Chapter 10

RATE-CHANGE: Up- and down-sampling module

In certain applications involving digitized speech, such as subjective evaluation of speech processed by digital algorithms, it may be preferable to use sampling higher than the typical rate used for the algorithms under test. This is desirable because simpler analog filters with less phase distortion can be built. Another advantage is that upper frequency components of the signal are not lost. It also allows for the convenient shaping of the input signal, such as IRS, Δ_{SM}, and psophometric weightings. Consequently there is a need to adapt the sampling rate of the digitized signal to that of the processing algorithm. For telephony applications, the typical sampling rate is 8000 Hz with a signal bandwidth in general of 300-3400 Hz, and for wideband speech applications, a bandwidth of 50-7000 Hz is desired with sampling rate of 16000 Hz. During the 2005-2008 ITU-T study period, greater audio bandwidth were considered and superwideband and fullband audio codecs were developed. Their typical samplings rates are respectively 32000 Hz with a signal bandwidth of 50-14000 Hz, and 48000 Hz with a signal bandwidth of 20-20000 Hz. Therefore, sampling rates above 8000 Hz and 16000 Hz are desirable, respectively. In several experiments [61] the sampling rate was 16 kHz. In others (see [62] and [63]), 48 kHz and 32 kHz were utilized. Hence the need for a software tool to carry out filtering and sampling rate change. Next, the rate change and spectral weighting routines implemented in the ITU-T STL are presented.

10.1 Description of the Algorithm

Signal processing theory describes the basic arrangement for decimation of signals; first the signal is low-pass filtered to limit its bandwidth in order to avoid aliasing when the rate is lowered and, second, to decimate the samples, i.e., to drop out samples from the input signal, such that the desired output rate is obtained. For example, if a rate reduction from 48 kHz to 8 kHz is desired, a decimation factor of 6:1 is necessary. This is equivalent to say that, after limiting the bandwidth of the digitized speech to 4 kHz, 5 out 6 samples are skipped, or alternatively, only 1 out of 6 samples will be kept (or saved) from the signal.

The up-sampling of signals requires that each of the input samples be followed by a number of zero samples, such that the desired output rate is achieved; after this, an interpolation

operation of these zero samples is performed to obtain a continuous-envelope signal. For example, up-sampling data from 8 kHz to 16 kHz requires interleaving each sample of the input signal with a zero sample followed by interpolation of the signal. This interpolation can be carried out by means of a polynomial, which is equivalent to a filtering operation.

The type of filtering required is determined by the application intended for the signals. For the tools needed in this version of the STL, three different groups of characteristics were defined:

- **High-quality**: Change in rate without changing the frequency response of the input signal. This is accomplished with a flat, linear phase, low-pass or bandpass FIR filter.

- **Spectral weighting**: Spectral weighting without rate change is necessary for some applications. For narrow-band speech, available are the IRS weighting specified in ITU-T Rec. P.48, the so-called "modified" IRS (annex D of ITU-T Rec. P.830), the far-to-near-field conversion Δ_{SM} weighting, and the psophometric noise weighting of ITU-T Rec. O.41. For wideband signals, the mask for wideband handsets, as defined in ITU-T Rec. P.341, is also available. For super-wideband signals, the mask for super-wideband videoconferencing terminals has been derived as an extension of ITU-T Rec. P.341.

- **PCM quality**: Change in rate accompanied with modification of the frequency response of the input signal according to the mask specified in ITU-T Recommendation G.712. This is accomplished with a non-linear phase low-pass IIR filter.

10.1.1 High-quality

The response of the filters in this type of rate change must minimize phase and amplitude distortion. For example, for decimation from 48 kHz to 16 kHz, the filter must be flat up to about 8 kHz (except for the transition, or cut-off, region), with a linear phase. In other cases, it may be desirable to remove the DC component and hum noise (50–60 Hz AC line noise) from the signal without additional phase distortion to the upper region of the spectrum.

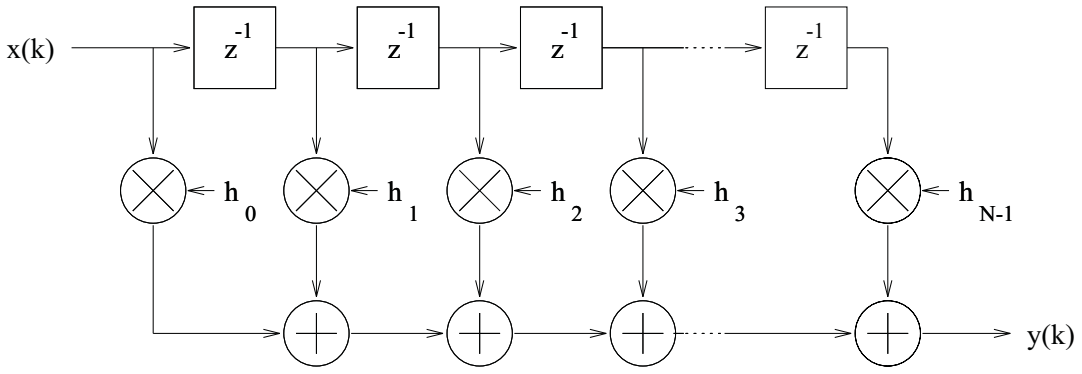


Figure 10.1: FIR filter block diagram.

One way to do this is to use a linear phase finite impulse response (FIR) digital filter, as

in figure 10.1. The input and output characteristic is defined by:

$$y(k) = \sum_{i=0}^{N-1} h(i) \cdot x(k-i)$$

Linear phase is guaranteed if the filter is symmetric, i.e.:

$$h(k) = h(N-1-k), \text{ for } k = 0..N-1$$

10.1.2 Telephony-band weighting

IRS weighting

The IRS weighting corresponds to a bandpass filtering characteristic whose mask can be found in ITU-T Recommendation P.48 [64]. The send and receive spectral shapes of the IRS weighting were obtained in a round-robin series of measurements made on a number of analog telephones in the early 1970's [65]. From these measurements, the average send and receive frequency-response characteristics were derived. However, for the loudness balance purposes for which the IRS was designed, it was also necessary to include a 300-3400 Hz bandpass filter, known as the SRAEN *(Système de Référence pour la détermination de l'Affablissement Équivalent pour la Netteté; Reference System for determining Articulation Ratings)* filter [66, 67]. The values of send and receive sensitivity currently given in ITU-T Rec. P.48 (columns 2 and 3 in Table 10.1) are therefore composed of the average send and receive responses for a number of telephones, as well as the response of the SRAEN filter (see column 4 of Table 10.1).

Because the P.48 IRS weighting used to be considered to model an average narrow-band telephone handset deployed in the PSTN, the IRS weighting has been chosen to simulate speech signals obtained from a regular handset. Examples of standardization efforts using the P.48 weighting characteristic are the ITU-T Recommendations G.711, G.721, and G.728. This weighting, as defined in P.48, is sometimes called "full-IRS" weighting.

While the weighting characteristic in P.48 was considered to model connections over analog transmission facilities in the past (although it is not clear why the SRAEN filter should be included in both the send and receive paths), it is no longer representative of connections over modern digital facilities. In particular, the low frequency roll-off gives rise to unnecessary quality degradation. For the purpose of low bit-rate coder evaluation, especially where the coder is located in the telephone handset, a better characteristic can be obtained by modifying the P.48 full-IRS response to remove the SRAEN filter as shown in columns 5 and 6 of Table 10.1. These values are specified in Annex D of ITU-T Recommendation P.830 [68] and define the so-called "modified" IRS weighting. The modified IRS has been used in the development of ITU-T Recommendations G.723.1 and G.729.

Table 10.1: Send and receive amplitude frequency characteristics for the IRS response as in ITU-T Rec.P.48, the SRAEN filter, and the modified IRS (P.48 IRS with SRAEN filter insertion loss removed).

Frequency	P.48 IRS		SRAEN	Modified IRS	
	Send	Receive	Filter	Send	Receive
(Hz)	(dbPa/V)	(dbPa/V)	(dB)	(dbPa/V)	(dbPa/V)
100	-45.8	-27.2	14.1	-31.7	-13.4
125	-36.1	-18.8	11.4	-24.7	-7.4
160	-25.6	-10.8	8.4	-17.2	-2.4
200	-19.2	-2.7	5.9	-13.3	3.2
250	-14.3	2.7	4.0	-10.3	6.7
300	-11.3	6.4	2.8	-8.5	9.2
315	-10.8	7.2	2.5	-8.3	9.7
400	-8.4	9.9	1.4	-7.0	11.3
500	-6.9	11.3	0.6	-6.3	11.9
600	-6.3	11.8	0.3	-6.0	12.1
630	-6.1	11.9	0.2	-5.9	12.1
800	-4.9	12.3	0.0	-4.9	12.3
1000	-3.7	12.6	0.0	-3.7	12.6
1250	-2.3	12.5	0.0	-2.3	12.5
1600	-0.6	13.0	0.1	-0.5	13.1
2000	0.3	13.1	-0.2	0.1	12.9
2500	1.8	13.1	-0.5	1.3	12.6
3000	1.5	12.5	0.5	2.0	13.0
3150	1.8	12.6	0.3	2.1	12.9
3500	-7.3	3.9	7.0	-0.3	10.9
4000	-37.2	-31.6	33.7	-3.5	2.1
5000	-52.2	-54.9	43.2	-9.0	-11.7
6300	-73.6	-67.5		-23*	
8000	-90.0	-90.0		-40*	
(*): Values estimated from the modified IRS implemented in the STL.					

The most important part of either the full or the modified IRS weighting is the transmission (or send) characteristic. The receive characteristic is less important because listening is in general done using handsets conforming to P.48 (which eliminates the need for filtering by the software, since it is done by the telephone terminal). In addition, the receive characteristic is relatively flat. Some studies also show that the use of headphones instead of handsets does not result in significantly different results while yielding lesser listener fatigue [69, 70]. Nevertheless, for cases where the receive-side MIRS filter is to be applied, a FIR implementation of this filter is available for 8000 Hz and 16000 Hz sampling rates.

An unspecified point in both P.48 and modified IRS is the phase response of the filter. There have been discussions within UGST on this topic and the conclusion was that, since the phase response is unspecified, it should be kept as generic as possible, what is better accomplished by keeping the phase linear[1]. If a certain non-linear phase characteristic is desired by the user, this can be implemented by cascading an all-pass filter with the desired phase response with one of the available FIR IRS implementations. Therefore, the IRS filters are implemented as FIR filters, as depicted in Figure 10.1.

ITU-T Recommendation P.48 presents the nominal values for the amplitude response in column 2 of its Table 1 (here reproduced in column 2 of Table 10.1) and then the upper and lower tolerances listed in its Table 2. For the STL approach, it was decided to design IRS filters whose characteristic would deviate no more than 0.5 dB from the average values in P.48 (see in Figure 10.12 the agreement of the nominal values, represented by dots, and the measured frequency response for the original P.48 IRS characteristic, represented by the continuous curve in the figure).

Other weightings

A filter that simulates the input response characteristic of certain mobile terminals was incorporated in the STL for data sampled at 16 kHz. Figures 10.10 and 10.11 display the respective frequency and impulse responses for the filter.

Another filter that models the input response characteristic of certain super-wideband videoconferencing terminals was incorporated in the STL for data sampled at 32 kHz. Figures 10.22 and 10.23 show the respective frequency and impulse responses for the filter.

10.1.3 Wideband weighting

P.341 weighting

While the IRS filter is applicable to telephony bandwidth (or narrowband) speech, for wideband speech the specification for the send and receive sides is given in ITU-T Recommendation P.341 [71]. The mask specified in P.341 is rather wide, and an implementation of the send-side mask agreed on by the experts has been incorporated in the STL.

[1] In spite of that, a non-linear phase IIR IRS filter is provided in the IIR module as an example of a cascade-form IIR filter implementation.

Other weightings

In the process to select a wideband codec at 32 and 24 kbit/s, a 50 Hz-5 kHz bandpass filter was developed and incorporated in the STL. Figures 10.24 and 10.25 display the respective frequency and impulse responses for the filter.

A 100 Hz - 5 kHz filter was also designed for tests of another wideband codec. Figures 10.26 and 10.27 show the respective frequency and impulse response of the filter.

10.1.4 Greater wideband weightings

P.341 extension weighting

For super-wideband signals, the mask for super-wideband videoconferencing terminals is based on the ITU-T Rec. P.341. The sensitivity/frequency characteristics of the P.341 filter were extended to a larger band [50 Hz - 14 kHz] with a sampling frequency of 32 kHz. The corresponding 50 Hz-14 kHz filter was developed and incorporated in the STL. Figures 10.22 and 10.23 display the respective frequency and impulse responses for the filter.

Other weightings

To generate anchors typically used in BS.1534 [72] "MUlti Stimulus test with Hidden Reference and Anchor (MUSHRA)" subjective tests, seven low-pass filters are also provided in the STL. Those anchors are generated by low-pass filters with cut-off frequencies 1.5, 3.5, 7, 10, 12, 14 and 20 kHz at sampling frequency of 48kHz. Their frequency responses are shown, respectively, in Figures 10.30, 10.32, 10.34,10.36, 10.38, 10.40, and 10.42. Their impulse responses are also given in Figures 10.31, 10.33, 10.35, 10.37, 10.39, 10.41, and 10.43. In addition a bandpass filter [20 Hz - 20 kHz) operating at sampling frequency of 48 kHz was also designed. Its frequency response is shown in figure 10.28 and its impulse responses in figure 10.29.

10.1.5 Noise weighting

Two weighting filters are available in this version of the STL, the psophometric and the Δ_{SM} weighting filters.

The psophometric weighting curve defined by ITU-T Recommendation O.41 is used for measuring the noise level in telephone circuits, accounting for the subjective perception of noise. The psophometric noise measure (given in dBmp) is related to the North-American C-message weighting curve (given in dBrnC), using to the following:

$$dBmp = dBrnC - 90.0dB$$

The other type of signal weighting filter is the Δ_{SM}, used for converting acoustic signals recorded in the far field using an omnidirectional microphone to the near-field equivalent of that signal if it were in the background of a telephone user. Owing to the directionality of the human mouth, head and torso, the high frequencies will mainly be radiated in the frontal direction, while the diffuse field will represent a spatial integration of the radiation

in all directions [73]. Hence, the Δ_{SM} filter is deployed for weighting acoustic noises (babble, vehicular, etc.) before electrical summation with clean speech files, in order to simulate speech corrupted by background noise. It is useful in subjective listening tests where precise control of the actual SNR is necessary.

Both these filters have been implemented as FIR filters. The psophometric filter has been designed for speech sampled at 8 kHz, and the Δ_{SM} filter for speech sampled at 16 kHz. It should be noted that these filters, like the IRS filters, are also frequency-specific and, unlike the low-pass high-quality FIR filters described before, cannot be used for arbitrary rate ratio conversion.

10.1.6 PCM Quality

There are applications requiring the simulation of the response of filters found in the A/D and D/A interfaces of current transmission systems, which are in general PCM systems satisfying ITU-T Recommendation G.711. The filters associated with G.711 are specified in Recommendation G.712 [74]. The main characteristic of these filters is the low out-of-band rejection of 25 dB.

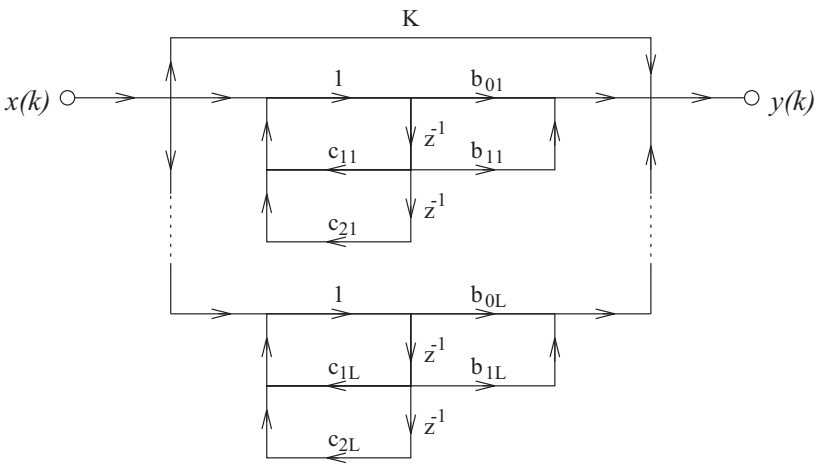


Figure 10.2: Parallel-form IIR filter block diagram.

In this context it is also necessary to simulate the convertion back to and forth the analog domain, e.g. to simulate multiple transcodings which are called *asynchronous trancodings*[2]. One way to simulate asynchronous transcodings is by means of a non-linear phase filter (non-constant group delay), which is most efficiently implemented using IIR filters.

Infinite impulse response (IIR) filters used in this tool are of the parallel form (see figure 10.2), described by the equation:

$$H_I^p(z) = K + \sum_{l=1}^{L} \frac{b_{0l} + b_{1l}z^{-1}}{1 + c_{1l}z^{-1} + c_{2l}z^{-2}}$$

[2]As the name indicates, there is no synchronization between sampling instants of the two digital systems, i.e., re-sampling in the succeeding A/D is not synchronous to the clock in the preceding D/A converter.

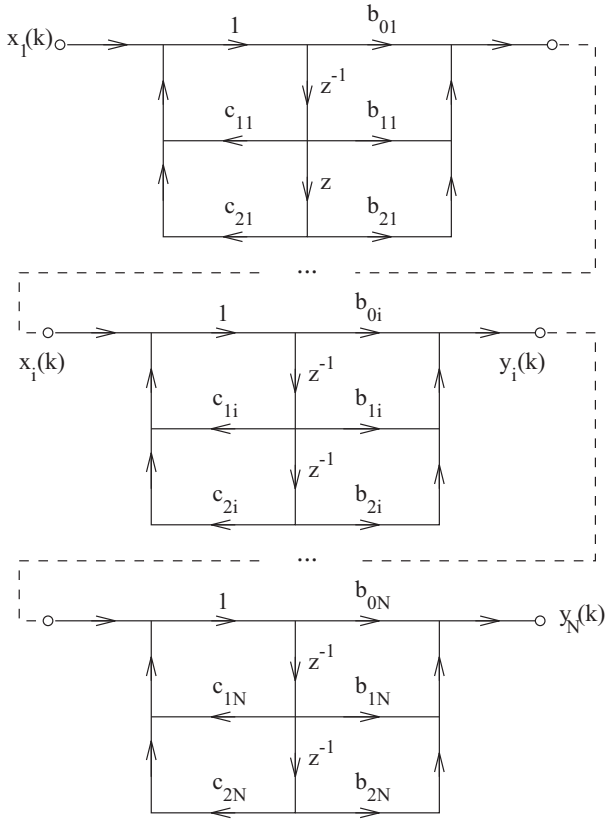


Figure 10.3: Cascade-form IIR filter block diagram.

and of the cascade form (see figure 10.3), described by the equation:

$$H_I^c(z) = \prod_{l=1}^{N} \frac{b_{0l} + b_{1l}z^{-1} + b_{2l}z^{-2}}{1 + c_{1l}z^{-1} + c_{2l}z^{-2}}$$

10.2 Implementation

The rate change algorithm is organized in two modules, FIR and IIR, with prototypes respectively in `firflt.h` and `iirflt.h`. It evolved from a version initially developed as part of the ETSI Half-rate GSM codec Host Laboratory exercise [75]. The rate-change functionality was incorporated in the STL92 in two main files, `hqflt.c` and `pcmflt.c`. To make these routines more flexible, the following modifications were included:

FIR: the FIR module was divided into a library source file (`fir-lib.c`) containing the basic filtering and initialization functions, as well as into source files for each kind of filter: `fir-flat.c` for high-quality low-pass and bandpass filters, `fir-irs.c` for the classical and modified IRS filters, and so on;

IIR: the IIR module was divided into a library file (`iir-lib.c`) containing basic filtering and initialization functions, as well as into source files for each kind of filter: `iir-g712.c` for G.712 filtering using the parallel-form filters, `iir-flat.c` for flat bandpass 1:3 and 3:1 asynchronization filtering using a cascade-form filter, and so on.

Files `fir-*.c` of the FIR module contain all the routines implementing FIR filters, i.e., the high-quality filters and IRS, Δ_{SM} and psophometric weighing filters. Files `iir-*.c` of the IIR module implement the IIR filters, i.e., the parallel-form PCM filter and the cascade-form 3:1 asynchronization filter.

Some of these filters have been implemented using 24-bit coefficients, thus allowing real-time, bit-exact hardware implementation of these routines. It may be noted that, for these filters in the STL, the calculations are performed in floating point by converting the coefficients from the range $-2^{23}..2^{23}-1$ to $-1..+1$, which is not needed in real time hardware with fixed point DSPs.

Frequency response and impulse response plots are provided for the STL filters in the forthcoming sections. It should be noted, however, that the impulse responses shown have been computed from the 16-bit quantized impulse responses of the filters, as generated by the demonstration programs, while the frequency responses were calculated as described in section 10.3. It should be noted that the apparent asymmetry in some impulse responses happens because an integer number of samples are generated, and linear interpolation is used to draw the figure. If the impulse responses were derived directly from the filter coefficients, the plot would be symmetric.

NOTE: When the same filter type is used by several independent speech materials (e.g. several speech files) within the same execution of an application program, the user must remember that the filters have memory. Hence, wrong results can be obtained if a given number of initial samples are not discarded. See section 10.4 for an example, where the first 512 samples are skipped when calculating the power level of the output tone.

10.2.1 FIR module

The frequency responses of the implemented high-quality low-pass filters are shown in figures 10.4 and 10.5 (for rate-change factors 2 and 3, respectively), while the telephone bandwidth bandpass filter is given in figure 10.6 (only a rate-change factor of 2 is available). The impulse responses of these filters are given in figures 10.7, 10.8, and 10.9, respectively for the up-sampling filters (factors 2 and 3), for the down-sampling filters (factors 2 and 3), and for the bandpass filter.

The transmit-side IRS filter has been implemented for the "regular" and modified flavors. The regular transmit-side P.48 IRS filter amplitude responses are shown in figure 10.12 (the available sampling rates are 8 and 16 kHz). The transmit-side modified IRS filter is available for sampling at 16 kHz and 48 kHz, and their frequency responses are shown in figure 10.14. The impulse response of these transmit-side IRS filters are in figures 10.13 and 10.15 for the regular and modified IRS filters, respectively. The receive-side modified IRS filter has also been implemented and the frequency responses for 8 kHz and 16 kHz sampling rate are found in 10.16. The impulse responses of the receive-side modified IRS filters are shown in figure 10.17.

The frequency response of the STL psophometric filter is given in figure 10.18, and that of the Δ_{SM} filter in figure 10.19.

For wideband signals, three weighting filters are available. The transmit-side ITU-T P.341 filter amplitude response is shown in figure 10.20, and its impulse response is shown in

figure 10.21. Alternatively to the P.341 filter, the frequency and impulse responses of the two bandpass filters, 50 Hz-5 kHz bandpass filter and 100 Hz -5 kHz, are shown, respectively, in figures 10.24 and 10.25, and in figures 10.26 and 10.27.

For super wideband signals, four weighting filters are available. The 50 Hz-14 kHz bandpass filter, extension of ITU-T P.341 filter, is presented in figures 10.22 (frequency response), and 10.23 (impulse response). Alternatively to this P.341 filter extension, the frequency responses of the three MUSHRA anchors filters, LP3.5, LP7 and LP10 filters, are shown in figures 10.32, 10.34 and 10.36, respectively. Their impulse responses are shown in figures 10.33, 10.35 and 10.37, respectively.

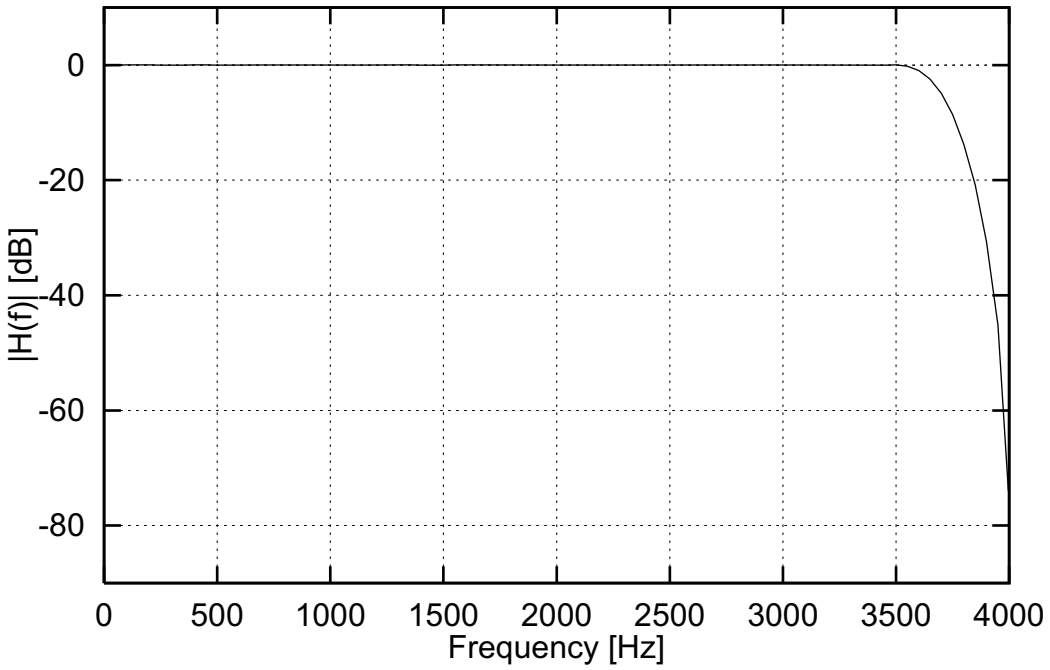
The high-quality filters were implemented for rate-change factors of 2 and 3. The IRS filters, band-limiting filters and MUSHRA anchors have been designed for specific *sampling rates* (e.g. 8 and 16 kHz). It should be noted that, while the high-quality filters are independent of the rate, these filters are not, because their masks are specified in terms of Hz, rather than normalized frequencies. This means that to carry out a high-quality up-sampling from 8 to 16 kHz, and from 16 to 32 kHz, the same routines are called, while for IRS, band-limiting or MUSHRA anchors, there is no rate-change routine from 16 to 32 kHz.

Since the digital filters have memory, state variables are needed. In this version of the STL, a type `SCD_FIR` is defined, containing the past sample memory, as well as filter coefficients and other control variables. Its fields, whose values shall never be changed by the user, are as follows:

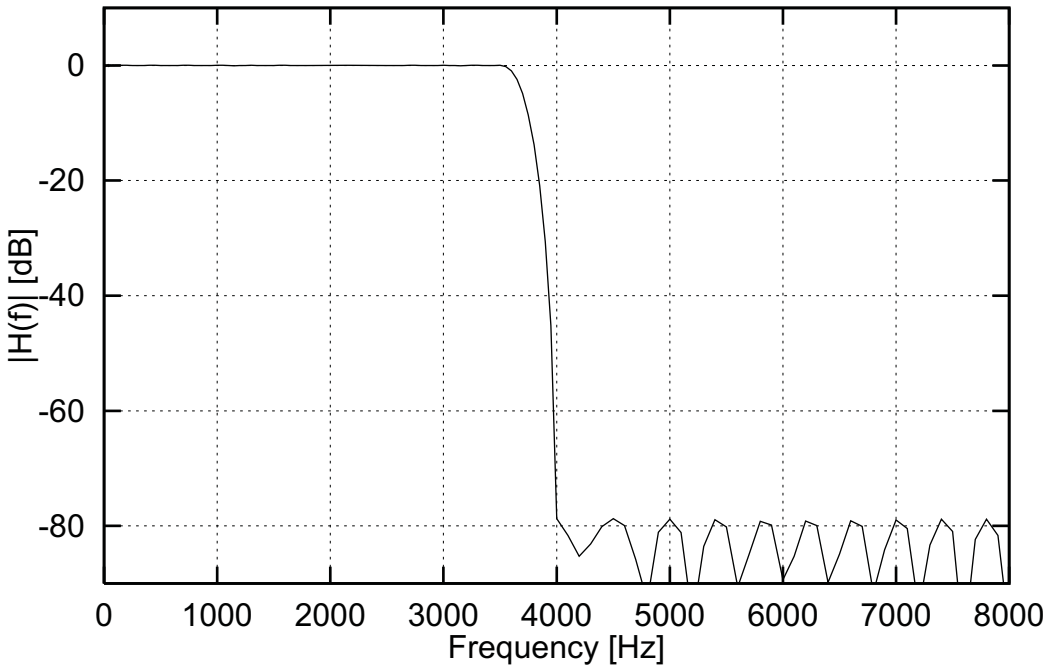
lenh0 .....Number of FIR coefficients
dwn_up ..... Down-sampling factor
k0 .....Start index in next segment (needed in segment-wise filtering)
h0 .....Pointer to array with FIR coefficients
T ..... Pointer to delay line
hswitch .....Switch to FIR-kernel: up- or down- sampling

The relevant routines for each module are described in the next sections.[3]

[3]It should be noted that in the source code files there are local (privately-defined) functions which are not intended to be directly accessed by the user and therefore are not described here.

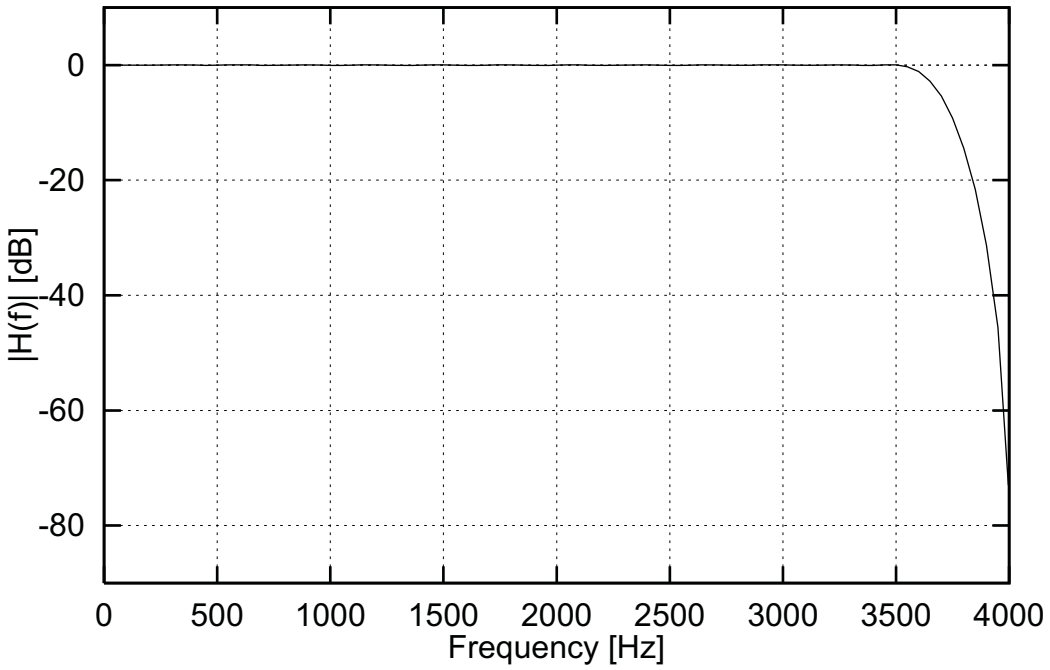


(a) High-quality filter for up-sampling.

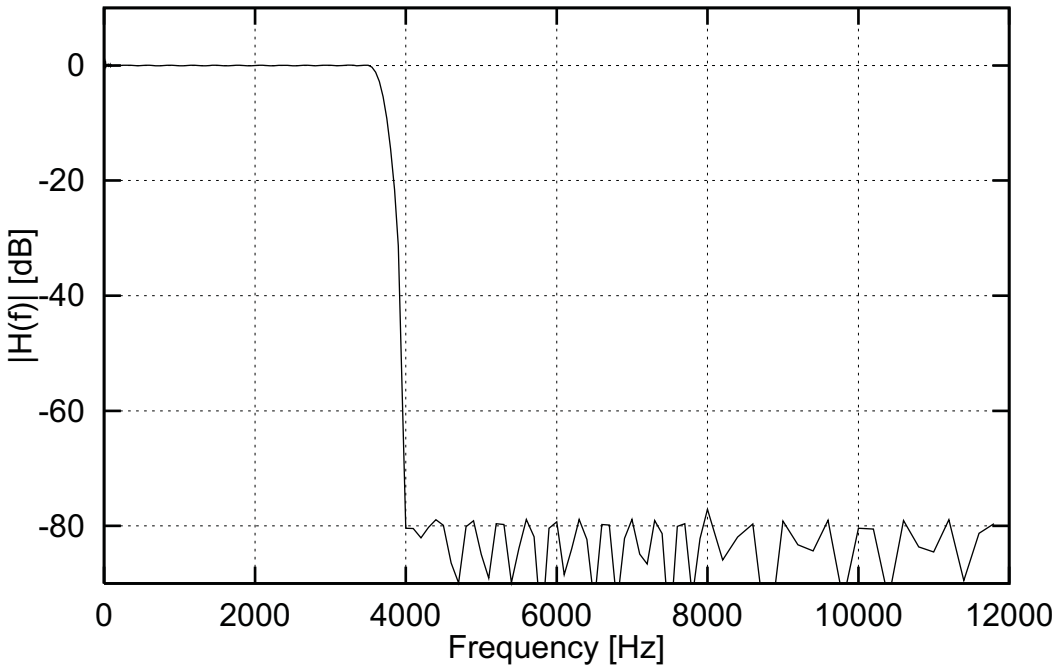


(b) High-quality filter for down-sampling.

Figure 10.4: High-quality filter responses for a factor of 2 and sampling rates of 8000 and 16000 Hz.

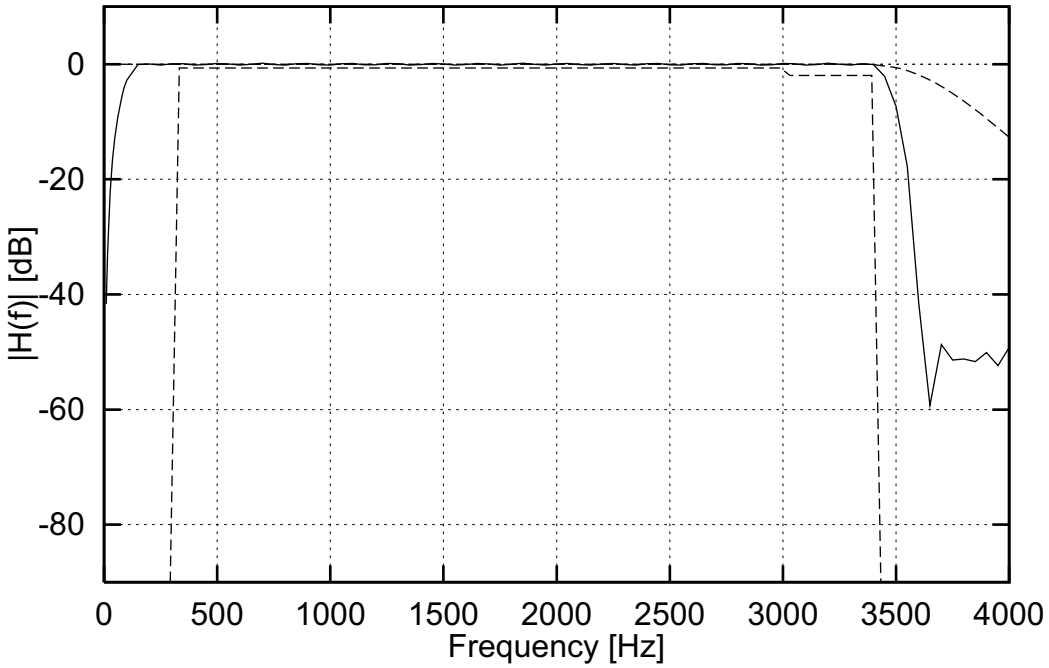


(a) High-quality filter for up-sampling.

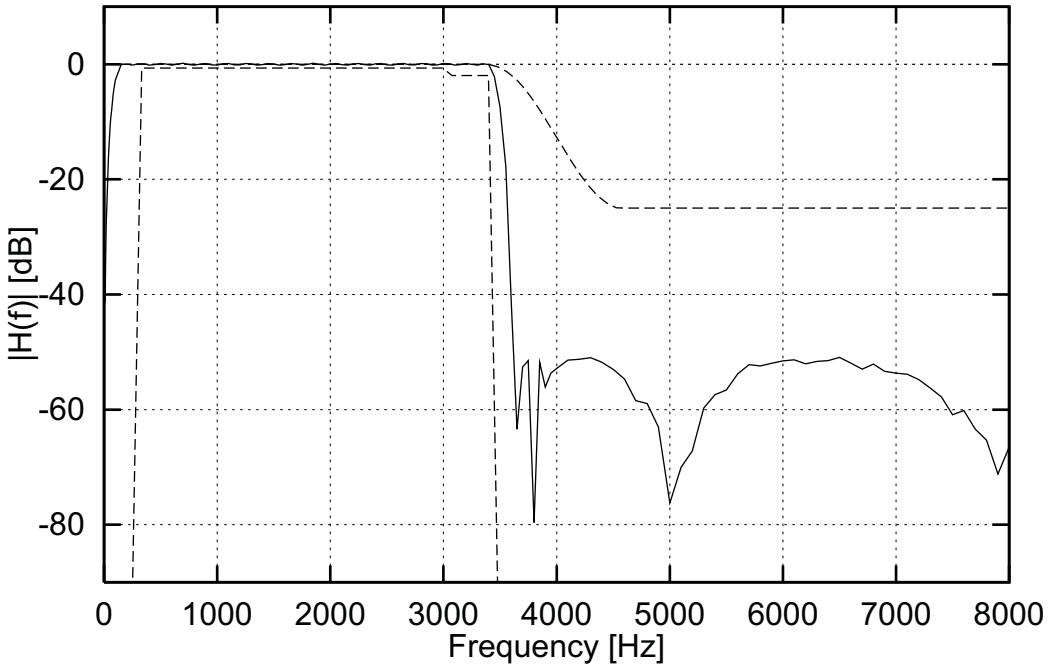


(b) High-quality filter for down-sampling.

Figure 10.5: High-quality filter responses for a factor of 3 and sampling rates of 8000 and 24000 Hz.



(a) High-quality bandpass for up-sampling (factor 1:2).



(b) High-quality bandpass for 2:1 down-sampling or for 1:1 filtering.

Figure 10.6: High-quality bandpass filter responses. Mask shown is that of the G.712 filter, for reference.

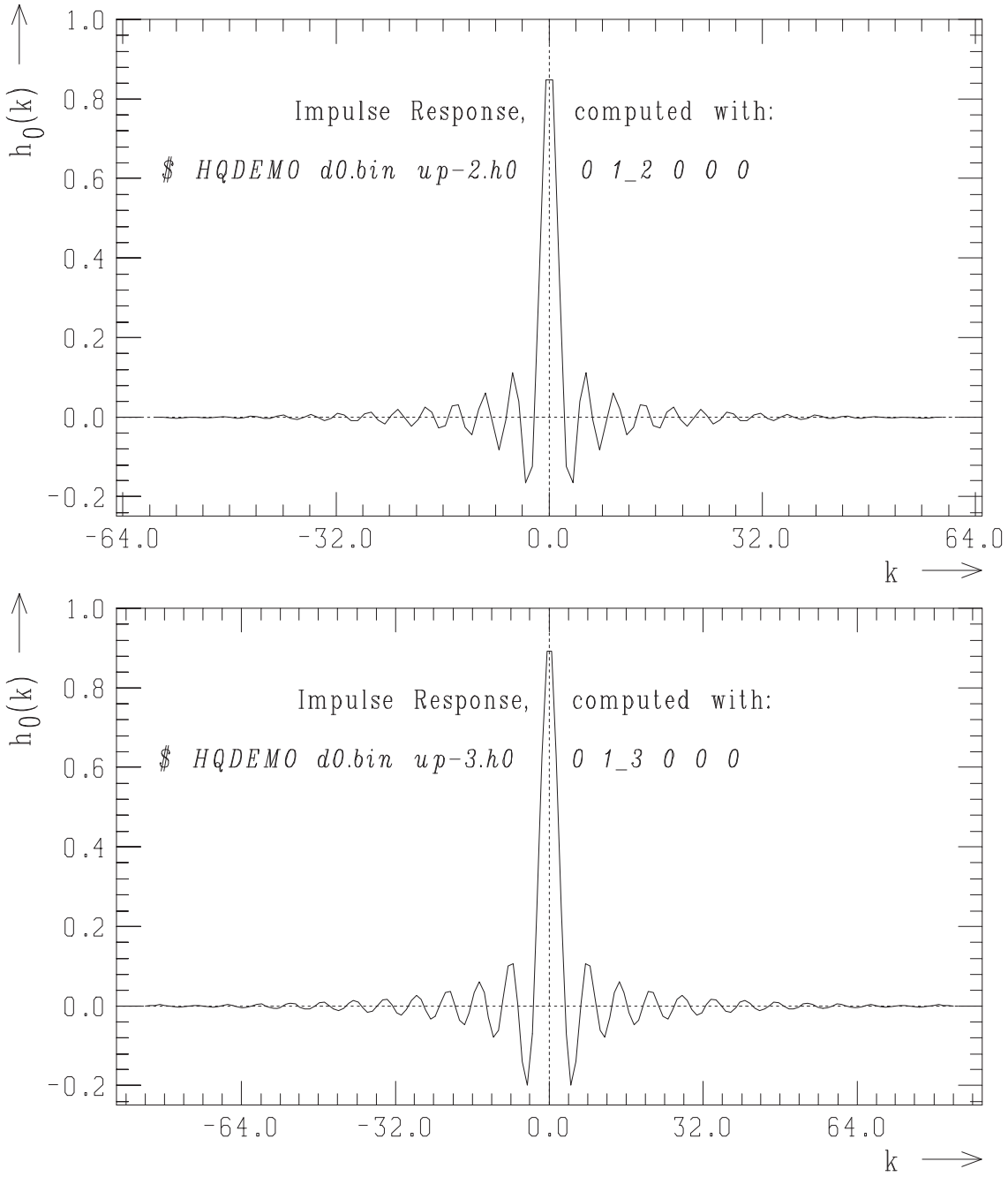


Figure 10.7: Impulse response for high-quality up-sampling filters (top, factor of 2; bottom, factor of 3).

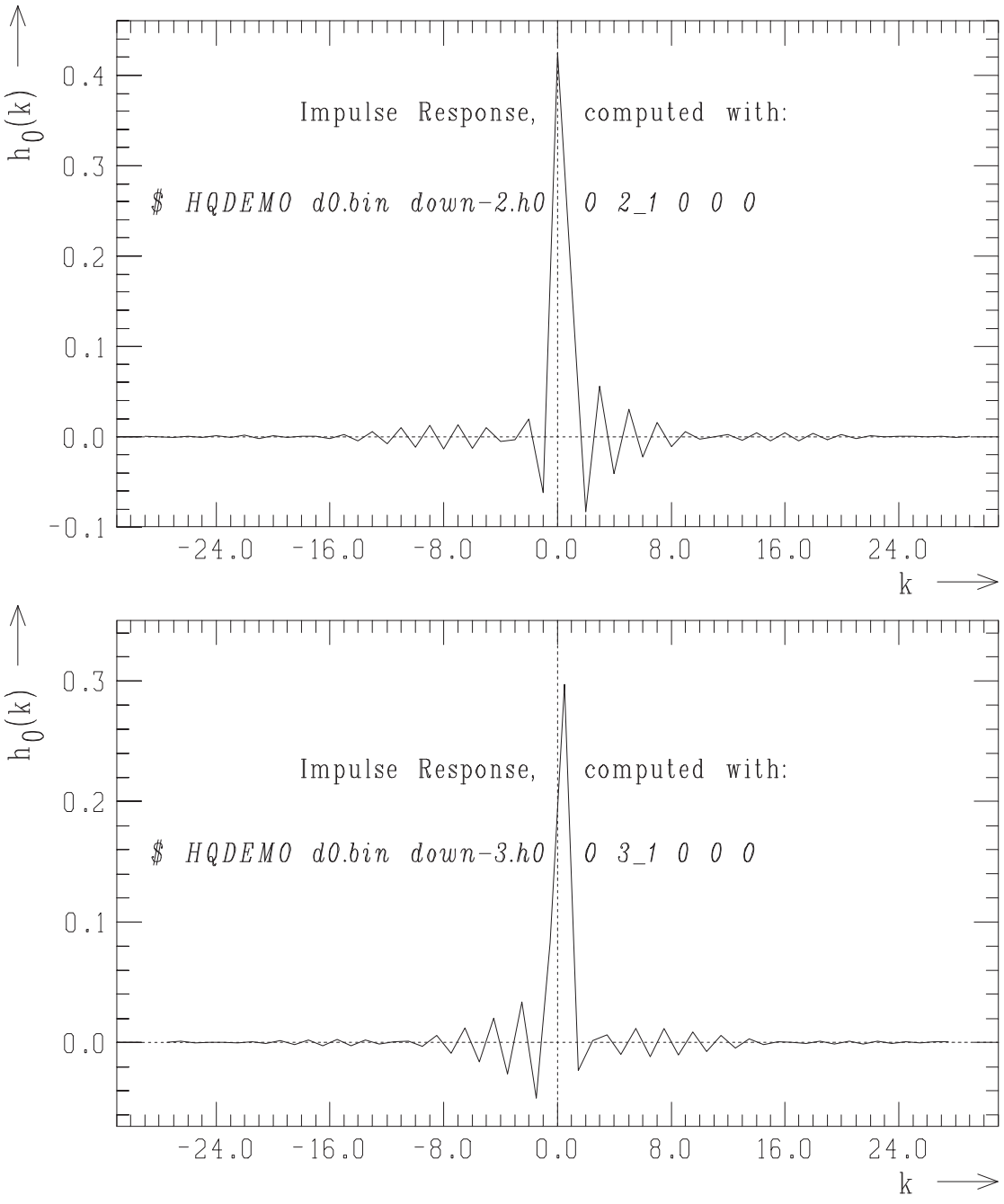
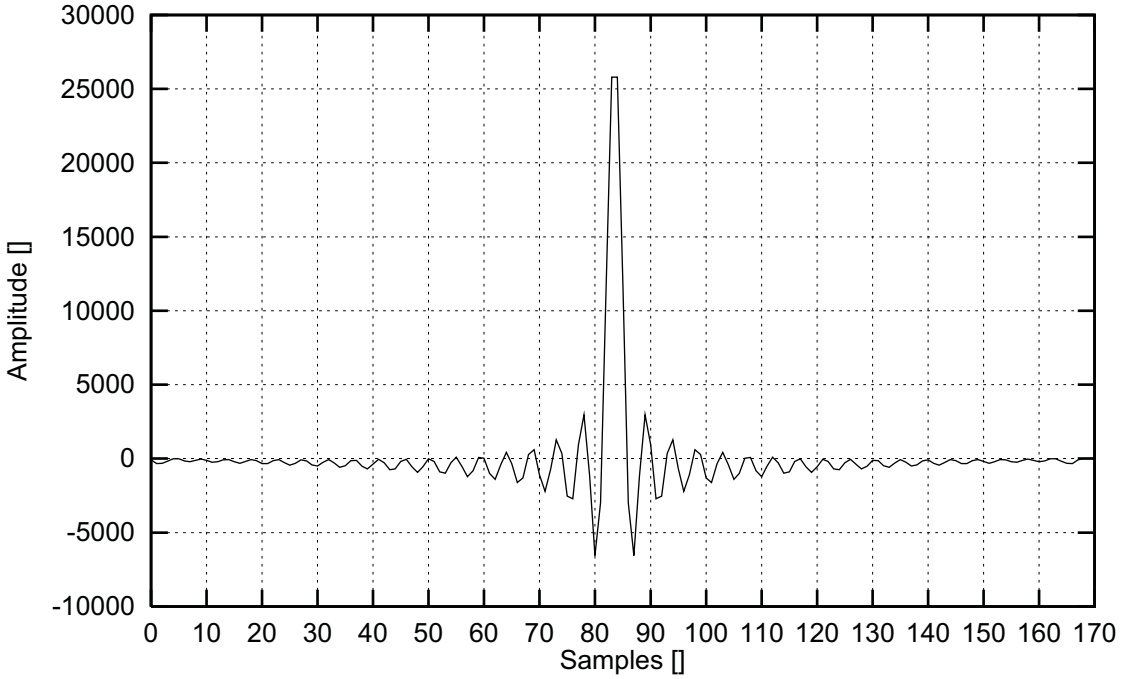
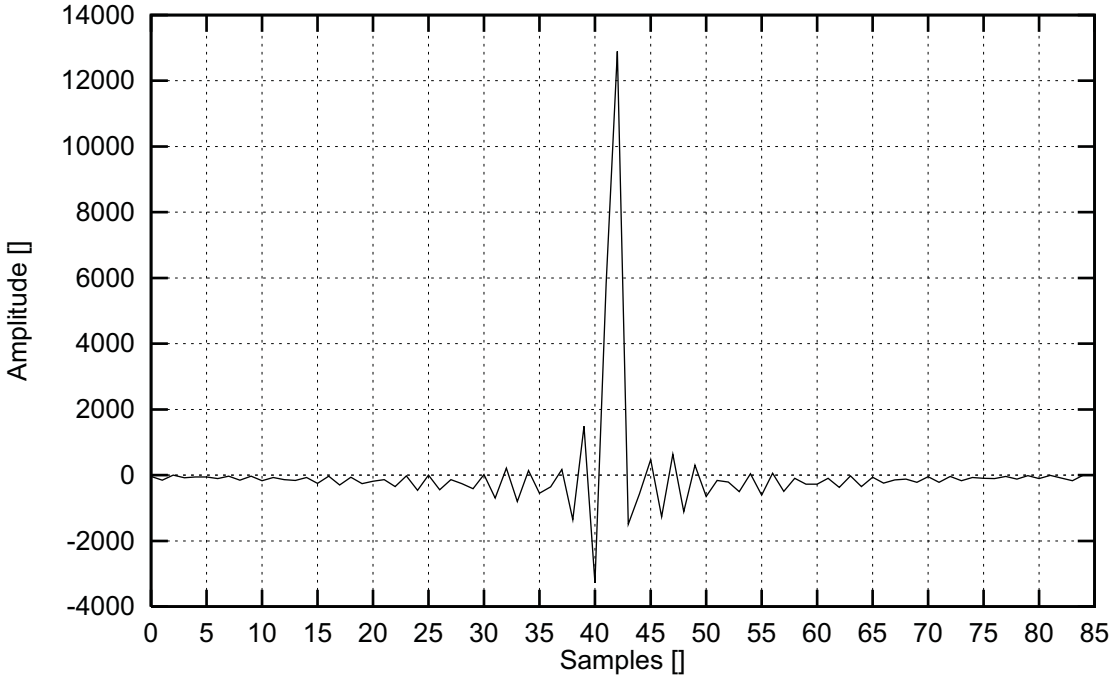


Figure 10.8: Impulse response for high-quality down-sampling filters (top, factor of 2; bottom, factor of 3).



(a) High-quality bandpass for up-sampling (factor 1:2).



(b) High-quality bandpass for down-sampling (factor 2:1).

Figure 10.9: Impulse response for high-quality bandpass filter (factors 2:1 and 1:1). Top is up-sampling by a factor of 1:2, and bottom is down-sampling by a factor of 2:1.

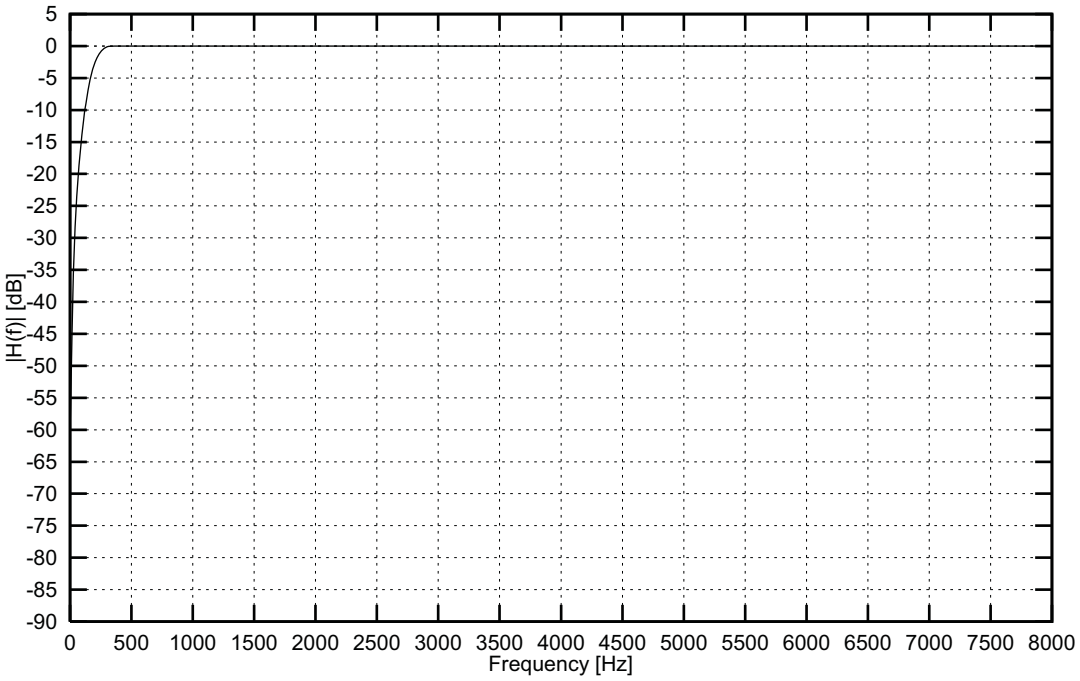


Figure 10.10: STL mobile station input (MSIN) frequency response for data sampled at 16 kHz (factor 1:1).

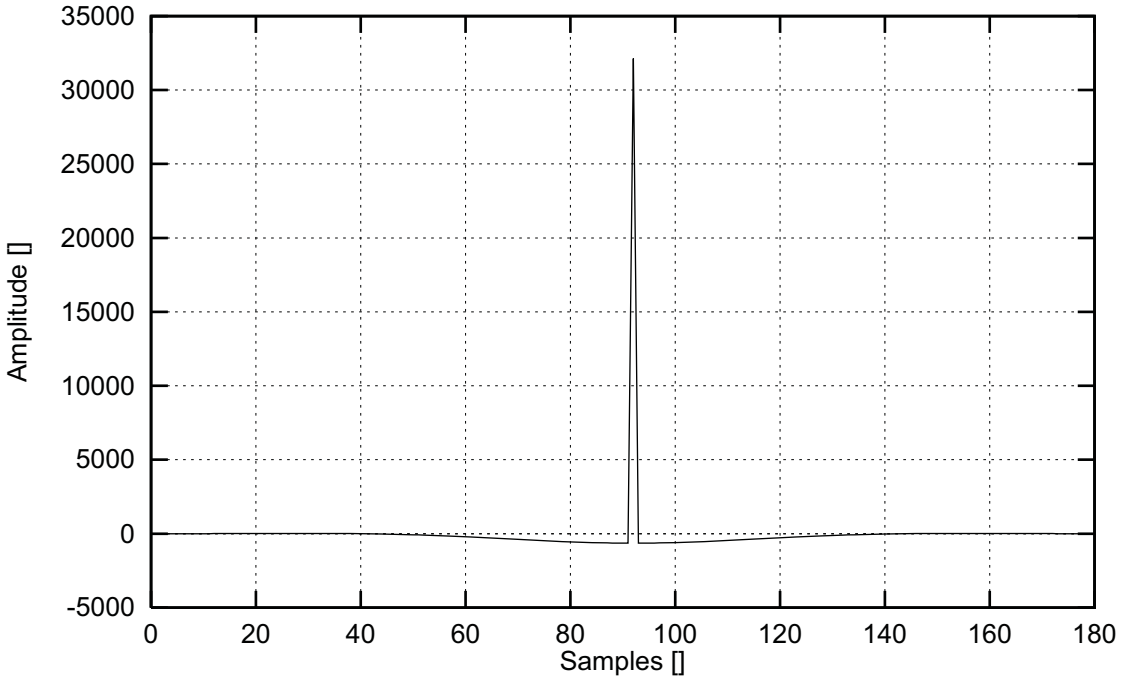
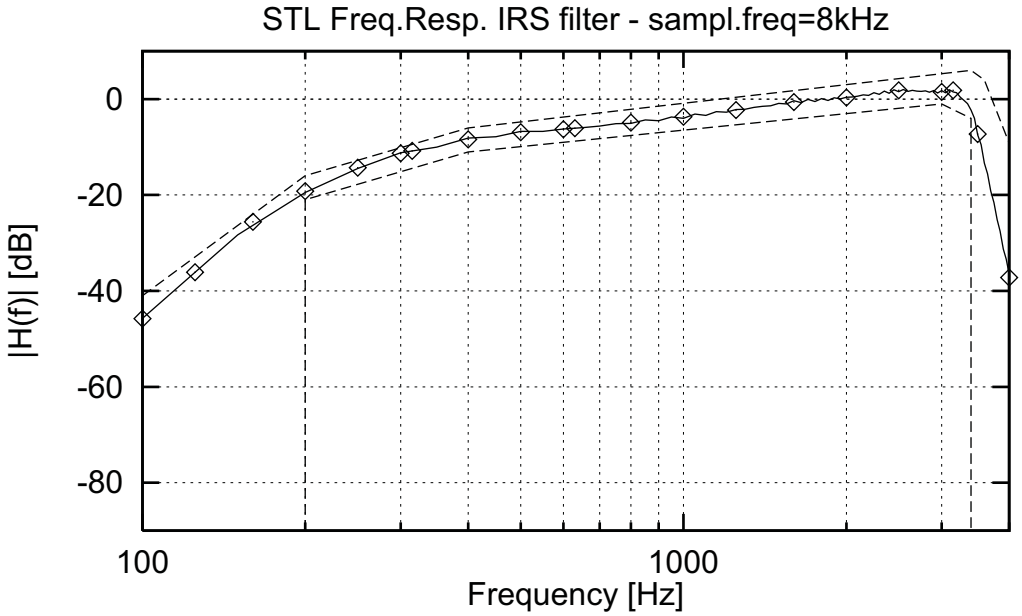
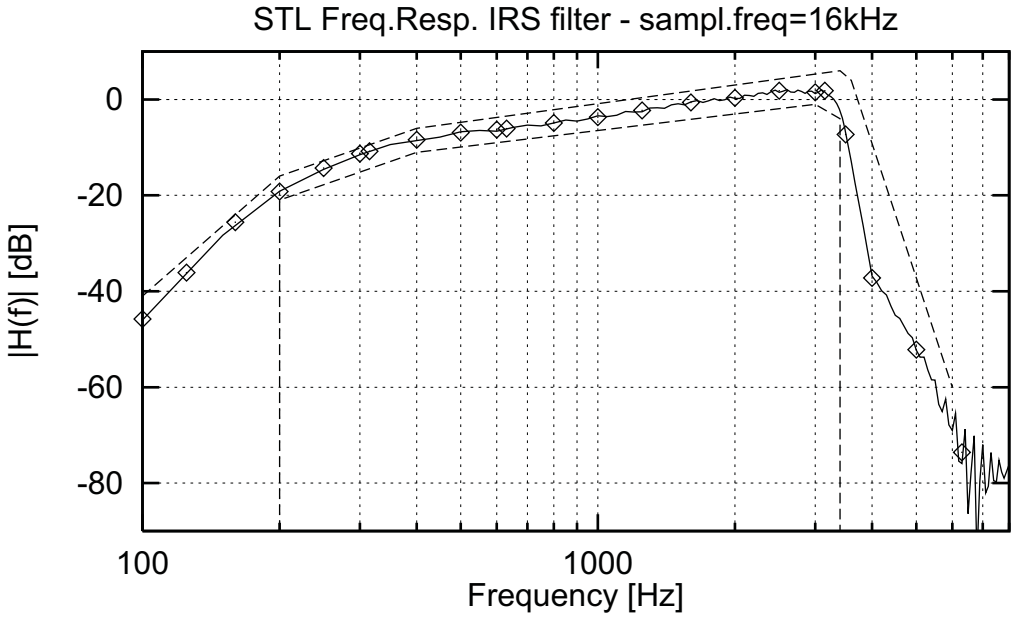


Figure 10.11: STL MSIN send-side filter impulse response.



(a) Transmission-side IRS for input samples at 8 kHz.



(b) Transmission-side IRS for input samples at 16 kHz.

Figure 10.12: Transmission-side IRS filter responses. The diamonds represent the nominal values of the "full" IRS characteristic and the interrupted line represent the mask of the "full" IRS, as shown in figure 2 of ITU-T Rec. P.48.

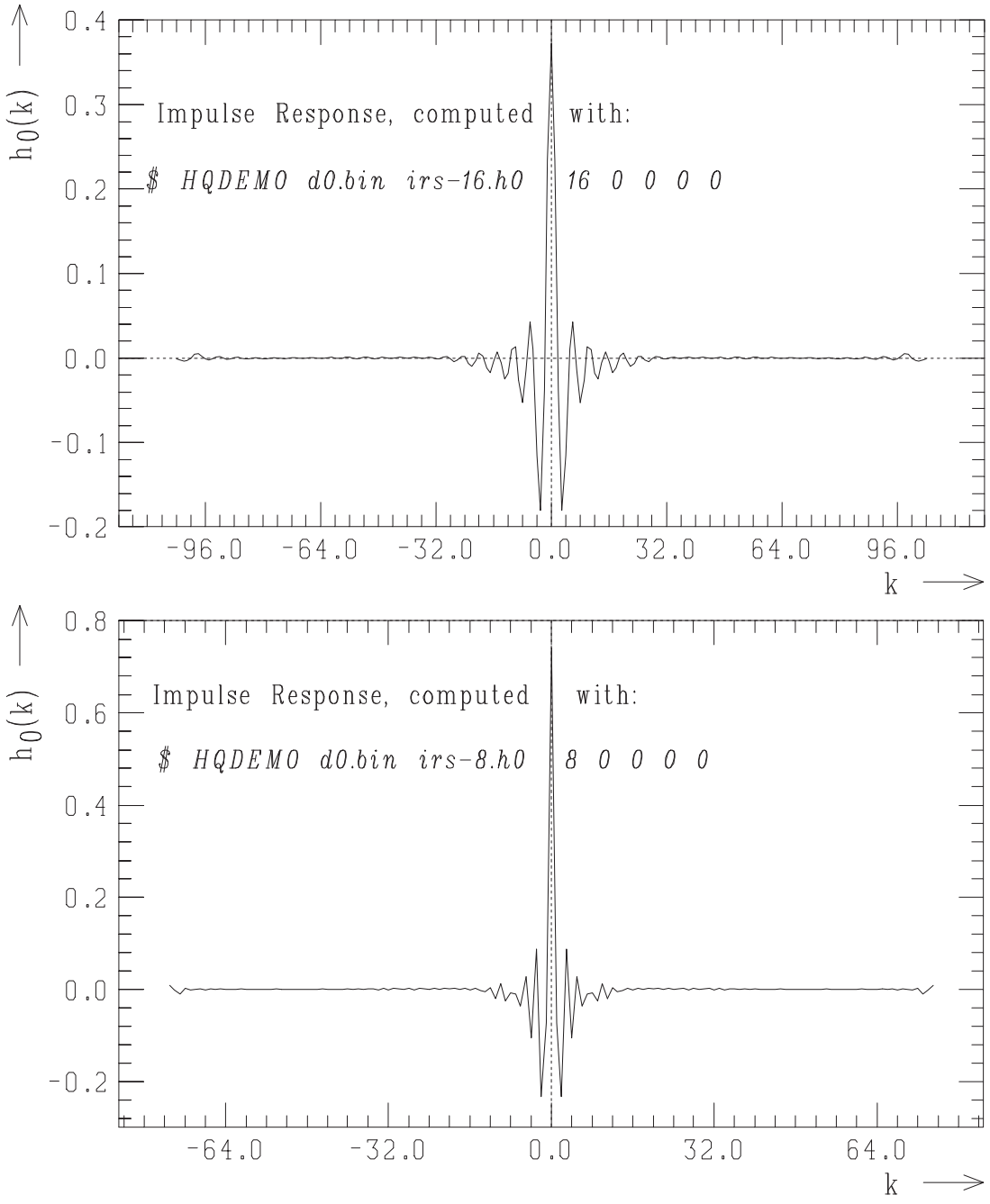
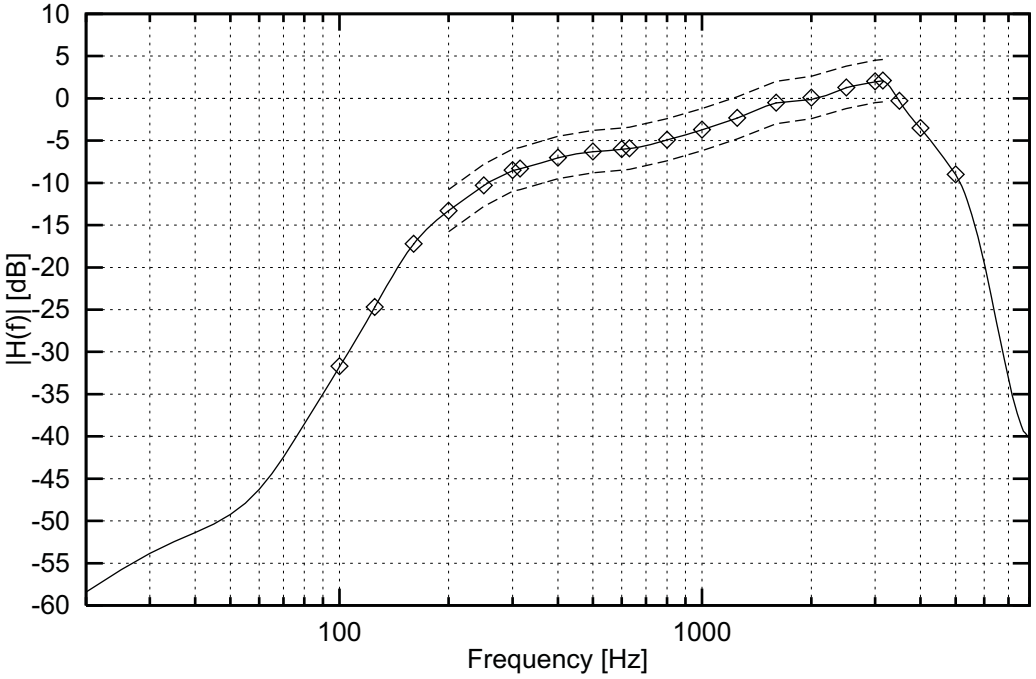
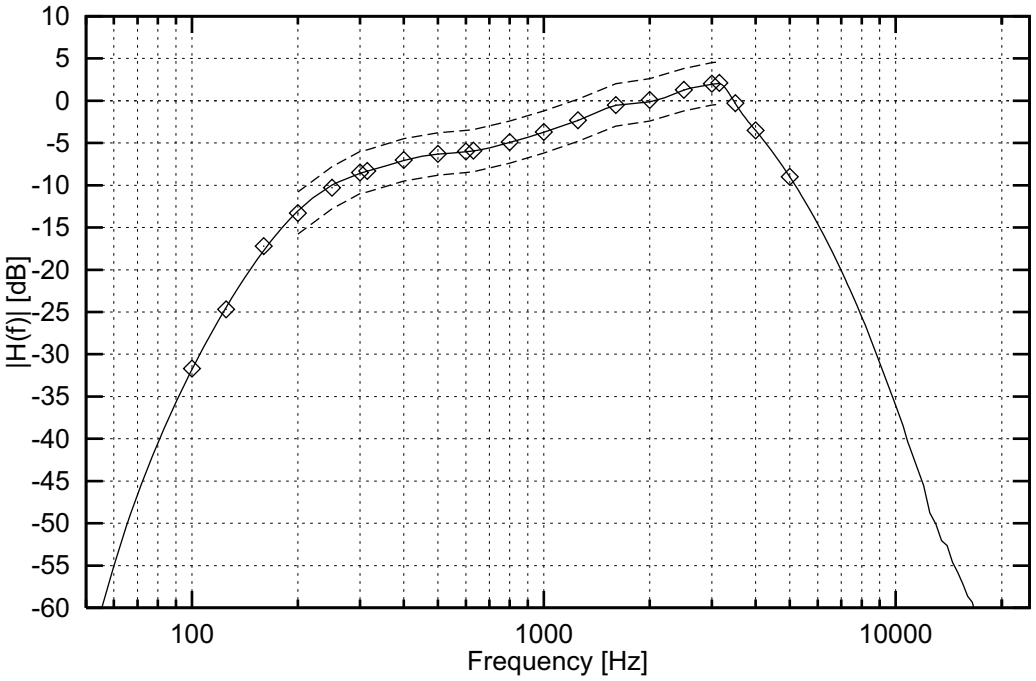


Figure 10.13: Impulse response of transmission-side IRS filters at 16 and 8 kHz.

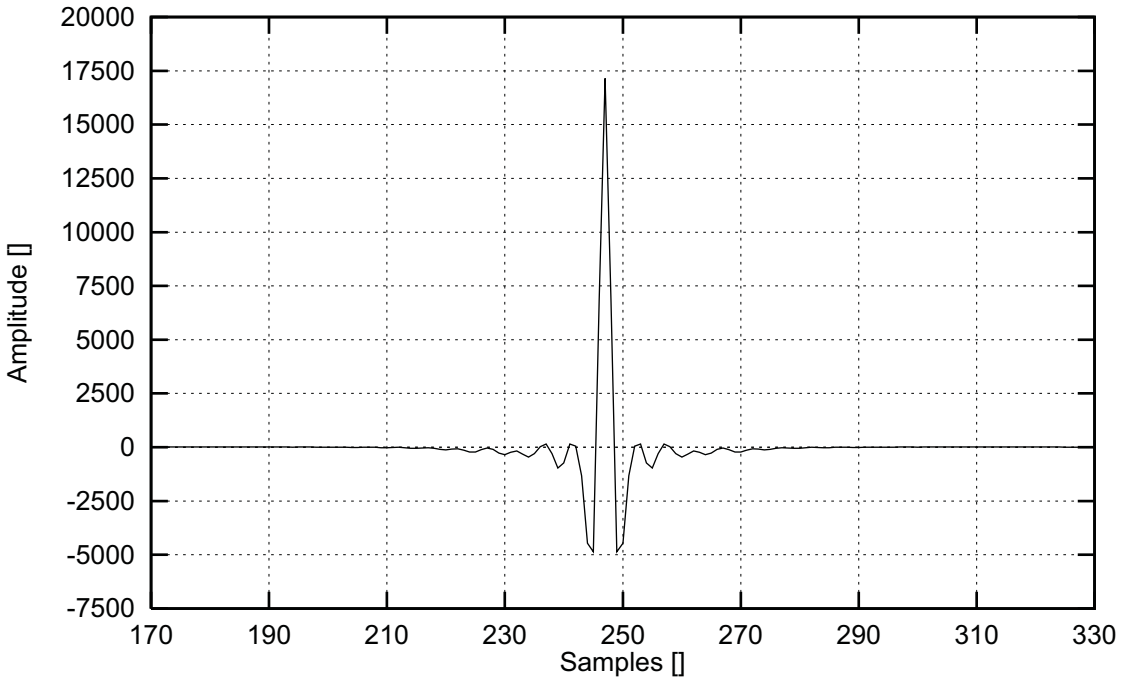


(a) Transmission-side modified IRS for input samples at 16 kHz.

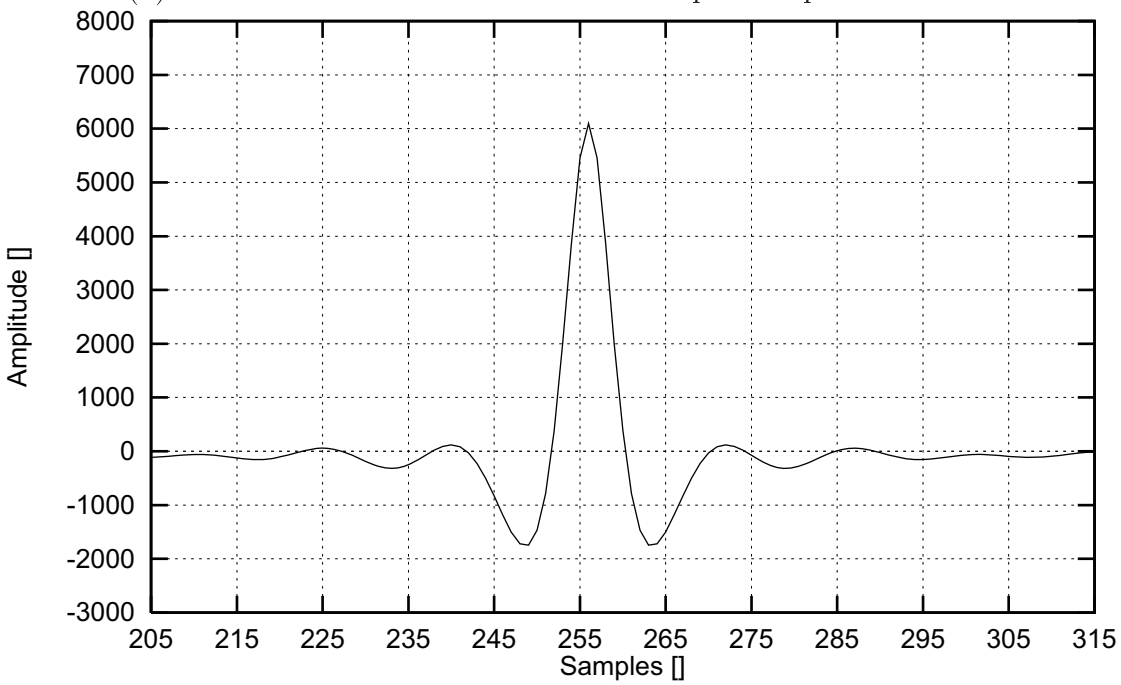


(b) Transmission-side modified IRS for input samples at 48 kHz.

Figure 10.14: Transmission-side modified IRS filter responses. The interrupted line represents the mask of the "full" IRS.

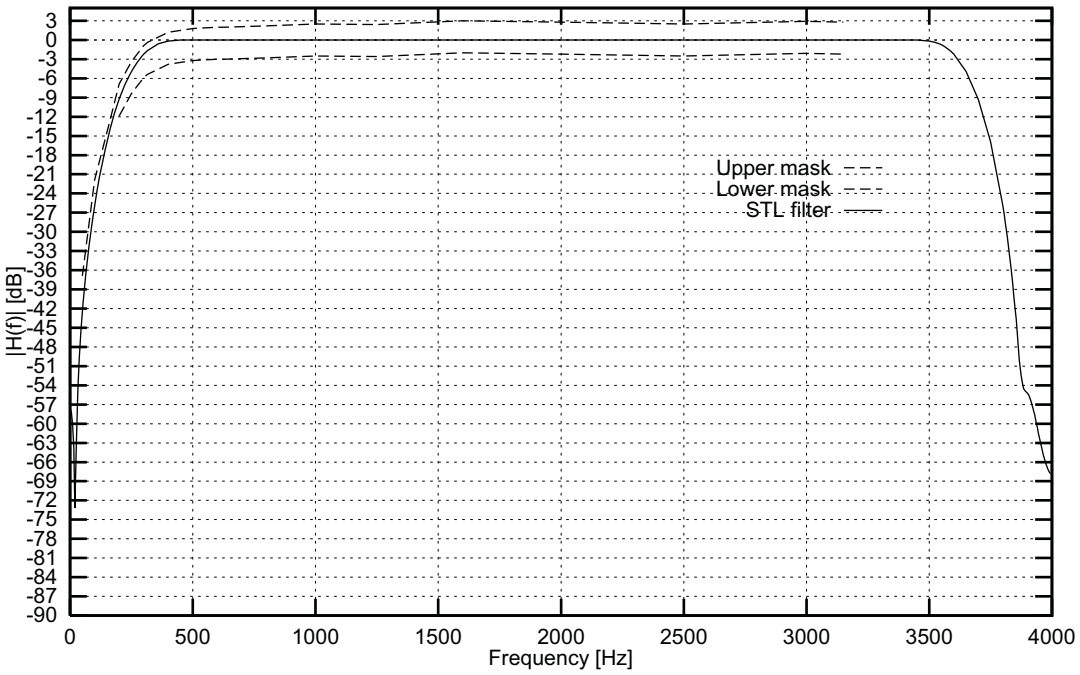


(a) Transmission-side modified IRS for input samples at 16 kHz.

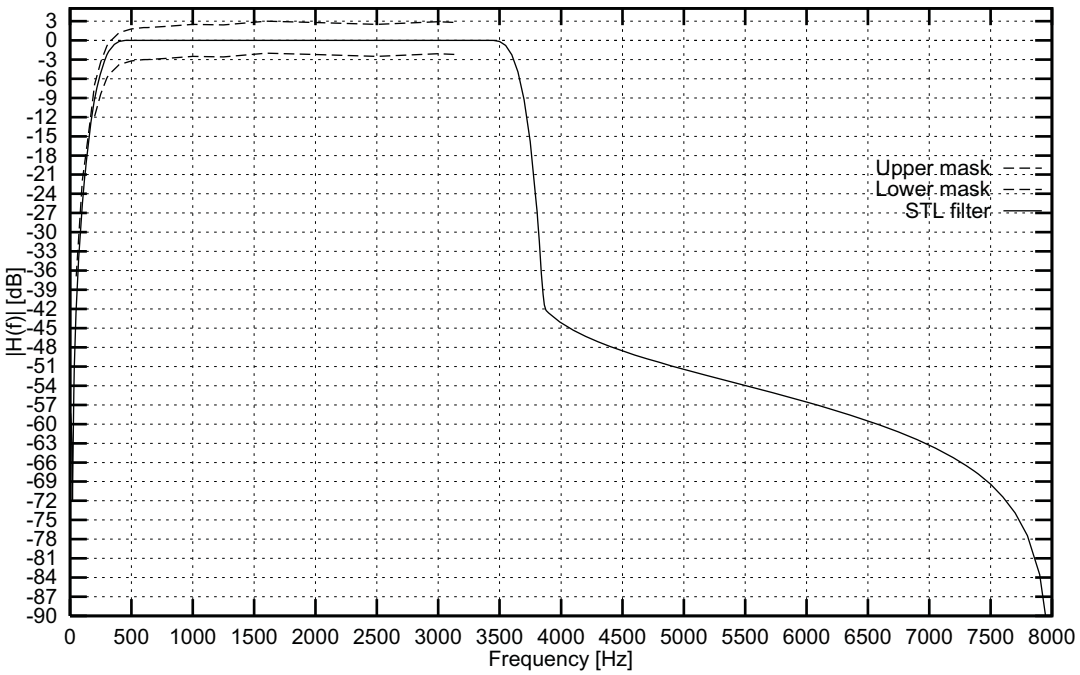


(b) Transmission-side modified IRS for input samples at 48 kHz.

Figure 10.15: Impulse response of transmission-side modified IRS filters at 16 kHz (top) and 48 kHz (bottom).

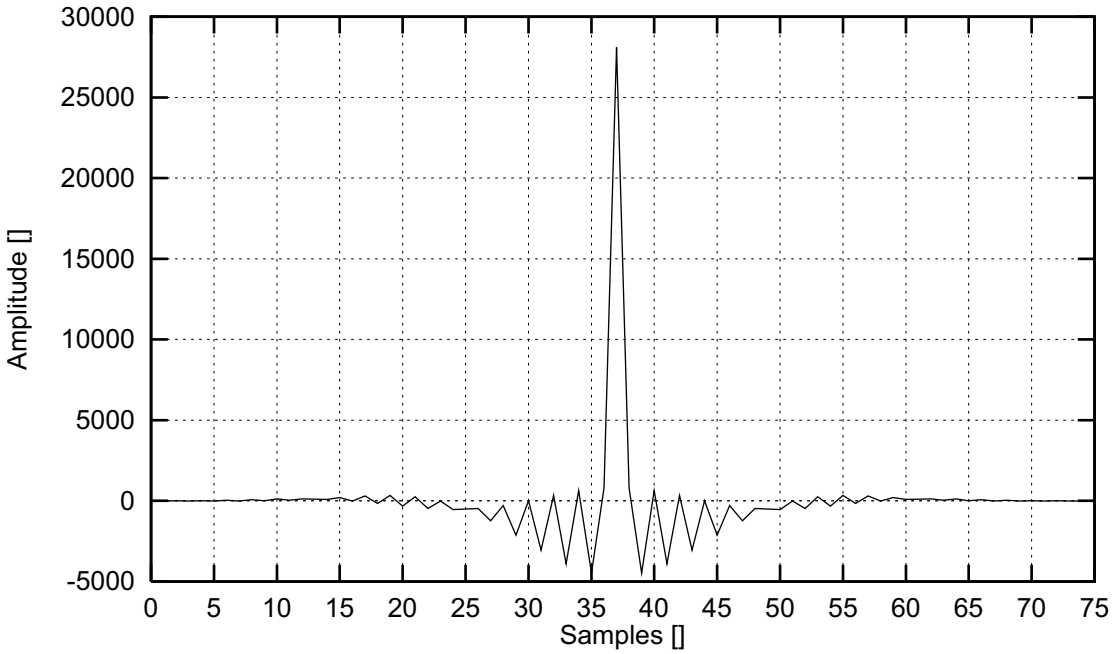


(a) Receive-side modified IRS for input samples at 8 kHz.

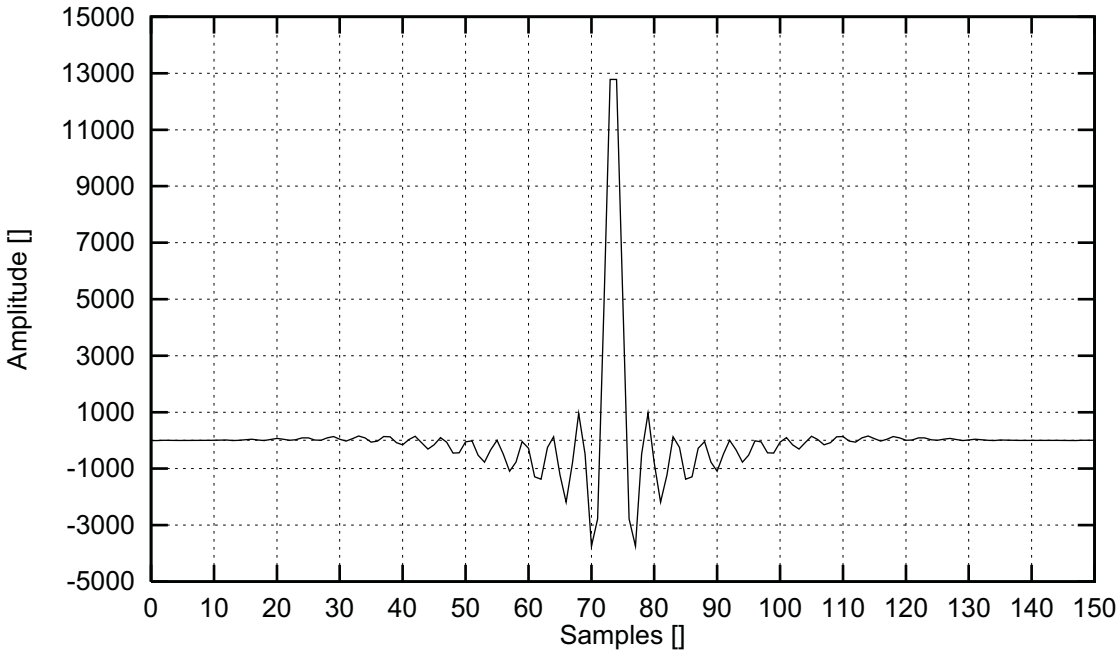


(b) Receive-side modified IRS for input samples at 16 kHz.

Figure 10.16: Receive-side modified IRS filter responses. The diamonds represent the nominal values of the modified IRS characteristic and the interrupted line represent the mask of the "full" IRS.



(a) Receive-side modified IRS for input samples at 8 kHz.



(b) Receive-side modified IRS for input samples at 16 kHz.

Figure 10.17: Impulse response of receive-side modified IRS filters at 8 kHz (top) and 16 kHz (bottom).

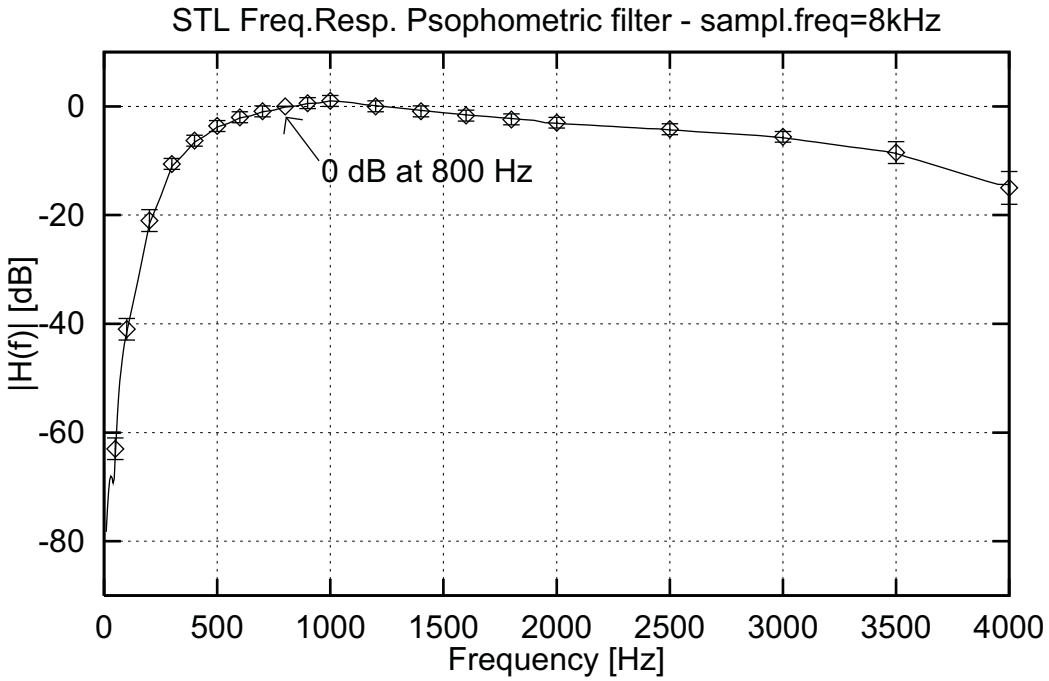


Figure 10.18: Frequency response for the psophometric filter. The points show the average points and the allowed range as per ITU-T Rec. O.41.

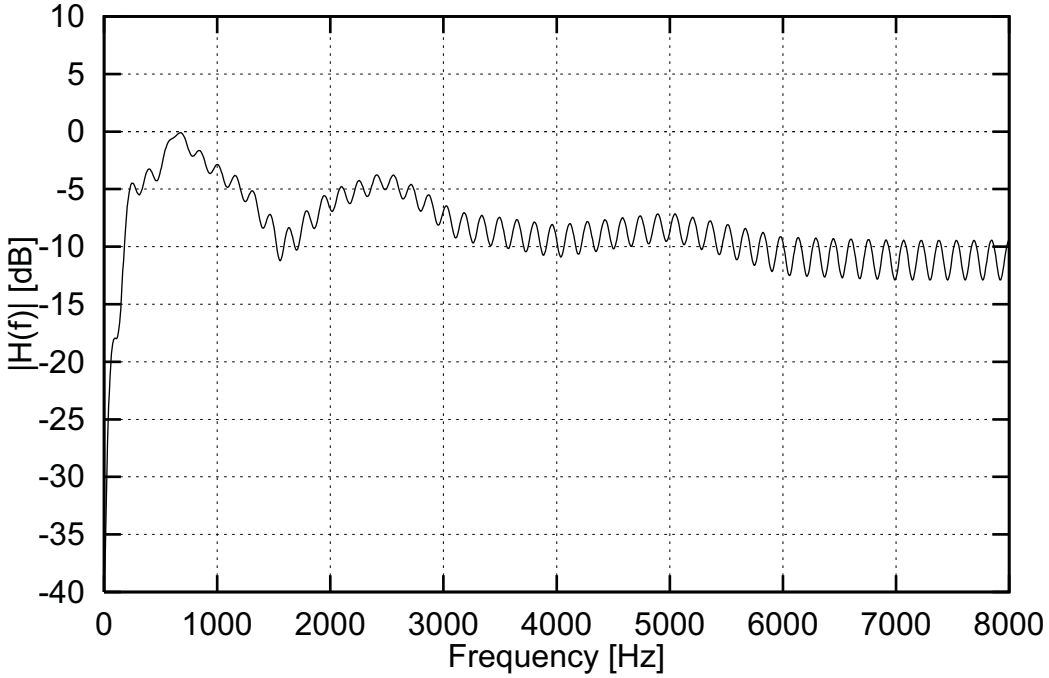


Figure 10.19: Frequency response for the Δ_{SM} filter.

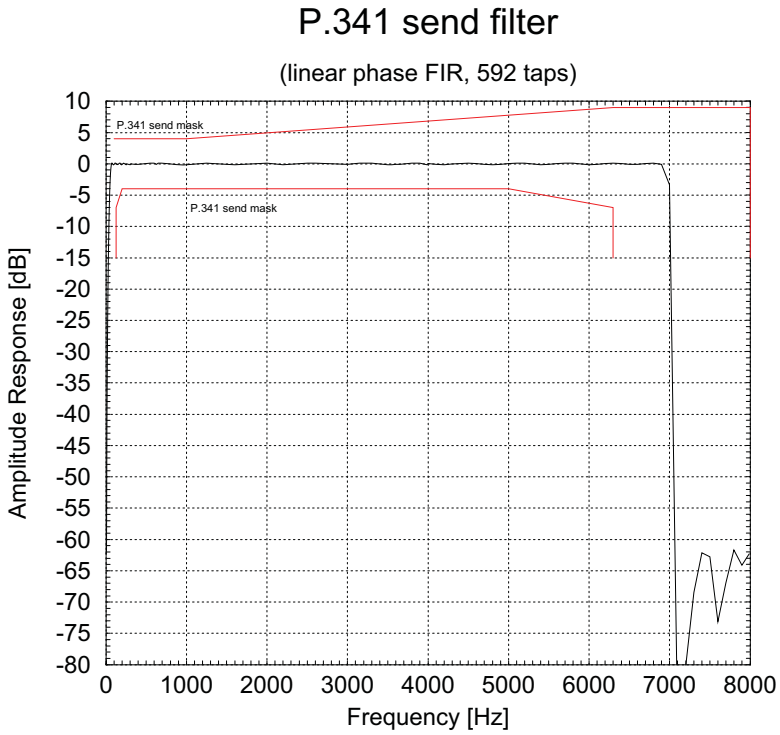


Figure 10.20: STL P.341 send-side filter frequency response for data sampled at 16 kHz (factor 1:1).

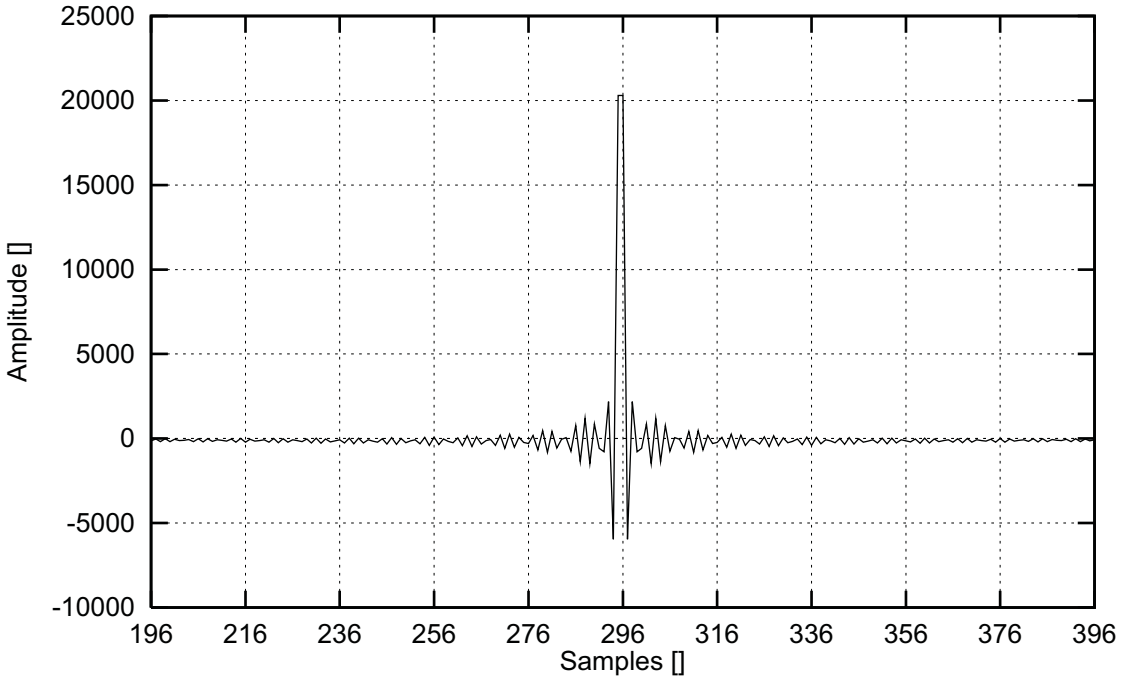


Figure 10.21: STL P.341 send-side filter impulse response.

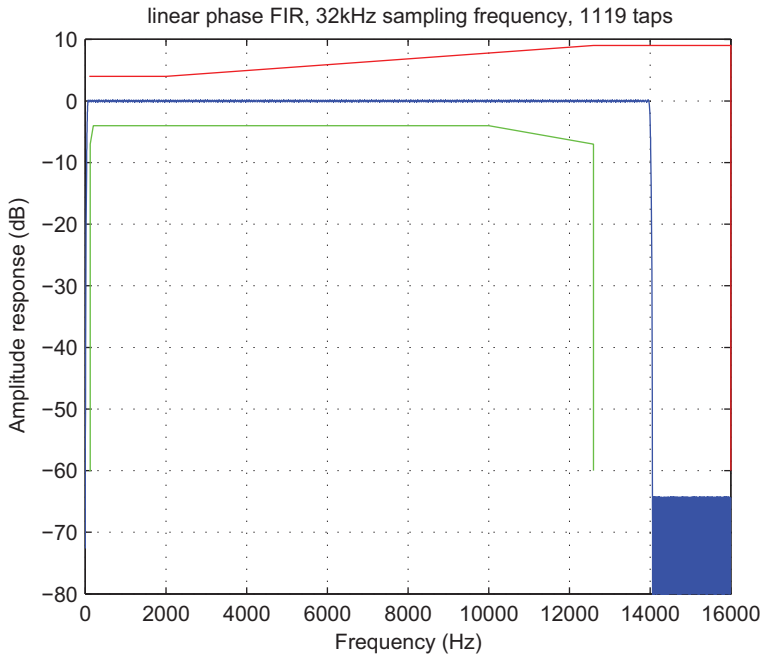


Figure 10.22: STL 50 Hz - 14 kHz band limiting filter frequency response for data sampled at 32 kHz (factor 1:1).

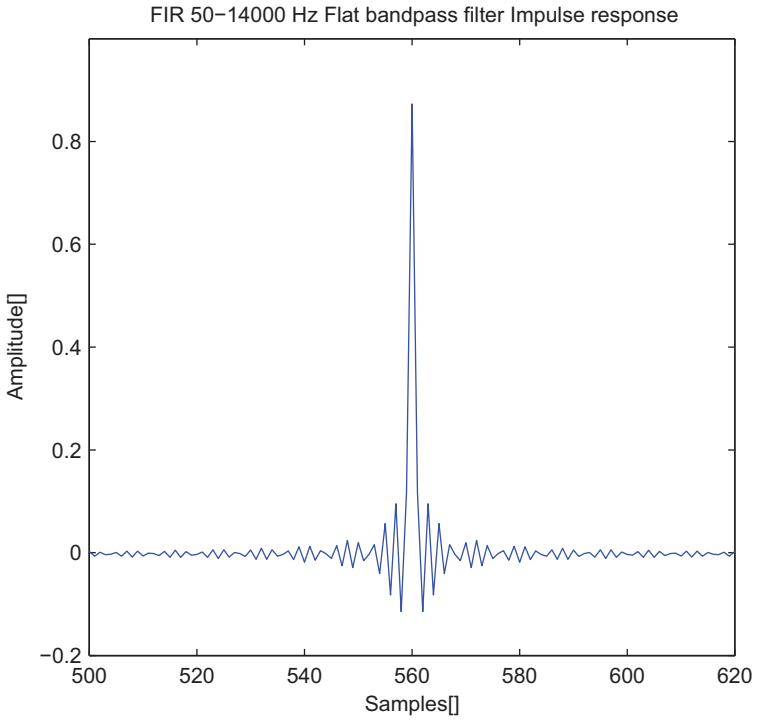


Figure 10.23: STL 50 Hz - 14 kHz band limiting filter impulse response.

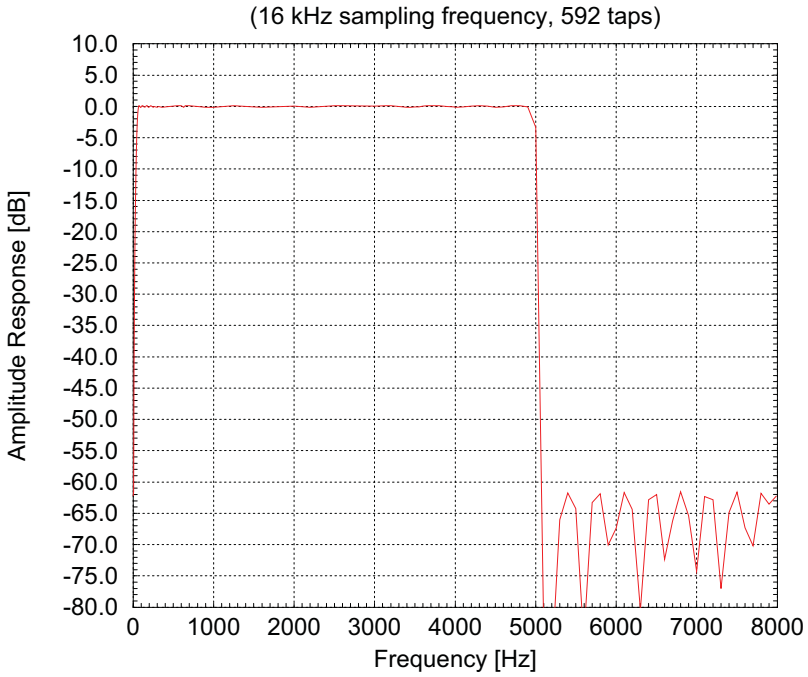


Figure 10.24: STL 50 Hz - 5 kHz band limiting filter frequency response for data sampled at 16 kHz (factor 1:1).

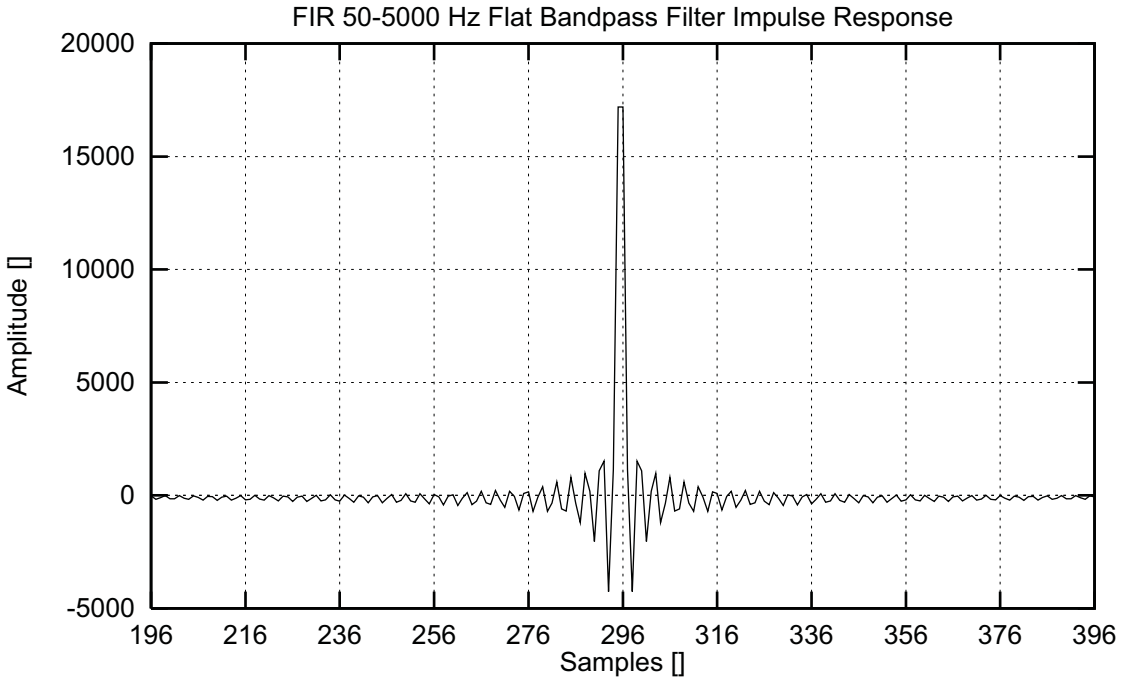


Figure 10.25: STL 50 Hz - 5 kHz band limiting filter impulse response.

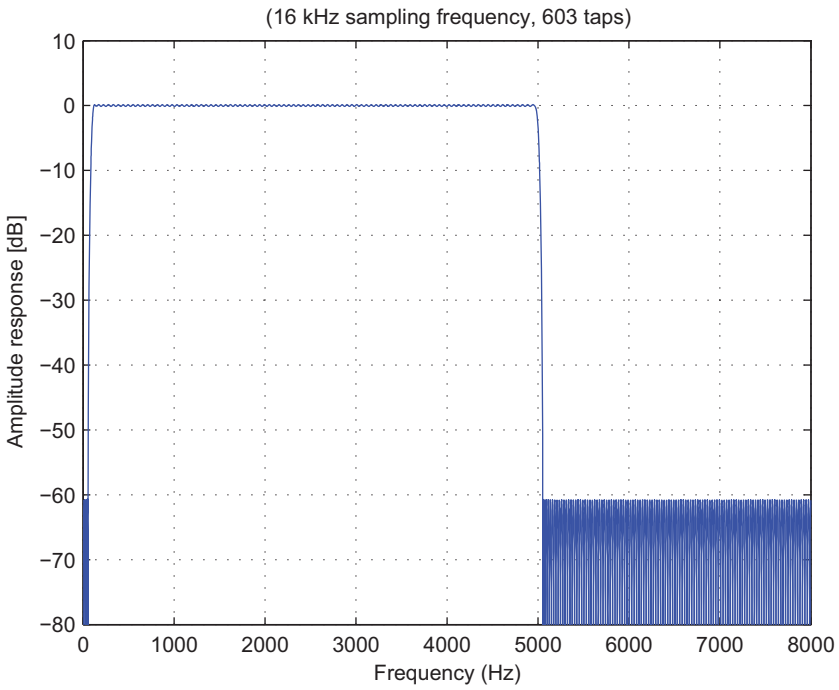


Figure 10.26: STL 100 Hz - 5 kHz band limiting filter frequency response for data sampled at 16 kHz (factor 1:1).

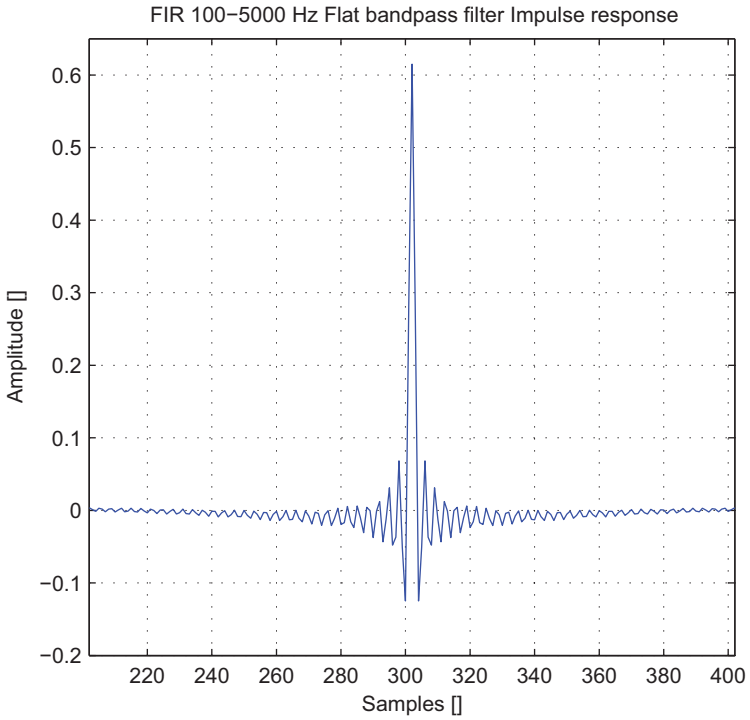


Figure 10.27: STL 100 Hz - 5 kHz band limiting filter impulse response.

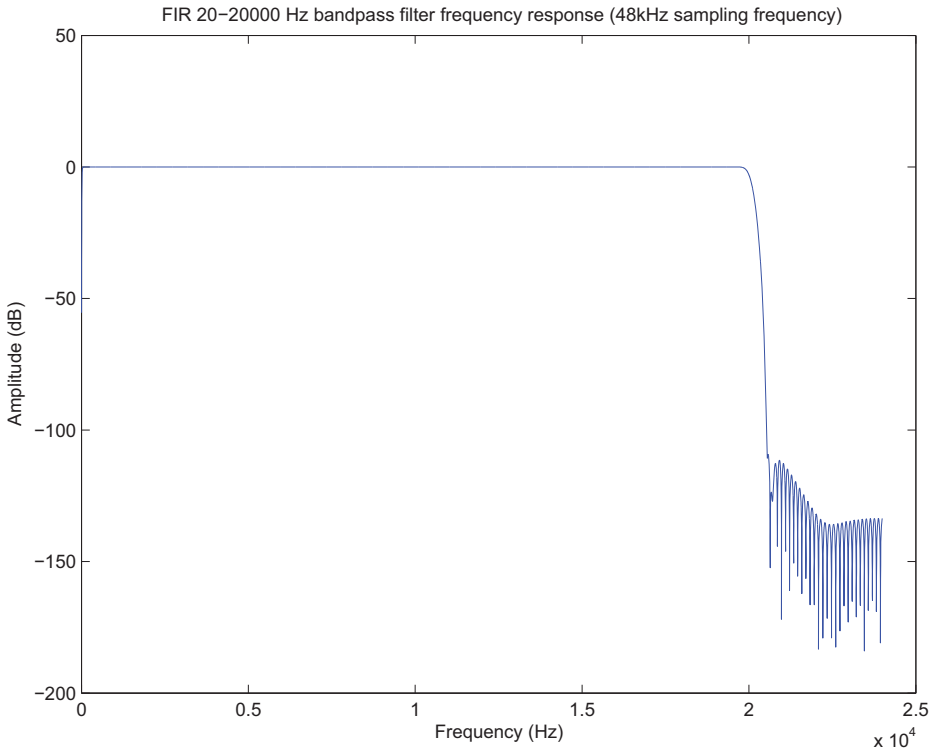


Figure 10.28: STL 20 Hz - 20 kHz band limiting filter frequency response for data sampled at 48 kHz (factor 1:1).

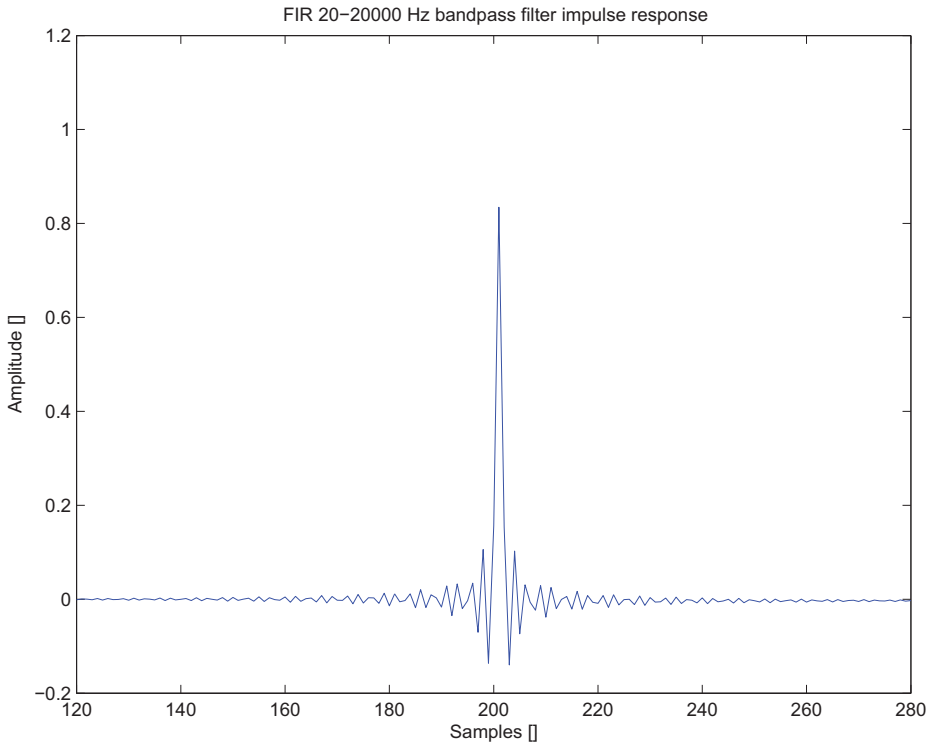


Figure 10.29: STL 20 Hz - 20 kHz band limiting filter impulse response, (the complete impulse response is of length 4001).

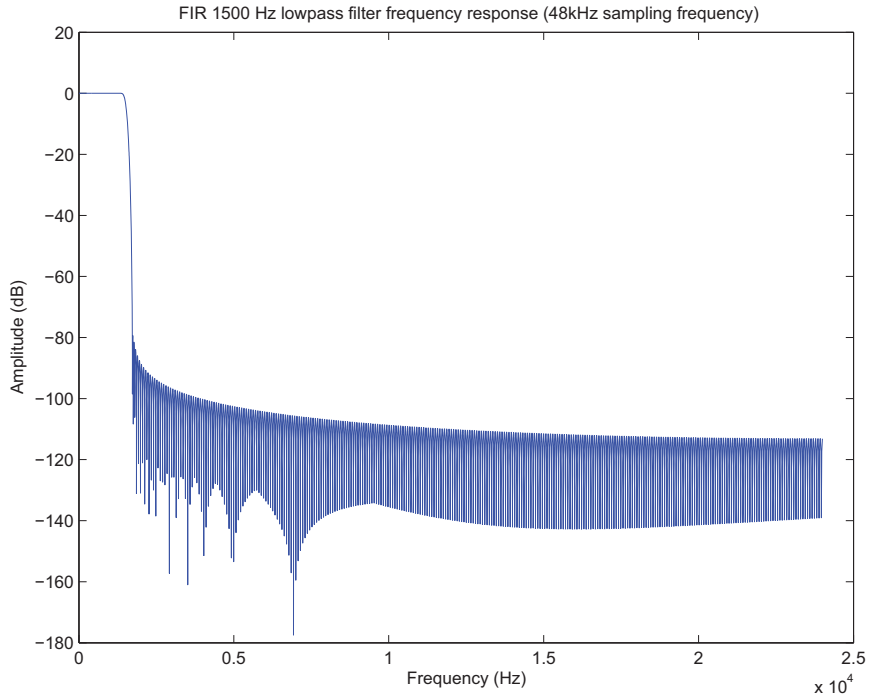


Figure 10.30: Frequency response of the STL MUSHRA anchor LP1.5 – Lowpass filter with cut-off frequency 1.5 kHz for a sampling frequency of 48 kHz (factor 1:1).

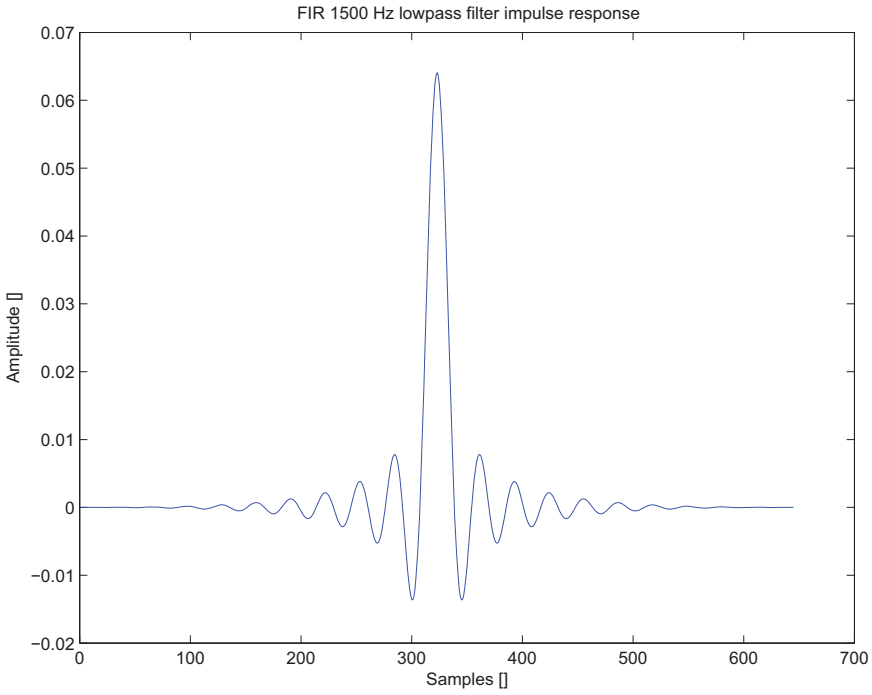


Figure 10.31: Impulse response of the STL MUSHRA anchor LP1.5 – Lowpass filter with cut-off frequency 1.5 kHz for a sampling frequency of 48 kHz (factor 1:1).

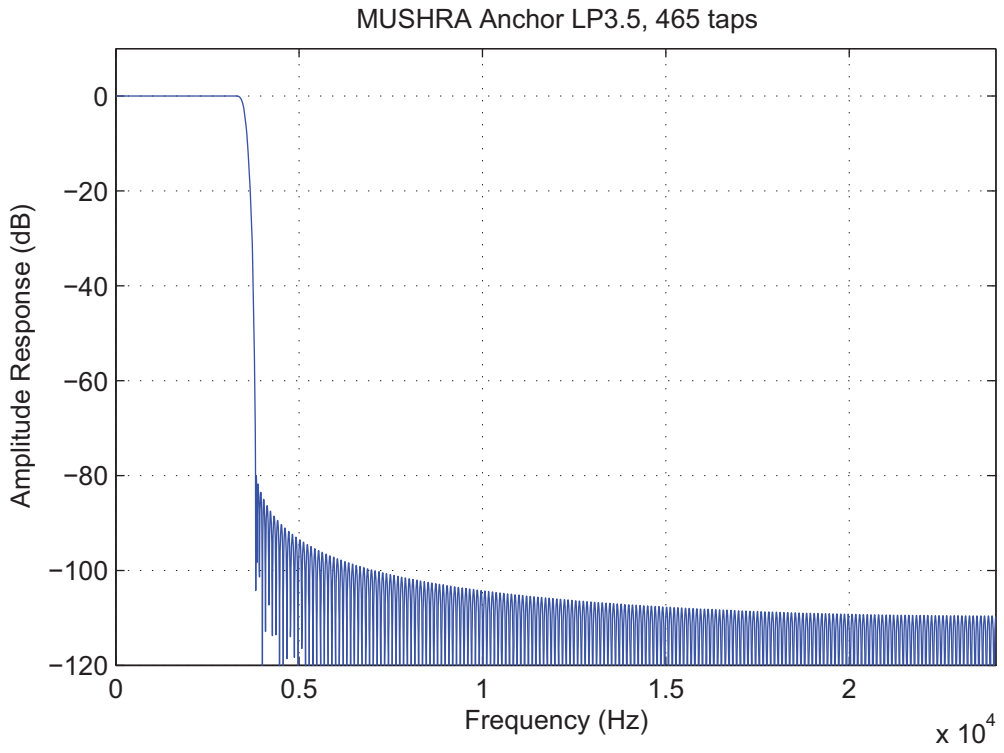


Figure 10.32: Frequency response of the STL MUSHRA anchor LP3.5 – Lowpass filter with cut-off frequency 3.5 kHz for a sampling frequency of 48 kHz (factor 1:1).

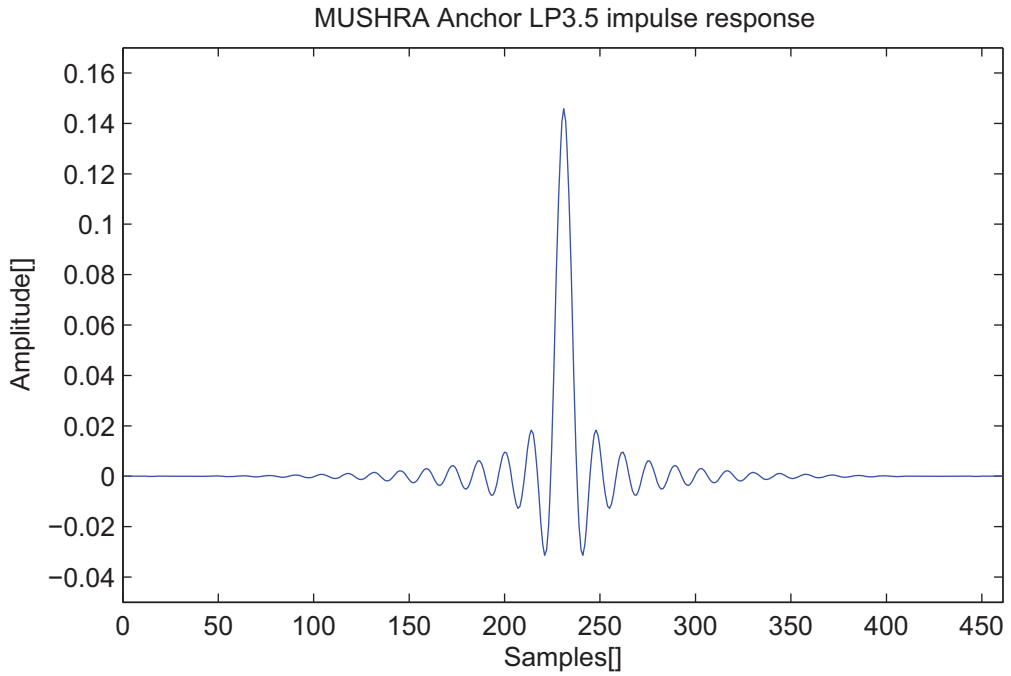


Figure 10.33: Impulse response of the STL MUSHRA anchor LP3.5 – Lowpass filter with cut-off frequency 3.5 kHz for a sampling frequency of 48 kHz (factor 1:1).

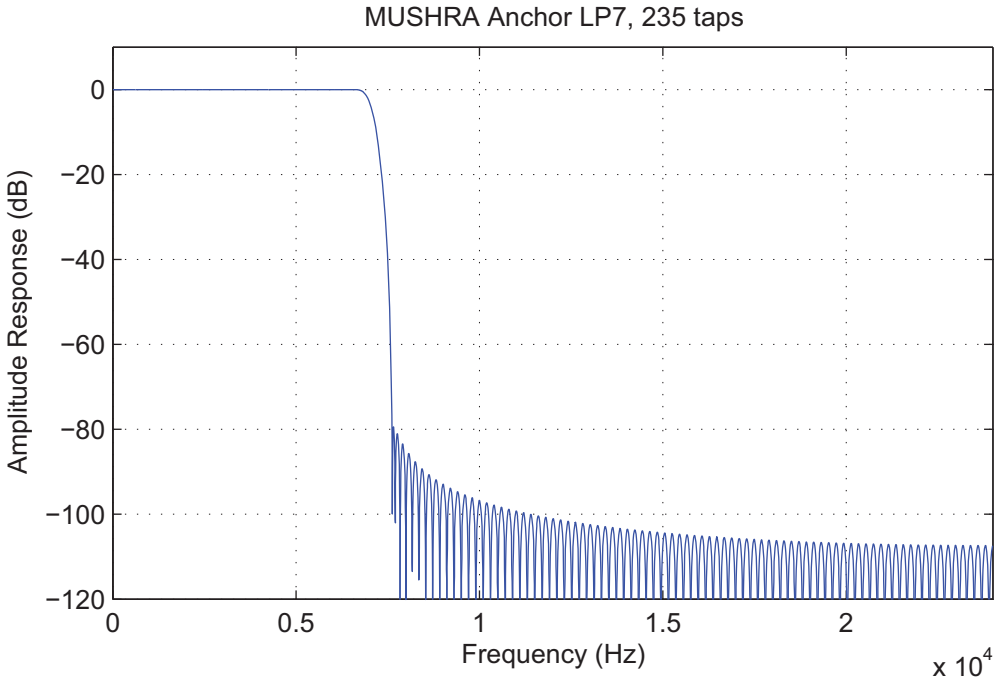


Figure 10.34: Frequency response of the STL MUSHRA anchor LP7 – Lowpass filter with cut-off frequency 7 kHz for a sampling frequency of 48 kHz (factor 1:1).

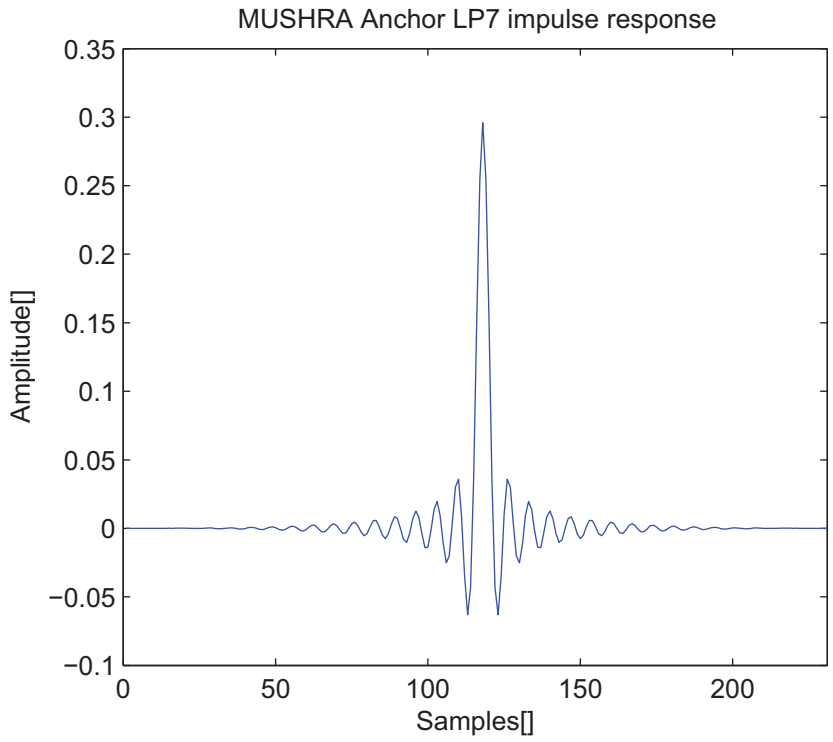


Figure 10.35: Impulse response of the STL MUSHRA anchor LP7 – Lowpass filter with cut-off frequency 7 kHz for a sampling frequency of 48 kHz (factor 1:1).

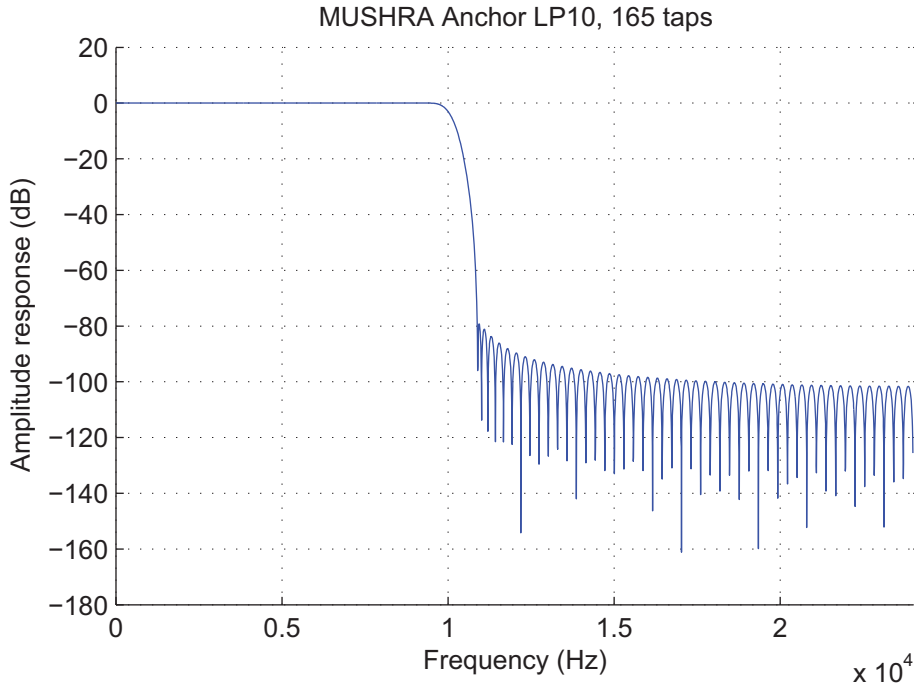


Figure 10.36: Frequency response of the STL MUSHRA anchor LP10 – Lowpass filter with cut-off frequency 10 kHz for a sampling frequency of 48 kHz (factor 1:1).

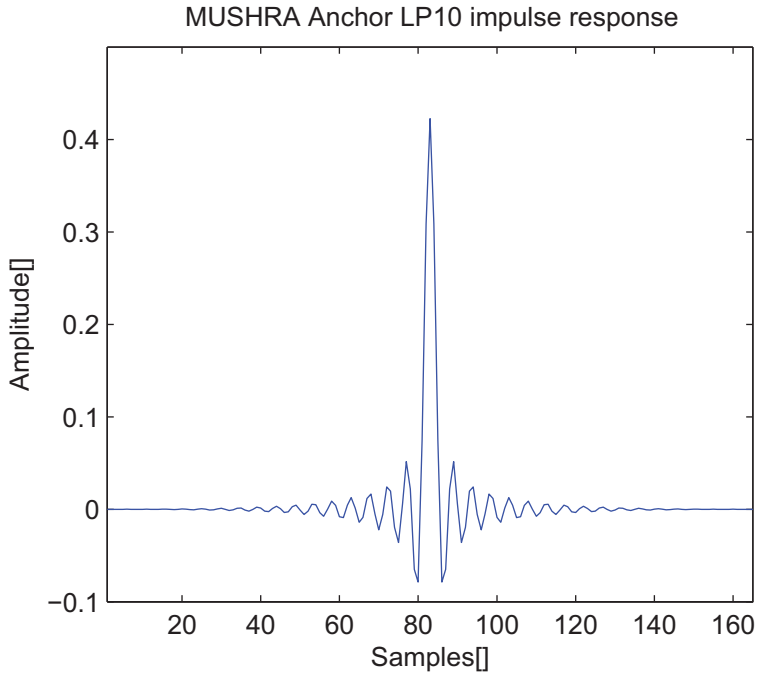


Figure 10.37: Impulse response of the STL MUSHRA anchor LP10 – Lowpass filter with cut-off frequency 10 kHz for a sampling frequency of 48 kHz (factor 1:1).

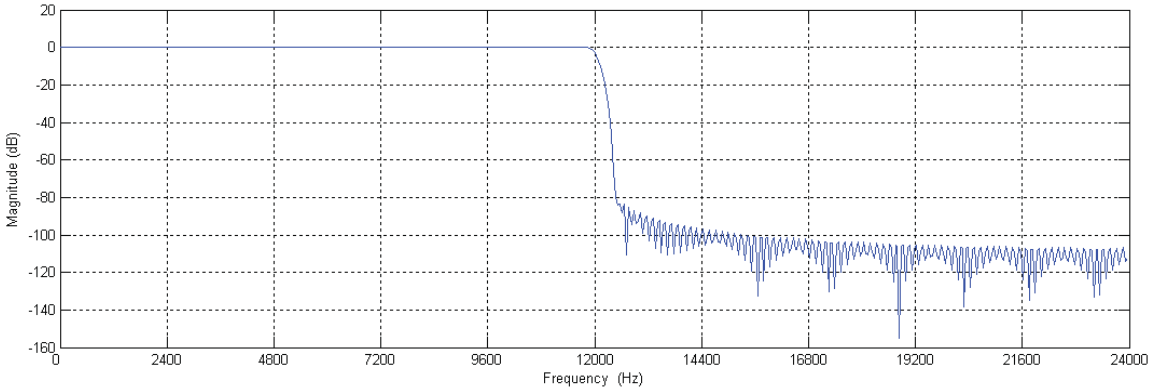


Figure 10.38: Frequency response of the STL MUSHRA anchor LP12 – Lowpass filter with cut-off frequency 12 kHz for a sampling frequency of 48 kHz (factor 1:1).

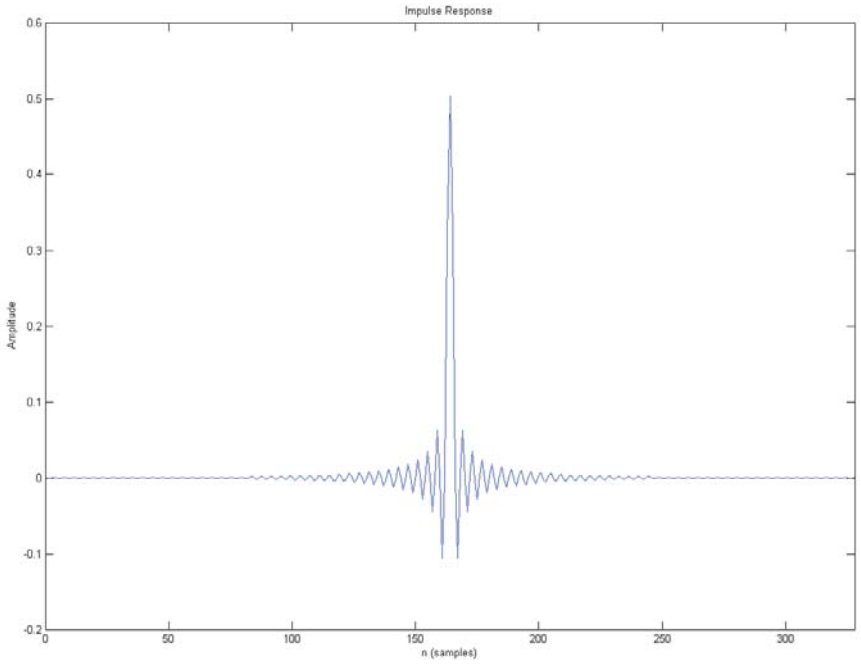


Figure 10.39: Impulse response of the STL MUSHRA anchor LP12 – Lowpass filter with cut-off frequency 12 kHz for a sampling frequency of 48 kHz (factor 1:1).

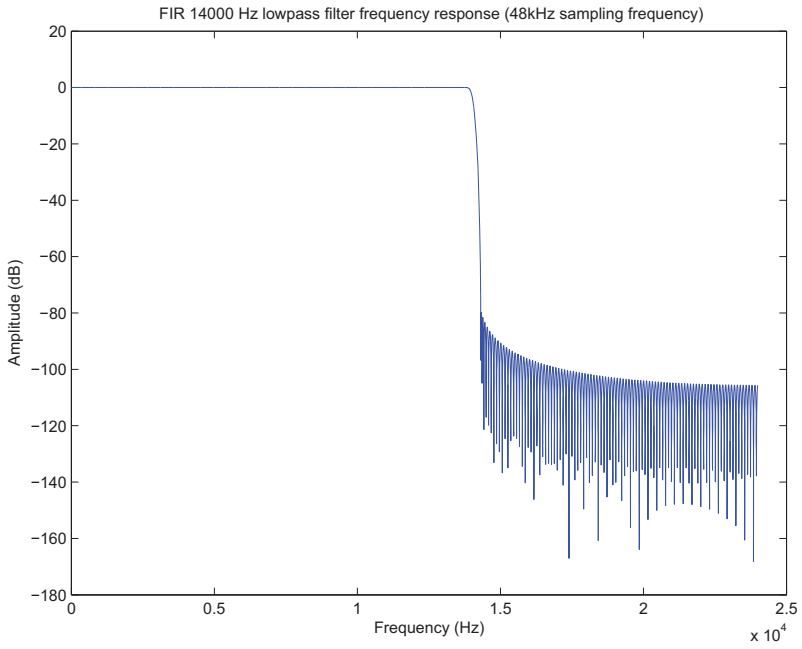


Figure 10.40: Frequency response of the STL MUSHRA anchor LP14 – Lowpass filter with cut-off frequency 14 kHz for a sampling frequency of 48 kHz (factor 1:1).

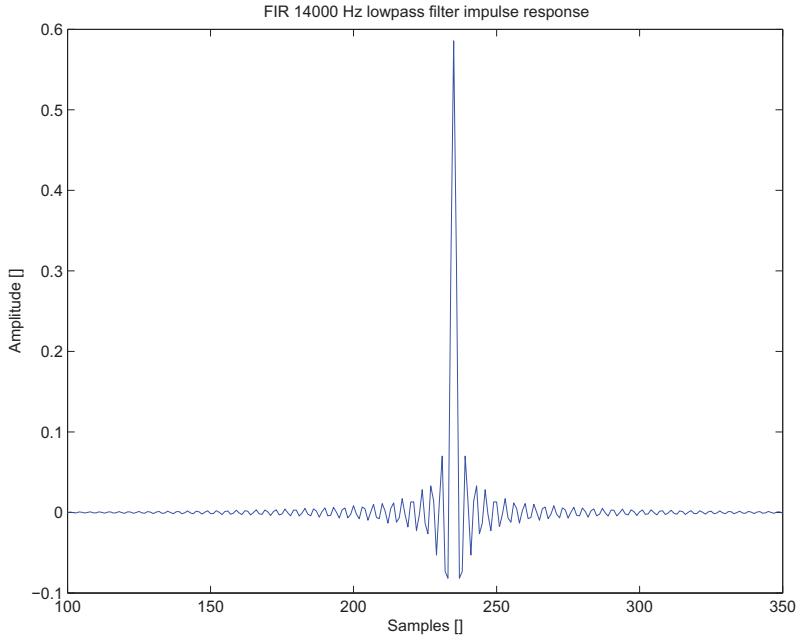


Figure 10.41: Impulse response of the STL MUSHRA anchor LP14 – Lowpass filter with cut-off frequency 14 kHz for a sampling frequency of 48 kHz (factor 1:1).

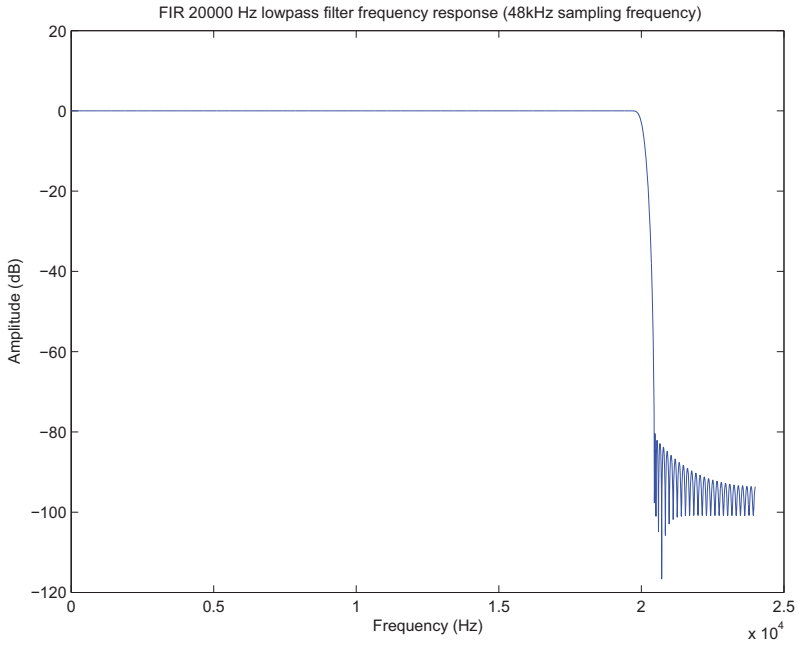


Figure 10.42: Frequency response of the STL MUSHRA anchor LP20 – Lowpass filter with cut-off frequency 20 kHz for a sampling frequency of 48 kHz (factor 1:1).

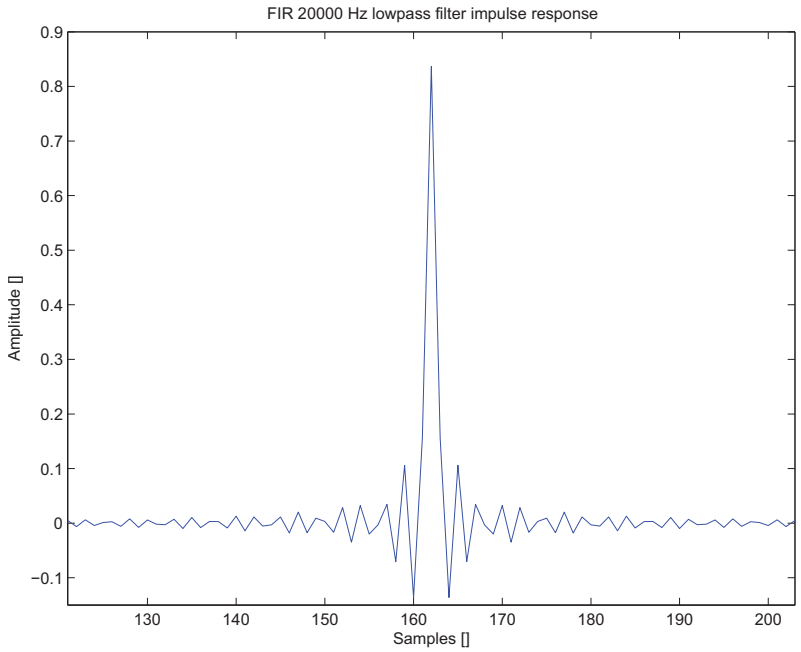


Figure 10.43: Impulse response of the STL MUSHRA anchor LP20 – Lowpass filter with cut-off frequency 20 kHz for a sampling frequency of 48 kHz (factor 1:1).

10.2.1.1 *_init for the FIR module

Syntax:

```
#include "firflt.h"
SCD_FIR *delta_sm_16khz_init (void);
SCD_FIR *hq_down_2_to_1_init (void);
SCD_FIR *hq_up_1_to_2_init (void);
SCD_FIR *hq_down_3_to_1_init (void);
SCD_FIR *hq_up_1_to_3_init (void);
SCD_FIR *irs_8khz_init (void);
SCD_FIR *irs_16khz_init (void);
SCD_FIR *linear_phase_pb_2_to_1_init (void);
SCD_FIR *linear_phase_pb_1_to_2_init (void);
SCD_FIR *linear_phase_pb_1_to_1_init (void);
SCD_FIR *msin_16khz_init();
SCD_FIR *mod_irs_16khz_init (void);
SCD_FIR *mod_irs_48khz_init (void);
SCD_FIR *rx_mod_irs_8khz_init(void);
SCD_FIR *rx_mod_irs_16khz_init(void);
SCD_FIR *psophometric_8khz_init (void);
SCD_FIR *p341_16k_init (void);
SCD_FIR *bp14k_32khz_init (void);
SCD_FIR *bp5k_16k_init (void);
SCD_FIR *bp100_5k_16khz_init (void);
SCD_FIR *bp20k_48kHz_init (void);
SCD_FIR *LP1p5_48kHz_init (void);
SCD_FIR *LP35_48kHz_init (void);
SCD_FIR *LP7_48kHz_init (void);
SCD_FIR *LP10_48kHz_init (void);
SCD_FIR *LP12_48kHz_init (void);
SCD_FIR *LP14_48kHz_init (void);
SCD_FIR *L20_48kHz_init (void);
```

Prototypes: firflt.h

Description:

`delta_sm_16khz_init` is the initialization routine for the Δ_{SM} weighting filter for data sampled at 16 kHz using a linear phase FIR filter structure. Input and output signals will be at 16 kHz. Code is in file `fir-dsm.c` and its frequency response is given in figure 10.19.

`hq_up_1_to_2_init` is the initialization routine for high quality FIR up-sampling filtering by a factor of 2. The -3 dB point for this filter is located at approximately 3660 Hz. Code is in file `fir-flat.c` and its frequency and impulse response are given in figures 10.4(a) and 10.7 (top), respectively.

`hq_down_2_to_1_init` is the initialization routine for high quality FIR down-sampling filtering by a factor of 2. The -3 dB point for this filter is located at approximately 3660

Hz. Code is in file `fir-flat.c` and its frequency and impulse response are given in figures 10.4(b) and 10.8 (top), respectively.

`hq_up_1_to_3_init` is the initialization routine for high quality FIR up-sampling filter by factor of 3. The -3 dB point for this filter is located at approximately 3650 Hz. Code is in file `fir-flat.c` and its frequency and impulse response are given in figures 10.5(a) and 10.7 (bottom), respectively.

`hq_down_3_to_1_init` is the initialization routine for high quality FIR down-sampling filtering by a factor of 3. The -3 dB point for this filter is located at approximately 3650 Hz. Code is in file `fir-flat.c` and its frequency and impulse response are given in figures 10.5(b) and 10.8 (bottom), respectively.

`linear_phase_bp_1_to_2_init` is the initialization routine for bandpass, FIR up-sampling filtering by a factor of 2. The -3 dB points for this filter are located at approximately 98 and 3460 Hz. Code is in file `fir-flat.c` and its frequency and impulse response are given in figures 10.6(b) and 10.9(b), respectively.

`linear_phase_bp_2_to_1_init` is the initialization routine for bandpass, FIR down-sampling filtering by a factor of 2. The -3 dB points for this filter are located at approximately 98 and 3460 Hz. Code is in file `fir-flat.c` and its frequency and impulse response are given in figures 10.6(a) and 10.9(a), respectively.

`linear_phase_bp_1_to_1_init` is the initialization routine for FIR 1:1 bandpass filtering. The -3 dB points for this filter are located at approximately 98 and 3460 Hz. Code is in file `fir-flat.c` and its frequency and impulse response are given in figures 10.6(a) and 10.9(a), respectively.

`msin_16khz_init` is the initialization routine for the high-pass, FIR 1:1 filter that simulates a mobile station input characteristic. The -3 dB point for this filter is located at approximately 195 Hz. Code is in file `fir-flat.c` and its frequency and impulse response are given in figures 10.10 and 10.11, respectively.

`irs_8khz_init` is the initialization routine for the transmit-side IRS weighting filter for data sampled at 8 kHz using a linear phase FIR filter structure. Input and output signals will be at 8 kHz. Code is in file `fir-irs.c` and its frequency and impulse response are given in figures 10.12 and 10.13, respectively.

`irs_16khz_init` is the initialization routine for the transmit-side IRS weighting filter for data sampled at 16 kHz using a linear phase FIR filter structure. Input and output signals will be at 16 kHz. Code is in file `fir-irs.c` and its frequency and impulse response are given in figures 10.12 and 10.13, respectively.

`mod_irs_16khz_init` is the initialization routine for the transmit-side modified IRS weighting filter for data sampled at 16 kHz using a linear phase FIR filter structure. Input and output signals will be at 16 kHz since no rate change is performed by this function. Code is in file `fir-irs.c` and its frequency and impulse response are given in figures 10.14 and 10.15, respectively.

`mod_irs_48khz_init` is the initialization routine for the transmit-side modified IRS weighting filter for data sampled at 48 kHz using a linear phase FIR filter structure. Input and output signals will be at 48 kHz since no rate change is performed by this function. Code is in file `fir-irs.c` and its frequency and impulse response are given in figures 10.14 and 10.15, respectively.

`rx_mod_irs_8khz_init` is the initialization routine for the receive-side modified IRS weighting filter for data sampled at 8 kHz using a linear phase FIR filter structure. The -3 dB points for this filter are located at approximately 285 Hz and 3610 Hz. Input and output signals will be at 8 kHz since no rate change is performed by this function. Code is in file `fir-irs.c` and its frequency and impulse response are given in figures 10.16 and 10.17, respectively.

`rx_mod_irs_16khz_init` is the initialization routine for the receive-side modified IRS weighting filter for data sampled at 16 kHz using a linear phase FIR filter structure. The -3 dB points for this filter are located at approximately 285 Hz and 3610 Hz. Input and output signals will be at 16 kHz since no rate change is performed by this function. Code is in file `fir-irs.c` and its frequency and impulse response are given in figures 10.16 and 10.17, respectively.

`psophometric_8khz_init` is the initialization routine for the O.41 psophometric weighting filter for data sampled at 8 kHz using a linear phase FIR filter structure. Input and output signals will be at 8 kHz since no rate change is performed by this function. Code is in file `fir-pso.c` and its frequency response is given in figure 10.18.

`p341_16khz_init` is the initialization routine for the P.341 send-side weighting filter for data sampled at 16 kHz. Input and output signals will be at 16 kHz since no rate change is performed by this function. Its frequency response is shown in figure 10.20 and its impulse response is shown in figure 10.21. The -3 dB points for this filter are located at approximately 50 and 7000 Hz. Code is in file `fir-wb.c`.

`bp14k_32khz_init` is the initialization routine for the [50 Hz - 14 kHz] filter for data sampled at 32 kHz. Input and output signals will be at 32 kHz since no rate change is performed by this function. Its frequency response is shown in figure 10.22 and its impulse response is shown in figure 10.23. The -3 dB points for this filter are located at approximately 50 and 14000 Hz. Code is in file `fir-wb.c`.

`bp5k_16khz_init` is the initialization routine for a 50 Hz - 5 kHz band limiting filter for wideband signals sampled at 16 kHz. Input and output signals will be at 16 kHz since no rate change is performed by this function. Its frequency response is shown in figure 10.24 and its impulse response is shown in figure 10.25. The -3 dB points for this filter are located at approximately 50 and 4990 Hz. Code is in file `fir-wb.c`.

`bp100_5k_16khz_init` is the initialization routine for a 100 Hz - 5 kHz band limiting filter for wideband signals sampled at 16 kHz. Input and output signals will be at 16 kHz since no rate change is performed by this function. Its frequency response is shown in figure 10.26 and its impulse response is shown in figure 10.27. The -3 dB points for this filter are located at approximately 100 and 5000 Hz. Code is in file `fir-wb.c`.

`bp20k_48khz_init` is the initialization routine for the [20 Hz - 20 kHz] filter for data sampled at 48 kHz. Input and output signals will be at 48 kHz since no rate change is performed by this function. Its frequency response is shown in figure 10.28 and its impulse response is shown in figure 10.29. The -3 dB points for this filter are located at approximately 20 and 20000 Hz. Code is in file `fir-wb.c`.

`LP1p5_48kHz_init` is the initialization routine for a 1.5 kHz lowpass filter for signals sampled at 48 kHz. Input and output signals will be at 48 kHz since no rate change is performed by this function. Its frequency response is shown in figure 10.30 and its impulse response is shown in figure 10.31. The -3 dB point for this filter is located at approximately 1500 Hz. Code is in file `fir-LP.c`.

`LP35_48kHz_init` is the initialization routine for a 3.5 kHz lowpass filter for signals sampled at 48 kHz. Input and output signals will be at 48 kHz since no rate change is performed by this function. Its frequency response is shown in figure 10.32 and its impulse response is shown in figure 10.33. The -3 dB point for this filter is located at approximately 3500 Hz. Code is in file `fir-LP.c`.

`LP7_48kHz_init` is the initialization routine for a 7 kHz lowpass filter for signals sampled at 48 kHz. Input and output signals will be at 48 kHz since no rate change is performed by this function. Its frequency response is shown in figure 10.34 and its impulse response is shown in figure 10.35. The -3 dB point for this filter is located at approximately 7000 Hz. Code is in file `fir-LP.c`.

`LP10_48kHz_init` is the initialization routine for a 10 kHz lowpass filter for signals sampled at 48 kHz. Input and output signals will be at 48 kHz since no rate change is performed by this function. Its frequency response is shown in figure 10.36 and its impulse response is shown in figure 10.37. The -3 dB point for this filter is located at approximately 10000 Hz. Code is in file `fir-LP.c`.

`LP12_48kHz_init` is the initialization routine for a 12 kHz lowpass filter for signals sampled at 48 kHz. Input and output signals will be at 48 kHz since no rate change is performed by this function. Its frequency response is shown in figure 10.38 and its impulse response is shown in figure 10.39. The -3 dB point for this filter is located at approximately 14000 Hz. Code is in file `fir-LP.c`.

`LP14_48kHz_init` is the initialization routine for a 14 kHz lowpass filter for signals sampled at 48 kHz. Input and output signals will be at 48 kHz since no rate change is performed by this function. Its frequency response is shown in figure 10.40 and its impulse response is shown in figure 10.41. The -3 dB point for this filter is located at approximately 14000 Hz. Code is in file `fir-LP.c`.

`LP20_48kHz_init` is the initialization routine for a 20 kHz lowpass filter for signals sampled at 48 kHz. Input and output signals will be at 48 kHz since no rate change is performed by this function. Its frequency response is shown in figure 10.42 and its impulse response is shown in figure 10.43. The -3 dB point for this filter is located at approximately 20000 Hz. Code is in file `fir-LP.c`.

Variables:

None.

Return value:

These functions return a pointer to a state variable structure of type `SCD_FIR`.

10.2.1.2 `hq_kernel`

Syntax:

```
#include "firflt.h"
long hq_kernel(long lseg, float *x_ptr, SCD_FIR *fir_ptr, float *y_ptr);
```

Prototype: firflt.h

Source code: fir-lib.c

Description:

This is the main entry routine for generic FIR filtering. It works as a switch to specific up- and down-sampling FIR-kernel functions. The adequate lower-lever filtering routine private to the filtering module (which is not visible by the user) is defined by the initialization routines. Currently, this function does not work properly for sample-by-sample downsampling operation, i.e. when $lseg = 1$. This limitation should be corrected in a future version.

Please note that prior to the first call to `hq_kernel`, one of the initialization routines `hq_*_init` must be called to allocate memory for state variables and the set the desired filter coefficients.

After returning from this function, the state variables are saved to allow segment-wise filtering through successive calls of `hq_kernel`. This is useful when large files have to be processed.

Variables:

lseg	.....	Number of input samples. Should be larger than 1 for proper downsampling operation.
x_ptr	.....	Array with input samples.
fir_ptr	.....	Pointer to FIR-struct.
y_ptr	.....	Pointer to output samples.

Return value:

The number of filtered samples as a `long`.

10.2.1.3 `hq_reset`

Syntax:

```
#include "firflt.h"
void hq_reset (SCD_FIR *fir_ptr);
```

Prototype: firflt.h

Source code: fir-lib.c

Description:

Clear state variables in `SCD_FIR` struct; deallocation of filter structure memory is not done. Please note that *fir_ptr* should point to a valid `SCD_FIR` structure, which was allocated by an earlier call to one of the FIR initialization routines `hq_*_init`.

Variables:

fir_ptr	.....	Pointer to a valid structure `SCD_FIR`.

Return value:

None.

10.2.1.4 `hq_free`

Syntax:

```
#include "firflt.h"
void hq_free (SCD_FIR *fir_ptr);
```

Prototype: firflt.h

Source code: fir-lib.c

Description:

Deallocate memory, which was allocated by an earlier call to one of the FIR initialization routines `hq_*_init`. Note that the pointer to the structure `SCD_FIR` must not be a null pointer.

Variables:

fir_ptr ..... Pointer to a structure of type `SCD_FIR`.

Return value:

None.

10.2.2 IIR Module

The IIR module contains filters whose main use is for asynchronous filtering. For telephony bandwidth asynchronous filtering, PCM filters are available in both cascade and parallel IIR filter forms. For wideband speech (50–7000 Hz), 3:1 and 1:3 rate-change factor filters are available. A transmit-side IRS filter for speech sampled at 8 kHz is also available in this module as an example of implementation of an IIR cascade-form filter.

The PCM filters have been designed for *sampling rates* of 8 and 16 kHz. It should be noted that the G.712 mask is specified in terms of Hz, rather than normalized frequencies. Therefore this applies only to rate conversions of factor 2, i.e., 8 kHz to 16 kHz and 16 kHz to 8 kHz. The frequency responses of the implemented PCM filters are shown in figure 10.48.

Since the digital filters need memory, state variables are needed. In the STL, a type `SCD_IIR` has been defined for parallel-form IIR filters, containing the past memory samples as well as filter coefficients and other control variables. Its fields are as follows:

nblocks ..... Number of coefficient sets
idown ..... Up-/down-sampling factor
k0 ..... Start index in next segment
gain ..... Gain factor
direct_cof ..... Direct path coefficient
b[3] ..... Pointer to numerator coefficients
c[2] ..... Pointer to denominator coefficients
T[2] ..... Pointer to state variables
hswitch ..... Switch to IIR-kernel: Up or down-sampling

For the cascade-form IIR filters, the state variable structure defined is `CASCADE_IIR` which is slightly different from the one for the parallel form structure:

nblocks ..... Number of stages in cascade
idown ..... Up-/down-sampling factor
k0 ..... Start index in next segment
gain ..... Gain Factor
a[2] .....Pointer to numerator coefficients
b[2] .....Pointer to denominator coefficients
T[4] ..... Pointer to state variables
hswitch ..... Switch to IIR-kernel: Up or down-sampling

It should be noted that the values of the fields must not be altered, and for most purposes they are not needed by the user. The relevant routines for each module are described in the next sections.

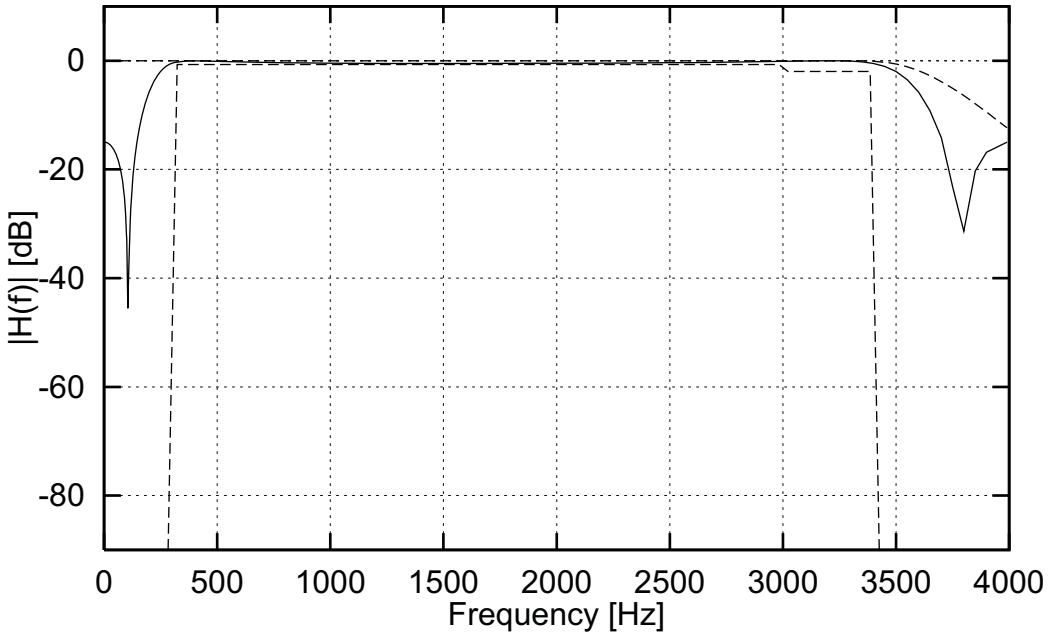


Figure 10.44: Frequency response of the cascade implementation of the G.712 standard PCM filter for data sampled at 8 kHz.

10.2.2.1 `iir_*_init`

Syntax:

```
#include "iirflt.h"
CASCADE_IIR *iir_G712_8khz_init (void);
CASCADE_IIR *iir_irs_8khz_init (void);
CASCADE_IIR *iir_casc_lp_3_to_1_init(void);
CASCADE_IIR *iir_casc_lp_1_to_3_init(void);
```

Prototypes: iirflt.h

Description:

`iir_G712_8khz_init` initializes an 8 kHz cascade IIR filter structure for a standard PCM

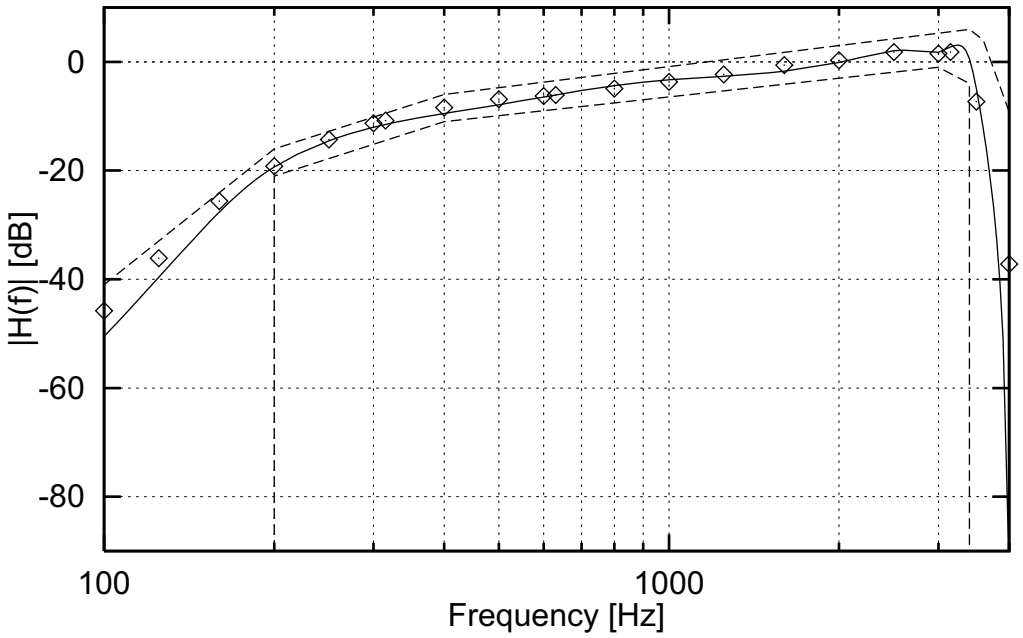


Figure 10.45: Frequency response of an IIR cascade implementation of the P.48 "full" transmit-side IRS weighting filter for data sampled at 8 kHz.

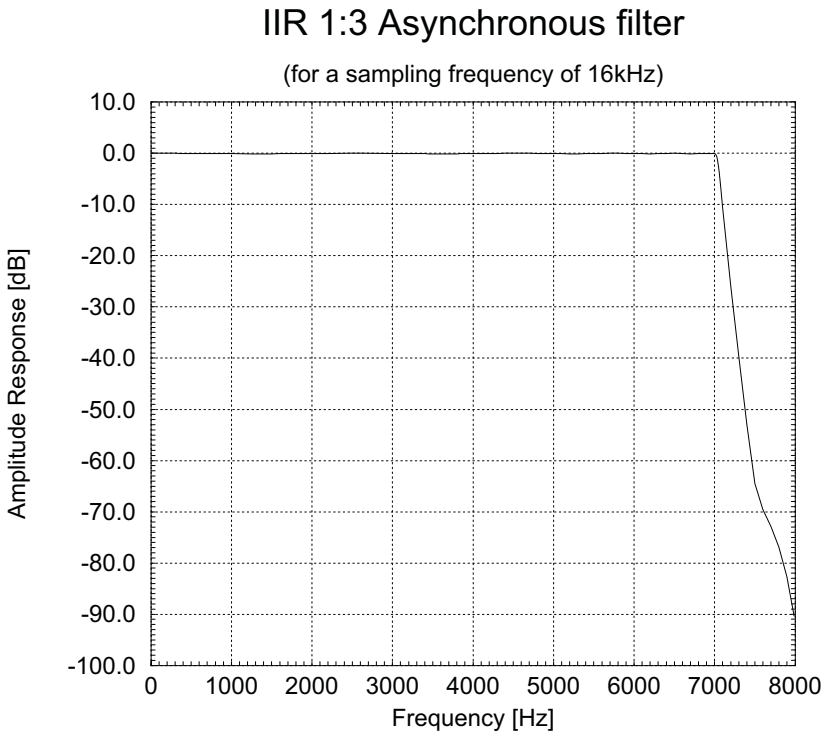
(G.712) filtering. Input and output signals will be at 8 kHz since no rate change is performed by this function. The -3 dB points for this filter are located at approximately 230 and 3530 Hz. Its source code is found in file `cascg712.c` and its frequency response is given in figure 10.44.

`iir_irs_8khz_init` initializes an 8 kHz cascade IIR filter structure for a transmit-side P.48 IRS non-linear phase filtering. Input and output signals will be at 8 kHz since no rate change is performed by this function. Its source code is found in file `iir-irs.c` and its frequency response is given in figure 10.45.

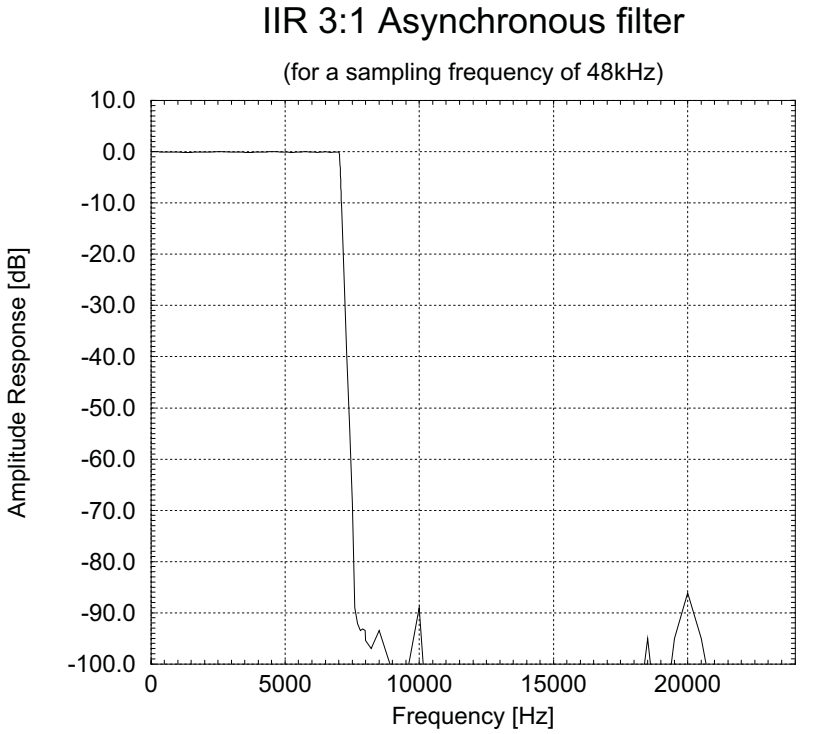
`iir_casc_lp_3_to_1_init` is the initialization routine for IIR low-pass filtering with a down-sampling factor of 3:1. Although this filter is relatively independent of the sampling rate,[4] it was originally designed for asynchronization filtering of 16 kHz sampled speech. The -3 dB point for this filter is located at approximately 7055 Hz. Its source code is found in file `iir-flat.c` and its frequency and impulse response are given in figures 10.46(a) and 10.47(a), respectively.

`iir_casc_lp_1_to_3_init` is the initialization routine for IIR low-pass filtering with a up-sampling factor of 1:3. Although this filter is relatively independent of the sampling rate, it was originally designed for asynchronization filtering of 16 kHz sampled speech. The -3 dB point for this filter is located at approximately 7055 Hz. Its source code is found in file `iir-flat.c` and its frequency and impulse response are given in figures 10.46(b) and 10.47(b), respectively.

[4]Since this is a low-pass filter, change of sampling rate implies in change of the lower and upper cutoff frequencies.

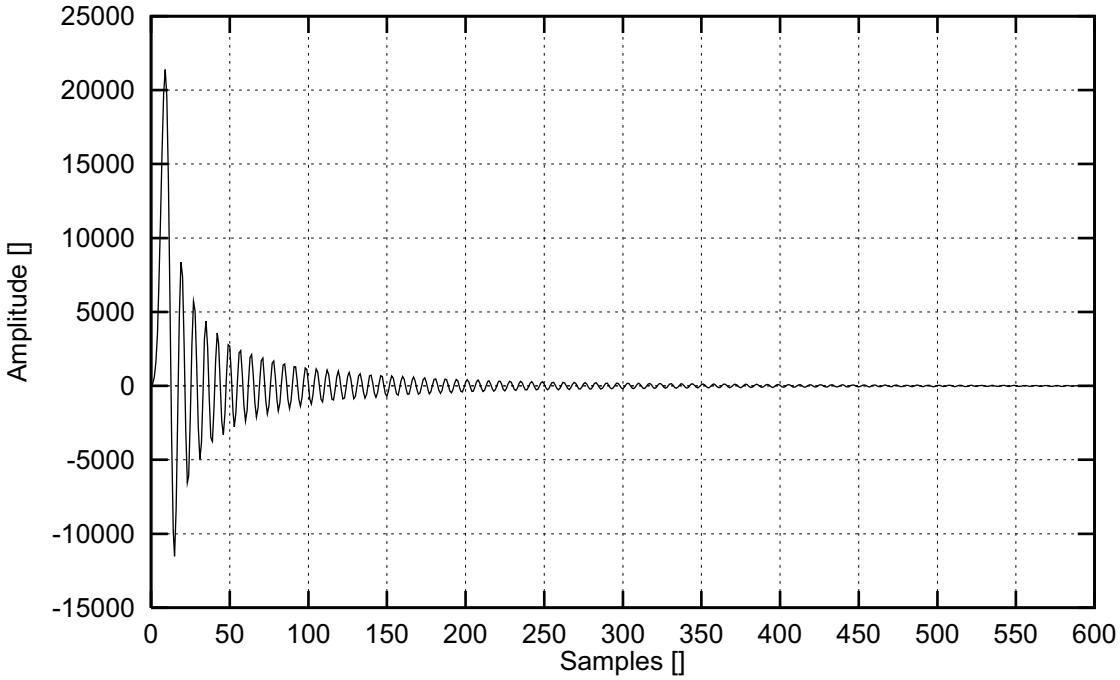


(a) Flat low-pass up-sampling by a factor of 3:1.

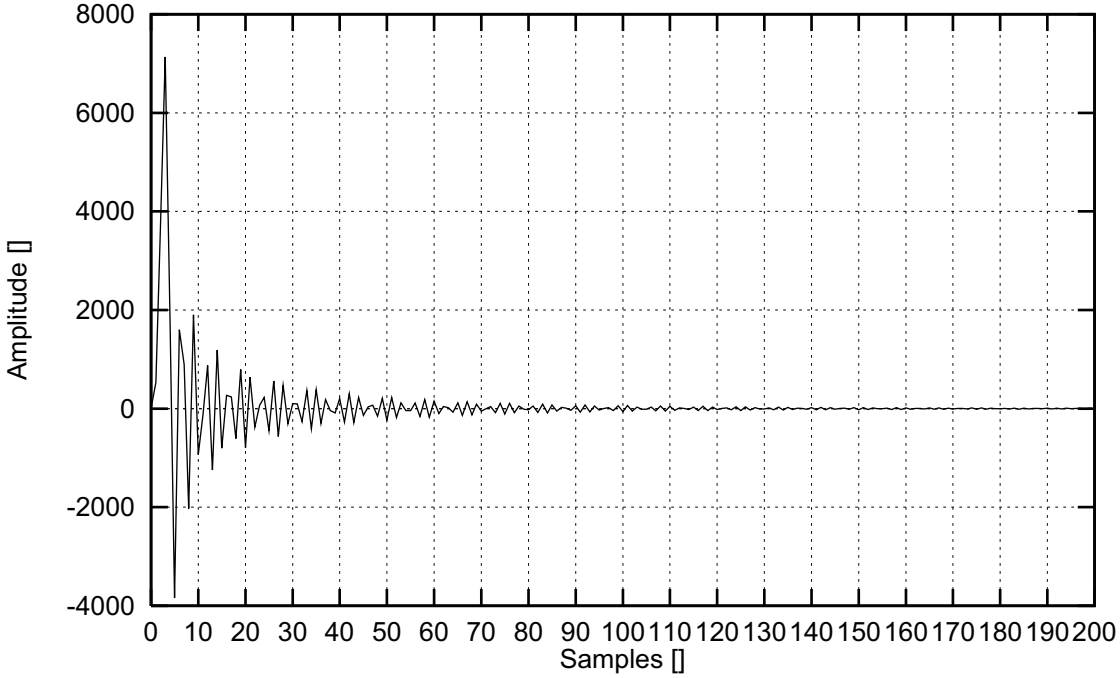


(b) Flat low-pass down-sampling by a factor of 1:3.

Figure 10.46: Flat low-pass IIR filter frequency response with factors 1:3 and 3:1 for sampling rates of 16000 and 48000 Hz.

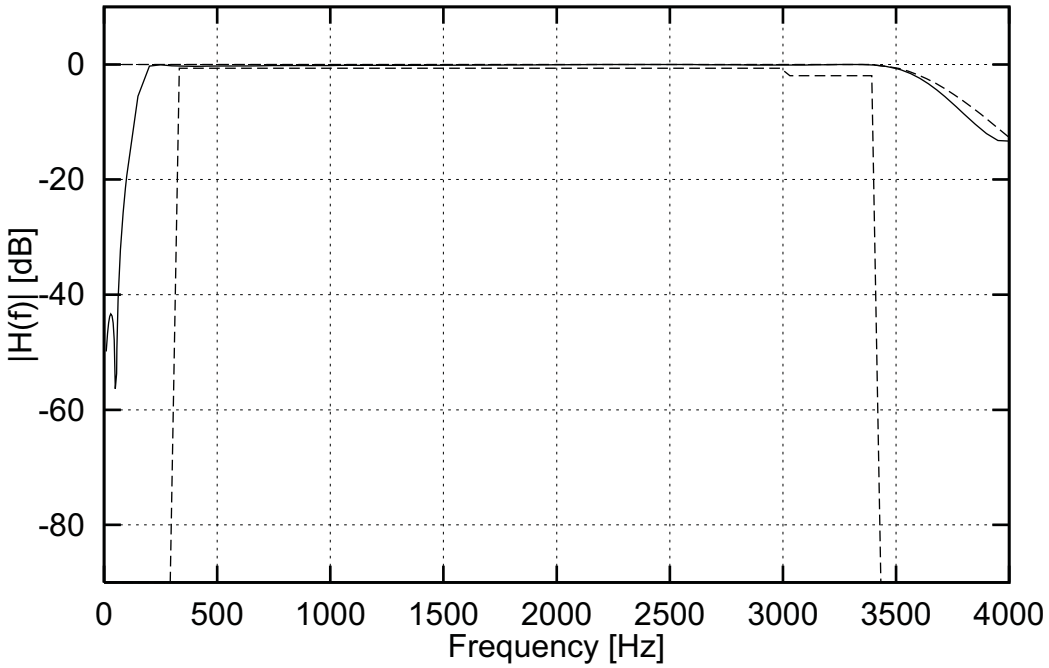


(a) 1:3 up-sampling factor

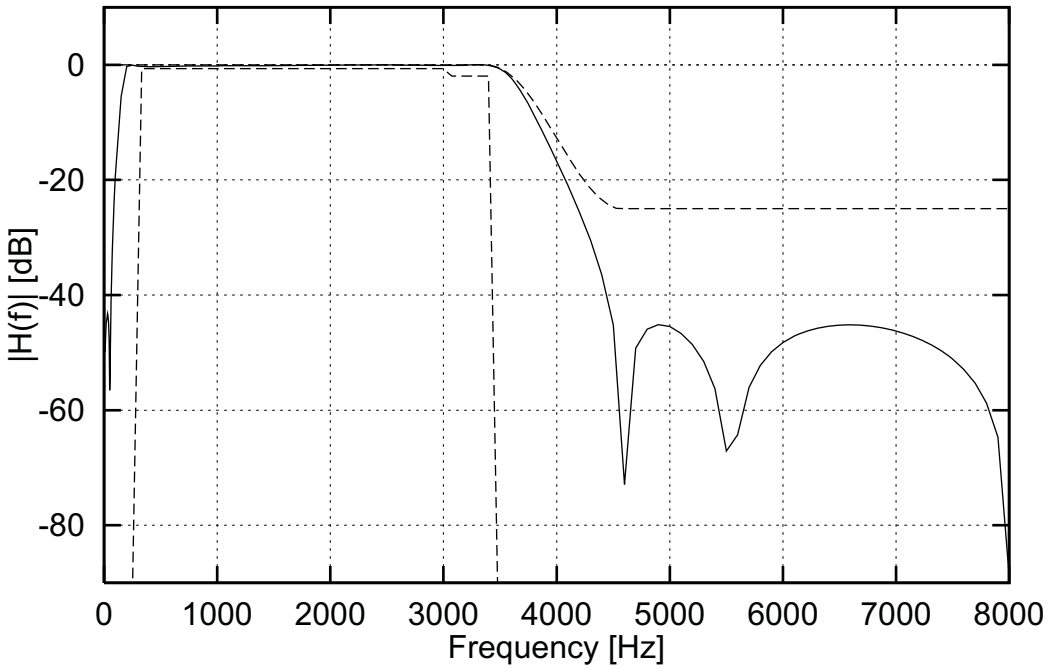


(b) 3:1 down-sampling factor

Figure 10.47: Impulse response for 1:3 and 3:1 cascade-form low-pass IIR filter.



(a) G.712 for input samples at 8 kHz, up-sampling factor 1:2



(b) G.712 for input samples at 16 kHz, down-sampling factor 2:1 or 1:1

Figure 10.48: Standard PCM (G.712) quality filter response.

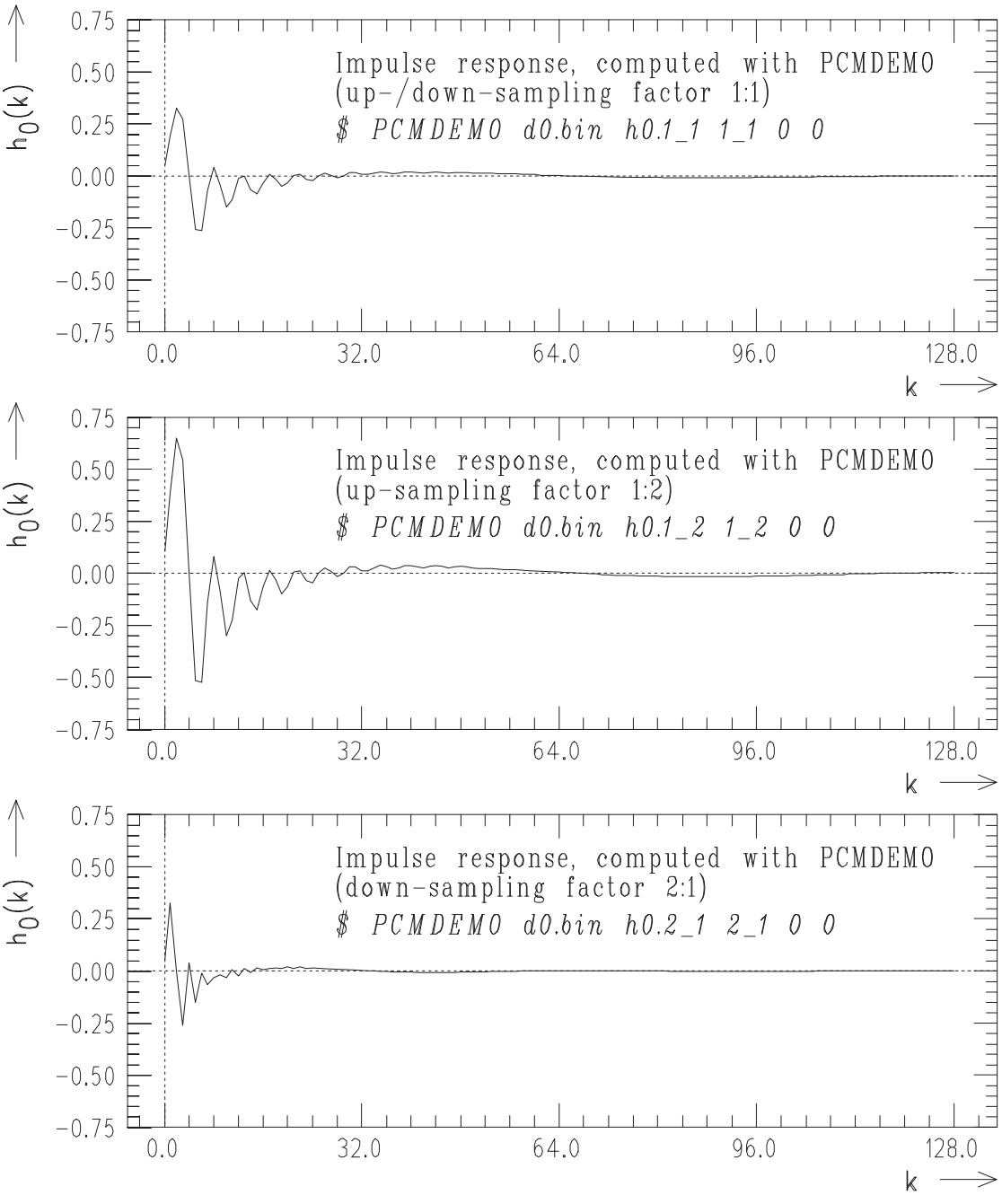


Figure 10.49: Impulse response for G.712 filters (Top: factor 1:1; Middle: factor 1:2; Bottom: factor 2:1).

10.2.2.2 cascade_iir_kernel

Syntax:

```
#include "iirflt.h"
long cascade_iir_kernel (long lseg, float *x_ptr, CASCADE_IIR *iir_ptr,
                         float *y_ptr);
```

Prototype: iirflt.h

Source code: iir-lib.c

Description:

General function for implementing filtering using a cascade-form IIR filter previously initialized by one of the `iir_*_init()` routines.

Variables:

lseg	.....	Number of input samples.
x_ptr	.....	Array with input samples.
iir_ptr	.....	Pointer to a cascade-form IIR-struct `CASCADE_IIR`.
y_ptr	.....	Pointer to output samples.

Return value:

The number of output samples is returned as a long.

10.2.2.3 cascade_iir_reset

Syntax:

```
#include "iirflt.h"
void cascade_iir_reset (CASCADE_IIR *iir_ptr);
```

Prototype: iirflt.h

Source code: iir-lib.c

Description:

Clear state variables in `CASCADE_IIR` structure, which have been initialized by a previous call to one of the initialization functions. Memory previously allocated is not released.

Variables:

iir_ptr	.....	Pointer to struct `CASCADE_IIR`, previously initialized by a call to one of the initialization routines.

Return value:

None.

10.2.2.4 `cascade_iir_free`

Syntax:

```
#include "iirflt.h"
void cascade_iir_free (SCD_IIR *iir_ptr);
```

Prototype: iirflt.h

Source code: iir-lib.c

Description:

Deallocate memory, which was allocated by an earlier call to one of the cascade-form IIR filter initialization routines described before. `iir_prt` must not be a NULL pointer.

Variables:

iir_ptr ..... Pointer to struct `CASCADE_IIR`, previously initialized by a call to one of the initialization routines.

Return value:

None.

10.2.2.5 `stdpcm_*_init`

Syntax:

```
#include "iirflt.h"
SCD_IIR *stdpcm_16khz_init (void);
SCD_IIR *stdpcm_1_to_2_init (void);
SCD_IIR *stdpcm_2_to_1_init (void);
```

Prototypes: iirflt.h

Description:

`stdpcm_16khz_init` initializes a 16 kHz IIR filter structure for standard PCM (G712) filtering. Input and output signals will be at 16 kHz since no rate change is performed by this function. The -3 dB points for this filter are located at approximately 174 and 3630 Hz. Source code is found in file `iir-g712.c` and its frequency and impulse response are given in figures 10.48(b) and 10.49 (top), respectively.

`stdpcm_1_to_2_init` initializes standard PCM filter coefficients for filtering by the generic filtering routine `stdpcm_kernel`, for input signals at 8 kHz, generating the output at 16 kHz. The -3 dB points for this filter are located at approximately 174 and 3630 Hz. Source code is found in file `iir-g712.c` and its frequency and impulse response are given in figures 10.48(a) and 10.49 (middle), respectively.

`stdpcm_2_to_1_init` initializes standard PCM filter coefficients for filtering by the generic filtering routine `stdpcm_kernel` for input signals at 16 kHz, generating the output at 8 kHz. The -3 dB points for this filter are located at approximately 174 and 3630 Hz. Source code is found in file `iir-g712.c` and its frequency and impulse response are given in figures 10.48(b) and 10.49 (bottom), respectively.

Variables:

None.

Return value:

This function returns a pointer to a state variable structure of type `SCD_IIR`.

10.2.2.6 `stdpcm_kernel`

Syntax:

```
#include "iirflt.h"
long stdpcm_kernel (long lseg, float *x_ptr, SCD_IIR *iir_ptr,
                    float *y_ptr);
```

Prototype: iirflt.h

Source code: iir-lib.c

Description:

General function to perform filtering using a parallel-form IIR filter previously initialized by one of the appropriate parallel-form `*_init()` routines available.

Variables:

lseg ..... Number of input samples.

x_ptr ..... Array with input samples.

iir_ptr ..... Pointer to a parallel-form IIR-struct `SCD_IIR`.

y_ptr ..... Pointer to output samples.

Return value:

This function returns the number of output samples as a `long`.

10.2.2.7 `stdpcm_reset`

Syntax:

```
#include "iirflt.h"
void stdpcm_reset (SCD_IIR *iir_ptr);
```

Prototype: iirflt.h

Source code: iir-lib.c

Description:

Clear state variables in `SCD_IIR` structure, which have been initialized by a previous call to one of the init functions. Memory previously allocated is not released.

Variables:

iir_ptr ..... Pointer to struct `SCD_IIR`, previously initialized by a call to one of the initialization routines.

Return value:

None.

10.2.2.8 `stdpcm_free`

Syntax:

```
#include "iirflt.h"
void stdpcm_free (SCD_IIR *iir_ptr);
```

Prototype: iirflt.h

Source code: iir-lib.c

Description:

Release memory which was allocated by an earlier call to one of the parallel-form IIR filter initialization routines described before. The parameter `iir_prt` must not be a null pointer.

Variables:

iir_ptr ..... Pointer to struct `SCD_IIR`, previously initialized by a call to one of the initialization routines.

Return value:

None.

10.3 Tests and portability

Compliance with the R&Os was verified by checking the frequency response of the filters and the size of the output files. Frequency response was obtained by feeding the filtering routines with sinewaves and calculating the ratio in dB, for each frequency of interest.

Portability of this module was checked by running the same speech file on a proven platform and on a test platform. Comparison of both processed files should show either no differences or yield equivalent results.[5] Tests were performed in the VAX/VMS environment with VAX-C and gcc, in MSDOS with Borland Turbo C++ Version 1.00 and gcc (DJGPP), in SunOS with cc, acc, and gcc, and in HPUX with gcc.

10.4 Examples

10.4.1 Description of the demonstration programs

Three programs are provided as demonstration programs for the RATE module, firdemo.c, iirdemo.c, and filter.c.

Programs `firdemo.c` and `iirdemo.c` were the first demonstration programs for the rate change module. The former is found in directory `fir` of the STL and contains a cascade

[5] Some differences may appear in the output files, but for a few samples and by no more than 1 LSB. As an example, in the tests for checking VAX and SUN-OS, one of the files differed in 3 samples out of 49152 for a cascade of high-quality up- and down-sampling of 1:6 and 6:1. For small rate change factors, differences are unlikely.

processing of the FIR filters available upto the STL96. The latter is found in directory `iir` of the STL and contains a cascade processing of the IIR filters available up to the STL96. However, because of the increasing static memory requirement for cascade processing that came with the introduction of new filters in the STL, these two programs became prohibitive and their maintenance was discontinued. They are still functional, although outdated.

Program `filter.c` is a single demonstration program that incorporates both IIR and FIR filters in the STL and has been kept up-to-date as new filters are added to the STL. Compared to the firdemo.c and iirdemo.c programs, filter.c can only perform one filtering operation per pass, while firdemo.c and iirdemo.c could perform a number of 1:1 operations combined with two up-sampling and two downsampling operations. Hence, several calls of the filter program are necessary to implement what was accomplished by a single call of firdemo.c and iirdemo.c, in addition to the cumulative quantization noise (from the successive float-to-short conversions). In applications where multiple filtering is needed and the user is concerned with the quantization noise accumulation, a custom-made program could be used e.g. based on a specialization of either firdemo.c, iirdemo.c, or filter.c.

10.4.2 Example: Calculating frequency responses

The following C code exemplifies the use of some of the filter functions available in the STL. The C code generates a number of tones which are specified by the user (lower, upper, and step frequencies). The frequency response is obtained by calculating the power change for each single frequency before and after filtered by the selected filter.

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

/* UGST MODULES */
#include "ugstdemo.h"
#include "iirflt.h"
#include "firflt.h"

/* Other stuff */
#define TWO_PI (8*atan(1.0))
#define QUIT(m,code) {fprintf(stderr,m); exit((int)code);}

void main(argc, argv)
  int             argc;
  char           *argv[];
{
  SCD_FIR        *fir_state;
  SCD_IIR        *iir_state;
  float          *BufInp, *BufOut;
  char            F_type[20];
  long            j, N, N2;
  long            inp_size, out_size;
  double          f, f0, fstep, ff, fs, inp_pwr, H_k;
```

```
char            is_fir;

/* Preamble */
N = 256; N2 = 20; inp_size = N * N2;

/* Read parameters for processing */
GET_PAR_S(1, "_Filter type(irs,hq2,hq3,pcm,pcm1): ... ", F_type);
GET_PAR_D(2, "_Start frequency [Hz]: ..... ", f0);
GET_PAR_D(3, "_Stop frequency [Hz]: ..... ", ff);
GET_PAR_D(4, "_Frequency step [Hz]: ..... ", fstep);
FIND_PAR_D(5, "_Sampling Frequency [Hz]: ..... ", fs, 8000);

/* Check consistency of upper and lower frequencies */
ff = (ff >= fs / 2)? (fs / 2) : ff;
if (f0 < 2.0 / (double) inp_size * fs && f0 != 0.0)
  f0 = 2.0 / (double) inp_size *fs;

/* Normalization of frequencies */
f0 /= fs; ff /= fs; fstep /= fs;

/* Set flag to filter type: IIR or FIR */
is_fir = (strncmp(F_type,"pcm",3)==0 || strncmp(F_type,"PCM",3)==0)
         ? 0 : 1;

/* ... CHOOSE CORRECT FILTER INITIALIZATION ... */
/*
 * Filter type: irs - IRS weighting 2:1 or 1:2 factor:
 *                     . fs ==  8000 -> up-sample: 1:2
 *                     . fs == 16000 -> down-sample: 2:1
*/
if (strncmp(F_type, "irs", 3) == 0 || strncmp(F_type, "IRS", 3) == 0)
{
  if (fs == 8000)
    fir_state = irs_8khz_init();
  else if (fs == 16000)
    fir_state = irs_16khz_init();
  else
    QUIT("IRS Implemented only for 8 and 16 kHz\n", 15);
}
/*
 * Filter type: hq2 - High-quality 2:1 or 1:2 factor:
 *                    . fs ==  8000 -> up-sample: 1:2
 *                    . fs == 16000 -> down-sample: 2:1
 *              hq3 - High-quality 3:1 or 3:1 factor
 *                    . fs ==  8000 -> up-sample: 1:3
 *                    . fs == 16000 -> down-sample: 3:1
*/
else if (strncmp(F_type,"hq",2)==0 || strncmp(F_type,"HQ",2)==0)
{
  if (fs == 8000)             /* It is up-sampling! */
    fir_state = F_type[2] == '2'
      ? fir_up_1_to_2_init()
```

```
      : fir_up_1_to_3_init();
  else                          /* It is down-sampling! */
    fir_state = F_type[2] == '2'
      ? fir_down_2_to_1_init()
      : fir_down_3_to_1_init();
}
/*
 * Filter type: pcm  - Standard PCM quality 2:1 or 1:2 factor:
 *                   . fs ==  8000 -> up-sample: 1:2
 *                   . fs == 16000 -> down-sample: 2:1
 *              pcm1 - Standard PCM quality with 1:1 factor
 *                   . fs ==  8000 -> unimplemented
 *                   . fs == 16000 -> OK, 1:1 at 16 kHz
*/
else if (strncmp(F_type,"pcm",3)==0 || strncmp(F_type,"PCM",3)==0)
{
  if (strncmp(F_type,"pcm1", 4)==0 || strncmp(F_type,"PCM1",4)==0)
  {
    if (fs == 16000)
      iir_state = stdpcm_16khz_init();
    else
      QUIT("Unimplemented: PCM w/ factor 1:1 for given fs\n", 10);
  }
  else
    iir_state = (fs == 8000)
      ? stdpcm_1_to_2_init()  /* It is up-sampling! */
      : stdpcm_2_to_1_init(); /* It is down-sampling! */
}

/* Calculate Output buffer size */
if (is_fir)
  out_size = (fir_state->hswitch=='U')
             ? inp_size * fir_state->dwn_up
             : inp_size / fir_state->dwn_up;
else
  out_size = (iir_state->hswitch=='U')
             ? inp_size * iir_state->idown
             : inp_size / iir_state->idown;

/* Allocate memory for input buffer */
if ((BufInp = (float *) calloc(inp_size, sizeof(float))) == NULL)
  QUIT("Can't allocate memory for data buffer\n", 10);

/* Allocate memory for output buffer */
if ((BufOut = (float *) calloc(out_size, sizeof(float))) == NULL)
  QUIT("Can't allocate memory for data buffer\n", 10);

/* Filtering operation */
for (f = f0; f <= ff; f += fstep)
{
  /* Reset memory */
  memset(BufOut, '\0', out_size * sizeof(float));
```

```
    /* Adjust top (NORMALIZED!) frequency, if needed */
    if (fabs(f - 0.5) < 1e-8/fs) f -= (0.05*fstep);

    /* Calculate as a temporary the frequency in radians */
    inp_pwr = f * TWO_PI;

    /* Generate sine samples with peak 20000 ... */
    for (j = 0; j < inp_size; j++)
      BufInp[j] = 20000.0 * sin(inp_pwr * j);

    /* Calculate power of input signal */
    for (inp_pwr = 0, j = 0; j < inp_size; j++)
      inp_pwr += BufInp[j] * BufInp[j];

    /* Convert to dB */
    inp_pwr = 10.0 * log10(inp_pwr / (double) inp_size);

    /* Filtering the whole buffer ... */
    j = (is_fir)
        ? fir_kernel(inp_size, BufInp, fir_state, BufOut)
        : stdpcm_kernel(inp_size, BufInp, iir_state, BufOut);

    /* Compute power of output signal; discard initial 2*N samples */
    for (H_k = 0, j = 2 * N; j < out_size - 2 * N; j++)
      H_k += BufOut[j] * BufOut[j];

    /* Convert to dB */
    H_k = 10 * log10(H_k / (double) (out_size - 4 * N)) - inp_pwr;

    /* Printout of gain at the current frequency */
    printf("\nH( %4.0f ) \t = %7.3f dB\n", f * fs, H_k);
  }
}
```