Universidad Pontificia Bolivariana
SECCIONAL BUCARAMANGA

PROCESO DIDÁCTICO DE DESARROLLO Y ELABORACIÓN DE EXPERIENCIAS PARA LA IMPLEMENTACIÓN DEL SISTEMA DE DESARROLLO ADSP-2181 DSPKIT

JUAN JOSÉ CASTAÑEDA PEREIRA
NICOLE TATIANA VÁSQUEZ TARAZONA

UNIVERSIDAD PONTIFICIA BOLIVARIANA
ESCUELA DE INGENIERÍA Y ADMINISTRACIÓN
FACULTAD DE INGENIERÍA ELECTRÓNICA
BUCARAMANGA
2009

PROCESO DIDÁCTICO DE DESARROLLO Y ELABORACIÓN DE EXPERIENCIAS PARA LA IMPLEMENTACIÓN DEL SISTEMA DE DESARROLLO ADSP-2181 DSPKIT

JUAN JOSÉ CASTAÑEDA PEREIRA
NICOLE TATIANA VÁSQUEZ TARAZONA

Trabajo de grado para optar al título de
Ingeniero Electrónico

Director:
FABIO ALONSO GUZMAN SERNA
Ingeniero Electrónico especialista en Telecomunicaciones

UNIVERSIDAD PONTIFICIA BOLIVARIANA
ESCUELA DE INGENIERÍA Y ADMINISTRACIÓN
FACULTAD INGENIERÍA ELECTRÓNICA
BUCARAMANGA
2009

A Dios,
A nuestros Padres por su insistencia, y por ser nuestros más grandes maestros,
A nuestras madres, por sus oraciones y su apoyo,
A nuestras familias, por el ánimo que siempre nos brindaron,
Y a nuestros amigos, por qué nunca falto su sonrisa,
Gracias a todos por estar ahí siempre.

AGRADECIMIENTOS

Los autores expresan sus agradecimientos a:

Fabio Alonso Guzmán Serna, Ingeniero Electrónico Especialista en Telecomunicaciones. Por ser un gran director, en quien siempre pudimos encontrar el consejo sabio, la palabra oportuna, y un gran amigo. Gracias por creer y confiar ciegamente en nosotros.

Alonso de Jesús Retamoso Llamas, Ingeniero Electricista Magister en Potencia Eléctrica. Por su orientación, amplio conocimiento del tema, y colaboración en el desarrollo de este proyecto.

Carlos Gerardo Hernández Capacho, Ingeniero Electrónico. Por ser un pilar fundamental en el proceso de nuestra formación como futuros profesionales. Por ser una gran e incondicional amigo.

Juan Carlos Villamizar, Ingeniero Electricista. Por su ayuda en la definición de la metodología a seguir en este proyecto, además de ser una persona que siempre nos ha enseñado a ver la vida con alegría y entusiasmo.

Diana Paola Blanco Rendón, Ingeniera Mecatrónica. Porque este proyecto no hubiese podido realizarse si su presencia y constante apoyo en todo el proceso de desarrollo y elaboración del mismo. Por ser una gran ingeniería y una persona muy especial.

Angélica María Jerez Navas, Ingeniera Electrónica. Porque a lo largo de mis cinco años de carrera ella fue, ha sido, y será la compañera más importante que he tenido. Culminé mi carrera gracias a su constante soporte y apoyo, muchas gracias Angélica por tanto aguante, por tus reflexiones, entusiasmo, pero por encima de todo, gracias por ser tan paciente.

A todos los profesores que durante seis años han brindado lo mejor de sí para que esta generación de futuros ingenieros electrónicos sea la mejor.

A todos los compañeros de la carrera que siempre estuvieron en las malas notas como en las buenas.

CONTENIDO

TABLA DE FIGURAS

LISTA DE TABLAS

RESUMEN GENERAL DEL TRABAJO DE GRADO

TÍTULO:
PROCESO DIDÁCTICO DE DESARROLLO Y ELABORACIÓN DE EXPERIENCIAS PARA LA IMPLEMENTACIÓN DEL SISTEMA DE DESARROLLO ADSP-2181 DSPKIT

AUTOR(ES):
JUAN JOSÉ CASTAÑEDA PEREIRA
NICOLE TATIANA VÁSQUEZ TARAZONA

FACULTAD:
Facultad de Ingeniería Electrónica

DIRECTOR(A):
FABIO ALONSO GUZMAN SERNA

RESUMEN

Los avances que ha tenido la electrónica en el campo del procesamiento digital de señales han generado una intensa búsqueda de una herramienta donde la velocidad y la economía jueguen un papel fundamental. Viendo la necesidad de un sistema basado en un procesador que posea un hardware y un software optimizados para aplicaciones que requieran operaciones numéricas de muy alta velocidad, y a su vez sea una herramienta adecuada para el procesamiento y representación de señales analógicas en tiempo real, se ha desarrollado un proyecto que contiene el proceso didáctico para la implantación del sistema de desarrollo ADSP-2181 DSPKIT como la herramienta más adecuada para la solución de los problemas que se plantean ante los factores de velocidad de procesamiento y economía. El sistema de desarrollo ADSP-2181 DSPKIT es la herramienta idónea para fines académicos, debido a su economía y fácil programación. Es por estas razones que el proyecto fue desarrollado sobre este sistema. En este documento se muestra el proceso a seguir por cualquier estudiante para aprender el uso del sistema de desarrollo y su respectivo software. Además se brinda al estudiante la oportunidad de realizar una serie de prácticas de laboratorio para que realice un proceso de formación de conocimiento progresivo, inicializando desde los conceptos más elementales de la plataforma, hasta llegar a un desafío de diseño de alto nivel donde se mezclan los diferentes conocimientos adquiridos en las experiencias previas realizadas.

PALABRAS CLAVES: DSP, filtro FIR, Assembler, ADSP-218x, Microprocesador, Unidad Computacional.

GENERAL OVERVIEW OF PROJECT

TITLE:
DIDACTIC PROCESS OF DEVELOP AND ELABORATION OF EXPERIENCES FOR THE IMPLEMENTATION OF THE DEVELOP SYSTEM ADSP-2181 DSPKIT

AUTOR(S):
JUAN JOSÉ CASTAÑEDA PEREIRA
NICOLE TATIANA VÁSQUEZ TARAZONA

FACULTY:
Facultad de Ingeniería Electrónica

DIRECTOR:
FABIO ALONSO GUZMAN SERNA

ABSTRACT

The advances that have had the electronic in the field of digital signal processing have generated a big searching of a tool where the speed and the economics are the most important role. Because of the necessity of a system based in a Processor which contains an optimized hardware and software to applications that require numeric operations with a high level of velocity, and at the same time being a suitable tool for the processing and representation of Analog signals in real time, it's been made a project that contains the didactic process for the implantation of the ADSP-2181 DSPKIT develop system as the precise tool for the solution of problems that are introduced in front of the processing speed and economic issues. The ADSP-2181 DSPKIT develops system is the ideal tool for academic issues, because of its economic and easy programming. It is for these reasons that the project was developed in this system. In this document it's shown the process to be followed to learn the way of use of the develop system and its respective software. Besides, we give to students the opportunity of develop some laboratory practices to do a formation process of progressive knowledge, beginning from the basics concepts of the platform, until reach to a design challenge of high level where the different knowledge's are mixed.

KEY WORDS: DSP, FIR Filter, Assembler, ADSP-218x, Microprocessor, Computational Unit.

GLOSARIO

FILTROS FIR: tipo de filtro no recursivo que para una señal impulso de entrada, la salida tendrá un número finito de términos no nulos (Finite Impulse Response)

FILTROS IIR: tipo de filtro no recursivo que para una señal impulso de entrada, la salida tendrá un número infinito de términos no nulos, es decir, no vuelve al reposo (Infinite Impulse Response)

ADC (Conversor Análogo-Digital): dispositivo encargado de transformar señales análogas en una secuencia de números de precisión finita.

DAC (Conversor Digital-Análogo): dispositivo encargado de conectar todos los puntos de una señal digital.

DMA: acceso directo a memoria que permite a cierto tipos de componentes de ordenador acceder a la memoria del sistema para leer o escribir independientemente de la CPU principal.

DAG (Data Adress Generator): generadores de direcciones que simplifican la organización de los datos manteniendo punteros a la memoria

CODEC: abreviatura de compresor-descompresor. Describe una especificación desarrollada en software, hardware o una combinación de ambos, capaz de transformar un archivo con un flujo de datos o una señal.

EPROM: memoria de acceso aleatorio, generalmente leída y eventualmente borrada y reescrita.

JACK: conector de audio utilizado en numerosos dispositivos para la transmisión de sonido en formato analógico

JUMPER: elemento para interconectar dos terminales de manera temporal sin tener que efectuar una operación que requiera una herramienta adicional.

EMULADOR: software que permite ejecutar programas de computadora en una plataforma diferente a aquella par la cual fueron escritos originalmente.

DUPURAR: revisión de la aplicación generada con el fin de eliminar los posibles errores que puedan existir en esta.

ENLAZADOR: programa que toma los ficheros de código objeto generado en los primeros pasos de compilación, la información de todos los recursos necesarios, quita aquellos recursos que no necesita y enlaza el código objeto con su biblioteca con lo que finalmente produce un fichero ejecutable.

CACHE: conjunto de datos duplicados de otros originales o zona de memoria de disco.

PILA: lista ordinal o estructura de datos en la que el modo de acceso a sus elementos es de tipo “último en entrar, primero en salir” que permite almacenar y recuperar datos.

BUCLE: sentencia que se realiza repetidas veces a un trozo aislado de código, hasta que la condición asignada a dicho bucle deje de cumplirse.

TELEMETRÍA: conjunto de procedimientos para medir magnitudes físicas y químicas desde un punto distante al lugar donde se producen los fenómenos, cuando existen limitaciones de acceso

INTRODUCCIÓN

El procesamiento digital de señales es un área de la ciencia y la tecnología que ha tenido un avance significativo en las últimas tres décadas. En un comienzo los ordenadores digitales y el hardware asociado con el procesamiento de señales eran relativamente grandes y costosos. Con el paso del tiempo surgieron avances en la electrónica que permitieron el rápido desarrollo de la tecnología y a su vez el estudio de esta ciencia se hizo más exacto y completo, logrando de esta forma establecer un hardware apropiado y costeable que tuvo un gran impacto en la industria y en la sociedad.

Dichos avances en la tecnología abrieron nuevas áreas de desarrollo basadas en DSP, tales como sensores inteligentes, visión de robots y automatización, como a su vez dieron bases para continuar con los avances en las áreas tradicionales del procesamiento digital de señales, tales como música, voz, radar, sonar, video, audio y comunicaciones.

Existen múltiples fabricantes de DSPs entre los que se encuentran Motorola, Texas Instruments, AT&T y Analog Devices. Estos fabricantes producen diferentes tipos de DSPs dependiendo de las necesidades de la aplicación. Este proyecto hace énfasis en el sistema de desarrollo ADSP-2181 EZ KIT Lite de Analog Devices, otorgando al lector la información necesaria para entender y evaluar el hardware del sistema, dando una amplia descripción sobre el contenido, requisitos y características del sistema, así como también, se introduce al lector en el manejo del software de programación del mismo.

Por último se exponen una serie de guías didácticas que tienen como objetivo enseñar el lenguaje de programación del sistema y como manipular el software, además de, mostrar algunas aplicaciones básicas que tiene el sistema con el fin de dar al lector una breve visión de sus alcances.

1 MARCO TEÓRICO

1.1 HISTORIA DEL PROCESAMIENTO DIGITAL DE LA SEÑAL

Desde la segunda guerra mundial, si no es más temprano, técnicos han especulado sobre la aplicabilidad de técnicas digitales para el desarrollo de tareas del procesamiento de señales. Por ejemplo, a finales de los años 40, Bode Shanon y otros investigadores de los Laboratorios de Telefonía Bell discutieron la posibilidad de usar elementos de circuitos digitales para implementar funciones de filtrado. En este tiempo, infortunadamente no había un hardware apropiado disponible.

A mediados de los años 50, la teoría de control, basada parcialmente sobre trabajos de Hurewiez se había establecido como una disciplina, y el muestreo y los efectos espectrales fueron bien entendidos. Un número de herramientas matemáticas como la transformada Z, eran ahora usadas dentro de la comunidad de ingenieros electrónicos. La tecnología en ese tiempo, sin embargo, estaba solo disponible para tratar con problemas de control a baja frecuencia o problemas de procesamiento de señales sísmicas a baja frecuencia. A pesar de que los científicos sísmicos hacían notable el uso de conceptos de filtros digitales para resolver problemas, este no lo era, hasta que a mediados de los años 60 una teoría formal del procesamiento digital de la señal empezó a emerger. Durante este periodo, la ventaja de la tecnología de los circuitos integrados de silicona hicieron posibles sistemas digitales completos, pero manteniendo un alto costo.

La primera mayor contribución en el área de la síntesis del filtrado digital fue hecha por Kaiser de Laboratorios Bell. Su trabajo mostró como diseñar satisfactoriamente filtros usando la transformada bilinear. Por otro lado, hacia 1965 el famoso documento por Cooley y Turkey fue publicado. En este documento, la FFT (Transformada de Fourier) fue demostrada como una manera eficiente y rápida de realizar la DFT (Transformada discreta de Fourier).

En este tiempo, un mejor hardware apropiado para la implementación de filtros digitales fue desarrollado y circuitos costeables empezaron a estar disponibles comercialmente. Largos filtros FIR (respuesta a impulsos finitos) pudieron ahora ser implementados eficientemente, convirtiéndose consecuentemente en un serio competidor a los filtros IIR (respuesta a impulsos infinitos), habiendo mejorado las propiedades de los pasa bandas para un determinado número de retardos. Al mismo tiempo, nuevas oportunidades emergieron, ahora era posible lograr

variables en el tiempo y filtros adaptables y no lineales que no podían ser construidos utilizando las técnicas análogas convencionales.

En el área de los filtros adaptables, B Windrow es un nombre importante, especialmente cuando se habla acerca del algoritmo LMS (Least Mean Square). Windrow había también hecho significantes contribuciones en el área de las redes neuronales a inicios de los años 60 y 70.

1.2 ORIGEN DE LAS SEÑALES DE TIEMPO REAL

Señal es definida como cualquier cantidad física detectable o medible que varía en el tiempo y con la cual se llevan mensajes o información que generalmente tratan acerca del estado o comportamiento de un sistema.

Por su propia naturaleza, las señales están divididas en dos categorías, las señales análogas y las señales digitales. Las señales análogas son señales que varían continuamente en el tiempo, mientras que una señal digital no varía en forma continua sino que cambian en pasos o en incrementos discretos.

En el caso especifico de un procesador digital de la señal (DSP), la señal análoga es convertida a una forma binaria por medio de un conversor análogo/digital. (ADC). La salida del ADC muestra una representación binaria de la señal análoga y esta es manipulada aritméticamente por medio de un procesador de señales. Después de procesada la señal, la información resultante debe ser convertida nuevamente a una señal continua o análoga mediante el uso de un conversor digital/análogo (DAC).

1.3 PROCESAMIENTO DE SEÑALES EN TIEMPO REAL

El procesamiento digital de la señal es una operación o transformación de una señal con un hardware digital sometido a reglas definidas las cuales son introducidas a él mediante un software. El procesamiento de señales en tiempo real tiene varias razones, una de ellas es el lograr obtener información de las señales procesadas. Esta información existe normalmente en la forma de la amplitud de la señal ya sea esta relativa o absoluta, en el contenido frecuencial o espectral, en la fase o en la relación de tiempo que tiene con otras señales, entre

otras; otra razón es comprimir la frecuencia contenida en la señal buscando no perder información significante en el proceso.

Una definición más estricta, se refiere al procesamiento digital de la señal como un procesamiento electrónico de dos señales (sonido, radio, microondas, etc.) haciendo uso de técnicas matemáticas para realizar transformaciones o extraer información.

La adquisición de datos en la industria y los sistemas de control hacen uso de la información extraída de los sensores para desarrollar apropiadas señales de retroalimentación con la cuales se controla el proceso por sí mismo. Se debe notar que estos sistemas necesitan de la implementación de conversores análogos/digital y digital análogos así como también el uso de sensores, acondicionadores de señal y por ultimo un DSP (o microprocesador).

1.4 MÉTODOS Y TÉCNICAS DISPONIBLES PARA EL PROCESAMIENTO DE SEÑALES EN TIEMPO REAL

Las señales pueden ser procesadas usando técnicas análogas (ASP o procesamiento análogo de la señal), técnicas digitales (DSP o procesamiento digital de la señal) o una combinación de técnicas análogas y digitales (MSP o procesamiento combinado de señales). En algunos casos la técnica a utilizar se hace evidente, por el contrario, en otros, no existe claridad en la elección y existen consideraciones de segundo orden que pueden ser usadas para hacer una elección final.

Con respecto al DSP, el factor que lo distingue del procesamiento o análisis computacional tradicional de datos es la velocidad y la eficiencia en desarrollar funciones sofisticadas del procesamiento digital tales como el filtrado, el análisis FFT y la compresión de datos en tiempo real.

El termino procesamiento combinado de señales (MSP) dice que el procesamiento digital así como el análogo son involucrados en el sistema. De acuerdo a esta definición, los conversores análogos/digitales y los conversores digitales/análogos son considerados como procesadores combinados de señales, siempre y cuando las funciones análogas y digitales sean implementadas en cada uno.

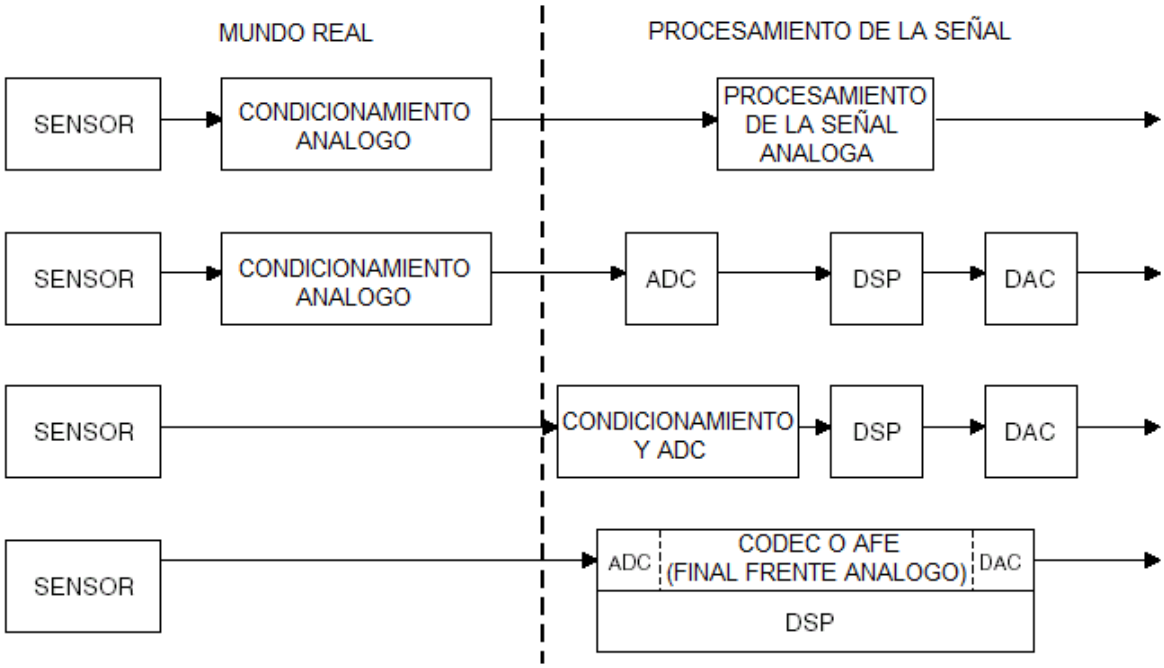
Hoy en día, el ingeniero se enfrenta a un desafío para seleccionar la técnica de procesamiento adecuada (combinada, análoga o digital). Es imposible procesar señales análogas en tiempo real mediante el uso exclusivo de técnicas digitales; por esta razón, todos los sensores, incluyendo micrófonos, termocuplas, galgas

extensiométricas, cristales piezoeléctricos y discos duros son sensores análogos. Por lo tanto, algún tipo de circuito acondicionador de señal es necesario a fin de preparar la salida de sensor. Los circuitos acondicionadores de señal son procesadores de señal análogos, donde se desarrollan funciones tales como la multiplicación, aislamiento, detección de presencia de ruido, rango dinámico de compresión y filtrado.

Varios métodos para lograr el procesamiento de la señal son mostrados en la figura 1. La parte superior de la figura muestra el método puramente análogo. La parte siguiente muestra el método DSP. Note que en cada uno de los casos se ha establecido el uso de técnicas DSP.

Por lo general el conversor análogo/digital es ubicado cerca al sensor actual, permitiendo que la señal análoga condicionada sea cargada al conversor. La alta complejidad del conversor puede lograr el incremento de la tasa de muestreo, una mayor resolución, un rechazo al ruido en la entrada, un filtrado en la entrada y una ganancia programable en los amplificadores, ventajas que añades funcionalidad y una simplificación en el sistema

Figura 1. Opciones del procesamiento de señal análogo o digital



Fuente: Los autores del proyecto.

1.5 PROCESADOR DIGITAL DE LA SEÑAL DSP

Un procesador digital de la señal (DSP) es un tipo de microprocesador muy rápido y poderoso. Un DSP se considera único debido a que es capaz de procesar señales en tiempo real. Esto logra que estos dispositivos sean ideales para aplicaciones que no permiten ningún tipo de retardo. . Por ejemplo, no es fácil conversar a través de un teléfono celular cuando existe un retardo en la línea. Esto lleva a que la señal se corte o a confusión ya que ambos usuarios hablan a la vez. Con los teléfonos celulares actuales, los cuales usan DSP's, es posible hablar normalmente. El DSP dentro del teléfono procesa el sonido (convirtiéndolo de una señal analógica a digital, filtrando, comprimiendo y realizando otras tareas en forma digital) tan rápidamente que uno puede hablar y escuchar sin problemas de retardo ni ninguna molestia que ello implica. O sea, se procesa en tiempo real.

Las aplicaciones de los DSP´s trabajan señales tales como sonido y ondas de radio las cuales tienen un origen análogo. Los computadores por otra parte, trabajan la información discontinuamente, como una serie de números binarios, por lo que se hace necesario desde un comienzo transformar las señales análogas a digitales.

Una vez terminada la etapa de transformación análogo/digital, los datos son entregados al procesador (DSP) , eventualmente el DSP deberá devolver los datos ya procesados razón por la cual es necesario en la etapa final la implementación de un conversor digital/análogo para que transforme la señal obtenida. Un ejemplo claro es el procesamiento de una señal de audio, después de adquirida (ADC) puede ser filtrada para eliminar el ruido, los crujidos de la estática, amplificar ciertas frecuencias de interés y eliminar otras, etc. Luego de esto la información es devuelta a través de la conversión digital/análoga (DAC).

En su interior, un DSP es altamente numérico y repetitivo. Cada vez que cada datos llega, este debe ser sumado, multiplicado y además transformado de acuerdo a formulas numéricas. La velocidad del dispositivo es la causa de que dichas operaciones se puedan realizar. Los sistemas basados en DSP´s deben trabajar en tiempo real capturando y procesando la información a la misma vez que esta está ocurriendo. El DSP también debe actuar rápido para no perder la información que le llega desde el ADC y además cumplir con el adecuado procesamiento de las señales.

Una de las más importantes características de un DSP es su capacidad de realizar operaciones de multiplicación y acumulación (MACs) en sólo un ciclo de reloj. No obstante ello, es necesario que el dispositivo posea la característica de manejar aplicaciones críticas en tiempo real. Esto requiere de una arquitectura que soporte un flujo de datos a alta velocidad hacia y desde la unidad de cálculo y memoria. Esta ejecución a *Menu*do requiere el uso de unidades DMA (Direct Memory Acess)

y generadores de direcciones duales (DAG's) que operan en paralelo con otras partes del chip. Los DAG's realizan los cálculos de direcciones, permitiendo al DSP buscar dos datos distintos para operar con ellos en un solo ciclo de reloj, de tal forma que es posible ejecutar algoritmos complejos en tiempo real.

1.6 OBJETIVOS

1.6.1 Objetivo general.

Diseñar e implementar diferentes experiencias didácticas para el aprendizaje del modo de funcionamiento del sistema de desarrollo ADSP-2181 DSPKit.

1.6.2 Objetivos específicos.

- Recopilar, analizar e incorporar el conocimiento correspondiente a la teoría del procesador digital de señales.
- Aprender el hardware y software del sistema de desarrollo ADSP-2181 DSPKit.
- Entender el flujo de diseño para aplicaciones sobre el sistema de desarrollo ADSP-2181 DSPKit, desde el establecimiento del problema, hasta la implementación sobre la tarjeta.
- Implementar por medio del software propuesto los algoritmos de aplicación en el lenguaje del sistema de desarrollo ADSP-2181 DSPKit.
- Elaborar las guías o instructivos de cada experiencia

2 INTRODUCCIÓN AL SISTEMA DE DESARROLLO ADSP-2181 EZ-KIT LITE

En este capítulo se encontrara la información necesaria para comenzar a utilizar el sistema de desarrollo ADSP-2181 EZ-KIT Lite.

2.1 CONTENIDO DEL EZ-KIT Lite

El paquete de sistema de evaluación ADSP-2181 EZ-KIT Lite contiene los siguientes ítems:

- Tarjeta ADSP-2181 EZ-KIT Lite.
- Cable de alimentación para fuente de alimentación de 8-10V DC.
- Puerto serial RS-232 cable de 9-pines.
- CD que contiene:
    - Visual DSP + + para procesador de 16-bit con una licencia limitada.
    - Software para el ADSP-2181 EZ-KIT Lite.
    - Ejemplos de programas.
    - Manual del Sistema de Evaluación ADSP-2181 EZ-KIT Lite.

2.2 REQUISITOS DEL SISTEMA

Para la correcta operación del software VisualDSP++ y el EZ-KIT Lite, el equipo debe tener como configuración mínima mostrada en la tabla 1.

Tabla 1. Requisitos del Sistema

Windows® 98, ME, 2000, XP.	Windows NT TM
Windows 98, ME, 2000, XP.	Windows NT 4.0, o posterior.
Procesador Pentium 166MHz o más rápido.	Procesador Pentium 166MHz o más rápido.
100 MB de espacio disponible.	100 MB de espacio disponible.
16 MB de RAM.	16 MB de RAM.
Monitor VGA de color y tarjeta de vídeo.	Monitor VGA de color y tarjeta de vídeo.
CD-ROM.	CD-ROM.

2.3 PROCEDIMIENTO DE INSTALACIÓN

Para el uso seguro y eficaz del ADSP-2181 EZ-KIT Lite se ha previsto la siguiente lista de instrucciones. El usuario debe seguir estas instrucciones a fin de garantizar el correcto funcionamiento del software y hardware.

- Configuración del hardware EZ-KIT Lite
- Instalación del software Visual DSP + + y EZ-KIT Lite.
- Instalación de la licencia del Visual DSP + +.

2.3.1 Configuración del Ez-Kit lite hardware. La tarjeta ADSP-2181 EZ-KIT Lite está diseñada para ejecutarse fuera de su equipo como una unidad independiente. Para conectar la tarjeta EZ-KIT Lite el usuario debe seguir las siguientes instrucciones:

Retirar la tarjeta EZ-KIT Lite del paquete. Se debe tener cuidado con la manipulación de la tarjeta para evitar la descarga de electricidad estática, la cual puede dañar algunos componentes.

Conectar un extremo del cable RS-232 a un Puerto COM disponible en el PC y el otro extremo en J3 en la tarjeta de desarrollo ADSP-2181.

Conectar el cable provisto en una toma de 120 voltios AC y enchufar el otro extremo del cable al conector j4 en la tarjeta de evaluación.

Comprobar visualmente que todos los LEDs se iluminan brevemente. El LED de comprobación de energía (verde) permanece encendido y FL1 parpadea. Si el LED no se enciende es necesario comprobar la conexión de energía. Para

configurar la tarjeta y tomar ventaja de las capacidades de audio de los demos de la tarjeta, utilizar el siguiente procedimiento:

Conectar un conjunto de altavoces de computador auto-alimentados en el jack J1 de la tarjeta. Encender los altavoces y ajustar el volumen a un nivel adecuado.

Conectar la línea de salida de un dispositivo electrónico de audio al jack J2 en la tarjeta. Fijar el jumper JP2 en modo LINE.

Abrir Jumper JP2 a GND para habilitar al AD1847 códec. (Esta se encuentra en la tarjeta por defecto).

2.3.2 Instalación del software VisualDSP++ y Ez-Kit lite. Este EZ-KIT Lite viene con la versión más reciente del VisualDSP++ para procesadores de 16-bits. La instalación del VisualDSP++ incluye las instalaciones del EZ-KIT Lite. Para instalar VisualDSP++ y EZ-KIT Lite software se deben seguir las siguientes instrucciones:

Insertar el CD de instalación del VisualDSP++ en la unidad de CD-ROM.

Si está habilitada la reproducción automática en el PC se observara el pantallazo del asistente de instalación *Install Shield Wizard Welcome*, de lo contrario, seleccionar Ejecutar del menú Inicio y escribir:
D: \ ADI_Setup.exe en el campo abrir, donde D es el nombre de la unidad local de CD-ROM.

Seguir las instrucciones mostradas en pantalla para continuar con la instalación del software.

En la pantalla *Customs Setup*, seleccionar el EZ-KIT Lite de la lista de los sistemas disponibles y elegir el directorio de instalación, hacer clic sobre el icono en el campo *Feature* Description, para ver descripción del sistema seleccionado. Cuando haya terminado, haga clic en Siguiente.

En la pantalla *Ready to Install*, hacer clic en *Back* para cambiar las opciones de instalación, hacer clic en *Install* para instalar el software, o hacer clic Cancel para salir de la instalación.

Cuando el EZ-KIT Lite se instale, la pantalla *Wizard* Complete aparecerá. Hacer clic en Finalizar.

2.3.3 Instalación de la licencia del VisualDSP++. Para instalar la licencia del VisualDSP++ se deben seguir las siguientes instrucciones:

Localizar el número de serie proporcionado en la calcomanía adherida al CD de instalación y el formulario de inscripción.

Desde el menú *Star*, seleccionar *Programs/Analog Devices/VisualDSP++ 3,5 for 16-bit Processors/VisualDSP++ Environment.*

La información en pantalla preguntará si se desea instalar una licencia. Hacer clic en *Yes*. La pantalla *About VisualDSP++* aparecerá.

Seleccionar la pestaña *Licences* y hacer clic en *New*.

En el dialogo *Install* a *New License* que se abre, seleccionar *Single User*.

Notar que el software VisualDSP++ que viene con el *EZ-KIT++ Lite* es una versión demo que limita el tamaño del archivo ejecutable a 8 Kbytes. Esta licencia permite ejecutar sesiones de *EZ-KIT++ Lite* solamente. La simulación y emulación no son soportadas.

Llenar el número serial en el campo provisto exactamente como aparece en la carátula del CD o en el formulario de registro y hacer clic en Yes. Una ventana de información avisa del éxito en la instalación de la licencia.

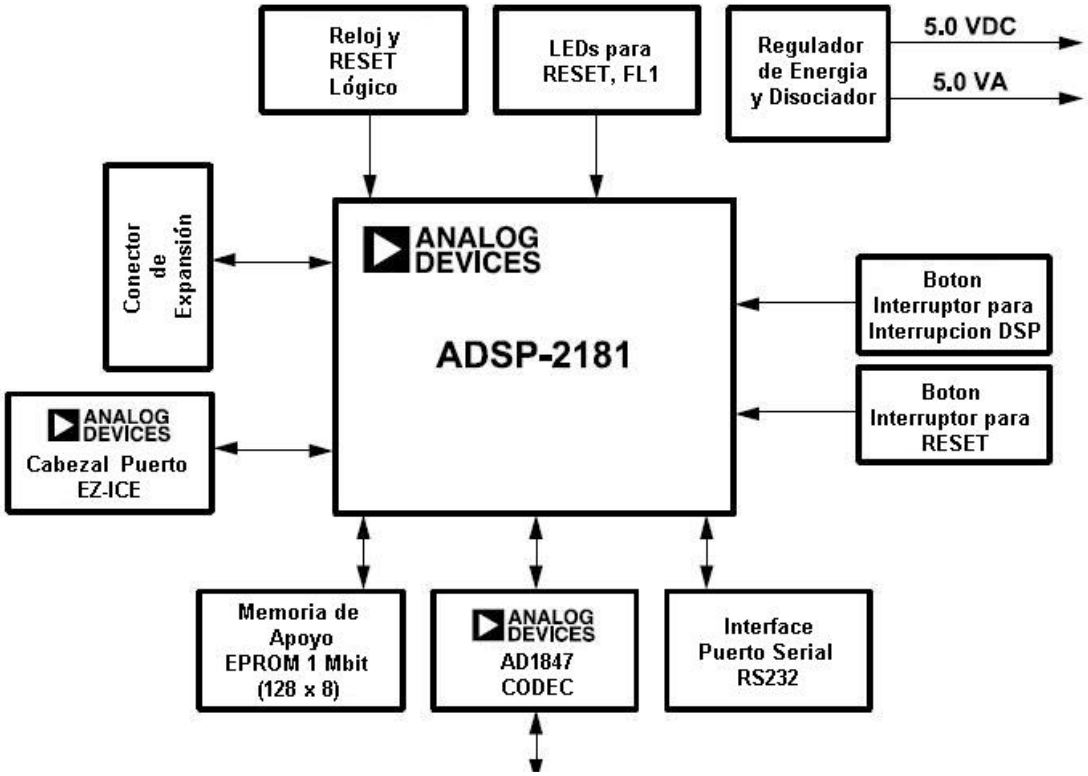
3 REFERENCIA DEL HARDWARE EZ-KIT

En este capítulo se analiza el hardware de diseño de la tarjeta ADSP-2181 EZ-KIT

3.1 ARQUITECTURA DEL SISTEMA

La figura 2 describe la configuración del procesador en la tarjeta del *EZ-KIT Lite*.

Figura 2. Diagrama de bloques del Sistema *ADSP-2181 EZ KIT Lite*

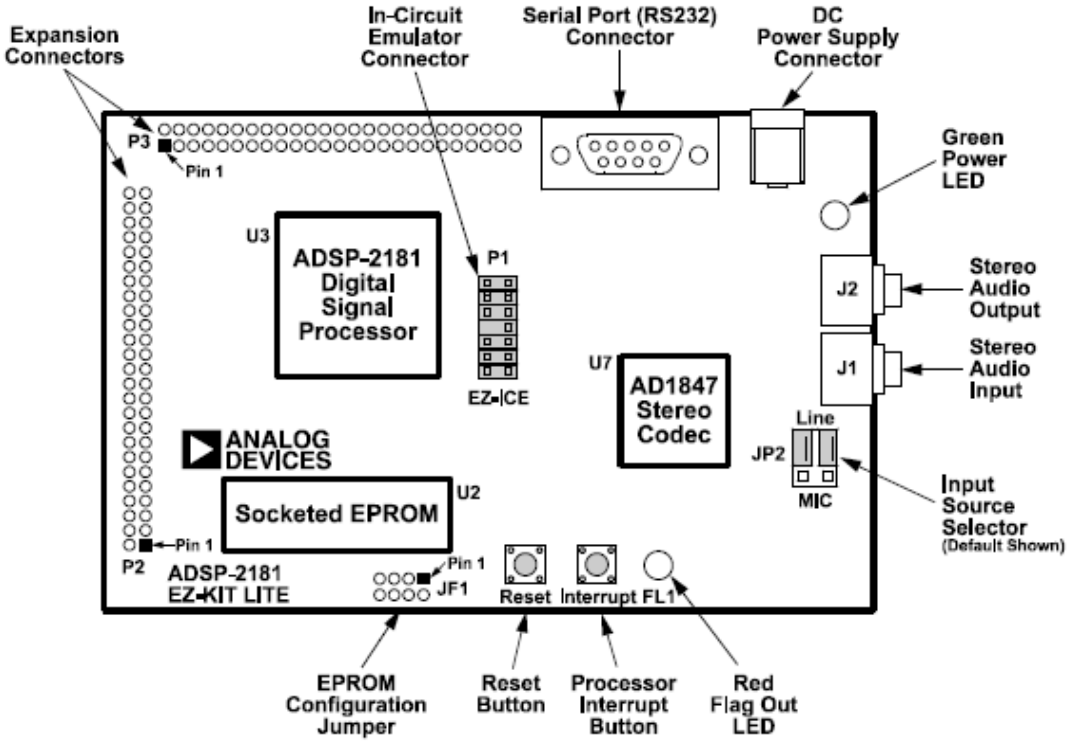


Fuente: Los autores del proyecto.

3.2 DISEÑO DE LA ARQUITECTURA DE LA TARJETA

La figura 3 muestra el diseño de la tarjeta EZ-KIT Lite. Esta figura orienta sobre la ubicación de los principales componentes y conectores.

Figura 3. Diseño de la tarjeta EZ-KIT Lite



Fuente: ADSP-2181 EZ-KIT Lite Evaluation System Board, cp.3-3.

3.2.1 Memoria desasociada del Ez-Kit Lite. La memoria EPROM proporciona hasta 128k x 8 bits de almacenamiento de programa que pueden ser cargados por el procesador de ADSP-2181 cuando se está programando para arrancar desde la memoria EPROM. Después el procesador de ADSP-2181 se restablece, la característica BDMA se utiliza para cargar las primeras 32 palabras de la memoria del programa desde el espacio de memoria byte. La ejecución del programa se realiza hasta que todas las 32 palabras estén cargadas.

3.2.2 LEDs de usuario. El LED D1 es un diodo emisor de luz roja, el cual está controlado por la salida FL1 del procesador ADSP-2181. El Software puede controlar el estado de este indicador escribiendo un registro interno. El LED D2 es un diodo emisor de luz verde, el cual esta encendido siempre que la tarjeta tenga energía.

3.2.3 Interruptores de la tarjeta. El interruptor S1 es el botón de *RESET*. Al pulsar este botón se activa el procesador ADSP-2181 y el códec AD1847, para entrar en el hardware, restablecer el estado y permanecer allí hasta que sea liberado. Los interruptores de salida son devueltos electrónicamente para prevenir múltiple transiciones debidas a los rebotes por contacto mecánico.

El interruptor S2 es el botón de interrupción. Al pulsar este botón el ADSP-2181 recibe una interrupción de entrada IRQE. El procesador entonces, ejecuta el actual interruptor manejador del software IRQE, si la interrupción está habilitada y el vector de interrupción IRQE está en su lugar. . Los interruptores de salida son devueltos electrónicamente para prevenir múltiple transiciones debidas a los rebotes por contacto mecánico.

3.3 CONECTORES

En esta sección se describen las cabeceras y los conectores suministrados con el ADSP-2181 EZ-KIT Lite.

El conector J2 es también un jack estéreo de 1/8 pulgadas (3,5 mm). Este jack se utiliza para llevar fuera la línea de nivel de audio de las señales de la tarjeta.

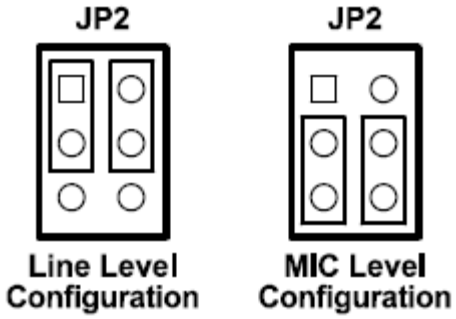
El conector J3 es un conector hembra de 9-pines. Se utiliza para comunicarse con un equipo host usando niveles de señales de RS-232 y protocolos seriales asíncronos.

El conector J4 es un jack de enchufe cilíndrico de 5,5 mm. Se utiliza para el suministro de energía a la tarjeta la cual se alimenta de voltajes DC. El pin central del jack es de 2 mm de diámetro y se debe conectar con el lado negativo de la fuente de energía.

El jumper JP1 es un sitio para un cabezal de ocho pines. Se puede utilizar para configurar la tarjeta por tamaños EPROM distintos que los EPROM (27C010) de 1 Mbit (128K byte) enviados con la tarjeta.

El jumper JP2, mostrado en la figura 4, es un cabezal de seis pines. Se utiliza para configurar la entrada al jack, J1, ya sea para nivel de línea o entrada de micrófono. El pin del centro en cada grupo de tres es conectado a uno de los pines de entrada del códec AD1847. Los *jumpers* (también conocidos como derivaciones o enlaces de cortocircuito) pueden ser usados para conectar estos pines, ya sea a la salida del micrófono o amplificador y a la salida de la línea de nivel del filtro de entrada.

Figura 4. Configuración jumper JP2



Fuente: ADSP-2181 EZ-KIT Lite Evaluation System Board, cp.3-6.

El conector P1 es un conector de cabezal de 14-pines utilizado para conectar a un ADDS-218x EZ-ICE® un circuito emulador. El pin 7 debe ser removido por efectos de introducción.

Los conectores P2 y P3 son *pads* para conectores de 50 pines. Estos conectores se pueden utilizar para acceder a las señales del procesador ADSP-2181 para expansión o propósitos de prueba.

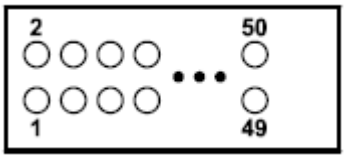
El socket U2 es un socket tipo DIP para una EPROM. Según su construcción, la tarjeta acepta un 27C512 (64K bytes) o 27C010 (128K byte). El cambio de conexiones en JP1 permite a la tarjeta aceptar un 27C256 (32K bytes), 27C020 (256K byte), 27C040 (512K byte), o 27C080 (1 Mb). Este socket está conectado a la interface de memoria del ancho de byte del procesador ADSP-2181

El pad marcado como R28 se emplea para ubicar un resistor de cero ohm. Esta resistencia es instalada en el procesador ADSP-2181 y puede restablecer la tarjeta bajo el software control. El software confirmaría el Reset por la configuración de la bandera PF0 como una salida.

El *pad* marcado como R29 es otro sitio para una resistencia de cero ohm. Si esta resistencia es instalada y X3 y C37 son removidos, el códec puede operar fuera con la señal CLKOUT del procesador ADSP-2181 en lugar de su propio reloj de 24576 MHz. También es necesario cambiar X1 a un valor de menor frecuencia para permanecer dentro de los rangos del códec.

3.3.1 Conectores del puerto de expansión. Mediante los *pads* de expansión (P2 y P3) se accede al bus de señales del procesador ADSP-2181, los cuales permiten ver la transmisión de datos. Además, la interfaz de host, interrupciones, y los pines del PWM_EVENT también están disponibles en dichos *pads*. La tabla 2 describe las señales disponibles en los pines de P2 y P3.

Figura 5. Pad de Expansión



Fuente: ADSP-2181 EZ-KIT Lite Evaluation System Board, cp.3-7.

3.3.2 Conector de alimentación. La tabla 3 resume los pines de salida del conector de energía. Si no se utiliza la fuente de alimentación suministrada con la tarjeta EZ-KIT Lite, sustituirla por una que tenga las conexiones indicadas en la tabla.

3.3.3 Conectores del códec AD1847. Cuando el códec AD1847 está habilitado en la tarjeta EZ-KIT Lite, se puede acceder a los jack de entrada y salida de audio en la tarjeta. Cada uno de los conectores de audio son jacks mini estéreo y aceptan el estándar disponible comercialmente en los enchufes de mini estéreo. El jack *Microphone/Line_in Input* se conecta a los pines del *Line_in_L (izquierda) y Line_in R (derecha)* o los MIC1 y MIC2 del códec AD1847 *SoundPort Stereo*, dependiendo de la configuración del jumper JP2. El jack LINE Output se conecta a los pines de la izquierda *(L) LINE_OUT* y de la derecha *(R) LINE_OUT* del códec.

Tabla 2. Conectores de Expansión

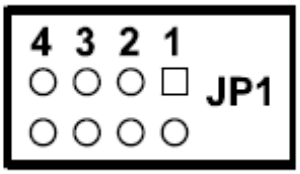
P2				P3			
Numero Del Pin	Nombre de La señal	Numero Del Pin	Nombre de La señal	Numero Del Pin	Nombre de La señal	Numero Del Pin	Nombre de La señal
1	A0	2	A1	1	GND	2	IAD0
3	A2	4	A3	3	IAD1	4	IAD2
5	A4	6	A5	5	IAD3	6	IAD4
7	A6	8	A7	7	IAD5	8	IAD6
9	A8	10	A9	9	IAD7	10	IAD8
11	A10	12	A11	11	IAD9	12	IAD10
13	A12	14	A13	13	IAD11	14	IAD12
15	D0	16	D1	15	IAD13	16	IAD14
17	D2	18	D3	17	IAD15	18	GND
19	D4	20	D5	19	IACK	20	IAL
21	D6	22	D7	21	IS	22	IWR
23	D8	24	D9	23	IRF	24	GND
25	D10	26	D11	25	PF0	26	PF1
27	D12	28	D13	27	PF2	28	PF3
29	D14	30	D15	29	PF4	30	PF5
31	D16	32	D17	31	PF6	32	PF7
33	D18	34	D19	33	FL0	34	FL1
35	D20	36	D21	35	FL2	36	CLK-OUT
37	D22	38	D23	37	RESET	38	IRQL0
39	WR	40	RD	39	IRQL1	40	IRQL2
41	IOMS	42	BMS	41	PWD	42	PWDACK
43	DMS	44	CMS	43	PWD	44	TXD0
45	PMS	46	BG	45	TFS0	46	RFS0
47	BGH	48	BG	47	RXD0	48	SCK0
49	VCC	50	GND	49	VCC	50	GND

Tabla 3. Conector de Alimentación

Terminal	Conexión
Centro de pin	8- 10V DC @ 300 mA
Anillo exterior	Positivo

3.3.4 Configuración del jumper de la EPROM. El jumper JP1 permite a la tarjeta ADSP-2181 EZ-KIT Lite ser configurada para cada uno de los seis diferentes tamaños de la EPROM. Por defecto, puede ser cualquiera, ya sea un 21C512 o 27C010. Si algún otro tamaño de la EPROM es instalado en el socket de U2, es necesario cambiar las conexiones en JP1, como se muestra en la figura 6.

Figura 6. Configuración del jumper JP1

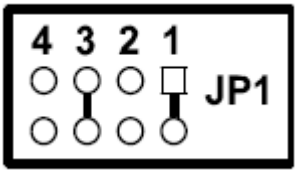


Fuente: ADSP-2181 EZ-KIT Lite Evaluation System Board, cp.3-10.

Las conexiones son hechas verticalmente entre los orificios destinados para el puerto de expansión. El par de orificios debajo de cada número constituye la posición del jumper asociado con ese número. Las conexiones se pueden realizar de varias maneras.

Las conexiones para la EPROM 27C256 debe ser similar a las mostradas en la figura 7.

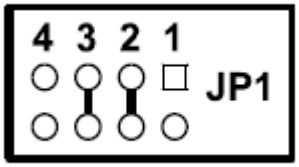
Figura 7. Configuración de los puentes JP1 para la EPROM 27C256



Fuente: ADSP-2181 EZ-KIT Lite Evaluation System Board, cp.3-11.

Las conexiones para una EPROM 21C512 o 27C010 son similares, se pueden observar en la figura 8.

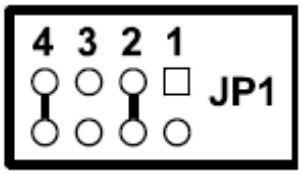
Figura 8. Configuración de los puentes JP1 para la EPROM 21C512 o 27C010



Fuente: ADSP-2181 EZ-KIT Lite Evaluation System Board, cp.3-11.

Las conexiones para una EPROM 27C020, 27C040, 27C080, se pueden observar en la figura 9.

Figura 9. Configuración de los puentes JP1 para la EPROM 27C020, 27C040, 27C080



Fuente: ADSP-2181 EZ-KIT Lite Evaluation System Board, cp.3-11.

3.3.5 Conector EZ-ICE. EL emulador ADSP-218x EZ-ICE ayuda a la depuración de un sistema ADSP-2181. El emulador consiste de hardware, del software residente del equipo host, y el conector de la tarjeta. La ADSP-2181 integra en el chip para el apoyo a la emulación una interface ICE-Port de 14-pines. Esta interfaz proporciona una conexión simple al objetivo de la tarjeta, que requiere menos consideraciones de remoción mecánica que otras ADSP-2100-EZ CIEM.

El dispositivo ADSP-2181 no necesita ser removido del sistema de destino cuando se está utilizando el EZ-ICE, ni tampoco son necesarios los adaptadores. Debido a la pequeña huella del conector EZ-ICE, la emulación puede ser soportada en los diseños finales de la tarjeta.

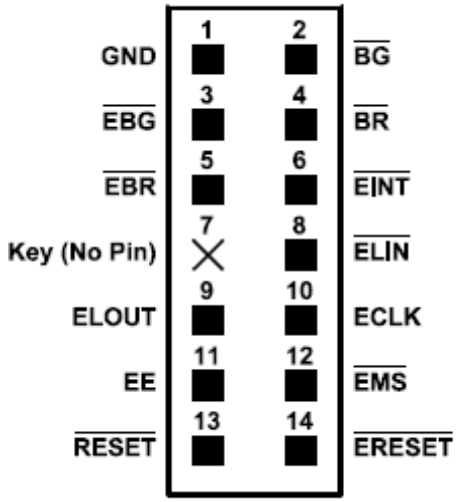
El EZ-ICE realiza y ofrece las siguientes opciones:

- Operaciones en la tarjeta.
- Configuración de hasta 30 puntos de interrupción
- Intensificación de la velocidad plena de operación.
- Examinar y modificar los registros de la memoria y los valores de carga y descarga de las funciones del PC.

- Emulación de la instrucción de nivel del programa de arranque y ejecución.
- Completar el montaje y desmontaje de instrucciones.
- Depuración de la fuente - nivel C.

El EZ-ICE es un encabezado de 14 pines con las conexiones que se indican en la figura 10.

Figura 10. EZ-ICE cabezal de 14 pines



Fuente: ADSP-2181 EZ-KIT Lite Evaluation System Board, cp.3-11.

4 INTRODUCCIÓN AL VISUAL DSP++

4.1 CARACTERÍSTICAS DEL VisualDSP++

El VisualDSP++ incluye todas las herramientas necesarias para construir y desarrollar proyectos en DSP.

4.1.1 Ambiente integrado de desarrollo y de depuración. El VisualDSP++ es un ambiente integrado de desarrollo y de depuración de proyectos, que proporciona gráficas de control de edición, construcción y proceso de depuración. En este entorno integrado, es posible moverse fácilmente entre la edición, la construcción, y las actividades de depuración.

4.1.2 Herramientas de desarrollo de código. Dependiendo del las herramientas de desarrollo DSP que se hayan comprado, VisualDSP++ incluye uno o más de los siguientes componentes:

- Compilador C/C++ con librerías de tiempo de ejecución.
- Ensamblador, linker, preprocesador, y archivador.
- Cargador y separador.
- Simulador.
- Sistema de desarrollo Ez-Kit lite

El VisualDSP++ da soporte a todos los formatos de archivos ejecutables producidos por el enlazador. Si el sistema está configurado con herramientas de desarrollo de terceros, el usuario puede seleccionar el compilador, ensamblador, ó enlazador para construcción en un dispositivo particular.

4.1.3 Características de edición del archivo fuente. El VisualDSP++ simplifica tareas que involucren archivos de origen. El usuario puede realizar fácilmente todas las actividades necesarias para crear, visualizar, imprimir, mover en el interior, y localizar la información.

4.1.3.1 Editar archivos de texto. Para crear y modificar archivos de código fuente, y ver el listado o mapa de los archivos generados por las herramientas de desarrollo de código del DSP. Los archivos fuente son en lenguaje C/C ++, ó archivos en lenguaje ensamblador que se toman para el proyecto. Los proyectos DSP pueden incluir archivos adicionales tales como archivos de datos y archivos .LDF, que contienen los comandos de entrada para el enlazador.

4.1.3.2 Editor de ventanas. Permite abrir múltiples ventanas de edición para ver y editar los archivos relacionados, o abrir varias ventanas para solo archivo. El editor de VisualDSP++ es una herramienta que permite enfocarse en el desarrollo de código.

4.1.3.3 Especificar color de las sintaxis. Las opciones de configuración que especifican el color de los objetos de texto vistos en una ventana de edición. Esta característica mejora la vista y ayuda a localizar partes del texto, citas, y comentarios, que aparecen en distintos colores.

4.1.3.4 Sensible al contexto de evaluación de expresión. Al mover el puntero del mouse sobre una variable que se encuentre en el entorno se logra observar el valor de la variable.

4.1.3.5 Íconos de estado. Visualización de íconos que indican puntos de interrupción, marcadores, y la posición actual del PC.

4.1.3.6 Visualizar detalles de errores y código erróneo. Desde la ventana de salida de la opción construir, se muestran los detalles resaltando el código de error (como cc0251) y pulsando la tecla F1. Haga doble clic en un error para así saltar a la línea donde está el error cometido en el código en una ventana de edición.

4.1.4 Características de administración de proyectos. El VisualDSP++ proporciona flexibilidad en la gestión de proyectos para el desarrollo de aplicaciones DSP, incluyendo el acceso a todas las actividades necesarias para crear, definir y construir proyectos en el DSP.

4.1.4.1 Definir y gestionar proyectos. Identifica archivos que la herramienta de desarrollo de códigos utiliza para construir el proyecto. Cree el proyecto una vez, o modifíquelo para poder cambiar lo necesario para el desarrollo.

4.1.4.2 Acceso y gestión de herramientas de desarrollo de código. Configura las opciones para especificar la forma en que las herramientas de desarrollo de código del DSP procesan entradas y generan salidas.

4.1.4.3 Vista y resultados de la construcción del proyecto. Ver el estado de la construcción del proyecto mientras que avanza y, de ser necesario, detener la construcción. El usuario debe hacer doble clic en el mensaje de error en la ventana de salida para ver el archivo fuente que ha causado el error, o alternar a través de los mensajes de error.

4.1.4.4 Administrar los archivos de origen. Administra los archivos de origen y realiza un seguimiento de las dependencias de archivos en el proyecto desde la ventana del mismo, para proporcionar una visualización de las relaciones del software con el archivo. Visual DSP + + utiliza las herramientas de desarrollo de código para el procesar el proyecto y para producir un programa DSP.

4.1.5 Características de depuración. Mientras se realiza la depuración del proyecto, el usuario puede:

Ver y depurar código mezclado de C/C++ y código ensamblador. Ver intercalados el código C/C++ con el código ensamblador. El número de línea y el símbolo de ayuda ayudan a depuración de archivos en ensamblador.

Ejecutar la línea de secuencias de comandos. Utilizar las secuencias de comandos para personalizarlas las características principales de depuración.

Utilizar expresiones de memoria. Utilizar expresiones que se refieren a la memoria.

Utilizar puntos ruptura para ver registros y memoria. Añadir y eliminar rápidamente, y activar y desactivar los puntos de ruptura.

Establecer los puntos de vigilancia simulados. Establecer el punto de vigilancia en pilas, registros, memoria, o símbolos para poner fin a la ejecución del programa.

Mostrar estadísticamente el perfil del procesador PC (únicamente en emulación de depuración de dispositivos JTAG). Tomar muestras aleatorias y mostrarlas gráficamente para ver cuando el programa utiliza la mayor parte de su tiempo.

Mostrar linealmente el perfil del dispositivo PC del procesador (únicamente para simulación). Muestra lo ejecutado en cada PC y proporciona una exacta y completa pantalla gráfica de lo que fue ejecutado en el programa.

Generar interrupciones mediante el flujo de datos I / O. Configura el puerto serial (SPORT) o la memoria mapeada I/O.

Crear ventanas de registro personalizado. Configurar ventanas de registro personalizadas para mostrar un determinado conjunto de registros.

Plotear los valores de la memoria del DSP. Elegir entre varios estilos de ploteo, opciones de procesamiento de datos y de representación.

Trazar el historial de la ejecución del programa. Trazar la forma en que su programa llega a un cierto punto, mostrando lo leído, lo escrito, y los nombres simbólicos.

4.1.6 Características del VisualDSP++3.5. El VisualDSP++3.5 incluye las siguientes nuevas características y mejoras.

4.1.6.1 Múltiple apoyo a proyectos. VisualDSP++ ofrece la posibilidad de alternar entre varios proyectos abiertos al mismo tiempo en la misma sesión. La ventana de proyecto muestra los proyectos activos.

4.1.6.2 Flujo y carga de datos. El VisualDSP++ ofrece ahora la posibilidad de flujo y carga de datos desde un dispositivo DSP sin detener al mismo. El ambiente integrado de desarrollo y depuración saca el mejor provecho de esta capacidad.

4.1.6.3 Perfiles de código con enlazador experto. Es posible utilizar un perfil de enlazador experto en las secciones objetivo de un programa. Cuando el programa se detiene, el enlazador experto grafica y muestra cuánto tiempo se gastó en cada uno de los objetivos de la sección. Se puede utilizar esta pantalla para localizar el código "hotspots" y, a continuación, mover rápidamente ese código a la memoria interna.

4.1.6.4 Menús con iconos. Los iconos ahora aparecen junto a los comandos de menú y tienen los botones correspondientes en la barra de herramientas.

4.2 DESARROLLO DE PROYECTOS

Durante el desarrollo de proyectos, el VisualDSP++ ayuda interactivamente a observar y modificar los datos en el procesador y en la memoria. La visión general de programación eficaz con el VisualDSP++ depende de un proceso de cuatro pasos donde el usuario debe aprender a:

- Trabajar con el VisualDSP++.
- Aplicar el diseño de software estructurado con VisualDSP++.
- Optimizar el rendimiento con VisualDSP++.
- Probar y depurar sus programas con el VisualDSP++.

Al trabajar con el Visual DSP++ el usuario debe tener un conocimiento práctico del software, de todos los objetivos, y de las plataformas. Así mismo debe saber cómo y cuándo utilizar sus diversas funciones y tener un compromiso firme con los elementos básicos del proyecto, tales como trabajar con páginas de propiedades, configurar las sesiones de depuración, conocer las diferencias entre las tres fases de desarrollo: la evaluación (a través de un sistema de desarrollo EZ- Kit Lite), la simulación y la emulación, comprender cómo las secciones del programa y los segmentos de memoria se relacionan con la memoria física del DSP y acceder a los periféricos. Esta tarea incluye la creación y manipulación interrupciones en C y en el lenguaje ensamblador.

El diseño estructurado de software del VisualDSP++permite al usuario considerar la posibilidad de elementos de diseño de software, de reutilización y la interoperabilidad del código.

Con la optimización de la ejecución del Visual DSP++ el usuario debe comprender en esta etapa la forma de acceder a las características del DSP y cómo utilizar una estructura para desarrollar software. Luego el usuario debe optimizar su software para sacar el máximo provecho del cálculo de la potencia del DSP. Este paso implica:

- Comprender el optimizador de compilador.
- Escribir programas tanto en C como en el lenguaje ensamblador.
- Acceder a estructuras de datos en C/C + + en lenguaje ensamblador.
- Aprovechar la potencia de C++.

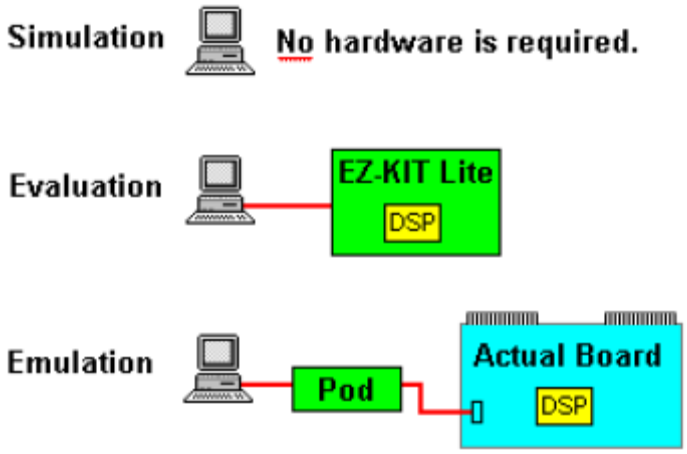
- Creación y uso de superposiciones.
- Configurar la memoria L1 para emulación (procesadores Blackin solamente).
- Uso de perfiles estadísticos.

Las pruebas y la depuración en el VisualDSP++ son el momento en donde el usuario debe tener una buena comprensión de las diversas instalaciones y servicios disponibles del software para la producción óptima del programa. El último paso es aplicar pruebas de software y técnicas de depuración, que incluyen:

- Recolectar datos y utilizar la opción avanzada de las ventanas de ploteo.
- Utilizar la simulación compilada.
- Utilizar el COM y el ActiveX probar los entornos y tomar ventaja de la interoperabilidad con otras aplicaciones.

4.2.1 Desarrollo de un proyecto en el DSP. Un típico proyecto incluye tres fases: la simulación, la evaluación, y la emulación. La figura 11 muestra las tres fases del desarrollo del proyecto. El VisualDSP++ se utiliza durante la simulación y la emulación de proyectos.

Figura 11. Fases de desarrollo de proyectos en DSP



Fuente: VisualDSP3.5 users Guide for 16-bit *processors*, cp.1-19.

4.2.1.1 Simulación. El usuario generalmente comienza el desarrollo de proyectos en un entorno de simulación mientras que los ingenieros de hardware se encuentran desarrollando el nuevo hardware (teléfono celular, equipo, y así sucesivamente). La simulación imita la memoria del sistema y la memoria de

entrada y salida (I/O), lo cual permite ver partes del dispositivo DSP. Un simulador es un software que imita el comportamiento del chip del DSP. Al correr el VisualDSP++ en simulación con un dispositivo predefinido (por ejemplo ADSP-2181), el usuario puede construir, editar y depurar su programa.

4.2.1.2 Evaluación. Al utilizar el EZ-KIT Lite ™ en el sistema de evaluación en la fase inicial de planificación del proyecto, se le permite al usuario determinar el DSP que mejor se adapta a sus necesidades. El PC es conectado a la tarjeta EZ-KIT Lite a través del cable, que le permite supervisar el comportamiento del DSP.

4.2.1.3 Emulación. Una vez que el hardware esté listo, el usuario puede pasar directamente a un JTAG emulador del hardware que se conecta al PC, simulando que es el objetivo real de la tarjeta DSP. Un emulador permite que el software de aplicación pueda ser descargado y depurado dentro del VisualDSP++. El software del emulador realiza las comunicaciones que le permiten ver la forma en que el código afecta el rendimiento del DSP.

4.2.1.4 Pasos para el desarrollo de un proyecto en el VisualDSP++. En el ambiente de desarrollo del VisualDSP++ el desarrollo del programa consiste en los siguientes pasos.

- Crear un proyecto.
- Configurar las opciones del proyecto.
- Agregar y editar los archivos fuente del proyecto.
- Definir las opciones de construcción del proyecto.
- Crear una versión depurada (archivo ejecutable) del proyecto.
- Crear una sesión depurada y cargar el ejecutable.
- Ejecutar y depurar el programa.
- Construir una versión libre del proyecto.

Siguiendo estos pasos, el usuario puede construir un proyecto en el DSP de forma coherente y con un mínimo de imprecisión en el desarrollo de los mismos. Este proceso disminuye el tiempo de desarrollo y le permite al usuario concentrarse en el desarrollo del código.

Paso 1: Crear un proyecto. Todo desarrollo en VisualDSP++ se produce dentro de un proyecto. El archivo de proyecto (.DPJ) almacena la información de construcción del programa: lista de archivos de origen y las opciones de configuración de las herramientas de desarrollo.

Paso 2: Configurar las opciones del proyecto. Definir el procesador a utilizar y establecer las opciones del proyecto (o aceptar la configuración por defecto) antes de añadir archivos al proyecto. El cuadro de dialogo de opciones de proyecto proporciona acceso al proyecto, el cual habilita las correspondientes herramientas de construcción para procesar los archivos de proyecto correctamente.

Paso 3: Añadir y editar los archivos fuente del Proyecto. Un proyecto normalmente contiene uno o más archivos fuente en C, C++, o en el lenguaje ensamblador. Después de crear un proyecto y definir el tipo de procesador, se pueden añadir nuevos archivos al proyecto ya sea importándolos o escribiéndolos. Es posible usar el editor del VisualDSP++ para crear nuevos archivos o modificar cualquier archivo de texto existente.

Agregar archivos al proyecto. El usuario puede agregar cualquier tipo de archivo al proyecto. Las herramientas de desarrollo del DSP permiten agregar archivos de forma selectiva, estos archivos sólo son reconocidos por el DSP cuando se genere el proyecto.

Creación de archivos para adjuntar al proyecto. El usuario puede crear nuevos archivos de texto. El editor puede leer o escribir archivos de texto con nombres arbitrarios. Al añadir archivos al proyecto, el VisualDSP++ actualiza el archivo del proyecto.

Edición de archivos. El usuario puede editar los archivos que se agregan al proyecto. Para abrir un archivo para ser editado, se debe hacer doble clic en el icono del archivo en la ventana del proyecto. El editor tiene un estándar al estilo de interfaz de Windows, el usuario puede manejar las operaciones de edición normal y así mismo, múltiples ventanas abiertas. Adicionalmente el DSP incluye las características personalizables como el lenguaje, sintaxis específica, color, marcador de capacidades (creación y búsqueda).

Administración de las dependencias de Proyectos. El usuario puede ejercer el control de las dependencias de los archivos fuente del proyecto y utilizar esta información en otros archivos y, en consecuencia, determinar el orden de construcción. El VisualDSP++ mantiene un archivo *makefile*, que almacena la información de cada dependencia para cada archivo en el proyecto. El VisualDSP++ tiene la información de las actualizaciones de las dependencias para cuando el usuario cambie la opción de construir, al añadir un archivo al proyecto, o cuando el usuario elige actualización de las dependencias del menú proyecto.

Paso 4: Definición de las opciones de construcción de proyecto. Después de crear un proyecto, establecer el tipo de procesador, añadir o editar el código fuente de archivos del proyecto, el usuario puede configurar las opciones para

construir el proyecto. Antes de usar las herramientas de desarrollo para crear un archivo ejecutable el usuario debe especificar las opciones o aceptar las opciones predeterminadas en VisualDSP++. El usuario puede especificar opciones para un proyecto o para un conjunto de archivos individuales. El VisualDSP++ mantiene los cambios en las opciones de construcción.

La configuración de un proyecto establece el control de su construcción. Por defecto, las opciones son *Debug* y *Release*. Seleccionando *Debug* y dejando todas las otras opciones en su valor por defecto se establece la construcción de un proyecto que puede ser depurado. El compilador genera la información de depuración. Seleccionando *Release* y dejando todas las otras opciones en su valor por defecto se establece la construcción de un proyecto con capacidad limitada o sin capacidad de depurar.

El usuario puede modificar la operación por defecto del VisualDSP++ para cualquier configuración, cambiando las entradas apropiadas en las páginas de Compile, *Assemble* y *Link*. Es posible crear configuraciones personalizadas que incluyen las opciones de construcción y los archivos de código fuente que desee.

Proyecto en todo el archivo y opciones de herramienta. A continuación se debe decidir entre utilizar la configuración para todo el archivo de proyecto o configuración de archivos individuales. Para los proyectos construidos totalmente dentro de Visual DSP++ sin objetos ni librerías de archivos preexistentes, se suelen utilizar las opciones de proyecto en todo el archivo. Los archivos nuevos añadidos al proyecto heredan estos ajustes.

Archivo individual y Opciones de la herramienta. Ocasionalmente, puede que se desee especificar la configuración de la herramienta para archivos individuales. Cada archivo está asociado con la propiedad de dos páginas: una página general, que permite que el usuario elija la extracción de los directorios intermedios y archivos de salida, y una página de propiedades que es una herramienta específica que le permite elegir las opciones (Recopilar, Ensamble, Link, y así sucesivamente).

Paso 5: Construcción de una versión depurada del proyecto. Ahora el usuario debe construir una versión depurada del proyecto. A medida que la construcción del proyecto avanza, en la ventana de salida aparecen mensajes del estado de cada una de las herramientas de desarrollo de código. El archivo de salida debe ser de tipo ejecutable (.DXE) para producir el archivo depurador compatible con la salida.

Paso 6: Crear una sesión de depuración y carga del ejecutable. Después de construir con éxito un archivo ejecutable se debe configurar una sesión de depuración. El usuario puede ejecutar proyectos DSP que se hayan desarrollado ya sea como sesión de hardware o sesión de software. Luego de especificar el

tipo de dispositivo y la información del procesador, se debe cargar el archivo ejecutable del proyecto. En el cuadro de diálogo *Preferences* de la página general, se configura el VisualDSP++ para cargar el archivo automáticamente y avanzar a la función *main* del código.

Paso 7: Ejecutar y depurar el programa. Después de crear con éxito una sesión de depuración, y construir y cargar el programa ejecutable, el usuario ejecuta y depura el programa. Si el proyecto no ha sido actualizado (hay archivos de código fuente o dependencias de información obsoletos), VisualDSP++ pregunta si desea generar el proyecto antes de la carga y la depuración en el archivo ejecutable.

4.3 HERRAMIENTAS DE DESARROLLO DE CÓDIGO.

Existen diferentes herramientas de desarrollo de código, dependiendo del tipo de procesador. El VisualDSP++ da soporte a todos los formatos de archivos ejecutables producidos por el enlazador. Si el sistema está configurado con herramientas de desarrollo de terceros, puede seleccionar el compilador, ensamblador, enlazador, o utilizarlo para construir un objetivo en particular.

4.3.1 Compilador. El compilador procesa el código de los programas en C/C++, y en el lenguaje ensamblador. El término compilador se refiere a la utilidad que la compilación suministra con el software del Visual DSP++. El compilador genera un archivo que tiene por objeto la elaboración de uno o más archivos fuente en C/C+ +. El compilador tiene como primer objetivo asociarse con un archivo de extensión .DOJ.

El usuario puede especificar las opciones del compilador para la construcción y selección de un proyecto Las opciones del compilador se agrupan en las categorías descritas en la tabla 4. Las opciones de compilación dependerán del objetivo del usuario en el DSP y de las herramientas de desarrollo de código.

Tabla 4. Opciones del compilador

Categoría	Propósito
General	Optimización, elaboración, y opciones de terminación.
Preprocessor	Macro y directorio de opciones de búsqueda
Processor	Procesador de opciones específicas
Warning	Advertencia y opciones de presentación de error

4.3.2 Ensamblador. El ensamblador produce un archivo objeto ensamblando la fuente, las cabeceras, y los archivos de datos. El ensamblador tiene como primer objetivo la producción de un archivo de extensión .DOJ. El usuario puede especificar las opciones del ensamblador seleccionando *Project/Project Options/Assemble*. El término ensamblador se define de la siguiente manera.

4.3.2.1 Set de instrucciones. El conjunto de instrucciones en ensamblador que corresponden a un DSP específico.

4.3.2.2 Comandos del preprocesador. Comandos que dirigen el preprocesador para incluir archivos, realizar sustituciones de macro, y control condicional del ensamblador.

4.3.2.3 Directivas del ensamblador. Las directivas se encargan de decirle al ensamblador la forma de procesar el código fuente y establecer las características del DSP. El usuario puede utilizar estas directivas para la estructura del programa ya sea en segmentos o secciones que apoyan el uso de un enlazador de descripción de archivos (. LDF), para construir una imagen adaptada al sistema de destino.

4.3.3 Enlazador. El enlazador reúne los vínculos de archivos (los archivos objeto y los archivos de librerías) para producir archivos ejecutables (. DXE), la memoria compartida de archivos (. SM), y de superposición de archivos (. OVL), que pueden ser cargados en el archivo fuente. Los archivos de salida en el enlazador (. DXE. SM. OVL) son binarios, ejecutables, y archivos asociados (ELF). Para hacer un archivo ejecutable, el enlazador procesa los datos de un archivo .LDF, y uno o más ficheros de objeto (. DOJ). El archivo ejecutable contiene el código del programa y la depuración de la información. El enlazador resuelve las direcciones en archivos ejecutables.

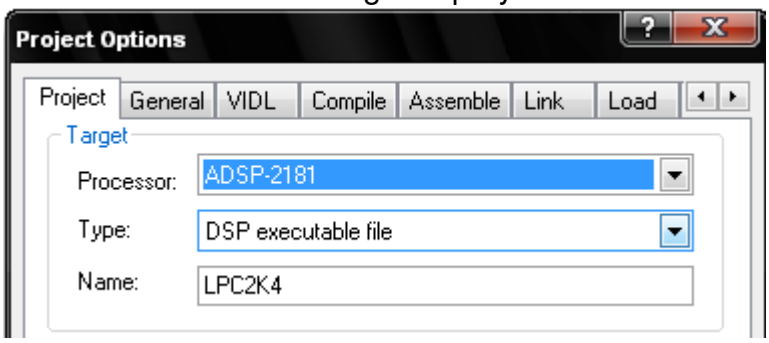
4.4 PROYECTOS EN EL DSP

El proyecto es la estructura en la que se construye el programa en DSP. El VisualDSP++ proporciona flexibilidad en la forma de configuración de los proyectos. El usuario configura las características para las herramientas de desarrollo de código del DSP, y también se especifica las características de construcción para el proyecto y para los archivos individuales. Es posible configurar inclusive las carpetas que contienen los archivos del código fuente.

4.4.1 Descripción de un proyecto. El usuario tiene como objetivo crear un programa que se ejecute en un sistema procesador (o multiprocesador). Todo el desarrollo en el VisualDSP++ se produce dentro de un proyecto. El término proyecto se refiere a la colección de archivos de código fuente y a las herramientas de configuración utilizadas para crear un programa en el DSP. Un archivo de proyecto (.DPJ) almacena el programa y la información de construcción. Se hace uso de la ventana de proyecto para administrar proyectos desde el inicio hasta el final.

4.4.2 Opciones de proyecto. El usuario especifica las opciones del proyecto, que se aplican a la totalidad del mismo en el DSP.

Figura 12. Opciones del cuadro de dialogo de proyecto



Fuente: Tomada del software VisualDSP3.5++

Para cada una de las herramientas de desarrollo de códigos (compilador, ensamblador, enlazador, divisor y cargador), existe una página con pestañas que ofrece las opciones que controlan como cada herramienta procesa entradas y genera salidas. Estas opciones se pueden definir o modificar según las necesidades del desarrollo del programa. Las opciones del proyecto especifican la siguiente información:

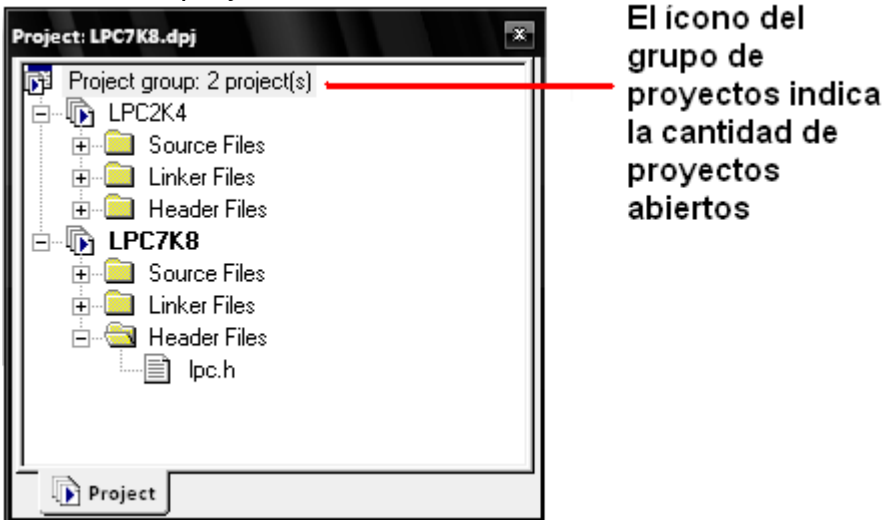
- Destino del proyecto
- Herramienta de cadena
- Archivo de directorios de salida
- Opciones post construcción

4.4.3 Grupos de proyecto. Grupos de proyectos permiten trabajar con un número de proyectos a la vez. Un grupo de proyectos puede estar vacío o contener cualquier número de proyectos. La apertura de un proyecto añade el mismo al grupo. El cierre de un proyecto lo elimina del grupo de proyectos. Una funcionalidad similar se encuentra en Microsoft Visual Studio.

La capacidad proporcionada por grupos de proyectos son particularmente útiles en el desarrollo VCSE, que implica múltiples proyectos. La ventana de proyecto muestra el icono de grupo de proyecto, como aparece en la figura 13.

Cada espacio de trabajo tiene un grupo de proyecto. Cuando se cambia entre los espacios de trabajo, el grupo del proyecto se carga y el mismo conjunto de archivos se abren igual que cuando se cerró por última vez el espacio de trabajo

Figura 13. Ventana de proyecto



Fuente: Tomada del software VisualDSP3.5++ y editada por los autores del proyecto

Solo un proyecto se encuentra activo al tiempo. Inicialmente el proyecto activo responde a los comandos y mensajes de los menús y de las barras de herramientas. En la ventana de proyecto muestra el proyecto activo con negrilla.

Un cuadro de proyecto, que se encuentra por defecto en la barra de herramientas, muestra el nombre del proyecto activo (ver figura 14).

Figura 14. Cuadro de proyecto que muestra el proyecto activo

Fuente: Tomada del software VisualDSP3.5++ y editada por los autores del proyecto

4.4.4 Construcción de un proyecto. El término construir se refiere al proceso de realización de las operaciones (como pre-procesamiento, montaje, y vinculación) en los proyectos y archivos. Durante la construcción, el VisualDSP++ tiene los procesos de archivos del proyecto que han sido modificados desde la anterior construcción, así como los archivos del proyecto que incluyen archivos modificados. Construir es diferente a una reconstrucción total del proyecto. Cuando se reconstruye el VisualDSP++ ejecuta el comando de reconstruir todos los procesos de todos los archivos en el proyecto, independientemente de si han sido o no modificados.

La construcción de un proyecto se basa en todos los archivos obsoletos del proyecto y le permite al usuario realizar el programa. El VisualDSP++ utiliza una dependencia de información para determinar qué archivos debe actualizar durante la construcción de un proyecto. El usuario debe tener en cuenta lo siguiente:

Un archivo con una extensión no reconocida es ignorado en la construcción del proyecto.

Si un archivo de cabecera incluido es modificado, el VisualDSP++ construye los archivos fuente que incluyen el archivo de encabezado (#include) a pesar de si los archivos de origen se han modificado desde la anterior construcción.

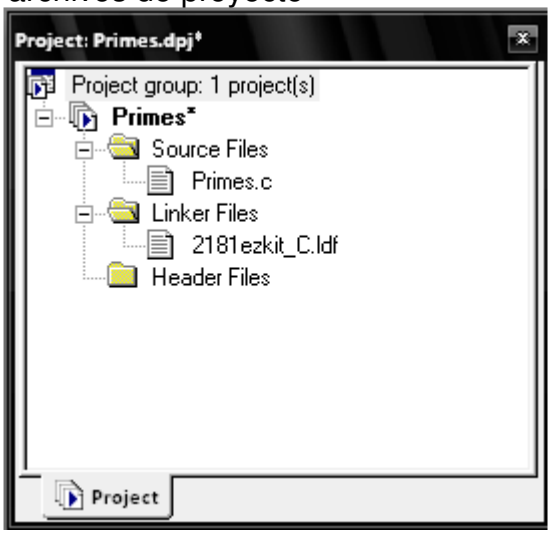
Los íconos en la ventana de proyecto indican el estado del archivo (por ejemplo, archivos excluidos o ficheros con opciones específicas que anulan la configuración de un proyecto).

4.4.5 Creación de un archivo. Se construye un archivo para compilar y ensamblar dicho archivo y de esta forma localizar y remover errores. El proceso de construcción actualiza la salida del archivo fuente (archivo .OBJ), y actualiza la información de depuración del archivo de salida. Construir un archivo es una operación muy veloz. Proyectos largos sin embargo pueden requerir de horas para ser construidos.

El usuario puede construir múltiples archivos seleccionados, similar a la construcción de un archivo individual, este proceso permite también actualizar los archivos de salida. Si el usuario cambia el archivo encabezado común que se requiere para una construcción total, es posible construir solamente el archivo actual para asegurar que el cambio arregle el error en dicho archivo.

4.4.6 Reglas de proyecto. La ventana de proyectos muestra los archivos de un proyecto, tal como se ve en la figura 15.

Figura 15. Ejemplo de archivos de proyecto



Fuente: Tomada del software VisualDSP3.5++

Las siguientes reglas determinan cómo los archivos y las subcarpetas se comportan en el árbol de archivos de la ventana de proyecto.

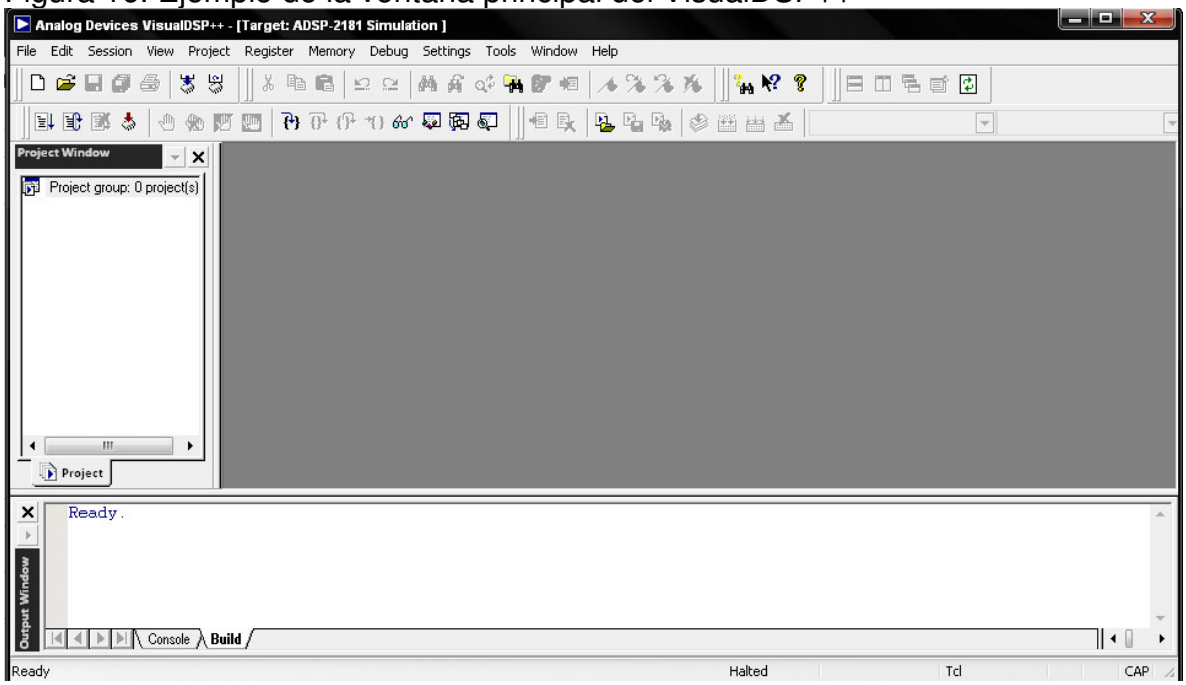
- El usuario puede incluir cualquier archivo en un proyecto.
- Sólo está permitido un archivo. LDF.
- No se puede añadir el mismo archivo en el mismo proyecto más de una vez.

5 AMBIENTE DE TRABAJO DEL VisualDSP++ 3.5

5.1 PARTES DE LA INTERFAZ DE USUARIO.

VisualDSP++ es una interfaz de usuario de manejo sencillo, para la programación de DSPs de la familia Analog Devices. Al momento de abrir el VisualDSP++ se hace presente la ventana principal de aplicaciones. La figura 16 muestra un ejemplo de la ventana principal del VisualDSP++.

Figura 16. Ejemplo de la ventana principal del VisualDSP++



Fuente: Tomada del software VisualDSP3.5++

Esta área de trabajo contiene todo lo necesario para poder construir, administrar, y depurar un proyecto DSP. Es posible establecer ciertas preferencias que especifican la apariencia de los objetos de la aplicación (fuentes, visibilidad, entre

otros). También se pueden abrir archivos del proyecto arrastrándolos y soltándolos dentro de la ventana principal.

La ventana principal del VisualDSP++ incluye las siguientes partes:

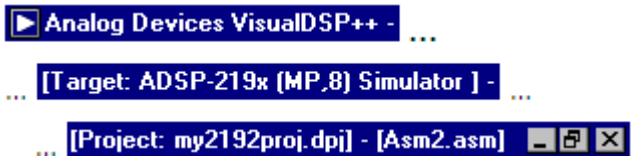
- Barra de título.
- Barra Menú.
- Ventana de proyecto.
- Barra de herramientas.
- Ventana de salida.
- Barra de estado.

VisualDSP++ también provee varias ventanas de depuración para facilitar el desarrollo del proyecto. Es necesario aprender únicamente una interface para depurar todas las aplicaciones DSP.

5.1.1 Barra de título. La figura 17 muestra las diferentes partes de la barra de título. La barra de título incluye los siguientes componentes

- Botón del Menú de Control
- Nombre de la aplicación – Analog Devices VisualDSP++
- Nombre del dispositivo activo
- Nombre del proyecto
- Nombre del archivo
- Botones de ventanas estándar

Figura 17. Ejemplo de la barra de título



Fuente: Los autores del proyecto

Haciendo clic en el botón del menú de control se abre el menú de control, el cuál contiene comandos para el posicionamiento, redimensionamiento, minimización, maximización, y cerrado de la ventana. Al hacer doble clic en el botón del menú de control se cierra el VisualDSP++. Hacer clic derecho (ver figura 18) en el menú de control o en la barra de título es lo mismo.

Figura 18. Clic derecho en la barra de título

Fuente: Los autores del proyecto

5.1.1.1 Información adicional en la barra de título. Una ventana de registro en la barra de título muestra su formato numérico (tal como hexadecimal). Una ventana de edición en la barra de título muestra el nombre del archivo fuente.

5.1.1.2 Menú clic derecho en la barra de título. Un menú tal como el mostrado en la figura 4-3 aparece cuando se hace clic derecho sobre la barra de título del VisualDSP++, o cuando se hace clic derecho sobre la barra de título de una sub ventana.

Desde el menú clic derecho de la barra de título es posible:

- Redimensionar o mover la ventana de aplicación.
- Cerrar el VisualDSP++.

5.1.2 Menú de control.

Los comandos sobre el menú de control (menú de sistema mostrado en la figura 19) permiten mover, dimensionar o cerrar una ventana.

5.1.2.1 Íconos del programa. Al hacer clic en uno de los íconos del programa me permiten abrir un menú de control.

- Ícono de programa para las ventanas de aplicación y depuración.
- Ícono de programa para las ventanas de edición.

Cuando se ubica el puntero del mouse sobre un comando del menú de control, una breve descripción del comando aparece en la barra de estado en la parte superior de la ventana de aplicación.

Figura 19. Menú de control del VisualDSP++

Fuente: Tomada del software VisualDSP3.5++

5.1.2.2 Ventanas de edición. Un editor flotante de la ventana de menú de control incluye *Next*, el cuál mueve el foco a otra ventana.

Cuando una ventana de edición flota en la ventana de la aplicación principal, su icono de programa se encuentra al lado izquierdo de su barra de título. Cuando una ventana de edición está maximizada, el icono de programa se encuentra en la parte final izquierda de la barra de menú.

5.1.2.3 Ventanas de depuración. Cada ventana de depuración tiene un menú de control. Es posible abrir el menú de control de la ventana de depuración únicamente cuando la ventana se encuentra flotante en la ventana principal.

5.1.3 Barra de menú. La barra de menú mostrada en la figura 4-5, aparece directamente debajo de la barra de título y muestra los encabezados del menú, tales como File y Edit.

Figura 20. Barra de menú del VisualDSP++

Fuente: Tomada del software VisualDSP3.5++

Para desplegar los comandos y submenús, se debe hacer clic en el encabezado del menú. También se puede tener acceso a muchos comandos de la barra menú de la siguiente manera:

- Haciendo clic en los botones de la barra de herramientas.
- Escribiendo los atajos de teclado.
- Por medio del clic derecho del mouse y seleccionando un comando del menú de contexto.

5.1.4 Información de comandos. Cuando el puntero del mouse se encuentra sobre los comandos de la barra de menú (o sobre un botón de la barra de herramientas), una pequeña descripción del comando aparece en la barra de estado en la parte superior de la ventana principal. Para cada comando hay disponible la ayuda de contexto sensible (puntero del mouse sobre el comando). Para aprender más sobre un comando individual de la barra de menú:

- Presione **Shift+F1** o haga clic en el botón de ayuda de la barra de herramientas  .
- El puntero se convertirá en un puntero de ayuda  .
- Mueva el puntero de ayuda sobre el comando del menú. Si es necesario navegue a través de los submenús.
- Haga clic en el mouse para mostrar la ayuda.

De esta manera se podrá observar la descripción del comando en la ventana de ayuda.

5.1.5 Barra de herramientas y herramientas de usuario. Una barra de herramientas es un set de botones por medio del cual es posible correr rápidamente un comando, haciendo clic en unos de sus botones. Se hace uso de la barra de herramientas para organizar las tareas que más se utilizan, posicionando la barra de herramientas en la pantalla para tener un acceso más veloz a las herramientas que el usuario planee usar. La aplicación incluye barras de herramientas estándar, aunque es posible crear barras de herramientas personalizadas.

5.1.5.1 Barras de herramientas de construcción. La tabla 5 muestra las barras de herramientas estándar (por defecto). Para obtener información acerca de una herramienta, se debe mover el puntero del mouse sobre la herramienta y presione la tecla F1.

5.1.5.2 Personalización de la barra de herramientas. Por defecto, nueve barras de herramientas aparecen cerca a la parte superior de la ventana de aplicación, debajo de la barra de menú. El usuario puede cambiar la apariencia de las barras de herramientas para:

- Moverse, acoplarse, o flotar.
- Adjuntar o remover botones de la barra de herramientas.
- Mostrar cool buttons, large buttons, o ambos.

Tabla 5. Barra de herramientas de construcción

Nombre	Barra de herramientas
File	
Edit	
Help	
Project	
Window	
Debug	
Multiprocessor	
User Tools	
Workspaces	

También es posible:

- Ocultar barras de herramientas desde la pestaña View.
- Adjuntar o quitar barras de herramientas de construcción por defecto.

5.1.5.3 Barra de herramientas: Acople vs. Flotar. Por defecto, las barras de herramientas se encuentran ubicadas bajo la barra de menú de la aplicación. El usuario puede moverlas hacia las siguientes ubicaciones:

- Sobre una ventana acoplada.
- Sobre la ventana principal.
- En cualquier parte del escritorio.

Cuando una barra de herramientas se adjunta a una ventana, se le llama acople de la barra de herramientas. El usuario puede decidir cuando una barra de herramientas será acoplada en forma y tamaño, moviendo su contorno y arrastrándola. Para prevenir que una barra de herramientas se acople, se debe presionar y mantener la tecla Ctrl, mientras que se arrastra la barra de herramientas a la nueva ubicación. Es posible separar una barra de herramientas de una ventana y moverla a otra ubicación cualquiera dentro del escritorio. Una barra de herramientas flotante es una ventana solitaria que no se encuentra acoplada. Una barra de herramientas acoplada no muestra su nombre, pero una barra de herramientas flotante si lo hace, la figura 21 muestra la barra de herramientas flotante de ayuda.

Figura 21. Ejemplo de barra de herramientas flotante

Fuente: Tomada del software VisualDSP3.5++

5.1.5.4 Apariencia de los botones de la barra de herramientas. Es posible seleccionar la apariencia de los botones de la barra de herramientas, existen dos opciones, *cool look* y *large buttons*. Estas opciones permiten diferenciar ligeramente la apariencia de los botones. La opción *cool look* incluye un par de barras verticales en el lado izquierdo de la barra de herramientas, pero a su vez remueve la caja cuadrar de cada botón. Las barras verticales separan visualmente los botones de las barras de herramientas en grupos. La opción *large buttons* hace mayor el área de cada botón. La tabla 6 muestra como aparecen los botones con la opción *cool look* habilitada (apagada), o deshabilitada (encendida).

5.1.5.5 Forma de las barras de herramientas. El usuario puede cambiar la forma de una barra de herramientas flotante. La tabla 7 muestra dos formas de barras de herramientas.

Dependiendo del número de herramientas en la barra es posible crear una nueva con arreglos de largo y ancho.

Tabla 6. Barras de herramientas en diferentes opciones de vista

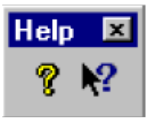
Opciones	Acoplado	Flotante
Cool look – Deshabilitado Large buttons - Deshabilitado		
Cool look –Habilitado Large buttons - Deshabilitado		
Cool look – Deshabilitado Large buttons - Habilitado		
Cool look – Habilitado Large buttons - Habilitado		

Tabla 7. Barra de herramienta en dos orientaciones

Horizontal	Vertical
	

5.1.5.6 Reglas de las barra de herramientas. Cuando se está trabajando con barras de herramientas se debe tener precaución con las siguientes reglas:

El usuario puede personalizar una barra de herramientas de construcción (por ejemplo es posible remover un botón de la barra de herramientas File), pero no se puede borrar la barra de herramientas de construcción. También se puede restablecer los botones en una barra de herramientas de construcción a su configuración por defecto

Se puede cambiar el nombre de una barra de herramientas definida por el usuario, pero no es posible cambiar el nombre de la barra de herramienta de construcción. Por ejemplo, no se puede cambiar el nombre de la barra de herramientas File.

5.1.5.7 Herramientas de usuario. Se ahorra tiempo corriendo comandos por medio de la configuración de herramientas de usuario. El usuario puede configurar hasta diez barras de usuario. Una barra de usuario corre un comando cuando:

- Contiene parámetros para lanzar una aplicación.
- Es una secuencia de comandos.

Se accede a la configuración de herramientas de usuario desde el menú Tools o desde la barra de herramientas User Tools, tal como se muestra en la figura 22.

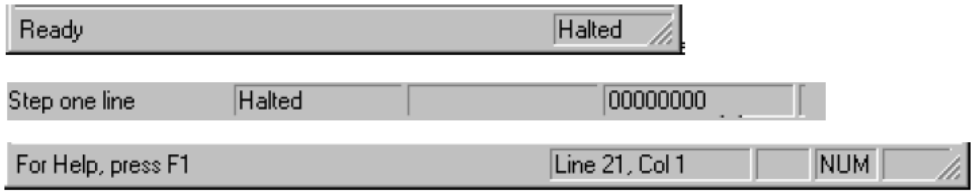
Figura 22. Herramientas de usuario por defecto

Fuente: Tomada del software VisualDSP3.5++

Cuando una herramienta de usuario es configurada, su nombre de menú (etiqueta) aparece en el menú Tools. La etiqueta también aparece cuando se mueve el puntero del mouse sobre un botón de la herramienta.

5.1.6 Barra de estado. La barra de estado, localizada en la parte superior de la ventana de aplicación principal, provee diferentes mensajes de información. La figura 23 muestra diferente información dada en la barra de estado.

Figura 23. Apariencia de la barra de estado en función del contexto



Fuente: Tomada del software VisualDSP3.5++

El tipo de información que aparece en la barra de estado depende de su contexto (de qué se está haciendo).

Cuando se mueve el puntero del mouse sobre un botón de la barra de herramientas o sobre un comando de la barra menú, aparecerá una pequeña descripción del botón o del comando. Cuando se detiene un programa con el comando de operación Halt, aparece en la barra de estado la dirección de donde se detiene el programa. Cuando se usa alguna secuencia de comandos la barra de estado provee información tal como se describe en la tabla 8.

Tabla 8. Información de la barra de estado cuando se está editando

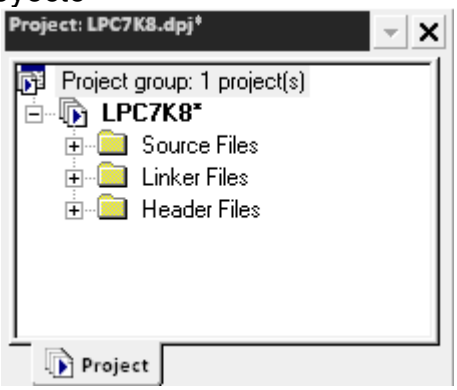
Item	Indicación
Line ###	Numero de la línea actual del cursor
Col ###	Numero de la columna actual del cursor
CAP	La tecla Caps Lock del teclado está activada
NUM	La tecla Num Lock del teclado está activada
SCRL	La tecla Scroll Lock del teclado está activada

5.2 VENTANAS DEL VisualDSP++.

Desde la ventana principal de la aplicación es posible abrir una ventana de proyecto, una ventana de edición, una ventana de salida, y diferentes ventanas de depuración.

5.2.1 Ventana de proyecto. Para abrir una ventana de proyecto se debe seleccionar *View/Project Window*. La figura 24 muestra una ventana de proyecto

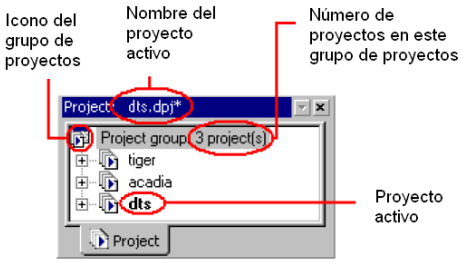
Figura 24. Ventana de proyecto



Fuente: Tomada del software VisualDSP3.5++

5.2.1.1 Vista del proyecto. La vista de proyecto muestra un grupo de proyectos el cuál contiene cualquier número de proyectos. Únicamente un proyecto, sin embargo, se encuentra activo al tiempo. Los nodos se organizan en una jerarquía similar a la estructura de los archivos en el explorador de Windows. La figura 25 muestra alguna de la información dada por la vista de proyecto.

Figura 25. Vista de proyecto

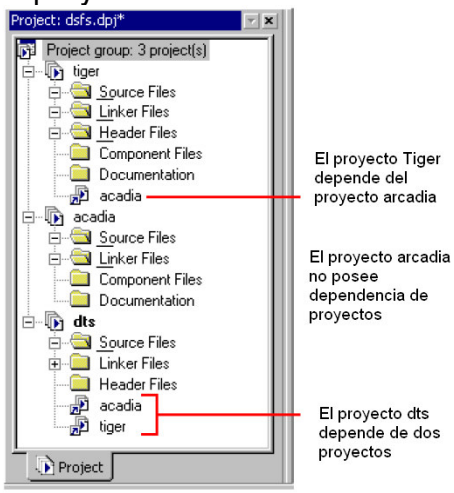


Fuente: Tomada del software VisualDSP3.5++ y editada por los autores del proyecto

5.2.1.2 Dependencia de proyectos. Un proyecto puede depender de otros proyectos para su ejecución. El icono indica dependencia e identifica la dependencia. Construir un proyecto también lo hace para el cual es dependiente. La figura 26 muestra como se indican las dependencias en la vista de proyectos.

5.2.1.3 Nodos de proyectos. La ventana de proyectos comprime los tipos de nodos descritos en la tabla 9.

Figura 26. Dependencia de proyectos



Fuente: Tomada del software VisualDSP3.5++ y editada por los autores del proyecto

Tabla 9. Tipos de nodos en la ventana de proyecto

Nodo	Ícono	Descripción
Grupo de proyectos		Únicamente un grupo de proyectos en una sesión de depurada.
Proyecto		Múltiples proyectos permitidos, pero solo uno se encuentra activo
Carpeta		Carpeta cerrada Carpeta abierta que muestra su contenido
Archivo		Archivo que utiliza características de proyecto Archivo cuyas opciones difieren de las opciones del proyecto Archivo excluido de la configuración actual Makefile activo
Dependencia de proyecto		Proyecto sobre el cual depende este proyecto

5.2.1.4 Carpeta de proyectos. Las carpetas de la ventana de proyectos organizan archivos dentro de un proyecto. El usuario puede especificar las propiedades de las carpetas. Ellas pueden ser anidadas a cualquier profundidad, y no pueden poseer atributos del proceso de construcción, ya que no reflejan el sistema de archivos. Las carpetas no aparecen en los listados de directorios, del mismo modo que en el explorador de Windows.

Cuando se adjuntan archivos al árbol del proyecto con ubicación automática de archivos, cada archivo es localizado en la primera carpeta que ha sido configurada con la misma extensión del archivo. Luego que se realiza la localización automática el usuario puede manualmente mover el archivo a cualquier parte. Para mover un archivo fuera de una carpeta a otra, se selecciona el archivo y luego se arrastra y libera dentro de la otra carpeta.

5.2.1.5 Archivos del proyecto. En la ventana de proyectos los archivos son representados por íconos mostrados en la tabla 10.

Los archivos aparecen en un nodo árbol expandible y compresible. Los archivos fuente son archivos en lenguaje C/C++ o assembler para su respectivo proyecto. Estos archivos proveen al proyecto de código y datos. El usuario puede adjuntar, quitar, y modificar los archivos fuente.

Cada proyecto debe incluir un archivo .LDF, el cuál contiene los comandos de entrada para el linker. Si no se incluye un archivo .LDF dentro del proyecto, este se construye con un .LDF por defecto. Un proyecto DSP puede también incluir archivos de datos y archivos de encabezados.

Tabla 10. Iconos de la ventana de proyectos

Ícono	Descripción
	Archivo que utiliza opciones de proyecto
	Archivo que utiliza opciones que difieren de las opciones del proyecto
	Archivos excluidos de la configuración actual
	Makefile activo

5.2.1.6 Íconos de la ventana de proyecto para el control de código fuente (SCC de sus siglas en inglés). Los íconos en la ventana de proyectos indican el estado del control de código fuente. Los archivos con una marca de chequeo de color verde (✔) están bajo SCC y han sido comprobados dentro de él. Los archivos con una marca de chequeo de color rojo (✔) se encuentran bajo SCC y han sido comprobados fuera de él. Cuando un archivo no está conectado a una copia controlaba bajo SCC, el ícono del archivo no tiene marca de chequeo. La tabla 11 muestra los íconos de que indican el estado SCC.

5.2.2 Ventana de edición. La ventana de edición se utiliza para observar y editar los archivos del proyecto. El usuario puede abrir una ventana de edición desde la ventana de proyecto haciendo doble clic sobre un archivo o seleccionando Open File del menú clic derecho del archivo. La figura 27 muestra los ítems que se pueden personalizar en la ventana de edición.

Tabla 11. Iconos de estado SCC

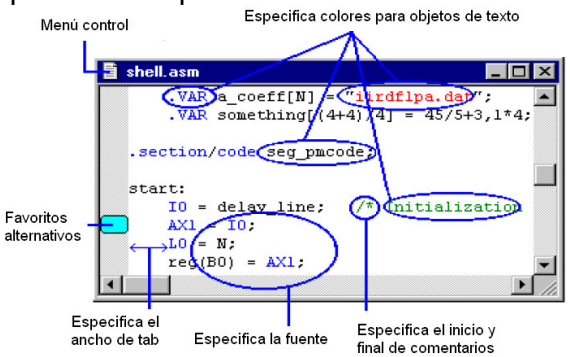
Ícono	Descripción
	El archivo está bajo SCC y ha sido comprobado dentro de él.
	El archivo está bajo SCC y ha sido comprobado fuera de él.
	El archivo de proyecto es chequeado fuera
	El archivo incluye un comando de archivo específico de construcción y es chequeado fuera.
	El Makefile es chequeado fuera.
	Archivo excluido de la construcción y es chequeado fuera.

El usuario puede abrir tantas ventanas de edición como quiera y hacer lo siguiente:

- Definir el color de comentarios, cadenas de palabras, y espaciados.
- Previsualizar e imprimir la ventana de datos.
- Cargar un script.
- Definir encabezados y pies de página.
- Establecer favoritos (*bookmarks*).
- Encontrar, reemplazar, y adecuar expresiones.
- Saltar a la siguiente o a la anterior sintaxis de error.
- Copiar, cortar, pegar, deshacer y rehacer más de 500 niveles de edición para cada archivo abierto.

Habilitar el modo *Editor Tab* para alternar rápidamente entre los archivos fuente.
Abrir archivos encabezados desde el menú clic derecho.

Figura 27. Ítems que pueden ser personalizados



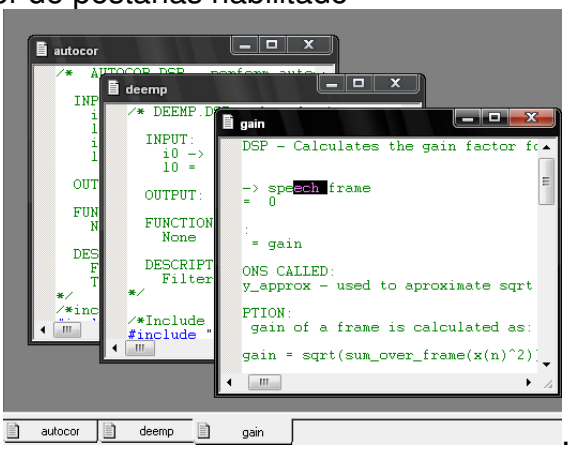
Fuente: Tomada del software VisualDSP3.5++ y editada por los autores del proyecto

5.2.2.1 Menú clic derecho. El menú de clic derecho de la ventana de edición provee los siguientes comandos:

- Hacer o deshacer la última edición.
- Cortar, copiar, o pegar texto.
- Colocar un favorito o ir al siguiente favorito.
- Mostrar el número de líneas o ir a un número de línea específica.
- Correr hasta cursor.
- Encontrar un valor específico indicando varios parámetros de búsqueda.
- Seleccionar formato (Hexadecimal, flotante, entero sin signo, entero, octal).

5.2.2.2 Modo editor de pestañas. El modo editor de pestañas provee una alternativa, basada en una interface de pestañas para administrar múltiples archivos fuente en la ventana de edición. Cuando se habilita este modo de funcionamiento desde el menú *View*, aparece una pestaña por cada archivo fuente abierto en la parte superior de la ventana de edición. El usuario hace clic sobre las pestañas para alternar entre archivos. La figura 28 muestra una ventana de edición con la opción de editor de pestañas habilitada.

Figura 28. Modo editor de pestañas habilitado



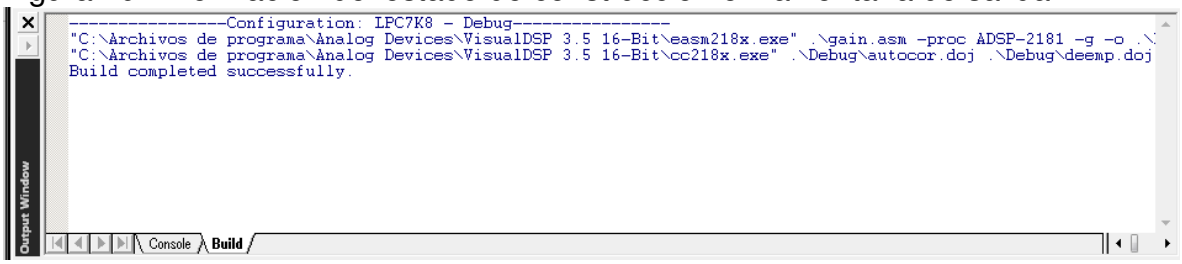
Fuente: Tomada del software VisualDSP3.5++

5.2.3 Ventana de salida. La ventana de salida se encarga de:

- Mostrar mensajes de texto estándar de entrada y salida tales como estado de carga de archivos y mensajes de error.
- Mostrar información del estado de construcción del proyecto actual.
- Provee acceso a errores en los archivos fuente.
- Actúa como una interface de scripts.

La ventana de salida mostrada en la figura 29 contiene la información del estado de construcción. Para mostrar la ventana se debe seleccionar View/Output Window.

Figura 29. Información del estado de construcción en la ventana de salida

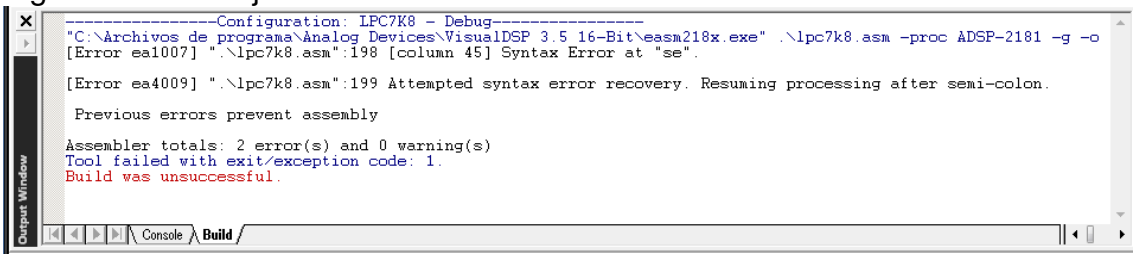


Fuente: Tomada del software VisualDSP3.5++

5.2.3.1 Pestañas de la ventana de salida. Haciendo clic sobre las dos pestañas de la ventana de salida (Console y Built), muestran páginas que contienen diferente información y capacidades.

- **Página de construcción**: la página de construcción (figura 30) muestra mensajes de error generados durante una construcción. El usuario puede hacer doble clic sobre un mensaje de error para saltar al código errado en una ventana de edición.

Figura 30. Mensajes de error en la ventana de salida



Fuente: Tomada del software VisualDSP3.5++

Es permitido moverse entre errores seleccionando la opción de siguiente o anterior error en el menú de edición. Por defecto, la salida del VisualDSP++ es azul y la herramienta de salida es negra, pero se pueden cambiar estos colores en el cuadro de diálogo llamado Preferences.

- **Pagina de consola**: desde esta página de la ventana de salida (figura 4-16) es posible:

  - Ver mensajes de error de VisualDSP++ o del dispositivo.
  - Ver la salida STDIO de programas C/C++.
  - Ver mensajes de I/O.
  - Realizar selección de múltiples líneas, copiado, pegado y borrado.
  - Auto completar secuencia de comandos.
  - Utilizar marcadores (*bookmarks*)
  - Alternar un marcador presionando Ctrl+F2.
  - Moverse al siguiente marcador presionando F2.

5.2.3.2 Mensajes de error de la ventana de salida. Las herramientas de desarrollo de código de DSP que realizan procesamiento por bloques pueden producir mensajes de error y advertencia cuando se entrega el resultado. Estos mensajes de información aparecen en la página de construcción en la ventana de salida.

Cada error es identificado con un código único de seis caracteres (tal como pp0019). La descripción de errores incluye una explicación de la condición que causa el error y una sugerencia para solucionar el problema. Cuando son aplicables, los mensajes de error incluyen el nombre de los archivos fuente y el número de la línea del código errado.

- **Gravedad de los mensajes de error.** cada mensaje de error tiene uno o más niveles de gravedad. El usuario puede cambiar el nivel de gravedad de un error que esté marcado como “discrecional”, mientras que es no es permitido cambiar el nivel de seguridad de un error marcado como “no discrecional”. La tabla 12 muestra el nivel de seguridad de los mensajes de error.

- **Sintaxis de ayuda para mensajes de error.** En *Help*, cada mensaje de error puede incluir varias partes. La información que es mostrada depende de la herramienta y del mensaje. La tabla 13 describe la sintaxis de ayuda de los mensajes de error.

- **Como observar detalles de los errores.** Cada mensaje de error de la herramienta DSP tiene asociado un texto de explicación que puede ser visto en la ventana de ayuda, seleccionando el identificador de error de seis caracteres de la página de construcción y presionando la tecla F1 de esta manera una explicación completa del mensaje de error aparecerá en la ventana de ayuda.

Tabla 12. Niveles de seguridad de los mensajes de error

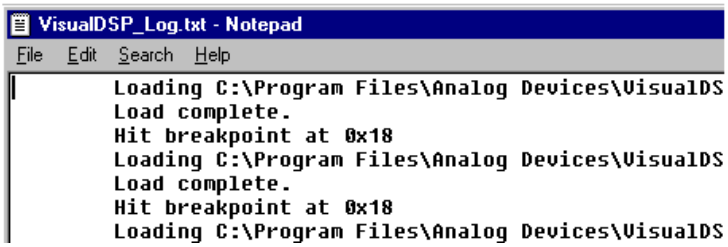
Nivel de seguridad	Descripción
Error fatal	Identifica errores tan severos se detiene la continuación del procesamiento de la señal de entrada.
Error	Identifica problemas que causan que la herramienta reporte una falla. Un error podría permitir la continuación del procesamiento de la señal de entrada para de este modo reportar problemas adicionales que puedan ser detectados.
Advertencia	Identifica situaciones que no previenen a la herramienta del procesamiento de la señal de entrada, pero puede indicar problemas potenciales.
Observación	Provee información de posible interés.

Tabla 13. Sintaxis de ayuda de los mensajes de error

Parte	Descripción
Código de identificación	Código de seis caracteres que identifican al error. Los primero dos caracteres identifican la herramienta: • ar (archiver) • cc (compiler) • ea (assembler) • el (expert linker) • li (linker) • pp (preprocessor) • vc (VILD, compiler) • vu (VCSE)
Texto de error	Texto que aparece luego del código de identificación en la ventana de salida.
Descripción	Descripción detallada del error.
Severidad	Nivel de dificultad impuesta por el error. Algunos mensajes de error pueden tomar más de un nivel de severidad.
Recuperación	Información extra provista únicamente si es utilizable.
Ejemplo	Código de ejemplo.
Como solucionar	El remedio para corregir el error.

5.2.3.3 Archivo de registro. El archivo de registro del VisualDSP++ contiene todos los estados y mensajes de error escritos en la página de consola de la ventana de salida. La figura 31 muestra un ejemplo del archivo de registro.

Figura 31. Ejemplo del archivo de registro



Fuente: Tomada del software VisualDSP3.5++

Todas las sesiones son añadidas al archivo de registro. Es recomendable ocasionalmente abrir el archivo y borrar partes de él para conservar espacio en el disco.

5.2.3.4 Personalización de la ventana de salida. En VisualDSP++ se pueden especificar preferencias que:

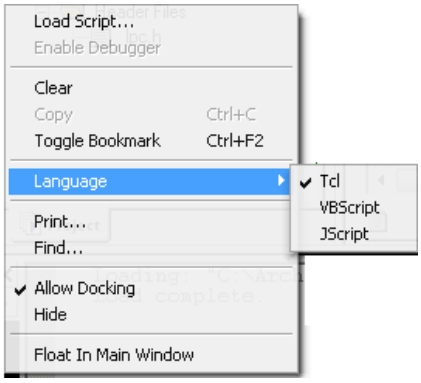
- Configuren tipo de letra y colores de la ventana de salida.
- Habiliten el comando de auto completar.

Por defecto, la ventana de salida se ubica en la parte inferior de la ventana principal de aplicación. El usuario puede redimensionar o mover la ventana de salida a diferentes partes de la pantalla, arrastrándola a la posición deseada. Del mismo modo es posible que la ventana se acople, se esconda, o flote. La página de consola de la ventana de salida puede interactuar con maquinas script.

5.2.3.5 Menú de clic derecho. El menú de clic derecho de la ventana de salida se muestra en la figura 32. Este menú permite:

- Cargar un Script o habilitar el depurador.
- Limpiar el texto en la ventana o copiar el texto seleccionado.
- Alternar favoritos.
- Seleccionar el lenguaje de scripting.
- Imprimir o encontrar texto en la ventana.
- Acoplar, esconder o flotar la ventana.

Figura 32. Menú de clic derecho de la ventana de salida



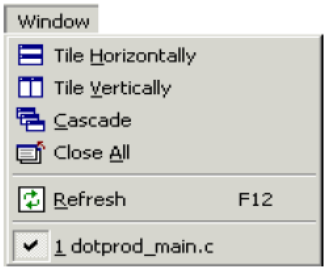
Fuente: Tomada del software VisualDSP3.5++

5.3 OPERACIONES DE VENTANA

De manera similar a las aplicaciones Windows, VisualDSP++ proporciona maneras para ajustar el modo en que se puede observar la interfaz de usuario.

5.3.1 Manipulación de la ventana. Los comandos del menú ventana mostrados en la figura 33, le permiten al usuario manipular la forma en que se muestra la ventana y al mismo tiempo actualizarla durante la ejecución del programa.

Figura 33. Comandos del menú de ventana



Fuente: Tomada del software VisualDSP3.5++

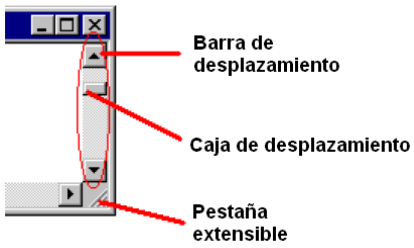
5.3.2 Menú de opciones de clic derecho. Al hacer clic derecho en una ventana o sobre su barra de título, aparecerá un menú cuyas opciones se muestran en la tabla 14.

Tabla 14. Comandos del menú clic derecho sobre ventanas

Opción	Descripción
Permitir acople	Habilita/Deshabilita acople
Cerrar	Cierra la ventana
Flotar en la ventana principal	Hace que la ventana en una ventana pequeña y deshabilita la posibilidad de acople

5.3.3 Barras de desplazamiento y pestaña extensible de redimensionamiento. Las barras de desplazamiento aparecen a lo largo del borde inferior y derecho de la ventana de aplicación, o de la ventana de documento, tal como se muestra en la figura 34.

Figura 34. Comandos del menú de ventana



Fuente: Los autores del proyecto

Las cajas de desplazamiento dentro de las barras de desplazamiento indican la localización vertical y horizontal en el documento. Utilice el mouse para desplazarse a otras partes del documento. Cuando una ventana de aplicación no se encuentra maximizada, la pestaña extensible de redimensionamiento aparece en la parte más baja de la esquina inferior izquierda de la ventana. Cliquee y arrastre la pestaña extensible para redimensionar la ventana de aplicación.

5.3.4 Ventanas: Acople Vs. Flotar. Una ventana adjunta al marco de la aplicación se conoce como ventana acoplada. Se puede separar una ventana de la ventana principal y moverla a otra posición en cualquier parte sobre el escritorio. Una ventana flotante se mantiene solitaria, debido a que no se encuentra acoplada.

Dependiendo de las necesidades es posible:

- Acoplar una ventana a la ventana principal de la aplicación.
- Flotar la ventana.

El menú de clic derecho de ventana provee los comandos para acoplar o flotar la ventana. La opción *Allow Docking y Float in a main window* son mutuamente exclusivas.

5.3.4.1 Ejemplo de una ventana acoplada. La ventana de proyectos mostrada en la figura 35 se encuentra acoplada (*Allow docking* esta seleccionado).

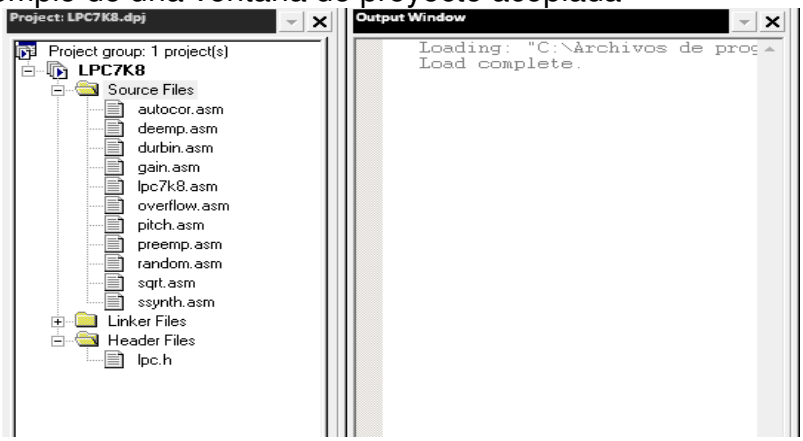
5.3.4.2 Ejemplo de una ventana flotante. La ventana de proyecto de las figuras 36 y 37 se encuentra flotando en la ventana principal (*Float in a main window* está seleccionado).

La ventana de proyecto de la figura 38 se encuentra flotando en la ventana principal (*Float in a main window* no se encuentra seleccionado).

5.3.5 Reglas del posicionamiento de ventanas. Las siguientes reglas se aplican al posicionamiento de ventanas:

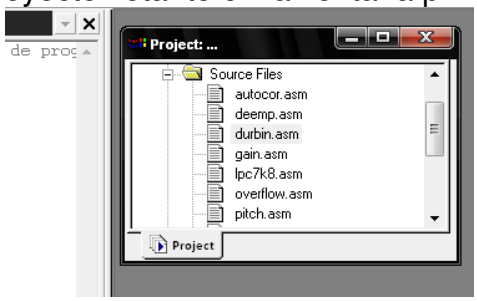
- A menos que la opción *Allow docking* esté deshabilitada una ventana residirá dentro de la ventana principal.
- Una ventana de edición no puede ser acoplada a una ventana principal.
- Una ventana especificada como ventana MDI pequeña, no puede ser posicionada sobre una ventana de salida.

Figura 35. Ejemplo de una ventana de proyecto acoplada



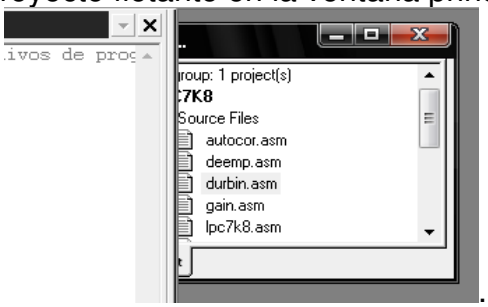
Fuente: Tomada del software VisualDSP3.5++

Figura 36. Ventana de proyecto flotante en la ventana principal (1 de 2)



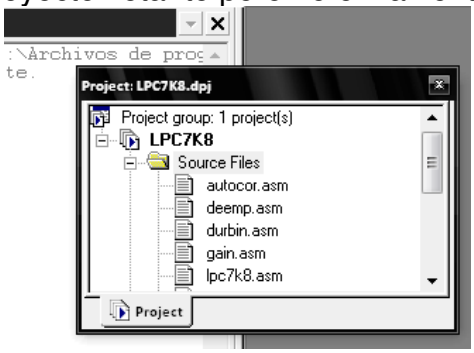
Fuente: Tomada del software VisualDSP3.5++

Figura 37. Ventana de proyecto flotante en la ventana principal (2 de 2)



Fuente: Tomada del software VisualDSP3.5++

Figura 38. Ventana de proyecto flotante pero no en la ventana principal



Fuente: Tomada del software VisualDSP3.5++

5.4 VENTANAS DE DEPURACIÓN

VisualDSP++ provee ventanas de depuración para exponer las operaciones y resultados de un programa DSP. La tabla 15 describe estas ventanas.

Tabla 15. Ventanas de depuración

Ventana	Provee
Salida	Una página de consola expone diferentes mensajes de texto tales como **load** y **status**, y mensajes de error, mientras que una página de construcción expone mensajes de construcción.
Edición	Coloración de las sintaxis, evaluación de expresiones sensitivas al contexto, favoritos, y la posición actual del PC.
Desmontaje	Código en formato de desmontaje. Esta ventana provee capacidad de lleno y volcado.
Expresiones	Los medios para ingresar una expresión y ver su valor en su momento a través de la ejecución del programa.
Trazo	Un historial de la actividad del procesador durante la ejecución del programa, incluyendo líneas de instrucción, contador de ciclos, e instrucciones ejecutadas tales como enganches de memoria, escritura de memoria de programa, y transferencia de datos/memoria.
Locales	Todas las variables locales en una función. Se utiliza esta ventana con los comando step y halt para mostrar variables que se mueven a través del programa.
Resultado lineal de los perfiles	(Solo para simulación) Muestras obtenidas de los registros del dispositivo en cada ciclo de instrucción, que proporciona una gráfica exacta de donde se ejecutan las instrucciones.
Registro personalizado	Valor actual de los registros. Se seleccionan los registros que se deseen monitorear.
Memoria	Una vista de la memoria del DSP. Formato de número similar y características de edición como en la ventana de registro, además capacidad de volcado y llenado.
Memoria BTC	Una vista de los antecedentes contenidos en el canal de telemetría en tiempo real. La ventana muestra los contenidos de las direcciones que se deseen ver.
Mapeo de memoria	El mapeo de memoria del procesador seleccionado.

Tabla 15 (Continuación). Ventanas de depuración

Ventana	Provee
Ploteo	Grafica exposición de valores de la memoria de direcciones. La ventana soporta modos de visualización para lineal y FFT (real y compleja), permitiendo exportar la imagen a un archivo, a él portapapeles, o a una impresora.
Multiprocesador	Estado actual de cada procesador en un sistema de multiprocesadores.
Visor de secuencias	Una vista de solo simulación de la secuencia de instrucción y detalles de eventos (no disponible para procesadores ADSP-218x).
Visor de caché	Análisis del uso del caché de la aplicación de un DSP, el cual es de útil en la optimización del rendimiento de la aplicación DSP
Visor de imágenes	Una vista de datos BMP, JPE, PPM, o MPEG, desde la memoria del DSP o desde un archivo en el PC. Es posible editar, imprimir, o exportar los datos de imagen

5.4.1 Ventanas de desmontaje. Por defecto, una ventana de desmontaje aparece cuando se abre una nueva sesión. También es posible abrirla seleccionando *View/Debug/Windows/Disassembly*. La figura 39 y la figura 40 muestran un ejemplo de ventana de desmontaje, una con la barra de direcciones habilitada y otra sin ella.

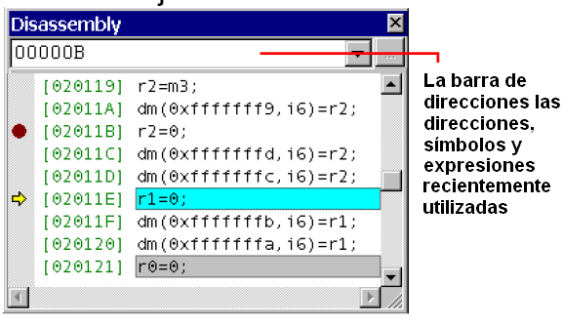
La ventana de desmontaje expone código de forma desmontada, la cuál es útil para modificar el código temporalmente para probar un cambio o para ver código cuando la fuente no está disponible. La ventana de desmontaje permite examinar el código de ensamblaje generado por el compilador C/C++. Seleccionando *View Source* del clic derecho de la ventana de desmontaje, es posible observar el código fuente C/C++ para el archivo cargado.

La ventana de desmontaje provee:

- Formato de número y características de edición, similar a la ventana registro.
- Capacidad de llenar y volcar.
- Símbolos en el extremo izquierdo de la ventana, que indican las etapas de ejecución del programa y el seguimiento de las etapas de desarrollo.

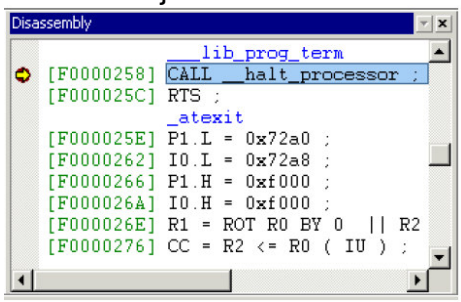
- Una barra de direcciones opcional que habilita la navegación entre direcciones, símbolos, o expresiones. La barra de direcciones mantiene un historial de las localidades más recientemente visitadas.

Figura 39. Ventana de desmontaje con barra de direcciones



Fuente: Tomada del software VisualDSP3.5++ y editada por los autores del proyecto

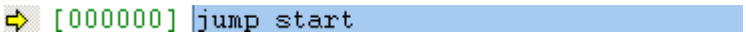
Figura 40. Ventana de desmontaje sin barra de direcciones



Fuente: Tomada del software VisualDSP3.5++

Por defecto, la línea de código fuente actual a ser ejecutada es resaltada por una barra horizontal azul, como se muestra en la figura 41.

Figura 41. Línea de código fuente actual en la ventana de desmontaje



Fuente: Tomada del software VisualDSP3.5++

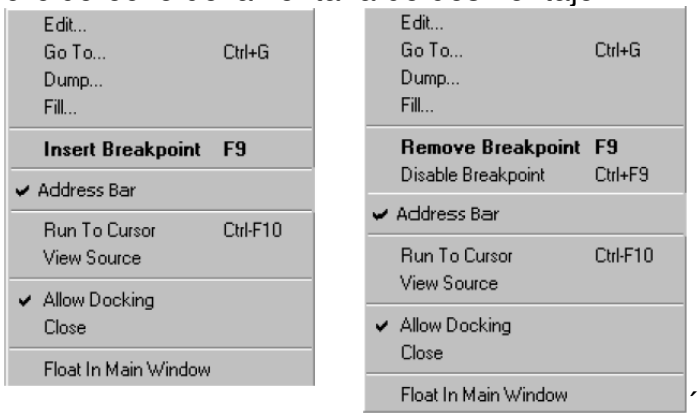
5.4.1.1 Otras características de la ventana de desmontaje. Desde la ventana de desmontaje se pueden realizar las operaciones descritas en la tabla 16.

Tabla 16. Operaciones de la ventana de desmontaje

Para…	Ubicar el puntero de mouse sobre...
Moverse a una dirección diferente	Un campo de dirección y hacer doble clic. A continuación, se selecciona la dirección a la que se desea ir. Nótese que se puede hacer uso de la barra de direcciones para navegar hacia una dirección, símbolo, o expresión.
Insertar o remover un punto de quiebre	Una instrucción y hacer doble clic.
Alternar un punto de quiebre (habilitar o deshabilitar)	Una instrucción y hacer clic derecho. A continuación, se selecciona el comando apropiado en el menú que se despliega.

5.4.1.2 Menú de opciones de clic derecho. El menú de opciones de clic derecho de la ventana de desmontaje proporciona acceso a los comandos mostrados en la figura 42.

Figura 42. Menú clic derecho de la ventana de desmontaje



Fuente: Tomada del software VisualDSP3.5++

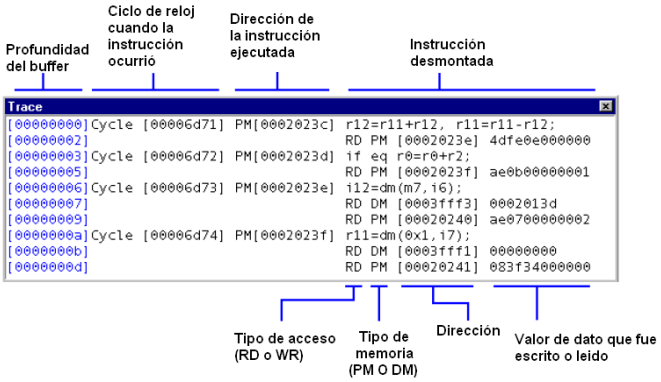
5.4.1.3 Símbolos de la ventana de desmontaje. Los símbolos ubicados en el extremo izquierdo de la ventana de desmontaje indican las etapas de la ejecución del programa. La tabla 17 muestra los símbolos de la ventana de desmontaje.

Tabla 17. Operaciones de la ventaja de desmontaje

Símbolo	Descripción
	Línea actual del código fuente
	La instrucción actual está siendo abortada por una instrucción de salto.
	Un punto de quiebre se encuentra habilitado
	Un punto de quiebre se encuentra deshabilitado.

5.4.2 Ventana trazo. Se realiza un trazo (también conocido como trazo de ejecución o trazo de programa) para analizar el comportamiento del DSP al momento de la ejecución, para habilitar las capacidades I/O, y para simular el flujo de datos entre fuente y dispositivo. Para abrir la ventana se selección View/Debug Windows/Trace. La figura 43 muestra los datos que se exponen en la ventana trazo.

Figura 43. Ejemplo de datos en la ventana trazo



Fuente: Tomada del software VisualDSP3.5++ y editada por los autores del proyecto

La ventana trazo muestra:

- Profundidad del buffer.
- El ciclo de reloj cuando la instrucción ocurrió.
- La dirección de la instrucción ejecutada.
- Instrucción desmontada.

Los resultados de memoria tienen los siguientes campos:

- Tipo de acceso (RD o WR).
- Tipo de memoria (PM o DM).
- La dirección entre corchetes ([ ]).
- El valor del dato que fue escrito o leído.

5.4.3 Ventanas de memoria. Las ventanas de memoria permiten:

- Visualizar y editar contenidos de memoria.
- Mostrar la dirección de un valor.
- Bloquear el número de columnas actualmente mostradas.

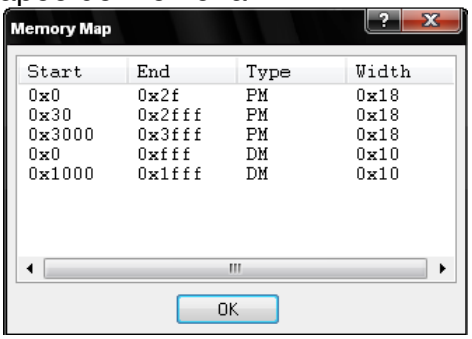
Las ventanas de memoria se abren desde el menú Memory**.** Para procesadores 21xx seleccione el tipo de memoria que se desea mostrar (Programa, Datos, Byte o I/O.

Las ventanas de memoria proporcionan:

- Formato de números y características de edición.
- Capacidad de llenado y volcado.
- Una barra de direcciones opcional para navegación veloz de las recientes direcciones utilizadas.

5.4.4 Ventana de mapeo de memoria. La ventana de mapeo de memoria (Figura 52) visualiza el mapa de memoria para el procesador seleccionado. Para abrir esta ventana se selecciona Memory/Memory Map.

Figura 44. Ventana de mapeo de memoria

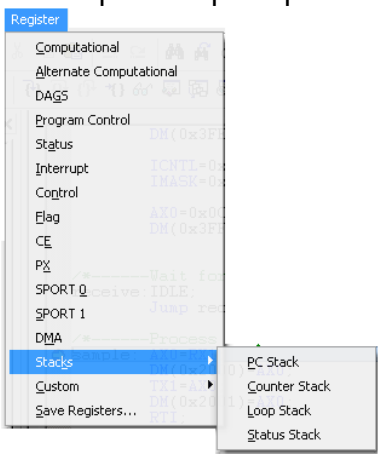


Fuente: Tomada del software VisualDSP3.5++

Si ningún programa DSP es cargado en el procesador, el mapeo de memoria visualizará toda la memoria disponible en el procesador. Si un programa es cargado en el procesador, el mapeo de memoria es el mapa definido en la sección de memoria del archivo .LDF del programa. Para cada porción de memoria, la ventana muestra la dirección de inicio, final, y ancho.

5.4.5 Ventanas de registro. Dependiendo del tipo de procesador, se tiene acceso a diferentes ventanas de registros desde el menú *Register*. El menú de registro se muestra en la figura 45

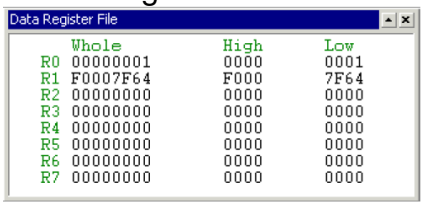
Figura 45. Ventana de registros disponible para procesadores 218x



Fuente: Tomada del software VisualDSP3.5++

La figura 46 muestra un ejemplo de archivo de datos de registro en una ventana de registro.

Figura 46. Ejemplo de ventana de registro



Fuente: VisualDSP++ 3.5 User 's Guide for 16-Bit *Processors*, cp.2-87.

Una ventana de registro le permite al usuario:

- Visualizar y cambiar los contenidos del registro.
- Cambiar la presentación (formato numérico),

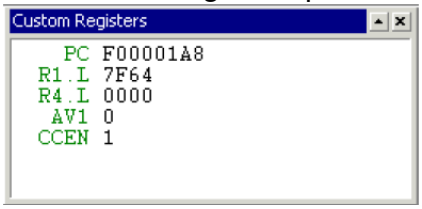
Los formatos numéricos de la ventana de registro incluyen formatos estándar, tales como hexadecimal, octal, y binario. Dependiendo del DSP, otros formatos son disponibles. El usuario puede cambiar los datos de un registro directamente desde la ventana de registro. La modificación del contenido del registro es utilizado durante la ejecución del programa. Editar datos no afecta el código fuente. Para hacer cambios permanentes, se debe editar el código fuente y reconstruir el proyecto.

5.4.6 Ventanas de pila. Dependiendo del tipo de procesador que se posea, el usuario puede tener acceso a diferentes ventanas de pila, incluyendo:

- Pila de PC.
- Pila de contador.
- Pila de bucles.
- Pila de estatus.

5.4.7 Ventana de registros personalizados. Mientras se realiza la depuración, se pueden configurar y visualizar ventanas de registros. Para crear una ventana de registros personalizados se debe seleccionar *Register/Custom/Manage*, entonces se adjuntan los registros que se quieren visualizar. La ventana de registros personalizados aparece inmediatamente después que se ha creado. Cada ventana muestra un título que le especifica al usuario únicamente los registros que se han seleccionado para monitorear. La ventana de registros personalizados mostrada en la figura 47 visualiza los contenidos de cinco registros.

Figura 47. Ejemplo de una ventana de registros personalizados



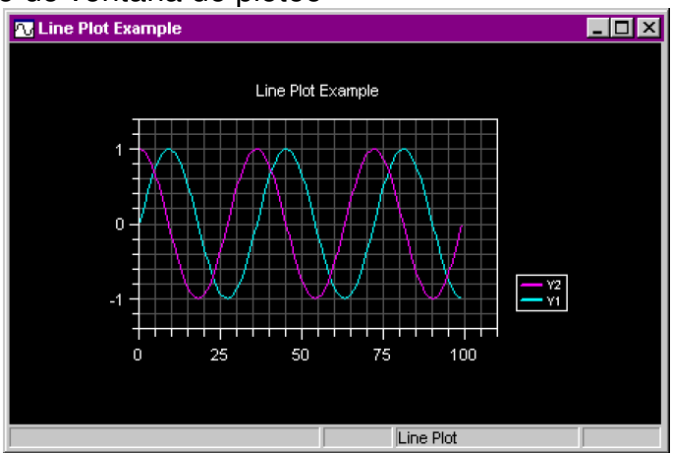
Fuente: VisualDSP++ 3.5 User 's Guide for 16-Bit *Processors*, cp.2-88.

5.4.8 Ventanas de ploteo. Se hace uso de las ventanas de ploteo para la visualización de los valores obtenidos desde la memoria del DSP. Es posible exponer uno o varios ploteos seleccionando View/Debug/Plot/New. La figura 48 muestra un ejemplo de un ploteo en la ventana de ploteo.

El usuario se encarga de elegir los contenidos dentro del ploteo, del mismo modo se pueden modificar la configuración de un ploteo e inmediatamente visualizar el ploteo con los cambios realizados. Para una ventana de ploteo es posible realizar zoom sobre una porción del ploteo y observar así los datos de un punto de datos.

El VisualDSP++ brinda la opción de imprimir, almacenar la imagen en un archivo, o almacenar los datos del ploteo en un archivo.

Figura 48. Ejemplo de ventana de ploteo

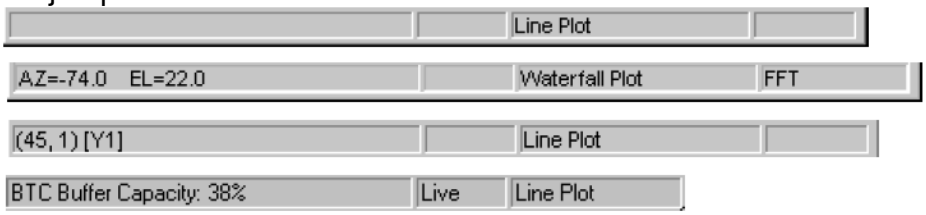


Fuente: VisualDSP++ 3.5 User 's Guide for 16-Bit *Processors*, cp.2-123.

5.4.8.1 Características de la ventana de ploteo. La ventana de ploteo incluye una barra de estado, de herramientas y un menú de clic derecho.

- **Barra de estado.** La barra de estado, localizada en la parte inferior de la ventana de ploteo, muestra el tipo de ploteo y otra información, dependiendo del tipo de ploteo y de otra configuración. En la figura 49 se muestran ejemplos que exponen distinta información en la barra de estado.

Figura 49. Ejemplos de la información mostrada en la barra de estado



Fuente: VisualDSP++ 3.5 User 's Guide for 16-Bit *Processors*, cp.2-124.

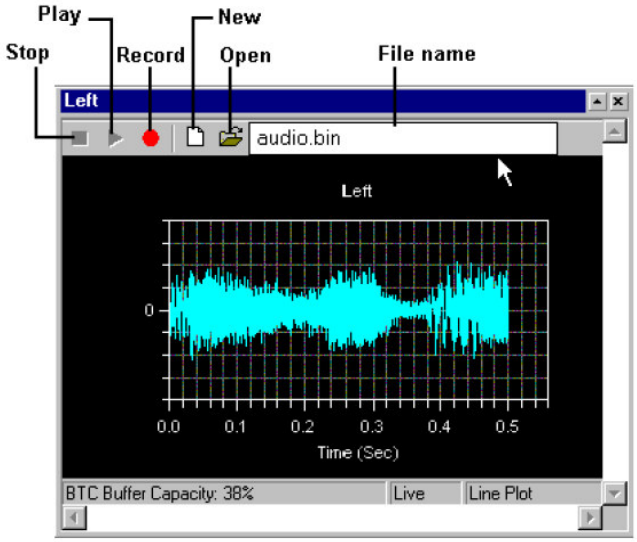
En un ploteo de tipo cascada, la barra de estado indica el azimut y la elevación del ángulo de vista. Si se realiza zoom sobre una región, la barra de estado indica que el zoom se encuentra habilitado. Cuando se hace uso del cursor de datos, la barra de estado muestra el valor de los datos en los puntos seleccionados.

Cuando el modo de auto actualización de una ventana de ploteo se encuentra activada, la barra de estado indica la capacidad actual del buffer y la condición de los registros de datos.

La capacidad del buffer para cambiar dinámicamente entre 0 y 100%, indica la porción del buffer que está actualmente en uso. El tamaño ideal es un poco por debajo de 100%. Lecturas del 100% indican pérdida de datos.

- **Barra de herramientas.** La barra de herramientas de la ventana de ploteo (ver figura 50) proporciona botones para grabar y reproducir un flujo de datos de archivos .BIN.

Figura 50. Barra de herramientas de la ventana de ploteo

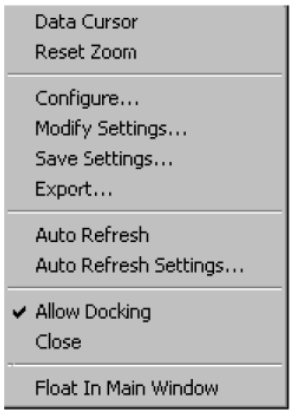


Fuente: VisualDSP++ 3.5 User 's Guide for 16-Bit *Processors*, cp.2-125.

- **Menú de clic derecho.** El menú de clic derecho de la ventana de ploteo se muestra en la figura 51.

Este menú provee acceso a las opciones estándar de la ventana (acople, cerrar, y flotar en la ventana principal), y las características de la ventana de ploteo descritas en la tabla 18.

Figura 51. Menú de clic derecho de la ventana de ploteo



Fuente: VisualDSP++ 3.5 User 's Guide for 16-Bit *Processors*, cp.2-126.

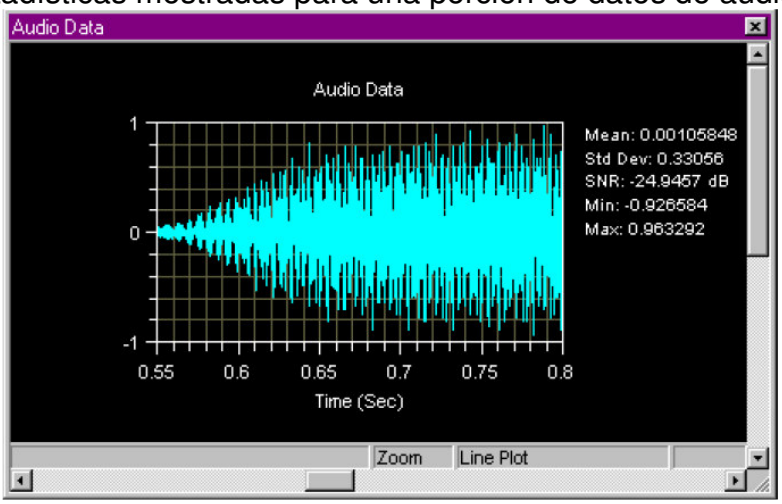
Tabla 18. Operaciones de la ventana de ploteo

Característica	**Descripción**
Cursor de datos	Expone el valor del dato asociado con la posición del cursor de datos en la ventana de ploteo.
Reset de zoom	Restablece la ventana de ploteo a su visualización de escala completa
Configuración	Abre el cuadro de diálogo de configuración del ploteo, desde el cual se puede adjuntar o remover datos establecidos.
Modificar configuración	Abre el cuadro de diálogo de configuración de ploteo, desde el cual se puede personalizar la apariencia del ploteo
Almacenar configuración	Salva las características de la configuración del ploteo para futuros usos.
Exportar	Exporta la imagen del ploteo a diferentes destinos incluyendo el portapapeles de Windows.
Auto actualización	Habilita al ploteo para actualizarse de forma automática basado en la configuración que se haya especificado.
Auto actualización de configuración	Habilita al usuario para controlar las características de auto actualización de un ploteo

5.4.8.2 Ploteo de ventana de estadísticas. Es posible ver diferentes estadísticas tales como la media, desviación estándar, relación señal a ruido (SNR), valor mínimo de datos y valor máximo de datos. Note que las estadísticas se aplican únicamente a la porción de datos que es visible. Cuando el ploteo se encuentra

con aumento (zoomed), las estadísticas son recalculadas únicamente para el área visible.

Figura 52. Estadísticas mostradas para una porción de datos de audio



Fuente: VisualDSP++ 3.5 User 's Guide for 16-Bit *Processors*, cp.2-128.

5.4.8.3 Configuración del ploteo. Una configuración del ploteo está compuesta por dos partes: valor de los datos y características de la presentación (configuración).

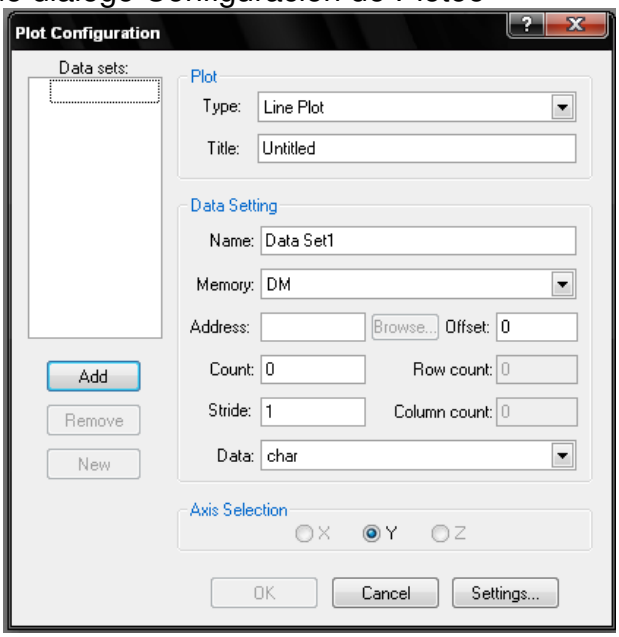
Una ventana de ploteo contiene por lo menos un set de datos, una serie de valores de datos en la memoria del DSP. El set de datos es creado en el cuadro de diálogo Configuración del ploteo, mostrado en la figura 53.

El usuario es quien especifica el tipo de ploteo, la localidad de memoria, el número de valores, el axis asociado a cada set de datos, y otras opciones que identifican lo información o los datos. Nótese que ploteos en 3-D requieren especificaciones adicionales.

El usuario es quien especifica el tipo de ploteo, la localidad de memoria, el número de valores, el axis asociado a cada set de datos, y otras opciones que identifican lo información o los datos. Nótese que ploteos en 3-D requieren especificaciones adicionales.

Los botones de configuración permiten configurar las opciones de presentación tales como títulos, grillas, fuentes, colores, y los valores axis de cada set de datos, VisualDSP++ utiliza esta configuración y lee la memoria del DSP para mostrar un ploteo en una ventana de ploteo.

Figura 53. Cuadro de diálogo Configuración de Ploteo



Fuente: Tomada del software VisualDSP3.5++

5.4.8.4 Presentación de la ventana de ploteo. En VisualDSP++ se puede personalizar la presentación de una ventana de ploteo para que se ajuste a las necesidades del usuario. La configuración de la presentación se realiza desde el cuadro de diálogo Configuración de ploteo, el cual se invoca de la siguiente manera:

Haciendo clic derecho dentro de la ventana de ploteo.
Haciendo clic en el botón de configuración en el cuadro de diálogo Configuración de ploteo.

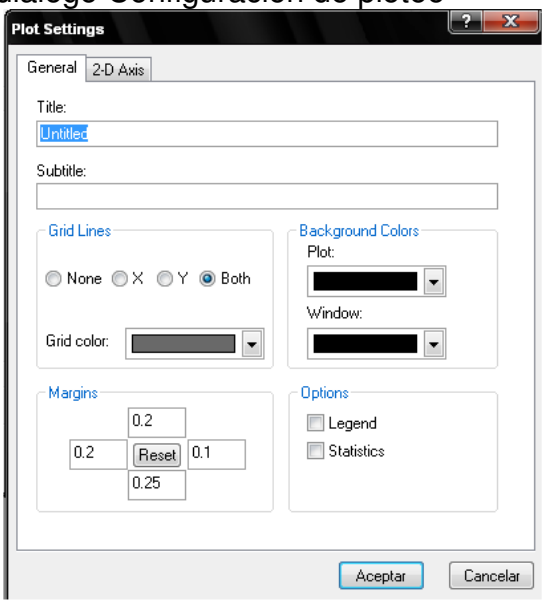
Las opciones dentro del cuadro de diálogo Plot Settings permiten configurar la presentación de la ventana de ploteo. La figura 54 muestra las opciones de configuración provistas por el cuadro de diálogo.

5.4.9 Visor de imágenes. La ventana visor de imágenes del VisualDSP++ permite que el usuario realice las siguientes operaciones:

- Ver una imagen, que puede ser un archivo de datos BMP, JPEG, PPM, ó MPEG, proveniente de la memoria del DSP o de un archivo en el PC.
- Configurar los atributos de la imagen tales como número de pixeles, bits por pixel, y el formato de la imagen.

- Corregir la gama de atributos de una imagen. Para una imagen de color se pueden ajustar los valores de pixel para rojo, verde, y azul. En una imagen en escala de grises, es posible ajustar solamente el nivel de oscuridad.
- Copiar una imagen a portapapeles de Windows.
- Imprimir una imagen o salvarla en un archivo.
- Exportar una imagen

Figura 54. Cuadro de diálogo Configuración de ploteo



Fuente: Tomada del software VisualDSP3.5++

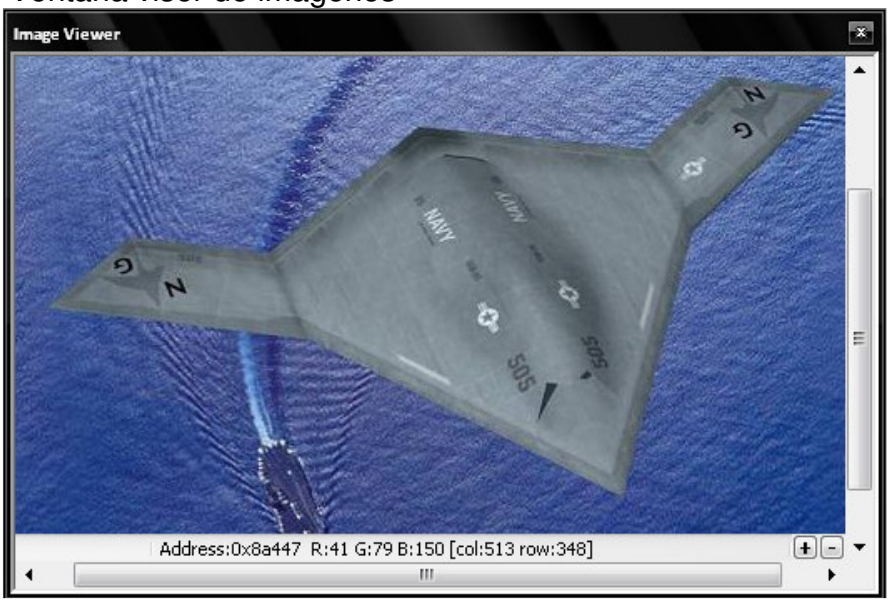
El usuario selecciona la fuente de la imagen y especifica los atributos de la misma. Si la imagen es cargada en la memoria del DSP, se debe especificar la dirección, el tamaño, y el formato de la imagen. Para abrir el visor de imágenes se selecciona View/Debug/Windows/Image Viewer.

La ventana visor de imágenes, mostrada en la figura 55, muestra la imagen y proporciona barras de desplazamiento y botones para lograr aplicación o distanciamiento de la imagen.

Cuando el puntero del mouse se mueve sobre la imagen, la barra de estado indicará:

- La dirección del DSP donde el pixel seleccionado se encuentra localizado.
- Valores de pixel de colores RGB, intensidad para escala de grises.
- Coordenadas del pixel (fila columna).

Figura 55. Ventana visor de imágenes



Fuente: Los autores del proyecto.

5.5 HERRAMIENTAS DE SIMULACIÓN

Antes de realizar cualquier operación con el procesador, se pueden generar interrupciones y flujo de datos dentro de VisualDSP++ para simular el comportamiento del procesador.

5.5.1 Interrupciones. Se utilizan las interrupciones para simular las interrupciones externas en el programa. Cuando se hace uso de las interrupciones con puntos de observación y flujo de datos, el programa simula operación del mundo real para el sistema DSP.

5.5.2 Simulación Entrada/Salida (Flujo de datos). En muchos productos, los procesadores existen como partes de un sistema muy extenso donde ellos pueden actuar como maestro o esclavo. Ellos a su vez pueden manejar otros dispositivos, o tomar parte en el procesamiento de un subconjunto de datos. Debido a sus extensas capacidades de I/O, los procesadores Analog Devices sobresalen en estas funciones. Se puede hacer uso de flujo de datos para transmitir datos entre:

- Un dispositivo y un archivo.
- Un dispositivo y otro dispositivo.

- Un dispositivo en un procesador y un dispositivo en otro procesador, en un sistema de multiprocesador.

Gracias a la tecnología del canal de telemetría, VisualDSP++ permite el flujo de datos desde un dispositivo DSP sin detener el DSP. Esta capacidad se aplica a tanto la simulación como a la emulación.

La ventana de ploteo recibe y muestra un flujo de datos desde la memoria del DSP. Si el dispositivo soporta telemetría, la ventana de ploteo lee la memoria y actualiza la gráfica mostrada sin detener el dispositivo, de lo contrario, la ventana de ploteo detiene el procesador, lee la memoria, actualiza la gráfica, y reanuda el procesador.

La ventana de ploteo permite que datos fluyan desde o hacia un archivo de datos binario. El archivo se puede convertir en un formato ASCII para ingresarlo en otras aplicaciones tales como MATLAB y Excel. La aplicación DSP podrá recopilar y transferir datos en cuatro diferentes formas:

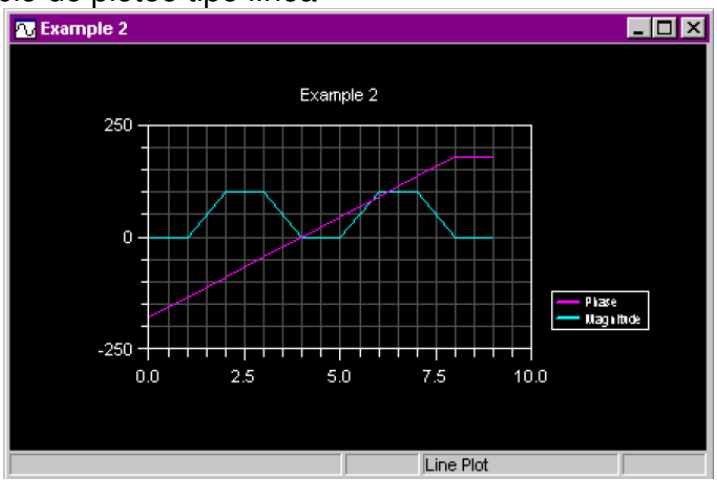
Muestreando un punto de prueba en el tiempo.
Transfiriendo un arreglo de datos sobre el canal de telemetría como un punto específico en la aplicación DSP.
Utilizando directamente GetMem().
Deteniendo periódicamente el dispositivo para leer la memoria.

5.5.3 Ploteos. La capacidad de ploteo de datos del VisualDSP++ ayuda a visualizar datos en la memoria del procesador. Los diferentes tipos de ploteos en VisualDSP++ son:

- Línea
- X-Y
- Constelación
- Diagrama de ojo
- Cascada
- Espectrograma

5.5.3.1 Ploteo tipo línea. Un ploteo tipo línea (figura 56) muestra un rango de valores en la memoria del procesador, conectados por una línea. Los valores leídos desde la memoria del procesador son asignados al Y-Axis. Los valores correspondientes al X-.Axis son automáticamente generados.

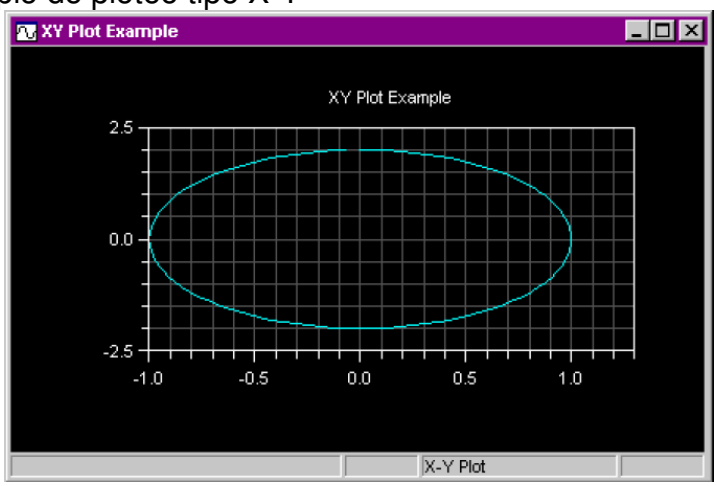
Figura 56. Ejemplo de ploteo tipo línea



Fuente: VisualDSP++ 3.5 User 's Guide for 16-Bit *Processors*, cp.3-19.

5.5.3.2 Ploteo tipo X-Y. Un ploteo tipo X-Y (figura 57) requiere un valor X y un valor Y para cada punto de datos, diferente a un ploteo tipo línea, un ploteo X-Y requiere los datos correspondientes al X-Axis.

Figura 57. Ejemplo de ploteo tipo X-Y

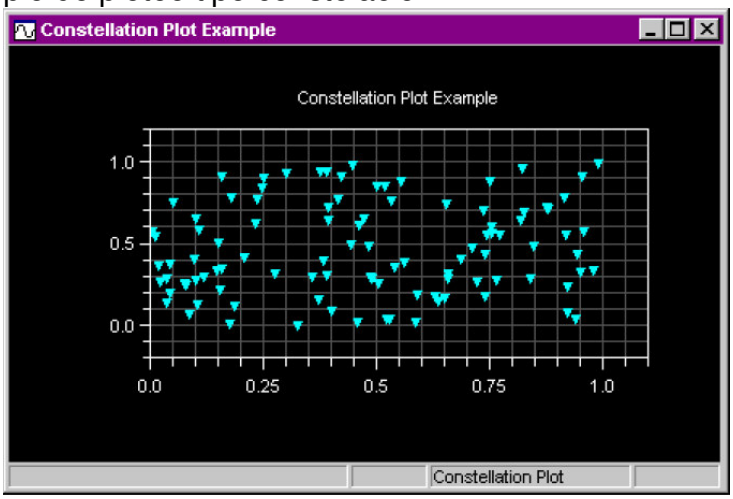


Fuente: VisualDSP++ 3.5 User 's Guide for 16-Bit *Processors*, cp.3-20.

Los datos X y los datos Y están especificados de forma separada en un localidad de memoria definida por el usuario. El número de X e Y puntos debe ser igual.

5.5.3.3 Ploteo tipo constelación. Un ploteo tipo constelación (figura 58) muestra un símbolo en cada punto de datos (X, Y).

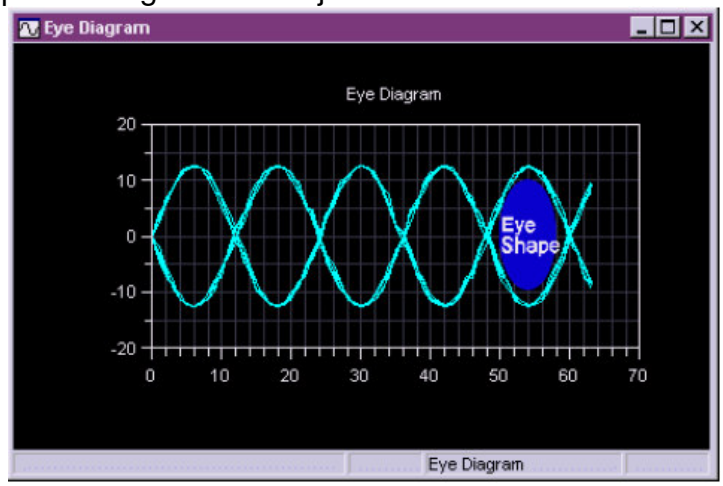
Figura 58. Ejemplo de ploteo tipo constelación



Fuente: VisualDSP++ 3.5 User 's Guide for 16-Bit *Processors*, cp.3-21.

5.5.3.4 Diagramas de ojo. Un ploteo diagrama de ojo (figura 59) es típicamente utilizado para mostrar la estabilidad de una señal basada en el tiempo.

Figura 59. Ejemplo de diagramas de ojo



Fuente: VisualDSP++ 3.5 User 's Guide for 16-Bit *Processors*, cp.3-22.

Esta gráfica funciona como un osciloscopio de almacenamiento que muestra una historia de señales superpuestas en el tiempo. El diagrama de ojo procesa los datos de entrada y opcionalmente busca un punto de cruce de umbral. Una traza es graficada cuando el umbral a cruzar es alcanzado. El ploteo continúa para el resto de la traza de datos.

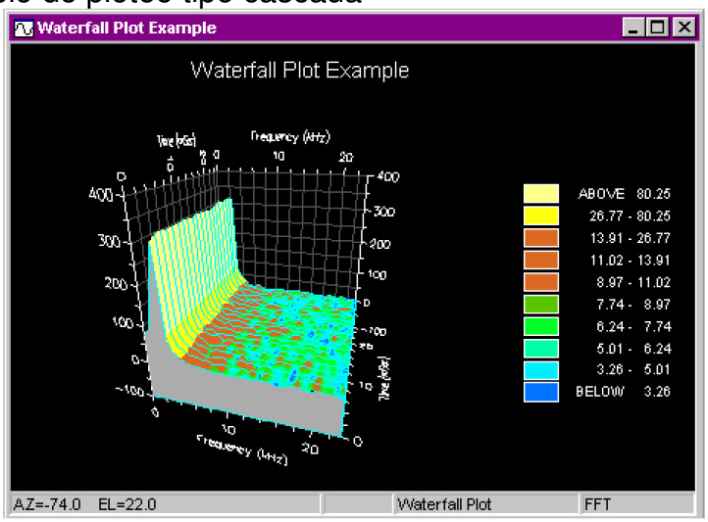
Cuando ocurre un punto de quiebre, el ploteo de datos es actualizado y una nueva traza es graficada. El diagrama de ojo utiliza una técnica de cambio de datos que almacena el número decidido de trazas en un buffer de ploteo. Cuando el número de trazas es excedido, la primera traza sale del buffer y la nueva traza se almacena dentro de la última localidad del buffer.

El usuario puede modificar las opciones del valor de umbral, activador de crecimiento, activador de decrecimiento, y el número de superposición de trazas.

5.5.3.5 Ploteo tipo cascada. Un ploteo tipo cascada (figura 60) es típicamente utilizado para mostrar el cambio en la frecuencia de una señal sobre el tiempo.

La gráfica está compuesta de múltiples trazas de líneas graficadas en vista 3-D. cada traza graficada representa un tramo de la gráfica en cascada. La manera más sencillas de crear un ploteo tipo cascada es definiendo un arreglo en 2-D en código C (una grilla). El arreglo de primera dimensión es el número de filas en la grilla, y el arreglo de segunda dimensión es el número de columnas en la grilla. El número de columnas es igual al número de puntos de datos en cada trazo de línea.

Figura 60. Ejemplo de ploteo tipo cascada

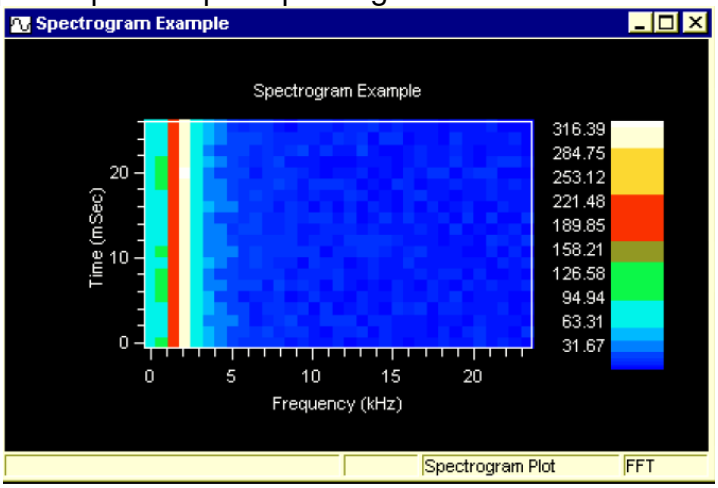


Fuente: VisualDSP++ 3.5 User 's Guide for 16-Bit *Processors*, cp.3-23.

El usuario puede utilizar un mapa de colores sobre la página Axis 3-D del cuadro de diálogo Configuración de Colores, para mejorar las características de la gráfica. Cada color corresponde a un rango de valores de amplitud. La grafica que se muestra tiene un texto que a cada color asociado con cierto rango de valores. El ploteo en cascada se puede rotar a cualquier azimut y elevación utilizando las teclas flecha del teclado.

5.5.3.6 Ploteo tipo espectrograma. Un ploteo tipo espectrograma (figura 61) muestra la misma información de un ploteo tipo cascada 3-D, exceptuando que el espectrograma lo hace en formato 2-D.

Figura 61. Ejemplo de ploteo tipo espectrograma



Fuente: VisualDSP++ 3.5 User 's Guide for 16-Bit *Processors*, cp.3-24.

6 INSTRUCCIONES DE REFERENCIA ESTABLECIDAS DE ASSEMBLER

6.1 MODELO DE PROGRAMACION

Desde un punto de vista de programación, el ADSP-218x DSP consta de tres unidades computacionales (ALU, MAC y Shifter), dos generadores de direccionamiento de datos y un secuenciador de programa, además de un dispositivo de periféricos y una memoria que varia con cada procesador. Casi todas estas operaciones usan esta arquitectura, los componentes requieren uno o más registros para guardar datos, hacer seguimiento de valores tales como punteros o especificar modos de operación. La eficiente transferencia de datos es conseguida con el uso de cinco buses internos (ver figura 62):

Bus de Direcciones de la Memoria de Programa (PMA)
Bus de Datos de la Memoria de Programa (PMD)
Bus de Direcciones de la Memoria de Datos (DMA)
Bus de Datos de la Memoria de Datos (DMD)
Bus de Resultados (R)

Los registros internos contienen datos, direcciones, información de control o información de estado. Por ejemplo, AX0 guarda un operando ALU (dato); I4 guarda un puntero DAG2 (dirección), ASTAT contiene banderas de estado de operaciones aritméticas.

Hay dos tipos de acceso para los registros. El primer tipo de acceso está dedicado a los registros como MX0 y IMASK. Estos registros pueden ser leídos y escritos explícitamente en lenguaje assembler. Por ejemplo:

```
MX0=1234;
IMASK=0XF:
```

El segundo tipo de acceso es hecho para el mapeo de registros de memoria como los registros del control del sistema, registros de control de estado de espera, registros de temporizadores y registros de los puertos (SPORT). Estos registros son accedidos mediante la lectura y escritura de la localización de los datos de memoria correspondientes.

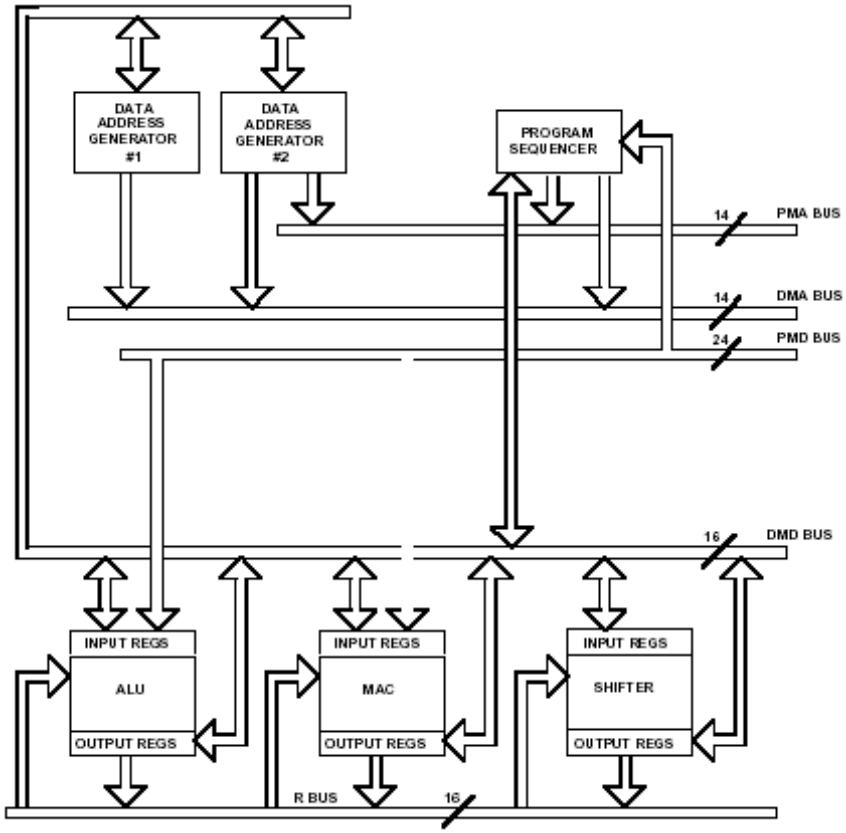
Por ejemplo el siguiente código limpia el registro de control de estado de espera, el cual es mapeado a la localización del dato de memoria 0x3FFE

AX0=0:
DM(0x3FFE)=AX0;

En este ejemplo, AX0 es usado para mantener la constante 0 debido a q no existe una instrucción para escribir un valor de dato inmediato a la memoria usando una dirección inmediata.

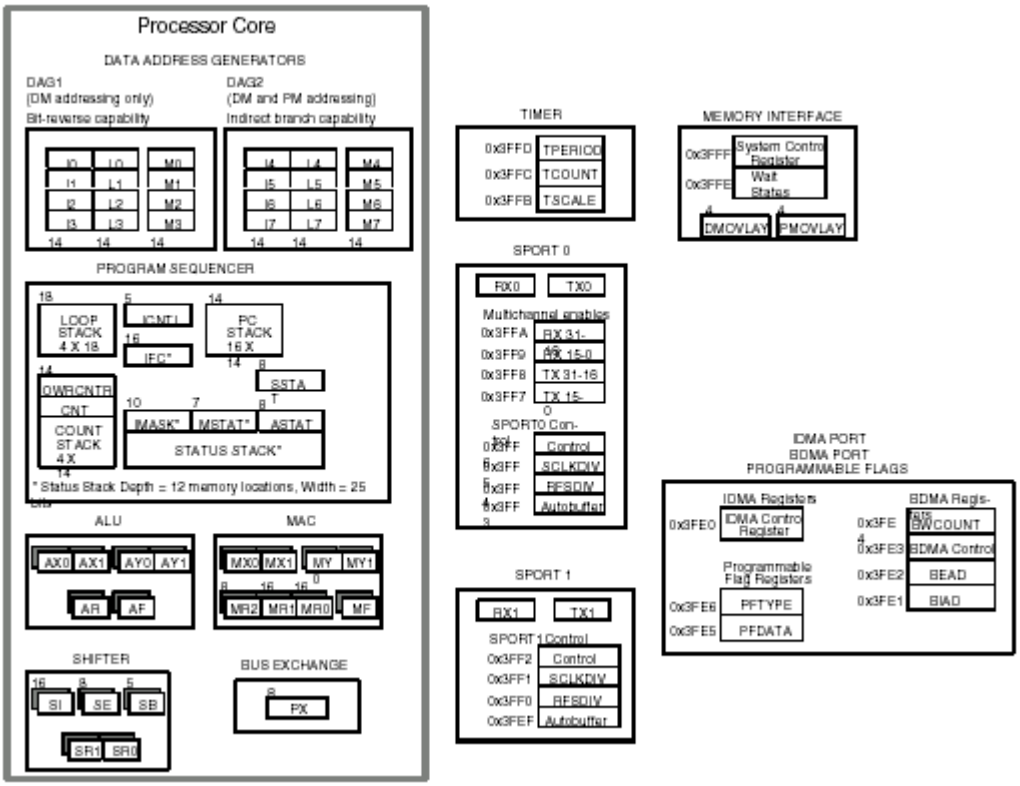
Los registros ADSP-218x son mostrados en la figura 63. Estos registros son agrupados por función: Generadores de direccionamiento de datos (DAGs), secuenciador de programa, unidades computacionales (ALU, MAC y Shifter), bis de intercambio (PX), interface de memoria, temporizadores, SPORTs, interface principal e interface DMA

Figura 62. Diagrama de bloques de la arquitectura del núcleo interno



Fuente: Digital Signal Processing Laboratory, NORTHEASTERN UNIVERSITY, Pag 4

Figura 63. Registros de ADSP-218x



Fuente: ADSP-218x DSP Instruction Set Reference

6.1.1 Generadores de direccionamiento de datos. DAG1 y DAG2 cada uno tiene 12 registros de 14-bit: cuatro registros indexados (I) para almacenamiento de punteros, cuatro registros modificables (M) para actualización de punteros y cuatro registros de longitud (L) para implementación de buffers circulares. DAG1 solo direcciona memoria de datos y tiene la capacidad de revertir sus bits de salida. DAG2 direcciona el programa y la memoria de datos, y puede proveer direcciones para sucursales indirectas (saltos y llamadas) tan bien como para acceder a datos. La figura 64 muestra el diagrama de bloques de un simple generador de direccionamiento de datos.

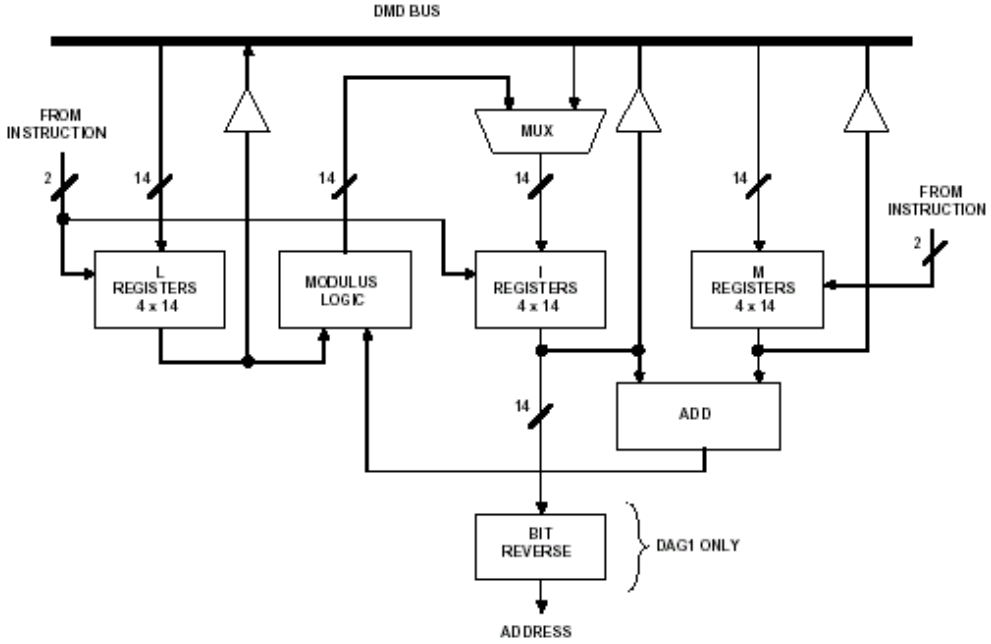
El siguiente ejemplo es una memoria de datos indirecta leída desde la localización del puntero dada por I0. Una vez la lectura se completa, I0 es actualizada por M0.

AX0=DM(I0,M0)

El siguiente ejemplo es un dato de memoria de programa indirecto escrito por la dirección del puntero dado por I4, el cual se modificara después por M5

PM(I4,M5)=MR1

Figura 64. Diagrama de bloques de un generador de direccionamiento de datos



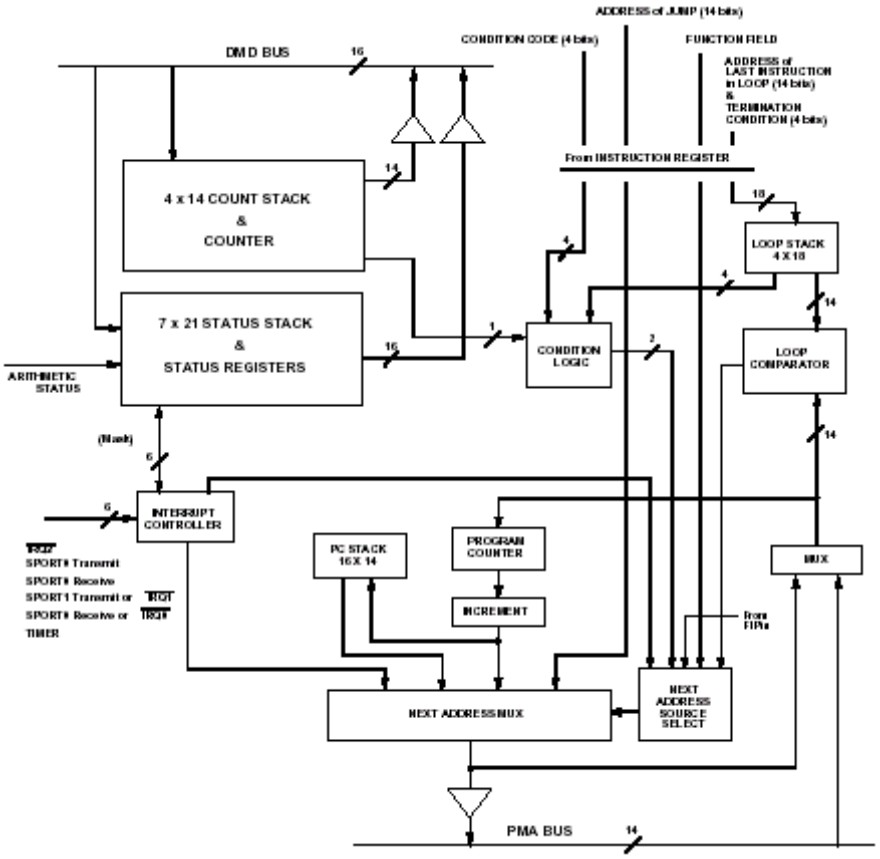
Fuente: Digital Signal Processing Laboratory, NORTHEASTERN UNIVERSITY, Pag 13

6.1.2 Secuenciador de programa. Los registros asociados con el secuenciador de programa controlan las subrutinas, interrupciones y bucles. Ellos también indican el estado y seleccionan los modos de operación. La figura 65 muestra el diagrama de bloques para el secuenciador de programa y las secciones de estado.

6.1.2.1 Interrupciones. El registro ICNTL controla las interrupciones anidadas y la sensibilidad de las interrupciones externas. El registro IFC el cual es de 16 bits de ancho permite forzar y limpiar las interrupciones en software. El registro IMASK el cual es de 10 bits de ancho enmascara (deshabilita) interrupciones individuales. Los procesadores ADSP-218x soportan 12 interrupciones, dos de las cuales (RESET y APAGADO) no son enmascarables.

El ADSP-2181 DSP soporta una instrucción que habilita una interrupción global (ENA INTS) y una instrucción que deshabilita una interrupción (DIS INTS). Al ejecutar la instrucción de deshabilitar interrupciones causa que todas las interrupciones sean deshabilitadas sin cambiar el contenido del registro IMASK.

Figura 65. Diagrama de bloques del secuenciador de programa



Fuente: Digital Signal Processing Laboratory, NORTHEASTERN UNIVERSITY, Pag 15

6.1.2.2 Bucles de contadores. El registro CNTR guarda el valor del contador para la ejecución del bucle actual. La pila del contador permite el anidado del contador basado en bucles de cuatro niveles. Al escribir en CNTR se empuja el valor actual dentro de la pila del contador antes de escribir el nuevo valor. El siguiente ejemplo empuja el valor actual de CNTR en la pila del contador y luego carga a CNTR con 10.

```
CNTR=10;
```

OWRCNTR es una sintaxis especial con la cual se puede sobrescribir el valor del contador por el bucle actual sin pulsar CNTR en la pila del contador.

6.1.2.3 Bits de estado y modo. El registro de pila de estado (SSTAT) contiene banderas llenas y vacías para pilas. El registro de estado aritmético (ASTAT) contiene banderas para las unidades computacionales. El registro de estado de modo (MSTAT) contiene bits de control para opciones varias.

6.1.2.4 Pilas. El secuenciador de programa contiene cuatro pilas que permiten bucles, subrutinas e interrupciones anidadas.

Las pilas del PC son de ancho de 14 bits y de 16 lugares de profundidad. Estas almacenan las direcciones de retorno de subrutinas, rutinas de servicio de interrupción, y direcciones en el tope del bucle para los lazos. El manejo de las pilas del PC es automático para llamado de subrutinas y manejo de interrupciones. En adición, las pilas del PC pueden ser manualmente puestas o guardadas usando las instrucciones de control de pila de PC TOPPCSTACK=reg y reg=TOPPCSTACK

La pila de bucle es de 18 bits de ancho, 14 bits para la dirección final del bucle y 4 bits para la condición de terminación del código. La pila de bucle es de 4 localidades de profundidad. Esta es automáticamente cargada durante la ejecución de una instrucción DO UNTIL. Esta es guardada automáticamente durante la salida del bucle si el bucle esta anidado. La pila de bucle debe ser guardada manualmente con la instrucción POP LOOP.

El estado de la pila, la cual es automáticamente cargada cuando el procesador sirve una interrupción, acomodando el registro de mascara de interrupción (IMASK), el registro de estado de modo (MSTAT) y el registro de estado aritmético (ASTAT). La profundidad y ancho de la pila de estado varía con cada procesador, desde que cada uno de los procesadores tenga un número diferente de interrupciones. La pila de estado es automáticamente guardada cuando la instrucción de retorno de interrupción (RTI) es ejecutada. La pila de estado puede ser cargada o guardada manualmente con las instrucciones PUSH STS y POP STS.

La pila de contadores es de 14 bits de ancho y mantiene el valor del contador (CNTR) para bucles basados en contadores anidados. Esta pila es cargada automáticamente con el valor de CNTR actual cuando hay escritura en CNTR. La pila de contadores puede ser guardada automáticamente con la instrucción POP CNTR

6.1.3 Unidades Computacionales. Todo procesador ADSP 21xx contiene tres unidades computacionales, unidad aritmética y lógica (ALU), unidad multiplicadora y acumuladora (MAC) y unidad de desplazamiento (Shift). Estas pueden operar con datos de 16 bits o de multiprecisión y solo almacenan datos

6.1.3.1 ALU. Proporciona el set estándar de funciones aritméticas y lógicas, la figura 66 muestra el diagrama de bloques. Los operadores de entrada son X y Y y el operador de salida es R, todos estos de 16 bits. Esta unidad acepta una señal de carry-in, que es el bit de carry del registro de estado del procesador aritmético (ASTAT). La ALU genera seis señales de estado o comúnmente llamadas Flags: cero (AZ), negativo (AN), carry (AC), overflow (AV), signo de operador de entrada X (AS), signo de cociente (AQ). Todas estas señales al fin de un ciclo están representadas en el registro ASTAT.

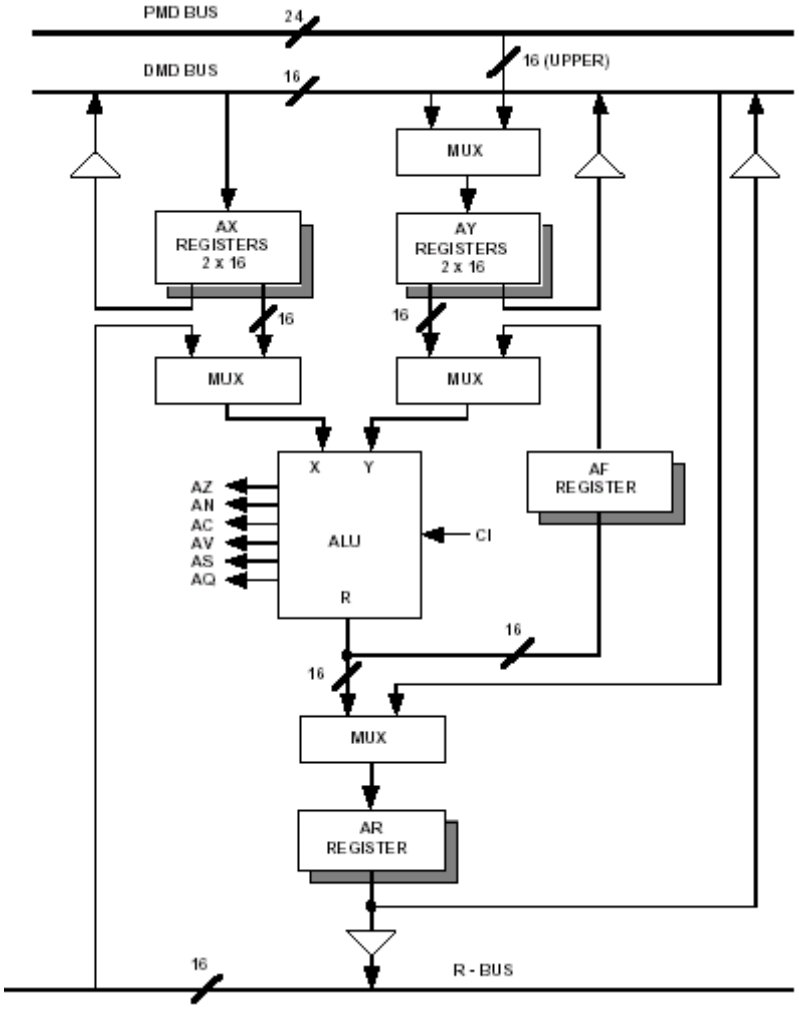
Los registros de entrada X de la ALU aceptan datos de dos fuentes: los registros AX y del bus de resultados (R). El bus R conecta los registros de salida de todas las unidades computacionales, permitiendo ser usados como operadores de entrada directamente. El registro AX está conformado por dos registros de 16 bits cada uno: AX0 y AX1. Estos registros pueden ser leídos y escritos desde el bus DMD.

Los registros de entrada Y de la ALU aceptan también datos de dos fuentes: el registro AY y el registro de realimentación de la ALU (AF). El registro AY está compuesto por dos registros de 16 bits cada uno: AY0 y AY1.

La salida de la ALU puede ser cargada en el registro de alimentación (AF) o en el registro de resultados (AR). El AF es un registro interno de la ALU y puede ser usado directamente como un operador de entrada Y. El registro AR puede ser manejado por el bus DMD o R y solo puede ser cargado directamente por el bus DMD.

6.1.3.2 MAC. Nos permite realizar operaciones de multiplicaciones y acumulación simultáneamente, la figura 67 muestra el diagrama de bloques del multiplicador/acumulador. Para realizar estas operaciones la MAC tiene dos grupos de operadores de entrada: X y Y de 16 bits y un operador P donde guarda el producto, el mismo es de 32 bits. Si la operación que deseamos realizar es de multiplicación, el operador P para directamente al registro MR (multiplier result) y si deseamos realizar un producto y acumulación, el operador P es sumado o restado al registro MR, quedando en MR el resultado final. El registro MR es de 40 bits y está formado por tres registros: MR2, MR1 y MR0. Estos dos últimos son de 16 bits cada uno y MR2 es de 8 bits.

Figura 66. Diagrama de bloques ALU



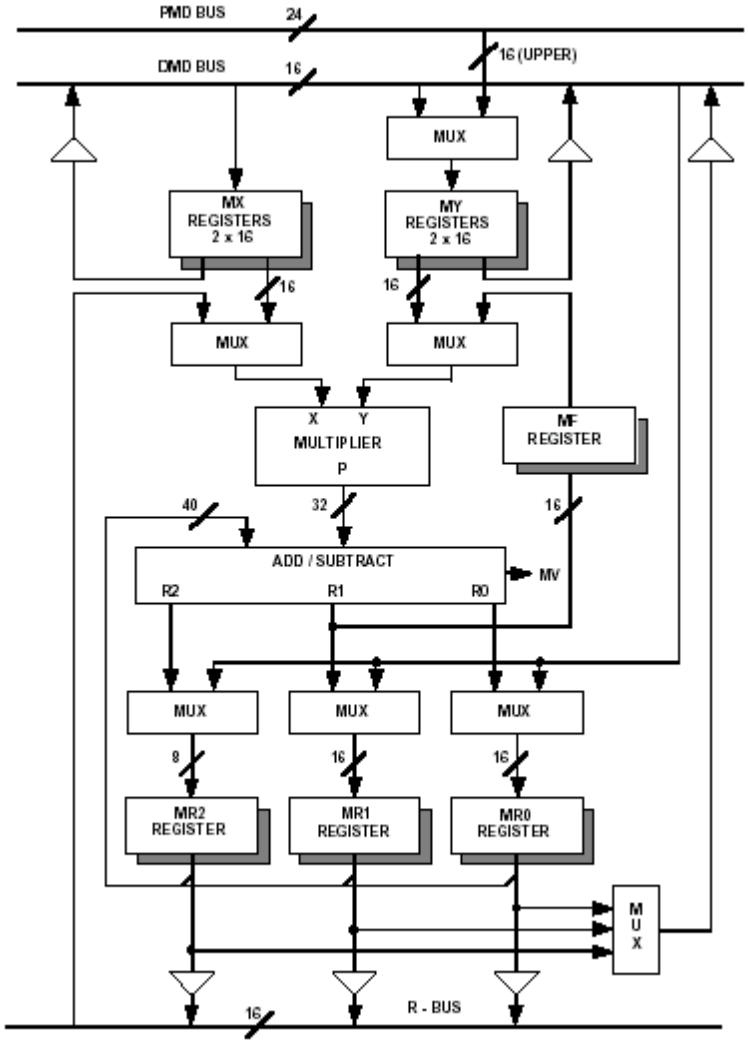
Fuente: Digital Signal Processing Laboratory, NORTHEASTERN UNIVERSITY, Pag 7

La adición y sustracción generan overflow y el bit que representa esto es el MV (MAC overflow) que su estado se puede ver en el ASTAT. Los registros de entrada y salida de la MAC son similares a los de la ALU. Los registros de entrada X aceptan datos de dos fuentes: MX y del bus de resultados (R). Hay dos registros dentro MX, MX0 y MX1. Estos registros pueden ser leídos y escritos desde el bus DMD. Los registros de entrada Y aceptan también datos de dos fuentes: el registro MY y el registro MF. El registro MY está compuestos por dos registros de 16 bits cada uno: M0 y M1. Estos registros pueden ser leídos y escritos desde el bus DMD y pueden ser escritos desde el bus PMD. La salida de la adición o la sustracción puede ser cargada en el registro MF o MR. El registro MF es también

un registro de realimentación en el cual están los bits 16-31 del resultado y puede ser usado directamente como operador de entrada Y en el próximo ciclo

El MAC del ADSP 21xx soporta dos modos para multiplicar y acumular: modo fraccionario para números fraccionarios y modo entero para números enteros. La selección del modo está representada en el bit 4 de la MSTAT. Si este bit vale 1, el modo seleccionado es el entero sino esta seleccionado del modo fraccional. La instrucción para realizar este cambio es (ENA o DIS) M_MODE.

Figura 67. Diagrama de bloques MAC



Fuente: Digital Signal Processing Laboratory, NORTHEASTERN UNIVERSITY, Pag 9

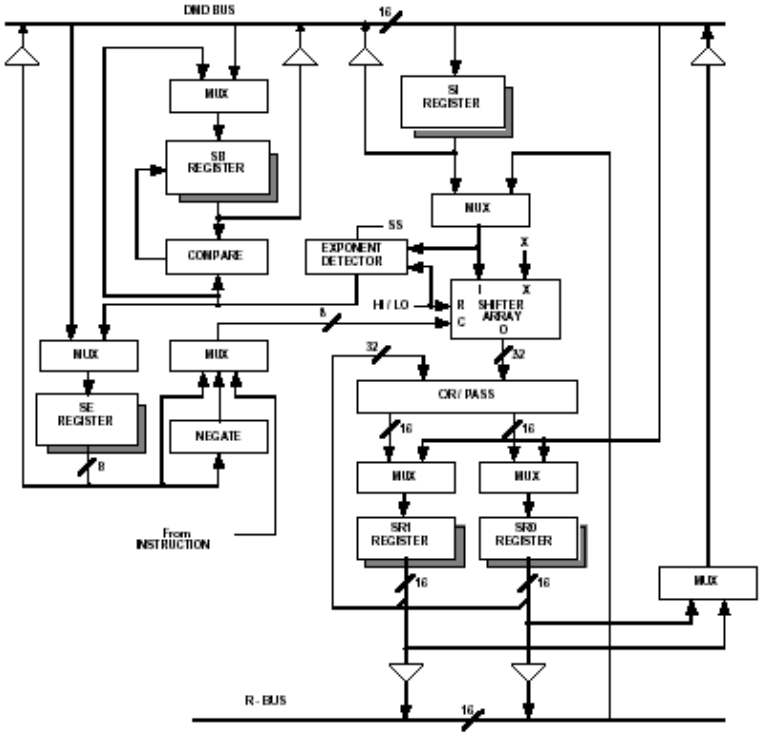
6.1.3.3 SHIFTER

Provee un completo set de funciones de desplazamiento (shifting) para datos de 16 bits de entrada dejándolos de 32 bits. Incluye shift aritmético, shift lógico, normalización y detección del máximo exponente. También permite detectar exponente y ajustarlo. Es capaz de realizar funciones básicas con números en formato punto flotante. La figura 68 muestra el diagrama de bloques de el Shifter

El Shifter tiene cuatro registros principales:

- SI: "*shifter input*" se utiliza para cargar los datos de entrada del array de desplazamiento y del detector de exponente. 16 bits.
- SR: "*shifter result*" se utiliza para guardar los datos de salida. 32 bits. Esta dividido en dos registros de 16 bits cada uno: SR0 y SR1
- SE: "*shifter exponent*" se utiliza para cargar el exponente en complemento a dos durante la normalización y desnormalización de números. 8 bits
- SB: "*shifter block*" se utiliza para cargar el valor del menor exponente posible y ser actualizado con el valor del exponente del numero en proceso si este es mayor que el valor cargado anteriormente en SB.

Figura 68. Diagrama de bloques SHIFTER



Fuente: Digital Signal Processing Laboratory, NORTHEASTERN UNIVERSITY, Pag 11

6.2 SET DE INSTRUCCIONES

El set de instrucciones proporciona movimientos desde algún registro a otro, o desde la mayoría de registros a/desde cualquier memoria. Para la combinación de operaciones, a casi cualquier operación ALU, MAC o Shifter puede ser combinada con algún movimiento registro a registro o con un movimiento de registro hacia o desde la memoria interna o externa.

Existen 5 categorías básicas de instrucciones:

- Instrucciones computacionales: ALU, MAC, Shifter
- Instrucciones de movimientos
- Instrucciones de control del flujo de programa
- Instrucciones multifunción
- Instrucciones misceláneas

El set de instrucciones está adaptado a los algoritmos intensivos de computación comunes en las aplicaciones DSP. El set de instrucciones provee un control total de las tres unidades computacionales del procesador. Las instrucciones aritméticas pueden procesar operandos sencillos con precisión de 16 bits directamente, como también, las operaciones de multi-precisión son posibles.

Al igual que muchos lenguajes de *asembler*, el set de instrucción es del ADSP-218x usa notaciones algebraicas para las operaciones aritméticas y el movimiento de datos.

En adición a JUMP y CALL, las instrucciones de control del set de instrucciones apoyan la ejecución condicional de la mayoría de los cálculos. Retornar de una interrupción (RTI) y de una subrutina (RTS) también es posible.

La instrucción IDLE es provista para el modo en espera de baja potencia del procesador hasta que una interrupción ocurre. IDLE pone el procesador dentro de un estado de baja potencia mientras espera por una interrupción.

6.2.1 Instrucciones computacionales. Este grupo de comandos ejecuta todas las instrucciones ALU, MAC y Shifter. Hay dos clases funcionales: instrucciones estándar, las cuales incluyen la mayoría de las operaciones computacionales y pueden ser ejecutadas condicionalmente (condición IF) y las instrucciones especiales las cuales forman un pequeño conjunto y deben ser ejecutadas individualmente. Las condiciones permisibles son listados en la tabla 19.

Cada unidad computacional tiene un set de registros de entrada y de salida. La lista de operandos de entrada y registros de resultados se muestra en la tabla 20.

Tabla 19. Condiciones permisibles para instrucciones computacionales

Condición	Keyword
Si el resultado ALU es:	
Igual a cero	EQ
No igual a cero	NE
Mayor que cero	GT
Mayor que o igual a cero	GE
Menor que cero	LT
Menor que o igual a cero	LE
Estado del Acarreo ALU	
Acarreo	AC
No Acarreo	NOT AC
Signo entrada x	
Positivo	POS
Negativo	NEG
Estado del overflow ALU	
Overflow	AV
No overflow	NOT AV
Estado del contador	
Expirado	CE
No expirado	NOT CE

Tabla 20. Registros computacionales de entrada y salida

ALU		
Fuente para entrada X (xop)	Fuente para entrada Y (yop)	Destino para puerto salida R
AX0, AX1, AR, MR0, MR1, MR2, SR0, SR1	AY0, AY1 AF	AR AF

Tabla 21 (Continuación). Registros computacionales de entrada y salida

MAC		
Fuente para entrada X (xop)	Fuente para salida Y (yop)	Destino para puerto salida R
MX0, MX1, AR MR0,MR1,MR2 SR0,SR1	MY0, MY1 MF	MR (MR2, MR1, MR0) MF

SHIFTER	
Fuente para entrada Shifter (xop)	Destino para salida Shifter
SI, SR0, SR1 AR, MR0, MR1, MR2	SR (SR1, SR0)

6.2.1.1 Grupo ALU

- **Funciones Estándar**. Las instrucciones ALU incluyen suma, resta y operaciones lógicas (AND, OR, NOT, exclusivo-OR), pasar, negar, incrementar, decrementar, limpiar y valor absoluto. La función "-" hace una resta de complemento a dos, mientras que la función NOT obtiene un complemento a uno. Las operaciones ALU también soportan constantes permitidas sumadas a yops. Estas constantes están disponibles para todas las operaciones ALU no multifuncionales usando tanto xop como yop con excepción a divisiones primitivas.

Un nuevo set de instrucciones de bit ALU permite operaciones test, set, clear, y *toggle* a través de una elección cuidadosa de la constante y la función ALU. La función PASS pasa el operando listado o constante permitida pero evalúa y guarda la información de estado para después evaluar signo y cero. Como un ejemplo, considere una instrucción de suma ALU para sumar/sumar con acarreo en la forma:

$$\begin{vmatrix} AR \\ AF \end{vmatrix} = xop \begin{vmatrix} + yop \\ + C \\ + yop + C \\ + value \end{vmatrix}$$

Las instrucciones para las operaciones de sustracción y operaciones lógicas son de una forma similar. Si las opciones AR y "+yop+C" son escogidas, y si xop y yop están contenidas en AX0 y AY0 respectivamente, la instrucción no condicionada se leería

$$AR0 = AX0 + AY0 + C$$

Esta expresión algebraica significa que el registro de resultado ALU AR obtiene el valor de la de los registros ALU de entrada x y entrada y mas el valor del bit de acarreo.

Cuando una condición opcional IF es incluida, y si el estado del bit de acarreo es escogido entonces la instrucción condicional se leería:

$$\text{if AC } AR = AX0 + AY0 + C$$

La expresión condicional, IF AC, evalúa el bit de acarreo ALU; si hay un acarreo de una instrucción previa, esta instrucción se ejecuta, de lo contrario un NOP ocurre y continúa la ejecución con la siguiente instrucción.

Finalmente, un set de operaciones ALU de resultado libre soportan la generación de códigos de condición basados en una operario ALU pero prescindiendo del resultado. Esto permite la evaluación de los valores de registros sin intervenir los valores de los registros AR o AF.

La tabla 21 muestra una lista completa de todas las instrucciones ALU.

6.2.1.2 Grupo MAC.

- **Funciones Estándar.** Las instrucciones MAC incluyen multiplicación. Multiplicación acumulada, multiplicación-sustracción, transferencia condicionada AR y limpiar. En adición, la ADSP 2181 tiene una modificación MAC la cual implementa instrucciones "tipo 9" las cuales permiten que el operando X, xop, sea usado como el operando Y, yop. Esto resulta en un ciclo sencillo x2 y también en operaciones $\sum x^2$. Como un ejemplo, considere una instrucción MAC para multiplicación acumulación en la forma:

$$\left[IF\ condicion\right]\ \left|\begin{matrix} MR \\ MF \end{matrix}\right| = MR + xop * \left|\begin{matrix} yop \\ xop \end{matrix}\right|\ \left|\begin{matrix} (SS) \\ (SU) \\ (US) \\ (UU) \\ (RND) \end{matrix}\right|$$

Si las opciones "MR" y "UU" son escogidas, si xop y yop están contenidos en MX0 y MY0 respectivamente, y si la condición overflow es escogida, entonces la instrucción condicional se leería:

$$IF\ NOT\ MV\ MR = MR + MX0 * MY0 \quad (UU)$$

Tabla 21. Instrucciones ALU

(IF Condición)	$\begin{vmatrix} AR \\ AF \end{vmatrix} = xop \begin{vmatrix} + \ yop \\ + C \\ + \ yop + C \\ + \ cons\tan te \end{vmatrix}$
(IF Condición)	$\begin{vmatrix} AR \\ AF \end{vmatrix} = xop \begin{vmatrix} - \ yop \\ - \ yop + C - 1 \\ - \ cons\tan te \end{vmatrix}$
(IF Condición)	$\begin{vmatrix} AR \\ AF \end{vmatrix} = yop \begin{vmatrix} - \ xop \\ - \ xop + C - 1 \end{vmatrix}$
(IF Condición)	$\begin{vmatrix} AR \\ AF \end{vmatrix} = xop \begin{vmatrix} AND \\ OR \\ IOR \end{vmatrix} \ yop$
(IF Condición)	$\begin{vmatrix} AR \\ AF \end{vmatrix} = PASS \begin{vmatrix} xop \\ yop \\ -1,0,1 \\ cons\tan te \end{vmatrix}$
(IF Condición)	$\begin{vmatrix} AR \\ AF \end{vmatrix} = - \begin{vmatrix} xop \\ yop \end{vmatrix}$
(IF Condición)	$\begin{vmatrix} AR \\ AF \end{vmatrix} = NOT \begin{vmatrix} xop \\ yop \end{vmatrix}$
(IF Condición)	$\begin{vmatrix} AR \\ AF \end{vmatrix} = ABS \begin{vmatrix} xop \end{vmatrix}$
(IF Condición)	$\begin{vmatrix} AR \\ AF \end{vmatrix} = yop \begin{vmatrix} +1 \\ -1 \end{vmatrix}$
(IF Condición)	$\begin{vmatrix} AR \\ AF \end{vmatrix} = \begin{vmatrix} TSTBIT \ \ n \ \ of \ \ xop \\ SBTBIT \ \ n \ \ of \ \ xop \\ CLBIT \ \ n \ \ of \ \ xop \\ TGTBIT \ \ n \ \ of \ \ xop \end{vmatrix}$
$NONE =< ALU >$	
$DIVS \quad yop, \quad xop$	
$DIVQ \quad xop$	

La expresión condicional, IF NOT MV, evalúa el bit de overflow MAC. Si la condición no es real un NOP es ejecutado. La expresión $MR = MR + MX0 * MY0$ es la operación de multiplicación acumulada: el registro de resultado de la multiplicación MR obtiene el valor de si mismo mas el producto de los registros de entrada X y Y seleccionados. El modificador en paréntesis (UU) trata al operando como sin signo. Solo puede existir un modificador seleccionado del set disponible. (SS) significa que ambos son signados, mientras que (US) y (SU) significa que solo el primer o el segundo operando tienen signo; (RND) significa redondear el resultado (implícitamente signado).

Un nuevo modo ha sido adherido para permitir el redondeamiento parcial en adición al redondeamiento normal no parcial. Esto es logrado por el establecimiento del bit BIASRND a 0 para un redondeo normal y a 1 para el redondeo parcial. Este modo es sumado para una implementación más eficiente de algoritmos de bit especificados los cuales especifican un redondeo parcial. El redondeo normal no parcial es el preferido por muchos algoritmos.

- **Funciones Especiales.** La saturación acumulada es la única función especial MAC

$$IF\ \ MV\ \ SAT\ \ MR$$

La instrucción evalúa el bit de overflow de MAC (MV) y satura el registro MR (solo para un ciclo) si el bit es establecido. La tabla 22 muestra una lista de todas las instrucciones MAC

6.2.1.3 Grupo Shifter

- Funciones Estándar: Las funciones estándar Shifter incluyen cambios aritméticos y lógicos, también como operaciones de escalamiento de punto flotante y bloques de punto flotante; exponente, normalización, des normalización y bloque de exponente ajustado. Como un ejemplo, considere una instrucción shifter para normalizar:

$$IF\ \ LT\ \ SR = SR\ OR\ \ NORM\ \ SI\ \ (HI)$$

La expresión condicional, IF LT, evalúa la condición “menor que cero”. Si la condición es falsa, un NOP es ejecutado. El destino de todas las operaciones Shifting es el registro de resultado del Shifter, SR. En este ejemplo, SI, el registro de entrada Shifer, es el operando. La cantidad y dirección del shift es controlada por el valor signado en el registro SE en todas las operación shift excepto un shift inmediato. Los valores positivos causan desplazamientos hacia la izquierda; los valores negativos causan desplazamientos hacia la derecha.

Tabla 22. Instrucciones MAC

(IF Condición)	$\begin{vmatrix} MR \\ MF \end{vmatrix}$	$= xop *$	$\begin{vmatrix} yop \\ xop \end{vmatrix}$	$\begin{vmatrix} (SS) \\ (SU) \\ (US) \\ (UU) \\ (RND) \end{vmatrix}$
(IF Condición)	$\begin{vmatrix} MR \\ MF \end{vmatrix}$	$= MR + xop *$	$\begin{vmatrix} yop \\ xop \end{vmatrix}$	$\begin{vmatrix} (SS) \\ (SU) \\ (US) \\ (UU) \\ (RND) \end{vmatrix}$
(IF Condición)	$\begin{vmatrix} MR \\ MF \end{vmatrix}$	$= MR - xop *$	$\begin{vmatrix} yop \\ xop \end{vmatrix}$	$\begin{vmatrix} (SS) \\ (SU) \\ (US) \\ (UU) \\ (RND) \end{vmatrix}$
(IF Condición)	$\begin{vmatrix} MR \\ MF \end{vmatrix}$	$= 0$		
(IF Condición)	$\begin{vmatrix} MR \\ MF \end{vmatrix}$	$= MR\ [(RND)]$		
$IF\ MV\ SAT\ MR$				

El modificador “SR OR” (el cual es opcional) es una OR lógica entre el registro de entrada con el contenido actual del registro SR; esto permite al usuario construir un valor de 32 bits en SR desde dos piezas de 16 bits. “NORM” es el operando y “(HI)” es el modificador el que determina si el desplazamiento es relativa a la mitad alta o baja (16 bits) del registro SR. Si “SR OR” es omitido, el resultado es pasado directamente dentro de SR.

- **Funciones Especiales.** *Shift* inmediato es la única función especial del Shifter. El número de lugares (exponente) de desplazamiento es especificado en la palabra instrucción.

La tabla 23 provee una lista de todas las instrucciones Shifter.

Tabla 23. Instrucciones SHIFTER

(IF Condición)	$SR = [SR\ OR]\ ASHIFT\ xop \left(\left\| \begin{matrix} HI \\ LO \end{matrix} \right\| \right)$
(IF Condición)	$SR = [SR\ OR]\ LSHIFT\ xop \left(\left\| \begin{matrix} HI \\ LO \end{matrix} \right\| \right)$
(IF Condición)	$SR = [SR\ OR]\ NORM\ xop \left(\left\| \begin{matrix} HI \\ LO \end{matrix} \right\| \right)$
(IF Condición)	$SR = EXP\ xop \left(\left\| \begin{matrix} HI \\ LO \\ HIX \end{matrix} \right\| \right)$
(IF Condición)	$SE = EXPDJ\ xop$
$SR = [SR\ OR]\ ASHIFT\ xop\ by\ \langle data \rangle \left(\left\| \begin{matrix} HI \\ LO \end{matrix} \right\| \right)$	
$SR = [SR\ OR]\ LSHIFT\ xop\ by\ \langle data \rangle \left(\left\| \begin{matrix} HI \\ LO \end{matrix} \right\| \right)$	

6.2.2 Instrucciones de movimientos de datos. Estas instrucciones mueven datos a y desde registros de datos de la memoria interna y externa. Los registros de la ADSP-2181 están divididos en 2 grupos, refiriéndose a ellos como reg los cuales incluyen casi todos los registros y como dreg o registros de datos, los cuales son un subconjunto. Solo el contador de programa (PC) y los registros de realimentación de la ALU y MAC (AF y MF) no son accesibles. La tabla 24 muestra cuales registros pertenecen a esos grupos. Muchos de los registros del sistema de control de la ADSP-2181 están mapeados en memoria y pueden ser leídos o escritos como locaciones de memoria en vez de utilizar nombres de registros.

Hay seis clases de instrucciones de movimientos de datos. Exceptuando por instrucciones inmediatas y nuevo espacio I/O, las direcciones de datos son calculadas por los DAGs por medio de los contenidos de sus registros *index* (I) y *modify* (M). Las clases de movimientos de datos son:

- Cargar registros inmediatos
- Movimiento registro a registro
- Movimiento inmediato de dirección DM
- Movimiento indirecto de dirección DM o PM
- Movimiento de espacio I/O
- Lectura de Multifunción DM y PM

Tabla 24. Conjunto de registros reg y dreg de la ADSP-2181

Registros Accesibles (reg)	Registros de datos (dreg)
SB PX I0-I7, M0-M7, L0'L7 CNTR ASTAT, MSTAT, SSTAT IMASK, INCTL, IFC TX0,TX1,RX0,RX1	AX0,AX1,AY0,AY1,AR MX0,MX1,MY0,MY1,MR0,MR1,MR2 SI, SE, SR0, SR1

En la descripción de cada uno de estas clases mostrada abajo, "valor inmediato" se refiere a un numero de 16 bits contenido en el campo de instrucción mientras que "dirección inmediata" se refiere a una dirección de 14 bits contenida en el campo de instrucción. La tabla 25 provee una lista de todas las instrucciones Move

6.2.2.1 Cargar Registro Inmediato. En esta instrucción, el dato es proveído por un valor inmediato de 16 bits en la palabra instrucción y es movido a reg.

$$reg = \langle data \rangle; \ \langle data \rangle = valor\ inmediato$$

6.2.2.2 Mover registro a registro. Aquí el calor de un permisible reg es movido dentro de otro permisible reg

$$reg = reg$$

6.2.2.3 Movimiento Inmediato de una Dirección de Memoria de Datos. Aquí una memoria de datos direccionada por una dirección inmediata de 14 bits en la palabra instrucción es movida dentro de un reg

Lectura de memoria de datos

$$reg = DM(\langle addr \rangle); \quad \langle addr \rangle = direccion\ inmediata$$

Escritura de memoria de datos

$$DM(\langle addr \rangle) = reg;$$

Tabla 25. Instrucción MOVE

$reg = reg$
$reg = DM(\langle address \rangle);$
$dreg = DM\left(\begin{vmatrix} I0 \\ I1 \\ I2 \\ I3 \\ - \\ I4 \\ I5 \\ I6 \\ I7 \end{vmatrix}, \begin{vmatrix} M0 \\ M1 \\ M2 \\ M3 \\ - \\ M4 \\ M5 \\ M6 \\ M7 \end{vmatrix}\right)$
$DM\left(\begin{vmatrix} I0 \\ I1 \\ I2 \\ I3 \\ - \\ I4 \\ I5 \\ I6 \\ I7 \end{vmatrix}, \begin{vmatrix} M0 \\ M1 \\ M2 \\ M3 \\ - \\ M4 \\ M5 \\ M6 \\ M7 \end{vmatrix}\right) = \begin{vmatrix} dreg \\ \langle data \rangle \end{vmatrix}$
$DM(\langle address \rangle) = reg;$
$reg = \langle data \rangle$
$dreg = PM\left(\begin{vmatrix} I4 \\ I5 \\ I6 \\ I7 \end{vmatrix}, \begin{vmatrix} M4 \\ M5 \\ M6 \\ M7 \end{vmatrix}\right);$
$PM\left(\begin{vmatrix} I4 \\ I5 \\ I6 \\ I7 \end{vmatrix}, \begin{vmatrix} M4 \\ M5 \\ M6 \\ M7 \end{vmatrix}\right) = dreg;$
$IO(\langle address \rangle) = dreg;$
$dreg = IO(\langle address \rangle)$

6.2.2.4 Movimiento Indirecto de una Dirección de Memoria. Esta instrucción hace que el dato sea movido entre dreg y DM o PM. La dirección de memoria para las operaciones actuales es dada por uno de los cuatro registros I (Im) de la DAG. El valor de I es entonces guardado nuevamente siendo modificado después por el contenido de uno de los cuatro registros M (Mn) y la longitud del buffer se guarda en uno de los cuatro registros L (Lm). La operación es:

Lectura/ Escritura de una dirección especificada por un puntero

$$direccion\ de\ memoria = \mathrm{Im}$$

Entonces modificando el puntero

$$\mathrm{Im} = (\mathrm{Im} + Mn) \bmod Lm$$

El direccionamiento indirecto puede acceder tanto a la memoria de datos como a la memoria de programa. El set de registros en DAG1 puede solamente ser usado para memoria de datos mientras que el set de registros en DAG2 puede ser usado para memoria de datos y para memoria de programa. Los registros de longitud de buffer Lm son emparejados con los registros index, por ejemplo, L5 es el registro modular para la locación de memoria indexada en I5.

Lectura de memoria dentro de los registros de datos

$$dreg = DM(\mathrm{Im}, Mn);$$
$$dreg = PM(\mathrm{Im}, Mn);$$

Escritura DM: Inmediata o desde un registro de datos

$$DM(\mathrm{Im}, Mn) = \left| \begin{matrix} dreg \\ \langle data \rangle \end{matrix} \right|$$

Escritura PM desde un registro de datos

$$PM(\mathrm{Im}, Mn) = dreg$$

6.2.2.5 Movimiento Espacio I/O

Esta es una nueva instrucción disponible en la ADSP 2181. Esta es usada para acceder a locaciones del espacio I/O.

$$\begin{matrix} IO(\langle addr \rangle) = dreg \\ dreg = IO(\langle data \rangle); \end{matrix} \qquad \langle addr \rangle = valor\ entre\ 0\ y\ 2048$$

6.2.2.6 Lectura de Multifunción para Memoria de Datos y de Programa
Aquí una instrucción combinada *Move* lee datos dentro de un par de registros ALU y MAC desde la DM y PM en el mismo ciclo.

6.2.3 Instrucciones Multifunción. Las operaciones de multifunción toman ventaja del paralelismo inherente de la arquitectura del ADSP-218x proporcionando combinaciones de movimientos de datos, lectura y escritura de memoria y computación, todas en un solo ciclo.

6.2.3.1 ALU/MAC con lectura de datos y programas de la memoria. Tal vez la operación simple más común en el algoritmo DSP es la suma de productos. El procesador ADSP- 218x puede ejecutar tanto la obtención de datos como la multiplicación/acumulación en un solo ciclo. Típicamente, un bucle de acumular/multiplicar puede ser expresado mediante un código fuente ADSP-218x en solo dos líneas de programa como se muestra a continuación:

*MR=MR+MX0*MYO(SS), MXO=DM(IO,MO), MYO=PM(I4,M5);*

Las partes computacionales de esta instrucción multifunción puede ser cualquier instrucción incondicional ALU excepto por la división o cualquier instrucción MAC excepto por la saturación. El resultado de la computación debe ir al registro resultado (MR o AR) no al registro de realimentación (MF o AF)

6.2.3.2 Lectura de Datos y programas de la memoria. Esta variación de una instrucción de multifunción es un caso especial de una instrucción multifunción. Esta se ejecuta solo al obtener un operando dual como se muestra a continuación:

AX0=DM(I2,M0), AY0=PM(I4,M6)

En este ejemplo se ha usado un registro de entrada ALU como destino.

6.2.3.3 Cálculos con lectura de la memoria. Si una simple lectura de memoria es realizada en lugar de una doble lectura de memoria de las dos instrucciones de multifunción previas, una gama más amplia de cálculos puede ser ejecutada. Los cálculos posibles incluyen todas las operaciones ALU, excepto división, todas las operaciones MAC y todas las operaciones Shifters excepto SHIFT IMMEDIATE. El cálculo debe ser incondicional. Un ejemplo de un tipo de esta instrucción multifunción es:

AR=AX0+AY0, DM(I0,M0)=AX0;

Aquí una adición es realizada en la ALU mientras una operación sencilla es obtenida desde una memoria de datos.

6.2.3.4 Cálculos con escritura de la memoria. Los cálculos con la instrucción de escritura de memoria son similares en estructura a los cálculos con la lectura de memoria: el orden de las cláusulas en la línea de instrucción, sin embargo, es revertido. Primero la escritura de memoria es realizada y luego el cálculo.

DM(I0,M0)=AR, AR=AX0+AY0

Invirtiendo el orden de las cláusulas implicaría que el resultado de los cálculos es escrito en la memoria cuando, en efecto, el valor previo del registro es el q se está escribiendo.

6.2.3.5 Cálculos con movimiento de registro de datos. Esta instrucción multifunción realiza un movimiento de un registro de dato a un registro de dato en paralelo con los cálculos.

AR=AX0+AY0, AX0=MR2

Aquí, una operación de adición ALU ocurre mientras un nuevo valor es cargado dentro de AX0 y MR2. El movimiento puede ser desde o para todas los registros de entrada y salida de ALU, MAC y Shifter excepto el registro de realimentación (AF y MF) y SB

En el ejemplo el movimiento de registro de datos carga el registro AX0 con el nuevo valor al final del ciclo. Todas las operaciones ALU excepto la división, todas las operaciones MAC y todas las operaciones Shifter excepto *SHIFT IMMEDIATE* son posibles.

En la tabla 26 se resumen las instrucciones multifunción, de una misma forma, en la tabla 27 se muestran las combinaciones posibles de las instrucciones multifunción. Se pueden combinar operaciones sobre la misma fila con cada una de las demás.

Tabla 26. Instrucciones Multifunción

$\begin{vmatrix} < ALU > * \perp \\ \\ < MAC > * \perp \end{vmatrix} . \begin{vmatrix} AX0 \\ \\ AX1 \\ MX0 \\ MX1 \end{vmatrix} = DM(\begin{vmatrix} I0 \\ \\ I1 \\ I2 \\ I3 \end{vmatrix} . \begin{vmatrix} M0 \\ \\ M1 \\ M2 \\ M3 \end{vmatrix}). \begin{vmatrix} AY0 \\ \\ AY1 \\ MY2 \\ MY3 \end{vmatrix} = PM(\begin{vmatrix} I4 \\ \\ I5 \\ I6 \\ I7 \end{vmatrix} . \begin{vmatrix} M4 \\ \\ M5 \\ M6 \\ M7 \end{vmatrix}).$
$\begin{vmatrix} AX0 \\ AX1 \\ MX0 \\ MX1 \end{vmatrix} = DM(\begin{vmatrix} I0 \\ I1 \\ I2 \\ I3 \end{vmatrix} . \begin{vmatrix} M0 \\ M1 \\ M2 \\ M3 \end{vmatrix}). \begin{vmatrix} AY0 \\ AY1 \\ AY2 \\ AY3 \end{vmatrix} = PM(\begin{vmatrix} I4 \\ I5 \\ I6 \\ I7 \end{vmatrix} . \begin{vmatrix} M4 \\ M5 \\ M6 \\ M7 \end{vmatrix});$
$\begin{vmatrix} < ALU > * \\ < MAC > * \\ < SHIFT > * \end{vmatrix} . dreg = DM(\begin{vmatrix} I0 \\ I1 \\ I2 \\ I3 \\ \\ I4 \\ I5 \\ I6 \\ I7 \end{vmatrix} . \begin{vmatrix} M0 \\ M1 \\ M2 \\ M3 \\ \\ M4 \\ M5 \\ M6 \\ M7 \end{vmatrix})$
$DM(\begin{vmatrix} I0 \\ I1 \\ I2 \\ I3 \\ \\ I4 \\ I5 \\ I6 \\ I7 \end{vmatrix} . \begin{vmatrix} M0 \\ M1 \\ M2 \\ M3 \\ \\ M4 \\ M5 \\ M6 \\ M7 \end{vmatrix}) = dreg. \begin{vmatrix} < ALU > * \\ < MAC > * \\ < SHIFT > * \end{vmatrix}$
$\begin{vmatrix} < ALU > * \\ < MAC > * \\ < SHIFT > * \end{vmatrix} .dreg = dreg$

Donde:

<ALU> Cualquier instrucción ALU (excepto DIVS, DIVQ)
<MAC> Cualquier instrucción multiplicación/acumulación
<SHIFT>| Cualquier instrucción Shifter (excepto Shift Inmediate)

Tabla 27. Combinaciones Instrucciones Multifunción

Cálculos Incondicionados	Movimiento Datos (DM=DAG1)	Movimiento Datos (PM=DAG2)
Ningún o cualquier ALU (excepto División) o MAC	Lectura DM	Escritura PM
Cualquier MAC	Lectura DM	-- Lectura PM
Cualquier ALU excepto División	--	--
Cualquier Shift excepto Inmedite	Escritura DM -- Registro a Registro	Escritura PM

6.2.4 Instrucciones de flujo de control de flujo. El flujo de control del programa en la ADSP-2181 es simple y muy poderoso. Este direcciona la secuencia del programa. En un orden normal, la secuenciador obtiene automáticamente la próxima operación continua para ser ejecutada. Este flujo puede ser alterado por estas instrucciones. El control de flujo de programa provee:

- Saltos a rutinas de servicio de interrupción y llamados a subrutinas
- Retorno de una interrupción o subrutina
- Condiciones de pin FLAG
- Lazos DO
- Instrucción IDLE

Una opcional condición “IF” puede primero evaluar alguno de los estados de las condiciones definidas anteriormente.

6.2.4.1 Instrucciones JUMP y CALL. JUMP es una construcción familiar de muchos otros procesos. Como un ejemplo considere el siguiente segmento:

$$IF\ EQ\ JUMP\ my\ \_\ label,$$

My_label es un identificador usado como una nueva dirección donde el control del programa es transferido para su ejecución. En vez de una etiqueta, un registro index en DAG2 puede ser explícitamente usado. Por otro lado, una instrucción CALL hace un llamado a una subrutina, y coloca el presente PC dentro de su propia pilas como la dirección retorno. El alcance por defecto de cualquier etiqueta es el modulo en el cual está declarada. La directiva assembler ENTRY hace una etiqueta visible como un punto de entrada para subrutinas fuera del modulo. Por el contrario, la directiva EXTERNAL hace posible usar una etiqueta declarada en otro modulo.

6.2.4.2 Instrucción RETURN. Existen dos estados de retorno: RTS de subrutina y RTI de una rutina de servicio de interrupción. En ambos casos, un RETURN muestra la dirección de retorno de la pila de PC. Un retorno de una interrupción también muestra el nuevo estado de la pila, retornando la aritmética, el modo de estado y los registros *mask* de interrupción a los valores que tenían antes de la interrupción

6.2.4.3 Instrucción FLAG. JUMP y CALL permiten la condicional adicional “FLAG IN” y “NOT FLAG_IN” que para ser utilizadas para derivaciones sobre el estado del pin FI, pero solo con direccionamiento directo, no con DAG2 como la fuente de dirección. Adicionalmente, el pin FO (*Flag Out*) puede ser establecido, limpiado o toggleado. Además esta instrucción no altera el flujo del programa, esta provee una estructura de control la comunicación del multiprocesador y es además incluida en este grupo.

6.2.4.4 Instrucción de Lazo. La instrucción DO UNTIL realiza la operación en lazo sin perdidas. Esta tiene la siguiente forma:

$$DO\ \langle addr \rangle\ (UNTIL\ condition),$$

La etiqueta “addr” designa la última instrucción en el lazo. La “condición” determina la terminación del lazo. Cuando el DO es empezado, addr y la condición de la terminación, son colocados dentro de la pila del lazo secuenciador y el actual PC+1 es colocado dentro de la pila del PC para convertirse en la siguiente dirección después de la terminación del lazo.

6.2.4.5 Instrucción IDLE. Esta instrucción provee una forma para parar por interrupción. IDLE logra que el procesador espera en un estado de baja alimentación hasta que una interrupción ocurra. Este usa una alimentación menor que un lazo creado con JUMP. Esta instrucción en la ADSP-2181tambien soporta una característica de baja potencia (*slow idle*). El *Slow IDLE* permite frenar el reloj por un factor de 16, 32, 64 o 128 durante el IDLE. Este tiene la siguiente forma:

$IDLE,$	Idle normal
$IDLE(in),$	Idle lento: n= 16, 32, 64, 128

La tabla 28 provee una lista completa de todas las instrucciones del flujo de control del programa.

Tabla 28. Instrucciones del flujo de control del programa

(IF Condición)	$JUMP$	$(I4)$ $(I5)$ $(I6)$ $(I7)$ $\langle address)$
(IF Condición)	$CALL$	$(I4)$ $(I5)$ $(I6)$ $(I7)$ $\langle address)$
IF	$FLAG\ IN$ $NOT\ FLAG\ IN$	$CALL\langle address)$
IF	$FLAG\ IN$ $NOT\ FLAG\ IN$	$JUMP\langle address)$
(IF Condición)	RTS	
(IF Condición)	RTI	
$DO\langle address\rangle$	$(UNTIL\ ter\min antion)$	
$IDLE,$ $IDLE(n)$	N? 13, 32, 64, 128	

6.2.5 Instrucciones Misceláneas. Existen muchas instrucciones misceláneas. NOP, de por sí, no es una instrucción de operación. La instrucción PUSH/POP permite un control explicito del estado, contador, PC y pilas de bucle.

Las instrucciones de control de modo (*enable/disable*) encienden o apagan muchos modos de operación. La instrucción gobierna modos comunes para la ADSP-2100 y el modo de control extendido de la ADSP-2181.

Un simple ENA o DIS puede ser seguido por numerosos identificadores de modo, separado por comas; ENA o DIS puede ser también repetidos. Todos los siete modos pueden habilitados, deshabilitados o cambiados en una simple instrucción.

La instrucción MODIFY modifica el puntero de dirección en el registro I seleccionado con el valor en el registro M seleccionado, sin cambiar ningún acceso de memoria actual. Como siempre, los registros I y M deben ser de la misma DAG, algún de I0-I3pueden ser usados solo con uno de M0-M3 y lo mismo para I4-I7 y M4-M7. Si un buffer circular esta en uso, se aplican módulos lógicos.

La ADSP-2181 soporta una instrucción para habilitar interrupción y una para deshabilitar interrupción. Las interrupciones son habilitadas por defecto cuando se resetea. La tabla 29 provee una lista completa de las instrucciones misceláneas.

6.2.6 Estructura de datos. El software de desarrollo de la ADSP-2181 soporta la declaración y uso de un simple set de estructura de datos: arreglos y puertos de una dimensión. Los arreglos pueden ser valores simples o valores múltiples. En adición, los arreglos pueden ser usados como un buffer circular. A continuación se da una breve discusión de cada elemento con un ejemplo de cómo son declarados y usados.

6.2.6.1 Arreglos. Los arreglos son las estructuras básicas en el set de instrucciones ADSP-2181. En la literatura, la palabra arreglo (array) y la expresión buffer de datos (data buffer) son usados intercambiablemente. Los arreglos son declarados con las directivas Assembler y pueden ser referenciados indirectamente por un nombre, pueden ser inicializados desde valores inmediatos en una directiva o desde archivos de datos externos y pueden ser lineales o circulares con una extensión automática.

Un arreglo es declarado con una directiva como se muestra a continuación

$$..VAR / DM\ coeficientes[129];$$

Esto declara un arreglo de 128 valores de 16-bits localizado en la memoria de datos (DM). Los operadores especiales ^ y % referencian la dirección y longitud, respectivamente, del arreglo. Esto podría ser referenciado como se muestra a continuación:

$$IO = \text{^}\,coeficientes \qquad \{punto\,de\,direccion\,de\,buffer\}$$
$$Mx0 = DM(I0, M0) \qquad \{c\arg a\,Mx0\,desde\,el\,buffer\}$$

Esta instrucción carga el valor dentro de Mx0 desde el inicio del buffer de coeficientes en la memoria de datos. Con la automática post-modificación de las DAGs, el usuario podría ejecutar la segunda de estas instrucciones en un lazo continuamente avanzando a través del buffer.

Tabla 29. Instrucciones Misceláneas

(IF Condición)	$\left\|\begin{matrix} SET \\ RESET \\ TOGGLE \end{matrix}\right\|$	$FLAG\_OUT$
NOP		
$\left\|\begin{matrix} PUSH \\ POP \end{matrix}\right\|$	$STS\,[, POP\,CNTR][, POP\,PC][, POP`LOOP]$	
$\left\|\begin{matrix} ENA \\ DIS \end{matrix}\right\|$	$\left\|\begin{matrix} BTI\,RRV \\ AV\_LATCH \\ AR\_SAT \\ SEC\_REG \\ TIMER \\ G\_MODE \\ M\_MODE \\ INTS \end{matrix}\right\|$	$[,\ldots 1$
$MODIFY$	$($ $\left\|\begin{matrix} I0 \\ I1 \\ I2 \\ I3 \\ \_ \\ I4 \\ I5 \\ I6 \\ I7 \end{matrix}\right\|$, $\left\|\begin{matrix} M0 \\ M1 \\ M2 \\ M3 \\ \_ \\ M4 \\ M5 \\ M6 \\ M7 \end{matrix}\right\|$ $)$	

Alternativamente, cuando solo la primera locación necesita direccionarse, se puede usar directamente el nombre del buffer como una etiqueta en muchas circunstancias así:

$$MX0 = DM(coeficientes)$$

El linker sustituye la dirección actual para la etiqueta. Esto también es posible para inicializar un completo arreglo/buffer desde un archivo de datos, usando la directiva INIT.

$$.INIT\ coeficientes : \langle filename.data \rangle$$

Esto lee el valor desde el archivo file*name*.dat dentro de un arreglo en un link de tiempo. Esta característica es soportada solo en el simulador ADSP-210X aun si no puede ser cargado directamente dentro del chip de memoria de datos por la secuencia de hardware.

Un arreglo o buffer de datos con una longitud de uno se comporta como una simple variable de palabra sencilla y es declarada de esta forma

$$.VAR / DM\ coeficientes$$

6.2.6.2 Arreglos/Buffer Circulares. Un requerimiento común en DSO son los buffer circulares. Este es directamente implementado por los DAGS de la ADSP-2181, usando los registros L (longitud). Primero, el buffer debe ser declarado como circular:

$$.VAR / DM / CIRC\ coeficientes[128]$$

Seguido, el registro L debe ser inicializado, típicamente usando el operador % (o una constante) y, como también el registro I y el registro M

$L0 = \%coeficientes$	$\{longitud\ del\ buffer\ circular\}$
$I0 = \hat{}\ coeficientes$	$\{punto\ de\ direccion\ del\ buffer\}$
$M0 = 1$	$\{Incremento\ de\ una\ locacion\ cada\ vez\}$

Ahora la instrucción quedara:

$MX0 = DM(I0, M0)$	$\{c\arg a\ MX0\ desde\ el\ buffer\}$

Los registros L deben ser inicializados por buffers de cualquier longitud que no sean circulares.

6.2.6.3 Direccionamiento Indirecto Linear (No Circular). El procesador ADSP-2181 permite dos modos de direccionamiento para archivos de datos de memoria, directo e indirecto. El direccionamiento indirecto es realizado cargando una dirección dentro de un registro index (I) y especificando uno de los registros M disponibles.

El registro L esta dado para facilitar el direccionamiento de los buffers de datos circulares. Un buffer circular es solo implementado cuando un registro L es establecido a un valor que no es cero. Para direccionamiento linear o no circular, el registro L correspondiente al registro I usado debe ser establecido a cero. No se debe asumir que los registros L son inicializados automáticamente o deben ser ignorados; los registros I, L y M contienen valores randomicos. El programa debe inicializar los registros L correspondientes a los registros I en uso. A continuación se da un ejemplo sencillo de un direccionamiento indirecto linear.

$$I3 = 0x3800;$$
$$M2 = 0;$$
$$L3 = 0;$$
$$AX0 = DM(I3, M2)$$

7 LABORATORIOS

7.1 LABORATORIO 1. UNIDAD COMPUTACIONAL ALU

Objetivos

- Familiarizar al estudiante con el proceso básico a seguir en la apertura, depuración y simulación de un proyecto para el sistema de desarrollo ADSP-2181 DSPKit, utilizando el software de programación VisualDSP 3.5++.

- Comprobar la funcionalidad del set estándar de funciones aritméticas y lógicas de la unidad computacional ALU.

- Hacer uso de las ventanas de registros del software de programación VisualDSP 3.5++.

Lecturas recomendadas

- Profundizar en el manejo de la interfaz del software de programación VisualDSP 3.5++.
- Consultar el set de instrucciones de la unidad computacional ALU

Marco Teórico.

La unidad aritmética y lógica (ALU) proporciona el set estándar de funciones aritméticas y lógicas. Los operadores de entrada son X y Y, y el operador de salida es R, todos estos de 16 bits.

Los registros de entrada X (xop) de la ALU aceptan datos de dos fuentes: los registros AX y del bus de resultados (R). El bus R conecta los registros de salida de todas las unidades computacionales, permitiendo ser usados como operadores de entrada directamente. El registro AX está conformado por dos registros de 16 bits cada uno: AX0 y AX1. Estos registros pueden ser leídos y escritos desde el bus DMD.

Los registros de entrada Y (yop) de la ALU aceptan también datos de dos fuentes: el registro AY y el registro de realimentación de la ALU (AF). El registro AY está compuesto por dos registros de 16 bits cada uno: AY0 y AY1.

La salida de la ALU puede ser cargada en el registro de alimentación (AF) o en el registro de resultados (AR). El AF es un registro interno de la ALU y puede ser usado directamente como un operador de entrada Y. El registro AR puede ser manejado por el bus DMD o R y solo puede ser cargado directamente por el bus DMD.

En las siguientes tablas se muestra un resumen de los operandos de entrada permitidos para cada registro de entrada y los registros de resultado y un resumen de todas las instrucciones de la ALU respectivamente.

ALU		
Fuente para entrada X (xop)	Fuente para entrada Y (yop)	Destino para puerto salida R
AX0, AX1, AR, MR0, MR1, MR2, SR0, SR1	AY0, AY1 AF	AR AF

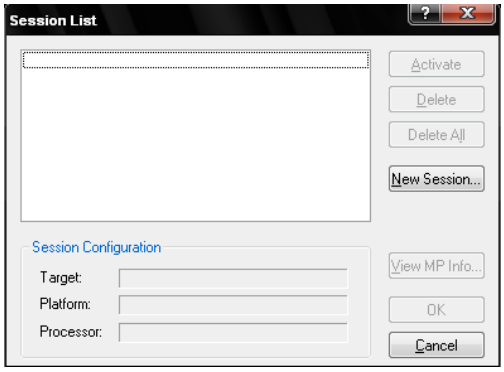
(IF Condición)	AR	$= xop$	$+ yop$	
	AF		$+ C$	
			$+ yop + C$	
			$+ cons\tan te$	
(IF Condición)	AR	$= xop$	$- yop$	
	AF		$- yop + C - 1$	
			$- cons\tan te$	
(IF Condición)	AR	$= yop$	$- xop$	
	AF		$- xop + C - 1$	
(IF Condición)	AR	$= xop$	AND	yop
	AF		OR	
			IOR	
(IF Condición)	AR	$= PASS$	xop	
	AF		yop	
			$-1,0,1$	
			$cons\tan te$	

(IF Condición)	$\begin{vmatrix} AR \\ AF \end{vmatrix} = - \begin{vmatrix} xop \\ yop \end{vmatrix}$
(IF Condición)	$\begin{vmatrix} AR \\ AF \end{vmatrix} = NOT \begin{vmatrix} xop \\ yop \end{vmatrix}$
(IF Condición)	$\begin{vmatrix} AR \\ AF \end{vmatrix} = ABS \begin{vmatrix} xop \end{vmatrix}$
(IF Condición)	$\begin{vmatrix} AR \\ AF \end{vmatrix} = yop \begin{vmatrix} +1 \\ -1 \end{vmatrix}$
(IF Condición)	$\begin{vmatrix} AR \\ AF \end{vmatrix} = \begin{vmatrix} TSTBIT \ \ n \ \ of \ \ xop \\ SBTBIT \ \ n \ \ of \ \ xop \\ CLBIT \ \ n \ \ of \ \ xop \\ TGTBIT \ \ n \ \ of \ \ xop \end{vmatrix}$
$NONE =< ALU >$	
$DIVS \quad yop, \quad xop$	
$DIVQ \quad xop$	

Procedimiento

- Inicializar el programa VisualDSP 3.5++. En caso de ser la primera vez que se ejecute el programa debe aparecer una pantalla como la que se muestra en la figura 69.

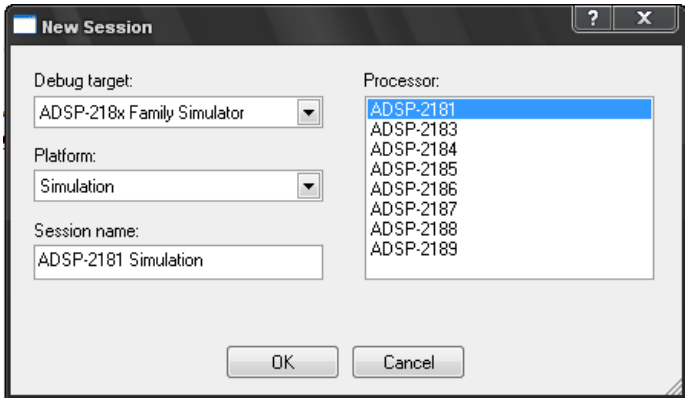
Figura 69. Figura 1-Laboratorio 1



- En la ventana *Session List* hacer clic en el botón *New Session*, aparcera la ventana correspondiente (ver figura 70), en el menú desplegable *Debug target* seleccionar la opción ADSP-218x *Family Simulator*, seleccione el procesador

ADSP-2181. La opción *Platform* permanece en *Simulation*, y por último se le asigna un nombre a la sesión en el cuadro de texto *Session name*. Finalizar dando clic en *OK*. Se retorna entonces a la ventana *Session List* donde se hace clic en el botón *Activate* dando como resultado la apertura de la ventana principal del VisualDSP 3.5++.

Figura 70. Figura 2- Laboratorio 1



Nota**:** En caso que no sea la primera vez en ejecutarse VisualDSP 3.5++ se abrirá la ventana principal con la última sesión trabajada.

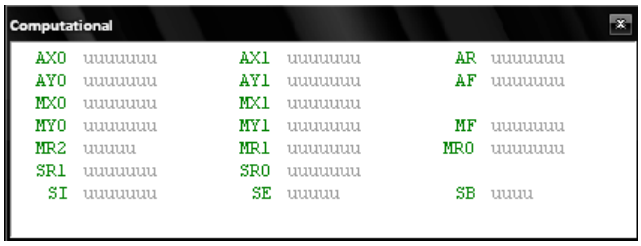
- En la barra *Menu*, en el menú *Project* seleccionar la opción Open y ubicar la carpeta con el nombre ALU que se encuentra en la dirección C:\Archivos de programa\Analog Devices\VisualDSP 3.5 16-Bit\2181_labs\ALU y seleccione el archivo ALU.dpj. Dar clic en *Open*. En la ventana *Project* se visualizaran las carpetas que conforman el proyecto.

- Depurar el proyecto, para esto en el menú Project de la barra *Menu*, seleccionar la opción *Build File* (F7). En la ventana de edición aparecerá el código fuente del proyecto. Verificar la correcta depuración del proyecto comprobando que en la pestaña *Console* de la ventana de salida se indique que la carga fue completa, y que en la pestaña *Build* se lea que la construcción fue completamente satisfactoria. Una ruta rápida para lograr la depuración es a través de la barra de Proyecto haciendo clic en el icono que se muestra en la figura 71.

Figura 71. Figura 3- Laboratorio 1

- Del menú *Register* seleccione la opción *Computational*, la ventana emergente (ver Figura 72) ubíquela en la parte inferior de la ventana Principal.

Figura 72. Figura 4-Laboratorio 1



- Ahora se procede a realizar la simulación paso a paso del programa, observe que después de haber sido depurado el proyecto un indicador (➪) aparece mostrando la línea actual del código fuente. Siempre después de realizar la depuración del proyecto el indicador se ubicara en la línea de código que realiza el salto al inicio del programa. Para recorrer el programa, ubíquese en el icono mostrado en la figura 73, este icono ejecuta la secuencia del programa paso a paso permitiendo ver los resultados de la ejecución de cada línea de código. El mismo proceso se logra al oprimir la tecla F11.

Figura 73. Figura 5 –Laboratorio 1

- Recorra paso a paso las primeras instrucciones.

```
AX0 = 0x0005;
AY0 = 0x003;
AR = AX0 + AY0;
AF = AX0 + 3;
```

Observe que a medida que se pasa a través de cada línea de código el contenido de los registros AX0, AY0, AF y AR cambian su valor en función de la instrucción que se ejecute.

¿De acuerdo al valor de los registros AR y AF, que diferencia existe entre las dos operaciones realizadas?
Rta. A pesar de que las dos instrucciones realizan una suma aritmética, la diferencia radica en que en la primera operación es el resultado de la suma de dos registros de entrada de la ALU cargados previamente con un valor cualquiera, mientras que la segunda operación es el resultado de la suma de un registro de entrada de la ALU con una constante.

- Retire la opción comentario (//) de la siguiente línea de instrucción. Cada vez que se realiza un cambio dentro del código fuente se debe depurar nuevamente el proyecto con el fin de cargar las modificaciones efectuadas.

```
//AR = AY0 + 3;
```

¿Que se visualiza en la ventana de salida luego de la depuración del programa? ¿Porque sucede esto?
Rta. La ventana de salida muestra que existe un error en la línea de código que se agregó, esto se debe a que no se cumple con la sintaxis de las instrucciones para la suma en la ALU que afirma que el primer sumando de la operación debe ser un registro de entrada X (xop)

- Ubicar la opción comentario (//) en la línea de instrucción anterior y depure nuevamente el proyecto.

- ¿Según el análisis previo que se puede deducir luego de recorrer las siguientes instrucciones?

```
AR = AX0 - AY0;
AR = AY0 - AX0;
AR = AX0 - 3;
```

Rta. A diferencia de la operación suma la sintaxis de la operación resta de la ALU permite que el sustraendo y el minuendo de la operación sean un registro de entrada X (xop) como también un registro de entrada Y (yop).

- Retire la opción comentario (//) de la siguiente línea de instrucción y depure nuevamente el proyecto.

```
//AR = AY0 - 3;
```

¿Que se visualiza en la ventana de salida luego de la depuración del programa? ¿Porque sucede esto?
Rta. La ventana de salida muestra que existe un error en la línea de código agregada, esto se debe a que no se cumple con la sintaxis de las instrucciones para la resta en la ALU, donde se afirma que cuando se va a restar una constante (minuendo) el sustraendo debe ser un registro de entrada X (xop)

- Ubicar la opción comentario (//) en la línea de instrucción anterior y depure nuevamente el proyecto.

- Recorra las siguientes instrucciones. Para observar mejor el resultado de las operaciones lógicas cambie el formato de visualización de los registros de la

ventana *Computational*, haciendo clic derecho dentro de la ventana y seleccionando la opción *Binary*.

```
AX0 = 3;
AY0 = 7;
AR = AX0 AND AY0;
```

Verifique que el valor del registro AR en la ventana *Computational* corresponde al resultado de la operación lógica AND entre AX0 y AY0. Repita el mismo procedimiento para las siguientes instrucciones y verifique las operaciones lógicas OR y XOR.

```
AR = AX0 OR AY0;
AR = AX0 XOR AY0;
```

- Retire la opción comentario (//) de las siguientes líneas de instrucción y depure nuevamente el proyecto.

```
//AR = AY0 AND AX0;
//AR = AY0 OR AX0;
//AR = AY0 XOR AX0;
```

¿Que se visualiza en la ventana de salida luego de la depuración del programa? ¿Porque sucede esto?
Rta. La ventana de salida muestra que existen tres errores a causa de las líneas de código agregadas, esto se debe a que no se cumple con la sintaxis de las instrucciones AND, OR y XOR.

- Ubicar la opción comentario (//) en las líneas de instrucción anteriores y depure nuevamente el proyecto.

- Recorra las siguientes instrucciones. En la ventana *Computational* verifique que los valores que toma AR después de ejecutada cada instrucción coincidan con el valor de los registros de entrada. Cambiar el formato de visualización de los registros de la ventana *Computational* a modo entero para tener claridad en la funcionalidad de la instrucción.

```
AR = PASS AX0;
AR = PASS AY0;
AR = PASS 5;
```

- Cambiar el formato de visualización de los registros de la ventana *Computational* a modo entero signado y recorra las siguientes instrucciones. En la

ventana *Computational* verificar que el valor que toma AR después de ejecutada la instrucción coincida con el valor absoluto del valor contenido en AX1.

```
AX1 = -8;
AR = ABS AX1;
```

- Recorra las siguientes instrucciones. En la ventana *Computational* verifique que el valor que toma AR después de ejecutada la instrucción coincida con el valor negado del valor contenido en el registro AX0.

```
AX0 = 0x5555;
AR = -AX0;
```

A continuación recorra la siguiente instrucción no sin antes haber cambiado el formato de visualización de los registros de la ventana *Computational* a modo binario

```
AR = NOT AX0;
```

¿Qué diferencia existe entre las dos operaciones anteriores?
Rta. La primera operación realiza una negación aritmética, donde se invierte el signo del valor contenido en el registro de entrada, mientras que la operación NOT realiza la negación lógica de cualquier valor.

- Recorra las siguientes instrucciones. En la ventana *Computational* verifique que el valor que toma AR y AF después de ejecutada la instrucción coincida con el incremento o decremento en 1 del valor contenido en el registro de entrada.

```
AF = AY0 + 1;
AR = AF - 1;
```

- Al llegar a la instrucción IDLE el programa se queda en un estado de baja potencia esperando la llegada de una interrupción externa.

Conclusiones

- El anterior procedimiento ilustró las características de la unidad computacional ALU, haciendo uso de la misma para realizar una variedad de funciones aritméticas que deben respetar la sintaxis propia del set de instrucciones de la ALU para que la depuración del proyecto se ejecute correctamente.

- Se hace clara la importancia de la funcionalidad básica del depurador en el proceso de localizar los errores causados por mala sintaxis dentro del código fuente.

- Las ventanas de registros y la opción de recorrer paso a paso el programa, son las herramientas propias del software que permiten observar los cambios provocados por la ejecución de las diferentes instrucciones del código fuente.

7.2 LABORATORIO 2. UNIDAD COMPUTACIONAL MAC

Objetivos.

- Comprobar la funcionalidad de la multiplicación, y de la multiplicación y acumulación simultanea, de la unidad computacional MAC.

- Hacer uso de las ventanas de registros del software de programación VisualDSP 3.5++.

Lecturas recomendadas.

- Profundizar en el manejo de las ventanas de registros del software de programación VisualDSP 3.5++.
- Consultar el set de instrucciones de la unidad computacional MAC

Marco Teórico.

La unidad multiplicadora y acumuladora (MAC) permite realizar operaciones de multiplicación y acumulación simultáneamente. Para realizar estas operaciones la MAC tiene dos grupos de operadores de entrada: X y Y de 16 bits y un operador P donde guarda el producto, el mismo es de 32 bits. Si la operación que deseamos realizar es de multiplicación, el operador P para directamente al registro MR (*multiplier result*) y si deseamos realizar un producto y acumulación, el operador P es sumado o restado al registro MR, quedando en MR el resultado final. El registro MR es de 40 bits y está formado por tres registros: MR2, MR1 y MR0. Estos dos últimos son de 16 bits cada uno y MR2 es de 8 bits

La adición y sustracción generan *overflow* y el bit que representa esto es el MV (MAC *overflow*) y su estado se puede ver en el registro ASTAT.

Los registros de entrada X (xop) aceptan datos de dos fuentes: MX y del bus de resultados (R). Hay dos registros dentro MX, MX0 y MX1, y los registros de entrada Y (yop) aceptan también datos de dos fuentes: el registro MY y el registro MF. El registro MY está compuestos por dos registros de 16 bits cada uno: M0 y M1. La salida de la adición o la sustracción puede ser cargada en el registro MF o MR. El registro MF es también un registro de realimentación en el cual están los bits 16-31 del resultado y puede ser usado directamente como operador de entrada Y en el próximo ciclo

La MAC del ADSP 21xx soporta dos modos para multiplicar y acumular: modo fraccionario para números de coma flotante y modo entero para números enteros. La selección del modo está representada en el bit 4 del registro MSTAT

En las siguientes tablas se muestra un resumen de los operandos de entrada permitidos para cada registro de entrada y los registros de resultado y un resumen de todas las instrucciones de la ALU respectivamente.

MAC		
Fuente para entrada X (xop)	Fuente para salida Y (yop)	Destino para puerto salida R
MX0, MX1, AR MR0,MR1,MR2 SR0,SR1	MY0, MY1 MF	MR (MR2, MR1, MR0) MF

(IF Condición)	$\begin{vmatrix} MR \\ MF \end{vmatrix}$	$= xop *$	$\begin{vmatrix} yop \\ xop \end{vmatrix}$	$\begin{vmatrix} (SS) \\ (SU) \\ (US) \\ (UU) \\ (RND) \end{vmatrix}$
(IF Condición)	$\begin{vmatrix} MR \\ MF \end{vmatrix}$	$= MR + xop *$	$\begin{vmatrix} yop \\ xop \end{vmatrix}$	$\begin{vmatrix} (SS) \\ (SU) \\ (US) \\ (UU) \\ (RND) \end{vmatrix}$

(IF Condición)	$\begin{vmatrix} MR \\ MF \end{vmatrix}$	$= MR - xop *$	$\begin{vmatrix} yop \\ xop \end{vmatrix}$	$\begin{vmatrix} (SS) \\ (SU) \\ (US) \\ (UU) \\ (RND) \end{vmatrix}$
(IF Condición)	$\begin{vmatrix} MR \\ MF \end{vmatrix}$	$= 0$		
(IF Condición)	$\begin{vmatrix} MR \\ MF \end{vmatrix}$	$= MR\ [(RND)]$		
$IF\ \ MV\ \ SAT\ \ MR$				

Procedimiento.

- Inicializar el programa VisualDSP 3.5++.

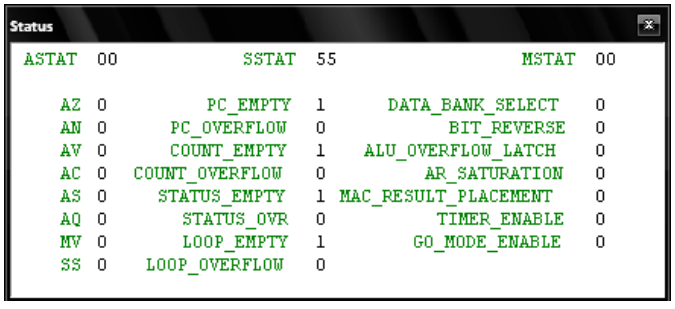
- En la barra *Menu*, en el menú *Project* seleccionar la opción *Open* y ubicar la carpeta con el nombre MAC que se encuentra en la dirección C:\Archivos de programa\Analog Devices\VisualDSP 3.5 16-Bit\2181_labs\MAC y seleccione el archivo MAC.dpj. Dar clic en Open. En la ventana *Project* se visualizaran las carpetas que conforman el proyecto.

- Depurar el proyecto.

- Del menú *Register* seleccione la opción *Computational*, la ventana emergente ubíquela en la parte inferior de la ventana Principal.

- Del menú *Register* seleccione la opción *Status*, la ventana emergente (ver figura 74) ubíquela en la parte lateral derecha de la ventana Principal.

Figura 74. Figura 1-Laboratorio 2



- Recorra paso a paso las primeras instrucciones.

```
MX0 = 10;
MY0 = 10;
MR = MX0 * MY0 (SS);
```

Observe el valor del registro MR0 en los distintos formatos que posee la ventana *Computational*. ¿Qué se puede deducir del valor presente en dicho registro? ¿Es el valor esperado del resultado de la operación de la MAC? ¿Por qué sucede esto?
Rta. Al observar el valor presente en el registro MR0 se deduce que no es el valor que se espera de la multiplicación de los dos registros de entrada de la MAC. El contenido del registro MR0 es un valor correcto y correspondiente con la multiplicación de los dos registros de entrada. Cuando se hace una operación de la MAC se debe tener en cuenta que por defecto la MAC asume los valores del los registros como números de coma flotante en formato 1.15 y el valor resultado en el registro MR0 es de tipo fraccional 1.31.

- Retire la opción comentario (//) de la siguiente línea de instrucción y depure nuevamente el proyecto.

```
//MR = MY0 * MY0 (SS);
```

¿Que se visualiza en la ventana de salida luego de la depuración del programa? ¿Porque sucede esto?
Rta. La ventana de salida muestra que existe un error en la línea de código agregada, esto se debe a que no se cumple con la sintaxis de las instrucciones para la multiplicación en la MAC, donde se afirma que el primer operando debe ser un registro de entrada X (xop)

- Ubicar la opción comentario (//) en la línea de instrucción anterior y depure nuevamente el proyecto.

- Retire la opción comentario (//) de la siguiente línea de instrucción y depure nuevamente el proyecto.

```
//MX0 = 0.5;
```

¿Que se visualiza en la ventana de salida luego de la depuración del programa? ¿Porque sucede esto?
Rta. La ventana de salida muestra que existe un error en la línea de código agregada, esto se debe a que no es posible asignar directamente un valor de coma flotante a un registro de entrada de la MAC.

- Ubicar la opción comentario (//) en la línea de instrucción anterior y depure nuevamente el proyecto.

- Recorra las siguientes instrucciones. Para observar mejor el resultado de las operaciones cambie el formato de visualización de los registros de la ventana *Computational* a modo *Fractional*.

```
MX0 = 0.5r;
MY0 = 0.5r;
MR = MX0 * MY0 (SS);
```

Observe el contenido del registro MR1. ¿Qué puede deducir del valor del registro? ¿Qué influencia tiene la letra "r" en los valores de los registros de entrada de la MAC?
Rta. El valor de MR1 es el resultado de multiplicar los números fraccionarios contenidos en los registros MX0 y MY0. La presencia de la letra "r" al final de los números de coma flotante, corresponde a la asignación que se le da al número para que sea identificado por la MAC como numero de coma flotante sin formato.

- Recorra las siguientes instrucciones. Para observar mejor el resultado de las operaciones cambie el formato de visualización de los registros de la ventana *Computational* a modo *Signed Integer*.

```
ENA M_MODE;
MX0 = 10;
MY0 = 10;
MR = MX0 * MY0 (SS);
MR = MX0 * MX0 (SS);
DIS M_MODE;
```

Observe el contenido del registro MR0. ¿Qué función cumple la presencia de la instrucción ENA M_MODE antes de las instrucciones de multiplicación y la instrucción DIS M_MODE al final?
Rta. La instrucción ENA M_MODE permite que la MAC asuma los valores contenidos en los registros de entrada de la MAC como números enteros, y la instrucción DIS M_MODE vuelve a colocar a la MAC en su modo por defecto de valores de los registros de entrada como números de coma flotante.

- ¿Qué relación tienen los registros MR0 y MR1 con respecto al resultado de las operaciones (MR) que se han ejecutado a lo largo del procedimiento?
Rta. Los registros de entrada de la MAC son valores de 16 bits cada uno, por consiguiente el resultado de su producto es un valor de 32 bits, el cual es consignado en dos registros que son MR0 y MR1, donde MR0 contiene los bits de menor valor y MR1 los bits de mayor valor.

- Recorra las siguientes instrucciones. Para observar mejor el resultado de las operaciones deje el formato de visualización de los registros de la ventana *Computational* en modo *Signed Integer*.

```
ENA M_MODE;
MR1 = 0x7FFF;
MX0 = 0x00FA;
MY0 = 0x00FB;
MR = MR + MX0 * MY0 (SS);
```

- Verifique que el contenido final del registro MR corresponda a la multiplicación de los registros de entrada más el valor previo contenido en MR1 y MR0.

- Recorra la siguiente instrucción.

```
MR = MR + MX0 * MY0 (SS);
```

Compruebe en la ventana Status el cambio de estado del bit 6 del registro ASTAT ¿Qué provocó este cambio?
Rta. El bit 6 del registro ASTAT es el indicador de overflow dentro de la MAC. Su cambio de estado indica que el registro MR ha superado su máximo valor positivo el cual corresponde a 0x7FFF.

- Observe el valor contenido en el registro MR, a continuación recorra la siguiente instrucción. Para observar mejor el resultado de la operación establezca el formato de visualización de los registros de la ventana *Computational* en modo Hexadecimal.

```
IF MV SAT MR;
```

Observe el nuevo valor de MR ¿Qué funcionalidad tiene la instrucción anterior?
Rta. La instrucción IF MV SAT MR evalúa el estado del bit 6 (MV) del registro ASTAT, si su valor es 1 entonces carga el registro MR (MR0 y MR1) con su valor máximo positivo.

- Al llegar a la instrucción IDLE el programa se queda en un estado de baja potencia esperando la llegada de una interrupción externa.

Conclusiones

- El anterior procedimiento ilustró las características de la unidad computacional MAC, haciendo uso de la misma para realizar una variedad de funciones aritméticas que deben respetar la sintaxis propia del set de instrucciones de la MAC para que la depuración del proyecto se ejecute correctamente.

- Se resaltó que el modo de operación que por defecto tiene la MAC para la ejecución de instrucciones corresponde al modo fraccionario, haciendo énfasis en que si se desea trabajar con números enteros el usuario debe cambiar el modo de trabajo de la unidad a modo entero.

- Se expusieron las diferentes opciones de representación de los valores contenidos en los registros dentro de la ventana Computacional los cuales facilitan el entendimiento y permiten la verificación de las operaciones que realiza la unidad.

7.3 LABORATORIO 3. UNIDAD COMPUTACIONAL SHIFTER

Objetivos.

- Comprobar la funcionalidad de las funciones de desplazamiento para datos de 16 bits de entrada, de la unidad computacional MAC.

- Hacer uso de las ventanas de registros del software de programación VisualDSP 3.5++.

Lecturas recomendadas.

- Consultar el set de instrucciones de la unidad computacional SHIFTER

Marco Teórico.

La unidad de desplazamiento (*Shifter*) Provee un completo set de funciones de desplazamiento (*shifting*) para datos de 16 bits de entrada dejándolos en 32 bits. Incluye *shift* aritmético, *shift* lógico, normalización, desnormalización y detección del máximo exponente. También permite detectar exponente y ajustarlo. Es capaz de realizar funciones básicas con números en formato punto flotante.

El Shifter tiene cuatro registros principales:

- SI: "*shifter input*" se utiliza para cargar los datos de entrada del array de desplazamiento y del detector de exponente. 16 bits.
- SR: "*shifter result*" se utiliza para guardar los datos de salida. 32 bits. Esta dividido en dos registros de 16 bits cada uno: SR0 y SR1
- SE: "*shifter exponent*" se utiliza para cargar el exponente en complemento a dos durante la normalización y desnormalización de números. 8 bits
- SB: "*shifter block*" se utiliza para cargar el valor del menor exponente posible y ser actualizado con el valor del exponente del numero en proceso si este es mayor que el valor cargado anteriormente en SB.

El set de instrucciones nos provee para esta unidad de tres modificadores: HIX, HI y LO. Además nos permite realizar un OR lógica entre el valor que tiene cargado anteriormente el SR y el valor que esta por adquirir en la nueva operación. En el ASTAT podemos encontrar el signo del desplazamiento (SS bit 7). El shifter posee una serie de operaciones que son:

- Desplazamiento aritmético (ASHIFT)
- Desplazamiento lógico (LSHIFT)
- Normalizar (NORM)
- Detección del exponente (EXP)
- Ajuste del exponente (EXPADJ)

En las siguientes tablas se muestra un resumen de los operandos de entrada permitidos para cada registro de entrada y los registros de resultado y un resumen de todas las instrucciones de la ALU respectivamente.

Shifter	
Fuente para entrada Shifter (xop)	Destino para salida Shifter
SI, SR0, SR1 AR MR0, MR1, MR2	SR (SR1, SR0)

(IF Condición)	$SR = [SR\ OR]\ ASHIFT\ xop \left(\begin{vmatrix} HI \\ LO \end{vmatrix} \right)$
(IF Condición)	$SR = [SR\ OR]\ LSHIFT\ xop \left(\begin{vmatrix} HI \\ LO \end{vmatrix} \right)$
(IF Condición)	$SR = [SR\ OR]\ NORM\ xop \left(\begin{vmatrix} HI \\ LO \end{vmatrix} \right)$
(IF Condición)	$SR = EXP\ xop \left(\begin{vmatrix} HI \\ LO \\ HIX \end{vmatrix} \right)$
(IF Condición)	$SE = EXPDJ\ xop$
$SR = [SR\ OR]\ ASHIFT\ xop\ by\ \langle data \rangle \left(\begin{vmatrix} HI \\ LO \end{vmatrix} \right)$	
$SR = [SR\ OR]\ LSHIFT\ xop\ by\ \langle data \rangle \left(\begin{vmatrix} HI \\ LO \end{vmatrix} \right)$	

Procedimiento.

- Inicializar el programa VisualDSP 3.5++.

- En la barra *Menu*, en el menú *Project* seleccionar la opción *Open* y ubicar la carpeta con el nombre SHIFTER que se encuentra en la dirección C:\Archivos de programa\Analog Devices\VisualDSP 3.5 16-Bit\2181_labs\SHIFTER y seleccione el archivo SHIFTER.dpj. Dar clic en *Open*. En la ventana *Project* se visualizaran las carpetas que conforman el proyecto.

- Depurar el proyecto.

- Del menú *Register* seleccione la opción *Computational*, la ventana emergente ubíquela en la parte inferior de la ventana Principal.

- Recorra paso a paso las primeras instrucciones. Para observar mejor el resultado de las operaciones cambie el formato de visualización de los registros de la ventana *Computational* a modo *Binary*.

```
SI = 0x9999;
SR = LSHIFT SI BY 2(HI);
```

Observe el valor contenido en los registros SR1 y SR0. ¿Qué labor desempeña la instrucción LSHIFT BY 2? ¿Qué labor desempeña el termino (HI) en la instrucción?
Rta. La instrucción LSHIFT es la encargada de realizar el desplazamiento lógico del valor contenido en el registro de entrada del shifter (SI). El segmento de código correspondiente a "BY 2", es el indicador del número de posiciones que el registro SI será desplazado ya sea a la izquierda o a la derecha. El término (HI) indica que el valor de 16 bits del registro SI será colocado en el registro SR1, para luego ser desplazado por la instrucción LSHIFT.

- Recorra la siguiente instrucción.

```
SR = LSHIFT SI BY 2(LO);
```

¿Con relación al valor en SR1 y SR0 que puede concluir del término (LO)?
Rta. El término (LO) indica que el valor de 16 bits del registro de entrada del shifter será ubicado en el registro SR0 para luego ejecutar la instrucción del de desplazamiento del shifter.

- Recorra s siguientes instrucciones.

```
SR = ASHIFT SI BY 2 (HI);
SR = ASHIFT SI BY 2 (LO);
```

Observe el valor contenido en los registros SR1 y SR0. ¿Qué labor desempeña la instrucción ASHIFT BY 2?
Rta. La instrucción ASHIFT BY 2 es la encargada de realizar el desplazamiento aritmético del registro de entrada del shifter (SI) un número predeterminado de posiciones, que para este caso en particular es de 2 posiciones en sentido positivo. Igual que para el caso de la instrucción LSHIFT los términos HI y LO son los encargados de indicar en qué registro (SR1 o SR0) será ubicado el valor contenido en SI.

- Recorra s siguientes instrucciones.

```
SI = 0xC000;
SR = ASHIFT SI BY -8 (HI);
```

Observe el valor contenido en el registro SR1. Antes de recorrer la siguiente instrucción tome nota de dicho valor en formato *Binary*, Hexadecimal, y *Signed Integer*.

```
SR = LSHIFT SI BY -8 (HI);
```

Compare el valor del registro SR1 luego de haber ejecutado las instrucciones de ASHIFT y LSHIFT. ¿Por qué el desplazamiento de las dos instrucciones se realizó hacia la derecha? ¿Por qué los 8 bits más significativos del registro SR1 son 1 luego de ejecutarse la instrucción de ASHIFT? ¿Qué quiere decir que el valor de SR1 sea 0x00C0 luego del LSHIFT y 0xFFC0 luego del ASHIFT?
Rta. El desplazamiento de las dos instrucciones se realizó a la derecha debido a que el valor indicado de corrimiento es negativo. Luego de que se ejecuta una instrucción de ASHIFT hacia la derecha los bits que ingresan al registro pueden ser 1 o 0 dependiendo del bit más significativo antes de desplazar. Debido a que la palabra que se desplaza es signada el desplazamiento aritmético asegura que el signo del resultado es igual al signo del operador. El valor contenido en SR1 en formato hexadecimal muestra como para la operación lógica el desplazamiento es de 8 bits a la derecha (0xC000 inicial, 0x00C0 final), mientras que para la operación aritmética el desplazamiento es de 8 bits a la derecha, y como el bit de signo de la palabra muestra que la palabra es signada los 8 bits que ingresan son todos uno para mantener el signo luego de la operación (0xC000 inicial, 0xFFC0 final)

- Recorra paso a paso las primeras instrucciones. Para observar mejor el resultado de las operaciones cambie el formato de visualización de los registros de la ventana *Computational* a modo *Binary*.

```
SR = LSHIFT SI BY -2 (HI);
SR = SR OR LSHIFT SI BY -8 (HI);
SR = SR OR ASHIFT SI BY -8 (HI);
```

Observe el resultado de las operaciones en el registro SR1. Verifique que las operaciones realizadas corresponden a una operación de desplazamiento del registro SI (ASHIFT y LSHIFT), y luego a una operación lógica OR del resultado del desplazamiento con el valor previo almacenado en SR.

- Recorra paso a paso las primeras instrucciones.

```
SE = -8;
SI = 0xFF00;
SR = LSHIFT SI (LO);
SR = ASHIFT SI (LO);
```

¿Qué se puede deducir del valor presente en SR0 y SR1 luego de ejecutarse las instrucciones de LSHIFT y ASHIFT? ¿Qué diferencia existe entre las operaciones que se realizaron a lo largo del procedimiento, comparadas con las últimas instrucciones ejecutadas?
Rta. Los valores que toman los registros SR0 y SR1 corresponden al resultado de las operaciones de desplazamiento permitidas por la unidad computacional Shifter.

La diferencia que existe radica en que a lo largo del procedimiento se indicaba en la instrucción del Shifter el valor correspondiente al número de posiciones de desplazamiento, mientras que en las últimas instrucciones ejecutadas el numero de posiciones de desplazamiento es indicado por un valor almacenado en el registro SE

- Al llegar a la instrucción IDLE el programa se queda en un estado de baja potencia esperando la llegada de una interrupción externa.

Conclusiones.

- El anterior procedimiento ilustró las características de la unidad computacional SHIFTER, haciendo uso de la misma para realizar una variedad de funciones de desplazamiento que deben respetar la sintaxis propia del set de instrucciones del SHIFTER para que la depuración del proyecto se ejecute correctamente.

- Se mostró como existen dos clases de desplazamientos, que para los lógicos los bits que ingresan son siempre cero, mientras que para los aritméticos los bits que ingresan pueden ser cero o uno dependiendo del signo del operador. En el caso del desplazamiento aritmético, este asegura que el signo del resultado será el mismo del operador luego de ejecutarse la instrucción.

- La unidad computacional Shifter acepta 16 bits de entrada que se pueden desplazar en cualquier sentido en el registro de 32 bits de salida. El desplazamiento se realiza en un solo ciclo sin importar que tipo de desplazamiento sea.

7.4 LABORATORIO 4. SECUENCIADOR DE PROGRAMA E INSTRUCCIONES MULTIFUNCION

Objetivos.

- Verificar como el secuenciador de programa controla el flujo de la ejecución del programa.

- Observar las ventajas de las instrucciones multifunción

Lecturas recomendadas.

- Consultar las teoría de la lógica de del secuenciador de programa.
- Consultar el conjunto de instrucciones de control de flujo del programa.

Marco Teórico.

Secuenciador de Programa. El secuenciador de programa provee direcciones para el encapsulado o para el acceso externo a la memoria. El secuenciador le da soporte a ciclos sencillos de ramificaciones condicionales y ejecuta programas cíclicos con pérdidas iguales a cero. El circuito del secuenciador de programa controla el flujo de la ejecución del programa. Contiene un controlador de interrupción y estado, y condición lógica. El secuenciador le provee un control flexible al flujo del programa. Permite la ejecución secuencial de instrucciones, bucles de "*zero-overhead*", sofisticado servicio de interrupciones, y ramificaciones de un solo ciclo con saltos de llamada (tanto condicional como incondicional).

Las siguientes son instrucciones utilizadas para el control del flujo del programa:

- DO UNTIL
- JUMP
- CALL
- RTS (retorno desde subrutina)
- RTI (retorno desde interrupción)
- IDLE

Una condición opcional "IF" puede primero evaluar alguno de los estados de las condiciones permisibles. En las siguientes tablas se muestra un resumen de condiciones permitidas para el servicio de interrupciones, llamado de subrutinas y saltos, y un resumen de todas las instrucciones de control del flujo del programa respectivamente.

Condición	Keyword
Si el resultado ALU es:	
Igual a cero	EQ
No igual a cero	NE
Mayor que cero	GT
Mayor que o igual a cero	GE
Menor que cero	LT
Menor que o igual a cero	LE

Condición	Keyword
Estado del Acarreo ALU	
Acarreo	AC
No Acarreo	NOT AC
Signo entrada x	
Positivo	POS
Negativo	NEG
Estado del overflow ALU	
Overflow	AV
No overflow	NOT AV
Estado del contador	
Expirado	CE
No expirado	NOT CE

(IF Condición)	*JUMP*	$(I4)$ $(I5)$ $(I6)$ $(I7)$ $\langle address)$	
(IF Condición)	*CALL*	$(I4)$ $(I5)$ $(I6)$ $(I7)$ $\langle address)$	
IF	*FLAG IN* *NOT FLAG IN*	*CALL* $\langle address)$	
IF	*FLAG IN* *NOT FLAG IN*	*JUMP* $\langle address)$	
(IF Condición)	*RTS*		
(IF Condición)	*RTI*		
DO $\langle address\rangle$	(*UNTIL ter* min *antion*)		
IDLE, *IDLE* (n)	N? 13, 32, 64, 128		

Instrucciones Multifunción. Las operaciones de multifunción toman ventaja del paralelismo inherente de la arquitectura del ADSP-218x proporcionando combinaciones de movimientos de datos, lectura y escritura de memoria y computación, todas en un solo ciclo.

- ALU/MAC con lectura de datos y programas de la memoria. Tal vez la operación simple más común en el algoritmo DSP es la suma de productos. El procesador ADSP- 218x puede ejecutar tanto la obtención de datos como la multiplicación/acumulación en un solo ciclo.

- Lectura de Datos y programas de la memoria. Esta variación de una instrucción de multifunción es un caso especial de una instrucción multifunción.

- Cálculos con lectura de la memoria. Si una simple lectura de memoria es realizada en lugar de una doble lectura de memoria de las dos instrucciones de multifunción previas, una gama más amplia de cálculos puede ser ejecutada.

- Cálculos con escritura de la memoria. Los cálculos con la instrucción de escritura de memoria son similares en estructura a los cálculos con la lectura de memoria: el orden de las cláusulas en la línea de instrucción, sin embargo, es revertido.

- Cálculos con movimiento de registro de datos. Esta instrucción multifunción realiza un movimiento de un registro de dato a un registro de dato en paralelo con los cálculos.

En las siguientes tablas se resumen las instrucciones multifunción y se muestran las combinaciones posibles de las instrucciones multifunción respectivamente. Se pueden combinar operaciones sobre la misma fila con cada una de las demás.

$<ALU>*\perp$	.	$AX0$	$=DM($	$I0$	.	$M0$	).	$AY0$	$=PM($	$I4$	.	$M4$	).
$<MAC>*\perp$		$AX1$		$I1$	.	$M1$		$AY1$		$I5$	.	$M5$	
		$MX0$		$I2$	.	$M2$		$MY2$		$I6$	.	$M6$	
		$MX1$		$I3$	.	$M3$		$MY3$		$I7$	.	$M7$	

$\begin{vmatrix} AX0 \\ AX1 \\ MX0 \\ MX1 \end{vmatrix} = DM(\begin{vmatrix} I0 \\ I1 \\ I2 \\ I3 \end{vmatrix} . \begin{vmatrix} M0 \\ M1 \\ M2 \\ M3 \end{vmatrix}) . \begin{vmatrix} AY0 \\ AY1 \\ AY2 \\ AY3 \end{vmatrix} = PM(\begin{vmatrix} I4 \\ I5 \\ I6 \\ I7 \end{vmatrix} . \begin{vmatrix} M4 \\ M5 \\ M6 \\ M7 \end{vmatrix});$
$\begin{vmatrix} <ALU>* \\ <MAC>* \\ <SHIFT>* \end{vmatrix} . dreg = \begin{vmatrix} DM(\begin{vmatrix} I0 \\ I1 \\ I2 \\ I3 \\ \\ I4 \\ I5 \\ I6 \\ I7 \end{vmatrix} . \begin{vmatrix} M0 \\ M1 \\ M2 \\ M3 \\ \\ M4 \\ M5 \\ M6 \\ M7 \end{vmatrix}) \end{vmatrix}$
$\begin{vmatrix} DM(\begin{vmatrix} I0 \\ I1 \\ I2 \\ I3 \\ \\ I4 \\ I5 \\ I6 \\ I7 \end{vmatrix} . \begin{vmatrix} M0 \\ M1 \\ M2 \\ M3 \\ \\ M4 \\ M5 \\ M6 \\ M7 \end{vmatrix}) = dreg . \begin{vmatrix} <ALU>* \\ <MAC>* \\ <SHIFT>* \end{vmatrix} \end{vmatrix}$
$\begin{vmatrix} <ALU>* \\ <MAC>* \\ <SHIFT>* \end{vmatrix} .dreg = dreg$

Donde:

<ALU> Cualquier instrucción ALU (excepto DIVS, DIVQ)
<MAC> Cualquier instrucción multiplicación/acumulación
<SHIFT>| Cualquier instrucción Shifter (excepto *Shift Inmediate*)

Cálculos Incondicionados	Movimiento Datos (DM=DAG1)	Movimiento Datos (PM=DAG2)
Ningún o cualquier ALU (excepto División) o MAC	Lectura DM	Escritura PM
Cualquier MAC Cualquier ALU excepto División Cualquier Shift excepto Inmedite	-- Lectura DM -- Escritura DM -- Registro a Registro	-- Lectura PM -- Escritura PM

Procedimiento.

- Inicializar el programa VisualDSP 3.5++.

- En la barra *Menu*, en el menú Project seleccionar la opción Open y ubicar la carpeta con el nombre SEQUENCER que se encuentra en la dirección C:\Archivos de programa\Analog Devices\VisualDSP 3.5 16-Bit\2181_labs\SEQUENCER y seleccione el archivo SEQUENCER.dpj. Con el puntero del ratón sobre la opción *Open* oprimir el botón izquierdo del mouse. En la ventana Project se visualizaran las carpetas que conforman el proyecto.

- Depurar el proyecto.

- Del menú *Register* seleccione la opción *Computational*, la ventana emergente ubíquela en la parte inferior de la ventana Principal.

- Del menú *Register* seleccione la opción Status, la ventana ubíquela en la parte lateral derecha de la ventana Principal.

- Recorra las primeras instrucciones paso a paso

```
AX0 = 8;
JUMP Etiqueta_Uno;
```

Observe que el registro AX0 ha sido cargado con un valor para ser utilizado más adelante en alguna parte del programa. También note que la instrucción JUMP

hace saltar el cursor a la posición en la que se encuentra la etiqueta marcada con el nombre de Etiqueta_Uno.

- Recorra las siguientes instrucciones.

```
Etiqueta_Uno:

        CALL Rutina_Uno;
        JUMP Etiqueta_Dos;
```

Verifique como la instrucción CALL hace un llamado de una subrutina que se encuentra dentro del mismo programa. No necesariamente la subrutina debe estar en el mismo programa sobre el que se está trabajando, los llamados de subrutinas pueden ser externos siempre y cuando el código fuente donde se encuentra la subrutina se encuentre dentro de los códigos del proyecto actual, y solamente si las subrutinas utilizadas son definidas como globales dentro del código al que pertenecen y como externas dentro del código donde se llaman. El CALL que se ejecutó realiza las siguientes instrucciones.

```
Rutina_Uno:

        SE = AX0;
        SI = 0x8888;
        SR = LSHIFT SI (HI);
        RTS;
```

¿Qué operación realizó la subrutina Rutina_Uno?
Rta. La operación que se realiza dentro del llamado de subrutina corresponde a una operación de la unidad computacional Shifter. El valor del número de posiciones de desplazamiento es el que almacenó antes del llamado de subrutina en el registro AX0.

Luego de retornar del llamado de subrutina el cursor se sitúa en la siguiente instrucción luego del CALL, la cual hace un JUMP a la Etiqueta_Dos que contiene el siguiente segmento de código.

```
Etiqueta_Dos:

        AX0 = 5;
        CNTR = AX0;
        AR = 0;

        DO Etiqueta_Tres UNTIL CE;
Etiqueta_Tres: AR = AR + 2;
        IDLE;
```

NOP;

Observe en la ventana Program Control como el valor de CNTR se reduce a medida que se alcanza la Etiqueta_Tres. ¿A qué se debe el valor de AR luego de que el contador llegó a un valor de cero?
Rta. La instrucción DO está condicionada al valor de CNTR, es decir que se van a realizar las instrucciones que existen entre DO y la etiqueta de condición de finalización del lazo tantas veces como sean indicadas por el valor del CNTR. Para el anterior caso se realiza un incremento del valor contenido en AR luego de cada bucle.

Luego de haber recorrido todo el programa, ¿Qué quiere decir la instrucción RTS?
Rta. Cuando se realiza un salto dentro de un programa la posición desde la cual se realizó el salto es almacenada dentro de una posición de memoria. Cuando la subrutina llega a la instrucción RTS esta le indica al programa que debe regresar a la siguiente instrucción de donde fue llamada.

- Al llegar a la instrucción IDLE el programa se queda en un estado de baja potencia esperando la llegada de una interrupción externa.

- En la barra *Menu*, en el menú *Project* seleccionar la opción *Close* para cerrar el proyecto del secuenciador de programas.

- A continuación en la barra *Menu*, en el menú *Project* seleccionar la opción *Open* y ubicar la carpeta con el nombre MULTI que se encuentra en la dirección C:\Archivos de programa\Analog Devices\VisualDSP 3.5 16-Bit\2181_labs\MULTI y seleccione el archivo Multifunction.dpj. Con el puntero del raton sobre la opción *Open* oprimir el botón izquierdo del mouse. En la ventana *Project* se visualizaran las carpetas que conforman el proyecto.

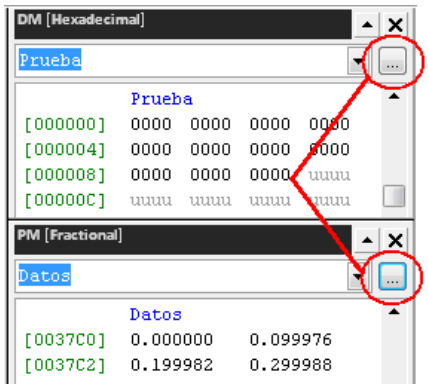
- En la barra *Menu,* en el menú *Memory* seleccione la opción *Program.* La ventana emergente ubíquela al lado derecho de la ventana principal. Repita este procedimiento para la opción *Data* del menú *Memory.*

- En la barra *Menu,* en el menú *Register* seleccione la opción *Computational.* La ventana emergente ubíquela al lado derecho de la ventana principal debajo de las ventanas de PM y DM.

- Las ventanas PM y DM permiten observar los buffers definidos para memoria de datos y memoria de programa. Haga clic izquierdo del mouse con el puntero ubicado sobre el botón que se encuentra junto al menú desplegable de la ventana *Data (DM)*. La posición del botón se muestra en la figura 75. En la ventana emergente seleccione la opción “Prueba”. ¿Qué se muestra en las posiciones de la ventana de memoria de datos?

Rta. Los valores iniciales del segmento de memoria asignado a “Prueba” se muestran con un valor de cero (valor por defecto).

Figura 75. Figura 1-Laboratorio 4



- Repita el procedimiento del paso anterior en la ventana Program (PM) para la opción “Datos”.

- Para la ventana *Program (PM),* selección que el formato de los datos de dicha ventana sea de tipo *Fractional.* ¿A que corresponden los valores mostrados en la ventana de memoria de programa para el segmento de “Datos”? ¿Por qué los valores mostrados se ven diferentes a los esperados?
Rta. Los valores mostrados en la ventana de programa corresponden a los valores que se encuentran almacenados en el archivo Datos.dat. La diferencia que se presenta entre los valores existentes en el archivo y los mostrados en la ventana PM, se debe a que VisualDSP++ toma los valores del archivo como fraccionarios que han sido aproximados o redondeados, es por eso que a pesar que se espera un valor específico proveniente del archivo, lo que se lee es un valor cercano al esperado.

- Ubique el cursor sobre la ventana de edición (la ventana donde se encuentra el código fuente), y oprima el botón derecho del mouse para desplegar el menú del cual debe seleccionar la opción *Line Numbers*.

- Las líneas de código 27 al 31 se muestran a continuación.

```
AX1 = 0xA0A0;
AY1 = 0;
AX0 = 0x0002;
AY0 = 0x0002;
AR = AX0 + AY0 , AY1 = AX1;
```

Recorra paso a paso el programa hasta la línea de código 30. Observe en la ventana *Computational* como los valores de los registros van cambiando a medida que se ejecuta cada instrucción.

- Recorra la línea de código número 30 al mismo tiempo que observa los registros AR y AY1. ¿Qué se puede deducir de la ejecución de la línea de código número 30?

Rta. La línea de código número 30 corresponde a una línea de código de multifunción, en donde se realiza una operación de la ALU y un movimiento de registros en un mismo ciclo.

- Las líneas de código 33 a la 46 se muestran a continuación.

```
I2 = Prueba;
M2 = 1;
L2 = LENGTH(Prueba);
AX0 = 15;
CNTR = 11;
DO Llenado UNTIL CE;
Llenado:            DM(I2,M2) = AX0;

ENA M_MODE;
AX0 = 0;
MX0 = 0x0002;
MY0 = 0x0002;
MR = MX0 * MY0 (SS), AX0 = DM(I2,M2);
DIS M_MODE;
```

Las líneas 33 a 39 corresponden a un ciclo DO como el visto en este laboratorio en el proyecto anterior, que se encarga de llenar las posiciones del buffer “Prueba” con el valor contenido en AX0. El modo de funcionamiento de Ix,Mx, y Lx (x es un valor entre 1 y 7) será explicado en próximos laboratorios. Recorra paso a paso el programa hasta la línea 44.

- Recorra la línea de código número 45 al mismo tiempo que observa los registros MR y AX0. ¿Qué se puede deducir de la ejecución de la línea de código número 45?

La línea de código 45 es una instrucción de multifunción en donde se realiza una operación de la MAC y una lectura de memoria en un mismo ciclo.

- Las líneas de código 49 a 53 se muestran a continuación.

```
AX0 = 0xAAAA;
I2 = 2;
SE = -1;
SI = 0x9999;
DM(I2,M2) = AX0 , SR = LSHIFT SI(LO);
```

Recorra las anteriores líneas de código. Al momento de recorrer la línea número 53 asegúrese de observar la ventana de memoria de datos (DM) y la ventana *Computational.* ¿Qué se puede deducir de la ejecución de la línea de código número 53?
Rta. La línea de código 53 es una instrucción de multifunción en donde se realiza una escritura de memoria y una operación del SHIFTER en un mismo ciclo.

- Las líneas de código 55 a 59 se muestran a continuación.

```
AY0 = 0;
I4 = Datos;
M4 = 1;
L4 = LENGTH(Datos);
AX0 = DM(I2,M2) , MY0 = PM(I4,M4);
```

Recorra las anteriores líneas de código. Al momento de recorrer la línea número 59 asegúrese de observar la ventana de memoria de datos (DM) y la ventana memoria de programa (PM). ¿Qué se puede deducir de la ejecución de la línea de código número 59?
Rta. La línea de código 59 es una instrucción de multifunción en donde se realiza de datos (DM) y lectura de memoria de programa (PM) en un mismo ciclo.

- Las líneas de código 61 a 64 se muestran a continuación.

```
I2 = 3;
AX0 = 5;
AY0 = 4;
AR = AX0 + AY0 ;AX0 = DM(I2,M2) , MY0 = PM(I4,M4);
```

Recorra las anteriores líneas de código. Al momento de recorrer la línea número 64 asegúrese de observar la ventana de memoria de datos (DM), la ventana memoria de programa (PM) y la ventana *Computational.* ¿Qué se puede deducir de la ejecución de la línea de código número 64?
Rta. La línea de código 64 es una instrucción de multifunción en donde se realizan tres operaciones diferentes que son, una operación de la ALU, lectura de memoria de datos (DM), y lectura de memoria de programa (PM)

Conclusiones.

- El anterior programa ilustró la forma en que el secuenciador de programa da sentido, orientación y flexibilidad al control del flujo del programa, utilizando instrucciones propias del secuenciador y la lógica del mismo.

- Se hace clara la importancia de la funcionalidad del secuenciador en el proceso de desarrollo de un proyecto ya que este ofrece las herramientas propias de programación para darle secuencia, y lógica al código fuente del proyecto. Las herramientas JUMP, CALL y DO UNTIL, permiten que el código que se desarrolla sea estructurado y ordenado, y que al mismo tiempo tenga una estructura orientada a programación por segmentos de código

- Las operaciones multifunción hace uso de las ventajas de la arquitectura de la familia de procesadores ADSP-218x al permitir el uso de varias instrucciones en un ciclo sencillo, donde dichas instrucciones pueden ser operaciones de la ALU, la MAC, el SHIFTER, mezcladas con lectura y escritura de la memorias de datos y de programa.

- El alto nivel de paralelismo permite optimizar el modo en que se estructura el programa al economizar líneas de código y salvando ciclos de máquina, haciendo que el programa sea efectivo, ordenado, estructurado y lo más importante, veloz.

7.5 LABORATORIO 5. EJECUCIÓN DE PROYECTOS EN TIEMPO REAL E INTERRUPCIÓN EXTERNA DEL SOFTWARE DE PROGRAMACIÓN VISUAL DSP3.5++

Objetivos.

- Familiarizar al estudiante con el proceso básico a seguir en la carga y ejecución en tiempo real de un proyecto para el sistema de desarrollo ADSP-2181 DSPKit, utilizando el software de programación VisualDSP 3.5++.

- Verificar el modo de funcionamiento de una interrupción externa.

- Verificar el cambio de estado de una bandera externa.

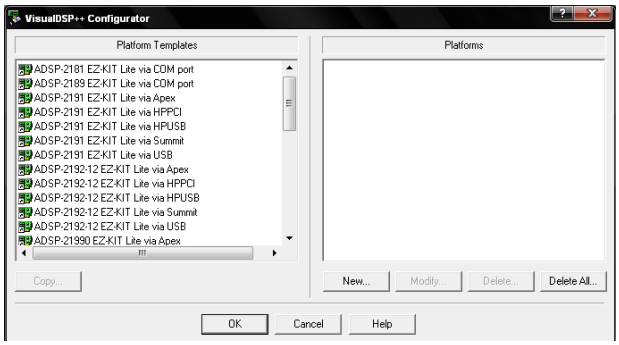
Lecturas recomendadas.

- Consultar las directivas para la asignación de interrupciones.
- Consultar el diagrama esquemático del sistema de desarrollo ADSP-2181 DSPKit.

Procedimiento.

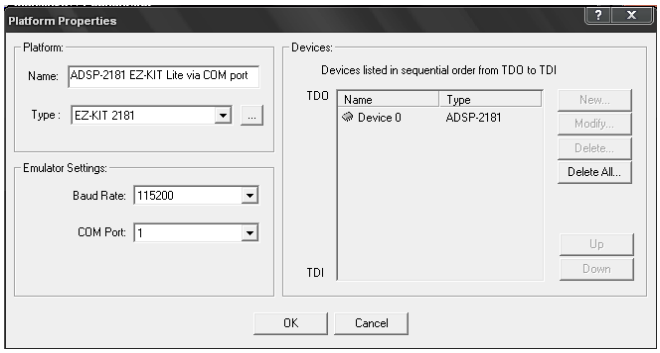
- Debido a que es la primera vez que se va a utilizar la tarjeta en un modo diferente al de simulación, es necesario configurar las opciones de comunicación entre la tarjeta y el computador. Para ello, diríjase a Inicio, Programas, Analog Devices, VisualDSP ++ 3.5 for 16-bit *processors*, y seleccione del menú la opción VisualDSP *Configurator* (Ver figura 76).

Figura 76.Figura 1-Laboratorio 5



- En el menú correspondiente a *Platform Templates* resalte la opción ADSP-2181 EZ-KIT Lite vía COM port y con el puntero del ratón sobre la opción *Copy* oprimir el botón izquierdo del mouse, de esta forma se hace visible la ventana *Platform Properties* (ver figura 77), en la cual se establecen las características de la comunicación.

Figura 77. Figura 2-Laboratorio 5



- En la ventana *Platform Properties* seleccione el nombre de la plataforma en el cuadro de texto *Name*, del mismo modo seleccione la opción EZ-KIT 2181 en el menú desplegable *Type*. Para las opciones de *Emulator Settings* seleccione una rata de baudios acorde a las capacidades del computador, de igual forma seleccione el puerto COM por el cual la tarjeta se va a comunicar con el computador y con el puntero del ratón sobre la opción *OK* oprimir el botón izquierdo del mouse. De esta manera se establecen las características de la comunicación entre la tarjeta y el computador, y al iniciar la sesión en modo de comunicación dispositivo/pc, el computador va a reconocer al sistema de desarrollo ADSP-2181.

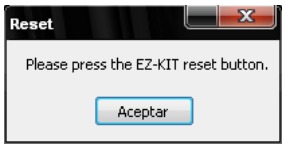
- Inicialice el programa VisualDSP 3.5++.

- En la barra *Menu* seleccionar la opción *Session* y *New Session*.

- En la ventana *New Session* en el menú desplegable *Debug Target* seleccionar la opción EZ-KIT 218x. A continuación en el menú desplegable *Platform* seleccione el nombre de la plataforma que se estableció en los pasos anteriores. En caso de no figurar la plataforma que se creó se debe repetir el procedimiento inicial para poder establecer la comunicación entre el dispositivo y el PC. En la opción *Session name* asigne un nombre a la sesión. Por último, en el cuadro Processor seleccione ADSP-2181 y haga clic en OK.

- Un cuadro de diálogo como el de la figura 78 se presentará en pantalla. Es importante tener en cuenta que este cuadro siempre se hará visible cuando se inicie una sesión de comunicación entre la tarjeta y el PC. Dicho cuadro le indica al usuario que debe hacer clic en el botón de Reset en la plataforma de desarrollo ADSP-2181 EZKit para establecer la comunicación entre las dos partes. Por el momento no haga clic en aceptar, permita que el cuadro de dialogo se mantenga flotante en la pantalla.

Figura 78. Figura 3-Laboratorio 5

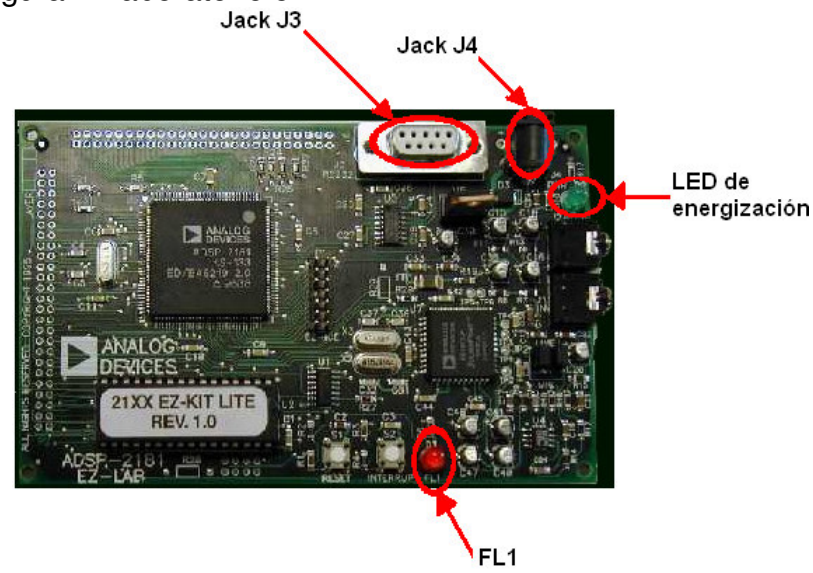


- Conectar un extremo del cable RS-232 a un Puerto COM disponible en el PC y el otro extremo en J3 en la tarjeta de desarrollo ADSP-2181 (ver figura 79).

- Conecte el cable de alimentación provisto por el Kit en una toma de 120 voltios AC y enchufar el otro extremo del cable al conector j4 en la tarjeta de

evaluación (ver figura 79). Compruebe visualmente que todos los LEDs se iluminan brevemente. El LED de comprobación de energía (ver figura 79) permanece encendido y FL1 parpadea continuamente (ver figura 79). Si el LED no se enciende es necesario comprobar la conexión de energía.

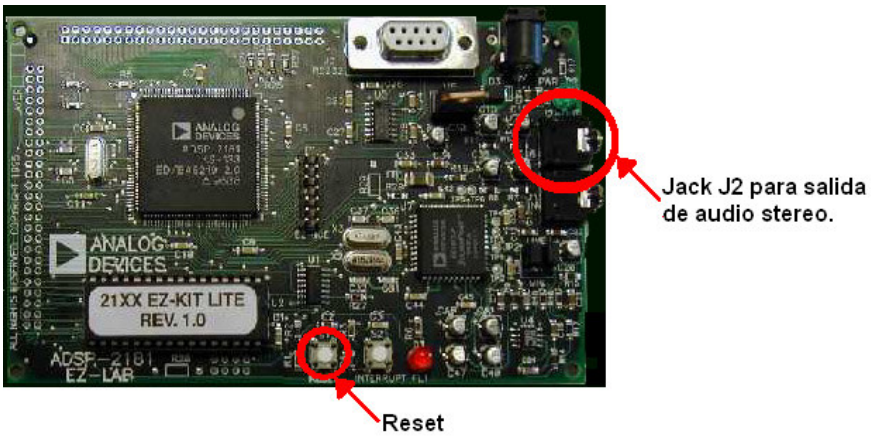
Figura 79. Figura 4-Laboratorio 5



- Conecte un dispositivo electrónico de audio al jack J2 de la tarjeta (ver figura 80).

- Oprima el botón *Reset* de la tarjeta (ver figura 80). Inmediatamente después de pulsar el botón de Reset se debe oprimir el botón izquierdo del mouse con el puntero del ratón ubicado sobre Aceptar del cuadro de dialogo de Reset. Cada vez que la plataforma de desarrollo es energizada, y cada vez que se pulsa el botón de Reset, una melodía podrá ser escuchada a través del dispositivo de audio conectado al jack J2. Esta melodía indica que el dispositivo se ha configurado en sus valores establecidos por defecto. Es necesario que luego de presionar el botón de Reset de la tarjeta y que mientras se escucha la melodía emitida por el sistema de desarrollo se debe oprimir el botón izquierdo del mouse con el puntero del ratón ubicado sobre Aceptar, de otra forma no se establecerá comunicación. En caso de oprimir el botón izquierdo del mouse con el puntero del ratón ubicado sobre Aceptar antes de pulsar el botón de Reset, o luego de que ha terminado de emitirse la melodía de establecimiento de la tarjeta, la comunicación no se realizará correctamente y se tendrá que repetir el procedimiento. Una vez se haya establecido una correcta comunicación entre la tarjeta y el PC el led FL1 dejará de parpadear y se mantendrá encendido de manera constante.

Figura 80. Figura 5-Laboratorio 5



- En la barra Menú, en el menú Project seleccionar la opción Open y ubicar la carpeta con el nombre IRQE que se encuentra en la dirección C:\Archivos de programa\Analog Devices\VisualDSP 3.5 16-Bit\2181_labs\IRQE y seleccione el archivo IRQE.dpj. Con el puntero del ratón sobre la opción *Open* oprimir el botón izquierdo del mouse. En la ventana *Project* se visualizaran las carpetas que conforman el proyecto.

- Depurar el proyecto. Cuando se trabaja en modo de ejecución de proyectos al momento de depurar, el archivo ejecutable es cargado de manera automática en la memoria del DSP, esto quiere decir que al momento de depurar, el software de desarrollo queda en un estado de espera para iniciar la ejecución (*Runtime*) del proyecto dentro del DSP.

- Para correr el proyecto cargado en memoria del DSP diríjase a la barra *Menu*, en el menú *Debug* seleccionar la opción *Run*, de esta manera se inicia la ejecución en tiempo real del proyecto. La forma abreviada de correr el proyecto es presionando la tecla F5 del teclado, o haciendo clic  en el ícono

- En la parte inferior derecha de la ventana principal se podrá observar un texto parpadeante (ver figura 81) que indica que el programa se está ejecutando.

Figura 81. Figura 6-Laboratorio 5

- Presione repetidas veces el botón *Interrrupt* del sistema de desarrollo ADSP-2181 EZKit. ¿Qué ocurre?

Rta. Al momento de pulsar el botón *Interrupt* el estado de la bandera FL1 cambia de alto a bajo dependiendo del estado anterior, indicando que inter*name*nte se generó una interrupción.

- Detenga la ejecución del programa. Para esto diríjase a la barra *Menu*, en el menú *Debug* seleccionar la opción *Halt*. La forma abreviada de detener el proyecto es presionando Shift+F5 en el teclado, o haciendo clic en el ícono 

- Presione la tecla F5 del teclado para correr de nuevo el proyecto. Pulse un par de veces el botón *Interrupt* del sistema de desarrollo ADSP-2181. ¿Qué ocurre con la bandera FL1? ¿Por qué considera que pasa esto?

Rta. La bandera FL1 no cambia de estado, indicando que la interrupción no se ejecuta. Esto ocurre debido a que al correr nuevamente el proyecto él se encuentra desactualizado y al iniciar la ejecución nuevamente, esta comienza justo en la línea en la que terminó al momento de detener el programa, evitando así que la plataforma de desarrollo se configure con los valores necesarios para poder iniciar la ejecución de forma correcta.

- Detenga la ejecución del proyecto, esto ubica al cursor en la última instrucción que se realizó antes de ser detenido. Una vez que el proyecto ha terminado su ejecución, para poder correrlo de nuevo es necesario volver a depurar, o también puede dirigirse a la barra *Menu*, en el menú *Debug* seleccionar la opción *Restart*. La forma abreviada de reiniciar la ejecución es oprimir el botón izquierdo del mouse con el puntero del ratón ubicado sobre el ícono 

Nota: es aconsejable que el usuario depure el proyecto siempre que se realice un cambio en el código fuente, o inclusive cuando se va a correr nuevamente el proyecto. Suele ocurrir que el *Restart* no toma algunos cambios que se realizan al archivo fuente, o en ocasiones inicia el proyecto sin actualizar el puntero en su posición inicial.

- Nuevamente ejecute el programa para comprobar que la interrupción de IRQE (*Interrupt*) se ejecuta correctamente.

- Repita todo el procedimiento para memorizar los pasos a seguir siempre que se desee correr un programa en tiempo real sobe la tarjeta.

Conclusiones.

- La interrupción externa IRQE (*Interrupt*) es una de las opciones disponibles en el sistema de desarrollo ADSP-2181 para que el usuario genere una interrupción en el momento en que él lo desee, con un fin específico que depende principalmente de la funcionalidad del programa que se desarrolla.

- Generar una interrupción externa no implica que el valor de los registros que utilizan a lo largo del programa pierdan su valor. Las opciones de configuración de interrupciones para el sistema de desarrollo ADSP-2181 permiten un almacenamiento temporal de todos los registros, para que al momento de ejecutarse la instrucción RTS, o RTI, los registros carguen los valores que tenían justo antes de darle prioridad a la interrupción generada.

- El anterior procedimiento mostró los pasos a seguir al momento de cargar un programa en memoria DSP, para luego correrlo sobre la tarjeta y poder observar los resultados de las tareas desarrolladas por el DSP.

7.6 LABORATORIO 6. MANEJO DEL DAG – SEÑAL RAMPA

Objetivos.

- Familiarizar al estudiante con el proceso básico a seguir en el ploteo de datos en memoria, utilizando el software de programación VisualDSP++ y la plataforma de desarrollo ADSP-2181 EZKit.

- Utilizar el DAG para generar la secuencia de direcciones necesarias para crear la señal rampa.

- Hacer uso de las ventanas de memoria del software de programación VisualDSP 3.5++.

- Reemplazar el uso de la bandera FL1 por la bandera del conector de expansión FL2.

Lecturas recomendadas.

- Profundizar en el manejo de la interfaz del software de programación VisualDSP 3.5++ para el ploteo de datos en PM y DM.

- Consultar el modo de funcionamiento del DAG.
- Consultar el diagrama esquemático de los conectores de expansión de la plataforma de desarrollo ADSP-2181 EZKit.

Materiales.

- Osciloscopio

Marco Teórico.

Todo dispositivo de la familia ADSP-218x contiene dos generadores de direcciones de datos independientes, lo que quiere decir que tanto la memoria de programa como la de datos pueden ser accedidas simultáneamente. Los DAGs (Generadores de direccionamiento de datos) proveen la capacidad de direccionamiento indirecto, ambos realizan modificaciones automáticas de direcciones, y para buffers circulares, los DAGs pueden realizar modificaciones en el módulo de direcciones.

Los dos DAGs difieren en: DAG1 genera únicamente direcciones de Memoria de Datos (DM), pero provee una capacidad opcional de inversión de bit; DAG2 puede generar tanto direcciones de Memoria de Programa (PM) como de Memoria de Datos, pero no tiene la capacidad de inversión de bit.

Existen tres archivos de registros: el archivo de registro de modificación (M), el archivo de registro índice (I), y el archivo de registro longitud (L). Cada uno de los archivos contiene cuatro registros de 14 bits que pueden ser leídos y escritos por medio del bus DMD (*data memory data*).

La familia de procesadores ADSP-218x permite dos modos de direccionamiento para obtención de la Memoria de Datos: directo e indirecto. El direccionamiento indirecto se logra cargando una dirección en el registro I (*Index*) y especificando uno de los registros M (*modify*) disponibles.

El registro L está provisto para rodear fácilmente el direccionamiento de buffers circulares de datos. Un buffer circular se implementa únicamente cuando un registro L es establecido en un valor que no es cero. Para direccionamiento lineal indirecto el registro L correspondiente al registro I utilizado, debe estar en cero.

Procedimiento.

- Inicializar el software de programación VisualDSP++ en sesión de simulación.

- En la barra *Menu*, en el menú Project seleccionar la opción Open y ubicar la carpeta con el nombre Ramp que se encuentra en la dirección C:\Archivos de programa\Analog Devices\VisualDSP 3.5 16-Bit\2181_labs\Ramp y seleccione el archivo Ramp.dpj. Oprima el botón izquierdo del mouse con el puntero ubicado en *Open*. En la ventana Project se visualizaran las carpetas que conforman el proyecto.

- Del menú *Register* seleccione la opción Program Control, la ventana emergente ubíquela en la parte lateral derecha de la ventana Principal.

- Del menú Memory seleccione la opción Data, la ventana emergente ubíquela junto a la ventana de Program Control. Haga clic en el botón izquierdo de mouse con el puntero ubicado sobre el icono  de la ventana y seleccione la opción data_dm.

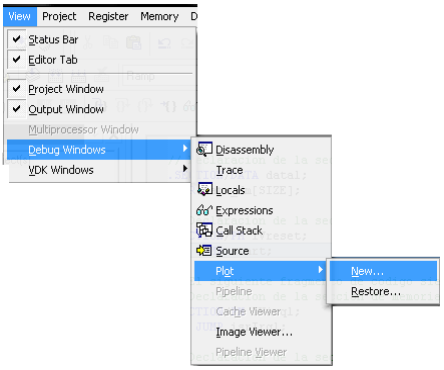
- Del menú *Register* seleccione la opción DAGS, la ventana emergente ubíquela debajo de las ventanas de Program Control y Data.

- Del menú *Register* seleccione la opción FLAG, la ventana emergente ubíquela debajo de la ventana DAG.

- Para observar mejor los resultados cambie el formato de visualización de las ventanas Program Control, Data, FLAG y DAGs, a modo entero signado.

- Del menú View seleccione la opción *Debug* Windows, del menú emergente seleccione la opción *Plot* y *New* (Ver figura 82).

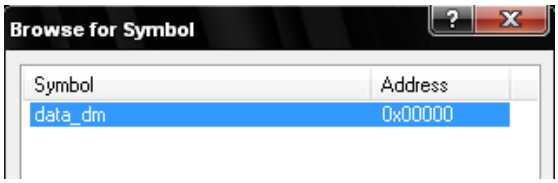
Figura 82. Figura 1-Laboratorio 6



- La ventana emergente *Plot Configuration* permite al usuario establecer los parámetros del ploteo que se desea realizar. Del cuadro de selección

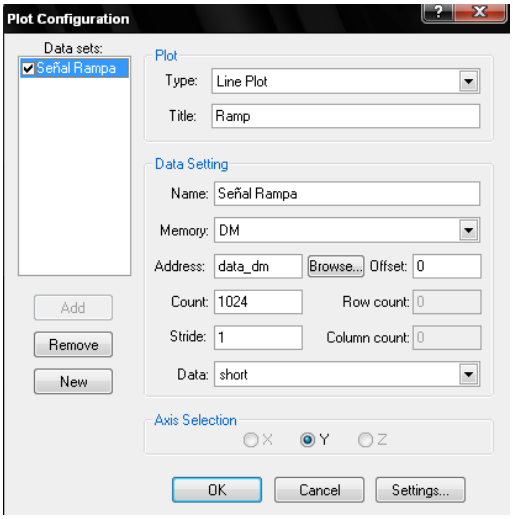
correspondiente a *Type* seleccione la opción *Line Plot*, y asígnele un título al ploteo en el cuadro de texto *title*. En las características de los datos, asígnele un nombre a los datos a graficar en el cuadro de texto *Name*. En el cuadro desplegable *Memory* seleccione la opción DM. Oprima el botón izquierdo del mouse con el puntero ubicado sobre el botón *Browse*, de la ventana emergente seleccione la opción data_dm (ver figura 83) y Ok.

Figura 83. Figura 2-Laboratorio 6



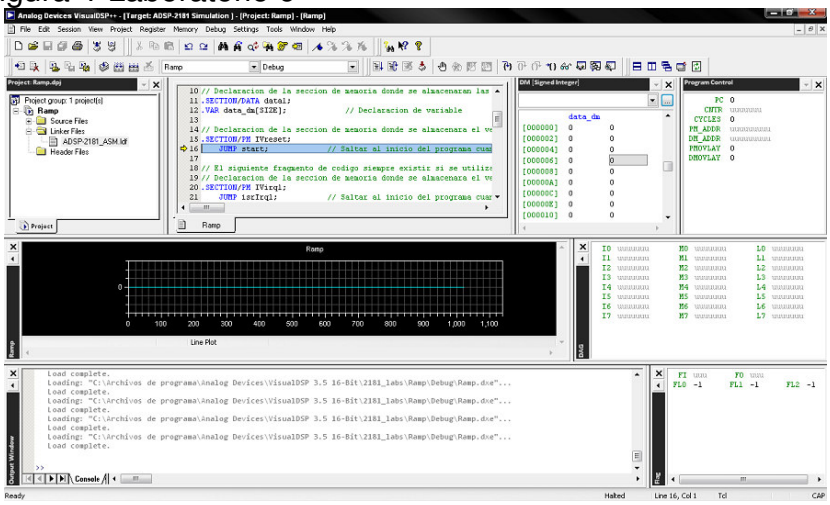
En el cuadro de texto *Count* de la ventana *Plot Configuration* escriba el tamaño del eje X. Para este caso el búfer (SIZE) es de 1024 posiciones. Del menú desplegable Data seleccione la opción short, a continuación haga clic izquierdo con el puntero ubicado sobre el botón *Add* y por último Ok. Al finalizar la configuración la ventana *Plot Configuration* debe tener la apariencia de la figura 99.

Figura 84. Figura 3-Laboratorio 6



- La ventana emergente corresponde a la ventana de ploteo de los datos de memoria antes de correr el programa. Ubique esta ventana junto a la ventana DAG, el aspecto de la ventana principal debe ser el de la figura 85.

Figura 85. Figura 4-Laboratorio 6



- Depure el proyecto.

- Ubique el cursor sobre la ventana de edición (la ventana donde se encuentra el código fuente), y haga clic sobre el botón derecho del mouse para desplegar el menú del cual debe seleccionar la opción *Line Numbers*.

- Recorra paso a paso el siguiente segmento de código.

```
        I0 = data_dm;
        M0 = 1;
        L0 = length(data_dm);
        AX0 = 0x0000;
        CNTR = L0;
        DO BorrarBuffer UNTIL CE;
BorrarBuffer: DM(I0,M0) = AX0;
```

Observe los registros I0, L0 y M0 de la ventana DAG. ¿Qué puede inferir de los valores contenidos en estos registros luego de recorrer el segmento de código? ¿Cuál es la función del DAG? ¿Qué labor desempeña el anterior segmento de código?

Rta. Lo valores contenidos en los registros I0, M0 y L0, corresponden al índice encargado de apuntar a la posición del búfer (I0), el tamaño de incremento del paso (M0=1), y el tamaño total búfer (L0). El DAG ubica el índice (I0) en la primera posición del segmento de memoria asignado para data_dm (DM(I0,M0)), y realiza automáticamente el incremento correspondiente (M0) después que se ha escrito un valor en la posición actual del DAG. Debido a que el valor de AX0 es cero, y que el tamaño del CNTR es igual al tamaño del búfer, el ciclo DO hace que el DAG inicialice todas las 1024 posiciones del búfer data_dm en cero.

- Pulse la tecla F11 del teclado repetidas ocasiones. Observe el valor de I0 en la ventana DAG. Observe el valor de CNTR en la ventana Program Control. ¿Qué quiere decir la variación del valor de estos registros?
Rta. El CNTR indica la cantidad de veces que aún es necesario realizar el bucle generado por el ciclo DO para salir del mismo (UNTIL CE = 0). El incremento del valor de I0 indica la posición del segmento de memoria asignado para data_dm al que se encuentra apuntando el DAG.

- Ubique el cursor sobre la línea de código número 38. En esta posición haga clic derecho en el mouse para desplegar el menú, inmediatamente después seleccione la opción *Run to* Cursor (Ctrl+F10). De esta manera se ejecutan todas las líneas de código presentes entre la línea de código en la que se encontraba y la línea de código hasta la que se corrió el programa (línea 38).

Observe el valor de I0 en la ventana DAG. ¿Por qué el valor de I0 es cero?
Rta. Como se dijo anteriormente al correr el cursor se ejecutó el bucle todas las veces necesarias para salir del ciclo DO, es decir que el valor de I0 en algún momento alcanzó el mismo valor del tamaño del búfer. Una característica del DAG define que cuando el puntero alcanza la última posición del búfer, este se posiciona nuevamente en la primera posición del segmento de memoria, cumpliendo de esta forma las características del búfer circular.

- Ejecute las líneas de código desde el número 39 hasta el 44.

- Ejecute paso a paso el siguiente segmento de código y a medida que lo hace observe con atención las ventanas Data (DM) y Program Control. Observe con atención la ventana de ploteo de la señal de salida.

```
DO Rampa UNTIL CE;
        AR = PASS AF;
        DM(I0,M0) = AR,     AF = AF + 1;
        AR = AX0 - AF;
Rampa:      IF EQ AF = PASS AY0;
```

- Mantenga presionada la tecla F11 del teclado para acelerar el proceso de ejecución paso a paso del segmento de código anterior. No deje de observar la imagen generada en la ventana de ploteo de la señal.

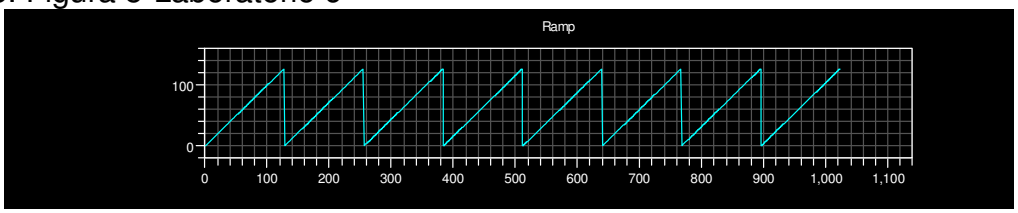
¿Qué relación existe entre los valores presentes en la ventana Data(DM), la ventana DAG y la imagen graficada en la ventana de ploteo?
Rta. Los valores de la ventana DAG corresponden a los valores del generador de direcciones que indican la posición actual de los datos. Los valores generados por el DAG indican la posición en memoria (ventana Data(DM)) en que se está

almacenando el resultado de la operación (incremento para este caso). Por último, los valores de la ventana Data(DM) corresponden a los valores del eje Y de la grafica de la ventana de ploteo.

- Ubique el cursor sobre la línea de código número 51 y corra el programa hasta esta posición de cursor (Run to cursor).

- Almacene la imagen del ploteo. Para esto haga clic derecho del mouse con el puntero ubicado dentro de la ventana de ploteo y del menú seleccione la opción *Export*. De la ventana emergente escoja como destino *ClipBoard* y a continuación *Export*. La imagen quedara en el portapapeles del PC para ser copiada en cualquier archivo de uso común para imágenes o para texto enriquecido tal como *Paint*, Word, Excel, Corel, entre otros. La imagen almacenada debe lucir de la siguiente manera:

Figura 86. Figura 5-Laboratorio 6



- Ubique el cursor en la línea de código 52, a continuación presione las teclas Ctrl+F10 del teclado para ejecutar el programa hasta esta línea de código. Presione la combinación Ctrl+F10 repetidas veces y observe el valor de FL2 en la ventana FLAG. ¿Qué quiere decir el cambio de valor de FL2? ¿Por qué el valor de FL2 es negativo?

Rta. El cambio de estado de FL2 corresponde a que se ha alcanzado el valor final del búfer data_dm (SIZE). La variación de FL2 muestra la frecuencia con la que se repite el ciclo de generación de toda la señal rampa. El valor de FL2 es negativo porque esta es la manera en que los *Flags* funcionan, esto es, si se revisa el esquemático del sistema de desarrollo ADSP-2181 EZKit se observará que la salida física de FL0, FL1, y FL2 es negativa.

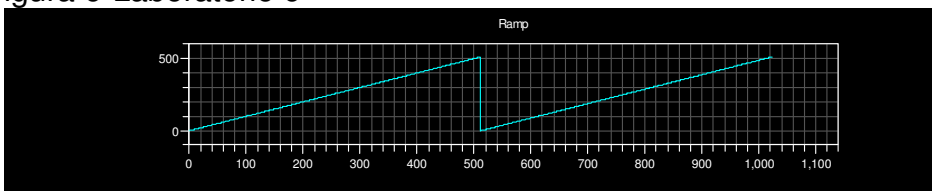
- Presione la tecla F7 del teclado para depurar el proyecto y observe como la ventana de ploteo muestra que la grafica se encuentra con los valores iníciales

- Presione la tecla F5 del teclado para correr el proyecto de manera continua. Luego de unos 10 segundos detenga la ejecución del programa presionando la combinación del teclado Shift+F5. Observe como nuevamente se observa la señal rampa en la ventana de ploteo.

- Asigne un valor de 512 a TOPE y ejecute de nuevo el programa. Observe y almacene la imagen de la ventana de ploteo. ¿Qué cambio se presentó en la grafica? ¿A qué se debió esto?

La imagen generada es la siguiente:

Figura 87. Figura 6-Laboratorio 6



Rta. Ahora la señal rampa posee menos segmentos que antes debido a que el tamaño del búfer para la generación de direcciones (DAG) es mucho mayor, es por esta razón que la señal rampa obtenida, solo posee dos máximos alcanzados correspondientes a la mitad del tamaño total de la señal.

- Asigne opción comentario (//) a la línea de código 45 de la siguiente manera:

```
//AR = PASS AF;
```

- Asigne nuevamente un valor de 128 a TOPE.

- Depure el proyecto.

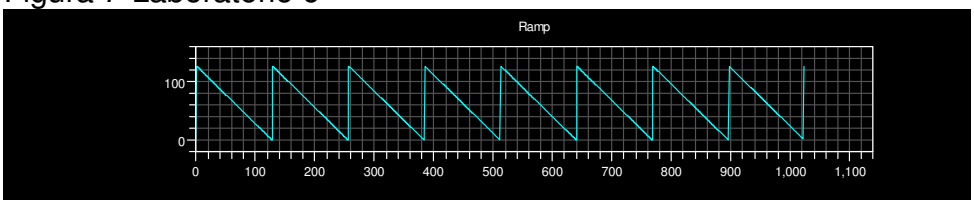
- Ejecute el proyecto.

- Detenga la ejecución del proyecto.

- Observe y almacene la imagen obtenida en la ventana de ploteo ¿Qué forma tiene la señal obtenida? ¿Explique porque razón la señal ahora tiene esa forma?

La imagen generada es la siguiente:

Figura 88. Figura 7-Laboratorio 6



Rta. La señal obtenida corresponde a una señal rampa, solo que la nueva señal inicia en su máximo valor y a medida que avanza sobre el eje X esta va decreciendo del mismo modo en que de TOPE se acerca a cero. Al ubicar la opción comentario (//), se logra que el primer valor ubicado en la memoria de datos corresponda al máximo valor que puede tener la señal (TOPE), y a medida que el DAG genera la siguiente dirección para la memoria de datos, esta se va llenando con el decremento del valor de TOPE.

- Retire la opción comentario (//) de la línea de código 54.

- Conecte el sistema de desarrollo ADSP-2181 EZKit y cambie la sesión a modo de ejecución de proyectos en tiempo real.

- Abra nuevamente el proyecto RAMP.dpj y depúrelo.

- Ubique todas las ventanas utilizadas en modo de simulación para comprobar que en ejecución en tiempo real (sobre la tarjeta) ocurre lo mismo que en sesión de simulación.

- Corra el programa.

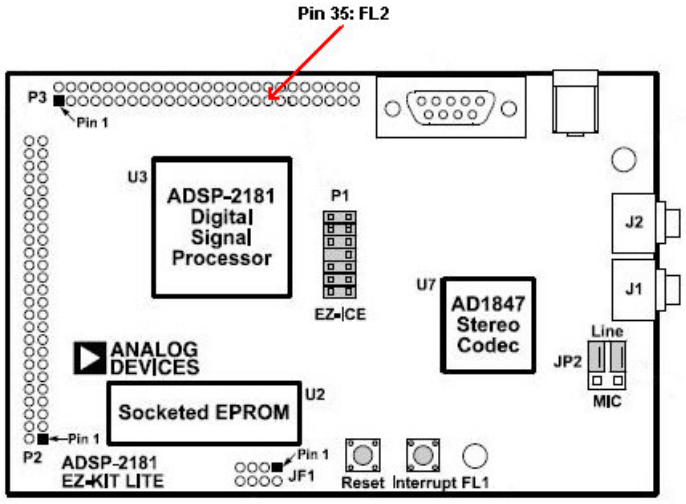
- Detenga la ejecución del programa y observe la gráfica de la ventana de ploteo. Note que la gráfica se está generando con los valores almacenados en la memoria del DSP.

- Asegure que el valor de SIZE se encuentra en 1024.

- Depure y ejecute nuevamente el programa.

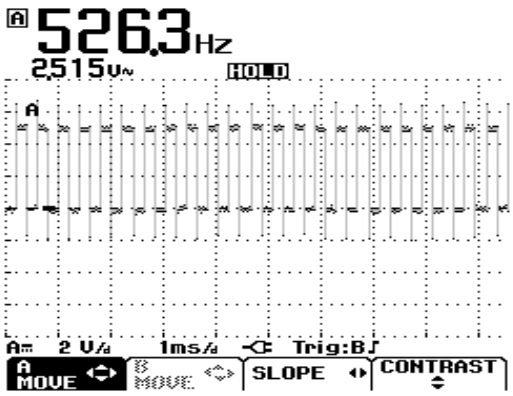
- Ubique una punta del osciloscopio en el pin 35 del puerto de expansión P3 correspondiente al pin externo para FL2 (ver figura 104). Dibuje la forma de la señal que se observa en el osciloscopio y calcule el valor de la frecuencia con la que se genera la señal de FL2.

Figura 89. Figura 8-Laboratorio 6



La señal obtenida es la que se muestra a continuación:

Figura 90. Figura 9-Laboratorio 6



La frecuencia de la señal es de 525.2 Hz.

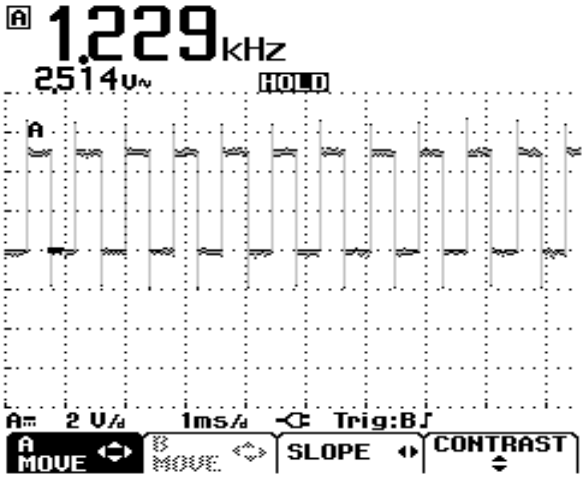
- Detenga la ejecución del programa.

- Cambie el valor de SIZE al doble del que tenia (2048). Asegure que en las características de la ventana de ploteo el cuadro de texto correspondiente a *Count* es del mismo valor que el de SIZE, de otra forma no se podrá observar toda la señal

- Una vez más ejecute el programa y ubique una punta del osciloscopio en el pin 35 del puerto de expansión P3. Dibuje la forma de la señal obtenida en el osciloscopio y calcule la frecuencia de la señal.

La señal obtenida es la que se muestra a continuación:

Figura 91. Figura 10-Laboratorio 6



La frecuencia de la señal es de 1.23 KHz.

- Detenga la ejecución del programa.

- Observe el dibujo de las graficas obtenidas por el osciloscopio. ¿Por qué la frecuencia en las dos señales varía y a que se debe esto?
Rta. La frecuencia de las señales cambia en función del tamaño de toda la señal rampa. Para uno de los casos el tamaño era de 1024 y para el otro caso el tamaño fue de 2048, de esta forma la frecuencia para una es el doble (o la mitad) de la otra. Debido a que existe el doble del espacio de memoria en un caso que en otro el tiempo de llenado de la memoria de datos corresponde al doble, y así mismo, la frecuencia con la que FL2 cambia de estado para la rampa de tamaño 2048, es el doble de la frecuencia de cambio para la rampa de tamaño 1024.

Conclusiones.

- Las herramientas del software de programación VisualDSP++ permiten que el usuario grafique distintas clases de datos utilizados en los procesos que se realizan a lo largo de los proyectos para la plataforma de desarrollo ADSP-2181 EZKit, para poder observar de forma visual el resultado el procesamiento en el DSP.

- Plotear una señal no es una característica propia del simulador, también es posible acceder a las distintas memorias del DSP para poder graficar el resultado del procesamiento luego de que este se ha ejecutado en tiempo real sobre la tarjeta (*On-Board*).

- Las rampas están fuertemente asociadas con el procesamiento digital de señales. Este laboratorio proporcionó una rutina sencilla generadora de señal llamada Rampa, para que el usuario aprendiera el uso de los DAGs con el fin de generar una secuencia de dirección con un fin específico.

- No es conveniente que en una sesión de ejecución en tiempo real se realice la ejecución del programa paso a paso (*Stepping*), ya que pasar de una instrucción a la otra toma un considerable tiempo de espera, además dependiendo del PC y de las características de cómo se establezca la comunicación dispositivo/PC, existe la posibilidad de que el sistema presente errores y fallos.

- En cualquier sesión el tamaño del búfer está limitado por el tamaño de la memoria del DSP. Cuando se trabaja en sesión de simulación el tamaño del búfer no presenta ninguna complicación respecto de tiempo de ejecución porque en simulación los tiempos dependen del mismo PC sobre el que se corre el software, mientras que en sesión de ejecución en tiempo real el tamaño del buffer (directamente relacionado con el tamaño de la memoria del DSP) implica demoras en la ejecución del programa dependiendo de qué tan grade sea el tamaño del búfer que se utilice y dependiendo de la conexión dispositivo/PC.

7.7 LABORATORIO 7. EJEMPLO DE DISEÑO: FILTRO FIR

7.7.1 Parte 1. Establecimiento de la plantilla de trabajo TalkThru.

Objetivos.

- Familiarizar al estudiante con la plantilla de trabajo TalkThru.
- Visualizar y analizar el procedimiento necesario para inicializar el CODEC AD1847 y el procesador ADSP-2181.

Lecturas recomendadas.

- Consultar el *datasheet* del CODEC AD1847.

- Consultar el *datasheet* del procesador ADSP-2181.

Materiales.

- Osciloscopio.
- Generador de señales.
- Dispositivo de audio con salida estéreo (mp3, mp4, ipod, etc).
- Audífonos o parlantes.
- Dos (2) cables estéreo-estéreo.
- Conector hembra estéreo.

Marco teórico.

El primer paso en el procesamiento digital de señales es el muestreo de señales análogas o conversión A/D. Similarmente, el paso final en la producción de la señal de salida procesada, requiere reconstrucción o conversión D/A.

La plantilla de trabajo TalkThru toma las muestras obtenidas desde los canales izquierdo y derecho del puerto de entrada y las dirige al puerto de salida a un predeterminado rango de muestreo. El programa comienza con la asignación de los comandos iniciales por medio de la directiva #*include* <def2181.h>, esta asignación contiene todos los comandos encargados de la configuración del CODEC. En las líneas siguientes se definen los buffers en DM, tres de los cuales son circulares. El buffer rx_buf y el buffer tx_buf son de una longitud de 3 y ellos serán utilizados para el auto almacenamiento, mientras que el buffer circular *init_cmds* de longitud 13 será utilizado para inicializar los registros índices del AD1847. El buffer escalar *stat_flag* es el encargado de chequear el estado de la inicialización del AD1847. Luego de haber terminado la inicialización del CODEC es necesario realizar la inicialización del procesador. El programa consta de dos sub secciones principales. Primero es necesario establecer los registros de direccionamiento de los buffer circulares para su adecuada operación. Segundo es necesario establecer la memoria de datos mapeada por los registro de control del ADSP-2181. De acuerdo con el manual de usuario de la familia 21xx, estos registros tienen valores por defecto al inicio y solo unos pocos registros requieren ser establecidos. Sin embargo, es una buena práctica de programación inicializar todos los registros de control y así evitar cualquier problema en el futuro. El primero de estos registros es el *System Control Register* ubicado en DM en la posición 0x3FFF. Al inicializarse el SPORT0 es habilitado. El registro *Wait State* Control en DM es resetado a cero, además todos los estados de espera de DM son establecidos en cero. Los registros restantes son para la inicialización del SPORT0. Desde que SPORT0 se encuentre habilitado, su registro de control localizado en 0x3FF6 es un registro importante, ya que en sus bits 0 al 3, 9, 10 y

15 son establecidos en uno, seleccionando así la operación de multicanal (para auto almacenamiento), formato de datos lineal (o sin comprensión), y longitud de palabra de 16 bits. Los bits 0 al 3 son el parámetro SLEN que denota el tamaño de la palabra serial, este tamaño esta dado por la función SLEN + 1. Por esta razón, SLEN está establecido en 15. La mayoría de los valores en este registro serán los mismos para muchos programas. Desde que la característica de multicanal de SPORT0 se encuentre habilitada los cuatro registros, desde la localidad 0x3FF7 a 0x3FFA, establecen los parámetros requeridos para la transmisión y recepción de palabras. Para auto almacenamiento existe el registro de control ubicado en 0x3FF3 en DM, el cual habilita el almacenamiento para recepción y transmisión. Finalmente dos registros no mapeados en memoria son establecidos. El registro ICNTL de 5 bits de longitud encargado de controlar la sensibilidad de las interrupciones, y el registro MSTAT encargado de habilitar el modo *Go Mode*.

Finalizada la inicialización del procesador, se dispara la interrupción de transmisión. Primero, *stat_flag* es establecido en uno para señalizar que la inicialización se encuentra en proceso, entonces la interrupción de transmisión es habilitada utilizando el registro IMASK y el valor de la parte superior del buffer *init_cmds* es transmitido a tx0 activando así la interrupción. Esto transfiere la ejecución al servicio de interrupción. Como *stat_flag* no es cero, el control de nuevo se transfiere a la etiqueta *next_cmd* donde la actual transferencia tiene lugar. Como hay trece valores para transferir, el procesador se mantiene en el lazo *check_init* utilizando nuevamente *stat_flag*. Como parte de la inicialización del códec, su característica de auto calibración es habilitada, y esta a su vez requiere ciclos de procesador. Esto es asegurado por dos lazos de seguridad el lazo check_aci1 para auto calibración en proceso, y el lazo check_aci2 para el reset de la auto calibración. Estos lazos logran que el códec sea adecuadamente inicializado, incluyendo silencio en la salida de los DAGs (canales izquierdo y derecho), para que ninguna señal de audio extraña sea generada durante este periodo. Este silencio es finalmente retirado por medio de la re inicialización de los registros índices 6 y 7.

Los registros de control IFC e IMASK, los cuales no son mapeados en memoria, son también importantes en todo el programa. El registro IFC limpia cualquier interrupción pendiente. El registro IMASK es de 10 bits de longitud, y su valor se establece para que la recepción de interrupción de SPORT0 sea habilitada. Es una buena práctica de programación el habilitar esta interrupción luego de inicializar todos los demás parámetros de los puertos seriales, y así evitar comportamientos erráticos.

A lo largo del programa se gastan muchas líneas en la configuración del procesador y del CODEC. Parece ser demasiado para un simple programa de conversión A/D – D/A, sin embargo son cubiertos todos los registros de control necesarios en ambos dispositivos así será posible programarlos para cualquier uso futuro. Luego de los anteriores procedimientos el programa se encuentra listo

para procesar las muestras de entrada. Los lazos del procesador sobre la instrucción IDLE utiliza la etiqueta *samples* hasta que la interrupción desde el SPORT0 es recibida. El programa es por esta razón dirigido por interrupción. Cuando la interrupción ocurre, el control del programa cambia a la rutina de servicio, saltando a la localidad *input_samples*. Toda futura actividad tiene lugar en la rutina de servicio de interrupción. Las muestras de entrada no se encuentran en RX0 debido al auto almacenamiento sino en rx_buf (canal izquierdo en la segunda posición y canal derecho en la tercera). Estos valores son entonces cargados en MR0 y MR1 respectivamente. Como este es un programa sencillo de TalkThru, estos valores son copiados a las debidas localidades del buffer tx_buff. Como la transmisión y recepción son sincronizadas en auto almacenamiento, los valores de entrada serán llevados por el códec cuando la interrupción de recepción ocurre.

Procedimiento.

- Inicializar el software de programación VisualDSP++ en sesión de simulación.

- En la barra *Menu*, en el menú Project seleccionar la opción Open y ubicar la carpeta con el nombre TalkThru que se encuentra en la dirección C:\Archivos de programa\Analog Devices\VisualDSP 3.5 16-Bit\2181_labs\TalkThru y seleccione el archivo TalkThru.dpj. Dar clic sobre el botón izquierdo del mouse con el puntero sobre la opción *Open*. En la ventana Project se visualizaran las carpetas que conforman el proyecto.

- Depure el proyecto.

- Observe la sección .SECTION/DM Data1 y explique el funcionamiento de los buffers definidos en ella.

Rta. Los buffers definidos en esta sección corresponden al buffer *state_flag* encargado de indicar el estado de transmisión de comando de inicialización del códec y del procesador, el buffer rx_buff que contiene el valor de control de recepción de palabras y las muestras recibidas y auto almacenadas, el buffer tx_buff que contiene lo valores a transmitir por el puerto de salida del códec, y por último el buffer *init_cmds* que contiene los valores a ser cargados en los parámetros del códec para su respectiva inicialización.

- Observe el siguiente segmento de código. ¿A que corresponde?

```
AX0 = 0x287;
DM (Sport0_Autobuf_Ctrl)  = AX0;
AX0 = 0;
DM (Sport0_Rfsdiv) = AX0;
```

```
AX0 = 0;
DM (Sport0_Sclkdiv) = AX0;
AX0 =0x860F;
DM (Sport0_Ctrl_Reg) = AX0;
AX0 =0x7;
DM (Sport0_Tx_Words0) = AX0;
AX0 =0x7;
DM (Sport0_Tx_Words1) = AX0;
AX0 =0x7;
DM (Sport0_Rx_Words0) = AX0;
AX0 =0x7;
DM (Sport0_Rx_Words1) = AX0;
```

Rta. El anterior código corresponde a la configuración de los registros mapeados en memoria correspondientes al modo de funcionamiento del Sport0.

- ¿Qué función desempeñan las etiquetas *check_init*, check_aci1 y check_aci2?

Rta. En la ejecución del programa las etiquetas mencionadas son las encargadas de realizar la finalización de la configuración inicial del códec y del procesador. *Check_init* se encarga de realizar la comprobación del envío de todos los comandos al códec, check_aci1 y check_aci2 realizan las tareas de auto calibración y comprobación de auto calibración.

- Observe el siguiente segmento de código. ¿Qué función desempeña la subrutina *next_cmd*?

```
next_cmd:
    ENA SEC_REG;
    AX0 = DM(I3, M1);
    DM (tx_buf) = AX0;
    AX0 = I3;
    AY0 = init_cmds;
    AR = AX0 - AY0;
    IF GT RTI;
    AX0 = 0xAF00;
    DM (tx_buf) = AX0;
    AX0 = 0;
    DM (stat_flag) = AX0;
    RTI;
```

Rta. La primera línea de código de la subrutina se encarga de habilitar el uso de registros secundarios, y así permitir que el valor de los registros que se utilizan en la interrupción, sean alterados durante la ejecución de la misma. En las líneas de

código siguientes de la subrutina *next_cmd* se realiza el enganche de la siguiente palabra de control y se ubica en la posición de transmisión 0 del buffer tx_buf. Luego se hace RTI si existen más palabras de control esperando a ser transmitidas, de lo contrario se establece la bandera para indicar que se ha realizado la transmisión y se remueve MCE si la inicialización también se ha realizado completamente. Las últimas líneas de código corresponden al reseteo de la bandera de estado y a la instrucción de retorno de la interrupción.

- Observe el siguiente segmento de código. ¿Qué función desempeña la subrutina *input_samples*?

```
input_samples:
    ENA SEC_REG;
    MR0 = DM (rx_buf + 1);
    MR1 = DM (rx_buf + 2);
    DM (tx_buf + 1) = MR0;
    DM (tx_buf + 2) = MR1;
    RTI;
```

Rta. Las dos primeras líneas de código se encargan de tomar las muestras almacenadas en las posiciones 1 y 2 del buffer encargado de la recepción de datos rx_buf, y moverlas a los registros MR0 y MR1 respectivamente. Las dos siguientes líneas de código toman los valores de MR0 y MR1, para ser llevados a las posiciones 1 y 2 del buffer tx_buf, el cual es el encargado de realizar la transmisión de datos. La última línea de código corresponde al retorno de la interrupción.

- Conecte la salida estéreo del dispositivo de audio al jack de entrada de la tarjeta ADSP-2181 EZKit, así mismo conecte los audífonos o parlantes al jack de salida de la tarjeta. Las conexiones deben verse del mismo modo en que se muestra en la siguiente figura.

Figura 92. Figura 1-Laboratorio 7, parte 1

- Conecte el sistema de desarrollo ADSP-2181 EZKit y cambie la sesión a modo de ejecución de proyectos en tiempo real. Abra y depure nuevamente el proyecto TalkThru.dpj.

- Encienda el dispositivo de audio y cerciórese que esté emitiendo una señal de audio.

- Corra el programa. ¿Qué sucede? ¿Qué diferencia existe entre la señal de entrada y la señal da salida?

Rta. La señal de entrada de audio es escuchada a través del dispositivo de salida conectado a la tarjeta, comprobando así la recepción y transmisión de datos a través del códec AD1847. Existe una evidente disminución en el volumen de la señal de entrada.

- Ubique el cursor sobre la ventana de edición (la ventana donde se encuentra el código fuente), y hacer clic sobre el botón derecho del mouse para desplegar el menú del cual debe seleccionar la opción *Line Numbers*.

- Diríjase a la línea de código 95. Esta línea de código corresponde a uno de los segmentos del buffer *init_cmds* encargado de la configuración del códec. Esta posición específicamente se encarga de seleccionar el reloj que será utilizado para la frecuencia de muestreo, selecciona el valor de la frecuencia de muestreo y por último selecciona el formato de los datos muestreados. La línea de código 95 se muestra a continuación.

```
0xc85c,        /* CLOR set, MCE set, Index reg address=8,
data=0x5c
```

Varíe la línea de código de la siguiente manera.

```
0xc85f,        /* CLOR set, MCE set, Index reg address=8,
data=0x5c
```

Depure el programa nuevamente y ejecútelo. ¿Qué diferencia existe entre la señal que se escucha actualmente y la anterior? ¿Al cambiar la línea de código 95 que se está afectando en la configuración del códec?

Rta. La señal que se escucha actualmente tiene menos nitidez que la anterior, además presenta tonos graves lo que la hace parecer más distante. Al variar el valor de la línea de código 95 se está afectando el valor de la frecuencia con que se muestrea la señal de entrada del códec, la frecuencia de muestreo para este caso es de 6.61 kHz la cual es la frecuencia más baja a la que se puede muestrear una señal con el códec AD1847.

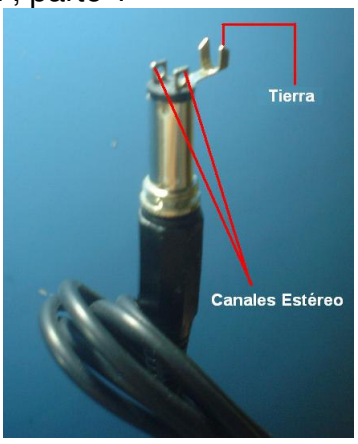
- Varíe la línea de código 95 tal y como se encontraba inicialmente, estableciendo de esta forma una frecuencia de 48 kHz la cual es la mayor frecuencia a la que se puede muestrear una señal con el códec AD1847.

- Desconecte el dispositivo de audio conectado al jack de entrada y los parlantes o audífonos conectados al jack de salida de la tarjeta.

- Conecte uno de los cables estéreo al jack de entrada de la tarjeta, y el otro cable estéreo al jack de salida.

- Destape los conectores hembra estéreo e insértelos a la punta libre de los cables estéreo, la conexión se muestra en la siguiente figura.

Figura 93. Figura 2-Laboratorio 7, parte 1



- Al finalizar el montaje de las conexiones el circuito debe verse de la siguiente manera.

Figura 94. Figura 3-Laboratorio 7, parte 1

- Conecte el generador de señales a ambos canales en 2 (ver figura anterior). Cerciórese que la señal emitida por el generador sea senoidal y de frecuencia 13 kHz, y que su voltaje pico - pico no sea mayor de 3 voltios rms. También conecte una punta del osciloscopio en este punto para tener la señal de referencia en uno de los dos canales.

- Conecte la otra punta del osciloscopio en 1 (ver figura anterior). La punta puede ser conectada en cualquiera de los dos canales.

- Depure y ejecute nuevamente el programa. Observe la señal graficada por el osciloscopio.

- Dibuje las señales del osciloscopio. ¿Qué diferencia existe entre la señal de entrada y la señal de salida? ¿A qué se debe esto? ¿Cuál es la frecuencia de ambas señales? ¿Por qué cree que la señal de salida se encuentra desplazada en comparación con la señal de entrada?
Rta. La diferencia que existe entre las dos señales es que la señal de salida presenta una atenuación de voltaje en comparación con la señal de entrada, esto se debe a que por motivos propios de los dispositivos (códec y procesador) siempre va a haber una atenuación en la señal de salida con respecto a la señal de entrada. Si se desea que la señal de salida sea igual a la señal de entrada es necesario implementar una etapa de amplificación a la señal de salida. La frecuencia para ambas señales sigue siendo la misma de 13 kHz, el muestreo afecta la amplitud de la señal pero no su frecuencia. Las señales obtenidas por el osciloscopio deben lucir como la figura 110. Observe que los parámetros de amplitud de señal del canal B están en 100mV/d, mientras que para el canal A son de 500mV/d. La señal de salida se encuentra desplazada en comparación con la señal de entrada debido a que todo el proceso de conversión A/D y D/A, hace que la señal presente una demora en relación con la señal de entrada. Se puede decir que el desplazamiento de la señal corresponde al tiempo de procesamiento.

- Detenga la ejecución del programa.

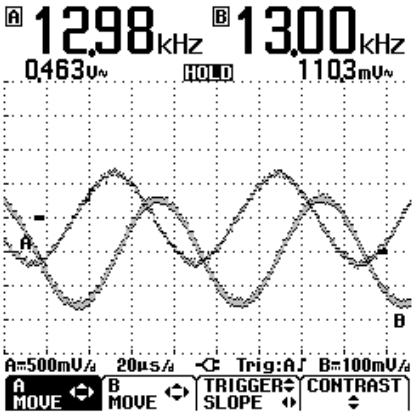
- La señal de salida de la figura 95 corresponde a una señal para la cual se ha establecido que la atenuación sea mínima. Ubique el cursor en la línea de código 83. El código es el siguiente.

```
0xc680,     /* CLOR set, MCE set, Index reg address=6,
data=0x80
```

Reemplace el anterior código por el siguiente

```
0xc68f,     /* CLOR set, MCE set, Index reg address=6,
data=0x80
```

Figura 95. Figura 4-Laboratorio 7, parte 1



- Ubique el cursor en la línea de código 89. El código es el siguiente.

  0xc780, /* CLOR set, MCE set, Index reg address=7, data=0x80

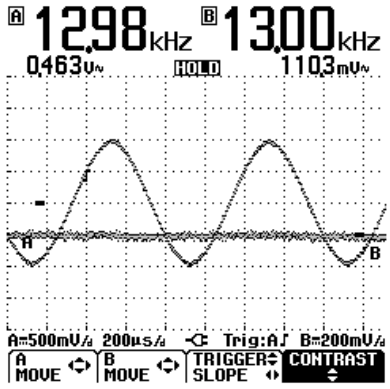
- Reemplace el anterior código por el siguiente

  0xc78f, /* CLOR set, MCE set, Index reg address=6, data=0x80

- Depure y ejecute nuevamente el programa. Observe y grafique las señales obtenidas en el osciloscopio. ¿A qué se debe que la señal de salida tenga ese aspecto?

Rta. La forma de la señal de salida es la de una señal casi igual a cero debido a que se estableció, que tanto canal izquierdo como canal derecho tuvieran una atenuación tal que la atenuación fuese tan grande como para hacer que la señal de salida fuese casi cero. La forma de las señales es la misma que la de la siguiente figura.

Figura 96. Figura 5-Laboratorio 7, parte 1



Conclusiones.

- La plantilla de trabajo TalkThru corresponde a la base de la mayoría de los programas que se elaboran en VisualDSP para la tarjeta ADSP-2181 EZKit. El TalkThru es utilizado para realizar el muestreo de la señal de entrada y a su vez se encarga de direccionar los datos de salida hacia el puerto de salida del códec.

- Es importante resaltar que el TalkThru se encarga del muestreo de la señal de entrada. Los datos obtenidos por dicho muestreo pueden ser utilizados para diferentes aplicaciones, es decir que la plantilla de trabajo solo se encarga de una parte del trabajo en el procesamiento digital de señales, la parte inicial de todo el proceso, el muestreo. De ahí su gran importancia y común uso.

- La correcta configuración e inicialización de los distintos dispositivos del sistema de desarrollo ADSP-2181 EZKit, son los causales del correcto funcionamiento del kit. Por esta razón se deben tener en cuenta todas las variables al momento de programar, desde la señal de muestreo con la que se va a configurar el códec, hasta la habilitación de las distintas interrupciones que se utilicen en el programa, sin olvidar que todos los parámetros de programación juegan un papel importante en la ejecución del proyecto.

- Toda señal procesada por el sistema de desarrollo tendrá inevitablemente una atenuación propia, debida a las características del sistema. Es por esta razón que en el proceso de reconstrucción de la señal de salida es posible y permitido colocar una etapa de amplificación para que dicha señal pueda ser la entrada correcta de otro sistema de procesamiento.

- Dado que el TalkThru es la plantilla de trabajo común a la mayoría de los programas de procesamiento digital de señales, es correcto mantener la misma plantilla, es decir que el código utilizado en el anterior laboratorio debe ser utilizado en todos los programas variando únicamente los factores necesarios para conseguir el correcto procesamiento de la señal.

- Para un mayor entendimiento de la configuración e inicialización del códec AD1847 y del procesador ADSP-2181, es conveniente leer los *datasheets* de ambos dispositivos, ya que el ellos se encuentran tanto los registros a configurar como el modo en que se debe programar para que la inicialización sea la adecuada.

7.7.2 Parte 2. Diseño y simulación del filtro FIR

Objetivos.

- Utilizar la herramienta de trabajo *Fdatool* de MatLab para el diseño del filtro FIR.

- Utilizar el entorno de trabajo de MatLab para comprobar el funcionamiento del filtro FIR.

- Hacer uso del entorno de simulación del VisualDSP++ para simular el filtro FIR.

Lecturas recomendadas.

- Profundizar en la teoría de filtros digitales.
- Profundizar en el diseño de filtros FIR.
- Consultar la herramienta de trabajo *fdatool* de MatLab.

Marco teórico.

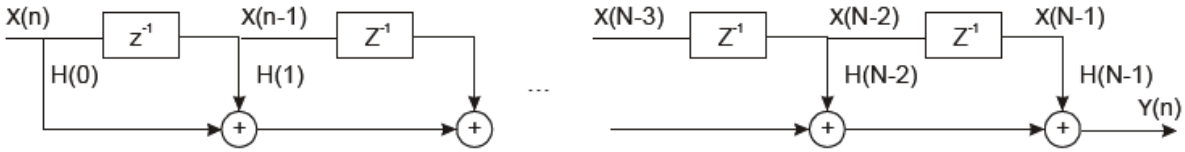
Los filtros FIR (*Finite Impulse Response*) se caracterizan por ser sistemas causales, lineales, que como lo dice su nombre, tienen una respuesta infinita al impulso, y es posible hacer que su respuesta sea de fase lineal. Un filtro FIR es un sistema lineal, discreto, e invariante en el tiempo, cuya salida es basada en la sumatoria ponderada de un número finito de entradas anteriores. Los filtros FIR, a diferencia de los filtros IIR (*Infinite Impulse Response*), son no recursivos y no requieren lazos de realimentación en su computación. Esta propiedad permite un simple análisis e implementación en microprocesadores tales como ADSP-2181. La ecuación en diferencias del filtro FIR es:

$$y(n) = \sum_{k=0}^{N-1} h(k)x(n-k)$$

Donde:

- N: Número de coeficientes del filtro. Determina el orden del mismo.
- h(k): Coeficientes del filtro.
- x(i): Señal a filtrar.

Esta ecuación corresponde a la convolución de la respuesta al impulso, es decir, los coeficientes del filtro, con la señal de entrada. Dicha ecuación en diferencias lleva fácilmente a la implementación con la estructura en forma directa. La representación grafica de un filtro FIR es mostrada en el siguiente diagrama de bloques.



La función de transferencia del filtro FIR es un polinomio de la forma:

$$H(z) = \sum_{k=0}^{N-1} h(k) z^{-1}$$

Entre las desventajas del filtro FIR se encuentra que para lograr determinada respuesta en frecuencia requieren de un número relativamente elevado de coeficientes, lo cual induce un considerable retardo de grupo, que si bien es constante en todo el rango de frecuencias, puede ser un inconveniente para determinado tipo de aplicaciones. Entre sus ventajas, especialmente para la implementación en plataformas DSP de punto fijo, está el hecho de que por ser posible implementar filtro FIR con estructuras no recursivas (forma directa, forma directa transpuesta, estructuras en cascada), con dichas estructuras no se presentaran efectos de ciclos limite, que consisten en una respuesta oscilatoria remanente aun en ausencia de señal de entrada. Sin embargo, debido a errores de cuantización ocasionados por la longitud finita de registros y a los errores de redondeo en operaciones internas (multiplicaciones y suma), es posible que la respuesta en frecuencia obtenida difiera de la frecuencia deseada con un diseño en precisión infinita.

Procedimiento

- El primer paso a realizar en esta práctica es decidir los parámetros del filtro FIR que se va a diseñar. El filtro será pasa banda de orden 53. Los parámetros de diseño son:

  - Frecuencia de muestreo (Fs) = 8000 Hz
  - Frecuencia de paso, inferior (Fp1) = 1000 Hz
  - Frecuencia de corte, inferior(Fs1) = 500 Hz
  - Frecuencia de paso, superior (Fp2) = 15000 Hz
  - Frecuencia de corte, superior (Fs2) = 2000 Hz

- Rizado máximo en la frecuencia de paso 1 dB
- Rechazo mínimo en la bandas de detención = 80 dB

- Inicializar MatLab.

- En el *Workspace* de MatLab escriba el siguiente comando y presione Enter en su teclado.

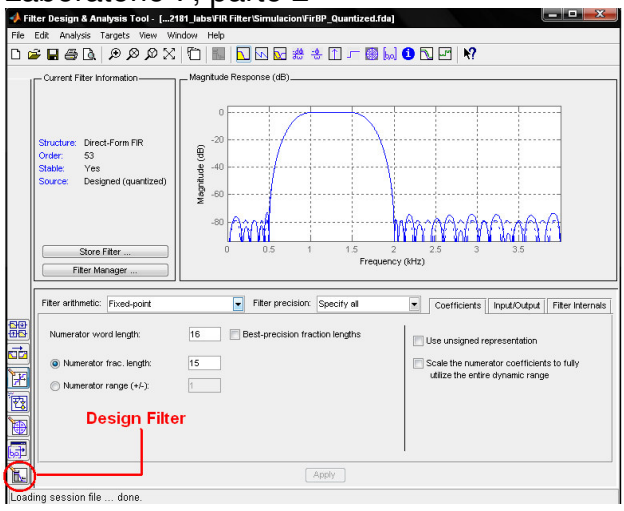
fdatool

- Se abrirá entonces la ventana que corresponde al toolbox de *fdatool* "*Filter Design & Analisis Tool*", la cual permitirá diseñar el filtro del modo en que se desea, y al mismo tiempo se encargará de generar todos los coeficientes del filtro, necesarios para poder realizar la convolución.

- Haga clic izquierdo con el puntero del mouse sobre la opción *File*, y a continuación seleccione la opción *Open Session*. Ubique la carpeta con el nombre FIR *Filter* que se encuentra en la dirección C:\Archivos de programa\Analog Devices\VisualDSP 3.5 16-Bit\2181_labs\FIR *Filter*. Dentro de esta carpeta se encuentra otra carpeta con el nombre de "Simulación". A continuación seleccione el archivo FirBP_Quantized.fda. Dar clic en Open.

- La forma de la ventana se muestra en la figura 97. Note la casilla resaltada por el círculo rojo, haga clic en ella.

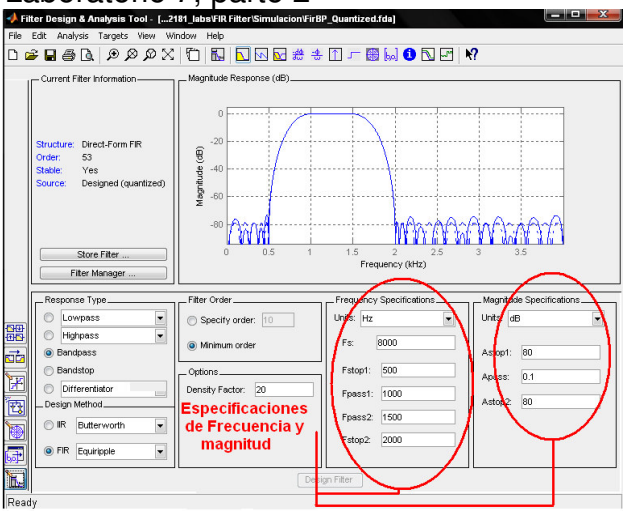
Figura 97. Figura 1-Laboratorio 7, parte 2



- La forma de la nueva ventana se muestra en la imagen 98. Note las casillas señaladas en rojo. Estas corresponden a los parámetros del filtro. Dichos parámetros ya están establecidos para un filtro de orden 53 que cumple con todos los requisitos establecidos para el filtro a trabajar.

Nota: es labor del estudiante profundizar en el modo de funcionamiento de la herramienta *fdatool*.

Figura 98. Figura 2-Laboratorio 7, parte 2



- Lo que es más importante de la herramienta *fdatools* es la capacidad para exportar los coeficientes del filtro al *Workspace*. Para realizar esta tarea haga clic izquierdo con el puntero del mouse sobre la opción *File,* y a continuación seleccione la opción *Export.* La ventana emergente se muestra en la imagen 99.

Figura 99. Figura 3-Laboratorio 7, parte 2

- Para exportar los coeficientes seleccione la opción *Workspace* en el cuadro de texto *Export To*. Seleccione la opción *Coefficients* en el cuadro de texto *Export As*. En el cuadro de texto Variable *Name*s en la opción *Numerator* escriba *Num* (este es el nombre que trae por defecto). Por último seleccione la casilla *Overwrite Variables*. Para finalizar haga clic sobre el botón izquierdo del mouse con el puntero del mouse sobre el botón *Export*.

- Ubíquese en el *Workspace* de MatLab.

- Haga clic izquierdo con el puntero del mouse sobre la opción File, y a continuación seleccione la opción Open. Ubique la carpeta con el nombre FIR *Filter* que se encuentra en la dirección C:\Archivos de programa\Analog Devices\VisualDSP 3.5 16-Bit\2181_labs\FIR *Filter*. Dentro de esta carpeta se encuentra otra carpeta con el nombre de "Simulación". A continuación seleccione el archivo *Fir Test*. Dar clic en Open.

- En el *Workspace* de MatLab escriba el siguiente comando y presione *Enter* en su teclado.

FirTest

- El programa que se está ejecutando requiere algunos datos para poder funcionar adecuadamente. Ingrese 8000 para la frecuencia de muestreo que solicita el programa, presione *Enter* en su teclado.

- Ingrese 1000 para el número de muestras, presione *Enter* en su teclado.

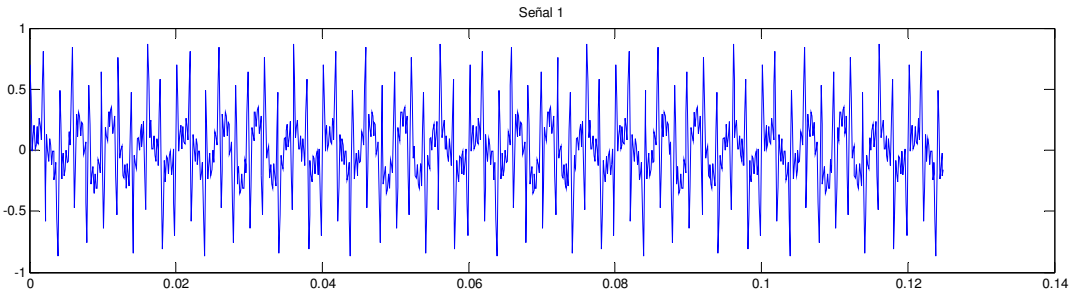
- Para el valor de las frecuencias senoidales ingrese el siguiente vector y luego presione *Enter* en su teclado.

[200 750 1250 1750 2250]

- Dibuje las figuras graficadas por MatLab. Como se interpreta la señal 1? Como se interpreta la señal 2? Como se interpreta la señal 3? Como se interpreta la señal 4? Observe atentamente la señal 3 y 4. Que puede concluir de estas dos graficas?

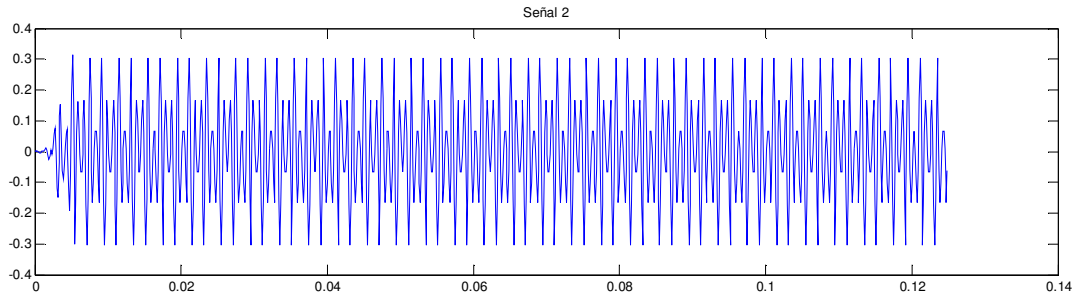
Rta. La señal 1 es mostrada en la figura 100, y corresponde a la señal generada, y compuesta de las frecuencias ingresadas en el paso anterior.

Figura 100. Figura 4-Laboratorio 7, parte 2



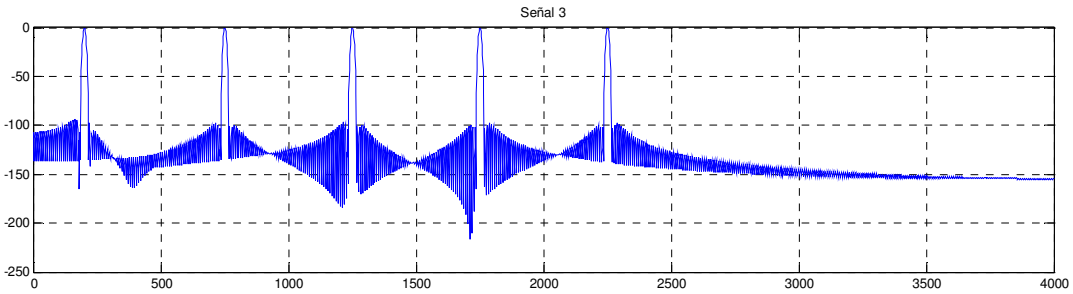
La señal 2 es mostrada en la figura 101, y corresponde a la señal generada una vez que esta ha pasado por el filtro.

Figura 101. Figura 5-Laboratorio 7, parte 2



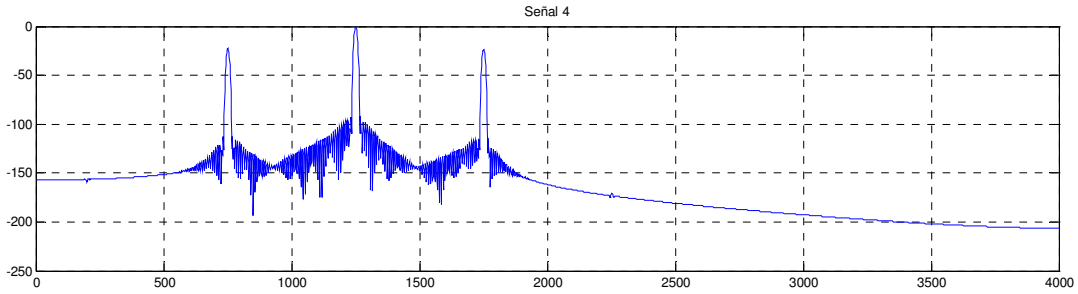
La señal 3 es mostrada en la figura 102, y corresponde al espectro de la señal generada y compuesta de las frecuencias ingresadas en el paso anterior.

Figura 102. Figura 6-Laboratorio 7, parte 2



La señal 4 es mostrada en la figura 103, y corresponde al espectro de señal generada una vez que esta ha pasado por el filtro.

Figura 103. Figura 7-Laboratorio 7, parte 2



Observando las señales 3 y 4 se puede concluir que la señal 3 es el espectro de la señal generada, y para esta señal es posible observar los picos para cada una de las frecuencias ingresadas. La señal 4 es el espectro de la señal una vez que ha pasado por el filtro, y para esta señal se puede observar como solo existen picos en el rango de frecuencias para el cual el filtro permite el paso de banda. El paso de banda total esta dado para el rango de 1000 y 1500, es por esta razón que el pico para la frecuencia de 1250 es el más alto en comparación con los demás picos, ya que estos se encuentran en el rango de las frecuencias de parada y de paso.

- En el *Workspace* de MatLab ingrese el siguiente nombre para el archivo de datos, y luego presione *Enter* en su teclado. Para la opción de generar otra señal presione la tecla N en su teclado y a continuación *Enter*.

'prueba. dat'

- Diríjase a la carpeta Simulación que se encuentra en la dirección C:\Archivos de programa\Analog Devices\VisualDSP 3.5 16-Bit\2181_labs\FIR Filter\Simulación y compruebe que se encuentran los siguientes archivos.

prueba.dat
Coeffs.dat

Abra los archivos prueba.dat y Coeffs.dat, para eso utilice un programa como el *WordPad*. ¿A que corresponden los datos en estos archivos? ¿Cuantos datos podría tener cada archivo y porque serian de dicha longitud? ¿Porque los datos de ambos archivos tienen una letra “r” al final de cada dato?
Rta. Los datos en los archivos Coeffs.dat y prueba.dat corresponden a los coeficientes del filtro y a la señal de prueba generada en MatLab respectivamente.

El archivo Coeffs.dat tiene 54 datos correspondientes a los 54 coeficientes del filtro FIR. El archivo prueba.dat tiene 1000 datos correspondientes a los coeficientes de la señal de prueba. Los datos de los archivos tienen la letra "r" para que cuando sean leídos por VisualDSP++ este reconozca los datos como fraccionarios y no como número en formato 1.15.

- Inicializar el software de programación VisualDSP++ en sesión de simulación.

- En la barra *Menu*, en el menú Project seleccionar la opción Open y ubicar la carpeta con el nombre FIR *Filter* que se encuentra en la dirección C:\Archivos de programa\Analog Devices\VisualDSP 3.5 16-Bit\2181_labs\FIR *Filter*. Dentro de esta carpeta se encuentra otra carpeta con el nombre de "Simulación". A continuación seleccione el archivo Simulacion2181.dpj. Dar clic sobre el botón izquierdo del mouse con el puntero sobre *Open*. En la ventana Project se visualizaran las carpetas que conforman el proyecto.

- Ubique el cursor sobre la ventana de edición (la ventana donde se encuentra el código fuente), y haga clic derecho para desplegar el menú del cual se debe seleccionar la opción *Line Numbers*.

- Las líneas de código 1 y 2 corresponden a las siguientes definiciones de constantes.

```
#define BUF_SIZE    1000
#define FIR_LEN     54
```

De acuerdo con el tamaño de las constantes BUF_SIZE y FIR_LEN, ¿a que corresponden dichas constantes?
Rta. Las constantes BUF_SIZE y FIR_LEN corresponden a las 1000 posiciones de la señal de prueba y los 54 coeficientes del filtro FIR respectivamente.

- Las líneas de código 8 al 13 corresponden a las siguientes declaraciones de buffer circulares.

```
.VAR/CIRC       Chan1_In[BUF_SIZE] = "prueba.dat";
.VAR/CIRC       Chan2_In[BUF_SIZE] = "prueba.dat";
.VAR/CIRC       Chan1_Out[BUF_SIZE];
.VAR/CIRC       Chan2_Out[BUF_SIZE];
.VAR/CIRC       Chan1_Dly[FIR_LEN];
.VAR/CIRC       Chan2_Dly[FIR_LEN];
```

¿A que corresponde cada buffer circular y porque su tamaño está definido por las constantes BUF_SIZE y FIR_LEN?

Rta. Los buffers Chan1_In[BUF_SIZE] y Chan2_In[BUF_SIZE] corresponden a los buffers en donde se almacena la señal de entrada, que está dada por los valores almacenados en la variable prueba.dat. Los buffers Chan1_Out[BUF_SIZE] y Chan2_Out[BUF_SIZE] corresponden a los buffers en donde se almacenara la señal filtrada de salida, por eso que su tamaño sea de 1000 posiciones(BUF_SIZE). Por último los buffers Chan1_Dly[FIR_LEN] y Chan2_Dly[FIR_LEN] corresponden a buffers temporales de 54 posiciones(FIR_LEN), en donde se almacenan de manera temporal 54 valores de la señal de prueba de entrada. Los datos van saliendo del buffer temporal hasta que cada valor almacenado en él se haya multiplicado con los 54 coeficientes del filtro FIR.

- Las líneas de código 18 al 19 corresponden a las siguientes declaraciones de buffer circulares.

```
.VAR/CIRC       FirCoeffs_1[FIR_LEN] = "Coeffs.dat";
.VAR/CIRC       FirCoeffs_2[FIR_LEN] = "Coeffs.dat";
```

¿Qué datos son definidos en los buffers FirCoeffs_1[FIR_LEN] y FirCoeffs_2[FIR_LEN]?
Rta. En los buffers FirCoeffs_1[FIR_LEN] y FirCoeffs_2[FIR_LEN] se definen los datos correspondientes a los coeficientes del filtro FIR, de ahí que su tamaño este dado por la constante FIR_LEN de 54 posiciones.

- La línea de código 30 realiza un llamado de subrutina a una etiqueta cuyo nombre es "*Initialization*", esta subrutina se muestra a continuación y se encuentra entre las líneas 67 y 79.

```
Initialization:
    i0 = Chan1_Dly;
    m0 = 1;
    l0 = length(Chan1_Dly);
    i1 = Chan2_Dly;
    m1 = 1;
    l1 = length(Chan2_Dly);
    ax0 = 0;
    cntr = FIR_LEN;
    do Clean until ce;
            dm(i0,m0) = ax0;
Clean:  dm(i1,m1) = ax0;
    rts;
```

La anterior subrutina utiliza a i0, e i1 como los índices que apuntan a los buffers Chan1_Dly y Chan2_Dly, m0 y m1 son el tamaño del incremento para el DAG. Por último l0 y l1 corresponde al tamaño del DAG que esta dado por el mismo tamaño de los buffers, es decir 54. Una vez que los DAG se han configurado se inicia un contador que va desde 54 hasta cero. En cada paso del contador se realiza la etiqueta *Clean*. ¿Qué labor desempeña la etiqueta *Clean*?
Rta. La etiqueta *Clean* se encarga de colocar cero (0) en cada posición de los buffers Chan1_Dly y Chan2_Dly, dicho en otras palabras, inicializa los buffers en cero.

- Luego de realizarse la subrutina "*Initialization*", las líneas de código 33 a la 44 se encargan de la configuración del canal 1, y las líneas 48 a la 59 de la configuración del canal 2. El código para el canal 1 y canal 2 es similar, la diferencia entre ellos radica en el buffer al cual apuntan, siendo para canal 1 los buffers asignados a él y para canal 2 los buffers asignados para canal 2. Este código entre las líneas antes mencionadas se encarga de configurar el DAG para las necesidades de cada canal.

- Las líneas 45 y 60 realizan un llamado de subrutina a una etiqueta cuyo nombre es "*Fir Process*", y a su vez esta se encarga de hacer llamado a otra etiqueta cuyo nombre es "*FirFilter*". Estas subrutinas se muestran a continuación y se encuentran entre las líneas 82 y 101.

```
FirProcess:
    cntr = BUF_SIZE;
    do FirProcessLoop until ce;
        ax0 = dm(i0,m0);
        call FirFilter;
FirProcessLoop:
        dm (i2,m2) = mr1;
    rts;
// Rutina del filtro FIR
FirFilter:
    dm(i1,m1) = ax0;
    cntr = FIR_LEN - 1;
    mr = 0,     mx0 = dm(i1,m1),   my0 = pm(i4,m4);
    do FirLoop until ce;
    FirLoop:    mr=mr+mx0*my0 (ss), mx0 = dm(i1,m1), my0 =
pm(i4,m4);      if mv sat mr;
    rts;
```

Las anteriores subrutinas realizan todo el procedimiento del filtro FIR. La primera subrutina se encarga de enviar las 1000 muestras de la señal de prueba a la

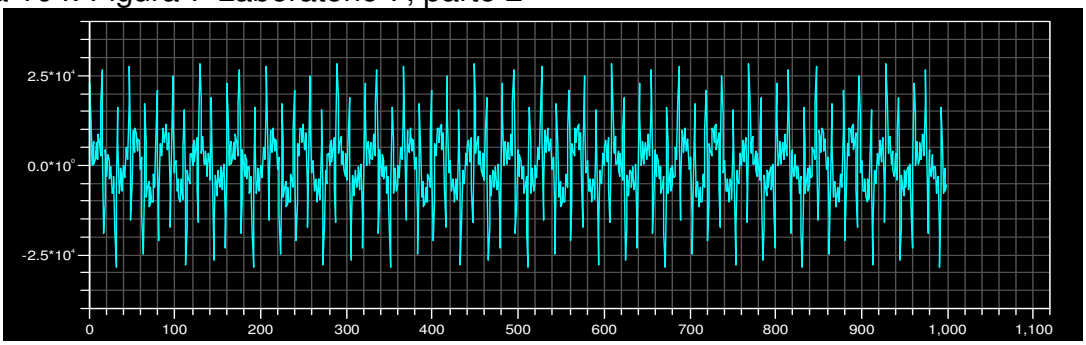
segunda subrutina, la cual se encarga de multiplicar y acumular el producto de las muestras con los coeficientes. En el código inicialmente se establece un contador con un valor BUF_SIZE (1000) y se realiza el bucle "*FirProcessLoop*" tantas veces como contador indique. Dentro del bucle se encuentra la segunda subrutina "*FirFilter*", en esta subrutina se toma el valor de la muestra que se encuentra en AX0 y se ubica en el buffer temporal. Un nuevo contador se establece con un valor de 53 (FIR_LEN-1). Luego se reinicia el acumulador (mr=0) y se ubican los valores de la primera muestra y el primer coeficiente en MX0 y MY0 respectivamente. A continuación se realiza la convolución entre los coeficientes del filtro y la señal de entrada en el bucle "*FirLoop*". En dicho bucle se multiplican los datos y se acumulan, para después adquirir el siguiente coeficiente y la siguiente muestra de la señal. Inmediatamente después se realiza el último cómputo y se redondea el resultado. Luego, en caso de haber existido overflow se satura MR con su máximo valor y se retorna a la primera subrutina, en donde se van llenando los valores del buffer de salida.

- Del menú View seleccione la opción *Debug* Windows, del menú emergente seleccione la opción *Plot* y New.

- En la ventana *Plot Configuration*, del cuadro de selección correspondiente a *Type* seleccione la opción Line *Plot*, y asígnele un título al ploteo en el cuadro de texto *title*. En las características de los datos, asígnele un nombre a los datos a graficar en el cuadro de texto *Name*. En el cuadro desplegable Memory seleccione la opción DM. Haga clic sobre el botón izquierdo del mouse con el puntero ubicado sobre el botón *Browse*, de la ventana emergente seleccione la opción Chan1_In. En el cuadro de texto *Count* de la ventana *Plot Configuration* escriba el tamaño del eje X. Para este caso el búfer (BUF_SIZE) es de 1000 posiciones así que digite este número. Del menú desplegable Data seleccione la opción short, a continuación haga clic izquierdo con el puntero ubicado sobre el botón *Add* y por último Ok.

- Observe que inmediatamente después que se cierra la ventana *Plot Configuration* aparece una grafica para los datos de Chan1_In. Dibuje y compare la imagen graficada por el VisualDSP++ con las gráficas de MatLab. ¿Qué se puede deducir de la señal graficada en comparación con las graficas de MatLab?
Rta. La señal graficada por el VisualDSP++ corresponde a la grafica de la señal de prueba elaborada en MatLab, en otras palabras, se trata de la grafica de la señal almacenada en el archivo prueba.dat. Si comparamos la señal con las obtenidas en MatLab, esta corresponde a la Señal 1 que es la señal generada y compuesta por las frecuencias ingresadas. La señal obtenida en el VisualDSP++ se muestra en la figura 104

.Figura 104. Figura 7-Laboratorio 7, parte 2



- Haga clic sobre el botón derecho del mouse con el puntero ubicado sobre la gráfica. Del menú desplegable mostrado en la figura 105 seleccione la opción *Modify Settings*. De la ventana emergente seleccione la pestaña *Data Processing* que se muestra en la figura 106.

Figura 105. Figura 9-Laboratorio 7, parte 2

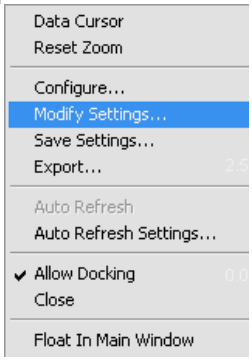
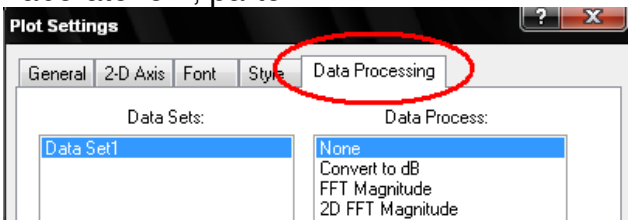
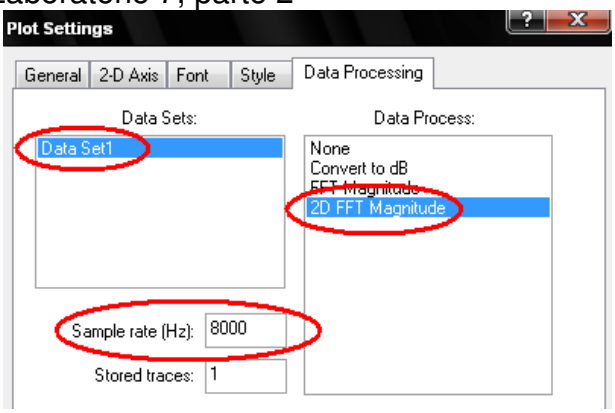


Figura 106. Figura 10–Laboratorio 7, parte 2



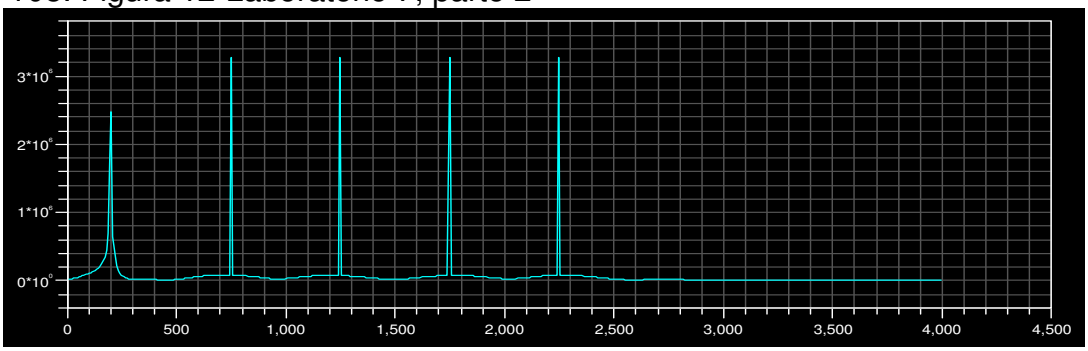
- En la ventana *Plot Settings* seleccione la opción que corresponde a la señal de entrada en el cuadro Data Sets. Para el cuadro *Data Process* seleccione la opción 2D FFT *Magnitude*. Por último, en el cuadro de texto *Sample Rate* (Hz) escriba la frecuencia de muestreo que corresponde a 8000Hz. La figura 107 muestra los anteriores parámetros. A continuación haga clic izquierdo en el mouse con el puntero ubicado sobre Aceptar.

Figura 107. Figura 11-Laboratorio 7, parte 2



- Observe, dibuje y compare la nueva imagen graficada por el VisualDSP++ con las gráficas de MatLab. ¿Qué se puede deducir de la señal graficada en comparación con las graficas de MatLab? ¿La señal graficada es la esperada?
Rta. La nueva señal graficada corresponde al espectro de las frecuencias de la señal de prueba generada en MatLab, es decir, es el espectro de la señal almacenada en el archivo prueba.dat. Si comparamos la señal con las obtenidas en MatLab, esta corresponde a la Señal 3, que es el espectro de las frecuencias ingresadas en la señal generada de prueba. La señal graficada es la esperada ya que en la grafica se pueden observar picos de frecuencia justamente en las posiciones correspondientes al vector de frecuencias que se ingreso en MatLab. La señal obtenida en el VisualDSP++ se muestra en la figura 108.

Figura 108. Figura 12-Laboratorio 7, parte 2



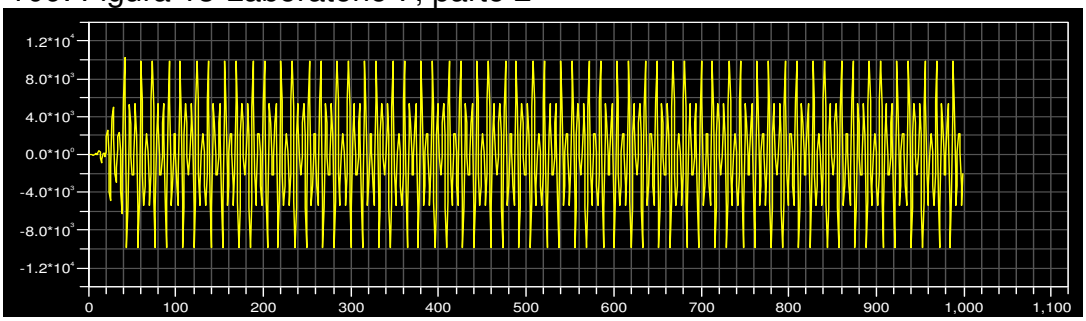
- Depure y corra el programa por lo menos por 10 segundos.

- Genere un nuevo ploteo para la señal de salida que corresponde a Chan1_Out.

- Observe, dibuje y compare la nueva imagen graficada por el VisualDSP++ con las gráficas de MatLab. ¿Qué se puede deducir de la señal graficada en comparación con las graficas de MatLab?

Rta. La señal graficada corresponde a la repuesta del filtro FIR, luego que ha pasado a través de él la señal de prueba. Si comparamos la señal graficada con las graficas de MatLab, esta gráfica corresponde a la Señal 2, es decir, la señal compuesta de las frecuencias ingresadas luego de haber sido filtrada. La señal obtenida en el VisualDSP++ se muestra en la figura 109.

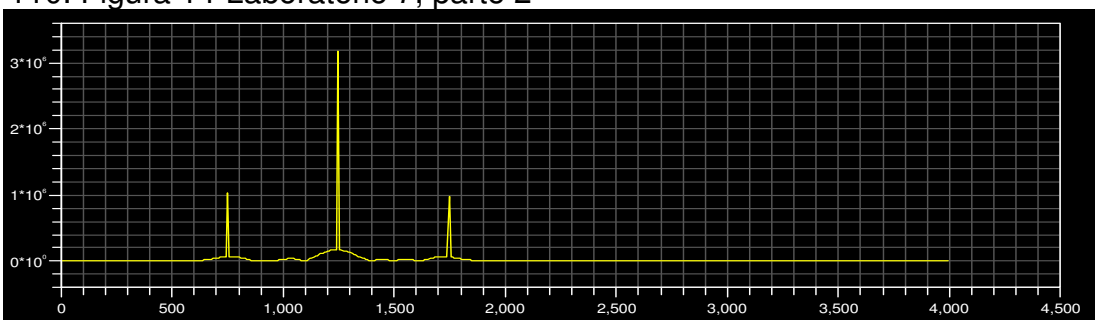
Figura 109. Figura 13-Laboratorio 7, parte 2



- Observe y dibuje el espectro de la anterior gráfica de VisualDSP++. Compare el espectro graficado con la Señal 4 de MatLab. ¿Qué se puede deducir de la señal graficada en comparación con la señal de MatLab?

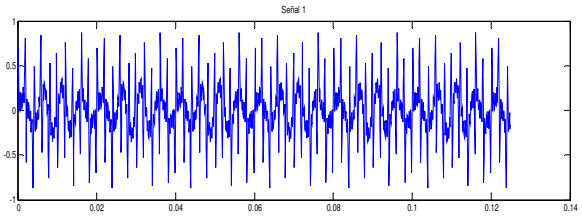
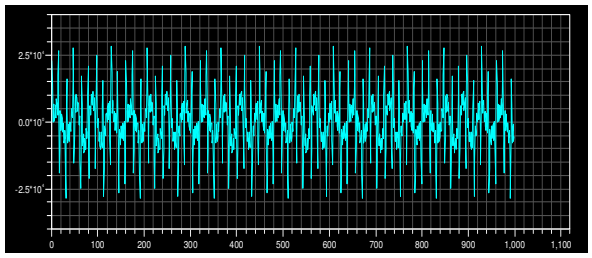
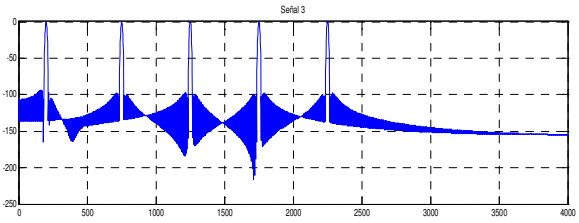
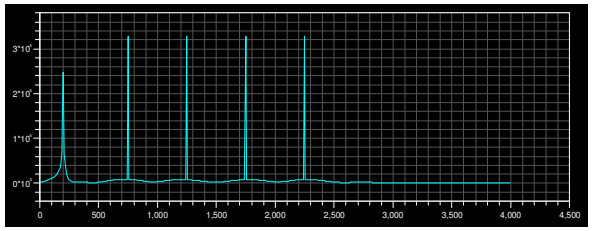
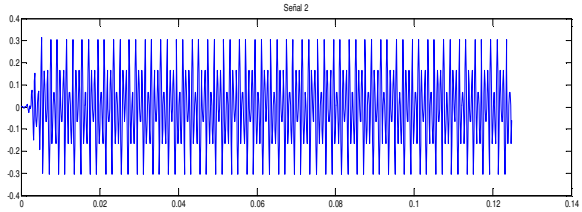
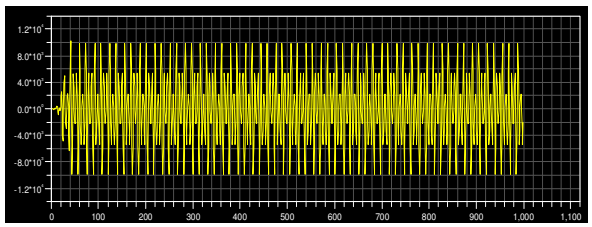
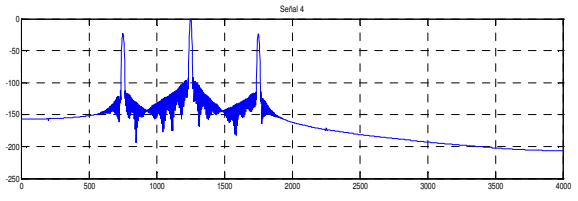
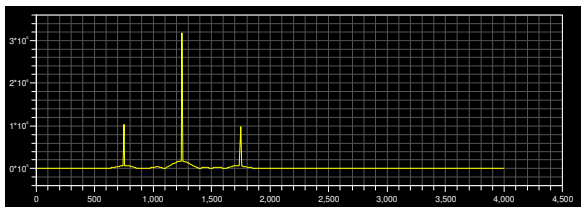
Rta. La gráfica obtenida por VisualDSP es la respuesta en frecuencia de la señal de prueba luego de haber sido filtrada. Esta señal es al igual que la Señal 4 de MatLab la muestra clara del funcionamiento del filtro FIR en el VisualDSP++, ya que solo permitió el paso de ciertos valores de frecuencia que se encuentran dentro del rango de las bandas de paso del filtro pasa bandas. La señal obtenida en el VisualDSP++ se muestra en la figura 110.

Figura 110. Figura 14-Laboratorio 7, parte 2



- Elabore una tabla que muestre gráficamente la respuesta del filtro FIR, tanto para MatLab como para VisualDSP++.

Tabla 30. Tabla de resultados laboratorio 7, parte 2

MATLAB	VisualDSP3.5++
	
	
	
	

Conclusiones.

- La herramienta de programación MatLab permitió, que haciendo uso de una de sus *Toolbox* fuera posible diseñar un filtro FIR de orden 53, cuyo planteamiento matemático presenta un alto grado de dificultad.

- La herramienta de programación MatLab también dio soporte para la creación de la señal de prueba que fue utilizada en el filtro FIR implementado para la simulación. La señal de prueba fue utilizada tanto en MatLab como en VisualDSP++ para poder observar la respuesta del filtro FIR en ambos entornos de programación.

- La simulación del filtro FIR para el ambiente de desarrollo VisualDSP++, comprobó de manera visual que su funcionamiento está al nivel de otros entornos de programación, y que su respuesta, tanto en el tiempo como en la frecuencia no presenta mayores diferencias respecto de la respuesta de otros softwares diseñados para el mismo fin.

7.7.3 Parte 3. Filtro FIR, ejecución en tiempo real.

Objetivos.

- Comprobar el funcionamiento del filtro FIR en modo de ejecución en tiempo real.

- Aprender a definir funciones globales y externas

Lecturas recomendadas.

- Profundizar sobre las directivas .GLOBAL y .EXTERN

Materiales.

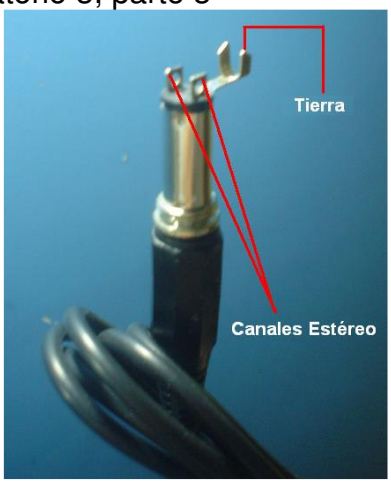
- Osciloscopio.
- Generador de señales.
- Dos (2) cables estéreo-estéreo.
- Dos (2) Conectores hembra estéreo.

Procedimiento.

- Conecte uno de los cables estéreo al jack de entrada de la tarjeta, y el otro cable estéreo al jack de salida.

- Destape los conectores hembra estéreo e insértelos a la punta libre de los cables estéreo, la conexión se muestra en la siguiente figura.

Figura 111. Figura 1-Laboratorio 3, parte 3



- Al finalizar el montaje de las conexiones el circuito debe verse de la siguiente manera.

Figura 112. Figura 2-Laboratorio 7, parte 3

- Conecte el generador de señales a ambos canales en 2 (ver figura anterior). Conecte una punta del osciloscopio en este punto para tener la señal de referencia en uno de los dos canales.

- Conecte la otra punta del osciloscopio en 1 (ver figura anterior). La punta puede ser conectada en cualquiera de los dos canales.

- Conecte el sistema de desarrollo ADSP-2181 EZKit y cambie la sesión a modo de ejecución de proyectos en tiempo real.

- En la barra *Menu*, en el menú Project seleccionar la opción Open y ubicar la carpeta con el nombre FIR *Filter* que se encuentra en la dirección C:\Archivos de programa\Analog Devices\VisualDSP 3.5 16-Bit\2181_labs\FIR *Filter*. Dentro de esta carpeta se encuentra otra carpeta con el nombre de "*FiltroFIR*". A continuación seleccione el archivo Fir_Runtime.dpj. Dar clic en Open. En la ventana Project se visualizaran las carpetas que conforman el proyecto.

- Observe que al abrir el proyecto Fir_Runtime.dpj, este está conformado por dos archivos de código, Fir.asm y TalkThru_Base_FIR.asm. Seleccione para ambos archivos la opción de *Line Numbers*.

- Observe rápidamente los archivos Fir.asm y TalkThru_Base_FIR.asm. Note que el archivo TalkThru_Base_FIR.asm es prácticamente igual a la plantilla de trabajo TalkThru, excepto por algunas diferencias que serán revisadas más adelante. También observe que el archivo Fir.asm es similar al archivo que se utilizó para la simulación del filtro FIR en la práctica anterior.

- Las líneas de código 1 y 2 del archivo Fir.asm se muestran a continuación.

```
#define     FIR_LEN_1       54
#define     FIR_LEN_2       54
```

¿Siendo el tamaño de las anteriores constantes 54 que función podrían cumplir estas constantes?
Rta. Conociendo que el valor de las constantes es de 54 se deduce que estas constantes corresponden al tamaño del vector de demora del filtro FIR para ambos canales, y a su vez también indican el número de coeficientes del filtro.

- Las líneas de código 5,6 y 7 del archivo Fir.asm se muestran a continuación.

```
.GLOBAL InitializationFir;
.GLOBAL FirFilter_Ch1;
.GLOBAL FirFilter_Ch2;
```

¿Qué se puede inferir de las anteriores asignaciones de directivas?
Rta. Por el hecho de ser .GLOBAL se puede inferir que las subrutinas *InitializationFir*, FirFilter_Ch1, y FirFilter_Ch2, propias del archivo Fir.asm, serán en algún momento llamadas desde otro archivo de código del proyecto dentro de la ejecución del programa, para este caso específico el archivo Fir.asm en algún momento de la ejecución del proyecto podría hacer un llamado de las subrutinas que se encuentren definidas bajo la directiva .GLOBAL.

- Observe que el archivo Fir.asm no presenta la etiqueta llamada *Firprocess*, esto es porque para ejecución en tiempo real no se tiene un archivo con una señal de prueba, se tienen muestras tomadas en tiempo real provenientes del códec que se dirigen directamente a la subrutina encargada del procesamiento FIR.

Las líneas de código 8,9 y 10 del archivo TalkThru_Base_FIR.asm se muestran a continuación. ¿Qué se puede inferir de las anteriores asignaciones de directivas?

```
.EXTERN   InitializationFir;
.EXTERN   FirFilter_Ch1;
.EXTERN   FirFilter_Ch2;
```

Rta. Por el hecho de ser .EXTERN se puede inferir que las subrutinas *InitializationFir*, FirFilter_Ch1, y FirFilter_Ch2, no pertenecen al código de programa del archivo TalkThru_Base_FIR.asm, pero si se encuentran en otro archivo dentro del mismo proyecto, y siempre que las mismas subrutinas estén definidas en su archivo fuente como .GLOBAL, estas podrán ser llamadas y traídas a cualquier archivo en donde estén definidas bajo la directiva .EXTERN.

Las líneas de código 296, 297 y 298 del archivo TalkThru_Base_FIR.asm se muestran a continuación. ¿Qué función realizan las anteriores líneas de código?

```
AX1 = 0;
DM (status_irqe) = AX1;
CALL InitializationFir;
```

Rta. Las líneas de código 296 y 297 se encargan de inicializar el estado de la interrupción externa IRQE en cero (0). Por otra parte, la línea de código 298 hace el llamado de la subrutina externa "*InitializationFir*" para configurar los parámetros iniciales del filtro FIR.

Las líneas de código 320 a la 326 del archivo TalkThru_Base_FIR.asm se muestran a continuación. ¿Cuál es el funcionamiento de la anterior subrutina de atención a interrupción?

```
isrIrqE:
  TOGGLE FL1;
  AX0 = DM (status_irqe);
  AR = AX0 XOR 0x0001;
  DM (status_irqe) = AR;
  RTI;
```

Rta. Inicialmente se cambia el estado lógico de la bandera externa FL1 indicando que el programa atendió al llamado de interrupción. La siguiente línea de código mueve el valor del estado de la interrupción al registro AX0, luego se realiza una

operación XOR para cambiar el estado lógico del valor contenido en AX0 y almacenarlo en AR. Por último, el valor de AR es movido a la posición de memoria correspondiente a *status_irqe*, indicando el cambio de estado de la interrupción.

- Las líneas de código 350 a la 371 del archivo TalkThru_Base_FIR.asm se muestran a continuación.

```
input_samples:
  ENA SEC_REG;
  AX0 = DM (status_irqe);
  AR = PASS AX0;
  IF EQ JUMP Filtrar;
  AX0 = DM (rx_buf + 1);
  DM (tx_buf + 1) = AX0;
  AX0 = DM (rx_buf + 2);
  DM (tx_buf + 2) = AX0;
  JUMP Final;
Filtrar:
  //canal izquierdo
  AX0 = DM (rx_buf  + 1);
  CALL FirFilter_Ch1;
  DM (tx_buf + 1) = MR1;
  //canal derecho
  AX0 = DM (rx_buf  + 2);
  CALL FirFilter_Ch2;
  DM (tx_buf + 2) = MR1;
Final:
  NOP;
  RTI;
```

La línea de código 351 se encarga de habilitar los registros secundarios. Las líneas de código 352, 353 y 354 son las encargadas de consultar el estado de *state_flag* y en función del valor almacenado en esta posición de DM, se continúa en el orden lógico del programa o se salta a otra etiqueta. En caso de seguir el orden lógico del programa, las líneas de código de la 355 a la 359 hacen que el programa se comporte como la plantilla de trabajo TalkThru. En caso que el programa salte a otra etiqueta, las líneas de código de la 360 a la 368, se encargan de adquirir los datos del buffer correspondientes a las muestras de la señal de entrada, para luego hacer un llamado de subrutina externa que realiza la labor de filtro FIR. El valor de retorno de la subrutina es entonces transmitido por el códec. El llamado de subrutina de filtro FIR se realiza tanto para canal izquierdo como para canal derecho.

- Ajuste el generador de señales con una señal senoidal, cuyo valor pico-pico no sea mayor de 3 Vrms.

- Ajuste la frecuencia del generador en un valor entre 500Hz y 1000Hz.

- Las líneas de código 18 a la 25 del archivo Fir.asm se muestran a continuación.

```
.VAR/CIRC     FirCoeffs_1[FIR_LEN_1] = "Coeffs1.dat";
.VAR/CIRC     FirCoeffs_2[FIR_LEN_2] = "Coeffs1.dat";

//.VAR/CIRC   FirCoeffs_1[FIR_LEN_1] = "Coeffs2.dat";
//.VAR/CIRC   FirCoeffs_2[FIR_LEN_2] = "Coeffs2.dat";

//.VAR/CIRC   FirCoeffs_1[FIR_LEN_1] = "Coeffs3.dat";
//.VAR/CIRC   FirCoeffs_2[FIR_LEN_2] = "Coeffs3.dat";
```

Observe que las líneas de código 21, 22, 24 y 25, se encuentran como comentario de programa, en caso de no encontrarse así por favor ajústelas a como se muestra en este paso del procedimiento.

- Depure y corra indefinidamente el programa.

- Observe las señales mostradas en el osciloscopio. La señal correspondiente a la señal de salida debe ser de un valor muy cercano a cero o cero. En caso de no cumplirse la condición anterior, presione el pulsador correspondiente a interrupción externa (IRQE) del sistema de desarrollo ADSP-2181 DSPKit y compruebe que el cambio en la señal de salida.

Varíe lentamente la frecuencia del generador de señales entre el rango de 500 Hz a 4000 Hz. Observe como la señal de salida aumenta y disminuye en amplitud a medida que se varia la frecuencia. ¿A qué se debe esto?
Rta. El filtro FIR está diseñado como un filtro pasa bandas. A medida que se varía la frecuencia, esta empieza a ingresar a las bandas características del filtro, es por eso que en algunos intervalos de frecuencia la señal tiene una amplitud que crece desde cero hasta un valor máximo en un intervalo, y en otro intervalo de frecuencias decrece desde un valor máximo a cero. Existe un intervalo de frecuencias para el cual la señal de salida tiene la misma amplitud a lo largo del intervalo.

- De acuerdo con las variaciones de la señal de salida, ¿cuáles son las bandas características del filtro pasa banda definido por el archivo “Coeffs1.dat”, establecido en las líneas 18 y 19 del archivo Fir.asm?

Rta. Frecuencia de corte, inferior (Fs1) = 1000 Hz
Frecuencia de paso, inferior (Fp1) = 1500 Hz
Frecuencia de paso, superior (Fp2) = 2000 Hz
Frecuencia de corte, superior (Fs2) = 2500 Hz

- En función de las bandas del filtro defina cinco posiciones para el generador de señales las cuales corresponden a: la primera posición para una frecuencia menor a la frecuencia de corte inferior, la segunda posición para una frecuencia entre la frecuencia de corte inferior y la frecuencia de paso inferior, la tercera posición para una frecuencia entre las frecuencias de paso superior e inferior, la cuarta posición para una frecuencia entre la frecuencia de paso superior y la frecuencia de corte superior, y la quita posición para una frecuencia mayor a la frecuencia de corte superior. Dibuje las señales mostradas por el osciloscopio en cada posición del generador, y ubique cada dibujo en la casilla correspondiente de la Tabla 31.

Rta. Las graficas adquiridas para cada posición del generador se muestran en la Tabla 31

- Detenga la ejecución del proyecto.

- Retire las opciones de comentario (//) de las líneas de código 21 y 22 del archivo Fir.asm para definir el archivo “Coeffs2.dat”. Coloque opciones de comentario a las líneas de código 18 y 19 del archivo Fir.asm. Depure, corra el proyecto, y realice nuevamente los últimos cuatro pasos.

Rta. Las bandas de paso para el archivo Coeffs2.dat son:

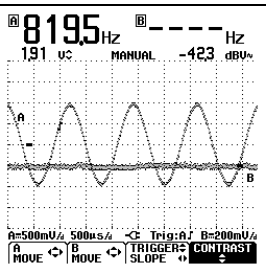
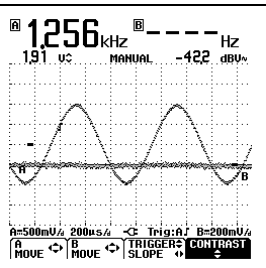
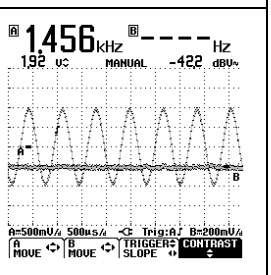
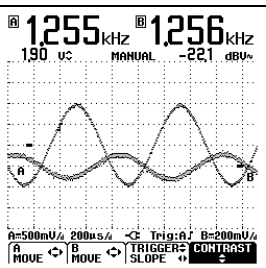
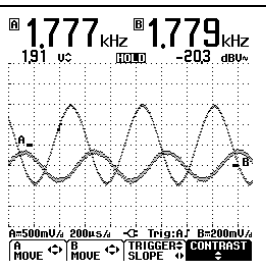
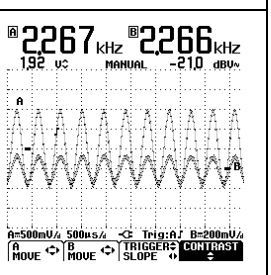
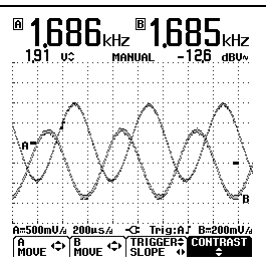
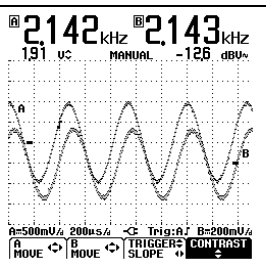
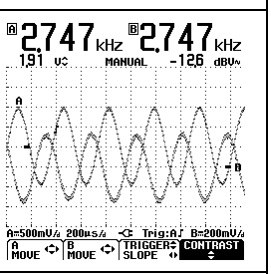
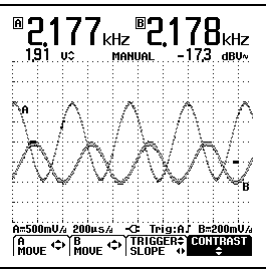
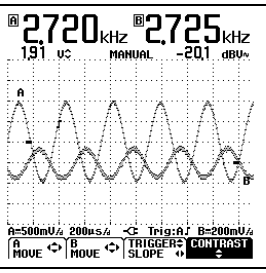
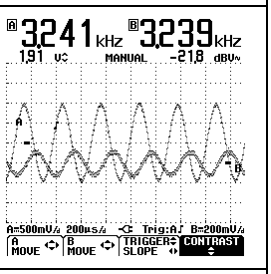
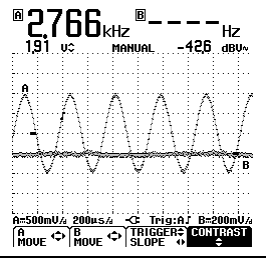
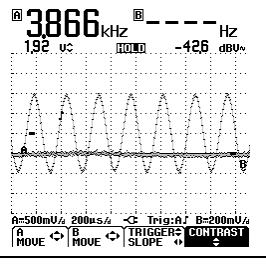
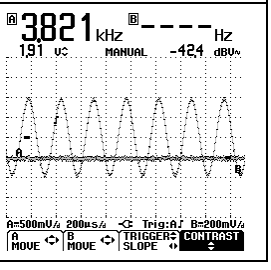
Frecuencia de corte, inferior (Fs1) = 1500 Hz
Frecuencia de paso, inferior (Fp1) = 2000 Hz
Frecuencia de paso, superior (Fp2) = 2500 Hz
Frecuencia de corte, superior (Fs2) = 3000 Hz

- Retire las opciones de comentario (//) de las líneas de código 24 y 25 del archivo Fir.asm para definir el archivo “Coeffs3.dat”. Coloque opciones de comentario a las líneas de código 21 y 22 del archivo Fir.asm. Depure, corra el proyecto, y realice nuevamente el procedimiento efectuado para los archivos “Coeffs1.dat” y “Coeffs2.dat”.

Rta. Las bandas de paso para el archivo Coeffs3.dat son:

Frecuencia de corte, inferior (Fs1) = 2000 Hz
Frecuencia de paso, inferior (Fp1) = 2500 Hz
Frecuencia de paso, superior (Fp2) = 3000 Hz
Frecuencia de corte, superior (Fs2) = 3500 Hz

Tabla 31. Tabla de resultados laboratorio 7, parte 3

Posición del Generador	Coeffs1	Coeffs2	Coeffs3
1			
2			
3			
4			
5			

Conclusiones.

- Se comprobó en tiempo real la funcionalidad del filtro FIR pasa banda para una señal compuesta de diferentes frecuencias. Las señales observadas en el osciloscopio fueron las esperadas para los distintos parámetros de banda del filtro, indicando de esta manera que el filtro funciona de forma correcta.

- Es posible implementar varios filtros pasa banda en el código que se expuso en este laboratorio sin alterar en gran medida el código propio del programa, es cuestión de cambiar el nombre del archivo que contiene los coeficientes del filtro de orden 53, y que el nuevo archivo ingresado en el programa exista dentro de la carpeta en la que se encuentran los códigos fuente del proyecto.

- El código del laboratorio anterior puede ser alterado levemente de tal forma que sea útil para filtros de orden diferente de 53, esto depende de que el archivo que contiene los coeficientes del filtro sea correspondiente con la constante definida al inicio del archivo Fir.asm.

- El uso de las directivas .EXTERN y .GLOBAL permiten que la elaboración del proyecto sea más estructurado y ordenado, dando la oportunidad de dividir el proyecto en diferentes archivos, y haciendo uso de estas directivas, hacer llamados a subrutinas que se encuentran definidas en otros archivos de código.

8 CONCLUSIONES

En el proceso de iniciar el trabajo correspondiente para el desarrollo de los objetivos planteados en el plan de trabajo, se presentaron una serie de inconvenientes los cuales no permitieron que el desarrollo de esta fluyera como se tenía planteado. En busca de superar estos inconvenientes fue necesario acudir a toda la información que estaba al alcance de nuestras manos, desde la información propia de los docentes, hasta la información brindada por los técnicos de soporte de Analog Devices. Es recomendable, que para el desarrollo de cualquier proyecto, el fundamento teórico sobre el cual se establecen las bases de la investigación tenga un soporte que vaya más allá de la información que existe en la web, es decir, es fundamental que la experiencia de proyectos anteriores, y el conocimiento adquirido por otros investigadores, sea parte vital del marco teórico y conceptual sobre el cual se desarrolla la investigación.

La documentación que existe sobre la arquitectura y modo de operación de los procesadores ADSP-218x es muy amplia y de fácil acceso, pero, no toda la información que se obtiene es útil e implementable para el sistema de desarrollo ADSP-2181 EZKit. La familia de procesadores ADSP-218x ha presentado con el pasar de los años un crecimiento y una evolución significativa. En ese proceso de cambio y crecimiento, Analog Devices se vio forzado, debido a la presión ejercida por la competencia, a evolucionar en sus herramientas de programación, como en sus dispositivos, dejando atrás una información propia de herramientas clásicas de programación y dispositivos que se fueron rezagando a medida que nuevos y mejores dispositivos fueron creados. Pese a que el procesador ADSP-2181 es un dispositivo clásico, es de los más óptimos en fines académicos, pero su documentación está basada en herramientas de programación clásicas y obsoletas, en comparación con las actuales. Para superar este inconveniente fue necesario indagar a fondo sobre las herramientas de programación actuales, con el fin de tomar toda la información clásica existente, mezclara con la actual, y poder crear proyectos DSP utilizando el sistema de desarrollo ADSP-2181 EZKit.

El lenguaje de programación que se utilizó para implementar los proyectos elaborados en la anterior investigación fue *Assembler.* Inicialmente se optó por utilizar C como lenguaje de programación, debido a la sencillez que presentan las estructuras cuando se programa en este lenguaje, ya que es más didáctico para fines académicos. Al final se decidió utilizar *assembler* como lenguaje de programación, porque una de las grandes ventajas que presenta programar procesadores ADSP-218x utilizando este lenguaje, y que ilustra el poder de la arquitectura del procesador, es como las operaciones de multifunción toman

ventaja del paralelismo inherente de la arquitectura de estos procesadores, proveyendo distintas combinaciones de movimientos de datos, lectura y escritura de memoria, y cálculos de unidades computacionales, todo en un ciclo sencillo. Esta es una ventaja que el lenguaje C desperdicia, y es por esta razón recomendable utilizar *assembler* ya que optimiza los proyectos y le da velocidad a la ejecución de los mismos, al ahorrar líneas de código gracias al alto grado de paralelismo.

El sistema de desarrollo ADSP-2181 EZKit es una herramienta de trabajo íntegra, que permite al usuario disponer de las herramientas necesarias para crear proyectos DSP, y que es sin duda alguna una herramienta diseñada con fines académicos. Una desventaja que presenta este sistema de desarrollo, son los canales de salida del codec AD1847. El AD1847 toma la señal de entrada y realiza sobre ella una conversión A/D, donde las muestras digitales son luego utilizadas en el procesamiento. Para generar la señal de salida es necesario realizar una conversión D/A. La señal que se tiene luego de la conversión D/A es sin duda alguna una señal que ha sido procesada correctamente, y que presenta una demora debido al tiempo de procesamiento, pero que es diferente en cuestión de amplitud si se compara con la señal de entrada o con la señal de salida que se espera. Esto se debe a que el codec provoca una atenuación significativa sobre la señal, y es por esta razón que no es posible lograr que la señal de entrada sea la misma de salida. Es necesario ubicar una etapa de amplificación a la salida del codec para generar una señal de salida cuyos parámetros sean los esperados.

Entre los objetivos planteados en el plan de trabajo estaba la creación de experiencias didácticas para el aprendizaje del modo de funcionamiento del sistema de desarrollo ADSP-2181 EZKit. Estas experiencias fueron diseñadas en un formato de explicación-procedimiento, en donde el estudiante inicia su aprendizaje desde las herramientas más básicas del sistema de desarrollo tales como las unidades computacionales, hasta llegar a un complejo desafío de diseño, en donde se implementa un filtro FIR sobre la tarjeta. El propósito que las experiencias hayan sido elaboradas de esta manera radica en el poco conocimiento que se tiene sobre el modo de funcionamiento del sistema de desarrollo. Las experiencias constantemente cuestionan al estudiante respecto del procedimiento inmediatamente anterior, para que el conocimiento sea visto en el software de programación, analizado debido a las preguntas, y asimilado al generar las respuestas.

Una ventaja que existe en utilizar procesadores ADSP-21xx es que el modo de programación entre una familia y otra no varía demasiado. De ahí que el procesador ADSP-2181 tenga un gran académico, ya que sobre él es posible aprender el modo de funcionamiento de la familia 21xx. Para futuros proyectos y debido a la similitud de programación de los dispositivos de *Analog Devices* (procesadores 21xx, *TigerSHARC*, SHARC, y *Blackfin*), es posible implementar

aplicaciones tales como sistemas karaoke moderno, codificación mp3, compresión de voz, control de motores, aplicaciones para tecnología celular, entre otras.

9 BIBLIOGRAFIA

ANALOG DEVICES, ADSP-2181EZ-KIT LITE Evaluation System Manual, Segunda edición, Analog Devices, 2003.p. 19-54

ANALOG DEVICES, ADSP-218x DSP Hardware Reference, Primera edición, Analog Devices, 2001.p. 45-210

ANALOG DEVICES, ADSP-218x Instruction Set Reference, Segunda edición, Analog Devices, 2004.p. 21-36

ANALOG DEVICES, VISUALDSP++ 3.5 User´s Guide for 16-Bit *Processors*, Primera edición, Analog Devices, 2003.p. 35-269

ANALOG DEVICES, AD1847 Serial-Port 16-Bit SoundPort Stereo Codec, Norwood, Analog Devices,1996.p. 1-19

ANALOG DEVICES, ADSP-2181 DSP Microcomputer, Norwood, Analog Devices, 1998.p. 3-11

ARANGURE CARDONA, Jaime Andrés. Nota de aplicación No 0003 filtros FIR en el ADSP-218x, SanJaac Electronics, Medellín, 2004.p. 1-5

PERALTA MARULANDA, Johan, Procesadores digitales de señal, familia ADSP 21xx, Bucaramanga, 2007.p. 45-120

RETAMOSO LLAMAS, Alonso de Jesús. Procesamiento Digital de Señales: Diseño y construcción de una tarjeta de propósito general, Bucaramanga, 2005.p. 112-135

TOMARAKOS, John, AD1847 SoundPort Codec-DSP Interface Training, Segunda edición, Analog Devices, 1998.p. 2-31

10 ANEXOS

Anexo A. Código fuente laboratorio ALU
Anexo B. Código fuente laboratorio MAC
Anexo C. Código fuente laboratorio SHIFTER
Anexo D. Código fuente laboratorio SECUENCIADOR
Anexo E. Código fuente laboratorio MULTIFUNCIÓN
Anexo F. Código fuente laboratorio Interrupción externa IRQE
Anexo G. Código fuente laboratorio Generación de señal Rampa DAG
Anexo H. Código fuente laboratorio Plantilla de trabajo TALKTHRU
Anexo I. Código fuente laboratorio Simulación FIR
Anexo J. Código fuente laboratorio Ejecución en tiempo real FIR
Anexo K. Código fuente laboratorio Simulación FIR en MatLab

Anexo A

```
// Se recomienda tener abierta la ventana de registros computacionales
// En el menu: Register -> Computational
// Avanzar en simulacion con F11 (Step Into)

// NOTA: Para utilizar el EzKit Lite, debe definirse la variable
__EZKIT_LICENSE_RESTRICTION_21xx__
// para eso, incluya la opcion -D__EZKIT_LICENSE_RESTRICTION_21xx__ en las
opciopnes del Linker

// Declaracion de la seccion de memoria donde se almacenara el vector de
interrucion por reset
.SECTION/PM IVreset;

      JUMP start;                     // Saltar al inicio del programa cuando se
presente un Reset

// Declaracion de la seccion de memoria donde se almacenara el programa
.SECTION/PM program;

start:                    // Declaracion de la entrada de programa

      AX0 = 0x0005;             // Inicializacion de variables
      AY0 = 0x0003;
      AR = AX0 + AY0;
      AF = AX0 + 3;
      //AR = AYO + 3;                 //Asignacion de laboratorio

      AR = AX0 - AY0;
      AR = AY0 - AX0;
      AR = AX0 - 3;
      //AR = AY0 - 3;                 //Asignacion de laboratorio

      // Operaciones logicas
      AX0 = 3;
      AY0 = 7;
      AR = AX0 AND AY0;
      AR = AX0 OR AY0;
      AR = AX0 XOR AY0;

      //AR = AY0 AND AX0;             //Asignacion de laboratorio
      //AR = AY0 OR AX0;              //Asignacion de laboratorio
      //AR = AY0 XOR AX0;             //Asignacion de laboratorio

      // Asignacion con PASS (operacion de la ALU, genera banderas)
      AR = PASS AX0;
      AR = PASS AY0;
      AR = PASS 5;

      // Valor absoluto
      AX1 = -8;
      AR = ABS AX1;

      // Operacion NEG (cambio de signo, es operacion aritmetica)
      AX0 = 0x5555;
      AR = -AX0;

      // Operacion NOT (inversion logica, es operacion logica)
      AR = NOT AX0;
```

```
// Incremento y Decremento
AF = AY0 + 1;
AR = AF - 1;

IDLE;        //Ciclo de espera de baja potencia de siguiente instruccion
```

Anexo B

```
// Se recomienda tener abierta la ventana de registros computacionales
// En el menu: Register -> Computational
// Se recomienda tener abierta la ventana de registros de estado
// En el menu: Register -> Status
// Avanzar en simulacion con F11 (Step Into)

// NOTA: Para utilizar el EzKit Lite, debe definirse la variable
__EZKIT_LICENSE_RESTRICTION_21xx__
// para eso, incluya la opcion -D__EZKIT_LICENSE_RESTRICTION_21xx__ en las
opciopnes del Linker

// Declaracion de la seccion de memoria donde se almacenara el vector de
interrucion por reset
.SECTION/PM IVreset;


JUMP start;         // Saltar al inicio del programa cuando se presente un Reset

// Declaracion de la seccion de memoria donde se almacenara el programa
.SECTION/PM program;

start:                                  // Declaracion de la entrada de programa

      // Multiplicacion por defecto en la MAC (Modo fraccional)
    MX0 = 10;
    MY0 = 10;
    MR = MX0 * MY0 (ss);
    //MR = MY0 * MY0 (ss);              //Asignacion de laboratorio

    //Multiplicacion de numeros fraccionales en la MAC

    //MX0 = 0.5;                                //Asignacion de laboratorio

    MX0 = -0.5r;
    MY0 = 0.5r;
    MR = MX0 * MY0 (SS);

    // Multiplicacion de modo entero en la MAC
    ENA M_MODE;
    MX0 = 10;
    MY0 = 10;
    MR = MX0 * MY0 (SS);
    MR = MX0 * MX0 (SS);
    DIS M_MODE;                 // Volver a modo fraccional.

      // Experimentar la saturacion de la MAC. Mirar MV y MR2
      ENA M_MODE;
      MR1 = 0x7FFF;
      MX0 = 0x00FA;
      MY0 = 0x00FB;
      MR = MR + MX0 * MY0 (SS);
      MR = MR + MX0 * MY0 (SS);

      // Si se presento overflow, saturar la MAC
      IF MV SAT MR;

      IDLE;
```

Anexo C

```
// Se recomienda tener abierta la ventana de registros computacionales
// En el menu: Register -> Computational
// Se recomienda tener abierta la ventana de registros de estado
// En el menu: Register -> Status
// Avanzar en simulacion con F11 (Step Into)

// NOTA: Para utilizar el EzKit Lite, debe definirse la variable
__EZKIT_LICENSE_RESTRICTION_21xx__
// para eso, incluya la opcion -D__EZKIT_LICENSE_RESTRICTION_21xx__ en las
opciopnes del Linker

// Declaracion de la seccion de memoria donde se almacenara el vector de
interrucion por reset
.SECTION/PM IVreset;
      JUMP start;                       // Saltar al inicio del programa cuando se
presente un Reset

// Declaracion de la seccion de memoria donde se almacenara el programa
.SECTION/PM program;

start:                                  // Declaracion de la entrada de programa

      SI = 0x9999;                      // Inicializacion del shifter
      SR = LSHIFT SI BY 2(HI);  // Ejecucion de shift, HI
      SR = LSHIFT SI BY 2(LO);  // Ejecucion de shift, LO

      SR = ASHIFT SI BY 2 (HI); // Ejecucion de shift, HI
      SR = ASHIFT SI BY 2 (LO); // Ejecucion de shift, LO

      // Comparar shift aritmetico con shift logico
      SI = 0xC000;
      SR = ASHIFT SI BY -8 (HI);// Mirar resultado como Signed Integer y Binary
      SR = LSHIFT SI BY -8 (HI);// Mirar resultado como Hexadecimal y Binary

      // Shift hacia la derecha (negativo)
      SR = LSHIFT SI BY -2 (HI);
      SR = SR OR LSHIFT SI BY -8 (HI);
      SR = SR OR ASHIFT SI BY -8 (HI);

      // Utilizacion del exponente
      SE = -8;                                  // Inicializacion del exponente
      SI = 0xFF00;                      // Inicializacion de registro
      SR = LSHIFT SI (LO);
      SR = ASHIFT SI (LO);

      IDLE;
```

Anexo D

```
// Se recomienda tener abierta la ventana de registros computacionales
// En el menu: Register -> Computational
// Se recomienda tener abierta la ventana de registros de estado
// En el menu: Register -> Status
// Avanzar en simulacion con F11 (Step Into)

// NOTA: Para utilizar el EzKit Lite, debe definirse la variable
__EZKIT_LICENSE_RESTRICTION_21xx__
// para eso, incluya la opcion -D__EZKIT_LICENSE_RESTRICTION_21xx__ en las
opciopnes del Linker

// Declaracion de la seccion de memoria donde se almacenara el vector de
interrucion por reset
.SECTION/PM IVreset;
      JUMP start;                    // Saltar al inicio del programa cuando se
presente un Reset

// Declaracion de la seccion de memoria donde se almacenara el programa
.SECTION/PM program;

start:                               // Declaracion de la entrada de programa
      // Ilustracion del JUMP
      AX0 = 8;
      JUMP Etiqueta_Uno;

Etiqueta_Dos:

      AX0 = 5;
      CNTR = AX0;
      AR = 0;

      DO Etiqueta_Tres UNTIL CE;
Etiqueta_Tres: AR = AR + 2;
      IDLE;
      NOP;


Etiqueta_Uno:// Ilustracion del CALL

      CALL Rutina_Uno;
      JUMP Etiqueta_Dos;

Rutina_Uno:

      SE = AX0;
      SI = 0x8888;
      SR = LSHIFT SI (HI);
      RTS;                     // Ojo: las subrutinas (llamadas con CALL), deben
retornar con RTS
```

Anexo E

```
// Se recomienda tener abierta la ventana de registros computacionales
// En el menu: Register -> Computational
// Se recomienda tener abierta la ventana de registros de estado
// En el menu: Register -> Status
// Avanzar en simulacion con F11 (Step Into)

// NOTA: Para utilizar el EzKit Lite, debe definirse la variable
__EZKIT_LICENSE_RESTRICTION_21xx__
// para eso, incluya la opcion -D__EZKIT_LICENSE_RESTRICTION_21xx__ en las
opciopnes del Linker
#define      test   11
#define      test2  10

.SECTION/DM  data1;
.VAR/CIRC    Prueba[test];

.SECTION/PM  data2;
.VAR/CIRC    Datos[test2] = "Datos.dat";

// Declaracion de la seccion de memoria donde se almacenara el vector de
interrucion por reset
.SECTION/PM IVreset;
      JUMP start;                    // Saltar al inicio del programa cuando se
presente un Reset

// Declaracion de la seccion de memoria donde se almacenara el programa
.SECTION/PM program;

start:

      AX1 = 0xA0A0;
      AY1 = 0;
      AX0 = 0x0002;
      AY0 = 0x0002;
      AR = AX0 + AY0 , AY1 = AX1;

      I2 = Prueba;
      M2 = 1;
      L2 = LENGTH(Prueba);
      AX0 = 15;
      CNTR = 11;
      DO Llenado UNTIL CE;
Llenado:           DM(I2,M2) = AX0;

      ENA M_MODE;
      AX0 = 0;
      MX0 = 0x0002;
      MY0 = 0x0002;
      MR = MX0 * MY0 (SS), AX0 = DM(I2,M2);
      DIS M_MODE;


      AX0 = 0xAAAA;
      I2 = 2;
      SE = -1;
      SI = 0x9999;
      DM(I2,M2) = AX0 , SR = LSHIFT SI(LO);
```

```
AY0 = 0;
I4 = Datos;
M4 = 1;
L4 = LENGTH(Datos);
AX0 = DM(I2,M2) , MY0 = PM(I4,M4);

I2 = 3;
AX0 = 5;
AY0 = 4;
AR = AX0 + AY0 ;AX0 = DM(I2,M2) , MY0 = PM(I4,M4);

IDLE;
NOP;
```

Anexo F

```
// NOTA: Para utilizar el EzKit Lite, debe definirse la variable __EZKIT_LICENSE_RESTRICTION_21xx__
// para eso, incluya la opcion -D__EZKIT_LICENSE_RESTRICTION_21xx__ en las opciopnes del Linker


// Declaracion de la seccion de memoria donde se almacenara el vector de interrucion por reset
// Saltar al inicio del programa cuando se presente un Reset
.SECTION/PM IVreset;
      JUMP start;

// Declaracion de la seccion de memoria donde se almacenara el vector de interrucion por edge (pushbutton)
// Saltar al inicio del programa cuando se presiona el pulsador
.SECTION/PM IVirqe;
      JUMP isrIrqE;

// Declaracion de la seccion de memoria donde se almacenara el vector de interrucion por SPORT1 Tx (Software UART)
// Saltar al inicio del programa cuando se presenta la interrupcion
.SECTION/PM IVirq1;
      JUMP isrIrq1;

// Declaracion de la seccion de memoria donde se almacenara el programa
// Declaracion de la entrada de programa
.SECTION/PM program;

start:

      // Habilitacion de IRQE
      AX0 = IMASK;
      AY0 = 0x0010;
      AR = AX0 OR AY0;
      IMASK = AR;
      jump Espera;

      // Rutina de atencion a la interrupcion por flanco (IRQE)
isrIrqE:
      TOGGLE FL1;
      RTI;

      // Ciclo permanente
Espera:
      IDLE;
      NOP;
      NOP;
      JUMP Espera;


// El siguiente fragmento de codigo siempre existir si se utiliza el EzKit
isrIrq1:
      POP STS;;
      ENA TIMER;              /* start timer now */
      RTS;                    /* note rts */
```

Anexo G

```
// NOTA: Para utilizar el EzKit Lite, debe definirse la variable __EZKIT_LICENSE_RESTRICTION_21xx__
// para eso, incluya la opcion -D__EZKIT_LICENSE_RESTRICTION_21xx__ en las opciopnes del Linker

#define             SIZE   1024         // Declaracion de tamano de bufer
#define             TOPE   128                // Declaracion del tope
//#define           TOPE   512
//#define           SIZE   4096


// Declaracion de la seccion de memoria donde se almacenaran las variables
.SECTION/DATA data1;
.VAR data_dm[SIZE];                     // Declaracion de variable

// Declaracion de la seccion de memoria donde se almacenara el vector de interrucion por reset
.SECTION/PM IVreset;
      JUMP start;                       // Saltar al inicio del programa cuando se presente un Reset

// El siguiente fragmento de codigo siempre existir si se utiliza el EzKit
// Declaracion de la seccion de memoria donde se almacenara el vector de interrucion por SPORT1 Tx (Software UART)
.SECTION/PM IVirq1;
      JUMP isrIrq1;             // Saltar al inicio del programa cuando se presenta la interrupcion

// Declaracion de la seccion de memoria donde se almacenara el programa
.SECTION/PM program;

start:                                  // Declaracion de la entrada de programa

      // Clareado de los buferes
      I0 = data_dm;
      M0 = 1;
      L0 = length(data_dm);
      AX0 = 0x0000;
      CNTR = L0;
      DO BorrarBuffer UNTIL CE;
BorrarBuffer:DM(I0,M0) = AX0;

      // Ciclo permanente
Espera:
      AR = 0x0000;              // Inicializar datos
      AX0 = TOPE;
      AY0 = 0x0000;
      AF = PASS AR;
      CNTR = SIZE - 1;
      DO Rampa UNTIL CE;
            //AR = PASS AF;
            DM(I0,M0) = AR,     AF = AF + 1;
            AR = AX0 - AF;
Rampa: IF EQ AF = PASS AY0;
      AR = PASS AF;       // Guardar ultima iteracion
      DM(I0,M0) = AR;

      TOGGLE FL2;               // Mostrar fin de ejecuciones
```

```
      JUMP Espera;


// El siguiente fragmento de codigo siempre existir si se utiliza el EzKit
/*****************************************************************************
 *  A high to low transition on flag_in signifies the start bit; it also
 *  triggers IRQ1 ISR which then turn on timer if the timer is off.  This is
 *  at to most 1/3 bit period too late but we shoudl still catch the byte.
 *****************************************************************************/
isrIrq1:
      POP STS;;
    ENA TIMER;              /* start timer now */
      RTS;                    /* note rts */
```

Anexo H

```
// NOTA: Para utilizar el EzKit Lite, debe definirse la variable
__EZKIT_LICENSE_RESTRICTION_21xx__
// para eso, incluya la opcion -D__EZKIT_LICENSE_RESTRICTION_21xx__ en las
opciopnes del Linker

#include <def2181.h>

// Declaracion de la seccion de memoria donde se almacenaran las variables
.SECTION/DM data1;
.VAR stat_flag;
.VAR/CIRC rx_buf[3];                                              //    Bufer    de
recepcion de datos del SPORT (proveniente del CODEC)
.VAR/CIRC tx_buf[3] = 0xc000, 0x0000, 0x0000; // Bufer de transmision de datos al
SPORT (hacia el CODEC)
.VAR/CIRC init_cmds[13] =                                    //     Comandos     de
inicializacion del CODEC
      0xc000,         /*    CLOR  set,  MCE  set,  Index  reg  address=0,  data=0x2
11000000 00 00 1100
                     *  Left input control reg
                     *  b7-6: 0=left line 1
                     *        1=left aux 1
                     *        2=left line 2
                     *        3=left line 1 post-mixed loopback
                     *  b5-4: res
                     *  b3-0: left input gain x 1.5 dB             */


      0xc100,      /*  CLOR set, MCE set, Index reg address=1, data=0x2
                     *  Right input control reg
                     *  b7-6:  0=right line 1
                     *         1=right aux 1
                     *         2=right line 2
                     *         3=right line 1 post-mixed loopback
                     *  b5-4:  res
                     *  b3-0:  right input gain x 1.5 db    */


      0xc280,      /*  CLOR set, MCE set, Index reg address=2, data=0x88   1000
                     *   Left aux 1 control reg
                     *   b7  :    1=left aux 1 mute
                     *   b6-5:    res
                     *   b4-0:    gain/atten x 1.5, 08= 0dB, 00= 12dB    */

      0xc388,      /* CLOR set, MCE set, Index reg address=3, data=0x88
                     *   Right aux 1 control reg
                     *   b7  :   1=right aux 1 mute
                     *   b6-5:   res
                     *   b4-0:   gain/atten x 1.5, 08= 0dB, 00= 12dB     */

      0xc488,      /* CLOR set, MCE set, Index reg address=4, data=0x88
                     *   left aux 2 control reg
                     *   b7  :   1=left aux 2 mute
                     *   b6-5:   res
                     *   b4-0:   gain/atten x 1.5, 08= 0dB, 00= 12dB     */

      0xc588,      /* CLOR set, MCE set, Index reg address=5, data=0x88
                     *   right aux 2 control reg
                     *   b7  :   1=right aux 2 mute
```

```
                 *   b6-5:   res
                 *   b4-0:   gain/atten x 1.5, 08= 0dB, 00= 12dB     */

       0xc680,       /* CLOR set, MCE set, Index reg address=6, data=0x80    1 0
000000
                 *   left DAC control reg
                 *   b7  :   1=left DAC mute
                 *   b6  :   res
                 *   b5-0:   attenuation x 1.5 dB       0=0 dB attenuation  */

       0xc780,     /* CLOR set, MCE set, Index reg address=7, data=0x80
                 *   right DAC control reg
                 *   b7  :   1 = right DAC mute
                 *   b6  :   res
                 *   b5-0:   attenuation x 1.5 dB       0=0 dB attenuation  */

       0xc85c,     /* CLOR set, MCE set, Index reg address=8, data=0x5c
                 *   data format register
                 *   b7  :   res
                 *   b5-6:   0 = 8-bit unsigned linear PCM
                 *           1 = 8-bit u-law companded
                 *           2 = 16-bit signed linear PCM
                 *           3 = 8-bit A-law companded
                 *   b4  :   0 = mono, 1 =  stereo
                 *   b0-3:   0 =   8.00000 Khz
                 *           1 =   5.51250 Khz
                 *           2 = 16.00000 Khz
                 *           3 = 11.02500 Khz
                 *           4 = 27.42857 Khz
                 *           5 = 18.90000 Khz
                 *           6 = 32.00000 Khz
                 *           7 = 22.05000 Khz
                 *           8 =   .
                 *           9 = 37.80000 Khz
                 *           a =   .
                 *           b = 44.10000 Khz
                 *           c = 48.00000 Khz
                 *           d = 33.07500 Khz
                 *           e =   9.60000 Khz
                 *           f =   6.61500 Khz
                 *  (b0) :   0 = XTAL1 24.576 Mhz; 1 = XTAL2 16.9344 Mhz    */

       0xc909,     /* CLOR set, MCE set, Index reg address=9, data=0x09
                 *   interface configuration reg
                 *   b7-4:   res
                 *   b3  :   1 = autocalibrate
                 *   b2-1:   res
                 *   b0  :   1 = playback enabled              */

       0xca00,     /* CLOR set, MCE set, Index reg address=0xa, data=0
                 *   pin control reg
                 *   b7  :   logic state of pin XCTL1
                 *   b6  :   logic state of pin XCTL0
                 *   b5  :   master - 1 = tri-state CLKOUT
                 *           slave  - x = tri-state CLKOUT
                 *   b4-0:   res                               */


       0xcc40,     /* CLOR set, MCE set, Index reg address=0xc, data=0x40
                 *   miscellaneous information reg
                 *   b7  :   1 = 16 slots per frame
```

```
                    *           0 = 32 slots per frame
                    *   b6  :   1 = 2-wire system
                    *           0 = 1-wire system
                    *   b5-0:   res                                 */


       0xcd00;     /* CLOR set, MCE set, Index reg address=0xd, data=0
                    *   digital mix control reg
                    *   b7-2:   attenuation x 1.5 dB
                    *   b1  :   res
                    *   b0  :   1 = digital mix enabled           */

// Declaracion de la seccion de memoria donde se almacenara el vector de
interrucion por reset
.SECTION/PM IVreset;
      JUMP start;                     // Saltar al inicio del programa cuando se
presente un Reset


// Declaracion de la seccion de memoria donde se almacenara el vector de
interrucion por edge (pushbutton)
.SECTION/PM IVirqe;
      JUMP isrIrqE;             // Saltar al inicio del programa cuando se
presiona el pulsador

// El siguiente fragmento de codigo siempre existir si se utiliza el EzKit
// Declaracion de la seccion de memoria donde se almacenara el vector de
interrucion por SPORT1 Tx (Software UART)
.SECTION/PM IVirq1;
      JUMP isrIrq1;             // Saltar al inicio del programa cuando se
presenta la interrupcion

// Declaracion de la seccion de memoria donde se almacenara el vector de
interrucion por SPORT0 Rx (CODEC)
.SECTION/PM IVsport0recv;
      JUMP input_samples;       // Saltar al inicio del programa cuando se
presenta la interrupcion

// Declaracion de la seccion de memoria donde se almacenara el vector de
interrucion por SPORT0 Tx (CODEC)
.SECTION/PM IVsport0xmit;
      AR = DM(stat_flag);
      AR = PASS AR;
      IF EQ RTI;
      JUMP next_cmd;                  // Saltar al inicio del programa cuando se
presenta la interrupcion

// Declaracion de la seccion de memoria donde se almacenara el programa
.SECTION/PM program;

start:                                // Declaracion de la entrada de programa

      //Inicializacion del ADSP-2181
      //Apagado del SPORT0  1000 0000 0000
      //Habulitacion del SPORT1 y configuracion del mismo como SPORT
      AX0 = 0x800;
      DM (Sys_Ctrl_Reg) = AX0;
      ENA TIMER;

      // Inicializacion de punteros para acceso al CODEC
      I0 = rx_buf;
```

```
	L0 = length(rx_buf);
	I1 = tx_buf;
	L1 = length(tx_buf);
	I3 = init_cmds;
	L3 = length(init_cmds);
	M1 = 1;
	M0 = 1;
	L5 = 0;

	// Configuracion SPORT0 (CODEC)
	AX0 = 0x287;
	DM (Sport0_Autobuf_Ctrl)  = AX0; //0010 1000 0111
	AX0 = 0;
	DM (Sport0_Rfsdiv) = AX0;
	AX0 = 0;
	DM (Sport0_Sclkdiv) = AX0;
	AX0 =0x860F;
	DM (Sport0_Ctrl_Reg) = AX0;
	AX0 =0x7;
	DM (Sport0_Tx_Words0) = AX0;
	AX0 =0x7;
	DM (Sport0_Tx_Words1) = AX0;
	AX0 =0x7;
	DM (Sport0_Rx_Words0) = AX0;
	AX0 =0x7;
	DM (Sport0_Rx_Words1) = AX0;

	// Configuracion de sistema
	AX0 = 0x1000;                    //ESTABA EN 1800, CON 1000 NO FUNCIONA
	DM (Sys_Crtl_Reg) = AX0;

	AX0 = 0xFFF;              //NO EXISTIA
	DM (Dm_Wait_Reg) = AX0;

	// Borrar interrupciones pendientes
	IFC = 0xFF;               //ESTABA EN FE
	NOP;

	ICNTL = 2;                        //ESTABA EN 2
	MSTAT = 0x60;             // Habilitar GO Mode y Timer estaba en 60

	// Inicializacion del CODEC
	AX0 = 0x0001;             // Inicializa banderas
	DM(stat_flag) = AX0;
	ENA INTS;                         // Habilita interrupciones
	IMASK = 0x0040;           // Habilita SPORT0 Tx y Timer    ESTABA EN 41
	AX0 = DM (I1, M1);        // Envia primer comando
	TX0 = AX0;

	// Chequear el envio de todos los comandos al CODEC
check_init:
	AX0 = DM (stat_flag);
	AF = PASS AX0;
	IF NE JUMP check_init;

	// Una vez inicializado, progreso de autocalibracion, la cual retorna como
bit 1 en la palabra de comando desde el CODEC
	AY0 = 2;
check_aci1:
	AX0 = DM (rx_buf);
	AR = AX0 AND AY0;
```

```
       IF EQ JUMP check_aci1;
       // Chequear que la autocalibracion este completa
check_aci2:
       AX0 = DM (rx_buf);
       AR = AX0 AND AY0;
       IF NE JUMP check_aci2;
       IDLE;

       // Cuando la autocalibracion esta completa, quitar el MUTE a los canales
DAC escribiendo a los reigstros respectivos
       AY0 = 0xBF3F;                    // Canal izquierdo
       AX0 = DM (init_cmds + 6);
       AR = AX0 AND AY0;
       DM (tx_buf) = AR;
       IDLE;                            // Esperar que la transmision termine
       AX0 = DM (init_cmds + 7);        // Canal derecho
       AR = AX0 AND AY0;
       DM (tx_buf) = AR;
       IDLE;                            // Esperar que la transmision termine

       // Borrar interrupciones pendientes
       IFC = 0xFF;               // ESTABA EN FE
       NOP;

       IMASK = 0x71;                     // Habilita SPORT0 Tx, Rx y Timer ESTABA
EN 61 ACA FUE EL PROBLEMA DE SIEMPRE PILAS ACA



       // Ciclo permanente, esperando interrupciones
Espera:

       NOP;
       NOP;
       NOP;
       NOP;
       NOP;
       NOP;
       NOP;
       NOP;
       NOP;
       NOP;
       NOP;

       JUMP Espera;


// El siguiente fragmento de codigo siempre existir si se utiliza el EzKit
/*************************************************************************
 *  A high to low transition on flag_in signifies the start bit; it also
 *  triggers IRQ1 ISR which then turn on timer if the timer is off.  This is
 *  at to most 1/3 bit period too late but we shoudl still catch the byte.
 *************************************************************************/

isrIrqE:
       TOGGLE FL1;                       // Conmutar el estado del led
       RTI;                              // Ojo: las interrupciones deben retornar
con RTI

isrIrq1:
```

```
      POP STS;;
    ENA TIMER;              /* start timer now */
      RTS;                    /* note rts */

// ISR para el envio del proximo comando al CODEC
next_cmd:
      ENA SEC_REG;
      AX0 = DM(I3, M1);
      DM (tx_buf) = AX0;
      AX0 = I3;
      AY0 = init_cmds;
      AR = AX0 - AY0;
      IF GT RTI;
      AX0 = 0xAF00;
      DM (tx_buf) = AX0;
      AX0 = 0;
      DM (stat_flag) = AX0;
      RTI;

// ISR para la recepcion ( y procesamiento) de las muestras recibidas
input_samples:
      //SET FL1;                                    // Encender LED
      ENA SEC_REG;                    // Utilizacion de registros secundarios
      MR0 = DM (rx_buf + 1);                  //  Obtencion  de  datos  del  canal
izquierdo    MR0
      MR1 = DM (rx_buf + 2);          // Obtencion de datos del canal derecho
            MR1

      DM (tx_buf + 1) = MR0;                  //   Envio   de   datos   del   canal
izquierdo                  MR0
      DM (tx_buf + 2) = MR1;           // Envio de datos del canal derecho
                  MR1
      //RESET FL0;                                  // Apagar LED
      RTI;
```

Anexo I

```
#define      BUF_SIZE     1000
#define      FIR_LEN            54


// Datos en DM
.SECTION/DM  data1;

.VAR/CIRC    Chan1_In[BUF_SIZE] = "prueba.dat";
.VAR/CIRC    Chan2_In[BUF_SIZE] = "prueba.dat";
.VAR/CIRC    Chan1_Out[BUF_SIZE];
.VAR/CIRC    Chan2_Out[BUF_SIZE];
.VAR/CIRC    Chan1_Dly[FIR_LEN];
.VAR/CIRC    Chan2_Dly[FIR_LEN];

// Datos en PM
.SECTION/PM  data2;

.VAR/CIRC    FirCoeffs_1[FIR_LEN] = "Coeffs.dat";
.VAR/CIRC    FirCoeffs_2[FIR_LEN] = "Coeffs.dat";

// Vector de interrupcion de reset
.SECTION/PM  IVreset;

       jump _main;

// Codigo de programa
.SECTION/PM  program;

_main:
       call Initialization;

       // Configuracion para procesamiento de Canal 1
       i0 = Chan1_In;
       m0 = 1;
       l0 = length(Chan1_In);
       i1 = Chan1_Dly;
       m1 = 1;
       l1 = length(Chan1_Dly);
       i2 = Chan1_Out;
       m2 = 1;
       l2 = length(Chan1_Out);
       i4 = FirCoeffs_1;
       m4 = 1;
       l4 = length(FirCoeffs_1);
       call FirProcess;

       // Configuracion para procesamiento de Canal 2
       i0 = Chan2_In;
       m0 = 1;
       l0 = length(Chan2_In);
       i1 = Chan2_Dly;
       m1 = 1;
       l1 = length(Chan2_Dly);
       i2 = Chan2_Out;
       m2 = 1;
       l2 = length(Chan2_Out);
       i4 = FirCoeffs_2;
       m4 = 1;
```

```
       l4 = length(FirCoeffs_2);
       call FirProcess;

       // Permanece en estado IDLE;
       idle;
       jump _main;

// Rutina de inicializacion del Buffer
Initialization:
       i0 = Chan1_Dly;
       m0 = 1;
       l0 = length(Chan1_Dly);
       i1 = Chan2_Dly;
       m1 = 1;
       l1 = length(Chan2_Dly);
       ax0 = 0;
       cntr = FIR_LEN;
       do Clean until ce;
              dm(i0,m0) = ax0;
Clean: dm(i1,m1) = ax0;
       rts;

// Procesamiento FIR
FirProcess:
       cntr = BUF_SIZE;
       do FirProcessLoop until ce;
              ax0 = dm(i0,m0);
              call FirFilter;
FirProcessLoop:
              dm (i2,m2) = mr1;
       rts;

// Rutina del filtro FIR
FirFilter:
       dm(i1,m1) = ax0;                                                    //
Almacena la muestra actual en la ultima posicion
       cntr = FIR_LEN - 1;                                                 //
Inicializa el lazo del contador
       mr = 0,      mx0 = dm(i1,m1),    my0 = pm(i4,m4);    //     Reinicia     el
acumulador, obtiene el primer dato y coeficiente
       do FirLoop until ce;

       FirLoop:     mr = mr+mx0*my0 (ss), mx0 = dm(i1,m1), my0 = pm(i4,m4);
       // Realiza la suma de productos
       mr = mr+mx0*my0 (rnd);
                                       // Realiza el ultimo calculo y redondea el
resultado
       if mv sat mr;
                                       // En caso de overflow sature MR
       rts;
```

Anexo J

```
#define      FIR_LEN_1          54
#define      FIR_LEN_2          54

// Funciones globales
.GLOBAL      InitializationFir;
.GLOBAL FirFilter_Ch1;
.GLOBAL FirFilter_Ch2;

// Datos en DM
.SECTION/DM  data1;

.VAR/CIRC    Chan1_Dly[FIR_LEN_1];
.VAR/CIRC    Chan2_Dly[FIR_LEN_2];

// Datos en PM
.SECTION/PM  data2;

.VAR/CIRC    FirCoeffs_1[FIR_LEN_1] = "Coeffs1.dat"; //1500
.VAR/CIRC    FirCoeffs_2[FIR_LEN_2] = "Coeffs1.dat";

//.VAR/CIRC  FirCoeffs_1[FIR_LEN_1] = "Coeffs2.dat"; //2000
//.VAR/CIRC  FirCoeffs_2[FIR_LEN_2] = "Coeffs2.dat";

//.VAR/CIRC  FirCoeffs_1[FIR_LEN_1] = "Coeffs3.dat"; //2500
//.VAR/CIRC  FirCoeffs_2[FIR_LEN_2] = "Coeffs3.dat";

// Codigo de programa
.SECTION/PM  program;

// Rutina de inicializacion de buffer
InitializationFir:
       // Inicializar canal 1
       i2 = Chan1_Dly;
       m2 = 1;
       l2 = length(Chan1_Dly);
       ax0 = 0;
       cntr = FIR_LEN_1;
       do Clean_1 until ce;
Clean_1:           dm(i2,m2) = ax0;

       // Inicializar canal 1
       i3 = Chan2_Dly;
       m3 = 1;
       l3 = length(Chan2_Dly);
       ax0 = 0;
       cntr = FIR_LEN_2;
       do Clean_2 until ce;
Clean_2:           dm(i3,m3) = ax0;

       // Punteros para el retardo y los coeficientes del canal 1
       i2 = Chan1_Dly;
       m2 = 1;
       l2 = length(Chan1_Dly);
       i4 = FirCoeffs_1;
       m4 = 1;
       l4 = length(FirCoeffs_1);

       // Punteros para el retardo y los coeficientes del canal 2
```

```
        i3 = Chan2_Dly;
        m3 = 1;
        l3 = length(Chan2_Dly);
        i5 = FirCoeffs_2;
        m5 = 1;
        l5 = length(FirCoeffs_2);

        rts;

InitializationFir.END:

// Fir filter rutine - Canal 1
// Entrada: AX0
// Salida: MR1
FirFilter_Ch1:
        dm(i2,m2) = ax0;
        cntr = FIR_LEN_1 - 1;
        mr = 0,       mx0 = dm(i2,m2),    my0 = pm(i4,m4);
        do FirLoop_Ch1 until ce;
FirLoop_Ch1:
              mr = mr+mx0*my0 (ss),      mx0 = dm(i2,m2),    my0 = pm(i4,m4);

        mr = mr+mx0*my0 (rnd);
        if mv sat mr;
        rts;

FirFilter_Ch1.END:

// Fir filter rutine - Canal 2
// Entrada: AX0
// Salida: MR1
FirFilter_Ch2:
        dm(i3,m3) = ax0;
        cntr = FIR_LEN_2 - 1;
        mr = 0,       mx0 = dm(i3,m3),    my0 = pm(i5,m5);
        do FirLoop_Ch2 until ce;
FirLoop_Ch2:
              mr = mr+mx0*my0 (ss),      mx0 = dm(i3,m3),    my0 = pm(i5,m5);

        mr = mr+mx0*my0 (rnd);
        if mv sat mr;
        rts;

FirFilter_Ch2.END:
```

Anexo K

```
% FirTest.m
%
% Exporta los coeficientes del filtro FIR y genera los datos de prueba
% Los coeficientes del filtro se encuentran en la variable 'Num'


fprintf(1, '\nExport FIR filter coefficients and generate test data\n');
fprintf(1, '-----------------------------------------------------\n\n');
fprintf(1, 'Los coeficientes del filtro se encuentran en la variable "Num"\n');
fprintf(1, 'Los coeficientes del filtro seran almacenados en un archivo llamado
"Coeffs.dat"\n\n')
fid = fopen('Coeffs.dat', 'w');
fprintf(fid, '%1.15fr\n\r', Num);
fclose(fid);

% Inputs
fprintf(1, 'Generar los datos de prueba\n');
fs = input('Ingrese la frecuencia de muestreo [Hz]: ');
N = input('Ingrese el número de muestras: ');
n = 0:N-1;

cont = 'y';
while (cont == 'Y' | cont == 'y')
    fc = input('Ingrese frecuencias senoidales en un vector [Hz]: ');
    % Generacion  de datos
    [n_m, f] = meshgrid(n,fc/fs);
    y = sin(2*pi*n_m.*f);
    y = sum(y,1)/length(fc);
    yf = filter(Num,1,y);

    % Ploteo de la señal de entrada y filtrada
    subplot(2,1,1), plot(n/fs, y),title('Señal 1');
    subplot(2,1,2), plot(n/fs, yf),title('Señal 2');
    % Ploteo del espectro
    figure, subplot(2,1,1), plotSpectr(y', N*4, fs, 2),title('Señal 3');
    subplot(2,1,2), plotSpectr(yf', N*4, fs, 2),title('Señal 4');

    % Exportar datos
    fileName = input('Ingrese el nombre del archivo para datos: ');
    fid = fopen(fileName, 'w');
    fprintf(fid, '%1.15fr\n\r', y);
    fclose(fid);

    % Limpieza de datos
    clear n_m;
    clear f;
    clear y;
    clear yf;

    cont = input('Desea generar otra señal? [Y/N]: ');
end
```