HamFilter

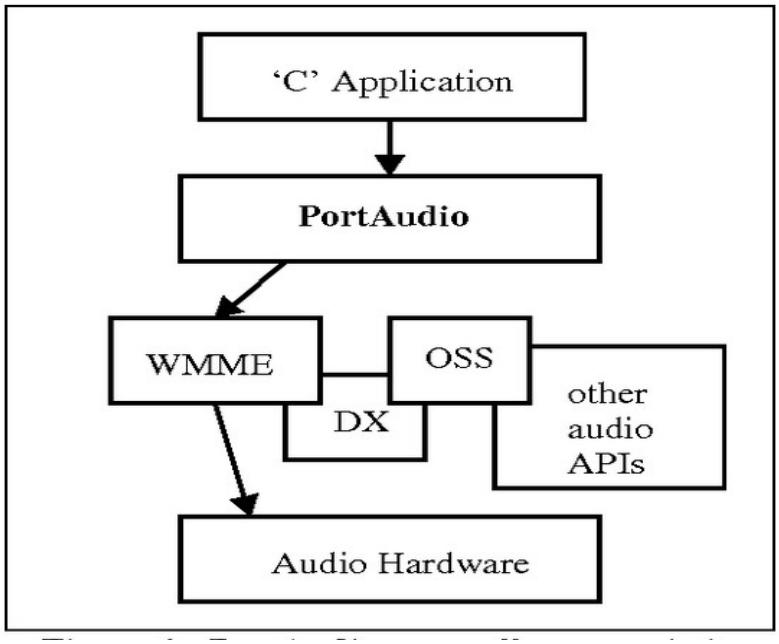
Già, si trovano programmi simili cercando un po' in rete ma allora, perchè crearne un altro ?
La risposta è non lo so. Stavo sperimentando con C, PortAudio, fft, filtri fir e questo è il risultato di qualche prova fatta pasticciando con gli stessi.

In realtà cercavo dei filtri CW per la mia vecchia radio che diventati ormai 'rari' costano quasi quanto la radio stessa. Non avendo molta possibilità di manovra e non volendo modificare la radio, il ripiego è filtrare in BF, come del resto facevano i vari filtri Datong e MFJ in uso anni fà.

Con molta fantasia, il nome del programma è HamFilter, che dovrebbe girare in tutte le versioni recenti di windows (da xp in poi). Scrivo dovrebbe perchè non ho avuto la possibilità di provarlo con tute le versioni (è stato sviluppato in Windows 7 e provato con XP e 10).

Allora, dicevo che stavo sperimentando come acquisire e manipolare flussi di dati audio in Windows. Windows mette a disposizione alcune API a livelli diversi che possono essere usate dagli sviluppatori per fare questo tipo di cose.
API è l'acronimo di Application Programming Interface; sono interfacce verso windows utilizzabili da un programma. In pratica, permettono ad un programma di 'dialogare' con windows e scambiare informazioni e dati con lo stesso; ad esempio scrivere/leggere files, leggere/scrivere dati da una porta seriale etc etc.
.
Ma, anzichè usare direttamente le API messe a disposizione da Windows, ho utilizzato una libreria Open Source che si chiama PortAudio la quale semplifica lo scambio dei flussi di dati audio tra il programma e Windows, rendendo relativamente più semplice il colloquio tra i due dal punto di vista dello sviluppatore.

PortAudio è una libreria di programmi per la riproduzione e la registrazione audio.
È una libreria multi piattaforma, quindi i programmi che la utilizzano possono essere eseguiti su molti sistemi operativi diversi , inclusi Windows MacOS e Linux .
In dettaglio PortAudio supporta le API fornite via Core Audio, MME, ALSA, ASIO, WASAPI su Windows.



Con riferimento alla figura precedente, PortAudio diventa una specie di ponte tra un programma 'C' ed il sistema operativo, mascherando allo sviluppatore la complessità delle api del sistema operativo stesso. Lo sviluppatore si deve focalizzare sul solo colloquio con PortAudio , anziché conoscere tutte le API messe a disposizione dai vari sistemi operativi che sono in genere complesse ed incompatibili tra loro.

Per approfondimenti e ulteriori informazioni fate riferimento al sito http://www.portaudio.com/

Parlando di audio, la qualità e quantità dei dati che dobbiamo manipolare dipende a grandi linee da tre cose: la frequenza di campionamento, la dimensione dei buffer che contengono i dati stessi, e la velocità a cui sono vengono a disposizione.

Un campionatore (convertitore analogico/digitale) è un circuito che a ritmo costante preleva dei campioni (ogni campione è pari all'ampiezza del segnale nell'istante in cui il campione viene prelevato) dal segnale analogico in esame. Il ritmo costante viene identificato con una frequenza detta frequenza di campionamento. Ogni campione memorizzato rappresenta l'ampiezza del segnale originario ad un determinato istante; questi campioni vengono memorizzati in buffers che PortAudio 'passa' al programma in tempi stabiliti.

Da ricordare che il teorema di Nyquist, sancisce che, se il campionamento viene eseguito ad una frequenza pari almeno al doppio della banda del segnale che si sta campionando il passaggio dall'analogico al digitale avviene senza perdita di informazione.

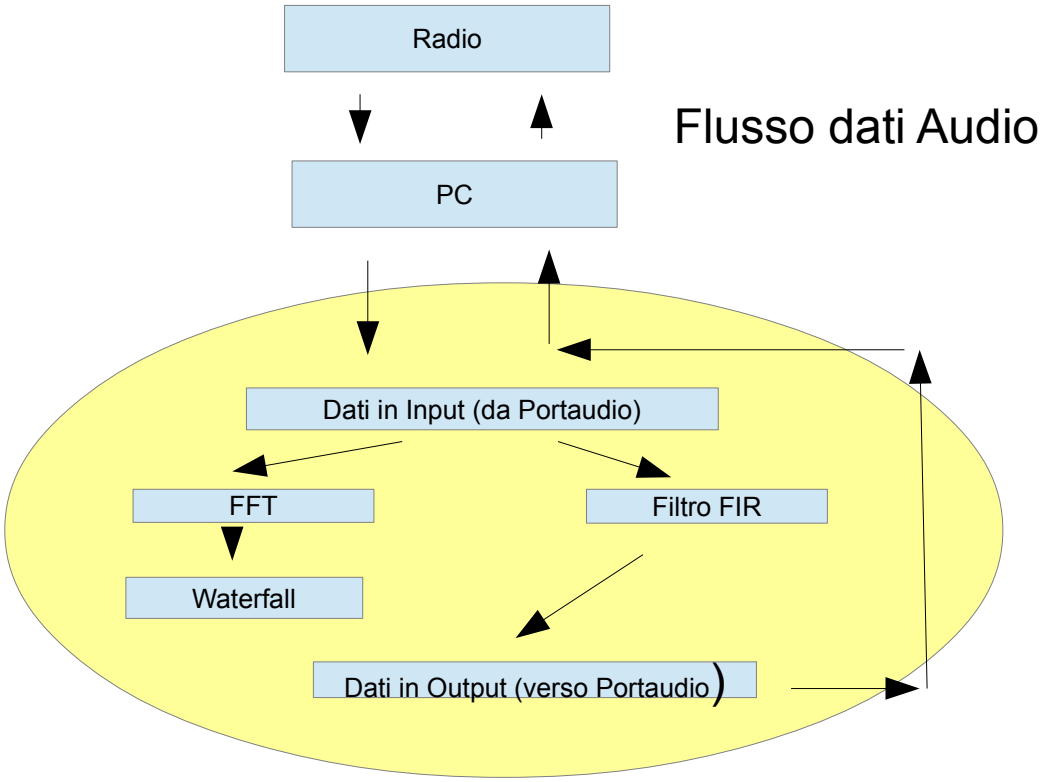
Nella implementazione di HamFilter, è stata scelta una frequenza di campionamento di 10240 Hz. e quindi in accordo con Nyquist la frequenza massima del segnale campionato sarà di 5120 Hz (frequenza di campionamento diviso 2) che 'copre' perfettamente la banda audio di nostro interesse che va più o meno da 300 a 3300 Hz per un canale modulato in SSB.

La dimensione del buffer che contiene i dati campionati che PortAudio si occupa di fornirci ha una lunghezza di 1024 campioni, nella nosta implementazione.
Quindi, ogni decimo di secondo circa avremo a disposizione un buffer contenente i dati campionati relativi al precedente decimo di secondo o in altre parole i dati che vediamo hanno un latenza di 1/10 di secondo.

A questo punto, quando PortAudio ha riempito il buffer con i dati audio questi vengono analizzati e processati dal programma seguendo due direzioni diverse.

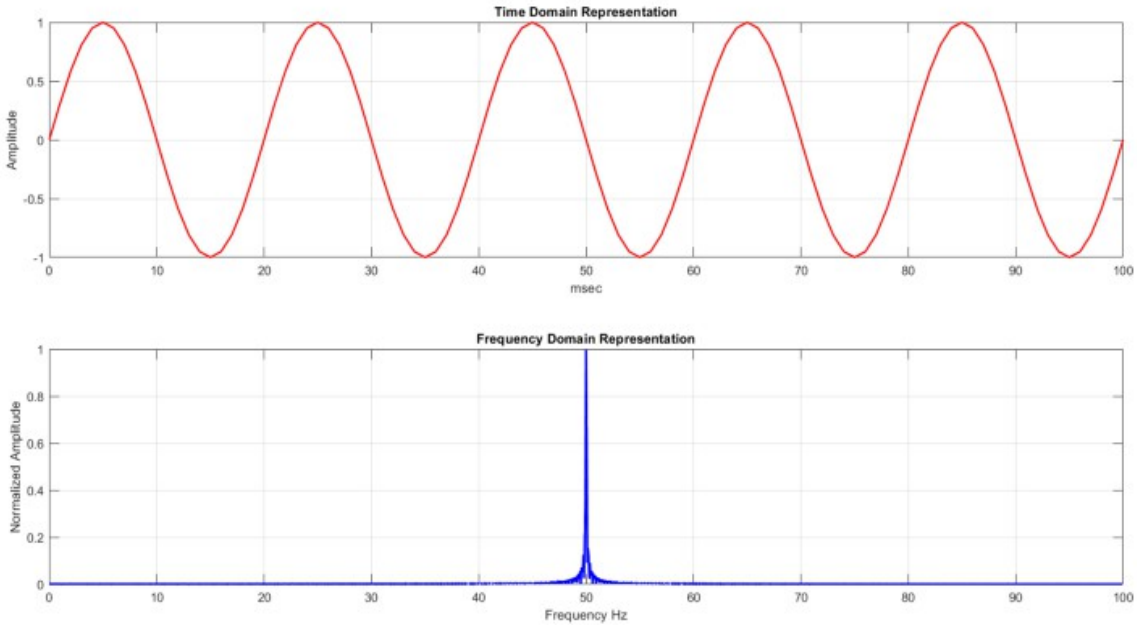
La prima, applica ai dati un algoritmo di FFT (Fast Fourier Transform) per avere una visualizzazione del segnale nel dominio delle frequenze (waterfall).

La seconda applica ai dati dei filtri FIR (Finite Impulse Response) tarati per le frequenze e banda passante di nostro interesse.



Il teorema di Fourier ci dice che una qualsiasi forma d'onda può essere approssimata sommando tra loro segnali sinusoidali di frequenza ed ampiezza opportuna.
Con un procedimento matematico, (FFT) si vanno a scoprire quali sono queste componenti, quale frequenza e quale ampiezza hanno; i dati vengo traslati dal dominio del tempo al dominio delle frequenze.

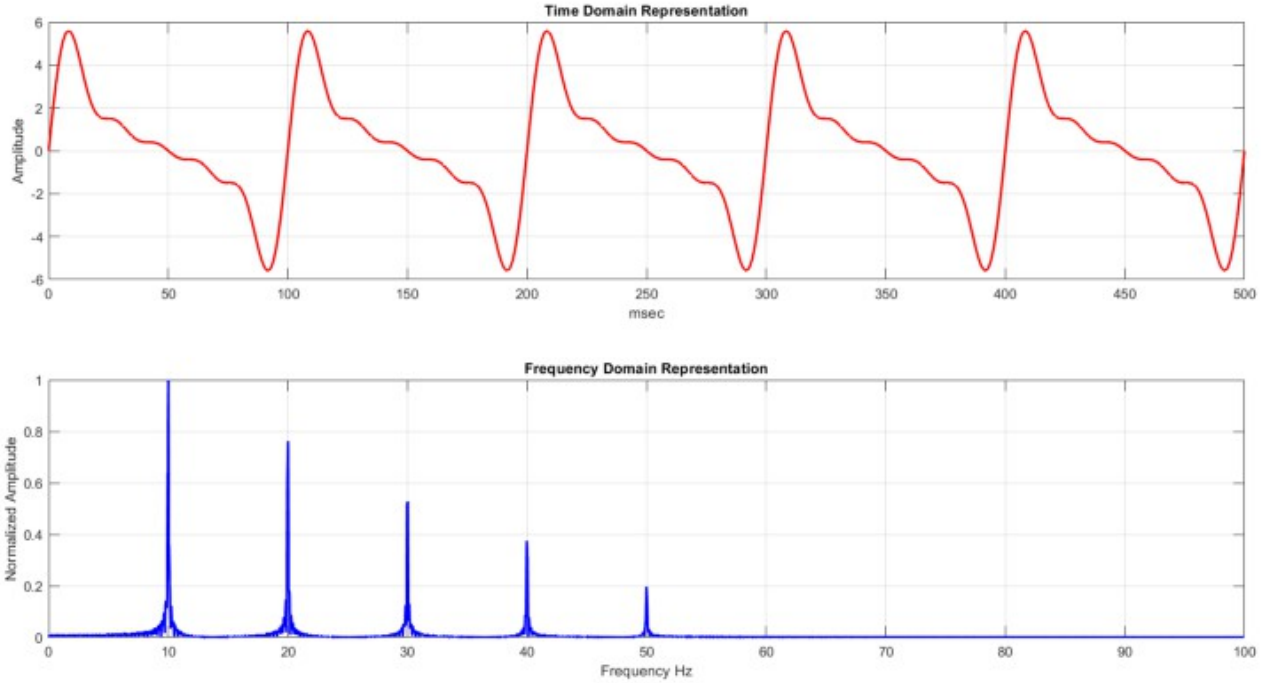
Con riferimento alla figura che segue, lo stesso segnale audio con forma sinusoidale e una frequenza di 50 Hz viene rappresentato nel dominio del tempo (in rosso)
e nel dominio delle frequenze (in blu).



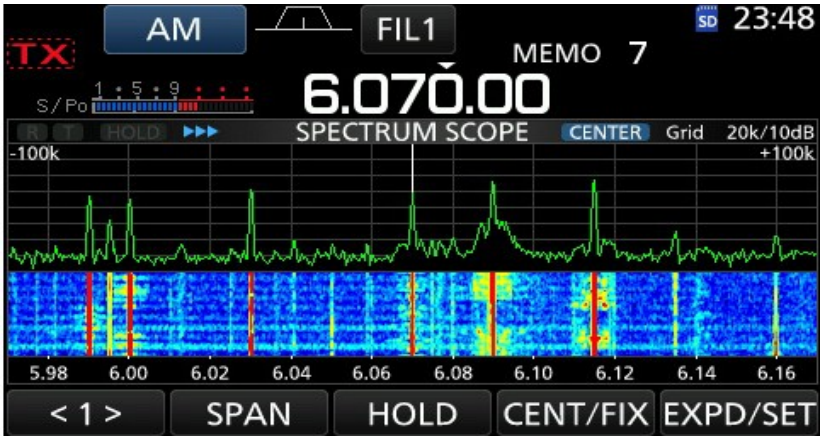
Nel dominio del tempo è visualizzato l'andamento dell'ampiezza del segnale nel tempo.
(Ampiezza del segnale nelle ordinate del grafico e tempo nelle ascisse)

Nel dominio delle frequenze viene mostrata l'ampiezza del segnale ad una determinata frequenza.
(Ampiezza del segnale nelle ordinate e frequenze nelle ascisse)

Se il segnale in esame fosse un po' piu complesso, composto ad esempio da 5 onde sinusoidali, FFT ci fornirà un output un po' diverso.
Con riferimento alla figura che segue vedremo che la forma d'onda in rosso è composta (si può approssimare) da 5 onde sinusoidali di frequenza 10, 20, 30, 40 e 50 hz con ampiezza rispettivamente di 1, 0,8, 0,4, 0,2.



Lo spettro ed il waterfall del segnale che vediamo sugli schermi delle nostra radio non sono altro che il risultato di una manipolazione dei dati (via FFT) raccolti (captati) dalle radio stesse e visualizzati nel dominio delle frequenze.



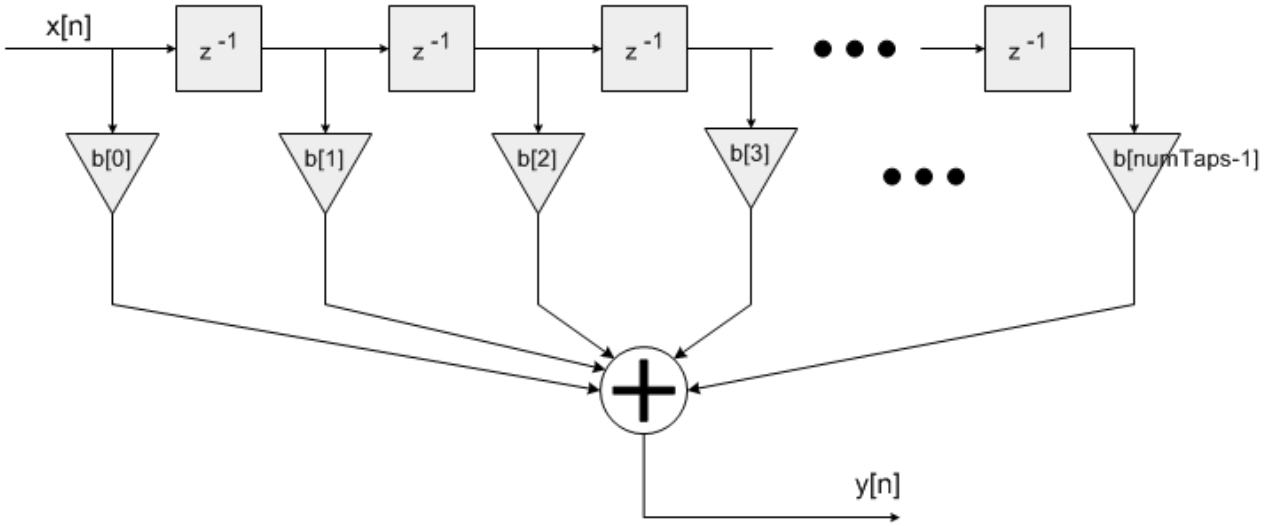
Per la implementazione in HamFilter della parte di programma che riguarda l'uso del FFT ho utilizzato un'altra libreria open source che ha nome FFTW.

Per approfondimenti sulla stessa → https://www.fftw.org/

La seconda direzione dei dati riguarda il loro filtraggio che viene fatta utilizzando un filtro FIR (Finite Impulse Response). Sono filtri digitali usati nelle moderne tecniche di Digital Signal Processing (DSP).

A grandi linee, il segnale in ingresso da filtrare viene moltiplicato/sommato con dei coefficienti; questi ultimi disegnano la curva di risposta del filtro e altre sue caratteristiche (fase etc).
Calcolando opportunamente questi coefficienti si possono realizzare via software filtri passa basso, passa alto, passa banda, notch etc...

```
y[n] = b[0] * x[n] + b[1] * x[n-1] + b[2] * x[n-2] + ...+ b[numTaps-1] * x[n-numTaps+1]
```

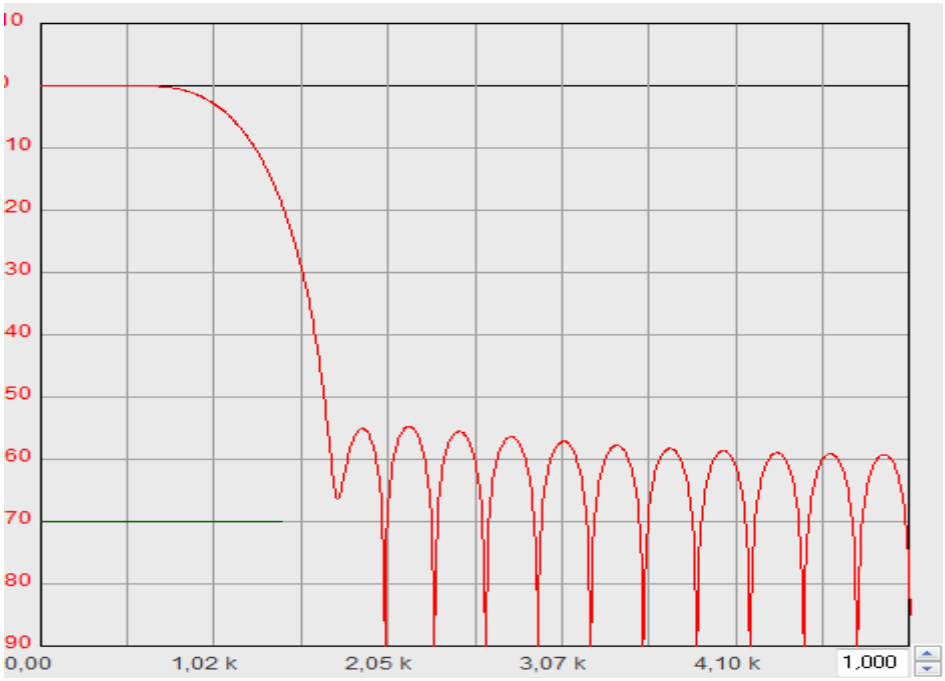


Per il calcolo dei coefficienti esistono diversi programmi anche online che permettono di disegnare e simulare la curva di risposta del filtro ad esempio → http://t-filter.engineerjs.com

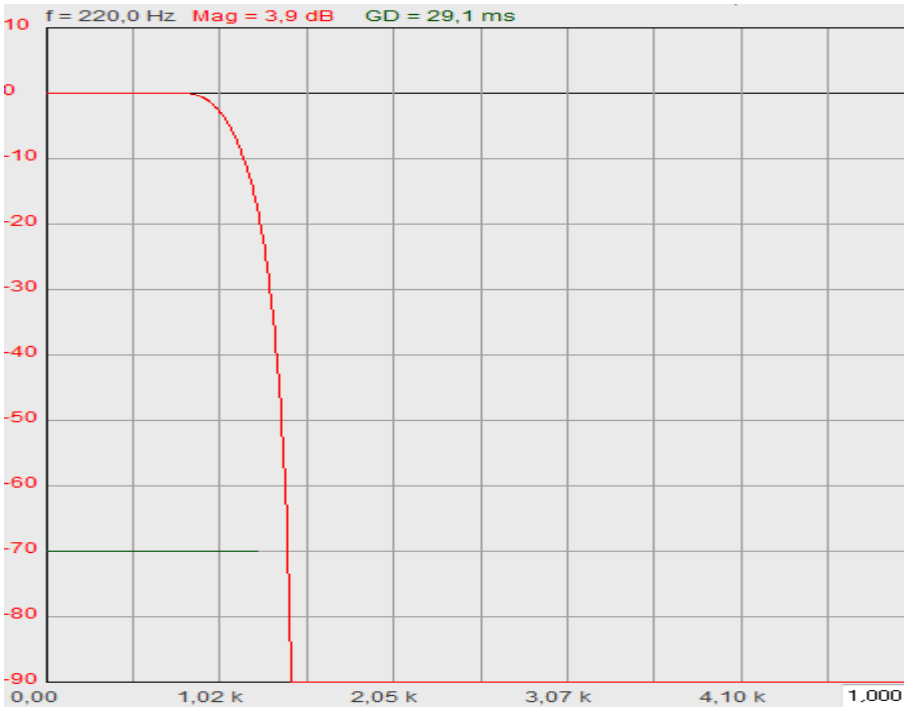
Tra i vari parametri usati per il calcolo, vale la pena citare il numero di 'tap'.
A grandi linee, maggiore è questo numero, più ripidi sono i fronti di salita e discesa del filtro.
Ma, va anche notato il fatto che il calcolo del filtro stesso diventa più oneroso in termini di cicli macchina necessari al microprocessore (CPU) usato; più tap, più calcoli...

Con un PC moderno non ci sono grossi problemi, mentre con microprocessori poco potenti il trattamento dell'audio fatto in questo modo diventa difficilmente realizzabile.
Da ricordare che nel nostro caso abbiamo 1/10 di secondo di tempo per effettuare tutti i calcoli, necessari per non perdere il prossimo buffer di dati in arrivo.....

Ad esempio questo è un filtro passa basso con frequenza di taglio di circa 1000Hz, calcolato con un numero di tap = 32

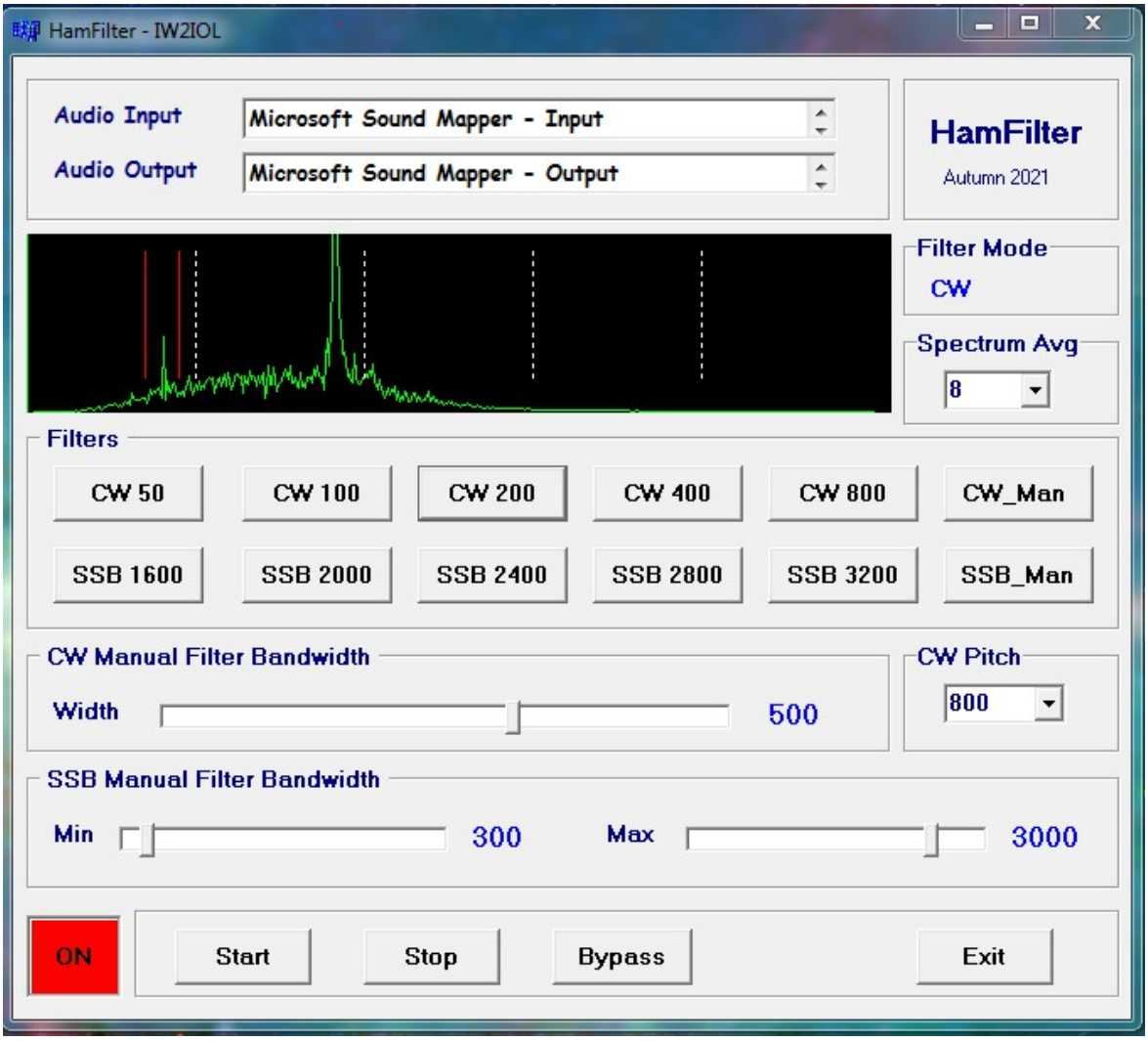


Questo è lo stesso filtro calcolato con 128 tap.
E' evidente il fatto che la curva ha i fianchi molto più ripidi; come esempio la frequenza di 1500Hz passa da -30dB a più di -90dB di attenuazione.



Per quanto riguarda i coefficienti, nella nostra implementazione in HamFilter il numero di tap è fissato a 200. E' un numero che ho ritenuto ottimale considerando risposta in frequenza del filtro ed i cicli macchina necessari per il calcolo dello stesso.
I coefficienti vengono ricalcolati ogni volta che viene variata la banda passante del filtro quando sono premuti i vari pulsanti o azionati i cursori.

Una volta fatto partire il programma e premuto il tasto Start, la finestra del programma si presenta così.



Audio Input e **Audio Output** sono rispettivamente, il canale di input del segnale audio proveniente dalla radio ed il canale di output.
Se avete più schede audio installate nel pc o adattatori audio tra radio e pc, i due list box vano settati di conseguenza.

CW Pitch seleziona la frequenza centrale del filtro CW.

Spectrum Avg media i valori usati per la visualizzazione dello spettro.
Più basso è il numero, più 'reattivo' diventa il grafico.

CW50 – CW800 selezionano la larghezza di banda del filtro passabanda CW.
Se per esempio la frequenza di pitch selezionata è 800 Hz, CW50 imposta un filtro che va da 775 a 825 Hz.

SSB1600 – SSB3200 selezionano la larghezza di banda del filtro passabanda SSB.
La frequenza minima è impostata a 300Hz, la massima viene impostata al valore selezionato.
Ad esempio SSB1600 imposta un filtro passa banda con banda compresa tra 300 e 1600 Hz.

CW_Man Permette di selezionare la larghezza di banda del filtro CW con un valore che va tra 10 e 800 Hz. Usando lo slider identificato con **CW Filter Bandwidth** si può selezionare la banda passante del filtro CW.

SSB_Man Permette di impostare la frequenza minima e la frequenza massima del filtro passabanda SSB. La frequenza minima è 100Hz, la frequenza massima 3600 Hz.

Il programma è stato pensato per essere usato con radio un po' datate che non hanno filtri, con radio auto-costruite (qrp) o radio pensate per il radio ascolto delle onde corte.
Non è stato pensato per essere usato con radio moderne che filtrano il segnale usando sofisticati sistemi di DSP in stadi precedenti la BF....

Ma, l'appetito viene mangiando, così si dice...

Per cui, perchè non portare la stessa idea su di un'altra piattaforma HW/SW diversa da un PC ? Non sempre abbiamo un PC a disposizione ad esempio quando siamo fuori casa o ancora non vogliamo usare un PC per queste cose.

Ho portato il SW su di un DSPIC della microchip come primo tentativo; ho usato un chip un pò datato perchè lo avevo in casa ma soprattutto perchè lo stesso è provvisto di un ADC (Analog to Digital Converter) e di un DAC (Digital to Analog Converter) interni.

Qui informazioni sul chip https://www.microchip.com/en-us/product/DSPIC33FJ128GP802#

Inoltre, sono disponibili librerie per la realizzazione di sw di 'tipo' DSP inclusi filtri FIR e FFT.
Da notare che il dspic menzionato sopra usa una aritmetica con numeri interi e non ha supporto HW per la gestione dell'aritmetica con numeri a virgola mobile (floating point).
Essendo sprovvisto di supporto per aritmetica floating point, la stessa è emulata dal SW caricando ulteriormente di lavoro extra il microprocessore...
Comunque, nonostante questa limitazione, un processore di questa famiglia accompagnato da un coded audio (TLV320AIC23B) è stato utilizzato anni fa per la realizzazione di sdr cube, un rtx sdr basato su softrock.

Qui informazioni su sdrcube https://www.sdr-cube.com/

Qui la pagina di OH2NLT (cheap dsp) che ha contribuito allo sviluppo del SW :

http://www.kolumbus.fi/juha.niinikoski/Cheap_dsp/Cheap_dsp.htm

In un secondo momento ho portato il SW su di un Teensy 3.2 (ARM Cortex-M4 at 72 MHz) usando la sua Audio Board. E' compatibile con Arduino IDE e anche per questo ci sono librerie per il trattamento dei segnali (DSP).

Qui informazioni su Teensy https://www.pjrc.com/teensy/

Le ultime versioni di Teensy (4.0 - ARM Cortex-M7 at 600 Mhz) hanno supporto nativo per i numeri in virgola mobile e una notevole potenza di calcolo. Sono stati usati per rtx sdr ad esempio con Teensy 3.2:

https://rheslip.blogspot.com/2015/03/teensy-sdr-transceiver-description_17.html

Cercando in rete ci sono altri interessanti progetti sviluppati attorno a Teensy 4.0 (o 4.1) per la realizzazione di rtx sdr con caratteristiche interessanti.

Ma, torniamo a noi, ho fatto un po' di prove e verificato che la stesa idea funziona anche su questi hw con capacità di calcolo diversa tra loro, OK, andiamo avanti...

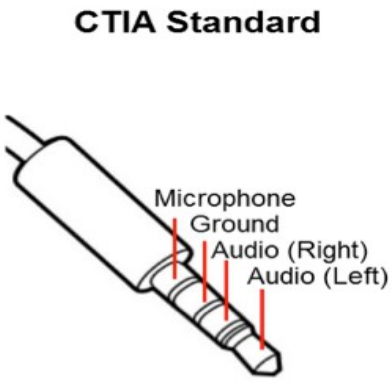
Al di là del trattamento del segnale audio, per motivi di praticità d'uso necessitiamo di qualche tasto e di un display per impostare filtri, vedere lo spettro del segnale etc etc.
Che si fa ? Aggiungo display tft, magari touch, tasti vari e inscatolo il tutto ?

Pensa e ripensa, e arriva l'idea malsana...

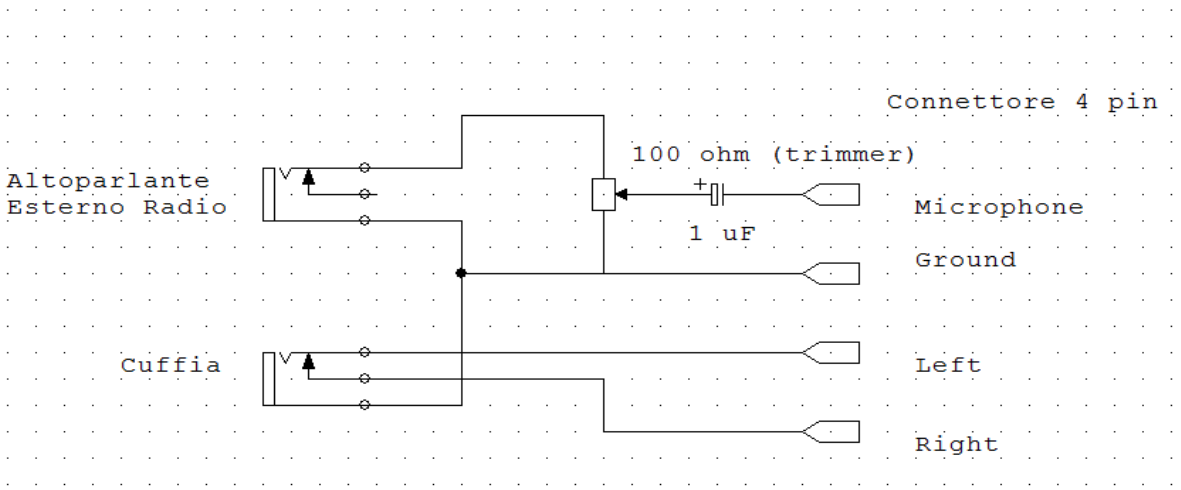
Uno smartphone o un tablet hanno già tutto quello che mi serve, incluso un bel contenitore, un bel display ed una certa potenza di calcolo.
Ho qualche vecchio smartphone nei cassetti in disuso da qualche anno, perchè non provare ?

Scartati modelli datati che non sono sufficientemente veloci in termini di CPU, ho provato con un Samsung S5, il massimo della tecnologia smartphone che avevo a disposizione.
Allora, per prima cosa dobbiamo pensare a come portare il segnale audio dal PC allo smartphone.
Ad alcuni di essi è possibile collegare una scheda audio USB esterna, ma non al mio accidenti.

Ultima spiaggia è il jack per il collegamento di una cuffia/microfono esterni



Questo è lo schema di collegamento del jack, abbiamo tutto quello che serve, ingresso microfono, uscita audio stereo. Non c'è un ingresso linea, ma con un paio di resistenze ed un trimmer
si può abbassare il segnale proveniente dall'uscita per l'altoparlante esterno della radio per iniettarlo nell'ingresso microfonico dello smartphone.



Cablato il tutto in una scatoletta 35 x 35 x 60 mm la cosa si presenta così.
Cavetto verso il jack cuffia/microfono dello smartphone e due jack uno per l'ingresso del segnale dalla radio (altoparlante o cuffia esterna) e l'altro per collegare una cuffia o amplificatore esterno (uscita audio dallo smartphone).

Il programma, o APP se preferite l'ho chiamato Poor Man DSP; stiamo raschiando il fondo, più povero di così...

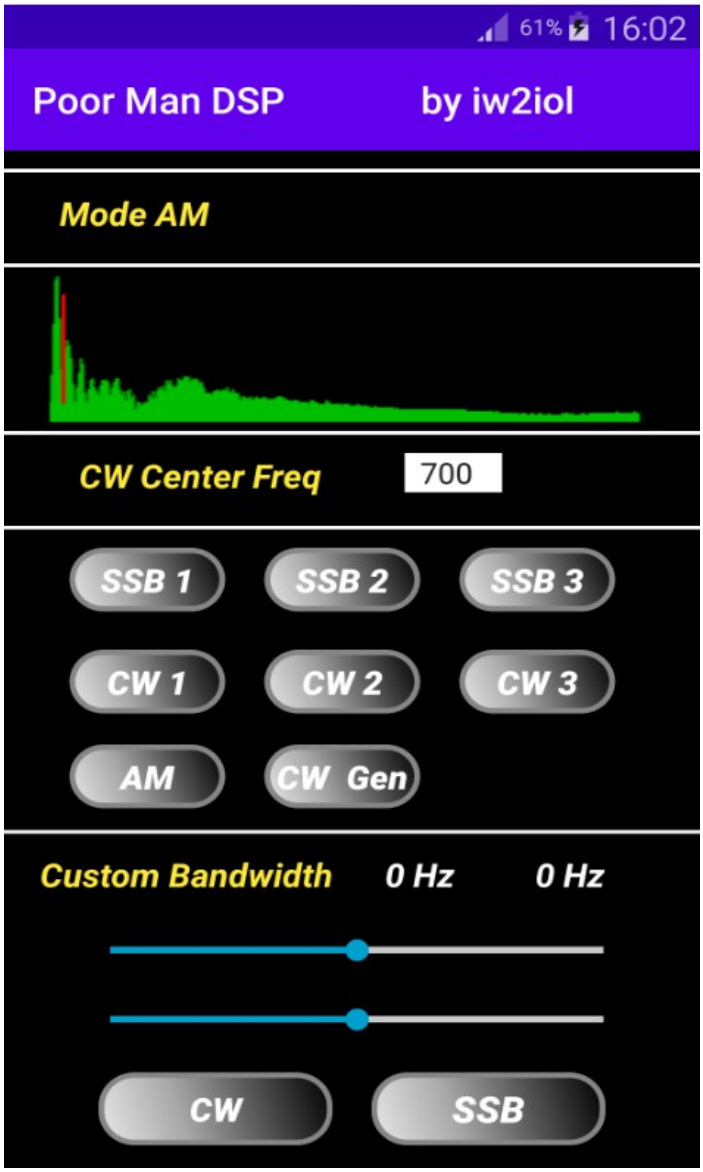
La APP è stata sviluppato usando Android Studio che si può scaricare liberamente dal sito di Google https://developer.android.com/ e si può caricare solo su dispositivi Android.
Il programma è stato sviluppato in linguaggio Java utilizzando le API di Android per la gestione dei flussi di dati Audio.

Qui altre info https://developer.android.com/reference/android/media/package-summary

La parte dsp che riguarda FFT e filtri FIR è stata riscritta per lavorare su questa piattaforma software; non sono state usate librerie di terzi.
Per curiosità, la frequenza di campionamento è di 8000 Hz e la dimensione del buffer usato per i dati audio contiene 2000 campioni. Quindi un nuovo buffer ogni quarto di secondo o se volete una latenza di 250 millisecondi.

Per installare la APP che trovate nel file .zip come file con nome app-release.apk non sono necessari particolari accorgimenti.
Va caricata nel telefono usando connessione USB o come allegato ad un mail.
Una volta scaricata è sufficiente premere sulla icona che rappresenta il file 'app-release.apk' per fare partire la sua installazione.
Necessita di autorizzazioni per l'uso della parte audio che vengono richieste dal sistema operativo (Android) durante l'installazione a cui và risposto affermativamente.

A installazione ultimata troverete una nuova icona con nome Poor Man Dsp.
Alla partenza mostrerà questa finestra



CW Center Freq è la frequenza centrale del filtro CW (o pitch nella versione PC)

SSB1 – SSB3 selezionano tre filtri SSB passabanda con taglio da 300 a 2200, 2600 e 3000 Hz

CW1 - CW3 selezionano tre filtri CW sentrati sulla frequenza Center Freq con ampiezza di 200, 400 e 600 Hz

CW e SSB nella sezione Custom Bandwidth della finestra, permetto di variare la frequenza del passabanda SSB o la ampiezza del filtro CW. Selezionando CW, il cursore in alto varia l'ampiezza del filtro CW

Start e Exit si spiegano da soli...