Universidad Técnica Federico Santa María

DEPARTAMENTO DE INGENIERÍA ELECTRONICA

ELO 314 - Laboratorio de procesamiento Digital de Señales

Proyectos en LCDK y Recomendaciones Practicas

Estudiante	ROL
Rodrigo Graves	201621009-1
Ricardo Mardones	201621036-9

Paralelo: 1

Profesor

Gonzalo Carrasco

Ayudante

Jaime Guzmán

Fecha : 10 de agosto de 2021

Índice

Índice de figuras

Índice de cuadros

1. Configuración de proyecto ejecutable: CCS

1. Se importa el proyecto de la carpeta *lab0/build* al IDE ccs entregada junto con el repositorio del curso. Una vez importado, al intentar efectuar la construcción haciendo click en *build proyect* lo que se obtiene es una serie de errores y mensajes de *warnings*

   Entre ellos se indica que la ruta hacia los archivos del proyecto importado, en la sección referente a la compilación, no es accesible pues indica directorios que no pertenecen al dispositivo en el que se está trabajando. Las que se indican en el proyecto inicialmente se muestran en la figura 1

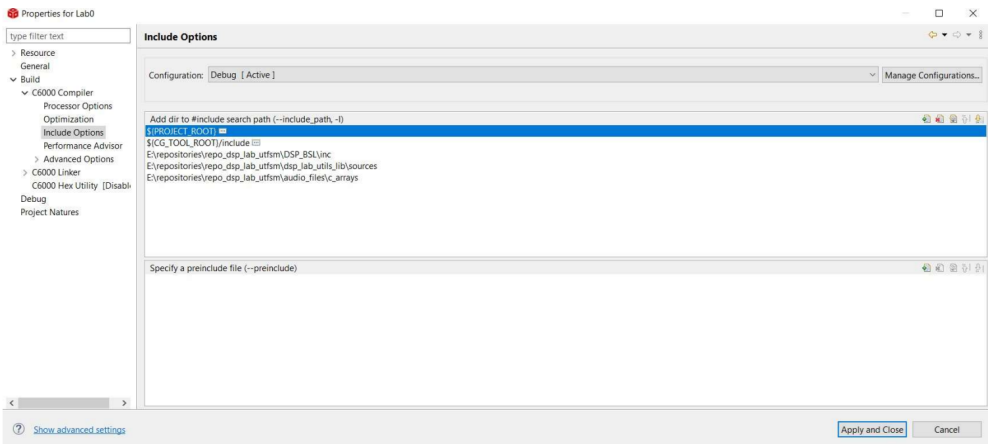

Figura 1: Configuración inicial del compilador con las rutas ajenas.

Modificando las rutas hacia la ubicación local de los archivos y añadiendo además la ruta hacia el directorio *DSP_AIC3016* , ya que también existían errores indicando que requiere archivos ubicados en ese directorio (ver figura 2), la sección *Build C000 Linker Include Options* de configura como se muestra en la figura 3

```
>> Compilation failure
subdir_rules.mk:16: recipe for target 'L1p2.obj' failed
"C:/Users/ricar/Desktop/labdsp_elo314_utfsm/lab0/sources/L1p2.c", line 13: fatal error #1965: cannot open
source file "L138_LCDK_aic3106_init.h"
1 catastrophic error detected in the compilation of
"C:/Users/ricar/Desktop/labdsp_elo314_utfsm/lab0/sources/L1p2.c".
Compilation terminated.
gmake: *** [L1p2.obj] Error 1
```

Figura 2: Error referente a la falta del directorio *DSP_AIC3016*

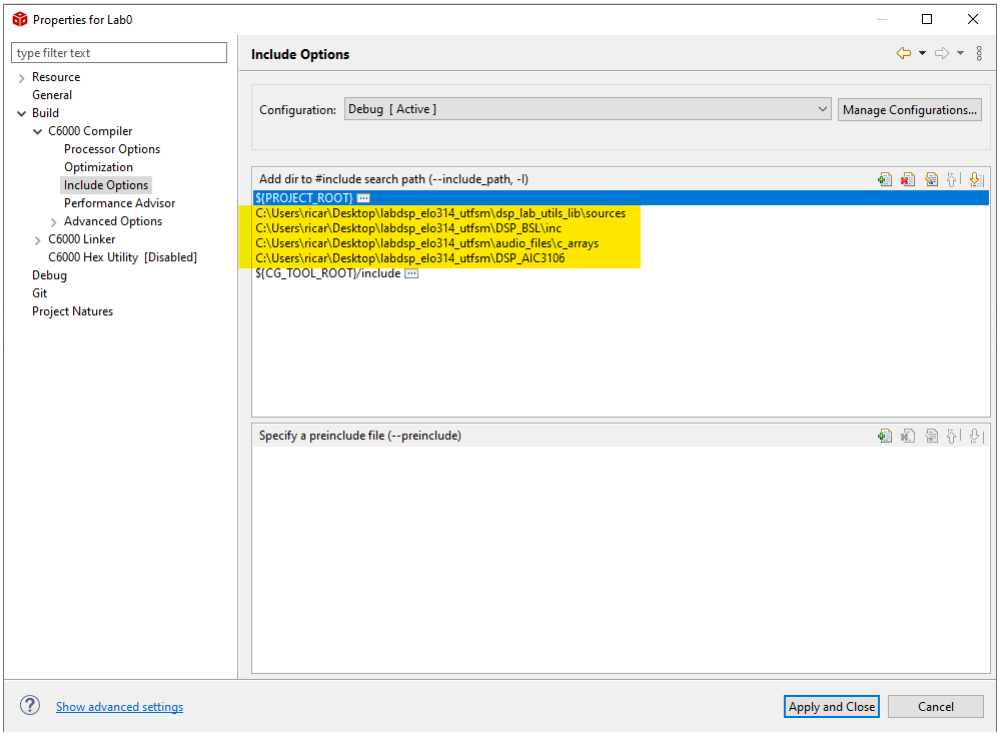

Figura 3: Configuración del compilador con las ruta locales de trabajo.

Por otro lado, se debe añadir la una dependencia extra en la sección *linker*, dado que inicialmente solo se encuentra una por defecto en el proyecto, esto se muestra en la figura 4

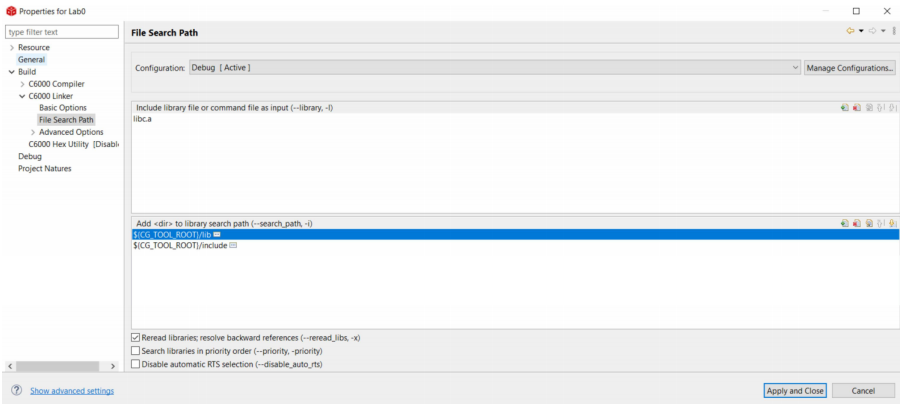

Figura 4: Configuración inicial del *linker* del proyecto.

Luego de añadir la dependencia necesaria la configuración queda de la siguiente forma

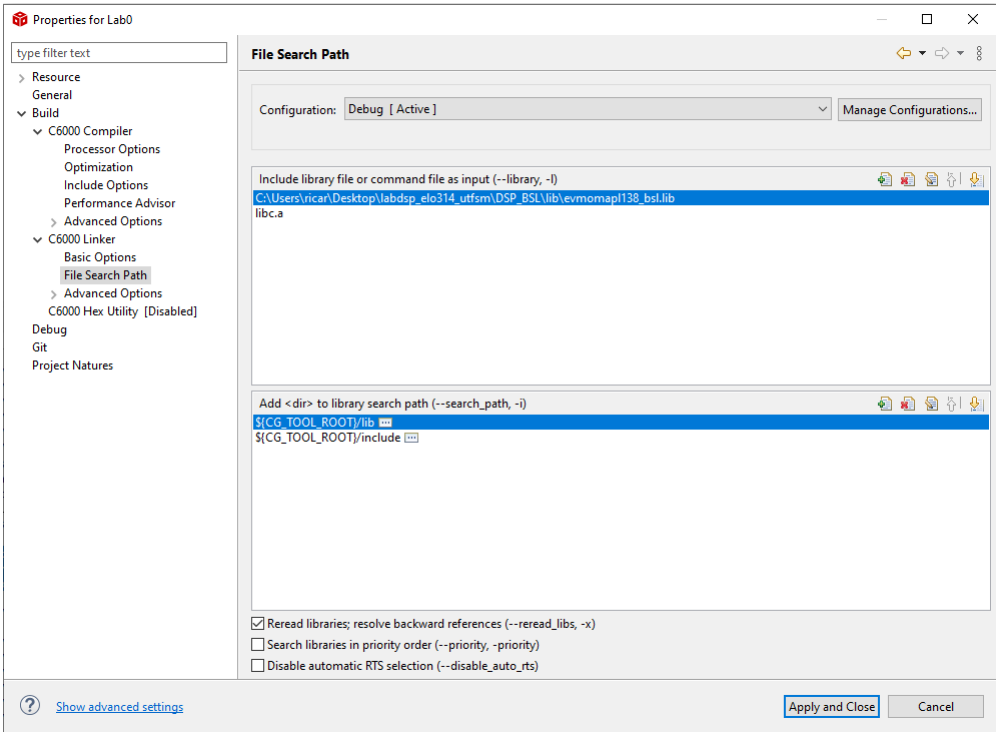

Figura 5: Configuración del *linker* del proyecto con dependencias ya añadidas.

Para solucionar el último error que impide la construcción correcta del proyecto se añade manualmente el archivo *L138_LCDK_aic3106_init.c* utilizando la ruta local de dicho archivo.

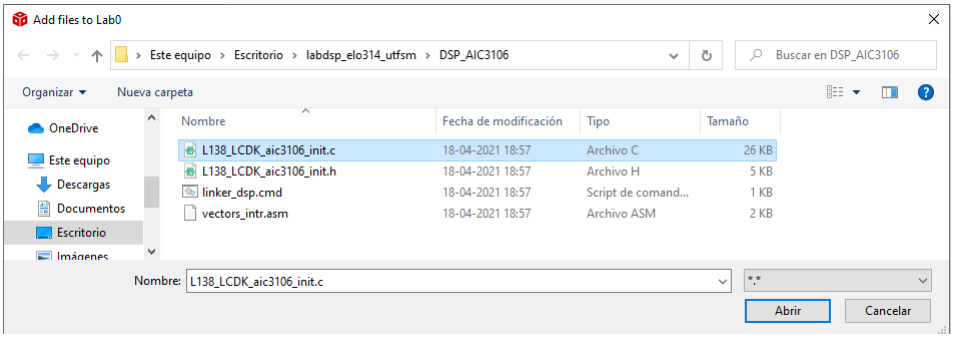

Figura 6: Se añade archivo *L138_LCDK_aic3106_init.c* requerido para construcción correcta del proyecto.

La figura 7 muestra como se ha logrado la correcta construcción del proyecto con las configuraciones hechas

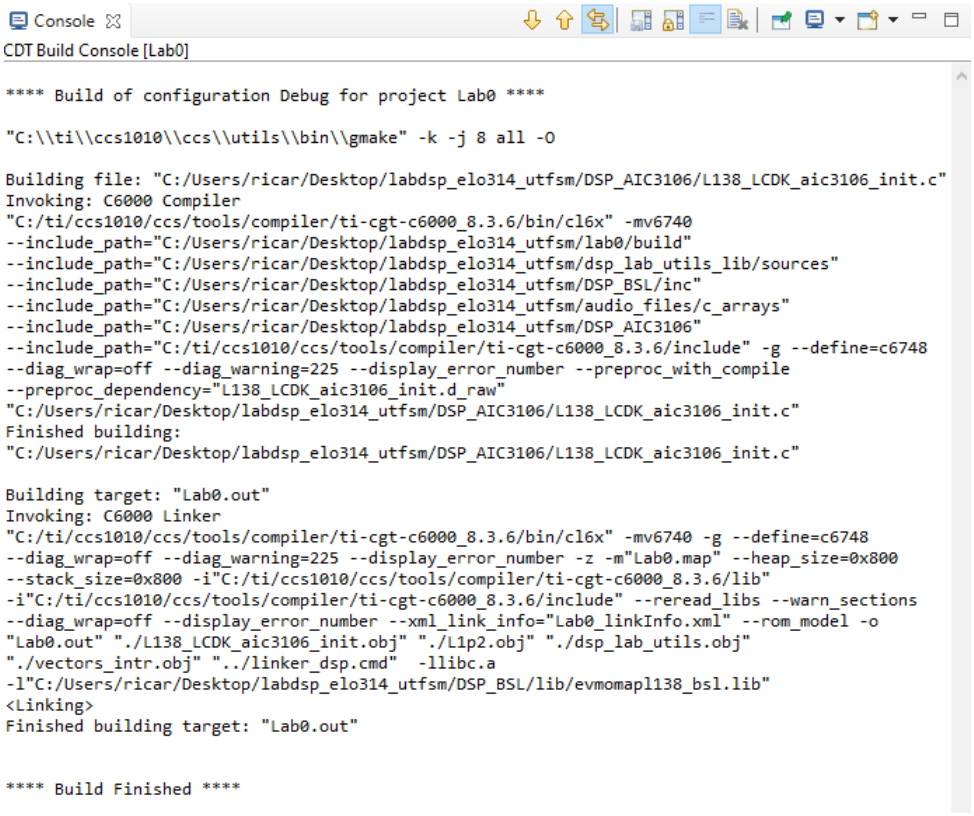

Figura 7: Construcción exitosa del proyecto

2. Inclusión de librería a un proyecto: MATHLIB

El proyecto sobre el cual se está trabajando utiliza en una de sus lineas la función *cosf()* incluida desde la librería *math.h*, sin embargo dado que es un proyecto pensado para ser implementado en una tarjeta LCDK que cuenta con un procesador de núcleo *C674x*, es recomendable reemplazar dicha función por alguna provista por la librería *MATHLIB*, que es recomendada por los fabricantes Texas Instrument (TI), en este caso se hará uso de la función *cossp()* (coseno *single-precision*, para *float* según MATHLIB).

Para esto se deben realizar algunos cambios en las configuraciones de compilación y enlace del proyecto. Primero se deben añadir las rutas que dirijan a los paquetes de la librería MATHLIB, como se muestran la figura 8

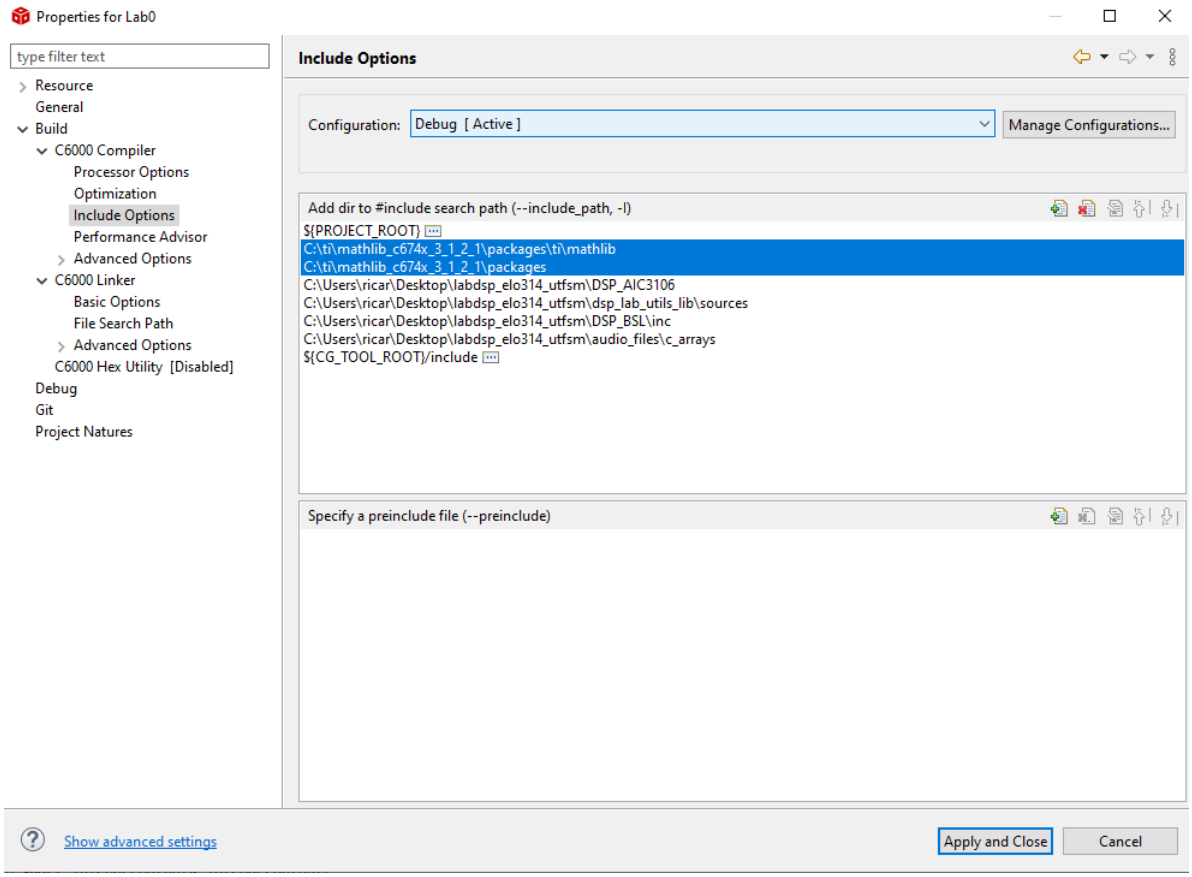

Figura 8: Configuración adicional al compilador para usar la librería MATHLIB

Además se debe agregar la dependencia *mathlib.lib*, indicando claramente el directorio en que se encuentra, esto se indica en la figura 9

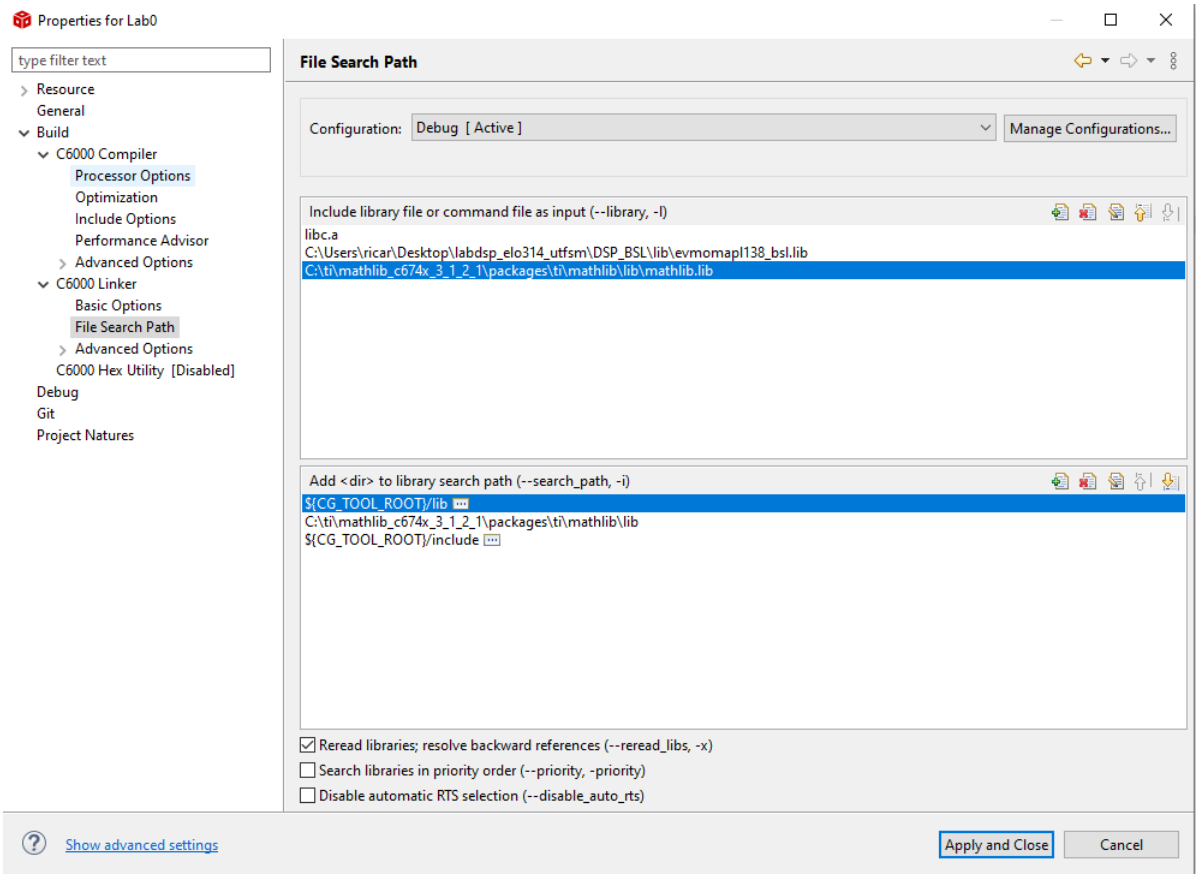

Figura 9: Configuración adicional al *linker* para usar la librería MATHLIB

Además en la sección de inclusión de librerías del código en el archivo *L1P2.c* se debe agregar la linea `#include ‘‘mathlib.h’’` y se debe modificar la línea 142 del código

```
1  #include "mathlib.h"
2
3  ...
4
5  ...
6
7  %linea original
8  cosSignal = (int16_t)( 2 * ampCosine * cosf(thetaCosineSignal) );
9
10 %linea modificada
11 cosSignal = (int16_t)( 2 * ampCosine * cossp(thetaCosineSignal) );
```

Revisando la documentación asociada a la librería MATHLIB posee diversas funciones que permiten trabajar de forma sencilla con elementos matemáticos, entre ellas funciones trigonométricas, logaritmos, exponenciales, y potencias. Todas estas implementadas tanto para trabajar con datos de precisión simple y doble, permitiendo una versatilidad para diversas arquitecturas al trabajar con distinto número de bits.

Esto resulta beneficioso ya que al estar estas funciones ya implementadas en una librería se reduce considerablemente el tiempo de procesamiento al momento de ejecutar operaciones matemáticas al trabajar con enfoques en tiempo real.

3. Estudio de la API de una librería: DSPLIB

1. La librería *DSPLIB* provee una serie de funciones diseñadas y optimizadas para facilitar el procesamiento digital de señales en tarjetas físicas, simplificando la implementación en lenguaje C, que suele ser el utilizado en este tipo de dispositivos. El que dichas funciones estén optimizadas con un objetivo claro permite reducir el tiempo de ejecución necesario para este tipo de procesamientos, que suele ser un punto de consideración al desarrollar proyectos que trabajen en tiempo real. Algunas de estas funciones se listan a continuación

   - **DSPF_sp_biquad:** Realiza filtrado de una señal usando un filtro biquad.
   - **DSPF_sp_convol:** Efectúa la convolución entre dos arreglos de entrada.
   - **DSPF_sp_fir_gen:** Realiza filtrado FIR a una matriz de entrada a partir de vectores con los coeficientes del filtro.
   - **DSPF_sp_iir:** Realiza filtrado IIR a una matriz de entrada a partir de vectores con los coeficientes del filtro.
   - **DSPF_sp_mat_mul:** Multiplica dos matrices de entrada

2. A continuación se detalla la forma de uso de algunas de las funciones antes mencionadas

   DSPF_sp_biquad
   Esta función realiza el filtrado de una señal cuyas muestras se almacenan en un arreglo, recibe seis parámetros en su implementación. La señal resultante se almacena en el arreglo y y la función no devuelve ningún valor (*void*) pues trabaja directamente con punteros a los parámetros.

   Prototipo función

```
1 void DSPF_sp_biquad(float *restrict x, float *b, float *a,
2                      float *delay, float *restrict y, const int n)
```

   Parámetros

   - float *x: Puntero al arreglo de entrada cuyo largo debe ser n, donde n corresponde al último de los parámetros requerido por la función y que se describe a continuación.
   - float *b: Puntero al arreglo de coeficientes B_k del filtro biquad cuyo largo debe ser 3. En este arreglo los coeficientes deben aparecer en el orden b_0, b_1, b_2
   - float *a: Puntero al arreglo de coeficientes A_k del filtro biquad cuyo largo debe ser 3. En este arreglo los coeficientes deben aparecer en el orden a_0, a_1, a_2
   - float *delay: Puntero arreglo de coeficientes de retraso asociado al filtro. Debe tener largo 2.

- float ∗y: Puntero al arreglo de valores de salida cuyo largo debe ser n, donde n corresponde al último de los parámetros requerido por la función y que se describe a continuación.
- const int n: Largo de los arreglos de entrada y de salida. Debe ser un número par.

DSPF_sp_mat_mul

La función realiza la multiplicación de dos matrices de entrada, cuyas dimensiones deben ser consistentes para poder ser multiplicadas. El resultado se almacena en la matriz de salida y y la función no retorna nada (*void*) pues trabaja directamente con los punteros a los parámetros.

Prototipo función

```
1 void DSPF_sp_mat_mul(float *x1, const int r1, const int c1,
2                      float *x2, const int c2, float *restrict y)
```

Parámetros

- float ∗x: Puntero a la primera matriz de entrada, de $r1 \cdot c1$ elementos.
- const int r1: Número de filas de la primera matriz de entrada.
- const int c1: Número de columnas de la primera matriz de entrada.
- float ∗x2: Puntero a la segunda matriz de entrada, de $c1 \cdot c2$ elementos.
- const int c2:Número de columnas de la segunda matriz de entrada.
- *restrict y: Puntero a la matriz resultante a partir de la multiplicación, el tamaño de este resultado es de $r1$ filas y $c2$ columnas.

4. Mediciones de validación de proyecto de ejemplo: Phaser

4.1. Prueba diseñada:

Los pasos propuestos para validar el correcto funcionamiento del efecto phaser en la LCDK son los que se presentan a continuación:

1. Con la placa LCDK programada en modo bypass, usar de señal de entrada al codec de audio una señal de ruido blanco (y por lo tanto de espectro distribuido) de 1 Vpp obtenido a partir de un generador de señales.

   Este es importante por 2 razones:

   - Verificar el funcionamiento del modo bypass.
   - Contar con señal de referencia para futuras comparaciones.

   Se utiliza una señal de 1 Vpp para no dañar el codec.

2. Importar archivo de audio en MATLAB y obtener su espectrograma con el script `Spectrogram_test`, previamente implementado.

   El espectrograma permitirá verificar que la señal sea de espectro distribuido en el tiempo

3. Repetir paso 1 con phaser activado.

   El espectro de está señal se verá afectado por el filtro dinámico.

4. Importar archivo de audio grabado en paso anterior en MATLAB y obtener su espectrograma con el script del paso 2.

   En el espectrograma se apreciarpa el movimiento de los nulos el filtro notch en el tiempo, por lo que el funcionamiento del phaser quedará validado.

4.2. Resultados de Validación:

Para esta parte se utilizó el código `Spectrogram_test`.

El espectrograma de la señal de ruido blanco pasada por la LCDK en modo bypass se muestra en a figura 10. Se aprecia el espectro plano esperado, salvo por otro lado se observa una caída en el espectro en torno a los 8 kHz. Esta caída se debe a que el códec de audio posee un filtro pasabajos a la frecuencia de Nyquist ($\frac{16}{2}$ kHz) con el fin de evitar los efectos de doblaje en frecuencia.

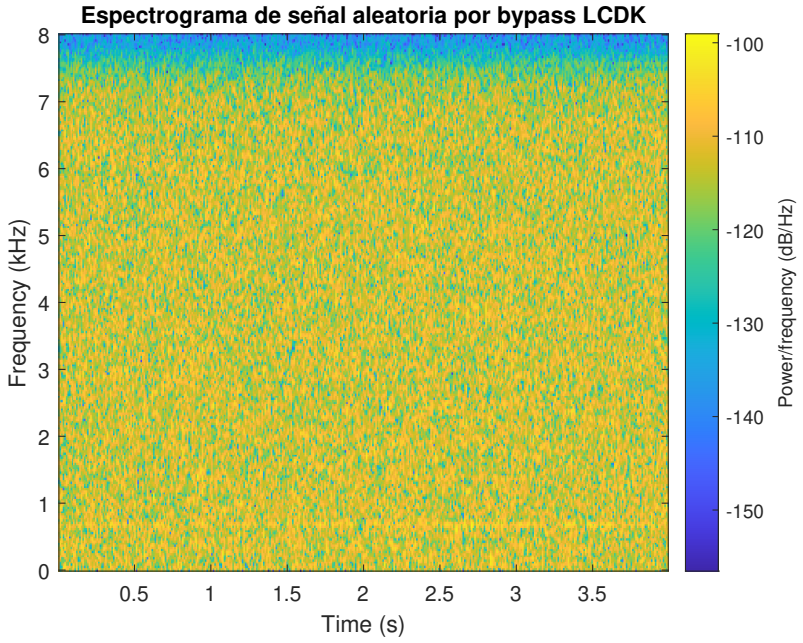

Figura 10: Espectrograma de señal de ruido blanco pasada por la LCDK programada en modo bypass. Ventanas 20 ms sin overlap.

El espectrograma de la señal de ruido blanco pasada por la LCDK en modo phaser se muestra en a figura 10. Se aprecia el cambio de la frecuecias atenuadas por el filtro notch en el tiempo, por lo que se valida el correcto funcionamiento del efecto de audio.

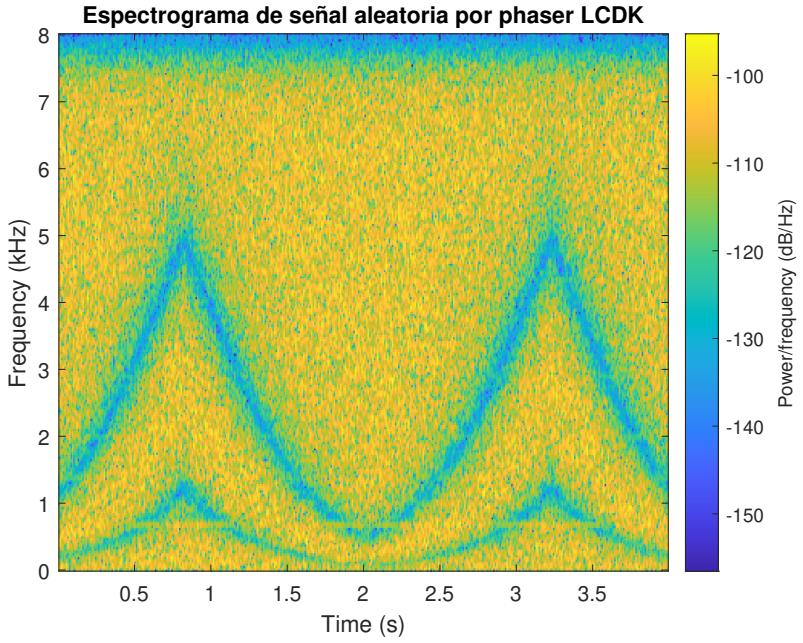

Figura 11: Espectrograma de señal de ruido blanco pasada por la LCDK programada en modo phaser. Ventanas 20 ms sin overlap.

5. Filtro de efecto de audio dentro del *codec* de audio y configuración

1. Antes de intentar hacer uso del *codec* se debe indagar sobre las características y forma de utilizar el método de filtrado que este dispositivo incluye. Para esto se recurre al *datasheet* de la tarjeta la TLV320AIC3106 provisto por Texas Instrument [1].

   En la sección **10.3.3.3.1 Digital Audio Processing for Playback** se describe la estructura del filtro y como programar el bloque asociado para poder utilizarlo en un proyecto. El filtro consta de dos filtros biquad en cascada, cuyos parámetros son modificables a conveniencia y se rigen por la siguiente ecuación

   $$H(z) = \left(\frac{N_0 + 2 \cdot N_1 \cdot z^{-1} + N_2 \cdot z^{-2}}{32768 - 2 \cdot D_1 \cdot z^{-1} - D_2 \cdot z^{-2}}\right) \cdot \left(\frac{N_3 + 2 \cdot N_4 \cdot z^{-1} + N_5 \cdot z^{-2}}{32768 - 2 \cdot D_4 \cdot z^{-1} - D_4 \cdot z^{-2}}\right)$$

   Donde tal como se mencionó se tiene libertad para definir los parámetros *N0-5* y *D0-5* asociados a ambos filtros. Además se debe tener en cuenta el signo en los parámetros dado que la expresión que se ha usado a lo largo de este curso para el diseño e implementación de filtros biquad difiere con la descrita en el *datasheet*

   $$H(z) = \left(\frac{b_0 + 2 \cdot b_1 \cdot z^{-1} + b_2 \cdot z^{-2}}{1 - 2 \cdot a_1 \cdot z^{-1} - a_2 \cdot z^{-2}}\right)$$

   Otra sección relevante del *datasheet* es la **10.6 Register Maps** la que describe la forma en que se guardan los parámetros definidos en los registros de memoria del dispositivos, donde se indica que los parámetros se dividen en dos registros, cada uno de 8 bits para poder abarcar el rango $[-32768, 32768]$

   Finalmente resulta útil conocer como modificar la amplitud de la salida generada por el dispositivo, esto se encontró en la sección **10.6.1 Output Stage Volume Controls** y será relevante para el proceso que se describirá a continuación.

2. Se implementa en MATLAB un script que permite encontrar los coeficientes de los filtros biquad con el fin de obtener una respuesta suficientemente cercana a la respuesta en frecuencia deseada. Cabe destacar que por la limitante impuesta por el número de bits se debe corregir la amplitud de uno de los filtros para poder trabajar dentro del rango de valores permitido, en este caso se corrige multiplicando por un factor de 0.8 veces.

[1] Ver en https://www.ti.com/lit/ds/symlink/tlv320aic3106.pdf?ts=1628519638980ref$_url$ = *https*

```
1
2  %Coeficientes del filtro de orden 4
3  F = [0 221 1764 4410 6615 8820 22050]/22050;
4  %correccion para entrar en [-32768, 32768]
5  A_corr = [2.1 2 1.3 1 1 1.2 1.5]*0.8;
6
7
8  %obtencion de coeficientes
9  [b,a] = yulewalk(4,F,A_corr);
10
11 [h,w] = freqz(b,a);
12
13
14 %obtencion de coeficientes para los filtros
15 [sos, g] = tf2sos(b,a);
16
17 b1 = sos(1,1:3)
18 a1 = sos(1,4:6)
19
20 b2 = sos(2,1:3)
21 a2 = sos(2,4:6)
```

En la figura 12 se pueden ver las gráficas obtenidas para la respuesta en frecuencia ideal y la respuesta en frecuencia generada con el script.

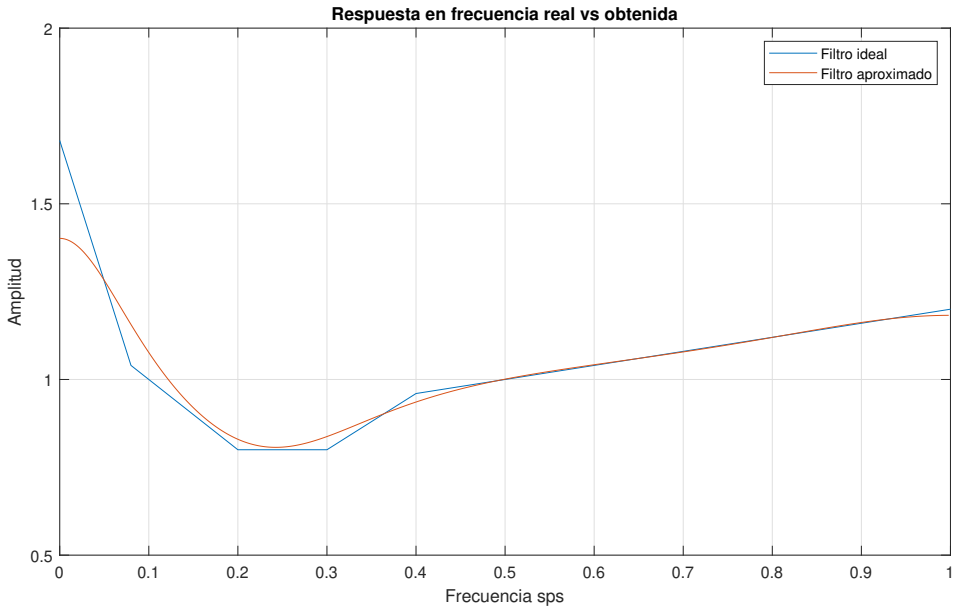

Figura 12: Respuestas en frecuencias ideal y obtenida mediante MATLAB

Los coeficientes para los filtros biquad obtenidos mediante la función *tf2sos* se muestran en la figura 13

```
b1 =

    1.0000   -0.2014   -0.1935

a1 =

    1.0000   -0.3184   -0.3038

b2 =

    1.0000   -0.9466    0.3990

a2 =

    1.0000   -0.7470    0.2785
```

Figura 13: coeficientes obtenidos mediante *tf2sos* para configurar los filtros biquad del *codec* de audio.

Se debe considerar la discrepancia de signos que se señaló anteriormente para los coeficientes antes de utilizarlos como parámetro en el dispositivo. Además se debe recordar que para poder trabajar dentro del rango adecuado permitido por el dispositivo corrigió la ganancia del filtro por un factor de 0.8, este efecto se debe corregir y se puede hacer corrigiendo el volumen de salida de audio del dispositivo (tal como se indica en la sección 10.6.1 Output Stage Volume Controls del *datasheet*)