# DSD

Digital Speaker Design

# Introduction

DSD stands for Digital Speaker Design. It is a set of GNU-octave scripts for loudspeaker crossover and equalization FIR filters. It contains tools for digital filter design in the form of Octave functions files that can be used separately from the Octave command line, and a main design script that uses them, and that is run from the operating system command line. The parameters used for one specific design are taken from configuration files written by the user.

GNU-octave is a programming environment for math calculations, largely compatible with Matlab, that can be used interactively from its own command line or by running scripts. You can get more information from the Octave website, https://www.gnu.org/software/octave/ .

## *Installation*

DSD can be executed on the more usual operating systems, with the only condition that there exist a Octave version for them. The following indications are detailed for the Ubuntu-Linux operating system, but the general procedure should be similar for others OSs. Beware that the way file paths are expressed may vary between systems.

Steps are:

- Install GNU-Octave, plus necessary function packets.
- Install git.
- Use git for DSD download.
- Add DSD directory to the Octave search path.
- Add DSD directory to the operating system executable search path.

## Octave

Open a terminal. Write: ‘`sudo apt install octave`’.

The following packets are necessary: control, general, io, signal y statistics. To install them write:

```
‘sudo apt install octave-control octave-general octave-io
octave-signal octave-statistics’
```

It is likely that some Octave distributions have some packets already installed. You can check it with the command *`pkg list`* on the Octave command line.

You can find specific info for installation on other operating systems in the Octave website.

## Git

```
‘sudo apt install git’.
```

You can find specific info for installation on other operating systems in the git website: https://git-scm.com/downloads.

## DSD

Change to the directory you want to install DSD under, for instance: ‘`cd /home/[user]`’, where [user] should be changed for the actual user name.

Use git to clone the DSD repository: ‘`git clone https://github.com/rripio/DSD`’.

## Octave path

In Octave command line write:

```
addpath(genpath('/home/[user]/DSD/src'));
savepath();
```

This will permanently save the DSD path in the Octave scripts search path.

## Operating system path

Edit **/home/[user]/.bashrc** and add:

```
PATH=$PATH:/home/[user]/DSD/src
```

With [user] being the actual user name.

# Making of the filters

The procedure consist of:

- Obtention and tailoring of frequency response files in **.frd** format for each driver in the loudspeaker.
- Writing of configuration files (**.xof** extension) with the necessary filter parameters (relative to crossover frequencies, type, etc.) for each driver.
- Running of '`dsd`' program, with the configuration file name as parameter, and optionally giving sampling frequency and filter class as additional parameter.
- Reviewing the attenuation levels given by the program in order to design the gain structure of amplification and/or digital gain.

Let's detail this steps hereunder:

## *Obtention and tailoring of frequency response files*

Filters are made by inverting the loudspeaker frequency response as given in some **.frd** file. It is the designer who has to tailor this file with proper tools for measurement and edition.

This way the designer have the maximum freedom to get a relevant response to invert, by windowing the impulse, stitching curves, adjusting level or applying smoothing, in order to have the best possible representation of the loudspeaker behavior.

If the relative levels between different driver curves are correct, we will end up with attenuation factors that are also correct relative to each other.

The configuration file basename will also be used to compose the basename of output files.

## *Writing of configuration files*

We are considering two filter classes: linear phase filters (*lp*) and minimum phase filters (*mp*).

When designing minimum phase filters we can opt for Butterworth responses of any order or Linkwitz-Riley responses of any even order. Also, we can use multiple high pass filters in a driver to have on it the same delay of another loudspeaker. A typical case is adding to a tweeter a Butterworth second order high pass that mimics the natural cutoff of the woofer.

The example folder have a set of files for a design of a 3 ways speaker. The **.xof** files on there can be copied, typically to the same folder that have the frequency response files, and modified to the designer needs.

In the source folder of DSD there is a dsd.ini file that have default options for every possible variable. The loudspeaker configuration files have precedence over this file.

Some parameters can be given in the command line. Command line parameters have precedence over both general and specific configuration files. This way the same configuration file can be used to make filters for different sampling frequency and filter class.

## *Running DSD*

Open a terminal and move to the project folder ('`cd [project_folder]`') and run '`dsd <configuration_file_basename>`'. **<configuration_file_basename >** is the name of the .**xof** file that configures the corresponding speaker filter. The program will generate a **.pcm** or **.wav** filter and some graphics explaining the results.

***GSFs*** (sampling frequency) and ***CFClass*** (filter class, *mp* or *lp*) can be given as aditional parameters. Sampling frequency is given as its numeric value. Also a parameter `nopause` can be given, that will avoid the usual pause to show graphics on screen:

```
‘dsd <configuration_file_basename> [fs <number>] [class <mp|lp>]
[nopause]’
```

For example, ‘`dsd t fs 44100 class lp`’ will calculate a filter following the directives in a **t.xof** file in the actual directory, tailored for a sampling frequency of 44100, and of linear phase class.

The configuration basename is a mandatory parameter and has to be given in first position. The rest of parameters are optional and its order is indifferent.

The name of generated files will be:

**xo-<class>-<configuration_file_basename>.pcm**

They will be written to a subdirectory of the project directory (the one containing the configuration file) named with the corresponding sampling frequency number.

Since we can give sampling frequency and filter class as parameters it is possible make a script that will generate a set of filters varying that parameters while maintaining the basic parameters, like crossover frequencies, filter orders, etc. The reader can check the script **calc_example** in the **example** folder to see the details or change it to their needs.

## Configuration parameters

The meaning of  the various configuration parameters is detailed hereunder, grouped by operation area. Operation area is expressed in parameter prefix.

## GS, General Settings

### *GSFs*

Sampling frequency.

### *GSLExp*

Power of two expressing the length of final filter. FFT require operands whose length is a power of two. Usual values will be *15* or *16*, resulting in filters of  2^15 or 2^16 coefficients, respectively.

### *GSLExpCalc*

Power of two expressing the filter length used in calculations. It will typically  be greater than ***GSLExp*** to get more precision.

### *GSLevelInterval*

Interval, in octaves, used in level estimations when two interval limits are not defined.

## FS File Settings

### *FSInputFile*

**.frd** filename of loudspeaker frequency response.

### *FSNormFile*

Filename of level normalization values. By default **levels.txt**.

### *FSOutPcmFilter*

Boolean (*true* or *false*) indicating if raw PCM filters are generated. Format is ‘float 32’.

### *FSOutPcmExt*

Extension of raw PCM files (usually **.raw** or **.pcm**).

### *FSOutWavFilter*

Boolean (*true* or *false*) indicating if WAV filters are generated.

### *FSOutWavDepth*

Bit depth of WAV files.

## CF Crossover Filter

### *CFLowF*

Crossover point frequency, high pass filter on the left side, in hertz. If *0* no crossover will be made at that end (on woofers, for instance).

Can be a vector with same length as ***CFLowType,*** with various crossovers on the same side to simulate a cascade filter. If ***CFClass*** = *'lp'* only first vector element is used.

### *CFHighF*

Crossover point frequency, low pass filter on the right side, in hertz. If *0* no crossover will be made at that end (on tweeters, for instance).

### *CFClass*

Filter class (*mp* or *lp*).

The parameters below only have effect if ***CFClass*** = *'mp'* :

### *CFLowType*

Type of high pass filter  on the left side of the band —low—. It can be *B* (Butterworth) or *LR* (Linkwitz-Riley). It is a cell array with as many elements as wanted, and one filter will be made and cascaded for each element.

It must be given between curly brackets ({}) which is the Octave language convention for cell arrays.

### *CFLowOrder*

Order of high pass filter. It can be any positive integer when corresponding ***CFLowType*** is *B* (Butterworth) or an even positive integer when corresponding ***CFLowType*** is *LR* (Linkwitz-Riley).

### *CFHighType*

Type of low pass filter  on the right side of the band —high—. It can be *B* (Butterworth) or *LR* (Linkwitz-Riley).

### *CFHighOrder*

Order of low pass filter. It can be any positive integer when ***CFHighType*** is *B* (Butterworth) or an even positive integer when ***CFHighType*** is *LR* (Linkwitz-Riley).

The parameters below only have effect if ***CFClass*** = *'lp'* :

### *CFLenghthCycles*

Crossover filter length in cycles at crossover frequency.

The absolute length of this part is limited to a maximum length of m/2, that has precedence over the length resulting from this setting,

### *CFDelayAsF*

Frequency of a crossover that gives the length of the impulse dedicated to XO function.

This is used to establish a common delay for all drivers, and usually this will be the ***CFHighF*** of the woofer.

It cannot be greater than the lower valid given XO frequency, and if *0* XO part length will be *m/2*.

### *CFLowAsMP*

When set to *true* and if a Butterworth filter is specified in the first element of ***CFHighType*** then a linear phase filter that emulates the magnitude of a Butterworth high pass filter id made.

## TF Tail Fixing

### *TFBeginLow*

Es la frecuencia en que comienza la zona de transición en el lado izquierdo de la banda, según en el sentido decreciente de las frecuencias. Si este parámetro es *0* o es menor que ***TFEndLow*** se ignora. Si el valor es válido prevalece sobre ***TFIntervalLow***.

### *TFEndLow*

Es la frecuencia final de la zona de transición y comienzo de la recta. Si es cero se toma como límite el primer bin de frecuencia por encima de cero.

### *TFIntervalLow*

Define el rango de transición como un intervalo en octavas desde ***TFBeginLow***.

### *TFSlopeLow*

Pendiente de la recta que modela la cola de la respuesta, en dB/oct, en el extremo izquierdo de la banda. Las pendientes descendentes según el sentido creciente de las frecuencias son negativas.

### *TFBeginHigh*

Es la frecuencia en que comienza la zona de transición en el lado derecho de la banda, según en el sentido decreciente de las frecuencias. Si este parámetro es *0* o es mayor que ***TFEndHigh*** se ignora. Si el valor es válido prevalece sobre ***TFIntervalHigh***.

### *TFEndHigh*

Es el límite superior de la ecualización. Los valores de magnitud a frecuencias superiores a este límite no se ecualizan. Si es muy alto se toma como límite el primer bin de frecuencia por encima de cero.

### *TFIntervalHigh*

Define el rango de transición como un intervalo en octavas desde ***TFEndHigh***.

### *TWSlopeHigh*

Pendiente de la recta que modela la cola de la respuesta, en dB/oct, en el extremo derecho de la banda. Las pendientes descendentes según el sentido creciente de las frecuencias son negativas.

## EQL EQ Limits

La estrategia general de ecualización y filtrado conjunto que lleva a cabo el programa puede implicar ecualizaciones con ganancias muy elevadas si se intenta ecualizar fuera de la banda pasante.

Para poner límites a esta ecualización y está el siguiente grupo de parámetros, con los que se define una frecuencia a partir de la cual, con una curva de transición, la ecualización es plana.

### *EQLBeginLow*

Es la frecuencia en que comienza la zona de transición en el lado izquierdo de la banda, según en el sentido decreciente de las frecuencias. Si este parámetro es *0* o es menor que ***EQLEndLow*** se ignora. Si el valor es válido prevalece sobre ***EQLIntervalLow***.

### *EQLEndLow*

Es el límite inferior de la ecualización. Los valores de magnitud a frecuencias inferiores a este límite no se ecualizan. Si es cero se toma como límite el primer bin de frecuencia por encima de cero.

### *EQLIntervalLow*

Define el rango de transición como un intervalo en octavas desde ***EQLBeginLow***.

### *EQLBeginHigh*

Es la frecuencia en que comienza la zona de transición en el lado derecho de la banda, según en el sentido decreciente de las frecuencias. Si este parámetro es *0* o es mayor que ***EQLEndHigh*** se ignora. Si el valor es válido prevalece sobre ***EQLIntervalHigh***.

### *EQLEndHigh*

Es el límite superior de la ecualización. Los valores de magnitud a frecuencias superiores a este límite no se ecualizan. Si es muy alto se toma como límite el primer bin de frecuencia por encima de cero.

### *EQLIntervalHigh*

Define el rango de transición como un intervalo en octavas desde ***TFEndHigh***.

## PS Plot Settings

Estos valores definen algunos aspectos de los gráficos de salida.

### *PSFLow*

Extremo izquierdo de frecuencia.

### *PSFHigh*

Extremo derecho de frecuencia.

### *PSVTop*

Valor máximo de presión sonora en los gráficos (dB).

### *PSVStep*

Escalones de presión sonora en los gráficos (dB).

### *PSVRange*

Rango total de presiones sonoras de salida (dB).

### *PSPos*

Posición en unidades normalizadas al tamaño de la pantalla (entre *0* y *1*), como vector fila de las dos coordenadas. *[0,0]* es la posición de la esquina superior izquierda.

### *PSSize*

Tamaño en unidades normalizadas al tamaño de la pantalla (entre *0* y *1*), como vector fila de las dos coordenadas.

### *PSOutRaster*

Valor booleano (*true* or *false*) que indica si se generan gráficos en formato raster.

### *PSOutRasterExt*

Extensión del archivo de gráficos raster.

### *PSOutVector*

Valor booleano (*true* or *false*) que indica si se generan gráficos en formato vectorial.

### *PSOutVectorExt*

Extensión del archivo de gráficos vectoriales.

### *PSShow*

Muestra el gráfico en pantalla y detiene el programa para examinarlo.

# Funciones del paquete DSD

Se describen a continuación las funciones auxiliares del paquete DSD, que forman un pequeño conjunto de utilidades de audio. Obsérvese que las funciones con prefijo **dsd_** no se describen aquí, pues son de uso exclusivo del programa **dsd**, sin utilidad independiente.

### *audioaxe*

Prepare axes for a frequency response graph with a conventional format.

```
audioaxe(magtop, magrange, magstep, flow, fhigh, plotitle)
```

```
         magtop   = Magnitude maximum (dB).
         magrange = Visible range of magnitude (dB).
         magstep  = dB steps for grid and labeling.
         flow     = Frequency lower limit (Hz).
         fhigh    = Frequency upper limit (Hz).
         plotitle = Graph title.
```

### *audioplot*

Draws a frequency response graph with a conventional format.

```
audioplot(F, dBmag, magtop, magrange, magstep, flow, fhigh, plotitle)
```

```
         F        = Frequencies vector.
         dBmag    = Magnitudes vector dB.
         magtop   = Magnitude maximum (dB).
         magrange = Visible range of magnitude (dB).
         magstep  = dB steps for grid and labeling.
         flow     = Frequency lower limit (Hz).
         fhigh    = Frequency upper limit (Hz).
         plotitle = Graph title.
```

### *centerimp*

Centers an impulse in a given length. Length of input impulse must be odd.

```
imp = centerimp(imporig,m)
```

```
         imp     = FIR filter coefficients.
         imporig = Impulse to center. Length must be odd.
         m       = Final impulse length.
```

### *crossButterworth*

Gets a n order FIR Butterworth filter.

```
imp = crossButterworth(Fs,m,nl,fl,nh,fh)
```

```
         imp = FIR filter Coefficients.
         Fs  = Sampling frequency.
         m   = Number of samples.
         nl  = Order of highpass filter.
         fl  = Highpass crossover frequency. 0 if only lowpass.
         nh  = Order of lowpass filter.
         fh  = Lowpass crossover frequency. 0 if only highpass.
```

### ***crossButterworthLP***

Gets the linear phase FIR filter that has the magnitude of a Butterworth filter of order n.

```
imp = crossButterworthLP(fs,m,nl,fl,nh,fh)

          imp = FIR filter Coefficients.
          Fs  = Sampling frequency.
          m   = Number of samples.
          nl  = Order of highpass filter.
          fl  = Highpass crossover frequency. 0 if only lowpass.
          nh  = Order of lowpass filter.
          fh  = Lowpass crossover frequency. 0 if only highpass.
```

### ***crossLinear***

Gets a windowed sinc linear phase FIR filter. Window is of Blackman-Harris type with a length, in cycles at crossover frequency, of ***nc***×***fs***÷***lf*** or ***nc***×***fs***÷***lf***. The window is centered in the total length ***m***.

```
imp = crossLinear(Fs,m,nc,fl,fh)

          imp = FIR filter Coefficients.
          Fs  = Sampling frequency.
          m   = Number of samples.
          nc  = Length factor of impulse (factor of fs/fl or fs/fh).
          fl  = Highpass crossover frequency. 0 if only lowpass.
          fh  = Lowpass crossover frequency. 0 if only highpass.
```

### ***crossLinkwitzRiley***

Gets a n order FIR Linkwitz-Riley filter, n even.

```
imp = crossLinkwitzRiley(Fs,m,nl,fl,nh,fh)

          imp = FIR filter Coefficients.
          Fs  = Sampling frequency.
          m   = Number of samples.
          nl  = Order of highpass filter.
          fl  = Highpass crossover frequency. 0 if only lowpass.
          nh  = Order of lowpass filter.
          fh  = Lowpass crossover frequency. 0 if only highpass.
```

### ***dB2mag***

Converts decibels to magnitude.

```
b = dB2mag(a)

          b = Magnitude.
          a = Decibels.
```

### ***delta***

Gets an m length delta impulse. First sample has value 1.

```
imp = delta(m)

          imp = FIR filter Coefficients.
          m   = Number of samples.
```

### *deltacentered*

Obtiene un impulso de longitud ***m*** con valor *1* en su muestra central.

```
imp = deltacentered(m)

         imp      = Coeficientes del filtro FIR.
         m        = Número de muestras. Debe ser impar.
```

### *frjoin*

Une dos respuestas en magnitud sobre el semiespectro, mezclándolas en un intervalo de índices dado.

```
ssp = frjoin(ssp1,ssp2,k1,k2)

         ssp      = Vector columna con la magnitud de la mezcla.
         ssp1     = Vector columna con la respuesta a mezclar por la izquierda.
         ssp2     = Vector columna con la respuesta a mezclar por la derecha.
         k1       = Primer índice del intervalo.
         k2       = Segundo índice del intervalo.
```

### *gainpcm*

Aplica una cierta ganancia en dB a un archivo **.pcm** y lo guarda.

```
gainpcm (filename, gaindB)

         filename = Nombres del archivo pcm.
         gaindB   = Ganancia a aplicar (dB).
```

### *loadfrd*

Carga datos de respuesta en frecuencia desde un archivo **.frd**.

Se ignoran las líneas con texto no numérico.

```
[freq, magdB, pha] = loadfrd (filename)

         freq     =             Vector             de             frecuencias.
         magdB    =               Magnitud                en                dB.
         pha      =                                                      Fase.
         filename = nombre completo del archivo .frd.
```

### *mag2dB*

Pasa un vector de magnitud o potencia a decibelios .

```
b = mag2dB(a)

         b        = Decibelios.
         a        = Magnitud.
```

### *minexcphsp*

Obtiene el espectro de fase mínima y el pasatodo con el exceso de fase a partir de un espectro completo.

```
[minph,excph] = minexcphsp(sp)

         minph    = Espectro completo de fase mínima con la misma magnitud de
                    espectro que imp.
         excph    = Espectro completo pasatodo de exceso de fase.
         sp       = Espectro completo. Longitud par.
```

### *minphsp*

Obtiene el espectro de fase mínima a partir de un espectro completo.

```
minph = minphsp(sp)

          minph     = Espectro completo de fase mínima con la misma magnitud que
          sp.
          sp        = Espectro completo. Longitud par.
```

### *savepcm*

(Tomada de DRC-fir) Escribe archivos **.pcm**.

```
savepcm(pcm,fname)

          pcm       = Vector del impulso.
          fname     = Nombre de archivo .pcm.
```

### *semiblackman*

Obtiene la mitad derecha de una ventana Blackman de longitud ***m***.

```
w = semiblackman(m)

          w         = Ventana.
          m         = Número de muestras.
```

### *semisp*

Obtiene el espectro de las frecuencias positivas a partir de un espectro completo.

```
ssp = semisp(wsp)

          ssp       = Semiespectro entre 0 y m/2.
          wsp       = Espectro completo entre 0 y m-1 (m par).
```

### *trwcos*

Genera ventanas de transición complementarias en un intervalo dado, según una función "raised cosine" en escala de frecuencias logarítmica.

```
[trw1,trw2] = trwcos(n1,n2)

          trw1      = Ventana de transición de izquierda a derecha.
          trw2      = Ventana de transición de derecha a izquierda.
          n1        = Índice del extremo izquierdo de la ventana.
          n2        = Índice del extremo derecho de la ventana.
```

### *wholesplp*

Obtiene el espectro simétrico completo a partir del espectro de las frecuencias positivas.

```
wsp = wholesplp(ssp)

          wsp       = Espectro completo entre 0 y m-1 (m par).
          ssp       = Semiespectro entre 0 y m/2.
```

### *wholespmp*

Obtiene el espectro causal completo a partir del espectro de las frecuencias positivas.

```
wsp = wholespmp(ssp)
wsp       = Espectro completo entre 0 y m-1 (m par).
ssp       = Semiespectro entre 0 y m/2.
```