Reporte 01A.- Formato archivos wav

Alumno: Monroy Martos Elioth
Profesor: Gutierrez Aldana Eduardo
Materia: Teoría de Comunicaciones y Señales
Grupo: 3CM6

12 de noviembre de 2017

Índice

1. Introducción

El formato de archivos de audio digital wav (o wave), es un formato sin compresión de datos que fue desarrollado por Microsoft e IBM, es utilizado para almacenar sonidos en una computadora. Admite archivos mono y estéreos (un canal o dos respectivamente) a diversas resoluciones y velocidades de muestreo.
Debido a que no presenta pérdida de calidad, es comúnmente usado profesionalmente para guardar audio en CDs o DVDs. Pero tiene como desventaja, que los archivos creados bajo este formato son bastante grandes en comparación con otros formatos comprimidos como mp3 u ogg.
El formato de archivos wav es una variante del formato RIFF (formato de archivo para el intercambio de recursos). Un archivo en formato RIFF, empieza con una cabecera de archivo (file header) seguida de una secuencia de fragmentos de datos (data chunks). Un archivo wav comúnmente, es solo un archivo RIFF pero con una sola "wave" chunk, la cual esta formada por dos sub-chunks (fmt sub-chunk y data sub-chunk). A esta forma se le conoce como la forma "canónica" de un archivo en formato wav.
A continuación se presenta una lista con todos los campos que almacenan la información dentro del archivo wav:

- ChunkID (4 bytes): Contiene las letras "RIFF" en ascii.
- ChunkSize (4 bytes): Es el tamaño del archivo entero menos los 8 bytes de los campos anteriores que no son contados.
- Format (4 bytes): Contiene las letras "WAVE".
- Subchunk1ID (4 bytes): Contiene las letras "fmt".
- Subchunk1Size (4 bytes): Contiene el tamaño del resto del subchunk que queda a partir de este número.
- AudioFormat (2 bytes): Regularmente almacena el número 1, si se guarda algún otro número este indica algún tipo de compresión.
- NumChannels (2 bytes): 1 para mono, 2 para stereo.
- SampleRate (4 bytes): La frecuencia de muestreo que se uso para muestrear el archivo (8000, 44100, etc).

- ByteRate (4 bytes): Es igual a: SampleRate*NumChannels*BitsPerSample/8.
- BlockAlign (2 bytes): Guarda el número de bytes por cada muestra (considerando todos los canales). Se obtiene mediante: NumChannels*BitsPerSample/8.
- BitsPerSampl (2 bytes): La cantidad de bits que se almacenan por muestra (8 bits, 16 bits, etc).
- SubChunk2ID (4 bytes): Contiene las letras "data".
- SubChunk2Size (4 bytes): Es el número de bytes de los datos, también puede considerarse como el resto del subchunk apartir de este dato.
- Data (tamaño del SubChunk2Size): Son los datos del sonido almacenado.

Los campos ChunkID, ChunkSize y Format componen la cabecera del archivo. Los campos Subchunk1ID, Subchunk1Size, AudioFormat, NumChannels, SampleRate, ByteRate, BlockAlign y BitsPerSample componen el primer sub-chunk llamado fmt. Los campos Subchunk2ID, Subchunk2Size y data componen el segundo sub-chunk llamado data.
La figura 1 muestra de forma gráfica las cabeceras y campos antes descritos:

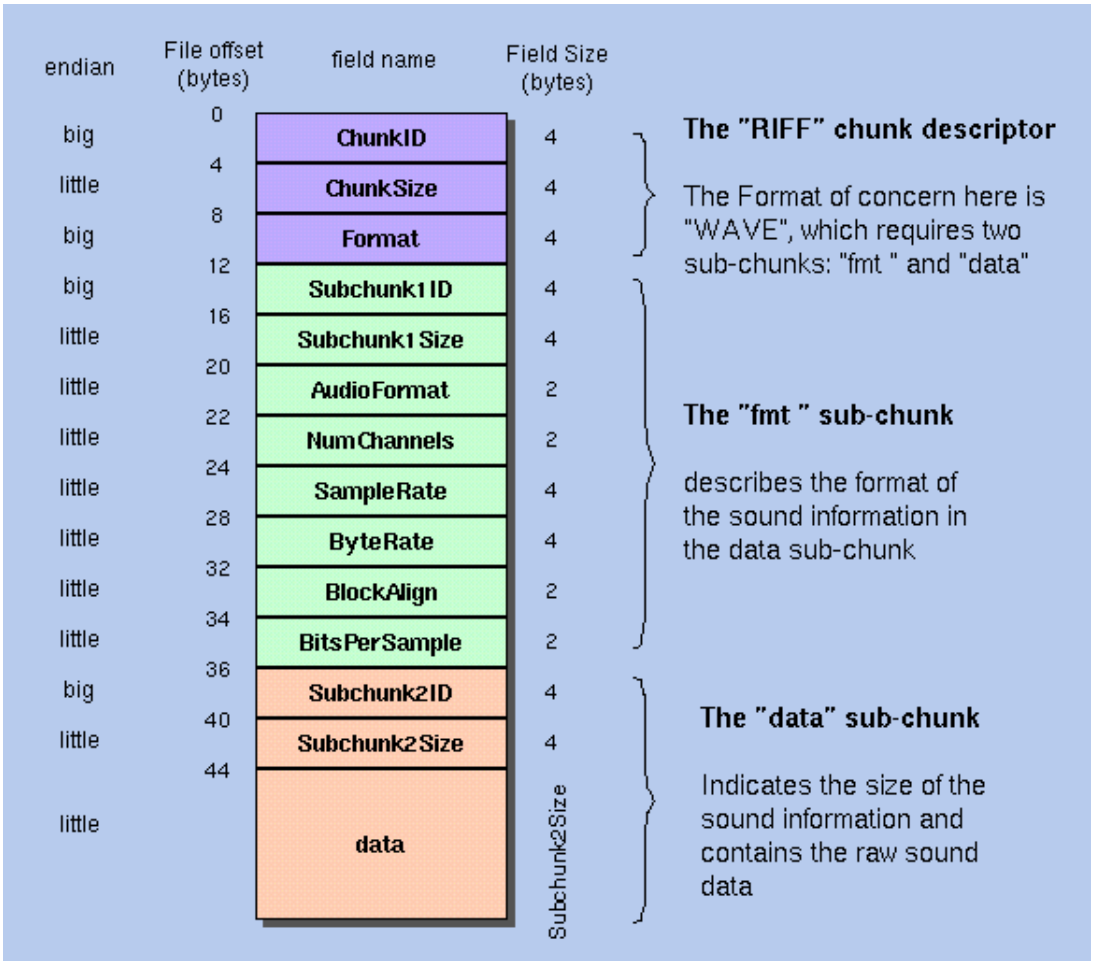

Figura 1: Campos que componen un archivo canónico wav

2. Código

A continuación se presenta el código del programa desarrollado que permite cambiar el volumen a un archivo en formato wav, el programa reduce el volumen del sonido a la mitad (lo cual resulta equivalente a dividir entre dos cada muestra en el archivo).
funciones.h:

```
#ifndef __FUNCIONES_H__
#define __FUNCIONES_H__
  //Librerías de C
  #include <stdio.h>
  #include <stdlib.h>
  //Métodos
  void leerCabeceras(char**);
  void leerMuestras(short*);
  void dividirMuestras(short*,float);
  void escribirArchivo(short*);
  //Cabeceras
  int chunkid;
  int chunksize;
  int format;
  int subchunk1id;
  int subchunk1size;
  short audioformat;
  short numchannels;
  int samplerate;
  int byterate;
  short blockalign;
  short bitspersample;
  int subchunk2id;
  int subchunk2size;
  //Archivos
  FILE* entrada;
  FILE* salida;
  //Variables para muestras
  short muestra;
  int total_muestras;
  short headers[37];
#endif
```

half.c:

```
#include"funciones.h"
int main(int argc, char *argv[]){
  //Leo las cabeceras
  leerCabeceras(argv);
  //Defino variables
  total_muestras=subchunk2size/blockalign;
  short muestras[total_muestras];
  //Leo las muestras
  leerMuestras(muestras);
  //Divido el volumen a la mitad
  dividirMuestras(muestras,.5);
  //Escribo el archivo
  escribirArchivo(muestras);
}
void leerCabeceras(char ** argv){
  //Abrir archivos
  entrada = fopen(argv[1], "rb");
  salida=fopen(argv[2],"wb");
  if(!entrada) {
    perror("\nFile opening failed");
    exit(0);
  }
  fread(&chunkid,sizeof(int),1,entrada);
  fread(&chunksize,sizeof(int),1,entrada);
  fread(&format,sizeof(int),1,entrada);
  fread(&subchunk1id,sizeof(int),1,entrada);
  fread(&subchunk1size,sizeof(int),1,entrada);
  fread(&audioformat,sizeof(short),1,entrada);
  fread(&numchannels,sizeof(short),1,entrada);
  fread(&samplerate,sizeof(int),1,entrada);
  fread(&byterate,sizeof(int),1,entrada);
  fread(&blockalign,sizeof(short),1,entrada);
  fread(&bitspersample,sizeof(short),1,entrada);
  fread(&subchunk2id,sizeof(int),1,entrada);
  fread(&subchunk2size,sizeof(int),1,entrada);
}
void leerMuestras(short *muestras){
  int i=0;
  while (feof(entrada) == 0)
  {
    if(i<total_muestras){
      fread(&muestra,sizeof(short),1,entrada);
      muestras[i]=muestra;
      i++;
```

```
45      }else{
46        fread(&headers,sizeof(short),37,entrada);
47        break;
48      }
49    }
50  }
51  void dividirMuestras(short *muestras, float factor){
52    int i;
53    for (i = 0; i < total_muestras; i++)
54      {
55        muestras[i]*=factor;
56      }
57  }
58  void escribirArchivo(short* muestras){
59    //Escribo el archivo
60    fwrite(&chunkid,sizeof(int),1,salida);
61    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(&muestras[i],sizeof(short),1,salida);
77    }
78    //Y por último los headers de goldwave
79    for(i=0;i<37;i++){
80      fwrite(&headers[i],sizeof(short),1,salida);
81    }
82  }
```

3. Pruebas

Para comprobar el funcionamiento del programa se ingreso la siguiente señal (como archivo wav):

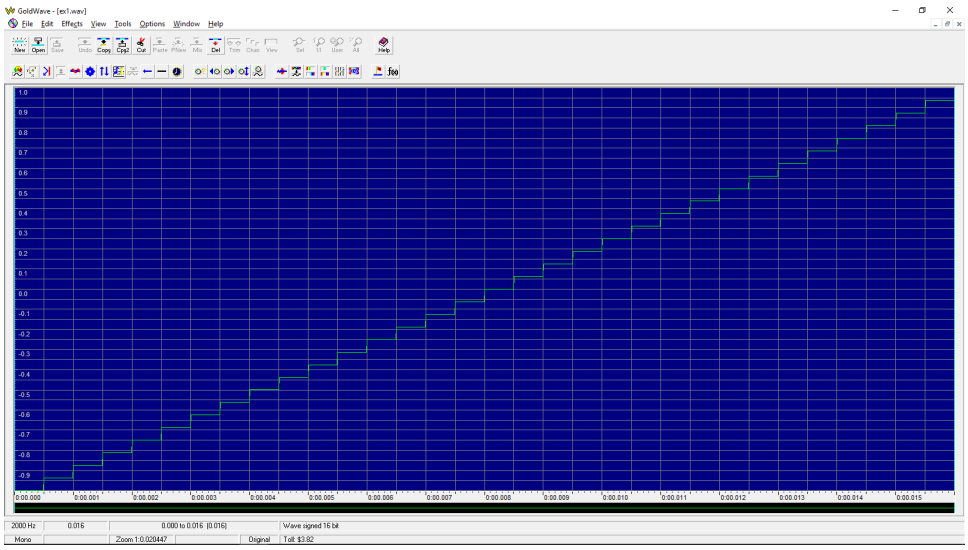

Figura 2: Entrada 1

Salida obtenida:

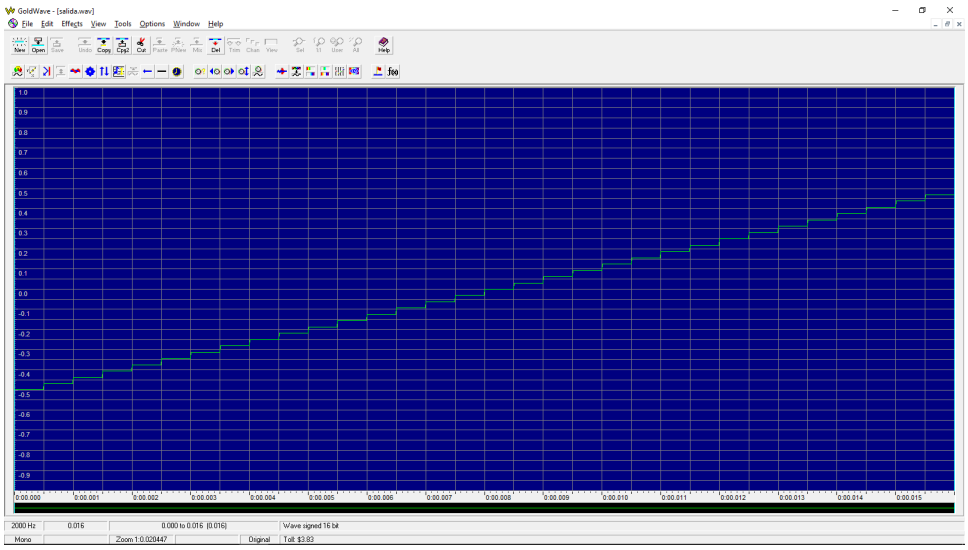

Figura 3: Salida 1

Posteriormente, se probó el programa con la siguiente señal de entrada:

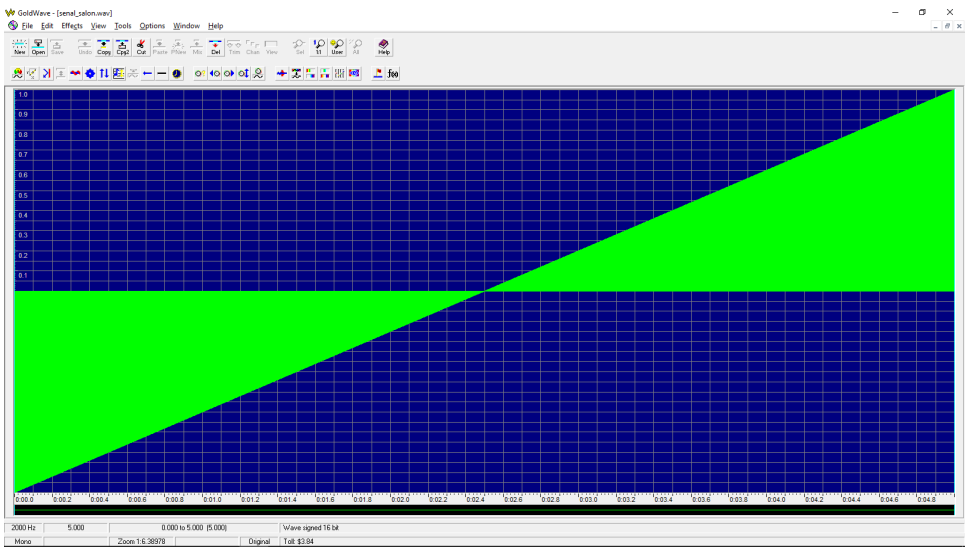

Figura 4: Entrada 2

Salida obtenida:

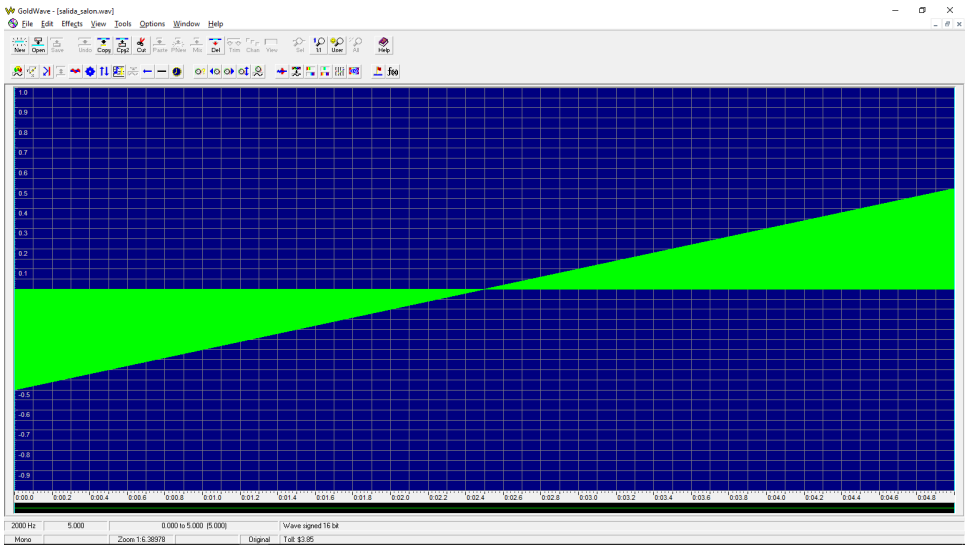

Figura 5: Salida 2

4. Conclusiones

El programa desarrollado en esta práctica, funcionó como un primer acercamiento hacia el formato de archivos wav, el cuál es un formato de audio digital. El familiarizarnos con el mismo nos permite poder realizar trabajos posteriores con este, debido a que es una forma conveniente de representar señales.
Durante este primer acercamiento, pude determinar como leer la información contenida en el archivo wav (las cabeceras y datos), además de comprender como tratar esta información para poderla modificar y así generar nuevos archivos en formato wav.
Una parte importante de este análisis, fue la lectura de cabeceras del archivo wav, ya que dependiendo del tamaño de la cabecera se tiene que usar un tipo de dato capaz de almacenarla, para las cabeceras de 4 bytes use variables de tipo int y para las cabeceras de 2 bytes use variables de tipo short (cada una puede almacenar hasta 4 y 2 bytes respectivamente).

Referencias

[1] S. University, "Wave pcm soundfile format [online]. disponible en: http://soundfile.sapp.org/doc/waveformat/."