UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA

Departamento de Electrónica

GUÍA DE LABORATORIO 1
INTRODUCCIÓN AL CODE COMPOSER Y GENERACIÓN DE SEÑALES

CURSO	LABORATORIO DE PROCESAMIENTO DIGITAL DE SEÑALES
SIGLA	ELO–385
PROFESOR	MARCO E. RIVERA A.
AYUDANTE	IGNACIO LIZAMA

Valparaíso, Febrero de 2009

Guía de Laboratorio 1
Introducción al Code Composer y Generación de Señales

En esta guía se han resumido los principales tópicos necesarios para el desarrollo del primer laboratorio. Muchos de los contenidos han sido obtenidos de Internet, y se ha acudido a los apuntes de años anteriores para elaborar este documento. Se ha sintetizado lo mejor posible, con el fin de dar una práctica guía al estudiante y no colapsarlo con exceso de información. No obstante, como primera versión está con falencias las que se esperan mejorar en un futuro, gracias a la retroalimentación de los estudiantes. Además, y para aquellos interesados en profundizar los temas, en la página del curso se ha dejado disponible el material completo para que los estudiantes puedan acceder a ellos. El **objetivo de este laboratorio** es introducir al estudiante en las herramientas de hardware y software que se usarán en este curso. En la primera sesión, usted se familiarizará con el hardware, el ambiente de trabajo y las herramientas de software para programar, depurar y verificar los programas DSP en tiempo real sobre la plataforma de evaluación DSK TMS320C6711, además de la generación de señales AM y FM. Con este fin se desarrollarán una serie de algoritmos que permiten generar patrones de señales, chequeándose su correcto funcionamiento a través del codec de audio disponible en el kit DSK.

Introducción al Code Composer

1. ¿Qué es un Procesador Digital de Señales?

Un Procesador Digital de Señales o DSP, es un rápido y poderoso tipo de microprocesador, el cual ha sido optimizado para la detección, procesamiento y generación de señales del mundo real que son continuas en el tiempo, tales como voz, video, música, etc. Este tipo de procesadores son ideales para aplicaciones que no toleran ningún retardo. Un ejemplo clásico para entender la aplicación del DSP, es la del teléfono celular, donde no es fácil conversar a través de él cuando existe un retardo en la línea, puesto que conlleva a que la señal se corte o a una confusión, ya que ambos usuarios hablan a la vez. El DSP en el teléfono, es el encargado de procesar el sonido tan rápidamente que la persona puede hablar y escuchar sin problemas de retardo ni alguna molestia que ello implica, es decir, las señales son procesadas en tiempo real.

2. ¿Qué es el Procesamiento Digital de Señales?

El Procesamiento Digital de Señales es una de las más poderosas tecnologías debido al impacto que ésta ha generado, pues ha conllevado a cambios revolucionarios en diferentes campos, tales como aplicaciones espaciales, médicas, comerciales, telefonía, militar, industrial, científica, etc. El DSP es distinguido de otras áreas de las ciencias de la computación puesto que el tipo de datos que éste usa son **señales**, las que, en la mayoría de los casos, se originan como senseo de datos del mundo real, tales como vibraciones sísmicas, visualización de imágenes, ondas sonoras, etc. El DSP es la matemática, los algoritmos y las técnicas usadas para manipular estas señales después que ellas han sido convertidas a una señal digital. En otras palabras, el DSP es un área que se dedica al análisis y procesamiento de señales que han sido originadas en el mundo real

representándolas como dígitos (números). Esto quiere decir, que una parte integral de aplicaciones con DSP es la conversión de señales del mundo real – voces análogas, música, video, velocidad de motores, vibraciones de la tierra – a valores numéricos para el procesamiento en un DSP. Esta conversión se realiza a través de una conversión analógica a digital y asimismo, la conversión de señales originadas por el DSP al mundo real se realiza a través de un conversor digital análogo.

3. ¿Dónde y para qué es usado el Procesamiento Digital de Señales?

Diversas son las áreas que han optado por incluir el Procesamiento Digital de Señales, y dentro de éstas se pueden nombrar algunas aplicaciones como:

Tabla I. Lista de las principales aplicaciones en las que el DSP juega un rol importante.

¿Dónde?	¿Para Qué?
DSP Propósito General	Filtrado Digital – Convolución - Correlación Transformadas - Filtros Adaptivos Generación de Formas de Onda
Gráficas / Imágenes	Rotación en 3D - Visión de Robots Transmisión y Compresión de Imágenes Patrones de Reconocimiento Animación y Mapeo Digital
Instrumentación	Análisis de Espectros - Generación de Funciones Análisis Transientes - Filtrado Digital
Voz	Mail de Voz - Reconocimiento de Voz Verificación de Voz
Control	Discos - Servos – Robots - Impresoras Láser Control Motores
Militar	Comunicaciones Seguras Procesamiento de Radares Procesamiento de Sonar Procesamiento de Imágenes Navegación Misiles Radio Frecuencia
Telecomunicaciones	Cancelación de Eco Líneas Repetitivas Multiplexión de Canales Modems - Ecualización Encripción de Datos Fax - Celulares Video Conferencias Comunicaciones
Autos	Control de Motores Análisis de Vibraciones Posicionamiento Global Navegación - Comandos de Voz Radio Digital Teléfonos Celulares
Industrial	Robótica Control Numérico Accesos de Seguridad Monitoreo de Líneas de Energía
Médicas	Monitoreo de Pacientes Equipamiento de Ultrasonido Herramientas de Diagnóstico Monitores Fetales

4. ¿Cuáles son los beneficios del Procesamiento Digital de Señales?

Obviamente el DSP presenta ventajas y desventajas, las que son indicadas a continuación.

Ventajas **:** Precisión.
Estabilidad.
Acumulación controlada del ruido.
Costo del hardware independiente de la complejidad.
Reprogramable.
Tarea a ser desarrollada por el DSP puede ser simulado.
Fácil de depurar, en un tiempo pequeño.
Menor costo y mayor fiabilidad.
Facilidad de transmisión y almacenamiento.

Desventajas **:** Ancho de Banda limitado por la razón de muestreo.
Rango dinámico limitado.
Cuantización del ruido.
Errores de redondeo.
No es posible una exacta reconstrucción de la señal analógica original a partir de muestras cuantificadas.

5. Comentarios Adicionales

Respecto a la capacidad del procesador, se tiene que ésta es una función de su ancho de datos (el número de bits manipulados) y el tipo de aritmética que posee (punto fijo o flotante). Un procesador de 32 bits tiene un ancho de datos mayor que uno de 24 bits el cual a su vez tiene un rango mayor que uno de 16 bits. DSP´s de punto flotante tienen rangos mayores que uno de punto fijo y cada tipo de procesador es ideal para un rango particular de aplicaciones. DSP´s de 16 bits son ideales para sistemas de voz tales como teléfonos, ya que ellos trabajan con un estrecho rango de frecuencias de audio. Stéreos de alta fidelidad requieren CAD de 16 bits y un procesador de 24 bits de punto fijo. Los 16 bits del conversor permiten capturar todo el rango de la señal de audio y los 24 bits del procesador permiten operar cómodamente los grandes valores resultantes de la operación con los datos. Procesamientos de imágenes, gráficos 3D y simulaciones científicas necesitan un rango dinámico mucho mayor y por lo tanto requieren procesadores de punto flotante de 32 bits y CAD de 24 bits.

6. Diseño e Implementación basadas en DSP

Existen 3 requerimientos básicos para aplicaciones de diseño e implementación basadas en DSP.

- Algoritmos.
- Software.
- Hardware.

Normalmente, en una primera etapa, la utilización de programas computacionales como MatLab es válida para implementar nuevos algoritmos, depurarlos y verificar su funcionamiento. Sin embargo, la materialización de los programas implementados en un PC bajo condiciones normalmente ideales dista de la implementación de esos programas en equipos tales como microcontroladores y/o procesadores digitales de señales (DSPs). Consideraciones como la aritmética finita, tiempo de procesamiento (normalmente un aspecto crítico), utilización de periféricos, etc., presentan restricciones que en un PC no se pueden emular o no se consideran por ser una etapa muy temprana de desarrollo.

Para llevar a cabo la implementación de algoritmos en sistemas basados en microcontroladores o en DSPs existen varios tipos de ambientes de trabajo que permiten programar, compilar y depurar las rutinas en un entorno gráfico de fácil manejo. Los sistemas más avanzados permiten la programación del hardware y la depuración de algoritmos en tiempo real. La finalidad de ello es poder depurar los algoritmos bajo condiciones reales de operación (al menos bajo condiciones lo más cercanas a la realidad posible), utilizar los datos provenientes de los periféricos y verificar las posibles restricciones temporales en la ejecución del programa implementado. Todo esto con el fin de reducir el tiempo entre la concepción de un nuevo producto y su salida al mercado.

7. Configuración de memoria del DSP, archivo .cmd

En cada proyecto se **debe** incluir un archivo de configuración de memoria, cuya extensión es **.cmd**, este archivo indica al compilador como está distribuida la memoria física de la que se dispone y en qué lugares se desea que ubique las diferentes variables y trozos de código que componen el proyecto.

- Directivas para la definición de la memoria:

El archivo .cmd se puede dividir en tres secciones:

- Definición de constantes.
- Definición de bloques de memoria (comando MEMORY).
- Ubicación del código y variables en la memoria del DSP (comando SECTIONS)

Estos dos últimos puntos se describen a continuación:

MEMORY

```
{
        name : o = constant, l = constant[, fill = constant];
}
```

El comando MEMORY divide la memoria en una serie de grupos definidos por el usuario. Estos grupos tienen directa relación con el hardware a utilizar ya que indica al compilador donde ubicar físicamente el programa a ejecutar y las variables a utilizar. Una mala definición de memoria puede provocar que trozos de código o variables se pierdan o a subutilizar el hardware del que se dispone.

Los parámetros de la sección MEMORY son:

name	:	Nombre del bloque a definir.
o	:	Dirección de origen del bloque.
l	:	Largo del bloque definido.
fill	:	Valor con el que debe pre-llenarse el bloque definido.

SECTIONS

{
name : load = constant
name : load > nom_bloque
name : >> nom_bloque1 | nom_bloque2 | ...
}

name	:	Nombre de la sección.
load = constant	:	Ubica la sección a partir de la dirección indicada por *constant*.
load > nom_bloque	:	Ubica la sección dentro del bloque definido en MEMORY y de nombre *nom_bloque*.
>> nom_bloque1 | nom_bloque2 | ... :	:	Intenta colocar la sección en alguno de los bloques definidos en MEMORY, en caso de no entrar en alguno de ellos lo dividirá entre los diferentes bloques.

- Secciones por defecto creadas por el compilador:

.text	:	Es una sección inicializada que contiene el código ejecutable, es decir, el programa en sí.
.bss	:	Es la sección donde se guardan las variables **no** inicializadas, reservando un espacio de memoria para este efecto.
.data	:	En esta sección se guardan las variables inicializadas.
.cinit	:	Sección para las variables globales y estáticas que han sido inicializadas.
.const	:	Idéntico a .cinit pero dedicado a variables de tipo string y variables definidas con el comando *const*..
.switch	:	Tablas para implementar instrucciones del tipo *switch*.
.stack	:	Sección para guardar las variables al realizarse un llamado por interrupción. Además

almacena variables locales y pasa argumentos a las diferentes funciones.

.sysmem : Memoria para las funciones de ubicación dinámica de memoria (*malloc*).

- Funciones especiales #pragma:

#pragma CODE_SECTION(símbolo, "nombre_de_sección");

La función #pragma CODE_SECTION envía una función o trozo de código de programa con el nombre *símbolo* a la sección *nombre_de_sección* definida en el archivo .cmd.

Ejemplo:

En el archivo C:

```
#pragma CODE_SECTION(fn, "my_sect")

int fn(int x)
{
   return x;
}
```

En el archivo cmd:

```
SECTIONS
{
...
.my_sect          >       SDRAM
}
```

#pragma DATA_SECTION(símbolo, "nombre_de_sección");

Trabaja de manera similar a CODE_SECTION, pero ubicando una variable en una dirección determinada de la memoria, para esto la variable **debe** ser una variable de tipo *global.*

Ejemplo:

En el archivo C:

```
#pragma CODE_SECTION(var_data, "my_sect")
float       var_data;
```

En el archivo cmd:

```
SECTIONS
{
...
.my_sect          >       SDRAM
}
```

- Verificación del mapa de memoria (archivo .map)

Una vez realizada la compilación del proyecto, se generarán una serie de archivos, los que el compilador ubica en la carpeta **Debug** dentro de la carpeta del proyecto actual. Entre los archivos generados destacan el archivo **.out** que es el que contiene el código compilado y listo para cargar en el DSP y el archivo **.map** que muestra el mapa de memoria generado a partir de las indicaciones dadas por el archivo **.cmd**.

El archivo **.map** es un archivo ASCII por lo que puede abrirse con cualquier editor de texto. Este archivo entrega información sobre la ubicación y tamaño de las secciones definidas, ubicación de las funciones definidas en el código C y aquellas definidas por las librerías incluidas en el proyecto, y finalmente la ubicaciones de todas las variables **globales** utilizadas ordenadas primero por nombre y después por dirección de memoria, de esta forma es posible verificar la correcta ubicación de las diferentes variables ya sea en la memoria interna o externa del DSP.

Generación de Señales

8. Definiciones Previas

Señal : Cualquier cantidad física que varía en el tiempo y que lleva información, generalmente acerca del estado o comportamiento de un sistema, como por ejemplo radar, música, voz, sonar, etc.

Tiempo Real : Capacidad de realizar un proceso de manera "instantánea", tal como lo hace un circuito análogo.

Instantáneo : Que se realiza con tal velocidad que el tiempo de cálculo es despreciable con respecto a la velocidad de respuesta del medio.

9. Representación de Señales

- Señales en Tiempo Continuo

Una señal es continua si toma todos los valores posibles en un intervalo tanto finito como infinito. Generalmente éstas pueden estar representadas en el dominio del tiempo de la forma $s(t) = A\cos(wt + \theta)$, donde A corresponde a la amplitud de la señal, w la frecuencia en radianes por segundo y θ es la fase en radianes, o también pueden ser representadas por fasores, de la forma $s(t) = A\angle\theta$.

- Señales en Tiempo Discreto

Esta señal se expresa como $s(kT) = A\cos(wkT + \theta) = A\cos(2\pi fkT + \theta)$. La señal es periódica si para algún k^* y N enteros se cumple, $2\pi fk^*T = 2\pi fN \rightarrow k^* = N/(fT)$, donde N es el menor entero positivo y k^* resulta ser el periodo discreto. Por medio de MatLab es posible crear un pequeño script que muestre la representación discreta de una señal sinusoidal de 1[Hz], tal como se indica en la **Fig. 1.**, donde se ha empleado la función *stem* para enfatizar que la señal es muestreada en vez de ser una señal continua.

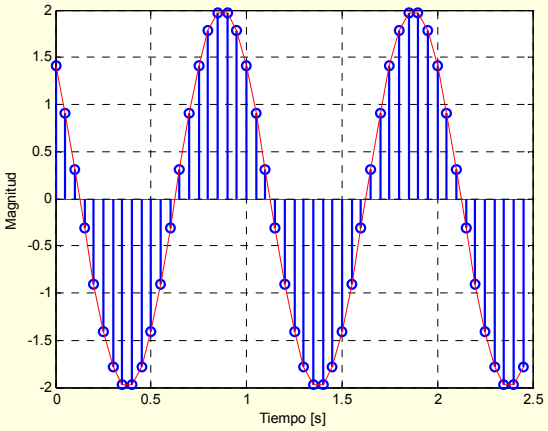

Fig. 1 Señal muestreada de una sinusoide.

Algunas señales elementales en tiempo discreto son las indicadas a continuación:

- Impulso $\delta[n]$, definida como: $\delta(n-m)=\begin{cases}1 & n=m\\ 0 & n\neq m\end{cases}$
- Escalón unitario $u[n]$, definido como: $u(n-m)=\begin{cases}0 & n<m\\ 1 & n\geq m\end{cases}$
- Rampa unitaria $u_r[n]$, definida como: $u_r(n-m)=\begin{cases}0 & n<m\\ n-m & n\geq m\end{cases}$

En las próximas sesiones se presentarán al estudiante los sistemas en tiempo discreto, deseables en aplicaciones donde se requiere diseñar un dispositivo o algoritmo que ejecute una operación sobre una señal en tiempo discreto.

10. Osciladores Digitales

- Oscilador digital bicuadrático

Este oscilador es un sistema de segundo orden que proviene de calcular la transformada Z a una señal sinusoidal de tiempo continuo y su función de transferencia es:

$$H(z)=\frac{R\cdot\sin(\omega_0)\cdot z^{-1}}{1-2\cdot R\cdot\cos(\omega_0)\cdot z^{-1}+R^2\cdot z^{-2}} \qquad (1)$$

Para que la amplitud de la oscilación se mantenga constante, es necesario que los polos del oscilador se encuentren sobre la circunferencia unitaria ($R=1$). Además, ω_0 representa la frecuencia de oscilación, obtenida como:

$$\omega_0 = 2 \cdot \pi \cdot \frac{f}{f_s} \tag{2}$$

Donde f_s corresponde a la frecuencia de muestreo y f la frecuencia de oscilación requerida.

La función de transferencia indicada en (1) puede ser sintetizada como se indica en la **Fig. 2**.

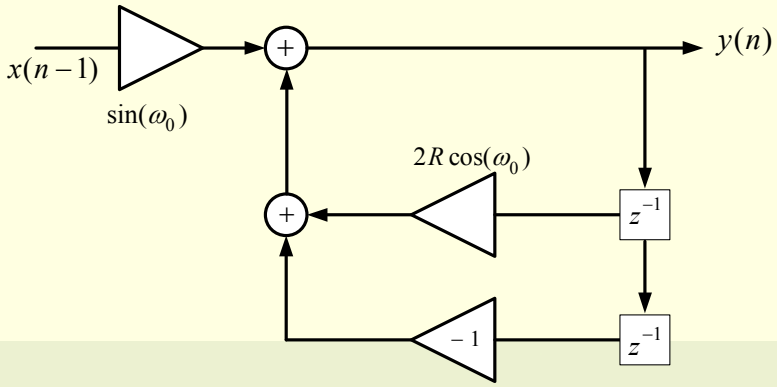

Fig. 2 Oscilador digital sinusoidal biquad.

Para que el sistema oscile, sólo se requiere que se aplique un impulso en la entrada x, y puesto que la transformada Z de un impulso es una constante, la salida será igual a la función de transferencia $H(z)$, obteniendo la oscilación deseada en la salida. En resumen, a partir de lo anterior, la ecuación recursiva que define la salida del oscilador digital está dada por:

$$Y(z) = \sin(\omega_0) \cdot U(z) + 2 \cdot \cos(\omega_0) \cdot Y(z-1) - Y(z-2) \tag{3}$$

- Oscilador digital en cuadratura

La principal característica de este oscilador es que genera dos señales de igual amplitud, pero en cuadratura, es decir, cuando una salida está en su valor alto, la otra está en valor cero, por lo que es posible tener fácilmente una señal *seno* y otra *coseno* a disposición.

Para generar dichas señales, es posible usar la siguiente representación matricial:

$$\begin{bmatrix} \hat{x}_1 \\ \hat{x}_2 \end{bmatrix} = \begin{bmatrix} \cos(\theta) & \sin(\theta) \\ -\sin(\theta) & \cos(\theta) \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \tag{4}$$

En donde $\hat{x}_i$, representa las salidas próximas, x_i, las salidas actuales ($i = 1, 2$) y θ representa la frecuencia normalizada.

La condición necesaria para generar las señales en cuadratura es que los autovalores de la matriz indicada en (4) estén sobre la circunferencia unitaria. De esta manera se pueden obtener distintas matrices que también producirán osciladores, pero, para que estén en cuadratura y tengan la misma amplitud, los elementos de dicha matriz deben cumplir ciertas condiciones, tal como se explica a continuación.

Al definir la representación matricial:

$$\begin{bmatrix} \hat{x}_1 \\ \hat{x}_2 \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}, \tag{5}$$

las condiciones necesarias son:

a.- El determinante de la matriz es 1, ($a \cdot d - b \cdot c = 1$), o que los autovalores de la matriz se encuentren sobre la circunferencia unitaria.

b.- Los valores propios de la matriz deben ser complejos conjugados ($|a+d| < 2$). Esto produce la oscilación.

c.- Para que el oscilador tenga salidas en cuadratura, se debe cumplir que $a = d$.

d.- Para que ambas salidas tengan igual amplitud, se debe cumplir que $b = -c$.

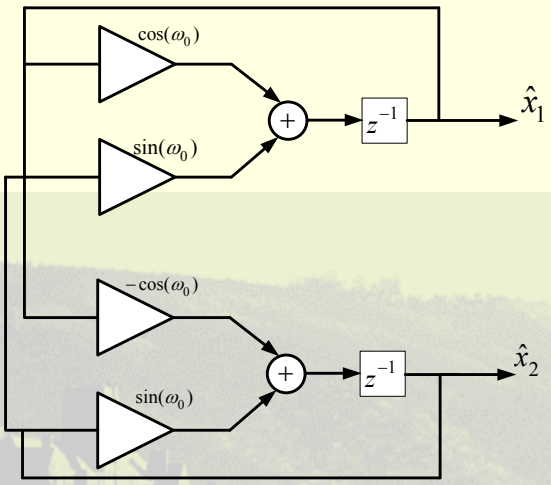

Fig. 3 Oscilador acoplado en cuadratura (Coupled - standard quadrature).

- Generación de señal digitales

Modulación de Amplitud (AM)

Esta señal tiene la forma general:

$$S(t) = A_c \cdot \left[1 + m \cdot f_s(t)\right] \cdot \cos(w_c \cdot t), \tag{6}$$

donde, $f_s(t) = \frac{w_c}{2 \cdot \pi}$ es la frecuencia de la portadora, $\|f_S(t)\| < 1$ es la señal de información y m corresponde al índice de modulación.

Una señal AM generada digitalmente puede ser realizada a través de la utilización de un oscilador sinusoidal más la adición de una señal externa cualquiera que cumpla con la restricción antes señalada.

Modulación de Frecuencia (FM)

La expresión de una señal portadora sinusoidal modulada en frecuencia tiene la forma:

$$S_{FM}(t) = A_c \cdot \cos(w_c \cdot t + K_f \cdot \int f(x) \cdot dx), \tag{7}$$

donde, $f(x)$ es una señal de información, K_f constante del sistema y w_c es la frecuencia angular de la señal portadora.

Para simplicidad en el análisis, se supondrá como señal de información un tono puro, esto es:

$$f(t) = A_M \cdot \cos(w_M \cdot t), \quad (8)$$

Entonces, la señal modulada toma la forma:

$$S_{FM}(t) = A_c \cdot \cos(w_c \cdot t + K_f \cdot \frac{A_M}{w_M} \cdot \sin(w_M \cdot t)), \quad (9)$$

La frecuencia instantánea de la señal portadora es:

$$w_i = w_c + K_f \cdot f(t) = w_c + K_f \cdot A_M \cdot \cos(w_m \cdot t) \ \ [rad/seg] \quad (10)$$

$$f_i = f_c + (K_f \cdot \frac{A_M}{2} \cdot p) \cdot \cos(w_m \cdot t) \ \ [Hz] \quad (11)$$

De la expresión anterior, se puede observar que la máxima desviación de frecuencia de la señal portadora será:

$$df = \frac{K_f \cdot A_M}{2\pi} \ \ [Hz] \text{ o bien } df = fd \cdot A_M \ \ [Hz] \quad (12)$$

donde, fd es la constante de desviación de frecuencia en $[Hz/V]$.

Se define el índice de modulación de una señal modulada en frecuencia como:

$$\beta = \frac{df}{f_M} \ \ [Hz] \text{ o bien } \beta = \frac{fd \cdot A_M}{f_M} \ \ [Hz] \quad (13)$$

Finalmente, la señal portadora modulada en frecuencia toma la forma:

$$S_{FM}(t) = A_c \cdot \cos(w_c \cdot t + \beta \cdot \sin(w_M \cdot t)). \quad (14)$$

Laboratorio

11. DSP a utilizar en el Laboratorio

Durante el desarrollo del laboratorio **se utilizará una tarjeta TMS320C6711 DSK** (C6711 DSK). La tarjeta C6711 DSK, está conectada al puerto paralelo del PC corriendo en Windows. Instalado en el PC, se encuentra el software que permite compilar, ensamblar, depurar y cargar las aplicaciones de la tarjeta DSK. En la **Fig. 4**, se muestra una vista superior de la tarjeta, con sus principales componentes. Para la entrada y salida de señal análoga, el DSK posee un **CODEC STEREO TLC320AD535,** con una frecuencia de muestreo de hasta 11025 Hz. Lea el 'DataSheet' del Codec para estar al tanto de sus principales características, disponible en la página web del curso. Para fines prácticos, el CODEC sólo muestrea a 8000 Hz. Para cada laboratorio necesitará estar equipado con un generador de señales y un osciloscopio, tanto para verificar el procesamiento

de sus algoritmos como para visualizar el procesamiento de las formas de onda. Para algunos laboratorios se aconseja utilizar en lo posible un osciloscopio con analizador de espectro.

La Tarjeta C6711 DSK es controlada por el PC mediante el programa **Code Composer Studio** (CCS). Ésta es una herramienta diseñada para agilizar y enriquecer el proceso de desarrollo para programadores que crean y prueban aplicaciones de procesamiento digital de señales en tiempo real proveyendo herramientas para configuración, construcción, depuración y análisis de programas.

Esta experiencia tiene como objetivo introducir las principales características del CCS. La intención no es proveer una exhaustiva descripción de cada característica, sino, sólo revisar las características fundamentales para comenzar los desarrollos de software para DSP.

Como primer paso en este laboratorio se debe comprobar la correcta conexión de la tarjeta C6711 DSK y la comunicación con el programa CCS. Para ello **conecte la tarjeta DSK al PC, energícela y abra la aplicación CCS**. Si todo sale bien se debería abrir el ambiente de trabajo, si no, un mensaje de error aparecerá. En ese caso verifique los pasos anteriores e intente nuevamente.

11.1 Creando un proyecto básico

Descargar la carpeta de códigos pertenecientes al laboratorio 1 desde la página web del curso. En el programa CCS seleccione el menú de ítems **Proyect→New**. En **Project Name** escribir un nombre al proyecto (por ejemplo Lab1), en **Location** seleccionar la carpeta de trabajo recién creada y presionar **finalizar**. Notará que CCS creó un archivo de proyecto "**.pjt"** donde se guardan los seteos y referencias de los archivos usados en el proyecto. Es importante no modificar el contenido de este archivo ya que ello seguramente hará que el proyecto no funcione adecuadamente. Ahora se comenzará a agregar archivos al proyecto, para esto, desde el menú de ítems seleccionar **Proyect→Add Files to Proyect**. Los programas necesarios para este proyecto son:

ejer.c	**:**	Contiene el código del programa.
func.c	**:**	Contiene las funciones de inicialización de registros para el DSP.
memoria.cmd	**:**	Indica al compilador la configuración de la memoria del DSP, es decir, cantidad de memoria interna y externa, para programa y datos, además de definir en qué secciones se almacenarán ciertos tipos de variables.
c6xdsk_RH.h	**:**	Definiciones de variables en función de las direcciones de memoria del DSP. (Este archivo no es necesario agregarlo al proyecto, pues el compilador lo hace solo).
rts6701.lib	**:**	Librería de funciones para la depuración en tiempo real. El archivo rts6701.lib se encuentra entre los archivos instalados por el compilador en la carpeta **.\c6000\cgtools\lib**.

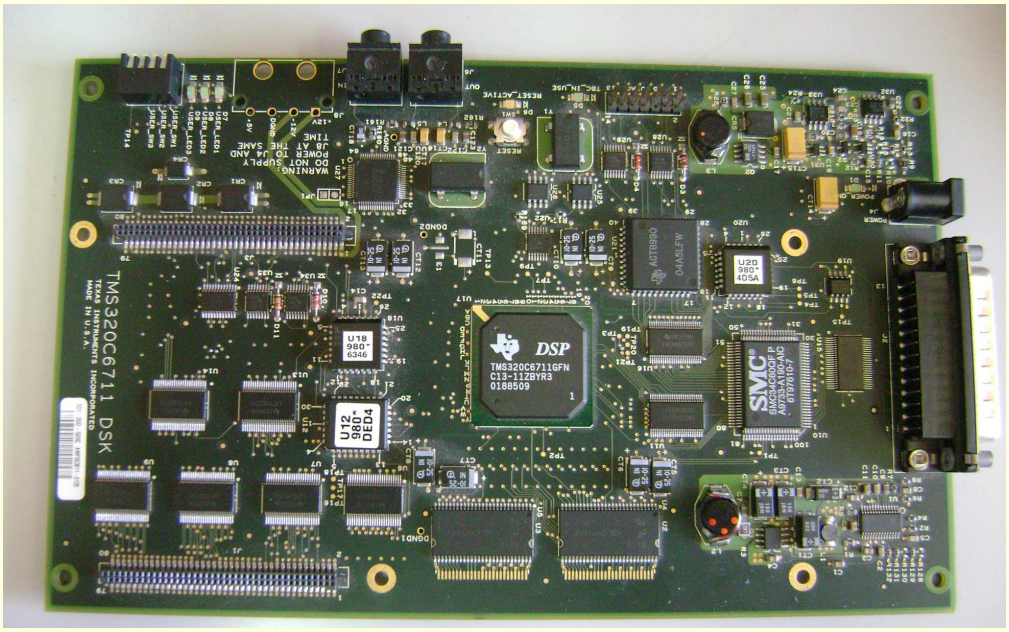

Fig. 4. Vista superior tarjeta C6711 DSK.

Para construir y correr el programa se deben seguir los siguientes pasos:

Desde el menú de ítems seleccionar **Proyect→Rebuild All (**o el ícono **)**, con esto, el programa CCS recompila, reensambla y reenlaza todos los archivos del proyecto. Mensajes acerca del avance de este proceso se observan en la ventana inferior en el CCS.

Para cargar el programa en la tarjeta seleccionar desde el menú de ítems **File→Load Program**. Buscar el Archivo **Proy1.out** en la carpeta **Proy1\Debug** y apretar el botón aceptar para cargar en la tarjeta. Esto abre automáticamente una ventana con el código Assembler del programa seleccionado.

11.2 Visualización de variables

- Watch Windows

Cuando se desarrollan o prueban programas es común tratar de verificar el valor de una variable durante la ejecución de un programa, para esto el programa CCS cuenta con **Breakpoints** y **Watch Window** para observar dichos valores. Los Breakpoints son 'banderas' que indican al DSP detener la ejecución del programa en el punto indicado. Esto permite, por ejemplo, conocer el valor de una variable justo después de la ejecución de algún algoritmo, de modo que se pueda revisar el correcto funcionamiento de la función o procedimiento que implementa dicho algoritmo, o verificar el correcto llamado a alguna rutina de interrupción. Para agregar un **Breakpoint** basta con hacer doble *click* en la línea del programa en C o en Assembler, otra forma, es ubicar el cursor en la posición deseada y seleccionar el icono de la barra de herramientas. Un punto rojo indicará la presencia del **Breakpoint.** Para quitar un **Breakpoint** se repite el procedimiento descrito anteriormente.

Las **Watch Window (**  **)** son ventanas que permiten visualizar el valor de las diferentes variables utilizadas en el programa (recuerde que las variables locales existen **sólo** dentro de las funciones que las crean). La **Fig. 5** muestra una **Watch Window** típica.

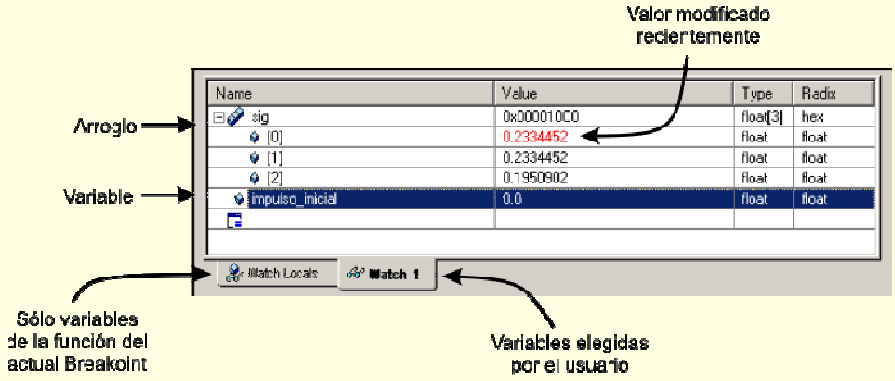

Fig. 5. Watch Window.

La primera columna indica el nombre (o dirección) de la variable a visualizar. La segunda muestra el valor de la variable (en rojo si ha cambiado su valor recientemente). La tercera indica el tipo de variable visualizada (int, long, unsigned, float, etc.). Finalmente la cuarta columna permite seleccionar el formato de la visualización de la variable (decimal, hexadecimal, char, etc.). Para agregar una variable en la lista, basta con hacer doble *click* al final de ella y escribir el nombre de la variable. Recuerde que en C una variable existe sólo dentro de la función donde es definida, salvo en el caso de las variables globales. Para ejecutar el programa en el DSP seleccione **Proyect→Run** o presionando en el ícono  , el programa se ejecutará hasta encontrar un **breakpoint** o llegar al final de la ejecución. Otra opción es seleccionar **Proyect→Animate** (), al igual que **Run** al llegar a un **Breakpoint** el programa se detendrá, pero continuará automáticamente pasada una cantidad de tiempo indicada en **Option→Customize→Animate Speed**.

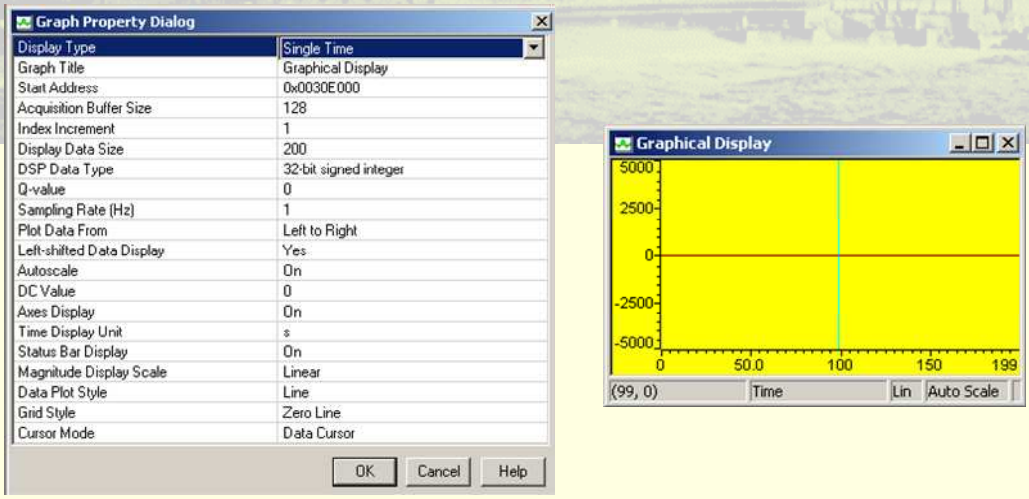

Fig. 6.a) Panel de propiedades para un gráfico; b) Gráfico

- Gráficos

Si bien las Watch Windows son muy útiles ya que permiten visualizar y modificar variables internas del DSP, cuando se trabaja con variables de tipo arreglo con un alto número de elementos, la visualización en forma de tabla puede no ser la más recomendable, siendo preferible un formato gráfico. Esto se logra a través de **View→Graph→Time/Frequency**, apareciendo la ventana indicada en la **Fig. 6**:

Ahí, las opciones más importantes son:

Display Type	**:**	Gráfico simple, doble, transformada de Fourier, entre otras.
Start Address	**:**	Dirección del primer dato (Recuerde que en C el nombre del arreglo coincide con la dirección inicial del mismo).
Acquisition Buffer Size	**:**	Cantidad de datos a ser tomados a partir de la dirección base.
Index Increment	**:**	Separación entre los datos a tomar.
Display Data Size	**:**	Cantidad de datos a mostrar en el gráfico.
DSP Data Type	**:**	Formato de los datos a ser representados: int, long, float.

NOTA1: Debido a algunos problemas en el driver que maneja la tarjeta DSK junto al PC, cada vez que se cargue un programa se deberá hacer un reset de la tarjeta (**Debug→Reset CPU**) y luego volver a cargar el programa. De no hacer esto podría, haber problemas en la carga y posterior ejecución.

NOTA2: En el caso de trabajar en un proyecto con múltiples archivos, no es necesario compilarlos todos cada vez que se ha modificado alguno. El programa CCS ofrece la posibilidad de realizar una compilación incremental, es decir compila sólo aquellos archivos que han sido modificados desde la última compilación, para esto haga *click* en el icono  **.**

NOTA3: Si el archivo **.out**, ha sido cargado al menos una vez en la tarjeta, es posible usar la opción de **File→Reload Program** para cargar el último programa compilado.

12. Uso del Codec de Audio y Generación de Señales

- Codec TLC320AD535

El TLC320AD535 es un codec de dos canales, uno de voz y uno de datos. Posee 2 puertos seriales independientes para la comunicación con el procesador. En la parte análoga cuenta con resistencias y condensadores para setear la ganancia y los polos de los filtros analógicos de entrada y salida.

Debido a la configuración de la tarjeta DSK varias de las capacidades del codec no son utilizadas o han sido limitadas. El ejemplo más significativo es la frecuencia de muestreo del conversor A/D y D/A, la cual es de

sólo 8000 [Hz]. Así, los filtros de entrada y salida se encuentran configurados para trabajar con una frecuencia de corte en torno a los 3.5 [KHz]. Ya que el codec posee sólo un canal de 16 bits para comunicarse con el DSP, este canal es utilizado para transmitir datos y para cambiar la configuración del codec. El protocolo utilizado depende del bit menos significativo:

- Si el bit es 0, significa que los 15 bits más significativos son de dato.
- Si el bit es 1, significa que la siguiente palabra que se escriba **modificará** la configuración del codec como las ganancias de salida y/o entrada.

Es sumamente importante tener en cuenta este protocolo a la hora de enviar los datos al codec para evitar errores en las señales. Más información con respecto al funcionamiento del codec se encuentra disponible en el datasheet del codec disponible en la página web del curso. En el archivo *func.c* se definen las funciones **mcbsp0_read()** y **mcbsp0_write(dato)** que permiten leer y escribir un dato del/en el codec de audio respectivamente. La definición de estas funciones es la siguiente:

```
void mcbsp0_write(int out_data)
{
  int output_reg;
  output_reg = McBSP0_SPCR & 0x20000;
  while ( output_reg == 0)
  {
    output_reg = McBSP0_SPCR & 0x20000;
  }
  McBSP0_DXR = out_data;
}


int mcbsp0_read()
{
  int input_reg;
  input_reg = McBSP0_SPCR & 0x2;
  while ( input_reg == 0)
  {
    input_reg = McBSP0_SPCR & 0x2;
  }
  input_reg = McBSP0_DRR;
  return input_reg;
}
```

Note que **mcbsp0_read()** es de tipo **int** ya que devuelve el dato leído en este formato, además no tiene argumentos, ya que no es necesario enviar datos adicionales al codec. Por otra parte, **mcbsp0_write()** es de tipo **void** ya que sólo escribe en el codec y no devuelve ningún valor, pero requiere del argumento *dato*, que corresponde al valor a enviar al codec. A pesar de estas diferencias, ambas funciones se basan en el uso del registro McBSP0_SPCR, mostrado en la **Fig. 7**. Tal como su nombre lo indica, este registro es el encargado de configurar la comunicación serial utilizada por el DSP para la comunicación con el codec. En particular el bit 1 **RRDY**, designado como de sólo lectura (letra R en la parte inferior), cambia a 1 cuando el DSP ha terminado de leer un dato desde el periférico (guardado en el registro **DRR**), en este caso el codec. Por otra parte el bit 17 **XRDY**, también designado como de sólo lectura, cambia a 1 cuando el DSP ha terminado de escribir en el periférico el dato almacenado previamente en **DXR**.

Serial Port Control Register (SPCR)

31					26	25	24
Reserved						FREE†	SOFT†
R-0						R/W-0	R/W-0

23	22	21		20	19	18	17	16
FRST	GRST	XINTM			XSYNCERR	XEMPTY	XRDY	XRST
R/W-0	R/W-0	R/W-0			R/W-0	R-0	R-0	R/W-0

15	14		13	12		11	10		8
DLB	RJUST		CLKSTP			Reserved			
R/W-0	R/W-0		R/W-0			R-0			

7	6	5		4	3	2	1	0
DXENA†	Reserved	RINTM			RSYNCERR	RFULL	RRDY	RRST
R/W-0	R-0	R/W-0			R/W-0	R-0	R-0	R/W-0

† Available only on C621x/C671x DSP and C64x DSP.
Legend: R = Read only; R/W = Read/Write; *-n* = value after reset

Fig. 7: Registro de configuración para el puerto serial McBSP.

13. Actividades

Aquí se detallan las actividades a realizar por el estudiante en el laboratorio y que deben ser reportadas en el informe final de laboratorio. Para este informe se requiere incluir tanto la parte teórica como la experimental, y todos los códigos usados y generados con sus respectivos comentarios. Se considerará la presentación del informe, el que esté completo, y trabajo extra realizado y debidamente comentado será recompensado con un máximo de 10 puntos. Presente sus resultados en forma ordenada, escrito en tercera persona, y evite errores ortográficos o de nomenclatura pues se descontarán puntos.

13.1 Parte Teórica

13.1.1 Explique al menos dos diferencias fundamentales entre un DSP y un microprocesador convencional.

13.1.2 ¿Por qué se necesita de un filtro pasabajos en la etapa previa a una conversión A/D?

13.1.3 Dibuje el diagrama de flujo de las funciones **mcbsp0_read()** y **mcbsp0_write(). E**xplique su funcionamiento utilizando "lenguaje normal" es decir **sin utilizar** nombres de registros, comandos, etc.

13.1.4 Utilizando Matlab, simule 40 [ms] de un oscilador digital de 50[Hz] y una frecuencia de muestreo de 8[kHz], utilizando el oscilador bicuad, bajo las siguientes condiciones:

- Valores exactos de las constantes $\sin(\omega_o)$ y $\cos(\omega_o)$.
- Error de +1% en el valor de las constantes $\sin(\omega_o)$ y $\cos(\omega_o)$.
- Error de -1% en el valor de las constantes $\sin(\omega_o)$ y $\cos(\omega_o)$.

Comente y explique los resultados obtenidos.

13.2 Parte Experimental

13.2.1 Cree un proyecto nuevo de acuerdo con lo explicado anteriormente, utilizando los archivos disponibles en la página web del curso, compílelo y ejecútelo. Mida la salida del codec con un osciloscopio.

13.2.2 Modifique el programa entregado de manera de almacenar en una tabla 320 puntos generados de manera consecutiva por el algoritmo entregado. Grafique estos resultados y muéstrelos a través de una Watch Window.

13.2.3 Modifique el programa original para que la salida del codec sea una señal senoidal de frecuencia 200[Hz]. Incluya los cálculos involucrados.

13.2.4 Genere en el DSP una señal de tipo AM utilizando como modulante la señal obtenida del punto anterior y como portadora una señal de 2[kHz] generada por un generador de señales externo. Obtenga la señal temporal y el espectro.

13.2.5 Grafique a lo menos dos periodos de las dos señales de salida que se obtienen de un generador de sinusoidales en cuadratura, cuando es configurado para operar a 1[kHz].

13.2.6 Utilizando la máxima amplitud posible del codec de audio, despliegue en un osciloscopio el producto de las dos señales obtenidas en el punto anterior.