardware. El módulo SWI del DSP/BIOS proporciona interrupciones de software. Éstas se activan con programación, a través de llamadas a las APIs. Las interrupciones de software tienen prioridades mas altas que las tareas (tasks).

Una aplicación puede programar al DSP/BIOS en un ambiente de tiempo real y manejar la transmisión de datos activados por eventos externos. Antes de que la función *main* entre a un ciclo infinito, lo que sucede con los datos de entrada/salida en una aplicación real es parecido a lo que sucede con el resultado de una interrupción periódica externa. El modo mas simple para simular una interrupción periódica externa es usar la interrupción del temporizador del chip.

El módulo SWI del DSP/BIOS da la capacidad de interrupciones de software, éstas se disparan con un programa, a través de las llamadas a las APIs. Los objetos SWI pueden crearse dinámica o estáticamente.

En vez de usar una interrupción de software, una interrupción de hardware puede ejecutar el procesamiento de la señal directamente. Sin embargo, el procesamiento de

la señal puede requerir un gran número de ciclos. Este procesamiento puede impedir la manipulación de la interrupción.

Ejemplo 4.2. ***Tareas (Task)***

Este programa muestra los subprocesos de menor prioridad: las tareas. Se usará la herramienta de configuración para activarlas y para notar como trabajan las prioridades en la ejecución. Cada tarea transfiere su argumento, y ejecuta la misma rutina, esta rutina es la de una multiplicación pero con diferente multiplicador. El multiplicador esta asociado con los argumentos transferidos.

Siga el procedimiento que a continuación se describe:

1. Inicie CCS, si aún no lo ha abierto.
2. Elija del menú **Project →New** . En la ventana **Project Creation** , escriba el nombre del proyecto y finalice.
3. Seleccione **File →New →Source File**. El código se presenta en el Programa 4.3, escribalo y guardelo.
4. Seleccione en el menú **Project →Add Files to Project** y abralo. En la ventana elija el nombre del archivo y oprima **Abrir**.
5. Escoja del menú **File→New→DSP/BIOS Configuration**. De la ventana New, elija la plantilla **dsk6711.cdb**. Pulse OK.
6. Para desplegar la lista de módulos haga clic sobre el signo + que esta al lado derecho de las categorías de programación (Scheduling) y de instrumentación (Instrumentation).
7. Genere cuatro tareas, eligiendo del menu despegable la opción **Insert TSK**, que se extrae oprimiendo el botón derecho del mouse sobre el icono de gestor de tareas. Del mismo menú, pero ahora de la opción **Properties**, asigna los siguientes parámetros:

Nombre	Función	Argumento	Prioridad
task0	_task	6	1
task1	_task	4	1
task2	_task	5	1
task3	_taskita	0	2

Observe que en la parte izquierda de la pantalla se muestran los niveles de prioridades de las tareas. La menor prioridad la tiene el objeto TSK_idle (0), este objeto tiene el propósito de activar la rutina idle cuando ninguna otra subtarea esta ejecutandose.

8. Despliegue la carpeta de **Instrumentation**, sobre el gestor de eventos Log (Event Log Manager) oprima el botón derecho del mouse para que se extienda el menú. Elija de **Insert LOG**, cambie el nombre del LOG a trace; también la longitud de buffer (buflen) a 512 y el tipo de buffer del Log (logtype) a fijo (fixed).

9. Guarde el archivo como tareas.cdb en el directorio. Agregue los siguientes archivos al proyecto:

   - tareas.cdb
   - tareascfg.cmd

Programa 4.3: tareas.c

```
#include <std.h>
#include <log.h>
#include <tsk.h>
#include "tareascfg.h"

#define SIZE 6
#define VALORES_A {2, 6, 4, 8, 10, 12}

Void main() {

return;
}


Void taskita() {
int i;
int a[SIZE] = VALORES_A;

    LOG_printf(&trace, "El arreglo original es:");
    for (i = 0; i < SIZE ; i++) {

          LOG_printf(&trace, "%d", a[i]);

          }

}
```

```
Void task(Arg id_arg) {
Int     id1 = ArgToInt (id_arg);
Int     i;
int     a[SIZE] = VALORES_A;

    LOG_printf(&trace, "Arreglo multiplicado por %d", id1);

    for (i = 0; i < SIZE ; i++) {

        LOG_printf(&trace, "%d",a[i] * id1);

        }


    TSK_yield();

}
```

10. Elija **Project → Rebuild All**.

11. Seleccione del menú **File → Load Program**. En la ventana **Load Program** haga clic en la carpeta Debug (Depurador) y pulse el botón **Abrir**. Haga doble clic sobre el programa `tareas.out`.

12. Escoja del menú **Debug→Go Main**.

13. Elija del menú **DSP/BIOS→ Message Log**.

14. Ejecute el programa, haciendo clic sobre el icono Correr (Run). Observe los resultados como los muestra la Figura 4.12.

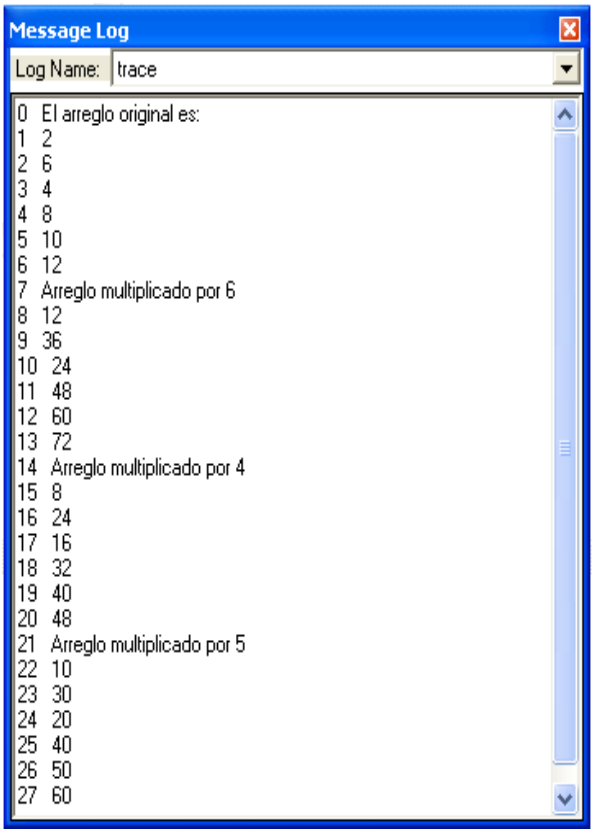


Figura 4.12: Programa tareas.c

Observe los siguientes puntos del Programa 4.3:

El programa incluye los archivos cabecera `std.h`, `log.h` y tsk.h. Esto permite al programa usar los módulos LOG y TSK. También se incluye el archivo cabecera tareascfg.h, que contiene declaraciones externas de los objetos del DSP/BIOS creados en el archivo de configuración.

Cuando concluye la función `main`, se ejecuta la rutina estática (idle) del DSP/BIOS. Con esta rutina, el DSP/BIOS aguarda que los eventos se lleven a cabo.

La función `taskita` tiene la prioridad más alta, e imprime el arreglo que se va a multiplicar. La función `task` ejecuta una rutina que imprime un arreglo multiplicado por un valor pasado a esta función, esta rutina se ejecuta tres veces por los objetos TSK. Note que el argumento de esta función es del tipo `Arg`. Este tipo de datos es capaz de manejar argumentos del tipo `Ptr` y de tipo `Int`. La función `ArgToInt` convierte el tipo de dato `Arg` a un entero.

Antes de finalizar la función task, encontramos la API `TSK_yield`, que permite a otra tarea de igual prioridad ejecutarse.

Las tareas se crean con la herramienta de configuración o con las APIs . Las tareas de igual prioridad, programan su ejecución de acuerdo a la lista que se presenta en la ventana de la herramienta de configuración (Figura 4.13). Seleccionando cualquier tarea, se puede observar su prioridad en la lista propiedades.

Si se requiere cambiar la prioridad de una tarea, arrastre el icono de la tarea al folder con la prioridad que desee y déjelo ahí o en la ventana de prioridades, desplegando el menú. Existen 16 niveles de prioridades, el nivel mas alto es 15 y el menor es 0, este nivel esta reservado para la tarea `idle`.

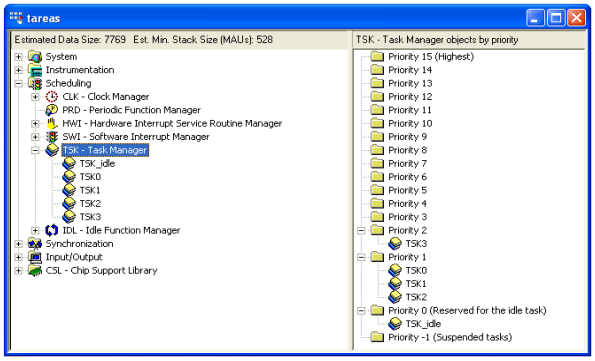


Figura 4.13: Prioridad de las tareas

Ejemplo 4.3. ***Interrupciones por software***

Otras particularidades las podemos observar en el siguiente ejemplo, que ilustra la manera de ejecutar interrupciones por software. El módulo SWI del DSP/BIOS tiene la capacidad de ejecutar estas interrupciones, a través de las APIs. Se vuelve a utilizar un algoritmo para matrices, pero esta rutina esta controlada por medio de una interrupción de software.

Para poner en práctica este ejemplo realice las siguientes acciones:

1. Inicie CCS.
2. Cree un nuevo proyecto.
3. Copie el Programa 4.4 en un archivo fuente nuevo, guarde y agregue este al proyecto creado.
4. Seleccione la plantilla para usar las herramienta de configuración del dsk C6711, **File→New→DSP/BIOS Configuration**.
5. Oprima el signo que esta a un lado del objeto **Instrumentación** (Instrumentation), para crear un objeto LOG, al que llamará `trace`. Cambie las propiedades:

Campo	Valor
buflen	512
logtype	circular

6. Expanda el objeto **Programación** (Scheduling), y cree un objeto **SWI**, llamado `matrica_SWI`. Introduzca las siguientes propiedades:

Campo	Valor
function	_matrica
priority	1
mailbox	10
arg0	_x
arg1	_y

7. Extienda el menú del objeto CLK, inserte un nuevo objeto CLK, renombrelo `interrupcion_CLK`. Abra sus propiedades y en el campo de la función (function), introduzca `_interrupcion`.
8. Guarde los cambios del archivo de configuración y cierre este archivo.
9. Agregue los archivos al `softwareInterrupt.cdb` y `softwareInterruptcfg.cmd` al proyecto.

10. Pulse el icono para construir el proyecto, ignorando dependencias,  .
11. Cargue el archivo `softwareInterrupt.out`, para ejecutarlo.
12. Avance hasta la línea donde se encuentra la instrucción main(), con **Go main**
13. Abra la ventana para imprimir los objetos Log (Message Log)
14. Ejecute el programa.
15. Compruebe los resultados del programa como en la Figura 4.14

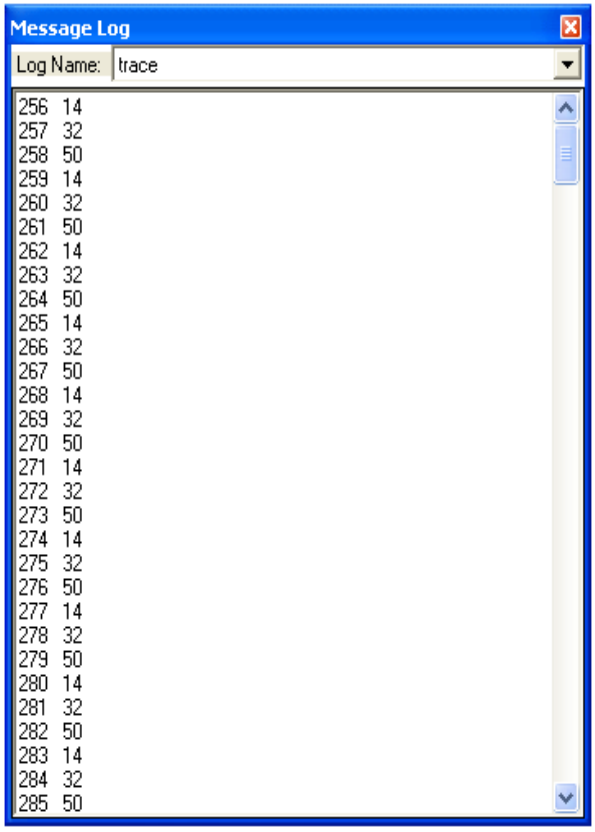


Figura 4.14: Programa softwareInterrupt.c

Programa 4.4: softwareInterrupt.c

```
#include <std.h>
#include <log.h>
#include <swi.h>
#include <clk.h>
#include <sts.h>
#include <trc.h>
#include <rtdx.h>
#include "softwareInterruptcfg.h"

#define SIZE1 9
#define SIZE2 3
#define COLU1 3
#define FILA1 3
#define COLU2 1
#define VALORES_X {1, 2, 3, 4, 5, 6, 7, 8, 9}
#define VALORES_Y {1, 2, 3}

    int x[SIZE1] = VALORES_X;
    int y[SIZE2] = VALORES_Y;
    int r[SIZE2] = {0,0,0};


Void main() {
    LOG_printf(&trace,"Inicia prueba\n");
    return;
}

Void matrica (Int *x,Int *y){
int c1 = COLU1;
int r1 = FILA1;
int c2 = COLU2; int i, j, k, suma;

for (i = 0; i < r1; i++)
    for(j =0; j < c2; j++)
    {
        suma = 0;
        for (k = 0; k < c1; k++)

          suma += x[k + i*c1]*y[j + k*c2];
          r[j+1*c2]= suma;
```

```
            LOG_printf(&trace, "%d", r[j+1*c2]);
        }

}

Void interrupcion() {
    SWI_dec(&matrica_SWI);
}
```

Ejemplo 4.4. ***Interrupciones***

Aquí se simula una aplicación para el DSP, que digitaliza una señal, ajusta este volumen, y produce una salida análoga con el ajuste del volumen. Se usa también las herramienta de análisis y herramientas de configuración.

Este programa simula una entrada de datos, sobre los cuales se aplicará un incremento a su valor, cuando la aplicación llamé a una de las APIs, el gestor SWI programará la función correspondiente a la interrupción para ejecutarla [2].

Para ejecutar este proyecto realice lo siguiente:

1. Seleccione el proyecto volume2.pjt de `c:\ti\tutorial\dsk6711\volume2`.
2. Copie este archivo a `c:\ti\mypojects`.
3. En CCS abra el proyecto volume2.pjt y visualice el archivo `volume.c` como se muestra en el Programa 4.5

Programa 4.5: volume.c

```
#include <std.h>
#include <log.h>
#include <swi.h>
#include <clk.h>
#include <sts.h>
#include <trc.h>
#include <rtdx.h>
#include "volumecfg.h"
#include "volume.h"

Int inp_buffer[BUFSIZE];          /* buffers de datos */
Int out_buffer[BUFSIZE];
Int gain = MINGAIN;               /* variable del volumen*/
Uns processingLoad = BASELOAD;    /* valor de carga  */

RTDX_CreateInputChannel(control_channel);

extern Void load(Uns loadValue);

Int processing(Int *input, Int *output);
Void dataIO(Void);
Void loadchange(Void);
```

```
Void main() {
    LOG_printf(&trace,"Inicia el programa de volumen\n");

    RTDX_enableInput(&control_channel);

    return;
}

/* ======== processing ========
 * FUNCIÓN: Llamada del objeto processing_SWI para aplicar una
 *          transformación a la señal de entrada.
 */
Int processing(Int *input, Int *output) {
    Int size = BUFSIZE;

    while(size--){
        *output++ = *input++ * gain;
     }

    /* Acciona la ejecución de la instrumentación
       solo si TRC_USER0 esta activo */
    if (TRC_query(TRC_USER0) == 0) {
        STS_set(&processingLoad_STS, CLK_gethtime());
    }
    /* Procesamiento adicional de load */
    load(processingLoad);
    if (TRC_query(TRC_USER0) == 0) {
        STS_delta(&processingLoad_STS, CLK_gethtime());
    }

    return(TRUE);
}

/*  ======== dataIO ========
 *
 * FUNCIÓN: Llamada del timer ISR para simular una interrupción
 *          periódica de hardware que lee la señal de entrada
 *          y conserva la señal procesada.
 */
Void dataIO() {
    SWI_dec(&processing_SWI);    /* llama a interrupción */
}
```

```
/*  ========= loadchange =========
 *
 * FUNCIÓN: Llamada de loadchange_PRD para actualizar
 *          periódicamente el valor de load.
 */
Void loadchange() {
    static Int control = MINCONTROL;

    /* Lee el nuevo control de load cuando el host lo envía */
    if (!RTDX_channelBusy(&control_channel)) {
    RTDX_readNB(&control_channel, &control, sizeof(control));
    if ((control < MINCONTROL) || (control > MAXCONTROL)) {
    LOG_printf(&trace,"Valor de control fuera del rango");
    }
    else {
    processingLoad = control;
    LOG_printf(&trace,"Valor de carga = %d",processingLoad);
        }
    }
}
```

El código incluye la referencia a los archivos cabecera del DSP/BIOS: `std.h`, `log.h`, `swi.h`, `clk.h`, `sts.h`, `trc.h`, `rtdx.h`, `volumecfg.h` y `volume.h`. El archivo `volumecfg.h`, se genera cuando el archivo de configuración se guarda.

La función *main* regresa después de que `LOG_printf` exhibe su mensaje. Cuando se reanuda la función *main*, el programa entra en la rutina estática del DSP/BIOS. En este punto, el programador del DSP/BIOS maneja la ejecución de las subtareas.

La función `processing` se invoca por una interrupción de software nombrada `processing_SWI`, que produce las interrupciones del hardware. El DSP/BIOS proporciona algunos tipos de subtareas, incluyendo interrupciones de hardware, interrupciones de software, tareas y subtareas estáticas.

La función `dataIO` llama a `SWI_dec`, que decrementa un contador asociado al objeto de la interrupción del software. Cuando el contador llega a 0, la interrupción del software prepara su función para ejecutarla y reinicia el contador.

La función `dataIO` simula los datos de Entrada/Salida del hardware. Un programa típico acumula los datos en un buffer hasta que tenga los suficientes datos para procesarlos. En este ejemplo, la función `dataIO` se ejecuta 10 veces por cada vez que la

función `processing` se ejecuta. El contador decrementado por `SWI_dec` controla esto.

Continue con los siguientes pasos:

1. Abra el archivo de configuración con doble clic sobre el archivo volume.cdb, examine los objetos que se han agregado a la configuración.

2. Despliegue la lista de módulos, observe los administradores CLK, LOG, y SWI. Esta configuración contiene los objetos de esos módulos, los objetos muestran sus propiedades en la parte derecha de la pantalla y para poder configurarlas se ingresa a la ventana de Propiedades (Properties), cambie las siguientes propiedades para ejecutar el programa:

   En el administrador LOG seleccione el objeto `LOG_system`, haga clic derecho y cambie el campo `buflen` a 256 palabras (words).

   Seleccione el objeto CLK nombrado `dataIO_CLK`. Note que la función que llama cuando se activa el objeto es `_dataIO`, que es la función `dataIO` en `volume.c`.

   Abra las propiedades del administrador del reloj CLK. Observe que el campo Microseconds/Int es de 1000. Esto significa que las funciones de reloj, incluyendo a `dataIO_CLK`, corren una vez en un milisegundo.

   Advierta que la propiedad de interrupción del CPU, se muestra en gris. Esto se debe a que esta propiedad actualmente esta configurada en el dialogo de propiedades del objeto HWI correspondiente.

   Expanda la lista de objetos HWI y seleccione el objeto listado en el campo de Interrupciones de CPU de las propiedades del administrador CLK.

   Este objeto ejecuta una función llamada `CLK_F_isr` cuando el temporizador del chip provoca una interrupción. Las funciones del objeto CLK corren en el contexto de la función `CLK_F_isr`. Por lo tanto, corren completamente sin tener una prioridad mayor a las interrupciones de software. (`CLK_F_isr` guarda el contexto del registro, así las funciones CLK no necesitan guardar y restablecer el contexto que se requiere para una función de interrupción de hardware.

   Extienda la lista de los objetos SWI y abra el cuadro de dialogo de propiedades del objeto `processing_SWI`.

   - **función** (function). Cuando esta interrupción de software esta activada, la función processing se ejecuta

- **buzón** (mailbox). Este valor puede controlar la ejecución de interrupción de software. Alguna API afectan el valor del buzón y pueden parar la interrupción de software dependiendo del valor generado. Una función de software se ejecuta cuando tiene la mayor prioridad
- **arg0, arg1**. Las direcciones de `inp_buffer` y `out_buffer` se pasan a la función `processing`.

3. Guarde los cambios realizados en `volume.cdb`, reconstruya el proyecto, cargue y abra el programa `volume.out`. Ejecute el programa a la función *main()*.

4. Escoja DSP/BIOS → Panel de Control RTA . En esta ventana se presenta un tablero de verificación para activar o desactivar los tipos de instrumentación. Por predeterminación, todos los tipos están activos.

5. Abra las ventanas de: Registro de Mensajes (Log Messages, Figura 4.15) y de Ejecución Gráfica (Execution Graph, Figura 4.16).

6. Ejecute el programa y observe las siguientes ventanas:

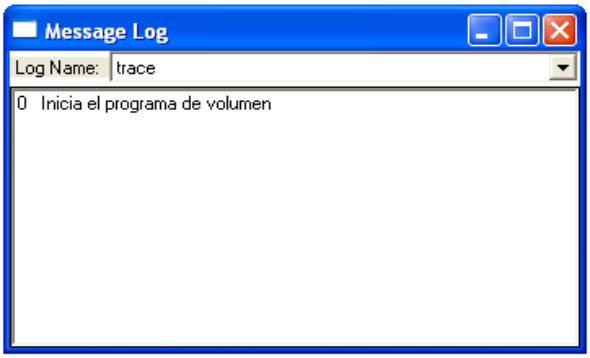


Figura 4.15: Mensaje de inicio.

Las marcas en la línea de tiempo muestra cada vez que el administrador de Reloj (Clock) realiza las funciones CLK. Enumere las marcas de tiempo en que el objeto `processing_SWI` se realiza. Suelen ser 10 marcas. Esto indica que el objeto `processing_SWI` ejecuta su función la décima vez que el objeto `dataIO_CLK` realiza su función. Esto era lo esperado ya que el valor del buzón decrementado por la función `dataIO` comienza en 10.

Usando las ejecuciones gráficas, vemos que el programa encuentra sus límites. Sin embargo, las funciones del procesamiento de la señal en un programa pueden ejecutar un trabajo mas complejo y consumir un número mayor de ciclos.

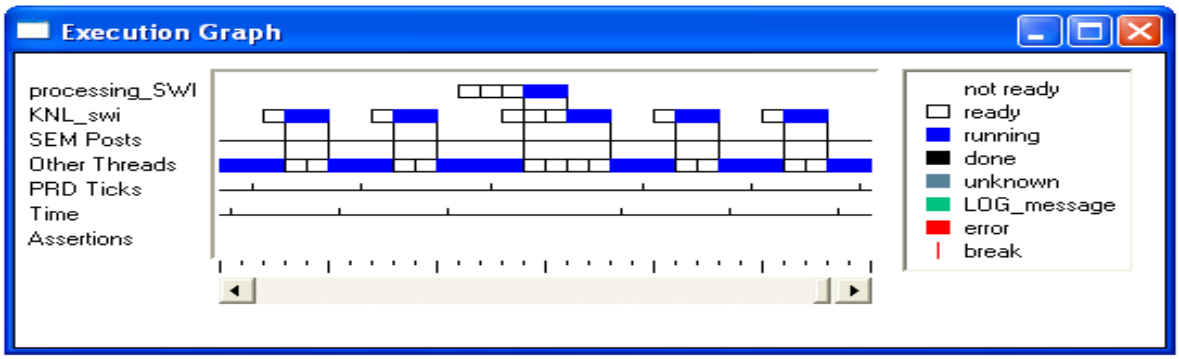


Figura 4.16: Ventana de gráficos.

La carga del CPU se define como el porcentaje de ciclos que el CPU trabaja en la aplicación, es decir, el porcentaje del tiempo total en el cual el CPU esta:

- efectuando ISRs, interrupciones de software, o funciones periódicas.
- ejecutando I/O del host.
- realizando cualquier otra rutina.

La Figura 4.17 representa este proceso.

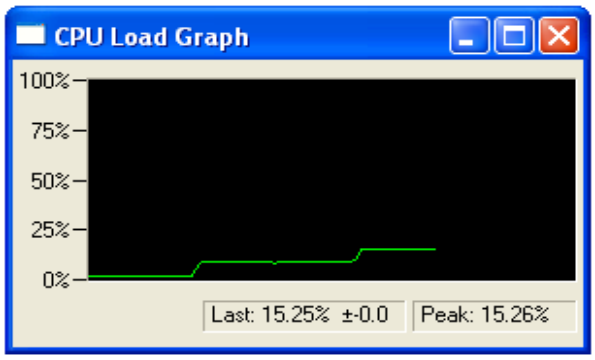


Figura 4.17: Gráfica de carga al CPU.

Capítulo 5

Intercambio de datos en tiempo real

- RTDX
- Interfaz JTAG
- Flujo de datos de la tarjeta al host
- Flujo de datos del host a la tarjeta

En este capítulo se presentan los recursos involucrados en la transferencia de datos desde y hacia la tarjeta del DSK.

5.1. Introducción

La principal razón de convertir una señal analógica a digital es la flexibilidad para programarse. El hardware del DSP puede usarse para diferentes aplicaciones, cambiando el código. Otra razón es que los circuitos proporcionan una salida mas estable comparada con la de los circuitos analógicos [20].

La ventaja de operar en el modo digital, es característico. Por ejemplo, un filtro de fase lineal puede solo diseñarse usando técnicas de procesamiento digital de señales, y muchos sistemas adaptables se consiguen únicamente por manipulación digital. La representación digital permite tratar datos de voz, sonido y video para propósitos de transmisión y almacenamiento.

El procesamiento digital de una señal puede ser implementado en varias plataformas, como un DSP, un circuito VSLI (Very Large Scale Integrated), o un microprocesador de propósito general. El éxito de los DSP parte del costo efectivo de los sistemas en tiempo real, por ejemplo teléfonos celulares, modems, y controladores de discos. Existen dos aspectos del procesamiento en tiempo real [29]:

a) rango de muestreo de entrada/salida

Aplicación	Rango de muestreo I/O	Latencia
Instrumentación	1 Hz	*sistema dependiente
Control	>0.1 KHz	*sistema dependiente
Voz	8 KHz	<50 ms
Sonido	44.1 KHz	*<50 ms
Video	1-14 MHz	*<50 ms

Tabla 5.1: Rangos de muestreo y latencias para aplicaciones selectas.

b) latencia del sistema (retardos)

Los rangos típicos de muestreo y latencias para diferentes aplicaciones aparecen en la Tabla 5.1.

Un sistema de tiempo real es aquel en el que para que las operaciones computacionales estén correctas no depende solo de que la lógica e implementación de los programas computacionales sean correctos, sino también en el tiempo en que dicha operación entregó su resultado. Un sistema en tiempo real debe responder a estímulos externos con unos plazos de respuesta finitos y acotados. Si las restricciones de tiempo no son respetadas el sistema se dice que ha fallado. Por lo tanto, es esencial que las restricciones de tiempo en los sistemas se cumplan. El garantizar el comportamiento en el tiempo requerido necesita que el sistema sea predecible. Es también deseable que el sistema obtenga un alto grado de utilización a la vez que cumpla con los requerimientos de tiempo.

5.2. RTDX

RTDX permite la ejecución en tiempo real y la continua visibilidad en las aplicaciones. RTDX transfiere datos entre un host y la tarjeta sin interferencia de la aplicación. En la plataforma del host, opera una librería RTDX junto con el CCS. Los datos pueden analizarse y visualizarse en el host usando la interfaz COM, proporcionada por RTDX. Los clientes como Visual Basic, Visual C++, Excel, LabView, Matlab y otros, utilizan esta interfaz. Estos permiten una representación real del modo que opera el sistema [2, 13, 16].

RTDX forma un vínculo de dos canalizaciones entre la aplicación y el cliente host. Estas canalizaciones de datos consisten en una combinación de componentes de hardware y software como se muestran en la Figura 5.1.

Los datos pueden enviarse de la aplicación al cliente host y viceversa. La canalización puede verse como un conjunto de una o más canalizaciones virtuales (canales) a través

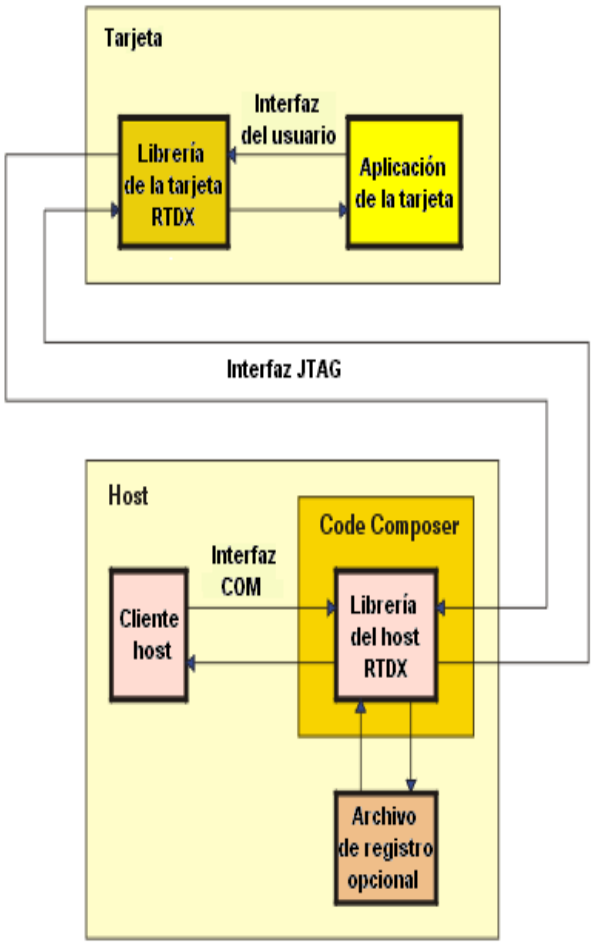


Figura 5.1: Diagrama de la interfaz JTAG.

de las que viajan datos. Esto permite etiquetar a los datos para que pertenezcan a una canalización virtual particular así que varios datos pueden ser diferenciados.

Estas canalizaciones virtuales son unidireccionales. Los datos pueden introducirse de forma asíncrona a una canalización virtual.

La aplicación envía datos a el host por medio de funciones de la librería RTDX. Estas funciones inmediatamente transmiten los datos y después regresan. La librería RTDX transmite el dato enviado a el host de modo que no interfiera con la aplicación de la tarjeta. El host registra los datos en un buffer de memoria o en un archivo de registros RTDX, dependiendo del modo especificado en el host RTDX. El dato registrado puede extraerse por cualquier aplicación del host que es un cliente de la interfaz del host RTDX. En computadoras que usan Windows, la interfaz del host RTDX es una interfaz COM.

De manera similar, un cliente host puede enviar datos a la tarjeta, con la librería del host RTDX. Después esta librería solicita los datos para la aplicación y existen suficientes datos en el buffer del host para satisfacer la solicitud los datos en el buffer del host se envían a la tarjeta. Los datos se escriben en las localizaciones solicitadas por la tarjeta sin la interferencia de la aplicación. El host notifica a la librería de la tarjeta RTDX cuando la operación se completa.

RTDX se compone de la tarjeta y de los componentes del host la librería RTDX se ejecuta en la aplicación. La aplicación hace las llamadas a las librerías API parar transmitir los datos o recibirlos. Esta librería usa un emulador de exploración para mover los datos a través de una interfaz JTAG. La transferencia de los datos ocurre en tiempo real mientras la aplicación esta en proceso de ejecución.

5.3. Interfaz JTAG (Joint Test Action Group)

El DSK C6711 proporciona una emulación JTAG, con la que se tiene acceso al puerto paralelo, también como soporte a un emulador externo XDS510. El modo de emulación se selecciona por la presencia o ausencia del cable conectado a la cabecera JTAG (J5). La emulación externa existe si un cable es detectado o emulación interna si el cable no esta conectado. El controlador de prueba del bus TI SN74ACT8990 permite controlar el software del host y también permite usar al CCS con el DSK sin un emulador externo.

CCS controla el flujo de datos entre los el host (PC) y la tarjeta (procesador TI).

5.4. Requerimientos en la aplicación para transmisión y recepción de datos

Una aplicación para transferencia o envío de datos debe incluir las siguientes directivas:

Archivo cabecera.
`#include <rtdx.h>`; los prototipos para la interfaz RTDX se definen en este archivo, y debe incluirse en la aplicación.

Declaración de macros.
`RTDX_CreateOutputChannel(canal)`/`RTDX_CreateInputChannel(canal)`; declaraciones de una canal de salida o de entrada, para enviar o recibir datos. Los canales RTDX son estructuras de datos en la memoria de la tarjeta que deben declararse como objetos globales. El nombre es arbitrario y depende de las necesidades que se requiera.

Estas macros declaran e inicializan en cero los canales, un canal puede usarse para adquirir o enviar datos pero no para ambas actividades. El contenido de los canales es desconocido para el usuario. Un canal tiene dos estados: activo o desactivo. Los canales se pueden cambiar de estado a través de CCS o de una interfaz COM.

Funciones.

`RTDX_channelBusy`; se usa junto con la función `RTDX_readNB`. Su valor indica si el canal esta actualmente en uso. Si un canal esta ocupado en lectura, el bit de la bandera de prueba/control (test/control) TC, del registro de estado 0 (STO) es 1. En caso contrario el bit TC es 0.

`RTDX_disableInput`, `RTDX_disableOutput`; la llamada a estas funciones provoca que el canal de se desactive.

`RTDX_enableOutput`, `RTDX_enableInput`; estas funciones activan un canal, para que ocurra la transferencia de datos. Los canales se activan llamando a una función de la tarjeta RTDX o a través del depurador.

`RTDX_read`; provoca una solicitud de lectura para enviarla al canal especificado. Si el canal esta activo, la función espera hasta que el dato llegue. Al regreso de la función, los datos han sido copiados en el buffer específico y el número de unidades de direcciones del dato actual es devuelto. La función regresa `RTDX_READ_ERROR` si el canal esta actualmente ocupado o si no esta activo.

Cuando se usa `RTDX_read`, la aplicación notifica a la librería RTDX Host que esta lista para recibir datos y después espera que esta librería escriba los datos al buffer de la tarjeta. Cuando los datos son recibidos, la aplicación continua con la ejecución. Los datos especificados se escriben al canal de salida, dado que este canal esta activo.

`RTDX_readNB`; es la forma de no bloquear la función `RTDX_read`. `RTDX_readNB` envía una lectura solicitada a un canal. Si el canal no esta activo o el canal esta ocupado, la función envía `RTDX_READ_ERROR`. La función devuelve 0 si no envía una lectura por falta de espacios en el buffer.

`RTDX_sizeofInput`; esta diseñada para usarse con `RTDX_readNB` después de una lectura este completa. La función regresa el número de unidades leídas del canal.

`RTDX_write`; Procesa los datos que se escriben en un canal de salida, si el canal esta activo. Al regreso de la función, los datos han sido copiados en el buffer de la tarjeta RTDX.

Otras macros.

`RTDX_isInputEnabled`, `RTDX_isInputEnabled`; examinan si un canal esta activo.

5.5. Flujo de datos de la tarjeta al host

Para registrar datos en la tarjeta, se debe declarar un canal de salida y escribir los datos para usarlos en rutinas definidas en la interfaz del usuario. Los datos son inmediatamente registrados en el buffer RTDX de la tarjeta definido por la librería RTDX. Los datos en el buffer se envían por medio de la interfaz JTAG. La librería RTDX del host recibe los datos de la interfaz JTAG y los registra. El host registra los datos en un buffer de memoria o en un archivo de registros RTDX (dependiendo del modo especificado de registro por el host). La transferencia de datos de la tarjeta a la librería RTDX del host se realiza en tiempo real.

Ejemplo 5.1. ***Transferencia de datos***

El Programa 5.1 ilustra el código para enviar datos de la tarjeta del C6711 a Matlab, este programa se enlaza con otros programas de extensión asm, cmd y algunas librerías, para su correcta ejecución. Además, se ejecuta otro programa en Matlab para recepción de datos.

Para abrir el proyecto, en CCS busque el directorio:

`ti/tutorial/dsk6711/sect_1/less_1`.

1. Del menú `Project`, seleccione `Build` para generar el ejecutable, `s1l1.out`.
2. Seleccione Load Program, del menu File, busque `s1l1.out`, y elija abrir.
3. En CCS, seleccione **Tools→RTDX→Configuration Control**. Esto abre la ventana para la configuración de RTDX. Active RTDX.
4. Ejecute el programa, cuando observe el mensaje de: Programa Completo!, inicie Matlab versión 7.0.

CCS permiten usar las funciones de Matlab para comunicarse con la información almacenada en memoria y los registros de la tarjeta. Con estos enlaces se puede transferir información de o hacia CCS y a los objetos internos.

El modo de Depuración incluye las operaciones que CCS maneja y que enlaza a CCS con Matlab. El enlace de CCS proporciona cuatro componentes que trabajan usando CCS IDE y TI Real-Time Data (RTDX).

Se pueden ejecutar las aplicaciones desde la ventana de comandos de Matlab, enviar y recibir datos de la memoria de la tarjeta, revisar el estado del procesador, así como otras funciones, tales como iniciar y detener la ejecución de el procesador digital de señales.

La interfaz de intercambio de los datos en tiempo real, proporciona una vía de comunicación entre Matlab y los procesadores digitales de señales instalados en la computadora. Usando objetos se pueden abrir los canales de los procesadores en la tarjeta y enviar y transmitir datos.

Programa 5.1: s1l1.c

```
#include <rtdx.h>
#include "target.h"
#include <stdio.h>
#define VALUE_TO_SEND {1,2,3,15,69,70}

/* Declaración de un canal de salida */
RTDX_CreateOutputChannel( ochan );

void main() {
        int data[] = VALUE_TO_SEND;
        int status;
        TARGET_INITIALIZE();
        /* Activando el canal de salida */
        RTDX_enableOutput( &ochan );
        /* Enviando los datos al  host */
        status = RTDX_write( &ochan, &data, sizeof(data) );

        if ( status == 0 ) {
                puts( "ERROR: RTDX_write fallo!\n" );
                exit( -1 );
        }

        while ( RTDX_writing != NULL ) {
#if RTDX_POLLING_IMPLEMENTATION
                RTDX_Poll();
#endif
        }
       /* Desactivando el canal de salida */
        RTDX_disableOutput( &ochan );

        puts( "Programa Completo!\n" );
}
```

Con el uso de los objetos en el enlace con CCS, se pueden abrir los canales para que el procesador de la tarjeta pueda enviar y extraer datos del procesador y ejecutarlos en la aplicación o viceversa.

El enlace de CCS y el enlace para RTDX facilitan el trabajo de un modelo real al mundo real de la tarjeta en el que el algoritmo se puede ejecutar. RTDX y los vínculos de CCS proporcionan el direccionamiento para manipular datos y procesar los programas en la tarjeta. RTDX ofrece intercambio en tiempo real en dos direcciones entre Matlab y la tarjeta de procesamiento.

Para introducir las técnicas y herramientas en los vínculos con CCS, se debe configura la tarjeta, abrir y activar los canales, enviar los datos y borrar todo antes de que finalice la prueba.

La Tabla 5.2 muestra los comandos y funciones para comunicación con el CCS que usa Matlab.

Configuración de los canales de comunicación

Para Matlab, los canales de comunicación no existen hasta que estos se abren y se activan a través del enlace de CCS. La apertura de canales consiste en crear y configurar cada canal para lectura o escritura.

Con la función `open`, se crea un canal, cuyo nombre será la cadena especificada en los parámetros de la función. La cadena debe indicar el nombre de un canal definido en el archivo en código C, en el que se definió el canal.

Continúe con el Ejemplo 5.1, ejecutando en Matlab linea por linea el Programa 5.2 o guardando este en un archivo y en matlab ejecutarlo con su nombre.

Programa 5.2: dC6711aMatlab.m

```
cc = ccsdsp;
rx = cc.rtdx;
open(rx, 'ochan','r');
enable(rx,'ochan');
pause(5);
outdata = readmsg(rx,'ochan','int32')
```

Instrucción	Descripción
ccsdsp	Crea enlaces con CCS y RTDX.
cd	Cambia el directorio de trabajo de CCS de Matlab.
run	Ejecuta el proceso en la tarjeta
close	Cierra los enlaces RTDX entre Matlab y la tarjeta.
configure	Determina cuantos canales se usan y fija el tamaño de cada buffer.
disable	Desactiva los enlaces RTDX antes de cerrarlos.
display	Despliega los resultados de las funciones get y set.
enable	Activa los canales abiertos de esta forma se pueden usar para enviar o transmitir datos a la tarjeta.
isenabled	Determina cuales canales están disponibles para la comunicación RTDX.
isreadable	Determina si Matlab puede leer un bloque de memoria.
iswritable	Determina si Matlab puede escribir a la tarjeta.
msgcount	Investiga cuantos mensajes en un canal están es espera.
open	Abre los canales RTDX.
readmat	Lee matrices de datos de la tarjeta en Matlab como un arreglo.
readmsg	Lee uno más mensajes de un canal.
writemsg	Escribe mensajes a la tarjeta sobre un canal.

Tabla 5.2: Instrucciones para la comunicación con RTDX.

5.6. Flujo de datos del host a la tarjeta

Para que la tarjeta reciba datos del host, primero se declara un canal de salida y los datos requeridos de esta, usando rutinas definidas en la interfaz del usuario. La petición de los datos se registra en el búfer RTDX de la tarjeta y se envía al host a través de la interfaz JTAG. Un cliente host envía datos a la tarjeta usando la interfaz COM. No existe la seguridad de que la transferencia de datos del host a la tarjeta ocurra en tiempo real; los datos se transfieren tan rápido como es posible. Todos los datos que se envía a la tarjeta se escriben a un búfer de la memoria con la librería RTDX del host. Cuando la librería recibe una petición de lectura de la aplicación, los datos en el búfer se envían a la tarjeta por medio de la interfaz JTAG. Los datos se escriben en las localizaciones solicitadas en tiempo real. El host notifica a la librería RTDX de la tarjeta cuando la operación se complementa.

RTDX proporciona control de diagnósticos para verificar que esta trabajando correctamente en el sistema. Los diagnósticos prueban la funcionalidad básica de transmisión.

La prueba interna simula la recepción de datos de una aplicación. Ejecutando esta prueba se asegura que CCS puede procesar los datos correctamente. No existe una aplicación asociada con esta prueba.

La prueba de la tarjeta al host valida la capacidad de la tarjeta para transmitir datos al host y recibir datos de la tarjeta.

La prueba del host a la tarjeta valida la capacidad para recibir datos del host y para transmitir a la tarjeta.

Ejemplo 5.2. ***Recepción de datos***

Para la recepción de datos en la aplicación de CCS, ejecute el Programa 5.3, el código es propio para recibir datos de Matlab, y presenta similitudes con el anterior. Se debe ejecutar el Programa 5.4, antes efectuar la aplicación.

Efectúe los siguientes pasos:

1. Abra el proyecto s1l1.pjt, cuya ruta es `ti/tutorial/dsk6711/sect_1/less_5`.
2. Abra el programa S1L5.c.
3. Seleccione Load Program, del menu File, busque `S1L5.out`, y elija abrir.
4. Seleccione **Tools→RTDX→Configuration Control**. Esto abre la ventana para la configuración de RTDX (Figura 5.2). Active RTDX, marcado la casilla **Enable RTDX**.

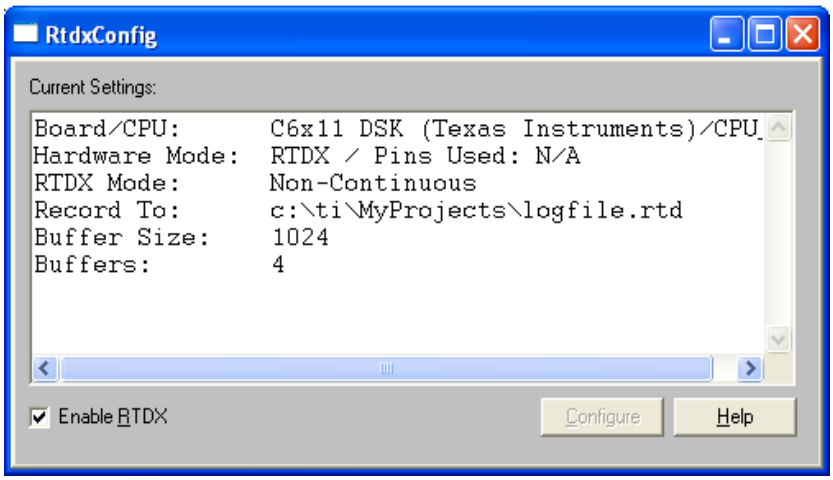


Figura 5.2: Ventana de configuración RTDX.

5. Realice un salto a `main()` en **Debug→Go main**

6. Abra Matlab y ejecute linea por linea las instrucciones del Programa 5.4 o guardelas en un archivo y ejecutelas por su nombre.

7. Regrese a CCS, y ejecute el programa para observar los resultados como en la Figura 5.3.

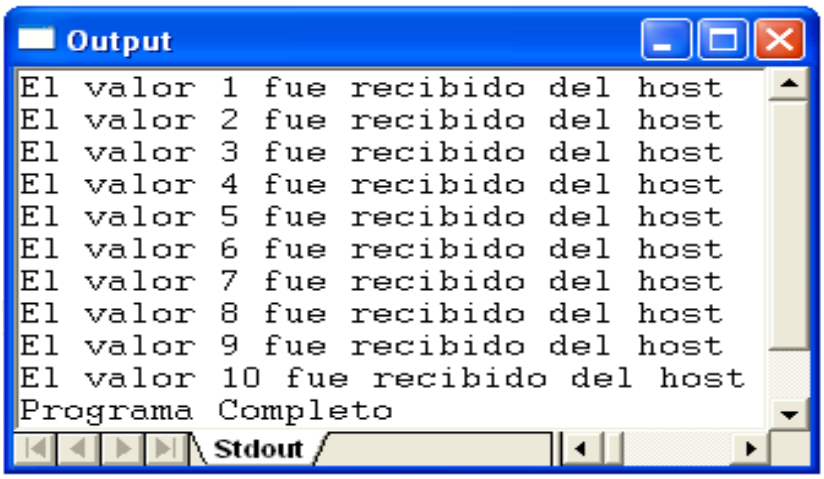


Figura 5.3: Ventana de salida.

8. Por último en Matlab, ejecute el comando `clear`.

Programa 5.3: S1L5.c

```
#include <rtdx.h>
#include "target.h"
#include <stdio.h>

/* Declaración de un canal de entrada */
RTDX_CreateInputChannel( ichan );

void main() {
     int arraydata[10];
     int status,i;

     TARGET_INITIALIZE();
/*Activando el canal de entrada */
     RTDX_enableInput( &ichan );

/*Recibiendo los datos del  host */
     status = RTDX_read( &ichan, arraydata, sizeof(arraydata));
     if ( status != sizeof(arraydata) ) {
          printf( "ERROR: RTDX_read fallo!\n" );
          exit( -1 );
        } else
           for( i=0; i<(sizeof(arraydata)/sizeof(int) ); i++)
               printf( "El valor %d fue recibido del host\n",
                       arraydata[i] );

/* Desactivando el canal de salida */
        RTDX_disableInput(&ichan);
        printf("Programa Completo\n");
}
```

Programa 5.4: dMatlabaC6711.m

```
cc = ccsdsp;
rx = cc.rtdx;
open(rx, 'ochan','w');
enable(rx,'ochan');
pause(5);
indata=(1:10);
writemsg(rx,'ichan',int32(indata))
```

Capítulo 6

Aplicación de audio para el DSK TMS320C6711

- Códec
- McBSP
- EDMA

Una de las áreas de mayor aplicación de los DSPs es en el procesamiento digital de audio y voz, la tarjeta DSK TMS320C6711, esta diseñada específicamente para este objeto, este capítulo incluye la aplicación para audio y voz.

6.1. Introducción

El propósito de esta aplicación es usar la tarjeta para muestrear una señal analógica de sonido en tiempo real. Es común, para manejar las señales en tiempo real, el uso de una rutina de servicio de interrupciones, que se puede usar en esta aplicación para obtener y procesar las muestras de la señal. Además se requiere del uso de periféricos como el códec, el McBSP (Multichannel Buffered Serial Port) y el EDMA (Enhanced Direct Memory Access), para procesar continuamente los datos de sonido digitalizados. La estructura de la aplicación se muestra en la figura 6.1.

6.2. Códec

Todos los ajustes del códec pueden hacerse a través del software de soporte. El software de soporte del DSP contiene funciones de C que se usan para trabajar con el McBSP, el códec y varias utilidades. La librería del códec se activa en el librería de objetos en el archivo drv.lib. La fuente de la librería esta incluida en el archivo

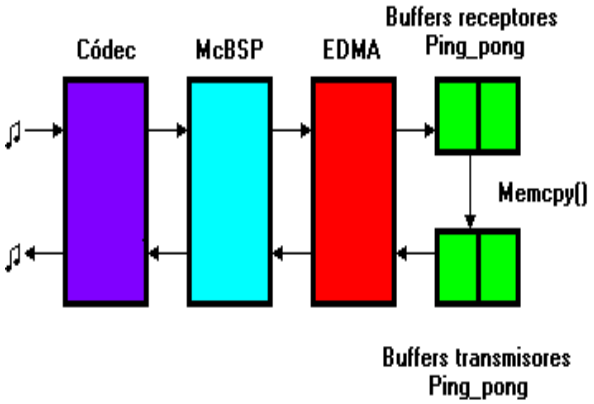


Figura 6.1: Diagrama de bloques para la aplicación.

drv6x.src. La librería del códec también se invoca por las funciones API que pueden usarse para configurar y controlar las operaciones del códec [7, 17].

Inicialización del Códec.

Para escribir un programa que use el códec y pueda muestrear una señal analógica entrante, se realizan algunas inicializaciones. Se debe inicializar la tarjeta del DSP, el McBSP, y el códec. El rango de muestreo del códec se establece, así como los ajustes de ganancia en la entrada del ADC y la salida del DAC. Las funciones API se usan para llevar a cabo todos los ajustes mencionados. Una vez hechas las inicializaciones, se necesita una interrupción para transferir al registro de recepción la salida del procesador y el salto a una rutina de servicio de interrupción. El programa final proporcionará como salida la misma muestra de la entrada al códec para sonido o video.

El próximo paso es abrir un gestor para el McBSP para enviar y recibir datos.

6.3. McBSP

El puerto serial multicanal con buffer (McBSP) se basa en las interfaces estándar del puerto serie, encontrada en los dispositivos con plataformas TMS320C2000 y C5000. Resumiendo, el puerto puede almacenar muestras seriales en un buffer de memoria automáticamente, con la ayuda del controlador EDMA. Este también tiene capacidad multicanal, compatible con los estándares de conexión de redes T1, E1, SCSA y MVIP. Las características del McBSP son: [17]

- Comunicación Full-Duplex.
- Registros de datos de doble buffer para flujo continuo de datos.

- Tramado independiente y temporización para dispositivos y transmisión.
- Interfaz directa a códecs estándar, chips de interfaz analógica (AIC) y otros dispositivos A/D y D/A conectados serialmente.

Tiene las siguientes capacidades:

- Interfaz directa a:
  1. Tramas T1/E1
  2. Dispositivos conforme a ST-BUS
  3. Dispositivos conforme a IOM-2
  4. Dispositivos conforme a AC97
  5. Dispositivos conforme a IIS
  6. Dispositivos conforme a SPI
- Transmisión y recepción multicanal de 128 canales.
- Un selector de ancho del tamaño del dato, que incluye 8,12,16,20,24 y 32 bits.
- Ley μ y Ley A de compresión y expansión
- Transferencia inicial de 8 bits con LSB o MSB.
- Polaridad programable para ambas tramas de sincronización y relojes de datos.
- Reloj interno altamente programable y generación de trama.

El McBSP consta de una vía de datos y una de control, como se muestra en la Figura 6.2, que se conectan a dispositivos externos. Los datos se comunican a los dispositivos externos a través de terminales separadas para transmisión y recepción. El control de información es a través de otras cuatro terminales. El dispositivo se comunica con el McBSP por medio de registros de control de 32 bits por conducto del bus periférico.

Los datos se comunican con el McBSP a través de la terminal datos de transmisión (DX) y de la terminal datos de recepción (DR). El control de la información de temporización y de la trama de sincronización es a través de CLKX, CLKR, FXR y FSR. Los dispositivos periféricos se comunican con el McBSP a través de los registros de control accesible del bus interno de periféricos. El CPU o el controlador del EDMA leen los datos recibidos del registro DRR y escriben los datos para transmitirlos, en el registro DXR. Los datos escritos en DXR se desplazan a DX a través del registro XSR. De igual forma, los datos recibidos en la terminal DR se desplazan por medio del registro RSR y se copian en el registro de buffer de recepción (RBR). RBR se copia en DRR, que puede ser leído por el CPU o el controlador del EDMA. Esto permite el movimiento

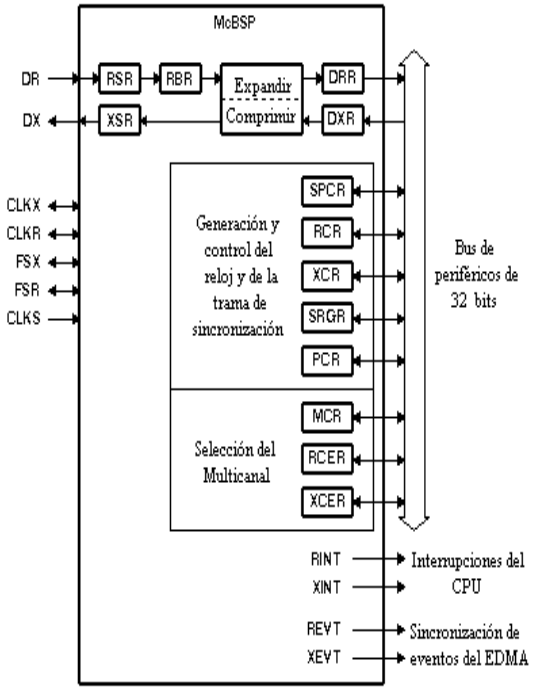


Figura 6.2: Diagrama de bloques del McBSP.

interno de los datos y la comunicación externa simultáneamente. Los registros restantes tienen acceso a la configuración del CPU del mecanismo de control del McBSP. Estos registros se listan en la Tabla 6.1. El bloque de control se compone de un generador de reloj interno, generador de las señales de la trama de sincronización, control de ambos, y selección del multicanal. Este bloque de control envía la notificación de eventos importantes al CPU y al controlador EDMA, a través de cuatro señales como se muestra en la Tabla 6.2.

6.4. EDMA

El controlador de acceso directo mejorado a la memoria (Enhanced Direct Memory Access), EDMA, maneja todas las transferencias entre la memoria caché del nivel L2 y los periféricos, como lo muestra la Figura 6.3. Estas transferencias de datos incluyen el servicio de caché, acceso a la memoria no caché, transferencia de datos programados por el usuario y acceso al host [17, 15].

Registro	Nombre
RBR	Registro de recepción del buffer
RSR	Registro receptor de desplazamiento
XSR	Registro transmisor de desplazamiento
DRR	Registro receptor de datos
DXR	Registro transmisor de datos
SPCR	Registro de control del puerto serial
RCR	Registro de control de recepción
XCR	Registro de control de transmisión
SRGR	Registro del generador de rango de muestras
MCR	Registro del multicanal
RCER	Registro de la activación del canal de transmisión
XCER	Registro de la activación del canal de recepción
PCR	Registro de la terminal de control

Tabla 6.1: Registros del McBSP

Registro	Nombre
RINT	Interrupción de recepción al CPU
XINT	Interrupción de transmisión al CPU
REVT	Sincronización del evento de recepción al EDMA
XEVT	Sincronización del evento de transmisión al EDMA

Tabla 6.2: Interrupciones del cpu y sincronización de eventos del EDMA

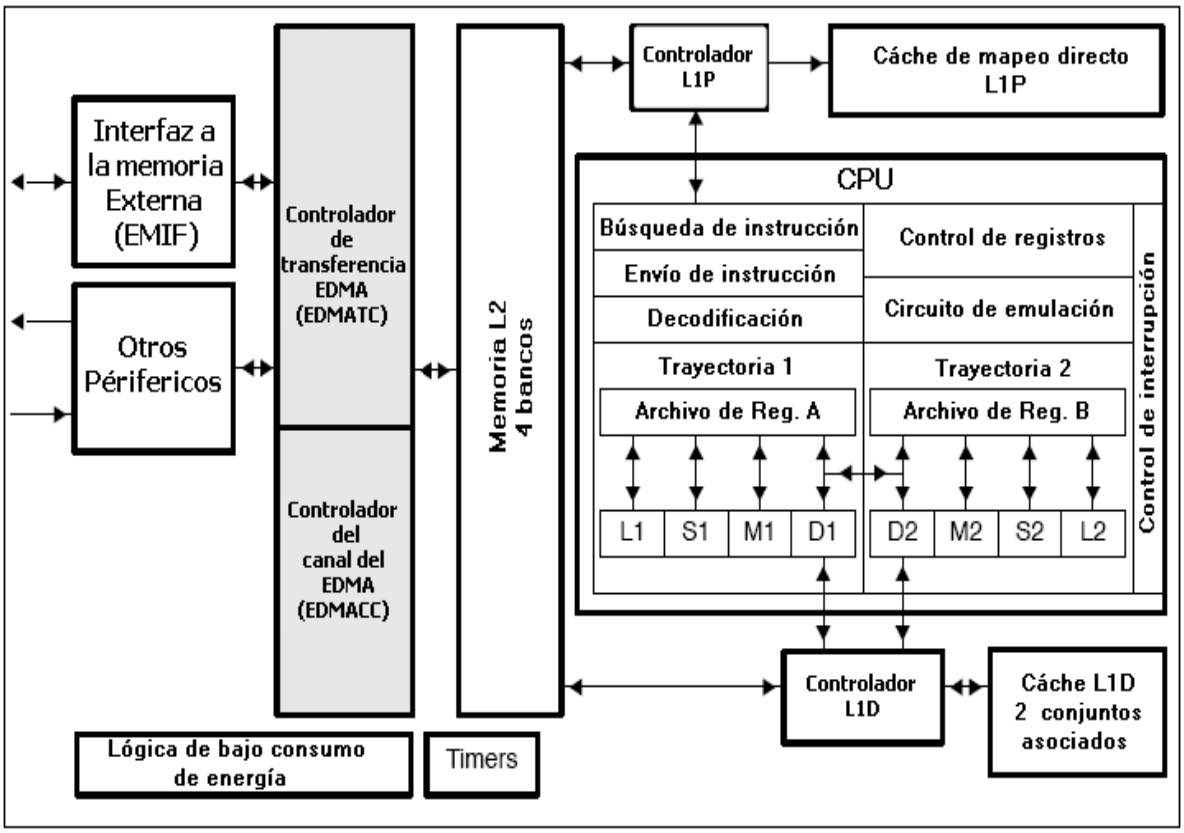


Figura 6.3: Diagrama de bloques del DSP TSM320C6711.

El EDMA del DSP C6711 incluye 16 canales, con prioridad programable, y la capacidad de enlazar la transferencia de datos. EDMA permite el movimiento de los datos de o hacia los espacios direccionables de la memoria, incluyendo a la memoria interna (SRAM L2), a los periféricos, y a la memoria externa. El EDMA consta de dos componentes primarios:

1. El controlador de transferencia (EDMATC), manipula toda la transferencia de datos entre el controlador de la memoria caché y los periféricos como se muestra en la Figura 6.3. En la transferencia de datos se involucra la transferencia al control del canal del EDMA, el acceso a o hacia la EMIF, acceso a la memoria no caché y acceso al periférico maestro.

2. El controlador del canal(EDMACC) es la parte programable del EDMA que soporta un flexible y poderoso conjunto de transferencias, incluyendo las transferencias 1D y 2D; transferencias flexibles de disparo incluyendo las de disparo de eventos, disparo de cadena y disparo del CPU; modos de direccionamiento flexibles que soportan los buffers ping-pong, para almacenamiento circular en buffers, extracción de trama y ordenación.

El EDMACC (EDMA Channel Controler) consta de:

- RAM de parámetros (PaRAM): contiene los parámetros de entrada para el canal y los parámetros de recarga. En la PaRAM se fija el contexto de la transferencia de los canales deseados y de los los parámetros de enlace (link). El EDMACC procesa las entradas basadas en un disparo y envía una solicitud de transferencia (TR) al EDMATC.
- Eventos y registros de procesamiento de interrupción: Permiten activar/desactivar eventos, activa los disparos; activa/desactiva las condiciones de la interrupción; borra y procesa las interrupciones que se complementan con el EDMA.
- Detección del complemento: Este bloque detecta el complemento de la transferencia con el EDMATC. El complemento de la transferencia puede oportunamente usarse para provocar el envío de nuevas transferencias (encadenamiento) o generar las interrupciones al CPU a través EDMA_INT.

.

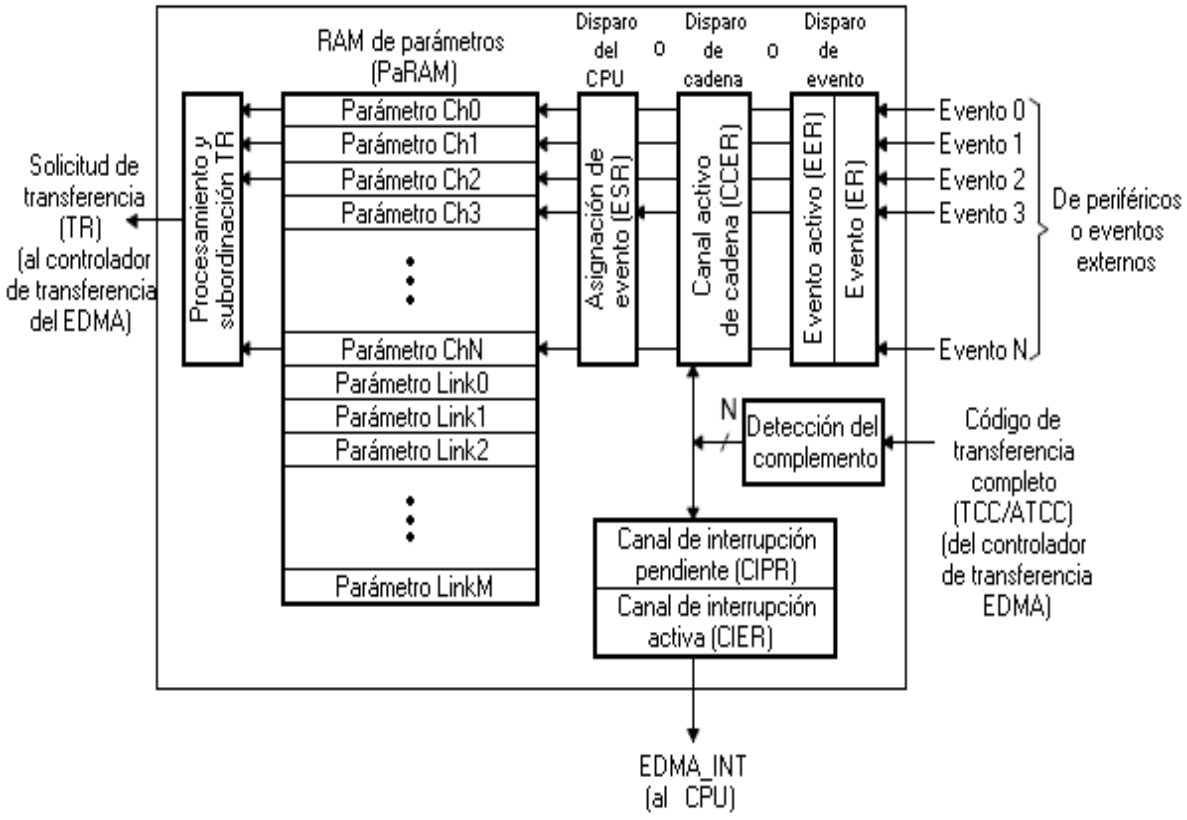


Figura 6.4: Diagrama de bloques del controlador de canales del EDMA.

Los periféricos, EDMA, McBSP y el códec, tienen un papel escencial en la transferencia de datos para cualquier aplicación de procesamiento digital de señales. Estos periféricos deben inicializarse y programarse para su funcionamiento. Además Texas Instruments proporciona un servicio de rutinas básicas usadas para manejar la transferencia de datos. Las canalizaciones (pipes) del DSP/BIOS se usan para transmitir con búfer las entradas y salidas de datos. La transferencia de datos se programan a través de interrupciones de software del DSP/BIOS. Estas interrupciones, diseñadas después de las rutinas de interrupción de hardware, son la base de la estructura de las aplicaciones del DSP/BIOS en una jerarquía prioritaria de subtareas en tiempo real.

6.5. Ejemplo de Audio

La habilidad de los procesadores digitales de señales para manejar una aritmética con rapidez, procesamiento de I/O e interrupciones requiere de una programación básica y servicios de I/O. La base de software del DSP/BIOS, incluido en CCS, suministra un pequeño firmware kernel con servicio básico de tiempo de ejecución la que los programadores pueden embeber en el hardware de la tarjeta del DSP. El DSP/BIOS incluye servicio de tiempo de ejecución optimizado así como subtareas de baja latencia y programación a través de canalizaciones de datos (pipes) diseñados para manejar bloques de I/O (también se les llama base de transmisión o I/O asíncronas). La librería embebida del DSP/BIOS y los soportes modulares son una nueva generación de prueba y herramientas de diagnósticos que permiten probar, trazar y monitorear una aplicación del DSP durante su curso de ejecución. Este monitoreo permite observar la ejecución del sistema en tiempo real así que se puede depurar y ajustar la ejecución del sistema antes de implementarlo [1, 5].

La aplicación de la tarjeta se diseña usando la herramienta de configuración del DSP/BIOS para crear y asignar atributos a los objetos (tareas, transmisiones, etc). A diferencia de otros sistemas en los que la creación del objeto y la inicialización ocurren en el tiempo de ejecución por medio de llamadas suplementarias a las API, incurriendo en el desbordamiento de la tarjeta -especialmente en el espacio del código-, todos los objetos son configurados estáticamente y limitados en un programa ejecutable usando las herramientas. Además para minimizar la memoria de la tarjeta elimina código y optimiza el empleo de estructuras de datos internos. La configuración estática programada por la herramienta de configuración del DSP/BIOS da los medios para la detección de errores semánticos a través de la validación de los atributos prioritarios para la ejecución del programa.

La herramienta de configuración del DSP/BIOS sirve como un editor virtual para la creación de objetos que se usan en la aplicación de la tarjeta a través de las APIs. Esta herramienta gráfica controla fácilmente un amplio rango de parámetros.

Módulos SWI y PIP.

El DSP/BIOS kernel esta internamente organizado en una colección de módulos firmware, cada uno implementa un coherente subconjunto de servicios invocados por la tarjeta a través de las APIs kernel. los módulos individuales pueden manejar uno o mas solicitudes de un objeto kernel y puede contar con valores de parámetros globales para controlar todo el ambiente.

Interrupciones de software o Módulo SWI

El módulo SWI maneja las rutinas de servicio de interrupción de software, que se crean después de las rutinas de interrupción de hardware, se disparan por medio de programación a través llamadas a las APIs, como SWI_POST. Una vez disparada, la ejecución de la rutina SWI tendrá la mayor prioridad a cualquier otra actividad o a cualquier otra SWI de menor prioridad; las rutinas de interrupción de hardware HWI tienen prioridad sobre las SWIs y permanecen activas durante la ejecución de todos los gestores, lo que les permite responder en el momento preciso al hardware del los periféricos. Las SWIs proporcionan subprocesos que tienen prioridad intermedia entre las funciones HWI y la rutina estática (idle loop).

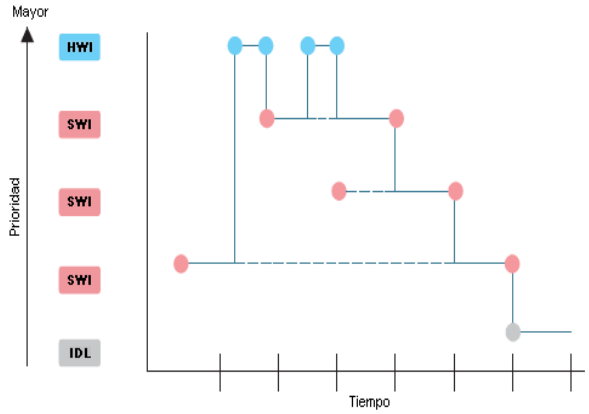


Figura 6.5: Prioridades de los subprocesos del DSP/BIOS.

Pipe o Módulo PIP

El módulo PIP maneja bloques de entrada/salida (también llamado base de transmisión o entrada/salida asíncrona) usado para la transmisión con el buffer de programas de entrada/salida, procesadas por aplicaciones del DSP embebidas. Cada objeto pipe tiene un buffer dividido en un número de tramas de longitud fija, especificadas por las propiedades `numframes` y `framesize`.

Todas las operaciones de entrada/salida en una pipe se distribuyen en la trama en un tiempo. Aun cuando cada trama tiene una longitud fija, la aplicación puede colocar un cantidad variable de datos en cada trama (superior a la longitud de la trama). Observe que una pipe tiene dos fines. La escritura, es donde el programa escribe las tramas de los datos. La lectura, es donde el programa lee las tramas de datos.

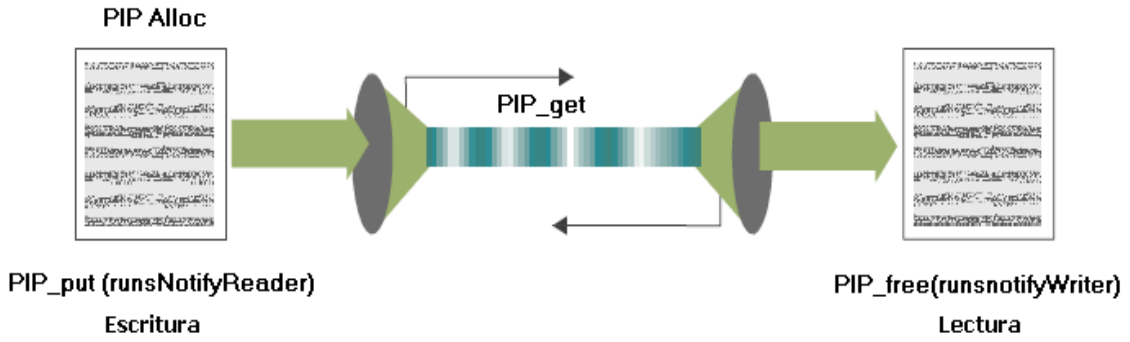


Figura 6.6: Canalizaciones de datos del DSP/BIOS (pipes).

Las funciones de notificación de datos (notifyReader y notifyWriter) se ejecutan para sincronizar la transferencia de datos. Estas funciones se disparan cuando una trama de datos se lee o se escribe para notificar al programa que la trama esta libre o que puede disponer de los datos. Estas funciones se ejecutan en el contexto de la función que llama a PIP_free o PIP_put. También pueden llamarse de los subprocesos PIP_get o PIP_alloc. Después de que se llama a PIP_alloc, el DSP/BIOS revisa si las tramas están llenas de datos. En tal caso, la función notyReader se ejecuta. Cuando se llama a PIP_alloc, y la trama esta vacía la función notyWriter se ejecuta.

Una pipe puede tener un lector y un escritor. Frecuentemente, un extremo de la pipe se controla por una ISR (por ejemplo Receptor del puerto serial ISR) y el otro extremo se controla por una función de interrupción de software llamada audioSWI y por el puerto serial conectado al códec.

En esta aplicación se usan dos objetos pipe para el intercambio de datos entre la interrupción de software y el puerto serial (McBSP) conectado al códec. Estas pipes son DSS_rxPipe y DSS_txPipe. La entrada de datos del códec fluye de la Rutina de servicio de interrupción del puerto serial (ISR) a través de DSS_rxPipe a la interrupción de software. Donde se copia a DSS_txPipe y se envía a la ISR del puerto serial para transmitirse al códec como se muestra en la Figura 6.7.

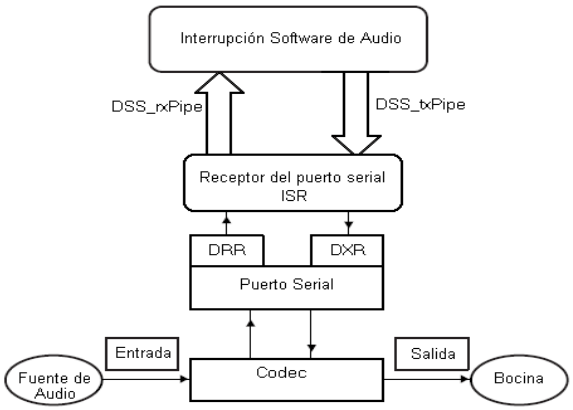


Figura 6.7: Diagrama del ejemplo de Audio.

La ISR para la interrupción receptora del puerto serial copia cada muestra de datos de 32 bits en el DDR (Registro Receptor de datos) a una trama del objeto pipe DSS_rxPipe. Cuando la trama esta llena, la ISR coloca a la trama de regreso en DSS_rxPipe para que sea leída por la función audio.

Como la función audio leerá solo una trama de DSS_rxPipe y la copia en una trama de DSS_txPipe. El rango de transmisión será el mismo que el rango de recepción: 48 kHz. Esto permite simplificar el ejemplo por la activación de la interrupción del puerto serial. la interrupción de transmisión para el puerto serial no esta activada.

La ISR para la interrupción de recepción también cuida del proceso de transmisión en el puerto serial.

Este tomará una trama llena de DSS_txpipe y escribirá una palabra de 32 bits de la trama al registro del puerto serial DXR (Data Transmit Register) cada vez que se maneja la interrupción. Cuando la trama completa ha sido transmitida, la trama vacía se recicla a DSS_txPipe para volver a usarse por la función audio.

La función DSS_init cuida la inicialización del puerto serial y del códec. DSS_init programa el rango de muestreo del códec y fija los bits en IMR e IFR para activar la interrupción del puerto serial, etc. Observe:

- DSS_init no activa las interrupciones y puede llamarse antes de que se activen las interrupciones por el DSP/BIOS al regresar de main.
- DSS_init no fija la tabla de vector de interrupción

Configuración

Todos los objetos se preconfiguran y se modifican en un programa imagen ejecutable. Esto se hace a través de la configuración del DSP/BIOS. Cuando se guarda un archivo de configuración, se crean archivos en ensamblador, archivos cabecera y un archivo de comando enlazador para que se adapten los parámetros de la configuración. Después estos archivos se enlazan con el código de la aplicación.

Los siguientes objetos se configuran o se crean en esta sección:

- Una interrupción de software, para ejecutar la función audio
- Dos canalizaciones (pipes), DSS_rxPipe y DSS_ txPipe, para intercambiar datos entre audioSWI y la ISR de la interrupción receptora del puerto serial.
- Conexión de la correspondiente ISR para el puerto serial usando la HWI (Gestor de la rutina de servicio de interrupción hardware.

Cuando una trama llena se ubica en DSS_rxPipe la función notifyReader borrará el segundo bit en audioSWI. Cuando una trama vacía esta disponible en DSS_txPipe, el primer bit para audioSWI se borra. De esta manera, audioSWI se envía únicamente cuando hay una trama llena disponible en DSS_rxPipe y un trama vacía disponible en DSS_txPipe.

notyWriter para DSS_rxPipe, DSS_rxPrime, es una función en C que se encuentra en dss.c. DSS_rxPrime llama a PIP_alloc para colocar una trama vacía para DSS_rxPipe que se usara por ISR para escribir los datos recibidos del códec. DSS_rxPrime es llamada dondequiera que una trama llena este disponible en DSS_txPrime. La ISR llama a DSS_txPrime después de que hace la transmisión de la trama para adquirir la próxima trama llena. Observe la Figura 6.8.

Es necesario conectar la correspondiente ISR para el puerto serial.

Cada uno de los objetos de HWI corresponden a una interrupción de la tabla de vectores de interrupción. La HWI_INT11 corresponde al puerto serial multicanal con buffer. Con la función _DSS_isr, en el campo function, se maneja la interrupción de recepción del puerto serial.

En las propiedades del objeto SWI se seleccionan microsegundos, para que la herramienta del análisis en tiempo real despliegue datos estadísticos para la interrupción del software en microsegundos.

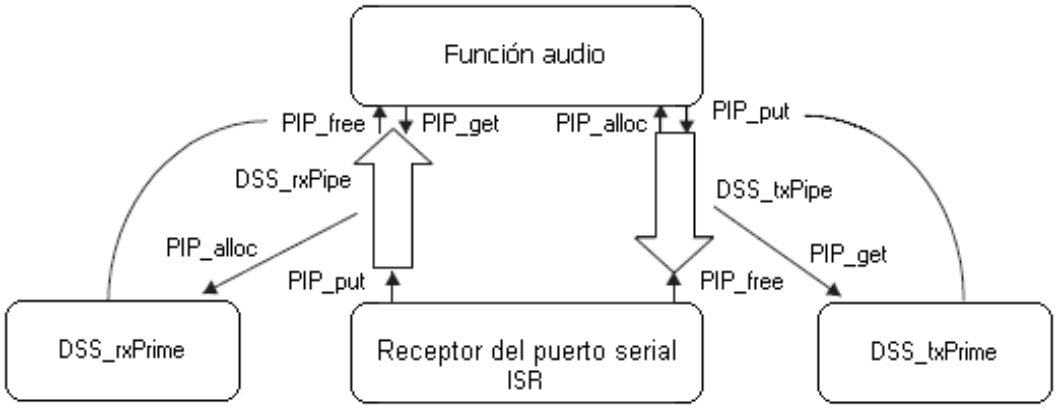


Figura 6.8: Funciones DSS.

6.6. Prueba de la aplicación de audio con el DSK TMS320C6711

El Programa 6.1, es para una aplicación de audio, utiliza los recursos de la herramienta de configuración para activar los objetos que programan la trasferencia de datos, para ejecutarlo y examinarlo siga los pasos descritos a continuación:

1. Copie la carpeta `c:ti\examples\dsk6711\bios\audio` a una nueva carpeta llamada audio en `c:ti\myprojects`

2. Para ejecutar este ejemplo es necesario conectar a la entrada del códec una señal acústica. Por ejemplo, se puede usar el reproductor del disco compacto de una PC y conectar la salida de los audífonos del CD-ROM al jack de entrada de la tarjeta. la salida de la tarjeta se conecta a una bocina u otro dispositivo de salida.

3. Abra el archivo audio.out y ejecute hasta la instrucción `main()`

4. En el panel de control RTA active SWI logging, CLK logging, SWI accumulators y global host enable.

5. Active las ventanas Load Graph y Execution Graph, también puede activar la ventana LOG.

6. Observe y cuente la veces que el objeto audioSWI se ejecuta. Este corre cada cuatro milisegundos que son los pulsos del reloj. La función audio corre cada vez que hay una trama llena en DSS_rxPipe y una trama vacía en DSS_txPipe. ISR se ejecuta cada 1/4800 microsegundos, escribe y lee una palabra en una vez, para tramas de 96 palabras, ISR se necesita ejecutar 96 veces antes de llenar una trama. De esta forma, la función audio puede ejecutarse 96/48000 microsegundos.

Programa 6.1: audio.c

```
#include <std.h>
#include <pip.h>
#include <log.h>
#include <trc.h>
#include <hst.h>
#include <string.h>     /* memset() */
#include <stdarg.h>     /* var arg stuff */
#include "dss.h"
#include "audio.h"

Int PHASE = 0;          /* número de muestras de
                           diferente fase */

static Uns allocBuf(PIP_Obj *out, Uns **buf);
static Void freeBuf(PIP_Obj *out);
static Uns getBuf(PIP_Obj *in, Uns **buf);
static Void process(Uns *src, Int size, Uns *dst);
static Void putBuf(PIP_Obj *out, Uns size);
static Void writeBuf(PIP_Obj *out, Uns *buf, Uns size);

static Int loadVal = 0;
static PIP_Obj *hostPipe = NULL;

/*
 *  ======== main ========
 */
Void main() {
    Int size;
    Uns *buf;

#if AUDIO_USEHST
    hostPipe = HST_getpipe(&hostOutputProbe);
#endif

    DSS_init();     /* Inicialización del códec
                       y del puerto serial*/

    LOG_printf(&trace, "El ejemplo de audio comienza!\n");
```

```
    /* Buffers de salida con silencio */
    size = allocBuf(&DSS_txPipe, &buf);
    memset(buf, 0, size * sizeof (Uns));
    putBuf(&DSS_txPipe, size);

    size = allocBuf(&DSS_txPipe, &buf);
    memset(buf, 0, size * sizeof (Uns));
    putBuf(&DSS_txPipe, size - PHASE);

    /* Rutina estática del BIOS */
    return;
}

/*
 *  ======== audio ========
 */
Void audio(PIP_Obj *in, PIP_Obj *out) {
    Uns *src, *dst;
    Uns size;

if (PIP_getReaderNumFrames(in)==0
    || PIP_getWriterNumFrames(out)==0)
  {
    error("Error: señal de audio falsa");
  }

    size = getBuf(in, &src);
    allocBuf(out, &dst);

/* procesa los datos de salida in src al buffer dst */
    process(src, size, dst);

/* Verifica el error de tiempo real */
if (DSS_error != 0)
  {
    LOG_printf(&trace,
    "Error: DSS fallo en tiempo real!
    (DSS_error = 0x%x)", DSS_error);
    loadVal -= 1;
    DSS_error = 0;
  }
```

```
    /* datos de salida y buffer de entrada libre */
    putBuf(out, size);
    freeBuf(in);
}


/*
 *  ======== error ========
 */
Void error(String msg, ...) {
    va_list va;
    va_start(va, msg);

    LOG_error(msg, va_arg(va, Arg));     /* escribe mensaje */
    LOG_disable(LOG_D_system);           /* Detiene a log */

    for (;;) {
        ;                                /* salto infinito*/
    }
}

/*
 *  ======== load ========
 */
Void load(Arg Arg_prd_ms) {
    static int oldLoad = 0;
    Int prd_ms = ArgToInt(Arg_prd_ms);

    /* visualiza la confirmación de cambio de load */
    if (oldLoad != loadVal ) {
        oldLoad = loadVal;
        LOG_printf(&trace,
        "load: new load = %d000 instrucciones por %d ms",
        loadVal, prd_ms);
        }

    if (loadVal) {
        AUDIO_load(loadVal);
    }
}
```

```
/*
 *  ======== step ========
 */
Void step(void) {
    static Int direction = 1;

    if (loadVal > MAXLOAD) {
        direction = -1;
    }
    if (loadVal <= 0) {
        direction = 1;
    }

    loadVal = loadVal + (100 * direction);
}

/*
 *  ======== allocBuf ========
 */
static Uns allocBuf(PIP_Obj *out, Uns **buf) {
    PIP_alloc(out);
    *buf = PIP_getWriterAddr(out);;
    return (PIP_getWriterSize(out));
}

/*
 *  ======== freeBuf ========
 */
static Void freeBuf(PIP_Obj *out) {
    PIP_free(out);
}

/*
 *  ======== getBuf ========
 */
static Uns getBuf(PIP_Obj *in, Uns **buf) {
    PIP_get(in);
    *buf = PIP_getReaderAddr(in);;
    return (PIP_getReaderSize(in));
}
```

```
/*
 *  ======== process ========
 */
static Void process(Uns *src, Int size, Uns *dst) {
    Int i;

    for (i = size - 1; i >= 0; i--) {
        dst[i] = src[i];
    }
    /* si hostPipe diferente a NULL escribe a la pipe */
    if (hostPipe != NULL) {
        writeBuf(hostPipe, dst, size);
    }
}

/*
 *  ======== putBuf ========
 */
static Void putBuf(PIP_Obj *out, Uns size) {
    PIP_setWriterSize(out, size);
    PIP_put(out);
}

/*
 *  ======== writeBuf ========
 */
static Void writeBuf(PIP_Obj *out, Uns *buf, Uns size) {
    Uns dstSize;
    Uns *dst;
    Int i;

    /* asigna a outBuffer para salida */
    dstSize = allocBuf(out, &dst);

    /* copia los datos al buffer de salida*/
    for (i = (dstSize >= size) ? size : dstSize; i > 0; i--) {
        *dst++ = *buf++;
    }
    /* coloca el buffer en la pipe de salida */
    putBuf(out, size);
}
```

Capítulo 7

Conclusiones

Abrir esta tecnología a usuarios principiantes solicita combinar, por lo menos, previas lecciones de programación básica en lenguaje C y un poco de arquitectura de computadoras. Sin embargo, a la par de las lecciones se pueden revisar temas relacionados. Es importante manifestar que con la llegada de los DSPs se abre un nuevo campo para el diseñador de sistemas.

La primera inconveniencia que se presentó para el desarrollo de este trabajo fue la poca información en Español, además la existente es cara. En contraste, la lista de publicaciones en Inglés es vasta, esto implica una alteración en el tiempo planeado.

Es una prioridad observar las ventajas que ofrece este dispositivo en cuanto a su programación, estas facilidades lo hacen tratable. Aun cuando estos sistemas son apropiados para múltiples tareas, antes de elegir un modelo se debe escoger el sistema más adecuado a nuestras necesidades.

Ante todo debido a sus ventajas los DSK son una opción de cambio para muchas tecnologías y aplicaciones, como telecomunicaciones, medicina, y otras áreas. Las ventajas que se encuentran con el uso de dispositivos como el DSK C6711, son:

- Las herramientas presentadas facilitan la programación.
- La interacción con otros programas amplia su uso.
- Su sistema programable permite flexibilidad en la reconfiguración de aplicaciones.
- Las operaciones pueden ser modificadas en tiempo real, frecuentemente con cambios en la programación, o por cambio de registros.
- La velocidad de ejecución es extraordinaria, pero en ocasiones nos impide, examinar o seguir la pista de la operación.

Apéndice A

Conjunto de instrucciones

Este Apéndice muestra muestra en resumen el conjunto de instrucciones para lenguaje ensamblador. Contiene instrucciones de carga, almacenamiento, instrucciones de operandos, así como control de programas [11].

Instrucciones aritméticas y lógicas	
Instrucción	**Descripción**
ABS	Valor entero absoluto con saturación
ABSDP	Valor absoluto de punto flotante y doble precisión
ABSSP	Valor absoluto de punto flotante y simple precisión
ADD(U)	Suma entera sin saturación
ADDAB/ADDAH/ADDAW	Suma entera usando modo de direccionamiento
ADDAD	Suma entera usando modo de direccionamiento de doble palabra
ADDK	Suma entera usando una constante de 16 bits
ADDDP	Suma de punto flotante y doble precisión
ADDSP	Suma de punto flotante y simple precisión
ADD2	Suma la parte superior e inferior de dos registros de 16 bits
AND	AND lógica bit a bit
CLR	Borra un bit especificado
CMPEQ	Comparación entera para igualdades

CMPEQDP	Comparación de punto flotante y doble precisión para igualdades
CMPEQSP	Comparación de punto flotante y simple precisión para igualdades
CMPGT(U)	Comparación entera con signo para mayor que
CMPGTDP	Comparación de punto flotante y doble precisión para mayor que.
CMPGTSP	Comparación de punto flotante y simple precisión para mayor que
CMPLT(U)	Comparación entera con signo para menor que
CMPLTDP	Comparación de punto flotante y doble precisión para menor que
CMPLTSP	Comparación de punto flotante y simple precisión para menor que
DPINT	Conversión de punto flotante de doble precisión a valor entero
DPSP	Conversión de punto flotante de doble precisión a valor de punto flotante de simple precisión
DPTRUNC	Conversión de punto flotante de doble precisión a valor entero.
EXT	Extrae y extiende el signo a un bit
EXTU	Extrae y extiende un cero a un bit
INTDP(U)	Conversión de entero a valor punto flotante de doble precisión
INTSP(U)	Conversión de entero a valor punto flotante de simple precisión
LMBD	Detección del bit más a la izquierda
MPY(U/US/SU)	Multiplicación entera de 16lsb × 16lsb
MPYDP	Multiplicación de punto flotante y doble precisión
MPYI	Multiplicación entera de 32 bits
MPYID	Multiplicación entera de 32 bits con resultado mayor a 64 bits
MPYH(U/US/SU)	Multiplicación entera de 16msb × 16msb

MPYHL(U)/MPYHULS/MPYHSLU	Multiplicación entera de 16msb × 16msb.
MPYSP	Multiplicación de punto flotante y simple precisión
NORM	Entero normalizado
NOT	NOT lógico
OR	OR lógico
RCPDP	Aproximación recíproca de punto flotante y doble precisión
RCPSP	Aproximación recíproca de punto flotante y simple precisión
RSQRDP	Aproximación recíproca de la raíz cuadrada de punto flotante y doble precisión
RSQRSP	Aproximación recíproca de la raíz cuadrada de punto flotante y simple precisión
SADD	Suma entera con saturación para el tamaño del resultado
SAT	Satura un entero de 40 bits a un entero de 32 bits
SET	Fija un bit
SHL	Desplazamiento aritmético izquierdo
SHR	Desplazamiento aritmético derecho
SHRU	Desplazamiento lógico derecho
SMPY (HL/LH/H)	Multiplicación entera con desplazamiento a la izquierda y saturación
SPDP	Conversión de punto flotante de simple precisión a valor de punto flotante de doble precisión
SPINT	Conversión de punto flotante de simple precisión a valor entero
SPTRUNC	Conversión de punto flotante de simple precisión a valor entero con truncamiento
SSHL	Desplazamiento a al izquierda con saturación
SSUB	Substracción entera con resultado de saturación

SUB(U)	Substracción entera sin saturación
SUBAB/SUBAH/SUBAW	Substracción entera usando modo de direccionamiento
SUBC	Substracción entera condicional y con desplazamiento
SUBDP	Substracción de punto flotante y doble precisión
SUBSP	Substracción de punto flotante y simple precisión
SUB2	Substrae la parte superior e inferior de dos registros de 16 bits
XOR	OR exclusiva

Carga y almacenamiento de instrucciones	
Instrucción	**Descripción**
LDB(U)/LDH(U)/LDW	Carga de la memoria una constante de 5 o 15 bits sin signo a un registro.
STB/STH/STW	Almacena en la memoria de un registro una constante de 5 o 15 bits sin signo.
LDDW	Carga una doble palabra de la memoria a una constante o registro.

Control de Registros	
Instrucción	**Descripción**
MV	Mueve un Registro a otro Registro
MVC	Desplazamiento entre el archivo de control y el archivo de registro.
MVK	Mueve una constante de 16 bits a un registro
MVKH/MVKLH	Mueve una constante de 16 bits a los bits superiores de un registro
MVKL	Coloca en un registro una constante de signo extendido de 16 bits.
NEG	Negativo
ZERO	Fija un registro a cero

Control del programa	
Instrucción	**Descripción**
B	Bifurcación usando un desplazamiento o un registro
B IRP	Bifurcación usando un puntero de regreso de interrupción
B NRP	Bifurcación usando un puntero de regreso NMI
IDLE	Multiciclo NOP finaliza hasta que exista una interrupción
NOP	No operación

Apéndice B

Segmentación encauzada (pipeline).

Las instrucciones del C6711 fluyen a través de las etapas de la segmentación encauzada: extracción, decodificación y ejecución. La etapa de extracción tiene cuatro fases y la decodificación tiene dos. La etapa de ejecución requiere de varias fases, dependiendo del tipo de la instrucción. Las etapas se muestran en la Figura B.1 [11].

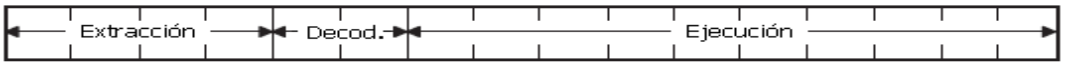


Figura B.1: Etapas de la segmentación encauzada de punto flotante.

Extracción

El C6711 usa un paquete de extracción (FP) de ocho instrucciones. Estas instrucciones continúan con el proceso de extracción a través de las fases:

- PG: Generar dirección del programa en el CPU.
- PS: Envíar dirección del programa a la memoria.
- PW: Esperar que el acceso al programa este listo
- PR: El CPU recibe el paquete de extracción del programa.

La Figura B.2(a) muestra las fases de extracción en orden secuencial. En la Figura B.2(b) se observa un diagrama funcional del flujo de las instrucciones a través de las fases de extracción. En la Figura B.2(c), el primer paquete (en PR) se compone de cuatro paquetes de extracción, el segundo y tercero (en PW y PS) contienen dos paquetes cada uno. El último paquete en PG consta de ocho instrucciones de un ciclo simple.

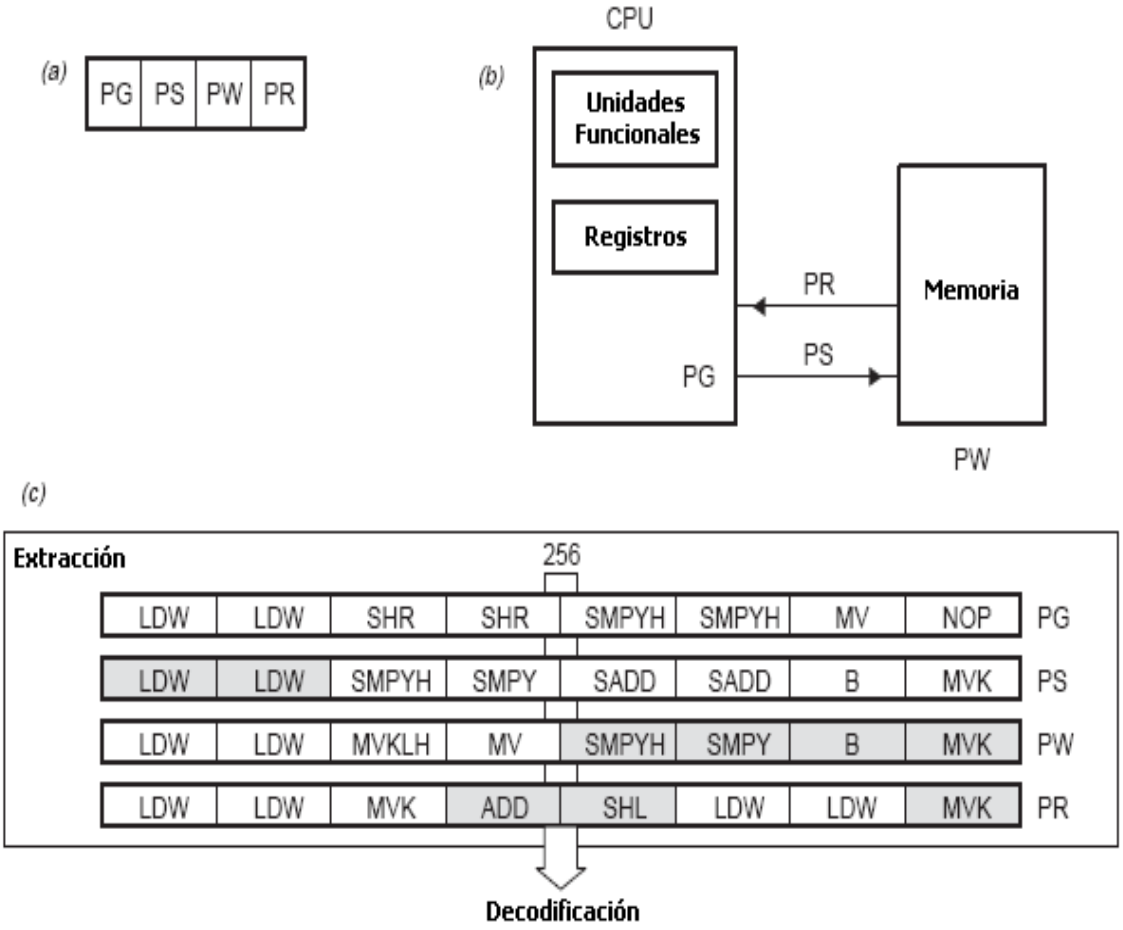


Figura B.2: Fases de la extracción.

Decodificación

Las fases de decodificación son DP despachar la instrucción y DC decodificar la instrucción. En la fase DP los paquetes extraídos se separan en paquetes de ejecución. Los paquetes de ejecución constan de una instrucción o dos en paralelo. Durante la fase DP, las instrucciones en un paquete de ejecución se asignan a las unidades funcionales correspondientes. En la fase DC, se decodifican los registros fuente, los registros de destino y los caminos asociados, para la ejecución de las instrucciones en las unidades funcionales.

La Figura B.3(a) muestra las fases de decodificación. La Figura B.3(b) muestra como un paquete extraído que contiene dos paquetes de ejecución se procesa a través de las etapas de la segmentación encauzada.

Las últimas seis instrucciones del paquete de extracción son paralelas y forman un paquete de ejecución (EP). Este paquete esta en la fase DP de la etapa de decodificar. Las flechas indican la asignación de cada instrucción a una unidad funcional para la ejecución durante el mismo ciclo. La instrucción NOP en la octava ranura de la FP no se remite ninguna unidad funcional porque no hay asociación con esta directiva.

Las primeras dos ranuras del paquete de extracción (sombreadas) representan un paquete de ejecución de dos instrucciones paralelas que fueron enviadas en un ciclo previo. Este paquete de ejecución contiene dos instrucciones MPY que son decodificadas (DC) un ciclo antes de la ejecución. No hay instrucciones decodificadas para las unidades .L, .S y .D.

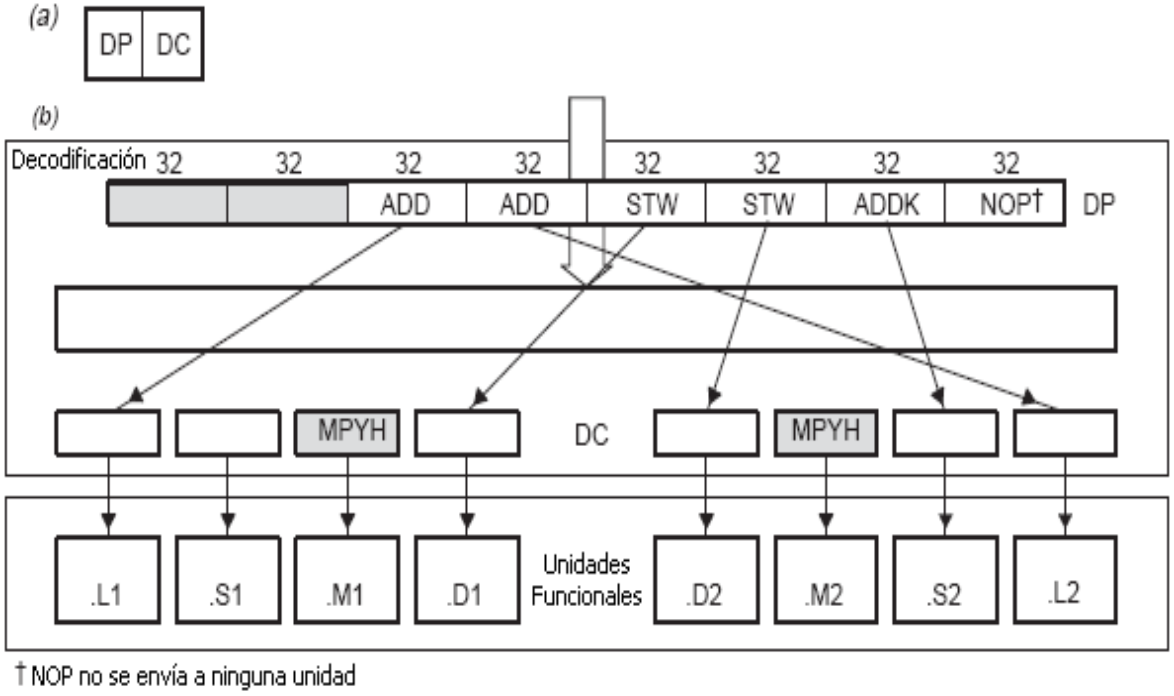


Figura B.3: Fases de la decodificación.

Ejecución

El segmento de ejecución se divide en 10 fases (E1-E10). La variedad tipos de instrucciones requieren diferentes numeros de fases para completar su ejecución. Estas fases de las segmentación encauzada tiene un papel importante en el comportamiento del dispositivo en los ciclos del CPU. La Figura B.4(a) muestra las fases de ejecución en orden secuencial. La Figura B.4(b) expone el diagrama funcional en que ocurre la ejecución

La Figura B.5 muestra todas las fases de las operaciones de la segmentación encauzada. También, muestra el flujo de paquetes extraídos que contienen ocho instrucciones. En este caso. donde la segmentación encauzada esta completa, todas las instrucciones de un paquete extraído están en paralelo y dividas en un paquete de ejecución por paquete extraído. Los paquetes extraídos siguen un modo escalonado a través de las fases de la segmentación encauzada. Si examinamos el ciclo 7 de la Figura B.5 , cuando la instrucción n de FP llega a E1, la instrucción en el paquete de ejecución de FP, n+1 principia a decodificarse. El FP n+2 esta en envío mientras n+3, n+4, n+5 y n+6 están en las cuatro fases del programa de extracción.

B.1. Otras consideraciones

En la programación del C6711, deben tenerse en cuenta algunas consideraciones, como el tener acceso a memoria.

Conflictos.
Una instrucción básica tiene cuatro niveles: buscar, decodificar, leer y ejecutar. Mientras una instrucción se ejecuta, las siguientes tres instrucciones son leídas, decodificadas y buscadas respectivamente. Varias acciones para ejecutar una instrucción se traslapan y se realizan paralelamente. La segmentación encauzada (pipelining) es el traslape de las fases de búsqueda, decodificación, lectura y ejecución de una instrucción. Un conflicto de pipeline ocurre cuando la secuencia del proceso de una instrucción esta lista para ir de un nivel pipeline a el próximo, y ese nivel no esta listo aun para aceptar la transición. Afortunadamente, estos conflictos son visibles al programador, y es importante cuando la velocidad es de consideración crucial.

Conflictos de registro.
Estos conflictos ocurren durante la lectura de un registro, en la escritura de un registro o con un grupo específico de registros por direccionamiento, cuando un registro del mismo grupo no esta listo para usarse. Mas específicamente, si una instrucción escribe en un registro, ningún otro registro puede ser decodificado hasta que el ciclo de escritura este completo.

Conflictos de memoria.
Estos conflictos ocurren porque la memoria interna (RAM0 o RAM1) puede soportar solamente dos accesos por ciclo. Por ejemplo, dos datos tienen acceso al bloque de la RAM y un programa busca el mismo bloque interno de la RAM. El C6711 proporciona una interfaz externa que soporta solo un acceso por ciclo. Los conflictos también ocurren cuando tres datos del CPU entran en un ciclo. Por ejemplo, un almacenamiento (escritura) seguido de dos cargas (lecturas) en paralelo. La escritura debe completarse antes de que las dos lecturas puedan concluir, el retardo de las lecturas es de un ciclo.

El mismo tipo de conflicto ocurre cuando dos escrituras (dos almacenamientos en paralelo) siguen a una lectura.

Eficiencia del Acceso de Memoria.
Si se desea tener un programa y otro o dos datos de acceso en un ciclo, un numero de alternativas pueden producir una ejecución máxima en un simple ciclo. Por ejemplo: un programa tiene acceso en el bus primario y dos datos tienen acceso en la RAM interna.

Caché.
La caché es una sección pequeña de la memoria que se usa para almacenar instrucciones de programas. Si una instrucción se busca en la memoria externa, la caché detecta automáticamente si la instrucción esta ya en la memoria caché. En tal caso, la instrucción se lee de la caché. Si no ocurre un error y la instrucción requerida se copia en la caché.

EDMA.
La transferencia de datos puede ocurrir sin la intervención del CPU. Esto ocurre en paralelo con la ejecución del programa. La separación de buses para el programa y datos, permiten ejecutar un programa paralelamente, leer datos y escribir una operación. Por ejemplo, el C6711 puede ejecutar un programa externo, introducir dos valores de datos en un bloque de la memoria RAM interna, y usar el EDMA para cargar un dato a otro bloque de la RAM. Para la realización de operaciones de entrada y salida, el EDMA puede reducir los efectos pipeline asociados con el CPU.

Estados de espera.
Con periféricos lentos como la memoria externa, los estados de espera pueden insertarse por el programador para acomodar el acceso a dicha memoria. Diferentes números de estados de espera pueden ser programados y aplicados a diferentes localidades de memorias con diferente rapidez.

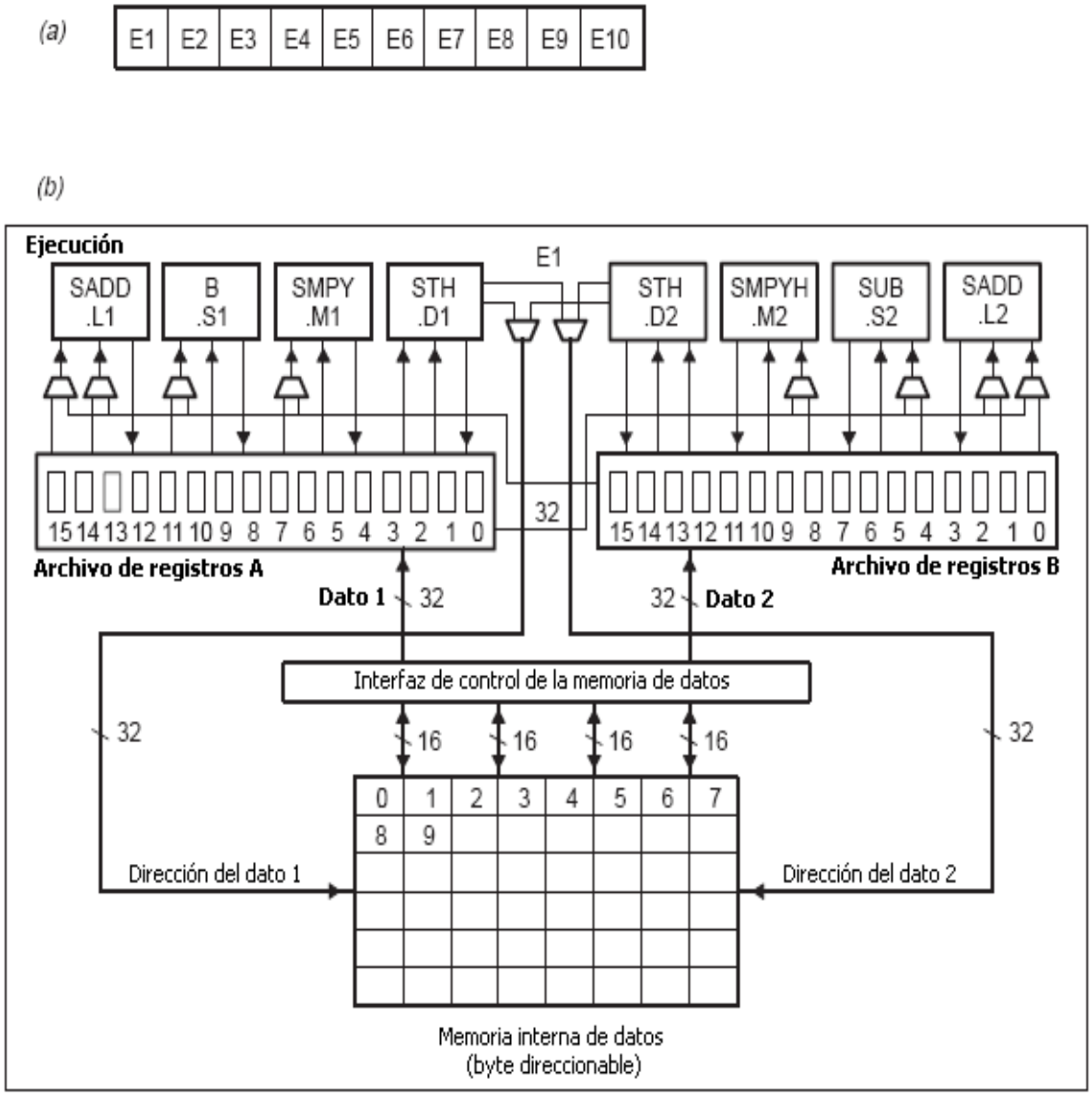


Figura B.4: Fases de la ejecución.

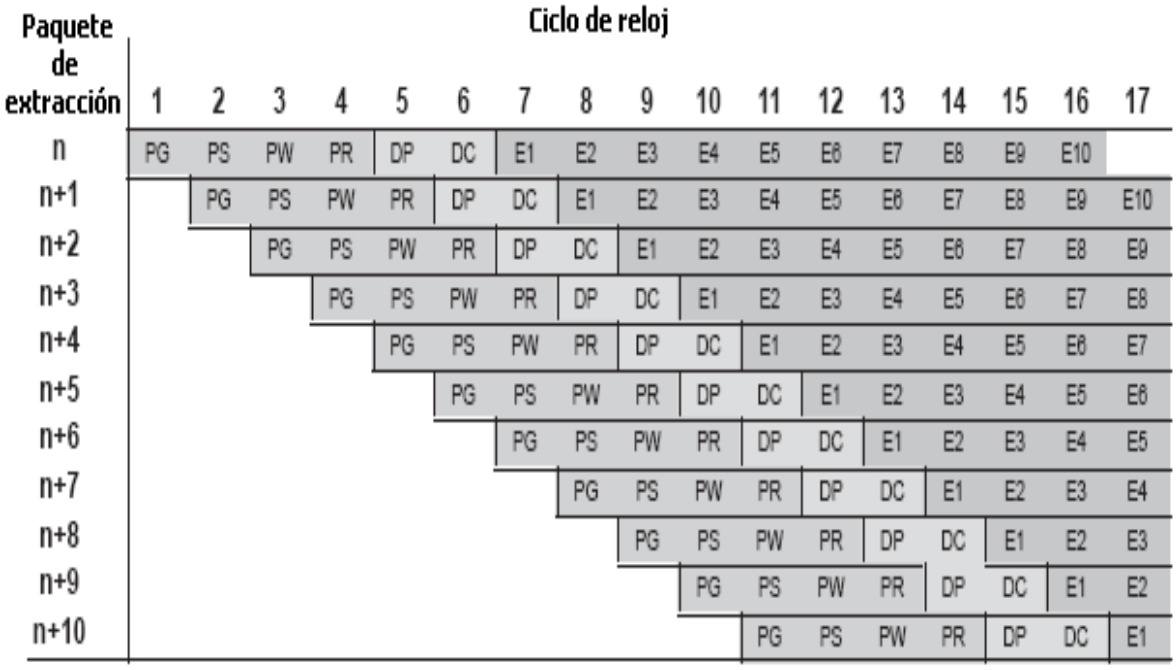


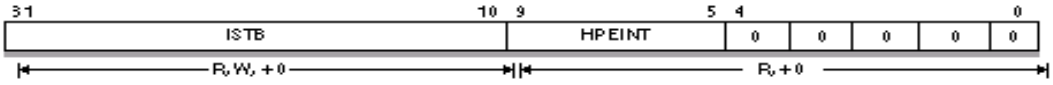
Figura B.5: Operación de la segmentación encauzada.

Apéndice C

Registros.

Registro ISTP

Puntero de Tabla con Servicio de Interrupción (Interrupt Service Table Pointer, ISTP), este registro se usa para localizar la rutina de servicio de interrupción. el campo ISTB, identifica la base de la dirección de la IST; el campo HPEINT, identifica la interrupción especifica y localiza el paquete de extracción con IST.



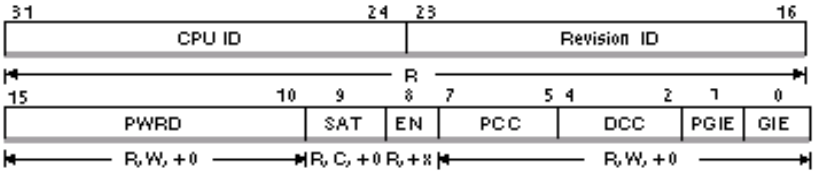
Simbología

R	Legible por la instrucción MVC.
W	Se escribe por la instrucción MVC.
+0	El valor se borra en la reinicialización.

ISTP		
Bits	**Nombre**	**Descripción**
0-4		Inicia a 0 Los paquetes de extracción deben ser de 8 palabras.
5-9	HPEINT	Activa la interrupción de mayor prioridad. Este campo proporciona el numero (relacionado el bit IFR) de la interrupción mas alta que se activa por el bit de IER. Así que, ISTP puede usarse para bifurcaciones manuales para activar la interrupción de mayor prioridad. Si no existe una interrupción pendiente y activa, HPEINT, contiene el valor 00000b. La interrupción no necesita activarse por NMIE o por GIE.
10-31	ISTB	Base de la tabla de servicio de interrupción. Este campo se activa con 0 en la reinicialización. De esta forma, el IST reside en la dirección 0. Después de la reinicialización, el IST se puede recolocar, escribiendo un nuevo valor a ISTB. El primer ISFP nunca se ejecuta a través de un proceso de interrupción, pues la reinicialización lo fija a 0

Registro CSR

El registro de control de estado (Control Status Register, CSR) contiene dos campos que controlan interrupciones: GIE y PGIE. las funciones de los campos de este registro se muestran en la Tabla C.1.



Simbología

R	Legible por la instrucción MVC.
W	Se escribe por la instrucción MVC.
+x	Valor definido después del reset.
+0	El valor es cero después del reset.
C	Se borra usando la instrucción MVC.

CSR		
Bits	**Nombre**	**Descripción**
31-24	CPU ID	CPU ID = 10b: indica al C67x.
23-16	Revision ID	Define la revisión del CPU. Los bits 31:16 para el C6711 son 0x0201.
15-10	PWRD	Controla los modos de interrupción de energía, los valores siempre se leen en ceros
9	SAT	El bit de saturación, se activa cuando alguna unidad esta saturada, puede borrarse solo por la instrucción MVC y puede activarse por una unidad funcional. Se activa en un ciclo completo después de que ocurre la saturación. No se modifica por una instrucción condicional
8	EN	Bit Endian : 1 = little endian, 0 = big endian
7-5	PCC	Control del programa caché
4-2	DCC	Control de datos caché
1	PGIE	GIE Previo (global interrupt enable); guarda el GIE cuando una interrupción se lleva a cabo.
0	GIE	Activa una interrupción global; activa (1) o desactiva (0) todas las interrupciones excepto las interrupciones reset y NMI (interrupción no mascarable).

Tabla C.1: Registro de control de estado

Registro IER

Con el registro de activación de interrupción (Interrupt Enable Register, IER) se puede activar o desactivar una interrupción individual fijando o borrando los bits correspondientes. El bit 0, corresponde a la reinicialización y siempre se encuentra en 1 de esta manera la interrupción de reinicialización siempre se activa. Los bits IE4-IE15 pueden fijarse a 1 o 0, dependiendo de la interrupción asociada. IER se muestra en la figura

Simbología

R	Legible por la instrucción MVC.
W	Se escribe por la instrucción MVC.
Rsv	Reservado.
+1	Valor después del reset.
+0	Valor después del reset.

Registro IFR

El registro bandera de interrupción (Interrupt Flag Register, IFR) contiene el estado de INT4-INT15 y de NMI. Cada bit del IFR se fija a uno cuando ocurre una interrupción correspondiente; por otro lado, los bits tienen un valor de 0.

Simbología

R	Legible por la instrucción MVC.
+0	Valor borrado en el reset.
rsv	Reservado.

Registro ISR

El registro de asignación de interrupción, (Interrupt Set Register, ISR) permite fijar interrupciones mascarables manualmente en IFR. Escribiendo 1 a los campos IS4-IS15 de IFR se activa la bandera de interrupción correspondiente. Si se escribe 0 no existe ningún efecto.

Simbología

W	Se escribe por la instrucción MVC.
Rsv	Reservado.

Registro ICR

El registro de desactivación de interrupción (Interrupt Clear Register, ICR), permite desactivar manualmente las interrupciones mascarables en el IFR. Si se escribe 1 en un bit del ICR provocará que la bandera de interrupción se desactive, si se escribe un 0 no causa ningún efecto. Las interrupciones tienen prioridad sobre cualquier escritura al ICR.

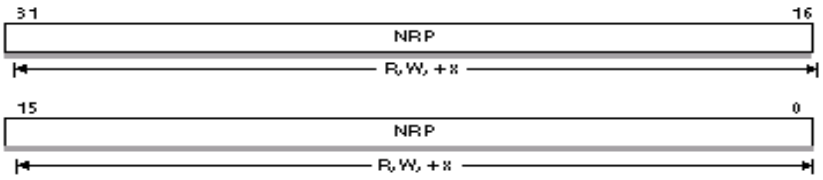
Simbología

W	Se escribe por la instrucción MVC.
Rsv	Reservado.

Registro NRP

El registro puntero de retorno de NMI contiene el puntero que direcciona el CPU a la localización propia para continuar la ejecución del programa después del proceso NMI. Una bifurcación usando la dirección en NPR (B NPR) en el servicio de rutina de interrupción retorna al programa cuando el servicio NMI esta completo.

El NPR contiene la dirección de 32 bits del primer paquete de ejecución en el programa que no fue ejecutado por causa de una interrupcion no mascarable. Así se puede escribir un valor a este registro, cualquier interrupción subsecuente puede sobre escribir ese valor. La figura abajo muestra este registro:



Simbología

R	Legible por la instrucción MVC.
W	Se escribe por la instrucción MVC.
+x	Valor indefinido después de Reset

Apéndice D

Conjunto de funciones API

Las APIs del BIOS se condicionan por los estándares de C. El formato para el nombre de una operación DSP/BIOS consta de tres o cuatro letras como prefijo del módulo que contiene la operación, un guión bajo, y la acción. En este Apéndice se hace una breve descripción de cada una [13].

Operaciones para el módulo ATM	
Función	**Operación**
ATM_andi, ATM_andu	AND atómica para dos localizaciones de memoria. Devuelve el valor previo de la segunda
ATM_cleari, ATM_clearu	Borrado atómico de la localización de memoria y regresa el valor previo
ATM_deci, ATM_decu	Decremento atómico de memoria, retorna un valor nuevo.
ATM_inci, ATM_incu	Incremento atómico de memoria y regresa un valor nuevo
ATM_ori, ATM_oru	OR atómica para una localización de memoria, devuelve el valor previo
ATM_seti, ATM_setu	Asignación atómica de memoria, retorna un valor pervio

Operaciones para el módulo CLK	
Función	**Operación**
CLK_countspms	Número de ejecuciones del hardware por milisegundos
CLK_gethtime	Consigue el tiempo de alta resolución
CLK_getltime	Obtiene el tiempo de baja resolución
CLK_getprd	Proporciona el valor del registro period

Operaciones para el módulo DEV	
Función	**Operación**
DEV_match	Señala el nombre de un dispositivo como controlador
Dxx_close	Cierra al dispositivo
Dxx_ctrl	Operación de control del dispositivo
Dxx_idle	Dispositivo Idle
Dxx_init	Inicialización del dispositivo
Dxx_issue	Envía un buffer al dispositivo
Dxx_open	Abre un dispositivo
Dxx_ready	Certifica si el dispositivo esta listo para entra/salida
Dxx_reclaim	Recupera un buffer de un dispositivo

Operaciones para el módulo HOOK	
Función	**Operación**
HOOK_getenv	Obtiene un puntero de ambiente para la combinación de HOOK y TSK
HOOK_setenv	Fija el puntero de ambiente para la combinación de HOOK y TSK

Operaciones para el módulo HST	
Función	**Operación**
HST_getpipe	Obtiene el objeto pipe correspondiente

Operaciones para el módulo HWI	
Función	**Operación**
HWI_disable	Desactivación global de interrupciones de hardware
HWI_enable	Activación global de interrupciones de hardware
HWI_enter	Prologo de la rutina de servicio de interrupción de Hardware
HWI_exit	Epílogo de la rutina de servicio de interrupción de Hardware
HWI_restore	Restablecimiento global del estado de la interrupción

Operaciones para el módulo IDL	
Función	**Operación**
IDL_run	Realiza un paso a través de las funciones idle

Operaciones para el módulo LCK	
Función	**Operación**
LCK_create	Crea un recurso de bloqueo
LCK_delete	Borra un recurso de bloqueo
LCK_pend	Adquiere una contraseña del recurso de bloqueo
LCK_post	Reemplaza una contraseña del recurso de bloqueo

Operaciones para el módulo LOG	
Función	**Operación**
LOG_disable	Desactiva un log
LOG_enable	Activa un log
LOG_error/LOG_message	Escribe un mensaje de error al usuario del log del sistema
LOG_event	Agrega un mensaje sin formato a un log
LOG_printf	Añade un mensaje con formato a un log
LOG_reset	Reinicia un log

Operaciones para el módulo MBX	
Función	**Operación**
MBX_create	Crea un buzón (mailbox)
MBX_delete	Borra un mailbox
MBX_pend	Espera un mensaje del mailbox
MBX_post	Envía un mensaje al mailbox

Operaciones para el módulo MEM	
Función	**Operación**
MEM_alloc, MEM_valloc, MEM_calloc	Asignación de una pila de memoria
MEM_define	Define una nueva pila de memoria
MEM_free	Libera un bloque de memoria
MEM_redefine	Redefine una pila de memoria existente
MEM_stat	Retorna el estado de la pila de memoria

Operaciones para el módulo PIP	
Función	**Operación**
PIP_alloc	Adquiere una trama vacía de una pipe
PIP_free	Recicla una trama que ha sido usada por una pipe
PIP_get	Recibe una trama completa de una pipe
PIP_getReaderAddr	Obtiene el valor del puntero readerAddr de la pipe
PIP_getReaderNumFrames	Capta el número de tramas de pipe libres para lectura
PIP_getReaderSize	Recaba el número de palabras en una trama de pipe
PIP_getWriterAddr	Obtiene el valor del puntero writerAddr de la pipe
PIP_getWriterNumFrames	Capta el número de tramas de pipe libres para escritura
PIP_getWriterSize	Recaba el número de palabras que pueden escribirse en una trama de pipe
PIP_peek	Logra el tamaño de la trama pipe y las direcciones sin reclamar la trama de la pipe
PIP_put	Coloca una trama completa en una pipe
PIP_reset	Reinicia todos los campos de un objeto pipe a sus valores originales
PIP_setWriterSize	Fija el número de palabras escritas a la trama de la pipe

Operaciones para el módulo PRD	
Función	**Operación**
PRD_getticks	Obtiene el contador del pulso presente
PRD_start	Crea una función periódica para de una sola ejecución
PRD_stop	Detiene una función periódica de la ejecución
PRD_tick	Avance del contador de pulso, envío de funciones periódicas

Operaciones para el módulo QUE	
Función	**Operación**
QUE_create	Crea una cola vacía
QUE_delete	Borra una cola vacía
QUE_dequeue	Remueve del tope de la cola (no atómica)
QUE_empty	Examen de una cola vacía
QUE_enqueue	Inserat al final de una cola (no atómica)
QUE_get	Obtiene el elemento del frente de la cola (atómica)
QUE_head	Retorna al elemento del frente de la cola
QUE_insert	Inserta en el medio de la cola (no atómica)
QUE_new	Asigna una cola para desalojarse
QUE_next	Retorna el elemento proximo de una cola (no atómica)
QUE_prev	Regresa el elemento previo de una cola (no atómica)
QUE_put	Coloca una elemento al final de la cola (atómica)
QUE_remove	Remueve de la mitad de la cola (no atómica)

Operaciones para el módulo RTDX	
Función	**Operación**
RTDX_channelBusy	Retorna el estado indicando si un canal esta ocupado
RTDX_CreateInputChannel	Declara una estructura para el canal de entrada
RTDX_CreateOutputChannel	Declara una estructura para el canal de salida
RTDX_disableInput	Desactiva un canal de entrada
RTDX_disableOutput	Desactiva un canal de salida
RTDX_enableInput	Activa un canal de entrada
RTDX_enableOutput	Activa un canal de salida
RTDX_isInputEnabled	Regresa el estado de un canal de entrada
RTDX_isOutputEnabled	Retorna el estado de un canal de salida
RTDX_read	Lectura de un canal de entrada
RTDX_readNB	Lectura de un canal de entrada sin bloqueo
RTDX_sizeofInput	Retorna el número de bytes leídos de un canal de entrada
RTDX_write	Escritura en un canal de salida

Operaciones para el módulo SEM	
Función	**Operación**
SEM_count	Obtiene el contador de semáforo presente
SEM_create	Crea un semáforo
SEM_delete	Borra un semáforo
SEM_ipost	Señala un semáforo (sólo interrupciones)
SEM_new	Inicializa un semáforo
SEM_pend	Espera un semáforo
SEM_post	Indica un semáforo
SEM_reset	Reinicia un semáforo

Operaciones para el módulo SIO	
Función	**Operación**
SIO_bufsize	Tamaño de los buffers usados por una transmisión
SIO_create	Crea la transmisión
SIO_ctrl	Ejecuta las operaciones de control de un dispositivo
SIO_delete	Borra una transmisión
SIO_flush	Desactiva una transmisión por el nivel de los buffers
SIO_get	Obtiene el buffer de la transmisión
SIO_idle	Desactiva una transmisión
SIO_issue	Envía un buffer a la transmisión
SIO_put	Coloca un buffer para una transmisión
SIO_reclaim	Solicita un buffer para una transmisión
SIO_segid	Sección de memoria usada por una transmisión
SIO_select	Selecciona un dispositivo
SIO_staticbuf	Adquiere un buffer estático de la transmisión

Operaciones para el módulo STS	
Función	**Operación**
STS_add	Agrega un valor a un objeto de estadísticas
STS_delta	Añade un valor calculado a un intervalo del objeto
STS_reset	Reinicia los valores almacenados en un objeto STS
STS_set	Almacena un valor inicial de un intervalo al objeto

Operaciones para el módulo SWI	
Función	**Operación**
SWI_andn	Borra los bits del buzón de SWI y lo envía si es 0
SWI_andnHook	Versión especial de SWI_andn
SWI_create	Crea una interrupción de software
SWI_dec	Decrementa el buzón de SWI y lo envia si es 0
SWI_delete	Borra una interrupción de software
SWI_disable	Desactiva una interrupción de software
SWI_enable	Activa una interrupción de software
SWI_getattrs	Consigue los atributos de una interrupción de software
SWI_getmbox	Retorna el valor del buzón de SWI
SWI_getpri	Regresa una mascara de prioridad de SWI
SWI_inc	Incrementa el buzón de SWI y lo envía
SWI_or	Fija o enmascara un buzón de SWI y lo transmite
SWI_oHook	Versión especial de SWI_or
SWI_post	Transmite una interrupción de software
SWI_raisepri	Aumenta la prioridad de SWI
SWI_restorepri	Restaura la prioridad de una SWI
SWI_self	Regresa la dirección del objeto SWI que se encuentra en ejecución
SWI_setattrs	Fija los atributos de una interrupción de software

Operaciones para el módulo SYS	
Función	**Operación**
SYS_abort	Aborta la ejecución de un programa
SYS_atexit	Apila un gestor detenido
SYS_error	Bandera de condición de error
SYS_exit	Termina la ejecución de un programa
SYS_printf, SYS_sprintf, SYS_vprintf, SYS_vsprintf	Salida formateada
SYS_putchar	Muestra un simple carácter

Operaciones para el módulo TRC	
Función	**Operación**
TRC_disable	Desactiva los controladores del conjunto de trace
TRC_enable	Activa los controladores del conjunto de trace
TRC_query	Prueba si los controladores del conjunto de trace están activos

Operaciones para el módulo TSK	
Función	**Operación**
TSK_checkstacks	Verifica el sobreflujo de la pila
TSK_create	Crea una tarea para ejecutarse
TSK_delete	Borra una tarea
TSK_deltatime	Actualiza la tarea STS con diferente tiempo
TSK_disable	Desactiva la tarea programada del DSP/BIOS
TSK_enable	Activa la tarea programada del DSP/BIOS
TSK_exit	Finaliza la ejecución de una tarea
TSK_getenv	Obtiene el ambiente de tareas
TSK_geterr	Obtiene el número de error de la tarea
TSK_getname	Obtiene el nombre de la tarea
TSK_getpri	Obtiene la prioridad de la tarea
TSK_getsts	Obtiene los objetos STS de la tarea
TSK_itick	Sistema avanzado de la alarma de reloj (sólo interrupciones)
TSK_self	Retorna un gestor para la tarea presente
TSK_setenv	Fija el ambiente de la tarea
TSK_seterr	Fija el número de error de la tarea
TSK_setpri	Fija la prioridad para la ejecución de la tarea
TSK_settime	Fija el tiempo para la tarea STS previa
TSK_sleep	Retarda la ejecución de la tarea actual
TSK_stat	Recupera el estado de una tarea
TSK_tick	Sistema avanzado de la alarma del reloj
TSK_time	Retorna el valor presente del sistema de reloj
TSK_yield	Cede el paso a una tarea de igual prioridad

Librería C stdlib.h	
Función	**Operación**
atexit	Registra una o mas funciones de salida usadas para abortar
calloc	Asigna el bloque de memoria inicializado con ceros
exit	Llama a las funciones de salida registradas en atexit
free	Libera bloques de memoria
getenv	Busca una combinación en el ambiente de cadena
malloc	Asigna de bloque de memoria
realloc	Redimenciona previamente el bloque de memoria asignado

Operaciones para el módulo TRC	
Función	**Operación**
ArgToInt(arg)	Intercambia el parámetro de tipo Arg para que se trate como de tipo entero (Int) en la tarjeta
ArgToPtr(arg)	Intercambia el parámetro de tipo Arg para que se trate como de tipo puntero (Ptr) en la tarjeta

Glosario

A

acceso directo a memoria mejorado: Acceso a la memoria que no usa al CPU; se usa para transferir datos directamente entre la memoria y un periférico.

AD535: Módulo API del audio codec. Soportado por el DSK 6711.

ALU: Ver unidad aritmética y lógica.

API: Ver interfaz programable para la aplicación.

ASIC: Ver circuito integrado de aplicación específica.

archivo de registro de control: Conjunto de registros de control.

archivo objeto: archivo que se ha ensamblado o enlazado y contiene código en lenguaje máquina.

B

bandera: Indicador binario que señala el estado de una condición que ha ocurrido o esta llevándose a cabo.

big endian: Protocolo de direccionamiento en el que los bytes son numerados de izquierda a derecha en una palabra. Los bytes mas significativos en una palabra tienen una dirección inferior. El orden endian es específico para hardware y se determina en el reset.

bit: Dígito binario, puede ser 0 o 1.

bit de activación de interrupción global: Bit del registro de control de estado (CSR) que se usa para activar o desactivar interrupciones mascarables.

bit más significativo: El bit de mayor orden en una palabra.

bit menos significativo: El bit de menor orden en una palabra.

boot: Proceso de cargar un programa a la memoria.

BSL: Ver librería de soporte de la tarjeta.

byte: Secuencia de ocho bits que operan como una unidad.

C

caché: Buffer de rápido almacenamiento en la CPU de una computadora.

caché de programa: Memoria caché rápida para almacenamiento de instrucciones de programa que permite la ágil ejecución.

cargador del boot (boot loader): Segmentos de código que transfiere código de una fuente externa a la memoria de un programa

circuito integrado de aplicación específica ASIC: Chip diseñado para una aplicación específica. Esta integrado por celdas de una librería.

ciclos de reloj: Una secuencia de eventos periódicos basados en la entrada de un reloj externo.

códec (codificador-decodificador, o compresión/descompresión): Dispositivo que codifica en una dirección de transmisión y decodifica en otra dirección de transmisión.

código: Conjunto de instrucciones escritas para ejecutar una tarea; un programa de computadora o parte de un programa.

compresión y expansión: esquema de cuantización para señales de audio en la que la señal de entrada se comprime y se reconstruye en la salida por expansión. Existen dos esquemas: Ley A (Para uso de Europa) y Ley μ (Para uso de Estados Unidos).

compilador: Programa de computación que traduce un programa de lenguaje de alto nivel a su equivalente en lenguaje ensamblador.

configuración del boot: Conjunto de parámetros definidos para la carga de un dispositivo.

controlador del acceso directo a memoria: circuitería especializada que transfiere los datos de la memoria a la memoria sin uso del CPU.

CPU: Ver unidad central de procesamiento.

camino cruzado (crosspath): Enlace entre archivos de registros para la comunicación entre las unidades del CPU.

D

dirección: La localización de almacenamiento de datos o de un programa; una localización de memoria de acceso individual.

direccionamiento circular: Tipo de direccionamiento en el que un conjunto finito de direcciones se reutiliza por el enlace de las direcciones menor y mayor.

direccionamiento indirecto: Modo de direccionamiento en el que una dirección apunta a otra en vez de apuntar al dato actual; este modo esta prohibido en la arquitectura RISC.

DRAM: Ver memoria dinámica de acceso aleatorio.

E

EDMA: Ver acceso directo a memoria mejorado.

ensamblador: Software que crea un programa en lenguaje máquina de un archivo fuente que contiene instrucciones de lenguaje ensamblador, directivas y definiciones de macros. El ensamblador substituye códigos de operaciones absolutas o códigos de localización por direcciones simbólicas.

estructura(frame): Espacio de 8 palabras en la cáche RAM. Cada paquete de búsqueda en la cáche reside en una sola estructura. La cáche actualiza las cargas con el paquete de búsqueda. La cáche contiene 512 estructuras.

E1: Servicio europeo de comunicación entre redes de alta rapidez que opera a 2.048 M bits por segundo y usa la ley de compresión y expansión A.

G

GIE: Ver bit de activación de interrupción global.

H

host: Dispositivo al que se conectan periféricos y a los que generalmente controla.

HPI: Ver interfaz del puerto host.

I

interfaz de memoria externa (EMIF): Microprocesador que se usa para para leer o escribir a la memoria de un chip.

interfaz del puerto host HPI: Interfaz paralela que usa el CPU para comunicarse con el procesador del host.

interfaz programable para la aplicación (API): se usa por programas de aplicación para interactuar con software de comunicaciones o para ajustarse a otros productos.

interrupción: Señal enviada por hardware o software para solicitar la atención del procesador. La señal le indica al procesador que suspenda su operación presente, guarde el estado de la tarea presente, y ejecute un conjunto particular de instrucciones. Las interrupciones se comunican con el sistema operativo y establece un orden de prioridades de las tareas para ejecutarse.

interrupción externa: Interrupción de hardware activada por un pin.

interrupción de hardware: Interrupción activada a través de las conexiones física de los periféricos o dispositivos externos.

interrupción mascarable: Interrupción de hardware que puede activarse o desactivarse por medio de software.

interrupción no mascarable NMI: Interrupción que no puede puede ser mascarada ni desactivada.

K

kernel: es la parte fundamental de un sistema operativo. Es el software responsable de facilitar a los distintos programas acceso seguro al hardware de la computadora o en forma más básica, es el encargado de gestionar recursos, a través de servicios de llamada al sistema. Es el cuerpo de una rutina con segmentación encauzada que se encuentra entre el prólogo y el epílogo.

L

latencia: Retardo entre el suceso de una condición y la reacción del dispositivo. También, en la segmentación encauzada, el retardo entre la ejecución de dos instrucciones en conflicto para asegurarse que los valores usados por la segunda instrucción son correctos.

ley de compresión y expansión A: Ver compresión y expansión.

ley de compresión y expansión μ: Ver compresión y expansión.

librería de soporte de la tarjeta BSL: Conjunto de APIs usadas para configurar y controlar los niveles de los periféricos.

LSB: Ver bit menos significativo.

M

mapa de memoria: representación gráfica del sistema de memoria de una computadora, que muestra las localizaciones del espacio del programa, el espacio de los datos, los espacios reservados y otros elementos residentes en la memoria.

McBSP: Ver puerto serial multicanal con buffer.

media palabra: Para este dispositivo se define como un dato de 16 bits.

memoria de acceso aleatorio RAM: Tipo de dispositivo de memoria en la que las localizaciones individuales pueden tener acceso en cualquier orden.

memoria de acceso aleatorio dinámica asíncrona SDRAM: RAM cuyo contenido se evoca periódicamente para prevenir perdidas de datos. La transferencia de datos tiene un rango fijo relativo a la rapidez del reloj del dispositivo.

memoria de datos: La región de la memoria usada para almacenamiento o manipulación de datos, separada de la región usada para almacenamiento del código del programa.

memoria del programa: La región de la memoria usada para almacenamiento y ejecución de programas, separada de la región usada para almacenamiento del datos.

memoria dinámica de acceso aleatorio DRAM: Memoria en la que el procesador puede leer o escribir y a cuyas localizaciones de almacenamiento se puede tener acceso pero deben ser evocadas periódicamente para retener datos o código de programa.

memoria flash: Ver memoria instantánea.

memoria sólo de lectura de borrado programable EPROM: Dispositivo de memoria cuyo contenido pueden borrarse (usualmente a través de luz UV) y es programable.

memoria instantánea Memoria que se borra electrónicamente, memoria programable inconstante (sólo de lectura)

millones de instrucciones por segundo MIPS: medida de la rapidez de ejecución de una computadora.

módulo de evaluación (EVM): Tarjeta y herramientas de software que permite al usuario evaluar un dispositivo específico.

MIPS: Ver millones de instrucciones por segundo.

multiplexor: Dispositivo para seleccionar una de las señales disponibles.

multiplicador: Componente del CPU que multiplica el contenido de dos registros.

N

normalización: Reducción de una estructura de datos a su forma mas simple o en un circuito a un número reducido de compuertas.

O

optimizador de ensamblador: Software que optimiza el código ensamblador lineal, que es código ensamblado que no ha sido destinado por los registros o no ha sido programado. El optimizador de ensamblador se solicita automáticamente con el programa shell.

P

palabra: Conjunto de 32 bits que se almacenan, direccionan, transmiten, u operan como una unidad.

paquete de ejecución: Un grupo de instrucciones que se ejecutan en paralelo.

paquete de búsqueda: Series de 8 palabras contiguas de instrucciones buscadas por el CPU y alineadas en un límite de 8 palabras.

paquete de búsqueda de instrucción: Un grupo mayor de ocho instrucciones guardadas en memoria para ser ejecutadas por el CPU.

paquete de servicio de interrupción: Paquete usado para servicio de interrupción. Cuando ocho instrucciones no son suficientes, el usuario debe bifurcar este bloque para un servicio de interrupción adicional. Si los retardos de la bifurcación no están con el ISFP, la ejecución continua en el paquete subsiguiente.

paralelismo: Eventos secuenciales que ocurren simultáneamente. El paralelismo se lleva a cabo en un CPU por el uso de la segmentación encauzada.

periférico: Dispositivo conectado y usualmente controlado por un host.

pipeline: Ver segmentación encauzada.

procesador digital de señales (DSP): Un semiconductor que convierte señales analógicas (tales como sonido o voltaje) en señales digitales que son impulsos eléctricos discretos o discontinuos, y pueden ser manipulados.

procesador de punto fijo: Procesador que hace operaciones aritméticas usando aritmética entera sin exponentes.

procesador de punto flotante: Procesador capaz de manejar aritmética de punto flotante en el que los operandos se representan usando exponentes.

puerto serial multicanal con buffer McBSP: Chip con circuito de transmisión full duplex que proporciona comunicación serial por medio de algunos canales a dispositivos externos.

R

registro: Pequeña área en la memoria de alta rapidez, localizada en un procesador o un dispositivo, que se usa para almacenamiento temporal de datos o instrucciones. Cada registro tiene un nombre, contiene bytes de información con referencia en los programas.

registro de control: Registro que contiene campos de bits que definen la forma de operar de un dispositivo.

S

segmentación encauzada: Método de ejecución para instrucciones en el que la salida de un proceso sirve de entrada a otro, como una linea de ensamble. Esos procesos son las etapas o fases de la segmentación encauzada.

sobreflujo: Condición en la que el resultado de una operación aritmética excede la capacidad del registro usado para ese resultado.

T

timer: Periférico programable usado para generar pulsos o para contabilizar eventos.

transferencia dimensional 1: Un grupo de tramas constituyen un bloque 1D. El número de tramas en un bloque puede estar entre 1 y 65536. El número de elementos por trama puede estar entre 1 y 65535. Cada elemento o trama puede transferirse una vez.

transferencia dimensional 2: Un grupo de arreglos constituyen un bloque 2D. La primera dimensión es el número de elementos en el arreglo, y la segunda dimensión es el número de arreglos. El número de arreglos en un bloque puede estar entre 1 y 65536. Cada arreglo o bloque entero puede transferirse una vez.

T1: Servicio americano de comunicación entre redes de alta rapidez que opera a 1.544 M bits por segundo; usa la ley de compresión y expansión μ.

U

unidad aritmética y lógica ALU: El hardware del CPU que efectúa las funciones aritméticas y lógicas.

unidad central de procesamiento CPU: La unidad que coordina las funciones de un procesador.

unidad de búsqueda del programa: Hardware del CPU que recupera instrucciones del programa.

V

VelociTI: Arquitectura desarrollada por TI que usa instrucciones de palabra muy larga.

1D: Ver transferencia dimensional 1.

2D: Ver transferencia dimensional 2.

Bibliografía

[1] An Audio Example Using DSP/BIOS, Número: SPRA598, Texas Instruments, U.S.A, noviembre 1999.

[2] Code Composer Studio Getting Started Guide, Número: SPRU509C, Texas Instruments, U.S.A, noviembre 2001.

[3] Code Composer Studio IDE Quick Start, Número: SPRU405A, Texas Instruments, U.S.A, febrero 2001.

[4] Code Composer Studio Online Documentation, Número: SPRU415A, Texas Instruments, U.S.A, abril 2001.

[5] DSP/BIOS by Degrees: Using DSP/BIOS Features in an Existing Application, Número: SPRA591, Texas Instruments, U.S.A, diciembre 1999.

[6] DSP/BIOS Quick Start Reference Guide, Número: SPRU426, Texas Instruments, U.S.A, febrero 2001.

[7] TLC320AD535C/I Dual Channel Voice/Data Codec Data Manual, Número: SLAS202B, Texas Instruments, U.S.A, 2000.

[8] TMS320 DSP/BIOS User's Guide, Número: SPRU423A, Texas Instruments, U.S.A, noviembre 2001.

[9] TMS320C6000 Assembly Language Tools User's Guide, Número: SPRU186I, Texas Instruments, U.S.A, abril 2001.

[10] TMS320C6000 Chip Support Library API User's Guide, Número: SPRU401C, Texas Instruments, U.S.A, octubre 2001.

[11] TMS320C6000 CPU and Instruction Set Reference Guide, Número: SPRU189F, Texas Instruments, U.S.A, octubre 2000.

[12] TMS320C6000 DSK Board Support Library API User's Guide, Número: SPRU432A, Texas Instruments, U.S.A, octubre 2001.

[13] TMS320C6000 DSP/BIOS Application Programming Interface API Reference Guide Literature, Número: SPRU403D, Texas Instruments, U.S.A, noviembre 2001.

[14] TMS320C6000 DSP/BIOS User's Guide, Número: SPRU303B, Texas Instruments, U.S.A, noviembre 2001.

[15] TMS320C6000 Enhanced Direct Memory Access EDMA Controller Reference Guide, Número: SPRU234B, Texas Instruments, U.S.A, marzo 2005.

[16] TMS320C6000 Optimizing Compiler User's Guide, Número: SPRU187I, Texas Instruments, U.S.A, abril 2001.

[17] TMS320C6000 Peripherals Reference Guide, Número: SPRU190D, Texas Instruments, U.S.A, febrero 2001.

[18] TMS320C6000 Programmer's Guide, Número: SPRU198F, Texas Instruments, U.S.A, febrero 2001.

[19] TMS320C6000 Technical Brief, Número: SPRU197D, Texas Instruments, U.S.A, febrero 1999.

[20] Alan V. Oppenheim. Alan S. Willsky. Señales y Sistemas. Segunda edición. Prentice Hall. México, 1998.

[21] Alan V. Oppenheim, Ronald W. Schafer. Tratamiento de Señales en Tiempo Discreto. Segunda edición. Prentice Hall. España, 2000.

[22] Ashok Ambardar. Procesamiento de Señales Analógicas y Digitales. Segunda edición. Thomson Learning. México 2002.

[23] Barry B. Brey. Los Microprocesadores Intel, Arquitectura, Programación e Inerfaz de los Procesadores 8086/8088, 80186/80188, 80286, 80386, 80486, Pentium, Pentium Pro y Pentium II. Quinta edición. Prentice Hall. México, 2001.

[24] Bohumil Pšenička. Omar Nieto Crisóstomo. C6X-Based Digital Signal Processing. Facultad de Ingeniería. México, 2002.

[25] H.M. Deitel. P.J. Deitel. Como programar en C/C++. Segunda edición. Prentice Hall. México, 1995.

[26] James L. Antonakos. Kenneth C. Mansfield Jr. Programación Estructurada en C. Primera edición. España, 2004.

[27] John G. Proakis, Vinay K. Ingle. A Self Study Guide for Digital Signal Processing. Pearson Prentice Hall. U.S.A, 2004.

[28] José B. Maniño Acebal, Francesc Vallverdú Bayés, Tratamiento Digital de la Señal. Segunda edición. Alfaomega, México 1999.

[29] Nasser Kehtarnavaz. Burc Simsek. C6X-Based Digital Signal Processing. Prentice Hall. U.S.A, 2000.

[30] Ronald J. Tocci. Neal S. Widmer. Sistemas Digitales Principios y Aplicaciones. Prentice Hall. Octava edición. México 2003.

[31] Rulph Chassaning. Digital Signal Processing Laboratory Experiments Using C and the TMS320C31 DSK. John Wiley Sonns. U.S.A, 1999.

[32] Vinay K. Ingle, John G. Proakis. Digital Signal Processing Using Matlab. Thomson Learning. Canada, 2000.

[33] Vuelapluma. Diccionario Inglés-Español de Informática y Telecomunicaciones. Mc.Graw Hill. España, 2002.