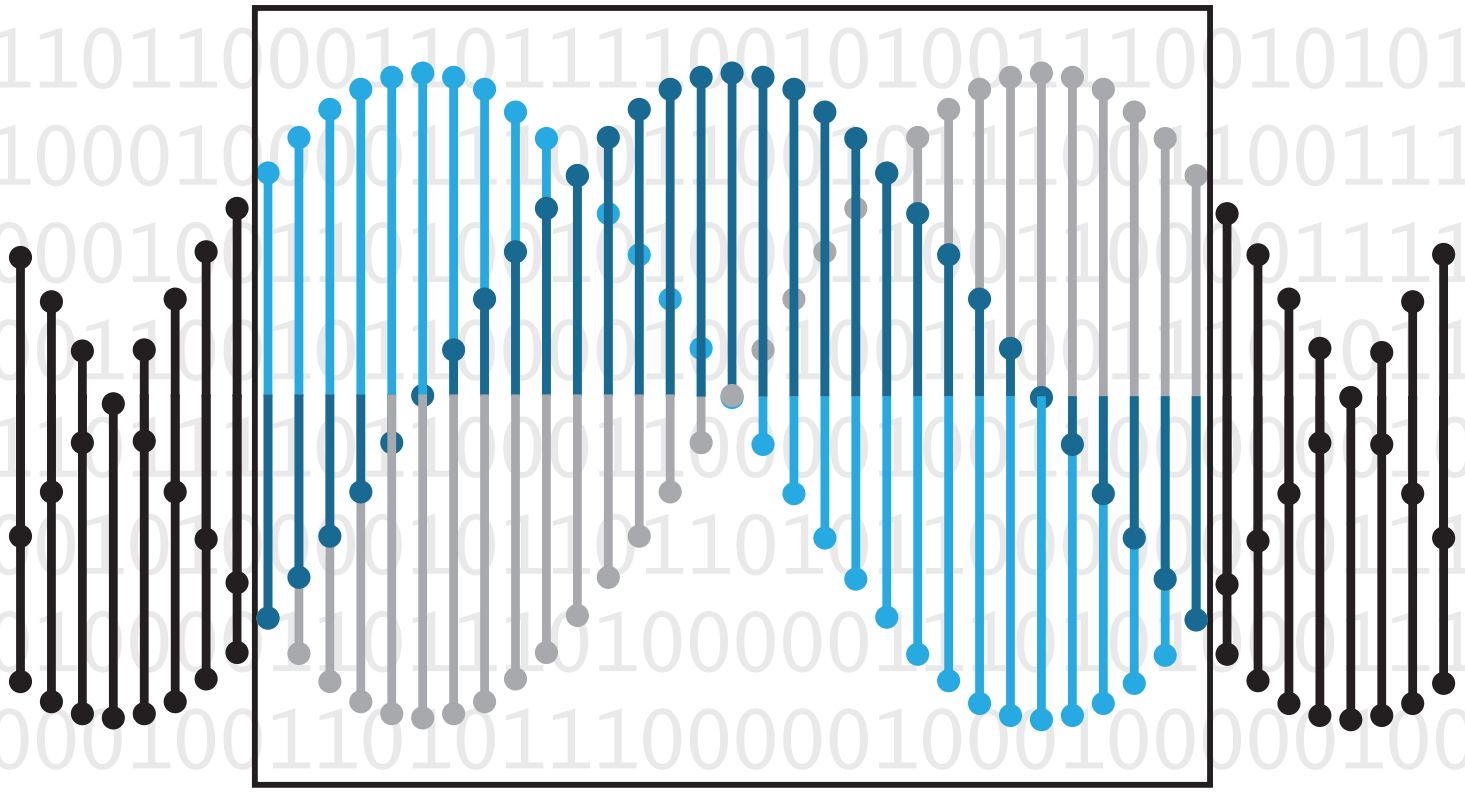
MANUAL DE ALGORITMOS Y APLICACIONES DE PROCESAMIENTO DIGITAL DE SEÑALES

Empleando la Familia **TMS320F2837xS**



LABORATORIO DE PROCESAMIENTO DIGITAL DE SEÑALES

Luis A. Álvarez Larry H. Escobar Miguel Flores

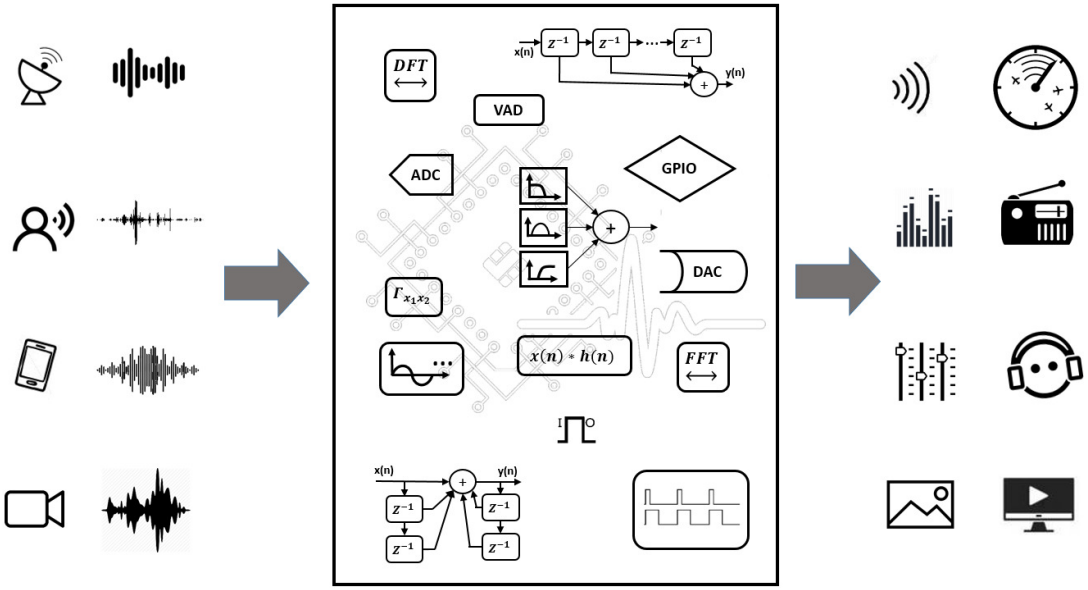
Carlos I. García Óscar I. Menendez Michel Olvera

MANUAL DE ALGORITMOS Y APLICACIONES DE PROCESAMIENTO DIGITAL DE SEÑALES

Álvarez, Escobar, Flores, García, Menendez, Olvera

Manual de Algoritmos y Aplicaciones de Procesamiento Digital de Señales

Empleando la Familia TMS320F2837xS



Luis A. Alvarez **Larry H. Escobar** **Miguel A. Flores**

Carlos I. García **Oscar I. Menendez** **Mauricio M. Olvera**

División de Ingeniería Eléctrica
Departamento de Procesamiento Digital de Señales

Universidad Nacional Autónoma de México (UNAM)

Facultad de Ingeniería

El desarrollo de este trabajo fue gracias al apoyo del proyecto PAPIME PE100616 con nombre “Servidor para prácticas de procesamiento digital de señales en tiempo real”

(2016 - 2017)

Ciudad Universitaria, FI, UNAM.
Ciudad de México, octubre 2018

Prólogo

En los últimos años el procesamiento digital de señales, abreviado como PDS, se ha convertido en un área de mucho interés para la investigación y la industria, esto gracias al avance tecnológico en los procesadores ya que la capacidad y velocidad de procesamiento ha aumentado considerablemente, permitiendo ejecutar importantes aplicaciones con respuesta en tiempo real.

Diferentes áreas tecnológicas están interesadas cada vez más en implementar procesamiento digital de señales en sus sistemas, dichos sistemas son utilizados en actividades de la vida moderna como: multimedia, computadoras personales, teléfonos celulares, comunicaciones vía Internet, telefonía, aplicaciones biomédicas, instrumentación, tabletas digitales, videojuegos y televisión digital, entre otras.

Todo procesamiento digital de señales requiere un hardware dedicado para poder ser implementado, los procesadores de señales digitales conocidos como DSPs por sus siglas en inglés, nacen con la idea de poder ejecutar de manera rápida las operaciones base del procesamiento digital de señales, tales como es la convolución y correlación. El diseño del hardware de los DSPs cada día mejora en función del desarrollo tecnológico, de tal manera que se han convertido en una herramienta de aplicación y soluciones a problemas del PDS.

Este manual está enfocado en poder ejecutar los algoritmos base y aplicaciones de procesamiento digital de señales en un hardware dedicado como es el caso de la familia C2000 de *Texas Instruments*, de la misma manera se expone la teoría básica de PDS para poder implementar los algoritmos.

El contenido de este manual se propone como un apoyo para los interesados en el área de procesamiento digital de señales utilizando DSPs, en donde se incluyen ejemplos elaborados en lenguaje ensamblador con diferentes formatos numéricos como punto fijo y punto flotante, con el objetivo de profundizar en el manejo del hardware para obtener la mejor precisión numérica. De la misma manera, estos dispositivos se puede programar en lenguaje C y C++, además de hacer una mezcla de lenguaje C con lenguaje ensamblador y así poder optimizar el código lo mejor posible.

Los ejemplos presentados van orientados al desarrollo de aplicaciones reales. La idea de los ejemplos es presentar la teoría de los algoritmos básicos del PDS y posteriormente implementar los algoritmos en el dispositivo, la implementación de los algoritmos se realiza en lenguaje ensamblador con formato numérico de punto fijo a 16 y 32 bits y posteriormente en punto flotante para analizar su diferencia en la precisión numérica.

Para las implementaciones presentadas en este manual, se seleccionó el microcontrolador de punto flotante de 32 bits **TMS320F28377S** [1, 2], porque cuenta con unidades de procesamiento adicionales al CPU C28x que pueden trabajar en conjunto, a 200 MHz para realizar operaciones de punto flotante, trigonométricas, de números complejos y relacionadas al uso de programación dinámica con el algoritmo de Viterbi. Adicionalmente cuenta con una unidad de procesamiento paralelo, periféricos para la adquisición de señales y puertos de comunicación serie, útiles en las aplicaciones de PDS, razones por las que en este manual, lo denominamos DSP.

Se recomienda que los interesados en este material tengan bases sólidas de procesamiento digital de señales, microcontroladores, diseño lógico y programación en lenguaje C y C++.

Organización del Manual

La organización del presente manual se realizó con el objetivo que el lector comprenda las bases para programación del dispositivo empleando algoritmos de procesamiento digital de señales y pueda emplear los conocimientos adquiridos a lo largo del mismo en los capítulos avanzados, por lo que está compuesto de la siguiente manera:

En el Capítulo 1 se explica de forma general la arquitectura del procesador digital de señales *TMS320F28377S* y define la tarjeta de evaluación *LAUNCHXL-F28377S* que contiene el DSC, indicando los puertos disponibles del procesador para su uso. El Capítulo 2 es una breve reseña de la interfaz de desarrollo *Code Composer Studio* (CCS), comenzando por la creación proyectos, exponiendo el imprescindible mapa de memoria y el formato de programas en lenguaje C y ensamblador. Además, para probar el funcionamiento de los programa,

como parte del entorno *Debug* se hace referencia a la visualización de datos en memoria, así como la estructura que deben tener los archivos para importar datos a la memoria del dispositivo, y las opciones con que se cuenta para analizar dichos datos de forma gráfica.

Los formatos numéricos son una pieza clave en cualquier aplicación del procesamiento digital de señales porque es la representación de los datos con punto decimal utilizando números enteros (punto fijo) o IEEE754 (para punto flotante). El Capítulo 3 presenta la conversión de datos a dichos formatos numéricos, además de su manejo al realizar operaciones aritméticas.

Presentadas las herramientas de trabajo, en el Capítulo 4 se exponen algoritmos y operaciones base del procesamiento digital de señales que son clave para el desarrollo de aplicaciones. En cada sección se describe brevemente la teoría del operador, algoritmo o método, y posteriormente se muestran las implementaciones propuestas, manejando diferentes formatos numéricos para analizar el desempeño y precisión.

En las aplicaciones, además de utilizar datos guardados en memoria, es importante la interacción del dispositivo con su entorno, por ejemplo la adquisición de señales por medio de sensores y transductores para ser procesadas en tiempo real. Dicha interacción se realiza por medio de los periféricos disponibles en el dispositivo. El capítulo 5 está dedicado a la configuración y manejo de los puertos disponibles en la tarjeta *LAUNCHXL-F28377S* tanto en lenguaje C, utilizando las bibliotecas de ControlSuite, como en lenguaje ensamblador.

Una de las últimas herramientas presentadas antes de las aplicaciones en tiempo real, es la capacidad del compilador C28x para desarrollar proyectos con un programa principal escrito en lenguaje C y con métodos o funciones por medio de lenguaje ensamblador. Esto le permite al usuario utilizar la arquitectura del DSP de forma específica para optimizar la implementación.

Posteriormente, en el Capítulo 6 se hace uso de la ejecución de un proyecto en el que se mezcla el lenguaje C con el lenguaje ensamblador para optimizar los recursos de la máquina. Tomando en cuenta que el código en lenguaje ensamblador se manda a llamar como una función desde el código principal en lenguaje C.

Finalmente, en el Capítulo 7 se combinan las operaciones y algoritmos con algunos periféricos y metodologías para implementar aplicaciones en tiempo real. Para ello, se diseñó un circuito impreso, como herramienta adicional, para adecuar las señales antes de adquirirlas por medio de la *LAUCHXL-F28377S* y de igual manera en los puertos de salida, de tal manera que se pueden adquirir dos señales de voz simultáneas por medio sus respectivos micrófonos y un canal tipo estéreo para adquisición de audio, además, cuenta con dos salidas de audio (una monoaural y otra estéreo), así como dos botones de entrada digital.

Los autores agradecemos a las autoridades de la Facultad de Ingeniería y a la Unidad de Apoyo Editorial por el soporte brindado para la publicación de este material.

Índice general

Capítulo 1

DSP TMS320F28377s y la Lauchpad Delfino

La capacidad y velocidad de procesamiento mejora cada día gracias a los avances en hardware, ésto ha permitido diseñar diferentes tipos de arquitecturas de procesadores para realizar tareas específicas. Para el procesamiento digital de señales, existen arquitecturas dedicadas a la implementación de operaciones del área de estudio, denominados DSP (*Digital Signal Processor*). Además, también existen microcontroladores (MCU) con unidades centrales de proceso (CPU) capaces de operar como un DSP, con la ventaja de disponer de puertos de comunicación y adquisición de señales.

Existen diferentes compañías de semiconductores que fabrican dispositivos para diversas aplicaciones, como los DSP. Sin embargo, pocas se han mantenido en constante desarrollo, innovación y demanda en el mercado. *Texas Instruments* (TI) es un ejemplo de ello.

TI es una compañía estadounidense fundada en 1951 que ha tenido mucho éxito a nivel comercial, reflejado en su amplio catálogo de dispositivos e información pública, necesaria para el desarrollo de proyectos. Además, cuenta con tarjetas de desarrollo enfocadas al aprendizaje y de módulos de hardware de conexiones inalámbricas, control de motores DC, Ethernet, por mencionar algunos.

Para el desarrollo de las operaciones, algoritmos y aplicaciones de PDS que se abordan en este manual, se eligió trabajar con la tarjeta LaunchXL-F28377S de TI. Esta tarjeta de desarrollo contiene el DSP TMS320F28377S con capacidades de procesamiento similares a las de un DSP, gracias a su CPU C28x. Cuenta con 1024KB de memoria flash, 15 canales ePWM, dos convertidores analógico digital de 12 bits y diversos protocolos de comunicación como I2C, SPI, UART entre otros. Adicionalmente, cuenta con un emulador JTAG XDS100 para cargar programas y sesiones de depuración desde una PC con el software Code Composer Studio [3].

Antes de comenzar a trabajar con los conceptos y el software de desarrollo, en este capítulo se describe brevemente la arquitectura y características del TMS320F28377S, así como la información necesaria para un manejo adecuado y uso de la LaunchXL-F28377S.

1.1. Características del TMS320F28377

La familia de MCU C2000, de 32 bits está optimizada para procesar, sensar y accionar actuadores, enfocada en mejorar el rendimiento de aplicaciones embebidas. El conjunto se divide en cuatro series de dispositivos, siendo de particular interés la Delfino, por su velocidad de procesamiento en formato de punto flotante simple.

El núcleo de toda la familia es el CPU C28x, el cual está acoplado a buses optimizados de datos e interrupciones. El C28x es una combinación de un MCU y un DSP, por su velocidad y capacidad de proceso también se conoce como un controlador digital de señales (DSC), sin embargo, en este trabajo lo nombraremos DSP.

Además el TMS320F28377S, cuenta con unidades dedicadas que le permiten hacer tareas en paralelo dentro del mismo CPU respetando los ciclos de máquina involucrados en cada instrucción. Las unidades de proceso adicionales que forman parte del CPU del DSP de trabajo son las siguientes:

- **Unidad de punto flotante (FPU):**
  Extiende las capacidades del CPU para procesamiento de datos de punto fijo agregando registros e instrucciones de punto flotante con precisión simple, utilizando el estándar 754 IEEE. Esta unidad agrega los siguientes registros de unidad de punto flotante:

  1. Ocho registros de resultados de punto flotante (RnH).
  2. Registro de estado de punto flotante (STF).
  3. Registro de bloqueo de repetición (RB).

- **Unidad matemática trigonométrica (TMU):**
  Extiende las capacidades del F28x agregando instrucciones y aprovechando las instrucciones existentes de la FPU para acelerar las operaciones trigonométricas básicas.

- **Unidad de Viterbi, complejos y CRC (VCU):**
  La unidad de Viterbi (VCU) agrega registros e instrucciones para acelerar el rendimiento de la Transformada discreta de Fourier (FFT) y algoritmos basados en comunicaciones.

De tal manera que el CPU está compuesto por C28x + FPU + TMU + VCU. Algunas características de esta arquitecturas son:

- Arquitectura tipo Harvard modificada.
- Unidad aritmética lógica (ALU) de 32 bits.
- Efectúa una operación multiplicación acumulación (MAC) de 32 x 32 bits en un ciclo de reloj.
- Efectúa dos operaciones MAC de 16 x 16 bits (DMAC) en un ciclo de reloj.
- Protección de código y registros protegidos.
- En un ciclo de instrucción puede ejecutar instrucciones que leen, modifican y escriben en memoria.
- Respuesta de interrupciones rápida con salvado automático del contexto.
- Sincronía de eventos con latencia mínima.
- Pipeline de 8 niveles, que permiten un solapamiento máximo de 8 instrucciones en niveles de ejecución diferentes:
  - Búsqueda de instrucción: F1 y F2.
  - Decodificación: D1 y D2.
  - Lectura de operandos: R1 y R2.
  - Ejecución: X.
  - Escritura: W.

Así mismo, este procesador cuenta con un gran número de periféricos, entre los que se encuentran:

- Convertidores Analógico-Digital (12/16 bits).
- Convertidores Digital-Analógico (12 bits).
- ePWM (PWM de alta resolución).
- GPIOS
- SCI, DMA, USB e I2C.

Adicionalmente, los DSPs F2837xS también cuentan con otra unidad de procesamiento en paralelo denominada CLA (*Control Law Accelerator*). Esta unidad es de 32 bits y maneja aritmética de punto flotante, además de tener acceso a los canales de comunicación y periféricos. También cuenta con un conjunto de instrucciones propio, las cuales operan a la velocidad de la máquina mejorando la velocidad de procesamiento en tiempo real de una aplicación [2].

En la Figura 1.1 se muestra el diagrama de bloques del procesador F2837xS con sus periféricos asociados.

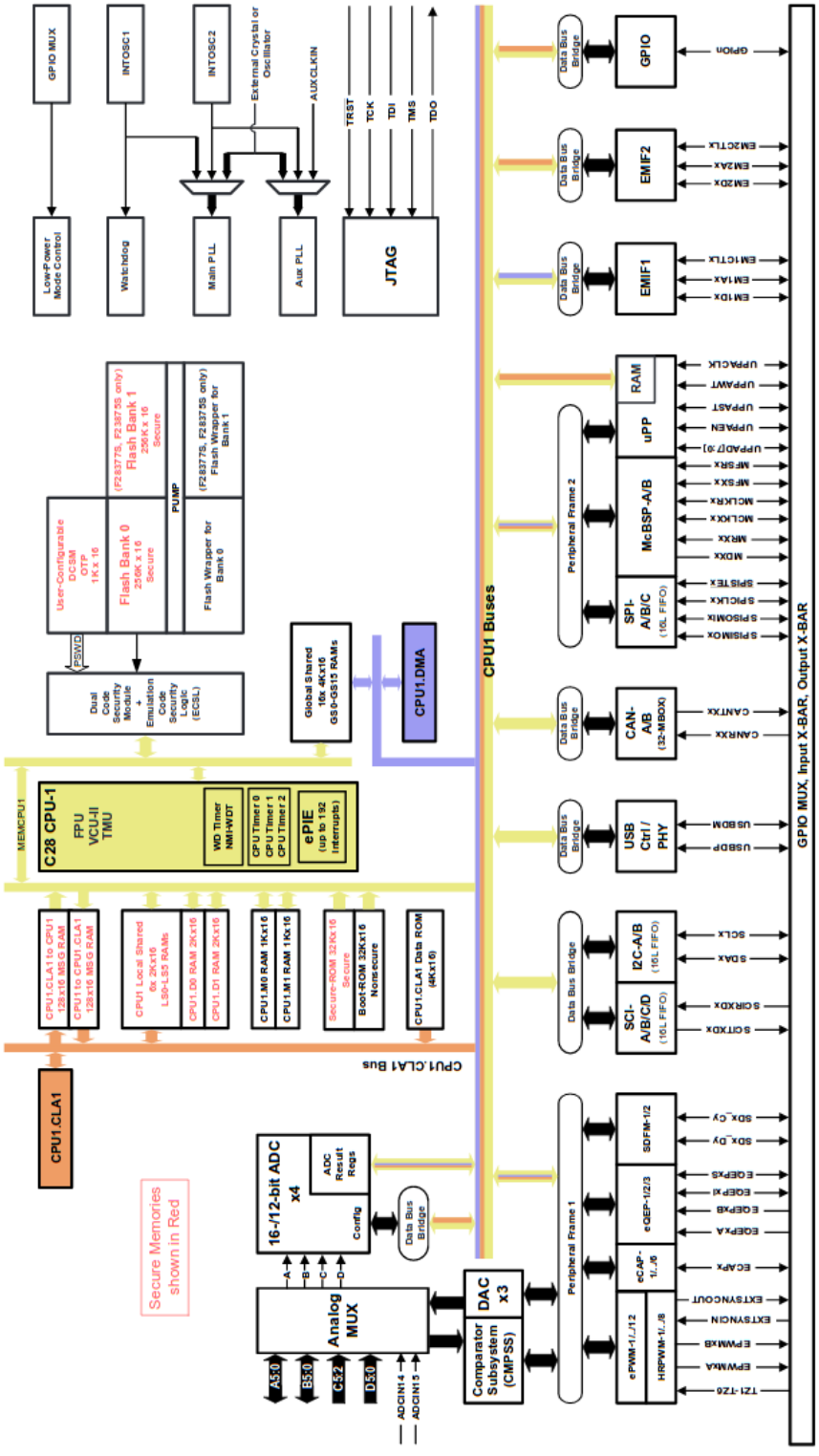


Figura 1.1: Diagrama a bloques del TMS320F28377s [4]

1.2. Descripción general de la tarjeta LAUNCHXL-F28377S

Para familiarizarse con esta familia de DSPs, *Texas Instruments* construyó una tarjeta de desarrollo denominada LauchPad Delfino C2000, teniendo dos opciones de dispositivo a escoger, TMS320F28377S o el TMS320F28379D, siendo esta última de dos núcleos, manteniendo la misma arquitectura.

Ambos LauchPad cuentan con una interfaz JTAG XDS100 para poder ejecutar instrucción por instrucción en tiempo real, ésto con efectos de depuración o análisis de código.

El desarrollo de este trabajo se realizó utilizando la tarjeta LaunchXL-F28377S, por ello a continuación se describen los puntos más relevantes de esta plataforma.

Alimentación de la tarjeta

La tarjeta LauchXL-F28377S funciona con un voltaje de 3.3V/1A DC. La principal fuente de energía de la tarjeta es por medio de la interfaz JTAG, la cual regula la diferencia de potencial suministrada por el puerto mini USB. Al estar aislado el hardware correspondiente al emulador XD100, tambien es posible suministrar los 3.3V por alguno de los conectores que dispone la tarjeta y señalados por la serigrafía de la tarjeta, por ejemplo, en el conector J10.

Por lo anterior, acorde al uso de la tarjeta, se cuenta con cuatro puntos de conexión/aislamiento mostrados en la Tabla 1.1 [3].

Tabla 1.1: Configuración del paso de corriente de energía del JTAG.

Jumper	Configuración del paso de la energía
JP1	Alimentación de 3.3V desde el módulo de JTAG, proviene del puerto USB
JP2	Aislamiento de GND entre el módulo JTAG con la sección del MCU.
JP4	Conexión/Desconexión de 3.3V a la terminal de 3.3V del bloque J5.
JP5	Conexión/Desconexión de 5V a la terminal de 5V del bloque J7.

Modos de arranque

Un programa necesita ser cargado a la memoria del dispositivo antes de que éste pueda ser ejecutado. El proceso de carga es la preparación de un programa para su ejecución inicializando la memoria del dispositivo con el código del programa y datos.

Un programa puede ser cargado por alguna de las siguientes formas:

- **Un depurador (*debugger*) conectado a una estación de trabajo:** El dispositivo está subordinado a un huésped que ejecuta un depurador como es el *Code Composer Studio* (CCS). El dispositivo es conectado por medio de un canal de comunicación como es el caso de la interfaz JTAG. CCS lee el programa y escribe la imagen de carga directamente a la memoria de la tarjeta a través de la comunicación de la interfaz.

- **Otro programa corriendo en el dispositivo:** Un programa que se está ejecutando puede crear la imagen de carga y transferir el control al programa de carga. Si está presente un sistema operativo, éste puede tener la habilidad de cargar y correr programas.

- **Grabar la imagen de carga en un módulo EPROM:** La EPROM se localiza dentro del mismo dispositivo y llega a ser una parte de la memoria del dispositivo, de tal manera que el convertido *(*hex2000) puede asistir con éste el archivo del objeto ejecutable en un formato adecuado para la entrada a un programador EPROM.

- **Carga *Bootstrap* desde un periférico dedicado, tal como un I2C:** El dispositivo puede requerir un pequeño programa llamado gestor de arranque (*Bootloader*) para realizar la carga desde el periférico.

Para sistemas embebidos, la imagen de carga de un programa es grabado en una EPROM/ROM. Los datos variables en el programa se deben poder escribir, de tal manera que se ubican en una memoria regrabable, generalmente RAM. Sin embargo, RAM es una memoria volátil , lo que significa que su contenido se pierde cada vez que se desenergiza el dispositivo.

La LaunchPad F28377s incluye una ROM de arranque, la cual permite al dispositivo realizar pruebas de arranque y de la misma manera tiene diferentes formas de arranque. Para cambiar la configuración de arranque, la LaunchPad tiene un interruptor S1 y dependiendo de su posición es el arranque. En la Tabla 1.2 se muestran las diferentes configuraciones de arranque.

Tabla 1.2: Configuraciones de arranque por medio del interruptor S1.

Switch	**Función**
1	GPIO84
2	GPIO72
3	TRSTn

Interfaz Debug

La interfaz de depuración TI utiliza cinco señales (JTAG) estandarizadas IEEE 11449.1 que son: TRST, TCK, TMS, TDI y TDO; Además cuenta con dos extensiones TI (EMU0 y EMU1). En la Figura 1.2 se muestran los 14 pines del encabezado JTAG que utiliza para la interfaz en el dispositivo.

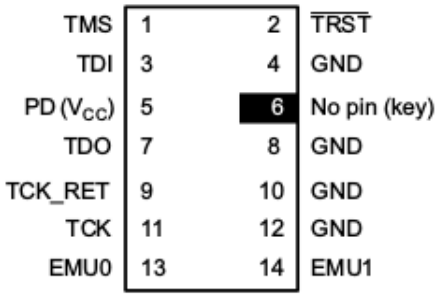


Figura 1.2: Interfaz JTAG

En la Tabla 1.3 se muestra la descripción de cada uno de los pines de la interfaz JTAG.

Tabla 1.3: Descripción de los pines de la interfaz JTAG

Señal	**Descripción**	**Estado del emulador**	**Estado de tarjeta**
EMU 0	Emulación pin 0	I	I/O
EMU 1	Emulación pin 1	I	I/O
GND	Tierra		
PD (Vcc)	Indica que el cable de emulación está conectado y tarjeta está encendido.	I	O
TCK	Es una fuente de reloj de emulación	O	I
TCK_RET	Prueba de entrada del reloj al emulador	I	O
TDI	Entrada de datos de prueba	O	I
TDO	Salida de datos de prueba	I	O
TMS	Selector de modo de prueba	O	I
TSTR	Reinicio de prueba	O	I

Distribución de puertos de conexión

La mayoría de las terminales del TMS320F28377S están disponibles y distribuidas en cuatro conjuntos de conectores principalmente. La posición de estos conectores tienen por

objetivo utilizar los diferentes puertos disponibles del DSP, de forma aislada o utilizando alguna de las tarjetas de expansión, fabricadas por TI y denominadas *Booster Packs*.

Los bloques de terminales de conexión están formados por dos pares señalados como J1-J3, J2-J4, J5-J7 y J6-J8.

En la Figura 1.3 se muestra la tarjeta LaunchPad donde se aprecian sus principales componentes, entre ellos: LEDs para uso del usuario (D2 y D3), headers para conexión de tarjetas de expansión (BoosterPacks) (J1-J3, J2-J4, J5-J7 y J6-J8). Cuenta con pines de alimentación externa (J10) y un botón para reiniciar el procesador (S3) [5].

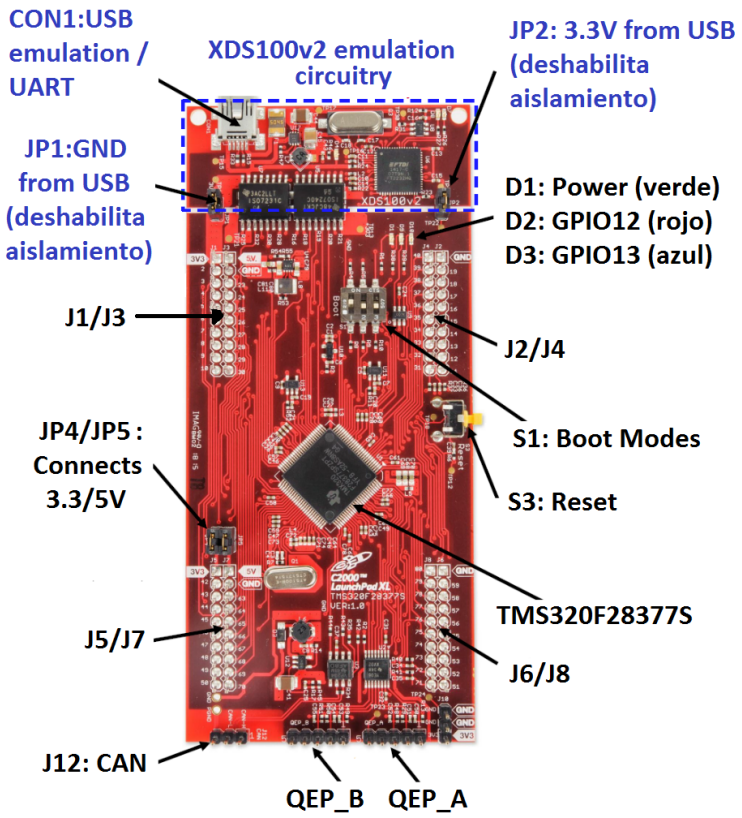
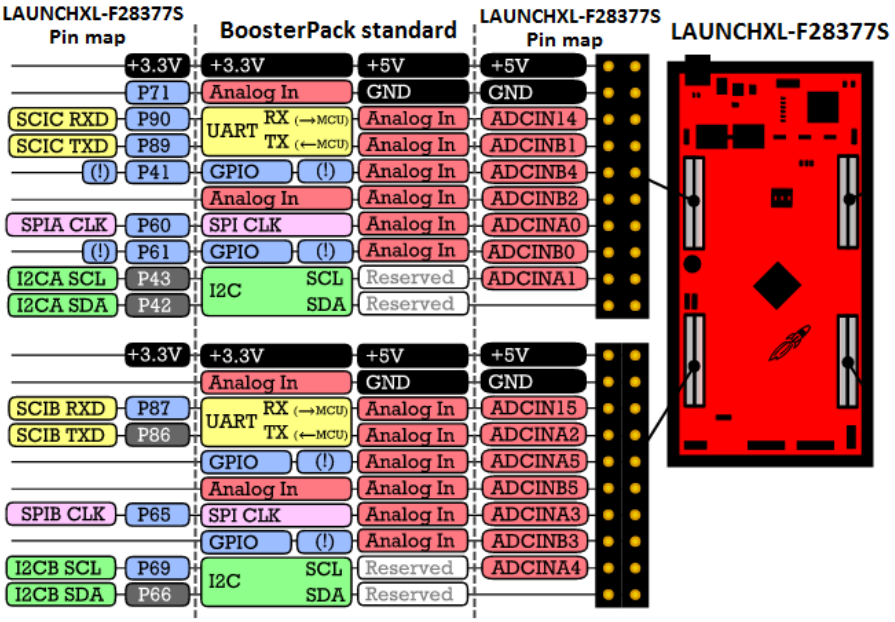


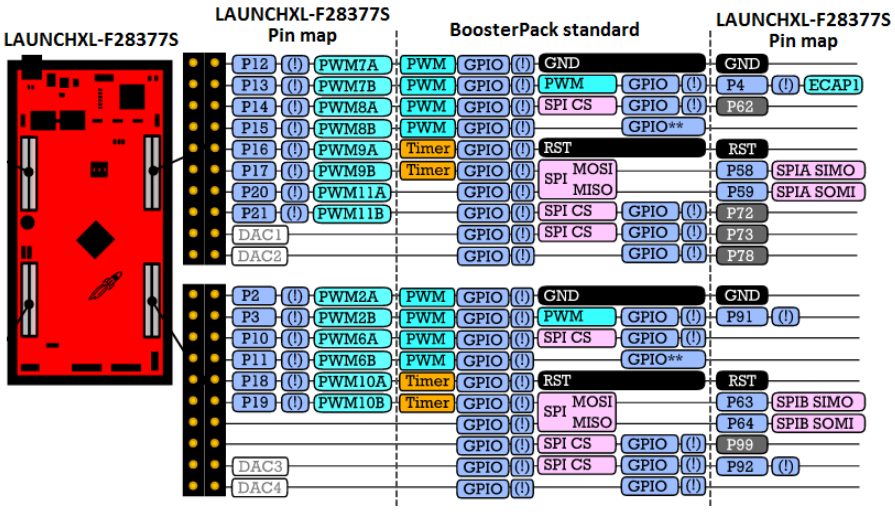
Figura 1.3: Tarjeta LaunchPad Delfino

En la Figura 1.4 se muestra el mapa de pines del LaunchPad. Los pines marcados como *Px* corresponden a los pines de entrada y salida (GPIO), así mismo se muestran los periféricos que se encuentran conectados a cada uno de los pines. La sección marcada como Booster-Pack Standard, corresponde a los pines empleados en las diferentes tarjetas de expansión

(BoosterPack) fabricadas por Texas Instruments.



(a) Mapa de pines de la LauchPad Delfino J1/J3 y J5/J7.



(b) Mapa de pines de la LauchPad Delfino J4/J2 y J8/J6.

Figura 1.4: Mapa de pines de la LauchPad Delfino [5].

Capítulo 2

Code Composer Studio

Texas Instruments provee un software llamado *Code Composer Studio (CCS)* que es un entorno de desarrollo diseñado para programar las diferentes familias de dispositivos que fabrica. Este software se puede instalar en diferentes sistemas operativos, tales como Windows, Linux y Mac OS. Sin embargo, los ejemplos realizados en el presente libro se desarrollaron utilizando la versión 7 del CCS[1] instalada una distribución del sistema operativo Linux, Ubuntu 16.04. CCS puede descargarse desde la página oficial de la empresa *Texas Instruments*, por medio del siguiente link *http://www.ti.com/tool/CCSTUDIO*.

Texas Instrumets tiene dos tipos de descargas para ejecutar la instalación del software: instalador en línea e instalador fuera de línea. La instalación en línea todos los archivos necesarios para instalar en el computador, por lo que es necesario tener buena conexión a la red mientras se ejecuta la instalación. El instalador fuera de línea contiene dichos archivos y bibliotecas para instalar, sin embargo, es muy importante seleccionar el o los dispositivos que se van a programar porque el software requiere los controladores de las tarjetas a programar para que sean reconocidas, y de la misma manera realizar una correcta ejecución del programa creado en el dispositivo.

Antes de instalar CCS en Linux, puede ser necesario instalar la biblioteca *libc6-i384*, esto se puede saber al iniciar el asistente de instalación si aparece una ventana emergente indi-

[1]Software de ambiente de desarrollo gratis, de la compañía Texas Instruments

cando un error por falta de la biblioteca mencionada. La instalación de dicha biblioteca, en Ubuntu, se hace al ejecutar la siguiente línea desde la terminal *sudo apt-get install libc6-i384*. Una vez instalada, el instalador del *Code Composer Studio* continuará con la instalación del software.

La comunicación entre los dispositivos embebidos en tarjetas de desarrollo de TI y CCS se mantiene mediante una interfaz serial denominada como XDS, cuyo soporte en los controladores JTAG permite embeber el hardware en la tarjeta de desarrollo, utilzando un puerto UART para conectarse a la PC. Dicha comunicación permite acceder en tiempo real al contenido de la memoria de los dispositivos, visualizar la información de los registros de la arquitectura del dispositivo, contabilizar ciclos de reloj entre dos segmentos de código, graficar segmentos de memoria, cargar programas, hacer sesiones para depurar o probar un código instrucción por instrucción, entre otras cosas.

En el presente capítulo se explican las características del entorno de desarrollo *Code Composer Studio*, así como la creación de nuevos proyectos, contenido de un programa en lenguaje ensamblador, compilación y ejecución en la tarjeta utilizada.

2.1. Creación de proyectos en Code Composer Studio

Cuando se ejecuta el software CCS, aparece una ventana como la que se muestra en la Figura 2.1, en la cual se solicita seleccionar un directorio de trabajo. En dicho directorio se van a alojar todos los proyectos generados por medio de ficheros con el nombre de cada proyecto.

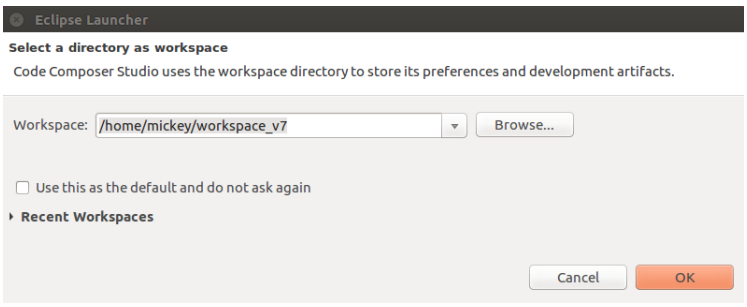


Figura 2.1: Selección del directorio de trabajo.

Posteriormente se muestra la ventana principal de *Code Composer Studio* donde se muestran botones para crear un nuevo proyecto, buscar ejemplos instalados, importar proyectos, etc. Para crear un nuevo proyecto se debe seleccionar el icono *New Project* (o bien

File→New→CSS Project), como se muestra en la Figura 2.2

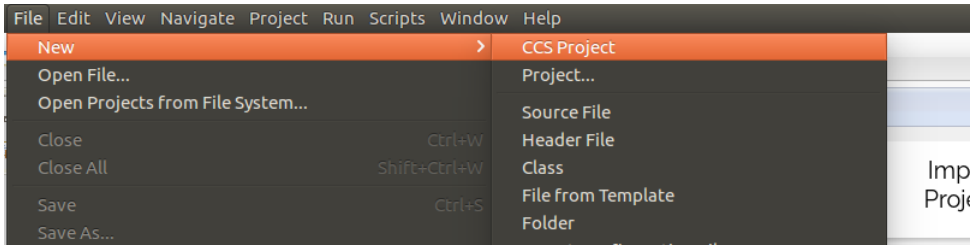


Figura 2.2: Pantalla de inicio

Es necesario configurar el proyecto con las características del sistema embebido a utilizar (microcontrolador o DSP), dichas características se ingresan en la ventana que aparece a continuación (ver Figura 2.3). A lo largo del presente manual, se desarrollaron ejemplos de programas implementados en el DSP TMS320F28377S, de tal manera que la configuración realizada en la Figura 2.3 contiene la información para dicha tarjeta.

La ventana mostrada en la Figura 2.3 se puede dividir en tres secciones, la primera es la mostrada dentro del rectángulo resaltado, en ella se seleccionan los datos relacionados de la tarjeta a utilizar y el nombre del proyecto, en este caso se seleccionó la tarjeta *2837xS Delfino* de la familia *TMS320F28377S* y el tipo de conexión *XDS100v2* por USB. El botón *Verify...* que se encuentra a la derecha del campo *Connection* genera una prueba de comunicación entre CCS y la tarjeta. Al conectar la tarjeta con el cable USB proporcionado en el Kit de evaluación y presionar el botón de *Verify...*, aparecerá una ventana emergente. Si la prueba fue exitosa, la ventana mostrará hasta el final del reporte un mensaje como el que se muestra en la Figura 2.4.

La segunda sección de la ventana de configuración del proyecto (Figura 2.3) es de ajustes avanzados (*Advanced settings*), al dar un clik en esta sección se despliega una ventana como se muestra en la Figura 2.5a. En esta sección se puede agregar un archivo con extension *cmd* en el apartado *Linker command file*, esto se logra por medio del botón *Browse...* y seleccionando la ruta en donde se encuentre dicho archivo.

El documento con extensión *cmd* es un archivo enlazador de comandos que describe el nombre, la localización y el tamaño del mapa de memoria del dispositivo, en la Sección 2.3 se explica con mayor detalle el contenido y declaración del archivo *cmd*. En el Apéndice A se expone el archivo *cmd* diseñado para las aplicaciones desarrolladas en el presente trabajo, mismo que se puede utilizar para desarrollar los ejercicios propuestos.

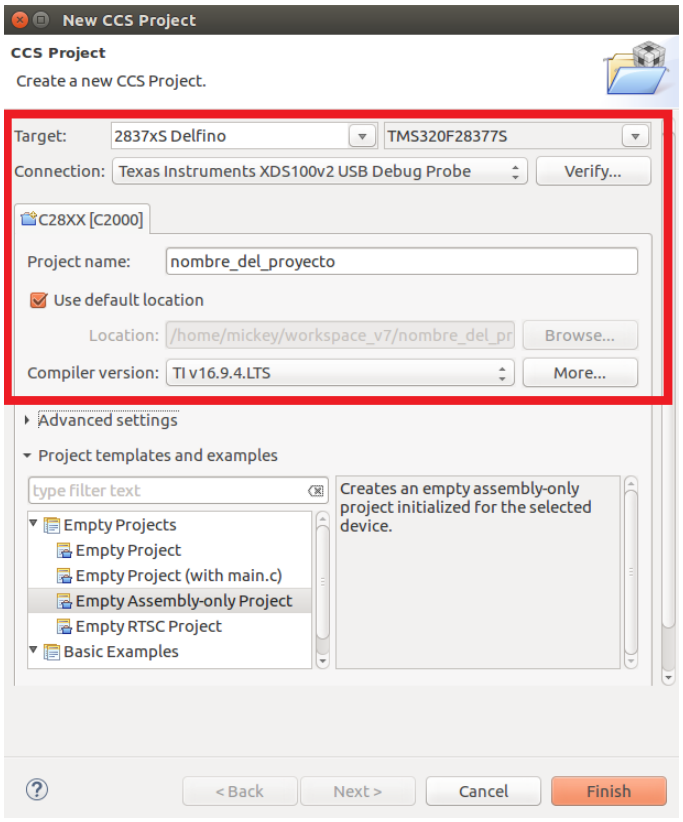


Figura 2.3: Nuevo proyecto con la tarjeta *LaunchPad Delfino*

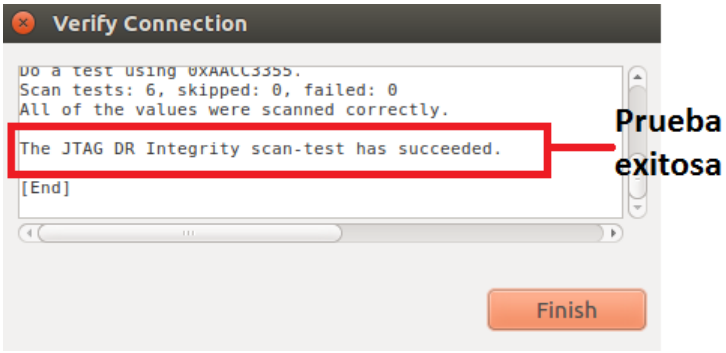
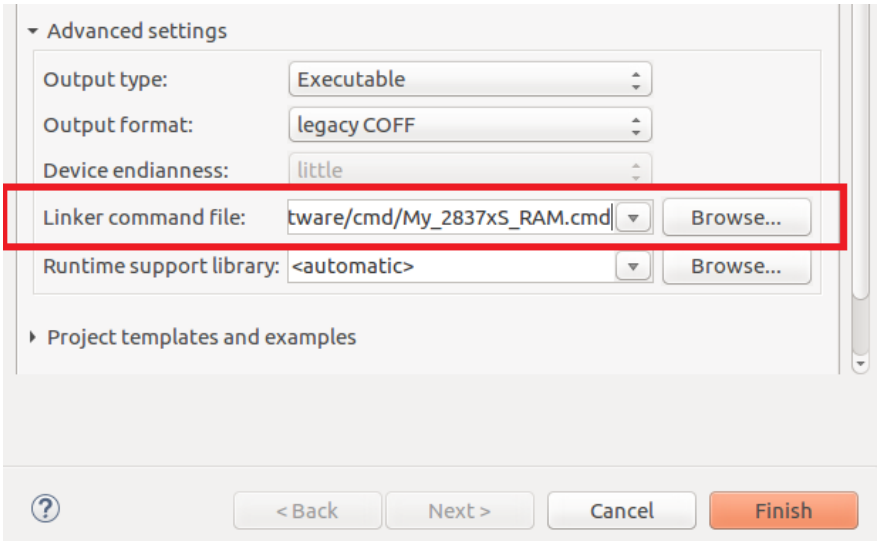
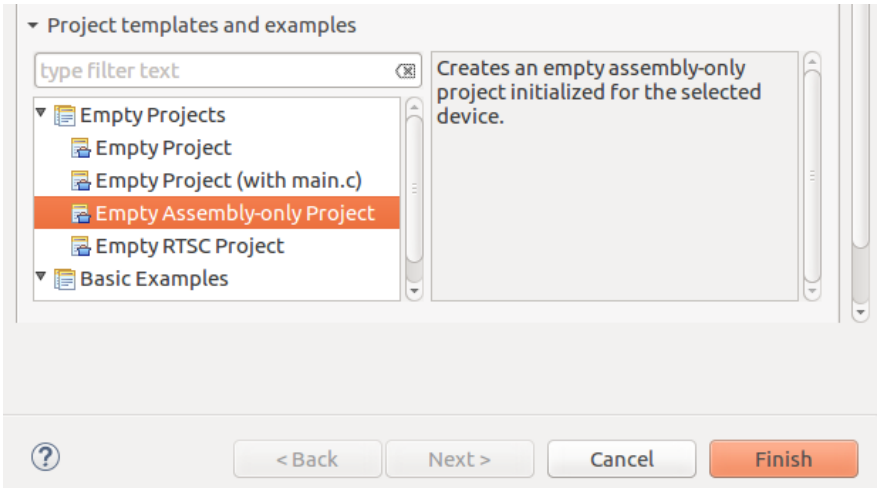


Figura 2.4: Prueba de conexión exitosa

Por último, en la sección de *Project templates and examples* se selecciona el tipo de proyecto, éste puede ser en lenguaje C o ensamblador entre otros. Los ejemplos de este manual se encuentran escritos en lenguaje ensamblador, por lo que se selecciona *Empty Assembly-only*



(a) Selección del archivo cmd.



(b) Selección del tipo de proyecto.

Figura 2.5: Selección del archivo cmd y tipo de proyecto.

Project, como se muestra en la Figura 2.5b.

Si se desea generar un proyecto para programar en lenguaje C, solo basta con seleccionar la opción *Empty Project (with main.c)* en la sección *Project temprates and examples* (ver Figura 2.5b) y automáticamente genera el archivo editor de texto con extensión *C* que contiene la función principal (*main*).

Despúes de crear el proyecto, la interfaz cambiará su apariencia y se verá como se muestra en la Figura 2.6, esta ventana se conoce como *CCS Edit* y es en donde se podrá crear, cargar

y editar un proyecto. Se pueden observar dos secciones dentro de la interfaz: el explorador de proyectos que se encuentra en la parte izquierda y la ventana de notificaciones en la parte inferior.

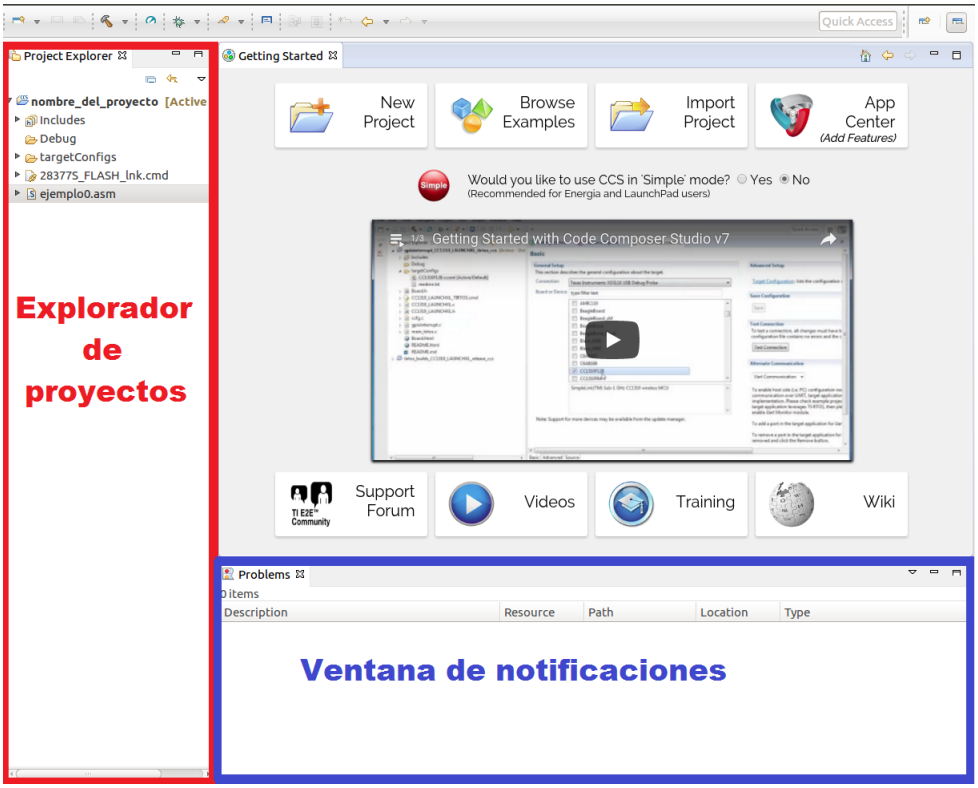


Figura 2.6: Interfaz *CCS Edit*

En el explorador de proyectos (Figura 2.6) se podrán observar todos los proyectos creados y almacenados dentro de la carpeta de trabajo seleccionada (al abrir Code Composer como se mostró en la Figura 2.1). Para activar un proyecto previamente almacenado es necesario dar clic derecho sobre el nombre del proyecto y seleccionar la opción *Open Project*, de esta manera será posible editar, compilar y ejetuar el proyecto, de la misma manera, la edición del proyecto se puede desactivar seleccionando la opción *Close Project*.

La ventana de notificaciones (Figura 2.6) muestra los problemas y avisos generados en la compilación de un programa.

En caso de no haber seleccionado el archivo *cmd* en la campo de *Linker command file* de la ventana *Advanced settings* (Figura 2.5a), *Code Composer* genera uno por defecto, mismo que puede ser utilizado para la creación del proyecto, sin embargo, es recomendable editar o

crear un archivo *.cmd* para mejorar la distribución de los bloques de memoria como el código (sección *.text*) y memoria de datos (sección *.data*).

Si se desea cargar un archivo *cmd* creado por el usuario, primero es necesario eliminar el *cmd* que generó por defecto el *Code Composer* para evitar fallas en el compilador.

Para agregar un nuevo *cmd* haga clic derecho en la carpeta del proyecto, ubicada en el explorador de proyectos, y seleccione *Add Files...*, posteriormente aparecerá una ventana en donde deberá buscar y seleccionar el archivo *.cmd* que desee insertar. Se recomienda utilizar el *cmd* que se localiza en el Apéndice A.

Después de realizar las configuraciones del proyecto, es necesario crear el archivo donde se escribirá el código del programa. Cuando el tipo de proyecto generado es en lenguaje C, el CCS crea automáticamente el archivo editor de texto con extensión *.c*, sin embargo, cuando es un proyecto para lenguaje ensamblador es necesario crearlo el archivo fuente, para ello, haga clic derecho sobre la carpeta del proyecto y seleccione *new* → *Source file*. Hecho lo anterior, emergerá una ventana como la que se muestra en la Figura 2.7. En *Source file* se escribe el nombre del archivo con terminación *.asm* (tipo de archivo para códigos en ensamblador).

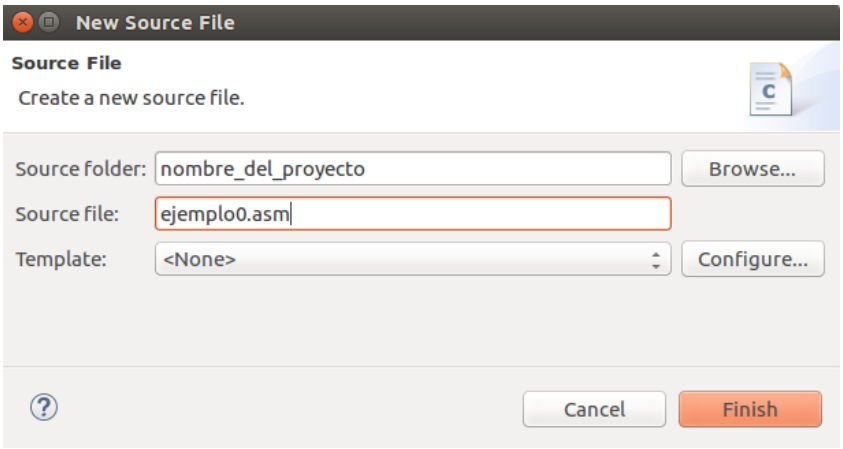


Figura 2.7: Creación de un nuevo archivo en ensamblador

2.2. Configuración del ControlSuite

Para facilitar el uso de los dispositivos programables de Texas Instrument, la empresa ha creó una herramienta llamada *ControlSuite*^{TM} la cual es un conjunto de herramientas,

bibliotecas y documentación útiles para su utilización con tarjetas embebidas de TI, teniendo un apartado dedicado para sus microcontroladores de la familia C2000 [6].

2.2.1. Instalación

ControlSuite^{TM} puede ser descargado de forma gratuita desde la página de Internet de TI [7], http://www.ti.com/tool/controlsuite. A la fecha de redacción de este trabajo, esta herramienta solo se encuentra disponible para sistemas operativos *Windows*, sin embargo, en la *wiki* de TI se pueden encontrar algunas opciones para su instalación en Linux. El desarrollo de este trabajo se realizó con la versión de Code Composer Studio (CCS) v7.4.0.00015, en una computadora con Ubuntu 16.04 LTS.

La descarga de *ControlSuite*^{TM} se puede realizar como un archivo ejecutable (*.exe*), o como una carpeta comprimida (*.zip*). Las implementaciones de ejemplo que se desarrollaron en este libro se realizaron con la versión 3.4.4 de la herramienta. Para trabajar con ControlSuite en el sistema operativo Ubuntu, se realizó la instalación en Windows siguiendo el asistente, teniendo presente la dirección que se ingresa para descargar los archivos de la herramienta. Posteriormente se exportaron todos esos archivos en la carpeta de trabajo en Ubuntu para poderlos utilizar con CCS.

2.2.2. Creación de un proyecto empleando ControlSuite

El primer paso es crear un proyecto nuevo para lenguaje C como se explicó en la Sección 2.1. Posteriormente, se selecciona la carpeta del proyecto en el *Explorador del proyectos* (Figura 2.6), dar clic derecho sobre el icono del directorio para desplegar el menú y seleccionar la opción *propiedades*.

Al desplegarse la ventana de propiedades del proyecto, seleccionar la pestaña *Resource* $\rightarrow$ *Linked Resources*, como se observa en la Figura 2.8. Aquí se encuentran las *direcciones* o *paths* que emplea CCS para compilar los proyectos, es importante señalar que **NO SE DEBEN MODIFICAR** los valores originales porque puede alterar la configuración del compilador y éste puede dejar de funcionar.

Se puede definir una variable global que será igual a la dirección general o *path* de la carpeta de trabajo de ControlSuite a utilizar. Esto se realiza, haciendo clic en el botón "New...", desglosando la ventana "Edit Variable" donde se deberá escribir el nombre de la variable, por ejemplo *CONTROLSUITE*. Posteriormente se debe seleccionar la dirección deseada, esto se realiza por medio del botón "Folder..". Al tener diferentes versiones la herramienta, se han creado diferentes versiones de bibliotecas, lo cual se puede ver en la carpeta

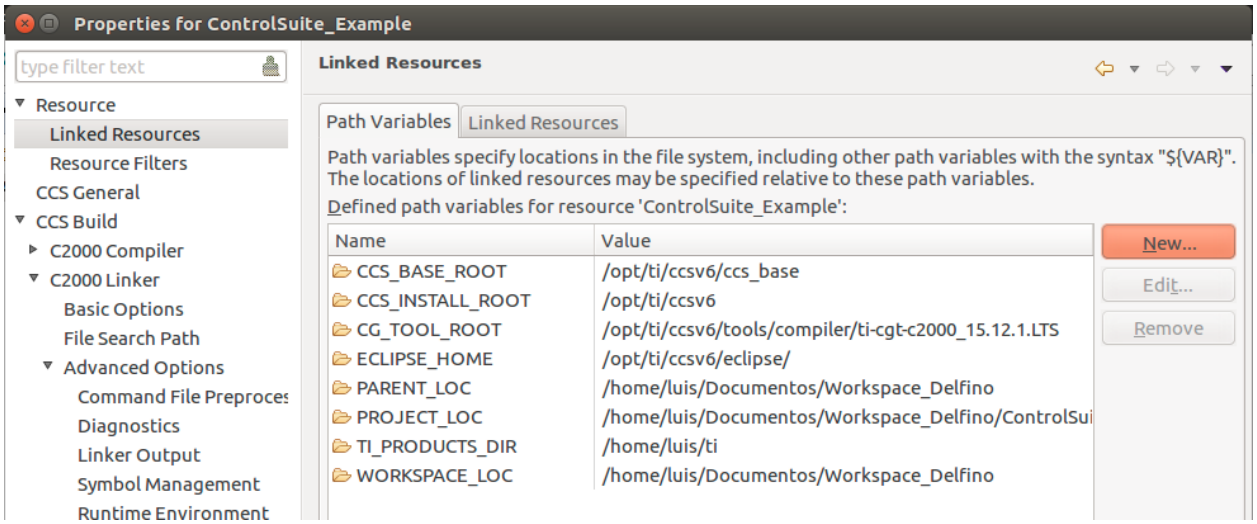


Figura 2.8: Direcciones o *paths* del compilador de CCS.

`device_support/F2837xS/` del directorio de ControlSuite. Entonces, un ejemplo de la dirección que se le asignará a la variable *CONTROLSUITE* es `/home/Documentos/Libs_Delfino/ControlSuite/device_support/F2837xS/v190`, donde **v190** indica la versión de las bibliotecas, como se muestra en la Figura 2.9.

Figura 2.9: Creación de nuevo path.

A continuación, **se comprobarán las opciones del procesador en la pestaña *Build* → *C2000 Compiler* → *Processor Options***. La configuración se muestra en la Figura 2.10, resaltando que esta corresponde al uso del DSP TMS320F28377S.

Posteriormente, **se tienen que incluir las direcciones o *paths*** donde se encuentran las bibliotecas a utilizar, para ello se accede a la pestaña *Build* → *C2000 Compiler* → *Include Options* y en la sección "*Add dir to #include search path*" se deben incluir una a la vez,

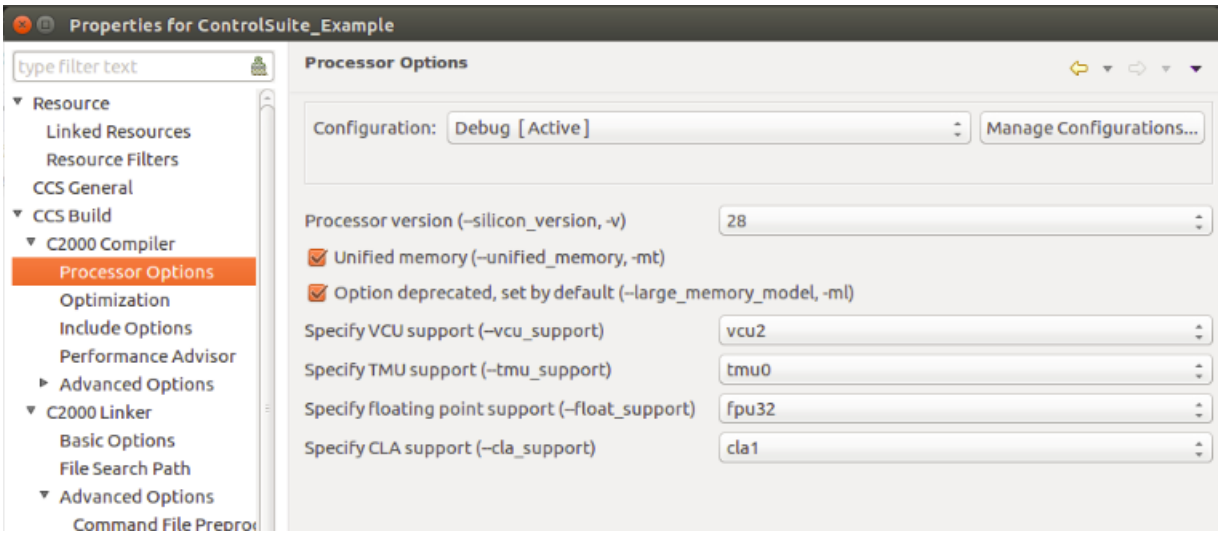


Figura 2.10: Configuración de las opciones del procesador.

las carpetas `F2837xS_headers/include` y `F2837xS_common/include`, utilizando la variable *CONTROLSUITE* para abreviar la dirección, por lo que se deberá dar clic al botón "Add", cuyo icono es una hoja con una cruz verde, señalado en la Figura 2.11 junto con la ventana que despliega con la dirección declarada.

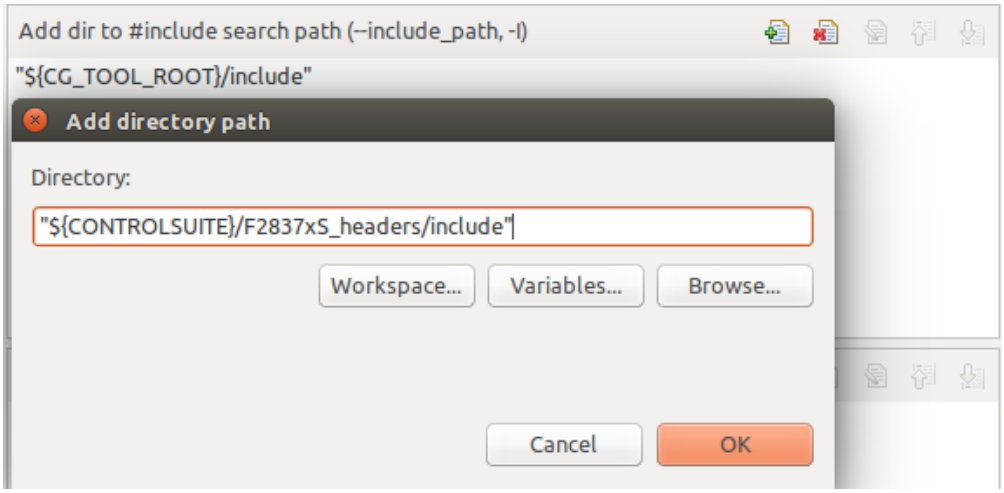


Figura 2.11: Ventana para incluir direcciones bibliotecas de ControlSuite a considerarse al compilar.

En la Figura 2.12 se observan las direcciones que se deben agregar al compilador para que pueda utilizar las bibliotecas de ControlSuite.

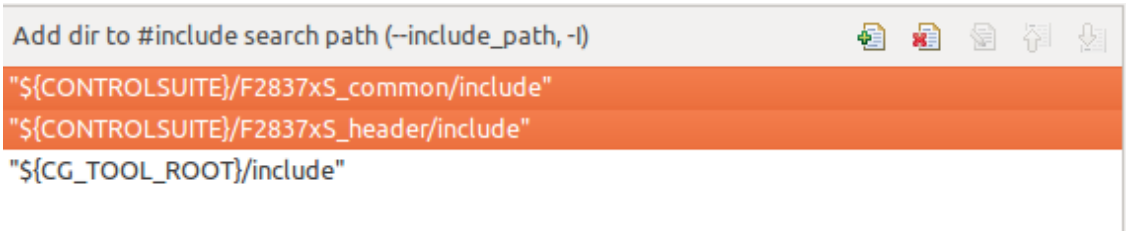


Figura 2.12: Direcciones que deben estar consideradas para incluir bibliotecas de ControlSuite.

Además, **se agregará un símbolo para indicarle al procesador el CPU a emplear (CPU1)**, esto se configura en la pestaña *CCS Build* → *C2000 Compiler* → *Predefined Symbols*. En la sección "*Pre-defined NAME*", se da clic al icono de la hoja con una cruz verde, para desplegar la ventana "Enter Value", donde se deberá escribir CPU1, como se muestra en la Figura 2.13 y dar clic en el botón de *OK*. Este símbolo es importante cuando se emplean procesadores de varios núcleos, para el caso de la Delfino F28377S no es necesario agregar dicho símbolo porque contiene un procesador de un núcleo.

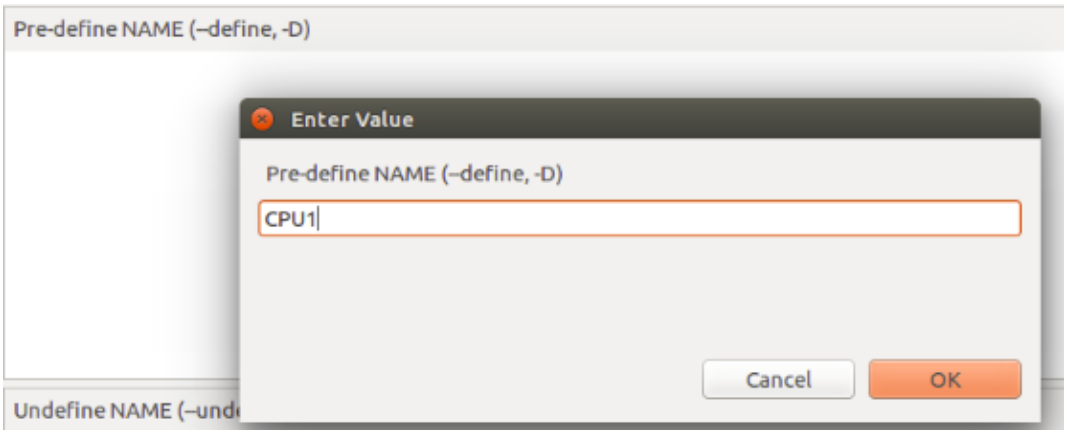


Figura 2.13: Definición del CPU a emplear en la compilación.

El mapa de memoria definido en los archivos *.cmd*, puede ser configurado de diferentes formas acorde a las necesidades de la aplicación como se verá en la Sección 2.3. ControlSuite contiene diferentes archivos *.cmd* de uso general, con diferentes características, las cuales son utilizadas por las diferentes bibliotecas que se pueden utilizar dentro de un proyecto para el uso de los periféricos, por lo que se debe de agregar al compilador la ubicación de dichos archivos *.cmd*.

Para realizar lo mencionado en el párrafo anterior, seleccionamos la pestaña *CCS Build* → *C2000 Linker* → *File Search Path*. En la sección "*Add <dir> to library search path*" se

agregarán las direcciones `F2837xS_headers/cmd` y `2837xS_common/cmd` (nuevamente con el botón cuyo icono es una hoja con una cruz verde).

Después, en la sección "*Include library file or command file as input*" se deben agregar los archivos `F2837xS_Headers_nonBios.cmd`, ubicado en la carpeta `F2837xS_headers/cmd` y la biblioteca `rts2800_fpu32.lib` que se encuentra en la raíz del CCS (`ti/ccsv6/tools/compiler/c2000_15.12.3.LTS/lib`). En la Figura 2.14 se muestra la lista de direcciones y archivos que se deben incluir en esta ventana.

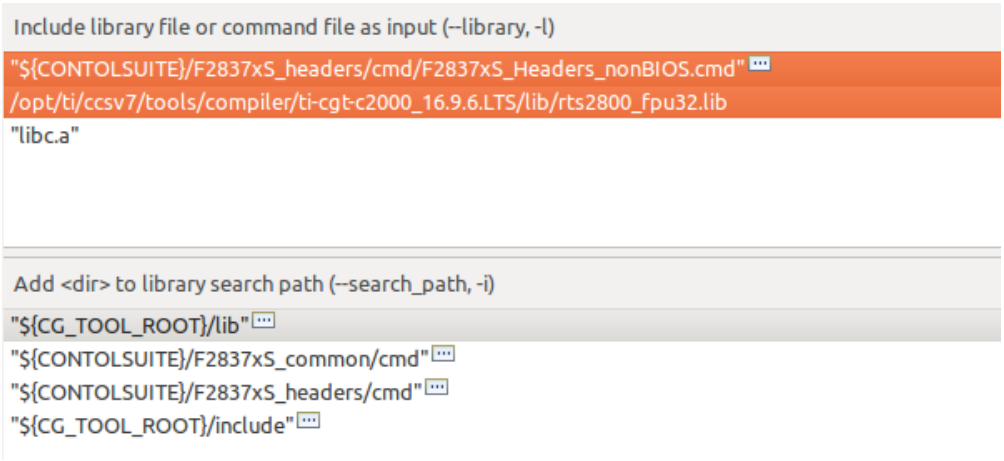


Figura 2.14: Inclusión de direcciones y archivos para el *linker*.

Para aplicar todas las configuraciones realizadas a las propiedades del proyecto, se debe hacer clic en el botón "OK" de la ventana *Properties*.

Por último, **se deben agregar las bibliotecas que configuran los periféricos a utilizar**, lo cual se hace seleccionando la carpeta del proyecto en el *Explorador de proyectos* y dar clic derecho sobre el icono seleccionado, para acceder a la opción *Add files* del menú desplegado. En la ventana "*Add files to ..*", se debe agregar el archivo `F2837xS_GlobalVariableDefs.c` que se encuentra en la dirección `ControlSuite/device_support/F2837xS/v190/F2837xS_headers/source`, como se muestra en la Figura 2.15, recordando que se puede usar la versión de bibliotecas que se desee, en este caso se está utilizando la v190. El archivo citado contiene las definiciones para las secciones de memoria de los diferentes periféricos que tiene el MCU.

Al dar clic en el botón "OK" para agregar el archivo, se desplegará la ventana "File operation" como se observa en la la Figura 2.16. En dicha ventana se muestran las dos formas posibles en las que se puede agregar el archivo. La primera de ellas es haciendo una copia del archivo original en nuestra carpeta del proyectos y la segunda opción genera un acceso

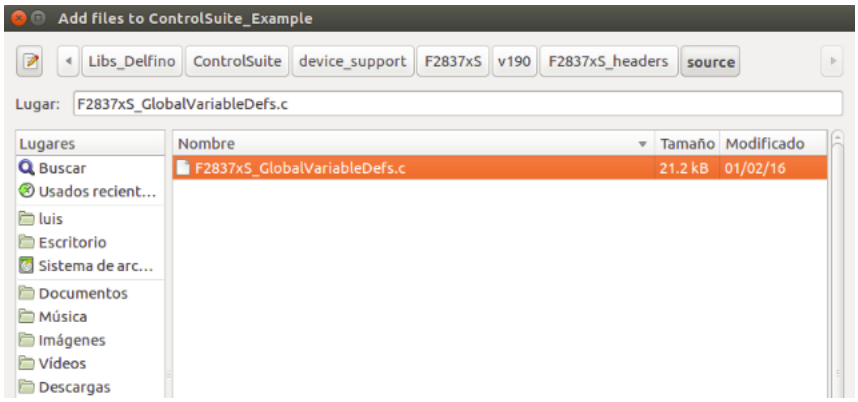


Figura 2.15: Ubicación del archivo *F2837xS_GlobalVariableDefs.c*

directo o *link* hacia el archivo original. Se recomienda utilizar la primer opción para no editar el código fuente de las bibliotecas del ControlSuite ya que no es recomendable modificar dichos archivos, a menos que se tenga el conocimiento necesario.

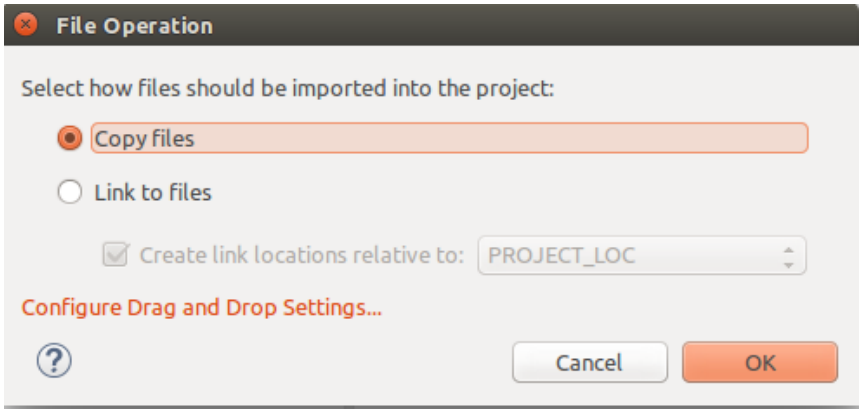


Figura 2.16: Opciones para agregar archivos al proyecto.

Una vez agregado el archivo de **configuración de variables**, se deben agregar los archivos fuentes.c de los periféricos que se van a utilizarán, los cuales se encuentran en `ControlSuite/device_support/F2837xS/v190/F2837xS_common/source`. Sin embargo, independientemente del proyecto que se vaya a desarrollar, es recomendable agregar siempre los siguientes archivos:

- `F2837xS_CodeStartBranch.asm`
- `F2837xS_SysCtrl.c`
- `F2837xS_usDelay.asm`

- `F2837xS_Gpio.c`

Estos archivos contienen funciones de uso común para varios proyectos. Una vez realizada esta configuración, se pueden crear códigos que empleen las bibliotecas previamente agregadas, sin embargo, los ejemplos que utilizan CONTROLSUITE en el presente libro se emplean en el Capítulo 5 para configurar los periféricos de la tarjeta.

2.3. Mapa de memoria

Antes de comenzar a escribir programas de diferentes aplicaciones, es necesario conocer el mapa de memoria del dispositivo de trabajo porque en cada proyecto, se utilizarán diferentes recursos del DPS, implicando manejar distintas partes de los registros de memoria dedicados, aunque los códigos solo manejen datos o periféricos.

En el diagrama de bloques de la Figura 1.1, se observa la arquitectura del TMS320-F28377S, donde en diferentes partes se encuentran algunos de los módulos que forman la memoria del DSP, los cuales se pueden clasificar en dos clases; registros de memoria para periféricos y registros para almacenamiento/transferencia de datos y de códigos. En la Sección seis del manual [4], se encuentran las tablas que contienen el nombre de las diferentes secciones de la memoria, así como su tamaño y direcciones.

El mapa de memoria está formado por tres tipos; RAM, ROM y FLASH organizados en las siguientes partes;

* Memoria general de los dispositivos de la familia C28x; formada por la memoria RAM es utilizada principalmente para correr el código en tiempo real y para definir variables globales inicializadas y no inicializadas, teniendo conexión directa con el CPU.

* Bancos de la memoria FLASH; comúnmente utilizada para guardar los programas, aunque en aplicaciones se deba de transferir las instrucciones del código a la memoria RAM.

* Memoria de interfaz externa EMIF; utilizada para conectar al DSP otras memorias externas a su arquitectura.

* Registros de memoria de periféricos; utilizados para configurar y administrar los datos que adquieren o se escriben como salidas por los diferentes puertos.

La memoria ROM es reservada a excepción de una parte en la cual pueden escribirse datos una sola vez para alguna aplicación, por ejemplo, para grabar coeficientes de algún filtro FIR. Para información más detallada de la memoria del dispositivo se recomienda al

lector consultar los manuales [4, 8].

En cada proyecto se tienen que declarar los módulos de memoria que se utilizarán, además de las correspondientes asignaciones para cada una de las secciones necesarias para implementar algún programa, dicha información se manifiesta por escrito en un archivo con terminación *.cmd*, denominado en los manuales como *linker command file*. La IDE CCS al crear un nuevo proyecto (proceso que se describirá en la Sección 2.1), recomienda al usuario un archivo *.cmd* configurado por TI para el desarrollo de proyectos básicos y generales, sin embargo, el usuario puede definir su propio archivo y para ello se describirá de forma básica a continuación la forma de hacerlo.

2.3.1. Linker command file o archivo *.cmd*

Referente a su nombre, este archivo permite declarar opciones para enlazar el archivo principal de un proyecto con otros secundarios (por ejemplo, al utilizar Control Suite que se verá en la Sección 2.2.2) y también declarar y asignar el mapa de memoria a utilizar. Estos archivos pueden contener

* Opciones de enlace para la crear el archivo ejecutable.
* Nombres de archivos objeto, que contienen asignaciones de bloques de memoria de entrada (datos de entrada).
* Declarar la inclusión de alguna biblioteca.
* Declarar variables globales.

El archivo se divide principalmente en dos partes; *MEMORY* es el apartado donde se definen los bloques de memoria a utilizar del dispositivo y en *SECTIONS* se asignan bloques de memoria a las secciones necesarias para que un programa funcione. El esquema general de este archivo se muestra a continuación

```
MEMORY{
// Declaración del mapa de memoria a utilizar
// (bloques de la memoria RAM/FLASH)
...

}

SECTIONS{
// Asignación de los bloques de memoria a las
// secciones necesarias para un programa
...

}
```

A continuación se presentarán las instrucciones y formato de cada parte del archivo *.cmd*.

Declaración de bloques de memoria

El objetivo y función del bloque **MEMORY** es asignar los nombres e intervalos de la memoria conectada directamente al CPU (RAM o FLASH) que se van a utilizar, declarando esta información con el siguiente formato

```
*
**** Formato para definir el uso de memoria ****
*
file1.obj    file2.obj          // Archivos de entrada
--output_file = prog.out        // Opciones del enlazador
#define zeros 0                 // Var. global

MEMORY
{
   PAGE 0: nombre 1 (atrib.)  :origin= dir. ini., length= tamaño, fill= const.
   PAGE 1: nombre 2 (atrib.)  :origin= dir. ini., length= tamaño, fill= zeros
    .
    .
   PAGE n: nombre n (atrib.)  :origin= dir. ini., length= tamaño, fill= const.
}
```

donde:

PAGE; es una referencia para identificar espacios de la memoria, es decir que define un conjunto de bloques de memoria con un objetivo común. Se pueden declarar hasta 32 767 páginas de memoria, siempre y cuando siempre la página 0 defina la memoria del programa y la página 1 la memoria para datos.

Nombre n (atrib.); es una denominación de referencia al bloque de memoria que se esta declarando para ser utilizado, una práctica común es ocupar los nombres asignados por el fabricante, especificados en la Sección 6.3 del manual [4]. Separado por un espacio, entre paréntesis se puede definir para dicho bloque, uno de los siguientes atributos

* **R** si el bloque es solo de lectura
* **W** si el bloque es solo de escritura
* **X** si el bloque contiene código del ejecutable
* **I** si el bloque puede ser inicializado con datos

Especificar un atributo no es necesario, al no declararlo se deja libre de restricciones el uso del bloque de memoria.

origin; este campo es para especificar la dirección de inicio del bloque de memoria que se esta declarando. La dirección debe estar en el intervalo de la memoria del dispositivo de libre

uso, especificada en bytes por un valor entero de 32 bits que puede ser escrito en decimal, octal o hexadecimal siendo este último el formato el más usual. En lugar de la palabra *origin* se puede utilizar la palabra *org* o la letra *o*.

length; especifica la longitud del bloque de memoria expresado en bytes, considerado por el compilador como un valor entero de 22 bits que se puede escribir en decimal, octal o hexadecimal. El tamaño declarado del bloque puede asignarse acorde a los valores definidos en los mapas de memoria especificados en la Sección 6.3 del manual [4], o pueden ser definidos libremente por el usuario respetando la longitud de palabra de esta variable, acorde a la aplicación. Este campo se puede abreviar con la palabra *len* o la letra *l*.

fill; declara un valor que se desea contenga todas las localidades de memoria del bloque que se esta definiendo. Dicho valor puede ser definido explícitamente o utilizando una variable global. Este campo no es obligatorio declararlo y también puede abreviarse como *fil* o con la letra *f*.

Un ejemplo breve sobre la declaración de la función *MEMORY* en el archivo *.cmd* se muestra a continuación, en donde se puede observar la definición de las dos páginas necesarias para el funcionamiento de un programa que no utiliza periféricos del MCU y la extensión del bloque *RAMGS0* que cubre los bloques del mapa predeterminado *RAMGS1* a *RAMGS12*, con la finalidad de contar con más espacio para cargar o guardar datos para una aplicación de cálculo.

```
MEMORY{
    PAGE 0:
    /* Memoria para el programa */
    /* Bloques de memoria (RAM/FLASH) pueden ser declarados en la */
    /* página 1 para usarlos como datos */
    /* BEGIN es utilizado para iniciar desde FLASH o RAM el modo */
    /* bootloader, en este caso se inicia desde FLASH */

    BEGIN           : origin = 0x080000, length = 0x000002 //obligatorio
    RAMM0           : origin = 0x000122, length = 0x0002DE
    RAMD0           : origin = 0x00B000, length = 0x000800
    RAMLS0          : origin = 0x008000, length = 0x000800
    RAMLS1          : origin = 0x008800, length = 0x000800
    RESET           : origin = 0x3FFFC0, length = 0x000002 //obligatorio

    /* Sectores de FLASH */

    FLASHA          : origin = 0x080002, length = 0x001FFE
    FLASHB          : origin = 0x082000, length = 0x002000
    FLASHC          : origin = 0x084000, length = 0x002000
    FLASHD          : origin = 0x086000, length = 0x002000
```

```
    FLASHE          : origin = 0x088000, length = 0x008000
    FLASHF          : origin = 0x090000, length = 0x008000

    PAGE 1:
    /* Memoria de datos */
    /* Bloques de memoria (RAM/FLASH) pueden ser declarados en la */
    /* página 0 para usarlos como datos */

    BOOT_RSVD       : origin = 0x000002, length = 0x000120 //obligatorio
    RAMM1           : origin = 0x000400, length = 0x000400
    RAMD1           : origin = 0x00B800, length = 0x000800

    RAMLS5          : origin = 0x00A800, length = 0x000800

    RAMGS0          : origin = 0x00C000, length = 0x00D000
    RAMGS13         : origin = 0x019000, length = 0x001000
}
```

Como nota adicional, no es necesario tener que utilizar los dos tipos de memoria FLASH y RAM, es posible utilizar solo un tipo de memoria acorde a la implementación que se vaya a desarrollar. Al ocupar FLASH el tiempo de escritura para entrar al modo *Debug* o almacenar el código en el dispositivo, es mayor que al utilizar solo memoria RAM, razón por la que como parte de este trabajo, se propone al usuario un archivo *.cmd* que utiliza solo RAM en diferentes aplicaciones, dicho mapa de memoria se encuentra en el Apéndice A.

Asignación de bloques de memoria

El compilador maneja la memoria del dispositivo como dos bloques lineales; uno para el programa y otro para datos, el contenido específico de cada uno de estos bloques es el siguiente

* La memoria de programa contiene el código ejecutable, registros de inicialización y tablas de casos al utilizar la instrucción *switch*.

* La memoria de datos contiene variables externas, estáticas y la pila del sistema.

Ambos bloques son organizados en *secciones* que cumplen un propósito particular en la etapa de compilación del proyecto y son declaradas en la segunda parte del archivo *.cmd* denotada como **SECTIONS**. Una *sección* se puede definir como la unidad mínima de un archivo objeto, que ocupa bloques de memoria consecutivos y se clasifican en dos tipos básicos

* Secciones inicializadas; contienen código o datos definidos al iniciar la ejecución del programa.

* Secciones no inicializadas; son localidades del mapa de memoria reservadas para datos que no han sido definidos al inicio de la ejecución del programa.

Para generar el código ejecutable, el compilador necesita definir las secciones base y asignarles bloques de memoria a cada una de ellas, en caso de que no se definan por el usuario CCS implementa un algoritmo para declarar una distribución, dicho procedimiento se puede consultar en la Sección 8.7 de [9]. En la Tabla 2.1 se listan las secciones base y algunas de uso especial

Tabla 2.1: Lista se las secciones básicas para generar el código ejecutable de un proyecto en CCS para el TMS320F28377S.

Sección	**Inicialización**	**Descripción**
.text	Si	Contiene el código ejecutable del programa
.ebss	No	Utilizada para variables globales y estáticas
.data	Si	Sección dedicada para datos COFF ABI ->Contiene datos inicializados
	Si/No	EABI ->Inicializado saliendo del ensamblador; cambiado a no inicializado por el linker
.econst	Si	Contiene constantes de cadena, literales de cadena, tablas de conmutación, declaración e inicialización de variables globales y estáticas, y datos definidos como *const* en C/C++. Esta sección normalmente es de solo lectura
.cinit	Si	Se usa para inicializar variables globales en un programa de C/C++
.stack	No	Utilizado para la pila de llamadas de función
.esysmem	No	Se usa para el grupo de asignación de memoria dinámica.
.switch	Si	Contiene los saltos que se pueden hacer al utilizar la instrucción switch
.cio	No	Buffer para funciones de la biblioteca stdio en programas de C/C++
.pinit	Si	Contiene la lista de constructores globales para programas de C/C++
codestart	Si	Sección para iniciar el modo bootloader desde el bloque asignado
ramfunctions	Si	Memoria para ejecutar funciones en la RAM

La sección *.data* puede no ser declarada, ya que otras secciones como *.ebss* ó *.esysmem* hacen parte de su función, en algunos programas del Capítulo 4 se utilizaa dicha sección con el objetivo mostrar su uso. Una descripción más detallada de las secciones listadas así como de otras que se pueden utilizar, se puede consultar en [9] y [10]. La declaración de las

secciones se realiza con la siguiente sintaxis

```
SECTIONS
{
    nombre 1: [propiedad [,propiedad][,propiedad]...]
    nombre 2: [propiedad [,propiedad][,propiedad]...]
    .
    .
    nombre n: [propiedad [,propiedad][,propiedad]...]
}
```

En el campo de *nombre* se escribe alguna de la secciones mostradas en la Tabla 2.1 seguida una o más propiedades, separadas por una coma teniendo las siguientes opciones:

Asignación de carga; define en qué lugar(es) de la memoria se cargará la sección declarada, la sintaxis para especificar esta propiedad es

```
Nombre de sección: load = nombre del bloque o
Nombre de sección: > nombre del bloque
```

También es posible declarar opciones alternas de asignación de memoria a una sección, para el caso donde el tamaño del bloque no sea el suficiente, el compilador podrá utilizar la segunda, tercera, cuarta, etcétera opción, por lo que la sintaxis anterior se reescribe como

```
Nombre de la sección: > bloque opción 1 | ... | bloque opción n
```

Otro caso que se puede presentar en la asignación, es que el tamaño de un bloque de memoria no sea suficiente para la sección, por lo que es posible hacer un tipo de expansión que consiste en utilizar otros bloques de memoria consecutivos, para que el compilador divida la información de forma uniforme en aquellos que se hayan declarado. La sintaxis de asignación de sección con opción de expansión es la siguiente

```
Nombre de la sección: >> bloque opción 1 | ... | bloque opción n
```

Algunas de las opciones extras que se pueden incluir en esta propiedad son;

* Dirección especifica; en lugar de declarar el nombre de un bloque de memoria para una sección, también se pude indicar la dirección inicial del bloque que se asignará utilizando el signo de igualdad, como se muestra en el siguiente ejemplo

  ```
  .text: load = 0x1000
  ```

* Alineación (**.align**); el ensamblador trabaja con un contador de programa dedicado a las secciones (SPC), el cual va apuntando a las direcciones de memoria asignadas a

alguna sección. La opción *.align* desfasa dicho contador acorde a la cantidad de palabras de 16 bits que se le indiquen y en caso de solo escribir la opción, el ensamblador desfasa la dirección hasta la siguiente sección. La sintaxis para utilizar esta opción se muestra a continuación

```
.align = 4    o    .align(4)    o    .align 4
```

En el archivo *.cmd*, al incluirlo en la asignación de memoria a una sección, desfasa la dirección inicial del bloque de memoria que se utilizará para la tarea de la sección, con el objetivo de tener localidades de separación para no sobreescribir datos.

* Página (**PAGE = n**); su función es especificar la página de memoria en la que se encuentran los o el bloque de memoria asignado a una sección.

Asignación para ejecución; esta propiedad define en qué parte de la memoria se ejecutará la sección, y es declarada con la siguiente sintaxis

```
run = nombre del bloque de memoria     o
run > nombre del bloque de memoria
```

Secciones de entrada; define secciones de entrada, declaradas en archivos objeto que constituyen una sección de salida. Se especifican al escribirse entre paréntesis tipo llave {*archivo_objeto.obj*}.

Tipo de sección; son tres clases que modifican la forma en la que el compilador trata a las secciones señaladas por alguna de estas denominaciones. Para especificar un tipo de sección se sigue el formato

```
type = clase o tipo
```

Las posibles clases o tipos que se pueden utilizar son

* DSECT; crea una sección ficticia que no tiene asignada memoria física al generar el mapa de memoria del archivo ejecutable. Tiene como funciones; declarar símbolos globales que si son asignados a localidades de memoria en el intervalo vinculado a la sección y para buscar símbolos externos no definidos (variables globales en el proceso de ensamble) en bibliotecas.

* COPY; las funciones de esta clase son similares al tipo DSECT, a excepción que este tipo contiene y asocia información que si se escribe en el archivo ejecutable.

* NOLOAD; este tipo hace que la sección de salida solo aparezca en el mapa de memoria del archivo ejecutable, pero no escribe el contenido de la sección, la información de reubicación y la información del número de línea del bloque de memoria.

Valor de relleno; define algún valor para escribirlo en alguna sección no inicializada, mediante la instrucción

```
fill = valor
```

A continuación se muestra un ejemplo de declaración de secciones

```
SECTIONS
{
   // Asignación de memoria a secciones del programa:
   .cinit          : > FLASHB       PAGE = 0, ALIGN(4)
   .pinit          : > FLASHB,      PAGE = 0, ALIGN(4)
   .text           : >> FLASHB | FLASHC | FLASHD | FLASHE  PAGE = 0, ALIGN(4)
   codestart       : > BEGIN                               PAGE = 0, ALIGN(4)
   ramfuncs        : LOAD = FLASHD,
                         RUN = RAMLS0 | RAMLS1 | RAMLS2 |RAMLS3,
                         LOAD_START(_RamfuncsLoadStart),
                         LOAD_SIZE(_RamfuncsLoadSize),
                         LOAD_END(_RamfuncsLoadEnd),
                         RUN_START(_RamfuncsRunStart),
                         RUN_SIZE(_RamfuncsRunSize),
                         RUN_END(_RamfuncsRunEnd),          PAGE = 0, ALIGN(4)

#ifdef __TI_COMPILER_VERSION__
   #if __TI_COMPILER_VERSION__ >= 15009000
     .TI.ramfunc : {} > FLASHD, PAGE = 0, ALIGN(4)
   #endif
#endif

   // Asignación de memoria a secciones no inicializadas para datos
   .stack          : > RAMGS14                             PAGE = 0
   .data           : > RAMGS0                              PAGE = 1
   .ebss           : >> RAMGS13                            PAGE = 1
   .esysmem        : > RAMLS5                              PAGE = 1

   // Asignación de memoria a secciones inicializdas
   // que van en memoria Flash
   .econst         : >> FLASHF | FLASHG | FLASHH           PAGE = 0, ALIGN(4)
   .switch         : > FLASHB,                             PAGE = 0, ALIGN(4)

   .reset          : > RESET,                          PAGE = 0, TYPE = DSECT

}
```

Esto fue un breve resumen sobre el *Linker command file* o archivo *.cmd*, la información detallada para escribir este archivo se puede consultar en los manuales [9] y [10].

2.3.2. Linker command file para periféricos

En los proyectos donde se utilicen algunos de los periféricos del TMS320F28377S, utilizando el conjunto de bibliotecas de *Control Suite* en lenguaje C/C++ (el cual se explicará en la Sección 2.2.2) es necesario incluir para compilar, el mapa de memoria donde se encuentran declarados los registros que utiliza cada puerto para su funcionamiento. El archivo *F2837xS_Headers_nonBIOS.cmd* ubicado dentro de la carpeta de *Control Suite*, en la siguiente dirección

```
device_support/F2837xS/vxxx/F2837xS_headers/cmd/
```

donde en */vxx* debe elegirse alguna de las versiones que se encuentran disponibles, por ejemplo la v200. A diferencia del archivo *.cmd* descrito en la subsección 2.3.1, las direcciones de los registros en la memoria son estáticos, por lo que no debe de modificarse el archivo *F2837xS_Headers_nonBIOS.cmd*, o en caso de que el usuario desee hacerlo es recomendable apoyarse del manual spruxh5d [8] para direccionar adecuadamente los datos a los registros correspondientes de cada periférico.

La estructura del archivo *Lniker command* para periféricos es la misma que se ha descrito en esta sección y sigue las mismas reglas descritas para declarar las asignaciones de memoria, como se puede observar en el breve contenido del archivo *F2837xS_Headers_nonBIOS.cmd* que se muestra a continuación:

```
MEMORY
{
 PAGE 0:    /* Memoria para el Programa */

 PAGE 1:    /* Memoria para Datos */

   // Registros para contener el resultado de la
   // conversión de alguno de las terminales de
   // los cuatro conjuntos que forman el ADC
   ADCA_RESULT   : origin = 0x000B00, length = 0x000020
   ADCB_RESULT   : origin = 0x000B20, length = 0x000020
   ADCC_RESULT   : origin = 0x000B40, length = 0x000020
   ADCD_RESULT   : origin = 0x000B60, length = 0x000020

   // Registros de configuración para el funcionamiento
   // de cada uno de los cuatro bloques que forman
   // el periférico ADC
   ADCA          : origin = 0x007400, length = 0x000080
   ADCB          : origin = 0x007480, length = 0x000080
   ADCC          : origin = 0x007500, length = 0x000080
   ADCD          : origin = 0x007580, length = 0x000080
.
.
```

```
.

SECTIONS
{
/*** PIE Vect Table and Boot ROM Variables Structures ***/
  UNION run = PIE_VECT, PAGE = 1
   {
      PieVectTableFile
      GROUP
      {
         EmuKeyVar
         EmuBModeVar
         FlashCallbackVar
         FlashScalingVar
      }
   }

   // Asignación de memoria a los registros para
   // guardar el resultado de la conversión del ADC
   // por cada uno de sus bloques A, B, C y D
   AdcaResultFile          : > ADCA_RESULT,   PAGE = 1
   AdcbResultFile          : > ADCB_RESULT,   PAGE = 1
   AdccResultFile          : > ADCC_RESULT,   PAGE = 1
   AdcdResultFile          : > ADCD_RESULT,   PAGE = 1

   AdcaRegsFile            : > ADCA,          PAGE = 1
   AdcbRegsFile            : > ADCB,          PAGE = 1
   AdccRegsFile            : > ADCC,          PAGE = 1
   AdcdRegsFile            : > ADCD,          PAGE = 1
.
.
.
}
```

2.4. Componentes de un programa en ensamblador

Un programa fuente en ensamblador debe contener una estructura básica para que se pueda compilador correctamente. Este consiste en expresiones que pueden contener directivas, instrucciones de ensamblador o macroinstrucciones y subrutinas con etiquetas, además de comentarios (opcionales). En la Figura 2.17 se muestra un código que muestra la estructura básica de un programa en lenguaje ensamblador.

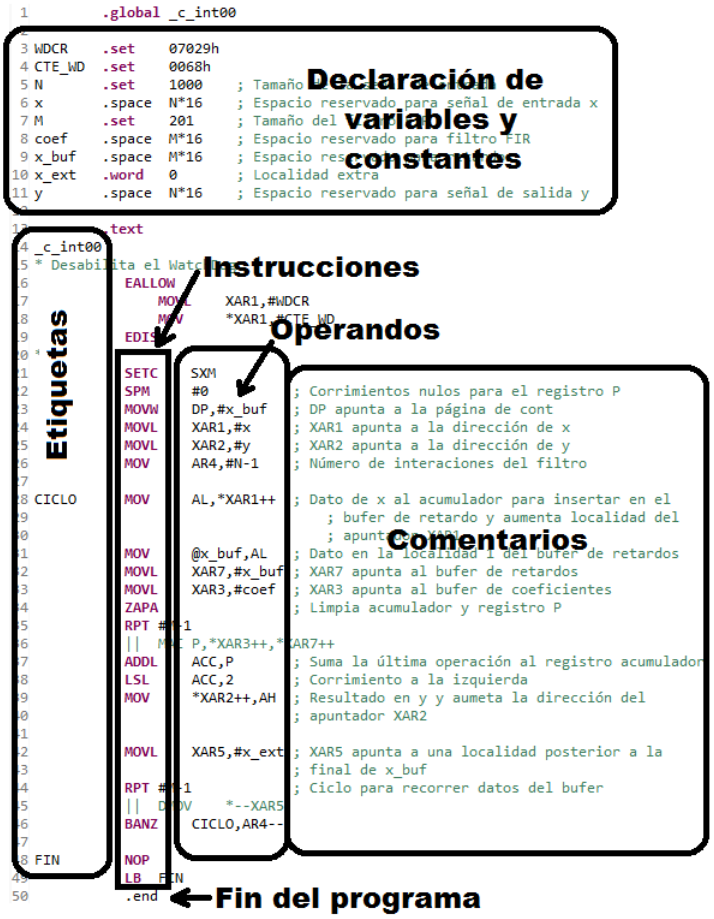


Figura 2.17: Estructura de un programa en lenguaje ensamblador.

A partir de la Figura 2.17 se puede observar que la sintaxis de una instrucción en ensamblador es la siguiente:

ETIQUETA ***MNEMONICO*** *OPERANDOS* ; *COMENTARIOS*

Cabe resaltar que no todos los programas contienen la sección de comentarios, eso dependerá del programador, sin embargo, es recomendable realizar comentarios sobre el código para una guía fácil del desarrollo del mismo. A continuación se explica cada uno de los elementos de la sintaxis el programa:

1. **Etiqueta.** Es un símbolo que indica el inicio de una sección de código de interés, es decir, se utiliza para transferir el control del código en función de ciertas decisiones como saltos a subrutinas de funciones, interrupciones o saltos condicionales, como es el caso de un ciclo o una sentencia *if*. Una etiqueta en un código es opcional y cuando se utiliza se debe colocar en la primera columna del código, desplazando a la derecha los mnemónicos y operandos.

2. **Mnemónico.** Está conformado por Instrucciones, macroinstrucciones y directivas en ensamblador. Los mnemónicos ejecutan la orden dentro de un programa, es decir, operaciones, direccionamientos, saltos condicionales e indirectos, transferencia de datos, etc.

3. **Operandos.** Los operandos varían dependiendo del tipo de instrucción que se utilice y son separados por comas. Existen diferentes tipos de operandos en un programa de ensamblador, estos pueden ser registros, símbolos, acceso a constantes, etiquetas o apuntadores.

4. **Comentarios.** Son optativos dentro del código y no generan recursos en la ejecución del programa. El programador los utiliza para agregar información extra como guía en el flujo del programa o recordatorios. Se recomienda utilizarlos porque se pueden explicar detalles de la implementación del programa. Se anteceden por un ";" cuando los comentarios se realizan después de la instrucción. Los comentarios también se pueden realizar al principio de la fila, para ello se utiliza el símbolo "*" previo al comentario.

2.4.1. Directivas

Dentro de la estructura de un código fuente en lenguaje ensamblador existen diferentes tipos de datos para ser procesados, además de símbolos y secciones. Estos elementos se conocen como directivas y existen diferentes tipos, entre los más utilizados se encuentran los siguientes:

1. Directivas que controlan el uso de la sección

2. Directivas que inicializan valores en la memoria

3. Directivas que reservan espacio en la memoria

4. Declaración de símbolo como constante

5. Directivas de símbolos y final del programa

A continuación se describen la directivas descritas anteriormente:

Directivas que controlan el uso de la sección

En este tipo de directivas se define el tipo de sección dentro de un código en ensamblador, por ejemplo, la declaración de variables o la sección del código. En la Tabla 2.2 se muestran las directivas de control de uso de sección [11].

Tabla 2.2: Directivas que controlan el uso de la sección

Mnemónico	Descripción
.data	Se ensambla en la sección *.data* (datos inicializados)
.sect	Se ensambla en una sección nombrada (inicializada)
.text	Se ensambla en la sección *.text* (código ejecutable)

Directivas para inicializar datos en la memoria

En cualquier programa ya sea de lenguaje ensamblador, C o cualquier otro tipo, es importante clasificar los datos que el programador y el usuario proporcionen para poder ejecutar el programa, es decir, la declaración del tipo de datos que se van a utilizar a lo largo del programa. Estos se almacenan en la memoria dependiendo del archivo *cmd* utilizado en el proyecto, como se explicó en la Sección 2.3 [11].

Existen diferentes directivas de inicialización de valores en un programa de ensamblador y el uso de cada una va a depender del tipo de dato que se requiera para el proceso. En el TSM320F28377s se pueden utilizar datos de 16 bits, 32 bits y flotantes. Para poder ser utilizadas es necesario realizar la declaración de la siguiente manera:

nombre ***mnemónico*** *valor o valores* *;comentarios*

En la Tabla 2.3 se muestran las directivas para inicializar un valor en lenguaje ensamblador.

Directivas que reservan espacio en la memoria

En algunos casos, los datos procesados en un programa no necesariamente requieren ser inicializados porque se pueden cargar directamente a la memoria de la tarjeta, son datos

Tabla 2.3: Directivas de lenguaje ensamblador para inicializar datos en la memoria.

Mnemónico	Descripción
.bits	Inicializa uno o más bits sucesivos en la sección actual.
.byte	Inicializa una o mas palabras sucesivas en la sección actual.
.char	Inicializa una o más palabras sucesivas en la sección actual.
.field	Inicializa un campo de bits de tamaño (1-32 bits) con valor.
.float	Inicializa una o más constantes de punto flotante de precisión IEEE de 32 bits.
.int	Inicializa uno o más enteros de 16 bits.
.long	Inicializa uno o más enteros de 32 bits.
.string	Inicializa uno o más cadenas de texto.
.ubyte	Inicializa uno o más palabras sucesivas sin signo.
.ulong	Inicializa uno o más enteros sin signo de 32 bits.
.uword	Inicializa uno o más enteros sin signo de 16 bits.
.word	Inicializa uno o más enteros signados de 16 bits.

resultado o son espacios de memoria para almacenamiento temporal, sin embargo, es necesario reservar el espacio de memoria correspondiente para poder utilizarlo. Por lo tanto, en la Tabla 2.4 se muestran las directivas que se pueden utilizar para reservar memoria.

Tabla 2.4: Directivas que reservan espacio en la memoria.

Mnemónico	Descripción
.bes	Reserva N bits en la sección actual; una etiqueta apunta al final del espacio reservado
.space	Reserva N bits en la sección actual; una etiqueta apunta al principio del espacio reservado

La sintaxis de estas directivas es la siguiente:

Nombre de variable **mnemónico** *Tamaño* *;Comentarios*

Declaración de símbolo como constante

La declaración de un símbolo constante es ampliamente utilizado en un programa fuente en ensamblador. Los símbolos constantes son valores que se declaran al inicio del programa y se mantienen constante a lo largo de todo el código, es decir, no se pueden cambiar, de tal manera que se pueden asignar nombres significativos a expresiones constantes como es el caso del número π o la longitud de una señal [11]. El mnemónico utilizado para asignar un

símbolo a una constante es **.set** y su sintaxis es la siguiente:

Nombre de la constante **.set** *valor* *;comentarios*

Directivas de símbolos y final del programa

En la siguiente Tabla 2.5 se muestran las directivas que se utilizan en los ejemplos a lo largo del presente trabajo para indicar la sección principal del código y el final del mismo.

Tabla 2.5: Directivas de símbolos y final de programa

Mnemónico	Descripción
.def	Identifica uno o más símbolos que son definidos en el módulo actual y que pueden ser utilizados en otros módulos.
.global	Identifica uno o más símbolos globales (externos)
.ref	Identifica uno o más símbolos utilizados en el modulo actual que son definidos en otro módulo.
.end	Fin de ensamblaje

La sintaxis utilizada para las directivas de símbolos es la siguiente:

mnemónico *símbolo*

2.5. Compilación y ejecución de un proyecto

Después de crear y emplear un algoritmo en código en CCS, el siguiente paso es compilar y verificar que éste no contenga errores. Se conoce como *Debugging* al proceso de depuración o identificación y corrección de errores. El CCS en éste proceso, verifica que se haya realizado el ensamble correctamente del código con la descripción de memoria (archivo *.cmd*), así como la configuración de puertos y errores de sintaxis. CCS genera el archivo ejecutable cuando se comprueba que tanto la configuración como el código son correctos.

En la parte superior de la ventana *CCS Edit* del *Code Composer Studio* (ventana de edición del programa) se pueden observar dos iconos llamados *Build* y *Debug* como se muestra en la Figura 2.18. El icono *Build* sirve depurar el programa y construir el ejecutable del mismo, es decir, verifica que no exista algún error en el código, configuración y ensamble con el mapa de memoria. Si existen errores, estos se mostrarán en la ventana de notificaciones. Es necesario resolver los problemas que muestre el compilador, ya que de no ser así, el programa no se podrá ejecutar en el DSP. Después de realizar las correcciones, será necesario compilar

el proyecto nuevamente.

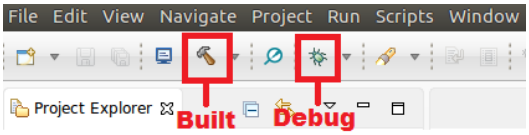


Figura 2.18: Herramienta *'Build'* y *'Debug'*

Una vez resueltos los problemas marcados por el compilador y obtener una compilación exitosa del proyecto, es necesario ejecutar el programa en la tarjeta mediante el icono de *Debug* (Figura 2.18), tomando en cuenta que se tuvo que compilar previamente parra que el compilador construya el ejecutable. Al presionar el botón, aparecerá una ventana emergente que notifica al usuario que se está cargando el proyecto e inmediatamente cambia la interfaz de *CCS Edit* a *CCS Debug* como la que se muestra en la Figura 2.19.

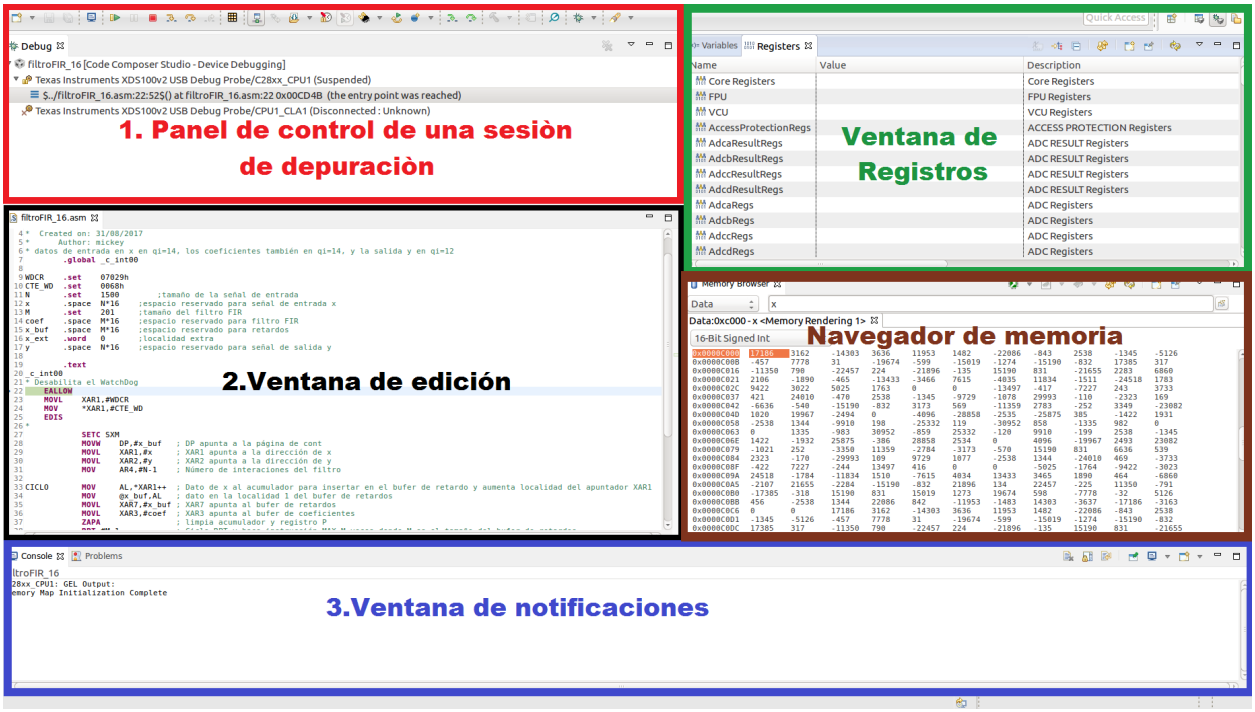


Figura 2.19: Interfaz *CCS Debug* al cargar el programa

Se pueden observar cinco diferentes secciones en la interfaz de *CCS Debug* de la Figura 2.19, las cuales son necesarias para monitorear el comportamiento del sistema, las cuales se

componen de:

1. Panel de control de una sesión de depuración. Cuando se está en una sesión de depuración o *debug*, permite manipular la forma de ejecución del programa en el DSP pausando, reiniciando, ejecutando paso a paso, etcétera.

2. Ventana de edición. Es la ventana donde se tiene acceso a los archivos del proyecto y durante la ejecución de algún programa paso a paso, muestra el lugar en donde el programa se va ejecutando.

3. Ventana de notificaciones. Muestra la actividad del CCS y notificaciones respecto a la compilación de código o la comunicación con el DSP.

4. Ventana de registros. Permite monitorear, y en algunos casos modificar los registros del DSP ya sean del CPU, interrupciones, módulo ADC, módulo PWM, entre otros.

5. Ventana de memoria. Muestra el mapa del contenido de la memorial del DSP.

Finalmente, el programa se puede ejecutar de dos maneras, instrucción por instrucción mediante el icono *Step Into* o presionando la tecla *F5*, o bien puede ejecutarse de forma continua mediante el icono *Resume* o presionando la tecla *F8*. Estos íconos se encuentran en la parte frontal de la ventana *CCS Debug* como se muestra en la Figura 2.20

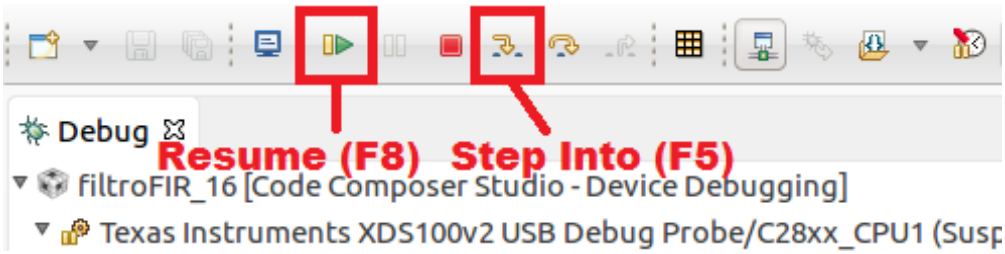


Figura 2.20: Herramienta *'Resume'* (F8) y *'Step Into'* (F5) para ejecución de código

2.6. Herramientas de acceso a memoria y visualización de datos en CCS

El *Code Compose Studio* cuenta con diferentes herramientas que facilitan al programador visualizar datos y el proceso de los mismos. Las dos herramientas más utilizadas en la ejecución de los ejemplos en el presente trabajo son el acceso a los datos de la memoria y

visualización por medio de gráficas una secuencia de datos alojados en la memoria.

El acceso a los datos de la memoria del dispositivo se hace por medio del *Memory Browser* y se puede abrir por medio de la ruta *View* → *Memory Browser* desde la ventana *CCS Debug*. Esto permite visualizar los datos contenidos a lo largo de la memoria, ingresando la dirección que se desea visualizar. De la misma manera se puede seleccionar la forma de visualizar dichos datos, ya sea en palabras de 16 o 32 bits en formatos binario, hexadecimal, flotante o entero signado o sin signo, además del Qi de interés cuando se trabaja en formato de punto fijo.

Además de visualizar datos de la memoria, también una herramienta muy importante en *Code Composer Studio* es poder cargar datos a la memoria del dispositivo desde un archivo de texto con extensión *.dat* y de la misma manera exportar los datos alojados en la misma. A continuación se explica como importar datos externos a la memoria del dispositivo y las características que debe contener el archivo para que sea compatible.

2.6.1. Importación de datos a la memoria

En algunos de los programas de ejemplo de este trabajo se emplean secuencias de datos (señales de prueba, coeficientes de filtros, etc.) para probar el funcionamiento de las operaciones y algoritmos de procesamiento digital programados, por esta razón, a continuación se aborda la forma de importar datos externos a la memoria del DSP TMS320F28377S.

En primera instancia, todos los archivos de datos que se deseen importar deben tener el encabezado como el mostrado a continuación:

No. clave	**Formato**	**Dir. de inicio**	**No. de página**	**No. de datos**

donde:

No. clave; es un indicador de *Texas Instruments* igual a 1651, este número es necesario para que Code Composer Studio identifique que el archivo contiene datos a cargar en la memoria.

Formato; indica el formato en el cual están los datos a cargar en la memoria, este campo toma valores del 1 al 4 dependiendo del tipo de formato como se muestra en la Tabla 2.6.

Dir. de inicio; es la dirección de la localidad de memoria, donde se comenzarán a escribir los datos contenidos en el archivo. Este campo depende directamente de la definición del mapa de memoria que haya sido declarado en el archivo *.cmd* y del orden en que hayan sido declaradas las variables de tipo arreglo, para alojar los datos del archivo. Este campo debe escribirse en hexadecimal.

Tabla 2.6: Formatos de datos soportados por CCS para importarse a la memoria del DSP.

Formato	Tipo	Descripción
1	hex	Datos en formato hexadecimal
2	int	Datos en formato entero con longitud de palabra de 16 bits
3	long	Datos en formato entero con longitud de palabra de 32 bits
4	float	Datos en formato flotante

No. de página; es la página de memoria donde se alojan las localidades destinadas a contener cada dato del archivo *.dat*. La memoria del DSP se distribuye en páginas y posteriormente en secciones.

No. de datos; es la cantidad total de datos a cargar en la memoria, este campo debe escribirse en hexadecimal.

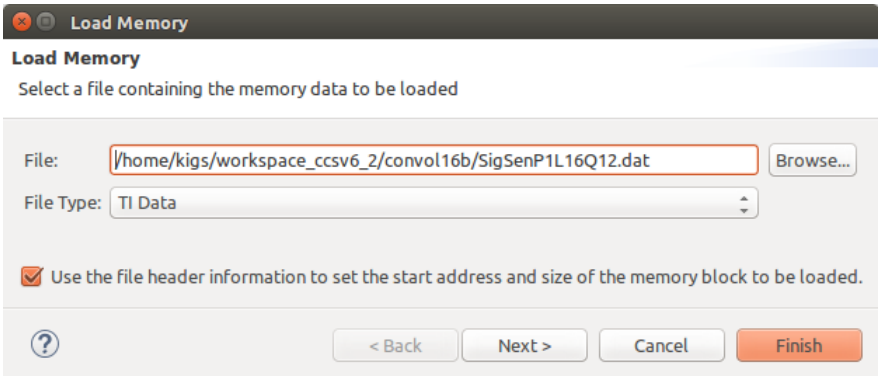
A continuación se muestra un ejemplo de como debe estar escrita la primer línea de un archivo *.dat* para cargar los datos contenidos a la memoria del DSP. Entre cada campo existe un espacio de separación.

1651 2 1A034 1 041A

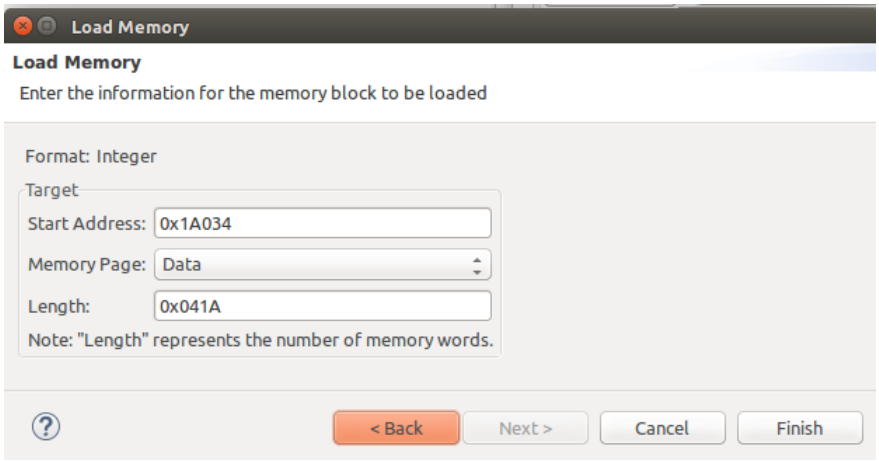
Una vez listo el archivo con el encabezado anterior, para cargar los datos a la memoria se tiene que tener listo un programa compilado y tener conectada la tarjeta Launchpad-F28377S a un puerto USB de la PC, para poder entrar al modo *Debug*, dando click derecho indicado en el icono cuyo dibujo es un escarabajo. Finalizada la escritura del programa en el DSP, como parte del entorno del modo *Debug* deberá aparecer a la derecha de la pantalla, una ventana llamada *Memory Browser* la cual permite ver el contenido de la memoria y escribir datos en ella. Ubicada la ventana *Memory Browser*, se debe de dar click derecho al icono resaltado en la Figura 2.21, para desplegar el menú de opciones y seleccionar *Load Memory* para que se despliegue la ventana mostrada en la Figura 2.22a.

Figura 2.21: Visualizador y editor del contenido de la memoria.

Con el botón de *Browser* se accede al explorador de archivos para poder buscar el archivo que nos interesa, y adicionalmente en la parte inferior izquierda tenemos la opción de habilitar para que lea la información de la primera línea de nuestro archivo y configure la exportación de archivos, en caso de habilitarla, al presionar el botón *Next* nos mostrará la configuración leída del archivo, en caso de no habilitar dicha opción, manualmente se tendrá que ingresar la configuración.



(a) Ventana para cargar datos a la memoria del dsp.



(b) Ventana para cargar datos a la memoria del DSP.

Figura 2.22: Cargar datos a la memoria del DSP.

En la Figura 2.22b se muestra la configuración leída del encabezado del archivo de datos *.dat* seleccionado. Se puede observar que los datos son de tipo entero, la dirección de inicio, la página en donde se encuentra la dirección inicial y la cantidad de datos a cargar. Al presionar el botón *Finish* la IDE CCS cargará los datos a la memoria del DSP.

2.6.2. Visualización de datos

Cuando se carga un programa al DSP y se ejecuta desde *Code Composer Studio*, es posible construir gráficas para visualización de los datos almacenados en la memoria del DSP. La comunicación entre la tarjeta y el software es por medio de la conexión USB. Las gráficas se generan a partir de una cantidad de datos seleccionados en la memoria del DSP especificando el formato de datos utilizado. El Software puede generar hasta seis tipos de gráficas diferentes tanto en el dominio del tiempo, como en el dominio de la frecuencia.

Para generar una gráfica a partir de los datos de la memoria es necesario asegurarse de que el programa se esté ejecutando, posteriormente seleccionar la ruta *Tools*→ *Graph* desde las opciones que se encuentran en la parte superior del software, como se muestra en la Figura 2.23.

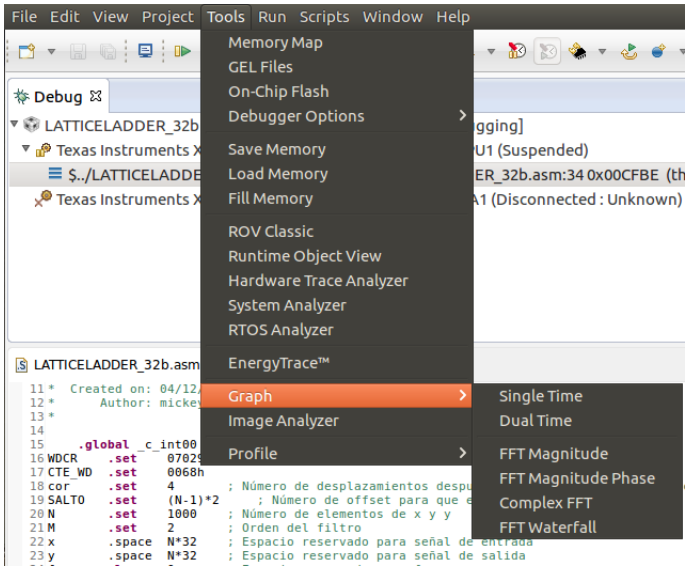


Figura 2.23: Creación de gráfica.

Para graficar una secuencia de tiempo es necesario seleccionar la opción *Single Time*, posteriormente emergerá una ventana como la que se muestra en la Figura 2.24a donde el usuario asigna las características de los datos que contendrá dicha gráfica. Los datos más importantes a ingresar son:

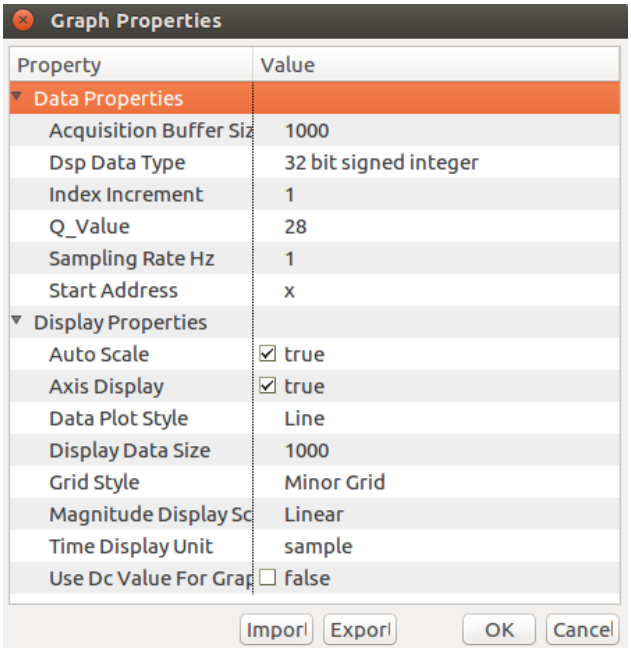
1. Tamaño del buffer de los datos a graficar.
2. Tipo de datos, éstos pueden ser de 64, 32, 26 y 8 bits, enteros signados, sin signo o flotantes.
3. El valor del Qi para el caso de números de punto fijo.

4. La dirección inicial de los datos en la memoria del DSP.

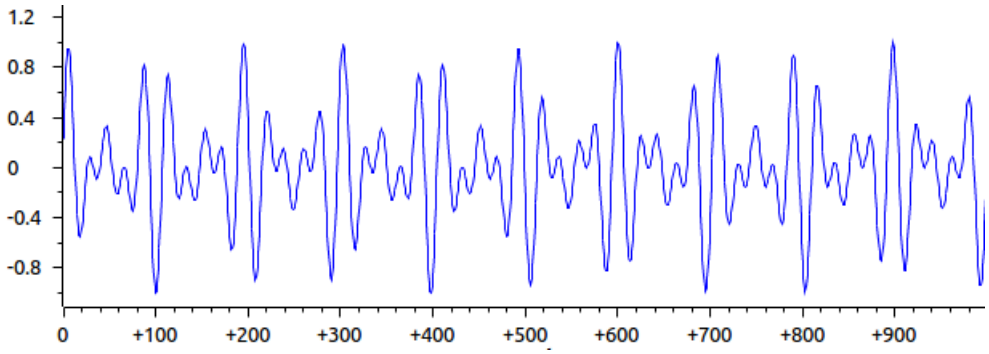
5. Número de datos que el usuario desea visualizar.

6. Cuadrícula y su tamaño.

En la Figura 2.24b se muestra el ejemplo de una gráfica generada con una señal con 1000 localidades de memoria, con datos de 32 bits enteros signados en Qi=28.



(a) Características de la gráfica.



(b) Gráfica de una secuencia de tiempo en *Code Composer Studio*.

Figura 2.24: Gráficación de una secuencia de tiempo en *Code Composer Studio*.

Cuando la interfaz del *Code Composer Studio* genera una gráfica y la ubica en una ventana localizada en la parte inferior de la interfaz, es decir, en la misma sección de notificaciones (ver Figura 2.19). Sin embargo el usuario puede acomodar las ventanas como lo desee.

Para el caso de las gráficas con la opción *Dual time*, genera dos secuencias de tiempo con las mismas características indicando las direcciones de inicio de los datos de cada secuencia.

Para las gráficas en el dominio de la frecuencia el software puede generar las gráficas *FFT Magnitude FFT*, *FFT Magnitude Phase*, *Complex FFT* (real e imaginario de la FFT) y *FFT Waterfall*. Al generar alguna de estas gráficas se requiere la misma información base, es decir:

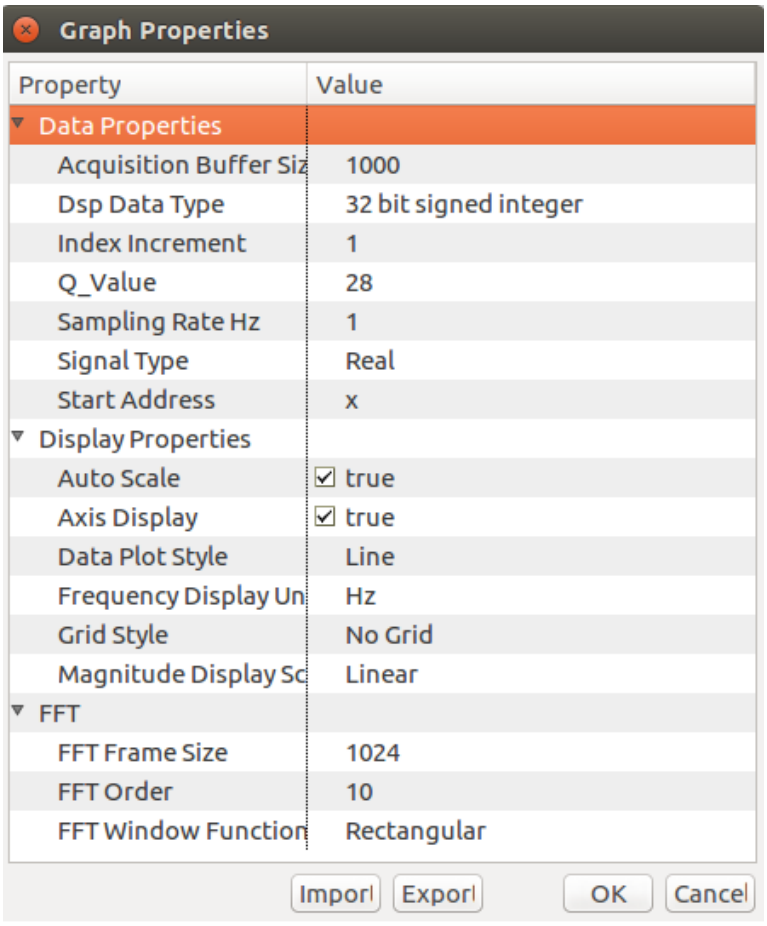
1. Tamaño del buffer de los datos para la gráfica.
2. Tipo de datos, éstos pueden ser de 64, 32, 26 y 8 bits, enteros signados, sin signo o flotantes.
3. El valor del Q_i para el caso de números de punto fijo.
4. Tipo de señal.
5. Dirección de inicio de los datos (para el caso de las gráficas de magnitud y magnitud y fase).
6. Dirección de inicio de los datos reales y dirección de inicio de los datos imaginarios (solo en caso de la gráfica *Complex FFT*).
7. Orden de la FFT.
8. Tipo de ventana para la FFT.

En la Figura 2.25a se muestra la ventana de selección de características para generar una gráfica de la Magnitud de la FFT de un buffer de datos, mientras que en la Figura 2.25b se muestra un ejemplo de la gráfica de magnitud de la FFT de la señal mostrada en la Figura 2.24b por medio de *Code Composer Studio*.

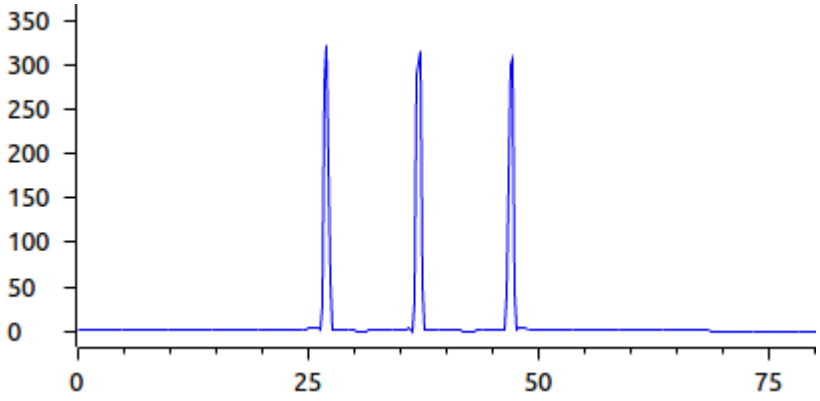
Dependiendo de la configuración de la gráfica, el eje de las abscisas puede verse en Hertz, para ello es necesario indicar la frecuencia de muestreo de la señal en las configuraciones de la gráfica (Figura 2.25a), de no ser así, la frecuencia que representa dicha gráfica es la frecuencia normalizada, de tal manera que esta dada por:

$$\omega = \frac{2\pi f}{f_s} \tag{2.1}$$

donde f es la frecuencia en Hz, y f_s es la frecuencia de muestreo.



(a) Características de la gráfica.



(b) Magnitud de la FFT de una señal en *Code Composer Studio*.

Figura 2.25: Gráficación de la magnitud de la FFT en *Code Composer Studio*.

2.7. Resumen del capítulo

El DSP empleado a lo largo del presente trabajo fue diseñado por *Texas Instruments*, de tal manera que el Software utilizado como herramienta de programación es el *Code Composer Studio*. En este capítulo se ha realizado una explicación de las funciones básicas de dicho software, mostrando la creación de un proyecto utilizando la LAUCHXL-F28377S tanto en lenguaje ensamblador como en lenguaje C y utilizando un mapa de memoria (archivo .cmd). Además, se mostró la utilidad y configuración de CONSTROLSUITE como herramienta de configuración para periféricos y programación en lenguaje C.

Se han mostrado las herramientas con las que cuenta el programa y los dos entornos que éste contiene (CCS Edit y CCS Debug), además de las herramientas para visualización de la memoria (Memory Browser), registros, variables y graficación en el dominio del tiempo y de la frecuencia. En muchas aplicaciones es necesario ingresar los datos de señales o coeficientes a la memoria del DSP, para después acceder a estos datos y realizar el procesamiento respectivo, para ello se explicó las características que debe contener un archivo que contiene los datos a importar en la memoria del DSP.

Capítulo 3

Formatos numéricos

En el diseño e implementación de un sistema de procesamiento digital de señales (PDS) en una arquitectura dedicada como un procesador digital de señales (DSP), los algoritmos son implementados en hardware con arquitecturas de longitud de palabra finita, por tanto, debe existir un compromiso entre el intervalo dinámico de las variables y la precisión de las mismas para evitar errores de precisión numérica. Esta característica es de suma importancia en procesadores de punto fijo, sin embargo, también es válido en procesadores de punto flotante [12].

En los formatos de representación numérica es importante tener presente algunos parámetros del sistema para lograr mejores desempeños, estos son, *el intervalo dinámico de las variables, la precisión numérica y la resolución*:

- *El intervalo dinámico (ID)* de un sistema numérico se define como la diferencia entre el número mayor y el menor que se pueda expresar (excluyendo al cero) en el formato, es decir la magnitud numérica más grande a representar. Cualquier número fuera de este intervalo se considera como sobreflujo. El ID en decibeles se expresa [13]:

$$ID_{db} = 20log_{10}\left(\frac{Max}{Min}\right) \tag{3.1}$$

  Por ejemplo, en un formato de punto fijo a 16 bits su intervalo dinámico es de 90 dB.

- *La precisión numérica (p)*, es el número real más pequeño a representar, se define como la diferencia entre dos números consecutivos y está determinado por el bit menos significativo (LSb).

- *Resolución (Δ)*, si con L bits se puede representar una variable X con un valor máximo y un mínimo, entonces la resolución es [14]:

$$\Delta = \frac{Xmax - Xmin}{2^L - 1} \tag{3.2}$$

Las computadoras y los sistemas digitales trabajan internamente con números representados en binario, los formatos de representación más utilizados son: punto fijo y punto flotante, los cuales sirven para clasificar a las máquinas digitales.

3.1. Formato numérico de punto fijo Q_i

La representación numérica en formato digital de punto fijo es similar a la representación de números reales, es decir, como un conjunto de dígitos con un punto decimal. En este formato la cantidad total de bits de la palabra digital se divide en:

- Q_E: número de bits para la parte entera
- $Q_F = Q_i$: los bits para la parte fraccionaria
- S: un bit de signo

es decir, que la longitud de la palabra se compone por la suma del número de cada parte, la cual se puede expresar como:

$$L = 1 + Q_E + Q_F \tag{3.3}$$

Como se muestra en la Figura 3.1, entre la parte entera y la parte fraccionaria existe un *punto hipotético*, el cual se encuentra *fijo* y que sólo lo interpreta el programador.

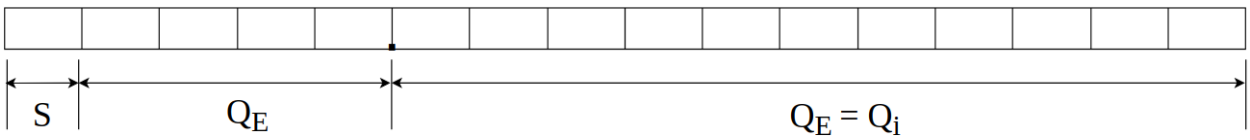


Figura 3.1: Secciones en las que se divide un número binario en el formato Q_i.

En general cualquier número real X expresado en punto fijo y complemento a dos se puede escribir para $Q_F \neq 0$ [12]:

$$X = (-1)^S b(Q_E + Q_F) + \sum_{i=1}^{Q_E} b(Q_F - 1 + i)2^{i-1} + \sum_{i=1}^{Q_F} b(Q_F - i)2^{-i} \tag{3.4}$$

donde y $b(.)$ es un número binario 0 o 1 en la posición correspondiente. Recordando que el bit más significativo (MSb) corresponde al signo, y está en la posición $L-1$, en el caso de números positivos $S = 0$, mientras que para números negativos $S = 1$. En (3.4) las posiciones binarias $b(.)$ están reflejadas a la forma usual de referirse a una palabra digital de longitud L, es decir, $b_{L-1}, b_{L-2}, b_{L-3}, \cdots, b_1, b_0$. Como se mencionó anteriormente, en este formato el punto decimal no se encuentra explícitamente en su representación binaria, por lo que un número puede representar diferentes valores dependiendo de la posición hipotética del punto decimal, por ejemplo, el número "*01101101*" puede ser interpretado de forma diferente de acuerdo al Q_i deseado:

$$\begin{aligned} Q_3 =& 01101.101 = 0x2^4 + 1x2^3 + 1x2^2 + 0x2^1 + 1x2^0 + 1x2^{-1} \\ &+ 0x2^{-2} + 1x2^{-3} = 13.625 \end{aligned}$$

$$\begin{aligned} Q_4 =& 0110.1101 = 0x2^3 + 1x2^2 + 1x2^1 + 0x2^0 + 1x2^{-1} + 1x2^{-2} \\ &+ 0x2^{-3} + 1x2^{-4} = 6.8125 \end{aligned}$$

$$\begin{aligned} Q_5 =& 011.01101 = 0x2^2 + 1x2^1 + 1x2^0 + 0x2^{-1} + 1x2^{-2} + 1x2^{-3} \\ &+ 0x2^{-4} + 1x2^{-5} = 3.40625 \end{aligned}$$

A continuación se muestran las operaciones básicas que se pueden realizar empleando aritmética de punto fijo.

3.1.1. Operación Suma

Para realizar sumas empleando aritmética de punto fijo, es necesario que los números tengan el mismo *Qi*, en caso contrario se deberán realizar corrimientos a la derecha o izquierda, según sea el caso, para igualar los *Qi*, este proceso se muestra en la Figura 3.2. Por ejemplo, si se tienen dos números $A = 3.170$ y $B = 2.025$, cuya representación en aritmética de punto fijo empleando una longitud de palabra de 16 bits y $Q_i = 12$ queda como:

- $3.170 = 0011.001010111000$
- $2.025 = 0010.000001100110$

Y se requiere obtener su suma (cuyo resultado debe de ser 5.195):

Tabla 3.1: Ejemplo de suma en punto fijo.

Formato	**Número binario**	**Número en Qi=12**	**Número en decimal**
+	0011.001010111000	12984	3.169921875
	0010.000001100110	8294	2.024902344
Total =	0101.001100011110	21278	5.194824219

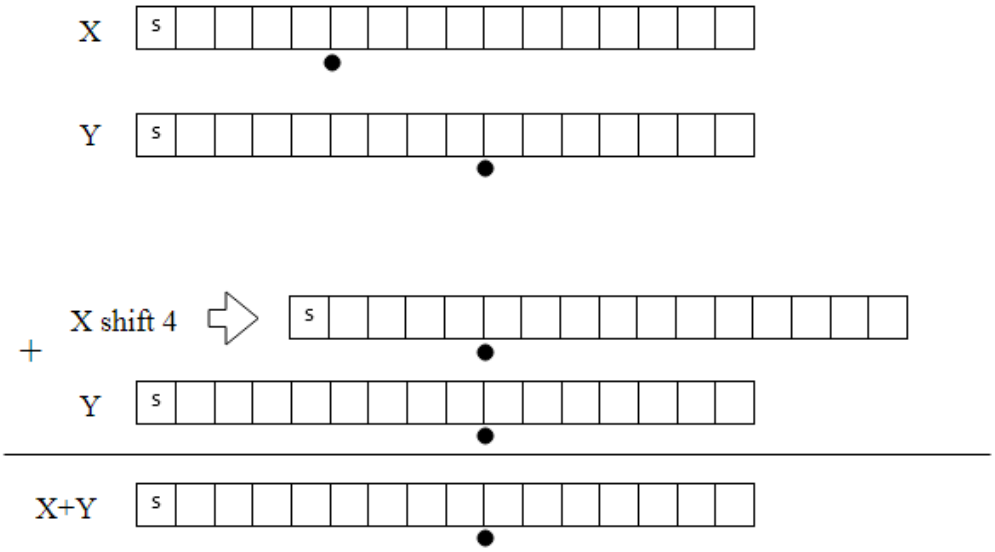


Figura 3.2: Suma en aritmética de punto fijo.

Cabe destacar que el empleo de este tipo de aritmética conlleva errores de precisión numérica la cual se ve reflejada en el resultado de la operación como se observa en la Tabla 3.1, estos errores fueron: $78.125x10^{-6}$, $97.656x10^{-6}$, para los operandos y de $175.781x10^{-6}$ para el resultado de la suma, así mismo, se debe tener cuidado de que el resultado no sobrepase el ID.

A continuación se muestra un programa en lenguaje ensamblador que realiza la suma de dos números en formato de punto fijo.

```
*
* Suma de dos números empleando
* aritmérica de punto fijo
*
* L  = 16
* Qi =  12
*
* El resultado queda en Q12 con L=16 en Al
* y se mueve a la localidad "z"

        .global     _c_int00

x       .word 12984     ;3.170 Q12
y       .word 8294      ;2.025 Q12
z       .word 0         ;para almacenar el resultado
        .text

_c_int00
        SETC SXM        ; Activación del modo extensión de signo
                        ; para aritmética
        SETC OVM        ; Habilitación de operaciones en modo overflow
        SPM 0           ; Corrimientos nulos en el registro P


        MOVL XAR1,#x    ; Apunta a la dirección de x
        MOVL XAR2,#y    ; Apunta a la dirección de y
        MOVL XAR3,#z    ; Apunta a la dirección de z

        MOV AL,*XAR1    ; AL=x
        ADD ACC,*XAR2   ; AL=AL+y

        MOV *XAR3,AL    ; z=AL

loop
        NOP
        LB loop         ; Ciclo infinito
        .end
```

3.1.2. Operación Multiplicación

Al realizar multiplicaciones en aritmética de punto fijo con números de longitud de palabra L, el resultado tendrá una longitud de palabra de $2L$ y con un $Q_i = Q_{ix} + Q_{iy}$, por lo que se deben de truncar los últimos L bits del resultado, en la Figura 3.3 se muestra el proceso para realizar multiplicaciones.

De la Figura 3.3, se observa que al truncar el resultado de la multiplicación, éste queda en un Q_i menor que el inicial, sin embargo, antes de realizar el truncamiento se pueden realizar

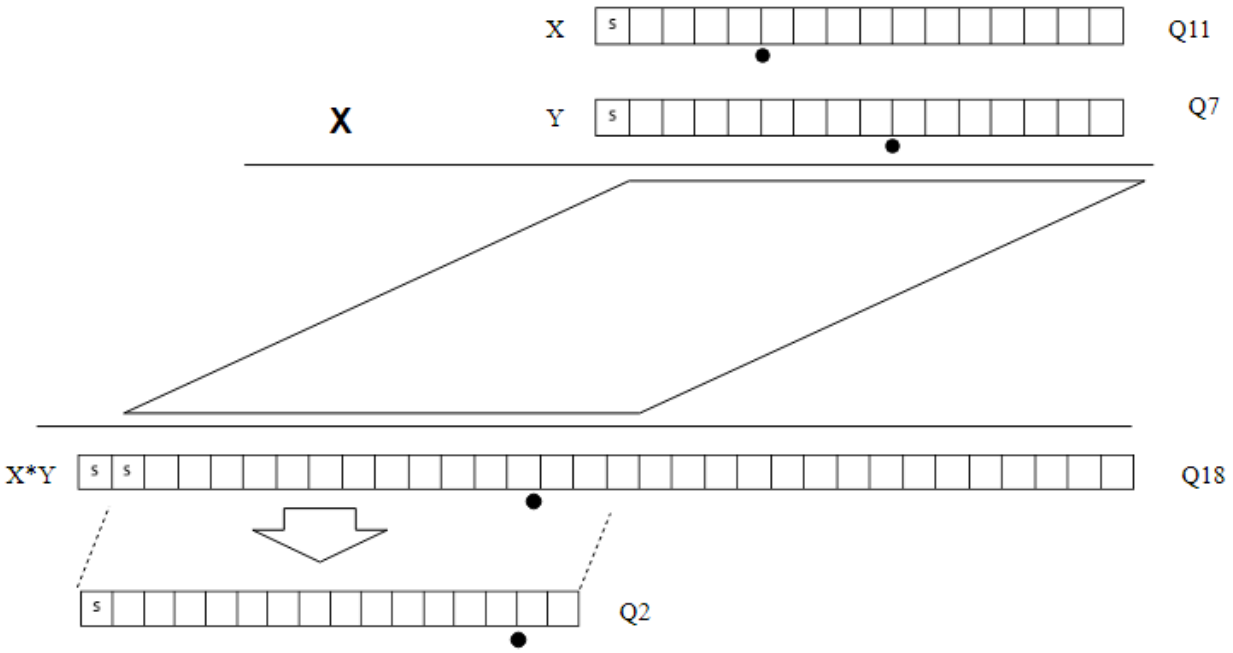


Figura 3.3: Multiplicación en aritmética de punto fijo.

corrimientos para obtener un Qi diferente[1]. Por ejemplo, si se desean multiplicar los números 1.25 y 2.12 (cuyo resultado debe ser 2.65), empleando una longitud de palabra $L = 8$ con Q_5:

- $1.25 = 001.01000$
- $2.12 = 010.00011$

En la Tabla 3.2 se muestra el resultado de la multiplicación en $L = 16$ y Q_{10}, por lo que para obtener el resultado en Q_5 se debe realizar un corrimiento a la izquierda de 3 bits, para después realizar el truncamiento de los 8 bits menos significativos, obteniéndose como resultado 2.59375.

Al igual que la suma, se tienen errores de precisión numérica de 0.0 y $27x10^{-3}$ para los operandos y de $56.25x10^{-3}$ para el resultado en Q_5. Cabe destacar que el resultado original de la multiplicación tiene mejor precisión numérica que el obtenido después del truncamiento, esto se debe a que se tiene un mayor número de bits en la parte fraccionaria, sin embargo, en muchas aplicaciones se prefiere obtener el resultado en el Q_i original.

[1]Mientras la parte entera del resultado pueda ser representada.

Binario	Entero	Representación
001.01000	40 Q_5	1.25
010.00011	67 Q_5	2.093
$\overbrace{\underbrace{000010.10}_{L=8,Q_2}\underbrace{01111000}_{L=8}}^{L=16} << 3$	2680 Q_{10}	2.6171875
$\underbrace{010.10011}_{L=8,Q_5}$	83 Q_5	2.59375

Tabla 3.2: Ejemplo de multiplicación en punto fijo.

A continuación se muestra un programa en lenguaje ensamblador que realiza la multiplicación de dos números en formato de punto fijo.

```
*
* Multiplicación de dos números empleando
* aritmérica de punto fijo
*
* L  = 16
* Qi =  5
*
* El resultado queda en Q10 con L=32 en ACC
* por lo que se corre 11 bits el resultado
* para que en AH quede el resultado en Q5
* y se mueve a la localidad "z"

        .global     _c_int00

x       .word 40        ; 1.25 Q5
y       .word 67        ; 2.12 Q5
z       .word 0         ; Varible para guardar resultado
        .text

_c_int00
        SETC SXM        ; Activación del modo extensión de
                        ; Signo para aritmética
        SETC OVM
        SPM 0           ; Corrimientos nulos en el reg. P


        MOVL XAR1,#x    ; Apunta a la dirección de x
        MOVL XAR2,#y    ; Apunta a la dirección de y
        MOVL XAR3,#z    ; Apunta a la dirección de z

        MOV T,*XAR1     ; T=x
        MPY P,T,*XAR2   ; P=x*y
```

```
        MOVL ACC,P      ; ACC=P
        LSL ACC,#11     ; ACC=ACC<<11
        MOV *XAR3,AH    ; z=AH

loop
        NOP
        LB loop         ; Ciclo infinito
        .end
```

3.2. Formato numérico de punto flotante

El formato de punto flotante se emplea en aplicaciones donde las señales tienen un intervalo dinámico muy grande, esto se debe a que la resolución decrece conforme incrementa el tamaño de los números sucesivos [15]. Esta representación se basa en que cualquier número puede ser expresado como:

$$X = M2^E \tag{3.5}$$

donde M es la *mantisa* y E es el *exponente*.

En la mayoría de las aplicaciones científicas y de ingeniería se emplea el estándar IEEE 754, el cual especifica cuatro formatos de punto flotante [16]:

- Precisión simple a 32 bits, Figura 3.4a.
- Precisión simple extendida ≥43 bits. Se utilizan para cálculos intermedios para evitar que los resultados finales de las operaciones se deterioren debido a errores de redondeo.
- Precisión doble a 64 bits, Figura 3.4b.
- Precisión doble extendida ≥79 bits.

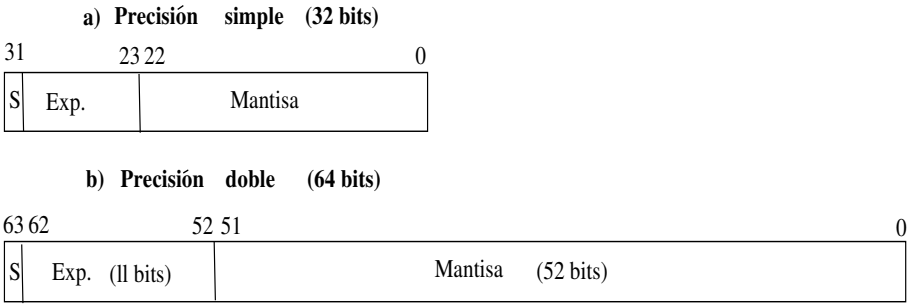


Figura 3.4: Formatos numéricos de punto flotante, IEEE 754

El formato de punto flotante de precisión simple es ampliamente utilizado en máquinas de 32 bits, donde un número de punto flotante se representa como:

$$X = (-1)^S M 2^{E-127}$$

donde $0.M$ es una fracción y $1.M$ es un número mixto con un bit para la parte entera (bit implícito) y 23 bits para la parte fraccionaria. El intervalo dinámico de esta representación es de:

$$\pm 2^{-126} \quad a \quad \pm\,(2 - 2^{-23}2^{127}) \qquad o \qquad \pm\,1.18x10^{-38} \quad a \quad \pm\,3.40x10^{38}$$

y su precisión $p = 2^{-23}$ es decir, que cualquier número fuera de este intervalo resulta en sobreflujo.

Por ejemplo, la representación del número 13.625 en forma binaria empleando el formato IEEE 754 se realiza de la siguiente manera:

1. Representar empleando el mejor Q_i posible: $13.625 \Leftrightarrow 1101.101$.
2. Representar el número en notación científica base dos: $1101.101x2^0$.
3. Normalizar la mantisa realizando corrimientos y ajustar el exponente: $1101.101x2^0 \Leftrightarrow 1.101101x2^3 = 1.703125x2^3$.
4. Dado que $E - 127 = 3$, el exponente será $130 = 10000010$.

Siguiendo estos pasos, el número quedará representado como[2]:

Signo	**Exponente**	**Mantisa**
0	10000010	10110100000000000000000

Tabla 3.3: Representación del número 13.625 en formato IEEE 754.

Este estándar incluye representaciones especiales tanto para el cero como para *no-números* como son [15]:

- Si $E = 255$ y $M \neq 0$, entonces X no es un número (NAN).
- Si $E = 255$ y $M = 0$, entonces $X = \pm\infty$ (dependiendo del bit de signo).
- Si $E = 0$ y $M \neq 0$, entonces $X \simeq 0$.
- Si $E = 0$ y $M = 0$, entonces $X = 0$.

[2]El bit que se encuentra a la izquierda del punto decimal en el tercer paso se denomina *bit implícito* y NO aparece en la representación binaria.

3.2.1. Operación Suma

Para realizar la suma de dos números representados mediante punto flotante, primero se requiere que ambos tengan el mismo exponente (E), esto se logra haciendo un ajuste en uno de los exponentes, realizando corrimientos en la mantisa (M). Una vez que los exponentes son iguales, se realiza la suma de las mantisas y posteriormente se reajusta el exponente. Por ejemplo, si se desean sumar los números 13.625 y 10.250 cuya representación está dada por:

Número	**Signo**	**Exponente**	**Mantisa**
13.625	0	10000010	10110100000000000000000
10.250	0	10000010	01001000000000000000000

En notación científica y agregando el bit implícito, estos números quedan como:

1.10110100000000000000000	$x2^3$
1.01001000000000000000000	$x2^3$
10.11111100000000000000000	$x2^3$
1.01111110000000000000000	$x2^4$

Tabla 3.4: Ejemplo de suma en punto flotante.

Donde el resultado es equivalente a $1.4921875x2^4 = 23.875$, el cual queda representado como:

Número	**Signo**	**Exponente**	**Mantisa**
23.875	0	10000011	01111110000000000000000

De forma general, la suma de dos números $X = (-1)^{s_x} X_M 2^{E_x}$ y $Y = (-1)^{s_y} Y_M 2^{E_y}$, cuando $E_y > E_y$, se puede expresar como:

$$X + Y = \left[(-1)^{s_x} X_M + (-1)^{s_y} Y_M >> (E_x - E_y)\right] 2^{E_x} \tag{3.6}$$

A continuación de muestra un programa en lenguaje ensamblador que realiza la suma de dos números en formato de punto flotante.

```
*
* Suma de números empleando
* aritmética de punto flotante IEEE 754
*

        .global     _c_int00

x       .float 13.625           ; IEE 754
y       .float 10.250           ; IEE 754
z       .float 0                ; Para almacenar el resultado
        .text

_c_int00
        SETC SXM                ; Activación del modo extensión de
                                ; Signo para aritmética
        SETC OVM
        SPM 0                   ; Corrimientos nulos en el reg. P

        MOVL XAR1,#x            ; Apunta a la dirección de x
        MOVL XAR2,#y            ; Apunta a la dirección de y
        MOVL XAR3,#z            ; Apunta a la dirección de z

        MOV32 R0H,*XAR1         ; R0H = x
        MOV32 R1H,*XAR2         ; R1H = Y

        ADDF32 R2H,R1H,R0H      ; R2H=x+y
        NOP

        MOV32 *XAR3,R2H          ; z=R2H

loop
        NOP
        LB loop                 ; Ciclo infinito
        .end
```

3.2.2. Operación Multiplicación

La multiplicación de números en punto flotante se realiza multiplicando las mantisas[3] y sumando los exponentes de ambos números, es decir:

$$XY = \left[(-1)^{s_x} X_M \cdot (-1)^{s_y} Y_M\right] 2^{E_x + E_y} \tag{3.7}$$

Por ejemplo, para obtener la multiplicación de los números $X = 1.25$ y $Y = 22.50$, se tendría que[4]

[3]Incluyendo el bit implícito.

[4]El bit implícito se muestra en negritas.

Número	Signo	Exponente	Mantisa
1.25	0	01111111 (0)	**1.**01000000000000000000000
22.50	0	10000011 (4)	**1.**01101000000000000000000
28.125	0	10000011 (4)	**1.**11000010000000000000000

A continuación de muestra un programa en lenguaje ensamblador que realiza la multiplicación de dos números en formato de punto flotante.

```
; Multiplicación de números empleando
; aritmética de punto flotante IEEE 754

        .global     _c_int00

x       .float 1.25     ; IEE 754
y       .float 22.5     ; IEE 754
z       .float 0        ; Para almacenar el resultado

        .text
_c_int00
        SETC SXM        ; Activación del modo extensión de signo
                        ; para aritmética
        SETC OVM
        SPM 0           ; Corrimientos nulos en el registro P


        MOVL XAR1,#x    ; Apunta a la dirección de x
        MOVL XAR2,#y    ; Apunta a la dirección de y
        MOVL XAR3,#z    ; Apunta a la dirección de z

        MOV32 R0H,*XAR1         ;R0H = x
        MOV32 R1H,*XAR2         ;R1H = Y

        MPYF32 R2H,R1H,R0H      ;R2H=x*y
        NOP

        MOV32 *XAR3,R2H         ;z=R2H



loop
        NOP
        LB loop         ; Ciclo infinito
        .end
```

3.3. Conversión entre formatos numéricos

En el siguiente ejemplo se muestra conversiones de formatos de punto flotante a punto fijo y viceversa utilizando el DSP.

```
; Este programa emplea dos números en punto flotante IEE 754
; los convierte a punto fijo L=32 Q16, realiza la suma de
; dichos números en ambos formatos y posteriormente
; el resultado en punto fijo lo convierte a punto flotante

        .global     _c_int00

xF      .float 13.625 ; IEE 754
yF      .float 10.250 ; IEE 754
zF      .float 0      ; Para almacenar el resultado IEEE 754

xQ      .long  0      ; Para almacenar x L=32 Q16
yQ      .long  0      ; Para almacenar y L=32 Q16
zQ      .long  0      ; Para almacenar el resultado L=32 Q16


        .text
_c_int00
        SETC   SXM          ; Activación del modo extensión de
                            ; signo para aritmética
        SETC   OVM
        SPM    0            ; Corrimientos nulos en P


        MOVL   XAR1,#xF             ; Apunta a la dirección de xF
        MOVL   XAR2,#yF             ; Apunta a la dirección de yF
        MOVL   XAR3,#zF             ; Apunta a la dirección de zF

        MOVL   XAR4,#xQ             ; Apunta a la dirección de xQ
        MOVL   XAR5,#yQ             ; Apunta a la dirección de yQ
        MOVL   XAR6,#zQ             ; Apunta a la dirección de zQ

        MOV32  R0H,*XAR1            ; R0H = xF
        MOV32  R1H,*XAR2            ; R1H = YF


        MPYF32 R2H,R0H,#65536.0     ; R2H = xF*(1<<16)
        MPYF32 R3H,R1H,#65536.0     ; R3H = yF*(1<<16)

        F32TOI32      R4H,R2H       ; R4H = xQ16 Convierte flotante
                                    ; a entero de L=32
        F32TOI32      R5H,R3H       ; R5H = yQ16
```

```
        MOV32   *XAR4,R4H               ; xQ=R4H
        MOV32   *XAR5,R5H               ; yQ=R5H

        ADDF32  R6H,R0H,R1H             ; R6H=xF+yF
        MOVL    ACC,*XAR4               ; ACC=xQ
        ADDL    ACC,*XAR5               ; ACC=ACC+yQ

        MOV32   *XAR3,R6H               ; zF=xF+yF
        MOVL    *XAR6,ACC               ; zQ=xQ+yQ

        I32TOF32        R0H,*XAR6       ; R0H = zQ Convierte entero
                                        ; L=32 a flotante
        NOP
        MPYF32  R7H,R0H,#(1/65536.0)    ; R7H = zQ*(1>>16) = zF

loop
        NOP                             ; Ciclo infinito
        LB      loop
```

3.4. Resumen del capítulo

Se abordaron los tipos de formatos numéricos utilizados en los datos para un procesamiento digital adecuado con microcontroladores y DSP. La longitud de palabra empleada en una aplicación depende principalmente de las necesidades de la misma, es decir, la precisión numérica y el tiempo de ejecución requerida. El presente capítulo se centró principalmente en hacer la diferencia entre los dos tipos de formato numérico utilizados en la arquitectura TMS320F283777, formato en punto fijo de 16 y 32 bits y formato flotante estándar IEEE 754. Además, se abordaron los tipos de operaciones base en procesamiento digital de señales, como es el caso de la suma y multiplicación tomando en cuenta el Qi utilizado en longitud de palabra de 16 y 32 bits, con la finalidad de tener una mejor precisión numérica en todo momento. Finalmente, se explica la conversión de datos entre los formatos numéricos (punto fijo y flotante), la cual es muy útil en la implementación de diversos algoritmos.

Ejercicios propuestos

1. Generar una señal conocida en cualquier software, almacenarla en un archivo de datos (con extensión *.dat*) en un formato numérico. Cargar dicha señal a la memoria del DSP y mostrar la serie de tiempo de dicha señal desde el CCS y su respectiva gráfica de Magnitud de la respuesta en frecuencia.

2. Dada la señal anterior en memoria del DSP, salvar esta información en diferentes formatos numéricos en un archivo de datos (con extensión *.dat*).

3. Dada la secuencia de números:

$$x(n) = \{1.3798, -7.88988, 5.2351, -18.987402, 0.999786, 11.349152, \\ -3.799823, 10.44568\}$$

   a) Calcular el valor medio utilizando un formato de punto fijo con L=16 y 32 bits utilizando la mejor precisión.

   b) Calcular el error cometido en el cálculo anterior.

4. Dados los siguientes puntos en coordenadas cartesianas:

$$p_1 = (5.27834, -4.43245)$$

$$p_2 = (13.996823, 2.98764)$$

$$p_3 = (-0.873423, 8.915299)$$

$$p_4 = (9.987651, -5.87923)$$

   a) Calcular la distancia cuadrática de cada punto al origen, utilizando un formato de punto fijo con L=16 bits utilizando la mejor precisión.

   b) Calcular la distancia cuadrática de cada punto al origen, utilizando un formato de punto fijo con L=32 bits utilizando la mejor precisión.

   c) Calcular la distancia cuadrática de cada punto al origen, utilizando un formato de punto flotante

   b) Calcular el error en cada caso utilizando un software de cálculo matemático.

Capítulo 4

Algoritmos y operaciones de PDS

El contenido de este capítulo consiste en diferentes implementaciones desarrolladas en el TMS320F28377S y válidas para la familia TMS320F28xxxx, de algunas operaciones y algoritmos fundamentales en el área de procesamiento digital de señales, como lo es la operación correlación, filtros digitales, la transformada discreta de Fourier, entre otros.

Cada sección aborda una aplicación diferente, comenzando con un breve marco teórico para explicar el contexto y proceder a describir la implementación, en cuanto al uso de señales de trabajo así como la metodología propuesta para el desarrollo de los diferentes programas que se presentan, utilizando los formatos de punto fijo y punto flotante con la una Unidad de Punto Flotante (FPU) del DSP. Antes de finalizar cada sección, se evalúan los resultados obtenidos de las aplicaciones en punto fijo a 16 y 32 bits y punto flotante, proyectando la influencia de la precisión numérica. Dichos resultados fueron comparados en cada caso con los obtenidos en simulaciones realizadas en Octave el cual utiliza doble precisión numérica.

Los archivos de datos que contienen las señales utilizadas en los ejemplos del presente capítulo, se ingresan en la memoria como se explicó en la Sección 2.6.1, están disponibles en el sitio web del Laboratorio de Procesamiento Digital de Señales de la Facultad de Ingeniería de la UNAM `http://odin.fi-b.unam.mx/labdsp/ManualDelfino`.

4.1. Producto punto entre vectores

Una operación fundamental en el área de procesamiento digital de señales, es la operación convolución, la cual, en términos de implementación, se realiza como la suma de productos de secuencias de datos finitos, es decir, que en un instante de tiempo discreto n, se calcula un producto punto entre dos vectores de longitud N, razón por la que a continuación se verá la implementación de esta última operación.

Entonces, dados dos vectores $\mathbf{x}$ y $\mathbf{h}$ de la misma longitud, el producto punto se denota como $y = \mathbf{x^T h} = \mathbf{h^T x}$, desarrollando la ecuación anterior tenemos

$$y = h_0 x_0 + h_1 x_1 + \cdots + h_{N-1} x_{N-1} = \sum_{i=0}^{N-1} h_i x_i \tag{4.1}$$

Describiendo textualmente la Ecuación (4.1), el producto punto entre dos vectores se define como la suma de cada uno de los productos realizados entre los elementos de cada vector, tomando en cuenta que cada producto se realiza con dichos elementos que tienen la misma posición dentro de los vectores, dando como resultado un escalar. De forma similar, en un sistema tipo filtro de respuesta finita al impulso (FIR), la salida $y\,(n)$ en un tiempo n se calcula como la convolución lineal de la entrada $x\,(n)$ con la respuesta al impulso $h\,(n)$.

Implementación del producto punto

A continuación se presentan tres programas que calculan el producto punto entre dos vectores, manejando aritmética de punto fijo con datos de 16 bits y 32 bits, además de punto flotante utilizando la FPU del TMS320F28377S.

4.1.1. Producto punto entre vectores en punto fijo a 16 bits

En este programa se consideró utilizar un formato de punto fijo de $Q_i = 0$, es decir, que se los datos de entrada serán enteros sin cifra decimal alguna, con el objetivo de que el lector pueda corroborar el resultado calculado por el código. Entonces, se definen en la memoria del DSP dos vectores x y h cada uno con 10 elementos, cuyos valores permiten al lector desarrollar el producto punto de forma escrita y el resultado de la operación se guarda en la variable llamada *total*.

```
*
*          Producto Punto entre Vectores
* Considerando datos con una longitud de 16 bits
*

        .global     _c_int00

* Declaración de variables globales del programa
x       .word 1,2,3,4,5,6,7,8,9,10                  ; Vector x
h       .word 1,2,1,2,1,2,1,2,1,2                   ; Vector y

total   .word 0                 ; Variable para guardar el resultado
N       .set 10                 ; Cantidad de elementos por vector
WDCR    .set 07029h             ; Dirección del registro de control
                                ; del WatchDog
        .text

* Inicio del programa
_c_int00
        EALLOW                  ; Habilitación de escritura a
                                ; registros protegidos
        MOVL XAR1,#WDCR         ; Direccionamiento al reg. WDCR
        MOV *XAR1,#0068h        ; Deshabilita el watchDog
        EDIS                    ; Deshabilita la escritura en
                                ; registros protegidos

        SETC SXM                ; Modo extensión de signo
        SPM 0                   ; Corrimientos nulos en P
        MOVW DP,#total          ; Direccionamiento a la pag. de
                                ; memoria donde está declarada la
                                ; variable total
        MOVL XAR1,#x            ; Direccionamiento indirecto a x
        MOVL XAR7,#h            ; Direccionamiento indirecto a h
        ZAPA                    ; ACC=0 y P=0

        RPT #N-1                        ; Ciclo de repeticiones
        || MAC P,*XAR1++,*XAR7++        ; Operación acumulación
                                ; y multiplicación -> ACC = ACC + P
                                ; T = dato apuntado por XAR1
                                ; P = T * (dato apuntado por XAR7)
                                ; XAR1 = XAR1 + 1
                                ; XAR7 = XAR7 + 1
        ADDL ACC,P<<PM          ; Acumulación del último producto
        MOV @total,AL           ; Escribe el resultado de la
                                ; operación en la variable total

FIN_R   NOP
        LB FIN_R                ; Ciclo infinito
        .end                    ; Fin del programa
```

El resultado del producto punto entre los vectores x y h es igual a 85, número que está en formato de punto fijo $Q_i = 0$. De forma adicional uno puede cargar vectores a operar con una mayor longitud, utilizando el procedimiento de importación de datos a la memoria del DSP y también se podrían utilizar datos con punto decimal siempre y cuando se conviertan al formato de punto fijo más adecuado y se tenga en cuenta el formato en el que se obtendrá el resultado, como se explicó en el Capítulo 3. Estas consideraciones podrán observarse en la segunda implementación.

4.1.2. Producto punto entre vectores en punto fijo a 32 bits

Con la finalidad de demostrar el uso de diferentes formatos de punto fijo, en esta implementación de 32 bits, se definen como vectores de entrada x y h, cada uno con cuatro elementos convertidos a un formato $Q_i = 24$, sin importar que no tengan cifras decimales, esto es con la finalidad de que se pueda corroborar el resultado del programa que se presenta a continuación.

```
*
*          Producto Punto entre Vectores
* Considerando datos con una longitud de 32 bits
*
          .global _c_int00

* Declaración de variables globales del programa

* Definición del vector x = [1,2,3,4] con un Qi=24.
x         .long 16777216,33554432,50331648,67108864

* Definición del vector h = [1,2,1,2] con un Qi=24.
h         .long 16777216,33554432,16777216,33554432

total     .long 0                   ; Variable para guardar el resultado
N         .set 4                    ; Cantidad de elementos por vector
WDCR      .set 07029h               ; Dirección del registro de control
                                    ; del WatchDog
          .text

* Inicio del programa
_c_int00
          EALLOW                    ; Habilitación de escritura a
                                    ; registros protegidos
          MOVL XAR1,#WDCR           ; Direccionamiento al registro WDCR
          MOV *XAR1,#0068h          ; Deshabilita el watchDog
          EDIS                      ; Deshabilita la escritura en
```

```
                                ; registros protegidos

        SETC SXM                ; Modo extensión de signo
        SPM 0                   ; Corrimientos nulos en P
        SETC OVM                ; Habilita el modo de desbordamiento
        MOVW DP,#total          ; Direccionamiento del DP a la
                                ; página de memoria que contiene a
                                ; total donde está declarada la
                                ; variable total

        MOVL XAR1,#x            ; Direccionamiento indirecto a x
        MOVL XAR7,#h            ; Direccionamiento indirecto a h
        ZAPA                    ; ACC=0 y P=0

        RPT #N-1                            ; Ciclo de repeticiones
        || QMACL P,*XAR1++,*XAR7++          ; Operación acumulación
                                ; y multiplicación a 32 bits
        ADDL ACC,P<<PM          ; Acumulación del último producto
                                ; realizado por la MAC
        LSL ACC,#8              ; Ajuste de formato a Qi=24
        MOVL XAR1,#total        ; Direccionamiento indirecto a la
                                ; variable total
        MOVL *XAR1,ACC          ; Escribe el resultado en total

FIN_R   NOP
        LB FIN_R                ; Ciclo infinito
        .end
```

El resultado obtenido en la variable *total* es igual a 268435456, lo que equivale al número 16 en formato $Q_i = 24$. En esta implementación, las variables de 32 bits que se declaran como globales, se especifican como datos de tipo **long**, además la operación de multiplicación y acumulación se realiza con la instrucción **QMACL** para poder operar correctamente los datos de entrada y al resultado obtenido, se le hace un ajuste de formato de punto entero, recorriendo ocho posiciones para conseguir que *total* este expresado en el formato $Q_i = 24$.

4.1.3. Producto punto entre vectores en punto flotante

La tercera implementación del producto punto entre dos vectores, contempla que los vectores puedan tener elementos con cifras decimales en formato de punto flotante de precisión simple bajo el estándar IEEE-754, esto se realiza con el objetivo de mostrar el manejo de este tipo de aritmética, utilizando la Unidad de Punto Flotante (FPU) del TMS320F28377S.

```
*
*          Producto Punto entre Vectores
* en formato de punto flotante de precision simple
*
        .global _c_int00

* Vectores de entrada a la operación
x       .float 1.1,2.2,3.3,4.4,5.5
h       .float 1.1,2.1,1.1,2.1,1.1

* Variables para guardar resultado
totf    .float 0.0              ; Resultado en punto flotante
totin   .word 0                 ; Resultado en punto fijo

N       .set 5                  ; Cantidad de elementos por vector
WDCR    .set 07029h             ; Dirección del registro de control
                                ; del WatchDog
        .text

; Inicio del programa
_c_int00
        EALLOW                  ; Habilitación de escritura a
                                ; registros protegidos
        CLRC XF                 ; Escribe cero en el bit XF
        MOVL XAR1,#WDCR         ; Direccionamiento al registro WDCR
        MOV *XAR1,#0068h        ; Deshabilita el watchDog
        EDIS                    ; Deshabilita la escritura en
                                ; registros protegidos

        MOVW DP,#x              ; Direccionamiento a la página de
                                ; memoria donde está declarada x
        MOVL XAR1,#x            ; Direccionamiento indirecto a x
        MOVL XAR2,#h            ; Direccionamiento indirecto a h

        ZAPA                    ; ACC=0 y P=0
        MOVF32 R1H,#0.0         ; R1H=0

        RPTB FIN_B,#N-1         ; N iteraciones
             MOV32 R2H,*XAR1++  ; R2H => dato apuntado por XAR1
             MOV32 R3H,*XAR2++  ; R3H => dato apuntado por XAR2
```

```
            MPYF32 R4H,R2H,R3H   ; R4H = R2H * R3H
            NOP                  ; Instrucción de operación nula
            ADDF32 R1H,R4H,R1H   ; R1H = R1H + R4H
FIN_B

        NOP                      ; Instrucción de operación nula
        MOV32 @totf,R1H          ; Escribe el resultado en
                                 ; formato flotante en totf
        F32TOI16 R5H,R1H         ; Conversión de formato
                                 ; flotante a entero
        NOP
        MOV32 @totin,R5H         ; Escribe resultado en totin

FIN_R   NOP
        LB FIN_R                 ; Ciclo infinito
        .end
```

En el programa anterior, podemos observar instrucciones de movimiento de datos, multiplicaciones y sumas en formato de punto flotante, las cuales corresponden a la FPU. El resultado obtenido por el código es 24.75, dato que se redondeó por truncamiento al utilizar la instrucción de conversión de formato flotante a punto fijo.

A diferencia de los programas del producto punto que operan datos en formato de punto entero, en el programa anterior se desarrolló la operación **MAC** en las instrucciones de suma y multiplicación, sin embargo, también podría utilizarse la instrucción **MACF32** para el cálculo del resultado, sustituyendo todo el bloque que comprende desde la instrucción **RPTB** hasta **FIN_R**. En las aplicaciones posteriores se continuará mostrando el uso de algunas instrucciones de la FPU.

4.2. Convolución

La convolución de $x(n)$ con $h(i)$ se calcula como el producto de la entrada actual $x(n)$ y las muestras retardadas $x(n-i)$ por los coeficientes de $h(i)$ de un sistema lineal e invariante en el tiempo discreto (SLITD). Esto se puede expresar como la sumatoria de una ecuación en diferencias descrita como

$$\begin{aligned} y(n) = \sum_{i=0}^{N-1} h(i)x(n-i) = h_0x(n) + h_1x(n-1) + \cdots \\ \cdots + h_{N-1}x(n-N+1) \end{aligned} \tag{4.2}$$

donde N es la cantidad total de términos de la convolución, la cual es igual a la suma de las longitudes de las secuencias que se operan. Para fines prácticos, N comúnmente se considera igual a la longitud máxima de las dos secuencias involucradas en la operación. El desarrollo

de la suma ponderada, es decir, la convolución, se observa en la Figura 4.1.

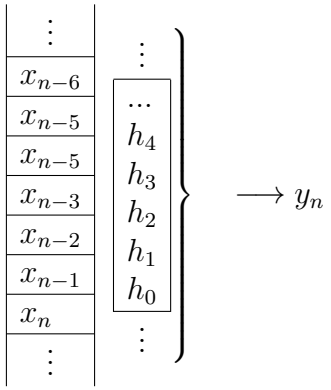


Figura 4.1: Convolución de una señal $x(n)$ con $h(n)$

La respuesta de la convolución $y\,(n)$, es una suma ponderada de los coeficientes $h(i)$ por la entrada actual $x(n)$ y las muestras retardadas, esto es equivalente a la respuesta de un sistema FIR, el cual se verá más adelante en este capítulo. La sumatoria que permite efectuar un filtro FIR es la operación de convolución de los coeficientes por una ventana temporal de una señal. Con los DSPs, la operación se puede calcular de una forma eficiente como se podrá observar en las implementaciones de la operación convolución.

Implementación de la convolución

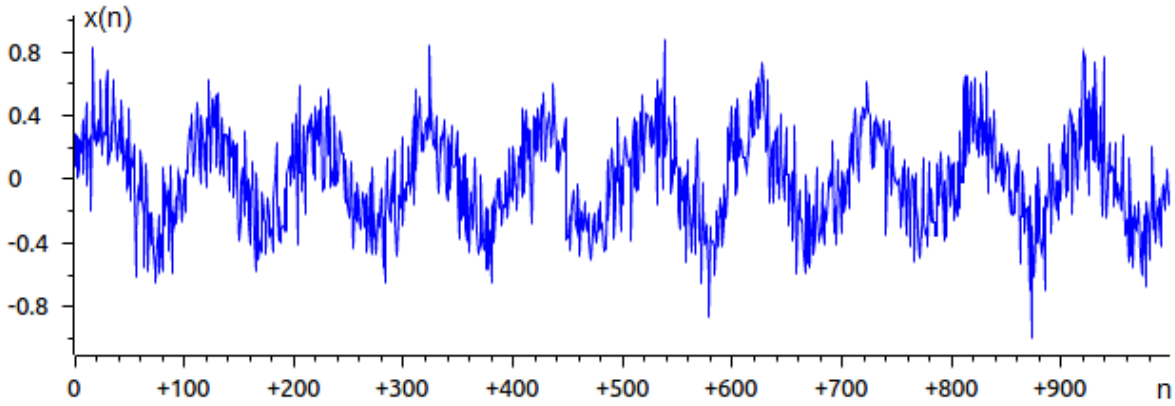
A continuación se presenta la implementación de la operación convolución utilizando dos secuencias de datos; $x\,(n)$ formada por 1000 puntos de una función sinusoidal con ruido blanco aditivo de tipo Gaussiano, considerando un SNR de 4 *dB*, y $h\,(n)$ corresponde a 51 puntos de una función triangular centrada en el origen, ambas secuencias son de amplitud máxima unitaria y fueron generadas utilizando como base las ecuaciones (4.3) y (4.4) en conjunto con un software libre (Octave). La Ecuación (4.3) se define como

$$x'\,(n) = \sin\left(\frac{2\pi f_0 n}{fs}\right) \tag{4.3}$$

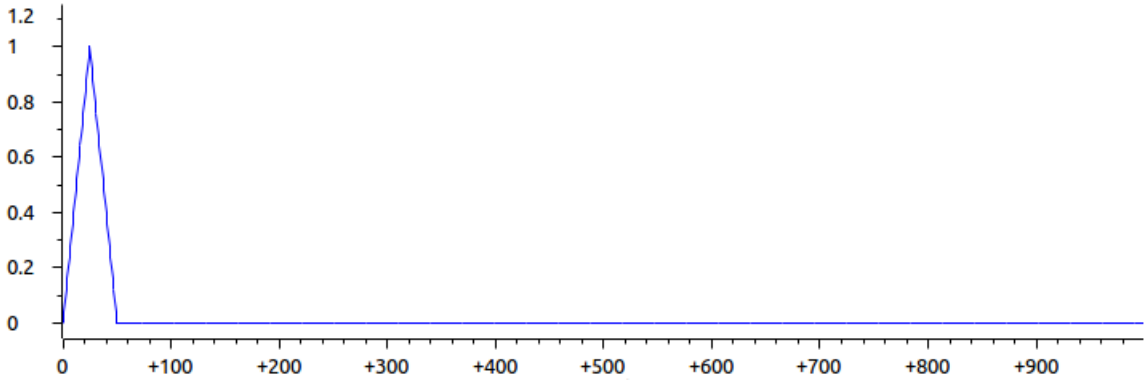
donde n es la variable que representa el tiempo discreto, el cual comprenderá de $n\epsilon\,[0, 1000]$, f_0 es la frecuencia de la secuencia en Hertz, $fs = 1000$ es la frecuencia de muestreo y $x'\,(n)$ es la señal generada sin ruido blanco Gaussiano. La secuencia $x\,(n)$ que se trabajará se observa en la Figura 4.2a y la señal $h\,(n)$ se define en 4.4

$$h\left(n\right) = \begin{cases} 1 - |n/(N-1)| & |n| \leq |(N-1)/2| \\ \\ 0 & |n| > |(N-1)/2| \end{cases} \tag{4.4}$$

donde $n\epsilon\left[-25, 25\right]$, generando los N puntos deseados de la secuencia, mencionados anteriormente logrando tener el punto máximo centrado en el eje horizontal, como se puede observar en la Figura 4.2b.



(a) $x\left(n\right)$ Señal sinusoidal con ruido blanco Gaussiano.



(b) $h\left(n\right)$ Señal triangular de 51 puntos.

Figura 4.2: Secuencias generadas para la implementación de la operación convolución.

Cada secuencia de datos se almacenó en un archivo de texto con terminación *.dat* en formato flotante y a partir de estos archivos, se realizaron las conversiones necesarias a formato de punto fijo, considerando longitudes de palabra de 16 y 32 bits, y diferentes cantidades de

bits para representar la parte fraccionaria del dato.

A continuación se presenta la implementación realizada de la operación convolución, empleando diferentes formatos numéricos. En los casos se implementa la operación convolución utilizando las secuencias de datos $x(n)$ y $h(n)$, mostradas en la Figura 4.2.

4.2.1. Convolución en formato de punto fijo a 16 bits

Para este programa, se convirtieron las secuencias de datos generadas $x(n)$ y $h(n)$ en formato de punto fijo, considerando 12 bits para representar las cifras decimales, tres bits para representar la parte entera del número y un bit para representar el signo, es decir Q_{12}. Las secuencias se guardaron en archivos individuales, incluyendo el encabezado necesario para importar los datos a la memoria. Los archivos para probar el funcionamiento del código son:

- xnL16Q12-conv.dat
- hnL16Q12-conv.dat

El programa de implementación de la operación convolución consiste en lo siguiente

```
*
*                   Convolución entre dos señales discretas
*                con datos en formato de punto fijo de 16 bits
*
        .global _c_int00

        .data
WDCR    .set 07029h             ; Dirección del registro de control WatchDog
N       .set 1050               ; Cantidad de puntos de la señal x(n)
Nw      .set 51                 ; Cantidad de muestras de hn
xb      .space 16*Nw            ; Reserva espacio para buffer x
xbf     .word 0                 ; Última localidad del buffer x

xn      .space 16*N             ; Espacio de memoria para x
hn      .space 16*Nw            ; Espacio de memoria para h
yn      .space 16*N             ; Espacio de memoria para yn

        .text

_c_int00
*Deshabilita el WatchDog
        EALLOW                  ; Habilita la escritura
        MOVL XAR1,#WDCR         ; Registro XAR1 apunta dir, WDCR
        MOV *XAR1, #0068h       ; Desactiva WatchDog
        EDIS                    ; Deshabilita escritura a registros
                                ; protegidos
```

```
*Configuracion e iniciaciones
        SETC SXM                ; Activación del uso de aritmética
                                ; con signo extendido
        MOVW DP,#xb             ; Direccionamiento de página a xb
        MOVL XAR1,#xn           ; Direccionamiento a xn
        MOVL XAR2,#yn           ; Direccionamiento a yn

        SPM 0                   ; Corrimientos nulos en registro P
        MOV AR4,#N-1            ; Escriba la cantidad de iteraciones
                                ; a realizar para el cálculo
CONVOL
        MOV AL,*XAR1++          ; Al = x(n)
        MOV @xb,AL              ; Coloca el dato x(n) al inicio del buffer xb
        MOVL XAR7,#xb           ; Direccionamiento indirecto a xb
        MOVL XAR3,#h            ; Direccionamiento indirecto a h
        ZAPA                    ; ACC=0 y P=0

        RPT #Nw-1               ; Declaración de ciclo RPT
        || MAC P,*XAR3++,*XAR7++

        ADDL ACC,P<<PM          ; Acumula el último producto
        LSL ACC,#2              ; Ajuste de Qi
                                ; Corrimiento para ajustar el Qi
        MOV *XAR2++,AH          ; Mueve la parte alta del acumulador
                                ; obteniendo un resultado de 16 bits
        MOVL XAR3,#xbf          ; Direccionamiento indirecto a
                                ; la localidad extra colocada al
                                ; final del buffer xb

; Ciclo para reacomodo de datos en el buffer xb el último dato lo
; desplaza hacia arriba del buffer

        RPT #Nw-1
        || DMOV *--XAR3         ; Mueve los datos del buffer,
                                ; recorriendo cada dato hacia la
                                ; localidad inicial del buffer xb

        BANZ CONVOL,AR4--       ; Repetición del bloque de
                                ; código etiquetado por la palabra
                                ; CONVOL, hasta que el registro AR4=0

iend    NOP
        LB iend                 ; Ciclo infinito
        .end
```

El flujo del programa se muestra en el diagrama de la Figura 4.3, el cuál corresponde al algoritmo utilizado para las tres implementaciones de la convolución.

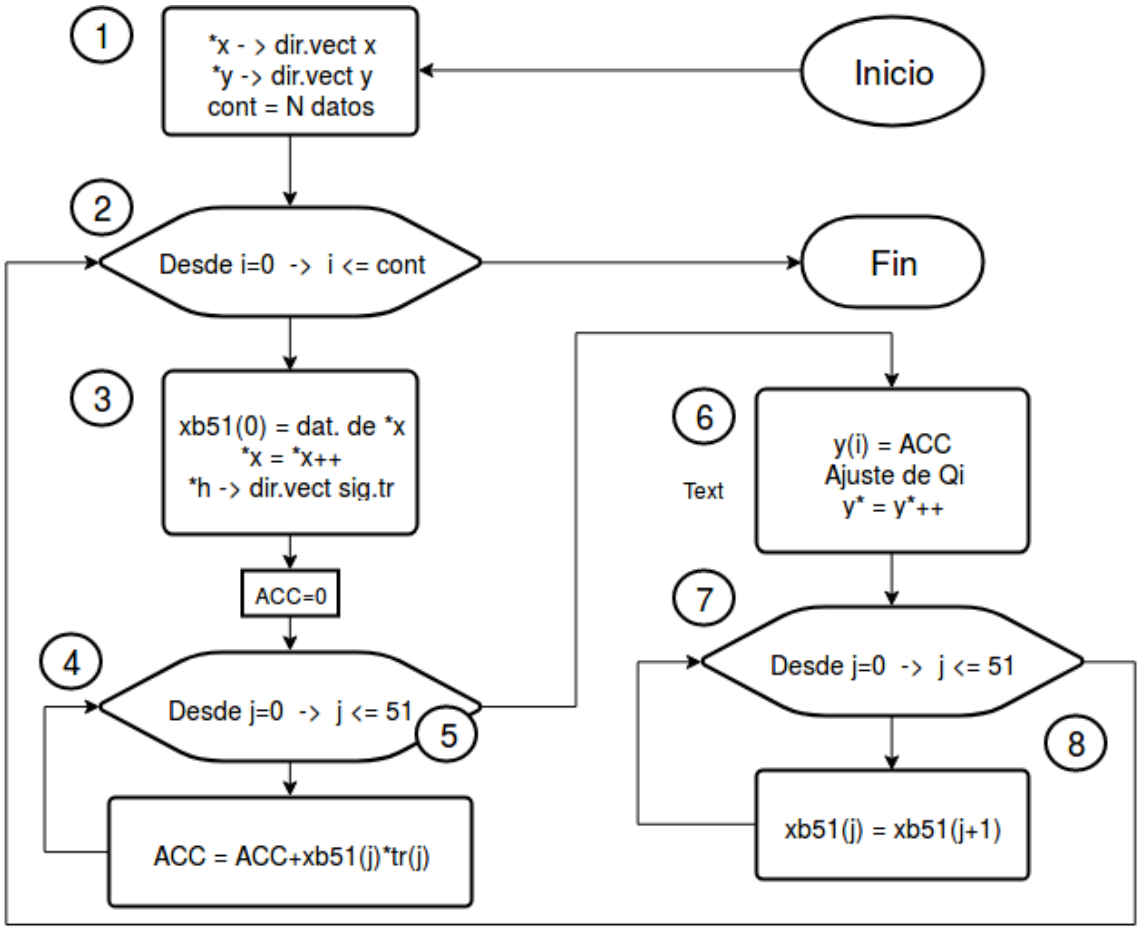


Figura 4.3: Diagrama de flujo del algoritmo planteado para la implementación de la operación convolución en lenguaje ensamblador.

Cada uno de los bloques del diagrama tiene un número que corresponde a la descripción de su implementación, como se muestra a continuación:

1. Se inician dos apuntadores, utilizando el modo de direccionamiento indirecto, señalando hacia la primera localidad de los arreglos x e y cuyas direcciones quedan en los registros auxiliares **XAR1** y **XAR2** respectivamente. También se carga en el registro **AR4** la cantidad de desplazamientos necesarios de la secuencia móvil para obtener la convolución, en este caso se tendrán 1050 iteraciones para realizar el cálculo completo.

2. Este bloque corresponde al ciclo que calculará cada uno de los términos de la convolución, siendo una especie de ciclo **for** en lenguaje C. El inicio del bloque de código que se repetirá en cada iteración está indicado por la etiqueta *CONVOL* y el final del bloque es la instrucción **BANZ** CONVOL, - -**AR4**.

3. La variable xb es un arreglo o vector con 51 localidades de memoria y tiene por objetivo ser la ventana que contendrá los datos de la secuencia $x(n)$ que se traslaparán en cada iteración del bloque 2, con la secuencia fija $h(n)$. Por ello, como primer paso se coloca el dato de la localidad apuntada por **XAR1** en la primer localidad de de xb y se post-incrementa la dirección del registro **XAR1**. Posteriormente se inician otros dos apuntadores utilizando el modo de direccionamiento indirecto, **XAR7** apunta a la primer localidad del buffer xb y **XAR3** apunta al primer valor de la secuencia $h(n)$.

4. Es el ciclo que calcula todas las operaciones **MAC** necesarias, entre el buffer xb y $h(n)$, por ende se realizan 51 iteraciones del bloque 5.

5. Cálculo de la operación **MAC** de los términos traslapados de la secuencia $x(n)$ con la secuencia $h(n)$.

6. La última operación **MAC** corresponde al valor de la convolución para la iteración correspondiente del bloque 2, por ello en este bloque 6 se realizan los corrimientos y ajustes necesarios para hacer un ajuste de formato de punto entero y se guarda el resultado en la localidad apuntada por el registro **XAR2**, post-incrementando la dirección de este para guardar el resultado de la siguiente iteración del bloque dos.

7. Cómo última parte del cálculo de un término de la convolución, este ciclo tiene por objetivo desplazar los datos del buffer xb una localidad de memoria de 16 bits, sacando el último dato del buffer y colocando el dato que se ingreso de la secuencia $x(n)$ en la segunda localidad de xb. La duración de este bloque es de 51 iteraciones.

8. Realiza el desplazamiento de datos mencionado en el punto anterior, utilizando la instrucción **DMOV**.

Las instrucciones en lenguaje ensamblador que se utilizaron para poder hacer la operación convolución principalmente fueron **MAC** y **DMOV**.

4.2.2. Convolución en formato de punto fijo a 32 bits

El formato de punto fijo que se eligió para convertir los datos de las secuencias de entrada $x(n)$ y $h(n)$ es el Q_{28}, obteniendo el resultado de cada término de la convolución en Q_{26}. Los archivos para probar el funcionamiento del código son:

- xnL32Q28-conv.dat
- hnL32Q28-conv.dat

El programa para calcular la operación convolución con datos de 32 bits en formato de punto entero es el siguiente.

```
*
*                  Convolución entre dos señales discretas
*               con datos en formato de punto fijo de 32 bits
*

        .global _c_int00

        .data
WDCR    .set 07029h
N       .set 1050                   ; Cantidad de puntos de la señal x(n)
Nw      .set 51                     ; Cantidad de puntos de la señal h(n)
xb      .space 32*Nw                ; Reserva espacio para buffer x
xbf     .long 0                     ; Última localidad del buffer x

x       .space 32*N                 ; Espacio de memoria para x(n) en Q28
h       .space 32*Nw                ; Espacio de memoria para h(n) en Q28
y       .space 32*N                 ; Espacio de memoria para y(n), resultado

        .text

_c_int00
*Deshabilita el WatchDog
        EALLOW                      ; Habilita la escritura
        MOVL XAR1,#WDCR             ; Registro XAR1 apunta dir, WDCR
        MOV *XAR1,#0068h            ; Desactiva WatchDog
        EDIS                        ; Deshabilita escritura a registros
                                    ; protegidos

*Configuracion e iniciaciones
        SETC SXM                    ; Modo extensión de signo
        SETC OVM                    ; Habilitacion de overflow
        SPM 0                       ; Corrimientos nulos en P

        MOVW DP,#xb                 ; Direccionamiento a la página de
                                    ; memoria del vector xb

        MOVL XAR1,#x                ; Direccionamiento a la secuencia x
        MOVL XAR2,#y                ; Direccionamiento a la secuencia y

        MOV AR4,#N-1                ; Registro auxiliar para el control
                                    ; de repeticiones N para realizar
                                    ; la convolución

CONVOL
        MOVL ACC,*XAR1++            ; Escribe el dato apuntado por XAR1
                                    ; en el ACC y post incrementa XAR1
        MOVL @xb,ACC                ; Coloca x(n) al inicio de xb
        MOVL XAR7,#xb               ; Direccionamiento al primer
                                    ; elemento del vector xb
```

```
        MOVL XAR3,#h                    ; Direccionamiento al primer elemento
                                        ; del vector h
        ZAPA

        RPT #Nw-1                       ; Ciclo de repetición de QMAC
        || QMACL P,*XAR3++,*XAR7++      ; Mult. y acumulación

        ADDL ACC,P                      ; Acumula el último producto
        ;LSL ACC,#4                     ; Ajuste de Qi
        MOVL *XAR2++,ACC                ; Escribe el resultado en y(AR4) y
                                        ; post incrementa la dirección que
                                        ; contiene XAR2

* Desplazamiento del buffer xb
* Se apunta a la localidad extra al final del buffer xb, y a la penúltima
* de xb, para copiar los datos de la parte final a la inicial en retroceso

        MOVL XAR3,#xbf
        MOVL XAR7,#xbf-2
        MOV AR5,#Nw-1

move_buffer
        MOVL ACC,*--XAR7
        MOVL *--XAR3,ACC
        BANZ move_buffer,AR5--

        BANZ CONVOL,AR4--               ; Ciclo de cálculo de cada término
                                        ; de la convolución

iend    NOP
        LB iend                         ; Ciclo infinito
        .end
```

Los cambios en comparación con el programa de la convolución que opera datos a 16 bits son los siguientes:

1. Las variables de tipo arreglo de enteros inicializadas para alojar los datos de entrada y de salida, cambian de tipo *int* a *long* o las localidades que se reservan para cargar los datos provenientes de los archivos *.dat* ahora se declaran de 32 bits.

2. La instrucción **MAC**, se cambia por **QMACL** para el manejo de la longitud de palabra de 32 bits.

3. Para recorrer los datos del buffer xb, se utilizó la repetición de un bloque de código (*move_buffer*), sustituyendo a la instrucción **DMOV**, porque maneja únicamente datos de 16 bits.

4.2.3. Convolución en formato de punto flotante IEEE 754

El tercer programa de ejemplo de la convolución, utiliza las secuencias de datos $x(n)$ y $h(n)$, en su formato de punto flotante de precisión simple. Los archivos para probar el funcionamiento del código son:

- xnL32Float-conv.dat
- hnL32Float-conv.dat

El programa de la implementación consiste en lo siguiente

```
*
*               Convolución entre dos señales discretas
*       con datos en formato de punto flotante de precisión simple
*

        .global _c_int00

        .data
WDCR    .set 07029h             ; Dirección del registro de control WatchDog
CTE_WD  .set 0068h              ; Constante para desactivar el WatchDog

N       .set 1050               ; Cantidad de puntos de la señal x(n)
Nw      .set 51                 ; Cantidad de puntos de la señal h(n)
xb      .space 32*Nw            ; Reserva espacio para buffer x
xbf     .float 0                ; Última localidad del buffer x

x       .space 32*N             ; Espacio de memoria para x(n)
h       .space 32*Nw            ; Espacio de memoria para h(n)
y       .space 32*N             ; Espacio de memoria para y(n) resultado

        .text

_c_int00
*Deshabilita el WatchDog
        EALLOW                  ; Habilita la escritura
        MOVL XAR1, #WDCR        ; Registro XAR1 apunta dir. WDCR
        MOV *XAR1, #0068h       ; Desactiva WatchDog
        EDIS                    ; Deshabilita escritura a registros
                                ; protegidos

* Configuraciones e inicializaciones
        SETC SXM
        MOVW DP,#xb

        MOVL XAR1,#x
```

```
        MOVL XAR2,#y
        MOV AR4,#N-1            ; Registro auxiliar para el control
                                ; de N repeticiones, para realizar
                                ; la convolución

CONVOL
        MOVL ACC,*XAR1++        ; ACC = x(n)
        MOVL @xb,ACC            ; Coloca x(n) en el buffer xb
        MOVL XAR7,#xb
        MOVL XAR3,#h

        ZERO R2H                ; Escribe cero en el registo R2H
        ZERO R3H                ; Escribe cero en el registo R3H
        ZERO R6H                ; Escribe cero en el registo R6H
        ZERO R7H                ; Escribe cero en el registo R7H

        RPT #Nw-1
        || MACF32 R7H,R3H,*XAR3++,*XAR7++

        ADDF32 R7H, R7H, R3H    ; Acumula el último producto
        NOP

        MOV32 *XAR2++,R7H       ; Mueve el acumulador al espacio de
                                ; la variable y

* Desplazamiento del buffer xb
* Se apunta a la localidad extra al final del buffer xb, y a la penúltima
* de xb, para copiar los datos de la parte final a la inicial en retroceso

        MOVL XAR3,#xbf
        MOVL XAR7,#xbf-2
        MOV AR5,#Nw-1

move_buffer
        MOVL ACC,*--XAR7
        MOVL *--XAR3,ACC
        BANZ move_buffer,AR5--

        BANZ CONVOL,AR4--

iend    NOP
        LB iend                 ; Ciclo infinito
        .end
```

Para el manejo de datos en formato de punto flotante, en el anterior programa se utilizó la Unidad de Punto Flotante (FPU), para poder realizar las operaciones MAC necesarias para obtener el resultado de la convolución. Cabe resaltar que la instrucción **ADDF32** necesita dos ciclos de máquina para tener el resultado, esta información puede ser consultada en [17].

4.2.4. Análisis de resultados de las implementaciones

En la Figura 4.4 se muestra la gráfica de la secuencia obtenida al calcular la operación convolución por un programa de cálculo matemático, manejando formato de punto flotante de doble precisión.

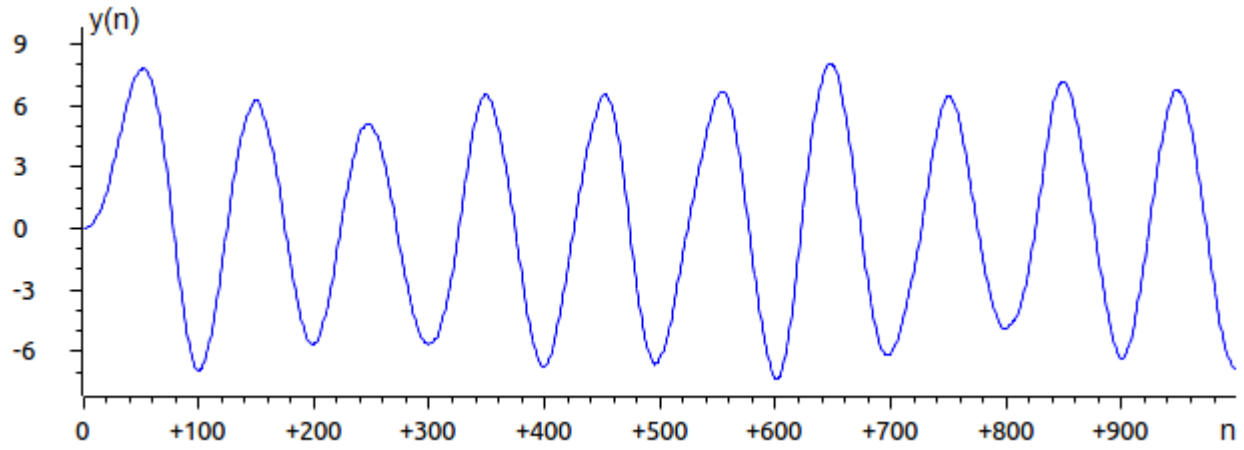


Figura 4.4: Gráfica de la secuencia resultante de aplicar la convolución con las secuencias $x\,(n)$ y $h\,(n)$ en formato de punto flotante de doble precisión.

Para analizar los resultados obtenidos por los tres programas implementados de la operación convolución, se calculó el error absoluto de cada secuencia obtenida, considerando como referencia la secuencia obtenida por el programa de cálculo Octave que utiliza un formato de punto flotante de doble precisión. En la Tabla 4.1 se observan los errores de precisión numérica obtenidos por los códigos propuestos.

Tabla 4.1: Errores absolutos obtenidos de la secuencias calculadas por los programas desarrollados de la convolución.

Formato de entrada	**Formato de salida**	**Error Absoluto**		
		Min.	**Prom.**	**Max.**
Punto flotante simple	Punto flotante simple	2.1000e-09	0.0094	0.004
Punto entero Q28 L=32 bits	Punto entero Q26	1.9238e-08	1.2488e-06	2.6558e-06
Punto entero Q12 L=16 bits	Punto entero Q10	6.4353e-05	0.0025	0.0049

El mínimo error absoluto se obtuvo al manejar datos y aritmética de punto flotante, sin embargo, analizando los errores absolutos promedio y máximo, se puede concluir que el mejor rendimiento en error de precisión numérica se obtiene al manejar formatos de punto entero a 32 bits, sin embargo, el mejor desempeño de las tres implementaciones estará en función de su aplicación.

Las cifras de la Tabla 4.1 muestran que las implementaciones realizadas son correctas al registrar errores significativos con respecto a la secuencia calculada utilizando formato de punto flotante de precisión simple. Por último, de forma significativa la Tabla 4.2 muestra algunos datos de la secuencia obtenida al hacer la convolución entre las secuencias $x\,(n)$ y $h\,(n)$, en el programa de cálculo de precisión doble y por los programas desarrollados.

Tabla 4.2: Comparación de algunos datos de la secuencia obtenida $y\,(n)$ por la operación convolución.

Punto flotante de precisión doble	**Punto flotante de precisión simple**	**Punto entero Q26 de L=32 bits**	**Punto entero Q10 de L=16bits**
0.0031187857	0.0031187997	0.0031187534	0.0029296875
0.0172677364	0.0172677583	0.0172677040	0.0166015625
0.0423501385	0.0423501581	0.0423499941	0.0419921875
0.0677119000	0.0677119121	0.0677117705	0.0673828125
0.0959098275	0.0959098265	0.0959095954	0.0957031252
0.1344896896	0.1344896550	0.1344894767	0.1337890625

4.3. Correlación

La correlación es una operación matemática similar a la convolución, se utiliza para determinar la similitud entre dos señales, calcular retardos entre señales, verificar y calcular la periodicidad de un señal, calcular energías, etc. La correlación entre señales se encuentra en aplicaciones como radar, sonar, síntesis de voz, reconocimiento de voz, procesamiento de imágenes, geología y muchas áreas de la ingeniería [15].

La correlación entre dos señales $x(n)$ e $y(n)$ se define como $r_{xy}(n)$

$$r_{xy}(l) = \sum_{n=\infty}^{\infty} x(n)y(n-l) \qquad ; \qquad l = 0, \pm 1, \pm 2, \cdots \tag{4.5}$$

si $x(n)$=$y(n)$, se tiene la auto correlación $r_{xx}(l)$ definida

$$r_{xx}(l) = \sum_{n=\infty}^{\infty} x(n)x(n-l) \qquad ; \qquad l = 0, \pm 1, \pm 2, \cdots \tag{4.6}$$

donde l es el desplazamiento entre señales cuando se calcula la sumatoria. La primera señal del subíndice se considera fija y la segunda móvil o desplazada en l.

Los programas presentados para la implementación de la operación correlación discreta entre dos señales, también puede utilizarse para realizar la auto correlación, siempre y cuando se utilicen los formatos de punto entero adecuados para representar el término $r_{xx}\,(0)$ o, se utilice la implementación FPU.

Implementación de la operación correlación

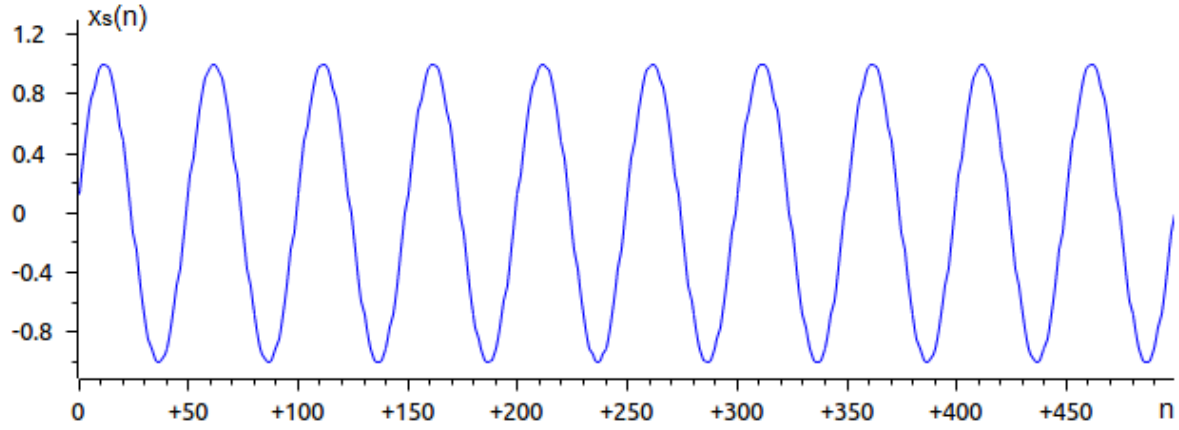
Se presentarán en esta sub-sección tres programas en lenguaje ensamblador que implementan la operación correlación utilizando dos secuencias discretas, las cuáles fueron generadas considerando las ecuaciones (4.7) y (4.8) respectivamente.

$$x_s\,(n) = \operatorname{sin}\left(\frac{2\pi f_0 n}{fs}\right) \tag{4.7}$$

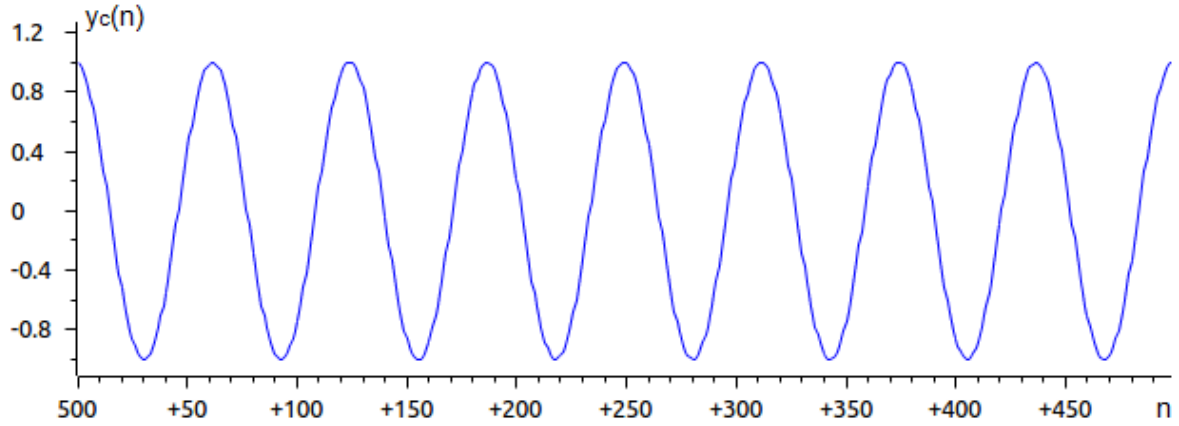
$$y_c\,(n) = \cos\left(\frac{2\pi f_1 n}{fs}\right) \tag{4.8}$$

En ambas ecuaciones, n es la variable que representa el tiempo discreto, la cual comprende el intervalo de $[0, 499]$, f_0 y f_1 son las frecuencias de interés en Hertz de las señales $x_s\,(n)$ y $y_c\,(n)$, fs es la la frecuencia de muestreo.

Para las implementaciones que se describirán en esta Sección, se consideró $f_0 = 10$ y $f_1 = 8$ Hz y $fs = 500$ Hz, además la amplitud de ambas señales se consideró como unitaria, para poder trabajar mejor los formatos de punto entero. En la Figura 4.5 se pueden observar las gráficas de las dos secuencias de datos en formato flotante.



(a) Secuencia discreta coseno $y_c\,(n)$ de 500 muestras.



(b) Secuencia discreta sinusoidal $x_s\,(n)$ de 500 muestras.

Figura 4.5: Secuencias generadas para la implementación de la operación correlación.

4.3.1. Correlación entre dos señal discretas a 16 bits en punto fijo

El primer programa, trabaja con las secuencias discretas de entrada $y_c\,(n)$ y $x_s\,(n)$ en formato punto fijo, considerando 12 bits para representar la parte fraccionaria, tres bits para la parte entera y un bit para el signo del número. Las secuencias de datos para probar el funcionamiento de la aplicación son los siguientes;

- xnL16Q12-corr.dat
- ynL16Q12-corr.dat

A continuación se presenta el código en lenguaje ensamblador de la operación correlación completa.

```
*
*              Correlación entre dos señales discretas
*
*           Usando datos con formato de punto fijo de 16 bits
*
        .global _c_int00

N       .set 500              ; Total de puntos de cada señal y(n) y x(n)
cont    .word 0               ; Contador de la cantidad de productos
                              ; a realizar por cada iteración de la
                              ; variable l
xn      .space N*16           ; Vector reservado para la secuencia  x(n)
basx    .word 0               ; Localidad de memoria para separar vectores

yn      .space N*16           ; Vector reservado para la secuencia y(n)
basy    .word 0               ; Localidad de memoria para separar vectores

rxy     .space 2*N*16         ; Vector reservado para guardar el resultado
incDir  .long 0               ; Variable para incrementar direcciones
                              ; del apuntador
        .text

_c_int00
        SETC SXM              ; Modo extensión de signo
        SETC OVM
        SPM 0                 ; Corrimientos nulos en P

        MOVW DP,#cont         ; Direccionamiento a la página  de memoria
                              ; que contiene la variable cont
        MOV AR4,#N-1          ; Carga la cantidad de iteraciones
                              ; para l<=0
        MOVL XAR5,#rxy+N      ; Direccionamiento al elemento 500
                              ; del vector rxy
        MOVL XAR2,#yn         ; Direccionamiento al vector de
                              ; datos y, secuencia móvil
        MOV AL,#N-1           ; Coloca el número 499 en el AL
        MOVW @cont,AL         ; Escribe el contenido de AL en la
                              ; variable cont
        MOVL XAR3,#incDir     ; Direccionamiento a incDir

*****************************************************************
* Primera parte de la correlacion cruzada para l <= 0
CORRELN
        MOVL XAR7,#xn         ; Direccionamiento a la secuencia x(n)
        ZAPA
```

```
        RPT @cont
        || MAC P,*XAR2++,*XAR7++          ; Cálculo de la MAC a 16 bits

        ADDL ACC,P<<PM          ; Acumulación del último producto
        ;LSL ACC,#4             ; Ajuste de formato Qi
        MOV *--XAR5,AH          ; Pre-decrementa el apuntador rxy y
                                ; guarda el resultado calculado de rxy(l)

        DEC @cont               ; Decrementa la cantidad de productos a realizar
                                ; en la siguiente iteración

* Desplazamiento de la secuencia móvil en la
* operación correlación para l<=0
        MOVL ACC,*XAR3          ; Se carga la variable incDir en ACC
        ADD ACC,#1              ; Suma 1 al valor de ACC, para
                                ; desplazar la secuencia
        MOVL *XAR3,ACC          ; Coloca el valor del ACC en incDir

        MOVL XAR2,#yn           ; Se apunta a la dirección de y
        MOVL ACC,XAR2           ; Se coloca la dirección anterior
                                ; en el ACC
        ADDL ACC,*XAR3          ; Se suma a la dirección el valor
                                ; de incDir
        MOVL XAR2,ACC           ; Se regresa el resultado
                                ; al apuntador de la secuencia
                                ; móvil y(n) regresando el valor
                                ; a la var. delDir

        BANZ CORRELN,AR4--      ; Ciclo de cálculo de los N elementos de la
                                ; correlación para l<=0

*************************************************************************
* Segunda parte de la correlación cruzada, para l > 0
        MOVW DP,#cont           ; Direccionamiento a la página de memoria
                                ; que contiene la variable cont
        MOV AR4,#N-1            ; Cantidad de iteraciones a realizar para l>0
        MOVL XAR2,#xn           ; Direccionamiento al vector x(n)
        MOV AL,#N-1             ; Coloca el número 499 en el AL
        MOVW @cont,AL           ; Escribe la cantidad de productos
                                ; a realizar en cont
        MOVL XAR5,#rxy+N-1      ; Se apunta con XAR5 al elemento 499 de rxy(l)

        ZAPA
        MOVL *XAR3,ACC          ; incDir=0

CORRELP
        MOVL XAR7,#yn           ; Se apunta a la dirección a y
        ZAPA
```

```
        RPT @cont           ; Ciclo para realizar la MAC
        || MAC P,*XAR2++,*XAR7++

        ADDL ACC,P<<PM      ; Acumulación del último productoC
        ;LSL ACC,#4         ; Ajuste de formato Qi
        MOVW *XAR5++,AH     ; Guarda el resultado total de la MAC en rxy(499)
                            ; y postincrementa
        DEC @cont           ; Decrementa la cantidad de productos a realizar
                            ; en la siguiente iteración

* Desplazamiento de la secuencia móvil en
* la operación correlación para l>0
        MOVL ACC,*XAR3      ; ACC=incDir
        ADD ACC,#1          ; Suma 1 al ACC
        MOVL *XAR3,ACC      ; incDir=ACC

        MOVL XAR2,#xn       ; XAR2 apunta a xn
        MOVL ACC,XAR2       ; Coloca la dirección en el ACC
        ADDL ACC,*XAR3      ; Suma la variabla incDir al ACC
        MOVL XAR2,ACC       ; Regresa la dirección obtenida del
                            ; ACC en el apuntador XAR2
        BANZ CORRELP,AR4—

iend    NOP
        LB iend             ; Ciclo infinito
        .end
```

El algoritmo que se utilizó en el programa anterior se puede observar en el diagrama de bloques de la Figura 4.6. Dicho planteamiento también se utilizó para operar con datos de 32 bits en punto fijo y en formato flotante de precisión simple.

A continuación se describe cada uno de los bloques que forman el diagrama, con el objetivo de que el lector observe la relación que hay entre el programa y el algoritmo, el cual básicamente está dividido en dos partes iguales, la primera comprende los bloques 1 al 7 y calcula todos los términos para $l < 0$ y los bloques del 8 al 14 para calcular los términos de $l \geq 0$.

1. Se apunta a la primera localidad del vector y la cuál será la secuencia de datos móvil, y a la localidad 500 del arreglo rxy que contendrá el resultado de la correlación. Estas operaciones se realizan utilizando el modo de direccionamiento indirecto utilizando los registros auxiliares **XAR2** y **XAR5** respectivamente. También se carga a la variable *cont* el número de iteraciones (500) que realizarán para el cálculo de cada término.

2. Inicia el bloque de código para calcular todos los términos de la correlación para $l < 0$. Este ciclo es controlado por el contador que se carga en el registro **AR4** y por la

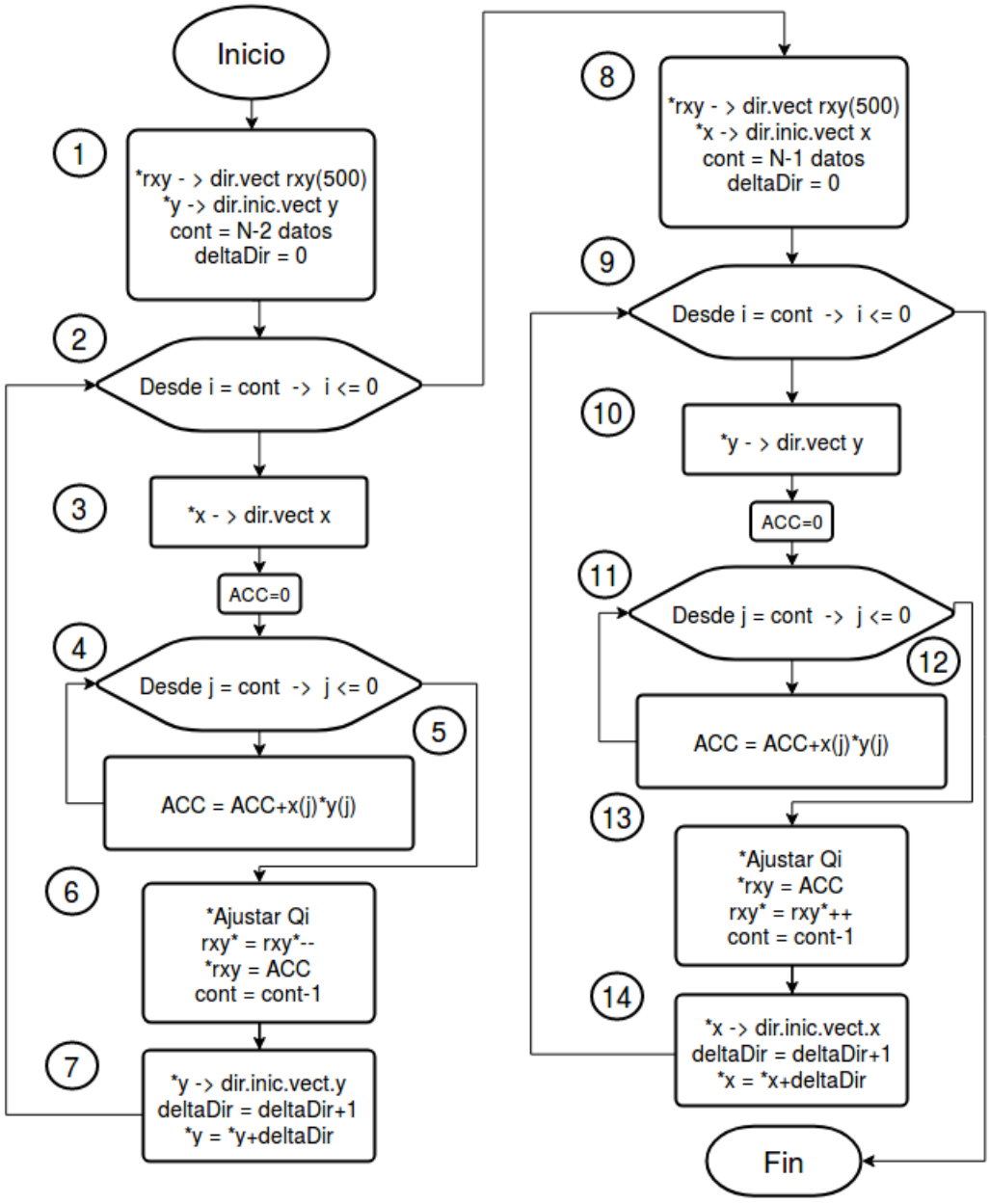


Figura 4.6: Diagrama de flujo del algoritmo utilizado para la implementación de la operación correlación.

instrucción **BANZ**, al finalizar cada iteración de este bloque, se obtiene el resultado de un término de la correlación, la cual tendrá una longitud de 1000 puntos, debido a la suma de las longitudes de las secuencias de entrada.

3. Utilizando el registro auxiliar **XAR7**, se apunta a la primer localidad de la secuencia

fija x, para comenzar con el ciclo de cálculo de operaciones mac.

4. La variable *cont*, se decrementará en cada iteración del ciclo indicado por el bloque 2, esto es con el fin de simular el movimiento de la secuencia y y operar únicamente los términos empalmados de ambas secuencias de datos. Este ciclo de cálculo se lleva a cabo con la instrucción **RPT**.

5. Se realiza la operación mac utilizando los registros auxiliares **XAR2** y **XAR7** que corresponden a las secuencias y y x respectivamente.

6. Calculado el término l de la correlación, se realizan los corrimientos hacia la izquierda necesarios para obtener el formato de punto fijo deseado, en este caso, nos interesa que el resultado se tenga con 12 bits para la parte fraccionaria. Después se guarda el valor en la localidad apuntada por el registro **XAR5** y se post-decrementa dicha dirección, finalizando el bloque con el decremento de la variable *cont*.

7. Para realizar correctamente la siguiente iteración del bloque 2, se vuelve a cargar el registro **XAR2** con la dirección de la localidad inicial del vector y y se recorre tantas localidades de memoria equivalentes al número de iteraciones del bloque 2, este desplazamiento de direcciones da el movimiento a la secuencia móvil. Por último se suma en cada iteración una unidad a la variable *dirCont* (iniciada en 0) para llevar a cabo los desplazamientos necesarios de la dirección del registro **XAR2**.
   En este bloque se finaliza el cálculo de los términos de la correlación para $l < 0$ una vez que todo el ciclo del bloque 2 se haya terminado de hacer.

8. Para calcular los términos en $l \geq 0$, ahora colocamos la dirección del primer elemento del vector x en el registro auxiliar **XAR2**, la variable *dirCont* se le asigna el valor cero y se carga a la variable *cont* el número de iteraciones que se van a realizar (500).

9. Este bloque corresponde al ciclo de cálculo de cada término de la correlación para $l \geq 0$. La configuración es la misma que el bloque 2.

10. Iniciado el ciclo de cálculo, se apunta hacia la primer localidad del vector y, secuencia que ahora estará estática en la realización de operaciones mac.

11. Similar al bloque 4, se inicia el ciclo de cálculo de todas las operaciones mac necesarias para calcular un término de la correlación después de n iteraciones, las cuáles se harán acorde al valor de la variable *cont*, la cual se decrementará cada vez que se realice una iteración del bloque 9, y el ciclo se lleva a cabo por la instrucción **RPT**.

12. Realización de las operaciones mac de los datos traslapados de ambas secuencias de datos.

13. Al finalizar el ciclo del bloque 11, se tiene un término de la correlación al cuál se le aplica un ajuste de punto fijo, realizando desplazamientos hacia la izquierda, de tal forma que obtengamos 12 bits para la parte fraccionaria. Después se guarda el resultado en la localidad apuntada por el registro **XAR5** y se post-incrementa dicha dirección. Por último la variable *cont* se decrementa una unidad para el control del cálculo de operaciones mac.

14. Para realizar el cálculo del siguiente término de la correlación, se vuelve a cargar el registro XAR2 con la dirección de la localidad inicial del vector x y se recorre tantas localidades de memoria equivalentes al número de iteraciones del bloque 9, este desplazamiento de direcciones da el movimiento a la secuencia móvil. Por último se suma en cada iteración una unidad a la variable *dirCont* (iniciada en 0) para llevar a cabo los desplazamientos necesarios de la dirección del registro **XAR2**.
    En este bloque se finaliza el cálculo de los términos de la correlación para $l \geq 0$ una vez que todo el ciclo del bloque 9 se haya terminado de hacer.

4.3.2. Correlación entre dos señales discretas a 32 bits en punto fijo

El segundo programa de ejemplo de correlación que se presenta a continuación, utiliza las secuencias de datos $y_c(n)$ y $x_s(n)$ convertidas a formato de punto entero considerando 28 bits para la parte fraccionaria, tres bits para la parte entera y un bit para el signo del número. Los archivos para probar el funcionamiento del código son

- xnL32Q28-corr.dat
- ynL32Q28-corr.dat

```
*
*              Correlación entre dos señales discretas
*
*          Usando datos con formato de punto fijo de 32 bits
*
        .global _c_int00

        .data
N       .set 500              ; Total de puntos de cada señal y(n) y x(n)
cont    .long 0               ; Contador de la cantidad de productos
                              ; a realizar por cada iteración de la
                              ; variable l
xn      .space N*32           ; Vector reservado para la secuencia  x(n)
basx    .long 0               ; Localidad de memoria para separar vectores

yn      .space N*32           ; Vector reservado para la secuencia y(n)
```

```
basy    .long 0              ; Localidad de memoria para separar vectores

rxy     .space 2*N*32        ; Vector reservado para guardar el resultado
incDir  .long 0              ; Variable para incrementar direcciones
                             ; de apuntador
        .text

_c_int00
        SETC SXM             ; Modo extensión de signo
        SETC OVM
        SPM 0                ; Corrimientos nulos en P

        MOVW DP,#cont        ; Direccionamiento a la página de
                             ; memoria que contiene la variable cont
        MOV AR4,#N-1         ; Carga la cantidad de iteraciones
                             ; para l<=0

        MOVL XAR5,#rxy+2*N   ; Direccionamiento al elemento 500
                             ; del vector rxy
        MOVL XAR2,#yn        ; Direccionamiento al vector de
                             ; datos y, secuencia móvil
        MOV AL,#N-1          ; Coloca el número 499 en el AL
        MOVW @cont,AL        ; cont=AL
        MOVL XAR3,#incDir    ; Direccionamiento a incDir

*****************************************************************************
* Primera parte de la correlacion cruzada para l <= 0
CORRELN
        MOVL XAR7,#xn        ; Direccionamiento al vector de
                             ; datos x, secuencia fija
        ZAPA

        RPT @cont            ; Ciclo para realizar la MAC
        || QMACL P,*XAR2++,*XAR7++          ; Cálculo de la MAC a 32 bits

        ADDL ACC,P<<PM       ; Acumulación del último producto
                             ; realizado por MAC
        ;LSL ACC,#4          ; Ajuste de formato Qi
        MOVL *--XAR5,ACC     ; Pre decrementa el apuntador rxy
                             ; y guarda el resultado de rxy(l)
        DEC @cont            ; Decrementa la cantidad de
                             ; productos a realizar en la
                             ; siguiente iteración

* Desplazamiento de la secuencia móvil en la
* operación correlación para l<=0
        MOVL ACC,*XAR3       ; Se carga la variable incDir en ACC
        ADD ACC,#2           ; Suma 2 al valor de ACC, para
                             ; desplazar la secuencia
```

```
        MOVL *XAR3,ACC      ; incDir=ACC

        MOVL XAR2,#yn       ; Se apunta a la dirección de y
        MOVL ACC,XAR2       ; Se coloca la dirección anterior en
                            ; el ACC
        ADDL ACC,*XAR3      ; Se suma a la dirección el valor
                            ; de incDir
        MOVL XAR2,ACC       ; Se regresa el resultado al
                            ; apuntador de la secuencia móvil
                            ; y(n) regresando el valor a delDir

        BANZ CORRELN,AR4--  ; Ciclo de cálculo de los N
                            ; elementos de la correlación
                            ; para l<=0

*************************************************************************
* Segunda parte de la correlación cruzada, para l > 0
        MOVW DP,#cont       ; Direccionamiento a la página
                            ; de memoria que contiene la
                            ; variable cont
        MOV AR4,#N-1        ; Escribe la cantidad de
                            ; iteraciones a realizar para l>0
        MOVL XAR2,#xn       ; Direccionamiento al vector
                            ; de datos x, secuencia móvil
        MOV AL,#N-1         ; Coloca el número 499 en el AL
        MOVW @cont,AL       ; Escribe la cantidad de
                            ; productos a realizar en cont

        MOVL XAR5,#rxy+2*N-1      ; Se apunta con XAR5 al
                            ; elemento 499 del vector rxy
        ZAPA
        MOVL *XAR3,ACC      ; incDir=0

CORRELP
        MOVL XAR7,#yn       ; Se apunta a la dirección de y
        ZAPA

        RPT @cont           ; Ciclo para realizar la MAC
        || QMACL P,*XAR2++,*XAR7++          ; Cálculo de la MAC a 32 bits

        ADDL  ACC,P<<PM     ; Acumulación del último producto
        ;LSL ACC,#4         ; Ajuste de formato Qi

        MOVL *XAR5++,ACC    ; Guarda el resultado total de la
                            ; MAC en rxy(l) y postincrementa
                            ; el apuntador de rxy
        DEC @cont           ; Decrementa la cantidad de
                            ; productos a realizar en la
                            ; siguiente iteración
```

```
* Desplazamiento de la secuencia móvil en
*  la operación correlación para l>0
        MOVL ACC,*XAR3      ; ACC=incDir
        ADD ACC,#2          ; Suma 2 al valor de ACC, para
                            ; desplazar la secuencia
        MOVL *XAR3,ACC      ; incDir=ACC
        MOVL XAR2,#xn       ; Se apunta a la dirección del
                            ; primer elemento del vector x
        MOVL ACC,XAR2       ; Coloca la dirección en el ACC
        ADDL ACC,*XAR3      ; Suma la variable incDir al ACC
        MOVL XAR2,ACC       ; Regresa la dirección obtenida
                            ; del ACC en el apuntador XAR2
        BANZ CORRELP,AR4--

iend    NOP
        LB iend             ; Ciclo infinito
        .end
```

4.3.3. Correlación entre dos señales discretas en punto flotante IEEE 754

Y por último, se presenta el programa utilizando aritmética y datos en formato de punto flotante de precisión simple. Los archivos para probar el funcionamiento del código son:

- xnFloat-corr.dat
- ynFloat-corr.dat

```
*
*            Correlación entre dos señales discretas
*
*         Usando datos con formato de punto flotante de 32 bits
*
*
        .global _c_int00

        .data
N       .set 500            ; Total de puntos de cada señal y(n) y x(n)
cont    .long 0             ; Contador de la cantidad de productos
                            ; a realizar por cada iteración de la
                            ; variable l
xn      .space N*32         ; Vector reservado para la secuencia  x(n)
basx    .long 0             ; Localidad de memoria para separar vectores

yn      .space N*32         ; Vector reservado para la secuencia y(n)
basy    .long 0             ; Localidad de memoria para separar vectores
```

```
rxy     .space 2*N*32       ; Vector reservado para guardar el resultado
incDir  .long 0             ; Variable para incrementar direcciones
                            ; de apuntador
        .text

* Inicio del programa
_c_int00
        EALLOW              ; Habilita la escritura de registros protegidos

        MOVL XAR1,#WDCR     ; Registro XAR1 apunta dir WDCR
        MOV *XAR1,#0068h    ; Desactiva WatchDog
        EDIS                ; Deshabilita escritura a registros
                            ; protegidos

        MOVW DP,#cont       ; Direccionamiento a la página de
                            ; memoria que contiene la
                            ; variable cont
        MOV AR4,#N-1        ; Carga la cantidad de iteraciones
                            ; para l<=0
        MOVL XAR5,#rxy+2*N  ; Direccionamiento al elemento 500
                            ; del vector rxy
        MOVL XAR2,#yn       ; Direccionamiento al vector de
                            ; datos y, secuencia móvil
        MOV AL,#N+1         ; Coloca el número 501 en el AL
        MOVW @cont,AL       ; Escribe el contenido de AL en cont
        MOVL XAR3,#incDir   ; Direccionamiento a incDir

;***************************************************************************
* Primera parte de la correlación cruzada para l <= 0
CORRELN
        MOVL XAR7,#xn       ; Direccionamiento al vector de
                            ; datos x secuencia fija
        ZAPA
        ZERO R2H            ; La operación MACF32
        ZERO R3H            ; auxiliares R2H, R3H, R6H y R7H
                            ; de la FPU,
        ZERO R6H            ; por ello antes de realizar
                            ; la operación
        ZERO R7H            ; se debe de escribir cero en
                            ; dichos registros

        RPT @cont           ; Ciclo para realizar la MAC
        || MACF32 R7H,R3H,*XAR2++,*XAR7++
                            ; Cálculo de la MAC a 32 bits
                            ; y(n) y x(n) apuntados por
                            ; los reg. aux.
                            ; XAR2 y XAR7 respectivamente
```

```
        ADDF32 R7H, R7H, R3H ; Acumulación del último producto
        NOP
        MOV32 *--XAR5,R7H    ; Pre decrementa el apuntador rxy y
                             ; guarda el resultado calculado
                             ; de rxy(l) en la direccion
                             ; de XAR5
        DEC @cont            ; Decrementa la cantidad de
                             ; productos a realizar en la
                             ; siguiente iteración

* Desplazamiento de la secuencia móvil en la operación
* correlación para l<=0
        MOVL ACC,*XAR3       ; Se carga la variable incDir en ACC
        ADD ACC,#2           ; Suma 2 al valor de ACC, para
                             ; desplazar la secuencia
        MOVL *XAR3,ACC       ; incDir=ACC
        MOVL XAR2,#yn        ; Se apunta a la dirección de y
        MOVL ACC,XAR2        ; Se coloca la dirección anterior
                             ; en el ACC
        ADDL ACC,*XAR3       ; Se suma a la dirección el valor
                             ; de incDir
        MOVL XAR2,ACC        ; Se regresa el resultado al
                             ; apuntador de la secuencia
                             ; móvil y(n) regresando el valor
                             ; a la var. delDir

        BANZ CORRELN,AR4--   ; Ciclo de cálculo de los N
                             ; elementos de la correlación para
                             ; l<=0

*************************************************************************
* Segunda parte de la correlación cruzada, para l > 0
        MOVW DP,#cont        ; Direccionamiento a la página de
                             ; memoria que contiene la
                             ; variable cont
        MOV AR4,#N-1         ; Escribe la cantidad de
                             ; iteraciones a realizar para l>0
        MOVL XAR2,#xn        ; Direccionamiento al vector
                             ; de datos x, secuencia movil
        MOV AL,#N+1          ; Coloca el número 501 en el AL
        MOVW @cont,AL        ; Escribe la cantidad de productos
                             ; a realizar en cont
        MOVL XAR5,#rxy+2*N-1          ; Se apunta con XAR5 al
                                      ; elemento 499 del vector rxy
        ZAPA
        MOVL *XAR3,ACC       ; incDir=0

CORRELP
        MOVL XAR7,#yn        ; Se apunta a la dirección de yn
```

```
        ZERO R2H
        ZERO R3H
        ZERO R6H
        ZERO R7H

        RPT @cont ; Ciclo para realizar la MAC
        || MACF32 R7H,R3H,*XAR2++,*XAR7++
                            ; Cálculo de la MAC a 32 bits
                            ; y(n) y x(n) apuntados por
                            ; XAR2 y XAR7 respectivamente

        ADDF32 R7H,R7H,R3H  ; Acumulación del último producto
        NOP
        MOV32 *XAR5++,R7H   ; Guarda el resultado total de
                            ; la MAC en rxy(l) y postincrementa
                            ; el apuntador de rxy

        DEC @cont           ; Decrementa la cantidad de
                            ; productos a realizar en la
                            ; siguiente iteración

* Desplazamiento de la secuencia móvil en la operación
* correlacion para l>0
        MOVL ACC,*XAR3      ; ACC=incDir
        ADD ACC,#2          ; Suma 2 al valor de ACC, para
                            ; desplazar la secuencia
        MOVL *XAR3,ACC      ; incDir=ACC
        MOVL XAR2,#xn       ; Se apunta a la dirección de x
        MOVL ACC,XAR2       ; Coloca la dirección en el ACC
        ADDL ACC,*XAR3      ; Suma la variable incDir al ACC
        MOVL XAR2,ACC       ; Regresa la dirección obtenida del
                            ; ACC en el apuntador XAR2
        BANZ CORRELP,AR4--

iend    NOP
        LB iend             ; Ciclo infinito
        .end
```

4.3.4. Análisis de resultados de las implementaciones

En la Figura 4.7 se muestra la gráfica de la secuencia calculada por la implementación de la operación correlación, entre las secuencias $x_s(n)$ e $y_c(n)$ realizado por programa de cálculo matemático, manejando precisión doble en punto flotante.

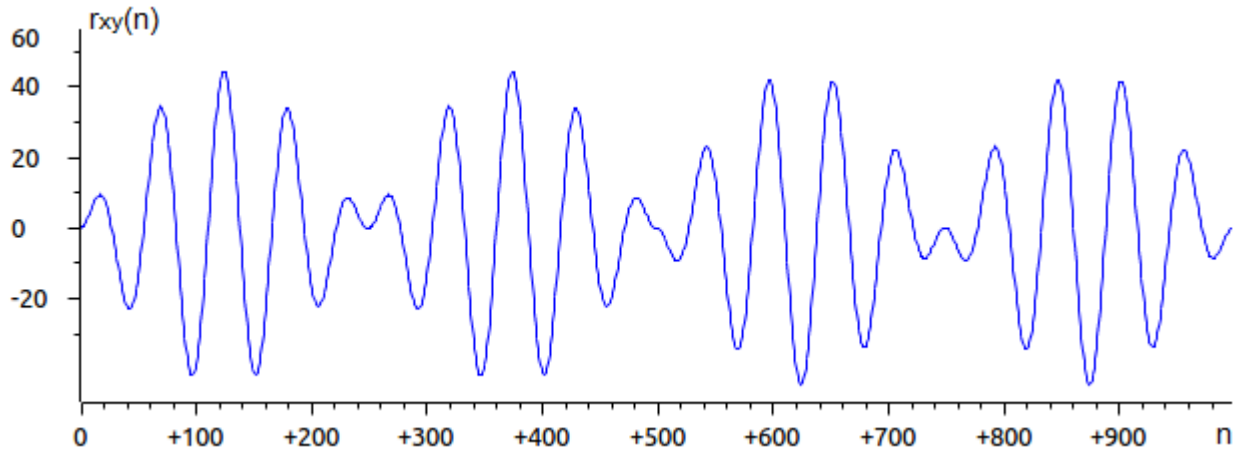


Figura 4.7: Gráfica de la secuencia resultante de hacer la operación correlación entre las secuencias $x_s(n)$ e $y_c(n)$ en formato de punto flotante de doble precisión.

Para analizar los resultados obtenidos por los tres programas implementados de la operación correlación, se obtuvo el error absoluto de cada secuencia obtenida, considerando como referencia la secuencia obtenida por el programa Octave utilizando formato de punto flotante de doble precisión. En la Tabla 4.3 se observan los errores de precisión numérica obtenidos por los códigos propuestos.

Tabla 4.3: Errores absolutos obtenidos de la secuencias calculadas por los programas desarrollados de la correlación.

Formato de entrada	**Formato de salida**	**Error Absoluto**		
		Min.	**Prom.**	**Max.**
Punto flotante simple	Punto flotante simple	1.0000e-09	7.6133e-05	2.4065e-04
Punto entero Q28 (L=32 bits)	Punto entero Q24 (L=32 bits)	1.3923e-08	7.3078e-06	1.5259e-05
Punto entero Q12 (L=16 bits)	Punto entero Q8 (L=16 bits)	1.6e-07	0.0023	0.0074

Con los registros desplegados en la Tabla 4.3, se demuestra que las implementaciones realizadas son correctas al registrar errores significativos con respecto a la secuencia calculada utilizando formato de punto flotante de precisión simple. Por último, de forma significativa la Tabla 4.4 muestra algunos datos de la secuencia obtenida al hacer la correlación entre las secuencias $x_s(n)$ e $y_c(n)$, en el programa Octave de precisión doble y por los programas desarrollados.

Tabla 4.4: Comparación de algunos datos de la secuencia obtenida $y(n)$ por la operación correlación.

Punto flotante de precisión doble	**flotante de precisión simple**	**Punto entero Q24 L=32 bits**	**Punto entero Q8 L=16 bits**
0.1255810390	0.1255810410	0.1255810260	0.125
0.3741285726	0.3741285231	0.3741285204	0.371094
0.7398270907	0.7398270378	0.7398269772	0.738281
1.2137709753	1.2137708791	1.2137708067	1.21094
1.7841366381	1.7841364109	1.7841364741	1.78125

4.4. Filtros digitales

El término *filtro* es utilizado para describir un sistema o un dispositivo que discrimina algunas características o atributos sobre las señales de entrada, es decir, que puede dejar pasar o modificar cierta información y suprimir otra. En procesamiento digital de señales, los filtros digitales realizan funciones similares a los filtros analógicos, además son utilizados en múltiples aplicaciones, como; separar dos señales que han sido combinadas o mezcladas, restaurar señales que han sido distorsionadas, extraer información de interés con base en un conocimiento a priori, remover ruido indeseable, detectar señales, compensar en frecuencias, realizar análisis espectral, modelar señales, entre otras.

Dentro de la clasificación de filtros lineales, existen básicamente dos tipos que son ampliamente utilizados en algoritmos de procesamiento, razón que lleva a su inclusión en este capítulo en las siguientes dos secciones.

4.4.1. Filtros de Respuesta Finita al Impulso FIR

Los filtros digitales de respuesta finita al impulso (FIR) son utilizados ampliamente en el PDS por su estabilidad y sus características de fase lineal que son de importancia en algunas aplicaciones de voz y audio. Debido a que la salida de este tipo de filtro sólo depende de la muestra actual de entrada y de *N-1* retardos de la entrada, también son conocidos como no recursivos, donde N es la longitud del filtro. Por las características de estos filtros, su implementación es precisamente la operación convolución entre la respuesta al impulso del filtro y una ventana de tiempo de la señal de entrada de longitud N [18].

Un sistema lineal e invariante en el tiempo discreto (SLITD) puede ser descrito por una ecuación en diferencias como

$$y(n) = \sum_{m=0}^{q} b_m x(n-m) - \sum_{k=1}^{p} a_k y(n-k) \tag{4.9}$$

Al aplicar la transformada Z (TZ), se obtiene la función de transferencia

$$H(z) = \frac{b_0 + b_1 z^{-1} + \cdots + b_q z^{-q}}{1 + a_1 z^{-1} + \cdots + a_p z^{-p}} \tag{4.10}$$

La respuesta al impulso unitario del sistema, $h(n)$ es la transformada inversa Z(TZI) de la función de transferencia $H(z)$ [12], [15].

Un filtro FIR sólo tiene coeficientes b_k=h_k, es decir, que tiene la característica de ser un sistema no recursivo, por lo que teóricamente siempre es estable. Su respuesta en frecuencia es

$$H(\omega) = \frac{Y(\omega)}{X(\omega)} = \sum_{k=0}^{q} h_k e^{-j\omega k} \tag{4.11}$$

Una desventaja de estos filtros es la necesidad de utilizar un mayor orden para lograr grandes pendientes en la banda de transición, esto implica un mayor tiempo de procesamiento y mayor retardo en la respuesta. La respuesta al impulso de estos filtros normalmente son funciones similares a funciones $sen(x)/x$ con longitud finita, como se verá más adelante.

De la Ecuación (4.10), un filtro FIR tiene la forma

$$H(z) = h_0 + h_1 z^{-1} + \cdots + h_{N-1} z^{-N+1} = \sum_{i=0}^{N-1} h(i) z^{-i} \tag{4.12}$$

con respuesta al impulso

$$h(n) = \begin{cases} h_i & 0 \leq i \leq N-1 \\ 0 & \text{otro } i \end{cases} \tag{4.13}$$

y la ecuación en diferencias

$$\begin{aligned} y(n) &= h_0 x(n) + h_1 x(n-1) + \cdots + h_{N-1} x(n-N+1) \\ &= \sum_{i=0}^{N-1} h(i) x(n-i) \end{aligned} \tag{4.14}$$

la cual es una convolución lineal de la entrada $x\,(n)$ con la respuesta al impulso $h(n)$ del filtro FIR, N es la longitud del filtro y es igual al número de coeficientes del filtro. Ésta es la estructura más sencilla y como su nombre lo indica, la Ecuación (4.1) se implementa directamente utilizando los bloques básicos de un sistema discreto. Para cualquier longitud N se presenta un diagrama de bloques en la Figura 4.8.

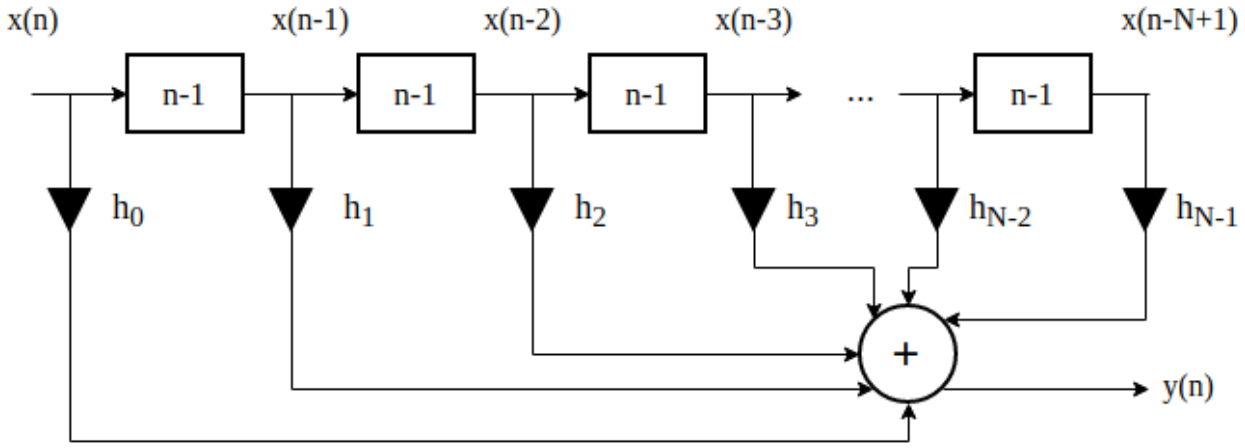


Figura 4.8: Forma directa de un filtro FIR

Filtro FIR promediador

Si se considera una señal con ruido agregado y con una relación señal a ruido (SNR) mediana, entonces una manera simple de eliminar el ruido o suavizar la señal, sería ir calculando el promedio de las muestras sobre una ventana de tiempo y recorrer la ventana. Es decir, que la $h(n)$ propuesta es $1/N$ en toda la ventana y cumple con la condición de simetría de los filtros FIR con diseño de fase lineal.

Este tipo de filtro también es conocido como *Moving average (MA)*, ya que realiza el promedio sobre la muestra actual de entrada $x(n)$ y *N-1* muestras pasadas. Si se visualiza

una señal a baja frecuencia con ruido agregado y aplicando promedios a una señal de entrada $x(n)$ en una ventana de tiempo de longitud N, se puede obtener la salida

$$y(n) = \sum_{i=0}^{N-1} \frac{1}{N} x(n-i) \tag{4.15}$$

si $h(n) = 1/N$, se tiene un filtro que realiza promedios sobre N muestras, aplicando la transformada Z

$$Y(z) = \frac{1}{N} \sum_{i=0}^{N-1} X(z) z^{-i} \tag{4.16}$$

Entre los filtros digitales, los filtros MA producen el menor ruido para bordes muy agudos, la cantidad de reducción de ruido es igual a la raíz cuadrada del número de puntos promediados, por ejemplo si $N = 100$, un filtro FIR MA reduce el ruido por un factor de 10 [18]. La función de transferencia es

$$H(z) = \frac{Y(z)}{X(z)} = \frac{1}{N} \frac{1 - z^{-N}}{1 - z^{-1}} \tag{4.17}$$

y si $z = e^{j\omega}$, su respuesta en frecuencia es

$$H(\omega) = \frac{sen(\pi\omega N/2)}{N sen(\pi\omega/2)} e^{-j(N-1)\omega/2} \tag{4.18}$$

Debido a la respuesta en frecuencia, este tipo de filtros no puede separar eficientemente una frecuencia de otra.

Implementaciones del Filtro FIR

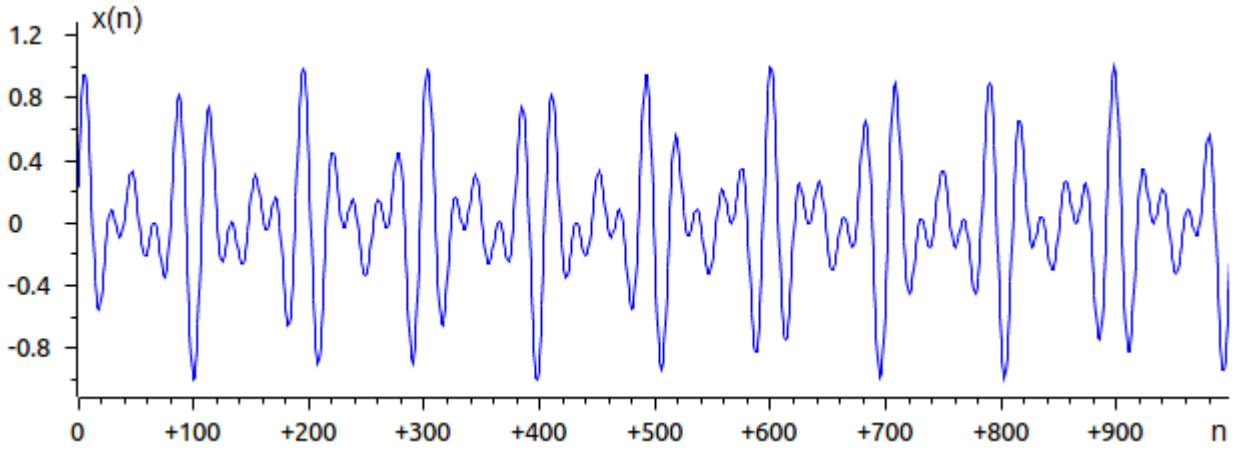
A continuación se presentarán dos implementaciones de un filtro FIR, la primera consiste en la aplicación de un filtro supresor de banda FIR de 255 coeficientes a una secuencia de tres tonos, considerando aritmética de punto fijo con datos de longitud de 16 y 32 bits, y en punto flotante. La segunda implementación es una propuesta de un filtro FIR *moving average*.

Para probar el funcionamiento de la primer implementación, como secuencia de entrada al filtro se generó una secuencia de 10,000 términos considerando como base la siguiente expresión

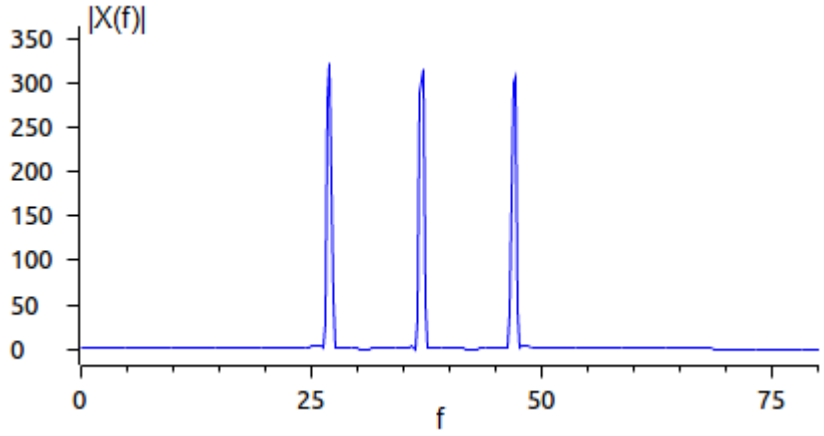
$$x\,(n) = \sin\left(\frac{2\pi f_0 n}{f_s}\right) + \sin\left(\frac{2\pi f_1 n}{f_s}\right) + \sin\left(\frac{2\pi f_2 n}{f_s}\right) \tag{4.19}$$

donde la frecuencia de muestreo es igual a $f_s = 1kHz$, la frecuencia que nos interesa suprimir es $f_1 = 37Hz$, para apreciar la calidad del filtro tenemos a $f_0 = 27Hz$ y $f_2 = 47Hz$, y n es el dominio discreto del tiempo que comprende de $[0, 10000]$. Los datos se almacenaron en archivos *.dat* en formato flotante, para que posteriormente se convirtieran a punto fijo acorde

a cada uno de los programas que se elaboraron. El espectro de la secuencia de entrada se observa en la Figura 4.9, así como los primeros 1000 elementos de la secuencia en el dominio del tiempo.



(a) Primeros 1000 elementos de la secuencia generada por (4.19).



(b) Espectro de magnitud de la secuencia de entrada definida por (4.19).

Figura 4.9: Gráficas de la señal 4.19; en el dominio del tiempo con la decima primer parte de sus puntos y el espectro de magnitud de la secuencia.

Filtro FIR con datos en punto fijo a 16 bits

El programa que se presenta en este apartado, utiliza la secuencia de datos de la Ecuación (4.19) y los coeficientes del filtro en formato de punto fijo, considerando 14 bits para representar la parte fraccionaria, 1 bit para la parte entera y 1 bit para el signo del número, Q_{14}. Las secuencias de datos se encuentran en los siguientes archivos:

- xnQ14-FIR-notch.dat
- hnQ14-FIR-notch.dat

Para probar el funcionamiento del programa, se deben de cargar los archivos de datos a la memoria del DSP, antes de ejecutarlo. El programa del filtro FIR supresor de banda se muestra a continuación.

```
*
*                Filtro FIR supresor de banda
*      Implementación de un fitro FIR con aritmética de
*                punto fijo y datos de 16 bits
*

        .global _c_int00

        .data
N       .set 10000              ; Longitud de la secuencia de datos a
                                ; filtrar x(n)
Ncof      .set 255              ; Cantidad de coeficientes del filtro
xn      .space 16*N             ; Secuencia de datos de entrada x(n)
xb      .space 16*Ncof          ; Reserva espacio para buffer x
xbf     .int 0                  ; Última localidad del buffer x
hn      .space 16*Ncof          ; Vector de coeficientes del filtro FIR
y       .space 16*N             ; Vector para guardar el resultado
WDCR    .set 07029h             ; Dirección registro de control WatchDog

        .text

_c_int00
* Desactivación del WatchDog Timmer
        EALLOW                  ; Habilita escritura a registros
                                ; protegidos
        MOVL XAR1,#WDCR         ; Registro XAR1 apunta a dir. WDCR
        MOV *XAR1,#0068h        ; Desactiva el WatchDog
        EDIS                    ; Deshabilita escritura a registros
                                ; protegidos

* Calculo del filtro FIR
        SETC SXM                ; Modo extensión de signo
```

```
        MOVW DP,#xb             ; Direccionamiento a la página de
                                ; de memoria donde está definida
                                ; la variable xb

        MOVL XAR1,#xn           ; Direccionamiento al vector xn
        MOVL XAR2,#y            ; Direccionamiento al vector y
        MOV AR4,#N-1            ; Ciclo de cálculo del filtro

FIR_CYCLE
        MOV AL,*XAR1++          ; Coloca el dato apuntado por XAR1
                                ; en ACC
        MOV @xb,AL              ; Trae al dato al buffer
        MOVL XAR7,#xb           ; Direccionamiento al buffer xb
        MOVL XAR3,#hn           ; Se guarda la dirección del
                                ; vector hn
        ZAPA

        RPT #Ncof-1             ; Ciclo de cálculo de convoluciones
        || MAC P,*XAR3++,*XAR7++

        ADDL ACC,P              ; Acumula el último producto
        ;LSL ACC,#4             ; Ajuste de Qi
        MOV *XAR2++,AH          ; Guarda el resultado en el vector y

* Desplazamiento del buffer xb
        RPT #Ncof-1             ;  Ciclo para reacomodo de datos en
                                ; el buffer xb
        || DMOV *--XAR7         ; Mueve Ncof veces los datos
                                ; del buffer

* Fin del desplazamiento de datos del buffer xb

        BANZ FIR_CYCLE, AR4--

iend    NOP
        LB iend                 ; Ciclo infinito
        .end
```

La estructura del programa permite implementar cualquier tipo de filtro FIR, puesto que se puede cambiar la cantidad de coeficientes, rediseñando el filtro y la cantidad de datos de entrada.

Filtros FIR con datos punto fijo a 32 bits

El segundo programa que se presenta en este apartado, utiliza la secuencia de datos de la Ecuación (4.19) y los coeficientes del filtro en formato de punto fijo, considerando 30 bits

para representar la parte fraccionaria, 1 bit para la parte entera y 1 bit para el signo del número. Las secuencias de datos se encuentran en los siguientes archivos:

- xnQ30-FIR-notch.dat
- hnQ30-FIR-notch.dat

Para probar el funcionamiento del programa, se deben de cargar los archivos de datos a la memoria del DSP, antes de ejecutarlo. El programa que aplica un filtro FIR supresor de banda a 32 bits, se presenta a continuación.

```
*
*                Filtro FIR supresor de banda
*      Implementación de un fitro FIR con aritmética de
*                punto fijo y datos de 32 bits
*

        .global _c_int00

        .data
N       .set 10000              ; Cantidad de muestras
Ncof    .set 255                ; Tamaño de la ventana
xn      .space 32*N             ; Secuencia de datos de entrada x(n)
xb      .space 32*Ncof          ; Reserva espacio para buffer x
xbf     .long 0                 ; Última localidad del buffer x
hn      .space 32*Ncof          ; Vector de coeficientes del filtro FIR
y       .space 32*N             ; Vector para guardar el resultado
WDCR    .set 07029h             ; Dirección del reg. de control WatchDog

        .text

_c_int00
* Desactivación del WatchDog Timmer
        EALLOW                  ; Habilita escritura a registros protegidos
        MOVL XAR1,#WDCR         ; Registro XAR1 apunta a dir. WDCR
        MOV *XAR1,#0068         ; Desactiva WatchDog
        EDIS                    ; Deshabilita escritura a registros
                                ; protegidos

* Cálculo del filtro FIR
        SETC SXM                ; Modo extensión de signo
        MOVW DP,#xb             ; Direcciona el apuntador de página
                                ; a la página de memoria a xb
        MOVL XAR1,#xn           ; Direccionamiento al vector xn
        MOVL XAR2,#y            ; Direccionamiento al vector y
        MOV AR4,#N-1            ; Ciclo de cálculo del filtro

FIR_CYCLE
```

```
        MOVL ACC,*XAR1++        ; Coloca el dato apuntado
                                ; por XAR1 en ACC
        MOVL @xb,ACC            ; Trae al dato al buffer
        MOVL XAR7,#xb           ; Direccionamiento al buffer xb

        MOVL XAR3,#hn           ; Se guarda la dirección del
                                ; vector hn
        ZAPA                    ; ACC=0 y P=0

        RPT #Ncof-1             ; Ciclo de cálculo
        || QMACL P,*XAR3++,*XAR7++

        ADDL ACC,P              ; Acumula el último producto
        LSL ACC,#4              ; Ajuste de Qi.
        MOVL *XAR2++,ACC        ; Guarda el resultado en el vector y

* Desplazamiento del buffer xb
* Se apunta a la localidad extra al final del buffer xb,
* y a la penúltima de xb, para copiar los datos de la parte final
* a la inicial en retroceso

        MOVL XAR3,#xbf
        MOVL XAR7,#xbf-2
        MOV AR5,#Ncof-1

move_buffer
        MOVL ACC,*--XAR7
        MOVL *--XAR3,ACC
        BANZ move_buffer,AR5--

        BANZ FIR_CYCLE, AR4--

iend    NOP
        LB iend                 ; Ciclo infinito
        .end
```

Filtros FIR en punto flotante IEEE 754

El último programa de un filtro FIR, opera los datos en formato de punto flotante, siendo las secuencias de datos se encuentran en los siguientes archivos:

- xnFloat-FIR-notch.dat
- hnFloat-FIR-notch.dat

```
*
*               Filtro FIR supresor de banda
*     Implementación de un fitro FIR con aritmética de
```

```
*            punto flotante y datos de 32 bits
*
        .global _c_int00

        .data
N       .set 10000              ; Puntos de la señal a filtrar
Ncof    .set 255                ; Cantidad de coeficientes
WDCR    .set 07029h             ; Dirección del reg. de control WatchDog

xn      .space 32*N             ; Secuencia de datos de entrada x(n)
xb      .space 32*Ncof          ; Reserva espacio para buffer x
xbf     .float 0                ; Última localidad del buffer x

hn      .space 32*Ncof          ; Vector de coeficientes del filtro FIR
y       .space 32*N             ; Vector para guardar el resultado

        .text

_c_int00
* Desactivación del WatchDog Timmer
        EALLOW                  ; Habilita escritura a registros
                                ; protegidos
        MOVL XAR1, #WDCR        ; Registro XAR1 apunta a dir. WDCR
        MOV *XAR1,#0068h        ; Desactiva WatchDog
        EDIS                    ; Deshabilita escritura a registros
                                ; protegidos
* Cálculo del filtro FIR
        SETC SXM                ; Modo extensión de signo
        MOVW DP,#xb

        MOVL XAR1,#xn           ; Direccionamiento al vector xn
        MOVL XAR2,#y            ; Direccionamiento al vector y
        MOV AR4,#N-1            ; Ciclo de cálculo del filtro

FIR_CYCLE
        MOVL ACC,*XAR1++        ; Coloca el dato apuntado por XAR1 en ACC
        MOVL @xb,ACC            ; Trae al dato al buffer
        MOVL XAR7,#xb           ; Direccionamiento al buffer xb
        MOVL XAR3,#hn           ; Se guarda la dirección del vector hn

        ZAPA

        ZERO R1H                ; Escribe cero en el registro R1H
        ZERO R2H                ; Escribe cero en el registro R2H
        ZERO R3H                ; Escribe cero en el registro R3H
        ZERO R7H                ; Escribe cero en el registro R7H

        MOV AR5,#Ncof-1         ; Carga la cantidad de iteraciones
        ZAPA
```

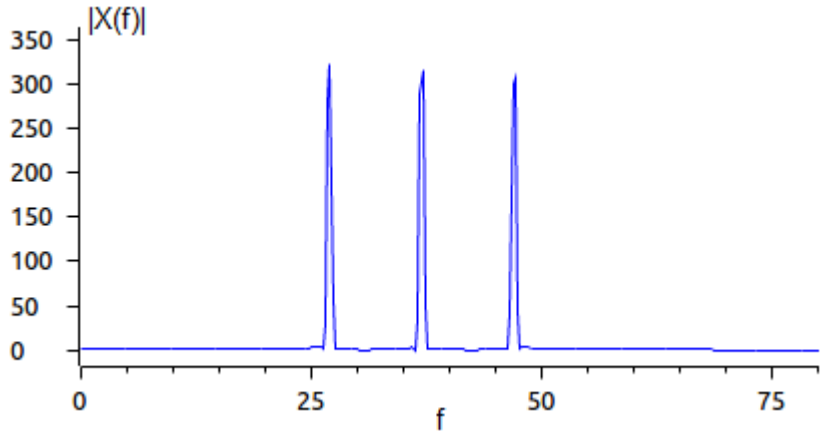
```
        RPTB mac_cycle,AR5      ; Ciclo de cálculo de la oper. MAC
            MOV32 R3H,*XAR3++
            MOV32 R7H,*XAR7++
            MPYF32 R2H,R3H,R7H
            NOP
            ADDF32 R1H,R2H,R1H
            NOP
mac_cycle:

        MOV32 *XAR2++,R1H       ; Guarda el resultado en el vector y

* Desplazamiento del buffer xb
* Se apunta a la localidad extra al final del buffer xb, y a la
* penúltima de xb, para copiar los datos de la parte final a la
* inicial en retroceso
        MOVL XAR3,#xbf
        MOVL XAR7,#xbf-2
        MOV AR5,#Ncof-1

move_buffer
        MOVL ACC,*--XAR7
        MOVL *--XAR3,ACC
        BANZ move_buffer,AR5--

        BANZ FIR_CYCLE, AR4--

iend    NOP
        LB iend                 ; Ciclo infinito
        .end
```

Análisis de resultados de las implementaciones del filtro supresor FIR

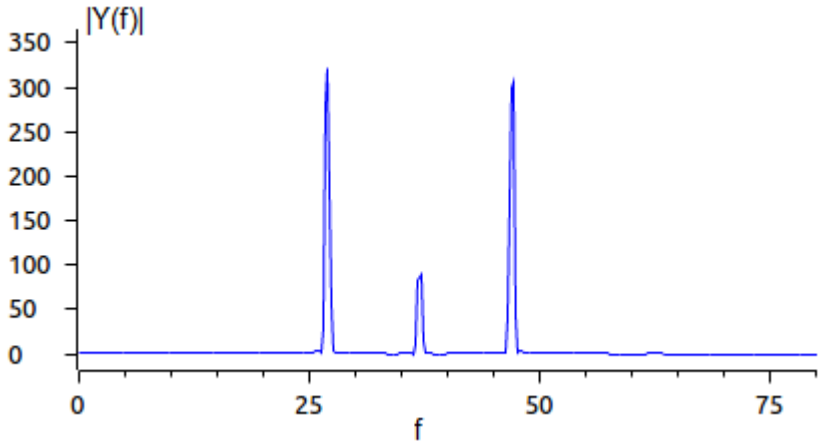
En la Figura 4.10 se muestran las gráficas del espectro de magnitud de la secuencia de entrada $x(n)$ al filtro, comparada con el espectro de magnitud de la respuesta obtenida $y(n)$ por el filtro supresor FIR propuesto en las implementaciones.

La atenuación que se obtuvo de la frecuencia de interés a suprimir, 37 Hz es aproximadamente de -10.8989 db, lo cual se puede observar y corroborar con la gráfica de la respuesta al impulso del filtro supresor FIR que se muestra en la Figura 4.11.

La banda de supresión en general, tiene un ancho de 20 Hz presentando una atenuación mayor a los -3 db en $\pm$ 5 Hz respecto a la frecuencia de interés a suprimir.



(a) Espectro de magnitud de la secuencia de entrada $x\,(n)$.



(b) Gráfica del espectro en magnitud de la respuesta del filtro supresor FIR.

Figura 4.10: Comparación del contenido espectral de las señales $x\,(n)$ y $y\,(n)$.

Para evaluar el desempeño de las implementaciones propuestas, se obtuvo la respuesta al impulso de cada uno de los filtros programados, respetando el formato de los datos de entrada y de salida, para apreciar los errores de precisión numérica. Las secuencias resultantes se guardaron en archivos y posteriormente fuera de línea fueron analizadas utilizando Octave.

Al convertir los formatos de punto entero a punto flotante, se obtuvo el espectro de magnitud de la respuesta al impulso de cada uno de los filtros y se calcularon los errores absolutos mínimos, tomando como referencia la respuesta obtenida en el programa de cálculo donde se extrajeron los coeficientes. Los errores absolutos de las respuestas al impulso obtenidos se muestran en la Tabla 4.5.

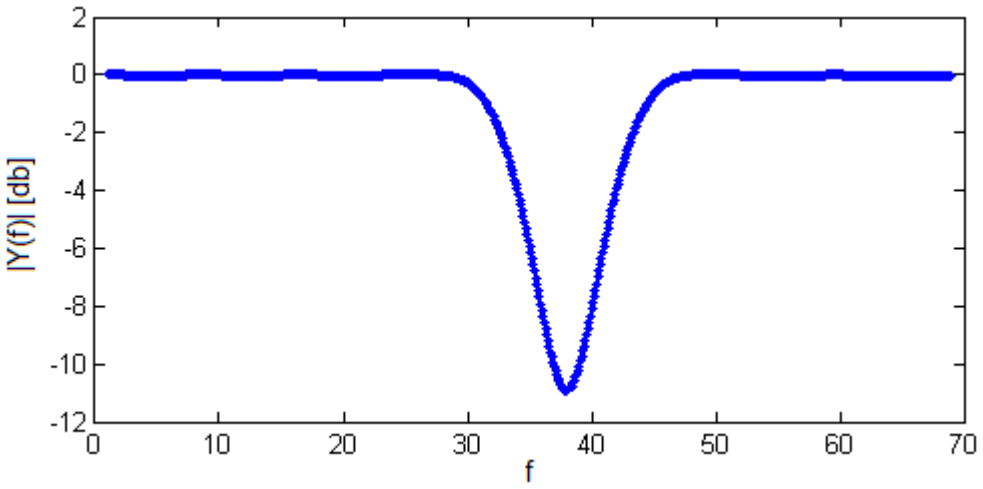


Figura 4.11: Respuesta al impulso del filtro supresor FIR de orden 255.

Tabla 4.5: Comparación de resultados obtenidos de los programas de implementación del filtro supresor FIR

Formato de entrada	**Formato de salida**	**Error absoluto mínimo**	**Atenuación db**
Q14 L=16 bits	Q14 L=16 bits	8.4487e-06	-10.8970
Q30 L=32 bits	Q30 L=32 bits	3.1700e-07	-10.8980
Flotante	Flotante	1.0806e-07	-10.8981

La cuarta columna de la Tabla 4.5, registra la máxima atenuación obtenida por cada uno de los filtros en la frecuencia de interés. ***Las conclusiones*** de los resultados obtenidos por las diferentes implementaciones de los filtros FIR de forma directa son:

- La atenuación de la frecuencia de interés no presentó cambios significativos debido al manejo de diferentes formatos numéricos, por lo que estos no influyen en la respuesta del filtro.

- La banda de supresión del filtro es de un tamaño considerable, afectando a otras frecuencias que podrían ser de interés conservarlas, sin embargo el factor de atenuación no es de una magnitud muy grande.

- Si se necesita, tener una alta precisión numérica en alguna aplicación de este filtro, es recomendable utilizar los formatos de 32 bits, en particular el de punto fijo.

El diseño propuesto de filtro supresor FIR de orden 256 no logró suprimir del todo la componente de frecuencia de 37 Hz, pero si la atenuó en cierta proporción, esto se debe principalmente al desempeño natural del filtro y a su orden, sin embargo al incrementar la cantidad de coeficientes, se necesitaría una mayor cantidad de memoria, lo cual sería uno de los costos de la aplicación para poder tener un mayor factor de atenuación.

Estas razones son las que se tienen que ponderar al momento de decidir el tipo de filtro que se puede aplicar a la implementación que se desea, ya que un filtro recursivo como un IIR de segundo orden puede obtener un mayor factor de atenuación, pero la fase del filtro no se conserva lineal a diferencia del filtro FIR estudiado en esta sección.

Filtro FIR promediador en punto fijo a 16 bits

La segunda implementación tiene por objetivo visualizar el efecto de un filtro FIR promediador, como reductor de ruido blanco, para lo cual se eligió una señal sinusoidal con ruido blanco aditivo considerando 2 db de SNR, como secuencia de entrada. La ecuación que se consideró como base para generar la secuencia, es decir, previo a sumarle ruido blanco fue la siguiente

$$x'(n) = \sin\left(\frac{2\pi f_0 n}{f_s}\right) \tag{4.20}$$

donde $f_0 = 10Hz$, $f_s = 1kHz$ y n es el tiempo discreto definido en el intervalo de $[0, 500]$.

Con el ruido blanco agregado, la secuencia se convirtió en formato de punto fijo representando la parte fraccionaria en $Q_i = 12$. El resultado de esta conversión se guardó en el archivo *xnQ12-FIR-Prom.dat*, para escribir los datos en la memoria.

El código propuesto considera una ventana de 32 muestras, para calcular el promedio de dicha ventana y obtener el n término de la secuencia filtrada, y previo a la siguiente iteración, los datos de la ventana se desplazan una posición para poder calcular el siguiente término, una vez que se haya introducido el nuevo elemento de $x(n)$ a la ventana, muy similar a lo que se realizó en las implementaciones de la convolución y correlación. A continuación se presenta el programa propuesto.

```
*
*           Filtro FIR promediador
* con aritmética y datos de 16 bits en punto fijo
*
*
        .global _c_int00

        .data
N       .set 500                ; Cantidad de puntos de x(n)
Nf      .set 32                 ; Datos a promediar
WDCR    .set 07029h             , Dirección del reg. WatchDog
DR      .set 5                  ; Cantidad de bits de corrimiento

xav     .space Nf*16            ; Buffer de la ventana a promediar
xavf    .word 0,0,0

x       .space N*16             ; Espacio para la señal x(n)
y       .space N*16             ; Espacio para la respuesta y(n)

        .text
_c_int00
* Deshabilitación del WatchDog
        EALLOW                  ; Habilita escritura a registros
                                ; protegidos
        MOVL XAR1, #WDCR        ; Registro XAR1 apunta a dir. WDCR
        MOV *XAR1,#0068h        ; Desactiva WatchDog
        EDIS                    ; Deshabilita escritura a registros
                                ; protegidos

        SETC SXM                ; Modo extensión de signo
        SETC OVM                ; Habilita operaciones con overflow
        SPM #0                  ; Corrimientos nulos al operar con P

        MOVW DP,#xav            ; Direcionamiento a la página donde
                                ; se alojó la variable xav
        MOVL XAR1,#x            ; XAR1 apunta a inicio de x(n)
        MOVL XAR2,#y            ; XAR2 apunta a inicio de y(n)
        MOV AR4,#N-1            ; Escribe la cantidad de iteraciones
                                ; a realizar

CICLO_MA
        MOV AL,*XAR1++          ; Muestrea el dato x(n)
        MOV @xav,AL             ; Escribe x(n) al inicio del buffer
        MOVL XAR3,#xav          ; XAR3 apunta al inicio del buffer

        ZAPA                    ; ACC = 0   y P = 0

        RPT #Nf-1
        || ADD ACC,*XAR3++      ; Suma los datos del buffer
```

```
        MOV T,#DR               ; Carga datos en T
        ASRL ACC,T              ; Divide datos entre 32
        MOV *XAR2++,AL          ; Escribe AL a salida y(n)
        MOVL XAR3,#xavf         ; XAR3 apunta al inicio del buffer

        RPT #Nf-1
        || DMOV *--XAR3         ; Mueve los datos del buffer

        BANZ CICLO_MA,AR4--     ; Regresa a CICLO_M si
                                ; AR4!=0 y AR4 = AR4-1

FIN_M   NOP
        LB FIN_M                ; Ciclo infinito
        .end
```

El programa anterior calcula la división, recorriendo a la derecha el registro ACC cinco posiciones, lo cuál a nivel de bits representa justamente una división entre 32, que es igual a la longitud de la ventana de datos a promediar.

Análisis de resultados del filtro promediador.

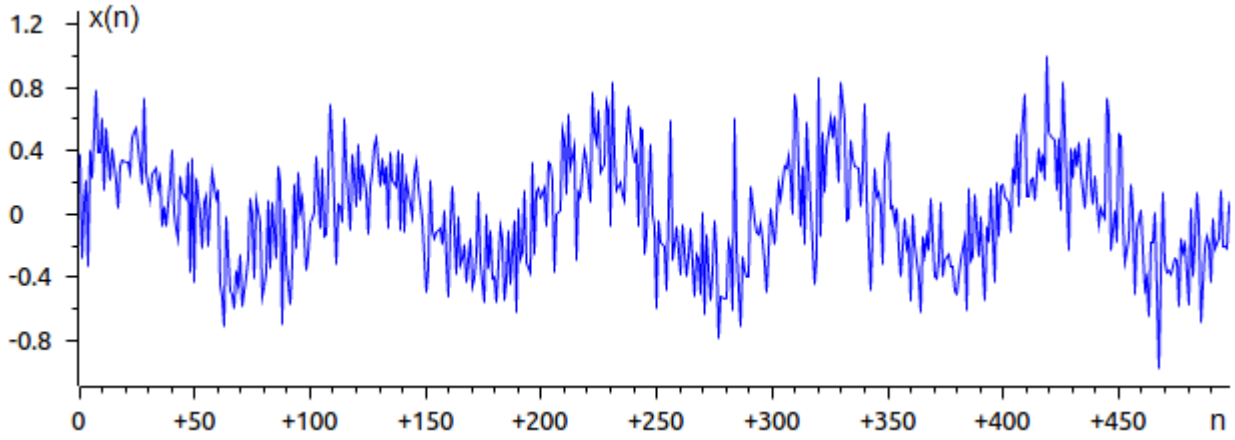
El resultado ideal obtenido por el filtro FIR promediador, se puede observar de forma gráfica en la Figura 4.12. Esta respuesta fue obtenida por un programa de cálculo que opera con formatos de doble precisión numérica de punto flotante, con el objetivo de poder analizar el desempeño del programa propuesto del filtro a 16 bits en formato de punto fijo.

La secuencia de resultado que se obtuvo, en formato Q_{12} se convirtió a formato de punto flotante y se cálculo el error absoluto tomando como referencia la secuencia de doble precisión numérica de punto flotante, y se obtuvo el comportamiento del error absoluto mostrado a continuación:

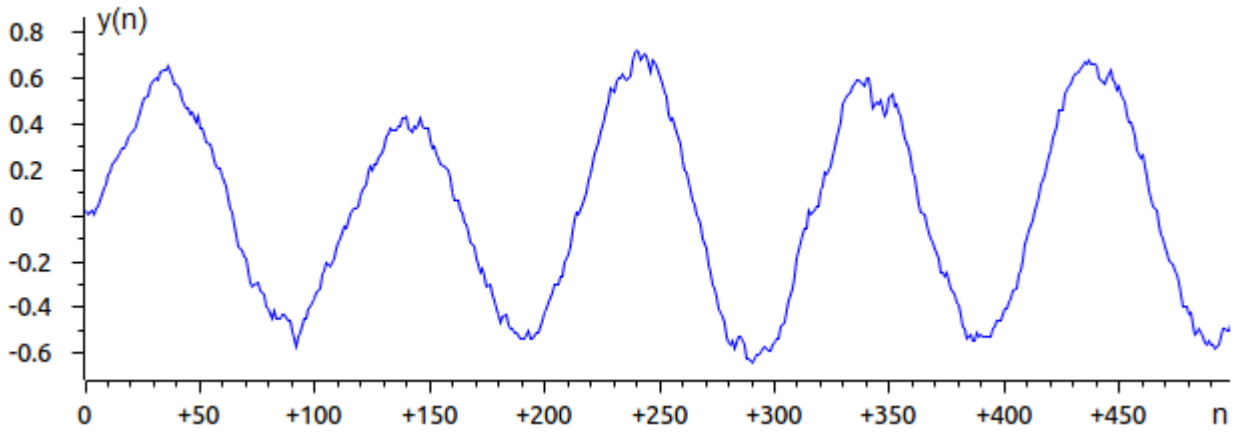
Tabla 4.6: Errores absolutos obtenidos por el resultado de la implementación del filtro FIR promediador.

Formato de datos en punto fijo L=16 bits	**Error absoluto**		
	Min.	**Prom.**	**Max.**
Q12	2.6465e-04	0.0083	0.0311

Este error se puede disminuir al manejar formatos de punto fijo o flotante de 32 bits, sin embargo, por el bajo desempeño de un filtro promediador, no se puede obtener la sinusoidal definida por la Ecuación (4.20), sin embargo, es evidente que este filtro si disminuyó el ruido blanco de la secuencia de entrada.



(a) Secuencia de entrada $x\,(n)$ del filtro FIR promediador.



(b) Gráfica de la secuencia $y\,(n)$ obtenida al filtrar $x\,(n)$ con el filtro promediador.

Figura 4.12: Comparación de las secuencias $x\,(n)$ y $y\,(n)$, que representan el efecto del filtro FIR promediador.

4.4.2. Filtros de Respuesta Infinita al Impulso IIR

Los filtros de respuesta infinita al impulso (IIR) también son llamados recursivos y consisten de dos partes; una que efectúa la suma ponderada de la entrada $x\,(n)$ y retardos de $x\,(n-i)$ y la otra efectúa una suma ponderada de las salidas $y\,(n-i)$ con retraso en el tiempo [18].

Todo filtro analógico genera una respuesta infinita al impulso, entonces al diseñar con esta metodología un sistema discreto IIR puede emular e incluso llegar a superar la respuesta analógica. Algunas de las características de los filtros IIR son:

- Debido a su recursividad, cuando se implementan en aritmética de punto fijo *pueden*

llegar a ser inestables aún cuando los polos se encuentren dentro del círculo unitario.

- *No son de fase lineal* pero presentan mejor repuesta en frecuencia que los filtro FIR.
- Con pocos coeficientes se obtienen grandes pendientes en la banda de transición.

La salida de un sistema lineal y discreto puede escribirse como la convolución de la entrada con su respuesta al impulso, entonces la respuesta al impulso de un filtro IIR se puede expresar como:

$$y(n) = x(n) * h(n) = \sum_{i=0}^{\infty} h(i)\, x(n-i) \tag{4.21}$$

Sin embargo, (4.21) no puede implementarse en aplicaciones, por ello para que un sistema discreto de componentes finitas pueda generar una salida infinita con una entrada impulso, el sistema debe de ser recursivo, entonces para poder realizar un filtro o sistema IIR es necesario tener una ecuación en diferencias que retro-alimente la salida retardada en el tiempo discreto, es decir

$$y(n) = \sum_{i=0}^{q} b(i)\, x(n-i) - \sum_{i=1}^{p} a(i)\, y(n-i) \tag{4.22}$$

Para implementar directamente (4.22) se desarrollan las sumatorias obteniendo:

$$\begin{aligned} y(n) =& b_0 x(n) + b_1 x(n-1) + \cdots + b_q x(n-q) - \\ & - a_1 y(n-1) - a_2 y(n-2) - \cdots - a_p y(n-p) \end{aligned} \tag{4.23}$$

En la expresión anterior, se observan dos líneas de retardo; una para la señal de entrada $x(n)$ y la otra para la señal de salida $y(n)$, esto implica que se tendrán $p+q$ bloques de retardo, como se muestra en el diagrama de bloques de la Figura 4.13.

La implementación de un filtro IIR se puede ver como la convolución de los coeficientes b_m con la señal de entrada menos la convolución de la señal de salida retardada con los coeficientes a_m, esto se aprecia en la Figura 4.14.

Implementación de un filtro supresor de banda IIR

A continuación, se presentará la implementación de un filtro IIR, considerando aritmética de punto fijo con datos de longitud de 16 y 32 bits, y en punto flotante. Se consideró filtrar una secuencia de 10,000 términos generada con la siguiente ecuación:

$$x(n) = \sin\left(\frac{2\pi f_0 n}{f_s}\right) + \sin\left(\frac{2\pi f_1 n}{f_s}\right) + \sin\left(\frac{2\pi f_2 n}{f_s}\right) \tag{4.24}$$

donde la frecuencia de muestreo es igual a $f_s = 1$ kHz, la frecuencia que nos interesa suprimir es $f_1 = 121$ Hz y como frecuencias de referencia, para apreciar la calidad del filtro tenemos a $f_0 = 111$ Hz y $f_2 = 131$ Hz, y n es el dominio discreto del tiempo que comprende de $[0, 10000]$.

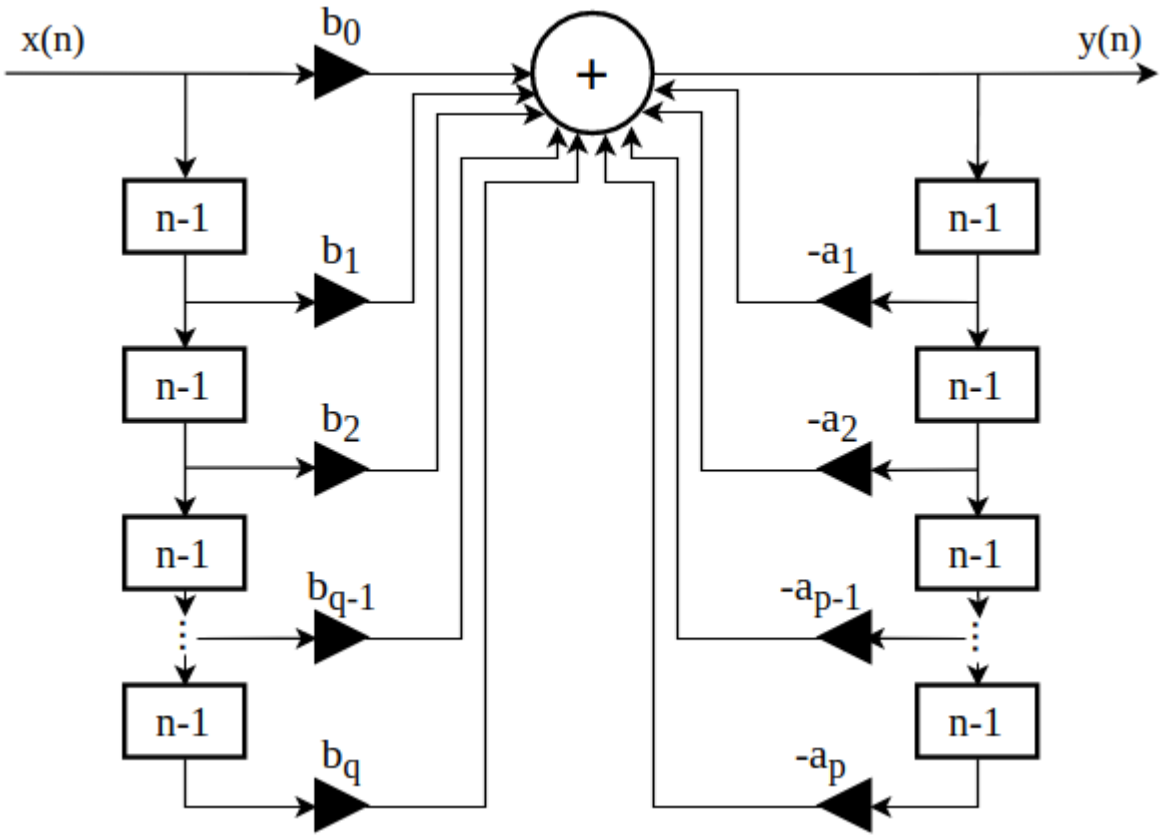


Figura 4.13: Diagrama de bloques de un filtro IIR en estructura directa I.

Los datos se almacenaron en archivos *.dat* en formato flotante, para que posteriormente se convirtieran a punto fijo acorde a cada uno de los programas que se elaboraron. El espectro de la secuencia de entrada se observa en la Figura 4.15.

Filtro IIR de segundo orden en forma directa IIR a 16 bits

El primer programa que se presenta, utiliza la secuencia de datos de la Ecuación (4.24) y los coeficientes del filtro en formato de punto fijo, considerando un formato para la representación decimal de $Q_i = 12$. Las secuencias de datos se encuentran en los siguientes archivos:

* xnQ12-IIR-notch.dat
* cof-apQ12-IIR-notch.dat
* cof-bqQ12-IIR-notch.dat

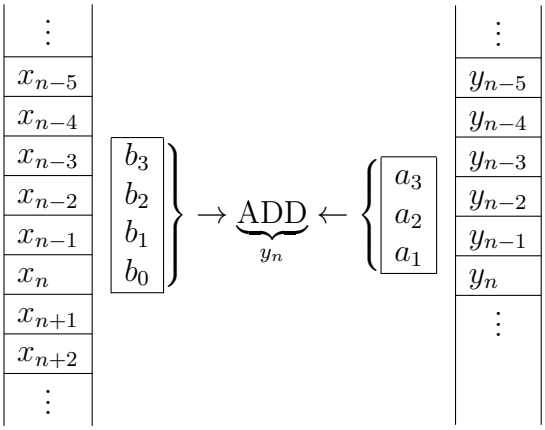


Figura 4.14: Líneas de retardo de un filtro IIR.

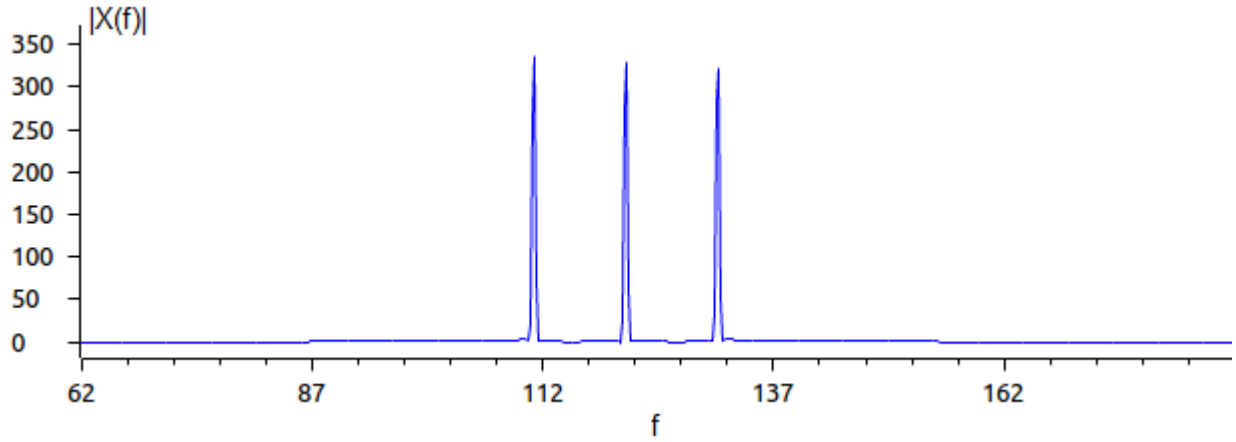


Figura 4.15: Espectro de magnitud de la secuencia de entrada definida por la Ecuación (4.24).

El código del filtro IIR de segundo orden propuesto consiste en lo siguiente:

```
*
* Programa de un filtro IIR de segundo orden
* considerando aritmética de punto fijo con datos de 16 bits
* El filtro implementado es un supresor de banda de 121 Hz,
*  aplicando la ecuación directa del filtro IIR de segundo orden
*
* y(n)=b0*x(n)+b1*x(n-1)+b2*x(n-2)+a1*y(n-1)+a2*y(n-2)
*
* donde los coeficientes ai estan multiplicados por -1
* El resultado se obtiene en punto fijo a Q12
```

```
*

        .global _c_int00
        .data
WDCR    .set    07029h          ; Dirección del reg. de control WD
N       .set    10000           ; Longitud de la secuencia a filtrar
Ncofb   .set    3               ; Cantidad de coeficientes para
                                ; retardos de x
Ncofa   .set    2               ; Cantidad de coeficientes para
                                ; retardos de y
xbsiz   .set    Ncofb           ; Buffer para guardar datos de
                                ; productos de x
Nprod   .set    5               ; Cantidad de multiplicaciones
xb      .space  xbsiz*16        ; Reserva espacio para buffer x
bufy1   .space  1*16
bufy2   .space  1*16
cofb    .space  Ncofb*16        ; Vector de coeficientes ai
cofa    .space  Ncofa*16        ; Vector de coeficientes bi
xn      .space  N*16            ; Secuencia de datos de entrada x(n)
yn      .space  N*16            ; Vector para guardar el resultado
basyn   .word   0               ; Localidad extra de separacion
                                ; de datos
        .text
_c_int00
* Desactivación del WatchDog
        EALLOW                  ; Habilita escritura a registros
                                ; protegidos
        MOVL    XAR1,#WDCR      ; Registro XAR1 apunta a dir. WDCR
        MOV     *XAR1,#0068h    ; Desactiva WatchDog
        EDIS                    ; Deshabilita escritura a registros
                                ; protegidos
        SETC    SXM             ; Modo extensión de signo
        SPM     #0              ; Corrimientos nulos en P
        MOVL    XAR6,#yn        ; Respaldo del direccionamiento al
                                ; vector y(n), para poder
                                ; conservar y(n-1)
        MOVL    XAR1,#xn        ; Direccionamiento a la secuencia
                                ; de datos de entrada
        MOV     DP,#xb          ; Apunte de página de memoria
                                ; tomando como referencia la xb
        MOVL    XAR2,#yn        ; Direccionamiento al vector y(n)
        MOV     AR4,#N-1        ; Carga del total de iteraciones

IIR_CYCLE
        MOV     AL,*XAR1++      ; Copia el dato apuntado por XAR1
                                ; en parte baja de ACC
        MOV     @xb,AL          ; Mueve de forma directa el dato
                                ; de AL al buffer xb
        MOVL    XAR7,#xb        ; Direccionamiento al buffer xb
```

```
        MOVL    XAR3,#cofb      ; Direccionamiento a b1
        ZAPA
        RPT #Nprod-1            ; Ciclo de 5 operaciones
        || MAC P,*XAR3++,*XAR7++    ; Cálculo de operaciones MAC
        ADDL    ACC,P<<PM       ; Acumulación del último resultado
        LSL     ACC,#4          ; Ajuste de formato punto fijo
        MOV     *XAR2++,AH      ; Guarda el resultado en la
                                ; localidad y(AR4)
        MOVL    XAR7,#bufy2     ; Direccionamiento a la localidad
                                ; y(n-2)
        RPT     #Nprod-1        ; Ciclo para reacomodar datos
                                ; en los buffers
        || DMOV *--XAR7         ; Movimiento de datos en buffers
        MOV     AL,*XAR6++      ; Carga el resultado obtenido
                                ; de y(AR4) en AL
        MOV     @bufy1,AL       ; Carga el dato y(AR4) en la
                                ; variable y(n-1)

        BANZ    IIR_CYCLE,AR4 --    ; Ciclo de cálculo para N
                                ; términos de la secuencia x(n)
iend    NOP
        LB      iend            ; Ciclo infinito
        .end
```

Para probar el funcionamiento del programa, los archivos listados previo al programa de implementación, deben ser grabados en la memoria del TMS320F28377S, siguiendo el procedimiento de la Sección 2.6.1. Esto debe realizarse, previo a ejecutar el programa o correrlo paso por paso.

Filtros IIR en punto fijo a 32 bits

El segundo programa utiliza la secuencia de datos de la Ecuación (4.24) y los coeficientes del filtro en formato de punto fijo, considerando un formato de $Q_i = 28$ para los datos de entrada y salida. Las señal a filtrar y los coeficientes que definen el filtro se encuentran en los siguientes archivos:

- xnQ28-IIR-notch.dat
- cof-apQ28-IIR-notch.dat
- cof-bqQ28-IIR-notch.dat

Antes de ejecutar el programa que se presenta a continuación, se deben grabar en la memoria del DSP los archivos previamente listados, para ver el funcionamiento de la aplicación.

```
*
* Programa de un filtro IIR de segundo orden
* considerando aritmética de punto fijo con datos de 32 bits
```

```
* El filtro implementado es un supresor de banda de 121 Hz,
* aplicando la ecuación directa del filtro IIR de segundo orden
*
* y(n)=b0*x(n)+b1*x(n-1)+b2*x(n-2)+a1*y(n-1)+a2*y(n-2)
* donde los coeficientes ai están multiplicados por -1
*
* El resultado se obtiene en punto fijo a Q28

        .global     _c_int00
WDCR    .set    07029h          ; Dirección del reg. de control WD
N       .set    1000            ; Longitud de la secuencia a filtrar
Ncofb   .set    3               ; Cantidad de coeficientes para
                                ; retardos de x
Ncofa   .set    2               ; Cantidad de coeficientes para
                                ; retardos de y
xbsiz   .set    Ncofb           ; Buffer para guardar datos de
                                ; productos de x
Nprod   .set    5               ; Cantidad de multiplicaciones
                                ; a realizar
xb      .space  xbsiz*32        ; Reserva espacio para buffer x
yn1     .long   0
yn2     .long   0
cofb    .space  Ncofb*32        ; Vector de coeficientes ap
cofa    .space  Ncofa*32        ; Vector de coeficientes bq
xn      .space  N*32            ; Secuencia de datos de entrada x(n)
yn      .space  N*32            ; Vector para guardar el resultado
basyn   .long   0               ; Localidad extra de separación
                                ; de datos
        .text

_c_int00
;       Desactivación del WatchDog Timmer
        EALLOW                  ; Habilita escritura a registros
                                ; protegidos
        MOVL    XAR1,#WDCR      ; Registro XAR1 apunta a dir. WDCR
        MOV     *XAR1,#0068h    ; Desactiva WatchDog
        EDIS                    ; Deshabilita escritura a registros
                                ; protegidos
        SETC    SXM             ; Modo extensión de signo
        SETC    OVM
        SPM     0               ; Corrimientos nulos en P
        MOVL    XAR6,#yn        ; Respaldo del direccionamiento
                                ; al vector y(n) para poder
                                ; conservar y(n-1)
        MOVL    XAR1,#xn        ; Direccionamiento a la secuencia de
                                ; datos de entrada
        MOV     DP,#xb          ; Apunta a página de memoria,
                                ; tomando como referencia xb
        MOVL    XAR2,#yn        ; Direccionamiento al vector y(n)
```

```
        MOV   AR4,#N-1          ; Carga del total de iteraciones

IIR_CYCLE
        MOVL  ACC,*XAR1++       ; Copia el dato apuntado por XAR1
                                ; en parte baja de ACC
        MOVL  @xb,ACC           ; Mueve de forma directa el dato
                                ; de ACC al buffer xb
        MOVL  XAR7,#xb          ; Direccionamiento al buffer xb
        MOVL  XAR3,#cofb        ; Direccionamiento a coeficiente b1
        ZAPA
        RPT #Nprod-1            ; Ciclo de operaciones MAC
        || QMACL P,*XAR3++,*XAR7++  ; Cálculo de operaciones MAC
        ADDL  ACC,P<<PM         ; Acumulación del último resultado
        LSL   ACC,#4            ; Ajuste de formato punto fijo
        MOVL  *XAR2++,ACC       ; Guarda el resultado en y(AR4)
        MOVL  XAR3,#yn2         ; Direccionamiento a y(n-2)
        MOVL  XAR5,#cofb        ; Direccionamiento al coeficiente b0
        MOV   AR7,#Nprod-2      ; Carga del total de iteraciones a
                                ; realizar para reacomodo de datos
move_data
        MOVL ACC,*--XAR3        ; Desplazamiento del dato
                                ; y(n-2) a y(n-1)
        MOVL *--XAR5,ACC        ; Desplazamiento de los datos
                                ; x(n-1) y x(n-2)
        BANZ move_data,AR7--    ; Ciclo de movimiento de datos
                                ; de retardo de x(n) y y(n)

        MOVL ACC,*XAR6++        ; Carga el resultado obtenido
                                ; de y(AR4) en ACC
        MOVL @yn1,ACC           ; Carga el dato y(AR4) en la
                                ; variable y(n-1)

        BANZ IIR_CYCLE,AR4--    ; Ciclo de cálculo para N terminos
                                ; de la secuencia x(n)

iend    NOP
        LB iend                 ; Ciclo infinito
        .end
```

Filtros IIR en punto flotante IEEE 754

Este programa utiliza la unidad FPU para implementar el filtro IIR con datos en formato de punto flotante. Los coeficientes y la señal de entrada al filtro están definidos en los siguientes archivos, los cuáles cuentan con el encabezado necesario y listo para escribir cada uno de ellos, en la memoria y con ello poder probar su funcionamiento:

- xnFloat-IIR-notch.dat

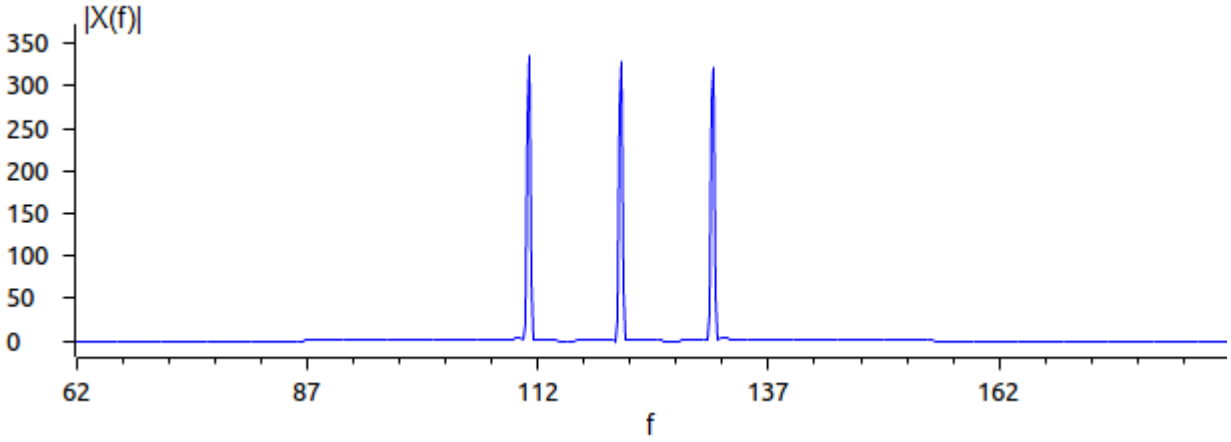
- cof-apFloat-IIR-notch.dat
- cof-bqFloat-IIR-notch.dat

```
* Programa de un filtro IIR de segundo orden
* considerando aritmética de punto flotante con datos de 32 bits
* El filtro implementado es un supresor de banda de 121 Hz,
* aplicando la ecuación directa del filtro IIR de segundo orden
*
* y(n)=b0*x(n)+b1*x(n-1)+b2*x(n-2)+a1*y(n-1)+a2*y(n-2)
*
* donde los coeficientes ai están multiplicados por -1

        .global     _c_int00
        .data
WDCR    .set    07029h          ; Dirección del reg. de WatchDog
N       .set    10000           ; Longitud de la secuencia a filtrar
Ncofb   .set    3               ; Cantidad de coeficientes para
                                ; retardos de x
Ncofa   .set    2               ; Cantidad de coeficientes para
                                ; retardos de y
xbsiz   .set    Ncofb           ; Buffer para guardar datos de
                                ; productos de x
Nprod   .set    5               ; Cantidad de multiplicaciones
xb      .space  xbsiz*32        ; Reserva espacio para buffer x
yn_1    .float  0               ; Variable y(n-1)
yn_2    .float  0               ; Variable y(n-2)
cofb    .space  Ncofb*32        ; Vector de coeficientes ai
cofa    .space  Ncofa*32        ; Vector de coeficientes bi
xn      .space  N*32            ; Secuencia de datos de entrada x(n)
yn      .space  N*32            ; Vector para guardar el resultado
basyn   .float  0               ; Localidad extra de separación
                                ; de datos
        .text
_c_int00
*Desactivación del WatchDog Timmer
        EALLOW                  ; Habilita escritura a registros
                                ; protegidos
        MOVL    XAR1,#WDCR      ; Registro XAR1 apunta a dir. WDC0R
        MOV     *XAR1,#0068h    ; Desactiva WatchDog
        EDIS                    ; Deshabilita escritura a registros
                                ; protegidos
        SETC    SXM             ; Modo extensión de signo
        SETC    OVM             ; Activación de operaciones
                                ; considerando desborde
        MOVL    XAR6,#yn        ; Respaldo del direccionamiento al
                                ; vector y(n), para conservar y(n-1)
        MOVL    XAR1,#xn        ; Direccionamiento a la secuencia
                                ; de datos x(n)
```

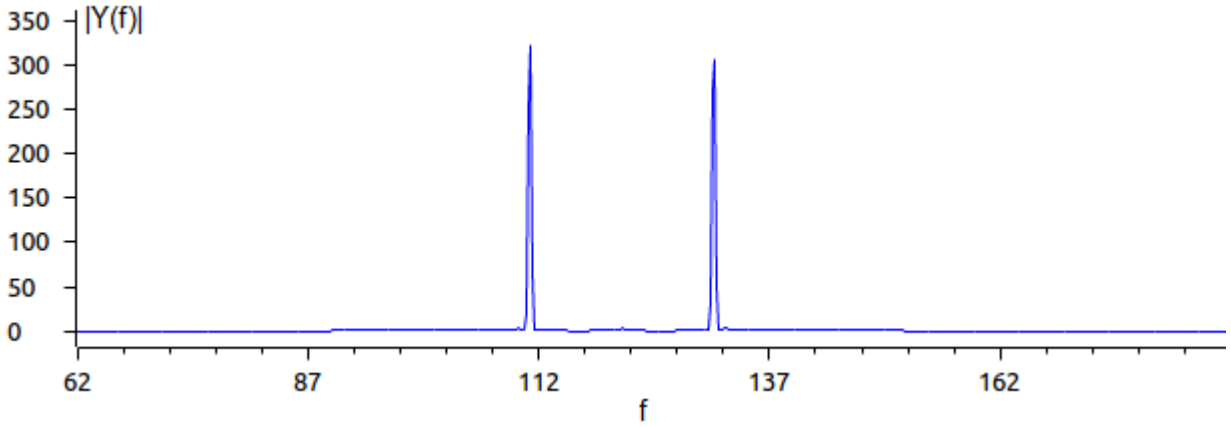
```
        MOVW    DP,#xb          ; Apunta a página de memoria
                                ; tomando como referencia xb
        MOVL    XAR2,#yn        ; Direccionamiento al vector y(n)
        MOV     AR4,#N-1        ; Carga del total de iteraciones

IIR_CYCLE
        MOVL    ACC,*XAR1++     ; Copia el dato apuntado por XAR1 en
                                ; parte baja de ACC
        MOVL    @xb,ACC         ; Mueve el dato de AL a xb
        MOVL    XAR7,#xb        ; Direccionamiento al buffer xb
        MOVL    XAR3,#cofb      ; Direccionamiento a coeficiente b1
        ZAPA
        ZERO    R1H
        ZERO    R2H
        ZERO    R3H
        ZERO    R7H

        MOV     AR5,#Nprod-1
        RPTB    mac_cycle,AR5
                MOV32   R3H,*XAR3++
                MOV32   R7H,*XAR7++
                MPYF32  R2H,R3H,R7H
                NOP
                ADDF32  R1H,R2H,R1H
                NOP
mac_cycle:
        MOV32   *XAR2++,R1H     ; Guarda el resultado en y(AR4)
        MOVL    XAR3,#yn_2      ; Direccionamiento a y(n-2)
        MOVL    XAR5,#cofb      ; Direccionamiento a b0
        MOV     AR7,#Nprod-2    ; Carga del total de iteraciones
                                ; para reacomodo de datos
move_data
        MOVL    ACC,*--XAR3     ; Desplazamiento del dato
                                ; y(n-2) a y(n-1)
        MOVL    *--XAR5,ACC     ; Desplazamiento de los datos
                                ; x(n-1) y x(n-2)
        BANZ    move_data,AR7-- ; Ciclo de movimiento de datos
                                ; de retardo de x(n) y y(n)
        MOVL    ACC,*XAR6++     ; ACC=y(AR4)
        MOVL    @yn_1,ACC       ; y(n-1)=y(AR4)

        BANZ    IIR_CYCLE,AR4-- ; Ciclo de cálculo para N términos
                                ; de la secuencia x(n)
iend    NOP
        LB      iend            ; Ciclo infinito
        .end
```

Análisis de Resultados de los filtros notch IIR

En la Figura 4.16 se muestran las gráficas del espectro de magnitud de la secuencia de entrada $x\,(n)$ al filtro, comparada con el espectro de magnitud de la respuesta obtenida $y\,(n)$ por el filtro supresor IIR de segundo orden propuesto para las implementaciones, realizando los cálculos con precisión doble en punto flotante.



(a) Espectro de magnitud de la secuencia de entrada $x\,(n)$.



(b) Gráfica del espectro en magnitud de la respuesta del filtro supresor IIR.

Figura 4.16: Comparación del contenido espectral de las secuencias $x\,(n)$ y $y\,(n)$.

La atenuación que se obtuvo de la frecuencia de interés a suprimir, 121 Hz es aproximadamente de -40.07 db, lo cual se puede observar y corroborar con la gráfica de la respuesta al impulso del filtro supresor IIR que se muestra en la Figura 4.17.

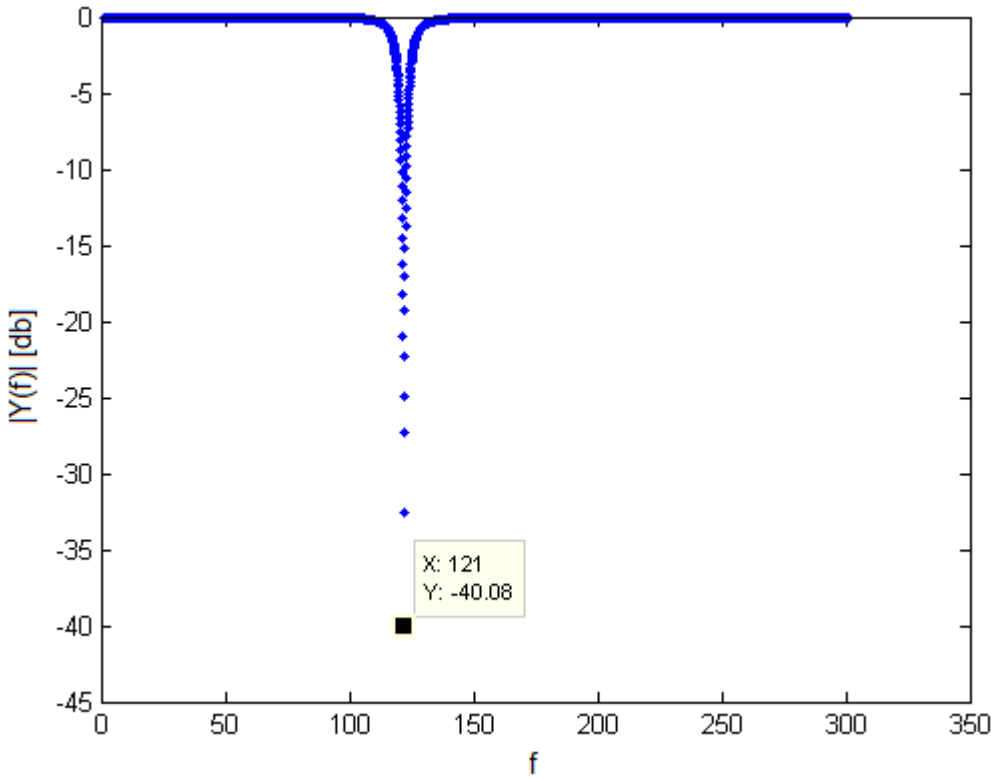


Figura 4.17: Respuesta al impulso del filtro supresor IIR de orden 2.

La banda de supresión del filtro IIR propuesto, tiene una atenuación mayor a los -3 db en ± 3 Hz respecto a la frecuencia de interés a suprimir. Para evaluar el desempeño de las implementaciones propuestas del filtro IIR, se obtuvo la respuesta al impulso de cada uno de los filtros programados respetando el formato de los datos de entrada y de salida, para apreciar los errores de precisión numérica que se tienen al manejar diferentes representaciones de números con punto decimal. Las secuencias resultantes se guardaron en archivos y posteriormente fuera de línea se analizaron con un programa de cálculo. Al convertir los formatos de punto entero a punto flotante, se obtuvo el espectro de magnitud de la respuesta al impulso de cada uno de los filtros y se calcularon los errores absolutos máximo, tomando como referencia la respuesta obtenida y mostrada en la Figura 4.17. Los errores absolutos de la respuesta al impulso de cada uno de los filtros programados se muestran en la Tabla 4.7.

La cuarta columna de la Tabla 4.7, registra la máxima atenuación obtenida por cada uno de los filtros en la frecuencia de interés. Se puede observar notoriamente que la precisión numérica afecto en cierto grado dicho factor de atenuación, sin embargo todos los factores son considerablemente altos, logrando prácticamente suprimir la componente espectral de 121 Hz. ***Las conclusiones*** de de los resultados obtenidos por las diferentes implementaciones

Tabla 4.7: Comparación de resultados obtenidos de los programas de implementación del filtro supresor IIR de segundo orden

Formato de entrada	**Formato de salida**	**Error absoluto máximo**	**Atenuación db**
Q14 L=16 bits	Q14 L=16 bits	0.0159	-38.3427
Q28 L=32 bits	Q28 L=32bits	6.8009e-05	-40.1132
Float	Float	1.9950e-05	-40.06

del filtro supresor IIR de segundo orden son:

- El factor de atenuación cambió en promedio 2 db, entre los formatos numéricos de 16 y 32 bits, manteniéndose en un margen menor de variación en estas últimas longitudes de datos.
- La banda de supresión del filtro es angosta, midiendo en promedio 6 Hz y la frecuencia de interés es prácticamente suprimida.
- La cantidad de localidades de memoria necesarias para realizar el cálculo del filtro, fue mucho menor por la cantidad de coeficientes del filtro de segundo orden.
- La precisión numérica es mayor en comparación con el filtro FIR, debido a la recursividad característica del filtro.

En términos generales, el desempeño obtenido por el filtro IIR superó el filtro FIR casi cuatro veces en materia del factor de atenuación logrado en las implementaciones propuestas, sin embargo, se vuelve a hacer notar que el filtro IIR cambia la fase de la secuencia de entrada, por lo que se deben de considerar las características que brinda cada uno de estas dos arquitecturas básicas de filtros digitales, antes de ponerlo en práctica en alguna aplicación.

4.5. Osciladores digitales

Los osciladores son sistemas con características inestables que son la base de otros sistemas más complicados como generadores sinusoidales, osciladores controlados por voltaje (VCO), moduladores, mallas de fase amarrada (PLL), etc [12].

En general, existen tres métodos para generar una señal sinusoidal, en este manual se documentó la implementación de un oscilador digital, el cual se describirá brevemente a continuación, considerando el método de diseño de un filtro paso banda de alta calidad, que

solo permite el paso de una componente de frecuencia, modelando como un sistema SLITD de segundo orden, el cual se puede escribir como

$$H(z) = \frac{b_0}{(1 - a_1 z^{-1} + a_2 z^{-2})} \tag{4.25}$$

donde $a_1 = -2\cos(\omega_0)$ y $a_2 = 1$, permaneciendo ambos coeficientes constantes. Estos valores se obtienen considerando que los polos de la función de transferencia (4.25) son conjugados $p_1 = {p_2}^*$ como se pueden observar en la Figura 4.18a. Si se aplican las fórmulas de tablas y propiedades de TZ se puede obtener la respuesta al impulso del sistema como

$$h(n) = \frac{b_0}{\text{sen}(\omega_0)}\text{sen}((n+1)\omega_0)U(n) \tag{4.26}$$

haciendo la constante $b_0 = \text{sen}(\omega_0)$ en la ecuación se obtiene la respuesta

$$h(n) = \text{sen}((n+1)\omega_0)U(n) \tag{4.27}$$

para que el sistema oscile y se tenga una salida $y(n)$ igual a la respuesta al impulso $h(n)$, la entrada debe ser un impulso $\delta(n)$ con ecuación en diferencias

$$y(n) = b_0 x(n) + a_1 y(n-1) - a_2 y(n-2) \tag{4.28}$$

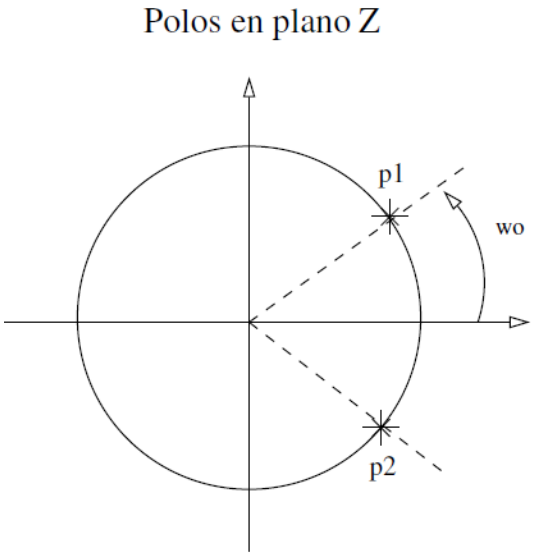
cuyo diagrama de bloques se muestra en la Figura 4.18b, con condiciones iniciales $y(-2) = 0$, $y(-1) = -A\text{sen}(\omega)$ y A la amplitud de la señal sinusoidal generada.

Por otro lado, un oscilador discreto también puede diseñarse considerando que su respuesta al impulso es una señal cosenoidal con la TZ para una entrada impulso $\delta(n)$. Para obtener la salida como una ecuación en diferencias se escribe $H(z)$ como el cociente de $Y(z)$ con $X(z)$, se calcula la transformada Z inversa y se despeja $y(n)$.

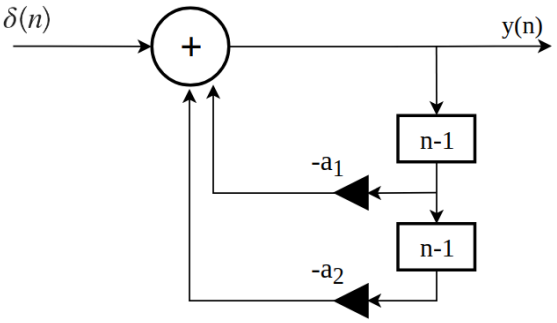
Implementación de un oscilador digital

Considerando el diagrama de bloques de la Figura 4.18b, a continuación se presentarán tres programas que generan una señal seno, considerando una frecuencia de interés de 45 Hz y una frecuencia de muestreo de 8 kHz. La metodología de la implementación propuesta, consiste en realizar el cálculo de los términos $y\,(0)$, $y\,(1)$ y $y\,(2)$ por separado y utilizar un ciclo de cálculo para todos los términos $3 \leq n \leq N$, acorde a la Ecuación (4.28) siendo $x\,(n)$ un impulso definido como

$$x\,(n) = \delta\,(n) = \begin{cases} 1 & n = 0 \\ \\ 0 & n \neq 0 \end{cases} \tag{4.29}$$



(a) Mapa de polos de un sistema SLITD oscilante de segundo orden.



(b) Diagrama de bloques de un oscilador digital de segundo orden.

Figura 4.18: Diagramas de un sistema SLITD de segundo orden.

4.5.1. Oscilador digital en punto fijo a 16 bits

Para este programa los coeficientes a_1, a_2 y b_0 se convirtieron en formato Q_{12} y fueron declarados directamente en el programa para que se graben junto con el código siempre y cuando el mapa de memoria declarado en el archivo *.cmd* lo permita, de lo contrario, antes de iniciar la operación del código se deberán de grabar dichas variables. Cabe recordar que la señal sinusoidal de 45 Hz es generada a partir de un señal impulso, lo cual permitió dividir el algoritmo en dos partes como se mencionó anteriormente y se podrá observar a continuación.

```
*
* Oscilador digital modelado por la ecuación en diferencias
* y(n)  = bo*x(n) + a1*y(n-1) - a2*y(n-2)
* donde x(n) es un impulso en el origen
* b0 = sin(w0)
* a1 = 2*cos(w0)
* a2 = -1
* w0 = 2*pi*f0/fs
* f0 = 45 Hz
* fs = 8 kHz
* Los datos fueron convertidos a formato de punto fijo Q12

        .global _c_int00
        ;.data
```

```
N       .set    1000
WDCR    .set    07029h
bzero   .word   145     ; Constante b0 en formato Q12
a1      .word   8187    ; Constante a1 en formato Q12
a2      .word   -4096   ; Constante a2 en formato Q12
yn1     .word   0       ; Variable y(n-1)
yn2     .word   0       ; Variable y(n-2)
ynaux   .word   0       ; Localidad de separación
yn      .space  N*16    ; Vector para guardar la secuencia
                        ; resultante
        .text
_c_int00
        EALLOW                  ; Habilita escritura a registros
                                ; protegidos
        MOVL    XAR1,#WDCR      ; Registro XAR1 apunta a dir. WDCR
        MOV     *XAR1,#0068h    ; Desactiva WatchDog
        EDIS                    ; Deshabilita escritura a registros
                                ; protegidos
        SETC    SXM             ; Modo extensión de signoo
        SPM     #0              ; Corrimientos nulos en P

        MOVW    DP,#yn1         ; Direcciona la página de memoria
                                ; de y(n-1)
        MOVL    XAR1,#yn        ; Direccionamiento a yn
        MOV     AL, @bzero      ; Mueve b0 a la parte baja
                                ; del acumulador
        MOV     @yn2,AL         ; Escribe b0 en la variable y(n-2)
        MOV     *XAR1++,AL      ; Escribe b0 en y(0)
        MOV     T,@yn2          ; Coloca el dato y(n-2) en T
        MPY     ACC,T,@a1       ; a1*y(n-2)=a1*b0
        LSL     ACC,#4          ; Ajuste de formato Q8 a Q12
        MOV     @yn1,AH         ; Escribe y(n-1) = a1*b0
        MOV     *XAR1++,AH      ; Escribe y(1)=a1*b0
        MOV     AR4,#N-1        ; Carga el total de iteraciones
OSC_CYCLE
        MOV     T,@a1           ; Escribe la constante a1 en T
        MPY     ACC,T,@yn1      ; ACC=a1*y(n-1)
        MOV     T,@a2           ; Escribe la constante a2 en T
        MPY     P,T,@yn2        ; P=a2*y(n-2)
        ADDL    ACC,P           ; Suma  a1*y(n-1)-a2*y(n-2)=ACC
        LSL     ACC,#4          ; Ajuste de formato Q8 a Q12
        MOV     *XAR1++,AH      ; Escribe el resultado en y(n-3)

        DMOV    @yn2            ; Desplaza el dato y(n-2) a ynaux
        DMOV    @yn1            ; Desplaza el dato y(n-1) a y(n-2)
        MOV     @yn1,AH         ; Escribe y(n) en y(n-1)

        BANZ OSC_CYCLE,AR4-- ; Ciclo de cálculo de cada término
                                ; del oscilador
```

```
iend    NOP
        LB      iend            ; Ciclo infinito
        .end
```

En este programa se utilizaron los modos de direccionamiento directo e indirecto para acceder a los datos y constantes participes del cálculo y se ejemplificó el uso de la instrucción **DMOV** con direccionamiento directo.

4.5.2. Oscilador digital en punto fijo a 32 bits

Para la implementación del sistema SLITD de segundo orden considerando datos de 32 bits, se decidió utilizar el formato de punto fijo Q_{28} para los coeficientes constantes de la Ecuación (4.28) y también para el resultado final.

```
*
* Oscilador digital modelado por la ecuación en diferencias
* y(n)  = bo*x(n) + a1*y(n-1) - a2*y(n-2)
* donde x(n) es un impulso en el origen
* b0 = sin(w0)
* a1 = 2*cos(w0)
* a2 = -1
* w0 = 2*pi*f0/fs
* f0 = 45 Hz
* fs = 8 kHz
* Los datos fueron convertidos a formato de punto fijo Q28

        .global         _c_int00
        ;.data
N       .set    1000
WDCR    .set    07029h
bzero   .long   9485317         ; Constante b0 en formato Q28
a1      .long   536535638       ; Constante a1 en formato Q28
a2      .long   -268435456      ; Constante a2 en formato Q28
yn1     .long   0               ; Variable y(n-1)
yn2     .long   0               ; Variable y(n-2)
ynaux   .long   0               ; Localidad de separación
yn      .space  N*32            ; Vector para guardar la secuencia
                                ; resultante
        .text
_c_int00
        EALLOW                  ; Habilita escritura a registros
                                ; protegidos
        MOVL    XAR1,#WDCR      ; Registro XAR1 apunta a dir. WDCR
        MOV     *XAR1,#0068h    ; Desactiva WatchDog
        EDIS                    ; Deshabilita escritura a registros
                                ; protegidos
```

```
        SETC    SXM             ; Modo extensión de signo
        SETC    OVM
        SPM     #0              ; Corrimientos nulos en P
        MOVW    DP,#yn1         ; Direcciona a página de memoria
                                : donde está y(n-1)
        MOVL    XAR1,#yn        ; Direccionamiento a yn
        MOVL    ACC,@bzero      ; Mueve b0 a la parte baja
                                ; del acumulador
        MOVL    @yn2,ACC        ; Escribe b0 en la variable y(n-2)
        MOVL    *XAR1++,ACC     ; Escribe b0 en y(0)
        MOVL    XT,@yn2         ; Coloca el dato y(n-2) en T
        QMPY    ACC,XT,@a1      ; a1*y(n-2)=a1*b0
        LSL     ACC,#4          ; Ajuste de formato Q24 a Q28
        MOVL    @yn1,ACC        ; Escribe y(n-1) = a1*b0
        MOVL    *XAR1++,ACC     ; Escribe y(1)=a1*b0
        MOV     AR4,#N-1        ; Carga el total de iteraciones

OSC_CYCLE
        MOVL    XT,@a1          ; Escribe a1 en el registro T
        QMPYL   ACC,XT,@yn1     ; Multiplica a1*y(n-1) y tiene
                                ; el resultado en el ACC
        MOVL    XT,@a2          ; Escribe a2 en T
        QMPYL   P,XT,@yn2       ; Multiplica a2*y(n-2) y tiene
                                ; el resultado en reg. P
        ADDL    ACC,P           ; ACC=a1*y(n-1)-a2*y(n-2)
        LSL     ACC,#4          ; Ajuste de formato Q8 a Q12
        MOVL    *XAR1,ACC       ; Escribe el resultado en y(n-3)

        MOVL    ACC,@yn2        ; Desplaza el dato y(n-2) a ynaux
        MOVL    @ynaux,ACC

        MOVL    ACC,@yn1        ; Desplaza el dato y(n-1) a y(n-2)
        MOVL    @yn2,ACC

        MOVL    ACC,*XAR1++     ; Escribe y(n) en y(n-1)
        MOVL    @yn1,ACC

        BANZ    OSC_CYCLE,AR4-- ; Ciclo de cálculo de cada
                                ; término del oscilador

iend    NOP
        LB      iend            ; Ciclo infinito
        .end
```

A diferencia del programa de implementación de 16 bits, en esta ocasión se sustituyó la forma de acomodar los datos $y(1)$ y $y(2)$ debido a que la instrucción **DMOV** funciona solo con 16 bits, con movimientos de datos a 32 bits mediante direccionamiento directo.

4.5.3. Oscilador digital en punto flotante IEEE 754

Aprovechando la unidad de punto flotante (FPU), se implementó la Ecuación (4.28) en el programa que se muestra a continuación, utilizando datos en formato de punto flotante de precisión simple, con el objetivo de ejemplificar el uso de algunas de sus instrucciones.

```
*
* Oscilador digital modelado por la ecuación en diferencias
* y(n)  = bo*x(n) + a1*y(n-1) - a2*y(n-2)
* donde x(n) es un impulso en el origen
* b0 = sin(w0)
* a1 = 2*cos(w0)
* a2 = -1
* w0 = 2*pi*f0/fs
* f0 = 45 Hz
* fs = 8 kHz

        .global     _c_int00
        ;.data
N       .set    1000
WDCR    .set    07029h
bzero   .float  0.03533556      ; Constante b0
a1      .float  1.998751008     ; Constante a1
a2      .float  -1              ; Constante a2
yn1     .float  0               ; Variable y(n-1)
yn2     .float  0               ; Variable y(n-2)
ynaux   .float  0               ; Localidad de separación
yn      .space  N*32            ; Vector para guardar la
                                ; secuencia resultante
        .text
_c_int00
        EALLOW                  ; Habilita escritura a registros
                                ; protegidos
        MOVL    XAR1,#WDCR      ; Registro XAR1 apunta a dir. WDCR
        MOV     *XAR1,#0068h    ; Desactiva WatchDog
        EDIS                    ; Deshabilita escritura a registros
                                ; protegidos
        SETC    SXM             ; Modo extensión de signo
        SETC    OVM
        SPM     #0              ; Corrimientos nulos en P
        MOV     DP,#yn1         ; Direcciona a página de memoria
                                ; donde está y(n-1)
        MOVL    XAR1,#yn        ; Direccionamiento a yn
        MOVL    ACC,@bzero      ; Mueve b0 a la parte baja
                                ; del acumulador
        MOVL    @yn2,ACC        ; Escribe b0 en y(n-2)
        MOVL    *XAR1++,ACC     ; Escribe b0 en y(0)
        MOV32   R1H,@yn2        ; Coloca y(n-2) en R1H
```

```
        MOV32   R2H,@a1
        MPYF32  R3H,R1H,R2H     ; a1*y(n-2)=a1*b0
        NOP
        MOV32   @yn1,R3H
        MOV32   *XAR1++,R3H     ; Escribe y(1)=a1*b0
        MOV     AR4,#N-1        ; Carga el total de iteraciones

OSC_CYCLE
        MOV32   R1H,@a1         ; Escribe a1 en R1H
        MOV32   R2H,@yn1
        MPYF32  R3H,R1H,R2H     ; ACC=a1*y(n-1)
        NOP
        MOV32   R1H,@a2         ; Escribe a2 en R1H
        MOV32   R2H,@yn2
        MPYF32  R4H,R1H,R2H     ; P=a2*y(n-2)
        NOP
        ADDF32  R5H,R3H,R4H     ; Suma  a1*y(n-1)-a2*y(n-2)=ACC
        NOP
        MOV32   *XAR1,R5H       ; Escribe resultado en y(n-3)
        MOV32   R1H,@yn1        ; Desplaza el dato y(n-1) a y(n-2)
        MOV32   @yn2,R1H
        MOV32   R1H,*XAR1++     ; Escribe y(n) en y(n-1)
        MOV32   @yn1,R1H

        BANZ OSC_CYCLE,AR4--    ; Ciclo de cálculo de cada término
                                ; del oscilador

iend    NOP
        LB      iend            ; Ciclo infinito
        .end
```

Del código anterior se resalta la necesidad de conocer la cantidad de ciclos de máquina que requieren ciertas instrucciones de punto flotante para poder obtener un resultado y verificar su desempeño, además de la posibilidad de utilizar direccionamiento directo utilizando los registros R0H-R7H de la FPU.

4.5.4. Oscilador digital utilizando Unidad Trigonométrica

El DSP TMS320F28377S además de la unidad de punto flotante, cuenta como parte de su arquitectura con una unidad matemática trigonométrica (TMU) la cual tiene la capacidad de generar funciones seno y coseno utilizando directamente una instrucción la cual necesita el argumento de la función normalizado y cuatro ciclos de máquina para obtener el resultado.

A continuación se presenta un programa que genera una secuencia de 1000 puntos de la función coseno en formato de punto flotante a 32 bits y en punto fijo a 16 bits con Q_0, una frecuencia de 45 Hz, frecuencia de muestreo de 8000 Hz y amplitud de 20 unidades para la secuencia en formato Q_0.

```
*
* Programa que genera dos señales seno discretas
* mediante la ecuación
* xc_flo(n) = sen(n*wo)
* xc_int(n) = 20*sen(n*wo)
* donde
* w0 = 2*pi*f1/fs
*
        .global         _c_int00
        .data
f1      .set    45.0    ; Frecuencia de interés para
                        ; la secuencia a generar en Hz
fs      .set    8000.0  ; Frecuencia de muestreo en Hz
N       .set    1000    ; Puntos de la secuencia a calcular
WDCR    .set    07029h  ; Dirección del reg. WatchDog Control
xc_flo  .space  N*32    ; Vector de 32 bits para guardar el
                        ; resultado en formato de pto. flotante
xc_int  .space  N*16    ; Vector de 16 bits para guardar el
                        ; resultado en formato de pto. fijo
        .text
_c_int00
        EALLOW                  ; Habilita escritura a registros
                                ; protegidos
        CLRC    XF
        SETC    SXM             ; Modo extensión de signo
        SETC    OBJMODE
        MOVL    XAR1,#WDCR      ; Reg. XAR1 apunta a dir. WDCR
        MOV     *XAR1,#0068h    ; Desactiva WatchDog
        EDIS                    ; Deshabilita escritura a registros
                                ; protegidos
        MOVL    XAR1,#xc_flo    ; Direccionamiento al primer
                                ; elemento del vector xc_flo
        MOVL    XAR2,#xc_int    ; Direccionamiento al primer
                                ; elemento del vector xc_int
        MOVF32  R2H,#f1         ; Escribe el valor de f1 en el
                                ; registro R2H
        MOVF32  R1H,#fs         ; Escribe el valor de fs en el
                                ; registro R1H
        NOP                     ; operación nula para que termine
                                ; la instrucción anterior
        DIVF32  R3H,R2H,R1H     ; Calculo de f1/fs

        MOVF32  R4H,#0.0        ; Escribe cero en el registro R4H,
```

```
                        ; que será la var. n

        RPTB    FIN_B,#N-1      ; Repeticiones del bloque de
                                ; codigo FIN_B
                NOP
                MPYF32 R5H,R3H,R4H    ; Cálculo de arg = n*f1/fs
                NOP
                NOP
                SINPUF32 R6H,R5H      ; Cálculo de R6H
                NOP                   ; cos(2*pi*(n*f1/fs))
                NOP
                NOP
                MOV32   *XAR1++, R6H  ; Escribe el resultado
                                      ; en xc_flo(n)
                ADDF32 R4H,R4H,#1.00  ; Incrementa una unidad la
                                      ; variable n
                NOP
                MPYF32 R2H,R6H,#20.0  ; Multiplica 20*cos(n*w0)
                NOP
                F32TOI16R R1H,R2H     ; Conversión del resultado de
                                      ; flotante de 32 bits a
                                      ; entero de 16 bits
                NOP
                MOV32   ACC,R1H       ; Escribe int16(20*cos(n*w0))
                NOP                   ; en ACC
                MOV     *XAR2++,AL
FIN_B
FIN_R   NOP
        LB      FIN_R                 ; Ciclo infinito
        .end
```

Análisis de resultados de los osciladores digitales de segundo orden

En la Figura 4.19 se muestra la gráfica de la secuencia obtenida al calcular 1000 puntos de la Ecuación (4.28) por un programa de cálculo matemático, manejando el formato de punto flotante de precisión doble.

Para analizar los resultados obtenidos por los programas implementados, se calculó el error absoluto de cada secuencia obtenida, considerando como referencia la secuencia calculada por el software Octave que utiliza un formato de punto flotante de doble precisión. En la Tabla 4.8 se observan los errores de precisión numérica obtenidos por los códigos propuestos.

La columna "Formato de entrada", indica el tipo de formato en el que se utilizaron las constantes a_1, a_2 y b_0 en la operación. La segunda columna "Formato de salida" señala el

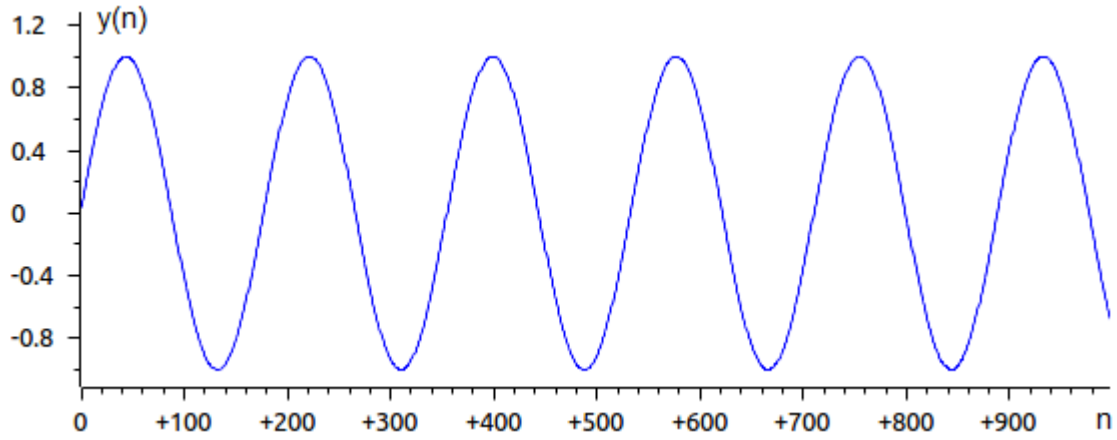


Figura 4.19: Gráfica de la secuencia resultante de evaluar la Ecuación (4.28) en formato de punto flotante de doble precisión.

Tabla 4.8: Errores absolutos obtenidos de la secuencias calculadas por los programas desarrollados del oscilador digital.

Formato de entrada	**Formato de salida**	**Error Absoluto**		
		Min.	**Prom.**	**Max.**
Q12	Q12	6.4353e-05	0.1041	0.2653
Q28	Q28	4.7969e-10	2.6217e-05	5.7747e-05
Float	Float	4.7762e-10	1.5243e-04	4.5609e-04
Float	Float-TMU	6.9378e-05	0.0227	0.0353

formato al que se ajusto cada dato calculado. En la tercer columna se observan los errores absolutos obtenidos, donde el mínimo error absoluto se obtuvo al manejar datos y aritmética de punto flotante, analizando los errores absolutos promedio y máximo.

Sin embargo ***se puede concluir*** que el mejor rendimiento en error de precisión numérica se obtiene al manejar formatos de punto entero a 32 bits, sin embargo, el mejor desempeño de las implementaciones estará en función de su aplicación. La última fila de la Tabla 4.8 corresponde a la comparación de errores de la secuencia generada por la unidad TMU, utilizando los valores definidos en tablas de CCS para este dispositivo.

4.5.5. Codificación o modulación de doble tono DTMF

La codificación o modulación por doble tono DTMF (*Dual tone modulation frequency*) es utilizada ampliamente en los sistemas telefónicos, cuando un usuario oprime una secuencia de números para comunicarse, por cada tecla oprimida se genera una señal de doble tono durante un tiempo determinado, es decir, se utilizan dos osciladores. Las señales DTMF generadas viajan por la red telefónica y a través de la decodificación en las subestaciones permite direccionar la llamada hasta el usuario final [12].

Un generador de doble tono se puede generar con dos osciladores sinusoidales en sus respectivas frecuencias estandarizadas como se muestra en la Figura 4.20, donde a cada dígito del teclado telefónico se le asocian dos frecuencias, de renglón y columna.

		Tonos columna [Hz]			
		1209	1336	1477	1633
Tonos renglón [Hz]	697	1	2	3	A
	770	4	5	6	B
	852	7	8	9	C
	941	*	0	#	D

Figura 4.20: Configuración del teclado matricial alfa numérico de modulación de doble tono DTMF.

Partiendo del análisis en el tiempo discreto, si se suman dos señales sinusoidales causales como

$$y(n) = \text{sen}(\omega_1 n) + \text{sen}(\omega_2 n) \tag{4.30}$$

donde las frecuencias de oscilación discretas bajas y altas corresponden a ω_1 y ω_2, y $\omega_i = 2\pi fosc/Fs$, $fosc$ es la frecuencia de oscilación analógica y Fs la frecuencia de muestreo. Si $y(n)$ es igual a la respuesta al impulso $h(n)$ del sistema, y aplicando la transformada Z donde $H(z) = Y(z)/X(z)$ tenemos

$$H(z) = \frac{Y(z)}{X(z)} = \frac{\text{sen}(\omega_1)z^{-1}}{1 - 2\cos(\omega_1)z^{-1} + z^{-2}} + \frac{\text{sen}(\omega_2)z^{-1}}{1 - 2\cos(\omega_2)z^{-1} + z^{-2}} \tag{4.31}$$

después de un manejo algebraico se obtiene (4.32)

$$H(z) = \frac{Y(z)}{X(z)} = \frac{Az^{-1} - Bz^{-2} + Az^{-3}}{1 - Cz^{-1} + Dz^{-2} - Cz^{-3} + z^{-4}} \tag{4.32}$$

donde:
$A = \text{sen}(\omega_1) + \text{sen}(\omega_2)$
$B = 2[\cos(\omega_1)\text{sen}(\omega_2) + \cos(\omega_2)\text{sen}(\omega_1)]$
$C = 2[\cos(\omega_1) + \cos(\omega_2)]$
$D = 2 + 4[\cos(\omega_1)\cos(\omega_2)]$

Aplicando la transformada Z inversa a (4.32), se obtiene una ecuación en diferencias para un sistema IIR de cuarto orden:

$$\begin{aligned} y\,(n) =& Ax\,(n-1) - Bx\,(n-2) + Ax\,(n-3) + Cy\,(n-1) \\ & - Dy\,(n-2) + Cy\,(n-3) - y\,(n-4) \end{aligned} \tag{4.33}$$

considerando la entrada $x(n) = \delta(n)$ y como condiciones iniciales $y(-4) = y(-3) = y(-2) = y(-1) = 0$, entonces se tienen los términos de $y\,(n)$ para $0 \leq n \leq 4$ salidas:

$y(0) = 0$; $\quad y(1) = A$; $\quad y(2) = -B + Cy(1)$
$y(3) = A + Cy(2) - Dy(1)$; $\quad y(4) = Cy(3) - Dy(2) + Cy(1) - y(0)$

es decir, que a partir de $n = 4$ se puede calcular la salida $y(n)$ en forma recursiva como

$$\begin{aligned} y(n) =& Cy(n-1) - Dy(n-2) + Cy(n-3) - y(n-4); \\ & \text{para} \quad n \geq 4 \end{aligned} \tag{4.34}$$

El diagrama de bloques de la implementación de una señal DTMF se muestra en la Figura 4.21, después se presenta la implementación de la generación la secuencia producida al presionar la tecla número cinco.

Implementación de un modulador de doble tono

Para implementar (4.34), los programas que se presentan a continuación se plantearon para que inicien con los valores calculados fuera de línea de $y(n)$ evaluada en el intervalo $[0, 3]$ y calculen mediante un ciclo los términos para $4 \leq n \leq N$, donde N corresponde a la cantidad de puntos que se desean calcular. Para este ejemplo, se propone trabajar con los tonos $f_1 = 770$ Hz y $f_2 = 1336$ Hz los cuáles corresponden a los tonos generados al presionar la tecla cinco, y una frecuencia de muestreo $f_s = 8$ kHz.

Para analizar y evaluar el desempeño de los resultados de los programas propuestos, se calculó la secuencia generada por (4.33) con las condiciones iniciales especificadas en un

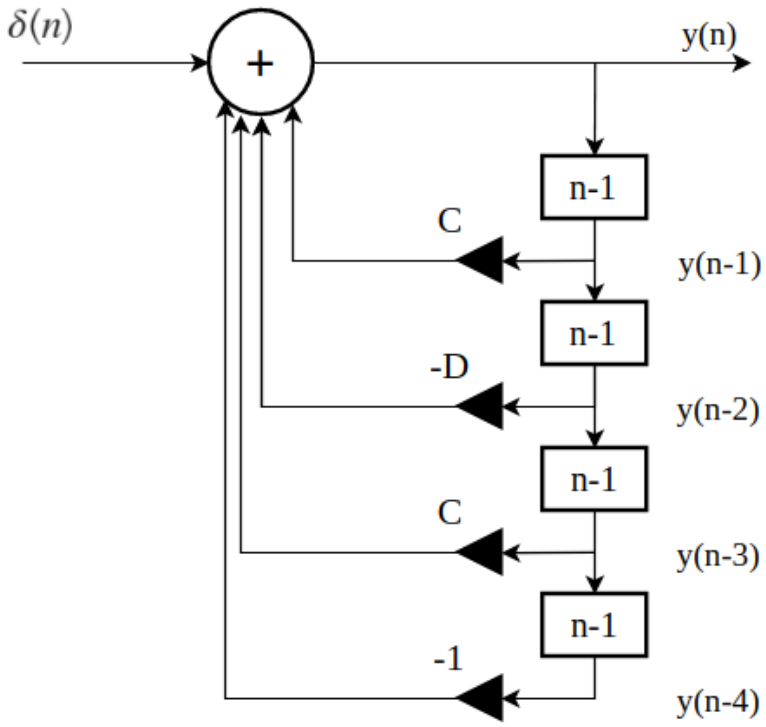


Figura 4.21: Diagrama de bloques del sistema SLITD de cuarto orden para generar la modulación de doble tono a partir de $n \geq 4$.

programa de cálculo utilizando formato flotante de doble precisión. La secuencia resultante se observa en las Figuras 4.22 y 4.23.

Modulador doble tono a 32 bits en punto fijo

Para su implementación en formato de punto fijo, es necesario tener presente la dinámica de las variables, de los coeficientes de (4.33) se observa que los máximos valores son $A \leq 2$, $B \leq 4$, $A \leq 4$ y $A \leq 6$, esto implica que la representación de los coeficientes debe manejarse considerando como mínimo 12 bits para la parte fraccionaria, $Q_i = 12$, formato que es adecuado para trabajar con datos cuya longitud es de 16 bits, sin embargo, el programa que se presenta a continuación trabaja con longitudes de datos de 32 bits, por lo que manejará datos en formato $Q_i = 28$.

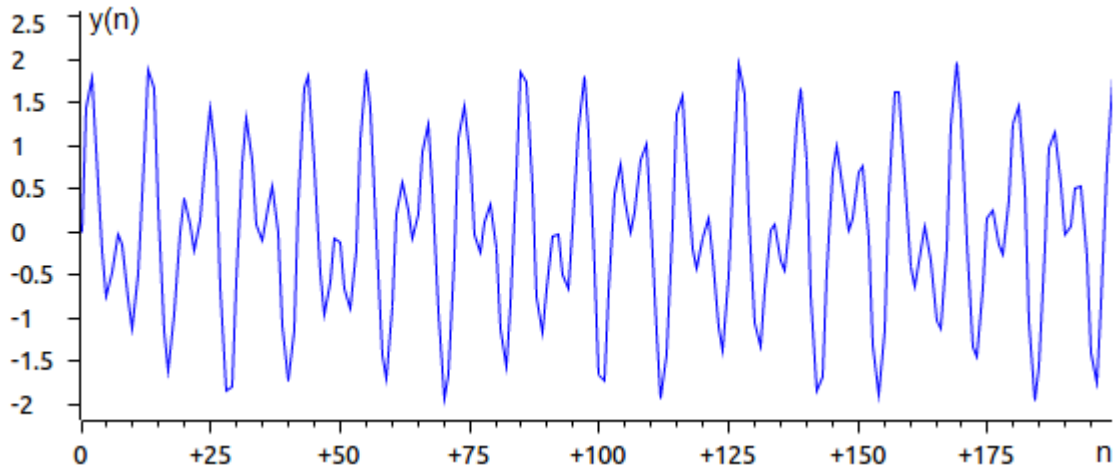


Figura 4.22: Gráfica de 200 puntos de la secuencia resultante de evaluar la Ecuación (4.33) en formato de punto flotante de doble precisión.

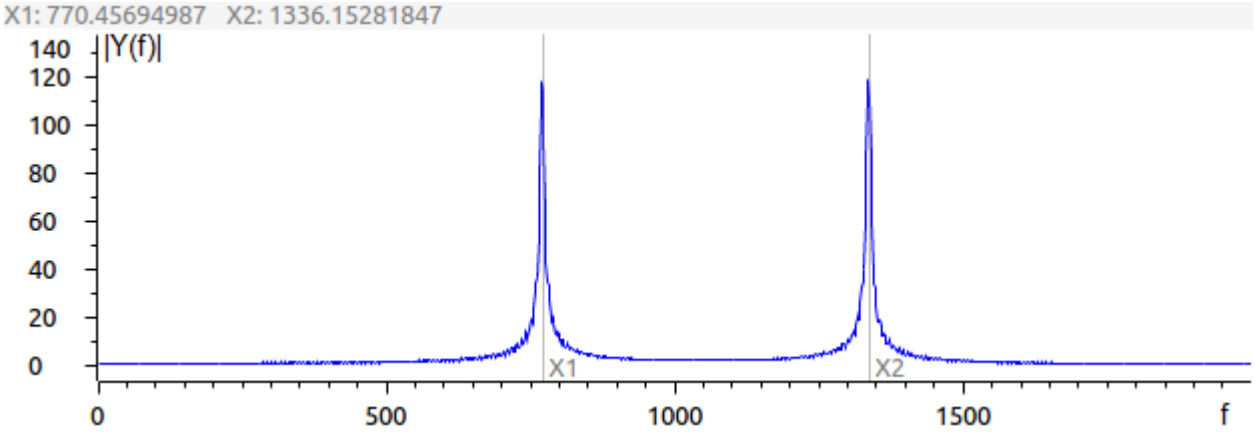


Figura 4.23: Espectro de magnitud de la secuencia generada al oprimir la tecla 5 del sistema DTMF.

```
* Programa generador de una señal de doble tono considerando
* datos de 32 bits en formato de punto fijo en Q28
*
* El programa inicia con valores iniciales de 0<=n<=3 de
* y(n) acorde a la ecuación
* y(n) = A*x(n-1)-B*x(n-2)+A*x(n-3)+C*y(n-1)-D*y(n-2)+C*y(n-3)-E*y(n-4)
*
* posteriormente, utiliza la simplificación para 4<=n<=N
* y(n) = C*y(n-1) - D*y(n-2) + C*y(n-3) - E*y(n-4)
* Las constantes de las ecuaciones son
* A = sen(w1)+sen(w2)
* B = 2*[cos(w1)*sen(w2)+cos(w2)*sen(w1)]
```

```
* C = 2*[cos(w1)+cos(w2)]
* D = 2+4*cos(w1)*cos(w2)
* E = -1
* Condiciones iniciales
* y(0) = 0
* y(1) = A
* y(2) = -B + C*y(1)
* y(3) = A + C*y(2) - D*y(1)
* Las frecuencias de los tonos consideradas en este programa son
* f1 = 770 Hz y f2 = 1336 Hz
* y una frecuencia de muestreo
* fs = 8 kHz
* teniendo entonces
* w1 = 2*pi*f1/fs y w2 = 2*pi*f2/fs
*
* Se calcularon las constantes y condiciones iniciales, y se
* convirtieron en formato de punto fijo Q28

        .global     _c_int00
WDCR    .set    07029h          ; Dirección de reg. de control
                                ; WatchDog
N       .set    1000            ; Cantidad de puntos a generar
Nprod   .set    4               ; Cantidad de productos a realizar
                                ; en la MAC
* Declaración de constantes de la ecuación en
* diferencias para n>=4
CD1     .long   709112857       ; 1ra cte. C de la ec. en dif
D       .long   -976919574      ; Constante D de la ec. en dif.
                                ; multiplicada por -1
C       .long   709112857       ; 2da cte. C de la ec. en dif.
E       .long   -268435456      ; Constante E de la ec. en dif
* Declaración de variables de retardo para la secuencia y(n)
yn0     .long   0               ; y(0) = 0 = y(n-4)
yn1     .long   258831729       ; y(1) = A = y(n-3)
yn2     .long   483014021       ; y(2) = -B+C*y(1) = y(n-2)
yn3     .long   385374679       ; y(3) = A+C*y(2)-D*y(1) = y(n-1)
yn4     .long   0               ; Inicia localidad para retardo
yn      .space  N*32            ; Reserva N localidades para
                                ; guardar y(n)

        .text
_c_int00
* Deshabilitación del WatchDog
        EALLOW                  ; Habilita escritura a registros
                                ; protegidos
        MOVL    XAR1, #WDCR     ; Registro XAR1 apunta a dir. WDCR
        MOV     *XAR1,#0068h    ; Desactiva WatchDog
        EDIS                    ; Deshabilita escritura a registros
                                ; protegidos
```

```
        SETC    SXM             ; Modo extensión de signo
        SETC    OVM
        SPM     #0              ; Corrimientos nulos en P
        MOVW    DP,#yn0         ; Direccionamiento del apuntador de
                                ; página donde se encuentra yn0
        MOVL    XAR2,#yn        ; Direccionamiento a y
        MOVL    ACC,@yn4        ; ACC=y(n-4)=0
        MOVL    *XAR2++,ACC     ; Escribe y(0) = y(n-4)
        MOVL    ACC,@yn3        ; Escribe y(n-3) en ACC
        MOVL    *XAR2++,ACC     ; Escribe y(1) = y(n-3)
        MOVL    ACC,@yn2        ; Escribe y(n-2) en ACC
        MOVL    *XAR2++,ACC     ; Escribe y(2) = y(n-2)
        MOVL    ACC,@yn1        ; ACC=y(n-1)
        MOVL    *XAR2++,ACC     ; Escribe y(3) = y(n-1)
        MOV     AR4,#N-1        ; Carga el total de iteraciones para
                                ; repetir el bloque de código
                                ; CALC_OSC que cálcula los términos
                                ; para n>=4
CALC_OSC
        MOVL    XAR6,#yn1       ; Direccionamiento a la variable yn1
        MOVL    XAR7,#CD1       ; Inicio de constantes ec. dif.
        ZAPA                    ; ACC=0 y P=0

        RPT #Nprod-1            ; Ciclo de cálculo de la ecuación
                                ; en diferencias
        ||      QMACL P,*XAR6++,*XAR7++     ; MAC a 32 bits
        ADDL    ACC,P           ; Acumulación del último producto
        LSL     ACC,#4          ; Ajuste de formato Qi

        MOVL    @yn0,ACC        ; Escribe y(N) en variable yn
        MOVL    *XAR2++,ACC     ; y(AR4) = ACC
* Desplazamiento de los datos retardados de y
        MOVL    ACC,@yn3        ; y(n-4) = y(n-3)
        MOVL    @yn4,ACC

        MOVL    ACC,@yn2        ; y(n-3) = y(n-2)
        MOVL    @yn3,ACC

        MOVL    ACC,@yn1        ; y(n-2) = y(n-1)
        MOVL    @yn2,ACC

        MOVL    ACC,@yn0        ; y(n-1) = y(n)
        MOVL    @yn1,ACC

        BANZ    CALC_OSC,AR4--

iend    NOP
        LB      iend            ; Ciclo infinito
        .end
```

Modulador doble tono a 32 bits en punto flotante

Para comprobar que el algoritmo propuesto para la implementación presentada, de un generador de señales DTMF funcione adecuadamente, se realizaron los cambios correspondientes al programa anterior, para poder trabajar con datos en formato de punto flotante, haciendo uso de la FPU. Por lo que las constantes calculadas fuera de línea y los primeros 4 términos de la secuencia $y(n)$ se declararon al inicio del programa acorde a los resultados obtenidos en la simulación realizada previamente en un programa de cálculo.

```
*
* Programa generador de una señal de doble tono considerando
* datos de 32 bits en formato de punto flotante
*
* El programa inicia con valores iniciales de 0<=n<=3 de
* y(n) acorde a la ecuación 1
*
* y(n) = A*x(n-1)-B*x(n-2)+A*x(n-3)+C*y(n-1)-D*y(n-2)
*        +C*y(n-3)-E*y(n-4)
*
* y posteriormente, utiliza la simplificación para 4<=n<=N,
* la cual es
* y(n) = C*y(n-1) - D*y(n-2) + C*y(n-3) - E*y(n-4)
*
* Las constantes de las ecuaciones son
* A = sen(w1)+sen(w2)
* B = 2*[cos(w1)*sen(w2)+cos(w2)*sen(w1)]
* C = 2*[cos(w1)+cos(w2)]
* D = 2+4*cos(w1)*cos(w2)
* E = -1
* Condiciones iniciales
* y(0) = 0
* y(1) = A
* y(2) = -B + C*y(1)
* y(3) = A + C*y(2) - D*y(1)
*
* Las frecuencias de los tonos consideradas en este programa
* son   f1 = 770 Hz          y      f2 = 1336 Hz
* y una frecuencia de muestreo
*       fs = 8 kHz
* teniendo entonces
*       w1 = 2*pi*f1/fs             y      w2 = 2*pi*f2/fs
*
* Se calcularon las constantes y condiciones iniciales

        .global _c_int00
WDCR    .set    07029h          ; Dirección del reg. de
                                ; control WatchDog
N       .set    1000            ; Cantidad de puntos a generar
```

```
Nprod  .set   5              ; Cantidad de productos a realizar
* Declaración de constantes de la ecuación en diferencias
* para n>=4
CD1    .float 2.6416512467   ; 1ra cte. C de la ec. en dif
D      .float -3.6393090125  ; Cte. D de la ec. en dif.
                             ; multiplicada por -1
C      .float 2.6416512467   ; 2da. cte. C de la ec. en dif.
E      .float -1.0,0         ; Constante E de la ec. en dif
*Declaración de variables de retardo para la secuencia y(n)
yn0    .float 0              ; y(0)=0=y(n-4)
yn1    .float 0.9642233295   ; y(1)=A=y(n-3)
yn2    .float 1.799367448    ; y(2)=-B+C*y(1)=y(n-2)
yn3    .float 1.4356325519   ; y(3)=A+C*y(2)-D*y(1)=y(n-1)
yn4    .float 0.0            ; Inicio de la localidad
                             ; para retardo
yn5    .float 0.0
yn     .space N*32           ; Reserva N localidades para
                             ; guardar y(n)

       .text
_c_int00
* Deshabilitación del WatchDog
       EALLOW                ; Habilita escritura a registros
                             ; protegidos
       MOVL   XAR1, #WDCR    ; Registro XAR1 apunta a dir. WDCR
       MOV    *XAR1,#0068h   ; Desactiva WatchDog
       EDIS                  ; Deshabilita escritura a registros
                             ; protegidos

       SETC   SXM            ; Modo extensión de signo
       SETC   OVM

       MOVW   DP,#yn0        ; Direccionamiento del apuntador de
                             ; página donde se encuentra yn0
       MOVL   XAR2,#yn       ; Direccionamiento a y
       MOVL   ACC,@yn4       ; Escribe en ACC=y(n-4)=0
       MOVL   *XAR2++,ACC    ; Escribe y(0) = y(n-4)
       MOVL   ACC,@yn3       ; Escribe y(n-3) en ACC
       MOVL   *XAR2++,ACC    ; Escribe y(1) = y(n-3)
       MOVL   ACC,@yn2       ; Escribe y(n-2) en ACC
       MOVL   *XAR2++,ACC    ; Escribe y(2) = y(n-2)
       MOVL   ACC,@yn1       ; Escribe y(n-1) en el ACC
       MOVL   *XAR2++,ACC    ; Escribe y(3) = y(n-1)
       MOV    AR4,#N-1       ; Carga el total de iteraciones para
                             ; repetir el bloque de código
                             ; CALC_OSC que calcula los términos
                             ; para n>=4
```

```
CALC_OSC
        MOVL    XAR3,#yn1       ; Direccionamiento a la variable yn1
        MOVL    XAR7,#CD1       ; Inicio de constantes ec. dif.
        ZAPA                    ; Escribe ceros en ACC y P

        ZERO    R1H
        ZERO    R2H
        ZERO    R3H
        ZERO    R7H

        MOV     AR5,#Nprod-1

        RPTB    mac_cycle,AR5
                MOV32   R3H,*XAR3++
                MOV32   R7H,*XAR7++
                MPYF32  R2H,R3H,R7H
                NOP
                ADDF32  R1H,R2H,R1H
                NOP
mac_cycle:
        MOV32   *XAR2++,R1H     ; Guarda el resultado en y(AR4)
        NOP
        MOV32   ACC,R1H         ; ACC=y(AR4)
        MOVL    @yn0,ACC        ; Escribe y(N) en variable yn0
* Desplazamiento de los datos retardados de y
        MOVL    ACC,@yn3        ; y(n-4) = y(n-3)
        MOVL    @yn4,ACC

        MOVL    ACC,@yn2        ; y(n-3) = y(n-2)
        MOVL    @yn3,ACC

        MOVL    ACC,@yn1        ; y(n-2) = y(n-1)
        MOVL    @yn2,ACC

        MOVL    ACC,@yn0        ; y(n-1) = y(n)
        MOVL    @yn1,ACC

        BANZ    CALC_OSC,AR4--

iend    NOP
        LB      iend            ; Ciclo infinito
        .end
```

Análisis de resultados de los osciladores DTMF

Para evaluar el desempeño de los programas propuestos que generan una secuencia DTMF, en particular de la tecla 5, se calcularon los errores absolutos de las secuencias calculadas por cada uno de los programas de implementación, considerando como referencia

la secuencia calculada en formato flotante de doble precisión. Los errores obtenidos se muestran en la Tabla 4.9, éstos no consideran el primer elemento de la secuencia que es igual a cero, con la finalidad de analizar las variaciones debido al uso de dos diferentes formatos.

Tabla 4.9: Error absoluto de las secuencias DTMF generadas por los programas de implementación, comparado con la secuencia ideal calculada en formato flotante de doble precisión.

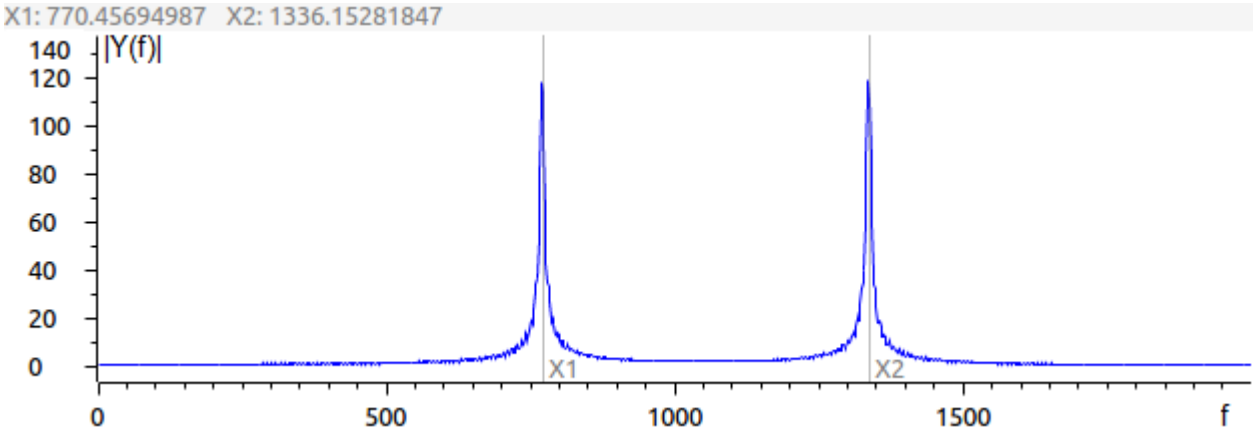
Formato de entrada en punto fijo	**Formato de salida en punto fijo**	**Error Absoluto**		
		Min.	**Prom.**	**Max.**
Q28 L=32 bits	Q28 L=32 bits	9.1553e-11	1.1771e-06	6.1232e-06
Float	Float	0	5.1252e-05	2.3829e-04

Se puede observar en la Tabla 4.9 que el error absoluto mínimo se obtiene con el programa que opera datos de 32 bits en formato de punto fijo. Sin embargo, considerando en términos prácticos, el error absoluto máximo menor, se obtiene con el programa que maneja datos en formato de punto flotante. Adicionalmente, se compararon los espectros obtenidos, con la finalidad de verificar que las componentes del espectro de mayor magnitud correspondieran a las frecuencias de interés correspondientes a la tecla número 5. En la Tabla 4.10 se observa la magnitud de las componentes de interés del espectro en magnitud de las señales generadas por las dos implementaciones, y en la Figura 4.24 se observan los tres espectros en magnitud.

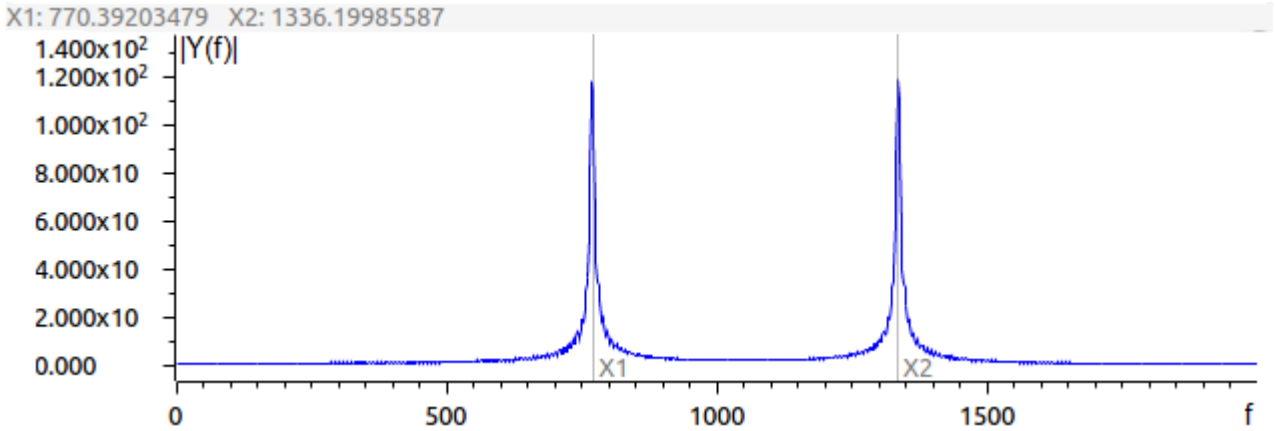
Tabla 4.10: Comparación de magnitudes de las componentes de frecuencia de la tecla 5 de la codificación DTMF.

Frecuencia Hz	**Magnitud de la componente espectral**		
	Punto fijo Q28 L=32 bits	**Punto flotante**	**Punto flotante doble (Octave)**
770	449.8052	449.7826	449.8045
1336	498.6659	498.6843	498.6657

A partir de los resultados registrados por los dos programas que generan una secuencia DTMF, de la ecuación en diferencias de cuarto orden, podemos ***concluir***:



(a) Espectro de magnitud de la secuencia generada al oprimir la tecla 5 del sistema DTMF, resultado del programa de punto fijo.



(b) Espectro de magnitud de la secuencia generada al oprimir la tecla 5 del sistema DTMF, resultado del programa de punto flotante.

Figura 4.24: Comparación de espectros de magnitud de las secuencias DTMF de la tecla 5 generadas por los programas propuestos.

- Los dos programas que generan la secuencia DTMF funcionan adecuadamente, siendo más precisa la implementación que maneja datos y aritmética en punto fijo.

- Las magnitudes de las componentes espectrales generadas, no presentaron cambios significativos debido a la precisión numérica de los dos formatos utilizados.

- Al tener errores absolutos muy bajos, cambiando las constantes correspondientes, los programas pueden generar cualquier señal de la codificación DTMF.

4.6. Transformada discreta de Fourier DFT

La ***Transformada Discreta de Fourier*** (DFT acorde a sus siglas en inglés) tiene un papel muy importante en la realización de una gran variedad de algoritmos del tratamiento digital de señales y es una herramienta computacional muy poderosa para el análisis en frecuencia de señales discretas.
Para una secuencia de duración finita $x(n)$ de longitud L, su transformada de Fourier en el tiempo discreto (DFT) es la descrita en (4.35)

$$X\left(e^{j\omega}\right) = \sum_{n=0}^{L-1} x\left(n\right) e^{-j\omega n} \tag{4.35}$$

donde ω va de 0 a 2π.

Debido a que $X(\omega)$ tiene periodo igual a 2π, solo las muestras dentro del rango de la frecuencia fundamental son necesarias. Cuando se muestrea $X(e^{j\omega})$ a frecuencias igualmente espaciadas $w_k = 2\pi k/N$ donde $k = 0, 1, 2, 3, \ldots, N-1$ y N es mayor o igual a L, las muestras resultantes obtenidas se describen como:

$$X\left(k\right) = X\left(2\pi k/N\right) = \sum_{n=0}^{L-1} x\left(n\right) e^{-j2\pi kn/N} \tag{4.36}$$

$$X\left(k\right) = \sum_{n=0}^{N-1} x\left(n\right) e^{-j2\pi kn/N} \tag{4.37}$$

La Ecuación (4.37) es la expresión matemática de la transformada de Fourier discreta o DFT en inglés (Discrete Fourier Transform) donde por conveniencia se ha cambiado el índice $L-1$ por $N-1$ [19].

Para recuperar la secuencia $x(n)$ a partir de las muestras en frecuencia se tiene (4.38), la cual, es la **Transformada Discreta Inversa de Fourier** o en inglés IDFT (inverse DFT).

$$X\left(n\right) = \frac{1}{N} \sum_{k=0}^{N-1} X\left(k\right) e^{j2\pi kn/N} \tag{4.38}$$

Las DFT y su inversa, pueden ser expresadas como se muestra en (4.39) y (4.40) respectivamente, si se define el término $W_N = e^{-j2\pi/N}$ y $W_N^{kn} = e^{-j2\pi kn/N}$.

$$X\left(k\right) = \sum_{n=0}^{N-1} x\left(n\right) W_N^{kn} \tag{4.39}$$

$$x\left(n\right) = \frac{1}{N} \sum_{k=0}^{N-1} X\left(k\right) W_N^{-kn} \tag{4.40}$$

donde $k = 0, 1, 2, 3, \ldots, N-1$.

Es de notar que para el cálculo de cada punto de la DFT se efectúan N multiplicaciones complejas y N-1 sumas complejas. Por consiguiente para el cálculo de N puntos de la DFT se efectúan N^2 multiplicaciones complejas y $N(N-1)$ sumas complejas [19].

4.6.1. Propiedades de la DFT

La transformada discreta de Fourier cumple con las siguientes propiedades [14]:

Linealidad

Dadas dos secuencias en el tiempo discreto $x_1(n)$ y $x_2(n)$ con su respectivas secuencias en el dominio de la frecuencia discreta $X_1(k)$ y $X_1(k)$ y a_1,a_2 como constantes, la transformada es lineal si se cumple:

- Superposición

$$DFT\{a_1 x_1(n) + a_2 x_2(n)\} = a_1 X_1(k) + a_1 X_1(k) \tag{4.41}$$

- Homogeneidad

$$DFT\{ax(n)\} = aX(k) \tag{4.42}$$

- Aditividad

$$DFT\{x_1(n) + x_2(n)\} = X_1(k) + X_2(k) \tag{4.43}$$

Inversión en el tiempo

$$DFT\{x(-n)\} = DFT\{x(N-n)\} \Rightarrow X(k) = X(N-k) \tag{4.44}$$

Desplazamiento circular en el tiempo

- Desplazamiento circular en el tiempo

$$DFT\{x(n-l)\} = X(k)\, e^{-j2\pi kl/N} \tag{4.45}$$

- Desplazamiento circular en la frecuencia

$$DFT\{x(nT)\, e^{j2\pi nl/N}\} = X(k-l) \tag{4.46}$$

Corrimiento circular en frecuencia de la DFT

$$X\left(k+m\right)=DFT\{x\left(n\right)W_N^{mn}\} \tag{4.47}$$

$$IDFT\{X\left(k+m\right)\}=x\left(n\right)W_N^{mn} \tag{4.48}$$

Simetría

Sea $x\left(n\right)$ una serie de tiempo discreto de longitud N y su respectiva representación en el dominio de la frecuencia discreta $X\left(k\right)$, $X\left(k\right)$ es simétrica respecto de $\frac{N}{2}$, esto es:

$$X\left(k\right)=X^*\left(-k\right) \tag{4.49}$$

Convolución Circular

La convolución circular ($\otimes$) entre dos secuencias de tiempo discreto, corresponde a la multiplicación de las mismas en el dominio de la frecuencia discreta, además, la convolución circular de dos secuencias en el dominio de la frecuencia corresponde a la multiplicación de dichas secuencias en el dominio del tiempo, esto se puede observar en (4.50) y (4.51) respectivamente.

$$DFT\{x_1\left(n\right)\otimes x_2\left(n\right)\}=X_1\left(k\right)X_2\left(k\right) \tag{4.50}$$

$$\frac{1}{N}X_1\otimes\left(k\right)X_2\left(k\right)=IDFT\{x_1\left(n\right)x_2\left(n\right)\} \tag{4.51}$$

Correlación Circular

$$DFT\{r_{x_1x_2}\}\rightarrow R_{x_1x_2}=X_1\left(k\right)X_2^*\left(k\right) \tag{4.52}$$

Teorema de Parseval

Para dos secuencias discretas $x_1\left(n\right)$ y $x_2\left(n\right)$ se cumple que:

$$\sum_{n=0}^{N-1}x_1\left(n\right)x_2^*\left(n\right)=\frac{1}{N}\sum_{k=0}^{N-1}X_1\left(k\right)X_2^*\left(k\right) \tag{4.53}$$

Además, si $x\left(n\right)=x_1\left(n\right)=x_2\left(n\right)$, entonces (4.53) representa la energía de la señal como se muestra a continuación:

$$\sum_{n=0}^{N-1}\left|x\left(n\right)\right|^2=\frac{1}{N}\sum_{k=0}^{N-1}\left|X\left(k\right)\right|^2 \tag{4.54}$$

4.6.2. Implementación de DFT en punto flotante

La implementación de la Trasformada discreta de Fourier básicamente consiste en realizar la sumatoria definida en (4.37), donde se puede observar que consta de dos ciclos anidados con las variables de interés "n" y "k". En la Figura 4.25 se muestra el diagrama de flujo del algoritmo de la DFT.

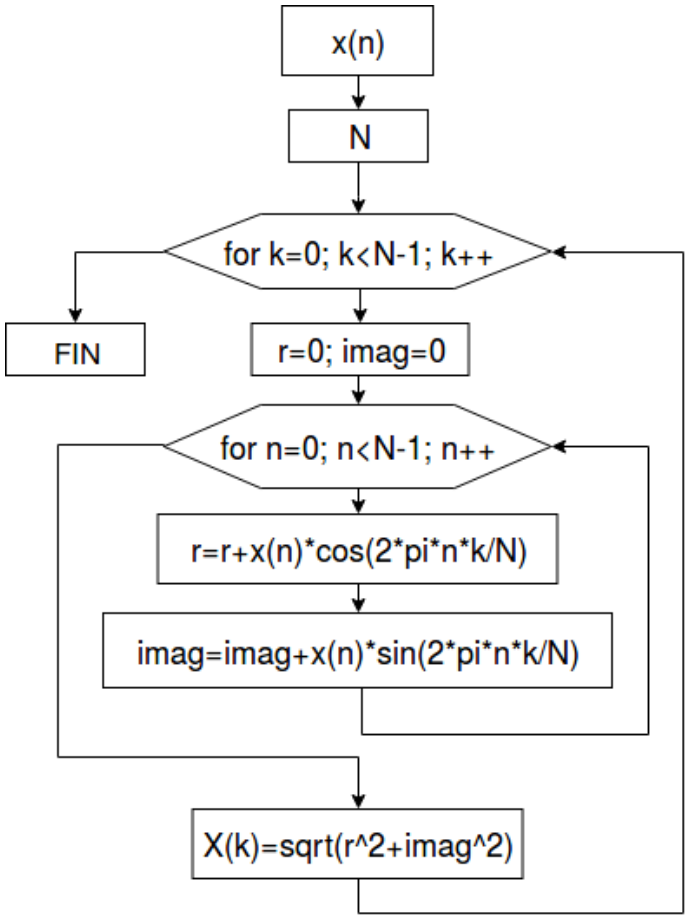


Figura 4.25: Diagrama de flujo de la Transformada discreta de Fourier (DFT).

A continuación se muestra el código en lenguaje ensamblador de la implementación de la transformada discreta de Fourier para una señal escalón de 10 elementos con aritmética de punto flotante IEEE 754.

```
*
*     TRANSFOMADA DISCRETA DE FOURIER
*                 DFT
*     En el DSP TMS320F28377S
*     en punto Flotante
*

        .global                 _c_int00
WDCR    .set    07029h          ;Dirección del reg. WatchDog Control
Nf      .set    256.0
N       .set    256
Ne      .set    512
contn   .word   0,0             ; Contador de n
contk   .word   0,0             ; Contador de k
TEMNK   .float  0,0
xi_flo  .float  1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0
xi_flo1 .space  16*Ne-10        ; Señal de entrada, llenar con ceros
xe_flo  .space  16*Ne           ; Espectro de salida

        .text
_c_int00
*****************
        EALLOW
                CLRC    XF
                SETC    SXM
                SETC    OBJMODE
                MOVL    XAR1, #WDCR
                MOV     *XAR1, #0068h
        EDIS
*****************
        MOVL    XAR2,#xe_flo    ; Espectro de salida X(k)
        MOVF32  R1H,#Nf         ; R1H = N
        NOP
        EINVF32 R3H,R1H         ; R3H = 1/R1H  = 1/N

        MOV     DP, #contn
        MOVF32  R6H,#0.0        ; Inicializa contador de k, con cero
        MOV32   @contk,R6H      ; R6H Contador de indice "n"
***** Ciclo k=0..N
        MOV     AR4,#N-1
DFTK
        MOVF32  R4H,#0.0        ; R4H=0,  parte Real de DFT
        MOVF32  R1H,#0.0        ; R1H=0,  parte Real de DFT
        MOVL    XAR1,#xi_flo    ; XAR1 apunta a inicio de Señal
                                ; de entrada x(n)

        MPYF32  R5H,R6H,R3H     ; R5H = k/N
        NOP
        NOP
```

```
        MOVF32 R6H,#0.0        ; Inicializa contador de n, con cero
        MOV32  @contn,R6H      ; R6H Contador de indice "n"
***** Ciclo   RPTB
        RPTB   FIN_B,#N-1
        MPYF32 R0H,R6H,R5H     ; R0H = n*k/N
        NOP
        NOP
        MOV32  @TEMNK,R0H

***** Calculo de parte reaL R4H = R4H +  x(n)*cos(2*pi*n*k/N)
        COSPUF32 R6H,R0H       ; R0H = cos(2*pi*n*k/N)
        NOP
        NOP
        NOP
        MOV32  R2H,*XAR1       ; R2H = dato x(n) apuntado por XAR1
        MPYF32 R0H,R6H,R2H     ; x(n)*cos(2*pi*n*k/N);
        NOP
        NOP
        ADDF32 R4H,R4H,R0H     ; R4H = R4H +  x(n)*cos(2*pi*n*k/N)
        NOP

***** Calculo de parte Imaginaria R4H = R4H + x(n)*cos(2*pi*n*k/N)
        MOV32  R0H,@TEMNK
        SINPUF32 R6H,R0H       ; R0H = sin(2*pi*n*k/N)
        NOP
        NOP
        NOP
        MOV32  R2H,*XAR1++     ;  R2H = dato x(n) apuntado por XAR1
        MPYF32 R0H,R6H,R2H     ;  x(n)*sin(2*pi*n*k/N);
        NOP
        NOP
        ADDF32 R1H,R1H,R0H     ;  R1H = R1H +  x(n)*sin(2*pi*n*k/N)
        NOP

        MOV32  R6H, @contn     ; R6H Contador de indice "n"
        ADDF32 R6H,R6H,#1.00   ;  INC  n = n+1
        NOP
        MOV32  @contn,R6H      ; R6H Contador de indice "n"
        NOP
FIN_B
***** Calculo de Magnitud  sqrt(Real^2 + Img^2) = sqrt(R4H^2 + R1H^2)
        MPYF32 R0H,R4H,R4H     ; Real^2
        NOP
        NOP

        MPYF32 R4H,R1H,R1H     ; Img^2
        NOP
        NOP
```

```
        ADDF32 R0H,R4H,R0H    ; R0H = Real^2 + Img^2
        NOP
        NOP
        SQRTF32 R4H,R0H       ; R4H = sqrt(X)
        NOP
        NOP
        NOP
        NOP
*****************
        MOV32  *XAR2++, R4H   ; Guarda Salida  sqrt(Real^2 + Img^2)

        MOV32  R6H, @contk
        ADDF32 R6H,R6H,#1.00  ; INC  k = k + 1
        NOP
        MOV32  @contk,R6H     ; R6H Contador de indice "k"
        BANZ   DFTK,AR4--

FIN_R   NOP
                LB      FIN_R   ; Ciclo infinito
                .end
```

Debido al alto número de operaciones que se necesitan para calcular la DFT, y a la relevancia de esta operación, se han desarrollado otros algoritmos para obtener el espectro de una señal con el objetivo de reducir el número de operaciones dentro del algoritmo. Uno de ellos es el algoritmo de Goertzel, el cual se se explica en en la siguiente sección.

4.7. Algoritmo de Goertzel para calcular DFT

Se utiliza para el cálculo de la DFT evitando efectuar muchos cálculos con números complejos, este algoritmo explota la periodicidad del factor W_N^{kn} y permite expresar a la DFT similar a la operación de un filtro lineal de Respuesta Infinita al Impulso (IIR).

El algoritmo de Goertzel utiliza la periodicidad del término W_N^{kn} para reducir el número de cálculos. Para la obtención del algoritmo primero se tiene en cuenta la Ecuación (4.55) [20].

$$W_N^{-kN} = e^{j(2\pi/N)Nk} = e^{j2\pi k} = 1 \tag{4.55}$$

Después se multiplica (4.56) por (4.55) y se obtiene la Ecuación (4.57).

$$X(k) = \sum_{n=0}^{N-1} x(n) W_N^{kn} \tag{4.56}$$

$$X(k) = W_N^{-kN} \sum_{r=0}^{N-1} x(r) W_N^{kr} = \sum_{r=0}^{N-1} x(r) W_N^{-k(N-r)} \tag{4.57}$$

Para el resultado final se define la secuencia:

$$y_k(n) = \sum_{r=-\infty}^{\infty} x(r) W_N^{-k(n-r)} u(n-r) \tag{4.58}$$

La Ecuación (4.58) se puede interpretar como una convolución discreta entre la secuencia de duración finita $x\,(n)$ y la secuencia $W_N^{-kN} u(n)$. En consecuencia $y_k(n)$ se puede ver como la respuesta de un sistema con respuesta al impulso $W_N^{-kN} u(n)$ a una entrada de longitud finita $x\,(n)$.

En lugar de calcular la DFT por medio de la convolución mostrada en (4.58), se puede utilizar la ecuación en diferencias correspondiente a un filtro de primer orden, el cual tiene un polo sobre la circulo unitario a la frecuencia $\omega = 2\pi k/N$. De esta manera se calcula $y_k\,(n)$ de forma recursiva, como se muestra en (4.59) [14].

$$y_k(n) = W_N^{-k} y_k(n-1) + x(n) \tag{4.59}$$

Es posible mantener esta simplificación reduciendo el número de multiplicaciones por un factor de 2. Esto se consigue multiplicando la función de transferencia correspondiente a la ecuación en diferencias (4.59), por un factor igual a $(1 - W_N^k z^{-1})$ tanto en el denominador como el numerador.

La función de transferencia resultante corresponde a la implementación de la Figura 4.26, por otro lado la ecuación en diferencias de los polos es la (4.60).

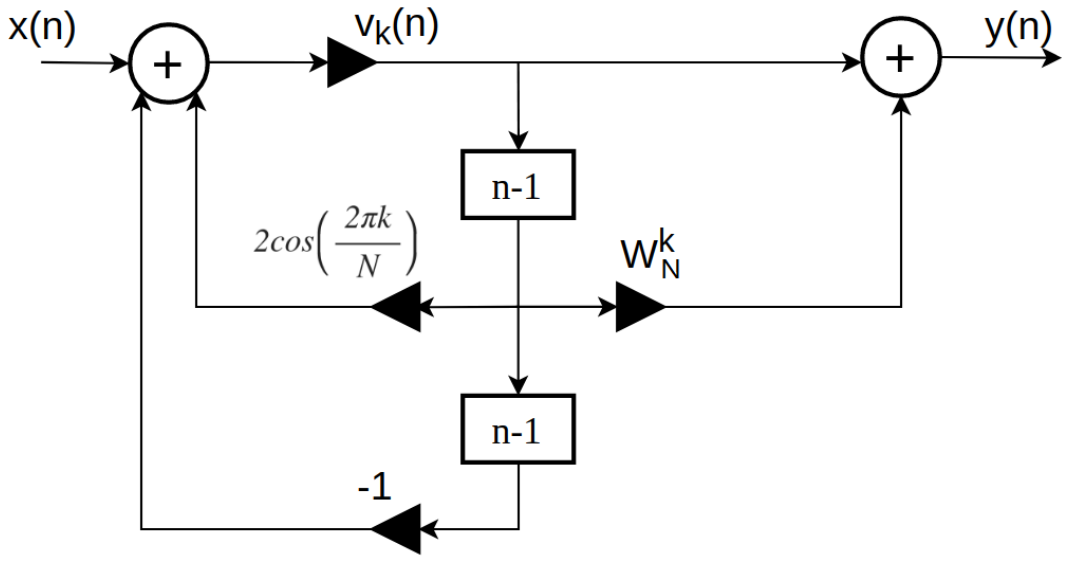


Figura 4.26: Flujo de señal para el computo del algoritmo de Goertzel.

$$v_k(n) = 2\cos\left(\frac{2\pi k}{N}\right) v_k(n-1) - v_k(n-2) + x(n) \tag{4.60}$$

Después de efectuar N iteraciones de (4.60) (comenzando con condiciones de reposo inicial), el valor esperado de la DFT se puede obtener implementando el cero de la función de transferencia como lo indica la Ecuación (4.61) [20].

$$X(k) = y_k(n)|_{n=N} = v_k(N) - W_N^k v_k(N-1) \tag{4.61}$$

Implementación del algoritmo de Goertzel

El programa de implementación propuesto, utiliza la unidad de punto flotante FPU y la de trigonométrica TMU del TMS320F28377S. Además de calcular la DFT por el algoritmo de Goertzel, también se obtiene el espectro en magnitud dicha señal. La secuencia de entrada es generada como

$$x_{osc}\,(n) = x_1\,(n) + x_2\,(n) \tag{4.62}$$

donde

$$x_1(n) = \cos\left(\frac{2\pi f_1 n}{fs}\right) \tag{4.63}$$

y

$$x_2(n) = \cos\left(\frac{2\pi f_2 n}{fs}\right) \tag{4.64}$$

donde $n = 0, 1, \ldots, N-1$, la frecuencia de muestreo es $f_s = 1024$ Hz, f_1 y f_2 tomarán los valores de prueba; 13 y 31 Hz, 23 y 41 Hz, y el último par de prueba 19 y 51 Hz, respectivamente. Por cada par de frecuencia, se generó una secuencia la cual fue almacenada en un archivo *.dat* en formato de punto flotante, teniendo los siguientes tres archivos:

- xnFloat-13-31Hz.dat
- xnFloat-23-41Hz.dat
- xnFloat-27-51Hz.dat
- xnImpFloat.dat

Para realizar las aplicaciones, se siguió el diagrama de bloques mostrado en la Figura 4.27.

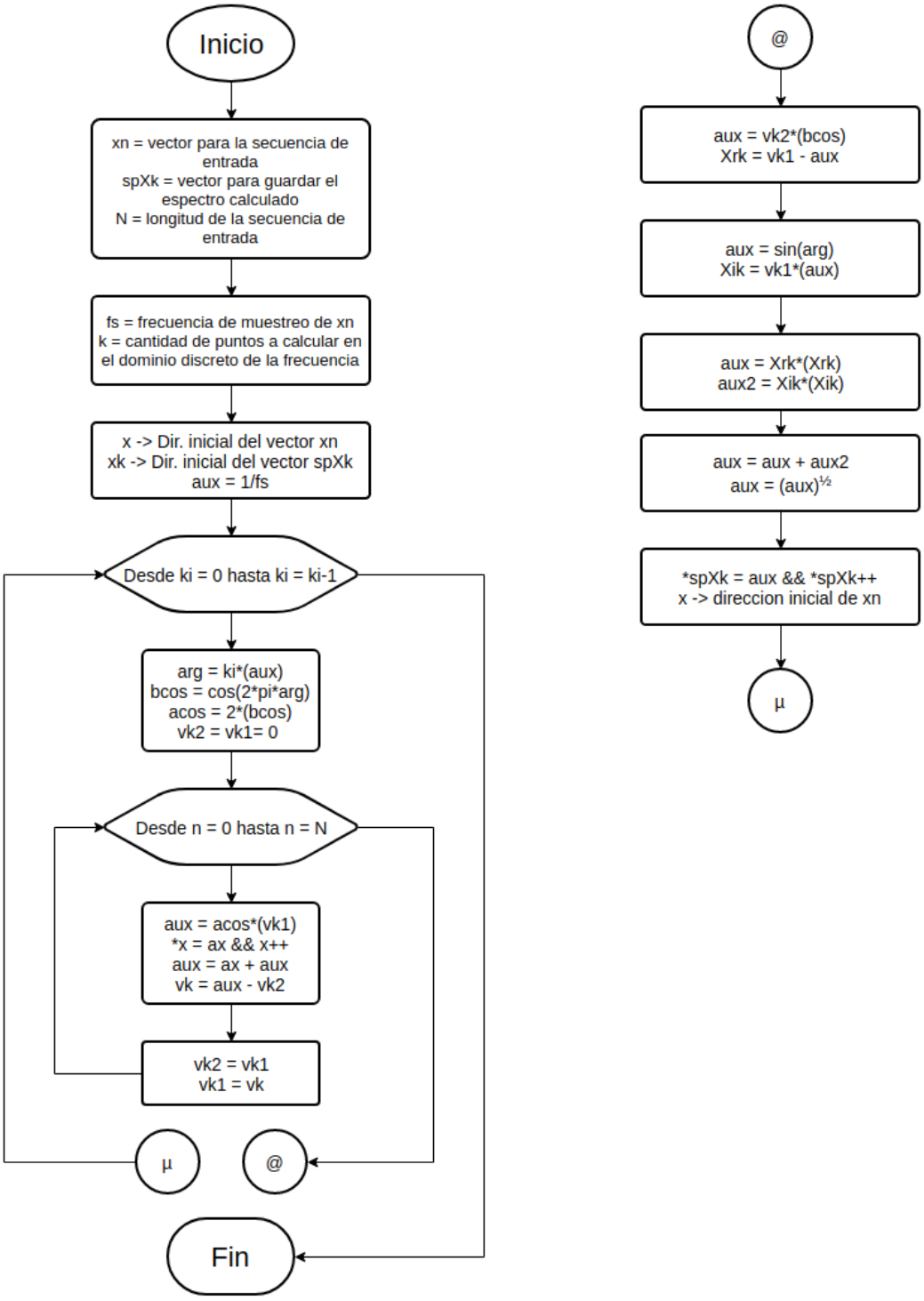


Figura 4.27: Diagrama de bloques del algoritmo de Goerzel.

4.7.1. Algoritmo de Goertzel en punto flotante IEEE 754

El programa que se presenta a continuación, es capaz de calcular la transformada discreta de Fourier para N términos de entrada, en este caso para probar su funcionamiento, se utilizarán tres diferentes secuencias de entrada $x(n)$, generadas a partir de (4.62), para observar el espectro generado por el cálculo de la DFT del código.

Para probar el funcionamiento del programa, se deberá de cargar a la memoria alguno de los tres archivos citados anteriormente, para poder observar sus componentes espectrales, con el cálculo de la DFT obtenido del programa de implementación propuesto, que se muestra a continuación.

```
*
* Programa para calcular la DFT
* utilizando el algoritmo de Goertzel
*
        .global         _c_int00
        ;.data
WDCR    .set    07029h ; Dirección del reg. de control
                       ; WatchDog
N       .set    1024   ; Longitud de la secuencia de entrada
Fsflo   .set    1024.0 ; Frecuencia de muestreo
xn      .space N*32    ; Espacio reservado para la secuencia
                       ; de entrada a la transformada DFT
arg     .float 0.0     ; Localidad de memoria auxiliar
cos_arg .float 0.0     ; Localidad de memoria auxiliar
spXk    .space N*32    ; Espacio reservado para guardar
                       ; el espectro
k       .float 0.0     ; Dominio discreto de la frecuencia
kint     .set  512     ; contador de control ciclo k

        .text
_c_int00
; Deshabilitacion del WatchDog
        EALLOW                  ; Habilita la escritura a registros
                                ; protegidos
        MOVL    XAR1, #WDCR     ; Registro XAR1 apunta dir, WDCR
        MOV     *XAR1,#0068h    ; Desactiva WatchDog
        EDIS                    ; Deshabilita escritura a registros
                                ; protegidos

* Calculo de la DFT por algoritmo de Goertzel
* Definición de condiciones iniciales
        MOVF32 R0H,#Fsflo       ; R0H = Nflo
        MOVL    XAR0,#xn        ; Apuntador dirigido a xn
        MOVL    XAR1,#spXk      ; Apuntador dirigido a spXk
        EINVF32 R1H,R0H         ; Obtiene 1/N
```

```
        MOV     DP,#k           ; Direccionamiento de página
        MOV     AR2,#kint-2     ; AR2 = Número de repeticiones
                                ; a realizar el ciclo k

* Inicio del ciclo de evaluación del término k
K_CYCLE
        MOVL    XAR3,#k         ; XAR3 = &k
        MOV32   R0H,*XAR3       ; R0H = k
        MPYF32  R2H,R1H,R0H     ; Multiplicación R2H = k*(1/N)
        MOVF32  R4H,#0.0        ; R4H = vk(n-1)
        MOVL    XAR3,#arg       ; XAR3 = &arg
        COSPUF32 R0H,R2H        ; R0H = cos(2pi*[R2H = k/N])
        MOVF32  R3H,#2.0        ; R3H = 2.0
        MOV32   *XAR3,R2H       ; arg = R2H = k/N
        MOVL    XAR3,#cos_arg   ; XAR3 = &cos_arg
        MPYF32  R2H,R3H,R0H     ; R2H = 2.0*cos(2pi*k/N)
        MOVF32  R5H,#0.0        ; R5H = vk(n-2)
        MOV32   *XAR3,R0H       ; cos_arg=R0H=cos(2pi*[R2H=k/N])

* Ciclo para hacer el barrrido de n (1ra. fase del algoritmo)
        RPTB    N_CYCLE,#N-1
                MPYF32  R0H,R2H,R4H     ; R0H=2.0*vk(n-1)*
                                        ; *cos(2*pi*k/N)
                MOV32   R3H,*XAR0++     ; R3H = x(n)
                NOP                     ; Ciclo extra para
                                        ; finalizar MOV32
                ADDF32  R6H,R3H,R0H     ; R6H=x(n)+
                                        ; +2.0*cos(2pik/N)*vk(n-1)
                NOP                     ; Ciclo extra para
                                        ; finalizar ADDF32
                NOP                     ; Ciclo extra para
                                        ; finalizar ADDF32
                SUBF32  R0H,R6H,R5H     ; R0H=x(n)+
                                        ; 2.0*cos(2pik/N)*vk(n-1)]-
                                        ; -vk(n-2)
                NOP                     ; Ciclo extra para
                                        ; finalizar SUBF32
                NOP                     ; Ciclo extra para
                                        ; finalizar SUBF32
                MOV32   R5H,R4H         ; R5H=R4H->vk(n-2)=
                                        ; =vk(n-1)
                NOP                     ; Ciclo extra para
                                        ; finalizar MOV32
                MOV32   R4H,R0H         ; R4H=R0H->vk(n-1)=
                                        ; =vk(n)
                NOP                     ; Ciclo extra para
                                        ; finalizar MOV32
```

```
N_CYCLE
; Cálculo de la parte imaginaria y real del término k-ésimo
; (2da. fase del algoritmo)
        MOV32   R6H,*XAR3       ; R6H = cos(2pi*[R2H=k/N])
        MPYF32  R6H,R5H,R6H     ; R6H=vk(n-2)*cos(2pi*[R2H=k/N])
        MOVL    XAR3,#arg       ; XAR3 = &arg
        MOV32   R0H,*XAR3       ; R0H = arg = k/N
        SUBF32  R6H,R4H,R6H     ; Xr(k)=R6H=
                                ; =vk(n-1)-vk(n-2)*cos(2pi*k/N)
        SINPUF32 R0H,R0H        ; R0H = sin(2pi*k/N)
        MOVL    XAR3,#k         ; XAR3 = &k
        MOV32   R7H,*XAR3       ; R7H = k
        ADDF32  R7H,R7H,#1.00   ; R7H = R7H +1
        NOP                     ; Ciclo extra para
                                ; finalizar ADDF32
        MPYF32  R5H,R5H,R0H     ; Xi(k)=R5H=vk(n-2)*sin(2pi*k/N)
        MOV32   *XAR3,R7H       ; XAR3 => k = k + 1
        NOP                     ; Ciclo extra para
                                ; finalizar MOV32

; Cálculo del espectro del término k-ésimo
        MPYF32  R6H,R6H,R6H     ; R6H = [Xr(k)]^2
        NOP                     ; Ciclo extra para finalizar MPYF32
        NOP                     ; Ciclo extra para finalizar MPYF32
        MPYF32  R5H,R5H,R5H     ; R5H = [Xi(k)]^2
        NOP                     ; Ciclo extra para finalizar MPYF32
        NOP                     ; Ciclo extra para finalizar MPYF32
        ADDF32  R7H,R6H,R5H     ; R7H = [Xr(k)]^2 + [Xi(k)]^2
        NOP                     ; Ciclo extra para finalizar ADDF32
        NOP                     ; Ciclo extra para finalizar MPYF32
        SQRTF32 R6H,R7H         ; R6H = sqrt([Xr(k)]^2 + [Xi(k)]^2)
        NOP                     ; Ciclo extra para finalizar SQRTF32
        NOP                     ; Ciclo extra para finalizar SQRTF32
        NOP                     ; Ciclo extra para finalizar SQRTF32
        MOVL    XAR0,#xn        ; Direccionamiento al primer
                                ; elemento de xn
        MOV32   *XAR1++,R6H     ; spX(k) = R6H

        BANZ K_CYCLE,AR2--      ; Ciclo de cálculo de cada término
                                ; de la DFT

iend    NOP
        LB iend                 ; Ciclo infinito
        .end
```

Como se puede observar, el algoritmo se desarrolla utilizando las herramientas de las unidades FPU y TMU, manejando datos en formato de punto flotante. Sin embargo, es posible trabajar con datos de entrada al programa anterior, si incluimos una rutina de conversión de formatos de punto fijo a punto flotante, como se muestra en el siguiente programa.

```
*
* Programa para calcular la DFT
* utilizando el algoritmo de Goertzel
* con entrada de datos en formato de
* punto fijo a 32 bits
*
        .global     _c_int00
        .data
WDCR    .set    07029h          ; Dirección del reg. de
                                ; control WatchDog
N       .set    1024            ; Longitud de la secuencia
                                ; de entrada
Fsflo   .set    1024.0          ; Frecuencia de muestreo
xn      .space N*32             ; Espacio reservado para la
                                ; secuencia de entrada a la
                                ; transformada DFT
Qi      .set    268435456.0     ; Valor para hacer la
                                ; conversión a formato de punto
                                ; entero de la señal generada
                                ; qi=2^(no. de bits para
                                ; parte fraccionaria)
                                ; este valor es para Qi=28
arg     .float 0.0              ; Localidad de memoria auxiliar
cos_arg .float 0.0              ; Localidad de memoria auxiliar
spXk    .space N*32             ; Espacio reservado para guardar
                                ; el espectro
k       .float 0.0              ; Dominio discreto de la frecuencia
kint    .set    512             ; contador de control ciclo k

        .text
_c_int00
* Deshabilitacion del WatchDog
        EALLOW                  ; Habilita la escritura de registros
                                ; protegidos
        MOVL    XAR1, #WDCR     ; Registro XAR1 apunta dir, WDCR
        MOV     *XAR1, #0068h   ; Desactiva WatchDog
        EDIS                    ; Deshabilita escritura a registros
                                ; protegidos

* Conversión de la secuencia de datos en formato de punto entero
* a punto flotante IEE754
        MOVL    XAR0,#xn        ; Apuntador hacia la dirección
                                ; inicial de la secuencia xn
        MOVF32 R0H,#Qi          ; Coloca el dato de conversión
                                ; a formato entero
        RPTB    INT2FL,#N-1     ; Ciclo de conversión de números
                                ; enteros a flotantes
            MOV32   R1H,*XAR0       ; R1H = xn(i)
```

```
                I32TOF32 R1H,R1H      ; Conversión de entero
                                      ; a flotante del dato xn(i)
                NOP
                NOP
                DIVF32 R2H,R1H,R0H    ; Conversión de formato
                                      ; R2H = xn(i)/Qi
                NOP
                NOP
                NOP
                NOP
                MOV32  *XAR0++,R2H    ; xn(i)=floatIEE754(xn(i))
INT2FL
* Cálculo de la DFT por algoritmo de Goertzel
* Definición de condiciones iniciales
        MOVF32 R0H,#Fsflo     ; R0H = Nflo
        MOVL   XAR0,#xn       ; Apuntador dirigido a la 1er
                              ; loc. de xn (XAR0 BLOCK)
        MOVL   XAR1,#spXk     ; Apuntador dirigido a la 1er
                              ; loc. de spXk (XAR1 BLOCK)
        EINVF32 R1H,R0H       ; Obtiene 1/N ( R1H BLOCK )
        MOV DP,#k             ; Direccionamiento de página
        MOV AR2,#kint-2       ; AR2 = Número de repeticiones
                              ; a realizar el (AR2 BLOCK)
                              ; ciclo k

* Inicio del ciclo de evaluación del término k
K_CYCLE
        MOVL   XAR3,#k                ; XAR3 = &k
        MOV32  R0H,*XAR3      ; R0H = k
        MPYF32 R2H,R1H,R0H    ; Multiplicación R2H = k*(1/N)
        MOVF32 R4H,#0.0       ; R4H = vk(n-1)       ( R4H BLOCK )
        MOVL   XAR3,#arg      ; XAR3 = &arg
        COSPUF32 R0H,R2H      ; R0H = cos(2pi*[R2H = k/N])
        MOVF32 R3H,#2.0       ; R3H = 2.0
        MOV32  *XAR3,R2H      ; arg = R2H = k/N
        MOVL   XAR3,#cos_arg  ; XAR3 = &cos_arg
        MPYF32 R2H,R3H,R0H    ; R2H = 2.0*cos(2pi*k/N)
        MOVF32 R5H,#0.0       ; R5H = vk(n-2)       ( R5H BLOCK )
        MOV32  *XAR3,R0H      ; cos_arg = R0H =
                              ; = cos(2pi*[R2H = k/N])

; Ciclo para hacer el barrrido de n (1ra. fase del algoritmo)
        RPTB N_CYCLE,#N-1
                MPYF32 R0H,R2H,R4H    ; R0H = 2.0*cos(2*pi*k/N)*
                                      ; *vk(n-1)
                MOV32  R3H,*XAR0++    ; R3H = x(n)
                NOP
                ADDF32 R6H,R3H,R0H    ; R6H = x(n) +
                                      ; +2.0*cos(2pik/N)*vk(n-1)
```

```
                NOP
                NOP
                SUBF32  R0H,R6H,R5H     ; R0H = [x(n) +
                                        ; +2.0*cos(2pik/N)*vk(n-1)]-
                                        ; - vk(n-2)
                NOP
                NOP
                MOV32   R5H,R4H         ; R5H = R4H -> vk(n-2) =
                                        ; = vk(n-1)
                NOP
                MOV32   R4H,R0H         ; R4H = R0H -> vk(n-1) =
                                        ; = vk(n)
                NOP
N_CYCLE
; Cálculo de la parte imaginaria y real del término k-ésimo
; (2da. fase del algoritmo)
        MOV32   R6H,*XAR3       ; R6H = cos(2pi*[R2H = k/N])
        MPYF32  R6H,R5H,R6H     ; R6H = vk(n-2)*cos(2pi*[R2H=k/N])
        MOVL    XAR3,#arg       ; XAR3 = &arg
        MOV32   R0H,*XAR3       ; R0H = arg = k/N
        SUBF32  R6H,R4H,R6H     ; Xr(k) = R6H = vk(n-1) -
                                ; - vk(n-2)*cos(2pi*k/N)
        SINPUF32 R0H,R0H        ; R0H = sin(2pi*k/N)
        MOVL    XAR3,#k         ; XAR3 = &k
        MOV32   R7H,*XAR3       ; R7H = k
        ADDF32  R7H,R7H,#1.00   ; R7H = R7H +1
        NOP
        MPYF32  R5H,R5H,R0H     ; Xi(k) = R5H =
                                ; = vk(n-2)*sin(2pi*k/N)
        MOV32   *XAR3,R7H       ; XAR3 => k = k + 1
        NOP

; Cálculo del espectro del término k-ésimo
        MPYF32  R6H,R6H,R6H     ; R6H = [Xr(k)]^2
        NOP
        NOP
        MPYF32  R5H,R5H,R5H     ; R5H = [Xi(k)]^2
        NOP
        NOP
        ADDF32  R7H,R6H,R5H     ; R7H = [Xr(k)]^2 +
                                ; + [Xi(k)]^2
        NOP
        NOP
        SQRTF32 R6H,R7H         ; R6H = sqrt( [Xr(k)]^2 +
                                ; + [Xi(k)]^2 )
        NOP
        NOP
        NOP
        MOVL    XAR0,#xn
```

```
        MOV32   *XAR1++,R6H     ; spX(k) = R6H

        BANZ    K_CYCLE,AR2--

iend    NOP
        LB      iend            ; Ciclo infinito
        .end
```

Como nota adicional, antes de iniciar el funcionamiento del programa anterior, se debe especificar el formato Qi en el que se encuentran los datos de entrada, en la variable correspondiente denominada Qi.

Análisis de resultados de la DFT utilizando el algoritmo de Goertzel

Para evaluar el desempeño de la implementación propuesta, se utilizó el método del algoritmo de Goertzel que tiene el programa de cálculo Octave para calcular la DFT y posteriormente los espectros en magnitud de las secuencias propuestas y de un impulso. Los resultados obtenidos en formato flotante de doble precisión (en Octave) se compararon con los datos del espectro en magnitud calculado por el programa propuesto, y se guardaron los datos obtenidos por las diferentes secuencias de entrada propuestas. El desempeño se evaluó mediante la obtención del error absoluto, obteniendo los resultados mostrados en las Tablas 4.11 y 4.12.

Tabla 4.11: Error absoluto entre los espectros en magnitud de la respuesta al impulso de los programas de implementación del algoritmo de Goertzel.

Formato de entrada	**Formato de salida**	**Error Absoluto**		
		Min.	**Prom.**	**Max.**
Float simple	Float simple	0	1.4133e-05	7.0632e-05
Q28 L=32 bits	Float simple	0	1.4132e-05	7.0631e-05

La rutina de conversión de números de formato de punto fijo a punto flotante simple, utilizando la FPU no influyó en el error obtenido al probar el programa propuesto con una secuencia de entrada impulso, por lo que se puede decir que el desempeño de la implementación es adecuado, por tanto, en la Tabla 4.12 solo se registran los errores absolutos obtenidos entre el espectro en magnitud calculado por el TMS320F28377S y el que se obtuvo en Octave, utilizando los datos de entrada en formato de punto flotante.

Tabla 4.12: Error absoluto de los espectros de magnitud de las secuencias $x\,(n)$ propuestas para evaluar las implementaciones propuestas.

Secuencia de datos de entrada	**Error Absoluto**		
	Min.	**Prom.**	**Max.**
xnFloat-13-31Hz	8.1957e-12	0.0015	0.0765
xnFloat-23-41Hz	6.2593e-12	0.0011	0.0435
xnFloat-27-51Hz	6.0066e-12	9.9115e-04	0.0458

Los datos de Tabla 4.12 corroboran los errores absolutos obtenidos al probar las implementaciones con un señal impulso, por lo que el programa del algoritmo de Goertzel se puede considerar con un funcionamiento aceptable, dependiendo de la aplicación en la que se quiera utilizar. Las Figuras 4.28, 4.29 y 4.30 muestran el espectro en magnitud calculado por el programa anterior, de cada una de las secuencias de entrada generadas a partir de (4.62).

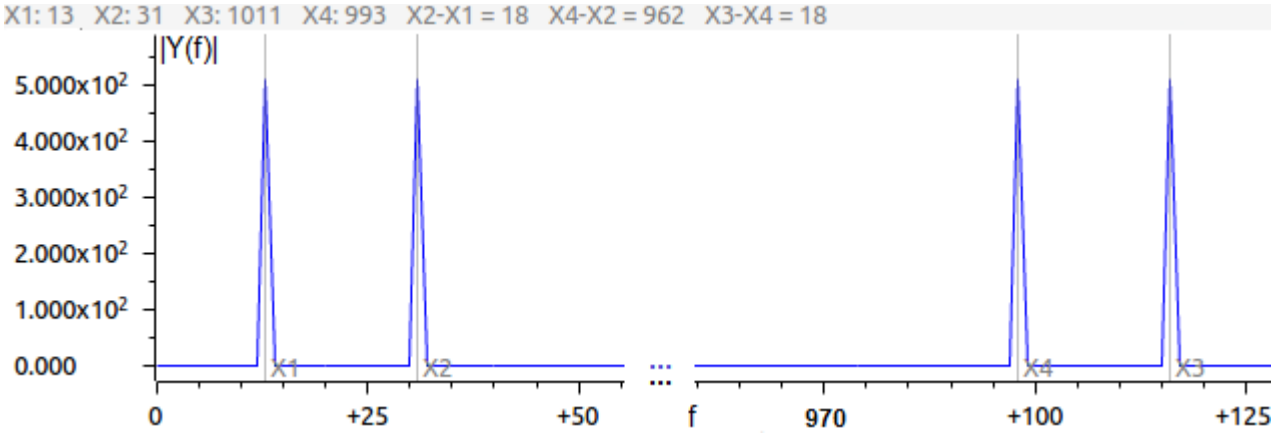


Figura 4.28: Espectro en magnitud de la Transformada Discreta de Fourier para la secuencia de entrada (4.62) con $f_1 = 13$ y $f_2 = 31$

En los tres espectros mostrados anteriormente, la magnitud de los picos coinciden con las componentes espectrales de las secuencias de entrada que se le ingresaron al programa para el cálculo de la DFT, respaldando los errores absolutos registrados en la Tabla 4.12.

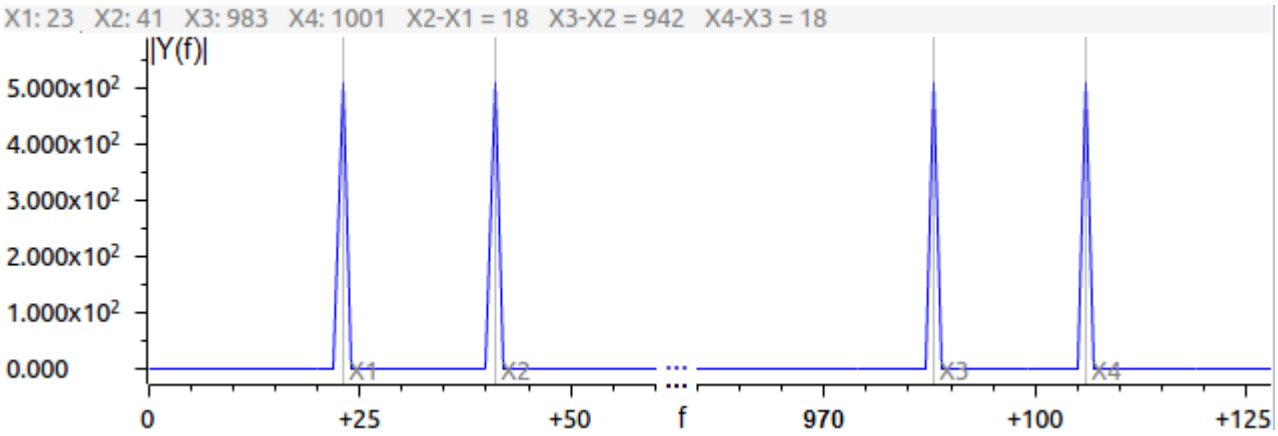


Figura 4.29: Espectro en magnitud de la Transformada Discreta de Fourier para la secuencia de entrada (4.62) con $f_1 = 23$ y $f_2 = 41$

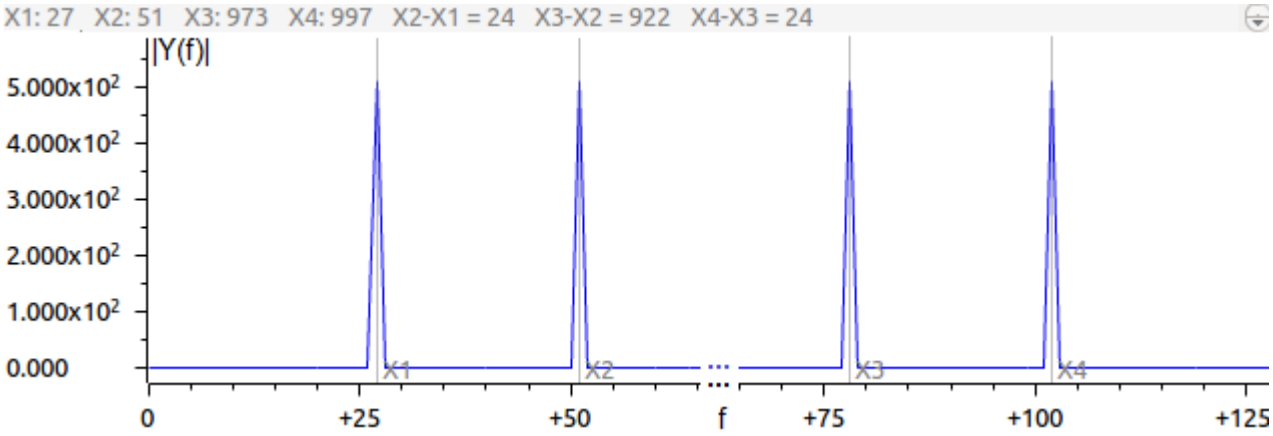


Figura 4.30: Espectro en magnitud de la Transformada Discreta de Fourier para la secuencia de entrada (4.62) con $f_1 = 27$ y $f_2 = 51$

4.8. La Transformada rápida de Fourier FFT

En la Sección 4.6, se explicó la DFT, la cual nos permite calcular el espectro discreto de una señal o un sistema discreto. No obstante, implementar la DFT involucra una alta demanda de recursos en un procesador de señales, esto se debe a que el número de operaciones complejas que se requieren efectuar están en el oden de $O(N^2)$, razón por la cual es muy complicado ejecutarla en tiempo real. Sin embargo, han surgido algunos métodos para el cálculo de dicha transformada, con el objetivo de reducir el número de operaciones y así poder implementarlos en aplicaciones de tiempo real como es el caso de la Transformada rápida de Fourier (FFT por sus siglas en inglés).

En esencia, la FFT es una forma eficiente de calcular la DFT y su éxito se debe a la reducción del número de adiciones y multiplicaciones requeridas para su cálculo, explotando las propiedades de simetría y periodicidad de la DFT y los factores W_N^k. La FFT reduce las operaciones a $O\left(N\log_2(N)\right)$ manteniendo el desempeño en el resultado [15] [14] [20].

Desarrollando (4.39) con una señal discreta $x\left(n\right)$ de orden $N=4$, además considerando que $k=0,1,2,3$, se genera un sistema de ecuaciones como el que se muestra a continuación:

$$\begin{aligned} X\left(0\right) &= x\left(0\right)W^0 + x\left(1\right)W^0 + x\left(2\right)W^0 + x\left(3\right)W^0 \\ X\left(1\right) &= x\left(0\right)W^0 + x\left(1\right)W^1 + x\left(2\right)W^2 + x\left(3\right)W^3 \\ X\left(2\right) &= x\left(0\right)W^0 + x\left(1\right)W^2 + x\left(2\right)W^4 + x\left(3\right)W^6 \\ X\left(3\right) &= x\left(0\right)W^0 + x\left(1\right)W^3 + x\left(2\right)W^6 + x\left(3\right)W^9 \end{aligned} \tag{4.65}$$

Se observa que (4.65) se puede expresar en forma matricial de la forma $\boldsymbol{X}_n\left(k\right) = \boldsymbol{W}_N\boldsymbol{x}_n\left(n\right)$ como muestra en (4.66).

$$\begin{bmatrix} X\left(0\right) \\ X\left(1\right) \\ X\left(2\right) \\ X\left(3\right) \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & W^1 & W^2 & W^3 \\ 1 & W^2 & W^4 & W^6 \\ 1 & W^3 & W^6 & W^9 \end{bmatrix} \begin{bmatrix} x\left(0\right) \\ x\left(1\right) \\ x\left(2\right) \\ x\left(3\right) \end{bmatrix} \tag{4.66}$$

Al sustituir los valores numéricos respectivos de los factores W_N^k, la matriz $\boldsymbol{W}_N$ de (4.66) queda de la siguiente manera:

$$\boldsymbol{W}_N = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{bmatrix} \tag{4.67}$$

Factorizando la Matriz $\boldsymbol{W}_N$ se obtiene:

$$\boldsymbol{W}_N = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & -j \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & j \end{bmatrix} \begin{bmatrix} 1 & 0 & 1 & 0 \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & -1 \end{bmatrix} \tag{4.68}$$

Sustituyendo las matrices resultantes de la factorización de $\boldsymbol{W}_N$ (4.68) en (4.66), se obtiene el siguiente sistema de ecuaciones:

$$\begin{aligned} X(0) &= x(0) + x(1) + x(2) + x(3) \\ X(1) &= x(0) - jx(1) - x(2) + jx(3) \\ X(2) &= x(0) - x(1) + x(2) - x(3) \\ X(3) &= x(0) + jx(1) - x(2) - x(3) \end{aligned} \tag{4.69}$$

Por lo que al implementar el sistema de ecuaciones dado en (4.69) se puede obtener el diagrama de la Figura 4.31, el cual describe la FFT de orden $N = 4$.

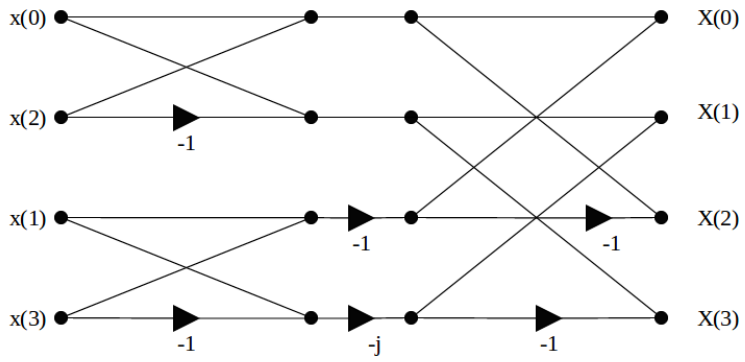


Figura 4.31: Diagrama de operaciones de la FFT para orden $N = 4$

De la Figura 4.31 se puede observar que existe una estructura base, la cual se repite a lo largo de la FFT, esta estructura se conoce como operación mariposa, misma que se muestra en la Figura 4.32, donde todos los operandos de entrada ($a,\ b$), salida ($A,\ B$) y factores W son valores complejos.

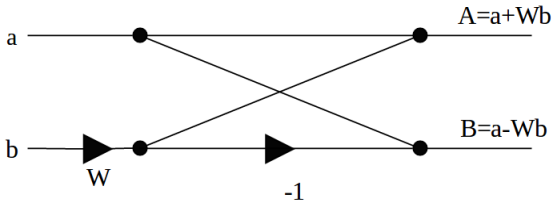


Figura 4.32: Operación mariposa.

En la Figura 4.33 se muestra un diagrama del algoritmo FFT para una secuencia de datos de $N = 16$, dicho algoritmo se conoce como Radix 2.

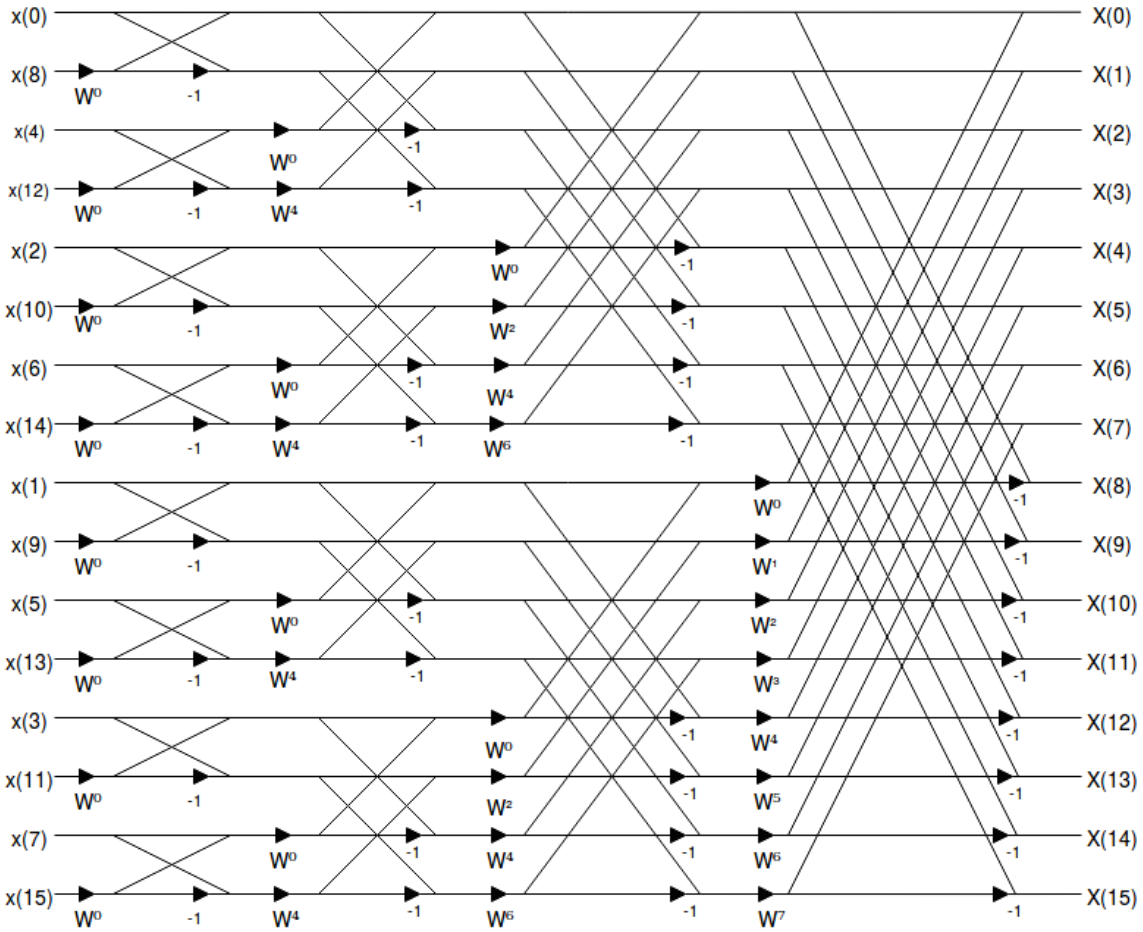


Figura 4.33: Ejemplo de decimación por acarreo inverso.

Se puede observar que los datos de la secuencia $x(n)$ de la Figura 4.31 no tiene un orden natural, sin embargo, los elementos de la secuencia $X(k)$ si se encuentran ordenados. Para poder efectuar la FFT por medio del diagrama de mariposas, es necesario realizar el re-ordenamiento de los datos de la secuencia de entrada $x(n)$, dicho reordenamiento es conocido como proceso de decimación el cual se explica a continuación.

4.8.1. Decimación

El proceso de decimación es un re-ordenamiento de forma especial de los elementos de un vector. Existen diferentes formas en las cuales se puede llevar a cabo la decimación, sin embargo, en la presente sección se explicará la decimación por acarreo inverso.

El método del acarreo inverso consiste en tomar el índice del primer elemento de la secuencia (elemento 0), y sumarle de manera inversa (de izquierda a derecha) $\frac{N}{2}$ unidades en

binario, obteniendo de esta manera el siguiente índice decimado, posteriormente se le vuelve a sumar $\frac{N}{2}$ unidades en binario de manera inversa, continuando el procedimiento hasta obtener el último índice el cual debe contener l bits de valor 1. En la Figura 4.34 se muestra un ejemplo del proceso de decimación con una secuencia $x\,(n)$ de longitud $N = 8$ para $L = 3$ bits.

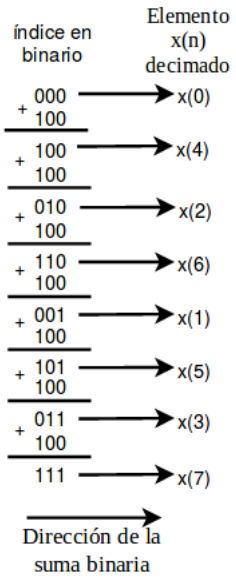


Figura 4.34: Ejemplo de decimación por acarreo inverso.

Existen otros métodos de decimación [12], sin embargo, los DSP utilizan este método del acarreo inverso para realizar la decimación de una secuencia, este proceso se muestra en el siguiente ejemplo, utilizando $N = 16$, el lector puede extender el valor de N.

```
*
* Decimacion de una secuencia de datos x(n)
* para realizar la FFT radix 2
* BR: significa bit reverse o acarreo inverso
*

        .global         _c_int00
WDCR    .set    07029h
N       .set    16
; Secuencia x(n) ordenada
xn      .word   0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15
; Secuencia x(n) decimada
xnd     .word   0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0

        .text
_c_int00
        EALLOW
```

```
            MOVL    XAR1,#WDCR
            MOV             *XAR1,#0069h
        EDIS

        SETC    SXM
        SPM     #0

        MOVL    XAR1,#xn        ; AR1=dir_inicio de xn
        MOV     AR0,#N/2        ; AR0=N/2 para la decimación
        MOVL    XAR2,#xnd       ; AR2=dir_inicio de xnd
        MOV     AR3,#N-1        ; AR3 contador de ciclo BANZ

CICLO
        NOP     *,ARP1          ; Selecciona a AR1 como registro
                                ; en uso
        MOV     AL,*BR0++       ; AL=dato xn,AR1=AR1+BR(AR0)
        MOV     *XAR2++,AL      ; Mueve xn a xnd
        BANZ    CICLO,AR3--     ; Retorna a Ciclo si
                                ; AR3 !=0, AR3=AR3-1SETC    SXM
FIN             NOP
                LB      FIN     ; Ciclo infinito
        .end
```

4.9. Implementación de la FFT en 16 y 32 bits en punto entero

La implementación de la FFT en el presente material se realizó con el algoritmo de decimación en el tiempo. El algoritmo consta en ejecutar operaciones mariposa a lo largo de los índices de la señal previamente decimada e ir sustituyendo los resultados de dichas operaciones en el mismo buffer de datos de la señal con el objetivo de utilizar la menor cantidad de memoria posible.

Es importante conocer la longitud de datos de la señal de entrada, porque el número de etapas del algoritmo está en función de la longitud de la señal. Ésto es $N\_etapas = \log_2(N)$, donde N es la longitud de $x(n)$.

El primer paso para ejecutar la FFT consta de realizar la decimación en el tiempo tal como se explicó en la Sección 4.8.1. La complejidad de la implementación de la FFT radix 2 se centra en la obtención de los índices para ejecutar la operación mariposa, es decir, mandar los datos adecuados de la señal a la función que ejecuta la operación mariposa, así como los coeficientes correctos.

El algoritmo de la FFT consta principalmente de tres ciclos anidados, el ciclo externo

está en función del número de etapas que tiene el algoritmo, y dicho valor está a su vez en función de la longitud de la señal como se explicó anteriormente.

En la Figura 4.33 se muestra un diagrama del algoritmo FFT Radix 2 para una señal de orden 16, en la cual se puede observar que consta de cuatro etapas. A su vez, contiene dos ciclos anidados en cada una de las etapas. El primero de ellos se relaciona con el número de bloques de operaciones mariposa y el tercer ciclo anidado está en función del número de operaciones mariposa que se ejecutan en cada bloque. Para visualizar mejor lo anterior, nos basaremos en el algoritmo Radix 2 de la FFT de la Figura 4.33, en el cual se observa que tiene una longitud de $L = 16$, por lo que consta de cuatro etapas.

En cada una de las etapas se puede observar que hay un número de bloques de operaciones de mariposa, por ejemplo, en la segunda etapa contiene cuatro bloques con dos operaciones de mariposa, la tercera etapa contiene dos bloques con cuatro operaciones de mariposa. De tal manera que entre mayor es el número de etapa, menor será el número de bloques y mayor la cantidad de operaciones mariposa definiendo así el comportamiento de los ciclos.

La operación mariposa, es la operación básica del algoritmo de la FFT, misma que se puede observar en la Figura 4.32. Esta operación involucra una suma de números complejos, una multiplicación de números complejos y una resta de números complejos. Se puede observar que es la misma operación a lo largo del algoritmo y lo único que cambian son los datos de entrada y los coeficientes de dicha operación, de tal manera que se recomienda crear una función externa que ejecute la operación mariposa, donde las variables de entrada son los índices donde apuntan los datos de los arreglos con los cuales se realiza la operación.

En la Figura 4.35 se muestra un diagrama de bloques para la implementación de dicha función.

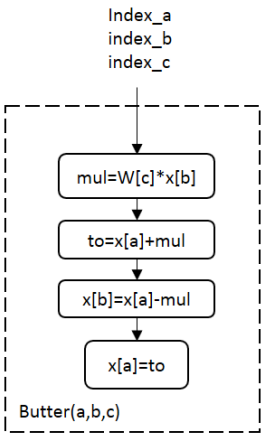


Figura 4.35: Diagrama de flujo de la función que realiza operación mariposa.

En la Figura 4.36 se muestra el diagrama de bloques que representa la implementación del algoritmo FFT Radix 2 tomando en cuenta que la longitud de la señal de entrada debe de ser $N = 2^{N\_etapas}$.

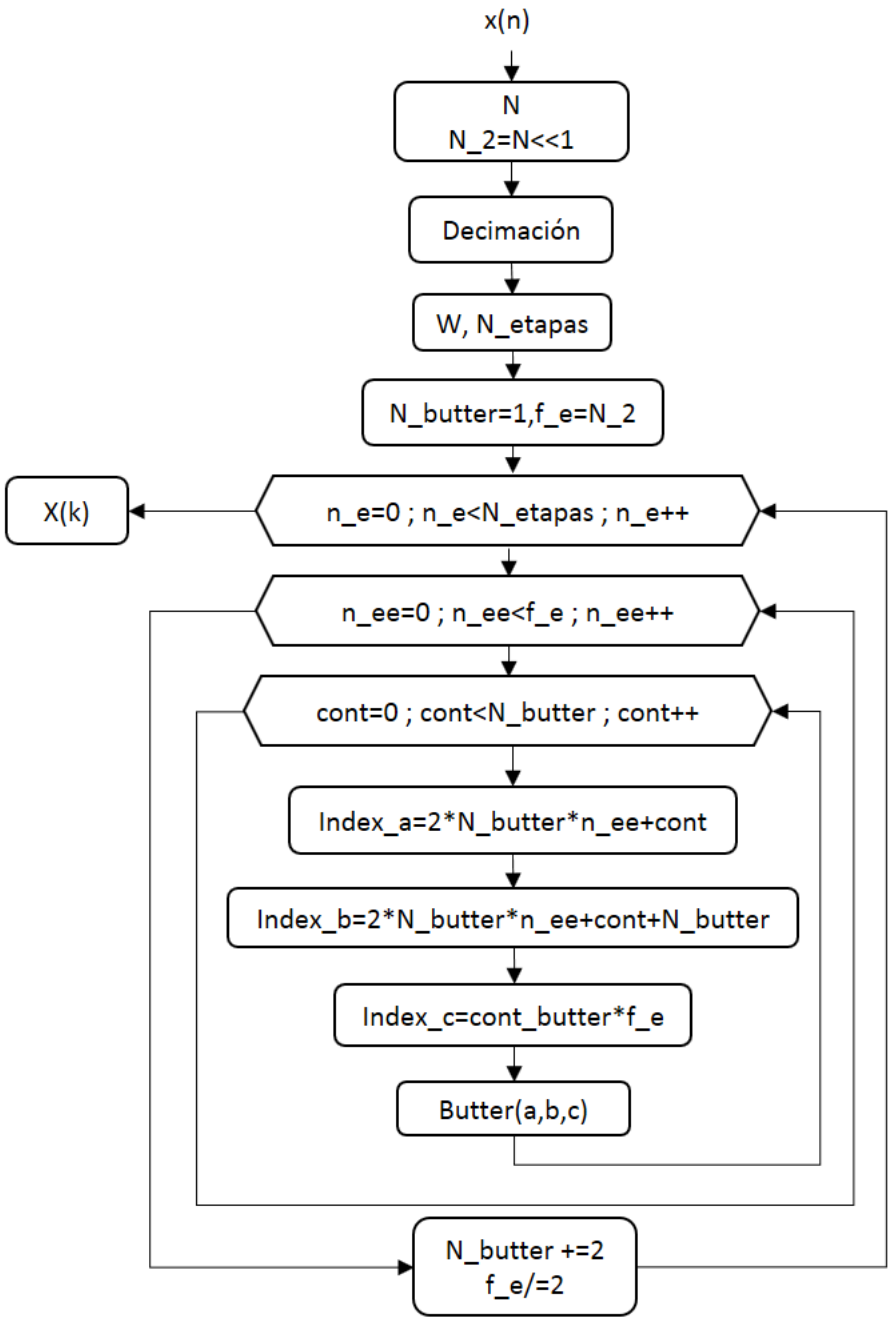


Figura 4.36: Diagrama de flujo para algoritmo FFT.

A continuación se muestra el código en lenguaje ensamblador del algoritmo FFT Radix 2 para una secuencia x(n) de longitud L=16 bits, sin embargo, dicho valor se puede extender, tomando en cuenta que las direcciones de inicio de x_k y Wr cambian dependiendo de la longitud de la señal, así como el número de etapas del algoritmo. Para un funcionamiento adecuado del presente programa es necesario ingresar los valores de los coeficientes por memoria en las localidades correspondientes (W_r < − Parte real de W, W_i < − parte imaginaria de W).

```
* Algoritmo FFT con decimación en el tiempo de una secuencia x(n)
* Los coeficientes se ingresan por memoria en Wr (parte real) y Wi
* (parte imaginaria). El número de corrimientos para ajustar el Qi se
* ingresa en corr. El número de etapas de la FFT (N_etapas) se ajusta
* dependidendo de la longitud de x(n), de la misma manera las
* direcciones de xk y wr. La señal resultado en el dominio de la
* frecuencia se almacena en xk (parte real) y xki (parte imaginaria).
*
        .global     _c_int00
DIRxk       .set    0x0C040; Dirección de inicio de xk parte real
DIRWr       .set    0x0C020; Dirección de inicio de coeficientes W
corr        .set    5       ; Número de corrimientos dependiento el qi
WDCR        .set    07029h ; Dirección para desactivar WatchDog
N           .set    32      ; Número de datos de entrada
N_2         .set    N/2     ; N/2
N_etapas    .set    5       ; Número de estapas en la FFT
*********Señal
xn          .space  16*N    ; Espacio reservado para señal de entrada
                            ; en el dominio del tiempo
Wr          .space  16*N_2 ; Espacio reservado para la parte real de
                            ; los coeficientes de la FFT
Wi          .space  16*N_2 ; Espacio reservado para la parte imaginaria
                            ; de los coeficientes de la FFT
xk          .space  N*16    ; Parte real de señal en frecuencia
xki         .space  N*16    ; Parte imaginaria de señal en frecuencia
f_e         .word   N/2     ; Límite del primer ciclo for
N_butter    .word   1       ; Indicador del número de operaciones
                            ; mariposas por etapa
*********índices para mariposa
;a_r        .long   0       ; índice 1 en operación de mariposa parte real
;b_r        .long   0       ; índice 2 en operación de mariposa
;c_r        .long   0       ; índice de coeficiente
*********contadores
cont1       .word   0
cont2       .word   0
********espacio resevado para operacion de complejos
complex1    .word   0       ; Espacio reservado para parte real de la
                            ; multiplicación de complejos
complex2    .word   0       ; Espacio reservado para la parte imaginaria
                            ; de la multiplicación de complejos
complex3    .word   0       ; Espacio extra reservado para la
```

```
                                ; multiplicación de complejos
********
RESULTADOr      .word  0        ; Espacio reservado para almacenamiento temporal
                                ; de operación de mariposa parte real
RESULTADOi      .word  0        ; espacio reservado para almacenamiento temporal
                                ; de operación de mariposa parte imaginaria
ffx             .space 16*N     ; Espacio reservado para almacenar el espectro
                                ; de la señal xn
        .text
_c_int00
******** Deshabilita WatchDog
        EALLOW
                MOVL    XAR1,#WDCR
                MOV     *XAR1,#0069h
        EDIS

        SETC    SXM             ; Modo extensión de signo
        SPM     #0              ; Sin corrimientos
        LC      DECIMACION      ; Subrutina que realiza la decimación
        MOV     AR0,#N_etapas-1 ; Carga el primer ciclo con el número de
                                ; etapas de la FFT
CICLO1
        MOVW    DP,#f_e         ; Apunta a la página en dirección de f_e
        MOV     AL,@f_e
        DEC     AL
        MOV     AR4,AL          ; Carga el número de ciclos de
                                ; operaciones mariposas
        ZAPA
        MOV     @cont1,AL
CICLO2
        MOV     AL,@N_butter    ; Carga el número de operaciones
                                ; mariposas por ciclo
        DEC     AL
        MOV     AR5,AL
        ZAPA
        MOV     @cont2,AL
CICLO3
***índices para butterfly índice a y b
        MOVL    XAR3,#N_butter
        MOVL    XAR7,#cont1
        MOV     T,*XAR3
        ZAPA
        MPY     P,T,*XAR7
        ADDL    ACC,P
        LSL     ACC,#1
        MOVL    XAR7,#cont2
        ADD     AL,*XAR7
        ADD     AL,#DIRxk
         ;MOV    @a_r,ACC        ; primer índice parte real
```

```
        MOVL    XAR1,ACC            ; APUNTA AL PRIMER DATO PARA OPERACIÓN
                                    ; MARIPOSA parte real
        ADD     AL,*XAR3
        ;MOV    @b_r,ACC            ; segundo índice parte real
        MOVL    XAR2,ACC            ; APUNTA AL SEGUNDO DATO DE OPERACIÓN
                                    ; MARIPOSA parte real
***índice para coerficientes
        MOVL    XAR3,#f_e
        MOV     T,*XAR7
        ZAPA
        MPY     P,T,*XAR3
        ADDL    ACC,P
        ADD     AL,#DIRWr
        ;MOV    @c_r,ACC
        MOVL    XAR6,ACC            ; APUNTA AL INICIO DE COEFICIENTES PARA
                                    ; OPERACION MARIPOSA parte real
        LC      OPBUTTERFLY         ; Salta a subrutina de operacion mariposa

        MOVW    DP,#f_e             ; Apunta a la página en dirección de f_e
        MOV     AL,@cont2           ; Incrementa contador 2
        INC     AL
        MOV     @cont2,AL
        BANZ    CICLO3,AR5--        ; Salto condicional (pregunta por
                                    ; fin de ciclo 3)
        MOV     AL,@cont1           ; Incrementa contador1 (contador del
                                    ; número de etapa)
        INC     AL
        MOV     @cont1,AL

        BANZ    CICLO2,AR4--        ; Salto condicional (pregunta por el
                                    ; fin del ciclo 2)
**************N_butter*=2
        MOV     AL,@N_butter
        LSL     AL,#1
        MOV     @N_butter,AL        ; Incrementa N_butter
**************f_e/=2
        MOV     AL,@f_e
        LSR     AL,#1
        MOV     @f_e,AL             ; Incrementa f_e

        BANZ    CICLO1,AR0--        ; Salto condicional (pregunta por el
                                    ; fin del ciclo 1)
        LC      ESPECTRO

fin     NOP
        LB      fin                 ; Ciclo infinito
.end

**************************SUBRUTINAS*****************************
```

```
**********DECIMACION
DECIMACION
        MOVL    XAR1,#xn                ; AR1=dir_inicio de xn
        MOV     AR0,#N_2                ; AR0=N/2 para la decimación
        MOVL    XAR2,#xk                ; AR2=dir_inicio de xnd
        MOV     AR3,#N-1                ; AR3 contador de ciclo BANZ

CICLO_DECIMA
        NOP     *,ARP1                  ; Selecciona a AR1 como registro
                                        ; en uso
        MOV     AL,*BR0++               ; AL=dato xn,AR1=AR1+BR(AR0)
        MOV     *XAR2++,AL              ; Mueve xn a xnd
        BANZ    CICLO_DECIMA,AR3--
        LRET
**********MULTIPLICACION DE COMPLEJOS
MULCOMPLEX      ;(*XAR2+*XAR3j)(*XAR6+XAR7j) = complex1+complex2j
        MOVW    DP,#complex1            ; DP apunta a la página donde está complex1

        ; Multiplicación de parte real
        MOV     T,*XAR2
        ZAPA
        MPY     P,T,*XAR6
        ADDL    ACC,P
        LSL     ACC,#corr
        MOV     @complex1,AH            ; Resultado parte real

        ; Multiplicación de parte real del primer dato  y
        ;       parte imaginaria del segundo dato
        ZAPA
        MPY     P,T,*XAR7
        ADDL    ACC,P
        LSL     ACC,#corr
        MOV     @complex2,AH            ; Resultado parte imaginaria

        ; Multiplicación de parte imaginaria de ambos datos
        MOV     T,*XAR3
        ZAPA
        MPY     P,T,*XAR7
        ADDL    ACC,P
        LSL     ACC,#corr
        MOV     @complex3,AH            ; Resultado para parte real
        MOV     AH,@complex1
        SUB     AH,@complex3            ; Resta de resultados para parte real
        MOV     @complex1,AH

        ; Multiplicación de parte imaginaria del primer dato
        ;           y parte real del segundo dato
        ZAPA
        MPY     P,T,*XAR6
```

```
        ADDL    ACC,P
        LSL     ACC,#corr
        ADD     AH, @complex2           ; Resultado parte imaginaria
        MOV     @complex2 ,AH           ; Suma de resultado para parte imaginaria
        LRET

**********Operación Mariposa**************
*                                        *
*   a-----------_--------_---------------A   *
*                -_    _-                *
*                  -_-                   *
*                 _- -_                  *
*               _-     -_                *
*   b-------w------*-1*-------------------B   *
*                                        *
******************************************
OPBUTTERFLY
        MOVL    ACC,XAR6                ; XAR6->Wr XAR7->Wi
        ADD     ACC,#N_2
        MOVL    XAR7,ACC

        MOVL    ACC,XAR2                ; XAR2->xk2r , XAR3->xk2i
        ADD     ACC,#N
        MOVL    XAR3,ACC
        LC      MULCOMPLEX              ; Salto a subrutina para
                                        ; multiplicación de complejos w*b

        MOVL    ACC,XAR1                ; XAR1->xkr  XAR6->xki
        ADD     ACC,#N
        MOVL    XAR6,ACC

        ;*****MARIPOSA*****
        MOVW    DP,#complex1            ; DP apunta a la página donde está complex1
        ZAPA
        MOV     AL,*XAR1
        ADD     AL, @complex1           ; AL=a_r+(b*w)_r
        MOV     @RESULTADOr,AL          ; Resultado_r=AL
        MOV     AL,*XAR6                ;
        ADD     AL, @complex2           ; AL=a_i+(b*w)_i
        MOV     @RESULTADOi,AL          ; resultado_i=AL

        MOV     AL,*XAR1
        SUB     AL, @complex1           ; AL= a_r-(b*w)_r
        MOV     *XAR2,AL                ; xk_r=AL
        MOV     AL,*XAR6
        SUB     AL, @complex2           ; AL=a_i-(b*w)_i
        MOV     *XAR3,AL                ; xk_i=AL

        MOV     AL,@RESULTADOr
```

```
        MOV     *XAR1,AL                ; xk_r=RESULTADOr
        MOV     AL,@RESULTADOi
        MOV     *XAR6,AL                ; xk_i=RESULTADOi
        ZAPA
        LRET

**********ESPECTRO
ESPECTRO
        MOVL    XAR1,#xk
        MOVL    XAR2,#xki
        MOVL    XAR3,#ffx
        MOV     AR4,#N-1
DEP
        I16TOF32 R4H,*XAR1++
        NOP
        NOP
        MPYF32  R0H,R4H,R4H             ; Real^2
        NOP
        NOP

        I16TOF32 R3H,*XAR2++
        NOP
        NOP
        MPYF32  R2H,R3H,R3H             ; Img^2
        NOP
        NOP

        ADDF32  R0H,R2H,R0H             ; R0H = Real^2 + Img^2
        NOP
        NOP
        NOP
        SQRTF32 R4H,R0H                 ; R4H = sqrt(X), magnitud del espectro
        NOP
        NOP
        NOP
        NOP

        F32TOI16 R1H,R4H
        NOP
        NOP
        MOV32   ACC,R1H
        MOV     *XAR3++,AL
        BANZ    DEP,AR4--
        LRET
```

Se puede hacer notar que con el DSP TMS329F28377s las operaciones entre números complejos y la operación mariposa a L=16 bits se pueden optimizar utilizando instrucciones y registros de la unidad VCU [1].

Implementación de la FFT en 32 bits

Para realizar la implementación de la FFT con palabras de 32 bits se utilizó la misma lógica utilizada para el algoritmo con palabras de 16 bits, tomando en cuenta que una palabra de 32 bits requiere dos localidades de memoria en el DSP de 16 bits. Esto significa que las reservas de memoria para la señal de entrada, señal resultado parte real e imaginaria, coeficientes (parte real e imaginaria) y espacios utilizados para almacenamiento temporal en la multiplicación de complejos y resultados de la operación mariposa, requieren el doble de la memoria para un correcto funcionamiento.

En las operaciones destinadas a la lógica del algoritmo se pueden seguir utilizando las palabras de 16 bits, sin embargo, hay que tener en cuenta que las direcciones a las que se apunta para efectuar la operación mariposa son de palabra doble, de tal manera que se debe multiplicar por dos el valor del contador antes de sumarle la dirección de *xk* y *wr*. Esto se logra realizando un corrimiento a la izquierda al valor del contador.

El siguiente código en lenguaje ensamblador muestra la implementación de la transformada rápida de Fourier utilizando el algoritmo Radix 2 para señales con longitud de palabra de 32 bits. Para un funcionamiento correcto es necesario ingresar los datos de la señal que se desea obtener la respuesta en frecuencia en las localidades de memoria de *xn*, los coeficientes en *Wr* para la parte real y *Wi* para la parte imaginaria, y el resultado se puede obtener en los espacios de memoria destinados para *xk* y *xki* respectivamente. El código mostrado es para una señal con longitud de L=32, sin embargo, dicho valor se puede extender tomando en cuenta que es necesario ingresar las direcciones de inicio de *xk* y *Wr* dentro del código.

```
*
* Algoritmo FFT con decimación en el tiempo de una secuencia x(n)
* con palabras de 32 bits de longitud. El número de corrimientos para
* ajustar el Qi se ingresa en corr. Las direcciones de xk y wr dependen
* de la longitud de x(n).

        .global     _c_int00
DIRxk   .set    0x0C080         ; Dirección de inicio de xk parte real
DIRWr   .set    0x0C040         ; Dirección de inicio de coeficientes W
corr    .set    5               ; Número de corrimientos dependiento el Qi
WDCR    .set    07029h          ; Dirección para desactivar WatchDog
N               .set    32      ; Número de datos de entrada
N_2             .set    N/2     ; N/2
N2              .set    N*2     ; N*2
N_etapas        .set    5       ; Número de estapas en la FFT
*********Señal
;xn             .long   1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
xn              .space  32*N    ; Espacio reservado para señal de entrada
                                ; en el dominio del tiempo
Wr              .space  32*N_2  ; Espacio reservado para la parte real de
```

```
                                ; los coeficientes de la FFT
Wi              .space 32*N_2   ; Espacio reservado para la parte imaginaria
                                ; de los coeficientes de la FFT
xk              .space 32*N     ; Parte real de señal en frecuencia
xki             .space 32*N     ; Parte imaginaria de señal en frecuencia
f_e             .word  N/2      ; Límite del primer ciclo for
N_butter        .word  1        ; Indicador del número de operaciones
                                ; mariposas por etapa
*********índices para mariposa
;a_r            .long  0        ; índice 1 en operación de mariposa parte real
;b_r            .long  0        ; índice 2 en operación de mariposa
;c_r            .long  0        ; índice de coeficiente
*********contadores
cont1           .word  0
cont2           .word  0
********espacio resevado para operacion de complejos
complex1        .long  0        ; Espacio reservado para parte real de la
                                ; multiplicación de complejos
complex2        .long  0        ; Espacio reservado para la parte imaginaria
                                ; de la multiplicación de complejos
complex3        .long  0        ; Espacio extra reservado para la
                                ; multiplicación de complejos
********
RESULTADOr      .long  0        ; Espacio reservado para almacenamiento temporal
                                ; de operación de mariposa parte real
RESULTADOi      .long  0        ; espacio reservado para almacenamiento temporal
                                ; de operación de mariposa parte imaginaria
ffx             .space 32*N     ; Espacio reservado para el espectrode la señal

        .text
_c_int00
******** Deshabilita WatchDog
        EALLOW
                MOVL    XAR1,#WDCR
                MOV     *XAR1,#0069h
        EDIS

        SETC    SXM                     ; Modo extensión de signo
        SPM     #0                      ; Sin corrimientos
        LC      DECIMACION              ; Subrutina que realiza la decimación
        MOV     AR0,#N_etapas-1         ; Carga el primer ciclo con el número de
                                        ; etapas de la fft
CICLO1
        MOVW    DP,#f_e                 ; Apunta a la página en dirección de f_e
        MOV     AL, @f_e
        DEC     AL
        MOV     AR4,AL                  ; Carga el número de ciclos de
                                        ; operaciones mariposas
```

```
        ZAPA
        MOV     @cont1,AL


CICLO2
        MOV     AL,@N_butter        ; Carga el número de operaciones
                                    ; mariposas por ciclo
        DEC     AL
        MOV     AR5,AL
        ZAPA
        MOV     @cont2,AL


CICLO3
***índices para butterfly índice a y b
        MOVL    XAR3,#N_butter
        MOVL    XAR7,#cont1
        MOV     T,*XAR3
        ZAPA
        MPY     ACC,T,*XAR7
        LSL     ACC,#1
        MOVL    XAR7,#cont2
        ADD     AL,*XAR7
        LSL     ACC,#1
        ADD     AL,#DIRxk
        ;MOV    @a_r,ACC            ; primer índice parte real
        MOVL    XAR1,ACC            ; APUNTA AL PRIMER DATO PARA OPERACIÓN
                                    ; MARIPOSA parte real
        ZAPA
        MOV     AL,*XAR3
        LSL     ACC,#1
        ADDL    ACC,XAR1
        ;MOV    @b_r,ACC            ; segundo índice parte real
        MOVL    XAR2,ACC            ; APUNTA AL SEGUNDO DATO DE OPERACIÓN
                                    ; MARIPOSA parte real
***índice para coerficientes
        MOVL    XAR3,#f_e
        MOV     T,*XAR7
        ZAPA
        MPY     ACC,T,*XAR3
        LSL             ACC,#1
        ADD     AL,#DIRWr
        ;MOV    @c_r,ACC
        MOVL    XAR6,ACC            ; APUNTA AL INICIO DE COEFICIENTES PARA
                                    ; OPERACION MARIPOSA parte real
        LC      OPBUTTERFLY         ; Salta a subrutina de operacion mariposa

        MOVW    DP,#f_e             ; Apunta a la página en dirección de f_e
        MOV     AL,@cont2           ; Incrementa contador 2
```

```
        INC     AL
        MOV     @cont2,AL
        BANZ    CICLO3,AR5-          ; Salto condicional (pregunta por
                                     ; fin de ciclo 3)
        MOV     AL,@cont1            ; Incrementa contador1 (contador del
                                     ; número de etapa)
        INC     AL
        MOV     @cont1,AL

        BANZ    CICLO2,AR4-          ; Salto condicional (pregunta por el
                                     ; fin del ciclo 2)
**************N_butter*=2
        MOV     AL,@N_butter
        LSL     AL,#1
        MOV     @N_butter,AL         ; Incrementa N_butter
**************f_e/=2
        MOV     AL,@f_e
        LSR     AL,#1
        MOV     @f_e,AL              ; Incrementa f_e

        BANZ    CICLO1,AR0-          ; Salto condicional (pregunta por el
                                     ; fin del ciclo 1)
        LC ESPECTRO

fin     NOP
        LB      fin             ; Ciclo infinito
.end

****************************SUBRUTINAS*****************************

**********DECIMACION para datos de 32bits
DECIMACION
        MOVL    XAR1,#xn             ; AR1=dir_inicio de xn
        MOV             AR0,#N       ; AR0=N/2 para la decimación
        MOVL    XAR2,#xk             ; AR2=dir_inicio de xnd
        MOV     AR3,#N-1             ; AR3 contador de ciclo BANZ

CICLO_DECIMA

        NOP     *,ARP1               ; Selecciona a AR1 como registro
                                     ; en uso
        MOVL    ACC,*BR0++           ; AL=dato xn,AR1=AR1+BR(AR0)
        MOVL    *XAR2++,ACC          ; Mueve xn a xnd
        BANZ    CICLO_DECIMA,AR3-
        LRET
```

```
**********MULTIPLICACION DE COMPLEJOS
MULCOMPLEX      ;(*XAR2+*XAR3j)(*XAR6+XAR7j) = complex1+complex2j

        MOVW    DP,#complex1            ; DP apunta a la página donde está complex1

        ; Multiplicación de parte real
        MOVL    XT,*XAR2
        ZAPA
        QMPYL   ACC,XT,*XAR6
        LSL     ACC,#corr
        MOVL    @complex1,ACC           ; Resultado parte real

        ; Multiplicación de parte real del primer dato  y
        ;       parte imaginaria del segundo dato
        ZAPA
        QMPYL   ACC,XT,*XAR7
        LSL     ACC,#corr
        MOVL    @complex2,ACC           ; Resultado parte imaginaria

        ; Multiplicación de parte imaginaria de ambos datos
        MOVL    XT,*XAR3
        ZAPA
        QMPYL   ACC,XT,*XAR7
        LSL     ACC,#corr
        MOVL    @complex3,ACC           ; Resultado para parte real
        MOVL    ACC,@complex1
        SUBL    ACC,@complex3           ; Resta de resultados para parte real
        MOVL    @complex1,ACC

        ; Multiplicación de parte imaginaria del primer dato
        ;             y parte real sel segundo
        ZAPA
        QMPYL   ACC,XT,*XAR6
        LSL     ACC,#corr
        ADDL    ACC,@complex2           ; Resultado parte imaginaria
        MOVL    @complex2,ACC           ; Suma de resultado para parte imaginaria
        LRET

**********Operación Mariposa**************
*                                         *
*  a-----------_--------_-----------------A   *
*               -_    _-                  *
*                 -_-                     *
*                _- -_                    *
*              _-     -_                  *
*  b------*W*-----*-1*-------------------B   *
*                                         *
*******************************************
```

```
OPBUTTERFLY

        MOVL    ACC,XAR6            ; XAR6->Wr XAR7->Wi
        ADD     ACC,#N              ; Datos de 32 bits
        MOVL    XAR7,ACC

        MOVL    ACC,XAR2            ; XAR2->xk2r, XAR3->xk2i
        ADD     ACC,#N2             ; Datos de 32 bits
        MOVL    XAR3,ACC
        LC      MULCOMPLEX          ; Salto a subrutina para
                                    ; multiplicación de complejos w*b

        MOVL    ACC,XAR1            ; XAR1->xkr  XAR6->xki
        ADD     ACC,#N2             ; Datos de 32 bits
        MOVL    XAR6,ACC

        ;*****MARIPOSA*****
        MOVW    DP,#complex1        ; DP apunta a la página donde está complex1
        ZAPA
        MOVL    ACC,*XAR1
        ADDL    ACC,@complex1       ; AL=a_r+(b*w)_r
        MOVL    @RESULTADOr,ACC     ; Resultado_r=AL
        MOVL    ACC,*XAR6           ;
        ADDL    ACC,@complex2       ; AL=a_i+(b*w)_i
        MOVL    @RESULTADOi,ACC     ; resultado_i=AL

        MOVL    ACC,*XAR1
        SUBL    ACC,@complex1       ; AL= a_r-(b*w)_r
        MOVL    *XAR2,ACC           ; xk_r=AL
        MOVL    ACC,*XAR6
        SUBL    ACC,@complex2       ; AL=a_i-(b*w)_i
        MOVL    *XAR3,ACC           ; xk_i=AL

        MOVL    ACC,@RESULTADOr
        MOVL    *XAR1,ACC           ; xk_r=RESULTADOr
        MOVL    ACC,@RESULTADOi
        MOVL    *XAR6,ACC           ; xk_i=RESULTADOi
        ZAPA
        LRET

**********ESPECTRO
ESPECTRO

        MOVL    XAR1,#xk
        MOVL    XAR2,#xki
        MOVL    XAR3,#ffx
        MOV     AR4,#N-1
```

```
DEP
        I32TOF32  R4H,*XAR1++
        NOP
        NOP
        MPYF32    R0H,R4H,R4H          ;  Real^2
        NOP
        NOP

        I32TOF32  R3H,*XAR2++
        NOP
        NOP
        MPYF32    R2H,R3H,R3H          ;   Img^2
        NOP
        NOP

        ADDF32    R0H,R2H,R0H          ;  R0H = Real^2 + Img^2
        NOP
        NOP

        SQRTF32   R4H,R0H              ;  R4H = sqrt(X), magnitud del espectro
        NOP
        NOP
        NOP
        NOP

        F32TOI32  R1H,R4H
        NOP
        NOP
        MOV32     ACC,R1H
        MOVL      *XAR3++,ACC
        BANZ      DEP,AR4--
        LRET
```

Análisis de resultados de la FFT Radix 2

Las pruebas realizadas a los algoritmos de la transformada rápida de Fourier implementados en 16 y 32 bits se ejecutaron con señales conocidas como es el caso de una señal senoidal y una señal un pulso como las que se muestran en las Figuras 4.37 y 4.38 respectivamente. Cabe resaltar que entre mayor es la longitud de la señal de entrada x(n), menor será la exactitud del resultado puesto que se se requiere mayor número de bits para representar la parte entera en la longitud de la palabra, es por ello que en la presente sección se realizarán comparaciones de exactitud del error con diferentes longitudes de la señal para observar como se degrada la exactitud de la respuesta dependiendo del Qi utilizado.

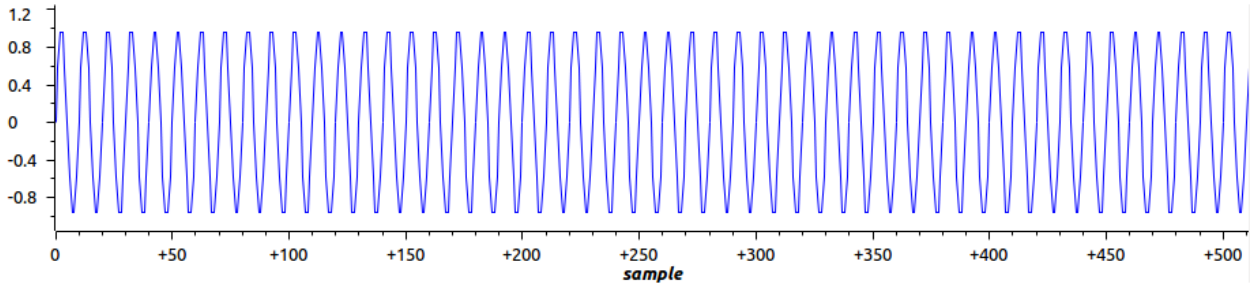


Figura 4.37: Señal senoidal de N=256 para prueba en la FFT.

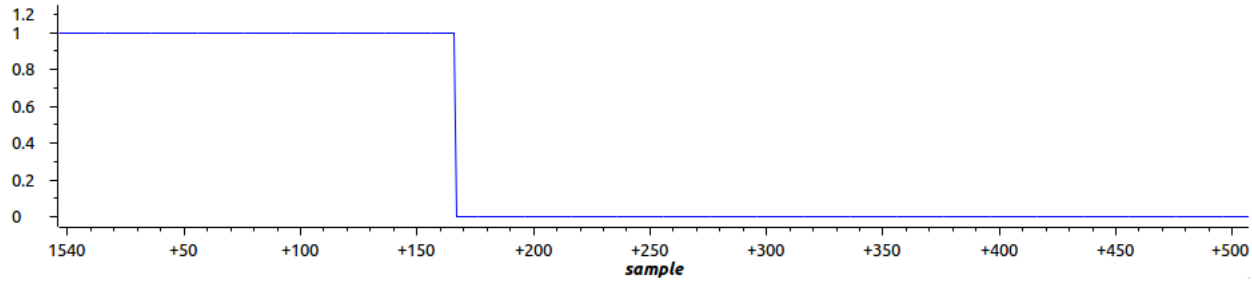
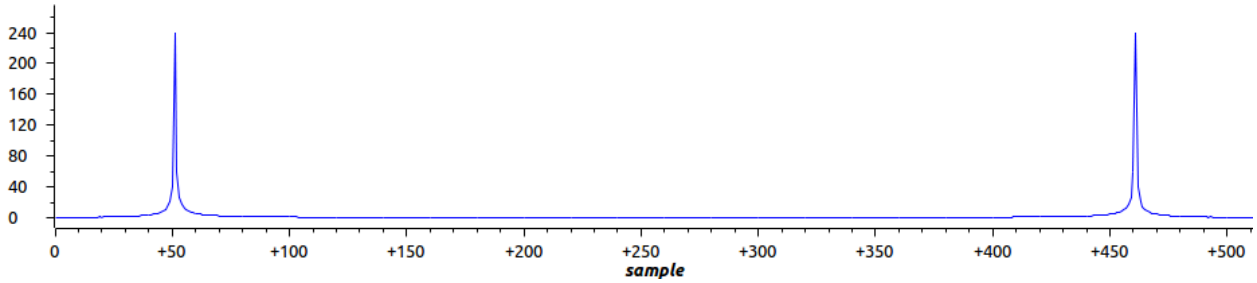


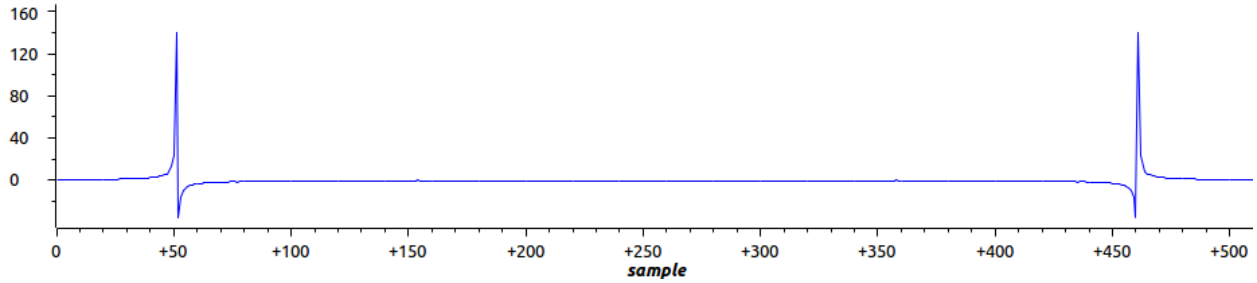
Figura 4.38: Señal pulso de N=256 para prueba en la FFT.

Los resultados obtenidos de la FFT implementada en 16 y 32 bits fueron comparados con los resultados de la FFT en Octave, para obtener el error promedio de cada una de las series. Además, las señales de prueba se utilizaron con diferentes longitudes N=32, 64, 128, 256 y 512.

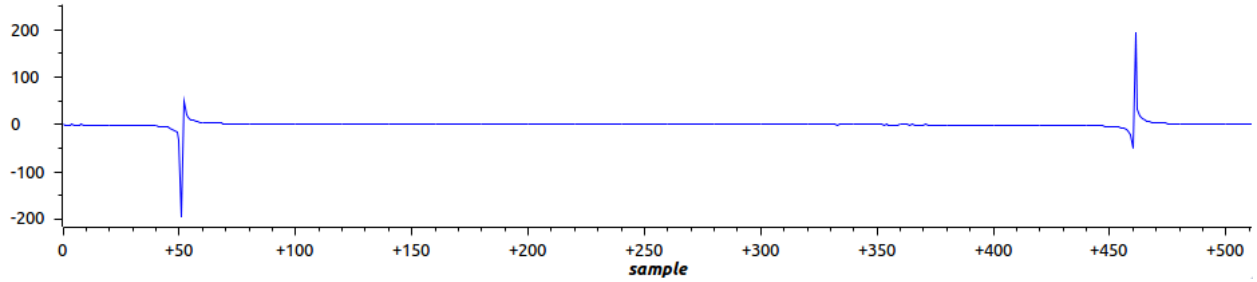
En la Figura 4.39 se muestran tres gráficas que representan la respuesta en frecuencia de la señal de entrada de la Figura 4.37 de longitud N=512 con longitud de palabra de 32 bits, dichas gráficas representan la densidad espectral de potencia y las partes real e imaginaria de la FFT.



(a) Magnitud del espectro de una señal senoidal de longitud N=512 de longitud de palabra de 32 bits.



(b) Parte real de la respuesta en frecuencia de una señal senoidal de longitud N=512 de longitud de palabra de 32 bits.



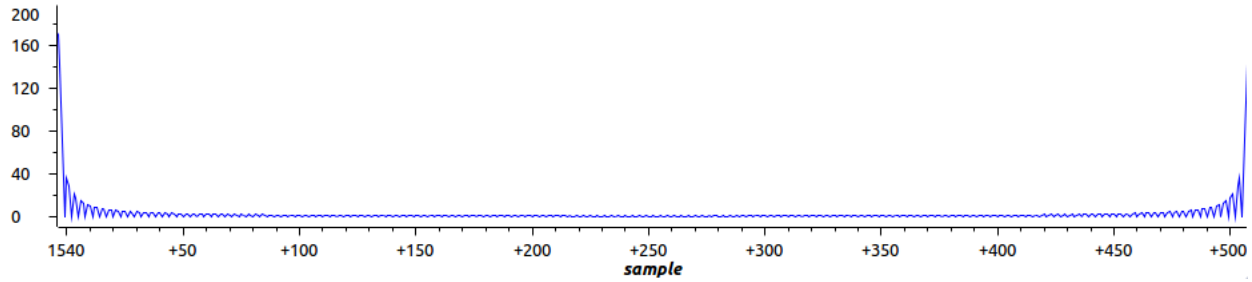
(c) Parte imaginaria de la respuesta en frecuencia de una señal senoidal de longitud N=512 de longitud de palabra de 32 bits.

Figura 4.39: Gráficas de señales de prueba para el algoritmo FFT.

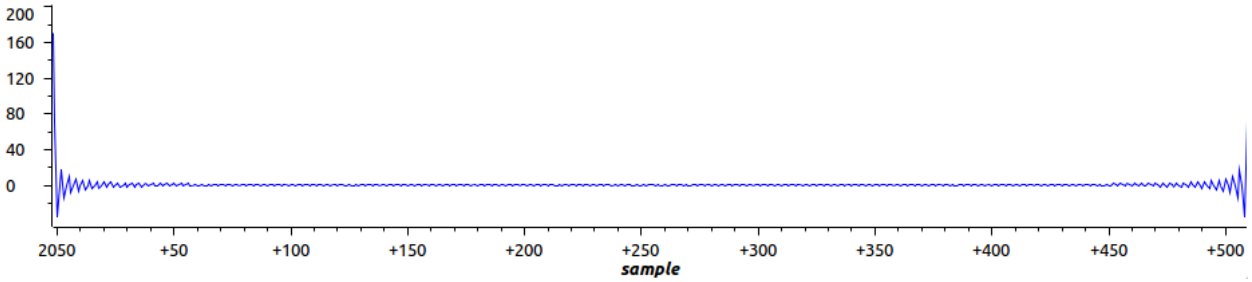
Es importante mencionar que el valor del Qi de los datos depende del máximo valor entero que se obtendrá después de ejecutar las respectivas multiplicaciones y sumas del algoritmo. A partir de la implementación de la FFT Radix 2, se notó que entre mayor es la longitud de la señal x(n), el valor del Qi en la longitud de palabra requiere ser menor para evitar el sobreflujo. Se encontró que para una señal de longitud L=16, la mejor precisión numérica es con Qi=11 para palabras de 16 bits y Qi=27 para palabras de 32 bits, para N=32 la precisión está dada por Qi=10 para palabras de 16 bits y Qi=26 para palabras de 32 bits. Para una señal de longitud N=512 la precisión numérica está dada por Qi=6 para 16 bits y Qi=23 para 32 bits. De tal manera que se encontró que el número de corrimientos realizados

después de efectuar las multiplicaciones de números complejos es igual al número de etapas a ejecutar en el algoritmo de la FFT.

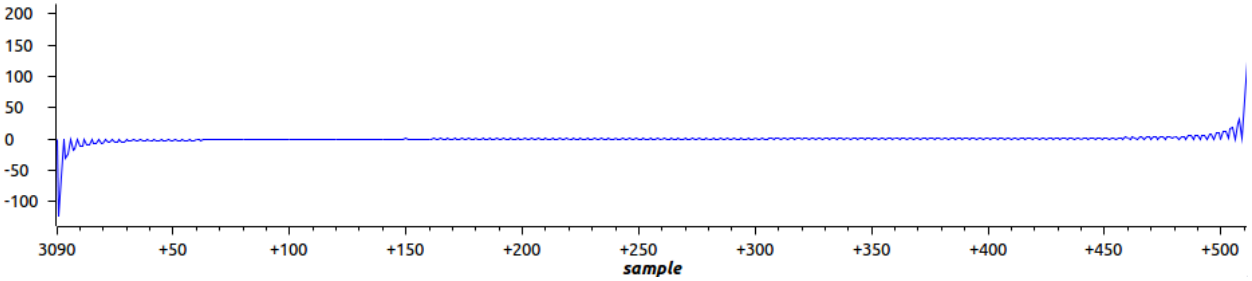
Por otro lado, las gráficas de la respuesta en frecuencia de la señal pulso que se muestra en la Figura 4.38 se pueden observar en la Figura 4.40, misma que contiene las gráficas de magnitud, parte real y parte imaginaria de la FFT. Las gráficas mostradas se realizaron con una señal de entrada x(n) de longitud $N = 512$ con longitud de palabra de 32 bits.



(a) Magnitud del espectro de una señal pulso de longitud N=512 de longitud de palabra de 32 bits.



(b) Parte real de la respuesta en frecuencia de una señal pulso de longitud N=512 de longitud de palabra de 32 bits.



(c) Parte imaginaria de la respuesta en frecuencia de una señal pulso de longitud N=512 de longitud de palabra de 32 bits.

Figura 4.40: Gráficas de señales de prueba para el algoritmo FFT.

Finalmente, en la Tabla 4.13 se muestra una comparativa del error absoluto promedio

de los respectivos resultados del algoritmo FFT Radix 2 con relación en los obtenidos del algoritmo FFT de la plataforma de programación Octave. En dicha Tabla la comparativa se realiza dependiendo del tipo de señal (senoidal y señal pulso) y la longitud de la misma, para N=16, 32, 64, 128 y 256.

Tabla 4.13: Comparación del error absoluto promedio en función de la longitud de la señal y el Qi.

	Señal senoidal		**Señal pulso**	
	Error absoluto promedio. Datos de 16 bits	**Error absoluto promedio. Datos de 32 bits**	**Error absoluto promedio. Datos de 16 bits**	**Error absoluto promedio. Datos de 32 bits**
N=32	$9.4404e^{-5}$ $Q_i = 11$	$2.8006e^{-8}$ $Q_i = 27$	$2.7676e^{-4}$ $Q_i = 11$	$1.2148e^{-7}$ $Q_i = 27$
N=64	$15.599e^{-4}$ $Q_i = 10$	$3.2799e^{-8}$ $Q_i = 26$	$11.69e{-}4$ $Q_i = 10$	$1.9058e^{-6}$ $Q_i = 26$
N=128	$4.9816e^{-4}$ $Q_i = 9$	$2.7835e^{-6}$ $Q_i = 25$	$30.712e^{-4}$ $Q_i = 9$	$4.9715e^{-6}$ $Q_i = 25$
N=256	$3.5674e^{-5}$ $Q_i = 8$	$1.1827e^{-5}$ $Q_i = 24$	$13.108e^{-3}$ $Q_i = 8$	$3.3471e^{-5}$ $Q_i = 24$
N=512	$9.0872e^{-4}$ $Q_i = 7$	$9.4797e^{-5}$ $Q_i = 23$	$21.705e^{-3}$ $Q_i = 7$	$1.6927e^{-4}$ $Q_i = 23$

Resumen

En el área del Procesamiento Digital de Señales es importante la ejecución de los algoritmos y que éstos tengan una respuesta en tiempo real utilizando un hardware dedicado, de tal manera que en el presente capítulo se mostró la teoría básica e implementación de diversos algoritmos básicos del procesamiento digital de señales en el DSP TMS320F28377. Las operaciones y algoritmos implementados en el presente capítulo son la convolución, correlación entre señales, filtros FIR e IIR, osciladores digitales y algoritmos de la transformada discreta de Fourier, tales como el algoritmos de Goertzel y la transformada rápida de Fourier (FFT). Se realizó un análisis en la respuesta y en la precisión numérica de los algoritmos implementados, utilizando los dos tipos de formato numérico (punto fijo y flotante) y comparando los resultados con los obtenidos en Octave, el cual es de doble precisión numérica.

Capítulo 5

Manejo de periféricos de la familia TMS320F2837xS

Una parte clave del procesamiento digital de señales en aplicaciones en tiempo real, es su proceso de adquisición, la cual se lleva a cabo por sensores y/o transductores conectados como entradas de datos a circuitos electrónicos que acoplan, reducen ruido o amplifican, entre otras etapas de pre-procesamiento, para poder aplicar un análisis digital y obtener información de interés. A su vez, después de aplicar un procesamiento mediante un circuito digital, como los DSP's, los resultados obtenidos se almacenan o son utilizados por otros dispositivos para llevar a cabo diferentes tareas.

En estas dos etapas de entrada y salida de información a un procesador, los canales de comunicación asumen el papel de interfaz para que las máquinas digitales trabajen con el mundo real, por ello algunas familias de DSP's incorporan en su arquitectura, módulos de diferentes tipos de periféricos como puertos seriales, de conversión analógica a digital, entre otros. En este capítulo se expone el uso de los módulos o periféricos más representativos del TMS320F28377S que permitirán realizar aplicaciones en tiempo real. Se hace notar que en este capítulo se utilizarán programas en lenguaje ensamblador, lenguaje C y las bibliotecas de *ControlSuite* de *Texas Instruments* para facilitar la configuración y manejo de periféricos.

5.1. Entradas y salidas de propósito general

El DSP C28x maneja puertos de entrada/salida de propósito general (GPIO), que le permite seleccionar y configurar sus terminales externos GPIO, cada uno de estos pines se puede utilizar para diferentes funciones [1]. En las familias F2837xS los pines GPIO se agrupan en diferentes puertos:

Puerto A	**Puerto B**	**Puerto C**	**Puerto D**	**Puerto E**	**Puerto F**
GPIO0 - GPIO31	GPIO32 - GPIO63	GPIO64 - GPIO95	GPIO96 - GPIO127	GPIO128 - GPIO159	GPIO160 - GPIO168

5.1.1. Configuración GPIO

Los GPIO tienen conectados diferentes periféricos los cuales pueden ser seleccionados mediante los siguientes registros:

- *GPyGMUX1-2*
- *GPyMUX1-2*[1]

La configuración de cada GPIO se encuentra detallada en la hoja de especificaciones [4], en la sección *Signal Descriptions*. En la Tabla 5.1 se muestran las diferentes configuraciones para el pin GPIO11[2].

Para configurar los GPIO como entrada o salida (I/O), se emplean los registros *GPyDIR*, escribiendo en el bit del GPIO correspondiente.

- 0: Para utilizar la terminal como entrada.
- 1: Para utilizar la terminal como salida.

Si el GPIO se configura como entrada, su valor se puede leer empleando el registro *GPyDAT*. En caso de emplear el pin GPIO como salida, se emplean los siguientes registros para modificar el valor del GPIO:

- *GPyDAT*: escribe un valor al GPIO.
- *GPySET*: escribe un "1" en el GPIO.

[1] "y" corresponde a los puertos A, B, C, D ,E y F, donde cada puerto contiene sus registros de configuración y escritura a sus correspondientes pines GPIO

[2] Si se emplea una configuración "Reservada" el estado del GPIO queda indefinido.

Tabla 5.1: Posibles configuraciones del pin GPIO11.

GPAGMUX1	**GPAMUX**	**Función**
00	00	GPIO
00	01	EPWB6B
00	10	SCIRXDB
00	11	OUTPUTXBAR7
01	00	GPIO
01	01	EQEP1B
01	10	SCIRXDB
10	00	GPIO
11	00	GPIO
11	11	UPP-START
Otras		Reservadas

- *GPyCLEAR*: escribe un "0" en el GPIO.
- *GPyTOGGLE*: cambia el estado del GPIO.

En la Figura 5.1 se muestra el diagrama del módulo GPIO del DSP.

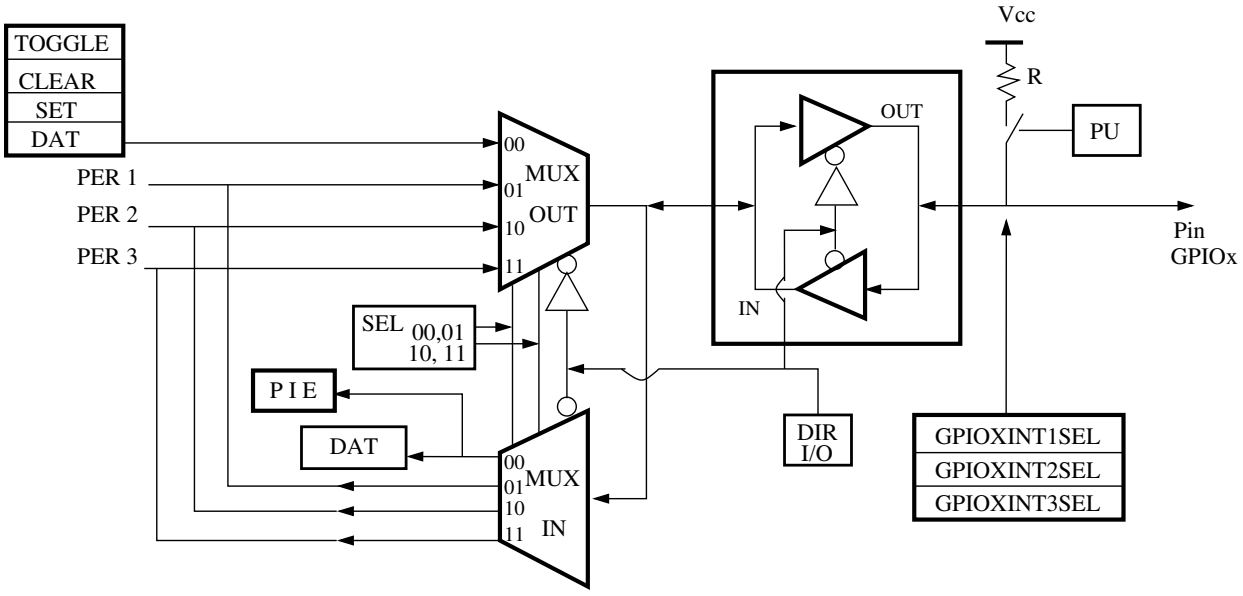


Figura 5.1: Módulo GPIO

5.1.2. Utilización de puertos GPIO

Para ejemplificar el uso y manejo del puerto en la tarjeta de desarrollo, a continuación se muestran dos programas que describen la configuración de entrada y salida para dos terminales independientes, programados en en lenguaje ensamblador y C.

Salida digital en lenguaje ensamblador

El programa que se presenta a continuación muestra la configuración del pin GPIO12 como terminal de salida y su estado binario (1/0) se cambia empleando el registro GPATOGGLE.

```
*       Salida digital GPIO12
*
* En este programa se configura la terminal 12 del
* modulo GPIOA como salida y se conmuta el estado
* utilzando el registro TOGGLE de este puerto.
*
        .global     _c_int00
* Direcciones de los registros
DIR_SP  .set    0400h   ; Dirección del registro
                        ; StackPointer
WDCR    .set    07029h  ; Dirección del registro de
                        ; control del WatchDog
GPADIR  .set    07C0Ah  ; Dirección del registro GPADIR
GPATOG  .set    07F06h  ; Dirección del registro GPATOGGLE
MASK12  .set    01000h  ; Máscara para utilizar el GPIO12
        .text
_c_int00
        SETC    INTM            ; Deshabilita INTERRUPCIONES
                                ; mascarables
        MOV     SP,#DIR_SP      ; Mueve el stack pointer a la
                                ; direccion DIR_SP
* Configuración en registros protegidos
        EALLOW                  ; Habilita escritura en registros
                                ; protegidos
        MOVL    XAR1,#WDCR      ; Direccionamiento indirecto al
                                ; registro del WatchDog
        MOV     *XAR1,#0068h    ; Escribe en el registro
                                ; WatchDog para deshailitarlo
        MOVL    XAR1,#GPADIR    ; Direccionamiento al registro de
                                ; dirección del puerto GPIO A
        MOV     *XAR1,#MASK12   ; Escribe el número "mascara"
                                ; que habilita el uso del GPIO12
        EDIS                    ; Deshabilita escritura en registros
                                ; protegidos
* Escritura del estado del GPIO12
```

```
        MOVL    XAR1,#GPATOG    ; Direccionamiento al registro
                                ; TOGGLE del puerto GPA
LED                             ; Etiqueta para realizar salto
                                ; para repetir el bloque
                                ; de codigo delimitado
        MOV     *XAR1,#MASK12   ; Escribe la máscara en el
                                ; registro Toggle a GPIO12
        B       LED,UNC         ; Salta a la etiqueta LED sin
                                ; evaluar alguna condición
        .end
```

Para poder ver la conmutación del estado del puerto GPIO12 de la tarjeta, se puede ejecutar el programa línea por línea, o colocar un breakpoint en la instrucción:

```
MOV *XAR1,#MASK12
```

para ver el encendido y apagado del LED conectado a esa terminal del DSP en la tarjeta de desarrollo.

Salida digital en lenguaje C

Utilizando las bibliotecas de ControlSuite, es posible realiza el código anterior en lenguaje C, como se muestra a continuación:

```
/* Configura el GPIO12 como salida
* y realiza un cambio de estado
* (TOGGLE) al pin GPIO
*/
// DSP28x Headerfile
#include "F28x_Project.h"

int main(void) {
        InitSysCtrl();
        DINT;

        EALLOW;
        GpioCtrlRegs.GPAMUX1.bit.GPIO12 = 0;
        GpioCtrlRegs.GPADIR.bit.GPIO12 =  1;
        EDIS;

        while(1){
                // Toggle GPIO12
                GpioDataRegs.GPATOGGLE.bit.GPIO12 = 1;
        }
        return 0;
}
```

Nuevamente, para poder ver la conmutación de la terminal GPIO12, se sugiere colocar un breakpoint en la instrucción:

```
GpioDataRegs.GPATOGGLE.bit.GPIO12 = 1;
```

Entrada y salida digital en lenguaje Ensamblador

El siguiente ejemplo configura el GPIO2 como entrada, el cual se emplea como interrupción externa, utilizando la interrupción XINT1 por medio del grupo de interrupciones del PIE 1.4. Cuando se detecta un flanco de bajada por medio del GPIO2, entra la interrupción y cambia el estado en el GPIO12.

El puerto GPIO12 se habilita por medio del registro GPADAT colocando un "1" en el bit correspondiente al GPIO que se desea utilizar como salida y un "0" al bit que se desea configurar como entrada. Posteriormente, es importante habilitar las interrupciones, para ello se utilizan los registros PIECRT, PIEACK1 y PIEIER1, donde PIECRT habilita las interrupciones del PIE, PIEACK1 habilita el reconocimiento de la interrupción 1 y el PIEIER1 habilita el tipo de interrupción dentro del grupo uno, para este caso se requiere la interrupción XINT1 que está en la interrupción 1.4.

Es necesario ligar la subrutina de interrupción (fragmento de código que va a ejecutar el programa cuando se active la interrupción), para esto, se requiere agregar la dirección del código de ejecución de la interrupción al registro XINT1. Por último, se debe ligar el puerto GPIO2 a la interrupción XINT1 para sea reconocida cuando se active el puerto, esto se logra por medio del registro XBARINPUT4, mismo que está relacionado a la interrupción XINT1 activando el GPIOx como interrupción dependiendo del valor que se ingrese en el registro, es decir, 0h para el GPIO0, 1h para el GPIO1, 2h para el GPIO2, etc.

```
*
* Este programa configura una interrupción externa (XINT1) que
* se activa por medio del GPIO2 cambiando el estado del GPIO12
* configurado como salida
*

                .global     _c_int00

DIR_SP          .set    0400h   ; Dirección del Stack Point
WDCR            .set    07029h  ; Dirección del reg. WatchDog
UNO             .set    001h    ; Grupo de interrupciones

* Direcciones de configuración para el puerto GPA
GPA_DIR         .set    07C0Ah  ; Dirección registro GPADIR
GPA_SET         .set    07F02h  ; Dirección registro GPASET
GPA_DAT         .set    07F00h  ; Dirección registro GPADAT
```

```
GPA_CLEAR     .set    07F04h ; Dirección registro GPACLEAR
GPA_TOGGLE    .set    07F06h ; Dirección registro GPATOGGLE

* Direcciones de configuración para el PIE
DIR_PIECRT    .set    0x0CE0 ; Control de PIE, 01 habilita INT_PIE
DIR_PIEACK    .set    0x0CE1 ; Reconocimiento de INT_PIE
DIR_PIEIER1   .set    0x0CE2

DIR_INTXINT1  .set    0x0D46 ; Dirección del vector de INTX1que ésta
DIR_XINT1CR   .set    0x07070; Dirección del registro XINT1CR
XBARINPUT4    .set    0x07903; Dirección del registro XBAR
                             ;entrada 4 (para IXNT1)

CTE_WR        .set    0068h  ; Cte. Para desactivar el WatchDog
MASK_PIEACK1  .set    00001h ; Máscara para PIE
MASK_PIE1_4   .set    00008h ; Habilita INT_PIE1.4 XINT1
MASK_GPIO12   .set    01000h ; Máscara para habilitar GPIO12 como salida
MASK_GIPO2    .set    0002h  ; Máscara para habilitar GPIO2
                             ; como interrupción (entrada)
XINT1_ENABLE  .set    0001h  ; Máscara para habilitar la interrpción XINT1

      .text
_c_int00
      SETC   INTM                          ; Deshabilita interrupciones
                                           ; mascarables
      MOV    SP,#DIR_SP
      EALLOW
            MOVL   XAR1,#WDCR              ; Direccionamiento indirecto al
                                           ; registro de control del watchdog
            MOV    *XAR1,#CTE_WR           ; Desactiva WatchDog

            MOVL   XAR1,#GPA_DIR
            MOV    *XAR1,#MASK_GPIO12      ; Configura al GPIO12 como salida
      EDIS

*CONFIGURACION INICIAL DE PIE
      MOVL   XAR1,#DIR_PIECRT
      MOV    *XAR1,#MASK_PIEACK1           ; Habilita INTs de PIE
      MOVL   XAR1,#DIR_PIEACK
      MOV    *XAR1,#MASK_PIEACK1           ; Habilita reconocimiento de
                                           ; INT1 PIE
      MOVL   XAR1,#DIR_PIEIER1
      MOV    *XAR1,#MASK_PIE1_4            ; Habilita INT1.4 (XINT1)
      OR     IER,#UNO                      ; Habilita interrupción entrada INT1


      EALLOW
*       LIGA SUBRUTINA DE INTERRUPCION _SUB_GPIO3 CON SU VECTOR DE INT
            MOVL   XAR2,#DIR_INTXINT1
```

```
                MOV     ACC,#_SUB_GPIO2
                MOVL    *XAR2,ACC
*       LIGA EL GPIO2 CON LA INTERRUPCION
                MOVL    XAR1,#XBARINPUT4
                MOV     *XAR1,#MASK_GIPO2
        EDIS

        MOVL    XAR1,#DIR_XINT1CR
        MOV     *XAR1,#XINT1_ENABLE          ; Habilita interrupción externa

        CLRC    INTM                         ; Habilita interrupciones
* Ciclo infinito que se interrumpe al habilitarse
* la interrupción
ESPERA_INT
        NOP
        NOP
        B       ESPERA_INT,UNC
**** Rutina de interrupción ****
_SUB_GPIO2
        MOVL    XAR1,#GPA_TOGGLE             ; Direccionamiento al
                                             ; registro GPATOGGLE
        MOV     *XAR1,#MASK_GPIO12           ; Escribe en el registro
                                             ; GPATOGGLE el numero MASK12
                                             ; para conmutar el estado de
                                             ; la salida del GPIO12

        MOVL    XAR1,#DIR_PIEACK             ; Direccionamiento al
                                             ; registro PIEACK
        MOV     *XAR1,#MASK_PIEACK1          ; Escribe la constante
                                             ; MASK_PIEACK1 para volver a
                                             ; habilitar la interrupción y
                                             ; pueda volver a ejecutarse
                                             ; la rutina
        IRET                                 ; Retorno al programa
                                             ; principal. Salida de la
                                             ; interrupción

        .end
```

Entrada y salida digital en lenguaje C

A continuación se muestra un ejemplo en lenguaje C que realiza la misma tarea que el programa en ensamblador anterior, el cual cambia el estado del GPIO12 cuando se detecta una interrupción externa por medio del GPIO2.

```
/*
 * Ejemplo_Analog.c
 *
 * Este programa configura la interrupción
 * externa 1 en el GPIO2 y enciende
 * el LED conectado al GPIO12 cuando se
 * detecta un flanco de bajada en la
 * interrupción externa.
 *

#include "F28x_Project.h"

//Funcion que atiende la interrupcion
__interrupt void XINT1(void){
        //Toggle GPIO12
        GpioDataRegs.GPATOGGLE.bit.GPIO12 = 1;
        PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;

}

int main(void){

        InitSysCtrl();
        InitGpio();
        DINT;

        InitPieCtrl();

        IER = 0x0000;
        IFR = 0x0000;

        InitPieVectTable();

        EALLOW;
        PieVectTable.XINT1_INT = &XINT1;
        EDIS;

        //Habilita el PIE
        PieCtrlRegs.PIECTRL.bit.ENPIE = 1;
        //Habilita la interrupción PIE 1.4
        PieCtrlRegs.PIEIER1.bit.INTx4 = 1;
        //Habilita la interrupción de CPU 1
        IER |= M_INT1;


        EALLOW;
        GpioCtrlRegs.GPAMUX1.bit.GPIO12 = 0;
        //GPIO12 como salidas
        GpioCtrlRegs.GPADIR.bit.GPIO12 = 1;
```

```
        GpioCtrlRegs.GPAMUX1.bit.GPIO2 = 0;
        //GPIO2 como entrada
        GpioCtrlRegs.GPADIR.bit.GPIO2 = 0;
        EDIS;

        //GPIO2 como interrupción externa
        GPIO_SetupXINT1Gpio(2);

        //Interrumpe en flanco de bajada
        XintRegs.XINT1CR.bit.POLARITY = 0;
        //Habilita la interrupción externa 1
        XintRegs.XINT1CR.bit.ENABLE = 1;

        EINT;           //Habilita interrupciones
        // Ciclo infinito
        while(1);       //todo el proceso se
                        //realiza en la interrupción
}
```

5.2. Temporizadores

La familia F2837xS cuenta con tres temporizadores de CPU (TIMER0/1/2) de 32 bits. El TIMER0 se maneja por interrupciones del módulo PIE, mientras que el TIMER 1 y el TIMER2 emplean las interrupciones INT13 e INT14, respectivamente. El TIMER2 está reservado para utilizarse con sistemas operativos en tiempo real [8].

El módulo es alimentado por el reloj SYSCLKOUT que se configura por los registros del PLL. El diagrama de bloques de operación del temporizador se presenta en la Figura 5.2.

La razón de tiempo de interrupción del temporizador está dada por la Ecuación (5.1):

$$f_{TINT} = \frac{1}{t_c(TDDRH:TDDR+1)(PRDH:PRD+1)} \tag{5.1}$$

donde:

f_{TINT}, frecuencia de interrupción del temporizador.
t_c, período de reloj SYSCLKOUT.
TDDRH:TDDR, escalamiento de 16 bits.
PRDH:PRD, escalamiento de 32 bits,

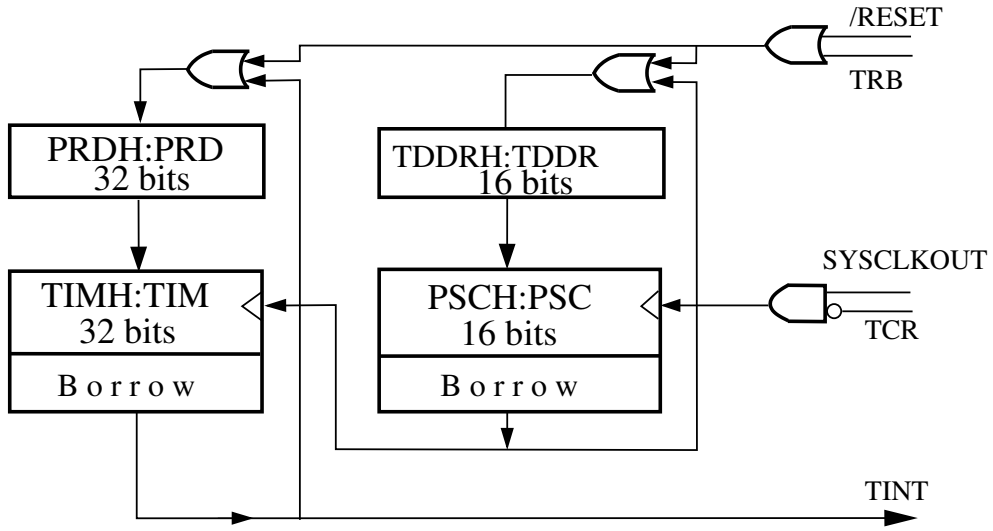


Figura 5.2: Temporizador de CPU [8]

5.2.1. Configuración del PLL

Para obtener obtener un periodo de interrupción constante en el TIMER, empleando (5.1), es necesario configurar el reloj del sistema, para esto se requiere:

1. Seleccionar la fuente de reloj (*CLKSRCCTL1.OSCCLKSRCSEL*).
2. Configurar la parte entera (*SYSPLLMULT.IMULT*) y fraccionaria del multiplicador del PLL (*SYSPLLMULT.FMULT*).
3. Configurar el divisor de reloj (*SYSCLKDIVSEL.PLLSYSCLKDIV*[3]).

La frecuencia del reloj se determina mediante (5.2):

$$f_{clk} = f_{osc}\frac{SYSPLLMULT.IMULT + SYSPLLMULT.FMULT}{SYSCLKDIVSEL.PLLSYSCLKDIV} \tag{5.2}$$

Donde f_{osc} es la frecuencia de una de las cuatro diferentes fuentes de reloj:

- Oscilador Interno Primario (INTOSC2): Es el reloj por defecto del sistema con una frecuencia de 10 MHz.
- Oscilador Interno de Respaldo (INTOSC1): Reloj redundante que se emplea para el WatchDog, tiene una frecuencia de 10MHz.

[3]El valor del divisor corresponde al doble del valor de este registro: 000000 = $x/1$, 000001 = $x/2$, 000010 = $x/4$, etc.

- Oscilador Externo (XTAL): emplea fuentes de reloj externas (cristal u osciladores), la frecuencia máxima se encuentra en la hoja de especificaciones [4].
- Entrada de Reloj Auxiliar (AUXCLKIN): Entrada de reloj externa adicional, solo está soportada en el GPIO133, emplea un reloj de 3.3V externo de una sola terminal. El límite de frecuencia de este reloj se detalla en la hoja de especificaciones [4].

El reloj empleado por el TIMER (PERx.SYSCLK) es de la misma frecuencia que el reloj empleado por el CPU (PLLSYSCLK) [8].

5.2.2. Utilización de un temporizador

Como ejemplo práctico, a continuación se presenta un programa que conmuta el estado de los LED's conectados a los puertos GPIO12 y GPIO13, en función de una interrupción interna que se activa en un período de tiempo configurado el temporizador 0.

Salidas digitales GPIO conmutadas por un temporizador en lenguaje ensamblador

En el siguiente código se muestra la configuración del TIMER0, de la interrupción interna vinculada a este temporizador y de las terminales 12 y 13 del puerto GPIOA como salidas en lenguaje ensamblador:

```
*
*      Conmutación de salidas digitales controladas
*      por un temporizador
*
* Este programa configura el TIMER0 para habilitar una
* interrupción cada vez que se cumple con cierto periodo
* de tiempo (contador de ciclos de reloj, configurado a
* 100 MHz). Cada vez que entra a la interrupcion, las
* terminales GPIO12 y GPIO13 conmutan su estado digital
* de salida, lo cual se visualiza en los leds conectados
* a estas terminales
*
              .global             _c_int00

DIR_SP     .set    0400h       ; Dirección del Stack Pointer
WDCR       .set   07029h       ; Direccion del reg. WatchDog
UNO        .set    001h

* Direcciones de configuración para el puerto GPA
GPADIR     .set    07C0Ah      ; Dirección del registro GPADIR
GPASET     .set    07F02h      ; Dirección del registro GPASET
GPACLEAR   .set    07F04h      ; Dirección del registro GPACLEAR
GPATOGLE   .set    07F06h      ; Dirección del registro GPATOGGLE
```

```
MASK12    .set   01000h      ; Número a escribir en el registro
                             ; GPADIR para usar como salida
                             ; la terminal GPIO12

* Direcciones de configuración para el Timmer 0
DIR_TIM       .set 0x0C00    ; Dirección del registro  TIM
DIR_TIMPRDL   .set 0x0C02    ; PRDL de TIMER0
DIR_TIMPRDH   .set 0x0C03    ; PRDH de TIMER0
DIR_TIM0TCR   .set 0x0C04    ; Control de TIMER0
DIR_PIECRT    .set 0X0CE0    ; Control de PIE, 01 habilita
                             ; INT_PIE
DIR_PIEACK    .set 0X0CE1    ; Reconocimiento de INT_PIE
DIR_PIEIER1   .set 0X0CE2
* Direcciones para habilitar inrterrupción del TIMER0
DIR_INTTIM0   .set 0x0D4C    ; Dir. del vector de TIMER0, INT1.7
DIR_PCLKCR0   .set 0x05d222  ; Reg. de Reloj periféricos

; Máscaras  y valores
C_WDCR        .set 0068h     ; Máscara de WD
PER_H         .set 0007h     ; Para PRD H 5fH   TIM  1s a SYSCLK 100 Mhz
PER_L         .set 0A120h    ; 100M = 5F5F100h, 25M = 17D7840h
                             ; 10M = 989680h, 2M = 1E8480
                             ; 1M = F4240h,  0.5 M = 7A120h
MASK_TIF      .set 08000h
MASK_TIE      .set 04000h
MASK_TRB      .set 00020h  ; Inicializa TIMER
MASK_TIE_TRB  .set 04020h  ; Inicializa TIMER y Carga Periodo 0402h
MASK_TIMER0   .set 00008h  ; Habilita reloj de TIMER0
MASK_PIEACK1  .set 00001h  ; Habilita PIECRT
MASK_PIE1_7   .set 00040h  ; Habilita INT_PIE1.7 DE TIMER0

              .text
_c_int00
       SETC   INTM                    ; Deshabilita INTERRUPCIONES
                                      ; mascarables
       MOV    SP,#DIR_SP
       EALLOW                         ; Habilita escritura a registros
                                      ; protegidos
              MOVL   XAR1,#WDCR       ; Direccionamiento indirecto al
                                      ; registro de control del watchdog
              MOV    *XAR1,#C_WDCR    ; Desactiva WatchDog
              MOVL   XAR1,#GPADIR
              MOV    *XAR1,#MASK12    ; GPIO12 salida
              MOVL   XAR1,#DIR_PCLKCR0 ; Alimenta reloj de TIMER0
              MOV    *XAR1,#MASK_TIMER0
       EDIS
```

```
* CONFIGURACION INICIAL DE PIE
        MOVL    XAR1,#DIR_PIECRT
        MOV     *XAR1,#MASK_PIEACK1         ; Habilita INTs de PIE
        MOVL    XAR1,#DIR_PIEACK
        MOV     *XAR1,#MASK_PIEACK1         ; Habilita reconocimiento de
                                            ; INT1 PIE
        MOVL    XAR1,#DIR_PIEIER1
        MOV     *XAR1,#MASK_PIE1_7          ; Habilita INT1.7, de PIE, TIMER0
        OR      IER,#UNO                    ; Habilita interrupción entrada INT1
* CONECTAR SUBRUTINA DE INTERRUPCION _SUB_TIM0 CON SU VECTOR DE INT
        EALLOW
                MOVL    XAR2,#DIR_INTTIM0
                MOV     ACC,#_SUB_TIM0
                MOVL    *XAR2,ACC
        EDIS
* Configura TIMER0
        MOVL    XAR1,#DIR_TIMPRDL
        MOV     *XAR1++,#PER_L
        MOV     *XAR1++,#PER_H
        MOVL    XAR1,#DIR_TIM0TCR
        MOV     AL,*XAR1
        OR      AL,#0x4020
        MOV     *XAR1,AL                    ; Carga TIMER e inicializa INT
                                            ; de TIMER0

        CLRC    INTM                        ; Habilita interrrupciones
* Ciclo infinito, que se interrumpe al habilitarse
* la interrupción
ESPERA_INT
        NOP                                 ; Instruc. de operación nula
        NOP                                 ; Instruc. de operación nula
        B       ESPERA_INT,UNC              ; Salto a la etiqueta
                                            ; ESPERA_INT sin condición
                                            ; a evaluar

**** Rutina de interrupción ****
_SUB_TIM0
        MOVL    XAR1,#GPATOGLE              ; Direccionamiento al
                                            ; registro GPATOGGLE
        MOV     *XAR1,#MASK12               ; Escribe en el registro
                                            ; GPATOGGLE el número MASK12
                                            ; para conmutar el estado de
                                            ; la salida del GPIO12

        MOVL    XAR1,#DIR_PIEACK            ; Direccionamiento al
                                            ; registro PIEACK
        MOV     *XAR1,#MASK_PIEACK1         ; Escribe la constante
                                            ; MASK_PIEACK1 para volver a
                                            ; habilitar la interrupción y
```

```
                                        ; pueda volver a ejecutarse
                                        ; la rutina
        IRET                            ; Retorno al programa
                                        ; principal. Salida de la
                                        ; interrupción

        .end
```

Salidas digitales GPIO conmutadas por un temporizador en lenguaje C

El programa anterior, se reduce considerablemente al utilizar las bibliotecas de Control-Suite programando en C, un código que realiza la misma tarea se presenta a continuación.

```
/*
 * Ejemplo_Timer.c
 *
 * Este programa configura el TIMER-0 con una
 * frecuencia de 100 MHz, el cual interrumpe
 * cada segundo, cambiando el estado del
 * GPIO2.
 *
 */

#include "F28x_Project.h"


// Función que se atiende al habilitarse la
// interrupción cada segundo
__interrupt void cpu_timer0_isr(void){
        // Toggle GPIO2
        GpioDataRegs.GPATOGGLE.bit.GPIO2 = 1;
        PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;
}

void main(void){

        InitSysCtrl();
        // Deshabilita interrupciones
        DINT;
        InitPieCtrl();

        IER = 0x0000;
        IFR = 0x0000;

        InitPieVectTable();

        EALLOW;
```

```
        // Declara el nombre de la función que se
        // atenderá a la interrupción
        PieVectTable.TIMER0_INT = &cpu_timer0_isr;
        EDIS;

        // Inicia los timers
        InitCpuTimers();

        // Define la frecuencia del reloj en 100MHz
        // y el periodo de interrupcion de 1 segundo
        ConfigCpuTimer(&CpuTimer0, 100, 1000000);

        /* Habilita la interrupción del timer
         * se DEBE escribir al registro completo
         * NO sirve si se usa CpuTimer0Regs.TCR.bit.TIE = 1
         */
        CpuTimer0Regs.TCR.all = 0x4000;


        //Configuración GPPIO2
        EALLOW;
        GpioCtrlRegs.GPAMUX1.bit.GPIO2 = 0;
        GpioCtrlRegs.GPADIR.bit.GPIO2 =  1;
        EDIS;

        IER |= M_INT1;
        //Configura el PIE
        PieCtrlRegs.PIEIER1.bit.INTx7 = 1;

        // Habilita interrupción
        EINT;
        ERTM;

        // Ciclo infinito para dejar funcionado el programa
        // para atender la funcion de interrupción
        while(1);      // Ciclo infinito
}
```

5.3. ePWM: salida PWM mejorada

El TMS320F28377S embebido en la tarjeta de desarrollo LanuchPad-F28377S, cuenta con 15 periféricos de modulación de ancho de pulso (ePWM), de los cuales nueve pueden pueden funcionar en modo de alta resolución (HRPWM).

Cada módulo ePWM está conformado por dos salidas; PWMxA y PWMxB, las cuales se pueden observar en el diagrama de bloques del periférico que se muestra en la Figura

5.3. Todos los módulos de ePWM se encuentran sincronizados para permitir que trabajen en conjunto como un mismo sistema [21].

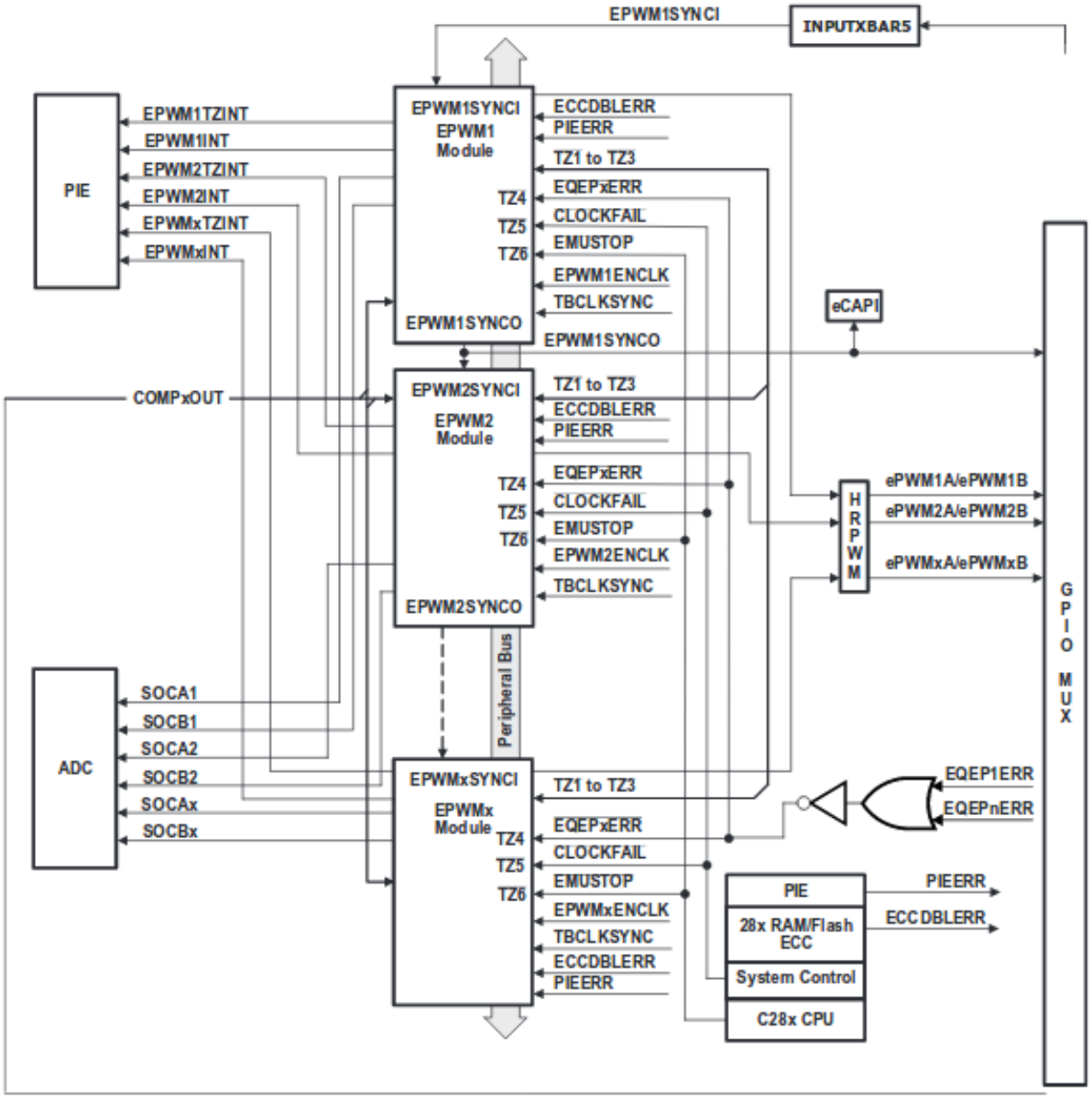


Figura 5.3: Diagrama a bloques del periférico ePWM del dsp TMS320F28377S [8]

Este periférico se puede emplear en diferentes modalidades, las cuáles son:

- Contador dedicado de 16 bits, con periodo y frecuencia de control.
- Las dos salidas de PWM (PWMxA y PWMxB) que pueden ser empleadas como:
  - Dos PWM independientes operando con flanco simple.
  - Dos PWM independientes en operación simétrica con flanco doble.
  - Un PWM con flanco doble operando de forma asimétrica.
- Generador de banda muerta con flanco de subida y bajada independientes y control de retraso.
- Generador de interrupciones de CPU e inicios de conversión (SOC) del ADC.
- Generador señales analógicas[4].

5.3.1. Utilización del ePWM

Como práctica de implementación, se expondrán dos programas que configuran el periférico para generar una señal escalón periódica de 1 kHz, con un ciclo de trabajo del 25 %, la cual se ve reflejada en la intensidad de iluminación del LED conectado al puerto GPIO12. El primer código que se presenta, esta desarrollado en lenguaje ensamblador.

```
*
*
* Configuración del PWM 7 con una frecuencia de 1KHz,
* un ciclo de trabajo del 25%. El reloj interno se configura
* a 50MHz y la señal generada se muestra en el GPIO12
*
        .global     _c_int00
N       .set 500
WDCR            .set    07029h ; Dirección del registro de
                               ; control del WatchDog
GPADIR          .set    07C0Ah ; Dirección del registro de
                               ; configuración del puerto
                               ; GPIOA (entradas/salidas)
GPASET          .set    07F02h ; Dirección del registro
                               ; GPASET
GPACLEAR        .set    07F04h ; Dirección del registro
                               ; GPACLEAR
GPATOGLE        .set    07F06h ; Dirección del registro de
                               ; conmutación de estado
                               ; GPATOGGLE
```

[4]Cuando el ciclo de trabajo es equivalente al valor analógico deseado.

```
* Direcciones de los registros de configuración del
* reloj para el sistema y periférico
DIR_CLKSRCCTL1      .set    0x0005D208
DIR_SYSPLLCTL1      .set    0x0005D20E
DIR_SYSPLLMULT      .set    0x0005D214
DIR_SYSCLKDIVSEL    .set    0x0005D222
* Direcciones de los registros de configuración de
* los periféricos GPIOA y ePWM7
DIR_GPIOGAMUX .set    07C20h        ; 0x07C+0x20
DIR_GPIOAMUXH .set    07C06h        ; 0x07C+0x06
DIR_PCLCLK0   .set    0x05D322      ; CAMBIO
DIR_PCLCLK2   .set    0x05D326      ; PCLKCR2 CAMBIO
DIR_PWM7TBCTL .set    0x04600
DIR_PWM7TBCTR .set    0x04604
DIR_PWM7TBPRD .set    0x04663
DIR_PWM7CMPA  .set    0x0466A
DIR_PWM7AQCTLA      .set    0x04640
* Números de habilitación de puertos GPIO de salida
MASK12        .set    01000h        ; Número a escribir
                                    ; en el registro GPASET
                                    ; para usar como salida
                                    ; la terminal GPIO12
MASK13        .set    02000h        ; Número a escribir en
                                    ; el registro GPASET
                                    ; para usar como salida
                                    ; la terminal GPIO13
MASK1213      .set    03000h

      .text
_c_int00
      CLRC   XF             ; Escribe cero en la bandera XF
      SETC   SXM            ; Modo extensión de signo
      EALLOW                ; Habilita la escritura en registros
                            ; protegidos
      MOVL   XAR1,#WDCR     ; Direccionamiento al reg.
                            ; del watchdog
      MOV    *XAR1,#0068h   ; Deshabilita el uso del
                            ; watchdog
      MOVL   XAR1,#GPADIR   ; Direccionmiento al reg.
                            ; GPADIR para indicar el
                            : uso de perifericos GPIO
      MOV    *XAR1,#MASK1213 ; Esribe el dato MASK1213
                            ; para configurar GPIO12
                            ; y GPIO13 como salidas
      DINT                  ; Deshabilita uso de interrupciones

* Configuración del ciclo de reloj del sistema a 50MHz
* f_osc = 10MHz*(imult+fmult)/(divsel)
      MOVL   XAR1,#DIR_CLKSRCCTL1 ; Direccionamiento al
```

```
                                    ; registro para elegir
                                    ; fuente de reloj
        MOV     *XAR1++,#0x01       ; Escribe en el regitro
                                    ; apuntado 1h para indicar
                                    ; el uso del circuito
                                    ; oscilador externo(cristal)
        MOVL    XAR1,#DIR_SYSPLLMULT
        MOV     *XAR1++,#0x014      ; imult=20, fmult=0
        MOVL    XAR1,#DIR_SYSCLKDIVSEL ; Direccionamiento al
                                    ; reg. PLL divisor
        MOV     *XAR1++,#0x02       ; Divide la señal de reloj
                                    ; en 4
* Configuración del ciclo de reloj del PWM
        MOVL    XAR1,#DIR_PCLCLK2   ; Habilita reloj de PWM7
        MOV     *XAR1,#0x040        ; Alimenta reloj a PWM7
* Configuración de puertos GPIO
        MOVL    XAR1,#DIR_GPIOGAMUX ; Direccionamiento al
                                    ; MUX del GPIOA para usar
                                    ; GPIO12 como pwm7
        MOV     *XAR1,#0x00         ; Habilita el GPIO12
                                    ; como PWM7
        MOVL    XAR1,#DIR_GPIOAMUXH
        ZAPA
        MOV     AH,#0x0500          ; cargamos el valor a AH
        MOVL    *XAR1,ACC           ; GPIO12 Y GPI13 Para EPWM7
                                    ; #0x01
* Configuración del contador de flancos para configurar
* el ciclo de trabajo del pwm
        MOVL    XAR1,#DIR_PCLCLK0
        MOV     ACC,*XAR1
        OR      AH,#0x04            ; Habilita el PWM7 como
                                    ; salida analógica
        MOVL    *XAR1,ACC           ; Habilita inicio conteo
                                    ; PWMs
        EDIS                        ; Deshabilita la escritura
                                    ; en registros protegidos
* Configuración del ciclo de trabajo del pwm7
        MOVL    XAR1, #DIR_PWM7TBCTL ; Conteo hacia arriba (UP)
        MOV     *XAR1,#00
        MOVL    XAR1,#DIR_PWM7TBPRD ;  Carga contador
                                    ; de Perido PWM
        MOV     *XAR1,#25000        ; f_pwm=50MHz/(2*25000)=1KH
        MOVL    XAR1,#DIR_PWM7CMPA  ; valor de comparación
        ZAPA
        MOV     AH,#5000            ; ciclo de trabajo del 25%
        MOVL    *XAR1,ACC
        MOVL    XAR1,#DIR_PWM7TBCTR
        MOV     *XAR1,#0x00         ; Limpia el contador
        MOVL    XAR1,#DIR_PWM7AQCTLA ; Configura el
```

```
                                        ; comportamiento del
                                        ; GPIO
        MOV     *XAR1,#0x012

FIN_R
        NOP
        LB      FIN_R                   ; Ciclo infinito
        .end
```

Generador de una señal analógica en lenguaje C

A continuación se presenta el programa equivalente en lenguaje C, del código anterior.

```
/*
* Configura el PWM 7 con una frecuencia de 1KHz
* con un ciclo de trabajo del 25%, el reloj interno
* se configura a 50MHz y la salida se muestra en
* el GPIO12
*
*/

// DSP28x Headerfile
#include "F28x_Project.h"


int main(void) {

        InitSysCtrl();
        DINT;

        EALLOW;
        //Configura el reloj del sistema a 50 Mhz
        InitSysPll(XTAL_OSC,IMULT_20,FMULT_0,PLLCLK_BY_4);

        //Alimenta reloj a PWM7
        CpuSysRegs.PCLKCR2.bit.EPWM7 = 1;
        //configrua el GPIO como PWM
        InitEPwm7Gpio();
        //Habilita inicio conteo PWMs
        CpuSysRegs.PCLKCR0.bit.TBCLKSYNC = 1;
        EDIS;

        //Conteo hacia arriba (UP)
        EPwm7Regs.TBCTL.all = 0x00;
        //f_pwm = 50MHz/(2*25000) = 1KH
        EPwm7Regs.TBPRD = 25000;
        //ciclo de trabajo del 25%
        EPwm7Regs.CMPA.bit.CMPA = 5000;
```

```
        //cuando TBCTR = 0 pone la salida en alto
        EPwm7Regs.AQCTLA.bit.ZRO  = 2;
        //cuando TBCTR = CMPA pone la salida en bajo
        EPwm7Regs.AQCTLA.bit.CAU = 1;

        while(1);       // Ciclo infinito

        //return 0;
}
```

5.4. Subsistema Analógico

Este subsistema es una de las principales entradas de datos para que el DSP opere en tiempo real. Consisten en convertidores Analógico-Digital (ADC)[5] y convertidores Digital-Analógico (DAC). En la Figura 5.4 se muestra el diagrama de la arquitectura del subsistema.

5.4.1. Convertidor ADC

El MCU cuenta con dos convertidores analógico-digital (ADC-A y ADC-B), que funcionan bajo el principio de aproximaciones sucesivas, con resolución seleccionable. El módulo está diseñado para muestrear y retener simultáneamente o de forma independiente cada canal [1], [22]. Algunas de las características de este periférico son:

- Cuenta con 16 canales[6].
- Número de muestras por segundo: 3.5 MSPS.
- El tiempo de conversión es de 290 ns.
- Resolución: 12 ó 16 bits.

En la Figura 5.5 se muestra el diagrama a bloques del ADC. La secuencia de conversión se da mediante el uso de los módulos *Start-of-Conversion* (SOC), estos determinan el inicio de conversión de los diferentes canales. Los módulos SOC pueden ser "disparados" de diferentes maneras, entre las que se encuentran:

- Por software.
- Con Timer 0/1/2.

[5]Cuenta con un sensor de temperatura conectado a uno de estos convertidores.
[6]No todos los canales se encuentran disponibles en la tarjeta.

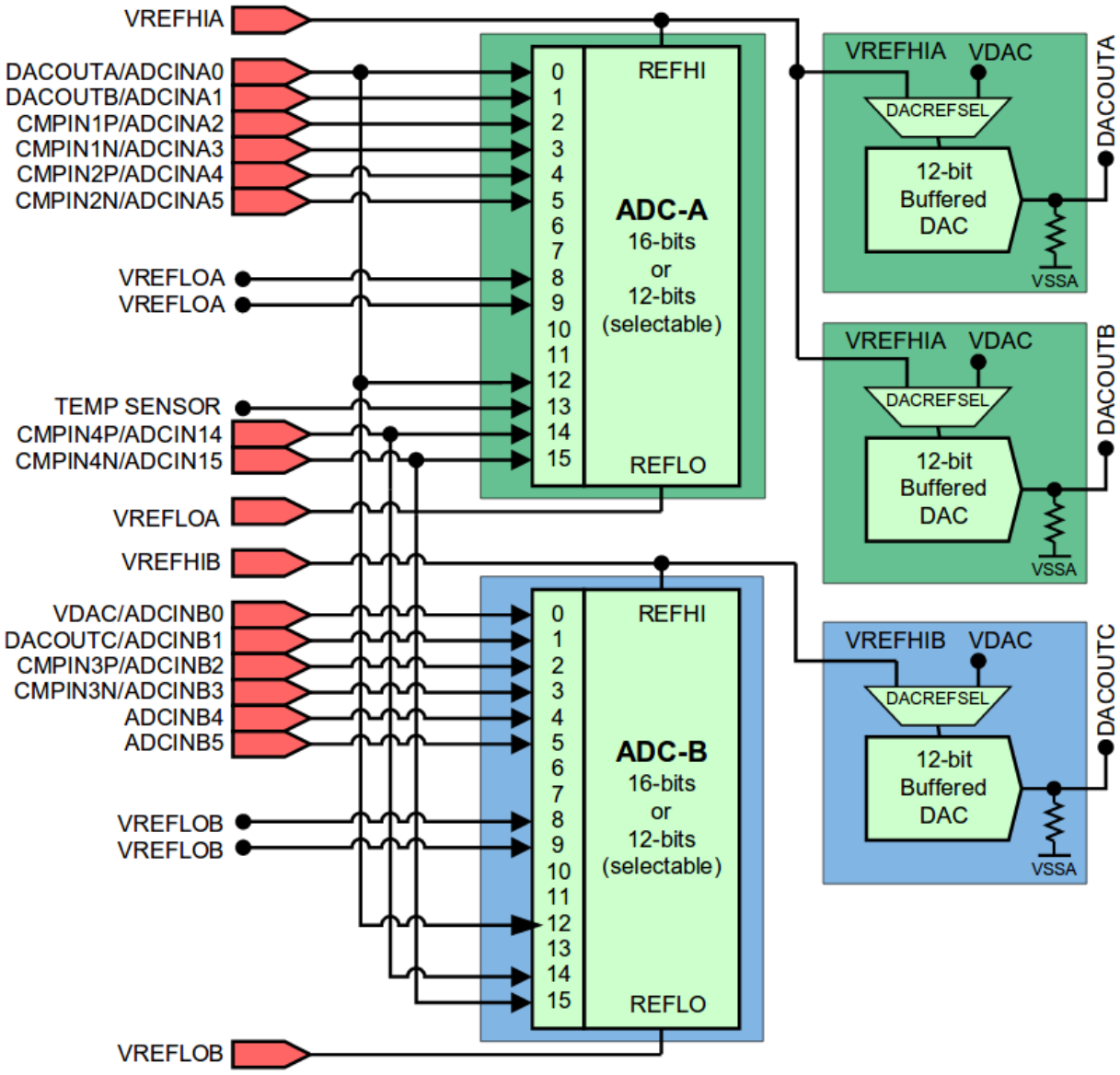


Figura 5.4: Diagrama a bloques del subsistema analógico [8]

- Por ePWM1 a ePWM12.
- Usando interrupción 1 ó 2 del ADC[7].

Cada módulo SOC tiene una señal de *End-of-Conversion* (EOC), la cual indica que se terminó de realizar la conversión del canal asignado a dicho módulo, estas señales pueden

[7]Esta configuración se emplea para realizar conversiones continuas

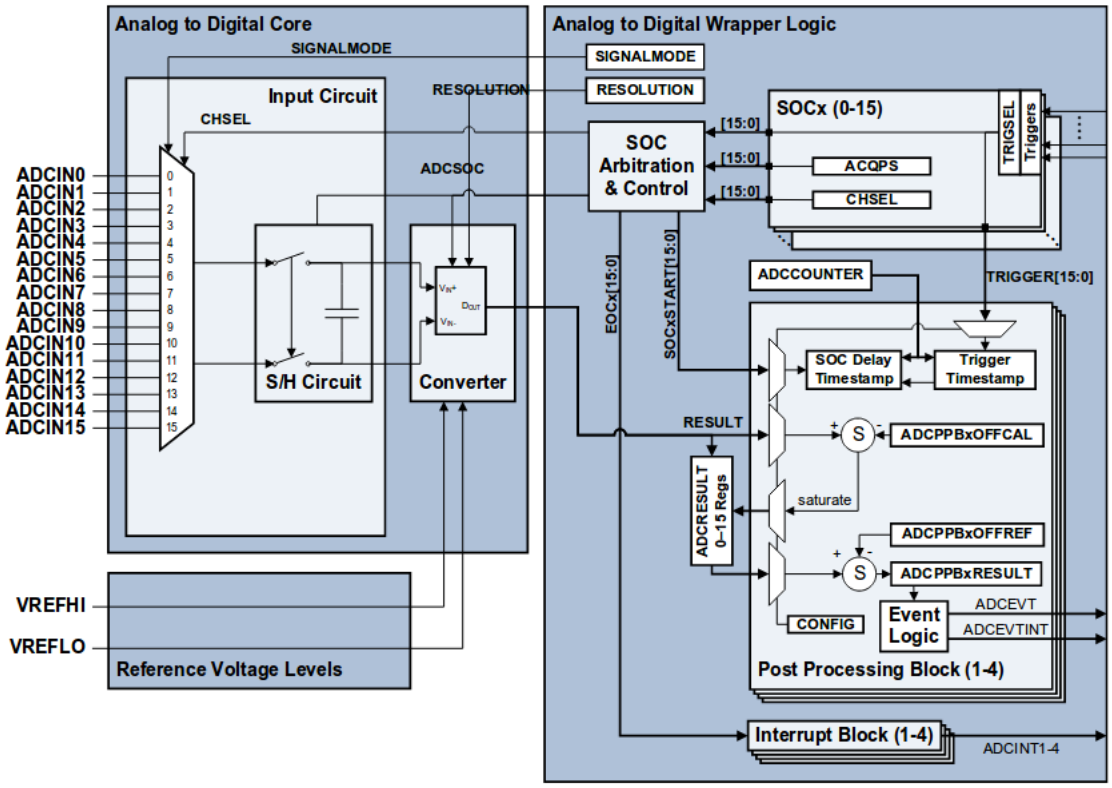


Figura 5.5: Diagrama a bloques del ADC [8]

disparar las banderas de interrupción del ADC. Cada ADC tiene cuatro banderas de interrupción (ADCINT1-ADCINT4), las cuales pueden ser leídas de forma directa o pueden ser empleadas en el PIE.

5.4.2. Convertidor DAC

El DSP TMS320F328377S tiene tres convertidores digital-analógico (DAC-A, DAC-B y DAC-C), con una resolución de 12 bits y con voltajes de referencia seleccionables, en la Figura 5.6 se muestra el diagrama a bloques de este convertidor.

El voltaje de salida de estos convertidores está determinado por la Ecuación (5.3) definida como

$$V_{DAC} = \frac{DACVALA * DACREF}{4096} \tag{5.3}$$

donde $DACVALA$ es el valor digital a convertir y $DACREF$ es el voltaje de referencia del convertidor DAC.

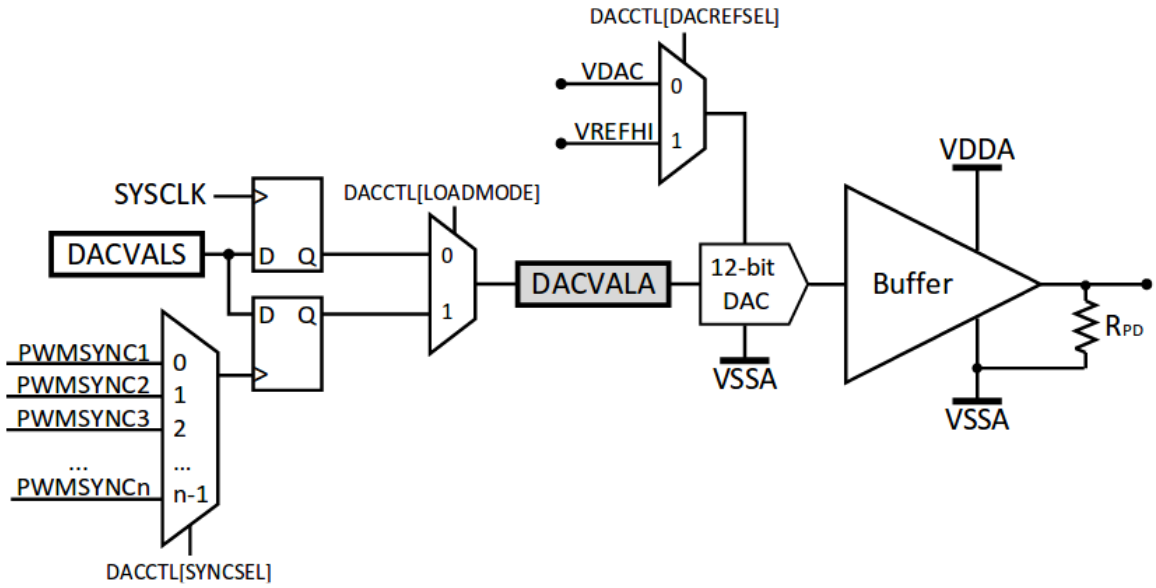


Figura 5.6: Diagrama a bloques del DAC [8]

5.4.3. Configuración y uso del ADC y DAC

Para utilizar el subsistema analógico, se propuso una biblioteca (código en el apéndice B), para facilitar la configuración de los periféricos ADC y DAC. El programa que se muestra a continuación es un ejemplo del uso de los métodos de dicha biblioteca, que habilita el canal 2 del módulo A del ADC para adquirir una señal analógica, considerando una frecuencia de muestreo de 8 kHz y configura el DAC-C para sacar la señal convertida a digital.

```
/*
 * Ejemplo_Analog.c
 *
 * Este programa configrua el ADC-A
 * canal 2, y el DACC (ADC-B1).
 *
 * La frecuencia de muestreo del ADC-A
 * es de 8 KHz.
 *
 * La entrada del ADC se envía al DAC
 *
 *  ADCINA2 -> J7-64
 *      ADCINB1 -> DACC -> J3-24
*/
```

```
#include "F28x_Project.h"
//Biblioteca de configuracion propuesta
#include "Analog.h"

// Funcion de proceso del ADCA
extern void ADCA_Process(void){
        Uint16 val;
        //Obtiene el valor del canal
        val = ADC_RESULT_PTR[ADCA]->ADCRESULT2;
        DAC_Send(DACC, val);
}

// Funcion de proceso del ADCB
extern void ADCB_Process(void){
        __asm(" nop");
}

int main(void){

        InitSysCtrl();
        InitGpio();
        DINT;

        InitPieCtrl();

        IER = 0x0000;
        IFR = 0x0000;

        InitPieVectTable();

        // Configura operación del ADC-A indicando la
        // frecuencia de muestreo
        ADC_Configure(ADCA,8000);

        // Configura e inicia el canal 2 del ADC-A
        ADC_Init(ADCA, 2);

        // Habilita la interrupción del ADC-A
        // para fin de conversion
        // el canal 2
        ADC_Int(ADCA, 2);

        // Configura el DAC-C
        DAC_Configure(DACC);

        // Inicia la operación del ADC-A
        ADC_Start(ADCA);

        // Ciclo infinito, para dejar el programa
```

```
        // funcionando y atendiendo la interrupcion
        while(1);       // Ciclo infinito

}
```

5.5. Puertos serial SPI

Los puertos SPI (*Serial Peripheral Interface*) son puertos seriales síncronos de alta velocidad, este periférico emplea cuatro[8] pines para realizar la comunicación [1], [23], los cuáles son:

- SPISOMI; salida del esclavo y entrada a maestro.
- SPISIMO; entrada del esclavo y salida a maestro.
- SPISTE; habilitación del esclavo.
- SPICLK; señal de reloj.

Las características de este periférico son:

- Puede funcionar como maestro y como esclavo.
- Hasta 125 velocidades de transmisión/recepción diferentes.
- Tamaño de palabra de 1 a 16 bits.
- Cuatro tipos de esquemas de reloj (polaridad y fase).
- Recibe y transmite de forma simultánea.
- Cuenta con una memoria FIFO de 16 niveles.
- Soporta acceso directo a memoria (DMA).

En la Figura 5.8 se muestra el diagrama de bloques de este módulo, en el cual se observa que cuenta con dos interrupciones: SPIINT/SPIRXINT[9] y SPITXINT[10]

La longitud de palabra se puede seleccionar empleando el registro *SPICCR*, sin embargo, debido a que los registro de transmisión (*SPITXBUF*) y recepción (*SPIRXBUF*) son de 16

[8]Se puede emplear en modo de tres pines.
[9]Si se usa la FIFO esta interrupción se llama SPIRXINT y se emplea para la recepción de datos.
[10]Si no se emplea FIFO esta interrupción no se emplea.

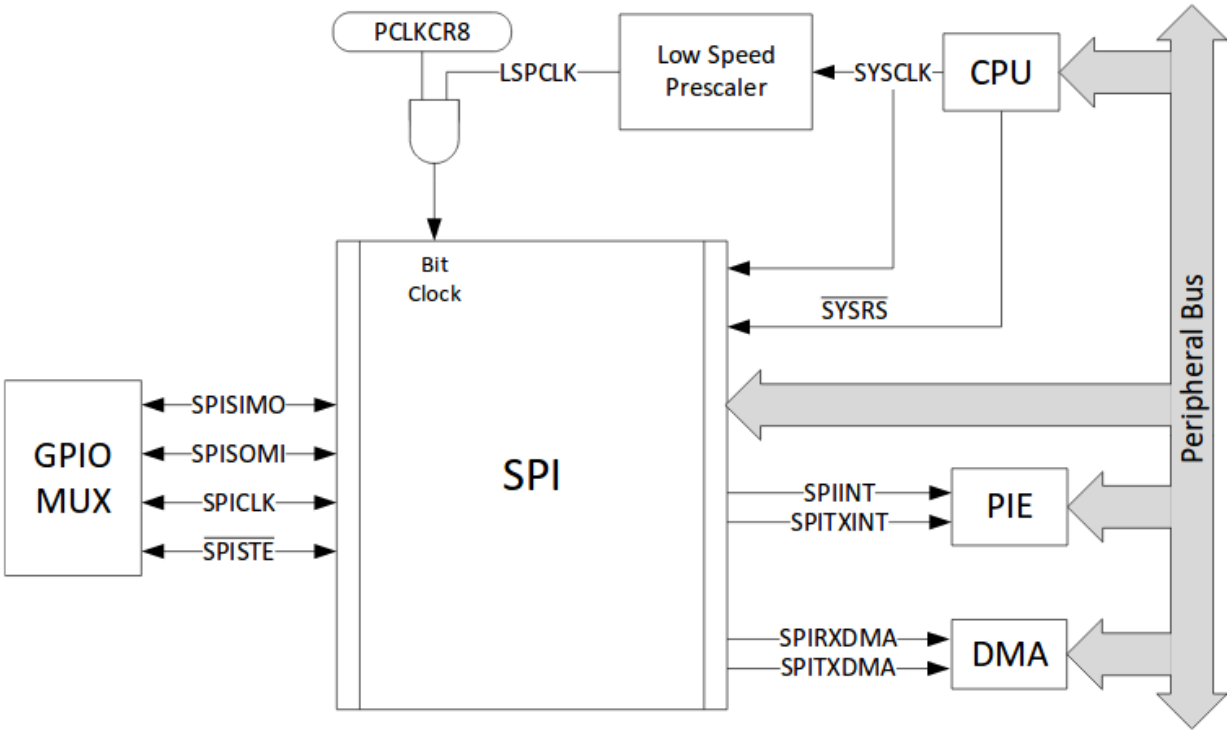


Figura 5.7: Diagrama a bloques del SPI [8]

bits, si se desean emplear palabras con un menor número de bits, los datos deben estar justificados a la izquierda para transmitir, mientras que al recibirlos estos estarán justificados a la derecha.

La configuración del reloj se realiza mediante el registro *SPIBRR* de acuerdo con (5.4), sin embargo, el reloj no puede ser mayor que LSPCLK/4.

$$SPI_{baudrate} = \frac{LPSCLK}{SPIBRR + 1} \tag{5.4}$$

En la Tabla 5.2 se muestran los cuatro diferentes esquemas de reloj soportados por el módulo SPI, los cuales definen la manera en la que se capturan y envían datos [8].

La configuración de este periférico se realiza de la siguiente manera:

1. Poner el módulo en reset (SPIWRESET = 0).
2. Configurar el módulo SPI:
   - Maestro/Esclavo.

Esquema SPICLK	CLK_Polaridad	CLK_Fase
Flanco de subida sin retraso	0	0
Flanco de subida con retraso	0	1
Flanco de bajada sin retraso	1	0
Flanco de bajada con retraso	1	1

Tabla 5.2: Esquemas de reloj.

- Polaridad y fase
- *Baud rate*
- Modo de alta velocidad (si se desea)
- Tamaño de palabra
- Borrar banderas
- Habilitar el modo de tres hilos (si se desea)
- Configurar FIFO (Fist input first output).

3. Activar interrupciones
4. Poner SPIWRESET = 0 para iniciar la operación.

5.5.1. Configuración y uso del SPI

A continuación se muestra un ejemplo de configuración de este periférico, enviando una variable tipo entero por el puerto.

```
/*
 * Ejemplo_SPI.c
 *
 * Este ejemplo configura el puerto SPI-A
 * los pines empleados son:
 *
 * SPISOMI:   GPIO17
 * SPISIMO:   GPIO16
 * SPISTE:    GPIO18
 * SPICLK:   GPIO19
 *
 * El pin CS se encuentra negado. Si se
 * conectan entre si los pines MISO y
 * MOSI se realiza un "eco" de los datos
 * enviados los cuales se pueden ver en la
 * variable "data"
 *
```

```
 * La configuración del puerto es:
 *
 * Bits: 16
 * SCLK: 12.5 MHz
 * Modo: Maestro
*/

#include "F28x_Project.h"

bool flag;
Uint16 i;
Uint16 data;

interrupt void SPIisr(void){

        flag = true;
        data = SpiaRegs.SPIRXBUF;

        // Limpia la bandera de OverFlow
        SpiaRegs.SPIFFRX.bit.RXFFOVFCLR=1;
        // Limpia la bandera de interrupcion
        SpiaRegs.SPIFFRX.bit.RXFFINTCLR=1;
        // Issue PIE ack
        PieCtrlRegs.PIEACK.all|=0x20;
        __asm(" NOP");
}

void main(){
        InitSysCtrl();
        InitPieCtrl();
        DINT;
        IER = 0x0000;
        IFR = 0x0000;

        InitPieVectTable();

        // Configura los GPIO
        InitSpiGpio();

        EALLOW;
        PieVectTable.SPIA_RX_INT = &SPIisr;
        EDIS;

        // Desactiva el módulo SPI
        SpiaRegs.SPICCR.bit.SPISWRESET = 0;
        // Selecciona Polaridad
        SpiaRegs.SPICCR.bit.CLKPOLARITY = 0;
        // Desactiva el modo de alta velocidad
        SpiaRegs.SPICCR.bit.HS_MODE = 0;
```

```
// Deshabilita el módo LoopBack
SpiaRegs.SPICCR.bit.SPILBK = 0;
// Define la longitud de palabra en 16 bits
SpiaRegs.SPICCR.bit.SPICHAR = 0xF;

SpiaRegs.SPICTL.bit.OVERRUNINTENA = 0;
// Selecciona el tipo de fase del clk
SpiaRegs.SPICTL.bit.CLK_PHASE = 0;
// Define al DSP como maestro
SpiaRegs.SPICTL.bit.MASTER_SLAVE = 1;
// Habilita la transmisión
SpiaRegs.SPICTL.bit.TALK = 1;
// Habilita la interrupción
SpiaRegs.SPICTL.bit.SPIINTENA = 1;

SpiaRegs.SPISTS.all = 0x00;
//LSPCLK/(SPIBRR+1)=100M/(7+1)=12.5M
SpiaRegs.SPIBRR.all = 0x07;

SpiaRegs.SPIFFTX.bit.SPIRST = 1;
SpiaRegs.SPIFFTX.bit.SPIFFENA = 0;
// Habilita la FIFO (Tx)
SpiaRegs.SPIFFTX.bit.TXFFIENA = 1;
// Genera la interrupción cuando ya no
// hay palabras en la FIFO
SpiaRegs.SPIFFTX.bit.TXFFIL = 0;

// Habilita la FIFO (Rx)
SpiaRegs.SPIFFRX.bit.RXFFIENA =1;
// Genera la interrupción cuando
// se recibió una palabra
SpiaRegs.SPIFFRX.bit.RXFFIL = 1;

// Habilita el SPI
SpiaRegs.SPICCR.bit.SPISWRESET = 1;


SpiaRegs.SPIFFTX.bit.TXFIFO=1;
SpiaRegs.SPIFFRX.bit.RXFIFORESET=1;

// Habilita el bloque de PIE
PieCtrlRegs.PIECTRL.bit.ENPIE = 1;
//Habilita la interrupción 1 del grupo 6
PieCtrlRegs.PIEIER6.bit.INTx1=1;
//Habilita la interrupción 6 del CPU
IER = 0x20;
EINT;          // Habilita interrupciones

i = 0;
```

```
        while(1){
                // Dato a enviar
                SpiaRegs.SPITXBUF = i;
                while(!flag); // Espera la interrupción
                flag = false;
                i++;          // Incrementa el dato a enviar
        }

}
```

5.6. Puertos serial SCI

La interfaz de comunicación serial (SCI) consiste en un puerto serial asíncrono, formado por dos canales de interfaz. La tarjeta de desarrollo LaunchXL-F28377S tiene disponibles los dos puertos seriales del DSP TMS320F28377S. Cada receptor y transmisor cuenta con una memoria FIFO de 16 niveles, así mismo, capaz de generar interrupciones tanto de transmisión como de recepción, además utiliza el formato estándar NRZ *non-return-zero* (NRZ) [1], [24]. Algunas de las características de este periférico son:

- Cada puerto consiste en dos terminales externas; una para transmitir (Tx) y otra de recepción (Rx).
- *Baud Rate* automático o programable (hasta 64K combinaciones diferentes).
- Longitud de palabra programable de 1 a 8 bits.
- Paridad: par, impar, ninguna.
- Uno o dos bits para detener la comunicación.
- Operación *Full-Duplex* o *Half-Duplex*.

En la Figura 5.8 se muestra el diagrama de bloques del SCI, cabe destacar que este periférico emplea el reloj de baja frecuencia (LSPCLK), el cual opera a (1/4) de la frecuencia del CPU, para determinar el *Baud Rate* se emplea (5.5).

$$BaudRate = \frac{LSPCLK}{(BRR + 1) * 8} \tag{5.5}$$

donde BRR es la combinación de los registros *SCIHBAUD.BAUD* y *SCILBAUD.BAUD*

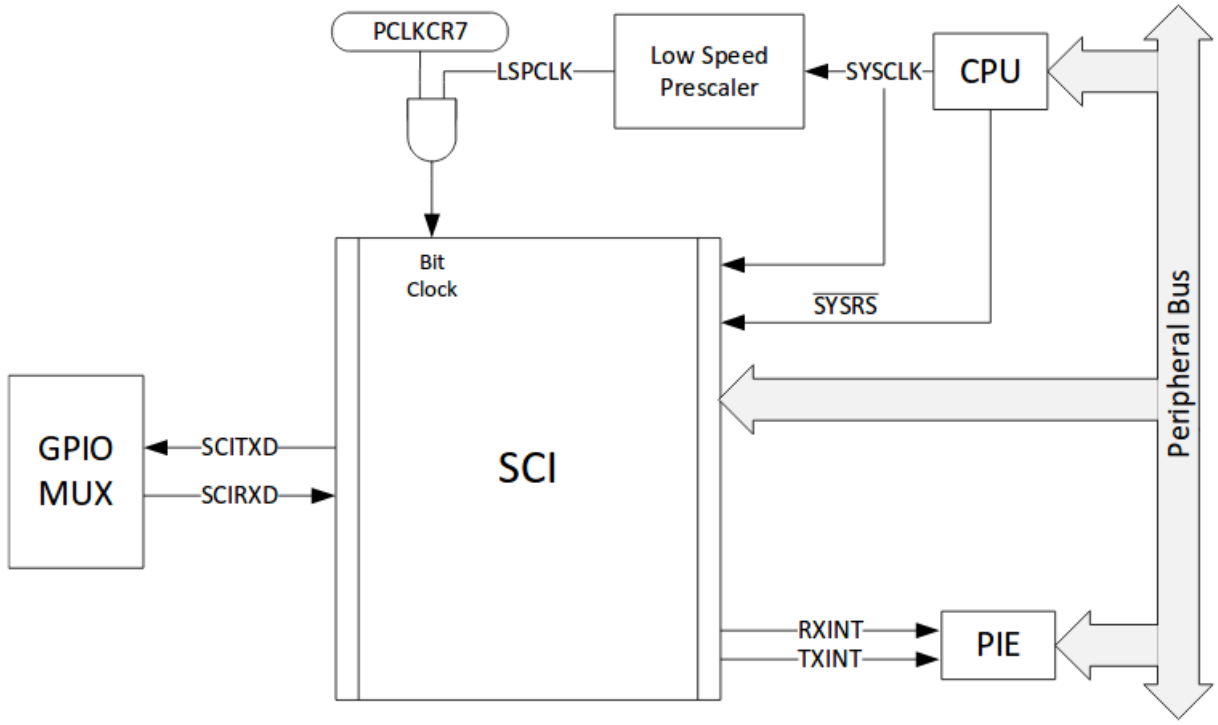


Figura 5.8: Diagrama a bloques del SCI [8]

5.6.1. Configuración y uso del SCI

De forma similar a los puertos ADC y DAC, se propuso una biblioteca para facilitar la configuración y uso de los puertos SCI, el código de dicha biblioteca esta disponible en el Apéndice C. A continuación se presenta un programa que ejemplifica el uso del puerto serial.

```
/*
 * Ejemplo_serial.c
 *
 * Este ejemplo recibe datos por el puerto serial-A
 * el cual se encuentra conectado al puerto USB (mismo
 * que se emplea para la programación de la tarjeta),
 * los datos son reenviados nuevamente por el puerto
 * serial realizando un "eco".
 *
 * El puerto se configura como:
 *
 * Bits = 8
 * Paridad = NO
 * Modo = Asíncrono
 * Stop = 1 bit
 *
```

```
 * Se puede emplear Putty o cualquier otro programa
 * para probar el funcionamiento, se aceptan las
 * siguientes velocidades de transferencia:
 *
 * BR9600    = 9600   Bauds
 * BR19200   = 19200  Bauds
 * BR38400   = 38400  Bauds
 * BR57600   = 57600  Bauds
 * BR115200  = 115200 Bauds
 */
#include "F28x_Project.h"
// Biblioteca desarrollada para configuración y
// uso de los puertos seriales SCI
#include "Serial.h"

Uint16 Rx;

// Funcion que utiliza el puerto serial
extern void Serial_Process(void){
        // Caracter recibido
        Rx = SciaRegs.SCIRXBUF.all;
        // Reenvió el caracter recibido
        Serial_putchar(Rx);
}

void main(){

        InitSysCtrl();
        InitPieCtrl();

        IER = 0x0000;
        IFR = 0x0000;

        InitPieVectTable();

        // Iniciacion del SCI
        Serial_Init();
        // Configuracion del puerto, definiendo
        // la velocidad de transmision
        Serial_Configure(BR9600);

        // Inicio de operacion del puerto serial
        Serial_Start();
        //
        Serial_print("Ejemplo de puerto serial A:\n\r\0");

        // Ciclo infinito
        while(1); //<- Todo se hace en la interrupción
```

```
}
```

5.7. Resumen del capítulo

La interacción de la máquina utilizada con su entorno es muy importante, en muchos casos se tiene un procesador muy completo el cual puede realizar procesos de mucha demanda computacional, sin embargo, no siempre es útil porque no contiene periféricos, lo que significa que no se puede tener un contacto con su entorno. La ventaja del TMS320F28377S es que se pueden realizar diferentes procesos de señales y de la misma manera cuenta con diferentes periféricos de entrada y salida de uso general (GPIO), PWM, ADC, DAC, I2C, SPI y SCI. En este capítulo se expuso la configuración de dichos periféricos y el uso de los mismos en lenguaje ensamblador y en lenguaje C por medio de ejemplos. De la misma manera se creó una librería (Analog.h) que contiene la configuración para la adquisición de señales acústicas, tales como voz o música, misma que se expone en el Apéndice B.

Ejercicios propuestos

Para complementar los ejemplos expuestos en este capítulo, se sugiere al lector una serie de ejercicios para practicar los temas abordados.

1. Programa que lea el ADC y envíe el resultado por el puerto serial.
2. Comunicar dos DSPs mediante el puerto SPI.
3. Generar tonos DTMF de forma digital y enviarlos al DAC y al PWM.
4. Iniciar el temporizador mediante un interrupción externa para encender un led de forma periódica durante 5 segundos.

Capítulo 6

Combinación de lenguaje C/C++ y ensamblador

La mayoría de los programas que se han presentado, implementan operaciones y algoritmos de PDS en lenguaje ensamblador y lenguaje C/C++, utilizando diferentes unidades de la arquitectura TMS320F28xxxx. Todos tienen en común funcionar utilizando datos escritos en la memoria del dispositivo, previo a la ejecución del programa.

Adicionalmente, en el *Capítulo* 5 se describió la configuración para usar algunos de los periféricos disponibles del DSP. El orden de los temas, tiene por objetivo brindar los conceptos y herramientas necesarias para desarrollar aplicaciones en línea, interactuando con señales mediante sensores que pueden ser procesadas para transmitir información a otros dispositivos o activar actuadores, por mencionar algunos ejemplos.

Como último paso antes de ver aplicaciones que combinan algoritmos y periféricos, se expondrá la capacidad del compilador para comunicar un programa principal en lenguaje C/C++, con instrucciones, rutinas y funciones escritas en ensamblador. Esta característica permite al usuario optimizar algunas operaciones y/o procesos, utilizando de forma específica la arquitectura acorde a ciertos lineamientos y reglas de acceso, que se describirán en este capítulo, finalizando con un ejemplo de implementación en donde se modifican algunos programas de la Sección 4.4.

6.1. Registros de interfaz

La capacidad de poder utilizar ambos lenguajes es posible por dos factores; las reglas de uso de los registros de la arquitectura, que sigue el compilador para los programas de C/C++ y por la metodología de atención a funciones. Existen cuatro formas de utilizar lenguaje ensamblador en conjunto con un programa escrito en C/C++, estos casos de implementación son:

* Utilizando funciones o subrutinas escritas en uno o varios archivos *.asm* diferentes al programa principal.
* Accediendo a constantes o modificando variables declaradas en ensamblador, desde programas en C/C++.
* Declarando directamente una instrucción de ensamblador en el programa principal.
* Implementando rutinas definidas en el compilador, conocidas como funciones intrínsecas.

Para implementar cualquiera de las opciones listadas, el compilador utiliza como interfaz de comunicación entre ambos lenguajes, los registros de la arquitectura que se listan en la Tabla 6.1 describiendo su función específica.

El procedimiento de atención al llamado de funciones, en cualquiera de los dos lenguajes se hace acorde a la convención especificada en la Sección 7.2 del manual [10], indicando la distribución de información en diferentes registros, para regresar al programa principal y continuar con su ejecución. Esta es la razón por la que solo los registros mencionados en la Tabla 6.1 pueden ser utilizados para la transferencia de variables, denominando a otros registros como ***dedicados*** cuya información inicial al atender una función debe preservarse al retorno, por lo que de ser necesario su uso, se deberá de respaldar su información y volver a colocarla previo a finalizar la rutina. Los registros dedicados son

* Auxiliares del CPU; XAR1, XAR2 y XAR3
* Registros de la FPU; R4H, R5H, R6H y R7H
* Como caso especial el Stack Pointer (SP)

Al poder transferir variables a una función con el SP, se debe tener cuidado de conservar los datos y direcciones que este pueda llevar a su pila, porque un mal manejo de dicha información puede impedir el retorno de la función, por eso se sugiere no utilizarlo a menos que se conozca a fondo su funcionamiento. Información más detallada y especifica sobre las convenciones que sigue el compilador para usar los registros, al construir un código de ejecución se puede consultar en las Secciones 7.2 y 7.3 del manual [10].

Tabla 6.1: Lista de registros de interfaz al combinar el uso de lenguaje ensamblador en un programa principal de C/C++.

Registro	Descripción de uso
ACC	Pasa un argumento de 32 bits a una función, o puede contener la parte alta de un valor de 64 bits y guardar un dato de 32 bits al término de una función.
AH	Transfiere un argumento de 16 bits a una función
AL	Traslada un argumento de 16 bits a una función y guarda un dato de 16 bits al término de una función
P	Introduce la parte baja de un dato de 64 bits, que se divide entre este registro y el ACC
XAR4	Pasa un argumento de 16 bits a una función o una dirección de alguna localidad de memoria
XAR5	Transfiere la dirección de una localidad de memoria o un argumento de 16 bits
XAR6	Retiene la dirección de una estructura, para ser utilizada al regreso del programa principal de C/C++
R0H ... R3H	Traslada argumentos de 32 bits tipo flotante El registro R0H puede retener un valor de 32 bits de tipo flotante para utilizarlo en el programa de C/C++
SP	El Stack Pointer (SP) habilita el uso de un stack de software en la memoria de datos. Por lo que al terminarse los registros posibles para el paso de variables de entrada a funciones, el SP selecciona las siguientes localidades de memoria baja que siguen, para poder llevar los datos restantes a la función de ensamblador

6.2. Uso de funciones en lenguaje ensamblador desde C/C++

Independientemente de como se distribuya un programa en lenguaje C/C++, al ser el principal, el compilador C28x le permite utilizar funciones escritas en ensamblador en archivos *.asm*. Esta característica tiene por objetivo optimizar secciones de código para el DSP TMS320C28x, que puedan tener un mejor desempeño al declararse por el usuario en lugar del compilador, siempre y cuando se sigan y respeten las siguientes reglas y convenciones

1. *Declarar como elemento externo* el nombre de la función o funciones en lenguaje ensamblador, antes del bloque principal del programa de C/C++.

2. En al archivo *.asm*, se debe declarar como **.global** el nombre de la función o funciones que contenga el archivo, variables, arreglos y constantes.

3. Cualquier nombre de una función definida por el usuario, debe iniciar con un guión bajo.

A continuación se muestra un ejemplo de las tres reglas descritas previamente para acceder a funciones en lenguaje ensamblador desde un programa en C/C++.

```
/* Programa principal
    archivo .c/.cpp   */

extern "C"{
//Declaración de la función externa en lenguaje ensamblador
extern int asmfunc(int a);
//Declaración de una variable global
int gVar = 0;
}

void main(){
   int i =5;
   i = asmfunc(i);
   while (1); // Ciclo infinito
}
```

```
* Subrutina o función en lenguaje ensamblador
*             archivo .asm

        .global _gVar
        .global _asmfunc

_asmfunc:
        MOVZ    DP,#_gVar
        ADDB    AL,#5
        MOV     @_gVar,AL

        LRETR
```

Del ejemplo, en el programa de C/C++ antes de la función principal se encuentra el bloque **extern "C"**, el cual contiene la declaración de la función y variable que se utilizan en el procedimiento escrito en el archivo *.asm*.

El ***orden de traslado de variables*** del programa principal a una subrutina, depende del tipo y cantidad de variables que se declaren. La Tabla 6.2 contiene algunas opciones posibles de variables de entrada, especificando la **jerarquía de transferencia**.

Tabla 6.2: Esquema de transferencia de variables del programa principal en C/C++ hacia una función escrita en ensamblador.

Variables de entrada	**Registros de transferencia**	**Descripción**
func(float a, float b, float c, ...)	R0H, R1H, R2H, R3H, SP	Al declarar variables tipo flotante los primeros cuatro valores se transfieren a los registros del FPU. Si hay más variables pasan por el stack
func(long double x, long double y, ...)	-	Argumentos de punto flotante de 64 bits se transfieren por su dirección de memoria
func(long long h, long long j, ...)	ACC/P, SP (Little endian)	Si son tipo entero de 64 bits, el primer dato es transferido en dos partes; ACC retiene la parte alta P pasa la parte baja del dato de 64 bits Si hay más variables del mismo tipo pasa por el stack como Little Endian[1]
func(long a, long long b, int c, float p)	ACC, SP (Little endian)	Si se declara algún dato de 32 bits, el primero de ellos pasa por el ACC y los demás por el stack
*func(int *a, int *b, long * c)*	XAR4, XAR5, SP	Al ser apuntadores, los primeros dos pasan respectivamente por XAR4 y XAR5, el resto pasa por el stack
func(int x, int y, int w, int z)	AH, AL, XAR4, XAR5, SP	Si se declaran únicamente datos de 16 bits, los primeros cuatro pasan en el orden de registros especificado, y los restantes se transfieren por el stack
*func(int x, int y, int *w, long *z)*	AH, AL, XAR4, XAR5	Al combinar variables con apuntadores, el uso de los registros lo determina la primer variable acorde a las combinaciones anteriores
*func(long x, int *y)*	ACC, XAR4	En este caso el dato de 32 bits pasa por ACC y el apuntador por el primer registro auxiliar destinado por el CPU

[1] *Little Endian: Primero almacena la parte baja y posteriormente parte alta del dato*

Al finalizar la ejecución de una función, como en cualquier programa también es posible enviar resultados. Particularmente, acorde al protocolo para el manejo de los registros por el compilador C28x, solo es posible regresar una variable por los registros especificados en la Tabla 6.3, en función del tipo de dato.

Tabla 6.3: Lista de registros para regresar valores al retorno de una función en lenguaje ensamblador.

Registro	**Dato de retorno**
AL	Valor tipo entero de 16 bits
ACC	Valor tipo entero de 32 bits
ACC/P	Valor tipo entero de 64 bits
XAR4	Apuntador de 32 bits
XAR6	Dirección de una estructura
R0H	Valor flotante de 32 bits

En caso de necesitar comunicar más de un dato al programa principal, al retorno de una función, es posible utilizar la definición de variables compartidas, como se verá en la Sección 6.3. Después de la última instrucción de una función en ensamblador, se debe escribir **LRETR** para regresar al programa principal de C/C++, como se puede observar en la rutina de ejemplo *asmfunc*.

NOTA. Sin importar que tengan diferente terminación, el nombre del programa principal *.c* debe ser único en el proyecto, es decir, que ningún otro archivo *.asm* debe tener el mismo nombre entre ellos y el *.c*.

6.3. Acceso compartido de variables

Cada una de las variables declaradas en lenguaje C/C++ o ensamblador, se le asigna una dirección de memoria, acorde a su tipo, todas estas direcciones se encuentran en la *tabla de símbolos* [9], y las constantes declaradas en códigos *.asm*, con la directiva *.set*, no ocupan espacio en memoria.

En la Sección 2.3 del *Capítulo* 2, se especificaron las secciones para alojar variables, donde *.ebss* es normalmente utilizada, *.data* es una opción alterna que tiene que ser declarada en el mapa de memoria en caso de ser utilizada, como se muestra en algunos programas y archivos *.cmd* del *Capítulo* 4. Adicionalmente, también es posible declarar una sección denominada como *.usect*.

Para acceder a variables declaradas en lenguaje ensamblador, desde un programa escrito en C/C++ se deben seguir los siguientes lineamientos

1. Si se utiliza un sección de datos diferente a **.ebss**, especificarla antes de iniciar la declaración de variables.

2. Utilizar la directiva **.global** o **.def** para hacer externa la declaración de la variable.

3. Iniciar el nombre de la variable por un guión bajo, por ejemplo

```
.global _var1
.def _var2
```

4. En lenguaje C/C++, declarar la variable como externa, para poder utilizarla de forma convencional, por ejemplo

```
extern int var1;
var1 = 0;
```

Y para acceder a constantes definidas en lenguaje ensamblador, desde un código de C/C++ se sigue una metodología similar con las variables, con la diferencia de tener que utilizar el *operador de dirección* & para dirigirse al valor deseado, como se muestra en el siguiente ejemplo

```
*** Declaración de la constante en lenguaje ensamblador ***

_samples        .set 10000              ; Definición de la constante
                .global _samples        ; Hace externa la definición

/* Declaración y uso de variables externas en C/C++ */

extern int samples;
#define n_samples ((int) (&samples))

.
.
.

for (int i=0; i<n_samples; i++)
```

Más información referente al acceso de variables entre lenguajes y su declaración, puede ser consultada en los manuales [10] y [9].

6.4. Instrucciones de ensamblador en línea

En los programas de C/C++ es posible utilizar instrucciones de lenguaje ensamblador, en una sola línea. Al tener esta capacidad el compilador ensambla directamente las instrucciones declaradas en el código de ejecución, al utilizar la siguiente declaración

```
__asm(``instrucción o texto en ensamblador'');
```

Un posible uso de esta herramienta, es imprimir en consola comentarios en la ventana de compilación, para ello, simplemente inicia el comentario de código con un ";" como se muestra a continuación:

```
__asm(``;*** ciclo sin errores ***'');
```

Es recomendable evitar el uso de instrucciones de salto o declarar etiquetas, porque puede afectar la manipulación de variables obteniendo resultados no deseados. También en caso de utilizar la instrucción **RPT**, toda la declaración debe hacerse en una sola línea utilizando los símbolos de salto de línea y tabulación, para incrustar adecuadamente las instrucciones en el código, por ejemplo:

```
__asm(``\tRPT #10\n\t||MAC P, *XAR4++, *XAR7++'');
```

6.5. Acceso intrínseco a instrucciones y rutinas de ensamblador

El compilador C28x reconoce ciertas instrucciones de lenguaje ensamblador que son inherentes a operadores o funciones de C/C++, agilizando en algunos casos su declaración convencional, haciendo uso de la arquitectura del DSP.

Esta implementación se auxilia del acceso de variables descrito en la Sección 6.4 y adicionalmente, cuenta con funciones de uso general y especifico, como puede ser la realización de una operación **MAC**. En la Sección 7.5.6 del manual [10] se describen los operadores y rutinas intrínsecas que pueden utilizarse en un programa en C/C++. Algunos ejemplos se muestran en la Tabla 6.4.

Al utilizar estas instrucciones y rutinas, se hace visible el uso de la arquitectura del DSP en un programa de C/C++. Su uso es opcional, ya que *Texas Instrument*, garantiza que su compilador está optimizado para aprovechar los recursos de sus diferentes dispositivos.

[2]src: fuente; dst: destino

Tabla 6.4: Ejemplos de operadores y rutinas intrínsecas al compilador C28x [10]

Sintaxis de la instrucción	**Proceso en lenguaje ensamblador**	**Descripción**
unsigned int __disable_interrupts();	PUSH ST1 SETC INTM,DBGM POP reg16	Deshabilita interrupciones y regresa el valor que tenía el vector de interrupción.
unsigned int __enable_interrupts();	PUSH ST1 CLRC INTM,DBGM POP reg16	Habilita interrupciones y regresa el valor que tenía el vector de interrupción.
long dst = __norm32(long src, int * shift);	CSB ACC LSLL ACC,T MOV @shift, T	Normaliza la variable src colocando el resultado en dst, y en shift escribe la cantidad de corrimientos realizados.
int __rpt_norm_inc(long src, int dst, int count);[2]	MOV ARx, dst RPT #count ‖ NORM ACC,ARx++	Repite la normalización del valor en ACC count+1 veces, teniendo de intervalo límite desde 0 a 255
void __f32_max_idx(double &dst, double src, double &idx_dst, double idx_src);	MAXF32 dst,src ‖ MOV32 idx_dst,idx_src	Si *src>dst* copia src en dst, y copia idx_src en idx_dst.
void __f32_min_idx(double &dst, double src, double &idx_dst, double idx_src);	MINF32 dst,src ‖ MOV32 idx_dst,idx_src	Si *src<dst* copia src en dst, y copia idx_src en idx_dst.

6.6. Ejemplo de Filtros FIR e IIR

En el *Capítulo* 4 en la Sección 4.4, se explicaron las implementaciones de los filtros FIR e IIR en lenguaje ensamblador. Se retomará la base de esos programas para ejemplificar el uso de funciones en lenguaje ensamblador, desde un código escrito en C/C++.

La aplicación consiste de un programa principal escrito en C, que utiliza dos funciones externas correspondientes a cada uno de los filtros. Estas rutinas están codificadas en lenguaje ensamblador en archivos separados, y reciben un dato de la señal a filtrar cada vez que son invocadas. El resultado de cada función se guarda en vectores separados, para observar y comparar de forma gráfica la respuesta de los filtros.

La señal a filtrar se genera por la ecuación 4.19 y muestreada a $f_s = 1kHz$, considerando las frecuencias $f_0 = 27Hz$, $f_1 = 37Hz$ y $f_2 = 47Hz$ y 1000 puntos. El objetivo de los filtros es suprimir o atenuar, la componente f_1 aplicando un *filtro supresor de banda o notch* tipo FIR e IIR.

El primer paso es crear un proyecto nuevo, cuya función principal esté en un programa de C, como se explicó en la Sección 2.1. El archivo *.c* que se genera, debe tener el siguiente código:

```
/*
 ****        Comparación de filtros supresores de banda          ****
 *                        FIR e IIR,
 *              combinando los lenguajes C y ASM.
*/

// Coeficientes del filtro FIR
#include "cof_hQ28_sup_37_fir.h"

// Cantidad de datos de la secuencia de entrada
#define samples 1000

// Declaración de funciones escritas en ensamblador
extern long firFilter( long, long * );
extern int iirFilter( long );

// Vectores para guardar en memoria cada una de las señales filtradas
long ynFir[samples] = {0};
int  ynIir[samples] = {0};


void main(void) {

        // Vector para alojar la secuencia de datos de entrada
        long xn[samples] = {0};
        int i = 0;

        // No. 1 en formato Q28
        //xn[0] = 268435456;

        // Ciclo de cálculo y aplicación de los filtros
        for (i=0; i<samples; i++){

                ynFir[i] = firFilter( xn[i], &h1[0] );
                ynIir[i] = iirFilter( xn[i] );
        }

        while(1){}     // Ciclo infinito
}
```

Previo a la función principal del programa, se declaran de forma global:

1. El número de muestras de la señal a filtrar *samples*, para poder ajustar dicha cifra en cualquier momento.

2. Las funciones *firFilter* e *iirFilter* que aplican cada uno de los filtros, siguiendo los lineamientos descritos en la Sección 6.2.

3. Los arreglos *ynFir()* e *ynIir()* que guardarán la respuesta de cada filtro. Son globales para poder recibir datos del retorno de rutina desde ensamblador.

Después de forma local, se declara el arreglo para guardar la señal de entrada y una variable para usarse en el ciclo **for**. En esta fase del programa, punto por punto de la señal de entrada, se filtra utilizando cada una de las funciones.

Para definir las rutinas de implementación de cada filtro, seleccionando la carpeta del proyecto en el explorador de CCS, se crearán dos nuevos archivos *.asm*, uno para cada filtro. En este ejemplo, el código del filtro FIR se nombró *fir_32bits_asm-c.asm* y contiene lo siguiente

```
* Función en lenguaje ensamblador para aplicar un filtro FIR
* de 255 coeficientes a 32 bits, en un formato
* de punto entero de 28 bits

        .global _firFilter

        .data
Nw      .set 255            ; Cantidad de coeficientes del filtro
xb      .space 32*Nw        ; Espacio reservado para buffer x
xbf     .long 0             ; Ultima localidad del buffer x
aux     .long 0             ; variable auxiliar

* Rutina que realiza el cálculo de un filtro FIR supresor,
* con 255 coeficientes cada etapa

_firFilter
* Cálculo de la primera etapa del filtro FIR
        SETC SXM            ; Habilita el uso de aritmética de signo extendido
        MOVW DP,#xb         ; Direccionamiento a la página de memoria que
                            ; contiene la variable xb

        MOVL @xb,ACC        ; El valor pasado a la función, se coloca en ACC
        MOVL XAR7,#xb       ; Direccionamiento indirecto a xb
        MOVL XAR0,#aux      ; Direccionamiento indirecto a la var. aux
        ZAPA

        RPT #Nw-1           ; Ciclo de cálculo de la convolución
        || QMACL P,*XAR4++,*XAR7++

        ADDL ACC,P          ; Acumula el último producto realizado
        LSL ACC,#4          ; Ajuste de Qi. El resultado original está en formato
                            ; Qacc = Q28+Q28 =Q56 que a 32 bits es Q24 + 4 => Q28
        MOVL *XAR0,ACC      ; Respaldo del valor obtenido del filtro
```

```
* Desplazamiento del buffer xb
* Se apunta a la localidad extra al final del buffer xb,
* y a la penúltima de xb, para copiar los datos de la
* parte final a la inicial en retroceso
        MOVL XAR4,#xbf
        MOVL XAR7,#xbf-2
        MOV AR5,#Nw-1

move_buffer
        MOVL ACC,*--XAR7
        MOVL *--XAR4,ACC
        BANZ move_buffer,AR5--

        MOVL ACC,*XAR0    ; Se coloca el valor resultante del filtro en ACC
                          ; para que se regrese a C

        LRETR             ; Regreso al programa principal
```

La función *_firFilter* trabaja con datos de 32 bits con un formato de punto entero $Q_i = 28$ de entrada y salida, por obtener los mejores resultados registrados en el *Capítulo* 4. La diferencia con el programa presentado en la Sección 4.4, es que se omite el ciclo **BANZ**, porque se sustituye por el **for** en el programa principal. Al entrar y salir de la función, el dato de entrada y el resultado obtenido por la rutina es transferido entre los lenguajes por el **ACC**, por ello la función se declara tipo **long**.

De forma análoga al filtro FIR, se define la rutina del filtro supresor IIR, cuyo código se guardo en el archivo *iir_16bits_asm-c.asm* como se muestra a continuación:

```
* Función en lenguaje ensamblador para aplicar un filtro IIR
* de segundo orden cuyo modelo es
*
* y(n)=b0*x(n)+b1*x(n-1)+b2*x(n-2)+a1*y(n-1)+a2*y(n-2)
*
* Nota: Los coeficientes a deben ser cargados con signo
* opuesto al calculado

        .global _iirFilter

        .data
Noper   .set 5          ; Cantidad de operaciones
Ncb     .set 3          ; Número de coeficientes b
Nca     .set 2          ; Número de coeficientes a

* Coeficientes del filtro
cofb    .int 3972,-7732,3972    ; Coeficientes b0,b1,b2
cofa    .int 7732,-3847         ; Coeficientes a1,a2
```

```
* Buffer para filtro IIR
bufx    .space Ncb*16   ; Buffer para guardar xn, x(n-1), x(n-2)
bufy1   .word 0         ; Localidad de memoria para y(n-1)
bufy2   .word 0         ; Localidad de memoria para y(n-2)
bufbas  .word 0         ; Localidad final de los retardos de yn


* Rutina del filtro supresor IIR
_iirFilter

        SETC SXM        ; Habilita el uso de aritmética de signo
                        ; extendido
        MOVW DP,#bufx   ; Direccionamiento a la página de memoria que
                        ; contiene la variable bufx
        SPM #0          ; Define cero corrimientos al postoperar con el reg. P

        MOVL XAR4,#cofb ; Direccionamiento al vector de coeficientes b
        MOV @bufx,AH    ; Escribe en el buffer x la parte alta de ACC, que
                        ; corresponde a la mitad del dato que pasa a la función

        MOVL XAR7,#bufx                 ; Direccionamiento al buffer x
        ZAPA

        RPT #Noper-1    ; Ciclo de operaciones acorde a la ecuación del filtro
        || MAC P,*XAR4++,*XAR7++        ; Multiplicaciones de cada uno de los
                                        ; factores que se suman en la ecuación

        ADDL ACC,P<<PM                  ; Suma el último producto al ACC
        LSL ACC,#4                      ; Ajuste de formato para obtener Q28


* Actualización de retardos de yn
        MOVL XAR7,#bufy2                ; Direccionamiento a la localidad y(n-2)

        RPT #Noper-1                    ; Ciclo para re acomodo de términos
        || DMOV *--XAR7                 ; Actualización de retardos y(n)

        MOV @bufy1,AH                   ; Escribe y(n-1) = y(n)

        MOV AL,AH                       ; Escribe en la pare baja de ACC el
                                        ; resultado obtenido para regresarlo al
                                        ; programa principal

        LRETR                           ; Regreso al programa principal
```

Con el objetivo de mostrar el funcionamiento del paso de datos en funciones, el filtro IIR se implementó considerando datos de 16 bits con un formato de punto entero $Q_i = 12$. En este caso, el dato que se transfiere desde el programa principal a la rutina de ensamblador,

se hace por el registro **AH** y el resultado se transfiere al código en lenguaje *C*, por el registro **AL**, razón por la que la función se declara como **int**.

Los coeficientes de los filtros FIR se definieron en un arreglo de 255 elementos, denominado *h1* declarado en el archivo *cof_hQ28_sup_37_fir.h* el cual es incluido al inicio del programa principal. De esta manera, los datos quedan declarados de forma global y son grabados en la memoria desde la carga del programa.

Respecto a los coeficientes del filtro de segundo orden IIR, estos fueron declarados directamente en el archivo *iir_16bits_asm-c.asm*, en formato $Q_i = 12$, divididos en dos arreglos; $cofb = [b_0, b_1, b_2]$ y $cofa = [a_1, a_1]$. Los datos son alojados en el arreglo de forma secuencial.

Para probar el funcionamiento de la aplicación, al entrar al modo *Debug*, cargando el programa a la tarjeta, se debe declarar el arreglo *xn*, para escribir en la memoria, los puntos de la señal de entrada. Hecho lo anterior, se puede colocar un *breakpoint* en la instrucción **while(1)** para utilizar la herramienta que gráfica el espectro de magnitud, y observar la respuesta obtenida por cada uno de los filtros.

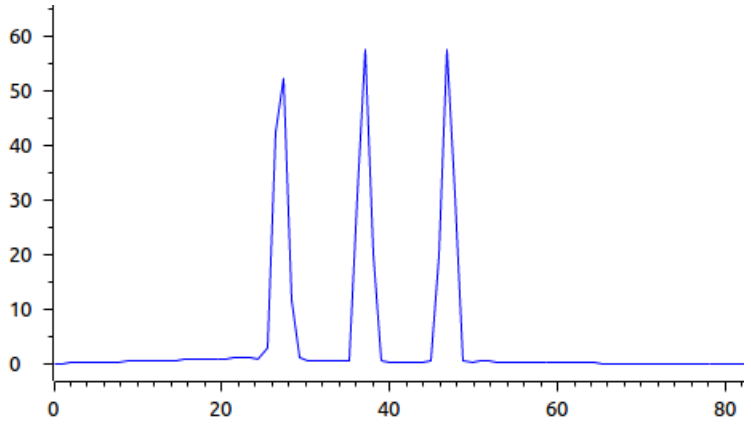
6.6.1. Resultados de la implementación

El espectro en magnitud de la señal de entrada $x\,(n)$ se muestra en la Figura 6.2a, donde se pueden observar las tres componentes de frecuencia de 27, 37 y 47 Hz. Las funciones que aplicaron cada uno de los filtros FIR e IIR fueron obtenidas considerando como base los programas de la Sección 4.4, obteniendo las mismas respuestas al impulso.

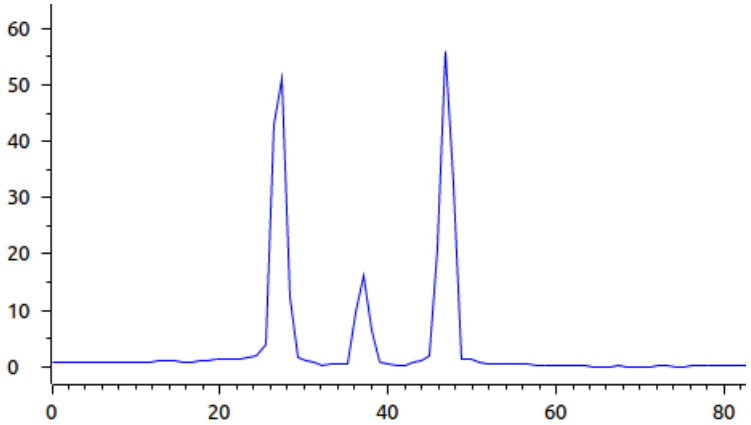
Las Figuras 6.2b y 6.2c, corresponden al espectro en magnitud de la señal filtrada, en donde es visible que el filtro IIR de 16 bits tuvo un mayor efecto al casi suprimir por completo la componente de 37 Hz. En contraste, el filtro FIR solo atenúo alrededor de 66 % de la magnitud original. Analizando los datos en Octave, se obtuvieron las mismas atenuaciones que se reportados en la Sección 4.4, y registrados para esta aplicación en la Tabla 6.5.

Tabla 6.5: Atenuación en db de la componente espectral de 37 Hz, de la señal $x\,(n)$ filtrada por los dos tipos de filtros.

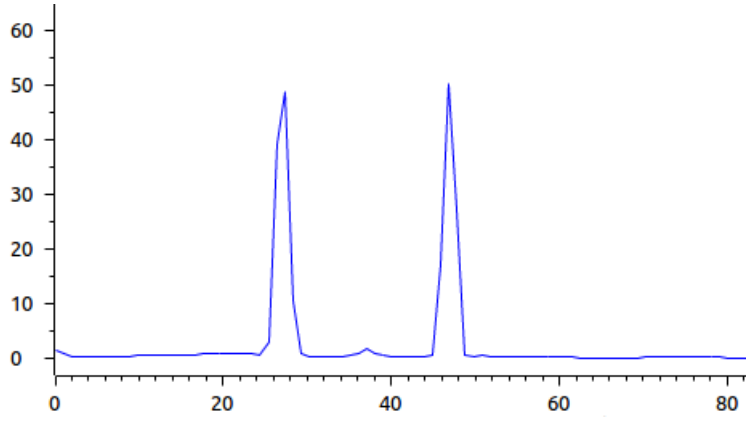
Filtro notch	**Formato de datos**	**Atenuación db**
FIR orden 255	32 bits / $Q_i = 28$	-10.8970
IIR 2do orden	16 bits / $Q_i = 12$	-38.3427



(a) Espectro de magnitud de la señal a filtrar generada por la ecuación 4.19.



(b) Espectro de magnitud de la señal filtrada calculada por el filtro supresor FIR.



(c) Espectro de magnitud de la señal filtrada calculada por el filtro supresor IIR.

Figura 6.1: Comparación del contenido espectral de los resultados obtenidos por la aplicación ejemplo que combina lenguaje C y ensamblador.

6.7. Ejemplo de Filtro FIR utilizando una estructura

Al respetar las reglas y convenciones de acceso de variables y uso de funciones de lenguaje ensamblador, en cualquier programa de C/C++, se tiene la libertad de desarrollar aplicaciones utilizando las diversas herramientas disponibles de dichos lenguajes. Para demostrarlo, en esta Sección se expondrá y describirá la implementación de un filtro FIR utilizando una estructura de lenguaje C.

En esta implementación, para definir la información involucrada en el filtro FIR, se propuso la estructura definida en el archivo *FIR.h*.

```
/*
 * FIR.h
 *
 * Estructura de datos involucrados
 * en la implementación del filtro FIR
 */

#ifndef FIR_H_
#define FIR_H_

#ifdef __cplusplus
extern "C" {
#endif

typedef struct {
        const int16 *B;    // apuntador a los coeficientes del filtro
        int16 *xd;         // apuntador a las muestras retrasadas del filtro
        int16 x;           // dato de entrada al filtro
        int16 y;           // dato de salida del filtro
        int16 Nb;          // orden del filtro (número de coeficientes)
}FIR;

typedef FIR *FIR_Handle;

//Funciones a definidas en lenguaje ensamblador
extern void FIR_init(void *);       // funcion para iniciar el filtro
extern void FIR_filter(void *);     // función que realiza el filtrado

#ifdef __cplusplus
}
#endif
#endif /* FIR_H_ */
```

Los coeficientes del filtro se declaran como un arreglo definido en un archivo *.h*, como se mostró en la Sección 6.6, por ello en la estructura se define un apuntador para acceder a cada uno de los valores. El arreglo para guardar las muestras de entrada de tiempos anteriores al

actual, se declaró en el programa principal, agregando otro apuntador para dicho arreglo. El resto de las variables son para especificar la cantidad de coeficientes del filtro y los datos de entrada y salida al proceso. Todos estos componentes de la estructura son etiquetados para su acceso, como *FIR.*

Posteriormente se define un apuntador del tipo de la estructura, para agilizar su acceso y especificar solo una dirección como variable de entrada a las funciones. Y por último, estás se declaran como externas escritas en lenguaje ensamblador, en el archivo *FIR16Q15.asm* el cual contiene lo siguiente:

```
;
; En este archivo se muestran las funciones empleadas para filtrar utilizando
; fitros FIR, estas funciones se llaman desde C empleando una estructura
; tipo FIR (ver FIR.h), y se direccionan empleando el registro
; XAR4 con sus respectivos ``offsets":
;
;
; typedef struct {                    <------- XAR4
;       const int16 *B;               <------- XAR4[0]
;       int16 *xd;                    <------- XAR4[2]
;       int16 x;                      <------- XAR4[4]
;       int16 y;                      <------- XAR4[5]
;       int16 Nb;                     <------- XAR4[6]
;
; }FIR
;
; Este filtro opera con datos de 16 bits en Q15
;
;

        .def _FIR_init
        .def _FIR_filter
        .text

;-------------------------------------------------------------------------------
; Escribe cero en todo el arreglo correspondiente al buffer de muestras
; de entrada retardadas xd, para asegurar la respuesta al impulso
;-------------------------------------------------------------------------------
_FIR_init:

        ZAPA
        MOVL XAR2,*+XAR4[2]       ; XAR2 = *xd
        MOV AL,*+XAR4[6]          ; AL = No. de coefcientes

        RPT AL
        || MOV *XAR2++,#0         ; xd[*XAR2] = 0
```

```
        NOP
        LRETR

;-----------------------------------------------------------------------
; Esta función realiza el filtrado de la señal, para esto se recuperan
; * El valor de entrada                                  (x[n]->XAR4[4])
; * Los coeficientes del filtro                          (*B->XAR4)
; * El buffer con las muestras retrasadas de la señal    (xd->XAR4[2])
; * El orden del filtro                                  (N->XAR4[6])
; * Y el resultado se coloca en la salida                (y->XAR4[5])
;-----------------------------------------------------------------------
_FIR_filter:

        SETC SXM                ; Modo extensión de signo
        SETC OVM
        SPM  #0

        MOVL XAR2,*+XAR4[0]     ; XAR2 = *B
        MOVL XAR7,*+XAR4[2]     ; XAR3 = *xd

        MOV AL,*+XAR4[4]        ; AL = x
        MOV *XAR7,AL            ; *xd[0] = x

        MOVZ AR0,*+XAR4[6]      ; AR0 = N

        ZAPA

        RPT AR0
        || MAC P,*XAR2++,*XAR7++

        ADDL ACC,P              ; Acumula el último producto
        LSL ACC,#1              ; Ajuste a formato Q15

       MOVH *+XAR4[5],ACC       ; y=filtrada


; Ciclo para reacomodo de datos en el buffer xd
        RPT AR0
        || DMOV *--XAR7         ; Mueve N veces los datos del buffer

        LRETR
```

Al estar definidas de forma secuencial las variables que forman la estructura, conociendo su dirección inicial en la memoria y el tipo de datos especificado, es posible dirigirse a cada elemento, sumando un "offset" a la primer localidad declarada que en este caso corresponde al apuntador de coeficientes. Utilizando la sintaxis ***+XARN[#]**, se logra direccionar a las localidades subsecuentes que ocupa la estructura, como es mostrado en el anterior código.

Y por último, el programa principal en lenguaje C *firPasoBanda_C-ASM.c*

```
/*
 *                Filtro FIR usando una estructura
 *                  utilizando lenguajes C y ASM
 *
 * Los coeficientes corresponden a un filtro pasa banda, utilizando
 * datos de 16 bits en Qi=15, se obtiene la respuesta al impulso.
 */

// Archivo donde se definen tipos de variables, registros y configuraciones
// de periféricos para un proyecto que use la familia F28x
#include "F28x_Project.h"

#include "FIR.h"

// Archivo de coeficientes del filtro paso banda
#include "B2.h"

// Objeto tipo FIR
FIR filter1;

// Apuntador al objeto estructurado
FIR_Handle hnd_filter1 = &filter1;

// Buffer de muestras retartadas x(n)
int16 xd1[400];
// Arreglo para guardar la señal filtrada
int16 out[512];

void main (void){

     Uint16 i;

     // Asignación de valores para aplicar el filtro
     hnd_filter1->Nb = 400;     // No. de coef. del filtro
     hnd_filter1->B = B2;       // Asignación al vector de coeficientes

     hnd_filter1->x = 32767;    // x[0] = 1 en formato Qi=15
     hnd_filter1->xd = xd1;     // xd -> xd1[0]

     // Cálculo de la respuesta del filtro para x[0]
     FIR_init(hnd_filter1);
     FIR_filter(hnd_filter1);

     // Guarda el resultado obtenido por el filtro
     out[0] = hnd_filter1->y;

     // Escribe cero en la variable de entrada, para todo lo que
```

```
    // resta del cálculo del filtro
    hnd_filter1->x = 0;

    for(i=1;i<2048;i++){     // Ciclo de cálculo
        FIR_filter(hnd_filter1);
        out[i] = hnd_filter1->y;
    }
    while(1);                // Ciclo infinito
}
```

Para compilar este proyecto, es necesario realizar toda la configuración especificada en la Sección 2.2.2, porque se utilizan tipos de variables definidas en diversos archivos llamados por *F28x_Project.h*.

El programa principal consiste asignar los datos de la estructura, para aplicar el filtro FIR, declarando de forma global todas las variables que interactúan con las funciones *FIR_init* y *FIR_filter*. La metodología de la aplicación es similar a la presentada en la Sección 6.6, en este caso ambas funciones reciben como dato de entrada una dirección, por el registro **XAR4**, correspondiente a la primera localidad de memoria asignada a la estructura.

Al definir en la estructura solo un término de la señal de entrada x, para obtener la respuesta al impulso primero se escribe ceros en todas las localidades del buffer *xd1*, llamando a la función *FIR_init*. Después se calcula la respuesta para el término $x(0) = 32767$, que es igual a uno en formato $Q_i = 15$ al aplicar *FIR_filter* una vez, y posteriormente se asigna $x = 0$ para obtener los términos restantes. En caso de otra señal de entrada, dentro del ciclo **for** se tendría que estar asignando el valor de x previo a llamar a la función *FIR_filter*.

6.7.1. Resultados de la implementación

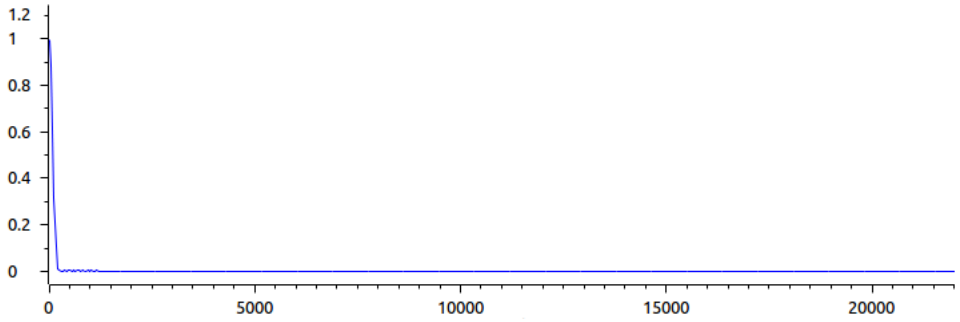
La aplicación descrita anteriormente, puede implementar cualquier tipo de filtro tipo FIR, para demostrarlo se obtuvo la respuesta al impulso de los siguientes tres filtros

1. Filtro paso bajas con frecuencia de corte 60 Hz
   Vector de coeficientes B1 definido en *B1.h*

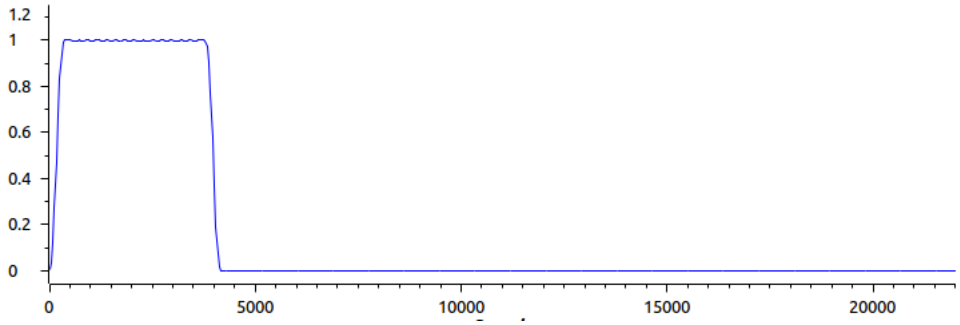
2. Filtro pasa banda definida de $400 < f_{bw} < 4500$ Hz
   Vector de coeficientes B2 definido en *B2.h*

3. Filtro paso alta con frecuencia de corte 8 kHz
   Vector de coeficientes B3 definido en *B3.h*

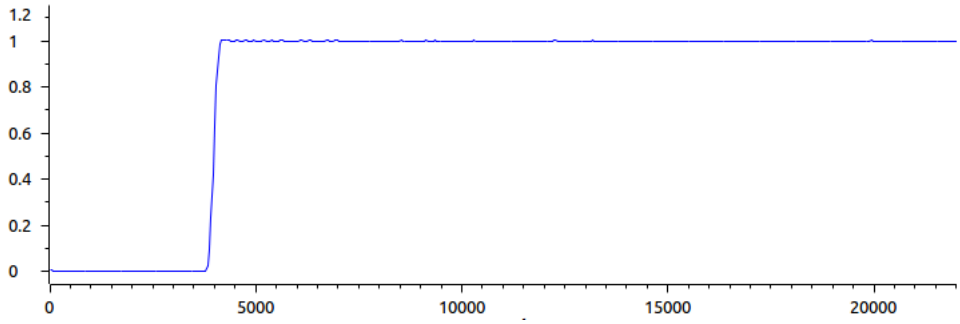
En todos los casos se consideró una frecuencia de muestreo de $f_s = 44100$ Hz, todos los filtros son de orden 400 y los coeficientes se calcularon por el método de ventanas y serie de



(a) Espectro de magnitud de la respuesta al impulso del filtro pasa baja con frecuencia de corte $f_c = 60$ Hz.



(b) Espectro de magnitud de la respuesta al impulso del filtro pasa banda definida en $400 < f_{bw} < 4500$ Hz.



(c) Espectro de magnitud de la respuesta al impulso del filtro paso altas con frecuencia de corte $f_c = 8$ kHz.

Figura 6.2: Espectros de magnitud de la respuesta al impulso de cada uno de los filtros FIR implementados en lenguaje C utilizando una estructura.

Fourier implementado con el método disponible en Octave. Para una señal impulso de 2048 términos, el espectro de magnitud obtenido en cada caso se muestra en la Figura 6.2.

Los archivos *.h* que contienen los coeficientes, se encuentran disponibles en el sitio `http://odin.fi-b.unam.mx/labdsp`.

6.8. Resumen del capítulo

Desde un enfoque práctico de aplicación, a lo largo de este capítulo se explicaron las reglas y maneras de poder utilizar funciones escritas en lenguaje ensamblador desde un programa de C/C++, y también cómo acceder a variables definidas en cualquiera de los dos lenguajes. Esta virtud del compilador C28x permite combinar las implementaciones presentadas en el Capítulo 4 y periféricos utilizando Control Suite para desarrollar aplicaciones en tiempo real, con señales adquiridas mediante diferentes sensores, como las que se mostrarán en el siguiente Capítulo.

La combinación de lenguajes es una herramienta opcional, que permite al usuario elegir como implementar operaciones de procesamiento o configuración de periféricos, utilizando y manejando directamente la arquitectura del dispositivo que se este utilizando, aprovechando en medida de lo posible todas sus características.

Ejercicios propuestos

1. Considerando la aplicación ejemplo de la Sección 6.6, haga las modificaciones pertinentes para aplicar dos filtros en cascada FIR e IIR, de tal forma que logré suprimir la componente espectral de 27 Hz, de la señal $x(n)$ definida por la ecuación 4.19.

2. Realice las modificaciones necesarias para aplicar el filtro FIR presentado en la Sección 6.6, con el programa de la Sección 6.7 utilizando la señal de entrada definida por la ecuación 4.19.

3. Utilizando la metodología de la aplicación de la Sección 6.7, desarrolle el primer ejercicio propuesto en esta lista.

4. Adecue el programa del algoritmo de Goertzel para poder obtener el espectro de las señales filtradas en los incisos anteriores, definiendo las respectiva función en lenguaje ensamblador considerando el código presentado en la Sección 4.7.

Capítulo 7

Aplicaciones en Tiempo Real

Uno de los retos más importantes dentro del área de procesamiento digital de señales es poder implementar diferentes aplicaciones en tiempo real. La respuesta de un sistema que interacciona con su entorno físico, es decir, que está inmerso en un ambiente real, no solamente requiere ser óptima, sino que además debe responder a los estímulos del entorno dentro de un plazo determinado, realizando el procesamiento con un tiempo de respuesta inferior al requerido por la aplicación. Los tiempos de respuesta varían dependiendo del tipo de aplicaciones.

En un sistema real, es necesario evaluar los algoritmos a implementar analizando el número de operaciones que éstos demanden en su ejecución para aprovechar al máximo los recursos que ofrecen las tarjetas de desarrollo y verificar si es posible ejecutar dicha aplicación en tiempo real en el dispositivo.

Este capítulo tiene como objetivo integrar los conceptos vistos en los capítulos anteriores para ejecutar aplicaciones de procesamiento de audio en tiempo real empleando el DSP Delfino TMS320F28377s. Se diseñó una tarjeta de acoplamiento electrónico para poder utilizar los puertos del DSP en las aplicaciones propuestas del presente trabajo, dicha tarjeta se expone en la Sección 7.1. Posteriormente se exponen diferentes aplicaciones empleando el DSP Delfino y la tarjeta de expansión, estas aplicaciones se encuentran programadas en lenguaje C, ensamblador y la mezcla de los mismos con el objetivo de mejorar el tiempo de respuesta

de los diferentes procesos implementados y tener un mejor manejo de memoria.

7.1. Tarjeta de Expansión

En el laboratorio de Procesamiento Digital de Señales de la UNAM se ha desarrollado una tarjeta de expansión para la LaunchPad LAUNCHXL-F28377S, de tal manera que los alumnos puedan realizar aplicaciones sencillas de procesamiento de audio y de voz. En esta sección se describen los elementos principales de dicha tarjeta así como su función, diagramas eléctricos y simulaciones.

La tarjeta de expansión se alimenta con voltajes superiores a 14V y cuenta con un regulador de voltaje de 12V, el cual alimenta los circuitos analógicos, así mismo cuenta con múltiples puntos de prueba (TP) conectados al voltaje de referencia para poder medir las señales de entrada y salida del DSP. En la Figura 7.1 se muestra un circuito esquemático del módulo de alimentación de dicha tarjeta.

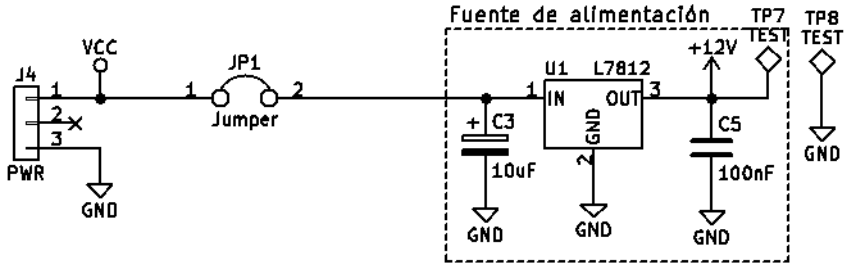


Figura 7.1: Módulo de alimentación de la tarjeta de expansión para el DSP

Como se mencionó previamente, esta tarjeta fue diseñada para realizar aplicaciones de procesamiento de audio, sin embargo, se agregaron ciertos periféricos que sirven para probar los diversos programas realizados en los capítulos anteriores, entre ellos se encuentran dos botones los cuales sirven para probar las entradas digitales además de las interrupciones externas. Estos botones cuentan con una etapa de filtrado analógico con el objetivo de evitar los "rebotes" mecánicos que presentan los botones al ser presionados. Las conexiones de dichos botones se encuentran conectados de la siguiente manera:

- SW1: GPIO2.
- SW2: GPIO3.

En la Figura 7.2 se muestra el diagrama de conexiones utilizado en la tarjeta de expansión para ambos botones con los pines GPIO del DSP. Se puede observar que cuando el usuario presiona un botón de la tarjeta, el pin del GPIO tendrá un nivel lógico bajo, esto se debe a que las entradas GPIO del DPS son negadas.

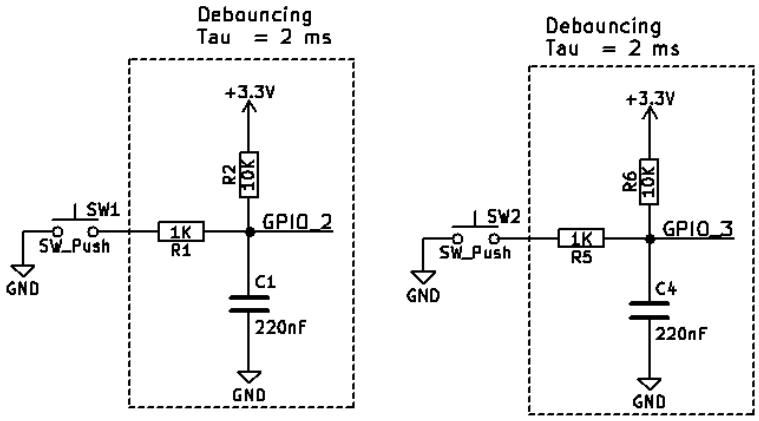


Figura 7.2: Diagrama de conexión de los botones de la tarjeta de expansión para controlar los GPIO2 y GPIO3 del DSP

Por otro lado, la tarjeta de expansión cuenta con un led RGB el cual se conecta con tres salidas de PWM, este led cambia de color cuando varía el ciclo de trabajo de cada uno de estos periféricos, estas salidas están conectadas como se muestra a continuación:

- Led verde: PWM_7A.
- Led azul: PWM_8A.
- Led rojo: PWM _9A.

La conexión descrita anteriormente se puede observar en la Figura 7.3, note que el *PWM_7A* y *PWM_8A* están conectados con los puntos de prueba 10 y 11 respectivamente.

La tarjeta se diseñó para utilizar dos entradas monocanal para micrófonos, las cuales se encuentran conectadas de la siguiente manera:

- Micrófono 1: ADC_A2.
- Micrófono 2: ADC_A3.

Los tipos de conexiones utilizadas tanto de entrada como salida de audio son conectores tipo *Jack* de 3.5 milímetros, en módulo de conexiones de entrada para los micrófonos se puede

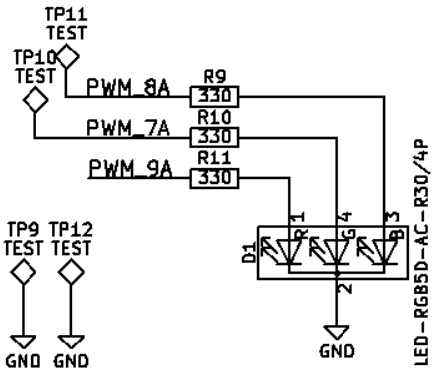


Figura 7.3: Leds indicadores para variación de PWM

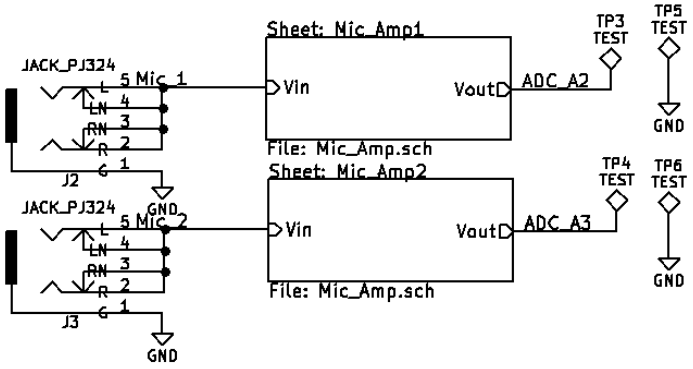


Figura 7.4: Conexiones para micrófonos

observar en la Figura 7.4.

Las entradas para micrófonos electret se encuentran conectadas al ADC-A2 y ADC-A3 del Delfino (Sección 5.4.1), sin embargo, la señal generada por este tipo de micrófonos no se puede emplear directamente debido a su baja amplitud, por lo que es necesario emplear una etapa de preamplificación. El diseño de ésta etapa se basa en la nota de aplicación "*Single-Supply, Electret Microphone Pre-Amplifier Reference Design*" de *Texas Instruments* [25].

En la Figura 7.5 se muestra el diagrama esquemático del módulo de preamplificación y acoplamiento de las señales generadas por los micrófonos, este módulo consta de tres etapas principales: preamplificación, amplificación y ajuste de voltaje offset.

La etapa de amplificación se encarga de aumentar la amplitud de la señal de entrada al DSP, esta etapa se basa en un amplificador inversor con ganancia fija en $G = 10$, esta ganancia está definida por (7.1).

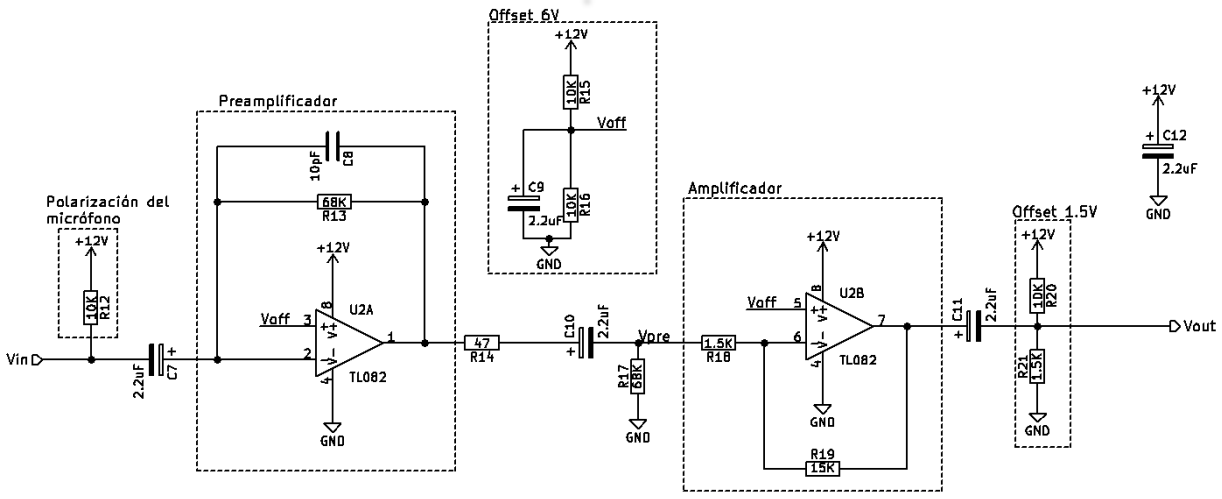


Figura 7.5: Circuito preamplificador para micrófono

$$G = -\frac{R_{19}}{R_{18}} \tag{7.1}$$

Esta ganancia puede ser ajustada de acuerdo al nivel de amplificación deseado modificando el valor de R_{19}.

Finalmente, los convertidores ADC del DSP operan en un intervalo de cero a 3.3 Volts, hay que tomar en cuenta que la señal amplificada tiene niveles de voltaje tanto positivos como negativos, es decir que está centrada en cero volts, por lo que es necesario sumarle un voltaje de corriente directa de tal manera que las variaciones de voltaje de dicha señal estén dentro del intervalo 0-3.3 Volts. Se agregó una etapa final en el preamplificador que suma un Offset de 1.56 volts a la señal amplificada, mismo que está definido por las resistencias R_{20} y R_{21}.

En la Figura 7.6 se muestra la magnitud de la respuesta en frecuencia del circuito preamplificador, donde se puede observar que mantiene una respuesta parcialmente plana en el intervalo auditivo ($20Hz - 20kHz$).

Por otro lado, la tarjeta de expansión propuesta en el presente trabajo también contiene una entrada de audio tipo estéreo, la cual se emplea para señales de audio con nivel de línea. Las señales de audio tipo estéreo constan de dos canales de audio, con la idea de simular la audición humana, donde cada canal la dirección izquierda y derecha respectivamente. El ADC-B del DSP se destinó para esta entrada realizando la siguiente conexión:

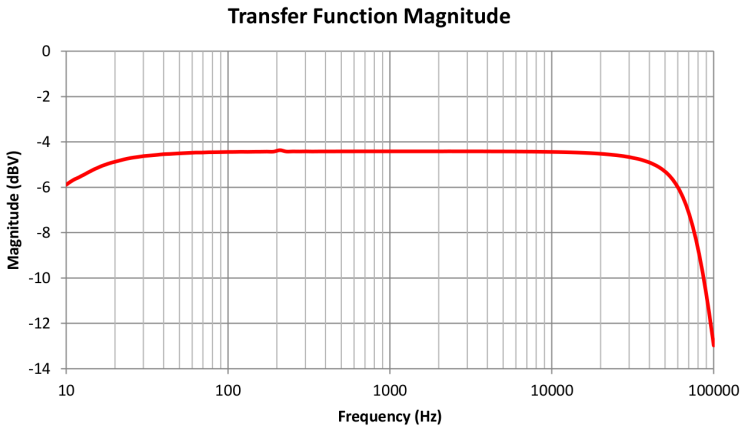


Figura 7.6: Respuesta en frecuencia del preamplificador [25]

- Auxiliar izquierdo (L): ADC_B4.
- Auxiliar derecho (R): ADC_B2.

En la Figura 7.7 se muestra el diagrama esquemático de la conexión estéreo realizada. Se puede observar que a cada canal canal se le agregó un voltaje de offset de 1.5 Volts para acoplar las señales con el convertidor ADC.

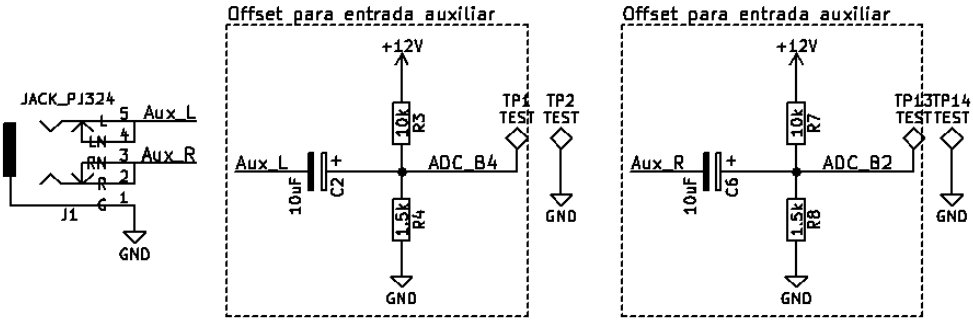
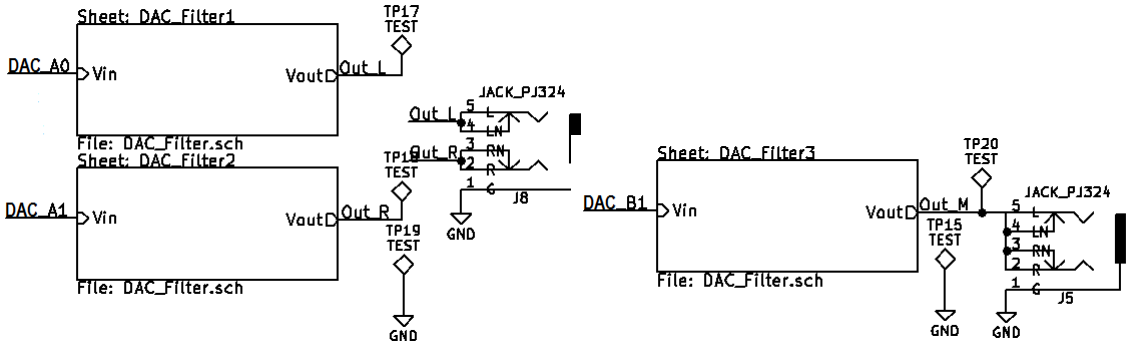


Figura 7.7: Conexión de entrada de audio estereo

El TMS320F28377s cuenta con tres DACs los cuales se emplean como salidas de audio con nivel de línea. En la tarjeta de expansión para el DSP se utilizaron los tres DACS de tal manera que se tuviera una salida estéreo y una monoaural, las conexiones de dichos DACs se realizaron de la siguiente manera:

- Salida estéreo izquierda (L): DAC_A.
- Salida estéreo derecha (R): DAC_B.
- Salida monoaural: DAC_C.

En la Figura 7.8 se muestran los diagramas de conexiones de la tarjeta de expansión donde el DAC A0 y DAC A1 corresponden a la salida de audio estéreo mientras que el DAC B1 corresponde a la monoaural.



(a) Conexión de los convertidores digital a analógico para salida de audio estéreo.

(b) Conexión de los convertidores digital a analógico para salida de audio monoaural

Figura 7.8: Conexión de los convertidores digital a analógico

El TMS320F28377s cuenta con tres DACs de 12 bits , sin embargo, para obtener una mejor respuesta se requiere de una etapa de filtrado analógico, la cual consiste en un filtro paso-bajas como el mostrado en la Figura 7.9. Este filtro tiene una topología *Sallen-Key*[1] de segundo orden [26].

Debido a que la tarjeta se diseñó para trabajar con señales de audio, este filtro se calculó con una frecuencia de corte $f_c = 15KHz$ como se observa en (7.2).

$$f_c = \frac{1}{2\pi\sqrt{R_{33}R_{36}C_{21}C_{22}}} = 15.025KHz \tag{7.2}$$

En la Figura 7.10 se muestra la respuesta en frecuencia del filtro diseñado para cada uno de los DACs,

[1]Filtro electrónico de segundo orden, este tipo de filtros son relativamente flexibles con las tolerancias de sus componentes

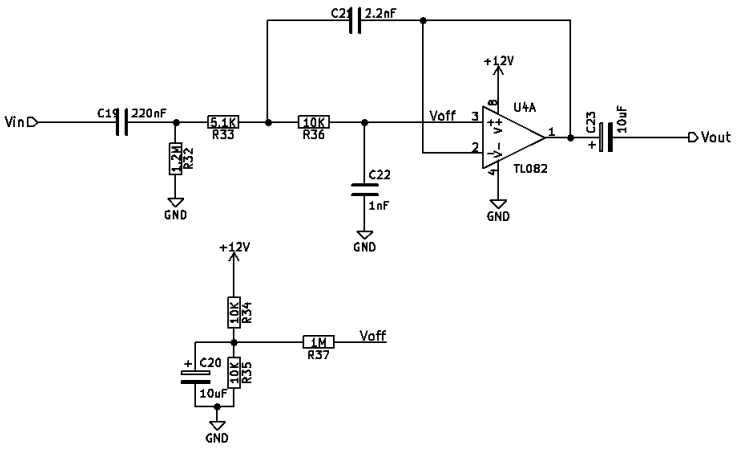


Figura 7.9: Filtro paso-bajas para el DAC

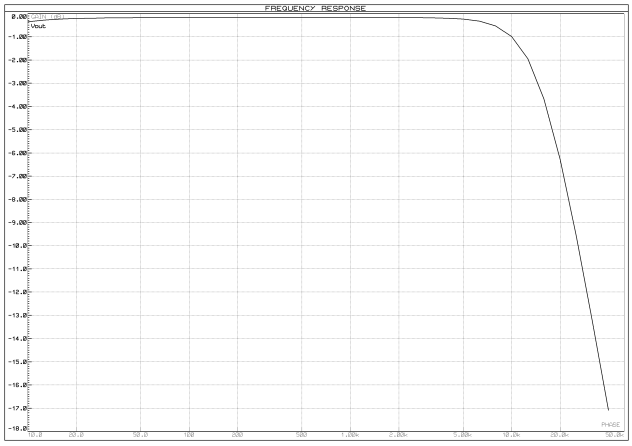


Figura 7.10: Respuesta en frecuencia del filtro paso-bajas

Circuito impreso (PCB)

El diseño del circuito impreso se basa en las dimensiones de la tarjeta del DSP, de tal manera que se pueda conectar directamente a la tarjeta por medio de los headers que contiene. El circuito impreso tiene 14.27cm de ancho mientras que de largo tiene 16.25cm.

En la Figura 7.11 se muestra el circuito integrado de la tarjeta de expansión para el DPS TMS320F28377s, donde se pueden observar cada uno de los módulos anteriormente descritos.

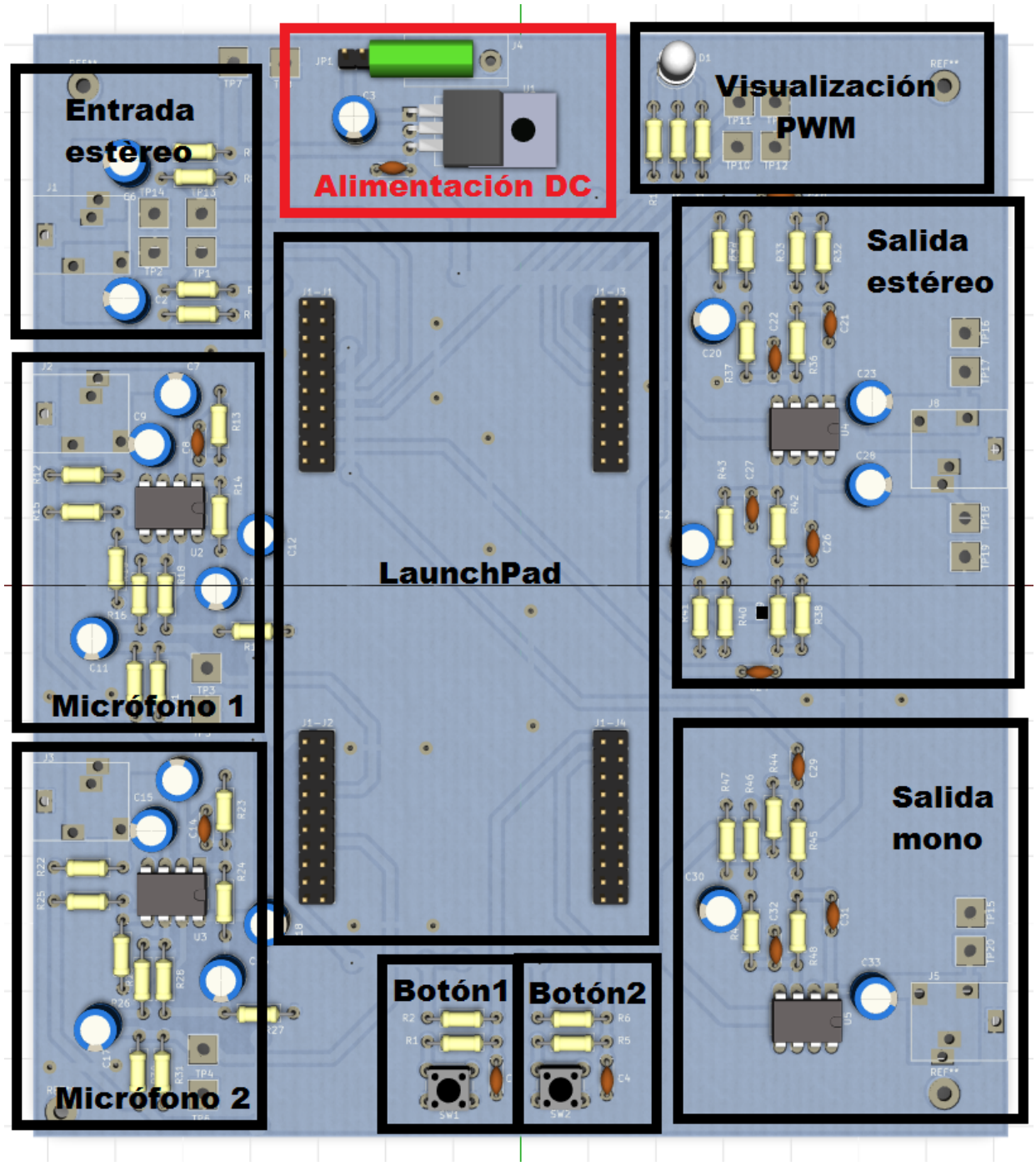


Figura 7.11: Tarjeta de expansión para el DSP TMS320F28377s

7.2. Detector de actividad de voz

Los algoritmos para el procesamiento digital de voz, tales como la estimación de dirección de arribo y reconocimiento, suponen que las señales de voz se encuentran activas en todo momento, sin embargo, las señales que emite una persona para comunicarse no son continuas, es decir que existen silencios entre frases o palabras, de tal manera que cuando un sistema de reconocimiento de voz o de estimación de dirección de arribo adquiere las señales cuando la fuente de voz se encuentra en silencio, éste solo obtendrá ruido de fondo provocando un error en la estimación. Por lo que es importante determinar si en un número finito de muestras existe presencia de voz o es únicamente ruido del ambiente.

Los algoritmos de detección de actividad de voz (VAD) están generalmente enfocados en el cálculo de la energía de fragmentos de la señal, éstos parten de una idea de que la señal adquirida por el micrófono contiene la emitida por la fuente de voz y el ruido de fondo cuando la fuente produce una emisión. Sin embargo, cuando no existe presencia de voz, únicamente tendrá el ruido de fondo como se muestra en (7.3).

$$H_0: \quad x(n) = s(n) + \eta(n) \tag{7.3}$$

$$H_1: \quad x(n) = \eta(n)$$

El VAD se centra en extraer las características de la señal de voz (por lo regular con base a la energía) para determinar si hay presencia de la señal de voz o no en un número finito de muestras, por lo regular la decisión es determinada por un umbral como se muestra en la Figura 7.12.

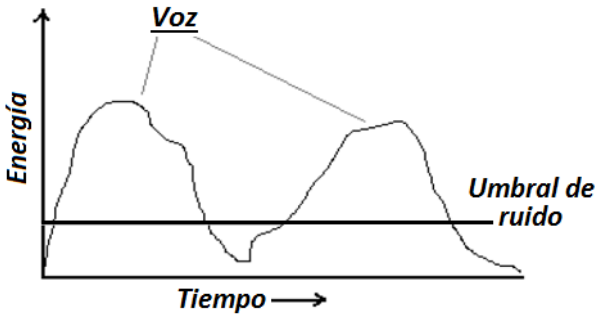


Figura 7.12: Detector de actividad de voz con umbral fijo [27].

El cálculo del umbral va a depender del ruido del ambiente en el tiempo t y es un valor clave en la selección de frames con voz y sin ella, de tal manera que antes de ejecutar el algoritmo de detección de actividad de voz es necesario determinar el umbral de energía del

ruido. La energía de una señal de longitud finita se determina por medio de (7.4).

$$E_x = \sum_{n=n1}^{n2} |x(n)|^2 \tag{7.4}$$

Suponiendo que en los primeros instantes en el que se ejecuta el sistema, únicamente contiene ruido ambiental, entonces se calcula la energía en un número finito de frames y se promedia el nivel de energía de ruido como se observa en (7.5).

$$E_{prom} = \frac{1}{I} \sum_{i=1}^{I} E_i \tag{7.5}$$

donde I es el número total de frames iniciales que únicamente contienen ruido ambiental. Finalmente, el umbral inicial se calcula como se muestra en (7.6)

$$T = E_{prom} + k_{VAD} E_{prom} \tag{7.6}$$

donde k_{VAD} es un factor de escala que define la tolerancia de energía de paso, de tal manera que entre mayor es este valor, la energía de voz deberá contener mayor energía para superar el umbral de decisión [27].

El detector de actividad de voz consiste en obtener la energía del frame adquirido en el tiempo t y verificar si es mayor a la energía del umbral de decisión, si es así, se considera un frame activo, es decir, contiene presencia de señales de voz, si no es así se considera como inactivo.

Una mejor aproximación es un detector de actividad de voz con umbral variable, el cual tiene la función de realizar un seguimiento de la energía del ruido actualizando su valor en frames inactivos. La actualización del umbral en un frame inactivo se calcula como se muestra en (7.7).

$$T_{nuevo} = (1 - p_{VAD})\, T_{anterior} + p_{VAD} E_\eta \tag{7.7}$$

donde T_{nuevo} es el valor del umbral actualizado, $T_{anterior}$ es el valor del umbral anterior y E_η la energía del ruido calculado en el frame inactivo. El valor de p_{VAD} se encuentra en un intervalo de $0 < p_{VAD} < 1$.

En el siguiente programa de lenguaje C, se muestra el código para ejecutar un detector de actividad de voz con actualización del umbral en el dominio del tiempo, tomando en cuenta que el umbral inicial se calcula en los primeros instantes de tiempo con el ruido ambiental.

Este programa utiliza el canal 2 de micrófono de la tarjeta de expansión para el LaunchPad (ver Sección 7.1) y el Led conectado en el GPIO12 de tal manera que cuando se detecta una ventana activa de voz, el LED se prende. El programa se ejecutó con las librerías de ControlSuite, además de utilizar la librería "Analog.h" descrita en el Apéndice B.

```
/*
 * Este programa ejecuta un detector de actividad de voz con umbral variable.
 * Calcula el umbral inicial por medio de la energía del ruido ambiental
 * en los primeros 20 frames. Los frames son de longitud 256 muestras.
*/

#include "F28x_Project.h"
#include "Analog2.h"          // Incluye librería Analog.h

Uint16 chanel=2;              // canal de adquisición
#define SIZE_BUF     256      // tamaño del buffer
#define FRAMES_NOISE     20   // número de frames iniciales
float    x[SIZE_BUF];         // vector en donde se almacena las señales
volatile int cont=0;

extern void ADCA_Process(void){

    Uint16  val;
    if (chanel==2){
        // almacena los datos adquiridos por el micrófono y los convierte
       // en flotantes
        x[cont++]=(float)(ADC_RESULT_PTR[ADCA]->ADCRESULT2*3.3/4096)-1.7;
    }else{
        // almacena los datos adquiridos por el micrófono y los convierte
       // en flotantes
        x[cont++]=(float)(ADC_RESULT_PTR[ADCA]->ADCRESULT3*3.3/4096)-1.7;
    }
    val=(Uint16)((x[cont]+1.7)*4096/3.3);
    DAC_Send(DACC, val);              // envía el resultado al DAC(A, B o C)
}
extern  void ADCB_Process(void){

    Uint16 val;
    if (chanel==2){
        val=ADC_RESULT_PTR[ADCB]->ADCRESULT2;
    }else{if(chanel==4){
        val=ADC_RESULT_PTR[ADCB]->ADCRESULT4;
        }
    }
    DAC_Send(DACC, val);
}

float    Energia (float *x){ // Función que calcula la energía
    int i;
```

```
    float E=0;
    for(i=0;i<SIZE_BUF;i++){
        E=x[i]*x[i]+E;
    }
    return E;
}

int main(void)
{
    InitSysCtrl();
    InitGpio();
    DINT;
    InitPieCtrl();
    float   E_r=0;          // almacenamiento de nivel de energía para umbral
    float   e,H,H_L;
    float   p=0.25;
    float   dn=0.2;
    int     i;

    IFR=0x0000;
    IFR=0x0000;

    InitPieVectTable();
    EALLOW;
        GpioCtrlRegs.GPAMUX1.bit.GPIO12=0;
        GpioCtrlRegs.GPADIR.bit.GPIO12=1;
    EDIS;

    ADC_Configure(ADCA,8000);       // Configura ADC con frecuencia de muestreo
    ADC_Init(ADCA,chanel);          // Configura el ADC(A) con chanel=2
    ADC_Int(ADCA,chanel);           // Configura interrupción para el ADC(A)
    ADC_Start(ADCA);                // Inicia conversión del ADC
    DAC_Configure(DACC);            // Configura el DAC

    for (i=0;i<FRAMES_NOISE;i++){   // frames de ruido iniciales
        cont=0;
        while(cont<(SIZE_BUF));     // realiza la conversión hasta llenar el
                                    // buffer x
        E_r+=Energia(x);            // almacena la energía de los frames
                                    // iniciales
    }
    H=(E_r/FRAMES_NOISE)+0.3;       // Umbral inicial

    while(1){
        cont=0;
        while(cont<(SIZE_BUF));     // realiza la conversión hasta llenar
                                // el buffer x
        e=Energia(x);               // calcula la energía del frame
        if (e>H){
```

```
            // frame activo (señal de voz)
            GpioDataRegs.GPASET.bit.GPIO12=1;
        }else{
            // frame inactivo (señal de ruido)
            GpioDataRegs.GPACLEAR.bit.GPIO12=1;
            // actualiza el umbral de desición
            H_L=(1-p)*H+p*e;
            H=H_L+H_L*dn;
        }
    }
}
```

7.3. Ecualizador de Audio

Un área importante dentro del procesamiento digital de señales es el audio, entre sus aplicaciones dentro del mundo digital se encuentran los ecualizadores. La función principal de un ecualizador es acentuar frecuencias aumentando o disminuyendo las ganancias de cada filtro. Los ecualizadores son muy utilizados principalmente en las mezcladoras para salas de conciertos y estudios de graduación, entre otros.

Se diseñó un ecualizador de tres bandas para un canal en la tarjeta de expansión explicada en la Sección 7.1. El ecualizador consta de tres filtros tipo FIR de orden 200, un filtro paso bajas, un filtro paso banda y un filtro paso altas con las siguientes frecuencias de corte:

1. Filtro paso bajas con frecuencia de corte en 400 Hz.

2. Filtro paso banda con frecuencias f1=400 Hz, f2= 5 KHz

3. Filtro paso altas con frecuencia de corte en 5 Khz

La respuesta de los filtros con una ganancia de cero dbs en la escala logarítmica se puede observar en la Figura 7.13.

El programa se desarrollo con las librerías que proporciona ControlSuite, además de la librería "Analog.h" descrita en el Apéndice B. Se utilizó el canal 2 del ADC que es la segunda entrada de micrófono en la tarjeta de expansión.

El programa principal está en lenguaje C, donde se realizan las configuraciones de adquisición y la ganancia, mientras que la función que ejecuta el banco de filtros está en lenguaje ensamblador. Las ganancias de los filtros de cada banda se actualizan dependiendo de la posición de los potenciómetros asignados a cada banda, ésto se logra realizando la conversión ADC del nivel de voltaje de un potenciómetro con rango dinámico de 0 a 3.3 Volts. Se utilizó el ADCA con la siguiente asignación de canales:

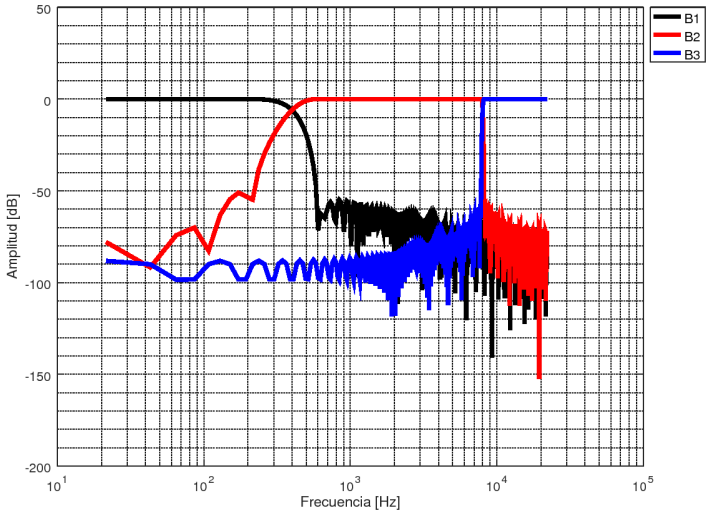


Figura 7.13: Respuesta de los tres filtros del ecualizador

1. Canal 2 <− x(n)
2. Canal 15 <− Ganancia de banda 1
3. Canal 5 <− Ganancia de banda 2
4. Canal 4 <− Ganancia de banda 3

A continuación se muestran el código en lenguaje C de la sección principal del programa en donde se puede observar que el procesamiento de las señales de voz se realiza en la función de adquisición llamada ADCA_Process.

```
#include "F28x_Project.h"
#include "Analog2.h"
#include <math.h>

Uint16 chanel[4]={2,5,4,15};// Canales utilizados para el ADC
                            // Canal 2 asignado para la señal de voz
                            // canal 15 asignado al potenciómetro de G1
                            // canal 5 asignado al potenciómetro de G2
                            // canal 4 asignado al potenciómetro de G3
int16 G1,G2,G3;             // Ganancias asignadas por los Potenciómetros

// Función en ensamblador que ejeecuta el banco de filtros del ecualizador.
// Los parámetros de entrada se asignan como: AL=aux,AH=G1, XAR4=G2,XAR5=G3
// y la muestra ecualizada retorna por medio de AL
```

```
extern int FIRFilter1_16( int16 aux, int16 G1, int16 G2, int16 G3);

extern void ADCA_Process(void){

    Uint16 val, valAux, aux;
    val = (ADC_RESULT_PTR[ADCA]->ADCRESULT2);
    G3 = (ADC_RESULT_PTR[ADCA]->ADCRESULT4);      // Ganancia de la banda 3
    G2 = (ADC_RESULT_PTR[ADCA]->ADCRESULT5);      // Ganancia de la banda 2
    G1 = (ADC_RESULT_PTR[ADCA]->ADCRESULT15);     // Ganancia de la banda 1

    valAux=val<<2;              // Ajuste de formato
    aux=valAux-8564;            // Quita el offset a la señal del ADC
    valAux=FIRFilter1_16(aux,G1,G2,G3);    // (AL=x(n),AH=G1,XAR4=G2,XAR5=G3)
    val=valAux + 8564;          //aumenta offset
    aux=val>>2;                 //Ajusta formato
    DAC_Send(DACC, aux);        //envía el resultado al DAC-C
}

extern void ADCB_Process(void){
    __asm("␣nop");
/*    Uint16 val;

        if (chanel==2){
            val = ADC_RESULT_PTR[ADCB]->ADCRESULT2;
        }else{ if (chanel==4){
            val = ADC_RESULT_PTR[ADCB]->ADCRESULT4;
            }
        }
        //considerar el offset
        DAC_Send(DACC, val); //envía el resultado al DAC-(A ,B o C)*/
}

int main(void)
{
    InitSysCtrl();
    DINT;
    InitGpio();
    InitPieVectTable();
    int i=0;
    IER=0x0000;
    IFR=0x0000;

    ADC_Configure(ADCA,16000);        // Configura ADC con frecuencia de
        muestreo
    for (i=0;i<4;i++){
        ADC_Init(ADCA, chanel[i]);    // Configura el ADC(A) y el chanel
        ADC_Int(ADCA, chanel[i]);     // Configura interrupción para el ADC(A)
    }
```

```
    ADC_Start(ADCA);                    // Inicia conversión del ADC
    DAC_Configure(DACC);                // Configura el DAC

    while(1);
}
```

La función declarada como *FIRFilter1_16*, es una función declarada como externa la cual se encuentra en lenguaje ensamblador. Dicha función contiene el banco de filtros de tres bandas del ecualizador de voz y tiene como variables de entrada el nuevo valor adquirido de x(n) y las ganancias de cada banda (G1, G2 y G3), mientras que el valor de salida es el dato x(n) filtrado.

Cuando se ejecuta la función *FIRFilter1_16*, las datos de las variables de entrada se alojan de la siguiente manera:

1. AL $\leftarrow$ x(n)
2. AH $\leftarrow$ G1
3. XAR4 $\leftarrow$ G2
4. XAR5 $\leftarrow$ G3

La función ejecuta los tres filtros de manera paralela y al final realiza la suma de las respuestas de los tres filtros. Se utiliza únicamente un buffer de muestras anteriores de la señal $x\,(n)$ para los tres filtros, además los coeficientes se almacenan seguidos en la memoria del DSP con la finalidad de que el apuntador no se direccione cada vez que se realice el producto punto de cada filtro, sino solamente aumentar la dirección a la que apunta. Cuando el programa termina de ejecutar el ciclo 'FILTROS' realiza el corrimiento de datos dentro del buffer 'x_buf' y regresa al programa principal con la muestra resultado del banco de filtros en la parte baja del acumulador.

En el siguiente programa en lenguaje ensamblador se muestra la función 'FIRFilter1_16' que ejecuta el banco de filtros para el ecualizador descrito anteriormente.

```
*
* Función en ensamblador que ejecuta la ecualización, los datos de entrada son
* x(n) almacenado en AL, ganancia G1 ealmacenado en AH, ganacia G2 almacenada
* en XAR4 y ganancia G3 almacenada en XAR5
*
                .global     _FIRFilter1_16

M       .set    201     ; Tamaño del filtro FIR
NB      .set    3       ; Número de bandas
coef    .space  M*16    ; Espacio reservado para filtro FIR
coef2   .space  M*16    ; Espacio reservado para filtro FIR
```

```
coef3   .space M*16     ; Espacio reservado para filtro FIR
x_buf   .space M*16     ; Espacio reservado para retardos
x_ext   .word  0        ; Localidad extra
Gain    .space NB*16    ; Espacio reservado para las ganacias de los filtros
suma    .word  0        ; Localidad de memoria para el resultado del ecualizador
Res     .word  0        ; Localidad de memoria para el resultado de los filtros

_FIRFilter1_16

        SETC    SXM
        SPM     #0              ; Corrimientos nulos para el registro P
        MOVW    DP,#x_buf       ; DP apunta a la página de cont
********* Almacena los datos de entrada
        MOV     @x_buf,AL       ; x(n) en la localidad 1 del bufer
                                ; de retardos
        MOVL    XAR7,#Gain      ; XAR7 apunta a la localidad Gain
        MOV     *XAR7++,AH      ; Almacena G1 en la localidad Gain
        MOVL    ACC,XAR4        ; Valor de entrada (G2) en el Acumulador
        MOV     *XAR7++,AL      ; Almacena G2 en la segunda localidad
                                ; de Gain
        MOVL    ACC,XAR5        ; Valor de entrada (G3) en el Acumulador
        MOV     *XAR7++,AL      ; Almacena G3 en la tercer localidad
                                ; de Gain
        ZAPA                    ; ACC=0,P=0
        MOV     *XAR7,AL        ; suma=0
        MOVL    XAR6,#coef      ; XAR6 apunta a la primera localidad
                                ; de coef
        MOVL    XAR5,#Gain      ; XAR5 apunta a la localidad de Gain
********* Filtro
        MOV     AR4,#NB-1       ; ciclo para aplicar los tres filtros
FILTROS
        MOVL    XAR7,#x_buf     ; XAR7 apunta al bufer de retardos
        ZAPA                    ; ACC=0, P=0
        RPT     #M-1            ; Ciclo RPT y hace instrucción MAC M veces
        ||      MAC P,*XAR6++,*XAR7++
        ADDL    ACC,P           ; suma última operación al registro
                                ; acumulador
        LSL     ACC,4           ; Ajusta el qi
        MOVL    XAR7,#Res       ; XAR7 apunta a la dirección de Res
        MOV     *XAR7,AH        ; Resultado del filtro en Res
********* Ganancia del filtro
        ZAPA                    ; ACC=0,P=0
        MOV     T,*XAR5++
        MPY     P,T,*XAR7       ; P=G*Res
        ADDL    ACC,P           ; ACC=P
        LSL     ACC,4           ; ACC=ACC<<4
        MOVL    XAR7,#suma      ; XAR7 apunta a la dirección de suma
        ADD     *XAR7,AH        ; suma=suma+AH
```

```
BANZ    FILTROS,AR4--

MOV     AL,*XAR7        ; AL=AH

MOVL    XAR5,#x_ext     ; XAR5 apunta a una localidad posterior
                        ; a la final de x_buf
RPT     #M-1            ; ciclo para recorrer datos del bufer
||      DMOV    *--XAR5
LRETR
```

7.4. Resumen del capítulo

El presente capítulo está enfocado en aplicar un procesamiento digital con señales reales utilizando los periféricos de la tarjeta LAUCHXL-F28377S. Para ello, se diseñó una tarjeta de expansión para realizar el acoplamiento de las señales de entrada por medio de micrófonos y un dispositivo electrónico reproductor de música, es decir, dos entradas tipo monocanal y una estéreo, también cuenta con dos botones para entrada digital tipo *push button* por medio de los puertos GPIO 2 y 3. De la misma manera, la tarjeta cuenta con dos salidas de audio (una tipo monocanal y una tipo estéreo), un led tricolor como indicador ya sea de un PWM o un simple Toggle. Se puede alimentar con una diferencia de potencial de 15 V utilizando corriente directa.

El diseño de la tarjeta es exclusivo para poder ejecutar diferentes aplicaciones del procesamiento digital de señales, en el presente capítulo se mostró un detector de actividad de voz que se ajusta a las características del ambiente y un ecualizador de voz de tres bandas, sin embargo, se proponen mas aplicaciones en ejercicios propuestos para que el lector pueda implementarlos utilizando los operaciones y algoritmos explicados en el Capítulo 4.

Apéndice A

Mapa de memoria propuesto

```
/*
 * Para mayor información acerca de la configuración
 * de la memoria se puede consultar:
 * http://processors.wiki.ti.com/index.php/C28x_Compiler_-
    _Understanding_Linking
 * http://www.ti.com/lit/ug/spru514m/spru514m.pdf (SPRU514m)
 * http://www.ti.com/lit/ug/spru513m/spru513m.pdf (SPRU513m)
 *
 */


MEMORY
{
PAGE 0 :
/* BEGIN es utilizado para el "boot to SARAM" modo gestor de arranque */
       BEGIN   : origin = 0x000000, length = 0x000002
       RAMM0   : origin = 0x000122, length = 0x0002DE
       RAMD0   : origin = 0x00B000, length = 0x000800
       RESET   : origin = 0x3FFFC0, length = 0x000002
       RAMGS0  : origin = 0x00C000, length = 0x004000 /* Esta
               * sección se amplio para abarcar las tres
               * siguientes secciones*/

/* Secciones que desaparecen en la expansión de memoria*/
       /*RAMGS1       : origin = 0x00D000, length = 0x001000*/
       /*RAMGS2       : origin = 0x00E000, length = 0x001000*/
       /*RAMGS3       : origin = 0x00F000, length = 0x001000*/

       RAMGS4  : origin = 0x010000, length = 0x001000
       RAMGS5  : origin = 0x011000, length = 0x002000

PAGE 1 :
       /* Parte de M0, BOOT rom será utilizado para el stack */
       BOOT_RSVD     : origin = 0x000002, length = 0x000120
```

```
        /* on-chip RAM block M1 */
        RAMM1   : origin = 0x000400, length = 0x000400
        RAMD1   : origin = 0x00B800, length = 0x000800

        RAMLS01         : origin = 0x008000, length = 0x001000
        RAMLS2  : origin = 0x009000, length = 0x000800
        RAMLS3  : origin = 0x009800, length = 0x000800
        RAMLS4  : origin = 0x00A000, length = 0x000800

        RAMLS5  : origin = 0x00A800, length = 0x000800
        RAMGS7  : origin = 0x013000, length = 0x004000 /* Esta
                * sección se amplio para abarcar las tres
                * siguientes secciones*/

/* Secciones que desaparecen en la expansión de memoria*/
        /*RAMGS8        : origin = 0x014000, length = 0x001000*/
        /*RAMGS9        : origin = 0x015000, length = 0x001000*/
        /*RAMGS10       : origin = 0x016000, length = 0x001000*/

        RAMGS11         : origin = 0x017000, length = 0x001000
        RAMGS12         : origin = 0x018000, length = 0x001000
        RAMGS13         : origin = 0x019000, length = 0x001000
        RAMGS14         : origin = 0x01A000, length = 0x001000
        RAMGS15         : origin = 0x01B000, length = 0x001000

        CANA_MSG_RAM    : origin = 0x049000, length = 0x000800
        CANB_MSG_RAM    : origin = 0x04B000, length = 0x000800

        CPU2TOCPU1RAM : origin = 0x03F800, length = 0x000400
        CPU1TOCPU2RAM : origin = 0x03FC00, length = 0x000400
}


SECTIONS
{
        codestart       : > BEGIN,      PAGE = 0
        ramfuncs        : > RAMM0       PAGE = 0
        /*Codigo ejecutable*/
        .text   : >> RAMGS0 | RAMGS4 | RAMGS5,    PAGE = 0
        .cio    : >> RAMLS4 | RAMLS5, PAGE = 1
        .sysmem         : > RAMD1, PAGE = 1
        .cinit  : > RAMM0,      PAGE = 0 /*Variables globales
                * y estáticas inicializadas*/
        .pinit  : > RAMM0,      PAGE = 0
        .switch         : > RAMM0,      PAGE = 0 /*Tablas para
                                        * "switch"*/
        .reset  : > RESET,      PAGE = 0, TYPE = DSECT

        .stack  : > RAMM1,      PAGE = 1   /*Stack*/
```

```
            /*Variables globales y estáticas*/
        .ebss   : >> RAMGS7 | RAMGS11 | RAMGS12,    PAGE = 1
        .econst         : >> RAMGS11 | RAMGS12,      PAGE = 1 /*Constantes*/
            /*Para memodia dinámica (malloc)*/
        .esysmem        : > RAMGS12,   PAGE = 1

        #ifdef __TI_COMPILER_VERSION
            #if __TI_COMPILER_VERSION >= 15009000
            .TI.ramfunc : {} > RAMM0,       PAGE = 0
            #endif
        #endif

/* Las siguientes definiciones de sección se necesitan cuando se utilizan los
*  IPC API Drivers */
        GROUP : > CPU1TOCPU2RAM, PAGE = 1
        {
            PUTBUFFER
            PUTWRITEIDX
            GETREADIDX
        }

        GROUP : > CPU2TOCPU1RAM, PAGE = 1
        {
            GETBUFFER :     TYPE = DSECT
            GETWRITEIDX :   TYPE = DSECT
            PUTREADIDX :    TYPE = DSECT
        }
}

/*
//===========================================================================
// End of file.
//===========================================================================
*/
```

Apéndice B

Biblioteca Analog.h

En este apartado se desglosa el código de la biblioteca *Analog.h*, la cual configura los módulos ADC y DAC utilizando las configuraciones propuestas en ControlSuite, encapsulando dichos procedimientos en métodos simples.

```
/*
 * Analog.h
 *
 * Esta libreria contiene funciones para la
 * configuración del subsistema analógico del
 * DSP TMS320F28377S que se encuentra en el
 * kit de desarrollo LAUNCHXL-F28377S. Este
 * subsistema cuenta con dos ADC (A y B), con
 * 6 canales cada uno (0-5) y dos canales
 * compartidos (14 y 15), los cuales pueden ser
 * usados en modo de 12 y 16 bits, la configuración
 * empleada usa 12 bits.
 *
 * Así mismo, tiene tres DAC de 12 bits (A,B y C),
 * los cuales emplean los voltajes de referencia
 * internos.
 *
 * Se emplea la configuración de reloj básica:
 * InitSysPll(XTAL_OSC,IMULT_20,FMULT_0,PLLCLK_BY_2),
 * con el cristal externo del kit de desarrollo
 * (f_xtal = 10 MHz).
 *
 *
 * IMPORTANTE: Esta librería emplea los ePWM1 y
 * ePWM2 para iniciar la conversión del ADC-A y
 * ADC-B, respectivamente, por lo que se recomienda
 * no emplear estos ePWM en otras aplicaciones, esto
 * con el fin de no modificar la frecuencia de muestreo.
 */
```

```
#define XTAL_FREQ 10000000  //frecuencia del cristal

/*   Definiciones para el ADC          */
#define ADCA 0
#define ADCB 1
//#define DEBUG



/*   Definiciones para el DAC          */
#define DACA 0
#define DACB 1
#define DACC 2
#define DAC_ENABLE 1
#define DAC_DISBLE 0
#define REFERENCE_VDAC              0
#define REFERENCE_VREF              1



volatile bool intrA = false;        //variable que indica que el
                                //ADC-A generó una interrupción
volatile bool intrB = false;        //variable que indica que el
                                //ADC-A generó una interrupción

volatile      struct ADC_RESULT_REGS*     ADC_RESULT_PTR[2] =
      {&AdcaResultRegs, &AdcbResultRegs};
volatile      struct ADC_REGS*      ADC_PTR[2] =
      {&AdcaRegs,&AdcbRegs};
volatile      struct DAC_REGS*      DAC_PTR[3] =
      {&DacaRegs,&DacbRegs,&DaccRegs};

void    ADCA_Process(void);
void    ADCB_Process(void);


interrupt     void    adca1_isr(void);
interrupt     void    adcb2_isr(void);

void    EPWM_Configure(Uint16 adc_num,  Uint32 Freq);

void    ADC_Configure(Uint16 adc_num,  Uint32 Freq);
void    ADC_Init(Uint16 adc_num, Uint16 channel);
void    ADC_Int(Uint16 adc_num, Uint16 channel);
void    ADC_Start(Uint16 adc_num);
Uint16 ADC_Read(Uint16 adc_num, Uint16 channel);

void    DAC_Configure(Uint16 dac_num);
```

```
void    DAC_Send(Uint16 dac_num, int dacval);


/*************************************************************
* Configura el ADC                                           *
*                                                            *
* adc_num: número de ADC a configurar (ADCA O ADCB)          *
* Freq: frecuencia de muestreo (Hz)                          *
*                                                            *
* IMPORTANTE: la frecuencia máxima de conversión está acotada *
* entre 382 Hz y 3.40 MHz (290 ns). Estos valores se pueden  *
* modificar si se emplea otra configuración de reloj.        *
*************************************************************/
void    ADC_Configure(Uint16 adc_num,  Uint32 Freq){

        if(Freq<382 || Freq>3400000){
        //No se puede muestrear tan rápido o tan lento
                __asm(" ESTOP0");
                return;                 //Error en las frecuencias
        }

        EALLOW;
        //Preescalador  /4
        ADC_PTR[adc_num]->ADCCTL2.bit.PRESCALE = 6;
        AdcSetMode(adc_num, ADC_RESOLUTION_12BIT, ADC_SIGNALMODE_SINGLE);
        //La interrupción se genera al terminar la conversión
        ADC_PTR[adc_num]->ADCCTL1.bit.INTPULSEPOS = 1;
        //Enciende el ADC
        ADC_PTR[adc_num]->ADCCTL1.bit.ADCPWDNZ = 1;
        //Tiempo de espera para que se encienda el ADC
        DELAY_US(1000);

        EDIS;

        EPWM_Configure(adc_num, Freq);
}       //end ADC_Configure



/*************************************************************
* Configura el el canal y el SOC  del ADC a emplear          *
*                                                            *
* adc_num: número de ADC a configurar (ADCA O ADCB)          *
* channel: canal a emplear (0,1,2,3,4,5,14 o 15)             *
* los canales 14 y 15 son compartidos en los ADC             *
*                                                            *
* IMPORTANTE: los SOC se configuran en orden respecto a los  *
* canales, de tal manera que en el caso de emplear varios    *
* canales por cada ADC, estos se adquieran de forma          *
```

```
* consecutiva.                                                    *
*****************************************************************/
void    ADC_Init(Uint16 adc_num, Uint16 channel){

        Uint16 acqps = 14;
        Uint16 Trigsel;

        EALLOW;

        if(adc_num == ADCA){
                Trigsel = 0x05; //EPWM1
#ifdef DEBUG
        /* Configuración extra para debuggear Fs  */
        /* GPIO 2 como salida                     */
        GpioCtrlRegs.GPAMUX1.bit.GPIO2 = 0;
        GpioCtrlRegs.GPADIR.bit.GPIO2 =  1;
#endif
        }
        else{
                Trigsel = 0x08;              //EPWM2
#ifdef DEBUG
        /* Configuración extra para debuggear Fs */
        /*    GPIO 3 como salida                 */
        GpioCtrlRegs.GPAMUX1.bit.GPIO3 = 0;
        GpioCtrlRegs.GPADIR.bit.GPIO3 =  1;
#endif
        }

        switch(channel){
        case 1:
                //Configura el canal 1
                ADC_PTR[adc_num]->ADCSOC1CTL.bit.CHSEL  = 1;
                //Tiempo de espera para el
                //S+H acqps + 1 SYSCLK ciclos
                ADC_PTR[adc_num]->ADCSOC1CTL.bit.ACQPS = acqps;
                //Conversion por EPWM
                ADC_PTR[adc_num]->ADCSOC1CTL.bit.TRIGSEL=Trigsel;
                break;
        case 2:
                //Configura el canal 2
                ADC_PTR[adc_num]->ADCSOC2CTL.bit.CHSEL  = 2;
                //Tiempo de espera para el
                //S+H acqps + 1 SYSCLK ciclos
                ADC_PTR[adc_num]->ADCSOC2CTL.bit.ACQPS = acqps;
                //Conversión por EPWM
                ADC_PTR[adc_num]->ADCSOC2CTL.bit.TRIGSEL=Trigsel;
                break;
        case 3:
                //Configura el canal 3
```

```
        ADC_PTR[adc_num]->ADCSOC3CTL.bit.CHSEL  = 3;
        //Tiempo de espera para el
        //S+H acqps + 1 SYSCLK ciclos
        ADC_PTR[adc_num]->ADCSOC3CTL.bit.ACQPS = acqps;
        //Conversión por EPWM
        ADC_PTR[adc_num]->ADCSOC3CTL.bit.TRIGSEL=Trigsel;
        break;
case 4:
        //Configura el canal 4
        ADC_PTR[adc_num]->ADCSOC4CTL.bit.CHSEL  = 4;
        //Tiempo de espera para el
        //S+H acqps + 1 SYSCLK ciclos
        ADC_PTR[adc_num]->ADCSOC4CTL.bit.ACQPS = acqps;
        //Conversión por EPWM
        ADC_PTR[adc_num]->ADCSOC4CTL.bit.TRIGSEL=Trigsel;
        break;
case 5:
        //Configura el canal 5
        ADC_PTR[adc_num]->ADCSOC5CTL.bit.CHSEL  = 5;
        //Tiempo de espera para el
        //S+H acqps + 1 SYSCLK ciclos
        ADC_PTR[adc_num]->ADCSOC5CTL.bit.ACQPS = acqps;
        //Conversión por EPWM
        ADC_PTR[adc_num]->ADCSOC5CTL.bit.TRIGSEL=Trigsel;
        break;
case 14:
        //Configura el canal 14
        ADC_PTR[adc_num]->ADCSOC14CTL.bit.CHSEL  = 14;
        //Tiempo de espera para el
        //S+H acqps + 1 SYSCLK ciclos
        ADC_PTR[adc_num]->ADCSOC14CTL.bit.ACQPS = acqps;
        //Conversión por EPWM
        ADC_PTR[adc_num]->ADCSOC14CTL.bit.TRIGSEL=Trigsel;
        break;
case 15:
        //Configura el canal 15
        ADC_PTR[adc_num]->ADCSOC15CTL.bit.CHSEL  = 15;
        //Tiempo de espera para el
        //S+H acqps + 1 SYSCLK ciclos
        ADC_PTR[adc_num]->ADCSOC15CTL.bit.ACQPS = acqps;
        //Conversión por EPWM
        ADC_PTR[adc_num]->ADCSOC15CTL.bit.TRIGSEL=Trigsel;
        break;
default:
        //Configura el canal 0
        ADC_PTR[adc_num]->ADCSOC0CTL.bit.CHSEL  = 0;
        //Tiempo de espera para el
        //S+H acqps + 1 SYSCLK ciclos
        ADC_PTR[adc_num]->ADCSOC0CTL.bit.ACQPS = acqps;
```

```
            //Conversión por EPWM
            ADC_PTR[adc_num]->ADCSOC0CTL.bit.TRIGSEL=Trigsel;
            break;

        }

    CpuSysRegs.PCLKCR0.bit.TBCLKSYNC = 1;
    EDIS;
}       //end ADC_Init



/*************************************************************
* Configura el canal que genera la interrupción del ADC      *
*                                                            *
* adc_num: número de ADC a configurar (ADCA O ADCB)          *
* channel: canal y soc a emplear (0-15)                      *
*                                                            *
* IMPORTANTE: para evitar problemas con las interrupciones   *
* se configuro el ADC-A con la interrupción 1, con el PIE    *
* PIEIER1_1.1, mientras que el ADC-B fue con la interrupción *
* 2, con el PIE PIEIER1_10.6                                 *
*************************************************************/
void ADC_Int(Uint16 adc_num, Uint16 channel){


        if( (channel>5 && channel <14) || channel>15){
            __asm(" ESTOP0");
            //El canal elegido no existe en el ADC
            return;
        }

        EALLOW;
        if(adc_num == ADCA){
            //nombre de la función de interrupción
            PieVectTable.ADCA1_INT = &adca1_isr;
            //Habilita la interrupción del PIE INT1.1
            PieCtrlRegs.PIEIER1.bit.INTx1 = 1;
            //canal que genera la interrupción INT1
            ADC_PTR[adc_num]->ADCINTSEL1N2.bit.INT1SEL = channel;
            //habilita INT1
            ADC_PTR[adc_num]->ADCINTSEL1N2.bit.INT1E = 1;
            //Habilita el grupo 1 de interrupciones
            IER |= M_INT1;

        }else{
            //nombre de la función de interrupción
            PieVectTable.ADCB2_INT = &adcb2_isr;
            //Habilita la interrupción del PIE INT10.6
```

```
            PieCtrlRegs.PIEIER10.bit.INTx6 = 1;
            //canal que genera la interrupción INT1
            ADC_PTR[adc_num]->ADCINTSEL1N2.bit.INT2SEL = channel;
            //habilita INT2
            ADC_PTR[adc_num]->ADCINTSEL1N2.bit.INT2E = 1;
            //Habilita el grupo 10 de interrupciones
            IER |= M_INT10;
        }


        ADC_PTR[adc_num]->ADCINTFLGCLR.all = 0x000F;

        EINT;           //Habilita interrupciones globales
        //Inicia el conteo de los EPWM
        CpuSysRegs.PCLKCR0.bit.TBCLKSYNC = 1;


    EDIS;
}       //end ADC_Int


/*************************************************************
* Configura el ePWM como trigger del ADC                     *
*                                                            *
* adc_num: número de ePWM a configurar                       *
*                                                            *
* epwm1->SCOA->ADCA                                          *
* epwm2->SOCB->ADCB                                          *
*************************************************************/
void EPWM_Configure(Uint16 adc_num, Uint32 Freq){

        Uint32 IMult, FMult, DivSel;
        Uint32 T;
        float f;

        //Obtenemos la configuración actual del reloj
        IMult = ClkCfgRegs.SYSPLLMULT.bit.IMULT;
        FMult = ClkCfgRegs.SYSPLLMULT.bit.FMULT;
        DivSel = ClkCfgRegs.SYSCLKDIVSEL.bit.PLLSYSCLKDIV;

        //calcula la frecuencia del reloj
        f = XTAL_FREQ*(float)(IMult+FMult)/(DivSel<<1);
        f = f/Freq;     //calcula el periodo del contador
        f = f/4;        //por el divisor /4 del ADC
        T = (int)f+1; //periodo del EPWM

        EALLOW;

        if(adc_num == ADCA){
```

```
            //Deshabilita el SOC-A
            EPwm1Regs.ETSEL.bit.SOCAEN = 0;
            //SOC cuenta up
            EPwm1Regs.ETSEL.bit.SOCASEL = 4;
            //Genera un pulso al primer evento
            EPwm1Regs.ETPS.bit.SOCAPRD = 1;
            //Frecuencia de muestreo
            EPwm1Regs.TBPRD = T; //0x0C36;
            //Detiene el contador
            EPwm1Regs.TBCTL.bit.CTRMODE = 3;
        }else{
            //Deshabilita el SOC-B
            EPwm2Regs.ETSEL.bit.SOCBEN = 0;
            //SOC cuenta up
            EPwm2Regs.ETSEL.bit.SOCBSEL = 4;
            //Genera un pulso al primer evento
            EPwm2Regs.ETPS.bit.SOCBPRD = 1;
            //Frecuencia de muestreo
            EPwm2Regs.TBPRD = T;//0x0209;
            //Detiene el contador
            EPwm2Regs.TBCTL.bit.CTRMODE = 3;
        }

        EDIS;

        return;
}       //end EPWM_Configure


/*************************************************************
* Inicia la operanción del ADC                               *
*                                                            *
* adc_num: número de ADC a iniciar (ADCA O ADCB)             *
*************************************************************/
void ADC_Start(Uint16 adc_num){
        if(adc_num == ADCA){
            //Habilita el SOC-A
            EPwm1Regs.ETSEL.bit.SOCAEN = 1;
            //Inicia el conteo del EPWM
            EPwm1Regs.TBCTL.bit.CTRMODE = 0;
        }else{
            //Habilita el SOC-B
            EPwm2Regs.ETSEL.bit.SOCBEN = 1;
            //Inicia el conteo del EPWM
            EPwm2Regs.TBCTL.bit.CTRMODE = 0;
        }

}       //end ADC_Run
```

```
/*************************************************************
* Lee el último valor del ADC                                *
*                                                            *
* adc_num: número de ADC a leer (ADCA O ADCB)                *
* channel: número de canal a leer                            *
*************************************************************/
Uint16 ADC_Read(Uint16 adc_num, Uint16 channel){

        Uint16 val = 0;
        //espera hasta que se presente la interrupción
        while(!intrA);
        //baja la bandera de la interrupción
        intrA = false;

        switch(channel){
        case 1:
                val = ADC_RESULT_PTR[adc_num]->ADCRESULT1;
                break;
        case 2:
                val = ADC_RESULT_PTR[adc_num]->ADCRESULT2;
                break;
        case 3:
                val = ADC_RESULT_PTR[adc_num]->ADCRESULT3;
                break;
        case 4:
                val = ADC_RESULT_PTR[adc_num]->ADCRESULT4;
                break;
        case 5:
                val = ADC_RESULT_PTR[adc_num]->ADCRESULT5;
                break;
        case 6:
                val = ADC_RESULT_PTR[adc_num]->ADCRESULT6;
                break;
        case 7:
                val = ADC_RESULT_PTR[adc_num]->ADCRESULT7;
                break;
        case 8:
                val = ADC_RESULT_PTR[adc_num]->ADCRESULT8;
                break;
        case 9:
                val = ADC_RESULT_PTR[adc_num]->ADCRESULT9;
                break;
        case 10:
                val = ADC_RESULT_PTR[adc_num]->ADCRESULT10;
                break;
        case 11:
                val = ADC_RESULT_PTR[adc_num]->ADCRESULT11;
                break;
        case 12:
```

```
            val = ADC_RESULT_PTR[adc_num]->ADCRESULT12;
            break;
        case 13:
            val = ADC_RESULT_PTR[adc_num]->ADCRESULT13;
            break;
        case 14:
            val = ADC_RESULT_PTR[adc_num]->ADCRESULT14;
            break;
        case 15:
            val = ADC_RESULT_PTR[adc_num]->ADCRESULT15;
            break;
        default:
            val = ADC_RESULT_PTR[adc_num]->ADCRESULT0;
            break;
        }

        return val;
}       //end ADC_Read




/*************************************************************
* Configura el DAC-C                                         *
*************************************************************/
void DAC_Configure(Uint16 dac_num){
        EALLOW;
        //Voltaje de referencia
        DAC_PTR[dac_num]->DACCTL.bit.DACREFSEL = REFERENCE_VREF;
        //Habilita el DAC
        DAC_PTR[dac_num]->DACOUTEN.bit.DACOUTEN = DAC_ENABLE;
        //Pone el 0V la salida
        DAC_PTR[dac_num]->DACVALS.all = 0;
        //Retraso para que encienda el DAC
        DELAY_US(10);
        EDIS;
}

/*************************************************************
* Envia información al DAC-C                                 *
*************************************************************/
void DAC_Send(Uint16 dac_num, int dacval){
        // Envia el valor al DAC
        DAC_PTR[dac_num]->DACVALS.all = dacval;
}
```

```
/*************************************************************
* Rutina de interrupción del ADC-A                           *
*************************************************************/
interrupt void adca1_isr(void){
        //bandera para indicar que ya se presentó la interrupción
        intrA = true;
        //saltamos al proceso
        ADCA_Process();
        //borra la bandera INT1
        AdcaRegs.ADCINTFLGCLR.bit.ADCINT1 = 1;
        //PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;
        PieCtrlRegs.PIEACK.bit.ACK1 = 1;

#ifdef DEBUG
        /* Configuración extra para debuggear Fs  */
        /*     GPIO 2 como salida                 */
        GpioDataRegs.GPATOGGLE.bit.GPIO2 = 1;
#endif

}//end adca1_isr

/*************************************************************
* Rutina de interrupción del ADC-B                           *
*************************************************************/
interrupt void adcb2_isr(void){
        //bandera para indicar que ya se presentó la interrupción
        intrB = true;

        //saltamos al proceso
        ADCB_Process();

        //borra la bandera INT2
        AdcbRegs.ADCINTFLGCLR.bit.ADCINT2 = 1;
        //PieCtrlRegs.PIEACK.all = PIEACK_GROUP10;
        PieCtrlRegs.PIEACK.bit.ACK10 = 1;

#ifdef DEBUG
        /* Configuración extra para debuggear Fs   */
        /*     GPIO 3 como salida                  */
        GpioDataRegs.GPATOGGLE.bit.GPIO3 = 1;
#endif

}       //end adcB1_isr
```

Apéndice C

Biblioteca Serial.h

Esta sección contiene el código de la biblioteca *Serial.h*, la cual define metodos de configuracion y uso de los puertos SCI, que sintetizan los procedimientos definidos por el conjunto de archivos de Control Suite.

```
/*
 * Serial.h
 *
 */


//   Definiciones para el BaudRate
#define BR9600   0
#define BR19200  1
#define BR38400  2
#define BR57600  3
#define BR115200 4

volatile bool intrRx = false;       //variable que indica que
                            //se recibió un dato

void Serial_Process(void);
interrupt void sciaRxFifoIsr(void);

void Serial_Configure(Uint16 BR);
void Serial_Init(void);
void Serial_Start(void);

void Serial_putchar(int a);
void Serial_print(char *msg);


/*************************************************************
 * Configura el puerto serial A (SCI-A)                      *
 *                                                           *
```

```
 * Bits = 8                                                        *
 * Paridad = NO                                                    *
 * Modo = Asíncrono                                                *
 * Stop = 1 bit                                                    *
 *                                                                 *
 * BR: baud rate a emplear (9600,19200,38400,57600,115200)         *
 *                                                                 *
 *******************************************************************/
void Serial_Configure(Uint16 BR){

        SciaRegs.SCICCR.all = 0x0007;
        SciaRegs.SCICTL1.all = 0x0003;
        SciaRegs.SCICTL2.bit.TXINTENA = 1;
        SciaRegs.SCICTL2.bit.RXBKINTENA = 1;

        //LSPCLK = 25MHz
        //BaudRate = LSPCLK/( (BRR+1)*8 )
        switch(BR){
                case 1:
                        // BRR = 161 = 19200 Bauds
                        SciaRegs.SCIHBAUD.all = 0x00;
                        SciaRegs.SCILBAUD.all = 0xA1;
                        break;
                case 2:
                        // BRR = 80 = 38400 Bauds
                        SciaRegs.SCIHBAUD.all = 0x00;
                        SciaRegs.SCILBAUD.all = 0x50;
                        break;
                case 3:
                        // BRR = 52 = 57600 Bauds
                        SciaRegs.SCIHBAUD.all = 0x00;
                        SciaRegs.SCILBAUD.all = 0x34;
                        break;
                case 4:
                        // BRR = 26 = 115200 Bauds
                        SciaRegs.SCIHBAUD.all = 0x00;
                        SciaRegs.SCILBAUD.all = 0x1A;
                        break;
                default:
                        // BRR = 324 = 9600 Bauds
                        SciaRegs.SCIHBAUD.all = 0x01;
                        SciaRegs.SCILBAUD.all = 0x44;
                        break;
        }

        SciaRegs.SCIFFTX.all = 0xC020;
        SciaRegs.SCIFFRX.all = 0x0021;
        SciaRegs.SCIFFCT.all = 0x0;
```

```
}//end Serial_Configure


/*************************************************************
* Inicializa el puerto serial A (SCI-A), para funcionar a    *
* través del puerto USB, configura la interrupción de        *
* recepción de datos.                                        *
*                                                            *
* TX = GPIO84                                                *
* RX = GPIO85                                                *
*                                                            *
*************************************************************/
void Serial_Init(void){

        EALLOW;
        //Configura los GPIO del puerto SCI-A
        GpioCtrlRegs.GPCMUX2.bit.GPIO84 = 1;
        GpioCtrlRegs.GPCMUX2.bit.GPIO85 = 1;
        GpioCtrlRegs.GPCGMUX2.bit.GPIO84 = 1;
        GpioCtrlRegs.GPCGMUX2.bit.GPIO85 = 1;

        //PieVectTable.SCIA_RX_INT = &sciaRxFifoIsr; //define la
                // rutina de interrupción
        EDIS;

        //PieCtrlRegs.PIECTRL.bit.ENPIE = 1; //habilita el bloque
                                             // del PIE
        //PieCtrlRegs.PIEIER9.bit.INTx1 = 1; //habilita la
        //interrupción del PIE Grupo 9, INT1 (SCIA_RX)


}//end Serial_Init



/*************************************************************
* Inicia la operación del puerto serial A (SCI-A)            *
*                                                            *
 *************************************************************/
void Serial_Start(void){

        SciaRegs.SCICTL1.all = 0x0023;  // Reinicia el puerto
                                        // serial
        SciaRegs.SCIFFTX.bit.TXFIFORESET = 1;
        SciaRegs.SCIFFRX.bit.RXFIFORESET = 1;

    //IER = 0x100; // Habilita CPU INT
```

```
    //EINT;

}//end Serial_Start



/*************************************************************
* Envía un caracter a través del puerto serial A (SCI-A)     *
*                                                            *
 *************************************************************/
void Serial_putchar(int a){

        while (SciaRegs.SCIFFTX.bit.TXFFST != 0);//espera que se
                                                 // pueda transmitir
        SciaRegs.SCITXBUF.all = a;

}//end Serial_putchar

/*************************************************************
* Envía una cadena de caracteres a través puerto serial      *
* A (SCI-A), la cadena debe terminar con "\0"                *
*                                                            *
*************************************************************/
void Serial_print(char *msg){
        int i = 0;

        while(msg[i] != '\0'){
                Serial_putchar(msg[i]);
                i++;
        }

}//end Serial_print


/*************************************************************
*       Rutina de interrupción de SCI-A                      *
*************************************************************/
interrupt void sciaRxFifoIsr(void){

        intrRx = true;
        Serial_Process();
        SciaRegs.SCIFFRX.bit.RXFFOVRCLR = 1;    // Limpia bandera de sobreflujo
        SciaRegs.SCIFFRX.bit.RXFFINTCLR = 1;    // Limpia bandera de intrrupción

        PieCtrlRegs.PIEACK.all |= 0x100;

}//end sciaRxFifoIsr
```

Bibliografía

[1] Texas Instruments. *TMS320F2837xS Delfino Microcontrollers*, 2017.

[2] Texas Instruments. *TMS320F2837xS Delfino Microcontrollers*, 2018.

[3] Texas Instruments. *LAUCHXLF28377S Overview*, 2017.

[4] Texas Instruments. *TMS320F2837xS Delfino Microcontrollers*, 2014.

[5] Texas Instruments. *TMS320F28377S Launchpad Quick Start Guide*, 2015.

[6] Texas Instruments. *controlSUITE software. Comprehensive. Intuitive. Optimized. Real-world software for real-time control.*

[7] Texas Instruments. *controlSUITE Getting Started Guide*, 2010.

[8] Texas Instruments. *TMS320F2837xS Delfino Microcontrollers Technical Reference Manual*, 2014.

[9] Texas Instruments. *TMS320C28x Assembly Language Tools v16.12.0STS rev. L*, 2016.

[10] Texas Instruments. *TMS320C28x Optimizing C/C++ Compiler v17.3.0STS rev. M*, 2017.

[11] Texas Instruments. *TMS320C28x Assembly Language Tools v18.1.0.LTS*, 2018.

[12] Escobar S. L. *Arquitecturas de DSP TMS320F28xxx y aplicaciones.* Facultad de Ingeniería, UNAM, 2014.

[13] Kuo M. S. & Woon-Seng G. *Digital signal Processors, Architecture, implementations and applications.* Prentice-Hall, New Jersey, USA, 2005.

[14] Proakis J. G & Manolakis D. G. *Digital Signal Processing, Principles, Algorithms and Applications.* Macmillan, New York, 1992.

[15] Escobar S. L. *Conceptos Básicos de Procesamiento Digital de señales.* Facultad de Ingeniería, UNAM, 2009.

[16] IEEE. *IEEE Standar for Binary Floating-Point Arithmetic*, 1985.

[17] Texas Instruments. *TMS320C28x Floating Point Unit and Instruction Set Reference Guide*, 2008.

[18] Escobar S. L. *Diseño de Filtros Digitales*. Facultad de Ingeniería, UNAM, 2006.

[19] Proakis J. G. *Digital Communications*. McGraw Hill, New York, 1995.

[20] V.A. Oppenheim and W.R. Shafer. *Tratamiento digital de señales*. Pearson Educación, 3a edition, 2011.

[21] Texas Instruments. *TMS320x2833x, 2823x Enhanced Pulse Width Modulator (ePWM) Module Reference Guide*, 2009.

[22] Texas Instruments. *TMS320x2833x, Analog-to-Digital Converter (ADC) Module*, 2007.

[23] Texas Instruments. *TMS320x2833x, 2823x DSC Serial Peripheral Interface (SPI) Reference Guide*, 2003.

[24] Texas Instruments. *TMS320x2833x, 2823x Serial Communications Interface (SCI) Reference Guide*, 2003.

[25] Texas Instruments. *Single-Supply, Electret Microphone Pre-Amplifier Reference Design*, 2015.

[26] Texas Instruments. *Analysis of the Sallen-Key architecture*, 1999.

[27] Miguel Angel Flores Gómez. Estimación de la dirección del ángulo de arribo de una fuente de voz. Master's thesis, Facultad de Ingeniería, Universidad Nacional Autónoma de México, 2017.

Glosario

A
AC97: “audio codec 97”.
ACC: registro acumulador.
ADC: convertidor análogo digital.
AH: acumulador parte alta, bits 31 a 16.
AL: acumulador parte baja, bits 15 a 0.
ALU: unidad aritmética lógica.
ARAU: unidad aritmética de registros auxiliares.
ARi: registros auxiliares o apuntadores de 16 bits, i:0,1,...,7. Parte baja de los XARi.
ARM: microprocesador avanzado RISC.
ARMA: filtros autorregresivos de movimiento promedio.

B
BIOS: sistema básico de entrada salida
Big endiand: forma de organizar los bytes o palabras en memoria, si un dato es de dos palabras de longitud, se almacena en la localidad más baja la palabra MSW y en seguida la LSW.
BOS: localidad baja de la pila.
bps: bit por segundo.
BR: “bit reverse” o acarreo inverso.
BSP: puerto serie buffereado.
bu: bits sin signo.

C
C1: complemento a uno.
C2: complemento a dos.
C28x: procesador digital de señales TMS320C28xx de TI.
CISC: conjunto complejo de instrucciones.
CLA: unidad aceleradora de operaciones matemáticas, sólo en algunas versiones C28x.
CMOS: tecnología de circuitos integrados “complementary metal oxide semiconductor”.
CCS: “code composer studio”, ambiente integrado de desarrollo.
COND: condición.

CRC: verificación de redundancia cíclica.
CSM: módulo de código de seguridad.
CPK: kernel del protocolo eCAN.
Compandor: compresor expansor.
cte: constante.

D
DAB: bus de direcciones de datos.
DARAM: memoria de doble acceso en un ciclo.
dat: dato.
DDB: bus de lectura de datos.
DFT: transformada discreta de Fourier.
dir: dirección.
DMA: acceso directo a memoria.
dma: dirección de memoria dato.
DMAC: multiplicación acumulación dual o doble de 16x16 bits.
DP: registro apuntador de página.
DSP: procesadores digitales de señales.
DSC: controladores de señales digitales.
dst: destino DTMF: "dual tone modulation frequency".
Δ: resolución.

E
EEPROM: memoria ROM borrable eléctricamente.
EOC: señal de fin de conversión.
EV: módulo manejador de eventos.
EOS: fin de conversión de una secuencia.

F
FA: filtro analógico.
FD: filtro digital.
FFT: transformada rápida de Fourier.
FIFO: tipo de registros o memoria con acceso primer dato en entrar, primer dato en salir.
FIR: filtros de respuesta finita al impulso.
FPU: unidad de punto flotante.

G
GIOP: entradas y salidas de propósito general.
Gw: giga palabras = mil millones de palabras.

H
h: símbolo posfijo usado en una constante en hexadecimal.
HRPWM: PWM de alta resolución.

I
I2C: interfaz “inter-integrated-circuit”.
I2S: interfaz “integrated interchip sound”.
IEEE: asociación internacional de ingenieros eléctricos y electrónicos.
ID: intervalo dinámico.
IDE: ambiente integrado de desarrollo.
IIR: filtros de respuesta infinita al impulso.
IOM-2: “oriented modular interface revision 2, bus-compliant device”
I/O: entrada/salida.

J
JTAG: “joint test action group”.

K
Kw: kilo palabras = mil palabras.
Khz: kilo Hertz = mil Hertz.

L
Little endian: forma de organizar los bytes o palabras en memoria, si un dato es de dos palabras de longitud, se almacena en la localidad más baja la palabra LSW y enseguida la MSW.
LSb: bit menos significativo.
LSB: byte menos significativo.
LSW: palabra menos significativa.
loc: localidad en memoria.
loc16: localidad de 16 bits.
loc32: localidad de 32 bits.

M
MA: filtros de movimiento promedio.
MAC: operación multiplicación acumulación.
McBSP: puerto serial multicanal buffereado.
Mhz: mega Hertz = un millón de Hertz.
MIPS: millones de instrucciones por segundo.
MMU: unidad manejadora de memoria.
MSPS: millones de muestras por segundo.

MSb: bit más significativo.
MSB: byte más significativo.
MSW: palabra más significativa.
MUX: multiplexor o selector.
Mw: mega palabra = un millón de palabras.

N
NRZ: no retorno a cero.
ns: nanosegundos.

O
Octave: Software libre y lenguaje de programación para realizar cálculos numéricos.
OTP: circuitos programables una sola vez: “one time programmable”.
OMAP: plataformas abiertas para aplicaciones multimedia.
OVM: Operación en modo saturado.

P
P: registro producto de 32 bits.
p: precisión numérica.
PAB: bus de direcciones de programa.
PDB: bus de datos de programa.
PC: contador de programa.
PDS: procesamiento digital de señales.
PIE: interfaz de expansión de interrupción de periféricos.
PH: 16 bits parte alta de P.
PL: 16 bits parte baja de P.
PLL: mallas de fase amarrada.
pma: dirección de memoria programa.
prog: programa.
PWM: modulación por ancho de pulso.

Q
QE: número de bits para la parte entera.
Qi: formato de punto entero y número de los bits para la parte fraccionaria.
QF: número de los bits para la parte fraccionaria.

R
RAM: memoria de acceso aleatoria.
RISC: conjunto reducido de instrucciones.
ROM: memoria de sólo lectura.
RPC: retorno del contador de programa.

S
S: bit de signo.
SARAM: memoria RAM de simple acceso.
SCI: interfaz de comunicación serial.
SCL: línea para reloj de I2C.
SDA: línea para datos de I2C.
SH(i): corrimientos de entrada y sobre el ACC.
SH(o): corrimientos de salida del ACC.
SH(m): corrimientos de salida del multiplicador.
SLITD: sistema lineal e invariante en el tiempo discreto.
SM: signo magnitud.
SNR: relación señal a ruido.
SPM: modo de corrimiento de la salida del multiplicador.
SP: apuntador de pila.
src: fuente SOC: inicio de conversión.
SPI: interfaz de puerto serie.
ST0: registro de estado 0 de 16 bits.
ST1: registro de estado 1 de 16 bits.
sig: signo o signado.

T
T: 16 bits parte alta del registro XT.
TI: Texas Instruments, compañía líder de DSP.
TL: 16 bits parte baja del registro XT.
TOS: localidad alta de la pila.
TZ: Transformada Z TZI: Transformada Z inversa
U
MMU: unidad manejadora de memoria.

V
V: volts.
Vcc: voltaje de alimentación.
VCO: osciladores controlados por voltaje.
VCU: unidad Viterbi, de matemáticas complejas y de CRC.
VLWI: “very large word instruction”, palabra de instrucción muy larga.

X
XARi: registros auxiliares o apuntadores de 32 bits, i:0,1,...,7.
XT: registro temporal de 32 bits.

W

w: palabra de 16 bits.