FIRtro: guía del usuario

la última versión en:

https://github.com/AudioHumLab/FIRtro

Contenido

Control de cambios

Fecha	Comentarios
2016-nov	Version inicial

Introducción

Qué es FIRtro.

FIRtro es un paquete de software de tipo 'black box' para filtrar y ecualizar altavoces activos, también para ecualización de sala (DRC). Dispone de funciones de preamplificador (control de volumen calibrable con Loudness ISO 226:2003 y selección de fuentes analógicas y digitales).

Para informarse y debatir sobre FIRtro se recomienda el foro www.che.es/uniforo.

Para instalar FIRtro por favor consultar la Guía de Instalación.

FIRtro es un proyecto de audio HUM, diseñado por Roberto Ripio y con la colaboración de otros aficionados en su estado actual. Su principal objetivo es beneficiarse de las técnicas digitales de procesamiento de señal (DSP) para el diseño y construcción de altavoces HUM.

El DSP se implementa en un PC o un microordenador a base de algortimos abiertos GPL. La infraestructura natural es un entorno Linux. RR proporciona unos algoritmos para DSP con FIRs de diseño propio (https://github.com/rripio/DSD). También se usan otras herramientas de audio disponibles con licencia GPL.

El sistema está basado principalmente en:

- El convolver de FIRs Brutefir (GPL https://www.ludd.ltu.se/~torger/brutefir.html).
- Para la selección de entradas y algunas facilidades avanzadas usa el servidor de audio digital Jack (GLP http://www.jackaudio.org/).
- Opcionalmente dispone de ecualización paramétrica (IIR) basada en el plugin filter del paquete fil-plugins (GPL http://kokkinizita.linuxaudio.org/linuxaudio), que es soportado en el host Ecasound (GPL http://nosignal.fi/ecasound/).

Como se ha comentado, FIRtro va ligado al software DSD (GPL https://github.com/rripio/DSD), que es una herramienta para **diseñar filtros FIR** para **cruce de vías (xover)** y para **EQ de la respuesta anecoica del altavoz**. Los filtros pueden ser de fase lineal o de fase mínima. La versión lp (lineal phase) consigue **compensar el group delay que ocurre en un altavoz de graves**, porque físicamente es un sistema pasa altos. Esta corrección proporciona un grave excepcional, coherente en el tiempo. Además se consiguen cruces de vía de altísima pendiente pudiendo extremar el rango de uso de los altavoces. Adicionalmente DSD también proporciona filtros FIR mp (minimun phase) que evitan el delay inherente a lineal phase. Los filtros para **correción de sala** (DRC) son minimun phase.

FIRtro también va ligado a MPD desde el inicio, que es software GPL (https://www.musicpd.org/) y es la mejor solución para **gestionar una biblioteca musical**. MPD es un diseño cliente / servidor. En general el servidor corre en la misma máquina que tiene los altavoces (el FIRtro en nuestro caso) y el cliente es la interfaz de usuario que puede estar en otra máquina (por ejemplo un portátil, una tablet, un smartphone).

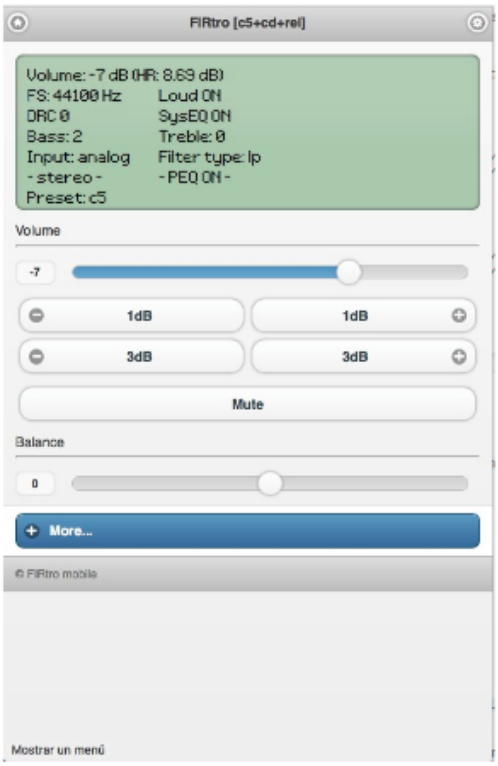
FIRtro es un sistema cliente / servidor. Un programa servidor se ocupa de manejar todos los subsistemas y un programa cliente se comunica con él pasándole comandos. La comunicación es por socket tcp/ip lo que facilita una localización distribuida.

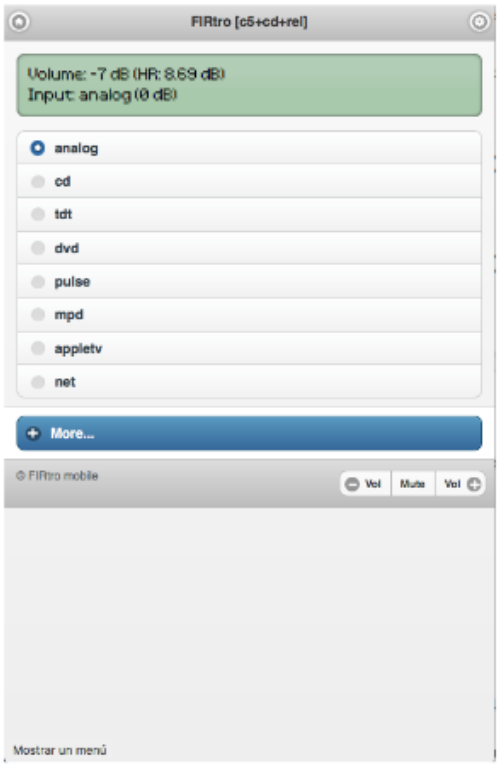
En cuanto a la interfaz del usuario de FIRtro:

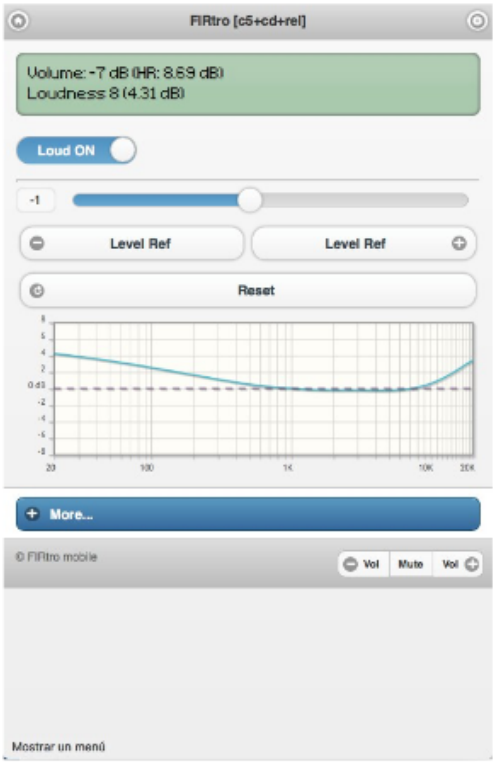
- Al tratarse de un proyecto HUM abierto, se sigue una filosofía “command line”, es decir todo se configura y se supervisa con comandos en un terminal.
- Los archivos de configuración son sencillos archivos de texto de tipo “.ini”.
- Los scripts que conforman el sistema FIRtro están escritos en Python.
- De cara al uso cotidiano, se dispone de herramientas gráficas y de un sistema de control **orientado a un usuario final** basado en una web cómodamente accesible desde cualquier navegador, por ejemplo un smartphone. También hay una interfaz basada en LIRC para mando a distancia IR (infrarojos).
- Como gestor de biblioteca de música, se ha elegido MPD por su universalidad. Cada usuario puede entonces usar su cliente mpd favorito, por ejemplo gmpc.

Ejemplo del cliente gmpc acceciendo a la biblioteca musical del servidor mpd:

La web de control:







Un poco de historia

FIRtro es un sistema inicialmente desarrollado por Roberto Ripio, aka RR en www.che.es/uniforo, y recientemente con la colaboración de otros aficionados al audio HUM. Destaca el diseño con espíritu HUM extremo: desde la modularización inicial del S.O. (Gentoo) para máquinas con CPU limitada hasta el diseño de algoritmos matemáticos en lenguaje Octave para la generación de los filtros de audio -paquete DSD (c)RR-.

El FIRtro original se implementó sobre Linux Slax y Linux Gentoo por su modularidad, pero actualmente disponemos de PC con potencia suficiente para usar por ejemplo Linux Debian o Raspbian para Raspberry Pi.

El nombre FIRtro es elegido conjuntamente por RR y mentero, foreros de www.che.es, con la siguiente idea inicial:

---------- Documento histórico fundacional: ----------

Sistema estéreo con posibilidad de utilización de FIR, IIR, ecualización, DCR, y básicamente cualquier cosa que se pueda hacer por convolución.
Programable desde un portátil externo, por conexión USB.

-Hard: Caja negra
- *Tarjeta base Pentium (?) y sin ventilador*
- *Fuente de alimentación*
- *Tarjeta de sonido en condiciones (tal vez con su propia fuente)*
- *Lector CD/DVD*

-Soft:
- *Instalación minima de Linux y programa FIRtro(c).*

Entradas
-USB
-RCA analógica estéreo
-SPDIF

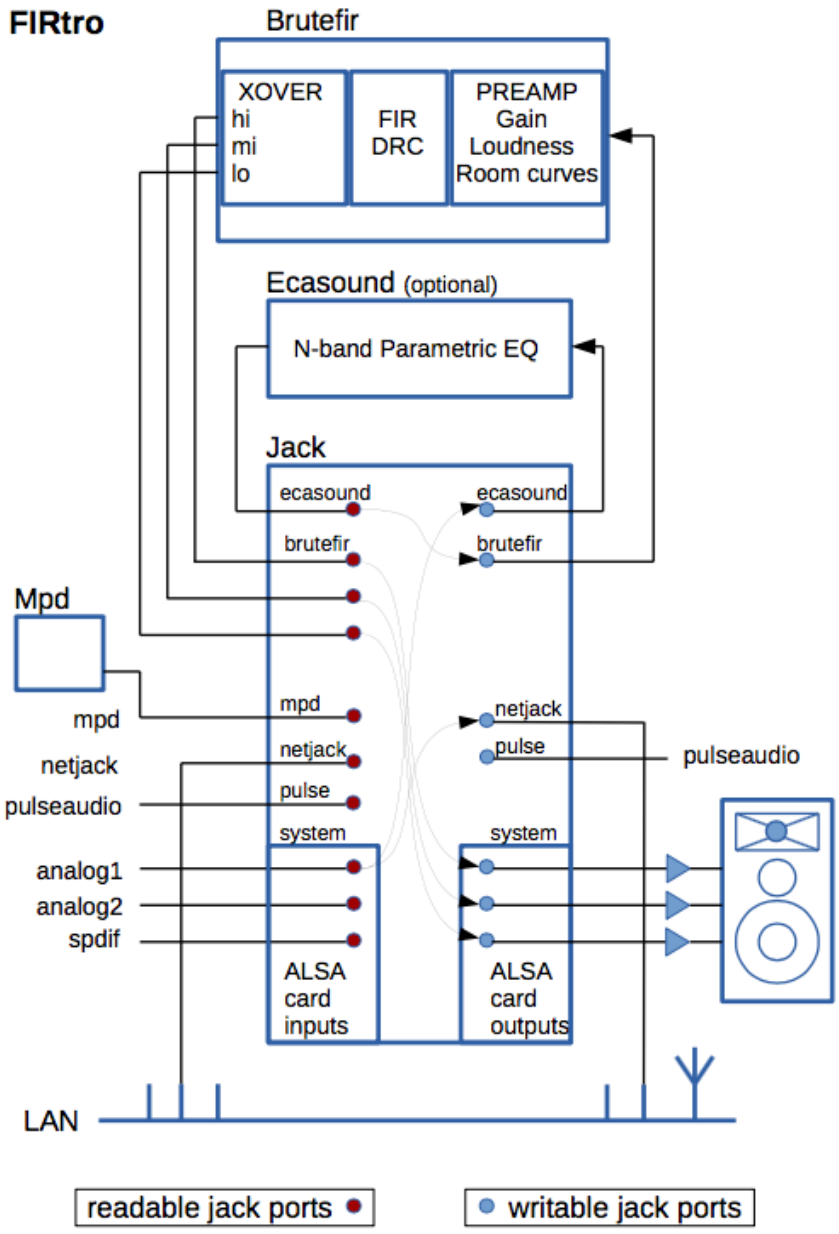
Salidas
-Ocho canales configurables.

Futuribles fuera del proyecto
Disco duro
Biblioteca de filtros bajables de la red.

FIRtro features:

- Community and user support (www.che.es/uniforo)
- Easy of install process ;-)
- Active loudspeaker crossover with bass group delay correction (lineal phase FIR based)
- Also for passive loudspeaker with bass group delay correction
- Lineal phase correction detachable to play video media w/o delay
- User configurable loudspeaker presets
- Volume control with iso 226:2003 loudness compensation and calibrated in room reference SPL.
- Room curves targets
- Digital Room Correcttion FIR and/or Parametric
- Parametric EQ module is Room EQ Wizard compatible
- Customizable FIR stages and length to adecuate available CPU power
- FIR / Parametric curves in a graph
- Preamp facilities: Mono/Stereo, Balance, Tone, Loudness
- External analog, spdif and network sources
- Internal CD, DVD playing
- Web remote control (a web server is integrated)
- Infrared remote control LIRC
- External LCD display optional
- Multiple sound cards I/O are supported
- Can integrate a DVB receiver to play DVB-Radio channels
- Can send & receive audio over a net (netjack, jacktrip, pulseaudio)
- Integrates on your own Desktop Linux PC, then play internet media via Pulseaudio
- Plays your Spotify Desktop app (via Pulseaudio)
- Runs on PC, Raspberry PI, etc ...
- Minimal operating system if needed, tailored for Bit-Perfect Audio Playback
- Works with any USB 2.0 DAC
- Music library management (local disk, usb disk or NAS) based on MPD standard.
- Supports all filetypes: FLAC, Alac, Aac, Vorbis, Mp3, DSD etc
- Custom playlist editing and creation (MPD)
- Airplay receiver playback
- Squeezeslave (SqueezeBox)
- DSD Over PCM support, for non Direct-DSD compatible hardware (as per MPD facilities)
- Mount CIFS and NFS Network Attached Storage.
- Installation is based on GitHub repository.

Diagrama funcional



Requisitos HW

Un PC compatible, preferiblemente “fanless” o con la utilidad `fancontrol` si fuera necesario, o un micro ordenador por ejemplo Raspberry Pi.

Disco duro es preferible SSD por la rapidez de encendido del sistema.

Un teclado, ratón y monitor son opcionales y solo son indispensables en el momento de instalación del S.O.

Tarjeta de sonido multicanal a elegir, compatible con ALSA (http://www.alsa-project.org/main/index.php/Matrix:Main).

Eligiremos la tarjeta de sonido con canales suficientes para manejar las vías de nuestro sistema de altavoces.

Para un altavoz pasivo basta con una tarjeta stereo de calidad, por ejemplo usb.

Se admite el uso de varias tarjetas de sonido si se desea más conectividad, serán remuestreadas para alcanzar la tarjeta principal del sistema en Jack (esto consume %CPU). ALSA proporciona las utilidades `alsa_in` y `alsa_out` para este propósito, pero aquí se prefire usar el software `zita-ajbridge` (ALSA to JACK bridge) proporcionado con licencia GPL por Fons Adriaensen (http://kokkinizita.linuxaudio.org/linuxaudio).

Algunas tarjetas (como ECHO Gina 3G) necesitan un FW específico para que los modulos del kernel funcionen, que hay que localizar al margen de lo que se proporciona de serie en el S.O.

Instalación de FIRtro

Ver **FIRtro – guía de instalación.pdf**

Confección de los filtros de los altavoces y de DRC

FIRtro se basa en filtros FIR. Para su confección disponemos de la herramienta DSD (c) Roberto Ripio (https://github.com/rripio/DSD). DSD necesita un entorno Octave operativo, fácilmente disponible en Linux y en otros SOs.

DSD proporciona filtros de cualquier longitud según la resolución en graves que se quiera conseguir, por ejemplo para reducir el retardo de grupo GD en graves se necesitan FIR largos 32K o más.

DSD proporciona a FIRtro juegos de **filtros lp y mp equivalentes** en su respuesta en frecuencia, al objeto de poder usar la versión mp (sin el delay intrínseco de lp) cuando queremos reproducir un programa audiovisual, por ejemplo una película. Basta pulsar un botón en el control de FIRtro.

DSD también proporciona versiones a distintas Fs si queremos disponer de un FIRtro con la posibilidad de trabajar a distintas "sample rate".

También podemos considerar algunas herramientas para confeccion de filtros:

Filtering Software	Driver Xover / EQ	DRC	FIR	IIR
DSD	yes / yes auto	yes	yes	no
rePhase	yes / yes man	yes	yes	no
Acourate	yes / yes	yes	yes	no
Room EQ Wizard	no / no	yes	yes	yes
DRC-fir de Denis Sbraigon	no / no	yes	yes	no

DSD

DSD proporciona FIRs en formato `.pcm` (raw pcm) utilizable por Brutefir. DSD ecualiza la respuesta del altavoz de forma automática a partir de una medida cuasi anecoica.

Room EQ Wizard

REW proporciona un archivo de filtros paramétricos de corrección de sala en formato ***.txt***, y si se desea también proporciona la IR de corrección de sala en formato (***.wav***).

FIRtro ofrece una utilidad para importar el ***.txt*** de paramétricos proporcionado por REW que será traducido a un archivo ***xxxx.peq*** de tipo "ini" utilizable por el módulo PEQ de FIRtro.

Si se desea importar la IR de corrección de sala (***.wav***) proporcionada por Room EQ Wizard, DSD proporciona un script Octave para convertirla a ***.pcm*** que será utilizable en el módulo Brutefir de FIRtro. También podemos recurrir a ***sox*** (la navaja suiza de los formatos de audio).

Traslado de los filtros confeccionados a la máquina FIRtro.

Es habitual que el PC de trabajo para confeccinoar filtros FIR o cualquier programa como Room EQ Wizard sea externo al FIRtro. Para volcar los filtros elaborados, podemos usar un pendrive, pero es más efectiva una herramienta sftp estándard como Filezilla, ya que FIRtro atiende conexiones ftp seguras (sftp).

Filtros digitales.

FIR

Los filtros empleados en FIRtro son FIR. En FIRtro los FIR están materializados en archivos `.pcm` que contienen una secuencia de “taps” que conforma una IR (Impulse Response) finita (de cierta longitud o “taps”).

La señal de entrada, muestreada y codificada también en PCM, será alterada (filtrada) con la secuencia FIR, por el mecanismo de convolución. La longitud de la secuencia (p.ej 32K o 64K) define la resolución en bajas frecuencias (graves) y la demanda de procesado %CPU.

El encargado de convolver la señal con los FIR es Brutefir.

IIR

También se dispone la posiblidad de usar filtros IIR (Infinite Impulse Response) en el bloque funcional PEQ (ecualizador paramétrico). Los filtros IIR responden a modelos analógicos. Los ajustes de un IIR se pueden anotar en un sencillo archivo de texto que da valor a cada parámetro como en el mundo analógico (p.ej.: frecuencia central Fo, ancho de banda BW ó Q, y ganancia G). Un filtro IIR se materializa en un cálculo matemático que puede hacer un chip especializado en DSP o un plugin DSP para un PC de uso general, como es el caso de FIRtro. Por su naturaleza infinita, no cabe implementar filtro IIR en una secuencia digital (un archivo binario) como el caso de los FIR.

El encargado de procesar la señal con un filtrado IIR es el plugin de audio *filter* del paquete *fil-plugins* de Fons Adriaensen *. El procesador de audio Ecasound se encarga de acoger este plugin (plugin host) e integrarlo en la cadena de audio gestionada por JACK.

> *: http://kokkinizita.linuxaudio.org/linuxaudio/index.html. Este plugin se ha elegido tras superar pruebas exigentes frente a otros plugins.

Los filtros de cruce de vías (xover)

El filtrado de las VÍAS del altavoz (también el filtrado DRC con FIRs) se basa en archivos ***.pcm*** localizados las subcarpetas para cada Fs que vayamos a usar, de un directorio dedicado al altavoz:

/home/firtro/lspk/mialtavoz/44100

En cada carpeta de Fs también está el archivo ***brutefir_config*** que entre otras cosas define el

'cableado' de cada una de las vías, es decir la correspondencia con las salidas de la tarjeta hacia los amplificadores. Atención: esto es muy delicado.

Los filtros de DRC

El filtrado de DRC basado en FIR también se basa en archivos ***.pcm*** localizados en subcarpetas.

Ver más abajo **Configuración de FIRtro**.

El EQ paramétrico

Opcionalmente podemos usar el EQ paramétrico, por ejemplo para hacer DRC.

El PEQ corre a acargo de Ecasound, que se controla por un puerto tcp/ip (telnet, netcat).

Para comodidad, el script **`peq_control.py`** es ejecutable desde la línea de comando, admite comandos que se explican más abajo en la sección **Comprobar Ecasound**.

PRESETS de usuario

Recientemente se ha incorporado a FIRtro presets de usuario, que permiten definir combinaciones de:

- **Vías a activar** y **coeffs* de filtrado** que deben cargarse en cada una de las vías.
- **Atenuación** y **retardo** que debe aplicarse a cada una de las vías.
- **Coeff* de drc** a usar en el preset, también se admite desactivar la etapa drc.
- **EQ paramétrico** (archivo ***xxx.peq***) que se quiere aplicar.
- Ajuste de **balance**.

* Nota: los coeff de filtrado seleccionables estarán cargados en la memoria de ejecución de Brutefir.

Ejemplos de uso:

- ✔ Preconfigurar distintas posiciones de las cajas o del punto de escucha.
- ✔ Activar o no el subwoofer cambiando incluso el filtrado.
- ✔ Ensayos para pruebas de filtrado de vías, de drc y/o de subwoofer.
- ✔ Activar distintos juegos de altavoces con filtrados específicos, con fines experimentales.

Definición de presets de usuario

Se definen en el archivo ***/home/firtro/lspk/mialtavoz/presets.ini*** (es un archivo de texto de estructura tipo “ini”).

Ver ejemplo en anexos.

Prueba del software base de audio

Los componentes básicos son JACK y BRUTEFIR. El usuario no configurará nada a nivel ALSA (la capa de drivers de tarjetas de sonido).

(i) Debemos tener presente el esquema funcional presentado al inicio de la guía.

JACK descubre automágicamente los canales disponibles de una tarjeta ALSA y abre los puertos de “capture” y “playback” correspondientes.

BRUTEFIR debe tener un mapeo I/O coherente de entradas y salidas de filtrado con los puertos disponibles en Jack.

Prueba del software usando el backend ficticio “dummy” de Jack

Para probar Jack y Brutefir, podemos abrir dos terminales y lanzar primero jack con los mismos parámetros del archivo audio/config “de fábrica”.

```
firtro@firtro:~ $ jackd -R -d dummy -C2 -P6 -r 44100 &
[1] 2420
firtro@firtro:~ $ jackdmp 1.9.10
Copyright 2001-2005 Paul Davis and others.
Copyright 2004-2014 Grame.
jackdmp comes with ABSOLUTELY NO WARRANTY
This is free software, and you are welcome to redistribute it
under certain conditions; see the file COPYING for details
JACK server starting in realtime mode with priority 10
self-connect-mode is "Don't restrict self connect requests"

firtro@firtro:~ $ jack_lsp  # con este comando vemos los puertos
system:capture_1
system:capture_2
system:playback_1
system:playback_2
system:playback_3
system:playback_4
system:playback_5
system:playback_6
```

Si no hay errores lanzamos Brutefir en el segundo terminal:

```
firtro@firtro:~ $ brutefir lspk/ejemplo3vias/44100/brutefir_config &
[2] 15752
firtro@firtro:~ $
BruteFIR v1.0m (November 2013)                                  (c) Anders
Torger

Internal resolution is 32 bit floating point.
SSE capability detected -- optimisation enabled.
Creating 4 FFTW plans of size 4096...finished.
```

```
Loading 21 coefficient sets...finished.
JACK I/O: Warning: JACK is not running with SCHED_FIFO or SCHED_RR
(realtime).
Realtime priorities are min = 4, usermax = 3, mid = 5 and max = 6.
Estimated CPU clock rate is 3292.468 MHz. CPU count is 4.
Filters in process 0: 0 1 2 3 4 5 6 7 8 9
Realtime priority 5 set for callback process (pid 15753)
Realtime priority 3 set for cli process (pid 15758)
Realtime priority 6 set for filter process (pid 15757)
Audio processing starts now

firtro@firtro:~$ jack_lsp -c
system:capture_1
system:capture_2
system:playback_1
   brutefir:hi_L
system:playback_2
   brutefir:hi_R
system:playback_3
   brutefir:mi_L
system:playback_4
   brutefir:mi_R
system:playback_5
   brutefir:lo_L
system:playback_6
   brutefir:lo_R
brutefir:input-0
brutefir:input-1
brutefir:hi_L
   system:playback_1
brutefir:hi_R
   system:playback_2
brutefir:mi_L
   system:playback_3
brutefir:mi_R
   system:playback_4
brutefir:lo_L
   system:playback_5
brutefir:lo_R
   system:playback_6
```

Este listado de puertos se corresponde con el presentado en el esquema funcional del inicio de esta guía.

Ver la sección Herramientas y troubleshooting para resolver posibles incidencias.

Podemos detener los procesos:

```
firtro@firtro:~$ killall brutefir
firtro@firtro:~$ killall jackd
```

Prueba del software con la tarjeta física (ALSA)

MUY IMPORTANTE: APAGAR LOS AMPLIFICADORES.

Ahora arrancaremos Jack usando nuestra tarjeta ALSA física:

```
jackd -R -d alsa -d hw:MiTarjeta -r 44100 &
```

Si no hay errores lanzamos Brutefir como hicimos arriba.

```
brutefir lspk/ejemplo3via/44100/brutefir_config &
```

NOTA: si MiTarjeta solo dispone de dos canales stereo, obtendremos un error. Pero podemos planear filtrar un altavoz pasivo usando la configuracón de ejemplo de 1 vía:

```
brutefir lspk/ejemplo1via/44100/brutefir_config &
```

Configuración de FIRtro

Preparar los filtros en la carpeta del altavoz

Con el paquete de instalación se entregan tres ejemplos de configuraciones típicas:

```
/home/firtro/lspk/ejemplo3vias/
/home/firtro/lspk/ejemplo2vias/
/home/firtro/lspk/ejemplo1via/
```

La carpeta del altavoz debe tener la siguiente estructura básica:

```
/home/firtro/lspk/mialtavoz/
                  └── 44100/
                      ├── drc-1/
                      ├── drc-2/ (... etc son opcionales)
```

Un ejemplo:

```
firtro@firtro:~$ tree lspk/mialtavoz/
lspk/mialtavoz/
├── 44100
│   ├── brutefir_config
│   ├── drc-1
│   │   ├── L escucha_3m.ini
│   │   ├── L escucha_3m.pcm
│   │   ├── R escucha_3m.ini
│   │   └── R escucha_3m.pcm
│   ├── drc-1-L.pcm (softlink --->  drc-1/L escucha_3m.pcm )
│   ├── drc-1-R.pcm (softlink --->  drc-1/R escucha_3m.pcm )
│   ├── drc-2
│   │   ├── L escucha_2m.ini
│   │   ├── L escucha_2m.pcm
│   │   ├── R escucha_2m.ini
│   │   └── R escucha_2m.pcm
│   ├── drc-2-L.pcm (softlink --->  drc-2/L escucha_2m.pcm )
│   ├── drc-2-R.pcm (softlink --->  drc-2/R escucha_2m.pcm )
...
... ...
│   ├── filters.ini
```

```
│   ├── lp-hi.ini
│   ├── lp-hi.pcm
│   ├── lp-lo.ini
│   └── lp-lo.pcm
├── rew
│   ├── L_REW_9pos_avg.txt
│   └── R_REW_9pos_avg.txt
├── parametrico1.peq
├── parametrico2.peq
├── parametrico2.peq
├── channels_peq.png
├── left_peq.png
├── right_peq.png
├── peqdump.txt
├── presets.ini
└── speaker
```

A continuación comentamos **resaltados** los archivos que debemos incorporar en la carpeta y subcarpetas.

Contenido de la subcarpeta *44100*:

La colección de archivos **`xxxx.pcm`** de **filtrado de cruce de vías** (o 'full range' para un sistema pasivo) para esta Fs=44100.

El archivo **`brutefir_config`**: es complejo y podemos generarlo con la herramienta descrita más adelante.

El archivo `filters.ini` es de uso interno, contiene la lista ordenada de todos los .pcm, es generado automáticamente por la herramienta referida. Sirve para referencia a la hora de confeccionar presets.ini. Muestra la ganancia de cada .pcm para comodidad.

Archivos **`xxx.pcm`** de **drc**. Con el fin de ordenar los archivos y de posibilitar un nombrado intuitivo, los .pcm de drc se han separado en subcarpetas. Una herramienta interna genera un ***symlink*** para compatibilidad con el *mecanismo original* de selección de filtros drc (comando `drc N`, ver la *Guia del servidor y comandos.pdf*). Ese mecanismo está cubierto actualmente con el *mecanismo de selección de presets de usuario* que se explica en secciones anteriores.

El nombre de los archivos .pcm de las carpetas de drc deben empezar por L o R para identificar el canal.

Contenido de la carpeta *rew*:

Esta carpeta es facultativa, resulta útil para descargar aquí los archivos de texto que definen un set de filtros paramétricos confeccionado con Room EQ Wizard.

Disponemos de la herramienta `rew2ini.py` para "traducir" estos archivos `.txt` a otro `.peq` (con estructura *INI)* adecuado para ser usado por el módulo PEQ de FIRtro.

Contenido de la carpeta del *altavoz*:

`presets.ini` es el archivo de presets de ususario (debemos confeccionarlo siguiendo el

ejemplo del anexo).

`xxx.peq` son archivos de ajustes de PEQ (EQ paramétrico) que estarán referidos en `presets.ini`. El nombre de estos archivos es facultativo, y podemos preparar tantos como queramos.

`speaker` este archivo contiene el calibrado de ganacia del altavoz para alcanzar el SPL de referencia en sala. También detalla los niveles de las curvas target de sala (room curves) que queremos aplicarle.

Aquí se encontrarán otros archivos como son las gráficas de los filtros (`.png`), o los volcados del ecualizador paramétrico (`peqdump.txt`).

NOTAS:

- ➔ La herramienta `brutefir_config.py` necesita que cada archivo de filtro `.pcm` se acompañe de otro `.ini` conteniendo la ganacia de ese filtro. Ese valor se trasladará al archivo de configuración del convolver Brutefir (`brutefir_config`). Ver más abajo los comentarios sobre la ganancia del los FIR. Esto es debido a que los archivos .pcm no pueden contener metadatos.
- ➔ Podemos tener tantos `.pcm` como deseemos, por ejemplo para evaluar distintos filtrados en el altavoz. La funcionalidad presets nos servirá para definir distintas combinaciones seleccionables al vuelo.

Configurar Brutefir: brutefir_config

Este archivo define:

- ✔ Los ajustes de procesamiento: Fs, profundidad de bits (32/64), longitud y particionado de los filtros, umbral de dB para *powersave*, avisos (run-time warnings).
- ✔ Los ajustes de I/O: mapeo de entradas y salidas, delays, dither.
- ✔ La colección *coeffs* que se podrán aplicar para filtrado.
- ✔ La estructura de filtrado (*filters*) con los correspondientes coeff de arranque.
- ✔ La atenuación y polaridad aplicada a cada salida (vía).

Podemos tomar como modelo uno de los ejemplos proporcionados en el paquete de instalación para confeccionar el archivo ***`brutefir_config`*** adecuado para nuestro altavoz.

Para comodidad, disponemos del script ***`brutefir_config.py`*** que permite generar de forma automática el archivo, en función de los archivos encontrados en la carpeta del altavoz. Ver los detalles en anexo.

(i) Los *coeff* cargados al arranque en las etapas *filter* pueden ser cambiados al vuelo.

Comentarios sobre la cadena de ganancias y el nivelado de vías.

Este asunto sería objeto de una guía de diseño de sistemas de altavoces activos que excede este texto. Haremos una simple reseña en lo que respecta a `/custom/brutefir.ini` y a `lspk/miAltavoz/brutefir_config`.

Son numerosos los posibles puntos de ajuste de ganacia a nivel de vía:

1. La ganacia de cada filtro FIR (`coeff`)
2. La de cada salida o vía de Brutefir (`filter`)
3. La de cada canal de la tarjeta de sonido (`alsamixer`)
4. Atenuadores fijos de linea antes del amplificador de cada vía.
5. La propia ganancia del amplificador de la vía.
6. Posible “lpad” antes de un tweeter.
7. La sensibilidad del propio driver.
8. La ganancia in room de un altavoz de graves (la influencia del espacio de radiación pared (2pi) +6dB, pared y suelo (pi) +12dB, o rincón (pi/2) +18dB.

Cuando diseñamos el firtro FIR de corte de una vía tomamos nota de la ganancia del firtro y la incorporamos en la sección `coeff` de Brutefir que carga dicho filtro, de manera que la convolución resulte en ganancia unitaria, ver comentarios más abajo.

Se recomienda que las salidas de Brutefir “vean” una ganacia SPL similar en todas las vías conectadas, es más que esta homogeneidad sea vista desde la propia tarjeta de sonido, dejando en ésta todos los canales a 0 dB para tener la mejor S/N posible de los DAC.

El equilibrio de ganancia SPL de todas las vías se debe conseguir jugando con los factores desde el punto 4 en adelante, aunque en la práctica no siempre es posible. El nivelado de vías lo verificaremos con medidas semianecoicas (exentas de la influencia de la sala).

Ajustes en Brutefir:

Podemos ajustar las salidas *`filter`* de Brutefir. El siguiente ejemplo corresponde al filtro de salida de un tweeter, en el que se ha necesitado atenuar 4.7 dB e invertir su polaridad.

```
filter "f_hi_L" {
        from_filters: "f_drc_L";
        to_outputs:   "hi_L"/4.7/-1;
        coeff:        "c_lp-hi3";
};
```

Los ajustes de atenuación y polaridad expuestos arriba se pueden configurar en el archivo `custom/brutefir.ini` para que la herramienta `brutefir_config.py` genere automáticamente el complejo archivo `brutefir_config` en la carpeta del altavoz.

Ajustes en los DAC de la tarjeta de sonido:

También podemos usar ganancias de los DAC distintas de 0 dB, las podemos definir en una sección INI del archivo `~/audio/cards.ini`, por ejemplo:

```
[M1010LT]
# Gains when FIRtro starts:
'DAC',0 = -3.00dB
'DAC',1 = -3.00dB
'DAC',2 = -2.50dB
'DAC',3 = -2.50dB
'DAC',4 = -4.30dB
'DAC',5 = -4.30dB
'DAC',6 = -0.00dB
'DAC',7 = -0.00dB
```

Comentarios sobre la ganacia de los FIR diseñados

En cuanto a los pcm de filtros FIR de cruce de vía (también los FIR de drc), éstos tienen una ganancia intrínseca que puede diferir de ser unitaria, dependiendo de cómo se hayan confeccionado. DSD proporciona ese valor. Ese valor debe trasladarse al parámetro `attenuation` de cada `coeff` declarado en `brutefir_config`, al objeto de que la convolución de ese `coeff` con la señal resulte de ganancia unitaria.

Podemos reflejar ese valor en un archivo .INI acompañando a cada archivo .PCM de filtrado para que la herramienta `brutefir_config.py` confeccione el archivo de configuración de brutefir automáticamente. Ejemplo:

```
lspk/miAltavoz/44100/lp-hi.pcm
lspk/miAltavoz/44100/lp-hi.ini
    [miscel]
    gain =  -9.8
```

Esto se refleja en `brutefir_config`:

```
coeff "c_lp-hi1" {
        filename: "/home/firtro/lspk/miAltavoz/44100/lp-hi.pcm";
        format: "FLOAT_LE";
        shared_mem: false;
        attenuation: -9.8;
};
```

Ganancia del PEQ (ecualizador paramétrico)

También debemos verificar la ganacia efectiva de un posible PEQ (eq paramétrico) que vayamos a usar. La ganacia resultante de la suma de varios filtros paramétricos puede exceder la unidad. Ello puede incurrir en clipping digital que será arrastrado a la entrada de Brutefir.

Podemos visualizar la curva resultante del ecualizador paramétrico del preset actual con el comando especial `peq_control.py PEQplot`.

Se recomienda someter a Ecasound (que es el host del plugin de filtros paramétricos), a un sweep de

rango completo de nivel -0.1 dBFS. Es una señal extrema que no ocurrirá nunca pero así nos aseguramos de la ganancia global que debemos ajustar en el PEQ para evitar clipping digital a cualquier frecuencia. Se puede discutir la conveniencia de ruido rosa comos señal de prueba.

Es fácil sintentizar la señal de pruebas por ejemplo con Audacity, guardándola en formato wav.

Usuarios avanzados: si nuestro FIRtro tiene escritorio también podemos reproducir directamente con Audacity sobre los puertos de entrada a Ecasound.

Para la prueba:

- Desconectar temporalmente los amplificadores.
- Reproducir la señal de prueba por ejemplo con MPD que la inyectará en los puertos ecasound de JACK.
- Medir posibles clips digitales a la salida de ecasound, con un medidor de peaks para JACK, por ejemplo:
  `ecasignalview -f:f32,2 jack,ecasound null`

Veremos algo así:

```
ecasignalview v20051112-10 (2.9.1) -- (C) K.Vehmanen, J.Cunningham
  Input/output: "jack,ecasound" => "null"
  Settings: f32,2 refresh=50ms bsize=128 avg-length=5000ms
-------------------------------------------------------------------------------
channel                                 avg-peak      max-peak   clipped
-------------------------------------------------------------------------------
Ch-01: ************************          0.62862       0.96344       0
Ch-02: ************************          0.58096       0.92078       0
Press spacebar to reset stats
```

(nota: es ilustrativo ver las conexiones de los puertos de jack con `jack_lsp -c` ó con `qjackctl` - requiere X, ver más abajo **Herramientras - Jack**)

Ajustaremos la ganancia global del PEQ en el archivo `lspk/miAltavoz/MiPeq.peq` que corresponda con el preset actual, hasta evitar clipping digital.

El comando `control peq_reload` se encarga de procesar el ajuste realizado en `MiPeq.peq`.

Ganacia total

Brutefir es “inmune” a clippng interno entre sus distintas etapas, ya que calcula en coma flotante. El clipping puede ocurrir en la salida, que no debe superar 0 dBFS.

Brutefir dispone de “peak warnings” visualizables en la consola del server de FIRtro, ver más abajo **Herramientas Brutefir.**

Y siguiendo el procedimiento de arriba, también podemos monitorizar la salida de Brutefir, por ejemplo para una configuración de 6 vías:

```
ecasignalview -f:f32,6 jack,brutefir null
```

NOTA: el script ecasignalview.py facilita la sintaxis:

```
ecasignalview.py brutefir  # monitoriza todas las vías
```

Dither

En FIRtro se puede aplicar dither en Brutefir y en Jack, a criterio del usuario. Puede ser deseable aplicar dither en la interfaf de Jack hacia Alsa si el driver de tarjeta requiere reducción de la profundidad de bits (por ejemplo tarjetas usb). Ver la documentación de cada software para ver cómo.

El archivo de estado (y de arranque del sistema)

/home/firtro/audio/status

Este archivo contiene parte de **la información con la que se iniciará FIRtro**:

```
[recording EQ]
treble = 0.0
bass = 0.0

[level]
replaygain_track = False
level = -26.0
headroom = 27.0
balance = 0

[general]
muted = False
polarity = +
fs = 44100
peq = c5
drc_eq = 0
filter_type = lp
preset = c5
clock = card

[loudness EQ]
loudness_track = True
loudness_ref = -10.0

[inputs]
radio_prev = 3
radio = 2
input = analog
resampled = no
mono = off
```

fs = xxxx establece la frecuencia de muestreo a la que se iniciará el sistema. En este caso se cargará la estructura correspondiente del directorio
/home/firtro/lspk/mialtavoz/44100

input = xxxx indica la entrada de inicio. Hace referencia a las entradas definidas en

/home/firtro/audio/inputs. En éste caso es la entrada analogica de la tarjeta, que normalmente aparece en Jack como ***system:capture_1*** y ***system:capture_2***.

preset = xxxx indica el preset de usuario con el que se configura el sistema al arranque.

El archivo de configuración de FIRtro

/home/firtro/audio/config

Nos centramos en tres líneas:

jack_options = -R -dalsa -p1024

ip_address = 192.168.1.xxx

loudspeaker = mialtavoz

- En la línea de jack options se deja sin especificar la frecuencia de muestreo (-r 44100), ni la tarjeta del sistema (-d hw:MiTarjeta,0) que serán incorporadaa por el script de arranque.
- Ajustar la línea de jack_options = ... con -S solo si usamos Jack2. También conviene ajustar `-p1024`, se puede experimentar con otros valores. Para tarjetas USB suele ir bien `-n3`. Para máquinas lentas como las primeras versiones de RPi podemos usar `-P16 -t2000` como opciones generales para limitar las posibles conexiones y aumentar el timeout de arranque, y poner `-P -o2` en las opciones para activar solo canales de salida estereo en un sistema sin xover (de altavoces pasivos o full range).

> Revisaremos el resto de líneas dejando las cargas opcionales "load_XXX" = False, salvo que nos interesen (True)

Usuarios avanzados: si hubiéramos compilado alguna versión especial de Jack, Brutefir, Mpd, etc hay que adecuar los ***path*** que definen la ubicación de los binarios ejecutables.

Primera prueba de arranque de FIRtro

IMPORTANTE: SE DEBEN APAGAR LOS AMPLIFICADORES.

Si todo va bien, en un terminal introducir la orden:

initfirtro.py all

Partimos de la base de tener ya la configuración deseada según se describe arriba.

(!) Atención:

Proceder con cautela, primero enchufar solo la vía de bajos, y cuando funcione, la de medios, y por último los tweeter. Se recomienda usar condensadores en los tweeters y medios, hasta asegurarse de que todo funciona.

¡Enhorabuena!

Tienes un sistema de filtrado activo de última generación con posibilidades impensables hasta ahora con las técnicas de filtrado tradicionales :-)

Control de FIRtro

La estructura cliente/servidor del sistema de control permite varias posibilidades:

Desde la línea de comandos

Desde una terminal local o remota (ssh), se pueden ordenar cambios al vuelo dialogando con el servidor

```
$ control "comando parametros"
```

Los comandos disponibles están descritos en la **Guía del servidor y de comandos.pdf**.

Web de control de FIRtro

Basta abrir en un navegador la url ***http://ipdelfirtro***

Control mediante infrarrojos (LIRC)

Es posible programar un mando a distancia convencional para recibir órdenes en un receptor de IR equipado en la máquina FIRtro, que serán traducidas a los comandos señalados arriba.

Visualización del estado de FIRtro

El comando ***status*** hará un volcado del archivo de estado.

La WEB de control muestra los ajustes del preamplificador y las gráficas de DRC.

Es posible equipar FIRtro con un display LCD, por ejemplo para presentar cíclicamente los ajustes, la pista reproducida por MPD, etc...

Autoarranque de FIRtro en el encendido

El sistema FIRtro debe autoarrancar al encendido de la máquina.

Máquina dedicada sin escritorio (black box)

Opción: rc.local

Editar `/etc/rc.local` insertando la orden:

```
su -l firtro -c "/home/firtro/bin/initfirtro.py all"
```

nota : `exit 0` debe ser la última línea de `rc.local`.

OjO es necesario revisar la policy para realtime del comando ***su***, editar `/etc/pam.d/su` y

añadir (o descomentar) esta linea:

```
session       required   pam_limits.so
```

Esto fuerza a aplicar los permisos realtime aunque no haya sesión iniciada.

Opción: init.d como servicio de sistema (demonio)

En la Guía de instalación se proporciona un anexo con el contenido del archivo ***/etc/init.d/firtro***

Lo editamos y lo hacemos ejecutable:

sudo chmod 755 /etc/init.d/firtro

Ahora lo podemos probar:

sudo service firtro start

Una vez comprobado que funciona, lo añadimos al boot:

sudo update-rc.d firtro defaults

Máquina con escritorio

Para autoarrancar FIRtro al inicio de la sesión de escritorio, consultar la documentación del gestor de escritorio de la máquina.

Por ejemplo para sistemas basados en Gnome se puede usar el cuadro de diálogo ***gnome-session-properties*** incluyendo una entrada llamada “FIRtro” que ejecute el comando “/home/firtro/bin/initfirtro.py all”.

Para (re)arrancar FIRtro manualmente desde una sesión de terminal usar el comando ***initfirtro.py all***. NOTA: previamente habremos añadido el path de ejecutables de FIRtro en nuestro `.bashrc` , como se indica al inicio de esta guía.

Herramientas y troubleshooting

(i) Se recomienda trabajar con varios terminales haciendo varias conexiones ssh desde un PC de trabajo, o con varios terminales abiertos en el escritorio del FIRtro si lo tuviera. Otra opción (menos cómoda) es usar las facilidades de múltiples consolas locales de Linux (Alt+F1, Alt+F2, etc).

Carga de la CPU, de la memoria, y supervisión de procesos

Con la utilidad **`htop`** de Linux podemos ver todos los procesos, seleccionar los de un usuario, filtrar algún proceso en el que queramos fijarnos especialmente, etc. La tecla de función [F1] nos ayuda a usar la herramienta.

El comando ***`free -h`*** de Linux nos informa del consumo de memoria RAM y del uso del swap en disco.

El comando ***`rti`*** de Brutefir también nos informará de la carga de su procesado. Se ha preparado un script bash (`~/bin/rti`) directamente ejecutalble desde un terminal.

Swap de memoria en disco

Brutefir recomienda desactivar el uso de swap. La instalación de FIRtro para Linux Debian lo deja desactivado.

> https://www.ludd.ltu.se/~torger/brutefir.html#config_1
>
> If there is any sound card used for input or output (or any other sample-clock dependent device), BruteFIR will automatically set its delay-sensitive processes to realtime priority, thus you will typically need to run the program as root. To maintain realtime performance, it is important that there is no memory belonging to the program in the swapfile, thus all memory must be locked to RAM. This is done if lock_memory is set to true. Note that the memory is never locked when realtime priority is not set (that is when there are only files used for input and output). Warning: there seems to be a bug in the Linux kernel which makes the shared memory to be locked one time for each process, meaning that when lock_memory is set to true, BruteFIR will seem to consume a lot more memory than it should. Also, it makes of course no sense to lock memory if your system does not have a swap activated. Due to this issue, the best thing to do is to have a system with no swap and avoid locking the memory.

El usuario avanzado por ejemplo de una máquina rápida con Desktop podrá evaluar si quiere activar la memoria “swap” de su sistema.

Ajustes de la tarjeta de sonido (ALSA)

ALSA es el sistema de audio de bajo nivel en Linux, maneja los driver de las tarjetas y ofrece funciones básicas de audio.

El comando ***cat /proc/asound/cards*** lista las tarjetas de sonido ALSA de la máquina.

Para ajustar los niveles y switches de la tarjeta:

```
alsamixer -D hw:MiTarjeta
```

- ✔ Llevar al máximo los sliders de alsamixer y delegar los ajustes niveles relativos de cada vía en ***brutefir_config***.
- ✔ OjO algunas tarjetas se quedan en mute [MM], presionar M en el slider correspondiente para quitar el mute [OO]. FIRtro se ocupa de cargar un ajuste del mixer alsa en cada arranque.
- ✔ Se recomienda atenuar la vía de agudos con un atenuador de línea fijo y pasivo (antes del amplificador). Luego un ajuste fino en brutefir.

FIRtro al arranque hace uso del módulo ***soundcards.py*** que asegura la correcta configuración del mixer ALSA de la(s) tarjeta(s). Previamente debe haberse guardado dicha configuración correcta en un archivo para cada tarjeta:

/home/firtro/audio/asound.MiTarjeta

Para ver y modificar los ajustes de la tarjeta

```
amixer -D hw:MiTarjeta argumentos...
```

Para salvar la configuración de una tarjeta de sonido:

alsactl --file /home/firtro/audio/asound.MiTarjeta store MiTarjeta

Ver la documentación de `amixer` con el comando `man amixer`. Este comando es de bajo nivel y da acceso a todas las funciones que permite el driver ALSA sobre la tarjeta. Es interesante por ejemplo para los ajustes de reloj.

Cada tarjeta es diferente y por tanto son diferentes los controles disponibles en amixer. En general, las tarjetas de uso doméstico disponen de controles como 'Master', 'Front', etc, y las tarjetas de tipo profesional tienen controles del tipo 'ADC', 'DAC', etc...

El script `soundcards.py` gestiona funciones comunes del `amixer` alsa y funciones específicas de tarjetas profesionales con posibilidad de configurar la referencia de reloj (interna /externa), como por ejemplo la M-Audio 1010LT.

El archivo `~/audio/cards.ini` es leído por `soundcards.py` y permite:

- ➢ Definir ajustes de ciertos `scontrol` del `amixer` de una tarjeta. Actualmente, en caso de ser la `system_card` estos ajustes serán establecidos al arranque de FIRtro. Esto es de utilidad para el nivelado de vías, como se describe en apartados anteriores.
- ➢ Declarar las tarjetas profesionales con posibilidad de configurar la referencia de reloj (interna/externa).

Ejemplo:

```
[proCards]
cards = M1010LT

[M1010LT]
# Gains when FIRtro starts:
'DAC',0 = -3.00dB
'DAC',1 = -4.50dB
'DAC',2 = -2.70dB
'DAC',3 = -3.00dB
'DAC',4 = -4.50dB
'DAC',5 = -2.70dB
'DAC',6 = -3.00dB
'DAC',7 = -3.00dB
```

Servidor de audio Pulseaudio

Pulseaudio es un sistema servidor de recursos de audio de amplio alcance, estandar de facto en Linux, principalmente en sistemas con escritorio. Normalmente es receptor de cualquier fuente de sonido que aparezca en el sistema, por ejemplo un navegador reproduciendo Youtube, un reproductor multimedia como VLC, sonidos del sistema (es recomendable desactivarlos), etc...

Pulseaudio accede por defecto a las tarjetas de sonido del sistema. Si queremos usar una tarjeta con el servidor de audio profesional JACK , deberemos inhibir su uso en Pulseadio, por ejemplo:

```
# Listado de tarjetas en Pulseaudio:
pactl list short cards
0 alsa_card.pci-0000_02_04.0 module-alsa-card.c
1 alsa_card.usb-miniDSP_miniStreamer-01 module-alsa-card.c

# Inhibimos el uso de la(s) tarjeta(s) que usará FIRtro:
pactl set-card-profile alsa_card.pci-0000_02_04.0 off
pactl set-card-profile alsa_card.usb-miniDSP_miniStreamer-01 off

# Lista de tarjetas ALSA:
cat /proc/asound/cards
 0 [PCH            ]: HDA-Intel - HDA Intel PCH
                      HDA Intel PCH at 0xfbff4000 irq 34
 1 [miniStreamer   ]: USB-Audio - miniStreamer
                      miniDSP miniStreamer at usb-0000:00:1a.0-1.2.3, full speed

# Arrancamos JACK sobre nuestra tarjeta ALSA
jackd -dalsa -dhw:PCH,0 -r44100 & (or whatever sample rate you need)
```

En este momento JACK está disponible para recibir/enviar audio a/desde cualquier aplicación compatible con Jack.

Si deseamos recoger en JACK los sonidos del Escritorio que seguirán apuntando a Pulseadio, navegador, players, etc... basta con activar el módulo que conecta Pulseadio con JACK:

```
# Instalamos el módulo de Pusleaudio a JACK:
sudo apt-get install pulseaudio-module-jack
# Carga el módulo y lo enlaza con JACK:
pactl load-module module-jack-sink channels=2 client_name=pulse_sink connect=False
```

```
# Configura a Pulseaudio para enrutar por defecto cualquier audio (p.ej Youtube) a Jack:
pacmd set-default-sink jack_out
```

Todo lo anterior es a título informativo y está configurado en el script de arranque de FIRtro.

Servidor de audio JACK

Jack es un sound server de baja latencia que proporciona conectividad de audio entre aplicaciones de procesado de audio y una interfaz de audio (tarjeta de sonido). Es un servidor orientado al audio profesional, estándar de facto en el mundo del audio Linux (http://jackaudio.org/).

Ver uso con el comando ***man jackd***.

Las utilidades de audio del ecosistema Linux vienen adaptadas para el uso de I/O vía Jack.

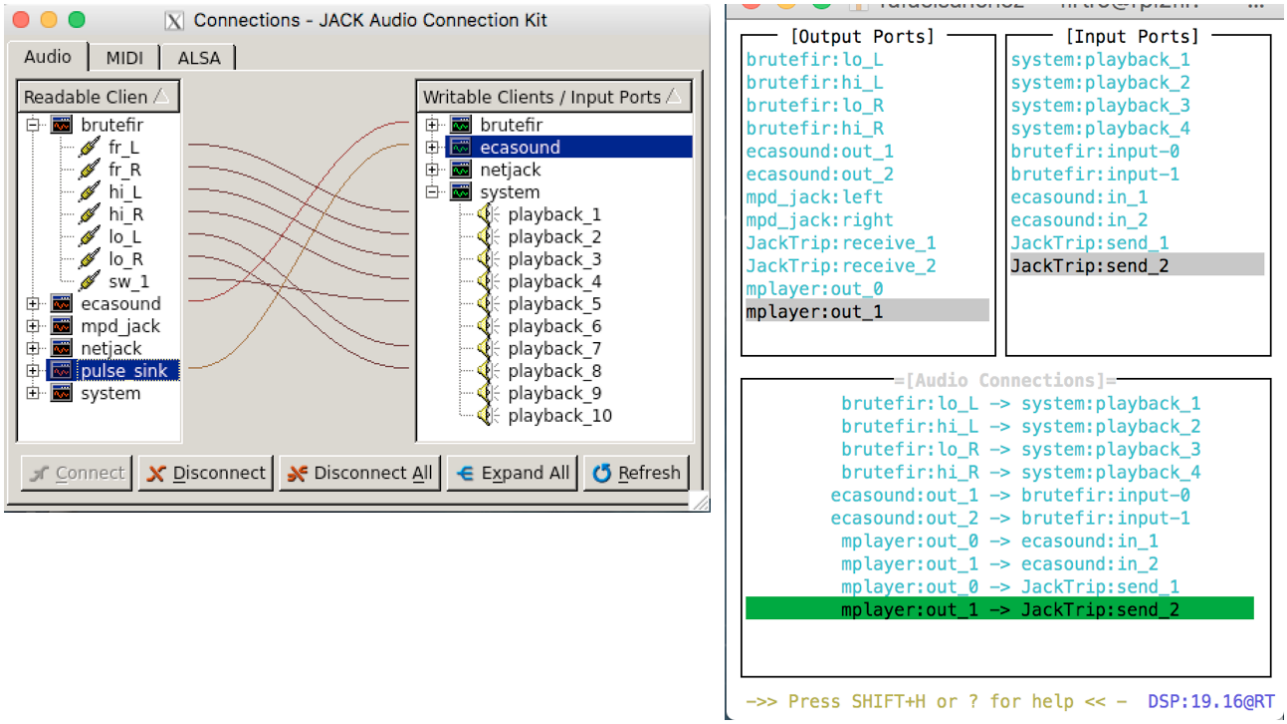
Jack dispone de comandos para hacer y listar conexiones entre sus puertos.

```
jack_connect     puerto_source puerto_playback
jack_disconnect  puerto_source puerto_playback
jack_lsp -c    # listado de puertos con su conexión
```

El script `jack_conexiones.py` muestra con flechitas las conexiones establecidas, bajo demanda.

Podemos operar dinámicamente sobre Jack, y ver cambios en las conexiones:

- En modo gráfico X windows con `qjackctl` Nota: necesitaremos un escritorio X local, o conectarnos al FIRtro con `ssh -X firtro@IPdeFIRtro`
- En un terminal de texto, con la herramienta `njconnect` (http://www.jackaudio.org/applications/). Para poder compilarla necesitaremos tener instalado el paquete `libncurses-dev`.



Brutefir, convolución, conceptos.

Brutefir es un convolver. Toma una señal de entrada y la procesa haciendo convoluciones, la salida puede resultar en varias señales, por ejemplo graves, medios y agudos.

Es totalmente abierto a configurar las etapas de convolución que necesitemos, tanto en serie (para EQs globales) como en paralelo (para cortes de vías).

En un convolver concurrente (multivía), la característica más relevante es la longitud de los filtros FIR .pcm. A mayor longitud, mayor resolución en graves pero mayor esfuero de %CPU.

Ajustaremos Brutefir a la capacidad de nuestra máquina:

- ✔ **Brutefir es el proceso que más carga la CPU del sistema**.
- ✔ Deberemos experimentar con las longitudes de los filtros y el consumo de CPU (real time index "`rti`"). Un punto de partida son longitudes de 32K taps.
- ✔ Brutefir permite el particionado de la convolución. Es una mejora del algoritmo interno que se traduce en una menor latencia de procesamiento entre la entrada y la salida de una muestra de señal. Esto es a costa de un mayor esfuerzo %CPU.

Cabe señalar que el uso de FIR de lp (lineal phase) introduce un rertardo ineludible en la señal debido a la correción temporal (de fase) que implementa. Este retardo no es causado por la CPU, es por la propia naturaleza de un filtro de fase lineal. Por contra un FIR de mp (minimum phase) no introduce retardo (solo la latencia del procesamiento). Tampoco introduce retardo un IIR, salvo la mínima latencia del cálculo DSP.

Las entidades principales en Brutefir son:

- **I/O Modules**. Usaremos su módulo JACK para definir una entrada y tantas salidas como necesitemos para nuestro altavoz multivía.
- **Coeffs**. Brutefir puede cargar un banco de coeffients en formato **.pcm** que podrán ser aplicados en las etapas de filtrado (convolución).
- **Filters**. Definen el mapa de routing del audio: desde dónde (from:) y hacia dónde (to:) aplicamos un coeff en convolución con la señal, para en definitiva filtrarla.
- **Logic Modules**. Dispone de módulos multipropósito:
  - ✔ **cli** (Command Line Interface) es el módulo que permite interactuar al vuelo con Brutefir (tcp/ip telnet)
  - ✔ **eq** (run-time equalizer) es un módulo que ofrece la funcionalidad de un EQ tradicional, pero controlable en amplitud y fase y con tantos “slider” como queramos. FIRtro lo usa para “dibujar” las curvas de tonos, de EQ de sala y de compensación loudness iso 226:2003.

Se recomienda familiarizarse con un archivo brutefir_config de FIRtro. Ver ejemplo anexo.

Ver más en http://www.ludd.luth.se/~torger/brutefir.html

Comprobar Brutefir

El script de FIRtro ***tb*** nos conecta con Brutefir (telnet) y podremos comprobar (y cambiar con precaución) su configuración al vuelo:

```
> tb
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.

Welcome to BruteFIR, type "help" for help.

> help
Commands:

lf -- list filters.
lc -- list coeffient sets.
li -- list inputs.
lo -- list outputs.
lm -- list modules.

cfoa -- change filter output attenuation.
        cfoa <filter> <output> <attenuation|Mmultiplier>
cfia -- change filter input attenuation.
        cfia <filter> <input> <attenuation|Mmultiplier>
cffa -- change filter filter-input attenuation.
        cffa <filter> <filter-input> <attenuation|Mmultiplier>
cfc  -- change filter coefficients.
        cfc <filter> <coeff>
cfd  -- change filter delay. (may truncate coeffs!)
        cfd <filter> <delay blocks>
cod  -- change output delay.
        cod <output> <delay> [<subdelay>]
cid  -- change input delay.
        cid <input> <delay> [<subdelay>]
tmo  -- toggle mute output.
```

```
        tmo <output>
    tmi  -- toggle mute input.
        tmi <input>
    imc  -- issue input module command.
        imc <index> <command>
    omc  -- issue output module command.
        omc <index> <command>
    lmc  -- issue logic module command.
        lmc <module> <command>

    sleep -- sleep for the given number of seconds [and ms], or blocks.
         sleep 10 (sleep 10 seconds).
         sleep b10 (sleep 10 blocks).
         sleep 0 300 (sleep 300 milliseconds).
    abort -- terminate immediately.
    tp    -- toggle prompt.
    ppk   -- print peak info, channels/samples/max dB.
    rpk   -- reset peak meters.
    upk   -- toggle print peak info on changes.
    rti   -- print current realtime index.
    quit  -- close connection.
    help  -- print this text.

    Notes:

    - When entering several commands on a single line,
      separate them with semicolons (;).
    - Inputs/outputs/filters can be given as index
numbers or as strings between quotes ("").
```

El comando **`rti`** del CLI de Brutefir muestra el **real time index**, que equivale al consumo de recursos de **CPU**.

Es conveniente revisar los filtros cargados antes de encender los amplificadores, para ello podemos leer la sección **`filters running`** que muestra el script **`read_brutefir_process.py`** (ver anexo). Este script también muestra el **mapeo de vías** y **las conexiones actuales desde Brutefir a la tarjeta de sonido**.

Los comandos **`lc`** y **`lf`** del CLI de Brutefir muestran información detallada de la polaridad y de la atenuación aplicada en cada etapa de filtrado (salidas).

El comando **`lo`** del CLI de Brutefir muestra los retardos en samples, en cada salida.

La polaridad y la atenuación de las etapas de filtrado de vía y los retardos en las salidas se pueden configurar en un archivo `custom/brutefir_xxx.ini` para generar automáticamente un archivo `brutefir_config`.

El ecualizador paramétrico

FIRtro dispone de un ecualizador paramétrico stereo de 12 bandas (se pueden adaptar más) por canal, proporcionado por el plugin *filter* del paquete *fil-plugins* de Fons Adriaensen *. El procesador de audio Ecasound sirve como host del referido plugin.

*: http://kokkinizita.linuxaudio.org/linuxaudio/index.html

El ecualizador paramétrico se configura en archivos de texto **`xxxx.peq`** (con estructura ".ini") en la carpeta del altavoz. El sistema dispone de archivos de configuración de Ecasound `~/audio/PEQxNN_defeat_FFFF.ecs` para cargar una configuración "plana" de NN filtros paramétricos por canal (FFFF es la Fs para Jack).

El comando **“control peq_reload”** carga el ecualizador con los ajustes del preset en curso.

El comando **“control peq_defeat”** pone “a cero” los ajustes el ecualizador.

El uso del PEQ es discreccional. Por ejemplo podemos indicar drc=off en un preset y conseguir el drc mediante los filtros paramétricos que ajustemos en un archivo `xxxx.peq`.

La opción `drc=` en el arhivo de prestes se refiere a la etapa drc FIR en Brutefir.

Comprobar Ecasound

Ecasound se controla por un puerto tcpip (telnet, netcat). Para comodidad, el script **`peq_control.py`** es ejecutable desde la línea de comando, admite como parámetros:

- Comandos especiales:

  `> peq_control.py PEQbypass` hace **bypass** en las chain left y right de ecasound.

  `> peq_control.py PEQgain nn` cambia la **ganacia** del ecualizador a nn dBs (el cambio no es persistente)

  `> peq_control.py PEQdump` **printa** la tabla de filtros paramétricos en curso en formato “ini”.

  `> peq_control.py PEQdump2ecs` printa la tabla de filtros paramétricos en curso, en formato .ecs (ecasound chain setup).

  `> peq_control.py PEQplot` muestra la **gráfica** dB/frecuencia de el EQ paramétrico en curso, también se guarda en archivos .PNG en la cerpeta del altavoz.

- Comandos EIAM (ver la man de *ecasound-iam*) que se lanzarán al servidor Ecasound, se mostrará la respuesta recibida. Por ejemplo, podemos comprobar los parámetros en ejecución:

```
> peq_control.py "cs-status" "aio-status"
```

Estado de FIRtro

El comando **`status`** printa el archivo de estado de FIRtro (`~/audio/status`)

La **consola de FIRtro** proporciona información del procesamiento de comandos y algunos parámetros del estado de FIRtro. Es muy útil visualizar la consola en un terminal rearrancando el server:

```
> pkill -f server.py; sleep 0.5; server.py &
```

La página web de control de FIRtro muestra los ajustes del preamplificador.

Frecuencia de trabajo del sistema FIRtro

El valor de `fs` en el archivo de estado determina la Fs a la que arranca FIRtro. El script de arranque `initfirtro.py` arrancará los programas de procesamiento de señal ajustados a dicha Fs (jackd,

brutefir, ecasound).

Cambio de la Fs del sistema

El cambio de la Fs del sistema puede establecerse:

- Manualmente mediante la edición del archivo de estado como se describe arriba, y la ejecucución del script `initfirtro.py audio`.
- Con el comando `control clock nuevaFs [nuevaRef]`.
- Con un comando `control input nuevaFuente`, para una fuente que especifique en `~audio/config` una fs distinta de la actual

Los comandos referidos implican el rearranque de los procesos jackd, brutefir y ecasound.

La opción `nuevaRef` es viable solo con tarjetas profesionales (ver apartado **Ajustes de la tarjeta de sonido**).

Funciones de preamplificador y otras

Preamplificador: gestión de las fuentes de sonido

FIRtro admite múltiples fuentes de señal:

- Fuentes locales externas: analógicas y digitales
- Fuentes locales internas: MPD, Pulseadio, CD, DVD-A, ¿SACD?
- Fuentes remotas: audio por la LAN.

La conmutación se realiza a nivel de puertos de captura en Jack. El usuario debe definir las fuentes en el archivo `/home/firtro/audio/inputs` , que tiene estructura de texto INI. Cada sección sirve para definir una posible fuente de señal. Las opciones admitidas son:

- **in_ports**: Los puertos “capture” de la fuente en Jack.
- **gain**: Ganancia en dB a añadir a la entrada, útil para equilibrar las fuentes analógicas.
- **xo**: [lp/mp] Tipo de crossover por defecto al conectar la entrada.
- **fs**: [44100|48000|…] Fs deseada (deben existir filtros en la carpeta del altavoz)
- **clock**: [card|word|spdif] Referencia para tarjetas con clock externo (ej:1010LT)
- **resampled**: [no|44100|48000|...] Si la entrada usa una tarjeta externa a jack “resampleada”.

La selección de una fuente de audio se consigue con la página web de control o mediante el comando `control input nombre_de_la_fuente.`

Servidor de la biblioteca musical del usuario.

FIRtro prevee la instalación del servidor musical MPD, que es la mejor solución para **gestionar una biblioteca musical**. MPD es un diseño cliente / servidor. En general el servidor corre en la misma máquina que tiene los altavoces (la de FIRtro en nuestro caso) y el cliente es la interfaz de usuario que puede estar en otra máquina (por ejemplo un portátil, una tablet, un smartphone).

Clientes típicos son mpc (command line), gmpc (GnomeMPC para cualquier escritorio Linux), Cantata (Linux/Windows), MPoD (iOS), MPDroid (Android), etc. Ver más en http://mpd.wikia.com/wiki/Clients.

El servidor MPD está enlazado a una biblioteca de música del usuario con archivos FLAC/MP3/etc. La biblioteca puede localizarse en red, en un disco usb o en un disco duro interno. MPD también puede usarse para escuchar streamings de Radio por internet, y las recientes versiones de MPD también permiten reproducir CDs de la unidad óptica instalada en la propia máquina si la hubiera.

Más información:

- Para configurar el servidor MPD ver la **Guia de Instalación**.
- La configuración del usuario que ejecuta MPD reside en **`~/.mpdconf`**
- Para ver las posibilidades de configuración ver la **`man mpd.conf`**.
- Para ver comandos típicos de un cliente ver la **`man mpc`**.
- Más información en https://www.musicpd.org.

Operativa:

La principal operativa es actualizar la biblioteca de música, este mecanismo hace que el servicio (demonio) `mpd` escanee los directorios de música que tenga declarados en **`~/.mpdconf.`** Esto se consigue con una función básica de cualquier cliente, por ejemplo con la orden **`mpc update`**. Esto puede tardar unos minutos si existieran miles de archivos musicales.

Futuros cambios (nuevos archivos o desaparición de archivos musicales) pueden ser automáticamente detectados si está habilitada la función `auto_update “yes”` en `.mpdconf`.

Recepcíon de DVB-T

Es posible instalar en la máquina un tarjeta DVB-T para recibir las emisoras de radio de nuestra región. El software `mplayer` se ocupa de manejar la tarjeta. Ver la **Guía de instalación mplayer CD DVB**.

FIRtro dispone de scripts para cambiar el canal sintonizado:

- `radio_channel.py`: canal seleccionable por n.º memorizado (`~/audio/radio`)
- `radio_channel_byName.py`: canal seleccionable por nombre de la emisora (`~/.mplayer/channels.conf`)

Audio por la LAN.

Algunas alternativas para enviar y recibir audio por la LAN:

- **Netjack1 o Netjack2**. La máquina master corre Jack sobre una tarjeta de sonido. Las máquinas slave corren Jack sobre el backend net mediante el cual envían y reciben audio con la master. Ejemplo: un portátil (slave) puede recibir o enviar audio con FIRtro.
- **JackTrip**. Permite habilitar puertos de audio por red en un servidor Jack que corre sobre tarjeta de sonido física. Por ejemplo para interconectar dos máquinas FIRtro.

  ALERTA: la versión actual de JackTrip en Linux Debian/Ubuntu hace que el extremo cliente se autoconecte a system:playback_X, esto peligroso en FIRtro ya que ahí están las vías de los altavoces activos sin atenuación. SE RECOMIENDA instalar la última versión (https://github.com/jcacerec/jacktrip/archive/master.zip) y ejecutar `jacktrip` con la opción `--nojackportsconnect` . Será necesario disponer en nuestro sistema de los paquetes `librtaudio-dev` y `qt4-default`.

- **Pulseaudio** (habitualmente en máquinas con escritorio). Permite enviar audio en broadacast RTP. Por ejemplo una máquina FIRtro con desktop autocarga el módulo module-rtp-recv, entonces si se recibe audio de una máquina remota, pulseaudio lo presentará en Jack de FIRtro como cualquier otra apliciación del escritorio. Para enviar audio, PA dispone del módulo module-rtp-send, y del módulo module-jack-source que presenta Jack como una fuente de sonido en PA.

Cabe destacar que con Netjack NO hay resampling, el mecanismo interno asegura que master y slave funcionan sincronizados. En el resto de soluciones mencionadas se hacer resampling, cada extremo tiene reloj autónomo.

Copia de seguridad de FIRtro

El script `backupfirtro.py` permite hacer una réplica (opción `-mirror`) de los archivos del sistema FIRtro en un soporte definido por el usuario, normalmente un medio externo USB. También permite generar una copia contendrá la configuración actual fechada, para poder recuperarla en un futuro (opción `-dated`)

Debemos definir los directorios de destino en el archivo `/custom/firtro.ini`. También podemos definir patrones que servirán para excluir arhivos o directorios a la hora de hacer la copia.

La manera más sencilla de conseguir un medio externo es “pinchar” un pendrive USB en la máquina FIRtro, que normalmente será automontado en `/media/sda1`.

Anexo. Ejemplo de archivo ~/lspk/mialtavoz/presets.ini

```
> cat lspk/c5+cd+rel/presets.ini

; Sintaxis:
;
; [preset_name]  nombre descriptivo
;
; --- Opción multivía:
; lo|mi|hi  = pcm_name*  attenuation**  delay**
; (idem para mi y hi)
;
; --- Opción full range:
; fr        = pcm_name*  attenuation**  delay**
;
; --- Opción subwoofer:
; sw        = pcm_name*  attenuation**  delay**
;
;             *   sin el prefijo de la fase ni la extensión
;                 en vías fr se admite "dirac_pulse" para no filtrar la vía
;             **  valores trasladados a la salida del filter de brutefir
;
; --- Ecualizador paramétrico PEQ
; peq       = nombre del archivo .peq (sin la extensión)
;             off
;
; --- Filtrado DRC:
; drc       = pcm_name sin el prefijo del canal ni la extensión
;             off
;

[c5]
; (i) nótese que aquí no hay SUBWOOFER
fr          = 0.0   0.0     dirac_pulse
peq         = c5
drc         = off
balance     = 0

[c5+sub]
fr          = 0.0   0.0     fr_oct2_80Hz_3thOrder
sw          = 0.0   0.0     sw_subRELterraza
peq         = c5
;drc         = RRreq C5
drc         = off
balance     = 0

[cd+sub(mueble)]
lo          = 9.0   0.0     lo_arp
hi          = 9.0   0.0     hi_celestion
;sw          = 7.0   7.3     sw_subRELterraza
sw          = 5.0   7.3     sw_subRELterraza
peq         = cd+sub(mueble)
;peq         = off
;drc         = RRreq CD mueble
drc         = off
balance     = 0

[cd+sub(front_1m)]
lo          = 9.0   0.0     lo_arp
hi          = 9.0   0.0     hi_celestion
sw          = 7.0   7.3     sw_subRELterraza
;peq         = CD+sub(front_1m)
peq         = off
```

```
drc         = RRreq CD front_1m
;drc         = off
balance     = 0
```

Anexo: información del script “read_brutefir_process.py”

```
$ read_brutefir_process.py


--- Outputs map:
"system:playback_1"/"fr_L"
"system:playback_2"/"fr_R"
"system:playback_3"/"hi_L"
"system:playback_4"/"hi_R"
"system:playback_7"/"lo_L"
"system:playback_8"/"lo_R"
"system:playback_5"/"sw_1"

--- Coeffs available:
              coeff#: coeff_name:     coeff_pcm_name:
               0      c_eq0           dirac pulse
               1      c_eq1           dirac pulse
               2      c_drc2_L        L RRreq FR_80Hz2nd_emula.pcm
               3      c_drc2_R        R RRreq FR_80Hz2nd_emula.pcm
               4      c_drc1_L        L RRreq C5.pcm
               5      c_drc1_R        R RRreq C5.pcm
               6      c_drc3_R        R RRreq CD mueble.pcm
               7      c_drc3_L        L RRreq CD mueble.pcm
               8      c_drc4_R        R RRreq CD front_1m.pcm
               9      c_drc4_L        L RRreq CD front_1m.pcm
              10      c_lp-fr1        lp-fr_oct2_80Hz_2ndOrd_EmulaOriginal.pcm
              11      c_lp-fr2        lp-fr_oct2_80Hz_3thOrder.pcm
              12      c_lp-hi3        lp-hi_celestion.pcm
              13      c_lp-lo4        lp-lo_arp.pcm
              14      c_lp-sw1        lp-sw_subRELterraza.pcm
              15      c_mp-fr1        mp-fr_oct2_80Hz_2ndOrd_EmulaOriginal.pcm
              16      c_mp-fr2        mp-fr_oct2_80Hz_3thOrder.pcm
              17      c_mp-hi3        mp-hi_celestion.pcm
              18      c_mp-lo4        mp-lo_arp.pcm
              19      c_mp-sw1        mp-sw_subRELterraza.pcm
              20      c_dirac-pulse   dirac pulse

--- Filters Running:
filter_name:  coeff#: coeff_name:     coeff_pcm_name:
f_eq_L         0      c_eq0           dirac pulse
f_eq_R         1      c_eq1           dirac pulse
f_drc_L       -1      (no filter)
f_drc_R       -1      (no filter)
f_fr_L        20      c_dirac-pulse   dirac pulse
f_hi_L        12      c_lp-hi3        lp-hi_celestion.pcm
f_lo_L        13      c_lp-lo4        lp-lo_arp.pcm
f_fr_R        20      c_dirac-pulse   dirac pulse
f_hi_R        12      c_lp-hi3        lp-hi_celestion.pcm
f_lo_R        13      c_lp-lo4        lp-lo_arp.pcm
f_sw_1        14      c_lp-sw1        lp-sw_subRELterraza.pcm

--- Jack:
brutefir:input-0               --<--  ecasound:out_1
brutefir:input-1               --<--  ecasound:out_2
brutefir:hi_L                  -->--  system:playback_3
brutefir:hi_R                  -->--  system:playback_4
brutefir:lo_L                  -->--  system:playback_7
brutefir:lo_R                  -->--  system:playback_8
brutefir:sw_1                  -->--  system:playback_5
```

Anexo: el script brutefir_config.py

El script de usuario `brutefir_config.py` permite generar un archivo `brutefir_config` en la carpeta del altavoz en función de:

- un archivo de configuración básica `~/custom/brutefir.ini` que especifica parámetros generales:
    - la longitud y particionado de filtros, si se usa dither, etc
    - el mapeo de entradas y salidas en JACK, así como la polaridad, atenuación y delays de las salidas.
- y de la relación de pcms de filtrado que se encuentre dentro de la carpeta del altavoz. Cada archivo `filtro.pcm` debe acompañarse de un archivo `filtro.ini` que contiene la ganancia del filtro, para ser tenida en cuenta a la hora de declarar los `coeff` en `brutefir_config`.

El script hace uso del módulo `scanfilters.py` que genera el archivo auxiliar `~/lspk/altavoz/Fs/filters.ini`, con la clasificación de filtros pcm y su ganancia que se ha encontrado en la carpeta del altavoz. Véase un ejemplo más abajo.

Este archivo `filters.ini` permitirá generar automáticamente el archvo `~/lspk/altavoz/Fs/brutefir_config`.

En función de los nombres de archivos de filtro encontrados, se construirá una estructura de filtrado multivía adecuada, o full range en su caso.

En caso de encontrar filtros "sw" para subwoofer se incluirá automáticamente una etapa de filtrado a partir de la mezcla ponderada de las señales L y R provinientes de la etapa de drc, la misma etapa que proporciona la señal a las etapas de filtrado de vías.

Anexo: ejemplo de archivo “brutefir_config”

Ejemplo de archivo **~/lspk/altavoz/44100/brutefir_config** generado con el script **brutefir_config.py**

```
# ------------------------------
# ------ GENERAL SETTINGS ------
# ------------------------------

sampling_rate:     44100;

filter_length:     2048,16;
float_bits:        32;

overflow_warnings: true;
allow_poll_mode:   false;
monitor_rate:      true;
powersave:         -80;
lock_memory:       true;
show_progress:     false;

# -------------------------
# --------  I/O -----------
# -------------------------

input "in_L", "in_R" {
        # Sin conexiones a priori en la entrada:
        device: "jack" { };
        sample: "AUTO";
        channels: 2/0,1;
};

output "fr_L", "fr_R", "hi_L", "hi_R", "lo_L", "lo_R", "sw_1" {
        # mapeo de las 7 salidas:
        device: "jack" { ports:
        "system:playback_1"/"fr_L", "system:playback_2"/"fr_R",
        "system:playback_3"/"hi_L", "system:playback_4"/"hi_R",
        "system:playback_7"/"lo_L", "system:playback_8"/"lo_R",
        "system:playback_5"/"sw_1";
        };
        sample:   "AUTO";
        channels: 7/0,1,2,3,4,5,6;
        maxdelay: 1000;
        dither:   true;
        delay:    0,0,0,0,0,0,321; # 'samples' that are equivalent in 'ms' to 0,0,0,0,0,0,7.3
};

# --------------------------------
# ----- EQ & LOUDNESS COEFFs -----
# --------------------------------

coeff "c_eq0" {
        filename: "dirac pulse";
        shared_mem: true;
        blocks: 1; # suficiente para hacer curvas de EQ suave
};
coeff "c_eq1" {
        filename: "dirac pulse";
        shared_mem: true;
        blocks: 1; # suficiente para hacer curvas de EQ suave
};

# --------------------------------
# ---------- DRC COEFFs ----------
# --------------------------------

coeff "c_drc1_L" {
        filename:    "/home/firtro/lspk/c5+cd+rel/44100/drc-1/L RRreq C5.pcm";
        format:      "FLOAT_LE";
        shared_mem:  false;
        attenuation: 0.0;
};
coeff "c_drc1_R" {
        filename:    "/home/firtro/lspk/c5+cd+rel/44100/drc-1/R RRreq C5.pcm";
        format:      "FLOAT_LE";
        shared_mem:  false;
        attenuation: 0.0;
};
coeff "c_drc3_L" {
```

```
        filename:    "/home/firtro/lspk/c5+cd+rel/44100/drc-3/L RRreq CD 3.1m.pcm";
        format:      "FLOAT_LE";
        shared_mem:  false;
        attenuation: 0.8;
};
coeff "c_drc3_R" {
        filename:    "/home/firtro/lspk/c5+cd+rel/44100/drc-3/R RRreq CD 3.1m.pcm";
        format:      "FLOAT_LE";
        shared_mem:  false;
        attenuation: 0.0;
};
coeff "c_drc2_R" {
        filename:    "/home/firtro/lspk/c5+cd+rel/44100/drc-2/R RRreq FR_80Hz2nd_emula.pcm";
        format:      "FLOAT_LE";
        shared_mem:  false;
        attenuation: 0.03;
};
coeff "c_drc2_L" {
        filename:    "/home/firtro/lspk/c5+cd+rel/44100/drc-2/L RRreq FR_80Hz2nd_emula.pcm";
        format:      "FLOAT_LE";
        shared_mem:  false;
        attenuation: 0.03;
};
coeff "c_drc4_L" {
        filename:    "/home/firtro/lspk/c5+cd+rel/44100/drc-4/L RRreq CD 3.1m BassOnly.pcm";
        format:      "FLOAT_LE";
        shared_mem:  false;
        attenuation: 0.0;
};
coeff "c_drc4_R" {
        filename:    "/home/firtro/lspk/c5+cd+rel/44100/drc-4/R RRreq CD 3.1m BassOnly.pcm";
        format:      "FLOAT_LE";
        shared_mem:  false;
        attenuation: 0.0;
};

# --------------------------------
# ---------- XO COEFFs -----------
# --------------------------------

coeff "c_lp-fr1" {
        filename:    "/home/firtro/lspk/c5+cd+rel/44100/lp-fr_oct2_80Hz_2ndOrd_EmulaOriginal.pcm";
        format:      "FLOAT_LE";
        shared_mem:  false;
        attenuation: -0.58;
};
coeff "c_lp-fr2" {
        filename:    "/home/firtro/lspk/c5+cd+rel/44100/lp-fr_oct2_80Hz_3thOrder.pcm";
        format:      "FLOAT_LE";
        shared_mem:  false;
        attenuation: -0.56;
};
coeff "c_lp-hi3" {
        filename:    "/home/firtro/lspk/c5+cd+rel/44100/lp-hi_celestion.pcm";
        format:      "FLOAT_LE";
        shared_mem:  false;
        attenuation: -15.8;
};
coeff "c_lp-lo4" {
        filename:    "/home/firtro/lspk/c5+cd+rel/44100/lp-lo_arp.pcm";
        format:      "FLOAT_LE";
        shared_mem:  false;
        attenuation: -22.63;
};
coeff "c_lp-sw1" {
        filename:    "/home/firtro/lspk/c5+cd+rel/44100/lp-sw_subRELterraza.pcm";
        format:      "FLOAT_LE";
        shared_mem:  false;
        attenuation: -17.52;
};
coeff "c_mp-fr1" {
        filename:    "/home/firtro/lspk/c5+cd+rel/44100/mp-fr_oct2_80Hz_2ndOrd_EmulaOriginal.pcm";
        format:      "FLOAT_LE";
        shared_mem:  false;
        attenuation: -0.58;
};
coeff "c_mp-fr2" {
        filename:    "/home/firtro/lspk/c5+cd+rel/44100/mp-fr_oct2_80Hz_3thOrder.pcm";
        format:      "FLOAT_LE";
        shared_mem:  false;
        attenuation: -0.56;
};
coeff "c_mp-hi3" {
        filename:    "/home/firtro/lspk/c5+cd+rel/44100/mp-hi_celestion.pcm";
        format:      "FLOAT_LE";
```

```
        shared_mem:  false;
        attenuation: -15.8;
};
coeff "c_mp-lo4" {
        filename:    "/home/firtro/lspk/c5+cd+rel/44100/mp-lo_arp.pcm";
        format:      "FLOAT_LE";
        shared_mem:  false;
        attenuation: -22.63;
};
coeff "c_mp-sw1" {
        filename:    "/home/firtro/lspk/c5+cd+rel/44100/mp-sw_subRELterraza.pcm";
        format:      "FLOAT_LE";
        shared_mem:  false;
        attenuation: -17.52;
};

# coeficiente comodín para vias full range sin filtrado
coeff "c_dirac-pulse" {
        filename:    "dirac pulse";
        format:      "FLOAT_LE";
        shared_mem:  false;
        attenuation: 0.0;
};

# ------------------------------
# --------  CONVOLVER   --------
# ------------------------------

# --- EQ filtering:

filter "f_eq_L" {
        from_inputs:  "in_L";
        to_filters:   "f_drc_L";
        coeff:        "c_eq0";
};
filter "f_eq_R" {
        from_inputs:  "in_R";
        to_filters:   "f_drc_R";
        coeff:        "c_eq1";
};

# --- DRC filtering:

filter "f_drc_L" {
        from_filters: "f_eq_L";
        to_filters:   "f_fr_L", "f_hi_L", "f_lo_L", "f_sw_1";
        coeff:        -1;
};
filter "f_drc_R" {
        from_filters: "f_eq_R";
        to_filters:   "f_fr_R", "f_hi_R", "f_lo_R", "f_sw_1";
        coeff:        -1;
};

# --- XOVER filtering:

filter "f_fr_L" {
        from_filters: "f_drc_L";
        to_outputs:   "fr_L"/0.0/1;
        coeff:        "c_lp-fr1";
};
filter "f_hi_L" {
        from_filters: "f_drc_L";
        to_outputs:   "hi_L"/14.0/-1;
        coeff:        "c_lp-hi3";
};
filter "f_lo_L" {
        from_filters: "f_drc_L";
        to_outputs:   "lo_L"/14.0/1;
        coeff:        "c_lp-lo4";
};
filter "f_fr_R" {
        from_filters: "f_drc_R";
        to_outputs:   "fr_R"/0.0/1;
        coeff:        "c_lp-fr1";
};
filter "f_hi_R" {
        from_filters: "f_drc_R";
        to_outputs:   "hi_R"/14.0/-1;
        coeff:        "c_lp-hi3";
};
filter "f_lo_R" {
        from_filters: "f_drc_R";
        to_outputs:   "lo_R"/14.0/1;
        coeff:        "c_lp-lo4";
```

```
};

# --- SUB filtering:

filter "f_sw_1" {
        from_filters: "f_drc_L"/3.0, "f_drc_R"/3.0;
        to_outputs:   "sw_1"/12.0/-1;
        coeff:        "c_lp-sw1";
};
```

Anexo: archivos de configuración del EQ paramétrico.

El ecualizador paramétrico se configura en archivos `xxxx.peq`. Son archivos de texto con estructura ".ini", en la carpeta del altavoz, ejemplo:

```
#    Active:     Frec:  BW(oct):   Gain:

[left]
global1  = 1                        -4.5
f1       = 1      54.4    0.1442    -5.7
f2       = 1       161    0.1442    -8.6
f3       = 1       260    0.1442   -13.2
f4       = 1       296    0.7140     8.2
global2  = 1                         0.0
f5       = 1       419    0.2287    -8.2
f6       = 1      1090    0.4490    -2.2
f7       = 1      3310    1.0097     2.2
f8       = 1      7320    0.5732     2.8
global3  = 1                         0.0
f9       = 0        10    1.0        0.0
f10      = 0        10    1.0        0.0
f11      = 0        10    1.0        0.0
f12      = 0        10    1.0        0.0

[right]
global1  = 1                        -4.5
f1       = 1      54.4    0.1442   -10.2
f2       = 1      87.7    0.1442    -8.6
f3       = 1       303    0.1442     9.0
f4       = 1       330    0.1442    -7.7
global2  = 1                         0.0
f5       = 1       419    0.1442    -4.5
f6       = 1       607    0.2287     3.3
f7       = 1      3110    1.0840     2.2
f8       = 1      7470    0.6502     2.6
global3  = 1                         0.0
f9       = 0        10    1.0        0.0
f10      = 0        10    1.0        0.0
f11      = 0        10    1.0        0.0
f12      = 0        10    1.0        0.0
```

Anexo: visualización de ajustes del EQ paramétrico.

Podemos consultar los ajustes en curso con el comando especial PEQdump:

```
> peq_control.py PEQdump

#   Active     Freq     BW    Gain

[left]
global1 =  1                    0.00
f1  =      1    47.40   0.08   -6.20
f2  =      1    50.60   0.19    4.00
f3  =      1    56.20   0.10   -9.90
f4  =      1   100.00   0.13   -4.60
global2 =  1                    0.00
f5  =      1   144.00   0.13   -6.60
f6  =      1   173.00   0.07   -4.70
f7  =      1   215.00   0.29   -3.50
f8  =      1   263.00   0.29   -1.70
global3 =  1                    0.00
f9  =      1   301.00   0.29    3.10
f10 =      1   534.00   0.48   -4.00
f11 =      0    10.00   1.00    0.00
f12 =      0    10.00   1.00    0.00

[right]
global1 =  1                    0.00
f1  =      1    56.60   0.16  -14.40
f2  =      1    88.20   0.17   -6.00
f3  =      1   100.00   0.16   -4.40
f4  =      1   129.00   0.03   -5.60
global2 =  1                    0.00
f5  =      1   144.00   0.05   -4.90
f6  =      1   174.00   0.06   -7.10
f7  =      1   253.00   0.29   -4.30
f8  =      1   272.00   0.29    4.00
global3 =  1                    0.00
f9  =      1   323.00   0.29   -3.80
f10 =      1   574.00   0.85   -4.20
f11 =      1  1356.00   0.29   -2.80
f12 =      0    10.00   1.00    0.00
```

Anexo: gráficas del EQ paramétrico.

Podemos visualizar las curvas aplicadas en curso con el comando especial PEQplot:

```
> peq_control.py PEQplot
```