

Reporte 05A.- Transformada Rápida de Fourier

Alumno: Monroy Martos Elioth

Boleta: 2016630258

Profesor: Gutierrez Aldana Eduardo

Materia: Teoría de Comunicaciones y Señales

Grupo: 3CM6

10 de diciembre de 2017

Índice

1. Introducción	1
2. Código	4
3. Pruebas	14
4. Conclusiones	22
Referencias	22

1. Introducción

Existen diversas formas de calcular la TDF, una de ellas es usando la Transformada Rápida de Fourier (Fast Fourier Transform, FFT). Esta produce la misma salida que la TDF pero en un tiempo significativamente menor. La principal diferencia entre la TDF y la FFT es el rendimiento (tiempo de ejecución) que tiene cada uno de los algoritmos. Siendo la FFT miles de veces más veloz que la TDF, por lo cual, es común que en el tratamiento de señales digitales el algoritmo usado por defecto para cualquier tipo de análisis sea la FFT.

La complejidad de la TDF es de: $O(N^2)$ mientras que el de la FFT es: $O(N \log(N))$, esto hace referencia al número de operaciones que necesita cada algoritmo.

La FFT trabaja de forma que descompone una señal en el dominio del tiempo de N puntos a N señales en el dominio del tiempo de un solo punto. Posteriormente calcula el espectro en frecuencia de las N señales y finalmente estos espectros en frecuencia son sintetizados dentro de uno solo obteniendo así la salida. En la figura 1 se muestra un ejemplo de la descomposición de una señal:

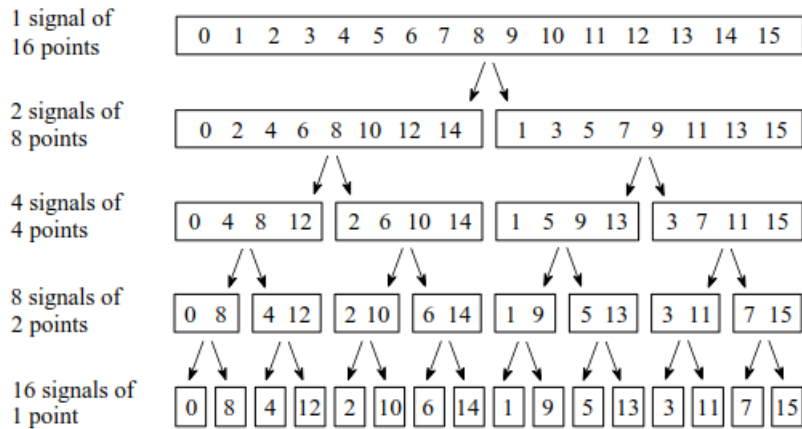


Figura 1: Descomposición de una señal de 16 puntos en 16 señales de 1 punto

El número de fases de esta descomposición esta dado por $\log_2(N)$. Posterior a esta descomposición, es necesario realizar un reordenamiento de las muestras, esto es logrado mediante la inversión de bits. El cual se puede observar en la siguiente figura:


Sample numbers in normal order			Sample numbers after bit reversal	
<i>Decimal</i>	<i>Binary</i>		<i>Decimal</i>	<i>Binary</i>
0	0000		0	0000
1	0001		8	1000
2	0010		4	0100
3	0011		12	1100
4	0100		2	0010
5	0101		10	1010
6	0110		6	0100
7	0111		14	1110
8	1000		1	0001
9	1001		9	1001
10	1010		5	0101
11	1011		13	1101
12	1100		3	0011
13	1101		11	1011
14	1110		7	0111
15	1111		15	1111

Figura 2: Inversión de Bits

Después del reordenamiento es necesario encontrar el espectro en frecuencia de cada uno de los puntos, y después realizar la síntesis de los mismos fase por fase, al final se obtiene el resultado de la FFT. Para esto se usa la siguiente operación, la cual es conocida coloquialmente como mariposa.

2. Código

Para la realización de esta práctica, se hizo uso del programa de multiplicación desarrollado en la práctica anterior (04A), y se modificó además el programa de la TDF para que esta imprimiera el tiempo de ejecución del algoritmo, los resultados de esto se pueden apreciar en la sección de pruebas. Para esta práctica se desarrollaron dos programas, uno que calcula la Transformada Rápida de Fourier (FFT) y otro que calcula la Transformada Rápida de Fourier Inversa (FFTI). De lo cuales se anexa el código elaborado a continuación:

funciones.h:

```
1 #ifndef __FUNCIONES_H__
2 #define __FUNCIONES_H__
3 //Librerías de C
4 #include <stdio.h>
5 #include <stdlib.h>
6 #include <math.h>
7 #include <string.h>
8 //Librería que contiene los máximos y mínimos de los
9 //diferentes tipos de datos en c
10 #include <limits.h>
11 //Librería para conocer tiempo de ejecución
12 #include <time.h>
13 //Metodos
14 void leerCabeceras(char**);
15 void escribirArchivo(short*, short*);
16 void leerMuestras(short*);
17 void leerMuestras2Canales(short*, short*);
18 void convertirFloat(short*, float*, float*);
19 void convertirShort(short*, short*, float*, float*);
20 //Cabeceras
21 int chunkid;
22 int chunksize;
23 int format;
24 int subchunk1id;
25 int subchunk1size;
26 short audioformat;
27 short numchannels;
28 int samplerate;
29 int byterate;
30 short blockalign;
31 short bitspersample;
32 int subchunk2id;
```

```

32  int subchunk2size;
33  //Archivo
34  FILE* entrada;
35  FILE* salida;
36  //Variables para muestras
37  short muestra;
38  int total_muestras_originales;
39  int total_muestras;
40  short headers[37];
41  //Métodos TDF
42  #define PI acos(-1.0)//Defino la constante PI
43  void calcularFFT(short*);
44  void calcularFFTI(short*,short*);
45  int calcularNuevoNumeroMuestras(int);
46  void intercambiar(float**,int,int);
47  //Inversión de bits
48  #define SWAP(x,y) do {typeof(x) _x = x;typeof(y) _y = y;x = _y
    ;y = _x;} while(0)
49  //Variables para obtener tiempo de ejecución
50  clock_t inicio , final;
51  double total;
52  #endif

```

fft.c:

```

1  #include"funciones.h"
2  int main(int argc , char *argv []) {
3      //Leo las cabeceras
4      leerCabeceras(argv);
5      //Defino variables
6      total_muestras_originales=subchunk2size/blockalign;
7      printf("Total muestras originales:%d\n",
        total_muestras_originales);
8      //Necesitamos que el total de muestras sea una potencia de 2
9      total_muestras=calcularNuevoNumeroMuestras(
        total_muestras_originales);
10     printf("Nuevo total de muestras:%d\n", total_muestras);
11     short *muestras=(short *)malloc(total_muestras * sizeof(short)
        );
12     //Leo las muestras
13     leerMuestras(muestras);
14     //Calculo la FFT
15     calcularFFT(muestras);
16 }
17 void leerCabeceras(char ** argv){
18     entrada = fopen(argv[1] , "rb");

```

```

19 salida=fopen(argv[2], "wb");
20 if(!entrada){
21     perror("\nFile opening failed");
22     exit(0);
23 }
24 fread(&chunkid, sizeof(int), 1, entrada);
25 fread(&chunksize, sizeof(int), 1, entrada);
26 fread(&format, sizeof(int), 1, entrada);
27 fread(&subchunklid, sizeof(int), 1, entrada);
28 fread(&subchunklsize, sizeof(int), 1, entrada);
29 fread(&audioformat, sizeof(short), 1, entrada);
30 fread(&numchannels, sizeof(short), 1, entrada);
31 fread(&samplerate, sizeof(int), 1, entrada);
32 fread(&byterate, sizeof(int), 1, entrada);
33 fread(&blockalign, sizeof(short), 1, entrada);
34 fread(&bitspersample, sizeof(short), 1, entrada);
35 fread(&subchunk2id, sizeof(int), 1, entrada);
36 fread(&subchunk2size, sizeof(int), 1, entrada);
37 }
38 void leerMuestras(short *muestras){
39     int i=0;
40     while (feof(entrada) == 0){
41         if(i<total_muestras_originales){
42             fread(&muestra, sizeof(short), 1, entrada);
43             muestras[i]=muestra;
44             i++;
45         }else{
46             fread(&headers, sizeof(short), 37, entrada);
47             break;
48         }
49     }
50     //Ajuste por si las muestras originales no fueron potencia de
    dos
51     if(total_muestras_originales<total_muestras){
52         for (i = total_muestras_originales; i < total_muestras; i++)
53         {
54             muestras[i]=0;
55         }
56     }
57     fclose(entrada);
58 }
59 void escribirArchivo(short* muestrasRe, short* muestrasIm){
60     //Escribo el archivo
61     fwrite(&chunkid, sizeof(int), 1, salida);
62     fwrite(&chunksize, sizeof(int), 1, salida);

```



```

62 fwrite(&format , sizeof(int) ,1, salida);
63 fwrite(&subchunk1id , sizeof(int) ,1, salida);
64 fwrite(&subchunk1size , sizeof(int) ,1, salida);
65 fwrite(&audioformat , sizeof(short) ,1, salida);
66 fwrite(&numchannels , sizeof(short) ,1, salida);
67 fwrite(&samplerate , sizeof(int) ,1, salida);
68 fwrite(&byterate , sizeof(int) ,1, salida);
69 fwrite(&blockalign , sizeof(short) ,1, salida);
70 fwrite(&bitspersample , sizeof(short) ,1, salida);
71 fwrite(&subchunk2id , sizeof(int) ,1, salida);
72 fwrite(&subchunk2size , sizeof(int) ,1, salida);
73 //Ahora escribo las muestras
74 int i=0;
75 for (i=0;i<total_muestras;i++){
76     fwrite(&muestrasRe[i] , sizeof(short) ,1, salida);
77     fwrite(&muestrasIm[i] , sizeof(short) ,1, salida);
78 }
79 //Y por último los headers de goldwave
80 for (i=0;i<37;i++){
81     fwrite(&headers[i] , sizeof(short) ,1, salida);
82 }
83 fclose(salida);
84 }
85 void calcularFFT(short *muestras){
86     //Aquí va el algoritmo para la FFT
87     float *Xre=(float *) malloc(total_muestras * sizeof(float));
88     float *Xim=(float *) malloc(total_muestras * sizeof(float));
89     int i;
90     //Convierto las muestras de short a float
91     convertirFloat(muestras , Xre , Xim);
92     //Iniciar reloj
93     inicio = clock();
94     //FFT
95     int j, k, fk, m, n, ce, c, w;
96     float arg, seno, coseno, tempr, tempi;
97     //Bit reversal
98     m=log((float)total_muestras)/log(2.0);
99     j=w=0;
100     for (i = 0; i < total_muestras; i++){
101         if (j>i){
102             SWAP(Xre[i] , Xre[j]);
103             SWAP(Xim[i] , Xim[j]);
104         }
105         w=total_muestras/2;
106         while (w>=2 && j>=w){

```

```

107     j-=w;
108     w>=>1;
109 }
110 j+=w;
111 }
112 ce=m;
113 c=0;
114 //Mariposas
115 for (i = 0; i < m; i++) {
116     for (j = 0; j < (int)pow(2,ce-1); j++){
117         n = (int)pow(2,i);
118         for (k = 0; k < n; k++){
119             fk=k*(int)pow(2,ce-1);
120             coseno=cos((-1)*2*PI*fk/total_muestras);
121             seno=sin((-1)*2*PI*fk/total_muestras);
122             tempr=Xre[c+n];
123             Xre[c+n]=(Xre[c+n]*coseno) - (Xim[c+n]*seno);
124             Xim[c+n]=(Xim[c+n]*coseno) + (tempr*seno);
125             tempr=(Xre[c]+Xre[c+n])/2;
126             tempi=(Xim[c]+Xim[c+n])/2;
127             Xre[c+n]=(Xre[c]-Xre[c+n])/2;
128             Xim[c+n]=(Xim[c]-Xim[c+n])/2;
129             Xre[c]=tempr;
130             Xim[c]=tempi;
131             c++;
132         }
133         c += n;
134     }
135     c = 0;
136     ce -= 1;
137 }
138 short *Reales=(short *)malloc(total_muestras * sizeof(short));
139 short *Imaginas=(short *)malloc(total_muestras * sizeof(
    short));
140 //Obtener tiempo e imprimir
141 final = clock();
142 total = (double)(final - inicio) / CLOCKS_PER_SEC;
143 printf("Tiempo de ejecucion: %f\n", total);
144 //Regreso las muestras calculadas a short
145 convertirShort(Reales, Imaginas, Xre, Xim);
146 //La salida ahora sera un archivo tipo estereo (2 canales)
147 //Por lo cual hay que cambiar el numero de canales del archivo
148 //y todas las demas cabeceras que dependan de esta
149 chunksize-=subchunk2size;
150 numchannels*=2;

```

```

151 byterate*=numchannels;
152 blockalign*=numchannels;
153 subchunk2size=total_muestras*blockalign;
154 chunksize+=subchunk2size;
155 escribirArchivo(Reales,Imaginas);
156 }
157 int calcularNuevoNumeroMuestras(int total){
158     if ((total & (total-1))==0){
159         puts("Ya es potencia de 2");
160     }else{
161         puts("No es potencia de 2");
162         int i;
163         i=(int) ceil((float)log(total_muestras_originales)/(float)log
164         (2));
165         printf("i:%d\n", i);
166         total=pow(2,i);
167     }
168     return total;
169 }
170 void convertirFloat(short *muestras, float *Xre, float *Xim){
171     int i;
172     for (i = 0; i < total_muestras; i++){
173         Xre[i]=(float)muestras[i]/(float)(SHRT_MAX);
174         Xim[i]=0.0;
175     }
176 }
177 void convertirShort(short *Reales, short *Imaginas, float *
178     Xre, float *Xim){
179     int i;
180     for (i = 0; i < total_muestras; i++){
181         Reales[i]=Xre[i]*(SHRT_MAX);
182         Imaginas[i]=Xim[i]*(SHRT_MAX);
183     }
184 }

```

fft.c:

```
1 #include "funciones.h"
2 int main(int argc, char *argv[]) {
3     //Leo las cabeceras
4     leerCabeceras(argv);
5     //Defino variables
6     total_muestras=subchunk2size/blockalign;
7     printf("Total muestras %d\n",total_muestras);
8     short *muestrasRe=(short *)malloc(total_muestras * sizeof(
9         short));
10    short *muestrasIm=(short *)malloc(total_muestras * sizeof(
11        short));
12    //Leo las muestras
13    leerMuestras2Canales(muestrasRe, muestrasIm);
14    //Calculo la TDF
15    calcularFFTI(muestrasRe, muestrasIm);
16 }
17 void leerCabeceras(char ** argv){
18     entrada = fopen(argv[1], "rb");
19     salida=fopen(argv[2], "wb");
20     if(!entrada){
21         perror("\nFile opening failed");
22         exit(0);
23     }
24     fread(&chunkid, sizeof(int), 1, entrada);
25     fread(&chunksize, sizeof(int), 1, entrada);
26     fread(&format, sizeof(int), 1, entrada);
27     fread(&subchunk1id, sizeof(int), 1, entrada);
28     fread(&subchunk1size, sizeof(int), 1, entrada);
29     fread(&audioformat, sizeof(short), 1, entrada);
30     fread(&numchannels, sizeof(short), 1, entrada);
31     fread(&samplerate, sizeof(int), 1, entrada);
32     fread(&byterate, sizeof(int), 1, entrada);
33     fread(&blockalign, sizeof(short), 1, entrada);
34     fread(&bitspersample, sizeof(short), 1, entrada);
35     fread(&subchunk2id, sizeof(int), 1, entrada);
36     fread(&subchunk2size, sizeof(int), 1, entrada);
37 }
38 void leerMuestras2Canales(short *muestrasRe, short* muestrasIm){
39     int i=0;
40     while (feof(entrada) == 0){
41         if(i<total_muestras){
42             fread(&muestrasRe[i], sizeof(short), 1, entrada);
43             fread(&muestrasIm[i], sizeof(short), 1, entrada);
44             i++;
45         }
46     }
47 }
```

```

43     } else {
44         fread(&headers, sizeof(short), 37, entrada);
45         break;
46     }
47 }
48 }
49 void escribirArchivo(short* muestrasRe, short* muestrasIm){
50     //Escribo el archivo
51     fwrite(&chunkid, sizeof(int), 1, salida);
52     fwrite(&chunksize, sizeof(int), 1, salida);
53     fwrite(&format, sizeof(int), 1, salida);
54     fwrite(&subchunklid, sizeof(int), 1, salida);
55     fwrite(&subchunklsize, sizeof(int), 1, salida);
56     fwrite(&audioformat, sizeof(short), 1, salida);
57     fwrite(&numchannels, sizeof(short), 1, salida);
58     fwrite(&samplerate, sizeof(int), 1, salida);
59     fwrite(&byterate, sizeof(int), 1, salida);
60     fwrite(&blockalign, sizeof(short), 1, salida);
61     fwrite(&bitspersample, sizeof(short), 1, salida);
62     fwrite(&subchunk2id, sizeof(int), 1, salida);
63     fwrite(&subchunk2size, sizeof(int), 1, salida);
64     //Ahora escribo las muestras
65     int i=0;
66     for (i=0; i<total_muestras; i++){
67         fwrite(&muestrasRe[i], sizeof(short), 1, salida);
68         fwrite(&muestrasIm[i], sizeof(short), 1, salida);
69     }
70     //Y por último los headers de goldwave
71     for (i=0; i<37; i++){
72         fwrite(&headers[i], sizeof(short), 1, salida);
73     }
74 }
75 void calcularFFTI(short *Re, short *Im){
76     //Aquí va el algoritmo para la FFT
77     int i;
78     float *Xre=(float *) malloc(total_muestras * sizeof(float));
79     float *Xim=(float *) malloc(total_muestras * sizeof(float));
80     for (i = 0; i < total_muestras; i++){
81         Xre[i]=(float)Re[i]/(float)SHRT_MAX;
82         Xim[i]=(float)Im[i]/(float)SHRT_MAX;
83     }
84     //Inicio relog
85     inicio = clock();
86     //FFTI
87     int j, k, fk, m, n, ce, c, w;

```

```

88     float arg, seno, coseno, tempr, tempi;
89     //Bit reversal
90     m=log((float)total_muestras)/log(2.0);
91     j=w=0;
92     for (i = 0; i < total_muestras; i++){
93         if (j>i){
94             SWAP(Xre[i],Xre[j]);
95             SWAP(Xim[i],Xim[j]);
96         }
97         w=total_muestras/2;
98         while(w>=2 && j>=w){
99             j-=w;
100            w>>=1;
101        }
102        j+=w;
103    }
104    ce=m;
105    c=0;
106    //Mariposas
107    for (i = 0; i < m; i++) {
108        for (j = 0; j < (int)pow(2,ce-1); j++){
109            n = (int)pow(2,i);
110            for (k = 0; k < n; k++){
111                fk=k*(int)pow(2,ce-1);
112                coseno=cos(2*PI*fk/total_muestras);
113                seno=sin(2*PI*fk/total_muestras);
114                tempr=Xre[c+n];
115                Xre[c+n]=(Xre[c+n]*coseno) - (Xim[c+n]*seno);
116                Xim[c+n]=(Xim[c+n]*coseno) + (tempr*seno);
117                tempr=(Xre[c]+Xre[c+n]);
118                tempi=(Xim[c]+Xim[c+n]);
119                Xre[c+n]=(Xre[c]-Xre[c+n]);
120                Xim[c+n]=(Xim[c]-Xim[c+n]);
121                Xre[c]=tempr;
122                Xim[c]=tempi;
123                c++;
124            }
125            c += n;
126        }
127        c = 0;
128        ce -= 1;
129    }
130    //Obtener tiempo e imprimir
131    final = clock();
132    total = (double)(final - inicio) / CLOCKS_PER_SEC;

```

```

133     printf("Tiempo de ejecucion: %f\n", total);
134     short *Reales=(short *)malloc(total_muestras * sizeof(short));
135     short *Imaginarias=(short *)malloc(total_muestras * sizeof(
        short));
136     convertirShort(Reales,Imaginarias,Xre,Xim);
137     //Escribo el resultado en el archivo
138     escribirArchivo(Reales,Imaginarias);
139 }
140 void convertirFloat(short *muestras, float *Xre, float *Xim){
141     int i;
142     for (i = 0; i < total_muestras; i++){
143         Xre[i]=(float)muestras[i]/(float)SHRT_MAX;
144         Xim[i]=0.0;
145     }
146 }
147 void convertirShort(short *Reales, short *Imaginarias, float *
    Xre, float *Xim){
148     int i;
149     for (i = 0; i < total_muestras; i++){
150         Reales[i]=Xre[i]*SHRT_MAX;
151         Imaginarias[i]=Xim[i]*SHRT_MAX;
152     }
153 }

```

3. Pruebas

Para comprobar el funcionamiento de los programas se usaron los siguientes archivos wav.

Para realizar el filtrado mediante el producto en frecuencia, se uso como entrada para el programa de la FFT el siguiente archivo:

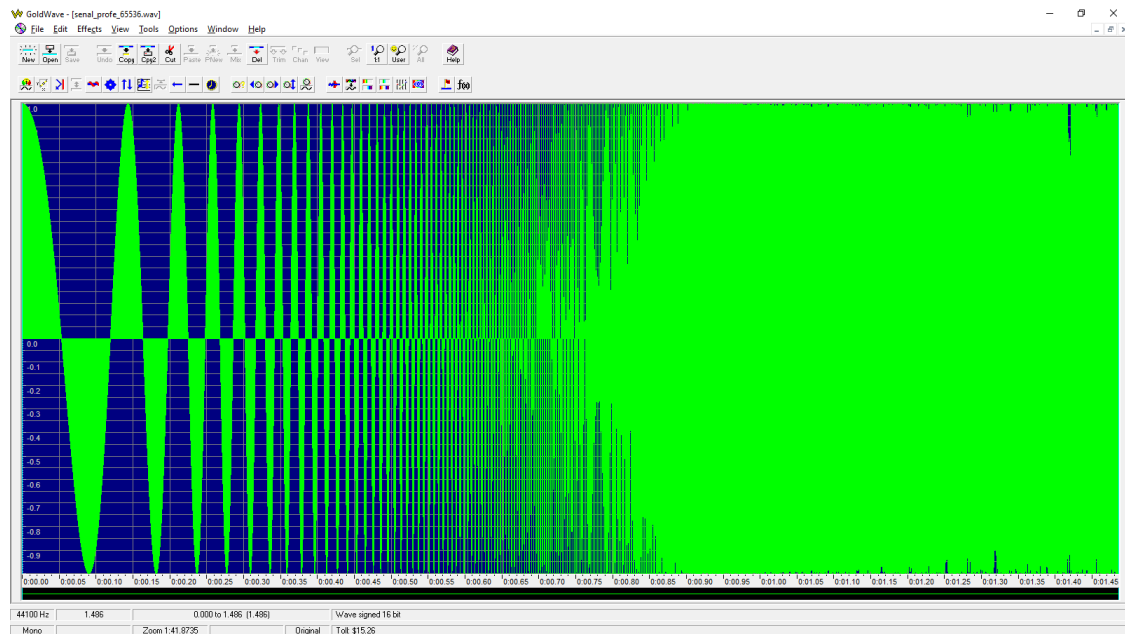


Figura 5: Archivo de entrada para la FFT

El cual tiene una frecuencia de muestreo de 44100 muestras/s y una duración de 1.486 segundos.

Esa duración fue seleccionada debido a que combinado con la frecuencia de muestreo, el número de muestras obtenidas es cercana a una potencia de dos, la cantidad de muestras que recibe la FFT debe ser una potencia de dos para que funcione correctamente.

La función usada en el archivo fue: $\cos(2\pi t * (\exp(\log(20) + n/N * 6.6)))$.

La salida obtenida del programa FFT fue la siguiente:

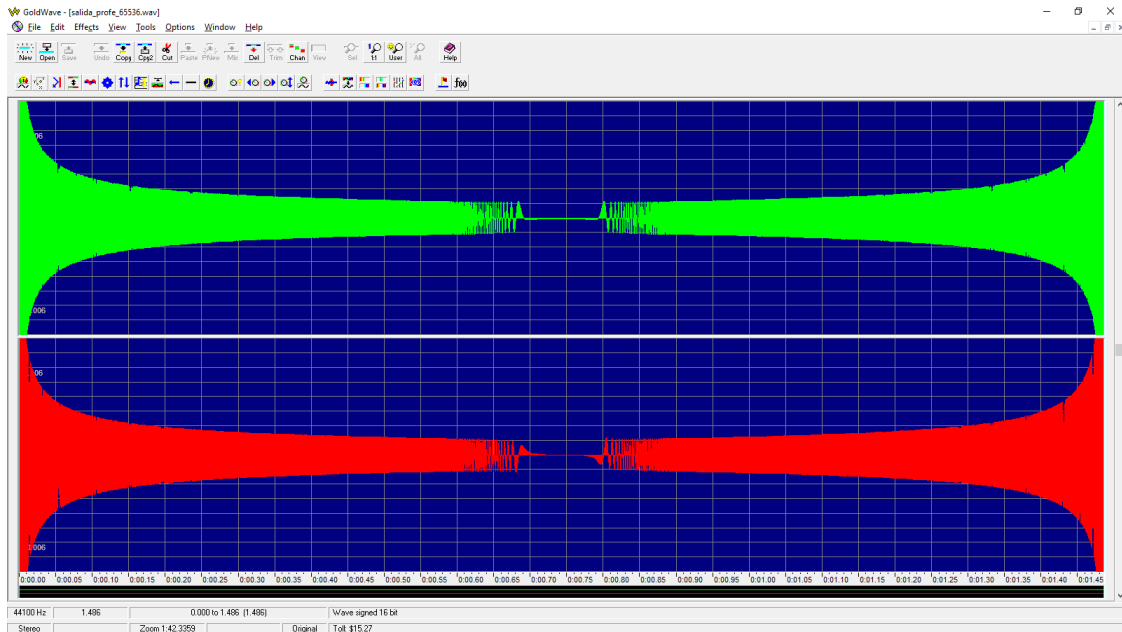


Figura 6: Salida FFT

El filtro para realizar el producto se muestra en la siguiente Figura, el filtro fue creado como un archivo wav de 2 canales para simular que el filtro se encuentra en el dominio de la frecuencia, además de que el filtro fue diseñado para ser un filtro ideal (el filtro tiene la misma frecuencia de muestreo y duración que la señal de entrada, y se uso la siguiente función para crearlo: $f(x)=(\text{step}(n)-\text{step}(n-1486))+\text{step}(n-(65536-1486)))$).

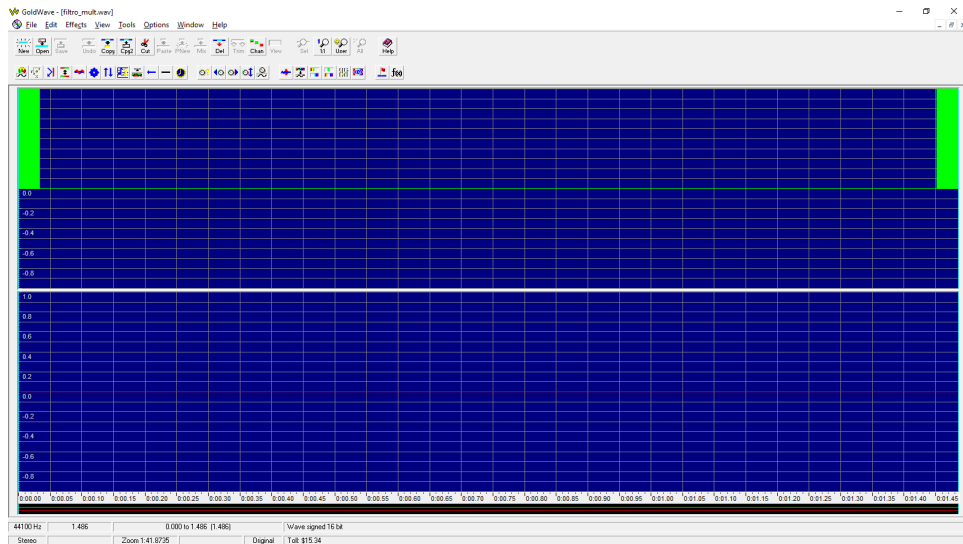


Figura 7: Filtro ideal en frecuencia

Ambos archivos (salida de FFT y el filtro) se multiplicaron y se obtuvo la siguiente la salida:

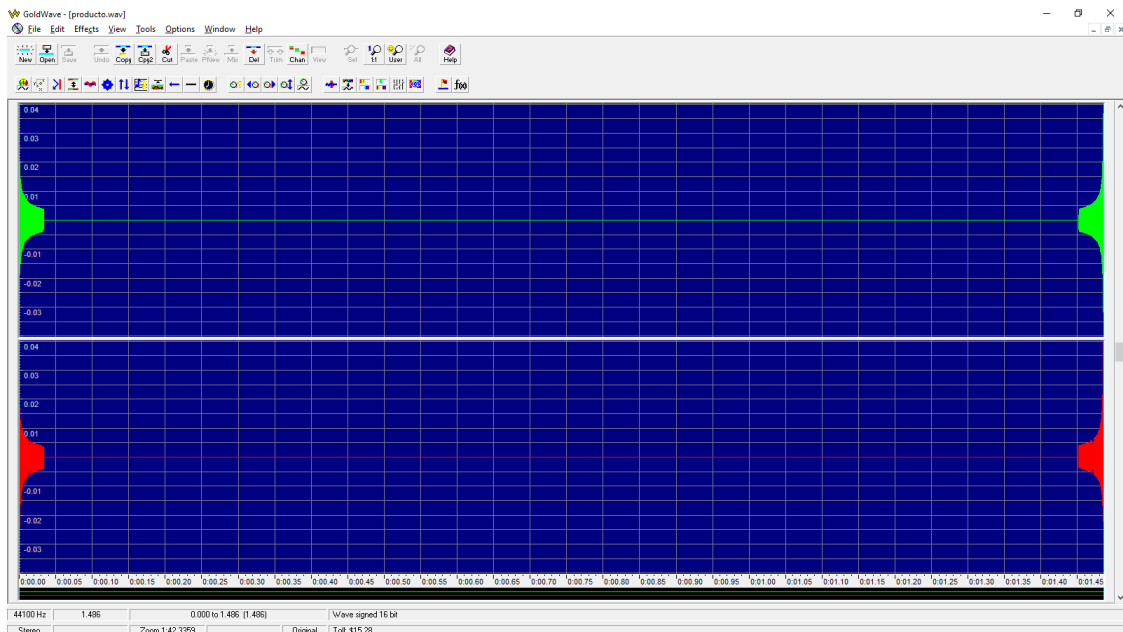


Figura 8: Salida del programa de multiplicación

Finalmente, la salida obtenida de la multiplicación fue ingresado al programa que calcula la FFT Inversa, para regresar del dominio de frecuencia al del tiempo.

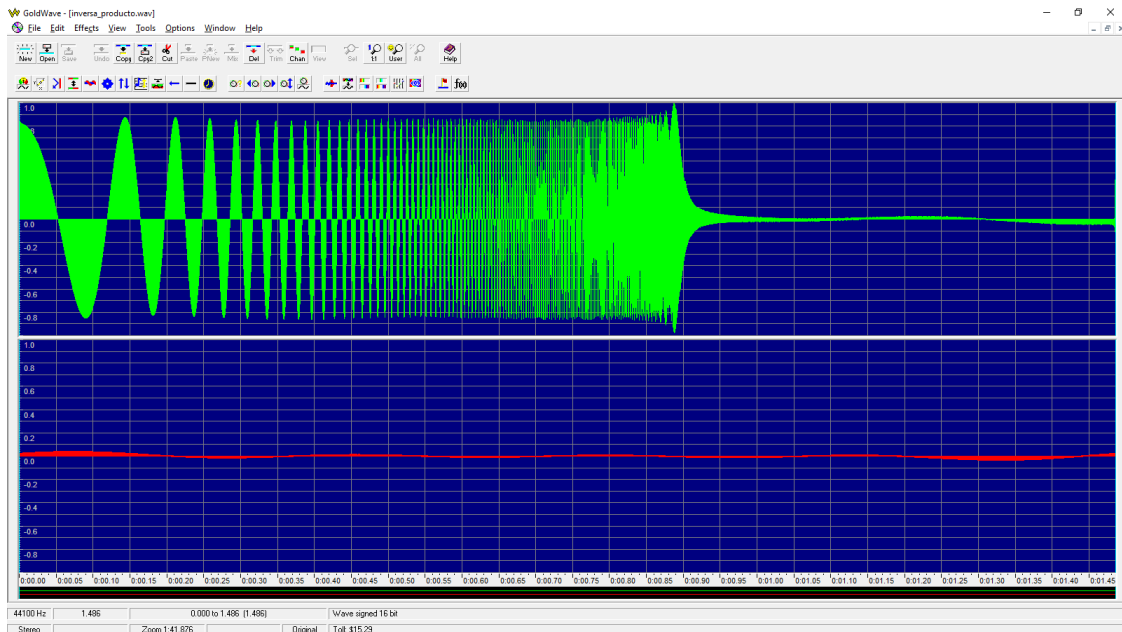


Figura 9: Salida obtenida FFT Inversa

Para comprobar si el funcionamiento de los programas fue el correcto, se realizó el filtrado de la señal original mediante el uso de Goldwave:

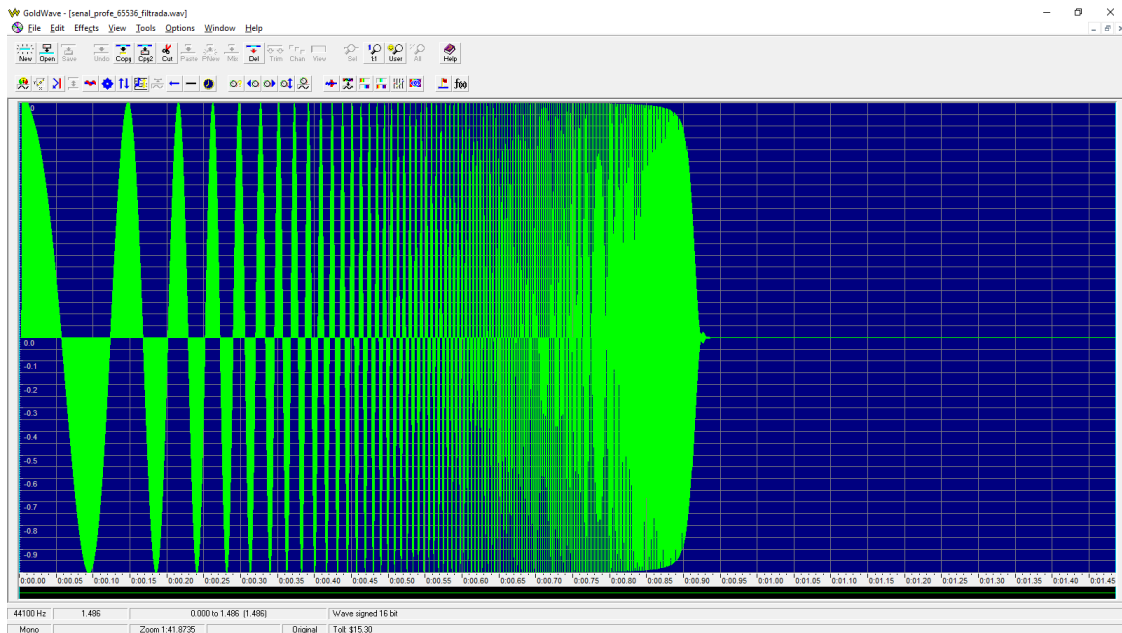


Figura 10: Filtrado de la señal original mediante Goldwave

Como se puede observar, el filtrado se realizó correctamente y es muy similar al realizado por Goldwave.

Como fue mencionado anteriormente, la FFT tiene un mejor rendimiento (menor tiempo de ejecución) que la TDF, a continuación se muestra un gráfico, del tiempo promedio (en segundos) que tarda la TDF en procesar N muestras.

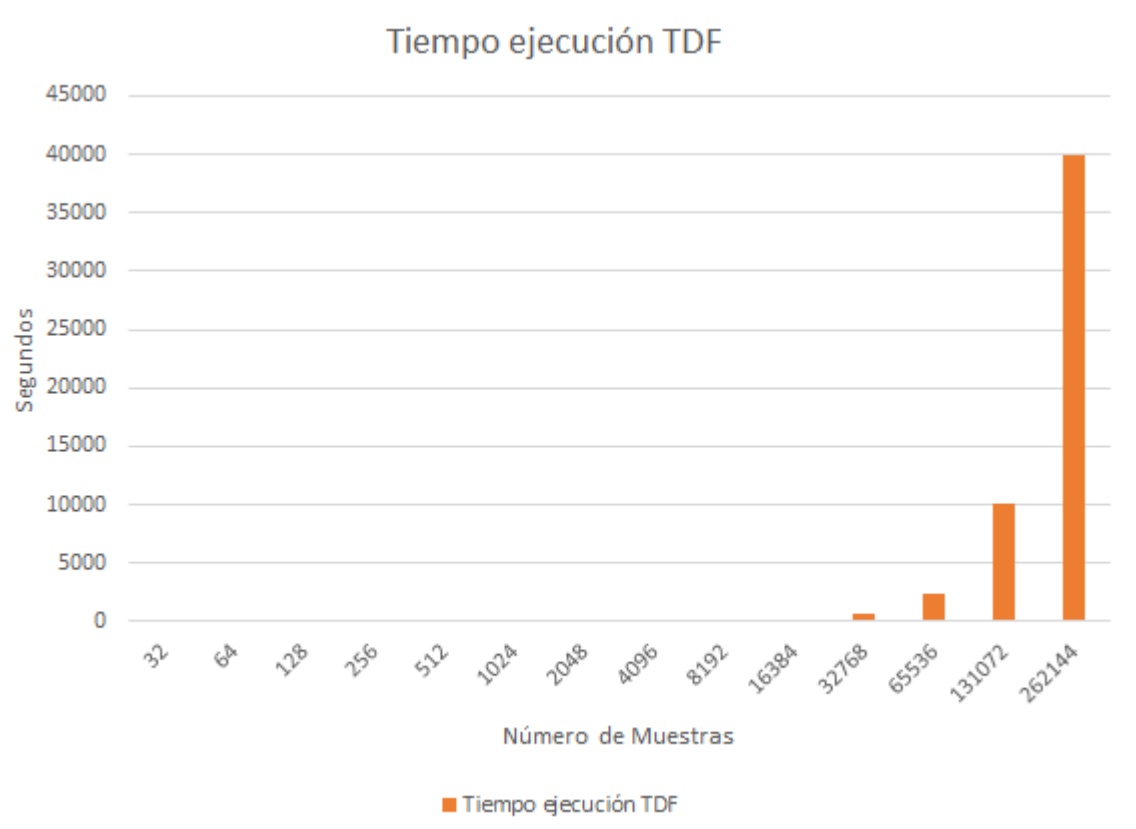


Figura 11: Tiempos de ejecución de la TDF

Ahora, se muestra otro gráfico de tiempos de ejecución pero usando la FFT, en la que se puede notar una enorme diferencia:

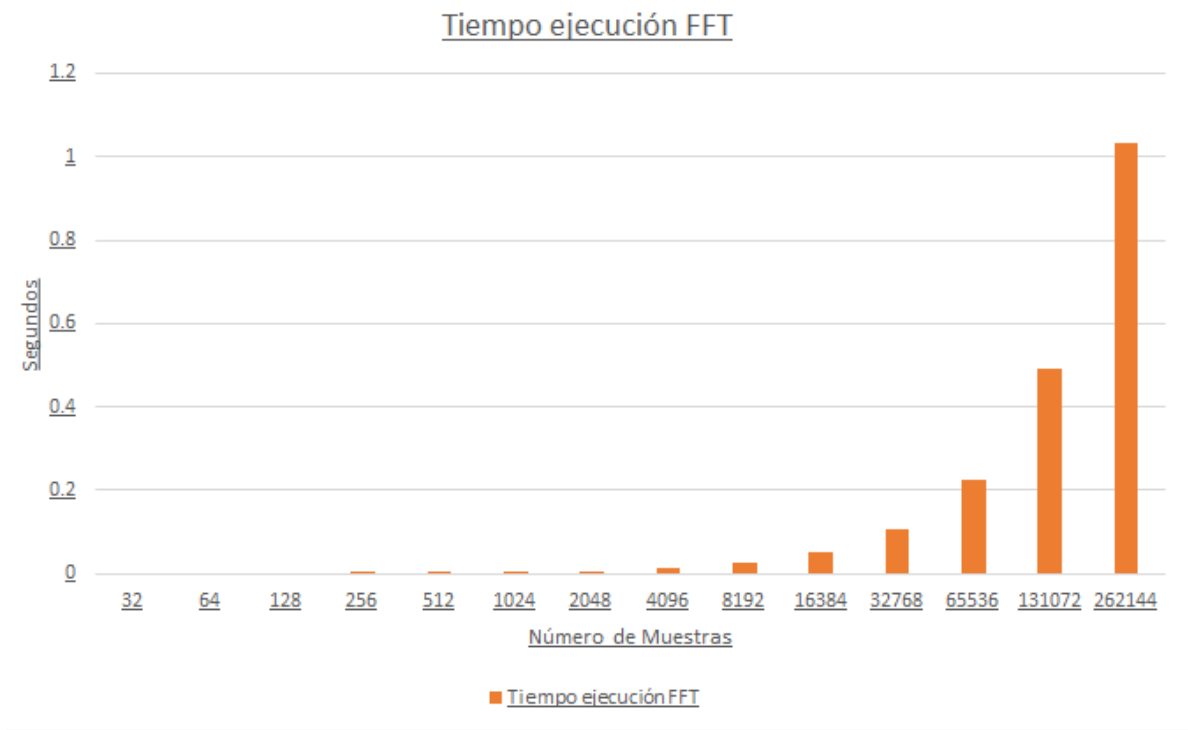


Figura 12: Tiempos de ejecución de la FFT

Mientras que la TDF puede tardarse cerca de 12 horas en completarse si recibe 262,144 muestras, la FFT se tarda cerca de 1.1 segundos en completarlo. Por lo cual, se puede observar la eficiencia del segundo algoritmo en comparación con el primero.

El último gráfico, muestra una comparación entre ambas:

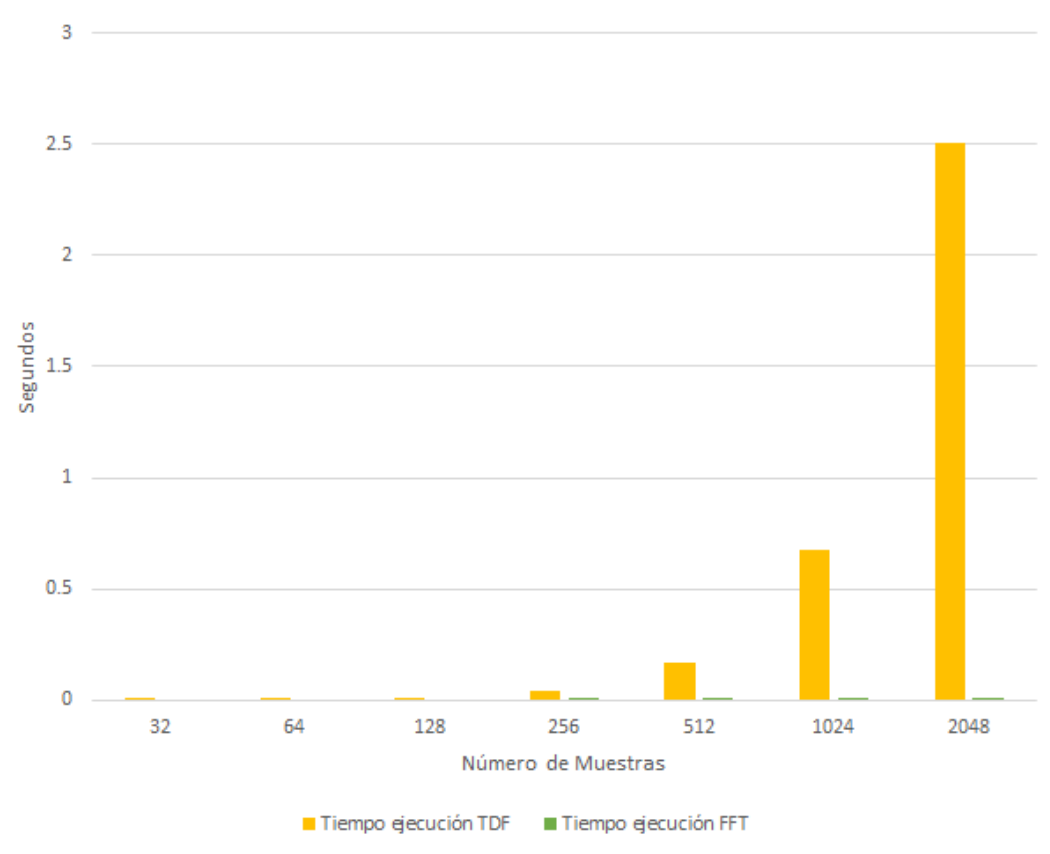


Figura 13: Comparación entre la TDF y la FFT

4. Conclusiones

El uso de algoritmos más veloces para calcular la TDF resulta necesario cuando se busca tener una aplicación real de la misma, debido a que el tiempo de ejecución de la TDF aumenta considerablemente dependiendo del número de muestras que reciba de entrada. En la práctica, se usan algoritmos con los que se obtenga la misma salida que con la TDF pero que sean mucho más eficientes, uno de ellos es la FFT, la cual mejora dramáticamente el tiempo de ejecución que tiene la TDF y se obtiene una salida relativamente igual a la de esta.

Los aspectos más importantes del calculo de la FFT es que esta a diferencia de la TDF, necesita que el número de muestras que recibe como entrada sea una potencia de dos, por lo cual, en muchas ocasiones resulta necesario agregar más muestras a las recibidas como entrada, todas estas con un valor de cero. Esto con la finalidad de hacer que el número de muestras final sea una potencia de dos. Además, la FFT recibe como parámetros no solo las muestras reales de la señal de entrada, si no que también recibe las muestras imaginarias, por lo cual, también es necesario realizar un ajuste para que esta reciba como ceros todas las muestras complejas (cabe aclarar, que este procedimiento solo es realizado cuando se calcula la FFT y no cuando se calcula la FFTI, debido a que esta última recibe como entrada un archivo que tiene un canal con muestras reales tanto otro canal con muestras complejas). Por lo cual, cuando se busque trabajar con un número elevado de muestras es necesario el uso de la FFT, tal como se pudo observar en el filtrado, la resolución obtenida por la FFT fue muy buena, y después de todo el procedimiento realizado, se obtuvo como resultado una señal muy parecida a la que se consiguió cuando se realizó el filtrado con Goldwave

Referencias

- [1] S. W. Smith, *The Scientist and Engineer's Guide to Digital Signal Processing*. California Technical Publishing, 2011.
- [2] Sengpielaudio, "Rc filter and cutoff frequency [online]. disponible en: <http://www.sengpielaudio.com/calculator-rcpad.htm>."