**INSTITUTO POLITÉCNICO NACIONAL**
**ESCUELA SUPERIOR DE CÓMPUTO**


# Teoría de Comunicaciones y Señales

## "Muestreo y Reconstrucción"

Resumen

Aplicación de la multiplicación de 2 archivos WAV (mono – mono, stereo – stereo), para muestrear una señal y posteriormente aplicar TDF para pasar al dominio de la frecuencia y multiplicar por un filtro pasa – bajas (ideal) para recuperar la señal original.

**Por:**

**Joel Mauricio Romero Gamarra**

Profesor:
EDUARDO GUTIÉRREZ ALDANA

Diciembre 2017

# Índice

## Contenido

## Introducción:

Multiplicar 2 señales puede tener distintas ventajas, uno de ellos podría ser el obtener información de ellas en el dominio de la frecuencia, por ejemplo, podemos calcular la transformada de Fourier de una señal f(t) para pasarla al dominio de la frecuencia y obtener F(W), posteriormente, crear una señal que represente al sonido de una vocal en el dominio de la frecuencia, y si al hacer la multiplicación queda 0, entonces eso quiere decir que las señales no tienen nada en común y por lo tanto descartamos esa vocal.

Otro uso que se le puede dar a la multiplicación de señales, es multiplicarla por un tren de impulsos en el dominio del tiempo, ya que hacer la multiplicación de cualquier señal con un tren de impulsos es muestrear la señal, por lo tanto, nos sirve para hacer un muestreo de señales.

La estructura de un archivo WAV nos permite ver muy fácilmente la forma en la que 2 señales se van a multiplicar, dependiendo si es mono o stereo, ya que, como sabemos, un archivo de tipo mono utiliza únicamente 1 canal a diferencia de un stereo que utiliza los 2, en la Figura 1 se observa la estructura de un archivo WAV.

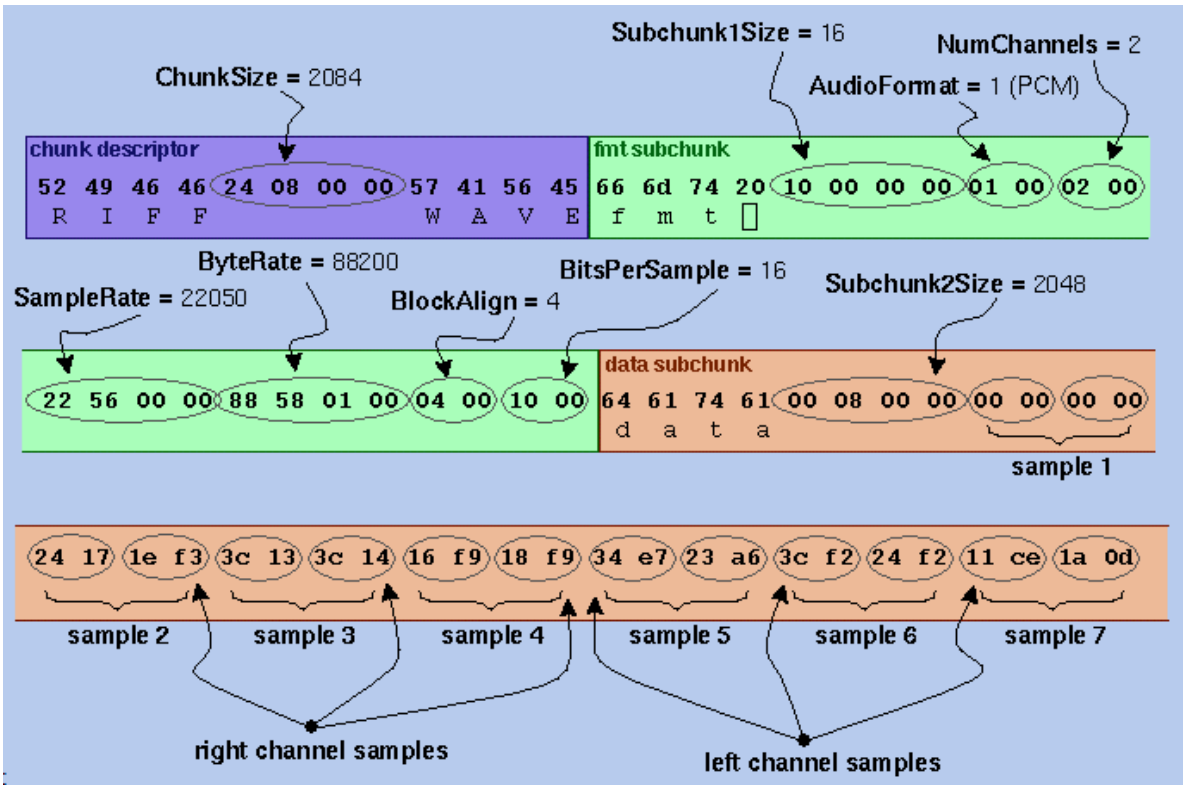


*Figura 1. Estructura de un archivo stereo*[1]

Como se puede observar, dice que el número de canales es 2, esto quiere decir que es un archivo estéreo y por lo tanto cada muestra se va a componer de 2 canales, el canal izquierdo y el canal derecho, en la Figura 2 se muestra un ejemplo de un archivo stereo.

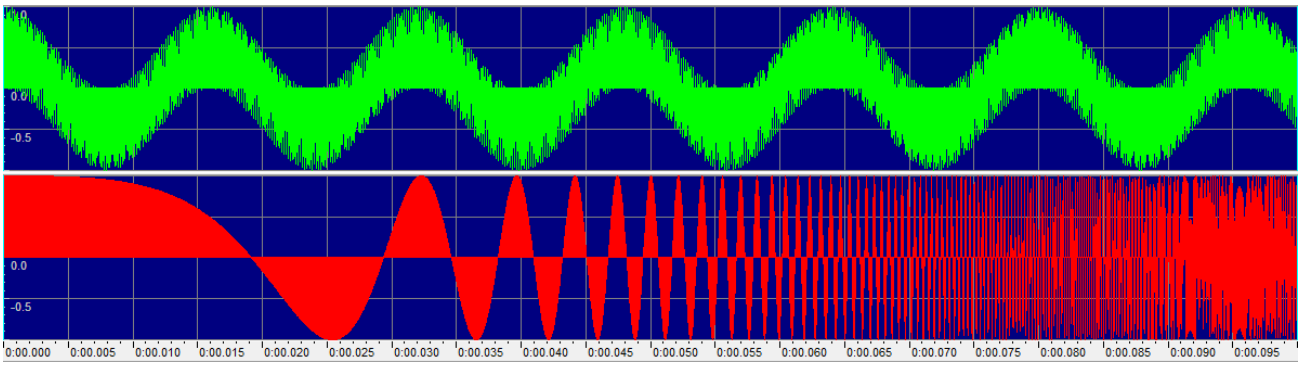


*Figura 2. Ejemplo de archivo stereo (2 canales)*

Como podemos ver, hay 2 señales distintas arriba y abajo, la señal que se encuentra arriba (la de color verde) representa al canal izquierdo en una muestra, mientras que la señal que se encuentra abajo (la de color rojo) representa al canal derecho de la muestra, como se puede ver podemos tener 2 señales distintas en cada canal. A continuación, en la Figura 3, se muestra un ejemplo de un archivo mono para que quede un poco más clara la diferencia.

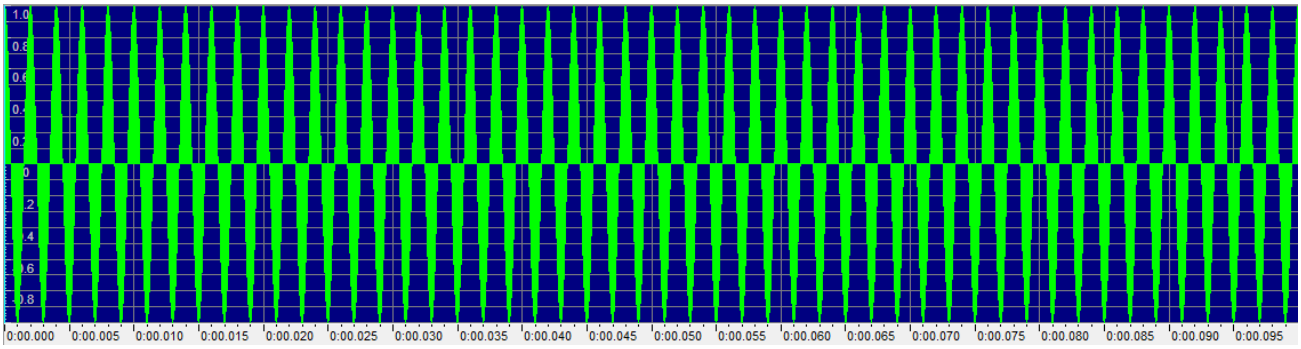


*Figura 3. Ejemplo de archivo mono (1 canal)*

Podemos ver que, en este tipo de archivo, solo existe un “color”, esto quiere decir que únicamente se está utilizando un canal y por lo tanto cada muestra podríamos decir que representa solamente 1 canal. En la siguiente sección, se explica cómo se tomarán los archivos para hacer la multiplicación de las señales.

## Análisis Teórico:

Como ya se mencionó en la Introducción, el archivo de tipo mono utiliza 1 canal mientras que el de tipo stereo utiliza 2, por lo tanto para realizar la multiplicación vamos a cada muestra del archivo mono como cualquier número real que toma valores entre -32, 767 y 32, 767, y al tomarlos como un real, entonces vamos a hacer la multiplicación de 2 números como la hacemos comúnmente, el único error que podría haber es cuando ambas muestras sean 32, 767 (de los 2 archivos) y esto es un problema debido a que el valor más grande que puede tomar un short es 32, 767, y al hacer la multiplicación nos quedaría 1, 073, 676, 289, y esto no puede ser representado en un short por lo cual se estaría regresando al valor más pequeño (-32, 767) hasta dar toda la vuelta y regresar hasta obtener un valor dentro del rango que puede tomar, lo cual sería incorrecto.

Para corregir este detalle, lo único que se tiene que hacer es un simple re – dimensionamiento de cada muestra, es decir que cada muestra que sea leída la voy a dividir entre 32, 767 (máximo valor de un short) para que nos de un 1, y al multiplicar 1 * 1 (que serían valores máximos), nos queda 1. Posteriormente, el valor final de la multiplicación va a estar en el rango de -1 a 1, y volvemos a multiplicar por 32, 767 para que la herramienta pueda hacer la graficación correctamente de cada multiplicación.

Este mismo principio aplicará para las muestras de un archivo stereo, sin embargo el cambio está en que en 2 archivos de tipo mono la multiplicación es tan sencilla como lo que se acaba de explicar, la primer muestra del primer archivo se multiplica por la primer muestra del segundo archivo y nos dará como resultado la primer muestra del archivo de salida, la segunda muestra del primer archivo por la segunda muestra del segundo archivo nos dará como resultado la segunda muestra del archivo de salida y así sucesivamente ya que como se mencionó anteriormente, estas muestras se van a tomar como números reales.

En el caso de los archivos stereo, al manejar 2 canales para hacer una correcta multiplicación se va a tomar 1 muestra (formada por 2 canales, el izquierdo y el derecho) como un número complejo, esto quiere decir que tendría su parte real y la parte imaginaria, en el caso de un archivo mono se mencionó que cada muestra era formada por 1 canal y en este caso de 2, entonces, el principio de hacer la multiplicación muestra a muestra se sigue cumpliendo, solamente que, ahora al tener un par de números complejos, la multiplicación no se hace de forma tan inmediata como si fueran 2 números reales, la fórmula que representa la multiplicación de 2 números complejos se muestra a continuación:

$$(a + ib) \cdot (c + id) = \sigma + i\omega$$

Realizamos la multiplicación como si se tratara de una multiplicación de binomios, es decir que todos los elementos se multiplican, se agrupan los elementos y se realiza la suma:

$$a \cdot c + ia \cdot d + ib \cdot c + i^2 b \cdot d$$

Pero, sabemos que $i^2 = -1$, por lo tanto, nos queda de la siguiente forma:

$$a \cdot c + ia \cdot d + ib \cdot c - b \cdot d$$

Agrupando términos:

$$(a \cdot c - b \cdot d) + i\,(a \cdot d + b \cdot c) = \sigma + i\omega$$

Donde:

- i: Parte imaginaria del número complejo (ω)
- a: Canal izquierdo de la muestra del primer archivo
- b: Canal derecho de la muestra del primer archivo
- c: Canal izquierdo de la muestra del segundo archivo
- d: Canal derecho de la muestra del segundo archivo

Como podemos ver, para obtener solamente el canal izquierdo de la muestra en el archivo de salida, es necesario a, c, b y d, esto quiere decir que es necesario leer la muestra completa de cada archivo

para poder encontrar solo 1 canal, sin embargo, en cada **fread** se lee un canal y esto hará que el ciclo se modifique un poco porque se van a hacer 2 fread por cada iteración en vez de 1.

## Software (librarías, paquetes, herramientas):

- Sublime Text 3[2]
- GoldWave versión 4.26[3]

## Procedimiento:

Lo primero que debemos hacer es recibir los 2 archivos y leer su cabecera (ya que aquí tenemos los datos importantes que servirán para todo el programa), por ejemplo, el número de canales que tiene el archivo (ya que de esto depende si es un archivo mono o un archivo stereo) y por lo tanto, de que forma hacer la multiplicación.

Posteriormente, debemos verificar cuál de los archivos tiene la mayor longitud, ya que nos quedaremos con el archivo que sea mayor y tomaremos esa longitud para que no haya problemas (el excedente, quedará con puros ceros a la salida ya que está multiplicando algo por 0).

El crear el archivo de salida no es tan simple como había sido para los primeros 2 programas, debido a que en esos la cabecera únicamente se copiaba del archivo de entrada al de salida, en este caso hay que copiar la cabecera del archivo mayor, así que, la función que habíamos creado para hacer la lectura y copiado de la cabecera se va a cambiar un poco para hacerlo después de verificar cual de los archivos es el mayor.

Una vez que ya tenemos copiada la cabecera, ya sabemos que tipo de archivo es, por lo que debemos comenzar a codificar el algoritmo propuesto en el análisis teórico para hacer la multiplicación de 2 archivos mono (mostrado a continuación):

```
for (i = 0; i < (cab_1.SubChunk2Size / 2); i ++)
{
          fread (&real1, sizeof (short), 1, archivo_1);
          real1_1 = (real1 / max);
          if (i >= (cab 2.SubChunk2Size / 2))
                    real2 = 0;
          else
                    fread (&real2, sizeof (short), 1, archivo_2);
          real2_1 = (real2 / max);
          real1 = (real1_1 * real2_1 * max);
          fwrite (&real1, sizeof (short), 1, archivoSalida);
}
```

En la parte superior, se muestra el código en lenguaje C para hacer la multiplicación de 2 archivos mono en caso de que el archivo 1 sea mayor o igual al archivo 1. Como podemos ver, la variable "real" se multiplica al final por max que es 32, 767 (valor máximo de short) para volver a escribir los datos en su dimensión correcta. Como ya se había mencionado, al tratarse de 1 canal cada muestra se toma como un real y por lo tanto la multiplicación es literalmente multiplicar 2 reales.

A continuación, se muestra la codificación del algoritmo para hacer la multiplicación de 2 archivos stereo:

```
for (i = 0; i < (cab_1.SubChunk2Size / 4); i ++)
{
          //Guardamos la parte real del n◆mero complejo
          fread (&real1, sizeof (short), 1, archivo_1);
          real1_1 = (real1 / max);
          if (i >= (cab_2.SubChunk2Size / 4))
          {
                    real2 = 0;
                    imaginario2 = 0;
          }else
          {
                    fread (&real2, sizeof (short), 1, archivo_2);
                    fread (&imaginario2, sizeof (short), 1, archivo_2);
          }
          real2_1 = (real2 / max);

          //Guardamos la parte imaginaria del n◆mero complejo
          fread (&imaginario1, sizeof (short), 1, archivo_1);
          imaginario1_1 = (imaginario1 / max);
          imaginario2_1 = (imaginario2 / max);

          real1 = (((real1_1 * real2_1) - (imaginario1_1 * imaginario2_1)) * max);
          imaginario1 = (((real1_1 * imaginario2_1) + (imaginario1_1 * real2_1)) * max);
          fwrite (&real1, sizeof (short), 1, archivoSalida);
          fwrite (&imaginario1, sizeof (short), 1, archivoSalida);
}
```

Como podemos ver, está explícitamente el proceso en las últimas 2 líneas (antes de los fwrite) de como se calcula tanto la parte real como la parte imaginaria y posteriormente se hace únicamente la estructura, es fácil ver como la programación es prácticamente la fórmula mostrada en la sección del análisis teórico.

En la sección de resultados, se muestran varios ejemplos de 2 señales de entrada (tanto de archivos con 1 canal y con 2 canales), así como la salida de la multiplicación de esas 2 señales. Como recordatorio, en las Figuras 4 a _ se muestra como crear distintos tipos de señales.

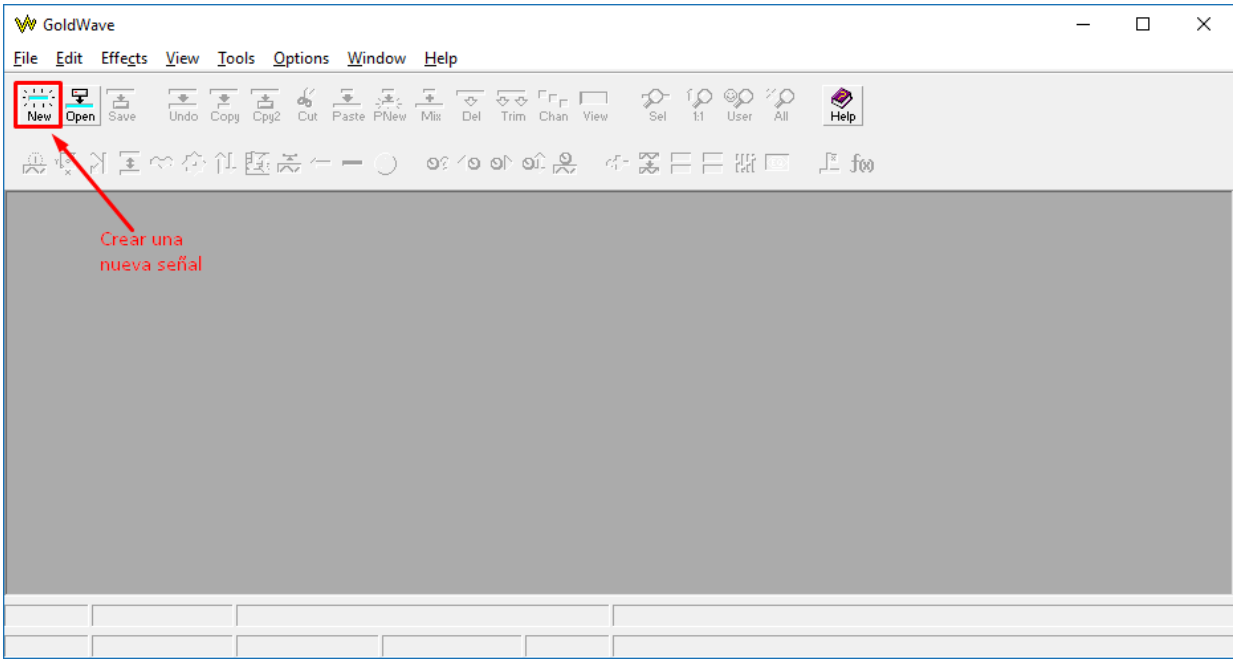


*Figura 4. Generar señal en GoldWave[3] (1)*

Primero debemos dar click en el botón que dice **New**, posteriormente, nos aparecerá la siguiente ventana mostrada en la Figura 5 para asignar las características a nuestra señal a crear:

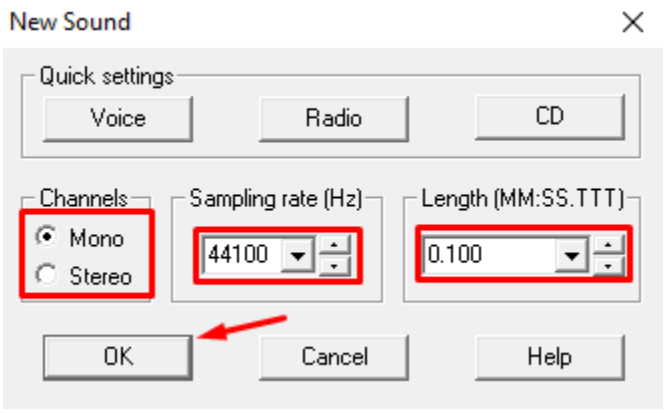


*Figura 5. Generar señal en GoldWave³ (2)*

En la ventana, vemos que tenemos 3 opciones para modificar, como la frecuencia de muestreo, la longitud y el número de canales o tipo de archivo. Posteriormente al dar click en **Ok**, nos generará un archivo como el siguiente:

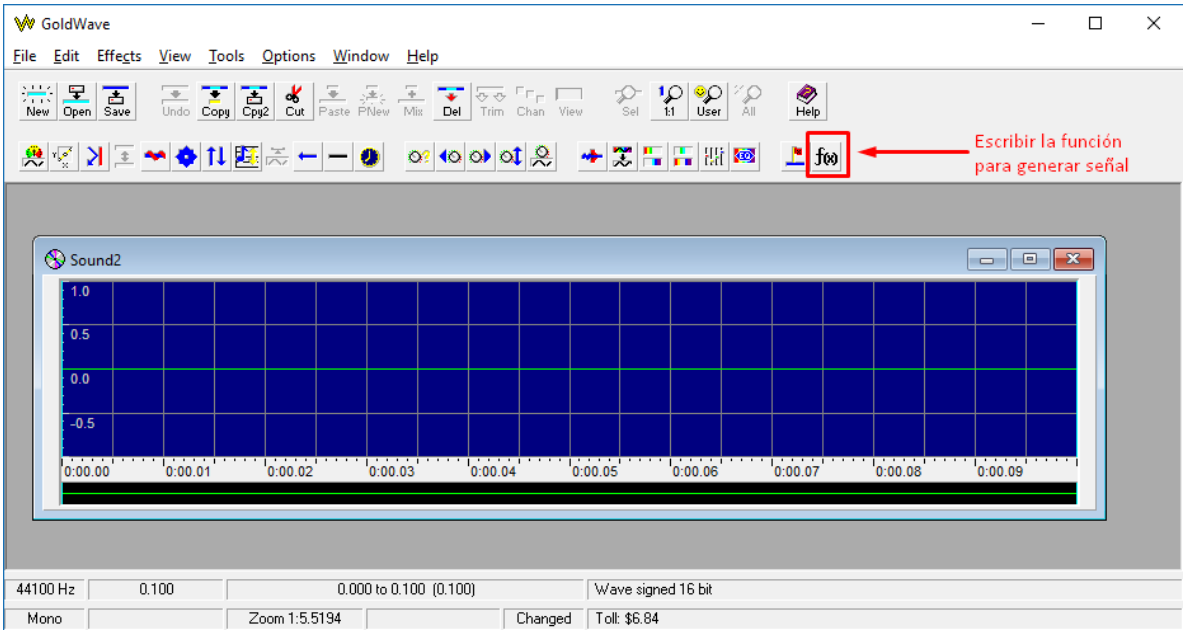


*Figura 6. Generar una señal en GoldWave³ (3)*

Como podemos ver, la señal en la Figura 6 se encuentra en 0, sin embargo, para cambiarlo damos click en el botón f(x) y escribimos una función para generarla. Por ejemplo, la función cos(2*pi*f*t) y quedará la siguiente señal creada mostrada en la Figura 7:

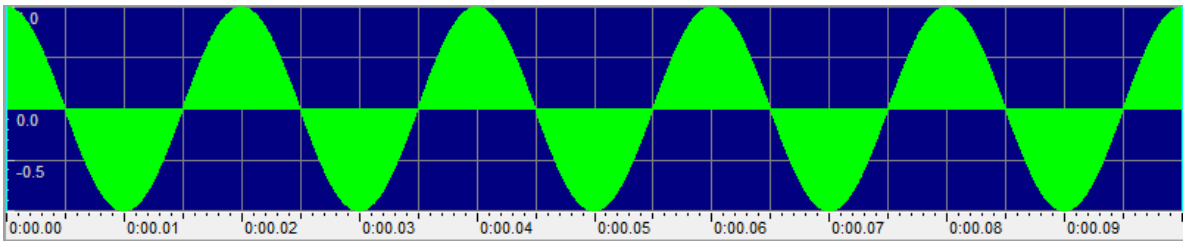


*Figura 7. Señal generada en GoldWave[3]*

Ya que sabemos como crear señales, procedemos a realizar las pruebas con varias señales mostrando las salidas de las multiplicaciones en la sección de resultados.

## Resultados

Como ya sabemos, Gold Wave únicamente grafica valores entre -1 y 1, por lo tanto, para probar al programa crearemos 2 señales muy simples (estéreo), ya que como ya se mencionó anteriormente, se toman como números complejos, y en teoría, si tenemos los números complejos: 1 + i y 1 + i, al hacer la multiplicación de estos 2 números complejos, el resultado será: **2i**

Este resultado, no se podría graficar, sin embargo, es una buena prueba para el programa ya que esto provocaría un desbordamiento y se regresaría el valor a los números negativos. La señal de entrada (ya que es la misma), se muestra en la Figura 8.

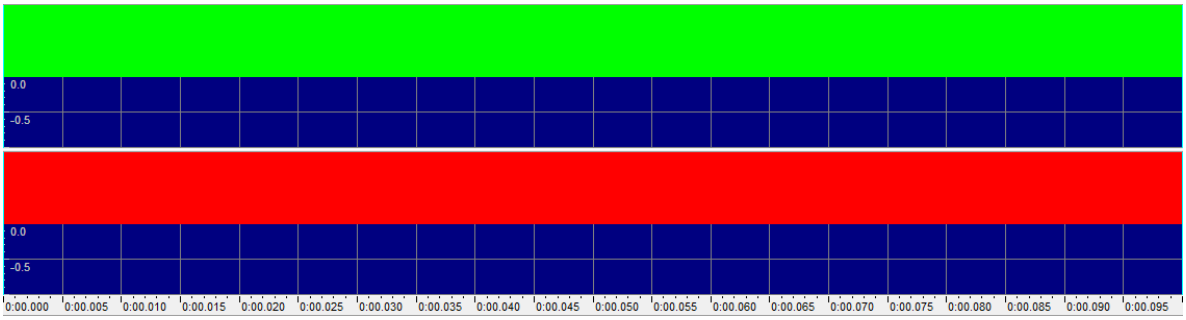


*Figura 8. Señales de entrada estéreo*

El resultado de hacer la multiplicación de estas señales se muestra en la Figura 9.

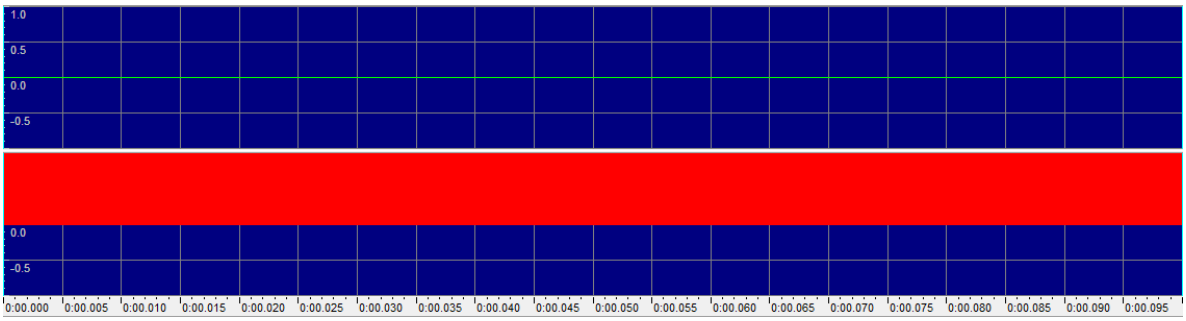


*Figura 9. Señal de salida resultado de multiplicar (1 + i) * (1 + i)*

Como podemos ver, la multiplicación fue de manera exitosa ya que el valor no regresó a los números negativos y en su lugar se quedó en 1.

A continuación, en las Figuras 10 y 11 se muestran las señales de entrada 1 y 2 que serán multiplicadas.

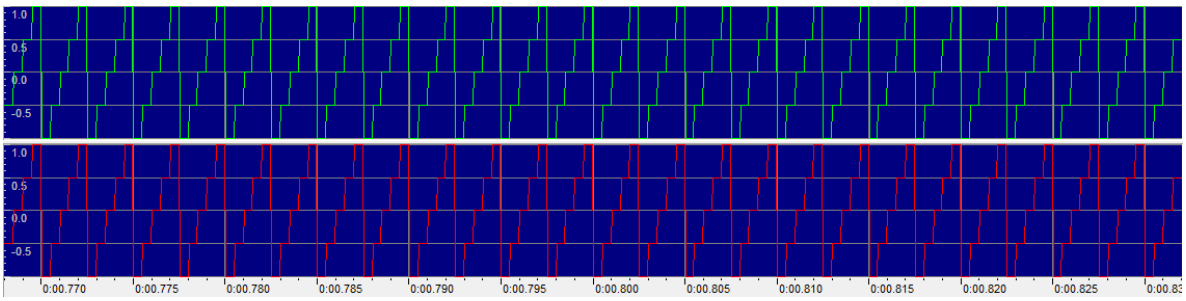


*Figura 10. Señal de entrada 1*

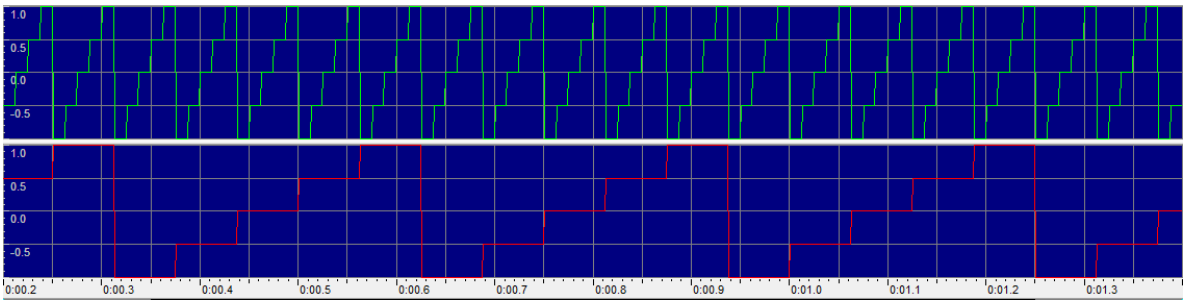


*Figura 11. Señal de entrada 2*

A continuación, en la Figura 12 se muestra el resultado de multiplicar las señales de las Figuras 10 y 11,

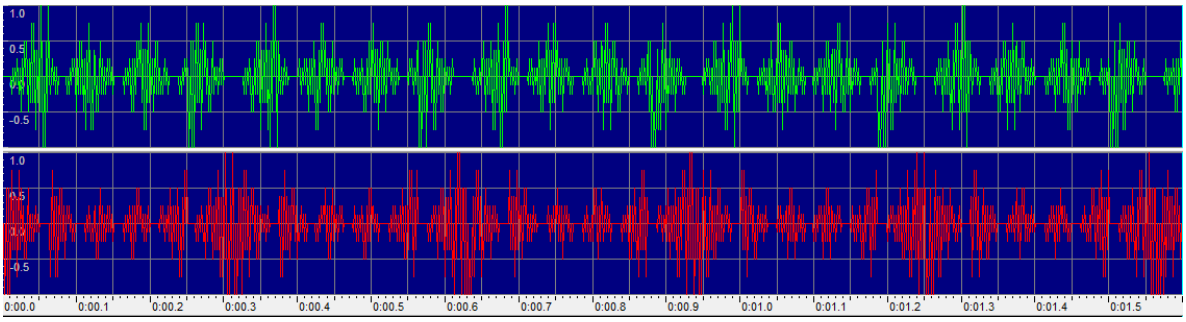


*Figura 12. Señal de salida (sin zoom)*

En la Figura 13, se muestra el mismo resultado de la Figura 12 pero aplicando un zoom para observar un poco mejor la señal.

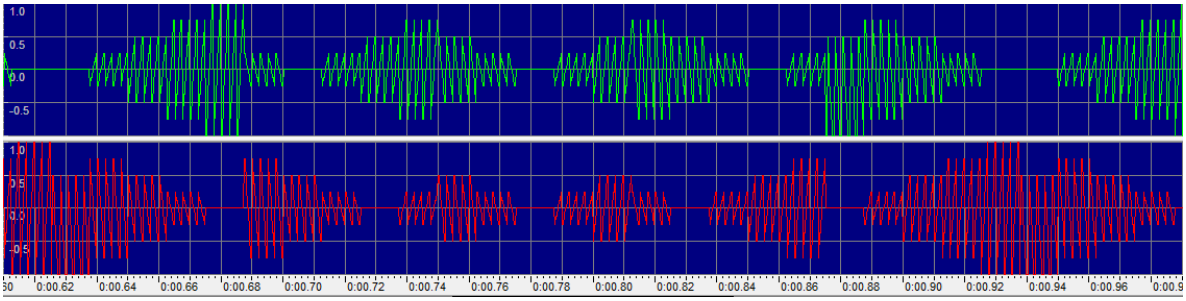


*Figura 13. Señal de salida (con zoom)*

A continuación, procedemos a realizar la multiplicación de 2 archivos mono. El archivo 1 es mostrado en la Figura 14.

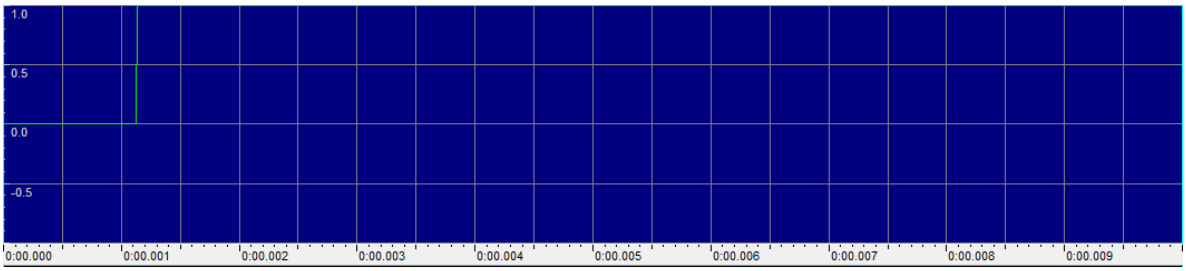


*Figura 14. Señal de entrada 1 (Función escalón)*

En la Figura 15 se muestra la señal de entrada 2, que es un coseno.

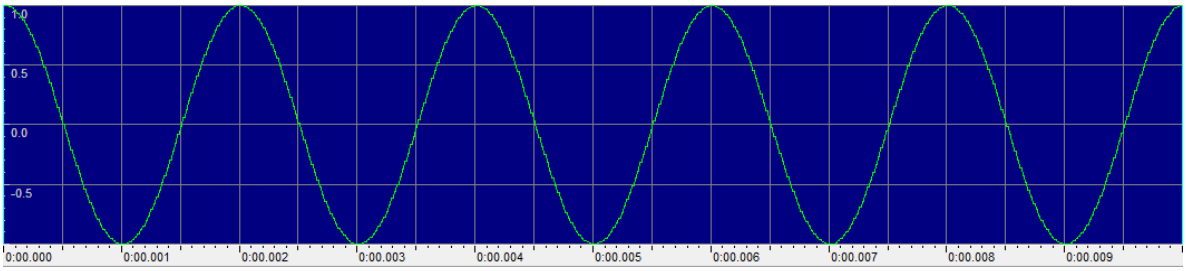


*Figura 15. Señal de entrada 2 (Coseno)*

Podemos hacer un análisis rápido de la multiplicación de estas 2 señales, como es una función escalón, al principio es 0 y después es puros 1, y como sabemos, multiplicar 1 por lo que sea, nos va a dar lo que sea, así que la señal de salida debería ser puro 0 al principio, y cuando comience a ser 1 la función heaviside, la señal de salida será el coseno original. La respuesta de esta multiplicación se muestra en la Figura 16.

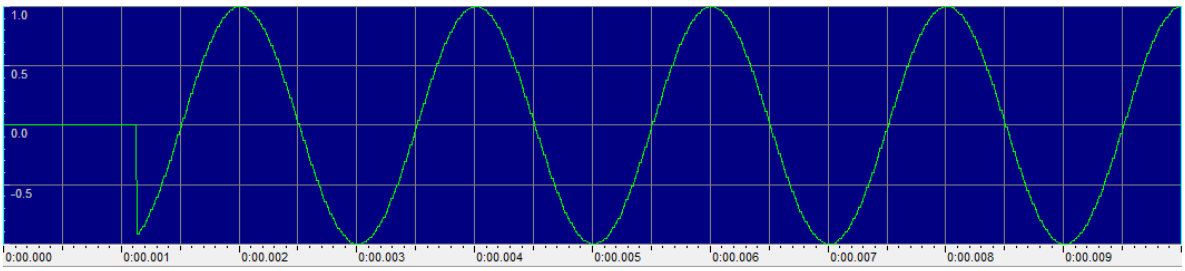


*Figura 16. Señal de salida (Heaviside * Coseno)*

Estas multiplicaciones se hicieron el dominio del tiempo, sin embargo, sabemos que al hacer una multiplicación en el tiempo se realiza una convolución en la frecuencia y viceversa, a continuación, ya que comprobamos que la multiplicación de las señales funciona correctamente, procedemos a realizar el muestreo de 1 señal (es decir, multiplicarla por un tren de impulsos en el dominio del tiempo), para esto, la Figura 17 muestra la señal de datos original que se propone a una frecuencia de muestreo de 44,100 Hz dada por la siguiente función:

$$f\ (x) = \sin(2 * \pi * f * t) + \frac{\sin(2 * \pi * f * t * 3)}{3}, \quad f = 250\ Hz$$

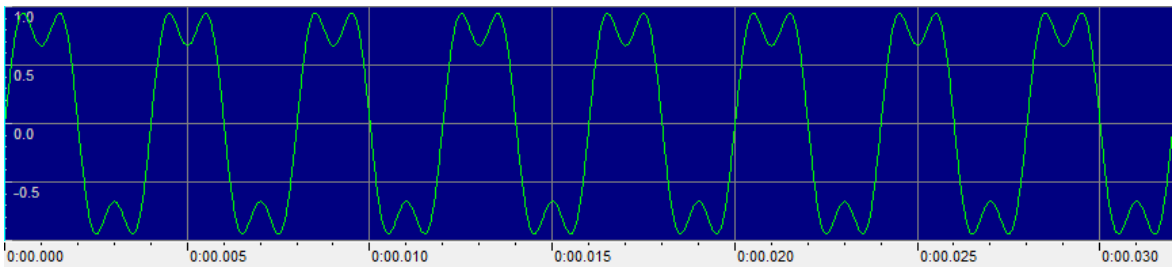


*Figura 17. Señal original*

Ahora, para realizar el muestreo de la señal, creamos una señal que tenga un tren de impulsos a una frecuencia de 2,000 Hz (esto, para evitar el efecto alias), dicho tren de impulsos se muestra en la Figura 18.

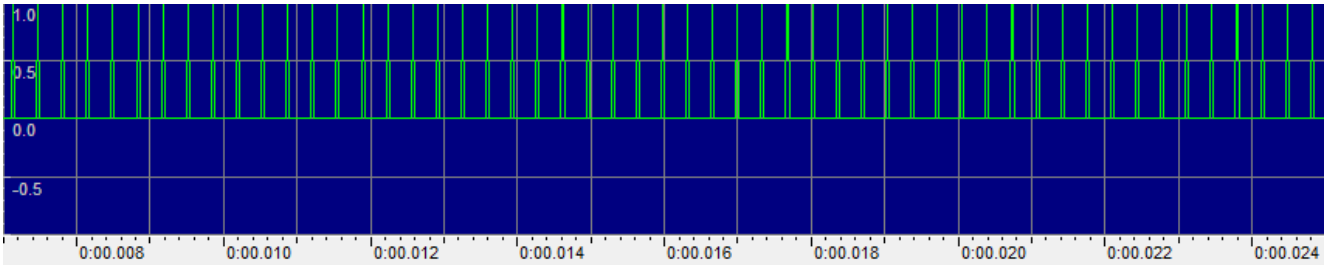


*Figura 18. Tren de impulsos para muestrear señal*

A continuación, ya tenemos la señal original y un tren de impulsos (ambas señales en el tiempo), así que, procedemos a realizar una multiplicación en el tiempo muestra a muestra (como en los ejemplos anteriores) para hacer el muestreo de la señal. El resultado de la multiplicación se muestra en la Figura 19.

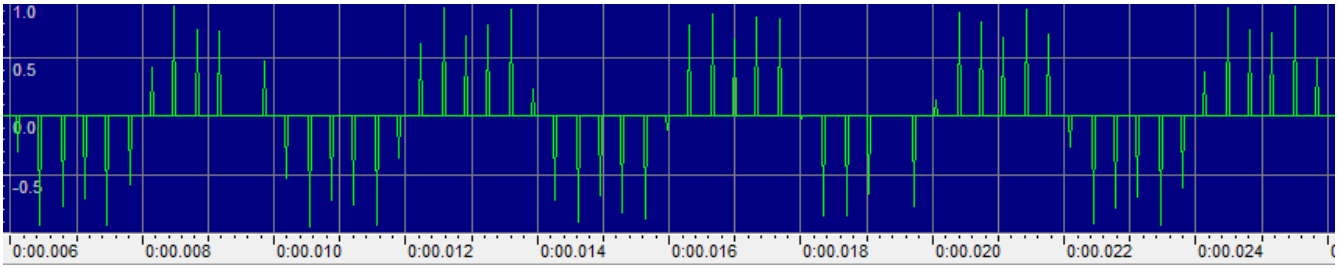


*Figura 19. Señal muestreada*

Al tener la señal muestreada, para recuperar la señal original deberíamos de hacer en el dominio del tiempo una convolución con una señal sampling (descrita a continuación):

$$sinc\ (\pi x) = \frac{\sin(\pi x)}{\pi x}$$

Sin embargo, ya se explicó anteriormente que realizar una multiplicación en el tiempo provoca una convolución en la frecuencia y viceversa, si hacemos una multiplicación en frecuencia, provoca una convolución en el tiempo, y como lo que debemos hacer es una convolución en el dominio del tiempo, lo que debemos hacer para simular este proceso (debido a que no encontré la forma de generar una función sampling en el tiempo), es pasar la señal muestreada al dominio de la frecuencia, generar un filtro pasa – bajas ideal (estéreo, ya que la transformada de la señal muestreada tiene parte real y parte imaginaria), para, posteriormente realizar la multiplicación de ambas señales en frecuencia y esto provocará una convolución en el dominio del tiempo, haciendo que recuperemos la señal original (Figura 17).

A continuación, se diseña el filtro pasa – bajas ideal en el tiempo que se pasará posteriormente al dominio de la frecuencia, el filtro se muestra en la Figura 20.

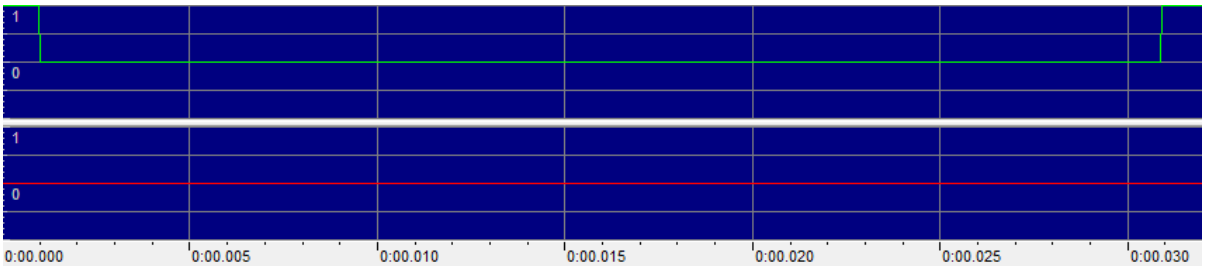


*Figura 20. Filtro pasa - bajas ideal con frecuencia de corte = 1,000 Hz*

A continuación, se realiza la transformada discreta de Fourier de la señal muestreada (en la opción 2) para pasarla al dominio de la frecuencia y hacer la multiplicación. La TDF de la señal muestreada (Figura 19) se muestra en la Figura 21.

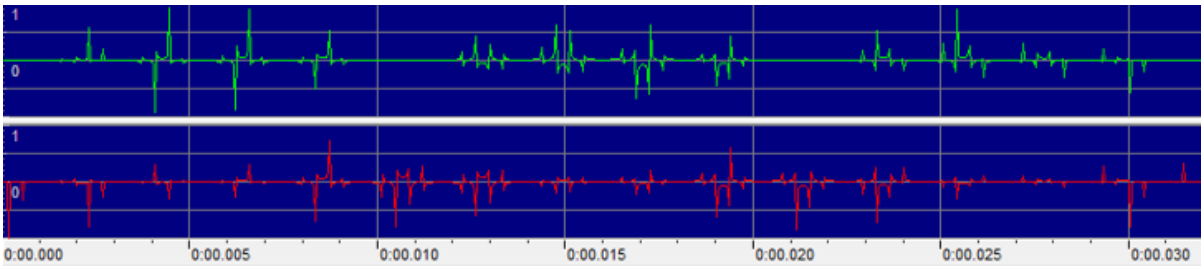


*Figura 21. Transformada de Fourier señal muestreada*

Ya que tenemos la señal muestreada en frecuencia y el filtro ideal que necesitamos a la frecuencia correcta (1,000 Hz), procedemos a realizar la multiplicación de ambas señales (entre Figura 20 y Figura 21, recordemos que se multiplican como si fueran números complejos) para obtener la señal original.

En la Figura 22, se muestra el resultado de multiplicar la TDF de la señal muestreada y el filtro ideal en frecuencia.

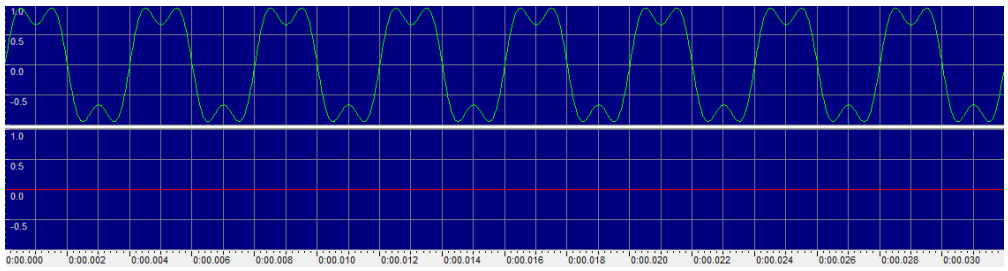


*Figura 22. Reconstrucción de señal original*

Como podemos ver, la Figura 22 es casi la misma señal que la original, salvo por un pequeño detalle, la señal original era mono y esta es una señal estéreo, sin embargo, esto se resolvería con un programa muy básico que lea las muestras y elimine todas las muestras del segundo canal (parte imaginaria) para quedarnos solo con la parte de arriba.

## Discusión:

En la sección anterior (resultados), no se puede observar, sin embargo, algunas señales son de una distinta duración, lo cual podría significar un posible error, pero no, ya que se soluciona de una manera muy fácil y rápida. Después de leer la cabecera de un archivo podemos saber el número de muestras que contiene, y a partir de eso, nos basamos para darle al archivo de salida (resultado de multiplicar las 2 señales de entrada) la duración del archivo más grande.

La multiplicación de 2 señales mono (de 1 solo canal) es demasiado fácil, debido a que es una multiplicación de 2 números reales, sin embargo, aún con su facilidad, es la que tiene más aplicaciones (por lo menos es la que más nos sirve en este curso), debido a que, a partir de esto, podemos aplicar modulación y por supuesto, demodulación para volver a recuperar los datos modulados por la señal moduladora en la señal portadora de los datos.

El multiplicar una señal por un tren de impulsos en el tiempo, es muestrear, es decir, tomar muestras en diversos puntos específicos de la señal original para ser tratados, es decir, hacer periódico el espectro. Posteriormente, como lo que queremos es hacer un filtro (que elimina todas las frecuencias por encima de la frecuencia de corte, en el caso del pasa – bajas), lo pasamos a la frecuencia y hacemos una multiplicación con una frecuencia de corte definida por una función f(x) que sea 1 hasta antes de la frecuencia de corte, y de ahí en adelante, sea 0 (porque es un filtro ideal que elimina todas las frecuencias superiores).

## Conclusiones:

Este programa es demasiado útil, debido a que es el principio de la modulación, hacer una multiplicación en el dominio del tiempo y después aplicar un filtro y otras cosas. Para hacer el proceso inverso (demodulación), hay que aplicar exactamente lo mismo, ya que si haces multiplicación en algún dominio (tiempo o frecuencia), se hace una convolución en el otro. Alguno de las utilidades que puede tener este programa, es el escalado de alguna señal en amplitud (como se realizó una prueba con el coseno, escalándolo a la mitad de su amplitud original, si se multiplica por alguna constante), el escalado nos puede servir por ejemplo, para modificar el volumen de un sonido (como se realizó la disminución del mismo en la práctica 1), y podemos aplicarlo tanto para aumentarlo o disminuirlo, además, se puede procesar señales multiplicando en el dominio de la frecuencia para obtener información deseada (demodulación), muestreo de una señal (multiplicar por un tren de impulsos).

La modulación es muy usada, por ejemplo, en una señal de radio (Amplitud Modulada o Frecuencia Modulada), una señal de televisión (digital), etc., esto nos sirve para poder aprovechar de una mejor manera el canal de comunicación y transferir una mayor cantidad de información a través del mismo, además, aumenta la resistencia que tiene la señal que contiene los datos (portadora) al ruido, generalmente las ondas portadoras son señales sinusoidales modificadas en alguno de sus parámetros (amplitud, frecuencia o fase), teniendo una frecuencia generalmente mayor a la de la señal de entrada debido a que esto permite la multiplexación de distintas señales en la frecuencia haciendo un uso más eficiente en comunicaciones.

La multiplicación de señales es muy utilizada ya que gracias a ella, pudimos reconstruir una señal con un concepto más simple que demodular, solo multiplicando y filtrando en la frecuencia con un filtro ideal a una frecuencia de corte definida.

## Referencias:

**[1]** craig@ccrma.stanford.edu, 'WAVE PCM soundfile format'. [Online]. Disponible en: http://soundfile.sapp.org/doc/WaveFormat/.

**[2]** Sublime HQ, 'Download', [Online]. Disponible en: https://www.sublimetext.com/3.

**[3]** Eduardo Gutiérrez Aldana, 'Software' [Online]. Disponible en: http://148.204.58.221/ealdana/gwave426.exe.

## Código

# Multiplicacion.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include "Cabecera.c"

int main (int argc, char const *argv[])
{
	FILE * archivo_1, *archivo_2, * archivoSalida;
	cabecera cab_1, cab_2;
	int i;
	short real1, imaginario1, real2, imaginario2;
	float max = 32767, real1_1, imaginario1_1, real2_1, imaginario2_1;
	char * salida = (char *) malloc (sizeof (char));
	char * archivo1 = (char *) malloc (sizeof (char));
	char * archivo2 = (char *) malloc (sizeof (char));
	system ("cls");
	if (argc < 4)
	{
		printf("\nError, faltan argumentos.\n\n");
		printf ("Ejemplo: '%s Archivo1.wav Archivo2.wav Salida.wav'\n\n", argv
[0]);
		exit (0);
	}else
	{
		archivo1 = (char *) argv [1];
		archivo2 = (char *) argv [2];
		salida = (char *) argv [3];
	}

	//Abrimos los archivos en modo binario
	archivo_1 = abre_archivo (archivo1, archivo2, salida, 1);
	archivo_2 = abre_archivo (archivo1, archivo2, salida, 2);
	archivoSalida = abre_archivo (archivo1, archivo2, salida, 3);

	//Leemos e imprimimos la cabecera del archivo wav
	leer_cabecera (archivo_1, archivo_2, &cab_1, &cab_2);
	imprimir_cabecera (&cab_1, 1);
	imprimir_cabecera (&cab_2, 2);

	if ((cab_1.ChunkSize) >= (cab_2.ChunkSize))
		copiar_cabecera (archivo_1, archivoSalida, &cab_1);
	else
		copiar_cabecera (archivo_2, archivoSalida, &cab_2);
```

```
	fseek (archivoSalida, 44, SEEK_SET);
	fseek (archivo_1, 44, SEEK_SET);
	fseek (archivo_2, 44, SEEK_SET);

	if ((cab_1.NumChannels == 1) && (cab_2.NumChannels == 1))
		//Si son 2 archivos mono
	{
		if ((cab_1.SubChunk2Size) >= (cab_2.SubChunk2Size))
		{
			for (i = 0; i < (cab_1.SubChunk2Size / 2); i ++)
			{
				fread (&real1, sizeof (short), 1, archivo_1);
				real1_1 = (real1 / max);
				if (i >= (cab_2.SubChunk2Size / 2))
					real2 = 0;
				else
					fread (&real2, sizeof (short), 1,
archivo_2);
				real2_1 = (real2 / max);
				real1 = (real1_1 * real2_1 * max / 2);
				fwrite (&real1, sizeof (short), 1, archivoSalida);
			}
		}else
		{
			for (i = 0; i < (cab_2.SubChunk2Size / 2); i ++)
			{
				fread (&real1, sizeof (short), 1, archivo_2);
				real1_1 = (real1 / max);
				if (i >= (cab_1.SubChunk2Size / 2))
					real2 = 0;
				else
					fread (&real2, sizeof (short), 1,
archivo_1);
				real2_1 = (real2 / max);
				real1 = (real1_1 * real2_1 * max / 2);
				fwrite (&real1, sizeof (short), 1, archivoSalida);
			}
		}
	}else
	{
		if ((cab_1.SubChunk2Size) >= (cab_2.SubChunk2Size))
		{
			for (i = 0; i < (cab_1.SubChunk2Size / 4); i ++)
			{
				//Guardamos la parte real del n?mero complejo
				fread (&real1, sizeof (short), 1, archivo_1);
				real1_1 = (real1 / max);
				if (i >= (cab_2.SubChunk2Size / 4))
				{
					real2 = 0;
					imaginario2 = 0;
				}else
				{
					fread (&real2, sizeof (short), 1,
archivo_2);
					fread (&imaginario2, sizeof (short), 1,
archivo_2);
				}
				real2_1 = (real2 / max);

				//Guardamos la parte imaginaria del n?mero complejo
				fread (&imaginario1, sizeof (short), 1, archivo_1);
				imaginario1_1 = (imaginario1 / max);
				imaginario2_1 = (imaginario2 / max);

				real1 = (((real1_1 * real2_1) - (imaginario1_1 *
imaginario2_1)) * max / 2);
				imaginario1 = (((real1_1 * imaginario2_1) +
(imaginario1_1 * real2_1)) * max / 2);
				fwrite (&real1, sizeof (short), 1, archivoSalida);
```

```
                                        fwrite (&imaginario1, sizeof (short), 1,
archivoSalida);
                              }
                    }else
                    {
                              for (i = 0; i < (cab_2.SubChunk2Size / 4); i ++)
                              {
                                        //Guardamos la parte real del n?mero complejo
                                        fread (&real1, sizeof (short), 1, archivo_2);
                                        real1_1 = (real1 / max);
                                        if (i >= (cab_1.SubChunk2Size / 4))
                                        {
                                                  real2 = 0;
                                                  imaginario2 = 0;
                                        }else
                                        {
                                                  fread (&real2, sizeof (short), 1,
archivo_1);
                                                  fread (&imaginario2, sizeof (short), 1,
archivo_1);
                                        }
                                        real2_1 = (real2 / max);

                                        //Guardamos la parte imaginaria del n?mero complejo
                                        fread (&imaginario1, sizeof (short), 1, archivo_2);
                                        imaginario1_1 = (imaginario1 / max);
                                        imaginario2_1 = (imaginario2 / max);

                                        real1 = (((real1_1 * real2_1) - (imaginario1_1 *
imaginario2_1)) * max / 2);
                                        imaginario1 = (((real1_1 * imaginario2_1) +
(imaginario1_1 * real2_1)) * max / 2);
                                        fwrite (&real1, sizeof (short), 1, archivoSalida);
                                        fwrite (&imaginario1, sizeof (short), 1,
archivoSalida);
                              }
                    }
          }
          printf ("\n\n");
          fclose (archivo_1);
          fclose (archivo_2);
          fclose (archivoSalida);
          printf ("Archivos '%s' y '%s' multiplicados correctamente en '%s'.\n\n", archivo1,
archivo2, salida);
          return 0;
}
```

Multiplicacion.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include "Cabecera.c"

int main (int argc, char const *argv[])
{
          FILE * archivo_1, *archivo_2, * archivoSalida;
          cabecera cab_1, cab_2;
          int i;
          short real1, imaginario1, real2, imaginario2;
          float max = 32767, real1_1, imaginario1_1, real2_1, imaginario2_1;
          char * salida = (char *) malloc (sizeof (char));
          char * archivo1 = (char *) malloc (sizeof (char));
          char * archivo2 = (char *) malloc (sizeof (char));
          system ("cls");
          if (argc < 4)
          {
                    printf("\nError, faltan argumentos.\n\n");
```

```
                    printf ("Ejemplo: '%s Archivo1.wav Archivo2.wav Salida.wav'\n\n", argv
[0]);
                    exit (0);
          }else
          {
                    archivo1 = (char *) argv [1];
                    archivo2 = (char *) argv [2];
                    salida = (char *) argv [3];
          }

          //Abrimos los archivos en modo binario
          archivo_1 = abre_archivo (archivo1, archivo2, salida, 1);
          archivo_2 = abre_archivo (archivo1, archivo2, salida, 2);
          archivoSalida = abre_archivo (archivo1, archivo2, salida, 3);

          //Leemos e imprimimos la cabecera del archivo wav
          leer_cabecera (archivo_1, archivo_2, &cab_1, &cab_2);
          imprimir_cabecera (&cab_1, 1);
          imprimir_cabecera (&cab_2, 2);

          if ((cab_1.ChunkSize) >= (cab_2.ChunkSize))
                    copiar_cabecera (archivo_1, archivoSalida, &cab_1);
          else
                    copiar_cabecera (archivo_2, archivoSalida, &cab_2);

          fseek (archivoSalida, 44, SEEK_SET);
          fseek (archivo_1, 44, SEEK_SET);
          fseek (archivo_2, 44, SEEK_SET);

          if ((cab_1.NumChannels == 1) && (cab_2.NumChannels == 1))
                    //Si son 2 archivos mono
          {
                    if ((cab_1.SubChunk2Size) >= (cab_2.SubChunk2Size))
                    {
                              for (i = 0; i < (cab_1.SubChunk2Size / 2); i ++)
                              {
                                        fread (&real1, sizeof (short), 1, archivo_1);
                                        real1_1 = (real1 / max);
                                        if (i >= (cab_2.SubChunk2Size / 2))
                                                  real2 = 0;
                                        else
                                                  fread (&real2, sizeof (short), 1,
archivo_2);
                                        real2_1 = (real2 / max);
                                        real1 = (real1_1 * real2_1 * max / 2);
                                        fwrite (&real1, sizeof (short), 1, archivoSalida);
                              }
                    }else
                    {
                              for (i = 0; i < (cab_2.SubChunk2Size / 2); i ++)
                              {
                                        fread (&real1, sizeof (short), 1, archivo_2);
                                        real1_1 = (real1 / max);
                                        if (i >= (cab_1.SubChunk2Size / 2))
                                                  real2 = 0;
                                        else
                                                  fread (&real2, sizeof (short), 1,
archivo_1);
                                        real2_1 = (real2 / max);
                                        real1 = (real1_1 * real2_1 * max / 2);
                                        fwrite (&real1, sizeof (short), 1, archivoSalida);
                              }
                    }
          }else
          {
                    if ((cab_1.SubChunk2Size) >= (cab_2.SubChunk2Size))
                    {
                              for (i = 0; i < (cab_1.SubChunk2Size / 4); i ++)
                              {
                                        //Guardamos la parte real del n?mero complejo
                                        fread (&real1, sizeof (short), 1, archivo_1);
```

```
                                        real1_1 = (real1 / max);
                                        if (i >= (cab_2.SubChunk2Size / 4))
                                        {
                                                real2 = 0;
                                                imaginario2 = 0;
                                        }else
                                        {
                                                fread (&real2, sizeof (short), 1,
archivo_2);
                                                fread (&imaginario2, sizeof (short), 1,
archivo_2);
                                        }
                                        real2_1 = (real2 / max);

                                        //Guardamos la parte imaginaria del n?mero complejo
                                        fread (&imaginario1, sizeof (short), 1, archivo_1);
                                        imaginario1_1 = (imaginario1 / max);
                                        imaginario2_1 = (imaginario2 / max);

                                        real1 = (((real1_1 * real2_1) - (imaginario1_1 *
imaginario2_1)) * max / 2);
                                        imaginario1 = (((real1_1 * imaginario2_1) +
(imaginario1_1 * real2_1)) * max / 2);
                                        fwrite (&real1, sizeof (short), 1, archivoSalida);
                                        fwrite (&imaginario1, sizeof (short), 1,
archivoSalida);
                                }
                        }else
                        {
                                for (i = 0; i < (cab_2.SubChunk2Size / 4); i ++)
                                {
                                        //Guardamos la parte real del n?mero complejo
                                        fread (&real1, sizeof (short), 1, archivo_2);
                                        real1_1 = (real1 / max);
                                        if (i >= (cab_1.SubChunk2Size / 4))
                                        {
                                                real2 = 0;
                                                imaginario2 = 0;
                                        }else
                                        {
                                                fread (&real2, sizeof (short), 1,
archivo_1);
                                                fread (&imaginario2, sizeof (short), 1,
archivo_1);
                                        }
                                        real2_1 = (real2 / max);

                                        //Guardamos la parte imaginaria del n?mero complejo
                                        fread (&imaginario1, sizeof (short), 1, archivo_2);
                                        imaginario1_1 = (imaginario1 / max);
                                        imaginario2_1 = (imaginario2 / max);

                                        real1 = (((real1_1 * real2_1) - (imaginario1_1 *
imaginario2_1)) * max / 2);
                                        imaginario1 = (((real1_1 * imaginario2_1) +
(imaginario1_1 * real2_1)) * max / 2);
                                        fwrite (&real1, sizeof (short), 1, archivoSalida);
                                        fwrite (&imaginario1, sizeof (short), 1,
archivoSalida);
                                }
                        }
                }
                printf ("\n\n");
                fclose (archivo_1);
                fclose (archivo_2);
                fclose (archivoSalida);
                printf ("Archivos '%s' y '%s' multiplicados correctamente en '%s'.\n\n", archivo1,
archivo2, salida);
                return 0;
}
```

# Cabecera.h

```
typedef struct CABECERA
{
	char ChunkID[4];	//Contiene las 'RIFF'
	int ChunkSize;	//Contiene el tama□o total sin contar este y el segmento anterior (8 bytes)
	char Format[4];	//Contiene 'WAVE'

	//Aqui comienza el primer subchunk 'fmt'
	char SubChunk1ID[4];	//Contiene 'fmt'
	int SubChunk1Size;	//Contiene el tama□o del resto de el primer subchunk
	short AudioFormat;	//Formato de audio, es es distinto de 1, es forma de compresi□n
	short NumChannels;	//Numero de canales, mono = 1, estereo = 2, etc.
	int SampleRate;	//8000, 44100, etc.
	int ByteRate;	//(SampleRate * Numero canales * Bits per Sample) / 8
	short BlockAlign;	//(Numero canales * Bits per Sample) / 8
	short BitsPerSample;	//8 bits, 16 bits, etc.

	//Aqui comienza el segundo subchunk 'data'
	char SubChunk2ID[4];	//Contiene 'data'
	int SubChunk2Size;	//Numero de bytes en los datos, es decir, bytes despues de este segmento
}cabecera;

FILE * abre_archivo (char * archivo1, char * archivo2, char * salida, int tipo);
void leer_cabecera (FILE * archivo1, FILE * archivo2, cabecera * cab1, cabecera * cab2);
void imprimir_cabecera (cabecera * cab, int tipo);
void copiar_cabecera (FILE * entrada, FILE * salida, cabecera * cab);
```

# Cabecera.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "Cabecera.h"

int i;	//Variable global para manejar ciclos

FILE * abre_archivo (char * archivo1, char * archivo2, char * salida, int tipo)
{
	FILE * archivo_1, * archivo_2, * archivo_salida;
	archivo_1 = fopen (archivo1,"rb");
	if (archivo_1 == NULL)
	{
		printf("Error al abrir archivo '%s'.\n", archivo1);
		exit (0);
	}

	archivo_2 = fopen (archivo2,"rb");
	if (archivo_2 == NULL)
	{
		printf("Error al abrir archivo '%s'.\n", archivo2);
		exit (0);
	}
```

```
        archivo_salida = fopen (salida,"wb");
        if (archivo_salida == NULL)
        {
                printf ("Error al crear el archivo '%s'.\n", salida);
                exit (0);
        }
        if (tipo == 1)
                return archivo_1;
        else if (tipo == 2)
                return archivo_2;
        else
                return archivo_salida;
}

void leer_cabecera (FILE * archivo1, FILE * archivo2, cabecera * cab1, cabecera * cab2)
{
        rewind (archivo1);
        rewind (archivo2);

        //ChunkID
        fread (cab1 -> ChunkID, sizeof (char), 4, archivo1);
        fread (cab2 -> ChunkID, sizeof (char), 4, archivo2);

        //ChunkSize
        fread (&cab1 -> ChunkSize, sizeof (int), 1, archivo1);
        fread (&cab2 -> ChunkSize, sizeof (int), 1, archivo2);

        //Formato "Fmt"
        fread (cab1 -> Format, sizeof (char), 4, archivo1);
        fread (cab2 -> Format, sizeof (char), 4, archivo2);

        //SubChunk1ID Formato de datos "fmt"
        fread (cab1 -> SubChunk1ID, sizeof (char), 4, archivo1);
        fread (cab2 -> SubChunk1ID, sizeof (char), 4, archivo2);

        //SubChunk1Size
        fread (&cab1 -> SubChunk1Size, sizeof (int), 1, archivo1);
        fread (&cab2 -> SubChunk1Size, sizeof (int), 1, archivo2);

        // Formato de audio
        fread (&cab1 -> AudioFormat, sizeof (short), 1, archivo1);
        fread (&cab2 -> AudioFormat, sizeof (short), 1, archivo2);

        //Canales
        fread (&cab1 -> NumChannels, sizeof (short), 1, archivo1);
        fread (&cab2 -> NumChannels, sizeof (short), 1, archivo2);

        //SampleRate
        fread (&cab1 -> SampleRate, sizeof (int), 1, archivo1);
        fread (&cab2 -> SampleRate, sizeof (int), 1, archivo2);

        //ByteRate
        fread (&cab1 -> ByteRate, sizeof (int), 1, archivo1);
        fread (&cab2 -> ByteRate, sizeof (int), 1, archivo2);

        //Block Align
        fread (&cab1 -> BlockAlign, sizeof (short), 1, archivo1);
        fread (&cab2 -> BlockAlign, sizeof (short), 1, archivo2);

        //Bits per Sample
        fread (&cab1 -> BitsPerSample, sizeof (short), 1, archivo1);
        fread (&cab2 -> BitsPerSample, sizeof (short), 1, archivo2);

        //SubChunk2ID
        fread (cab1 -> SubChunk2ID, sizeof (char), 4, archivo1);
        fread (cab2 -> SubChunk2ID, sizeof (char), 4, archivo2);

        //SubChunk2Size
        fread (&cab1 -> SubChunk2Size, sizeof (int), 1, archivo1);
        fread (&cab2 -> SubChunk2Size, sizeof (int), 1, archivo2);
}
```

```
void imprimir_cabecera (cabecera * cab, int tipo)
{
          char * formatoArchivo = (char *) malloc (sizeof (char));
          if (tipo == 1)
                    strcpy (formatoArchivo, "Archivo 1");
          else if (tipo == 2)
                    strcpy (formatoArchivo, "Archivo 2");
          else
                    strcpy (formatoArchivo, "Archivo de Salida");
          printf ("\n\n\n\t\t\t\t%s\n", formatoArchivo);
          printf ("(1-4) Chunk ID: %s\n", cab -> ChunkID);
          printf ("(5-8) ChunkSize:  %u\n", cab -> ChunkSize);
          printf ("(9-12) Format: %s\n", cab -> Format);
          printf ("(13-16) SubChunk 1 ID: %s\n", cab -> SubChunk1ID);
          printf ("(17-20) SubChunk 1 Size: %u\n", cab -> SubChunk1Size);
          if (cab -> AudioFormat == 1)
                              strcpy (formatoArchivo, "PCM");
                    printf ("(21-22) Audio Format: %u, %s\n", cab ->
AudioFormat,formatoArchivo);
          if (cab -> NumChannels == 1)
                              strcpy (formatoArchivo, "Mono");
                    else
                              strcpy (formatoArchivo, "Stereo");
          printf ("(23-24) Number of Channels: %u, Tipo: %s\n", cab ->
NumChannels,formatoArchivo);
          printf ("(25-28) Sample Rate: %u\n", cab -> SampleRate);
          printf ("(29-32) Byte Rate: %u BitRate: %u\n", cab -> ByteRate,cab -> ByteRate *
8);
          printf ("(33-34) Block Align: %u\n", cab -> BlockAlign);
          printf ("(35-36) Bits Per Sample: %u\n", cab -> BitsPerSample);
          printf ("(37-40) SubChunk 2 ID: %s\n", cab -> SubChunk2ID);
          printf ("(41-44) SubChunk 2 Size: %u\n", cab -> SubChunk2Size);
}

void copiar_cabecera (FILE * entrada, FILE * salida, cabecera * cab)
{
          rewind (entrada);
          rewind (salida);

          //ChunkID
          fread (cab -> ChunkID, sizeof (char), 4, entrada);
          fwrite (cab -> ChunkID, sizeof (char), 4, salida);

          //ChunkSize
          fread (&cab -> ChunkSize, sizeof (int), 1, entrada);
          fwrite (&cab -> ChunkSize, sizeof (int), 1, salida);

          //Formato "Fmt"
          fread (cab -> Format, sizeof (char), 4, entrada);
          fwrite (cab -> Format, sizeof (char), 4, salida);

          //SubChunk1ID Formato de datos "fmt"
          fread (cab -> SubChunk1ID, sizeof (char), 4, entrada);
          fwrite (cab -> SubChunk1ID, sizeof (char), 4, salida);

          //SubChunk1Size
          fread (&cab -> SubChunk1Size, sizeof (int), 1, entrada);
          fwrite (&cab -> SubChunk1Size, sizeof (int), 1, salida);

          // Formato de audio
          fread (&cab -> AudioFormat, sizeof (short), 1, entrada);
          fwrite (&cab -> AudioFormat, sizeof (short), 1, salida);

          //Canales
          fread (&cab -> NumChannels, sizeof (short), 1, entrada);
          fwrite (&cab -> NumChannels, sizeof (short), 1, salida);

          //SampleRate
          fread (&cab -> SampleRate, sizeof (int), 1, entrada);
          fwrite (&cab -> SampleRate, sizeof (int), 1, salida);
```

```
        //ByteRate
        fread (&cab -> ByteRate, sizeof (int), 1, entrada);
        fwrite (&cab -> ByteRate, sizeof (int), 1, salida);

        //Block Align
        fread (&cab -> BlockAlign, sizeof (short), 1, entrada);
        fwrite (&cab -> BlockAlign, sizeof (short), 1, salida);

        //Bits per Sample
        fread (&cab -> BitsPerSample, sizeof (short), 1, entrada);
        fwrite (&cab -> BitsPerSample, sizeof (short), 1, salida);

        //SubChunk2ID
        fread (cab -> SubChunk2ID, sizeof (char), 4, entrada);
        fwrite (cab -> SubChunk2ID, sizeof (char), 4, salida);

        //SubChunk2Size
        fread (&cab -> SubChunk2Size, sizeof (int), 1, entrada);
        fwrite (&cab -> SubChunk2Size, sizeof (int), 1, salida);
}
```