PRÁCTICA 6

TRANSFORDA RÁPIDA DE FOURIER

Teoría de comunicaciones y señales

Profesor: Gutiérrez Aldana Eduardo

Alumno:

Amador Nava Miguel Ángel

Grupo: 3CM8

Índice

Introducción

Se implementará la Transformada Rápida de Fourier (FFT), y su inversa (IFFT) comparando el tiempo de ejecución con la Transformada Discreta de Fourier.

Objetivos

Generales: Implementación de la FFT y IFFT

Específicos:

- Análisis del Algoritmo de la FFT
- Comprobar que los resultados de la FFT son igual a los de la DFT

Software y equipo utilizado

- Ubuntu 17.10
- Wine 1.7
- GoldWave 4.26
- Sublime Text 3
- Biblioteca de tiempo.

Marco teórico

Transformada Rápida de Fourier

La FFT (del inglés Fast Fourier Transform) es un algoritmo clásico con un costo de $N \log N$ que permite calcular la transformada de Fourier discreta (DFT). La DFT tiene un costo de N^2 por lo cual este algoritmo es muy importante.

Donde N es el número de muestras de la señal y en la FFT debe ser potencia de 2 (si no lo es se puede rellenar con ceros para que lo sea).

La transformada rápida de Fourier, organizar las muestras de entrada y realiza transformadas más pequeñas en las filas y posteriormente en las columnas.

Las muestras son organizadas en una matriz de L filas, con M columnas.

Donde N=L*M;

L y M pueden ser cualquier número (solo que multiplicados den N)

$$x(l,m) = \begin{bmatrix} x(0) & x(4) & x(8) & x(12) \\ x(1) & x(5) & x(9) & x(13) \\ x(2) & x(6) & x(10) & x(14) \\ x(3) & x(7) & x(11) & x(15) \end{bmatrix}$$

Seguidamente se debe realizar la transformación por filas, haciendo la transformada de M muestras. Esta operación se realiza en función de la siguiente ecuación:

$$F(l,q) = \sum_{m=0}^{M-1} \left[x(l,m) W_M^{mq} \right]$$

$$0 \leq l \leq (L-1)$$

$$0 \leq q \leq (M-1)$$

$$W_M^{mq} = e^{-j\frac{2\pi mq}{M}} = \cos\left(\frac{2\pi mq}{M}\right) - j\sin\left(\frac{2\pi mq}{M}\right)$$

La matriz $F(l,q)$ es un arreglo de valores complejos correspondiente a las transformadas M, puntos sobre cada una de las filas. Seguidamente la matriz $F(l,q)$, se multiplica por los factores de fase W_n esta operación se define en la siguiente ecuación:

$$G(l,q) = W_N^{lq} F(l,q)$$

$$0 \leq l \leq (L-1)$$

$$0 \leq q \leq (M-1)$$

$$W_N^{lq} = e^{-j\frac{2\pi lq}{N}} = \cos\left(\frac{2\pi lq}{N}\right) - j\sin\left(\frac{2\pi lq}{N}\right)$$

Por último se realiza la transformada por columnas L muestras. Esta operación se resume en la siguiente ecuación:

$$X(p,q) = \sum_{l=0}^{L-1} \left[G(l,q) W_L^{lp} \right]$$

$$0 \leq p \leq (L-1)$$

$$0 \leq q \leq (M-1)$$

$$W_L^{lp} = e^{-j\frac{2\pi lp}{L}} = \cos\left(\frac{2\pi lp}{L}\right) - j\sin\left(\frac{2\pi lp}{L}\right)$$

La matriz $X(p,q)$, es la transformada final de las N muestras iniciales. Esta matriz contiene en cada uno de sus términos un número complejo que representa la magnitud y la fase de las componentes espectrales. La matriz $X(p,q)$, se debe leer por filas, como se aprecia en la siguiente matriz:

$$x(l,m) = \begin{bmatrix} X(0) & X(1) & X(2) & X(3) \\ X(4) & X(5) & X(6) & X(7) \\ X(8) & X(9) & X(10) & X(11) \\ X(12) & X(13) & X(14) & X(15) \end{bmatrix}$$

Transformada Rápida de Fourier Inversa

Lo único que difiere de la FFT es que los W_M , W_N y W_L pasan a ser negativos y es opcional dividir en la FFT entre N o en la IFFT

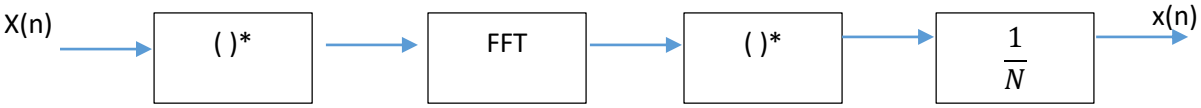

Ilustración 1 FFT inversa usando FFT directa

Preparación del entorno para el desarrollo de la práctica

GoldWave 4.26 fue creado para Windows por lo cual para poder ejecutarlos en Ubuntu 17.10 necesitaremos instalar wine.

Introduciremos los siguientes comandos:

sudo add-apt-repository ppa:ubuntu-wine/ppa

sudo apt-get update

sudo apt-get install wine1.7 winetricks

Una vez instalado wine para ejecutar GoldWave lo haremos de la siguiente manera:

WINEPREFIX~=/.wine32 wine "nombre del ejecutable"

Planteamiento del problema

Se pide implementar un programa el cual calcula la FFT (si bandera =0) o la FFT inversa (si bandera=1); en el caso del cálculo de la FFT se aceptará un archivo monoaural o estéreo como entrada (parte real en el canal izquierdo y parte imaginaria en el derecho); en el caso de la FFTI el archivo de entrada será estéreo. En caso de que el tamaño del archivo de entrada no sea una potencia de dos el programa añadirá ceros al final hasta completar la potencia de dos inmediatamente superior antes de hacer el cálculo, el tamaño del archivo de salida será de tamaño potencia de dos.

El programa se ejecutará de la siguiente manera:

./FFT.exe bandera entrada.wav salida.wav

Análisis Teórico

Cabecera de un archivo wav

Bytes	Campo	Descripción
4	ChunkID	Contiene las letras “RIFF” en ASCII
4	ChunkSize	36 bytes más SubChunk2Size. Este es el tamaño entero del archivo en bytes menos 8
4	Format	Contiene las letras “WAVE”
4	Subchunk1ID	Contiene las letras “fmt”
4	Subchunk1Size	Este es el tamaño del resto del Subchunk
2	AudioFormat	Los valores distintos de 1 indican alguna forma de compresión.
2	NumChannels	Numero d canales: 1= mono, 2=Stereo
4	SampleRate	Número de muestras por segundo
4	ByteRate	SampleRate * NumChannels * BitsPerSample/8
2	BlockAlign	NumChannels * BitsPerSample/8. El número de bytes para una muestra incluyen todos los canales
2	BitsPerSample	Bits por muesta. 8,16,etc.
4	Subchunk2ID	Contiene las letras “data”
4	Subchunk2Size	NumSamples * NumChannels * BitsPerSample/8. Este es el número de bytes en la data
*	Data	Datos del sonido

Tabla 1 Información de la cabecera de un archivo wav

Transformada Rápida de Fourier

Para simplificar los cálculos sacaremos el promedio en la FFT y en la IFFT solo pasaremos a negativo los W. Como en l DFT podemos dividir entre la raíz de N en la FFT y en IFFT.

Análisis del tiempo

Para medir el tiempo de ejecución del programa usaremos una biblioteca la cual su documentación menciona:

En Unix, se dispone de temporizadores ejecutables (en concreto time) que nos proporcionan medidas de los tiempos de ejecución de programas. Estos temporizadores nos proporcionan tres medidas de tiempo:

* real: Tiempo real que se ha tardado desde que se lanzó el programa a ejecutarse hasta que el programa finalizó y proporcionó los resultados.

* user: Tiempo que la CPU se ha dedicado exclusivamente a la computación del programa.

* sys: Tiempo que la CPU se ha dedicado a dar servicio al sistema operativo por necesidades del programa (por ejemplo para llamadas al sistema para efectuar I/O).

El tiempo real también suele recibir el nombre de elapsed time o wall time. Algunos temporizadores también proporcionan el porcentaje de tiempo que la CPU se ha dedicado al programa. Este porcentaje viene dado por la relación entre el tiempo de CPU (user + sys) y el tiempo real, y da una idea de lo cargado que se hallaba el sistema en el momento de la ejecución del programa.

El grave inconveniente de los temporizadores ejecutables es que no son capaces de proporcionar medidas de tiempo de ejecución de segmentos de código. Para ello, hemos de invocar en nuestros propios programas a un conjunto de temporizadores disponibles en la mayor parte de las librerías de C de Unix, que serán los que nos proporcionen medidas sobre los tiempos de ejecución de trozos discretos de código.

Vamos a emplear una función que actúe de temporizador y que nos proporcione los tiempos de CPU (user, sys) y el tiempo real. En concreto, vamos a emplear el procedimiento uswtime listado a continuación.

Este procedimiento en realidad invoca a dos funciones de Unix: getrusage y gettimeofday. La primera de ellas nos proporciona el tiempo de CPU, tanto de usuario como de sistema, mientras que la segunda nos proporciona el tiempo real (wall time). Estas dos funciones son las que disponen de mayor resolución de todos los temporizadores disponibles en Unix.

Modo de Empleo:

La función uswtime se puede emplear para medir los tiempos de ejecución de determinados segmentos de código en nuestros programas. De forma esquemática, el empleo de esta función constaría de los siguientes pasos:

1.- Invocar a uswtime para fijar el instante a partir del cual se va a medir el tiempo.

```
uswtime(&utime0, &stime0, &wtime0);
```

2.- Ejecutar el código cuyo tiempo de ejecución se desea medir.

3.- Invocar a uswtime para establecer el instante en el cual finaliza la medición del tiempo de ejecución.

```
uswtime(&utime1, &stime1, &wtime1);
```

4.- Calcular los tiempos de ejecución como la diferencia entre la primera y segunda invocación a uswtime:

```
real:   wtime1 - wtime0
user:   utime1 - utime0
sys :   stime1 - stime0
```

El porcentaje de tiempo dedicado a la ejecución de ese segmento de código vendría dado por la relación CPU/Wall:

CPU/Wall = (user + sys) / real x 100 %

Compilación donde se usará la biblioteca:

```
gcc –c tiempo.c

gcc nombreArchivo.c  -o nombreEjecutable tiempo.o
```

Resultados

En señales con el número de muestras potencia de dos, los resultados son idénticos a la DFT y la IDFT.

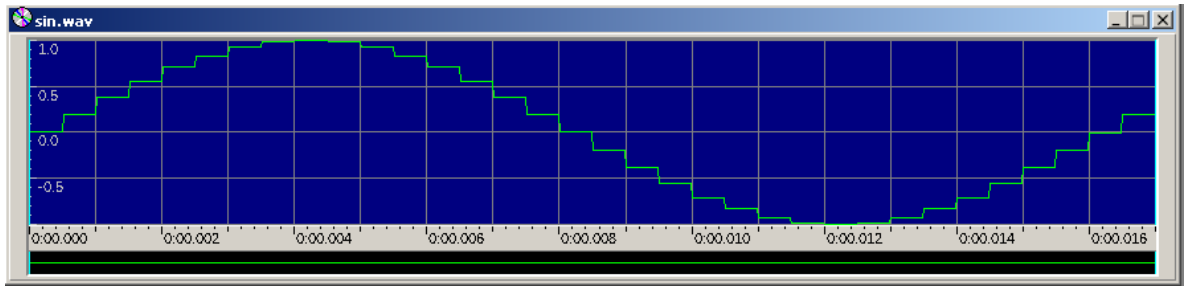

Ilustración 2 Seno de 62.5 Hz con número de muestras distintas de una potencia de 2

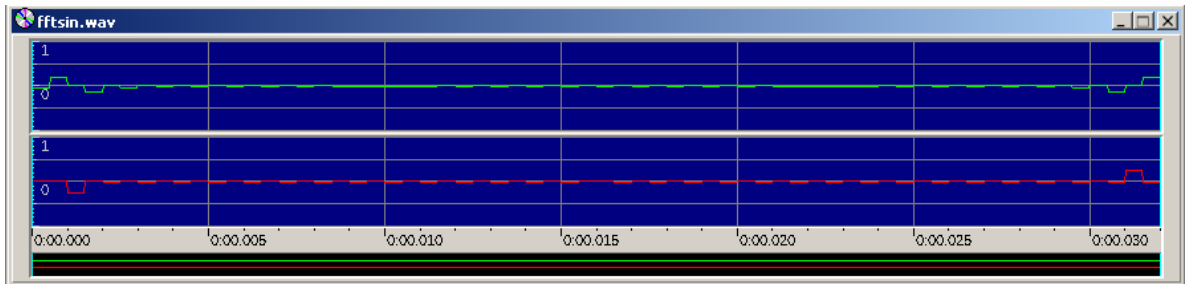

Ilustración 3 FFT del seno de 62.5 Hz con relleno de ceros.

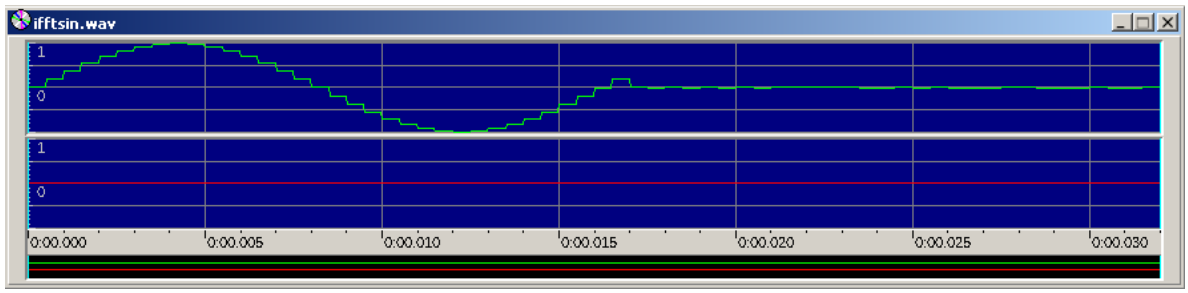

Ilustración 4 IFFT del seno de 62.5 Hz con relleno de ceros

Análisis Práctico

No. De Muestras	DFT (tiempo en segundos)	FFT (tiempo en segundos)
16	0.0002839565	0.0002048016
32	0.0008680820	0.0005338192
64	0.0028629303	0.0009860992
128	0.0074658394	0.0031270981
256	0.0255978107	0.0072100163
512	0.0899648666	0.0242428780
1024	0.3423740864	0.0890450478
2048	1.3495500088	0.3208038807
4096	5.3304669857	1.2427258492
8192	21.0436520576	4.8374810219
16384	89.0457410812	18.9519710541
32768	335.4351091385	75.1833989620
65536	1340.6462161541	299.9703528881

Tabla 2 Resultados de la DFT y FFT en segundos

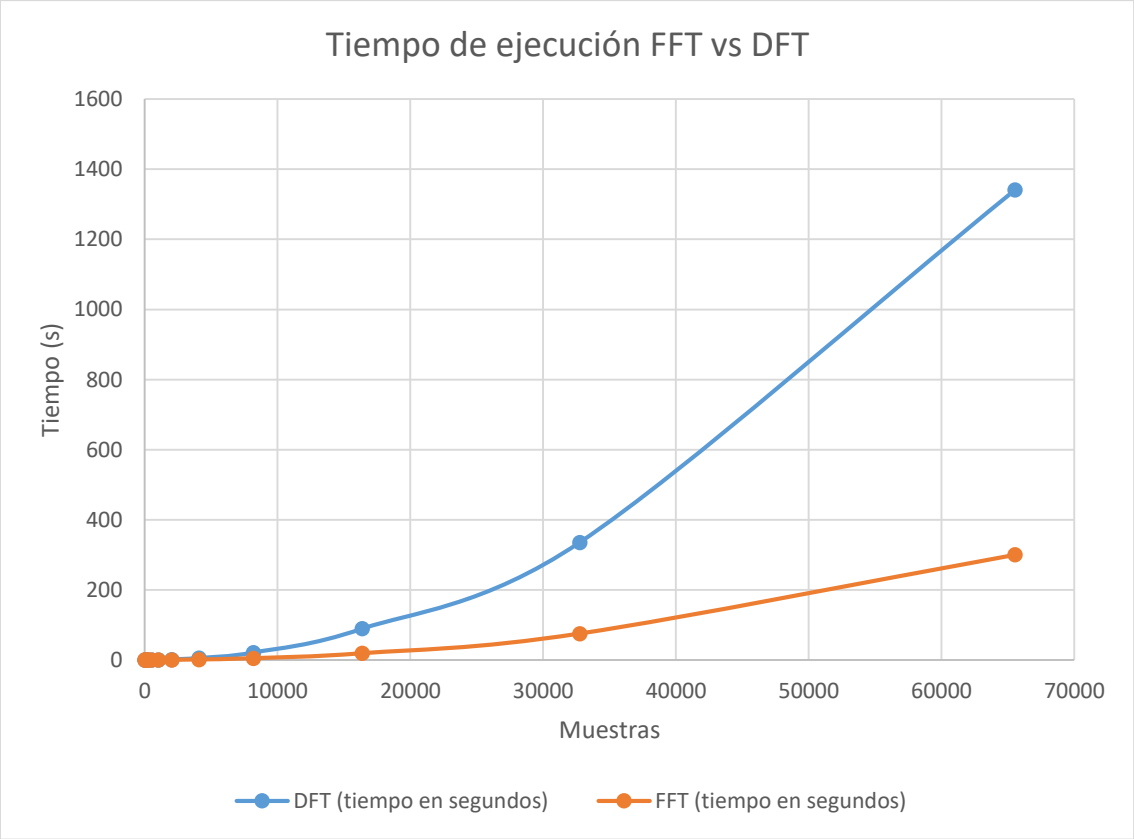

Ilustración 5 Grafica del tiempo de ejecución de la FFT y la DFT en segundos

Conclusiones

Con base en los resultados podemos decir que la FFT es un algoritmo optimizado para la obtención de la transformada discreta de Fourier, la cual necesita menos poder de cómputo y esto se ve reflejado en menor tiempo de ejecución y por consiguiente menor tiempo de espera de los resultados.

Código

wav.h

```
#include <stdio.h>
#include <stdlib.h>

typedef struct CabeceraWAV{
	unsigned char ChunkID[4];
	unsigned int ChunkSize;
	unsigned char Format[4];
	unsigned char Subchunk1_ID[4];
	unsigned char Subchunk1_Size[4];
	unsigned char AudioFormat[2];
	unsigned char NumChannels[2];
	unsigned int SampleRate;
	unsigned int ByteRate;
	unsigned short BlockAlign;
	unsigned short BitsPerSample;
	unsigned char Subchunk2_ID[4];
	unsigned int Subchunk2_Size;//Catidad de bytes de informacion de la
señal
}Cabecera;

void informacionAudio(Cabecera cabeceraAudio){
	printf("%c%c%c%c\n",cabeceraAudio.ChunkID[0],cabeceraAudio.ChunkID[
1],cabeceraAudio.ChunkID[2],cabeceraAudio.ChunkID[3]);
	printf("ChunkSize: %d\n",cabeceraAudio.ChunkSize);
	printf("%c%c%c%c\n",cabeceraAudio.Format[0],cabeceraAudio.Format[1]
,cabeceraAudio.Format[2],cabeceraAudio.Format[3]);
	printf("%c%c%c%c\n",cabeceraAudio.Subchunk1_ID[0],cabeceraAudio.Sub
chunk1_ID[1],cabeceraAudio.Subchunk1_ID[2],cabeceraAudio.Subchunk1_ID[3])
;
	printf("Subchunk1_Size:%d
%d%d%d\n",cabeceraAudio.Subchunk1_Size[0],cabeceraAudio.Subchunk1_Size[1]
,cabeceraAudio.Subchunk1_Size[2],cabeceraAudio.Subchunk1_Size[3]);
	printf("Formato del audio: %d\n",cabeceraAudio.AudioFormat[0] );
	printf("Numero de canales: %d\n",cabeceraAudio.NumChannels[0] );
	printf("Muestras por segundo %d\n",cabeceraAudio.SampleRate);
	printf("Bytes por segundo: %d\n",cabeceraAudio.ByteRate);
	printf("Numero de bytes por muestra %d\n",cabeceraAudio.BlockAlign
);
	printf("Numero de bits por muestra: %d\n",
cabeceraAudio.BitsPerSample);
	printf("%c%c%c%c\n",cabeceraAudio.Subchunk2_ID[0],cabeceraAudio.Sub
chunk2_ID[1],cabeceraAudio.Subchunk2_ID[2],cabeceraAudio.Subchunk2_ID[3])
;
	printf("Numero de bytes de la informacion
%d\n",cabeceraAudio.Subchunk2_Size);
	printf("Numero de muestras
%d\n",cabeceraAudio.Subchunk2_Size/(cabeceraAudio.NumChannels[0]*(cabecer
aAudio.BitsPerSample/8)));
	printf("Bytes de sobra (informaciondesconocida):
%d\n",cabeceraAudio.ChunkSize-36-cabeceraAudio.Subchunk2_Size);
	printf("Tam total del archivo: %d\n",cabeceraAudio.ChunkSize+8);
}
```

```
//Resegra el tamaño completo de el archivo en bytes
unsigned int getFileSize(Cabecera CW){
	return CW.ChunkSize+8;
}

//Regresa el tamaño de las muestras en bytes
char getSampleSize(Cabecera CW){
	return (CW.BitsPerSample/8);
}

//Regresa el numero de canales;
char getCannalNumber(Cabecera CW){
	return CW.NumChannels[0];
}


//Regresa el numero de muestras de audio si es momo o estero regresa la
misma cantidad por lo cual en estereo se debe multiplicar por 2
unsigned int getNumberAudioSamples(Cabecera CW){
	return CW.Subchunk2_Size/(CW.NumChannels[0]*(CW.BitsPerSample/8));
}
//Regresa el numero de muestras de audio totales
unsigned int getTotalNumberSamples(Cabecera CW){
	return CW.Subchunk2_Size/(CW.BitsPerSample/8);
}

//Agrega una cantidad de muestras a ka cabecera sin importar si es audio
o estereo
Cabecera addNumberAudioSamples(Cabecera CW, int numberSamples){
	CW.Subchunk2_Size+=(numberSamples*2*CW.NumChannels[0]);
	CW.ChunkSize+=(numberSamples*2*CW.NumChannels[0]);
	return CW;
}

//Regresa el tamanio total del archivo wav-8
unsigned int getNumberBytesAudioInformation(Cabecera CW){
	return CW.Subchunk2_Size;
}


//Convierte una cabecera de tipo mono a una de tipo estereo
Cabecera setAudioStereo(Cabecera CW){
	CW.ChunkSize+=CW.Subchunk2_Size;
	CW.Subchunk2_Size*=2;
	CW.NumChannels[0]=2;
	CW.ByteRate*=2;
	CW.BlockAlign*=2;
	return CW;
}


//regresa el numero de bytes del pie del archivo wav
unsigned int getNumberBytesFoot(Cabecera CW){
	return CW.ChunkSize-36-CW.Subchunk2_Size;
}
```

```
//Regresa la cabecera del archvo wav
Cabecera getHeader(char *archivoWavEntrada){
	Cabecera CW;
	FILE *archivo=fopen(archivoWavEntrada,"r");
	fread(&CW,44,1,archivo);
	fclose(archivo);
	return CW;
}

//Regresa las muestras de un archivo con tamanio de 8 bits
char* getAudioSamples8bits(char *archivoWavEntrada){
	Cabecera CW;
	FILE *archivo=fopen(archivoWavEntrada,"r");
	fread(&CW,44,1,archivo);
	unsigned int
numeroMuestras=CW.Subchunk2_Size/(CW.NumChannels[0]*(CW.BitsPerSample/8))
;
	char *Samples=malloc(sizeof(char)*numeroMuestras);
	fread(Samples,numeroMuestras,1,archivo);
	fclose(archivo);
	return Samples;
}

//Regresa las muestras de audio de 16 bits mono/Stereo
short* getAudioSamples16bits(char *archivoWavEntrada){
	Cabecera CW;
	FILE *archivo=fopen(archivoWavEntrada,"r");
	fread(&CW,44,1,archivo);
	unsigned int
numeroMuestras=CW.Subchunk2_Size/(CW.NumChannels[0]*(CW.BitsPerSample/8))
;
	short
*Samples=malloc(sizeof(short)*numeroMuestras*CW.NumChannels[0]);
	fread(Samples,numeroMuestras*2*CW.NumChannels[0],1,archivo);
	fclose(archivo);
	return Samples;
}
short* getAudioSamples16bitsStereo(char *archivoWavEntrada){
	Cabecera CW;
	FILE *archivo=fopen(archivoWavEntrada,"r");
	fread(&CW,44,1,archivo);
	unsigned int
numeroMuestras=CW.Subchunk2_Size/(CW.NumChannels[0]*(CW.BitsPerSample/8))
;
	numeroMuestras*=2;
	short *Samples=malloc(sizeof(short)*numeroMuestras);
	fread(Samples,numeroMuestras*2,1,archivo);
	fclose(archivo);
	return Samples;
}

//Regresa las muestras de 16bits del canal1
short* getAudioSamples16bisCanal1(char *archivoWavEntrada){
	Cabecera CW=getHeader(archivoWavEntrada);
	if(getCannalNumber(CW)==2){
		short
*muestrasAudio=getAudioSamples16bitsStereo(archivoWavEntrada);
```

```
            unsigned int tam=getNumberAudioSamples(CW);
            short *canal1=malloc(tam*sizeof(short));
            for(unsigned int i=0;i<tam;i++){
                  canal1[i]=muestrasAudio[2*i];
            }
            return canal1;
      }else{
            printf("El archivo: %s no es estereo\n",archivoWavEntrada);
            return NULL;
      }
}

//Regresa las muestras de 16bits del canal2
short* getAudioSamples16bisCanal2(char *archivoWavEntrada){
      Cabecera CW=getHeader(archivoWavEntrada);
      if(getCannalNumber(CW)==2){
            short
*muestrasAudio=getAudioSamples16bitsStereo(archivoWavEntrada);
            unsigned int tam=getNumberAudioSamples(CW);
            short *canal2=malloc(tam*sizeof(short));
            for(unsigned int i=0;i<tam;i++){
                        canal2[i]=muestrasAudio[2*i+1];
            }
            return canal2;
      }else{
            printf("El archivo: %s no es estereo\n",archivoWavEntrada);
            return NULL;
      }
}



//Regresa las bytes del pie del archivo wav
char* getFileFoot(char *archivoWavEntrada){
      Cabecera CW;
      FILE *archivo=fopen(archivoWavEntrada,"r");
      fread(&CW,44,1,archivo);
      unsigned int bytesSobrantes=CW.ChunkSize-CW.Subchunk2_Size-36;
      char *bytes=malloc(sizeof(char)*CW.Subchunk2_Size);
      fread(bytes,CW.Subchunk2_Size,1,archivo);
      bytes=malloc(sizeof(char)*bytesSobrantes);
      fread(bytes,bytesSobrantes,1,archivo);
      return bytes;
}



short* getSalidaStereo(short *canal1,short *canal2,unsigned int
tamCanal){
      short *salida=malloc(sizeof(short)*tamCanal*2);
      for(unsigned int i=0;i<tamCanal;i++){
            salida[2*i]=canal1[i];
            salida[2*i+1]=canal2[i];
      }
      return salida;
}
```

FFT_IFFT.c

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "wav.h"

#define PI 3.141592653589793

typedef struct NumeroComplejo
{
	double real;
	double imaginario;
}complejo;

void main(int argc, char *argv[]){

	char *archivo1;
	FILE *archivoWavEntrada1=fopen(argv[2],"r");
	int bandera=atoi(argv[1]);
	if(bandera!=0 && bandera != 1){
		printf("Error de bandera\n");
		exit(1);
	}


	if(archivoWavEntrada1==NULL){
		printf("Error en la lectura del archivo de audio\n");
	}else{

		FILE *archivoWavSalida=fopen(argv[3],"w+");

		if(archivoWavSalida!=NULL){

				char *pie=getFileFoot(argv[2]);
				Cabecera CW1=getHeader(argv[2]);
				unsigned int
numMuestrasAudio1=getNumberAudioSamples(CW1);
				unsigned int bytesPie=getNumberBytesFoot(CW1);
				float log=log2(numMuestrasAudio1);
				unsigned short logi=log2(numMuestrasAudio1);
				char portenciaDeDos=0;
				unsigned int numMuestrasAntes=numMuestrasAudio1;

				if(log-logi!=0){
					logi++;
					numMuestrasAudio1=pow(2,logi);

	CW1=addNumberAudioSamples(CW1,numMuestrasAudio1-numMuestrasAntes);
					portenciaDeDos=1;
				}

				unsigned int
bytesAudio=getNumberBytesAudioInformation(CW1);
				if(bandera==0)
					CW1=setAudioStereo(CW1);
```

```
                        fwrite(&CW1,44,1,archivoWavSalida);

                        short *canal1;
                        short *canal2;
                        short *x;

                        if(bandera==1){
                              canal1=getAudioSamples16bisCanal1(argv[2]);
                              canal2=getAudioSamples16bisCanal2(argv[2]);

                              if(portenciaDeDos){
                                    short
*Acanal1=(short*)malloc(sizeof(short)*numMuestrasAudio1);
                                    short
*Acanal2=(short*)malloc(sizeof(short)*numMuestrasAudio1);
                                    for(unsigned int
i=0;i<numMuestrasAntes;i++){
                                          Acanal1[i]=canal1[i];
                                          Acanal2[i]=canal2[i];
                                    }

                                    for(unsigned int
i=numMuestrasAntes;i<numMuestrasAudio1;i++){
                                          Acanal1[i]=0;
                                          Acanal2[i]=0;
                                    }


      canal1=(short*)malloc(sizeof(short)*numMuestrasAudio1);

      canal2=(short*)malloc(sizeof(short)*numMuestrasAudio1);
                                    canal1=Acanal1;
                                    canal2=Acanal2;
                              }
                        }
                        else if(bandera==0){
                              x=getAudioSamples16bits(argv[2]);
                              if(portenciaDeDos){
                                    short
*y=(short*)malloc(numMuestrasAudio1*2*sizeof(short));
                                    for(unsigned int
i=0;i<numMuestrasAntes;i++){
                                          y[i]=x[i];
                                    }
                                    for(unsigned int
i=numMuestrasAntes;i<numMuestrasAudio1;i++){
                                          y[i]=0;
                                    }

      x=(short*)malloc(numMuestrasAudio1*2*sizeof(short));
                                    x=y;
                              }
                        }
                        //FFT e IFFT
                        //L=filas M=columnas
                        unsigned char L=4;
                        unsigned int M=numMuestrasAudio1/L;
```

```
                        complejo **F=(complejo
**)malloc(sizeof(complejo*)*L);
                        complejo **X=(complejo
**)malloc(sizeof(complejo*)*L);
                        for(unsigned int i=0;i<L;i++){
                              F[i]=malloc(sizeof(complejo)*M);
                              X[i]=malloc(sizeof(complejo)*M);
                        }

                        double r1,i1,r2,i2;
                        double C_W;
                        double cons=(2*PI/M);
                        for(unsigned int l=0;l<L;l++){
                              for(unsigned int q=0;q<M;q++){
                                    C_W=cons*q;
                                    if(bandera==1)
                                          C_W=-C_W;
                                    F[l][q].real=0;
                                    F[l][q].imaginario=0;
                                    for(unsigned int m=0;m<M;m++){
                                          if(bandera==1){
                                                r2=cos(C_W*m);
                                                i2=sin(C_W*m);

      F[l][q].real+=canal1[(L*m)+l]*r2+canal2[(L*m)+l]*i2;

      F[l][q].imaginario+=canal2[(L*m)+l]*r2-canal1[(L*m)+l]*i2;
                                          }else if(bandera==0){

      F[l][q].real+=x[(L*m)+l]*cos(C_W*m);

      F[l][q].imaginario+=x[(L*m)+l]*sin(C_W*m);
                                          }
                                    }
                                    if(bandera==0)
                                          F[l][q].imaginario*=-1;
                              }
                        }
                        cons=2*PI/numMuestrasAudio1;

                        for(unsigned int l=0;l<L;l++){
                              C_W=cons*l;
                              if(bandera==1)
                                    C_W=-C_W;
                              for(unsigned int q=0;q<M;q++){
                                    r1=F[l][q].real;
                                    i1=F[l][q].imaginario;
                                    r2=cos(C_W*q);
                                    i2=sin(C_W*q);
                                    F[l][q].real=(r1*r2+i1*i2);
                                    F[l][q].imaginario=(i1*r2-r1*i2);
                              }
                        }

                        cons=(2*PI/L);
                        for(unsigned int p=0;p<L;p++){
```

```
                                    for(unsigned int q=0;q<M;q++){
                                          C_W=cons*p;
                                          if(bandera==1)
                                                C_W=-C_W;
                                          X[p][q].real=0;
                                          X[p][q].imaginario=0;
                                          for(unsigned int l=0;l<L;l++){
                                                r1=F[l][q].real;
                                                i1=F[l][q].imaginario;
                                                r2=cos(C_W*l);
                                                i2=sin(C_W*l);
                                                X[p][q].real+=(r1*r2+i1*i2);
                                                X[p][q].imaginario+=(i1*r2-
r1*i2);
                                          }
                                          if(bandera==0){

      X[p][q].real/=(numMuestrasAudio1);//

      X[p][q].imaginario/=(numMuestrasAudio1);//
                                          }
                                    }
                              }

                              //FIN FFT e IFFT
                              short
*muestras1=malloc(sizeof(short)*numMuestrasAudio1*2);
                              int indice,y;
                              for(unsigned int l=0;l<L;l++){
                                    for(unsigned int m=1;m<=M;m++){
                                          indice=2*(l*M+m);
                                          y=m-1;
                                          muestras1[indice-
2]=(short)X[l][y].real;
                                          muestras1[indice-
1]=(short)X[l][y].imaginario;
                                    }
                              }
                              if(bandera==0)
                                    bytesAudio*=2;
                              fwrite(muestras1,bytesAudio,1,archivoWavSalida);
                              fwrite(pie,bytesPie,1,archivoWavSalida);
                              fclose(archivoWavSalida);
                              fclose(archivoWavEntrada1);

            }else{
                              printf("No se pudo crear el archivo de salida\n");
            }
      }
}
```

Biblioteca del tiempo

```
//*************************************************************
//TIEMPO.C
//*************************************************************
//*************************************************************
//M. EN C. EDGARDO ADRIÁN FRANCO MARTÍNEZ
//Curso: Análisis de algoritmos
//(C) Enero 2013
//ESCOM-IPN
//Ejemplo de medición de tiempo en C y recepción de parametros en C bajo
UNIX
//Compilación de la libreria: "gcc -c tiempo.c " (Generación del código
objeto)
//*************************************************************
//*************************************************************
//Librerias incluidas
//*************************************************************
#include <sys/resource.h>
#include <sys/time.h>
#include "tiempo.h"

//*************************************************************
//uswtime (Definición)
//*************************************************************
//Descripción: Función que almacena en las variables referenciadas
//el tiempo de CPU, de E/S y Total actual del proceso actual.
//
//Recibe: Variables de tipo doble para almacenar los tiempos actuales
//Devuelve:
//*************************************************************#inclu
de <stdio.h>
void uswtime(double *usertime, double *systime, double *walltime)
{
      double mega = 1.0e-6;
      struct rusage buffer;
      struct timeval tp;
      struct timezone tzp;
      getrusage(RUSAGE_SELF, &buffer);
      gettimeofday(&tp, &tzp);
      *usertime = (double) buffer.ru_utime.tv_sec +1.0e-6 *
buffer.ru_utime.tv_usec;
      *systime  = (double) buffer.ru_stime.tv_sec +1.0e-6 *
buffer.ru_stime.tv_usec;
      *walltime = (double) tp.tv_sec + 1.0e-6 * tp.tv_usec;
}
```

```
//*****************************************************************
//TIEMPO.H
//*****************************************************************
//*****************************************************************
//M. EN C. EDGARDO ADRIÁN FRANCO MARTÍNEZ
//Curso: Análisis de algoritmos
//(C) Enero 2013
//ESCOM-IPN
//Ejemplo de medición de tiempo en C y recepción de parametros en C bajo
UNIX
//Compilación de la libreria: "gcc -c tiempo.c " (Generación del código
objeto)
//*****************************************************************


//*****************************************************************
//uswtime (Declaración)
//*****************************************************************
//Descripción: Función que almacena en las variables referenciadas
//el tiempo de CPU, de E/S y Total actual del proceso actual.
//
//Recibe: Variables de tipo doble para almacenar los tiempos actuales
//Devuelve:
//*****************************************************************
void uswtime(double *usertime, double *systime, double *walltime);
```