Design and Implementation of an Interpolation Filter for Hearing-Aid Application

Pere Llimós Muntal

Supervisors:
Erik Bruun [DTU]
Peter Pracný[DTU]

Master Project, June 2012

Technical University of Denmark
DTU Elektro
Department of Electrical Engineering
Integrated Circuits Group

Universitat Politècnica de Catalunya (UPC)
Escola Tècnica Superior Enginyeria Industrial de Barcelona (ETSEIB)
Departament d'Electrònica

Preface

This master thesis is the result of my five year education in engineering. Even though these studies have embraced a wide variety of fields, this final project will be focused in electronics. I would like to thank my both supervisors from Denmark Technical University, Dr. Erik Bruun and the PhD candidate Peter Pracný for the huge amount of time spent in educating me and giving me feedback. I strongly believe that the learning curve during this project has been exponential which really kept me interested and willing to learn more. This project could not have been possible without both of you, so many thanks.

Pere Llimós Muntal, June 2012

Permission for use of content

Pere Llimós Muntal, June 2012

Design and Implementation of an Interpolation Filter for Hearing-Aid Application

by

Pere Llimós Muntal

Master thesis submitted to achieve the academical degree of:
Enginyeria Industrial Superior

Academical year 2011-2012

Supervisors: Dr. Erik Bruun, PhD candidate Peter Pracný

Escola Tècnica Superior d’Enginyeria Industrial de Barcelona (ETSEIB)
Universitat Politècnica de Catalunya (UPC)

Summary

This master thesis will deal with the design and implementation of an interpolation filter which is part of the back end stage of the D/A converter of a hearing aid. Since this filter will be used in hearing-aid applications the design goal will be to achieve low current consumption and hardware requirements, and reduced area needed for the implementation.

Key words

Hearing aids, D/A converter, interpolation filter, sigma-delta modulator

Design and Implementation of an Interpolation Filter for Hearing-Aid Application

Pere Llimós Muntal

Supervisors: Dr. Erik Bruun and PhD candidate Peter Pracný

Abstract – **In this master thesis the design and implementation of an interpolation filter for hearing-aid applications will be discussed. The aim of the design will be to minimize the current consumption, hardware demand and area needed for the implementation of the design.**

Keywords – **Hearing aids, interpolation filter, sigma-delta modulator, D/A converter**

Hearing aids are devices with very strict specifications. They are attached in its owner's ear, therefore to make them comfortable they must be small and light. The biggest and heaviest part of a hearing aid is the battery. This means that if a significant reduction of the size and weight of the device wants to be achieved, the battery's dimensions and in consequence, its capacity, must be reduced. This solution conflicts with the fact that since the user will probably be wearing it all day, the working time per charge should be maximized. The design of the hearing aid will aim at the reduction of hardware demands that will lead to a reduction of current consumption that will allow the battery to be smaller.

The D/A converter, which is the back-end stage of the audio processing chain of a hearing aid can be seen in Figure 1. It consists of an interpolation filter, a sigma-delta modulator, a digital pulse width modulation, a class-D output stage, a feedback chain and an output filter. The sigma-delta modulator is an oversampled data converter, so a previous oversampling is needed for its correct operation. This oversampling will be performed by the interpolation filter.

This oversampling process will increase the sampling frequency by the oversampling ratio needed for the sigma delta modulator, which will lead to an increase of the band of interest. Since the input of the interpolation filter is discrete, its frequency spectrum will be repeated at every multiple of fs. Those frequency spectrum repetitions or images, will appear in the output band of interest of the filter, therefore they will need to be suppressed by the interpolation filter.

The interpolation filter of this design will be separated into four stages as it can be seen in Figure 2. The first stages of the filter will be the sharpest ones, and in consequence the most hardware demanding ones since the will have to attenuate the closest images. The last stages of the filter will attenuate the furthest images, so filters with less hardware requirements will be suitable. This multistage approach will also allows the first stages of the filter to work at lower frequency, since the sampling frequency will be increased step by step.

This paper will deal with the design of the first stage of the interpolation filter since it is the most hardware demanding one. An optimized design will be critical regarding the overall hardware savings of the interpolation filter.

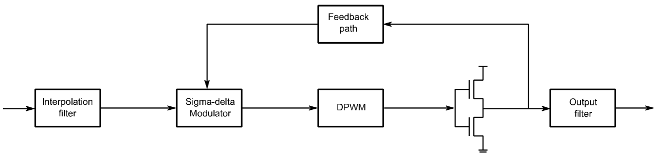


Figure 1: Back-end stage structure.

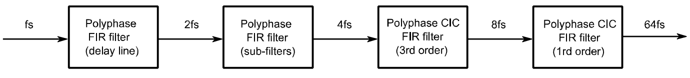

fs
Polyphase FIR filter (delay line)
2fs
Polyphase FIR filter (sub-filters)
4fs
Polyphase CIC FIR filter (3rd order)
8fs
Polyphase CIC FIR filter (1rd order)
64fs

Figure 2: Four stages filter structure.

Contents

List of acronyms

A/D	analog to digital
ASIC	application-specific integrated circuit
CIC	cascaded integrator-comb
CSD	canonic signed digits
DCM	digital clock manager
DFT	discrete Fourier transformation
DPWM	digital pulse width modulation
DSP	digital signal processing
D/A	digital to analog
FFT	fast fourier transformation
FIR	finite impulse response
FPGA	field programmable gate array
FSM	finite state machine
IIR	infinite impulse response
MAC	multiplier and accumulator
NTF	noise transfer function
RAM	random access memory
ROM	read only memory
SDNR	signal to distortion noise ratio
SNR	signal to noise ratio
SQNR	signal to quantization noise ratio
STF	signal transfer function
VHDL	VHSIC hardware description language
VHSIC	very high speed integrated circuit

Chapter 1

1. Introduction

Nowadays hearing aids are devices with very strict specifications. They are attached to its wearer's ear, so they must be small, light and comfortable. Also, one battery charge should have decent time duration since it will be used during most of the day. Hearing aids normally function with a 1,4 V Battery, and the analog parts are regulated with a voltage of 1 V while the digital parts are regulated at least 100 mV below them. The number of working hours per battery charge is set to 140 hours, so the current consumption of the hearing aid should be lowered as much as possible. This current consumption will be lowered by designing a filter with the aim of saving hardware. In addition, since the battery is the heaviest part of the device, it should be kept as small as possible to save area, which will make the hearing aid smaller, and to save weight which will make it lighter.

Hearing aids structure consists in an A/D converter stage followed by a signal processing block and a D/A back end stage Figure 1.1. The back-end stage of the hearing aid that will be treated in this paper consists of an interpolation filter, a sigma-delta modulator, a digital pulse width modulation (DPWM), a class-D output-stage, an output filter and a feedback path Figure 1.2. The DPWM block will generate the pulse width modulation signal that will drive the class-D output stage, while the sigma-delta modulator will shape the noise improving this way the signal to noise ratio (SNR), which is the power of the signal divided by the power of the noise, and will prepare the signal for the DPWM. The usage of a sigma-delta modulator in audio applications is adequate since the bandwidth of interest of the audio signals is only a fraction of the bandwidth achievable with integrated digital technologies of today, allowing taking advantage of the effects of over-sampling and noise shaping.

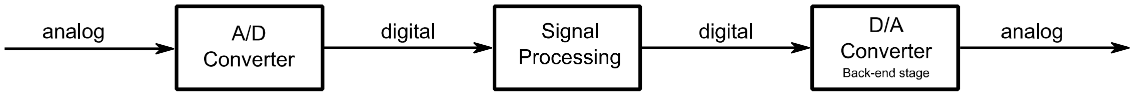


Figure 1.1: Hearing aid structure.

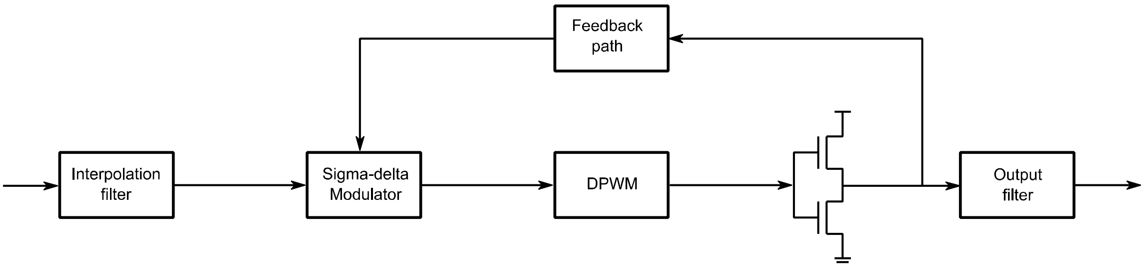


Figure 1.2: Back-end stage structure.

The sigma-delta modulator that is used in this back-end stage will be a 3rd order 3 bits sigma-delta modulator with an oversampling ratio OSR=64 with optimized zeros. That means that, ideally, this sigma delta modulator will need an input frequency 64 times faster than the original sample frequency [3], so an interpolation filter will be needed to achieve that goal. Since the input signal will be discrete with a sample frequency fs, its frequency spectrum is repeated at every multiple of fs which means that when the sampling frequency is augmented, 32 repetitions or images of this spectrum will appear in the new band of interest, Figure 1.3. For this reason the interpolation filter will have to multiply the input frequency by 64 and also attenuate the 32 repetitions of the frequency spectrum of the input signal.

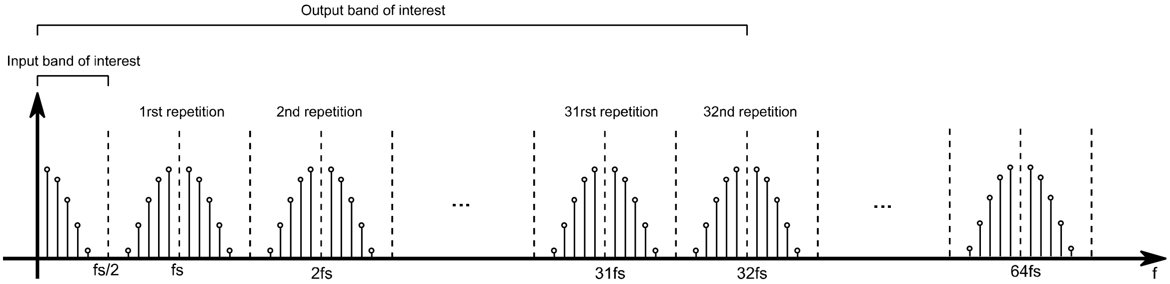


Figure 1.3: Repetitions that appear in the output band of interest.

The interpolation filter will be divided into four steps, which is a good approach for hardware saving purposes [1]. The idea of the multi-step approach is to suppress the closest frequency spectrum repetitions or images with the first steps of the filter while using the latest steps of the filter to suppress the further ones. This means that the first steps of the filter will have to be sharper and in consequence more hardware demanding, while the last steps of the filter will be smooth with very low hardware demand [1]. Also, the frequency will be increased step by step which means that the first filters, the most hardware demanding, will work at lower frequency improving the current consumption [1].

The filter will consist in a first polyphase FIR filter with a delay line approach that doubles the frequency, a second polyphase FIR filter with a sub-filter approach that doubles again the frequency, followed by a 3rd and 1st order cascaded integrator-comb filters (CIC filters) that multiply by two and eight the frequency respectively Figure 1.4. The attenuation of the input signal frequency spectrum repetitions, or images, distributed in the four filters of the four step interpolation filter can be seen in Figure 1.5. In every step a certain images are suppressed, and are shown in the next step with discontinuous lines. After the four steps, all the 32 images will be suppressed. Obviously, the sharpest filter of all four will be the first one since it is the one that has to suppress the closest image [1]. This means that a higher order will be needed to satisfy the specifications, which leads to higher hardware requirements, higher current consumption and higher area required. It can be said that a high percentage of the whole interpolation filter´s hardware requirements is from this first filter. Therefore, an optimized design of this first filter will be very important to meet the low hardware requirements specifications. This paper will deal with the design of the first step of the interpolation filter which is, for the reasons exposed before, the most critical one.

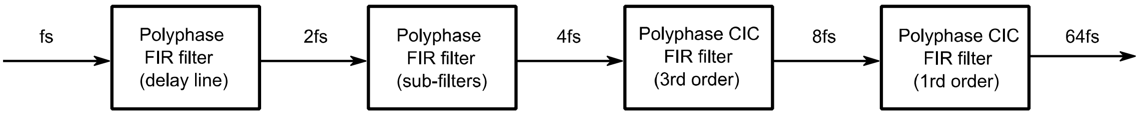
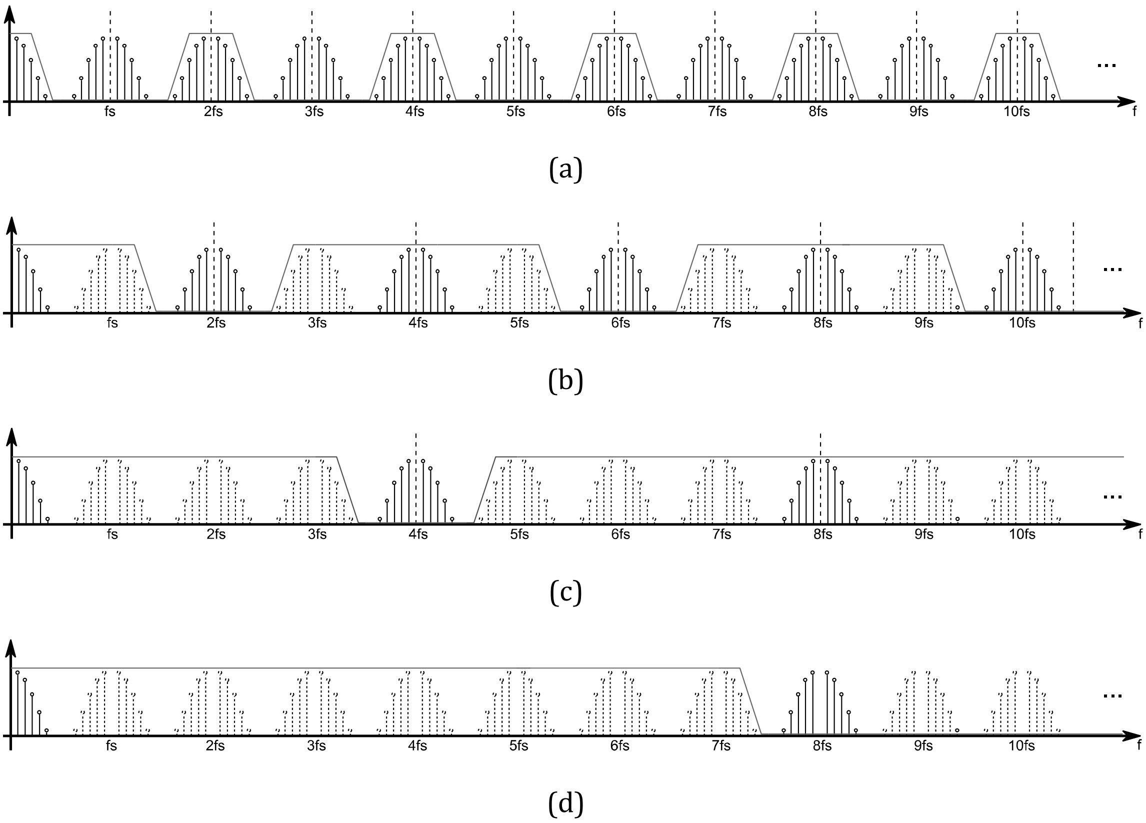

fs
Polyphase FIR filter (delay line)
2fs
Polyphase FIR filter (sub-filters)
4fs
Polyphase CIC FIR filter (3rd order)
8fs
Polyphase CIC FIR filter (1rd order)
64fs

Figure 1.4: Four steps filter structure.



(a)

(b)

(c)

(d)

Figure 1.5: Images attenuation of the whole interpolation filter divided in the four steps: (a) First step; (b) Second step; (c) Third step; (d) Fourth step.

The most important specification for the interpolation filter is the attenuation needed for the images. This parameter will be very important because the more attenuation needed the higher hardware requirements will the filter have. The filter that was previously situated in the first stage of the interpolation filter used the sigma-delta modulator noise to transfer function to decide the attenuation. The noise transfer function, NTF, of the sigma-delta modulator is shown in Figure 1.6. The images were suppressed under the NTF curve. More attenuation would be pointless because the amount of noise defined by the NTF curve would be added anyway by the sigma-delta modulator. As it can be seen in the NTF curve, the attenuation needed for the images would be -80dB. However, an attenuation of -60dB will be enough to suppress the images below the hearing threshold of a normal hearing person for a speech signal of normal intensity, Figure 1.7(a). As it can be seen in Figure 1.7(b) even less suppression could be used because the hearing aids are aimed to impaired-hearing persons, but since the hearing aids are tested with normal hearing persons, the -60dB suppression will be used. The specifications are less restrictive than before, which means that the filter will need to be less sharp and in consequence less hardware demanding. For this reason this last approach will be used in the new design of the filter to decide the attenuation of the images, which will now be -60dB.

In this project the filter will be designed and implemented in an FPGA because of its flexibility. During the design process lots of mistakes can be done, so an adaptable device that can be reprogrammed in case that the filter designed does not work is very useful. The downside of the FPGA is that, even if being more versatile, the implementation is not optimal. Not specific components will be used, but generic ones that give the FPGA its flexibility. Therefore, the area used for the microchip will also be bigger than it could be with specific components. For that reason, after the design of the filter in FPGA, the future work will be to export the design to ASIC. The integrated circuit in ASIC will be customized for the particular use of the filter therefore specific components for the exact tasks will be used, and the optimal chip design regarding hardware saving and area reduction will be achieved. The ASIC implementation will not be dealt with in this paper.

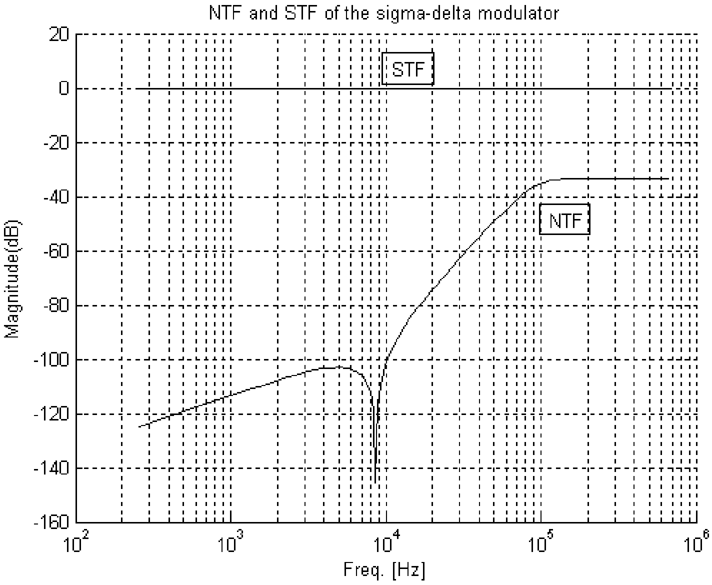
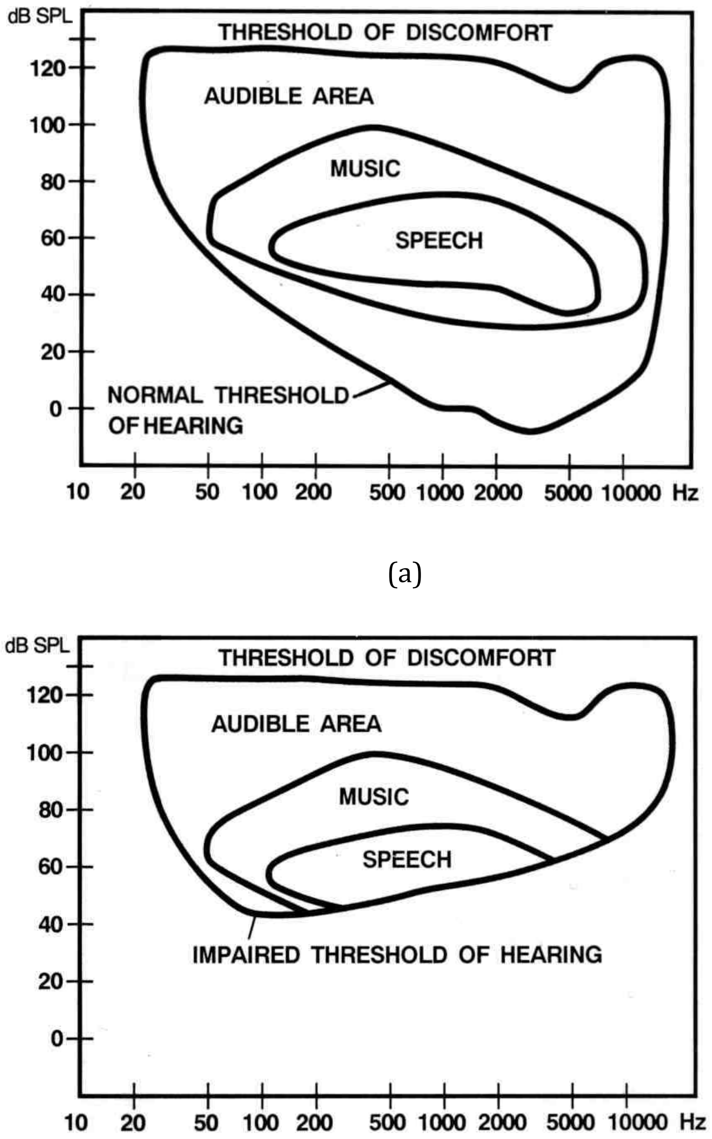


Figure 1.6: NTF and STF of the sigma-delta modulator.



(a)

(b)

Figure 1.7: Hearing threshold of: (a) normal hearing person; (b) hearing-impaired person. Pictures were originally published in [8]. Used with permission from Widex A/S.

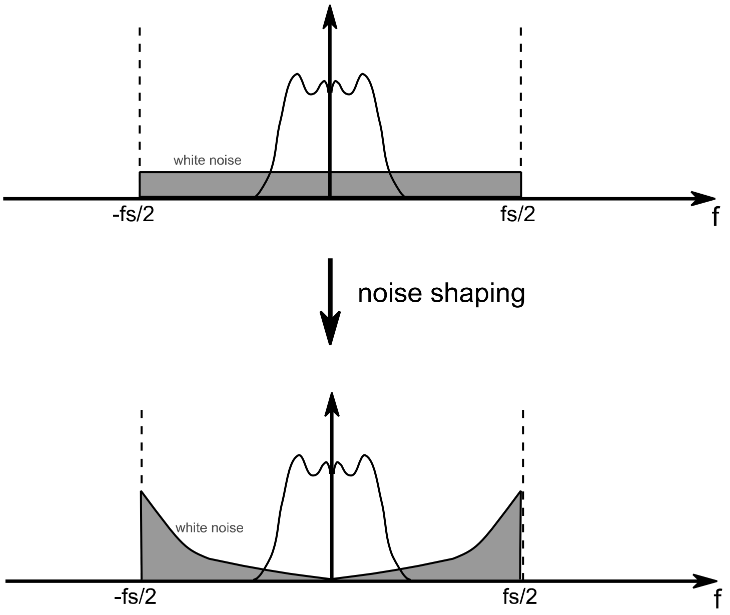
Chapter 2

2. Theoretical overview (2)

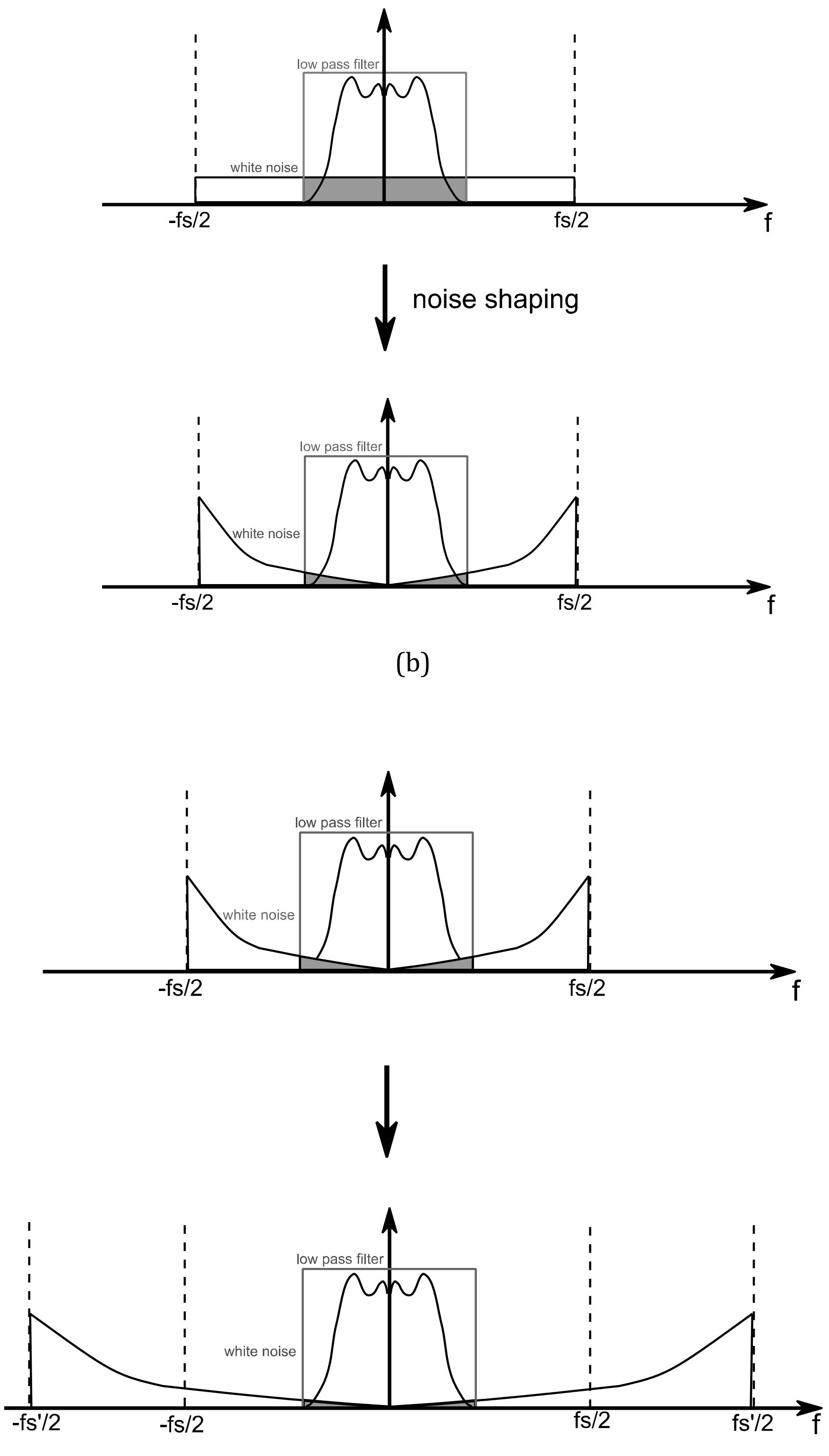
In this chapter, the theoretical background needed to understand the following parts of this master thesis will be exposed. The filter that will be designed is the first stage of the interpolation filter but the other stages and the subsequent sigma-delta modulator will also be described in order to contextualize.

2.1 Sigma-delta modulator

As it was stated, the interpolation filter is used as an oversampling method for the 3rd order 3 bits sigma-delta modulator with an oversampling ratio OSR=64 with optimized zeros. The sigma-delta modulator uses noise shaping in order to reduce the overall noise in the system [3]. In Figure 2.1(a) the effect of the noise shaping can be seen. It basically consists in pushing the noise into each extreme of the band of interest. The aim of this noise shaping is that if a low pass filter is used afterwards, a significant higher amount of noise is filtered away Figure 2.1(b) [3]. It can also be seen that if the band of interest is wider, the noise shaping becomes even more effective since using the same low pass filter even more noise will be filtered away, Figure 2.1(c) [3]. This is the main reason for the previous oversampling before the noise shaping. Oversampling the signal by 64, the band of interest is also 64 times wider, which means that there will be more space to throw out the noise with the noise shaping, which will lead to more noise attenuated when a low pass filter is applied.



(a)



(b)

(c)

Figure 2.1: (a) Noise shaping effect; (b) Low pass filter with and without noise shaping; (c) Low pass filter with noise shaping with two different sample frequencies: fs and fs'.

The transfer function of the sigma delta modulator used in the back-end of the hearing aid of this design can be seen in Figure 2.2.

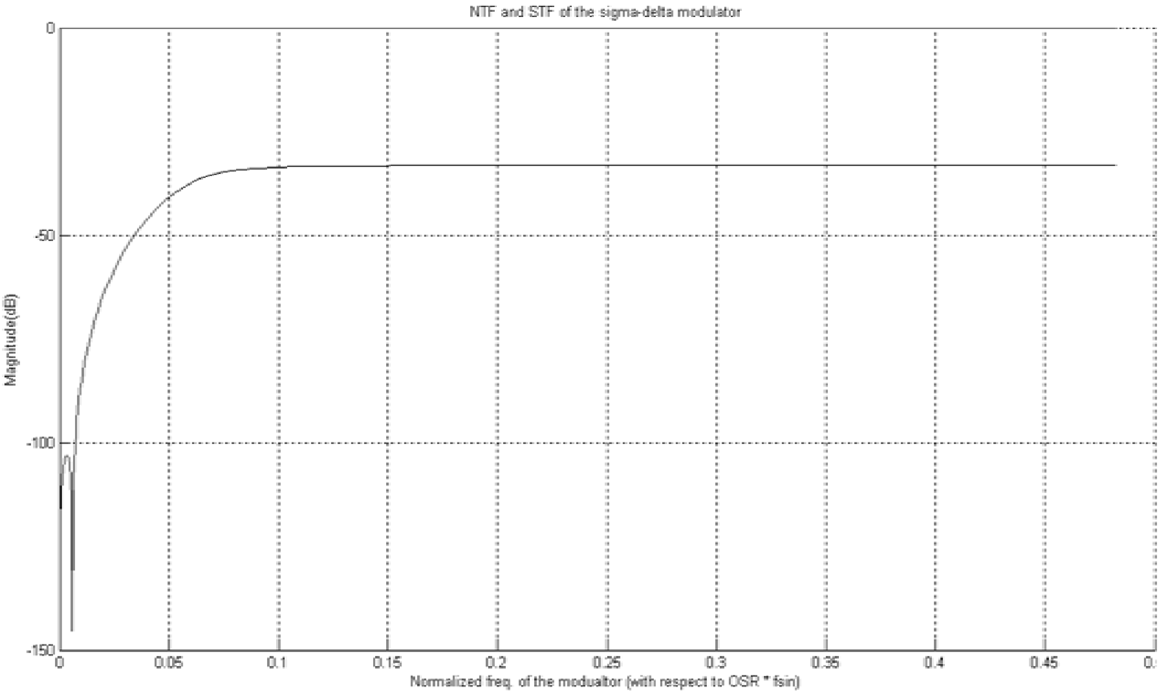


Figure 2.2: Transfer function of the sigma delta modulator.

2.2 Second stage of the interpolation filter

The second stage of the interpolation filter is a halfband hardware efficient filter designed with the sub-filters approach that is proposed by Saramäki in [5], which is normally used for decimation or interpolation purposes associated to sigma-delta modulators. This sub-filters approach consists of generating a filter with various identical repetitions of the same subfilter, which means that not just the coefficients of the filter should be found but also the coefficients of the subfilter. This approach can be understood easily with an example. It is worth to notice that since it is a halfband filter the cutoff frequencies will be symmetrical around fs/4 and the ripples in the pass band and stop band will be the same. This means that just by defining the first cutoff frequency and one of the ripples the filter can be designed. The specifications of the filter in the example will be a cutoff frequency of 0.2fs, where fs is the input sample frequency, and a ripple less than 10^{-5}. The structure of the filter can be seen in Figure 2.3, where the F blocks represent the identical repetitions of the subfilter. The structure of the subfilter can be seen in Figure 2.4.

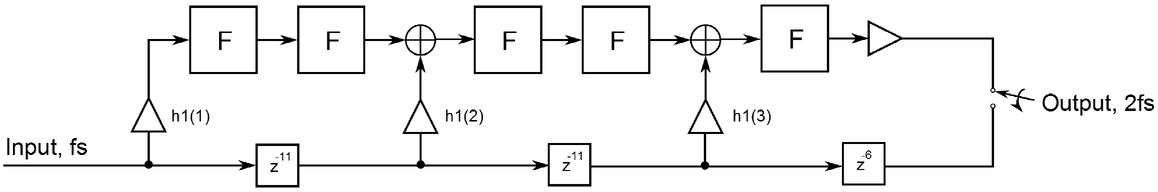


Figure 2.3: Filter structure.

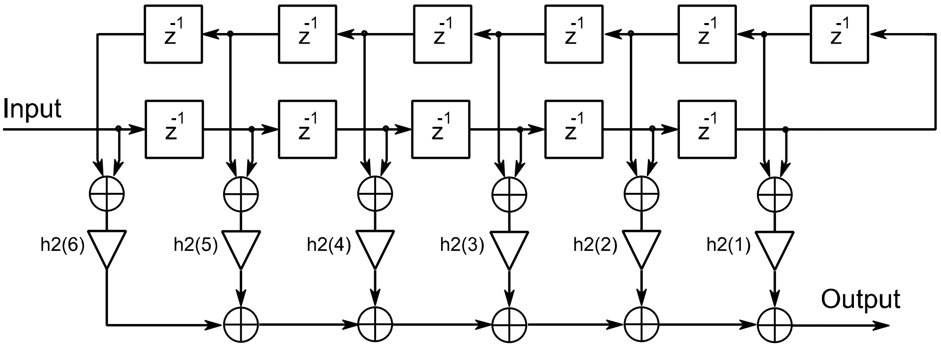


Figure 2.4: Subfilter structure.

The filter is designed using the "designhbf" function contained in the delta-sigma toolbox for Matlab that uses the same design process described by Saramäki [5]. The input variables that must be defined for the "designhbf" function are the normalized pass band cutoff frequency and the pass band and stop band ripple in absolute value. The output values will be the filter and the subfilter coefficients.

As it was explained before this second stage filter will be used to interpolate by two the input signal coming from the first stage filter. The input frequency will be 2fs, and the output frequency will be 4fs, where fs is the original sampling frequency of the input signal of the whole interpolation filter.

Since the whole interpolation filter is designed with the aim of hardware reduction, this kind of filters are very adequate because of their hardware efficiency that comes from the property of being multiplier-less[5].

2.3 CIC Filters

Cascaded integrator-comb (CIC) filters are implementations of narrowband low pass filters which are especially computationally efficient and they are used in anti-aliasing filtering before decimation and image filtering after interpolation systems [1]. The interpolation filter of the back-end of the hearing aid has four stages, and the last two are two CIC filters whose order is three and one respectively. Normally the structure of an interpolation system with CIC filters is similar to the structure in Figure 2.5.

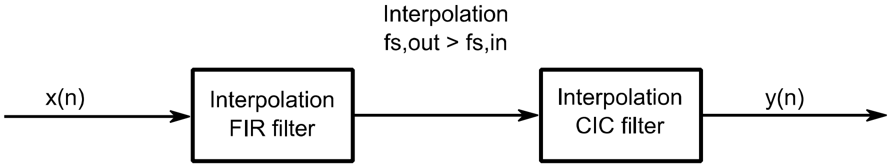


Figure 2.5: CIC filter interpolation usage in interpolation systems.

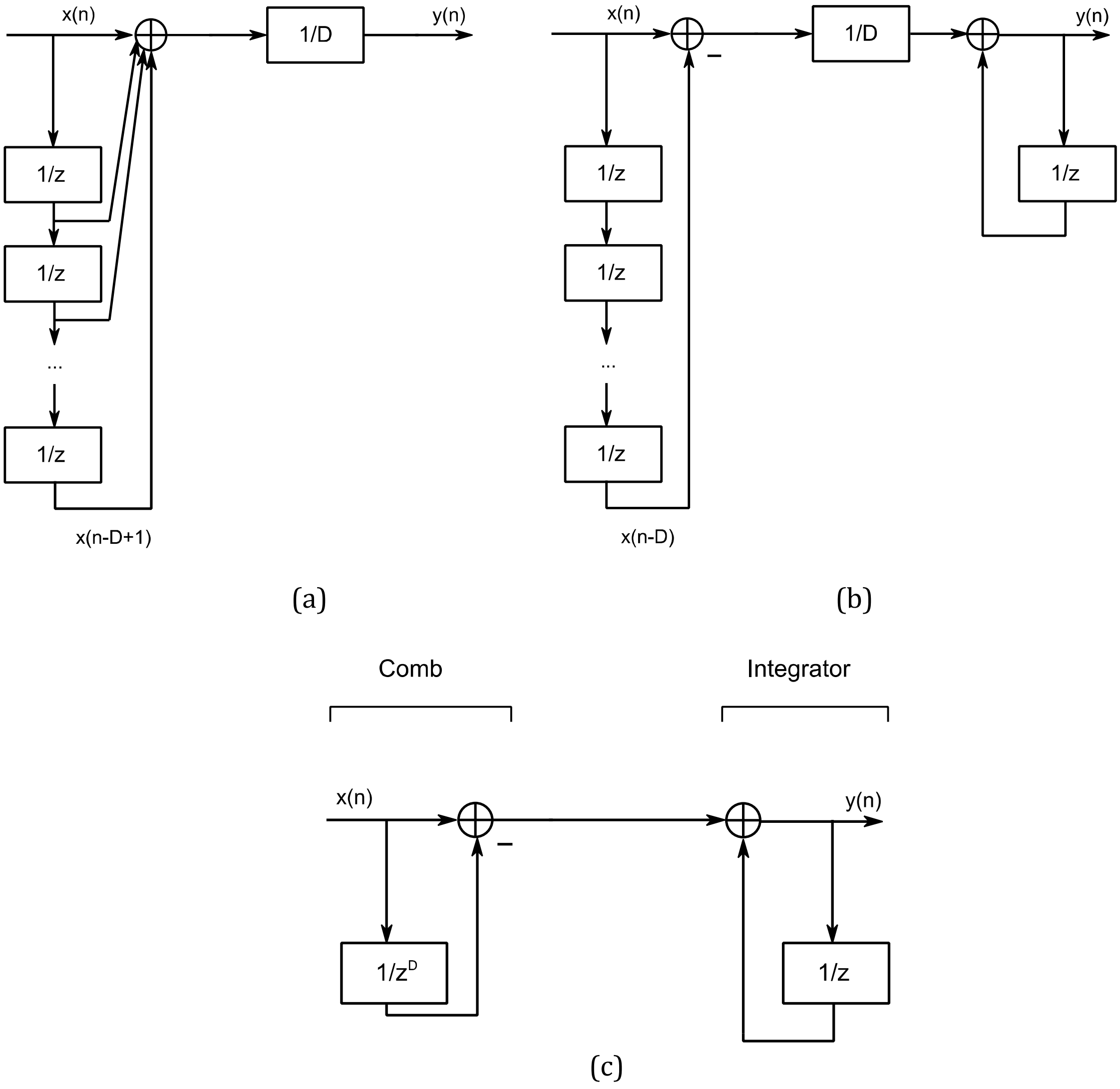


Figure 2.6: (a) D-point averager; (b) Recursive running sum filter; (c) First order CIC filter structure.

The basic idea of the CIC filters is the recursive running sum filter. This idea can be easily understood with Figure 2.6. In Figure 2.6(a) a simple structure of a D-point averager can be seen. This first approach is the standard delay-line approach of FIR filters, which will be explained in detail later. The output in this case will be:

$$y(n) = \frac{(x(n) + x(n-1) + \cdots + x(n-D+1)}{D}$$

In Figure 2.6(b) the recursive running sum filter idea is implemented. The oldest sample in the delay line is subtracted from the current sample, divided by D, and then added to the last output value. In this case the output y(n) will be expressed in function of the previous output sample y(n-1) obtaining:

$$y(n) = \frac{x(n) - x(n-D)}{D} + y(n-1)$$

Finding all the expressions for y(n), y(n-1),…y(0) it can be seen that the output is the same as the one obtained in the previous D-point averaging structure.

$$y(0) = \frac{x(0) - 0}{D} + 0;$$

$$y(1) = \frac{x(1) - 0}{D} + y(0) = \frac{x(1) + x(0)}{D};$$

$$y(2) = \cdots = \frac{x(2) + x(1) + x(0)}{D};$$

…

$$y(D-1) = \frac{x(D-1) - 0}{D} + y(D-2) = \frac{x(D-1) + \cdots + x(1) + x(0)}{D};$$

$$y(D) = \frac{x(D) - x(0)}{D} + y(D-1) = \frac{x(D) - x(0) + x(D-1) + \cdots + x(1) + x(0)}{D} =$$

$$= \frac{x(D) + x(D-1) + \cdots + x(1)}{D};$$

…

$$y(n) = \cdots = \frac{x(n) + x(n-1) + \cdots + x(n-D+1)}{D};$$

The delay line is the part of the CIC filter called Comb, the part that add that adds the last output sample is the part called Integrator, and both together generate the structure of a first order CIC filter Figure 2.6(c).

In order to reduce the hardware power consumption of this CIC filters using non recursive structures, a simple trick can be used. This trick only requires that the sample ratio that the interpolation CIC filter increases the frequency with must be multiple of two. The basic idea of this trick is to divide the CIC filter in lower order sub-filters that are overall simpler and less hardware demanding than the original one [1]. It's an approach similar to the four step splitting of the interpolation filter.

The recursive and non recursive expressions of the z-domain transfer function of an M-th order CIC filter are:

$$H_{CIC}(z) = \left(\frac{1 - z^{-D}}{1 - z^{-1}}\right)^M = (1 + z^{-1} + z^{-2} + \cdots + z^{-D+1})^M$$

Assuming that D is an integer positive power of two, $D = 2^k$, the non recursive expression of the transfer function can be factored as:

$$H_{CIC}(z) = (1 + z^{-1})^M (1 + z^{-2})^M (1 + z^{-4})^M \ldots (1 + z^{-2^{K-1}})^M$$

The upside of this factorization is that now the CIC filter is implementable with K non recursive stages without any feedback as it can be seen in Figure 2.7 [1]. Since the frequency between blocks is doubled all the stages will be the same, so no extra design is needed for each part.

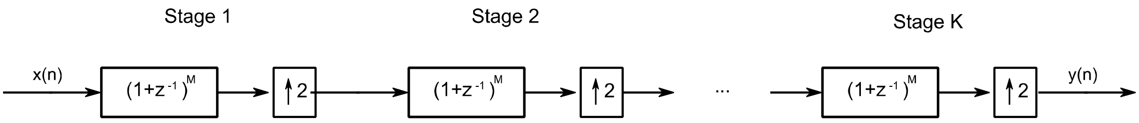


Figure 2.7: Factorized structure of the CIC filter.

The transfer function of the two CIC filters used in the third and fourth stage of the interpolation filter can be seen in Figure 2.8 and Figure 2.9 respectively.

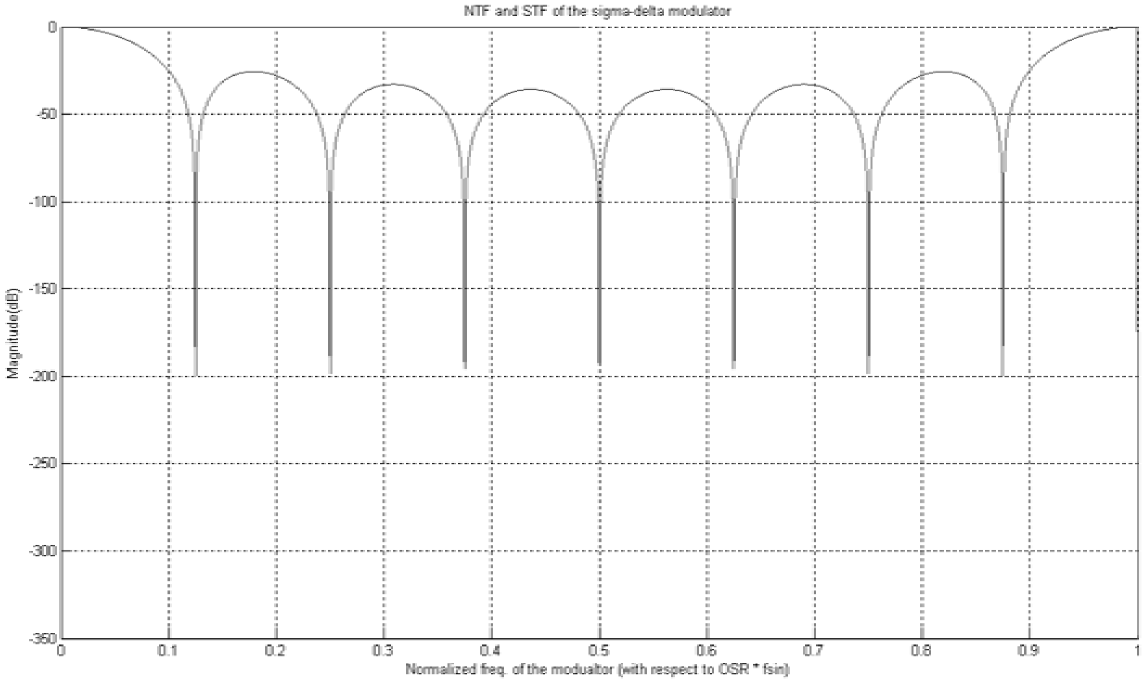


Figure 2.8: Transfer function of the 3rd order CIC filter used in the 3rd stage of the interpolation filter.

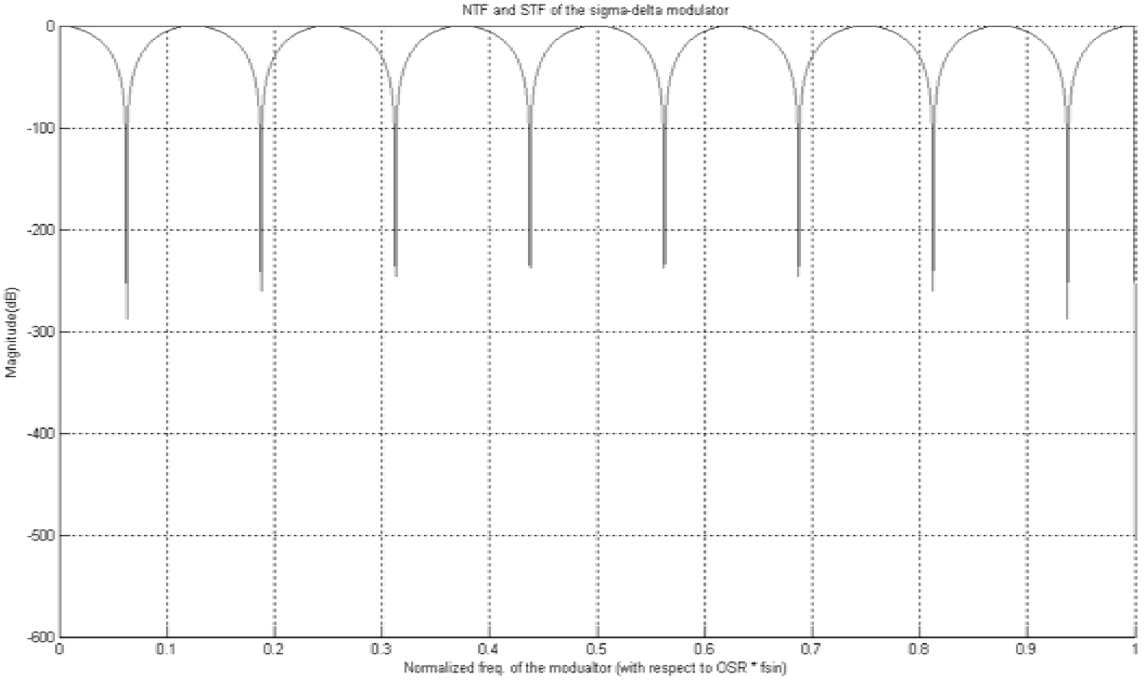


Figure 2.9: Transfer function of the 1st order CIC filter used in the 4th stage of the interpolation filter.

2.4 FIR filters

For the first stage of the interpolation filter, a FIR filter is chosen. Although for the same attenuation needed the order of an IIR filter is usually lower which might translate into a lower hardware demand, a FIR filter offers the possibility of achieving linear phase, an adequate characteristic for a filter used in audio applications because the group delay is constant, which means that all the frequencies are delayed equally [1]. IIR filter can only try to approximate linear phase but they will never achieve a constant group delay. FIR filters are also more stable, and easier to control, which will help to keep the design simple.

A finite impulse response (FIR) filter only operates with present and past samples. For that reason, a finite number of nonzero input samples will generate a finite number of nonzero output samples [1]. The way in which these kinds of filters calculate the output samples is based on weighted addition of a certain number of input samples. It could be said that it's a process similar to averaging. An example will be used to illustrate how a FIR filter operates. Let's assume that the average of the number of people crossing a certain street every four hours needs to be found. In Figure 2.10 and Table 2.1 the arbitrary numbers are shown.

Hour index $\boldsymbol{n}$	**People crossing the street in an hour** $\boldsymbol{x(n)}$	**Average of people crossing the street in the last 4 hours** $\boldsymbol{y_{av}(n)}$
0	6	-
1	9	-
2	8	-
3	12	8,75
4	3	8
5	11	8,5
6	7	8,25
7	6	6,75
8	6	7,5
9	13	8
10	17	10,5
11	15	12,75
12	0	11,25
13	0	8
14	0	3,75
15	0	0
16	0	0
17	0	0

Table 2.1: Averager example sample table.

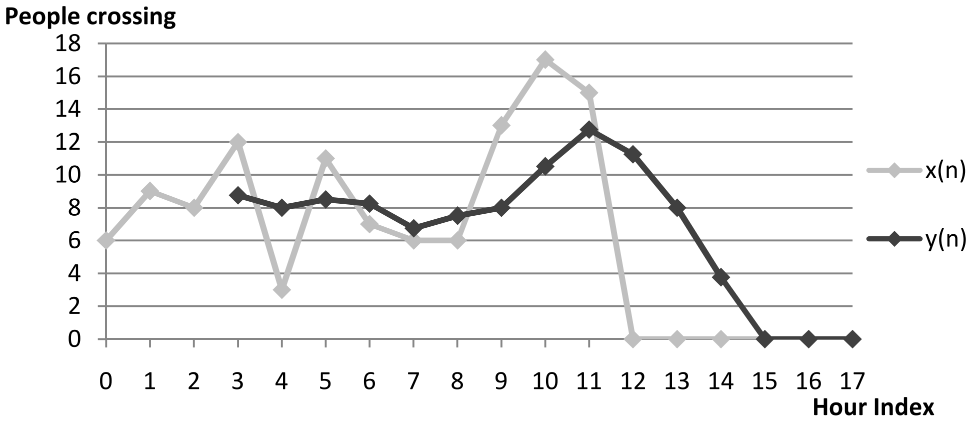


Figure 2.10: Plot of the Table 1.

The average of people crossing the street in the last four hours is obtained by adding the number of people crossing the street in each of those hours, and dividing it by four:

$$y_{av}(3) = \frac{x(3) + x(2) + x(1) + x(0)}{4} = \frac{6 + 9 + 8 + 12}{4} = 8{,}75$$

$$y_{av}(4) = \frac{x(4) + x(3) + x(2) + x(1)}{4} = \frac{9 + 8 + 12 + 3}{4} = 8{,}00$$

...

$$y_{av}(11) = \frac{x(11) + x(10) + x(9) + x(8)}{4} = \frac{15 + 17 + 13 + 6}{4} = 12{,}75$$

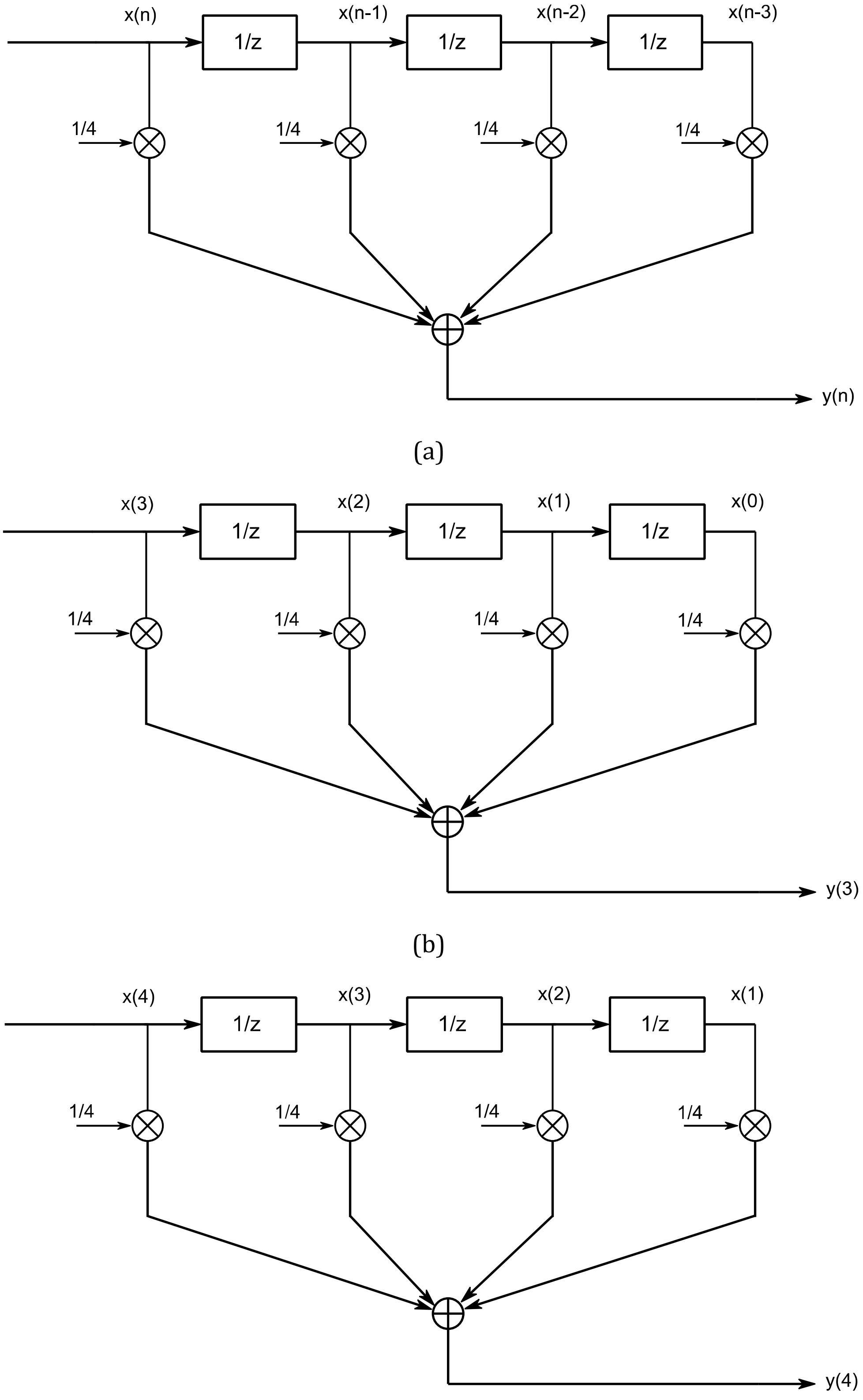
...

$$y_{av}(15) = \frac{x(15) + x(14) + x(13) + x(12)}{4} = \frac{0 + 0 + 0 + 0}{4} = 0{,}00$$

In a general case, the *nth* output sample will be:

$$y_{av}(n) = \frac{x(n) + x(n-1) + x(n-2) + x(n-3)}{4} = \frac{1}{4} \cdot \sum_{n-3}^{n} x(m)$$

This expression suggests that the *nth* output sample is obtained averaging the *nth* input sample and the three previous input samples. The block diagram in Figure 2.11 formalizes this averager into a digital FIR filter structure.



(a)

(b)

(c)

Figure 2.11: (a) General structure of the averager; (b) Step one; (c) Step two.

In Figure 2.11(a) it can be seen that the input samples keep shifting through the delay line, in where they are stored temporally to make the calculation of the output before being shifted again. In Figure 2.11(b) x(0), x(1), x(2) and x(3) are multiplied by 1/4 and added all together. After a shift, the situation in Figure 2.11(c) is reached. Now, x(1), x(2), x(3) and x(4) are used in the same way. Note that the expression of the block diagram in Figure 2.11(a) is the following one, which exactly matches with the expression of the averager.

$$y_{av}(n) = \frac{1}{4} \cdot x(n) + \frac{1}{4} \cdot x(n-1) + \frac{1}{4} \cdot x(n-2) + \frac{1}{4} \cdot x(n-3) = \frac{1}{4} \cdot \sum_{n-3}^{n} x(m)$$

That same principle can be applied in the calculation of the output samples of a FIR filter; however the coefficients that are multiplied by each input sample does not have to be the same. In fact, the value of each of those coefficients, *coefficients of the filter*, and the number of input samples used to calculate the output, *number of taps*, are the parameters that define the behavior of the FIR filter. A block diagram of a general *M-tap* FIR filter with coefficients *h(k)* is shown in the Figure 2.12.

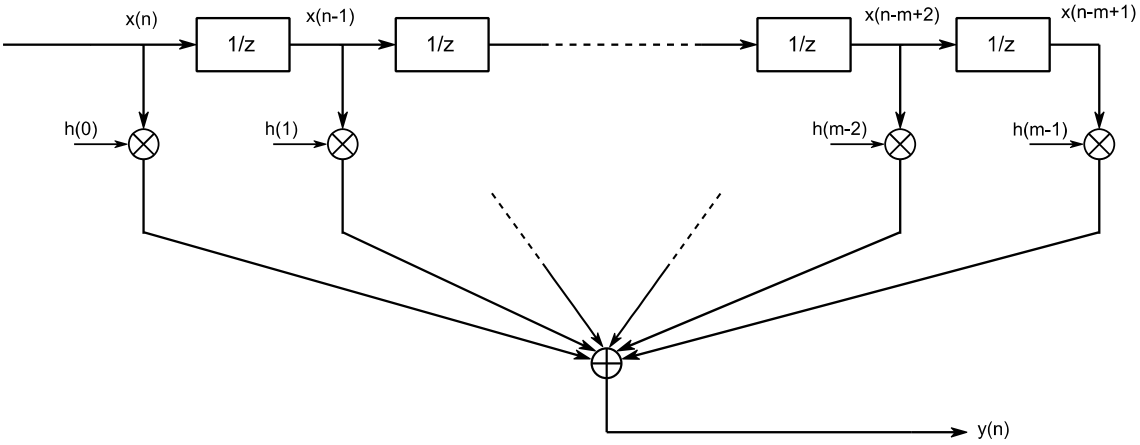


Figure 2.12: General structure of an m-tap FIR filter.

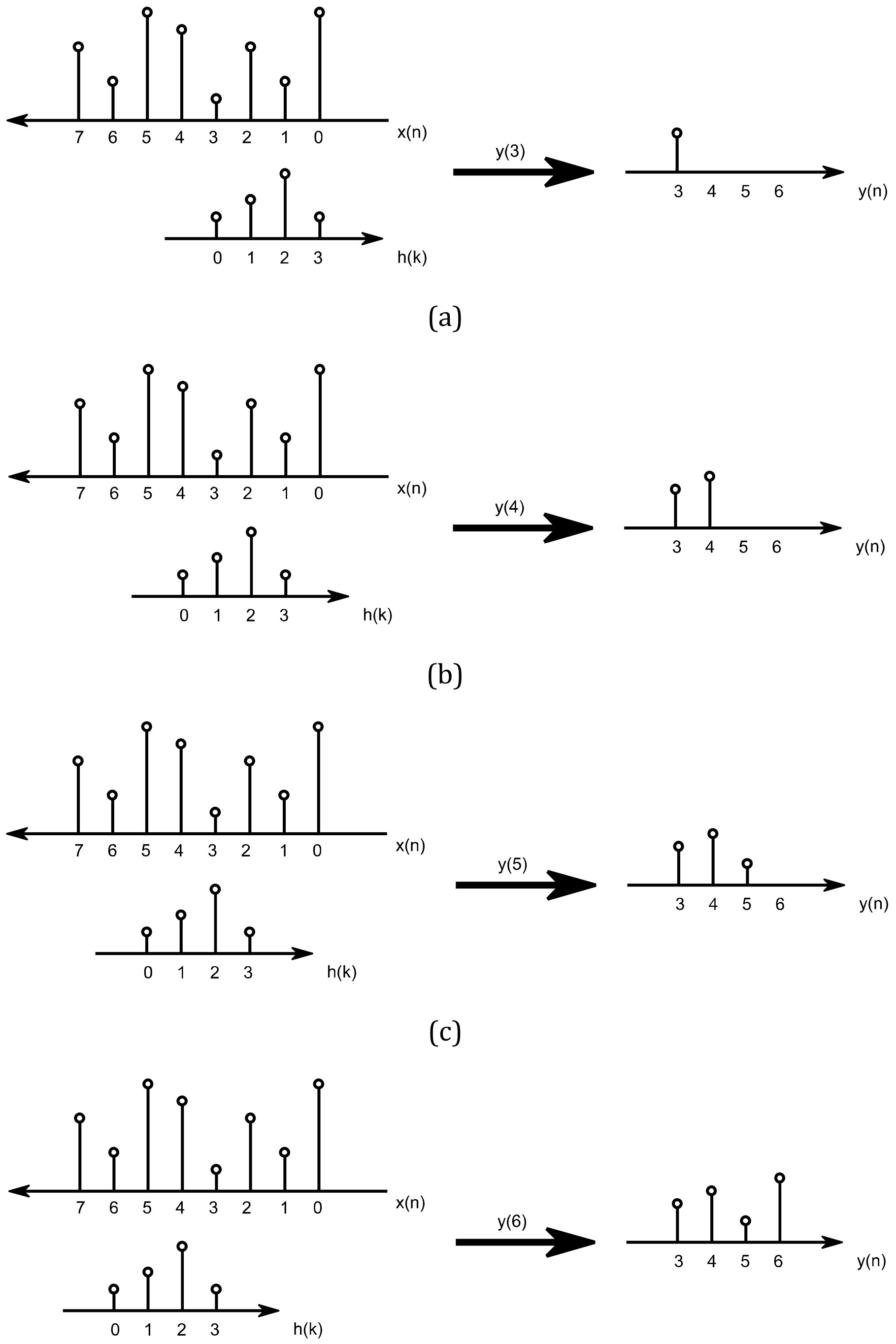
As it can be seen in Figure 2.12, the expression of the *nth* output of a general *m-tap* FIR filter is Eq.(2.1), where *h(k)* are the FIR filter coefficients.

The result obtained for the *nth* output sample, is the *convolution* between *h(k)* and *x(n)*, so the Eq.(2.1) can be rewritten with the abbreviated notation of the *convolution* (with the symbol *) as Eq.(2.2).

$$y(n) = h(m-1) \cdot x\big(n-(m-1)\big) + \cdots + h(0) \cdot x(n) = \textstyle\sum_{k=0}^{m-1} h(k) \cdot x(n-k) \qquad (2.1)$$

$$y(n) = \textstyle\sum_{k=0}^{m-1} h(k) \cdot x(n-k) = h(k) * x(n) \qquad (2.2)$$

The way that the *convolution* between two functions works can be simply understood with the following example of a *4-tap* FIR filter.



(a)

(b)

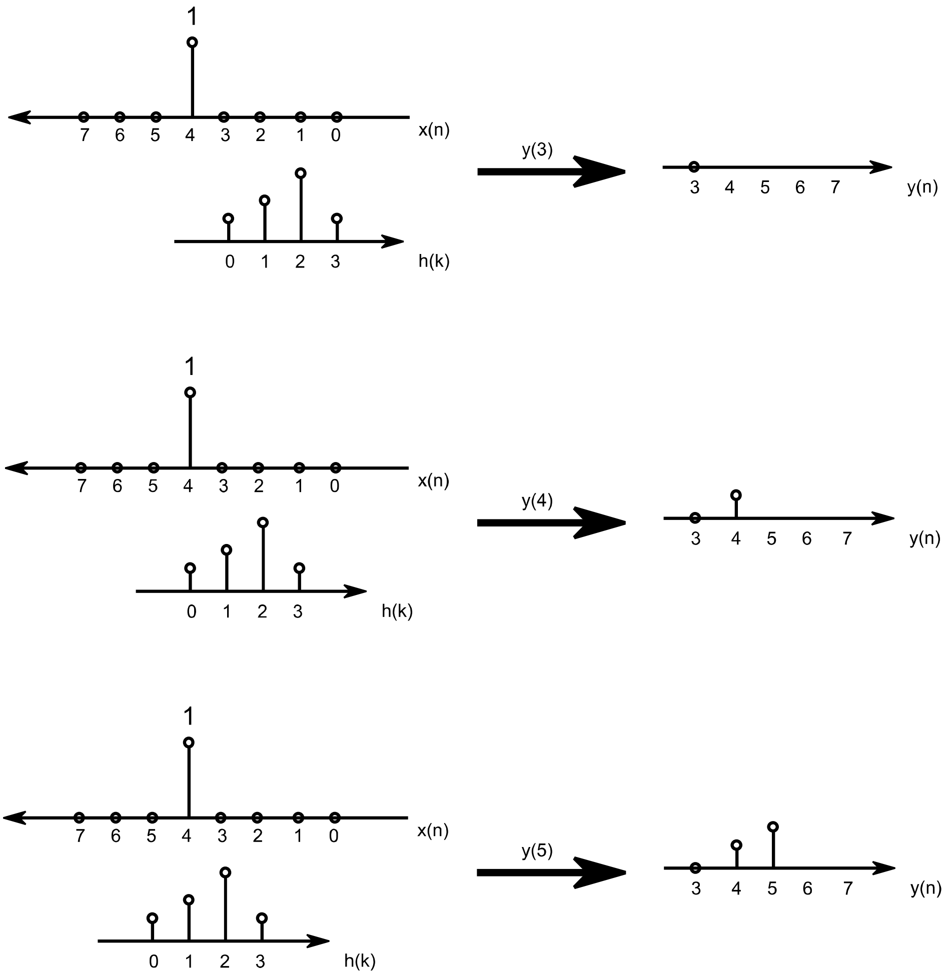
(c)

(d)

Figure 2.13: Convolution example, output calculation of: (a) y(3); (b) y(4); (c) y(5); (d) y(6);

In Figure 2.13(a) the first step of the convolution is shown; the coefficients are multiplied by the input samples, and added all together forming the first output sample. Shifting left the coefficients from Figure 2.13(a), Figure 2.13(b) is obtained. The operation procedure to obtain the second output sample is the same used in the first step; multiplication and addition of the coefficients and the input samples. The following output samples are generated the same way shifting the coefficients to the left every step Figure 2.13(c) and Figure 2.13(d).

Once understood how the convolution operates, it is easy to see that the impulse response of a FIR filter exactly corresponds to the coefficients of the filter [1]. The input will be an impulse, which means that just one of the samples of the input will be 1, being zero all the others. Once being convoluted with the filter coefficients, all the multiplications will be zero except for the one that matches the input sample equal to 1. The result of the nonzero multiplication will be the filter coefficient itself. Shifting the filter coefficients through all the input samples, all the *y(n)* output values will be found, which will be the same sequence of values as the filter coefficients. In Figure 2.14, an example of the convolution of an impulse input signal and the coefficients of a filter is exposed.



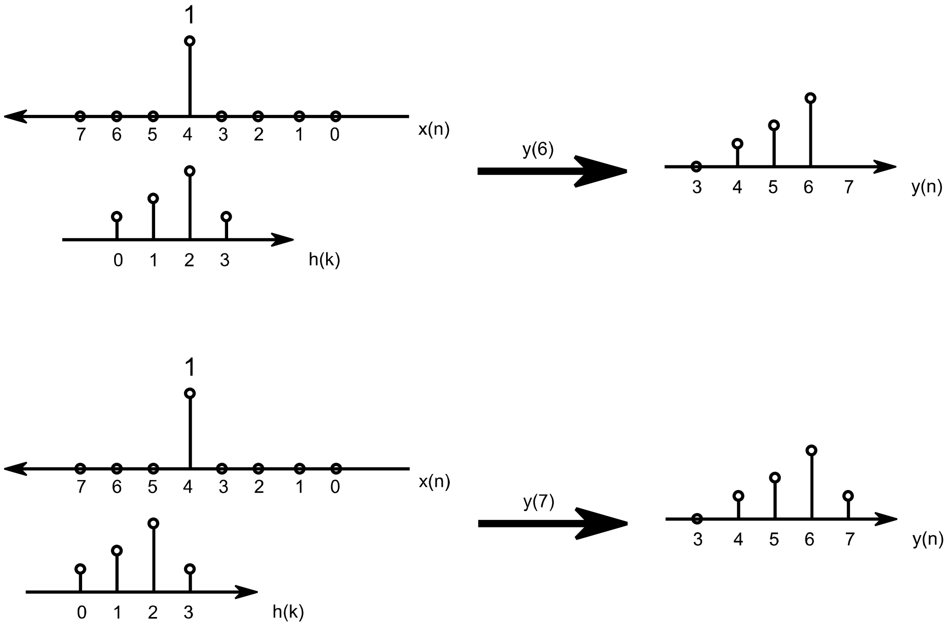
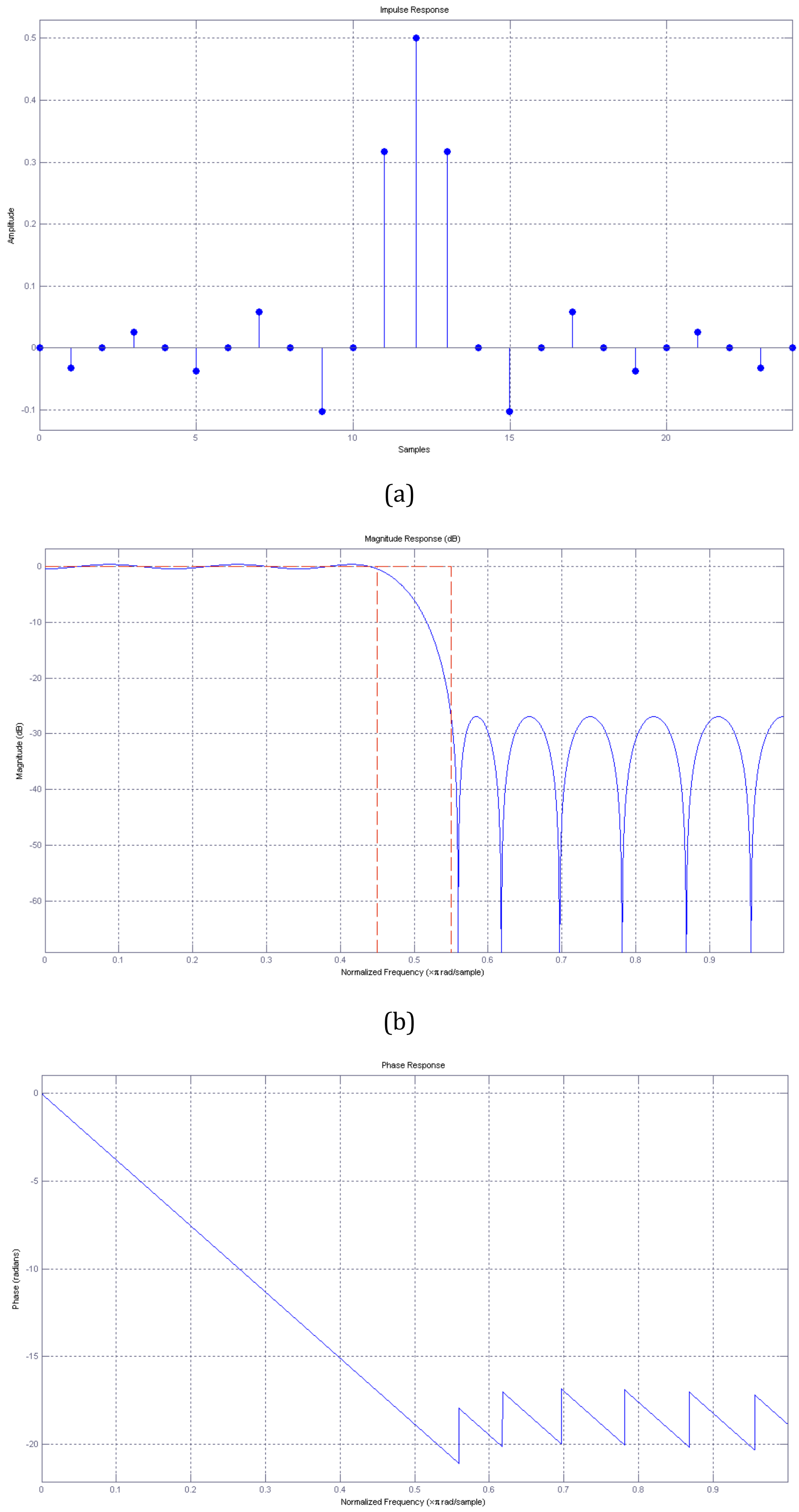


Figure 2.14: Showing the convolution with an impulse.

The importance of the convolution between the input signal and the filter coefficients resides in the fact that in frequency domain, this convolution is translated into a simple multiplication [1]. That means that assuming that $X(m)$ and $H(m)$ are the discrete Fourier transformation of $x(n)$ and $h(k)$ respectively, the discrete Fourier transformation of $y(n) = h(k) * x(n)$ will be $Y(m) = H(m) \cdot X(m)$. The inverse process is also true; if $H(m)$ and $X(m)$ are found, the time domain output $y(n)$ can be calculated as the inverse discrete Fourier transformation of $H(m) \cdot X(m)$. As it will be seen in the design of our interpolation filter, this characteristic is very useful for the design of a FIR filter with certain specifications.

One of the reasons why FIR filters are used is the property of being able to achieve linear phase [1]. This characteristic can easily be seen with an example. A DFT of a 25 symmetrical coefficients FIR filter Figure 2.15(a) is performed, obtaining the H(m) argument Figure 2.15(b) and phase Figure 2.15(c) in where the linear phase can clearly be seen. This linear phase comes from the symmetry of the FIR filter coefficients [6]. The mathematical process to deduce that fact can be seen with the following example. In Figure 2.16 the odd number of symmetric coefficients of a FIR filter can be seen. This coefficients can be expressed with the Eq.(2.3), where z^{-1} corresponds to a delay. Since the coefficients are symmetric, Eq.(2.3) can be expressed as Eq.(2.4) since h(0)=h(6), h(1)=h(5), h(2)=h(4). Eq.(2.4) can also be arranged to find the expression in Eq.(2.5) just by taking common factor of z^{-3}. With the expression in Eq.(2.5) the inverse discrete Fourier transformation (IDFT) can be used to find the expression of the coefficients in frequency domain obtaining Eq.(2.6). $H_R(\omega)$ is a real function that can be either positive or negative in any frequency, and the phase comes from the $e^{j\theta(\omega)}$ factor which in this example is $e^{-j3\omega}$. The phase angle $\theta(\omega)$ will be 3ω which is clearly proportional to the frequency, so it can be said that this filter has linear phase. If the coefficients wouldn't had been symmetrical these simplifications wouldn't had been possible and the resulting phase wouldn't had been linear with the frequency.



(a)

(b)

(c)

Figure 2.15: (a) 25 FIR filter coefficients; (b) Argument; (c) Phase.

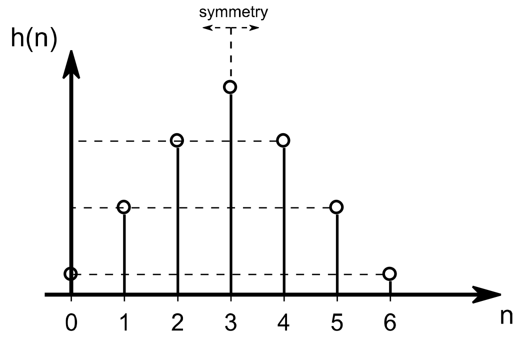


Figure 2.16: FIR filter coefficients. Odd number of coefficients, and symmetrical.

$$H(z^{-1}) = h(0) + h(1)z^{-1} + h(2)z^{-2} + h(3)z^{-3} + h(4)z^{-4} + h(5)z^{-5} + h(6)z^{-6} \quad (2.3)$$

$$H(z^{-1}) = h(0)[1 + z^{-6}\,] + h(1)[z^{-1} + z^{-5}] + h(2)[z^{-2} + z^{-4}] + h(3)z^{-3} \quad (2.4)$$

$$H(z^{-1}) = z^{-3}\{h(0)[z^{3} + z^{-3}\,] + h(1)[z^{2} + z^{-2}] + h(2)[z^{1} + z^{-1}] + h(3)\} \quad (2.5)$$

$$H\left(e^{-j\omega}\right) = e^{-j3\omega}\{2h(0)\cos(3\omega) + 2h(1)\cos(2\omega) + 2h(2)\cos(\omega) + h(3)\} =$$

$$= e^{j\theta(\omega)}\{H_R(\omega)\} \quad (2.6)$$

2.5 Low pass filters

The low pass filters theory that will be explained in this part can be found in [1] and [7]. As it was described before, the interpolation filter that will be designed doubles the frequency at the output ($f_{s_{out}} = 2 \cdot f_{s_{in}}$). This means that the band of interest at the output will be twice as big as the band of interest in the input. While the band interest in the input is from $-f_{s_{in}}/2$ to $+f_{s_{in}}/2$, the band of interest in the output is $-f_{s_{out}}/2$ to $+f_{s_{out}}/2$ which is equivalent to $-f_{s_{in}}$ to $+f_{s_{in}}$. Since the input of the filter is discrete with sample frequency $f_{s_{in}}$, there will be repetitions of its frequency spectrum every multiple of $f_{s_{in}}$, which are all out of the input band of interest but not from the output band of interest. In fact, the first repetition of the spectrum situated at $-f_{s_{in}}$ and $+f_{s_{in}}$ is contained in it and needs to be suppressed because they will be in the output band of interest. This explanation can be visualized in Figure 2.17.

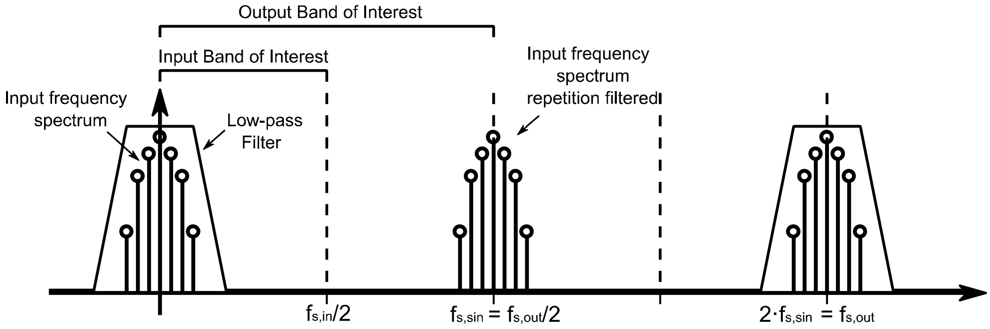
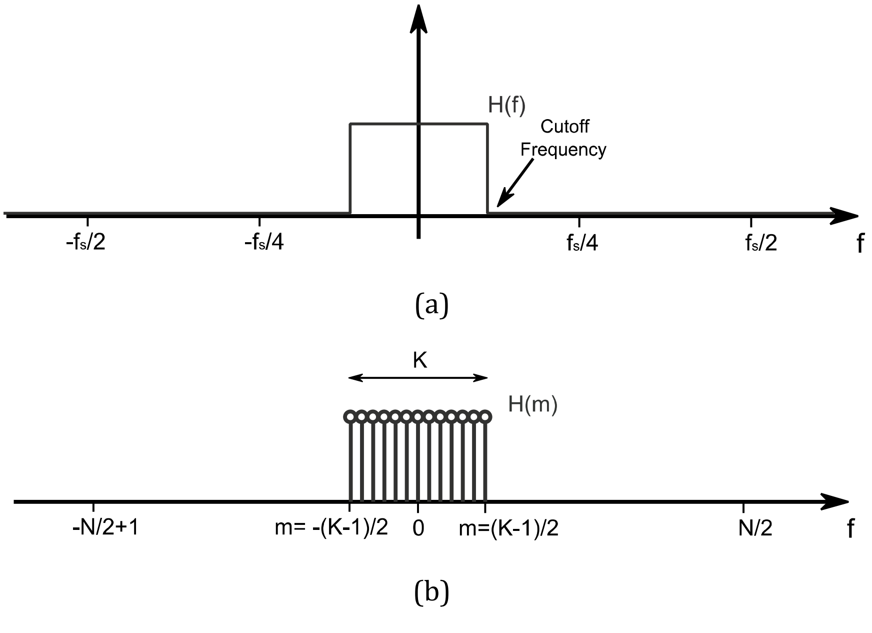
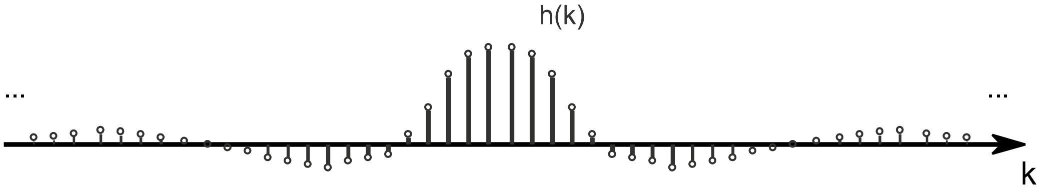

Output Band of Interest
Input Band of Interest
Input frequency spectrum
Low-pass Filter
Input frequency spectrum repetition filtered
$f_{s,in}/2$
$f_{s,sin} = f_{s,out}/2$
$2 \cdot f_{s,sin} = f_{s,out}$

Figure 2.17: Frequency spectrum with the input and output bands of interest.

As it was stated before a FIR filter is defined by its coefficients hence designing a FIR filter directly means finding the time domain coefficients that satisfies the specifications of the frequency response needed. This process consists in various steps, being the first one defining the ideal continuous frequency response of the filter Figure 2.18(a). This ideal response has to be adjusted according to the cut off frequency needed, obtaining this way the H(f) filter coefficients. In the second step, H(f) is discretized obtaining an infinite number of H(m) coefficients of the filter that generates that continuous ideal frequency response H(f) Figure 2.18(b). Afterwards the inverse discrete Fourier transformation will be applied to H(m) generating the infinite time domain h(k) coefficients that also have the ideal frequency response specified Figure 2.18(c). The function h(k) obtained has been derived after some algebraic operation applying the direct definition of IDFT to the H(m) Eq.(2.7), until Eq.(2.8) is reached. The shape of h(k) is a sinc(x)= sin(x)/x function, which shows that the coefficients of the low pass filter will be symmetric. As it was stated before, this symmetry will lead to a linear phase response. All this procedure is normally carried out by mathematical software because the algebraic complexity of the IDFT that might lead to errors.





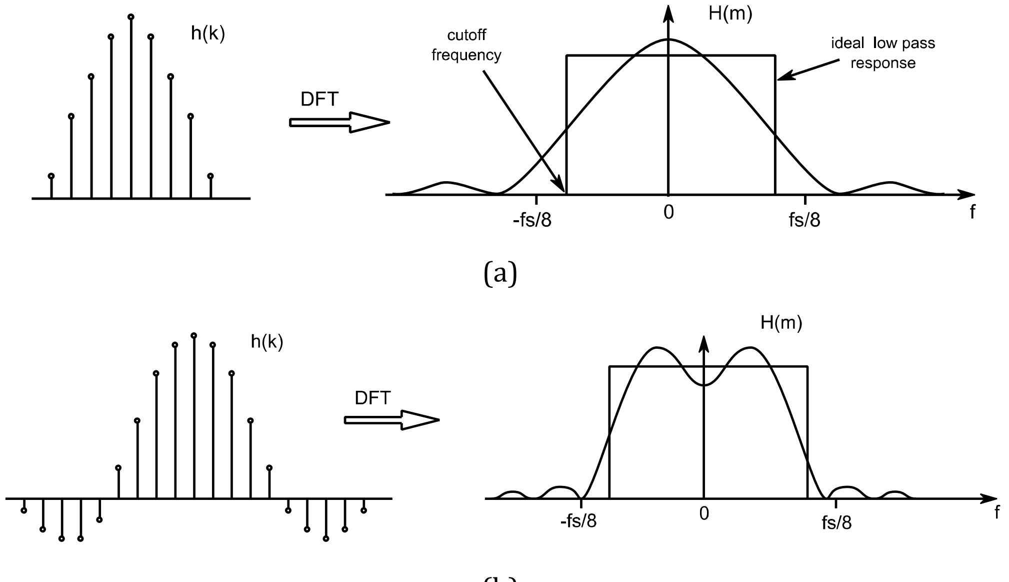
(c)

Figure 2.18: (a) Generic continuous frequency response H(f); (b) Generic discrete frequency response H(m); (c) Time domain h(k) coefficients.

$$h(k) = \frac{1}{N} \sum_{m-\left(\frac{N}{2}\right)+1}^{N/2} H(m) \cdot e^{j2\pi mk/N} \tag{2.7}$$

$$h(k) = \frac{1}{N} \cdot \frac{\sin(\pi kK/N)}{\sin(\pi k/N)} \tag{2.8}$$

Obviously, the last step will consist in deciding how many of those infinite h(k) coefficients should be taken in order to approximate the ideal frequency response. As it can be seen in Figure 2.19, the more coefficients taken the more ideal the response will be, which means that the transition band will be narrower, and the filter will be sharper. However, there will be also more side lobs also and more hardware will be needed to implement the filter. Basically it is a tradeoff between audio quality and hardware saving that depending on the application of the filter will be more balanced to one side or the other. The approach that will be used in this paper to balance this tradeoff will be to find the lowest number of coefficients that satisfies the specifications of the filter. Because the application of the filter that will be designed in this paper is for hearing aids, the hardware used should be as low as possible in order to consume less current and save battery.



(a)

(b)

Figure 2.19: Coefficients and frequency response of two FIR low pass filters: (a) 9-tap; (b) 19-tap.

2.6 Windows

During the low-pass filter design, there is a step that allows the designer to minimize the ripple and the amplitude of the side lobs. As it has been stated, the h(k) function is the time domain infinite response that ensures the ideal frequency response desired. From this h(k) a certain amount of coefficients are taken, which is equivalent of multiplying the h(k) function per a w(k) rectangular window Figure 2.20. Using this concept, instead of multiplying the h(k) per a rectangular window, another smoother shape can be used, attenuating this way the side lobes of the h(k), like it can be seen in Figure 2.21 [1]. In this case this the smoother window used is called Blackman window.

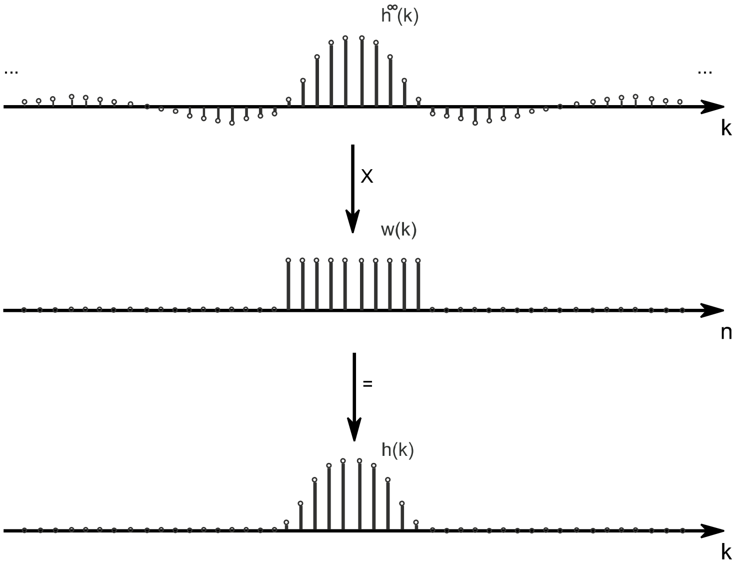


Figure 2.20: Multiplication of the infinite h(k) coefficients by a rectangular window.

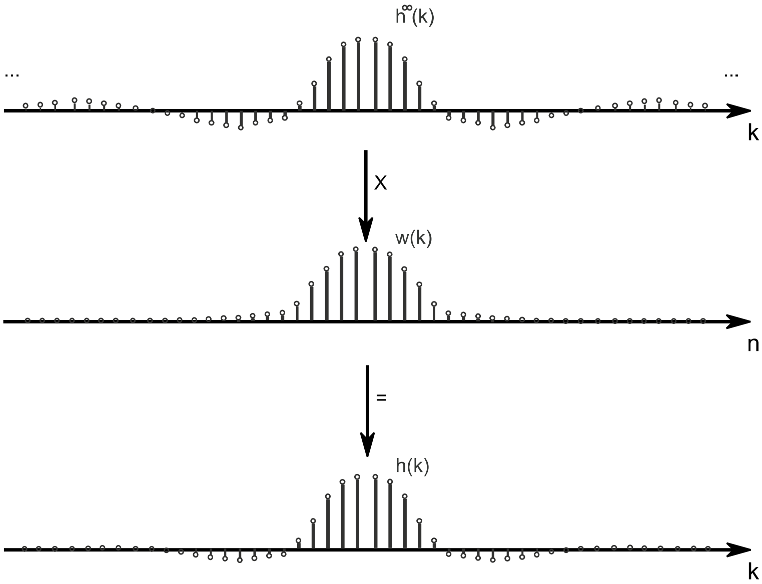
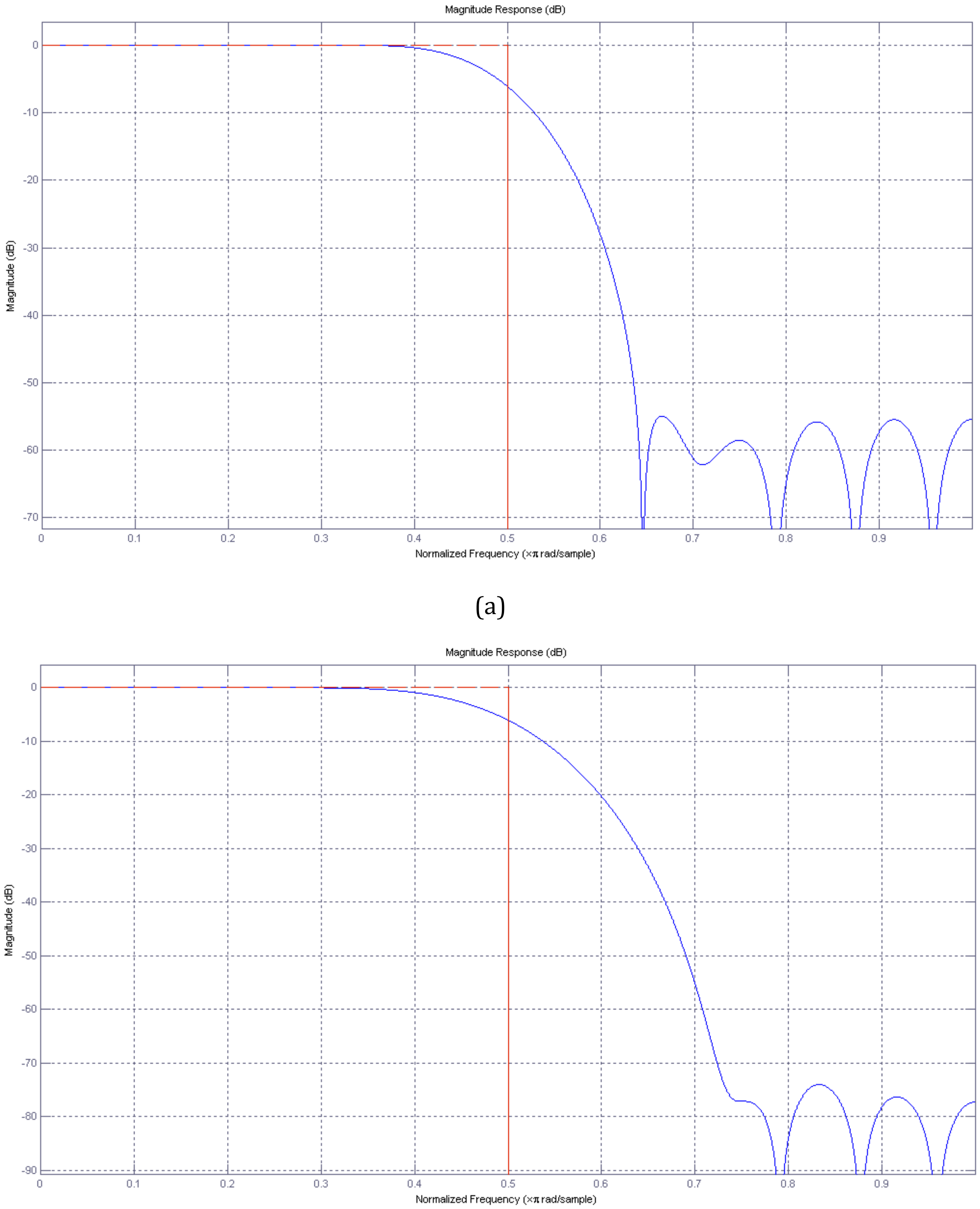


Figure 2.21: Multiplication of the infinite h(k) coefficients by a Blackman window.



(a)

(b)

Figure 2.22: Windows frequency response (a) Rectangular and; (b) Blackman window.

In order to see the effect in frequency domain of the different windows an example will be used. A halfband FIR filter with 25 coefficients with different windows applied can be seen in Figure 22. A rectangular window was used in Figure 2.22(a), which leads to a sharp frequency response but it also has higher side lobs. In Figure 2.22(b) the Blackman window was used. As it can be seen the frequency response has lower gain and has less side lobs and more attenuated, however it is smoother than when the rectangular window was used. Basically, the smoother window attenuates the frequency response but not the same amount all the frequencies. The higher frequencies are attenuated more than the lower ones, which makes the difference between the gain of the pass-band and the stop-band higher. A high number of different window shapes can be used in order to achieve different goals and specifications which leads to a very high design freedom to obtain the frequency response desired for the filter.

2.7 Polyphase filters

As it was stated before the interpolation filter that will be designed will have finite impulse response (FIR). The way this filter obtains an output sample, is multiplying the input samples and the filter coefficients, which are constant, and adding all the multiplications together. The problem here is that since the filter has to double the frequency at the output, two output samples will need to be calculated for each new input sample that enters to the delay line. In the filter structure proposed at the beginning of the paper, Figure 2.23, there is a one to one match between input samples and output samples.

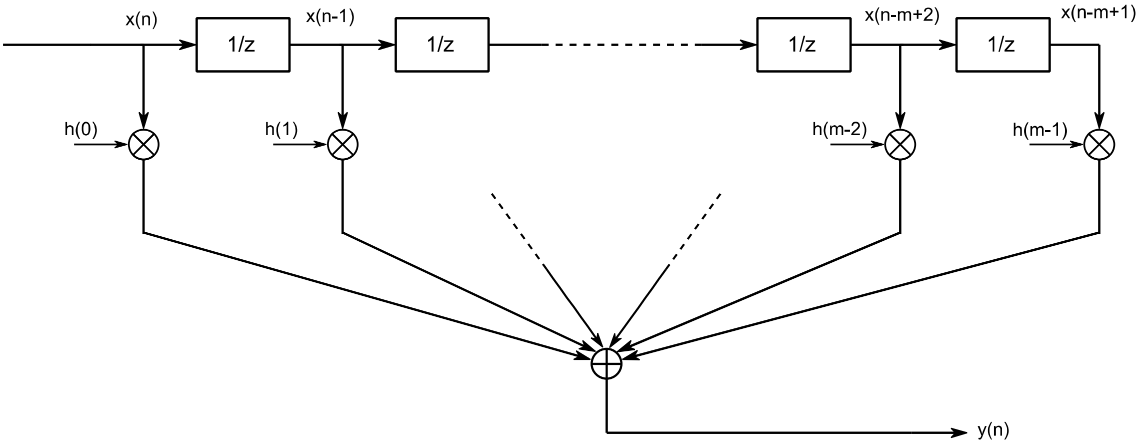


Figure 2.23: Basic m-tap FIR filter structure.

In order to match the double output frequency, a polyphase filter structure is chosen to be used. Most of the theory explained in this part can be found in [1]. Polyphase filters work very similar to a normal m-tap filter, actually the basic structure is the same, Figure 2.23, but with the difference that the discrete signal at the input is sampled with higher frequency, so the whole filter frequency is increased. Sampling a discrete signal with higher frequency is equivalent of inserting zeros between its initial samples. Once the frequency is increased, then the standard convolution that an m-tap filter uses to calculate the output is used also in the polyphase filter.

An example of this approach can be seen in Figure 2.24. In the Figure 2.24(a) the input signal sampled at fs sample frequency is shown. Sampling this input with triple frequency will mean to interpolate two zeros between each of the original input samples Figure 2.24(b). Once the input operates at the desired frequency, the normal convolution process can be performed with the coefficients of the filter Figure 2.24(c). As it can be seen in the convolution in Figure 2.24(d), the number of nonzero multiplications per output sample does not change in comparison to a standard m-tap filter since the interpolated input samples are zero and the multiplications by zero don’t need to be done. The value of the output samples will be in Figure 2.24(d) will be:

$$y_0(1) = h(0)\cdot \boldsymbol{x}(\mathbf{1}) + h(3)\cdot \boldsymbol{x}(\mathbf{0});$$

$$y_1(1) = h(1)\cdot \boldsymbol{x}(\mathbf{1}) + h(4)\cdot \boldsymbol{x}(\mathbf{0});$$

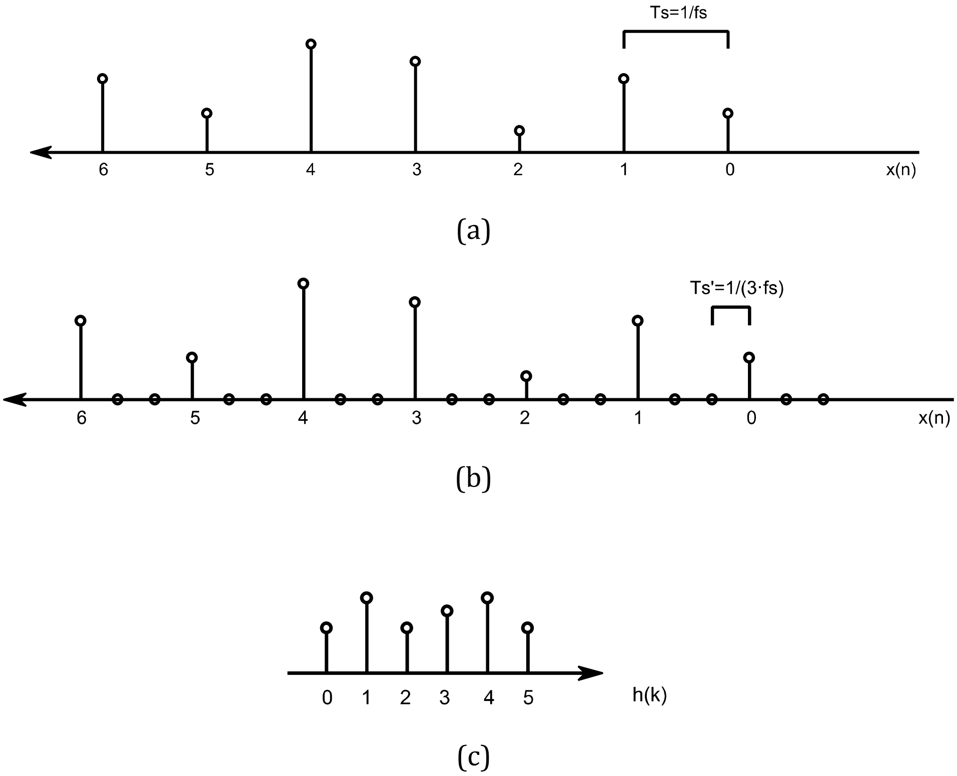
$$y_2(1) = h(2)\cdot \boldsymbol{x}(\mathbf{1}) + h(5)\cdot \boldsymbol{x}(\mathbf{0});$$

$$y_0(2) = h(0)\cdot \boldsymbol{x}(\mathbf{2}) + h(3)\cdot \boldsymbol{x}(\mathbf{1});$$

$$y_1(2) = h(1)\cdot \boldsymbol{x}(\mathbf{2}) + h(4)\cdot \boldsymbol{x}(\mathbf{1});$$

$$y_2(2) = h(2)\cdot \boldsymbol{x}(\mathbf{2}) + h(5)\cdot \boldsymbol{x}(\mathbf{1});$$

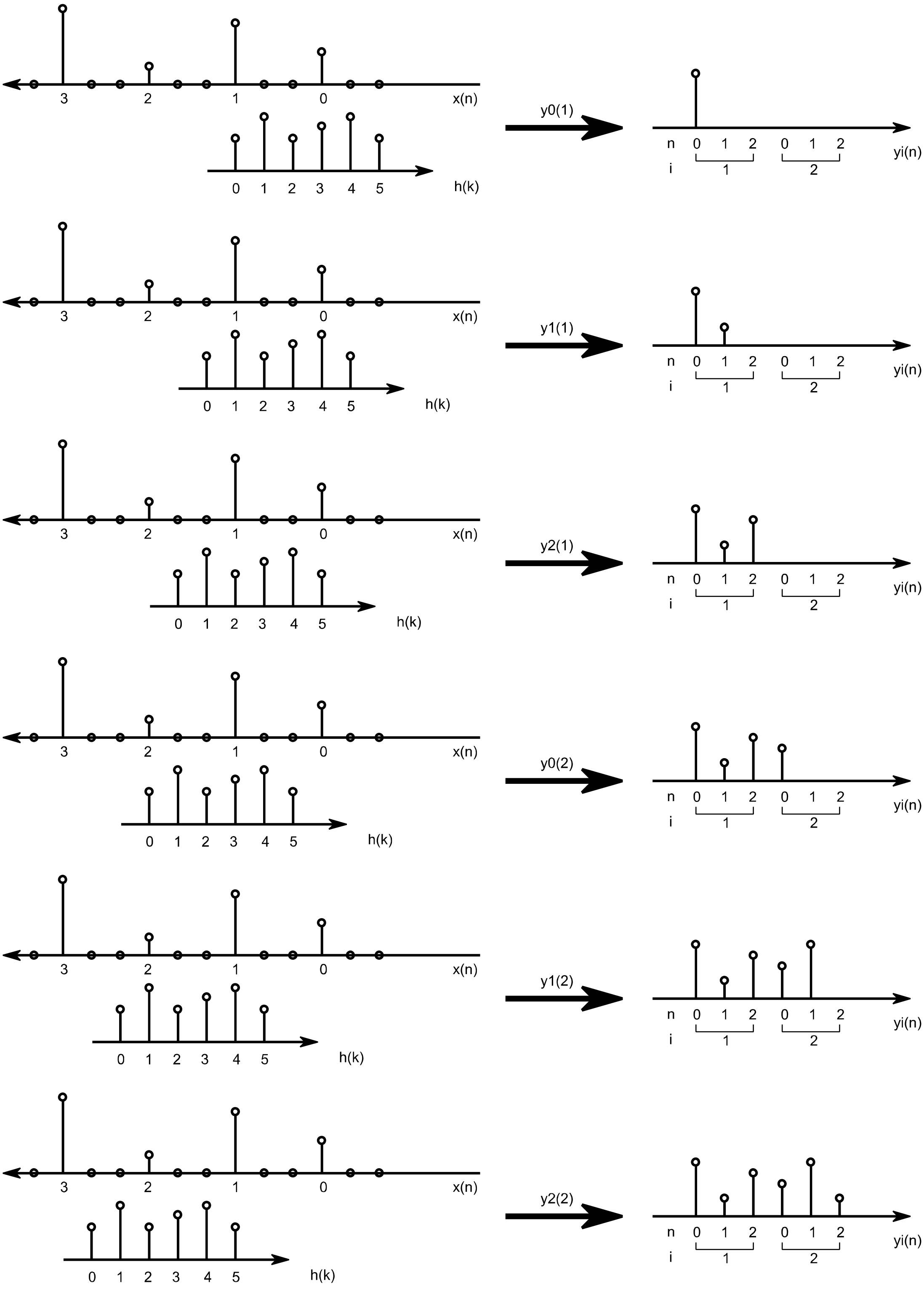
As it can be seen, the every group of three output samples are calculated with the same input samples, but multiplied by different coefficients. Using this approach, and with also the aim of saving all these useless zero multiplications, an improved structure is suggested in Figure 25.



(a)

(b)

(c)



(d)

Figure 2.24: (a) Input samples; (b) Input samples interpolated; (c) Filter coefficients; (d) Convolution.

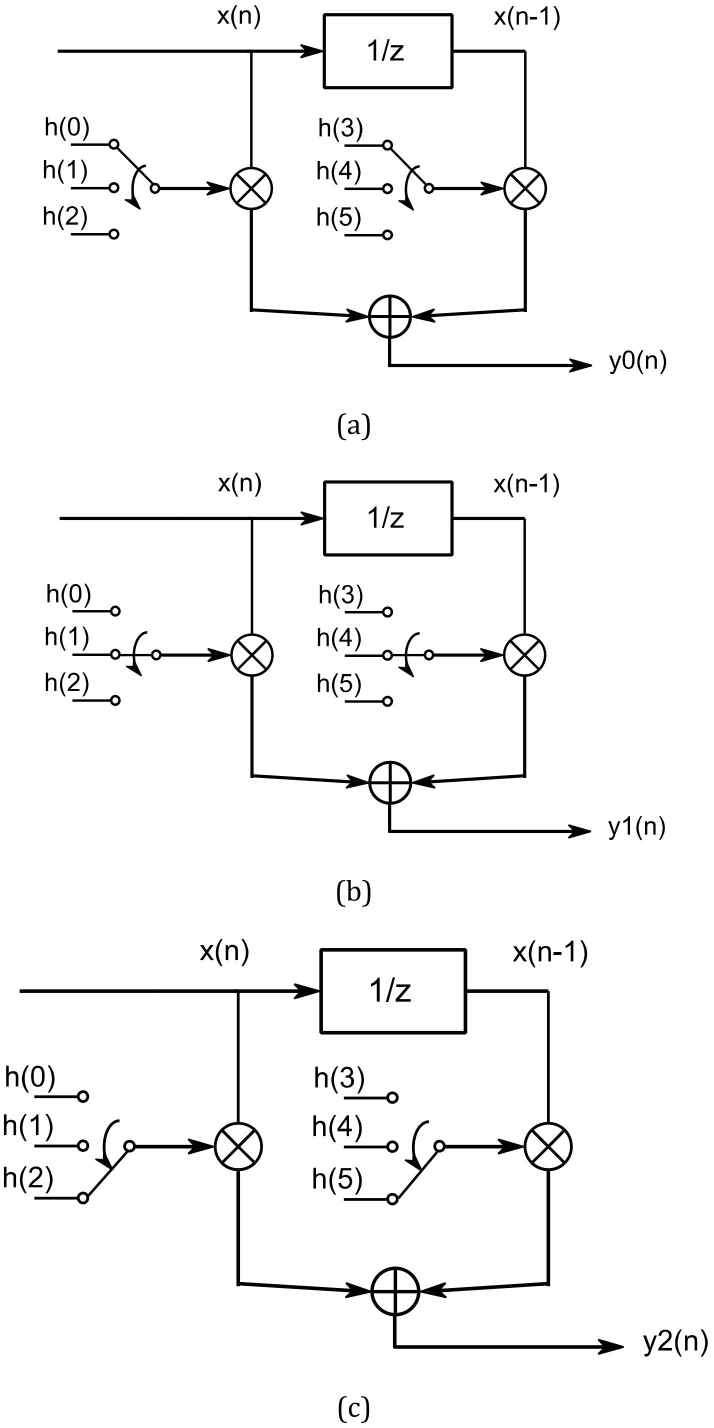


Figure 2.25: Structure proposed of the polyphase filter, calculation of the: (a) Output sample 0; (b) Output sample 1; (c) Output sample 2.

In this structure, the input samples frequency is the original one, but the selectors that switch between coefficients operate at triple frequency. This basically means that there will be three output samples calculated per each new input sample that enters to the delay line. In the first step Figure 2.25(a), the input samples will be multiplied by the coefficients on the top of the selectors, and added all together. In the following step Figure 2.25(b) all the selectors will switch down performing another multiplication of the input samples by the new coefficients selected. Finally the selectors switch again calculating the last output sample Figure 2.25(c). The expression of the three outputs calculated that can be seen in Eq.(2.9), Eq.(2.10) and Eq.(2.11), completely matches to the previous output calculations with the original structure of the polyphase filter.

$$y(0) = h(0) \cdot x(n) + h(3) \cdot x(n-1) \tag{2.9}$$

$$y(0) = h(1) \cdot x(n) + h(4) \cdot x(n-1) \tag{2.10}$$

$$y(0) = h(2) \cdot x(n) + h(5) \cdot x(n-1) \tag{2.11}$$

The upsides of this new structure are:

- The whole filter works at the original frequency and not at higher rate, which will save current consumption and hardware.
- All the useless zero multiplications are avoided.
- There is also hardware saved in the structure itself of the filter, since the number of taps is reduced by three.

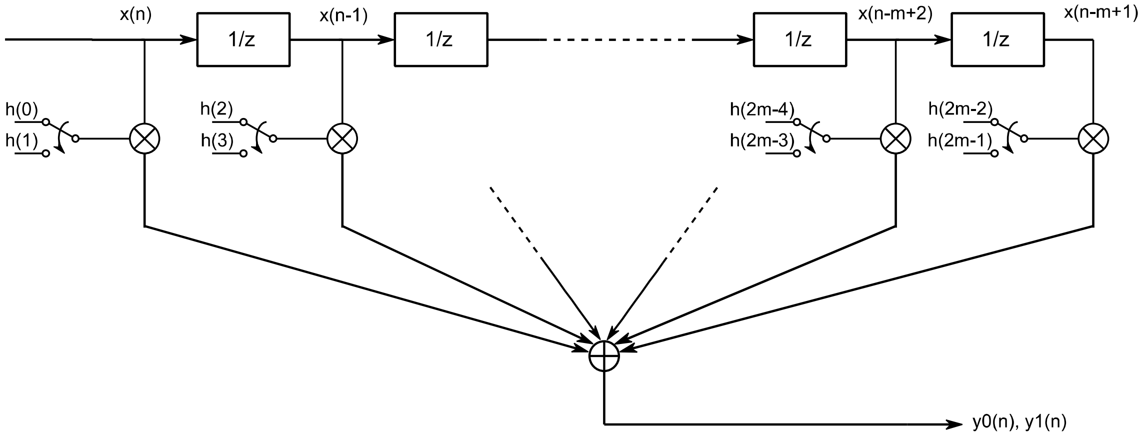


Figure 2.26: Two interpolation m-taps polyphase FIR filter.

As a final note, it's obvious to see that relation between the input frequency and the output frequency is exactly the number of coefficients per tap, since that is the number of output samples obtained every input sample (one in every switch of the selectors). For the design of the filter needed in this paper, since the output frequency needs to be double as the input frequency the number of coefficients per tap is two. A generic two interpolation m-tap filter structure can be seen in Figure 2.26.

2.8 Hardware structure of the FIR filter

Once this point is reached, the structure of the filter that needs to be designed is the structure in Figure 2.26. As it was stated before, the aim of the design of this filter is to save as much hardware and area in the microchip as possible. For that reason a generic assumption of how the hardware structure of the filter will be done in order to be able to make more improvements in the design of the filter, Figure 2.27.

The filter will have n taps (n will be decided later in the design) and since the polyphase structure will interpolate by 2, 2n coefficients will be need. These coefficients will need to be stored in a memory and will not need to be changed; hence a ROM memory should be used for that task. These coefficients will be multiplied by the n input samples that are in each moment in the delay line. These samples will need to be temporarily stored and refreshed every time that a new input sample enters to the delay line, so a RAM memory will be used for that purpose. Also, a block that multiplies and accumulates will need to be implemented in order to calculate the output samples, MAC block. Finally, a finite state machine block (FSM) that controls all the circuit will need to be designed according to the operation of the whole circuit.

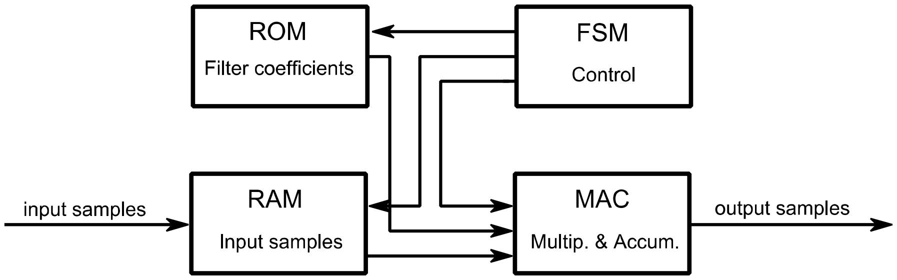


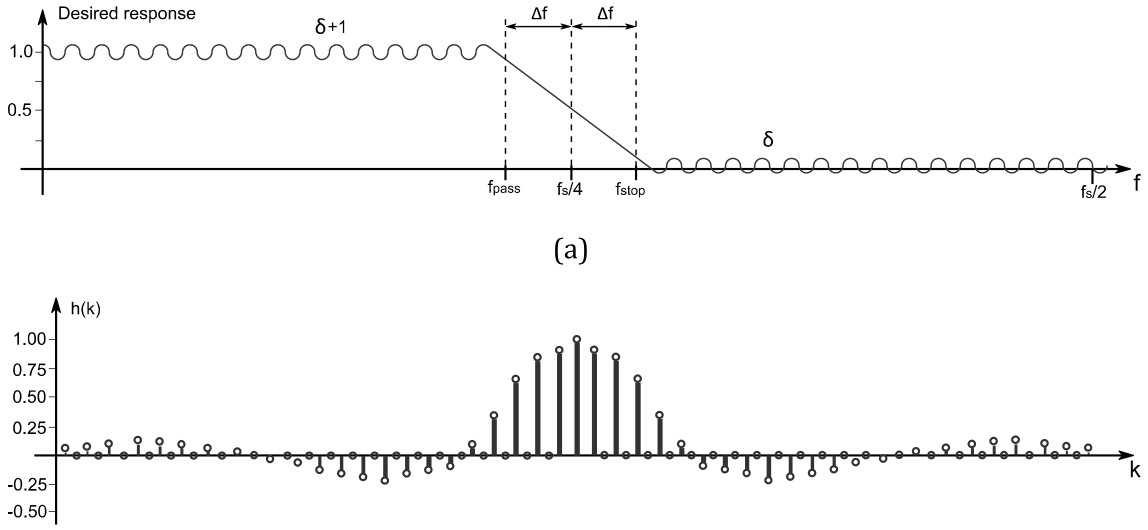
Figure 2.27: Generic hardware structure of the FIR polyphase filter.

It is very difficult to make assumptions of what will the MAC and FSM blocks have inside, so it's complicated to see what hardware or area saving improvements can be done in these blocks. This improvements will be dealt later in the paper. However, it is obvious that the less size of the ROM and RAM memory the more hardware and especially the more area will be saved. Both memory sizes will be reduced as much as possible in order to achieve this goal. The size of the RAM memory will be decided in the design phase according to the n number of taps of the filter, which will be the least possible that accomplishes the filter specifications. The size of the ROM memory depends on the number of coefficients, which at first glance will be double as the number of taps.

With the purpose of reducing the number of coefficients needed to store in the ROM memory some new approaches will be given to the FIR interpolation filter.

2.9 Halfband filters

A half band filter is a type of filter that has the special property of being symmetric from the fs/4 point. This means that the transition band is centered in fs/4, and that the ripples in the pass-band and the stop-band are the same [1]. The transfer function of a half band filter looks like Figure 2.28(a). The reason why this property is interesting for the design of this paper is that approximately half of the coefficients of a half band filter are zero, which basically means that half of the multiplications don't need to be done [1]. Specifically, it can be said that a half band filter alternates nonzero and zero coefficients except for the middle pick coefficient, which is situated in between two non zero coefficients and its value is the highest among all the coefficients Figure 2.28(b). This structure is symmetric from each side of the middle peak coefficient, which leads to the fact that this kind of filters need to have an odd number of coefficients. What's more, a half band filter with n coefficients, being n+1 multiple of four, will be the same as a half band filter with n+2 coefficients. The reason is that these two added samples will be two zeros, one at the beginning and one at the end to preserve symmetry, which will not change the output generated by then filter. As a general statement, it can be said that a half band filter with symmetric coefficients (which, as it was stated before, means linear phase) needs to have n number of coefficients being n+1 multiple of four.



(a)

(b)

Figure 2.28: (a) Desired frequency response; and (b) Coefficients; of the half band filter.

A part from saving the time and current consumption of the multiplications, all the zero coefficients of the filter will not need to be stored in the ROM memory, and assuming that the coefficients are normalized to the middle peak coefficient, which will be reasoned later, this one will also not need to be saved since it will be one. Summing up, using a half band filter in the design implies a significant reduction of the ROM memory mostly halving its size.

This half band approach will be used in the polyphase structure mentioned before. Looking at the operation of the resulting structure combined Figure 2.29, it is clear that in the first step (switches up), n multiplications need to be done, while in the second step (switches down), only one multiplication need to be done. Precisely, this multiplication involves the middle peak coefficient, which normalized is one, and an input sample which leads to an output equal to the input sample itself. The first state works like a simple m-tap filter and the second step directly passes one of the input samples to the output. So basically, the two output samples per each input will be calculated differently, and its expressions are found in Eq.(2.12) and Eq.(2.13).

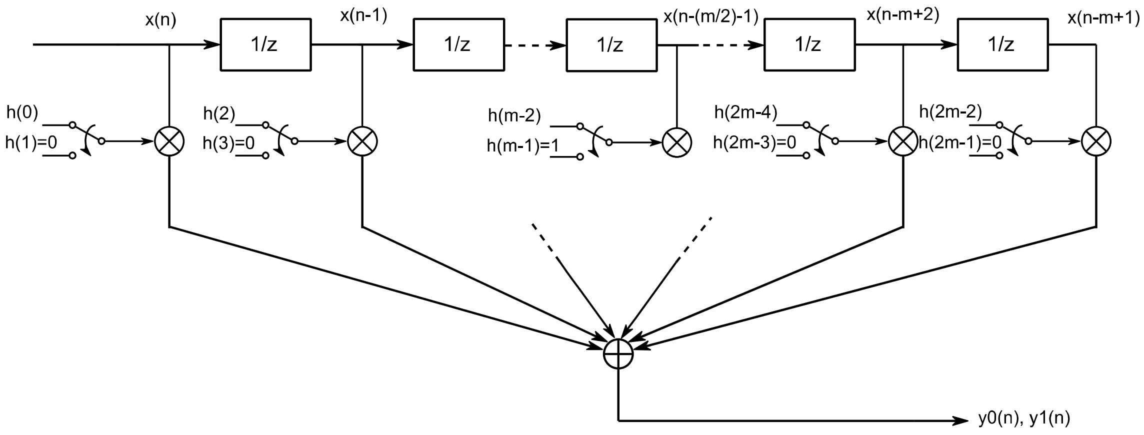


Figure 2.29: Structure of the halfband polyphase filter with the zero coefficients.

$$y_0(n) = x(n) \cdot h(0) + x(n-1) \cdot h(2) + \cdots + x(n-m+1) \cdot h(2m-2) \qquad (2.12)$$

$$y_1(n) = x(n-(m/2)-1) \qquad (2.13)$$

There is even a further significant reduction of the size of the ROM memory. As it has been said, the coefficients of the half band filter are symmetrical from the middle peak coefficient, which means that only half of them should be stored in the ROM memory, because the other half is the same.

After all the different approaches used in the basic FIR filter in order to reduce the hardware consumption and the size of the area, the characteristics of the filter designed will be:

- FIR filter.
- Polyphase filter interpolated by m=2.
- Half band filters.

Assuming n taps, since the filter is interpolating by two in a form of a polyphase filter, 2n coefficients are expected. Half minus one of them are zero because of being half band, and one of the will be 1 since all the coefficients will be normalized to the middle peak (the highest one). This leaves n symmetric coefficients, which means that just n/2 to save in the ROM memory.

- n taps
- 2n coefficients
- n-1 zero coefficients
- 1 coefficient equal to 1
- n symmetrical nonzero coefficients
- n/2 coefficients stored in the ROM memory

The final structure of the filter that will be designed is shown in Figure 2.30.

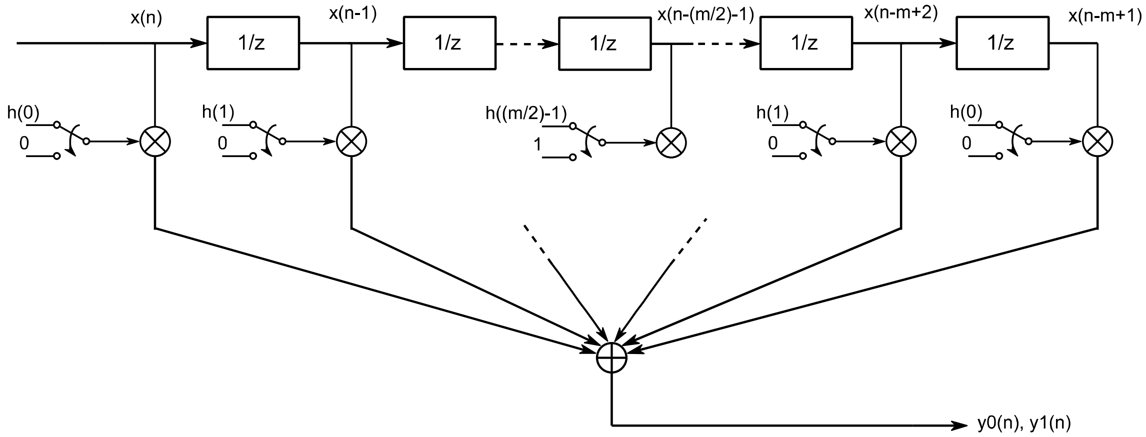


Figure 2.30: Structure with all the approaches used of the FIR filter to design.

Chapter 3

3. Design of the filter

This chapter will deal with the design of the first step of the interpolation filter using the approaches explained in the previous chapter and other further improvements for the structure proposed.

3.1 Specifications

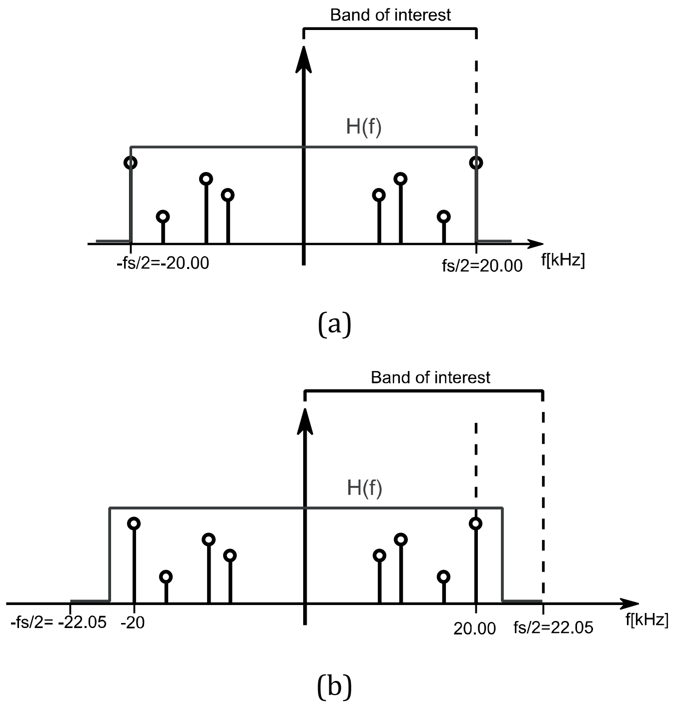
Once the structure of the filter has been set the design process starts. The first step of the process will be to establish the specifications of the filter.

- Input sample frequency: 22,05 kHz
- Output sample frequency: 44,1 kHz
- Input bit precision: 1 integer bit, 15 fractional bits
- Ripple of the pass-band and stop-band: 0,1dB
- Cut off frequencies: 10kHz and 12,05kHz
- -60dB attenuation in the cut off band pass frequency.
- Preserve a signal to noise ratio (SNR=Power of the Signal/Power of the Noise) around 98dB.

The ripples will be the same in the pass band and stop band, because as it was stated before the filter will be designed as a half band filter. The cut off frequencies are equally distant from the fs/4 for the same reason.

In hifi applications the bandwidth is 20 kHz, so in order to avoid aliasing the sample frequency must be at least 40 kHz (Nyquist criteria). The problem with exactly 40kHz is that if there is a component of exactly 20kHz in the input, an infinite sharp filter will be needed to get rid of it Figure 3.1(a). For this reason a sample frequency of 44,1kHz is normally used, to give some space to the filter to attenuate Figure 3.1(b). Since the hearing aids are aimed to hearing-impaired persons, the bandwidth considered can be lowered. Even though a 4 kHz bandwidth would be enough to ensure the speech understandability a bandwidth of 10 kHz is normally used so that also the environmental sounds can be also heard properly. Since the bandwidth is halved from the original 20 kHz, a sample frequency of 44,1 kHz/2=22,05kHz will be used as an input sample frequency. Since the filter has to double the frequency at the output, it is clear that the output sample frequency will need to be 2·22,05 kHz = 44,1kHz.

The bit precision in the input of the filter comes from the output of the DSP (digital signal processing) stage of the hearing aid, and it is assumed to be 16, since it is an standard quality used for example in the CD-ROMs.



(a)

(b)

Figure 3.1: Frequency spectrum; (a) fs=40 kHz, infinite sharp filter; (b) fs=44,1kHz.

A pass-band ripple of 1 dB is audible, and a pass-band ripple of 0.1 dB is the one used in hifi applications, so an average pass-band ripple of 0.5 dB will be used in the whole interpolation filter chain. From this whole ripple 0.1 dB will be allocated in the first interpolation filter leaving the 0.4dB ripple for the three following stages of the filter.

The cutoff frequency of the pass-band must be 10 kHz, since it is the hearable bandwidth that is going to be used. Since the cutoff frequencies must be symmetrical around fs/4=11,025 kHz, the stopband cutoff frequency must be (11,025 kHz-10 kHz) + 11,025 kHz = 12,05 kHz.

Finally, as it was explained before, -60 dB is the attenuation needed to suppress the images below the hearing threshold of a normal hearing person, for a speech signal of normal intensity Figure 1.7(a). Again, since the hearing aids are tested with normal hearing persons, the hearing threshold of normal hearing persons is used instead of the hearing threshold of hearing-impaired persons.

Finally, a SQNR of 98 dB must be maintained, since this is the SQNR of the input, so the filter should not increase it, which is the equivalent of saying that the filter should not lower the quality of the input audio signal.

The noise that in the input of the filter comes from quantization, Figure 3.2, this is why we talk about SQNR and not about directly SNR. The possible noise that the filter can add to the system is also from quantization, but the noise that the class-D output stage can add is from distortion Figure 3.3. It is unknown the amount of distortion that this class-D output stage can generate, but the overall SNDR (signal to noise and distortion rate) at the output of the back end must be 90dB by specification. In order to leave the widest margin possible of noise addition in the class-D output, the 98dB in the SQNR will be maintained at the end of the filter so that the full 98-90= 8dB can be left as a possible distortion noise addition.

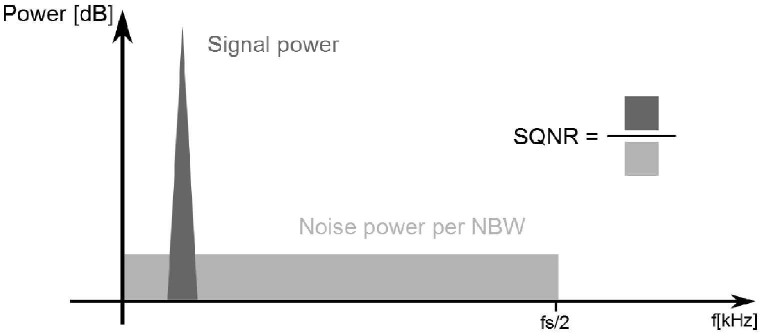


Figure 3.2: Input signal power and quantization noise in the input.

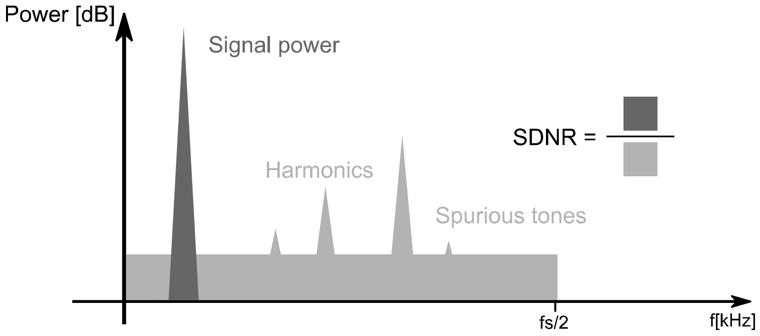


Figure 3.3: Input signal power, quantization noise, and distortion noise.

3.2 Design process with Matlab

The design process will consist of various steps, being the first one to obtain the coefficients of the filter using Maltab. Once the coefficients are obtained, a floating point model of the filter will be designed in Matlab. A sinusoidal signal will be used as an input of the model in order to test its functionality. After the floating point model, a fixed point model will be also designed in Matlab. The same input will be used to test this new model, but the sinusoidal wave will not have floating point precision now, but it will be quantized to 16 bits (as the specifications of the input requires). With both models working, the input 16 bit precision and the fixed point filter coefficients will be exported from Matlab to the VHDL design. The filter will be designed in VHDL, and tested in simulation using the Matlab input. In order to make sure that the filter designed in VHDL works properly the output samples generated by it will be exported back to Matlab and compared to the output samples obtained with the fixed point model. If the comparison is positive, it will be assumed that the VHDL design is correct.

3.2.1 Filter coefficients obtention

The digital signal processing (DSP) system toolbox will be used in Matlab, which will help to find the coefficients of the filter needed, concretely, the function "firhalfband". The inputs of the "firhalfband" function are the filter order and the cut of frequency of the pass band normalized to half of the sample frequency. Using that normalization the sample frequency does not need to be specified. With these two parameters Matlab calculates the coefficients of the halfband FIR filter. It's worth to keep in mind that the coefficients obtained are all the coefficients of the half band filter, so not all of them will need to be saved in the ROM memory. Obviously, just one of the cutoff frequencies need to be designed since the other one is symmetric to 0,5 (fs/4 normalized to fs/2); in this case is the pass-band cutoff frequency which by definition of the halfband filter has to be lower than 0,5. Also, the filter order must be an even integer since the order is always the number of coefficients minus one [1], and as it was explained before, the number of coefficients of a halfband filter must be an odd number. The expression used is the one in Eq.(3.1) where fir_b are the floating point precision coefficients, fir_order is the filter's order, fir_cutoff_f(1) is the pass-band cutoff frequency and fir_fs is the sample frequency.

$$fir\_b = firhalfband(fir\_order, fir\_cutoff\_f(1)/(fir\_fs/2)); \tag{3.1}$$

As it was stated before the coefficients that will be stored will be normalized to the higher one, the middle peak coefficient. The expression in Eq.(3.2) was used to achieve this purpose, where fir_b_s are the normalized floating point precision coefficients that will be used in the floating point model.

$$fir\_b\_s = fir\_b/max(fir\_b); \tag{3.2}$$

There are two reasons why this normalization is done. The first reason is that this way the highest coefficient will be 1, which makes the multiplication of the input samples by this coefficient not needed. Therefore, the coefficient does not need to be saved in the ROM memory saving area in the chip. The second and the most important reason is that using normalization the overall precision will increase. Since the range of the values of the coefficients is very wide, the first bit will be used to express the highest coefficient. This will leave all the other bits to express all the remaining coefficients. All the operations will be done with the normalized coefficients, and when the result is obtained it will be multiplied again by the highest coefficient.

Afterwards the floating point precision coefficient will need to be quantized in order to obtain the 18 bit precision needed for the design (1 integer bit and 17 fractional bits). This number of bits is chosen since it's the maxim number of bits that the multiplier inside the

FPGA can operate. It has been found to be more than enough in order not to destroy the signal with too much noise. For now 18 bits quantization will be used, and will be discussed later in the paper.

The quantization is carried out by the piece of code in Eq.(3.3), where fir_b_q_fixed are the fixed point normalized coefficients that will be used in the fixed point model, T is a numeric type defined as a total number of 18 bits where 1 will be integer and 17 fractional and F a variable that defines the way in which the quantization will be done.

$$F \ = \ fimath('RoundMode', 'nearest', 'OverflowMode', 'saturate');$$

$$T = numerictype(1,18,17);$$

$$fir\_b\_q\_fixed = fi(fir\_b\_s, T, F); \qquad (3.3)$$

Now the discussion will be how to find the order of the filter that meets the specifications. The procedure that will be used will be to assign a value to the order, obtain the frequency response, and then check the attenuation of this response in the normalized stopband cutoff frequency (12050Hz/(44100Hz/2)=0,2732.

The initial order was set to 100 in the previous design, where the gain in the 0.2732 frequency is -77.34 dB as it is shown in Figure 3.4. The red line corresponds to the floating point precision transfer function, and the blue line corresponds to the 18bits quantized transfer function.

The suppression is too much so the order will be halved to 50. In the frequency response, the attenuation obtained with this new order in the stopband frequency is -45.6 dB Figure 3.5, which is not enough for our specifications, so the order must be increased.

The next order tried is 74. Notice that as it was explained before, order 74 is equivalent to order 76, since the only difference is adding 2 zeros at each extreme of the sequence of time domain coefficients. The result in this case is an attenuation of -61,84dB Figure 3.6. This attenuation is close to the desired one, but one more iteration of the process will be done in order to see if the order can be reduced even more.

The order, now, will be set to the order directly below 74. Since the order must be even, and order 72 exactly corresponds with order 70 (for the same reason as 76 and 74), the order will be set to 70 obtaining a gain of -59dB Figure 3.7. This attenuation does not preserve our specifications so it cannot be used.

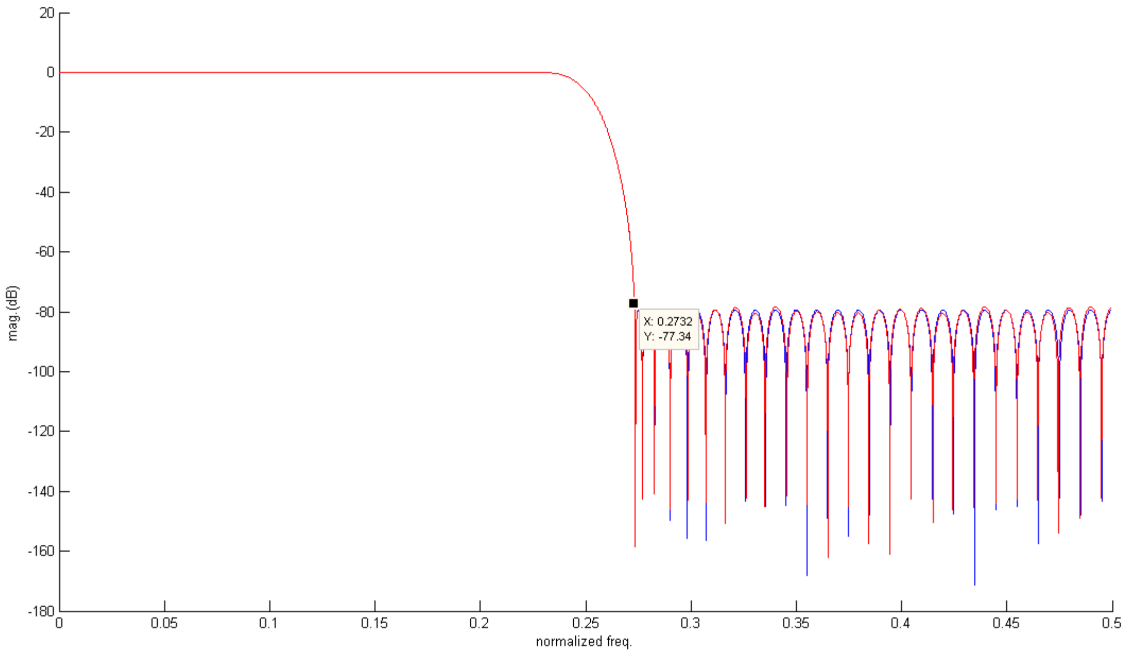


Figure 3.4: Frequency response of the filter with order=100.

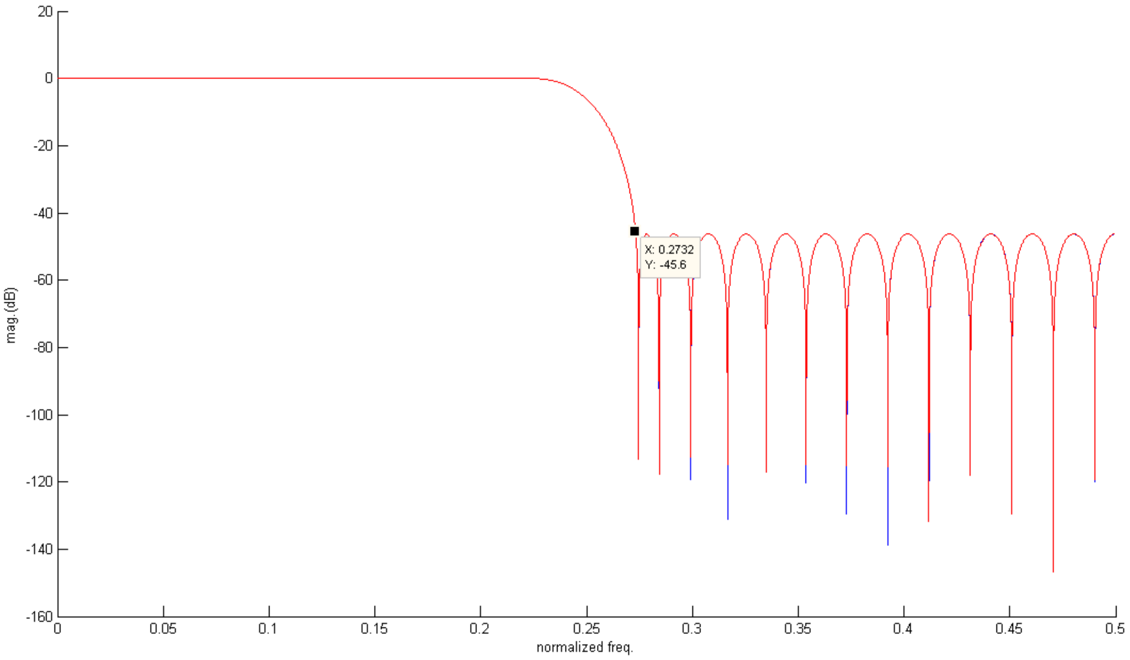


Figure 3.5: Frequency response of the filter with order=50.

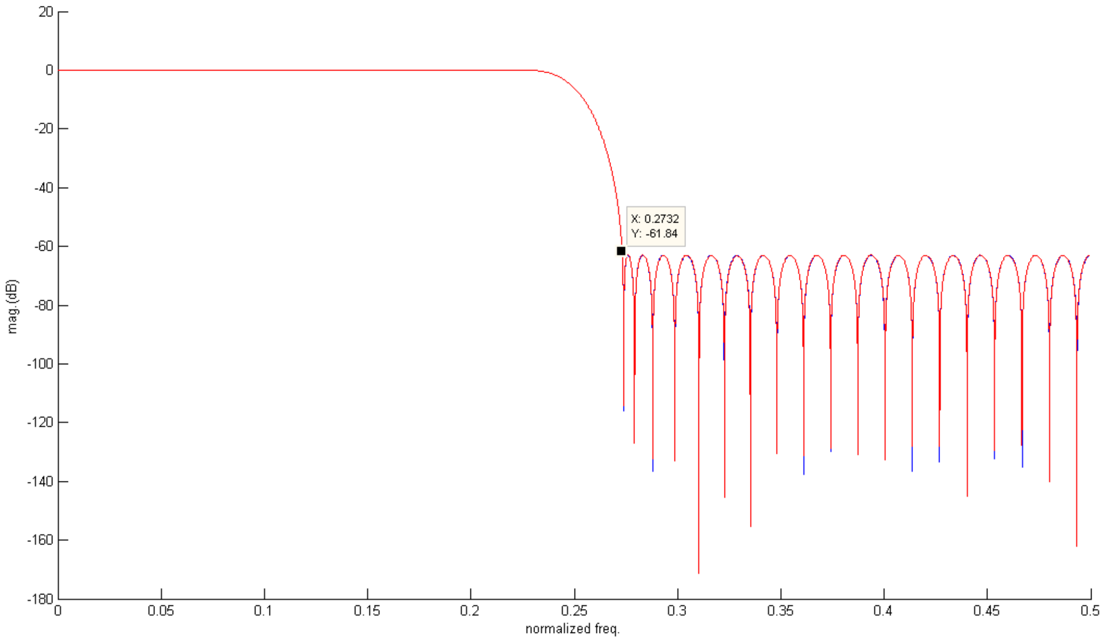


Figure 3.6: Frequency response of the filter with order=74.

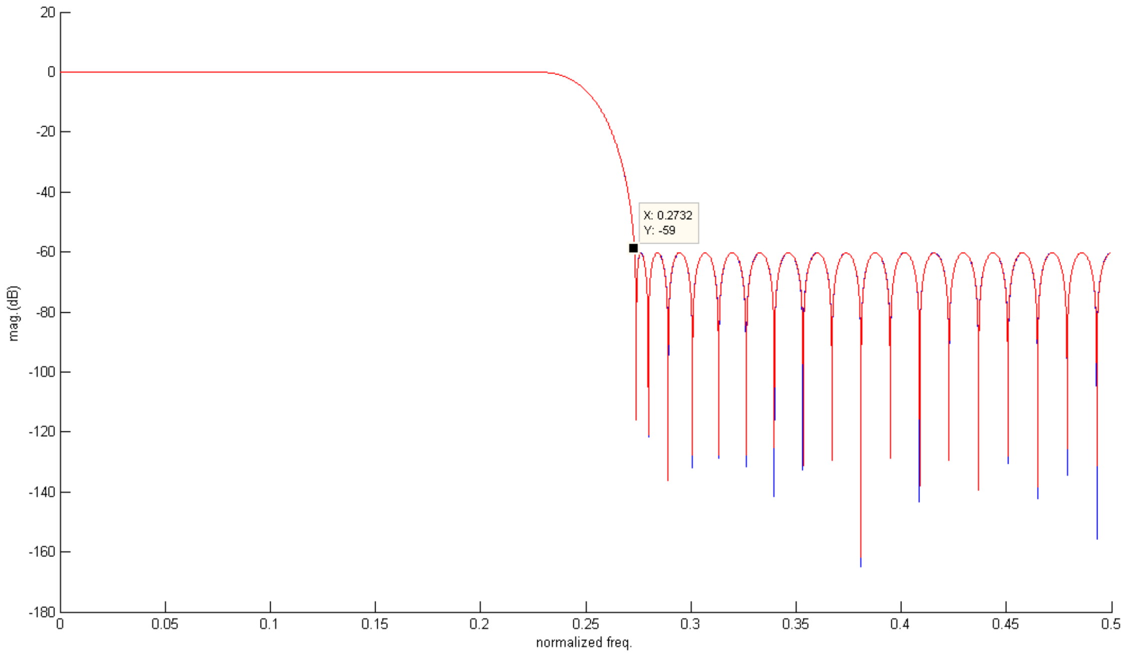


Figure 3.7: Frequency response of the filter with order=70.

It can be concluded that the minimum order that satisfies the specifications of the FIR halfband filter is 74, so it is the one that will be used in the design in order to save as much hardware as possible. Now that the order has been found, the number of bits of the quantization should be discussed. Starting with the plot in the Figure 3.6, the transfer function of the filter with floating point precision (red) and the filter quantized by 18 bits (blue) is shown. It can be seen that the transfer function it's mostly exact, and the noise added with the quantization is below the noise in the input, so it doesn't add noise in the overall response. If the number of bits is reduced, the amount of noise added will grow until a point in which the overall noise will be increased significantly destroying the original transfer function. In Figure 3.8 the quantization by 16 bits can be observed. The ideal transfer function and the quantized one are similar, but much more different than with 18 bits. Finally, if the quantization is done with 14 bits, the transfer function becomes clearly different and the overall noise becomes higher than the original one, lowering the attenuation in the stopband Figure 3.9. Therefore the quantization will be to 18 bits for the filter coefficients, since it is the quantization in which the transfer function is mostly exact as the floated point precision one.

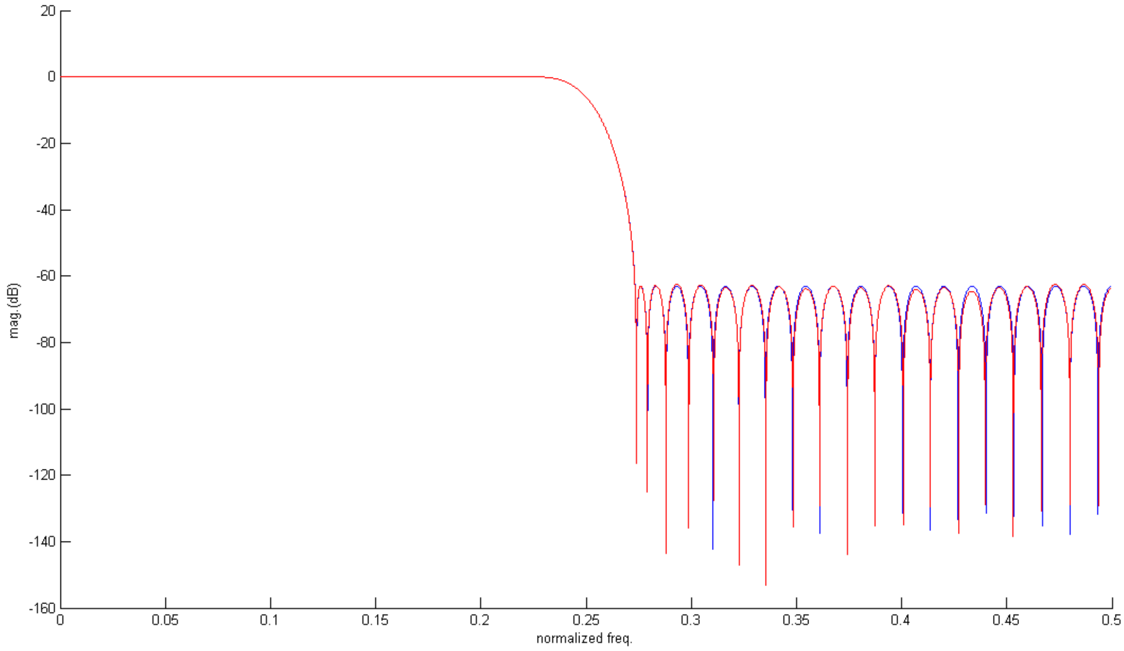


Figure 3.8: Transfer function with and without quantization, 16 bits.

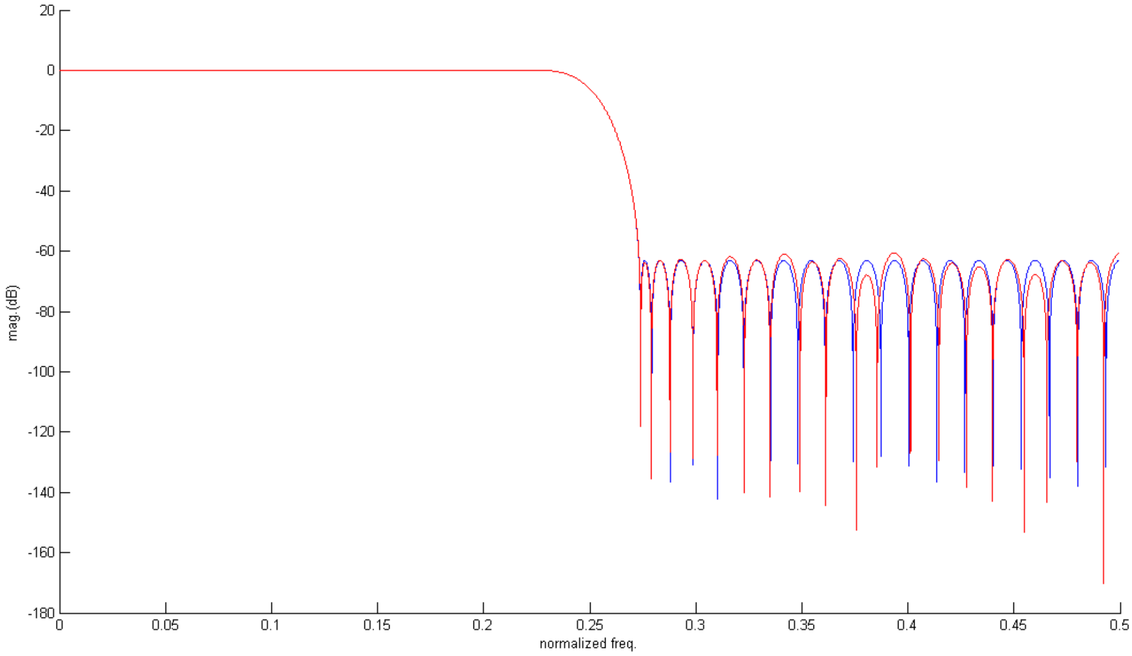


Figure 3.9: Transfer function with and without quantization, 14 bits.

3.2.2 Input signal generation

Now that the coefficients of the filter are decided, the next step will be to create models of the filter. In order to test these models, an input for the filter should be generated. A sinusoidal wave of 43,0664Hz frequency with an attenuation of the amplitude of -0,2dB is chosen to be used. The frequency is chosen randomly just making sure that it is an exact frequency of DFT's output bins so that leakage is avoided. The sinusoidal wave is not full scale, because otherwise during the whole process the signal can saturate, which means that a number greater than the highest that can be expressed with a certain number of bits is obtained. The saturation is caused by the fact that the numbers are expressed in two's complement. With n bits, the range of the standard bit notation is from 0 to $2^n - 1$ but with two's complement this range is from -2^{n-1} to $2^{n-1} - 1$. For n=3 the values that can be expressed can be seen in Table 3.1. As it can be seen not the same range of numbers can be expressed in positive than in negative, which means that if the sinusoidal wave has an amplitude of 4, the signal will saturate when the highest positive point is reached, causing distortion to the signal.

Binary number	**Number that it represents**
000	0
001	1
010	2
011	3
100	-4
101	-3
110	-2
111	-1

Table 3.1: Numbers that can be represented in two's complement with 3 bits.

In the case of this design, the binary numbers that expresses the input has 1 integer bit and 15 fractional bits. That means that the maximum positive number that can be expressed will be 0111111111111111= 0.999969482421875 and the maximum negative number achievable will be 1000000000000000=-1. If the sinusoidal wave is full scale, from -1 to 1, when the signal reaches the positive peak it will saturate Figure 3.10. The -0,2dB attenuation will prevent the signal from reaching +1 so that it doesn't saturate.

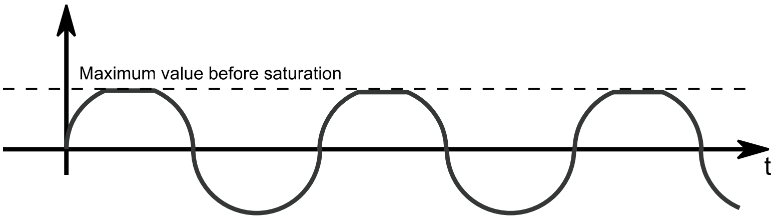


Figure 3.10: Sinusoidal wave saturating.

3.2.3 Floating point model

A floating point model of the FIR filter will be designed in Matlab Simulink. The polyphase structure proposed before will be used for the design of the model. A part of the design can be seen in Figure 3.11. A little change in the structure proposed has been done while modeling the filter; instead of switching between the coefficient that is being multiplied, both of them are multiplied by the input sample, and then a switch chooses which of these two results is passed to the output to be added all together with the other results. This change is just done in order to simplify the model design in Matlab, but the filter will be implemented with the original structure. This change does not affect the result or the principle of the operation of the filter, it is completely equivalent. The switches also work with a frequency double as the input sample frequency.

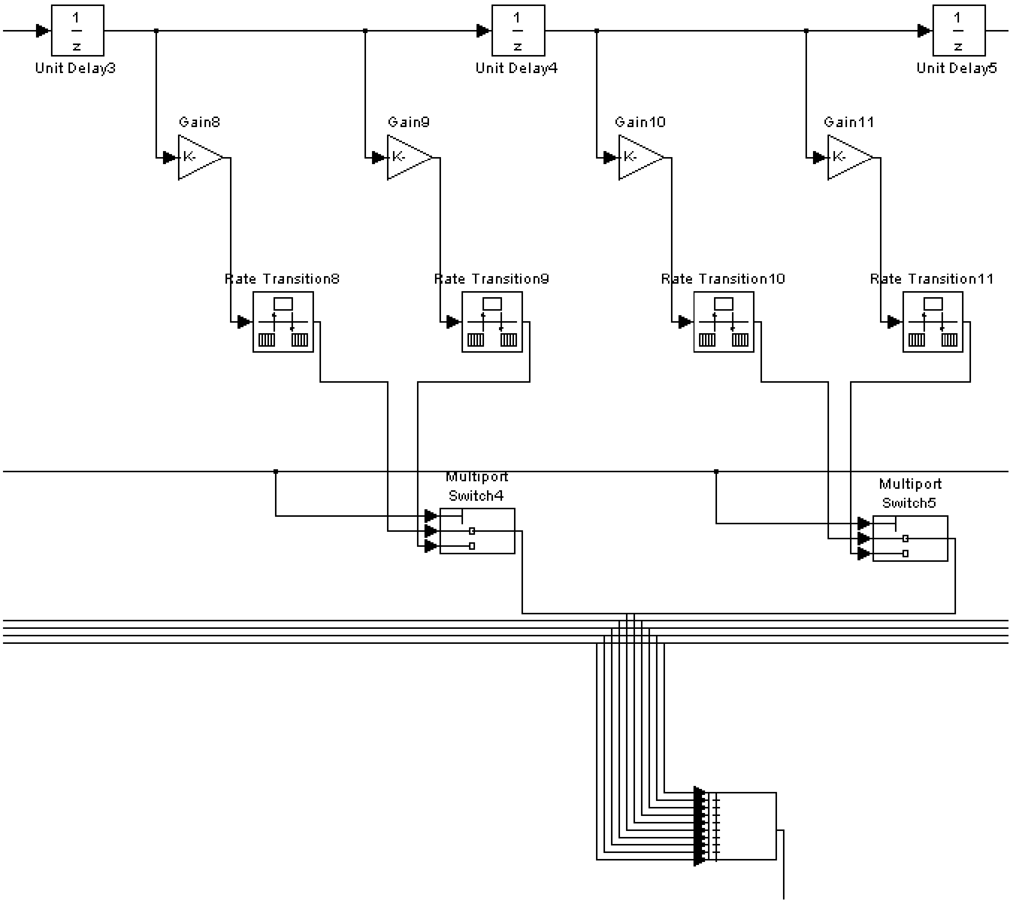


Figure 3.11: Floating point model.

In Figure 3.12 the equivalence between blocks and its function can be seen. The 1/z blocks are the delay blocks that create the delay line. The gain blocks contain the filter coefficients, and multiply their inputs by their own coefficient. The rate transition blocks double the frequency so that the switching blocks can pick both multiplications per tap in one input period. The switching blocks select which of the two multiplications per tap are passed to the output every half input period. Finally the adder block is the block that adds all the multiplications together to obtain the output sample. The output of the filter obtained with the floating point model with floating point precision can be seen in Figure 3.13. As it can be seen the filter attenuates the 60dB correctly in the stop-band. It is also worth to notice that the peak power in the input frequency is repeated every input frequency because it is a discrete signal. While the first peak is not suppressed, the first repetition (located in fs_{in}) is suppressed down to -60dB accomplishing this way the specifications. It is worth to notice that the second repetition (located at $2 \cdot fs_{in}$) is not attenuated; the following stage of the interpolation filters will suppress that image. Observing the noise it can be noticed that in the stop band the noise is also attenuated.

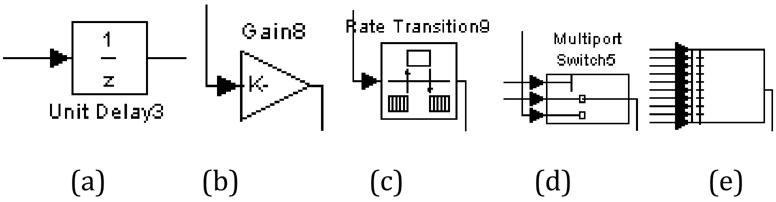


Figure 3.12: (a) 1/z block; (b) Gain block; (c) Rate transition block; (d) Switching block; (e) Adder block.

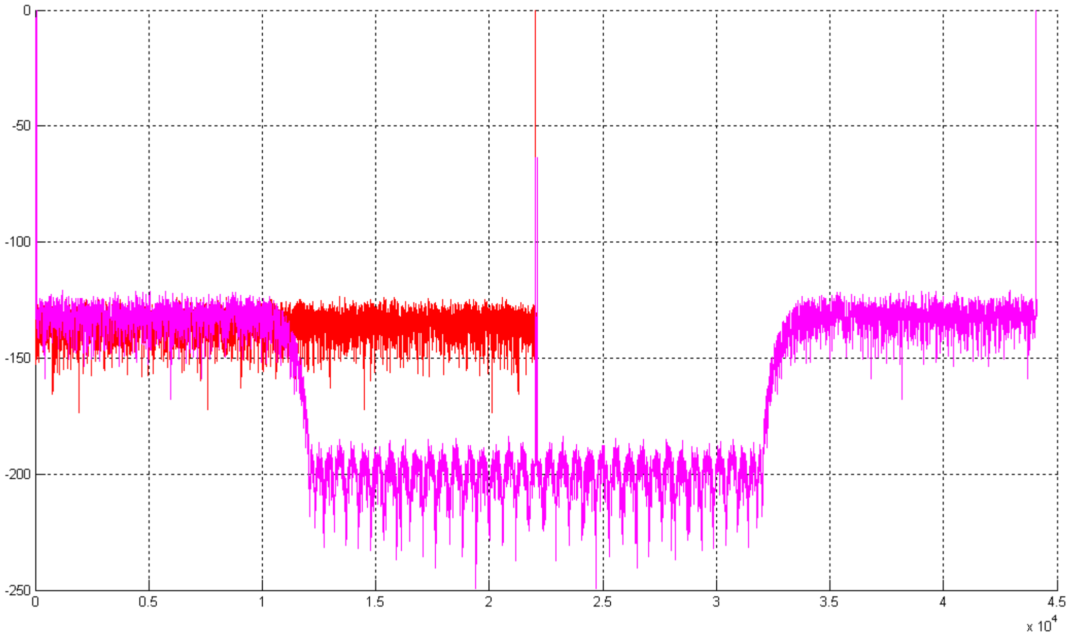


Figure 3.13: Output of the floating point model. Input(red) Output(magenta).

3.2.4 Fixed point model

Once the floating point model works, a fixed point model will be designed using the same approach. Now, the quantized to 18 bits coefficients will be used, and also the quantized input signal. A sample part of the model can be seen in Figure 3.14. This model works in the same way as the floating point model but with the coefficients, input, and operations quantized.

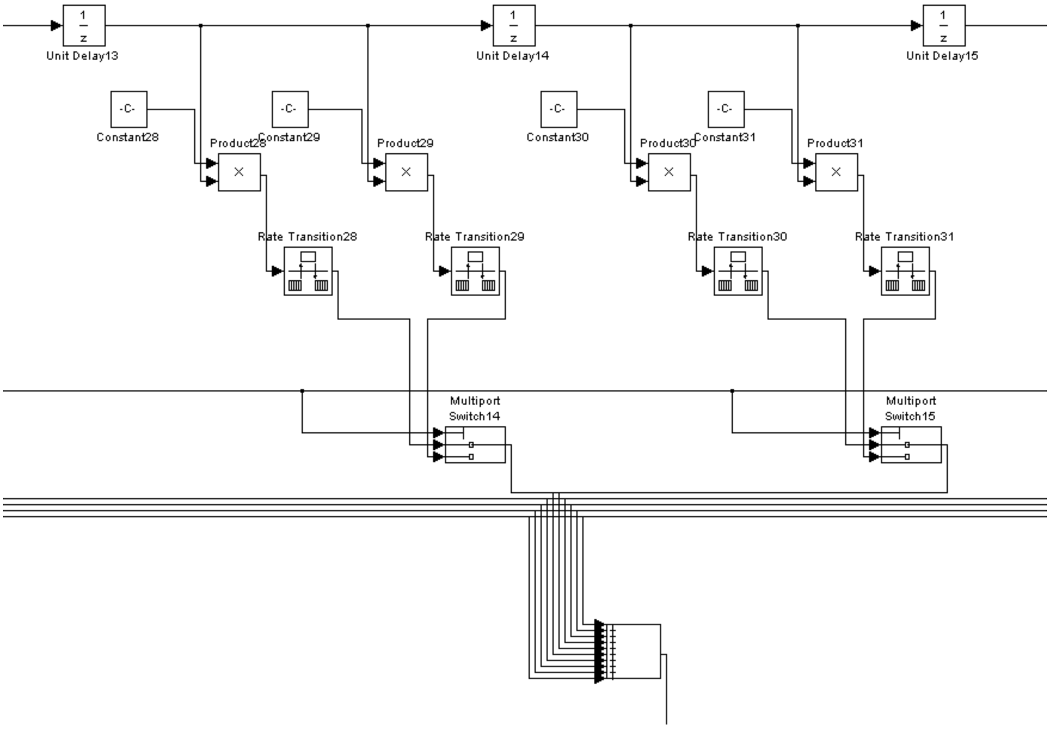


Figure 3.14: Fixed point model.

In Figure 3.15 the equivalence between blocks and its function can be seen. The 1/z blocks are the delay blocks that create the delay line. The constant blocks are the ones where the filter coefficients are stored. The product blocks are the ones that multiply the input samples by each filter coefficient. The rate transition blocks double the frequency so that the switching blocks can pick both multiplications per tap in one input period. The switching blocks select which of the two multiplications per tap are passed to the output every half input period. Finally the adder block is the block that sums all the multiplications together to obtain the output sample.

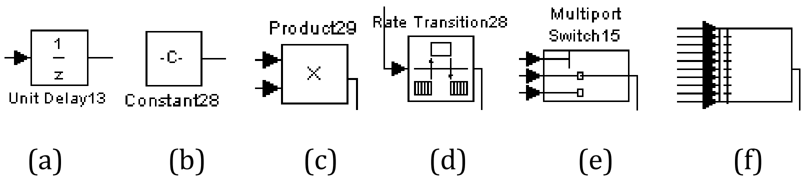


Figure 3.15: (a) 1/z block; (b) Constant block; (c) Product block; (d) Rate transition block; (e) Switching block; (f) Adder block.

The input samples are 16 bits, 1 integer and 15 fractional, and the coefficients are quantized at 18 bits, 1 integer and 17 fractional. After the multiplication, if the full precision wants to be achieved 2 integer bits and 32 fractional bits should be used. In order to reduce the number of bits of the output a quantization will be applied. This quantization will add quantization noise to the output signal so the number of bits should be reduced as much as possible without compromising the 98dB of the SQNR at the output of the filter. In Figure 3.16 the fixed point full precision output can be seen. The level of the noise from the pass band is around -125dB and the noise in the stop band is around -190dB. The approach that was used in the previous design of this filter was to start from this fixed point full precision and then keep quantizing bits step by step until the minimum number of bits that satisfy the specifications is reached. The minimum number of bits found was 2 integer bits and 20 fractional bits, and this quantization is the one that will be used in this design. In Figure 3.17 the output with this quantization can be seen.

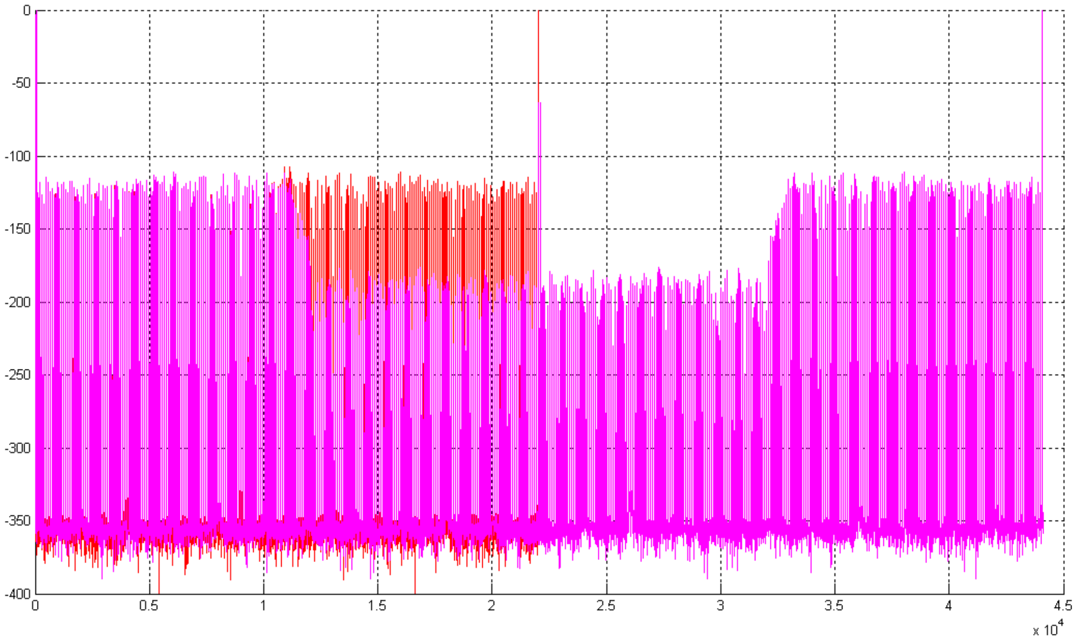


Figure 3.16: Output with fixed point full precision, 2 integer bits and 32 fractional bits. Input(red) Output(magenta).

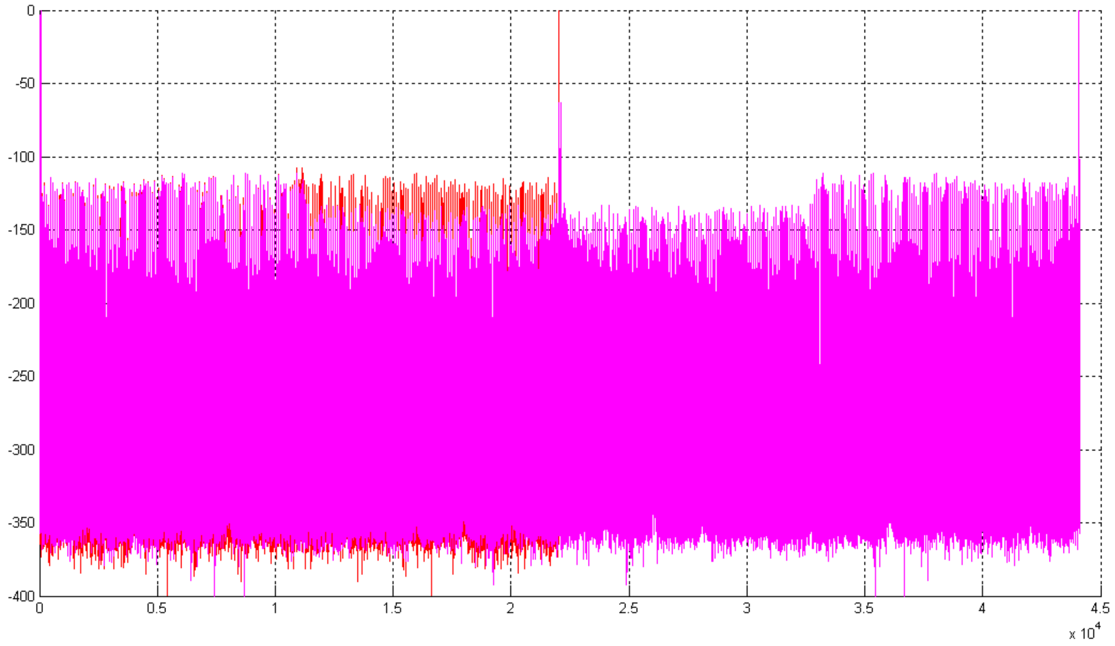


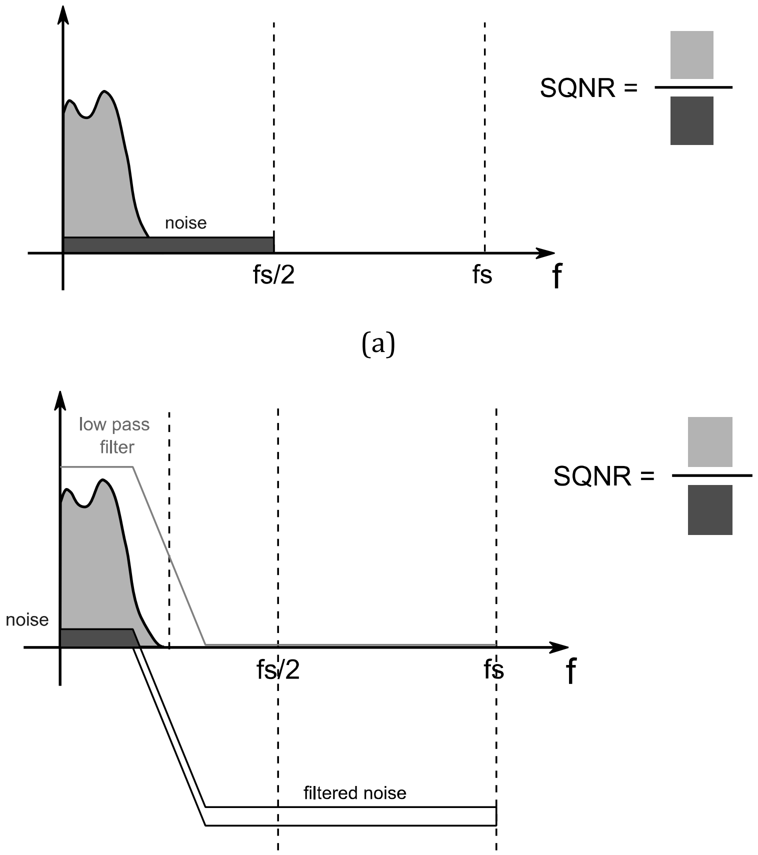
Figure 3.17: Output quantized to 2 integer bits and 20 fractional bits. Input(red) Output(magenta).

The SQNR of the input and output of with this quantization are:

- SQNR(input)=97.3725 dB
- SQNR(output)=97.8933 dB

As it can be seen, the output SQNR is not exactly 98dB, but close to it. This little difference comes from the fact that, as it was explained before, the input signal is not full-scale but it was attenuated 0.2dB. Adding this 0.2dB into the 97.8933dB the result of 98.0933dB is obtained which is a little more than 98dB.

It is also worth to notice that the input SQNR is lower than the output SQNR. The reason for this can be seen in Figure 3.18. The input SQNR is calculated with the signal power and the quantization noise contained in the input band of interest which stops at fs/2. The output SQNR is calculated with the filtered signal power and the filtered quantization noise contained in the output band of interest which stops at fs. As it can be seen in Figure 3.19 the filtered output noise is a lower than the input unfiltered input noise because of the attenuation that the filter applies to the noise just after the passband frequency.

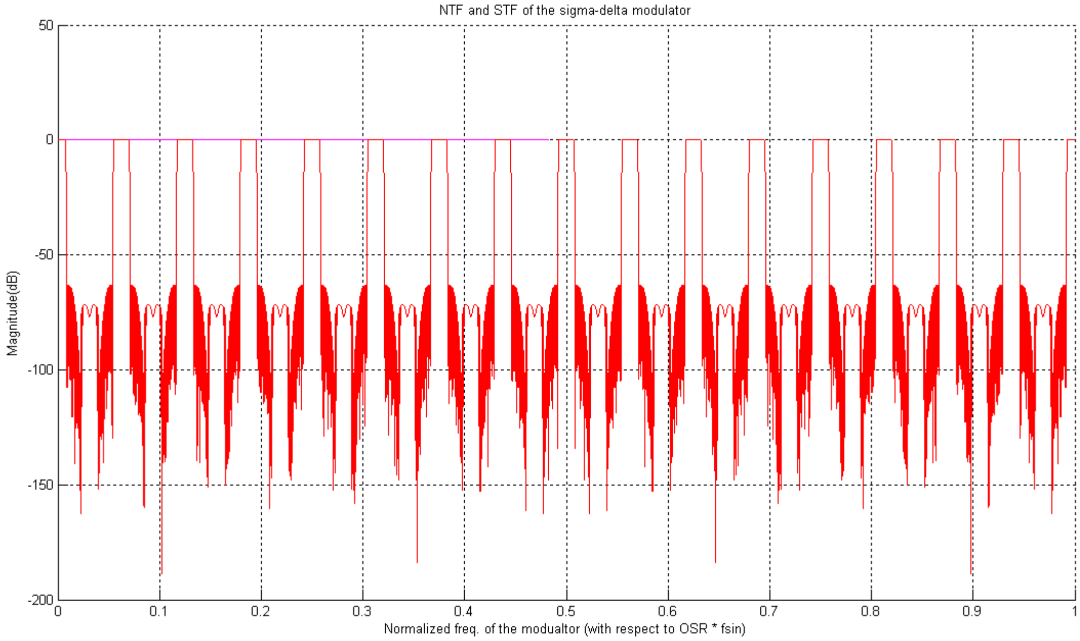


(a)

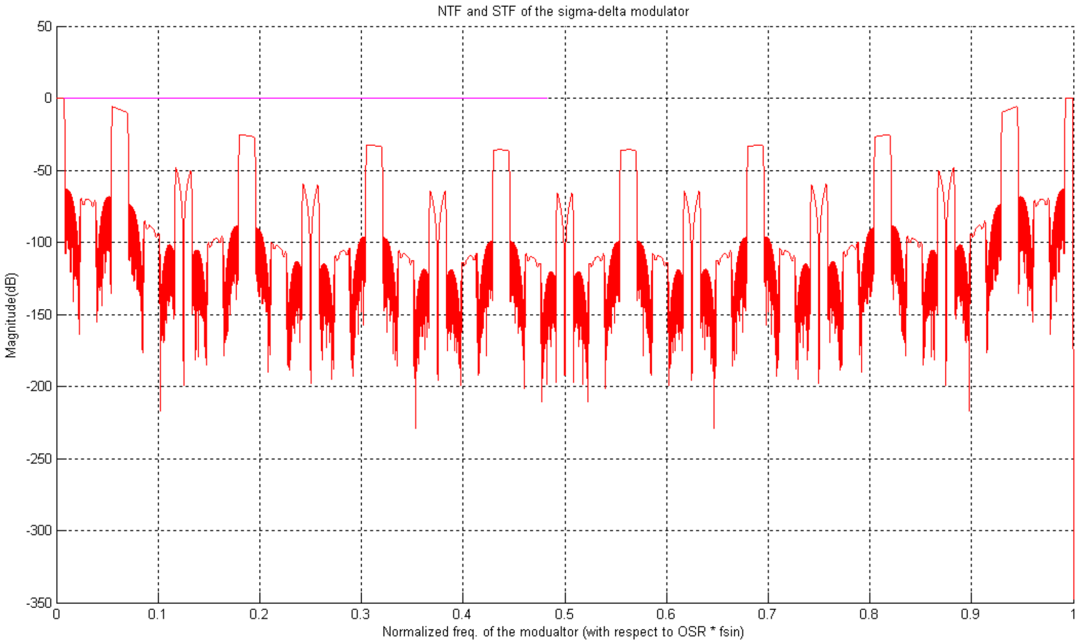
(b)

Figure 3.18: SQNR of the: (a) input; (b) output.

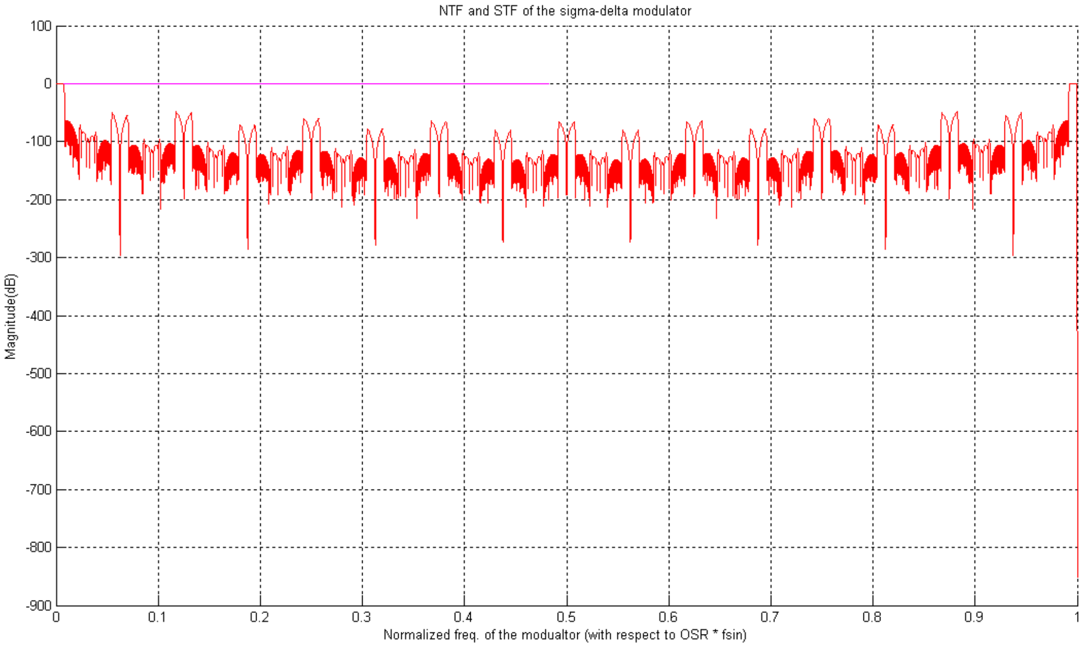
Once the Matlab model of the filter is finished, the combined transfer function of all the stages of the FIR filter can be seen in Figure 3.19.



(a)



(b)



(c)

Figure 3.19: Combined transfer functions of the stages of the interpolation filter: (a) First and second stage; (b) First second and third stages; (c) First, second, third and fourth stages.

3.3 Design process in VHDL

After the Matlab models, the filter will be designed and implemented in the FPGA. For the prototype of the filter the FPGA Xilinx Spartan 6 was used. The knowledge that needs to be acquired in order to design the filter in VHDL can be found in [2] and [4]. The filter will be designed in VHDL code and the simulation will be tested with the same input generated with Matlab. The output obtained with the simulation will be saved and compared with the output obtained in Matlab to make sure that the VHDL design works properly Figure 3.20. After the simulation works, the filter will be implemented and tested using an audio analyzer. The audio analyzer that will be used is the “Audio Precision Cascade SYS-2522” and it’s capable of generating and analyzing signals. In order to connect the Audio Precision with the FPGA Xilinx Spartan 6 board an I2S interface will be needed to synchronize both machines together. The I2S that will be used in this project will be the “Programmable Serial Interface Adapter PSIA-2722 (Audio Precision)”. The I2S is a standardized interface to connect audio devices together that transfers the bits between devices in serial form. The filter designed receives parallel 16 bits samples, and the output are parallel 22 bits samples, which means that a serial to parallel and parallel to serial interfaces will need to be designed in order to test the filter with the Audio Precision. An I2S s/p and p/s interface will be designed to connect together the FPGA Xilinx Spartan 6 with the prototype implemented and the Audio Precision Figure 3.21.

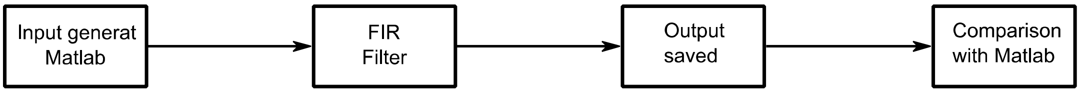


Figure 3.20: Simulation.

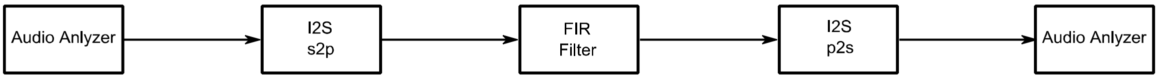


Figure 3.21: Implementation.

The I2S bus that connects the two devices consists at least of a bit clock line, a left-right clock line and a multiplexed data line. The bit clock line is used achieve the most exact synchronization possible between the two machines and the left-right clock will have the frequency of the input signal, which means that every cycle of the left-right clock a new input sample will be generated.

First of all, the system frequency must be selected. The frequency for the clock that will be used in the system, sys_clk, is chosen to be the highest of the whole system, the DPWM block. The quantizer in sigma delta modulator has 3 bits, so the operation frequency must be:

$$f_{\ system} = (8-2)\cdot OSR \cdot f_{sin} = 6\cdot 64\cdot 23.4\, kHz = 8.9\, MHz \tag{3.4}$$

The OSR value is the total oversampling rate of the filter, which is 64 (divided in four steps). The 3 bit quantizer, that can generate 8 possible values (-4, -3, -2, -1, 0, 1, 2, 3), but since the signal must be symmetrical from zero just the values from -3 to 3 are used, and since the zero does not add any clock cycles to turn the signal into PWM just -3,-2,-1,1,2,3 values are used, reducing this way the number of possible values from 8 to 6, which is the factor used in Eq.(3.4).

Once this frequency is set, the bit_clk is defined 6 times faster as the sys_clk, and the lr_clk is defined to be 64 times faster than the bit_clk. The lr_clk will correspond to the frequency in which the new input samples arrive to the delay line. Overall, 64·6=384 sys_clk periods will be contained in one lr_clk period, which basically means that, since the system is synchronized to sys_clk, a maximum number of 384 operations can be used to calculate the two outputs per input sample. In order to count these operations, four counters will be created Table 3.2. This counters will allow the designer to know in which of the 384 operations per period the filter is in each moment, simplifying this way the control circuit of the filter. In Figure 3.22 the clocks and counters of the system can be seen.

Name	**Counting**	**Synchronization**
count_1	0-191	Rising edge of sys_clk
count_2	0-191	Falling edge of sys_clk
count_3	0-383	Rising edge of sys_clk
count_4	0-383	Falling edge of sys_clk

Table 3.2: Counters.

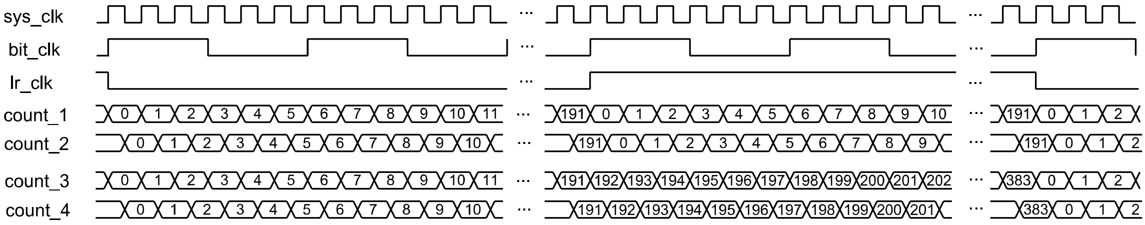


Figure 3.22: Clocks and counters of the system.

The basic structure of the design of the filter can be seen in Figure 3.23. A *ROM memory* to store the filter coefficients, a *RAM memory* to store the input samples that are in the delay line, a *multiplication-accumulation (MAC) block* that operates the input samples and the coefficients and a *Finite State Machine* to control all the processes of the filter, will be needed.

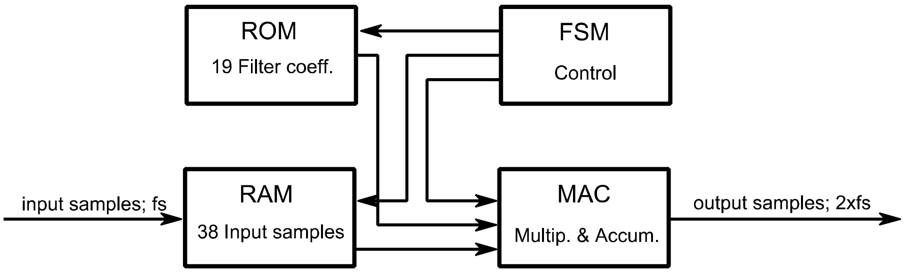


Figure 3.23: Basic structure of the FIR filter.

3.3.1 ROM memory

The order of the filter is 74, this means that 75 coefficients were found with Matlab. There are 2 coefficients per tab, which means that a zero coefficient will need to be added in the last tap. There is an alternation of non-zero and zero values in these 76 coefficients, but the zero sample that should be in the middle of the coefficients is one instead. This results into 38 non-zero and non-one coefficients. These 38 coefficients are symmetrical so just 19 of them are different from each other. As a result, just 19 coefficients will need to be stored. The ROM memory size will be 19 positions of 18 bits. This ROM memory will be initialized with the values of the filter coefficient obtained in Matlab and will not be changed during the filter operation. The values of the coefficients in the 19 sized ROM memory can be seen in Figure 3.24.

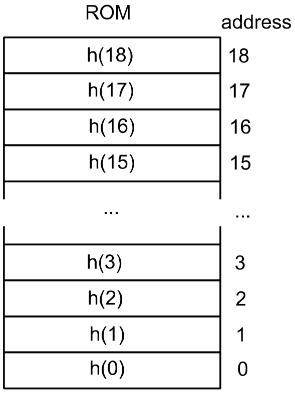


Figure 3.24: Coefficients of the filter stored in the ROM memory.

The ROM memory block inputs and outputs can be seen in Figure 3.25. The ROM memory will be synchronized with the input clock (clk) and will operate only if the enabler signal(en) is one. In the output of the ROM memory (do_a) there will be the value stored in the address chosen by the input(addr), and the output value will be updated every rising edge of the input clock.

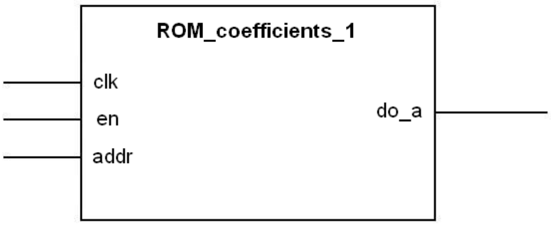
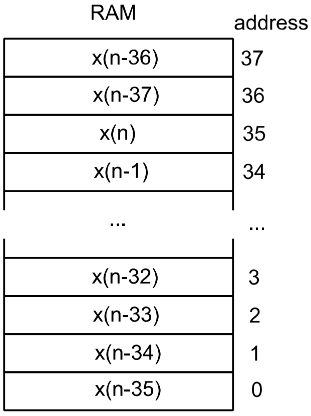


Figure 3.25: ROM memory structure.

3.3.2 RAM memory

The size of the RAM memory will be 38 since there are 38 taps, and each position will contain 16 bits, the number of bits of the input sample. The RAM memory will be initialized with all zeros, and will be written every time that a new input sample enters to the delay line over the old input sample that goes out of the delay line Figure 3.26. The inputs and outputs of the RAM memory are shown in Figure 3.27.



RAM	address
x(n-36)	37
x(n-37)	36
x(n)	35
x(n-1)	34
...	...
x(n-32)	3
x(n-33)	2
x(n-34)	1
x(n-35)	0

Figure 3.26: RAM memory full with input samples. The newest sample written is x(n).

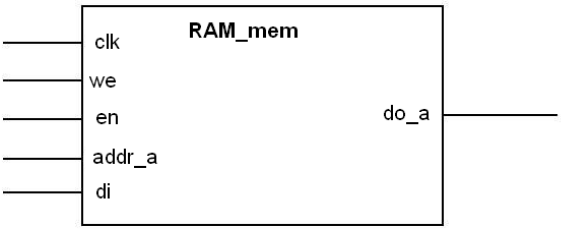
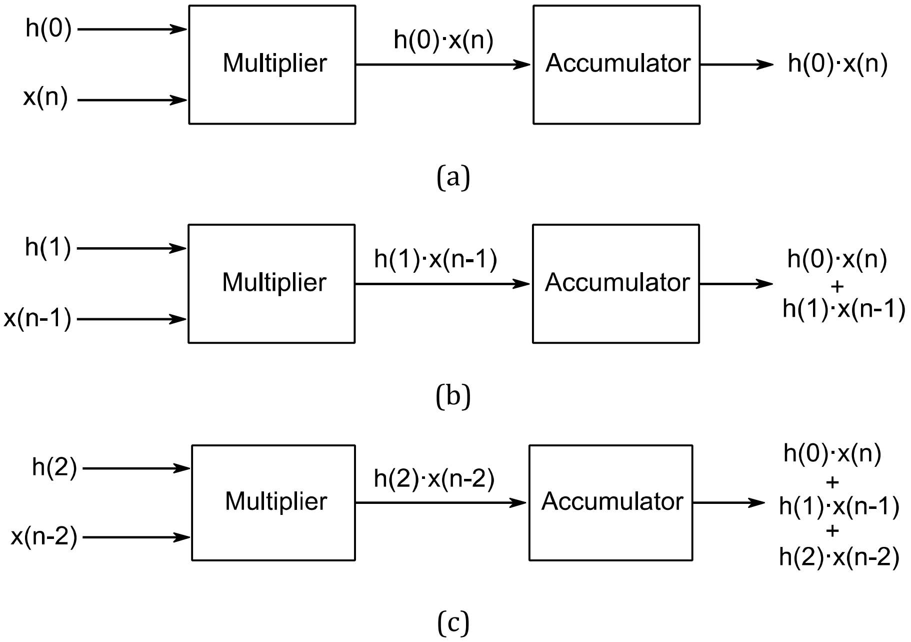


Figure 3.27: RAM memory structure.

The RAM memory works in a similar way as the ROM memory, but it is also capable of writing. If the writing enabler(we) is zero, it works exactly as a ROM memory, in the output (do_a) the value in the address defined by the input(addr_a) is shown. If the writing enabler is one, the value in the input(di) will be stored in the address defined by the input(addr_a).

3.3.3 MAC block

The MAC block will generate both output samples, so it will have to work differently for each of them. The first output will be generated in the first half of the lr_clk. This first output sample will be calculated by multiply all the input samples stored in the RAM memory by each of the filter coefficients stored in the ROM memory, and add all the results together. The multiplier which is already built in the FPGA will be used for this purpose and will calculate all the operations. This means that instead of needing 38 multipliers, just one will be used. This will save hardware and space in the microchip though it will make the output calculation slower. The speed of the output calculation is not a critical factor as long as it doesn't exceed half of a lr_clk period. This multiplier will get an input sample and a coefficient and will multiply both obtaining a result that will be saved in a register. Afterwards another input sample and another coefficient will be multiplied obtaining another result that will be accumulated together with the previous one in the register. This process can be seen in Figure 3.28 and will be done until the whole delay line and the whole coefficients have been multiplied and accumulated.



(a)

(b)

(c)

Figure 3.28: Process of the MAC: (a)Step1;(b)Step2;(c)Step3.

In the second half of the lr_clk the MAC block the second output sample will be calculated. The MAC block will go through all the RAM memory finding the middle sample and passing it to the output, generating the second output per input sample. In this case the register that accumulates is not needed, so two different reset signals will need to be used, one for the whole multiplier block and one specifically for the register that accumulates. The inputs and outputs of the MAC block can be seen in Figure 3.29.

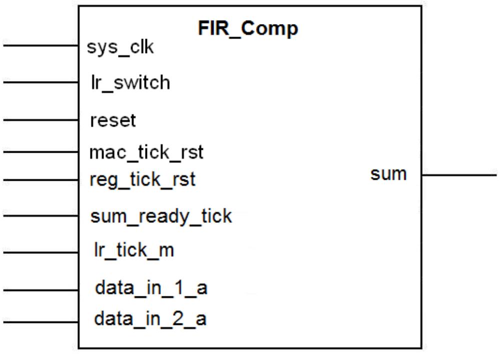


Figure 3.29: MAC structure.

The MAC works with the rising edge of the input clock(sys_clk) and can be reset with the mac_tick_rst input signal. The reg_tick_rst input signal resets the register that accumulates the results of the multiplications. The lr_switch, a signal which is a copy of lr_clk, will tell the MAC which of the two outputs should be calculated. The sum_ready_tick tells the MAC when all the multiplications and accumulations are done, so that the first output sample is ready, and the lr_tick_m tells the MAC when the middle sample of the RAM memory is found, so that it can pass it directly to the output. All these ticks are generated using the counters created before

Obviously the data_in_1_a is the input sample coming from the RAM memory, the data_in_2_a is the filter coefficient coming from the ROM memory and sum is the result for both output samples, which will be the first output in the first half of the lr_clk and the second output in the second half of the lr_clk.

It's worth to notice that the multiplier in the FPGA has delay, which means that every result of the multiplication will be delayed six clock cycles, so this will have to be taken in consideration during the design of the FSM.

3.3.4 FSM block

The FSM block will consist in one register that will control when the state is switched, and a next-state sequential block that will generate the next state. In order to generate the next state, the current state and the inputs will be used. The FSM block will be synchronized with the system clock, and the hardware structure used to design the base of the FSM is shown in Figure 3.30. When there is a rising edge of the system clock the D-flip flop updates the current state with the next state, and the next state is decided by the next-state logic block. This next-state logic block will be designed later when all the states and transitions are defined.

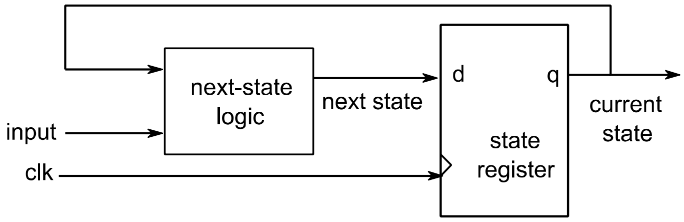


Figure 3.30: FSM basic hardware structure.

The states of the FSM will be:

- Idle state(idle_st): Starting state in which the filter starts after a reset.
- Writing state(wr_st): State in which a new sample is written in the RAM memory.
- Low operating state(low_op_st): State in which the first output sample is calculated by multiplying the delay line input samples by the filter coefficients.
- High operating state(high_op_st): State in which the second output sample is calculated by just passing the middle input sample.
- Ready state(ready_st):State in which the filter waits.
-

After a reset, the filter will start with idle state, going to writing state when all the clocks are initialized. Once the filter is working, the standard cycle will be: writing a new sample in the RAM memory (wr_st), calculate the first output sample (low_op_st), waiting (ready_st), waiting (wr_st), calculate the second output sample (high_op_st), waiting (ready_st) and back to writing a new input sample. It is worth to notice that there is a wr_st between the first ready_st and the high_op_st in which not any sample is written in the RAM memory. The reason for that is that after every ready_st, the FSM will go to writing state either if it comes from the low or high op_states, which simplifies a lot the transition conditions between the states.

Once all the states have been defined, they have to be fit in the 384 operations per lr_clk period. The first operation will be left without use, and the second operation will be used to write a new sample in the RAM memory, wr_st. The following 38 operations will be used to multiply the coefficients and the input samples, low_op_st, and then the ready_st will be kept until the end of the first half of the lr_clk period. The structure of the second half will be the same as the first half, but the high_op_st will be used instead of using the low operating. This means that the 193 operation will be left without use and in the 194 operation the state will be shifted to wr_st but without writing any input sample in the RAM memory. After that, the following 38 operations will be used to go through the RAM memory to find the middle sample and pass it to the output, high_op_st. The rest of the operations will be kept in ready_st waiting for the new cycle to begin. The structure presented for the cycle can be seen in Figure 3.31.

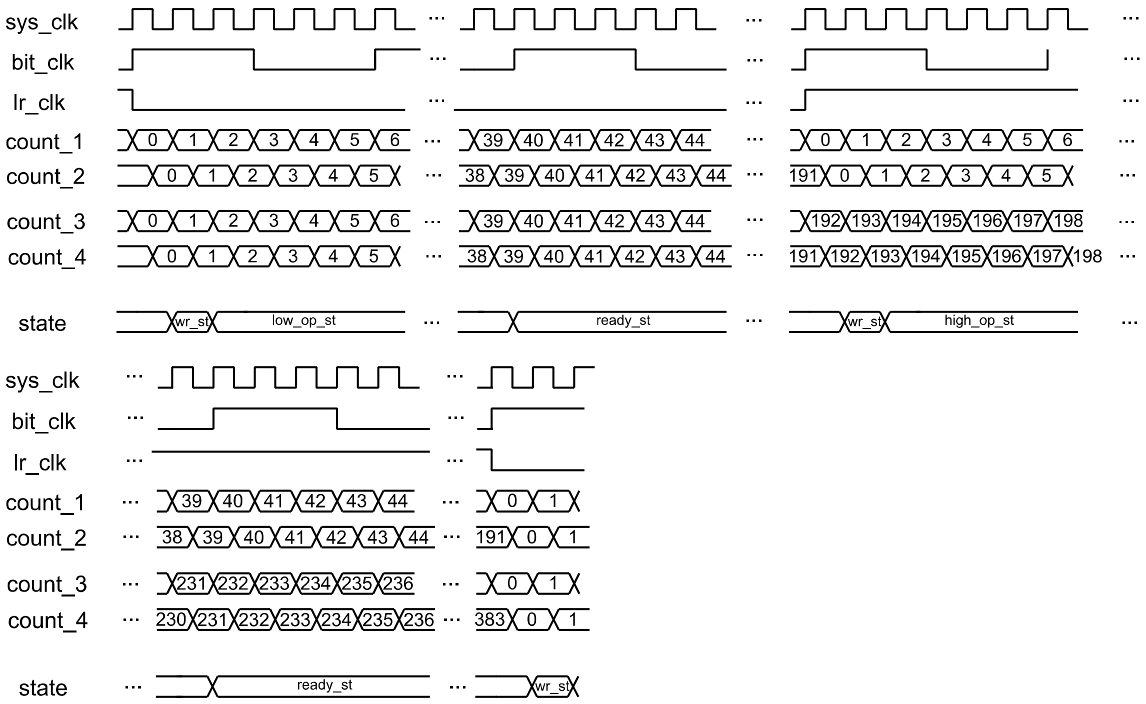


Figure 3.31: States fit into the lr_clk period.

Now, the states have been designed, and also their duration in the lr_clk period. The next step will be to generate the transition conditions to switch from one state to another. These transitions together with the states can be seen in the state diagram in Figure 3.32. By looking at Figure 3.31, two ticks have been created to simplify the logic of the next state. Tick_0 is generated by checking if the count_2 is 0, and tick_39 is generated by checking if count_2 is 39. These ticks will be checked by the FSM in the rising edges of the sys_clk, so it is important to make sure that the ticks update in the falling edge in order to avoid synchronization problems. These two ticks together with the lr_clk control the next state logic, and they will be used according to Figure 3.32.

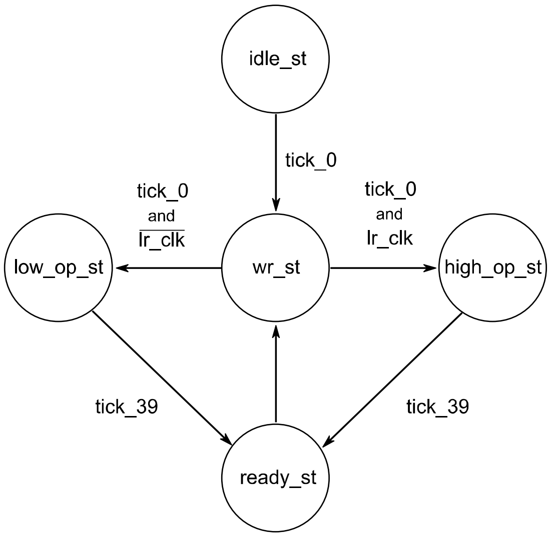


Figure 3.32: State diagram of the FSM.

Now that the state diagram has been defined, the logic of the output signals that control all the other blocks of the filter must be designed. These outputs will be exposed separated in different blocks according to which block are they controlling. All these outputs correspond to the inputs of the RAM, ROM and MAC blocks discussed before.

The outputs that control the RAM memory will be:

- RAM_en: Enables the RAM memory functionality.
- RAM_we: Writing enabler. If this signal is 1, the RAM memory will write in the RAM address. Otherwise it will read in the RAM address.
- RAM_addr: Address in where the RAM memory reads or writes.
- RAM_pr: Pointer that aims at the newest input sample written. It's the reference that will be used in order to match the multiplications.

The outputs that control the ROM memory will be:

- ROM_en: Enables the ROM memory functionality.
- ROM_addr: Address in where the ROM memory reads.

The output that control the MAC block will be:

- mac_rst: Reset of the whole MAC block.
- reg_rst: Reset the register that accumulates the output in the MAC block.

The RAM_en, RAM_we and ROM_en are signals of 1 bit, and it's easy to see in which state each of them should be one, or zero. The Table 3.3 sums up the value of these outputs in each state. It is worth to mention that since there are two wr_st in the cycle in just one of them the RAM_we should be one, and it corresponds to the first part of the lr_clk cycle, when lr_clk is 0. This is why the value of RAM_we in wr_st is not(lr_clk). Also, in the high_op_st the ROM_en will be zero since in this state only the RAM memory is used.

	idle	**writing**	**ready**	**low op.**	**high op.**
RAM_en	1	1	1	1	1
RAM_wr	0	not(lr_clk)	0	0	0
ROM_en	1	1	1	1	0

Table 3.3: Output logic depending on the state.

Regarding to the mac_rst and reg_rst, it is quite intuitive to say that their values depending on the state should be the ones in Table 3.4, but there is something that should be taken in consideration, the multiplier has some delay. During the low_op_st, 38 multiplications are performed, and after them the state is switched to ready_st. The problem is that the delay of the multiplier makes the last result of the 38th multiplication not available instantly, so it will be available during the ready_st. If both MAC resets are set to 1 in the ready_st the correct value of the last multiplications will never be ready because the whole block will be reset before. In order to solve that problem, the value of mac_rst and reg_rst in ready_st will be kept in 0 in the first six cycles of the sys_clk and then switched to 1. This will lead to the correct output table of the mac_rst and reg_rst, Table 3.5.

	idle	writing	ready	low opp.	high opp.
mac_rst	1	1	1	0	1
reg_rst	1	1	1	0	1

Table 3.4: Output logic depending on the state without considering the delay of the multiplier.

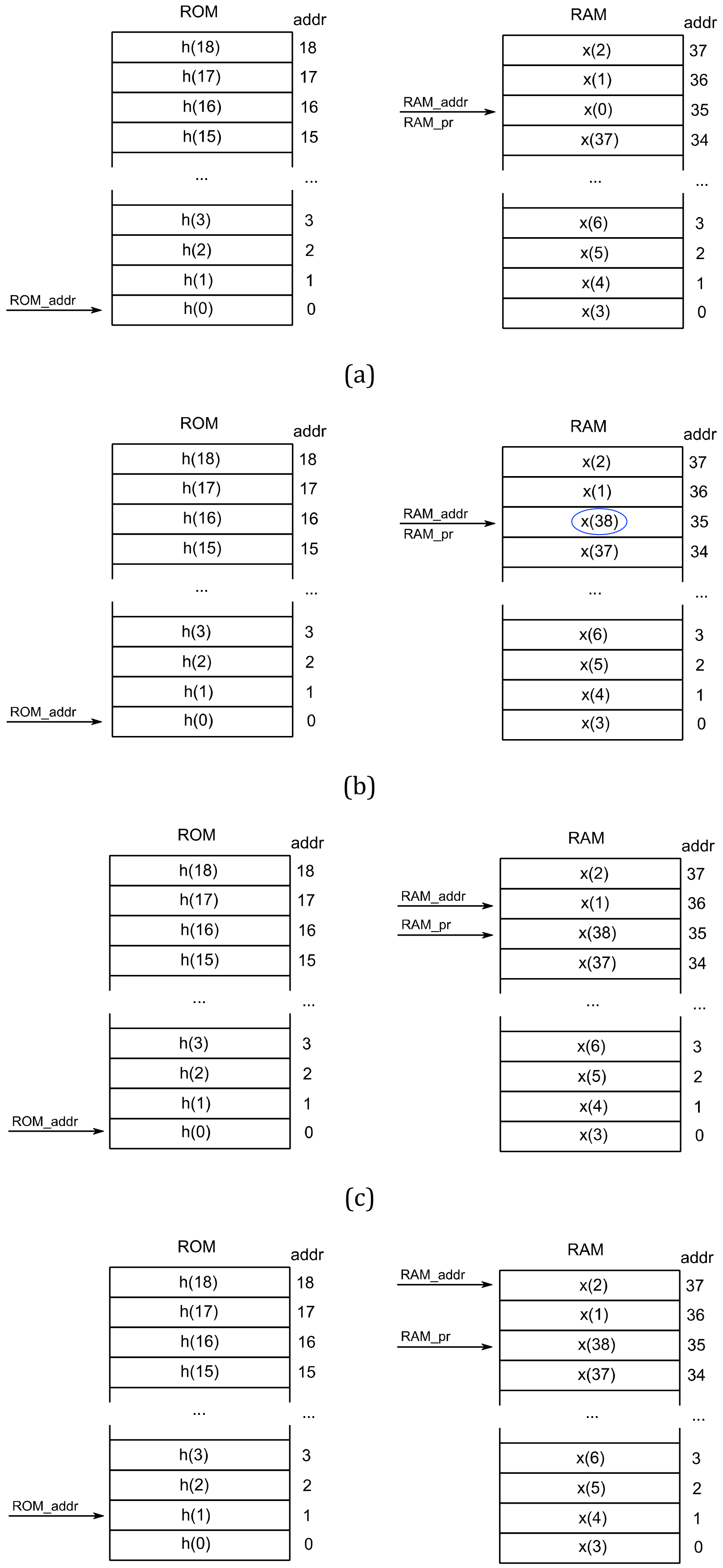
	idle	writing	ready	low opp.	high opp.
mac_rst	1	1	0 for 6 cycles, 1 afterwards	0	1
reg_rst	1	1	0 for 6 cycles 1 afterwards	0	1

Table 3.5: Output logic depending on the state considering the delay of the multiplier.

The last 3 outputs that need to be designed are RAM_addr, RAM_pr and ROM_addr. The RAM_addr will point in the address in where the input sample that needs to be multiplied is stored or the address in where the new input sample should be written into. The RAM_pr is a reference point that will be used in the control of the RAM_addr, and will only be switched one per cycle. It basically indicates where the newest input sample has been written, so every time that a new sample is written the pointer must be changed first. The ROM_addr points in the address in which the filter coefficient that will be multiplied is stored. These three addresses will not need to change its value during the idle_st, ready_st or wr_st, but the logic in the low and high operating states will be more complicated.

Following the structure presented on the filter, the first coefficient of the filter must be multiplied by the newest sample and the oldest sample, the second coefficient must be multiplied by the second newest sample and the second oldest sample etcetera. An easy way to control the addresses of both ROM memory and RAM memory and the pointer of the RAM memory must be found.

The way that these addresses will be controlled can be easily understood with an example of one cycle of the filter. Te assumptions that will be made in this example can be seen in Figure 3.33(a). Basically, the RAM memory is full of input samples from x0 to x37 and a new sample x38 has to come in. The ROM memory is full of coefficients from h0 to h18. Initially, RAM_addr and RAM_pr will be situated in x0, and ROM_addr will be situated in h0.

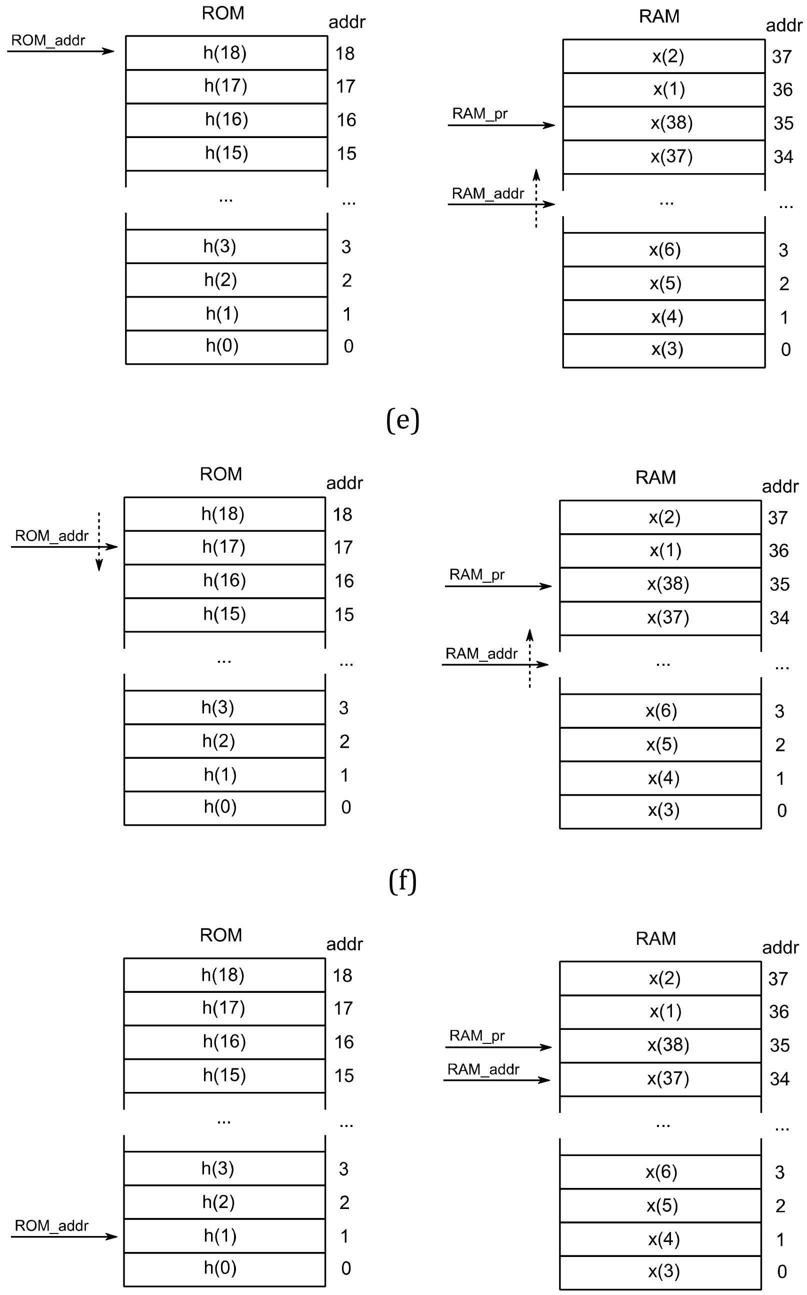

ROM
addr
h(18) 18
h(17) 17
h(16) 16
h(15) 15
... ...
h(3) 3
h(2) 2
h(1) 1
h(0) 0
ROM_addr
RAM
addr
x(2) 37
x(1) 36
x(0) 35
x(37) 34
... ...
x(6) 3
x(5) 2
x(4) 1
x(3) 0
RAM_addr
RAM_pr
ROM
addr
h(18) 18
h(17) 17
h(16) 16
h(15) 15
... ...
h(3) 3
h(2) 2
h(1) 1
h(0) 0
ROM_addr
RAM
addr
x(2) 37
x(1) 36
x(38) 35
x(37) 34
... ...
x(6) 3
x(5) 2
x(4) 1
x(3) 0
RAM_addr
RAM_pr
ROM
addr
h(18) 18
h(17) 17
h(16) 16
h(15) 15
... ...
h(3) 3
h(2) 2
h(1) 1
h(0) 0
ROM_addr
RAM
addr
x(2) 37
x(1) 36
x(38) 35
x(37) 34
... ...
x(6) 3
x(5) 2
x(4) 1
x(3) 0
RAM_addr
RAM_pr
ROM
addr
h(18) 18
h(17) 17
h(16) 16
h(15) 15
... ...
h(3) 3
h(2) 2
h(1) 1
h(0) 0
ROM_addr
RAM
addr
x(2) 37
x(1) 36
x(38) 35
x(37) 34
... ...
x(6) 3
x(5) 2
x(4) 1
x(3) 0
RAM_addr
RAM_pr

(a)

(b)

(c)

(d)



(e)

(f)

(g)

Figure 3.33: (a) Initial state; (b) Writing new sample, x(38) and first; (c) Second; (d) Third multiplications and accumulations; (e) ROM_addr keeps at h(18) for two periods while RAM_addr keeps incrementing; (f) ROM_addr starts decreasing; (g) Last multiplication and accumulation.

The calculation of the first output sample will be shown. The first step will be to write the new sample x38 in the x0 position. Now, with the delay line containing the correct input samples the multiplication process can begin Figure 3.33(b). With the current state the x38 sample is multiplied by h0, saving the result in a register. Then RAM_addr will increment one position reaching x1 and ROM_addr will keep in its place. Now, x0 and h0 are multiplied, and the result is accumulated with the previous one in the same register Figure 3.33(c). Both addresses will increment, multiplying this time x2 and h1 and accumulating the result Figure 3.33(d). Both addresses will keep incrementing one every step, multiplying the corresponding input sample and coefficients until the h18 coefficient is reached. At this point ROM_addr will stay two cycles in the h18 coefficient while RAM_addr keep increasing Figure 3.33(e). After this, the ROM_addr will decrease one position every clock cycle until h0 is reached, while the RAM_addr will keep increasing Figure 3.33(f). The last multiplication will be x(37) per h(1) Figure 3.33(g). Finally RAM_pr will jump to the position in where x(1) is situated and RAM_addr will be set where the pointer is situated. Note that ROM_addr goes back to its starting point by itself. Obviously, when RAM_addr reaches the highest address, it will wrap around finding the lowest address. The output result using this method will be the one in Eq.(3.4), which exactly matches the desired value.

$$y_1(0) = \; x(38)\cdot h(0) + \; x(1)\cdot h(0) + x(2)\cdot h(1) + \cdots$$

$$\ldots + x(18)\cdot h(17) + x(19)\cdot h(18) + \cdots$$

$$\ldots . + x(20)\cdot h(18) + x(21)\cdot h(17) + \cdots$$

$$\ldots x(37)\cdot h(1); \tag{3.4}$$

Summing up, during the low_op_st RAM_addr must be increasing step by step and wrapping around when it reaches the highest address, address 37. ROM_addr will have to stay in the zero position for two cycles, increment to its highest position, position 18, staying 2 cycles there and then decrease again until the zero position.

Now, the calculation of the second output sample will be shown. After the calculation of the first output samples, the pointer and RAM_addr will be situated in the position of x(1). The ROM memory will not be needed so ROM_addr will be kept in the same value. RAM_addr will just increment wrapping around if needed, going from the position of RAM_pr through all the input samples until it reaches the pointer again. The aim of that is to find the middle sample which will be situated always in the same relative place from the pointer position.

The behavior of these last three outputs the FSM block can be seen in Table 3.6, where "+" means incrementing one position per sys_clk period, "-" decrement one position per sys_clk period and "=" maintain its value.

State	**low_op_st**				**ready_ st**	**wr_st**	**high_op_st**
Operation	*1-19*	*20*	*21-38*	*39*	*40-191*	*0*	*op 1-38*
RAM_addr	+	+	+	Jump to RAM_pr	=	=	+
RAM_pr	=	=	=	+	=	=	=
ROM_addr	+	=	-	=	=	=	=

Table 3.6: RAM address, RAM pointer and ROM address sequence.

In order to control the behavior changes of the ROM_addr during the low_op_st, two ticks have been created using the counters described before: rom_tick_inc and rom_tick_dec. Each of them will be generated by forcing them to be one when the ROM_addr should increase or decrease respectively, an zero otherwise. The ROM_addr will be programmed to increase when rom_tick_inc is one, decrease when rom_tick_dec is one, and stay in its value otherwise.

The inputs of the FSM block will be the signals needed to generate the outputs described before. The structure of the whole FSM block with the inputs needed to generate those control outputs can be seen in Figure 3.34.

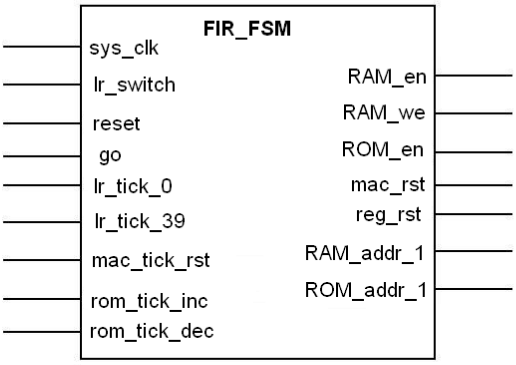
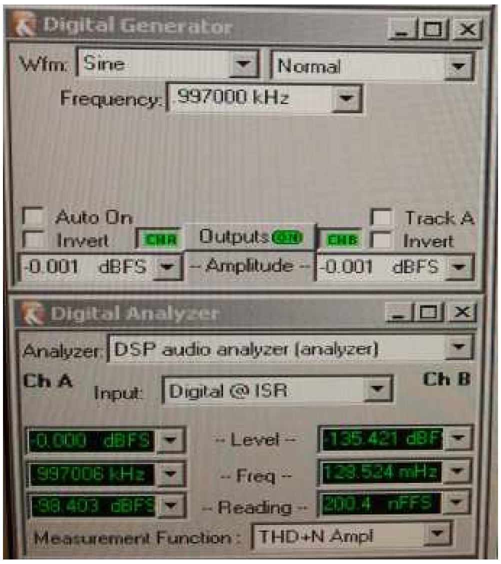


Figure 3.34: Structure of the FSM block.

Chapter 4

4. Simulation and implementation

Now that the whole filter has been designed in VHDL, the fixed point input signal generated by Matlab is used to test the VHDL model. The output of the filter will be saved and compared to Matlab. Once both outputs matches, it can be assumed that the VHDL code is correct, so the implementation process can start. The filter, now, will be connected between the I2S serial to parallel and parallel to serial interface designed, and will be implemented as a whole block. The code will be downloaded to the FPGA and it will be connected to the audio analyzer in order to test it. The sample frequency generated with code for the simulation was exactly 22.05kHz, but the sample frequency that will be used for testing the filter will be generated by the oscillator in the FPGA so the closest frequency that is possible to generate with that hardware will be chosen. In this case 23.4kHz will be the sampling frequency chosen and in consequence the output sampling frequency will be double, 46.8kHz. The input signal frequency set in the audio analyzer will be a sinusoidal signal with frequency 997Hz, with a SQNR around 98dB. The settings in the Audio-Analyzer software can be seen in Figure 4.1.



Digital Generator

Wfm: Sine Normal

Frequency: .997000 kHz

□ Auto On □ Track A

□ Invert CHA Outputs ON CHB □ Invert

-0.001 dBFS -- Amplitude -- -0.001 dBFS

Digital Analyzer

Analyzer: DSP audio analyzer (analyzer)

Ch A Input: Digital @ ISR Ch B

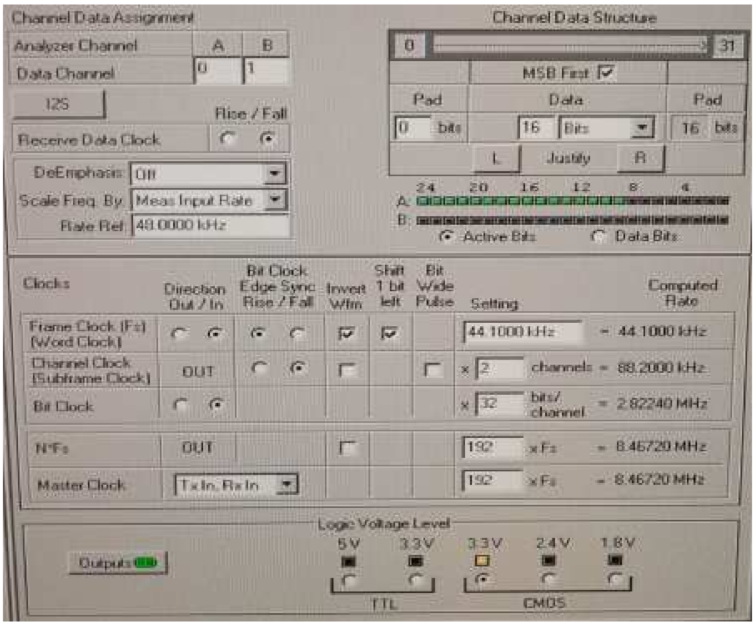
-0.000 dBFS -- Level -- -135.421 dBF

.997006 kHz -- Freq -- 128.524 mHz

-98.403 dBFS -- Reading -- 200.4 nFFS

Measurement Function : THD+N Ampl

(a)



Channel Data Assignment

Analyzer Channel	A	B
Data Channel	0	1

Receive Data Clock Rise / Fall ○ ●

DeEmphasis: Off

Scale Freq. By: Meas Input Rate

Rate Ref: 48.0000 kHz

Channel Data Structure

0 – 31

MSB First ☑

Pad	Data	Pad
0 bits	16 Bits	16 bits

● Active Bits ○ Data Bits

Clocks	Direction Out / In	Bit Clock Edge Sync Rise / Fall	Invert Wfm	Shift 1 bit left	Bit Wide Pulse	Setting	Computed Rate
Frame Clock (Fs) (Word Clock)	○ ●	● ○	☑	☑		44.1000 kHz	= 44.1000 kHz
Channel Clock (Subframe Clock)	OUT	○ ●	□		□	x 2 channels	= 88.2000 kHz
Bit Clock	○ ●					x 32 bits/channel	= 2.82240 MHz
N*Fs	OUT		□			192 x Fs	= 8.46720 MHz
Master Clock	Tx In, Rx In					192 x Fs	= 8.46720 MHz

Logic Voltage Level

TTL		CMOS		
5V	3.3V	3.3V	2.4V	1.8V
○	○	●	○	○

(b)

Figure 4.1: Audio analyzer software configuration: (a) Input generation; (b) Output signal receiving and configuration of the clocks for the I2S interface.

In order to see the real effect of the filter and not the effect of the devices used for the measuring, two outputs will be registered. The first output will be generated just by passing the input signal through all the system but without the filter Figure 4.2, and the second output will be generated by passing the input signal through all the system including the filter Figure 4.3. The maximum frequency in Figure 4.2 and Figure 4.3 is 23 kHz since it is the highest frequency of the audio analyzer frequency band.

The first noticeable fact is that the original frequency spectrum's gain has also decreased around -6dB, Figure 4.3. The reason for this suppression is the fact that even if the output signal is expressed with 2 integer bits and 20 fractional bits, since the coefficients are normalized and the values of the input signal are contained between -1 and 1, the most significant integer bit will not be used. The Audio Analyzer interprets that the full-scale signal is 22 bits, but since the first bit is not used, the signal is just 21 bits. Reducing the most significant bit is equivalent to halving the amplitude of the signal, which is the same as suppressing it 20·log(1/2) ≃ -6dB. In Figure 4.2 the input is not passed through the filter, so the -6dB attenuation is not applied.

Ideally, the frequency spectrum of a discrete signal should be repeated with the same gain every multiple of fs. However, as it can be seen in Figure 4.2, the gain of the image that has to be filtered is around -23dB. This suppression comes from the low-pass filter that the Audio Analyzer applies to the input signal. This low-pass filter is applied both in the unfiltered frequency spectrum, Figure 4.2, and the filtered frequency spectrum, Figure 4.3. This image will be suppressed down to -80dB after the filtering but, as it was said, 6 from these dB come from the bit reduction of the output, so basically (-23)-(-80+6)=51dB are attenuated by the filter. This attenuation is 9dB lower than the suppression specified. The filter designed in VHDL had been compared to the ideal fixed point model of the filter and the output matches perfectly, so a correct design of the filter can be assumed. In that design, the minimum attenuation of the stop-band was -60dB, so the output obtained with the Audio Analyzer don't match the output obtained with the VHDL design. The design was implemented by the VHDL software, but not any time, area or placement constraints were used, which means that the software places the components at its will. This component placing might cause delays of signals, and can even desynchronization the system, which is probably the reason why there is a lack of 9dB attenuation in the fast Fourier transformation (FFT) of the output. A further study of the constraint specifications that can be given to the software for the implementation should be done in order to make sure that the design is implemented correctly without causing any delays. These constraints should place the components of the same blocks close to each other in order to minimize the delays, and also make sure that the system clock arrives to all the blocks sufficiently quick.

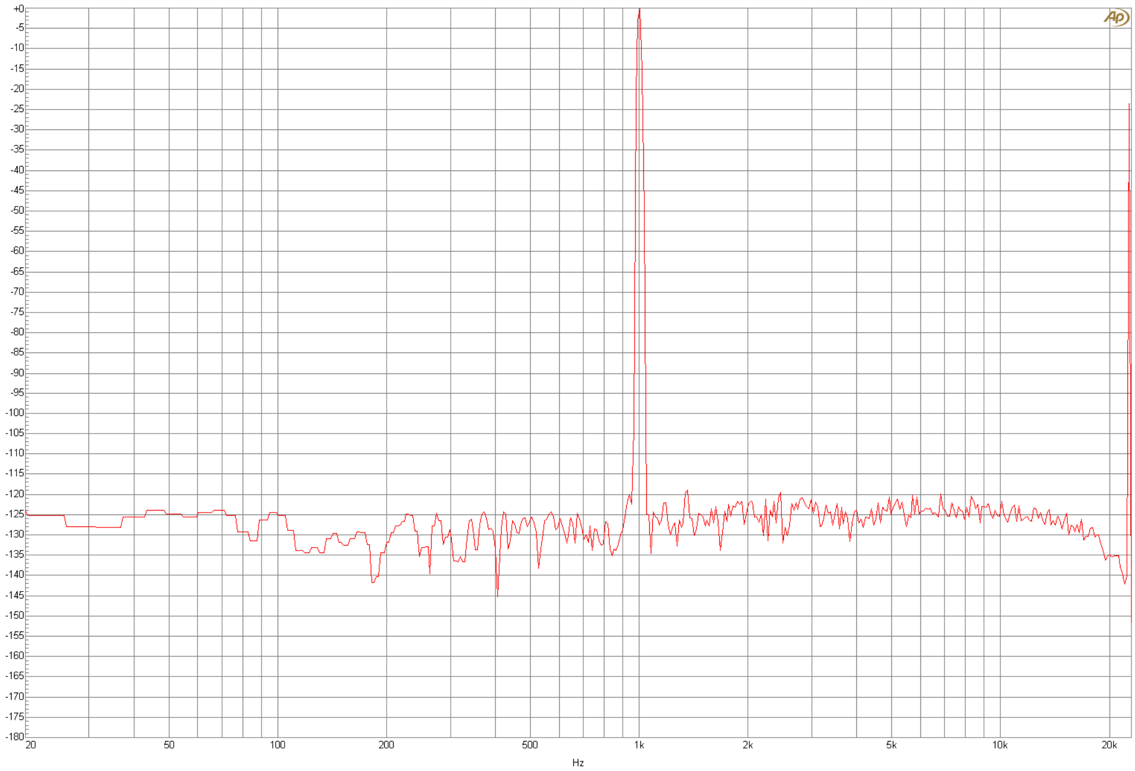


Figure 4.2: Frequency spectrum of the unfiltered.

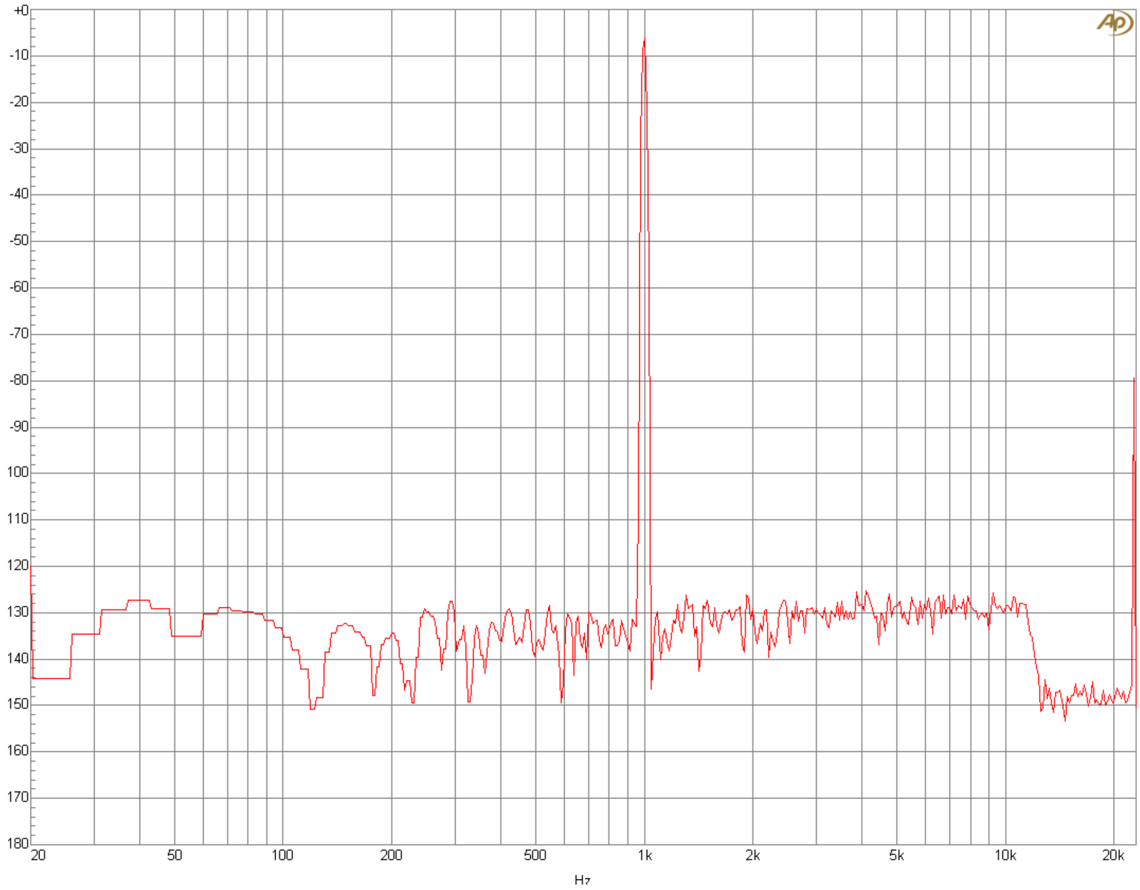


Figure 4.3: Frequency spectrum of the filtered output.

Chapter 5

5. Discussion

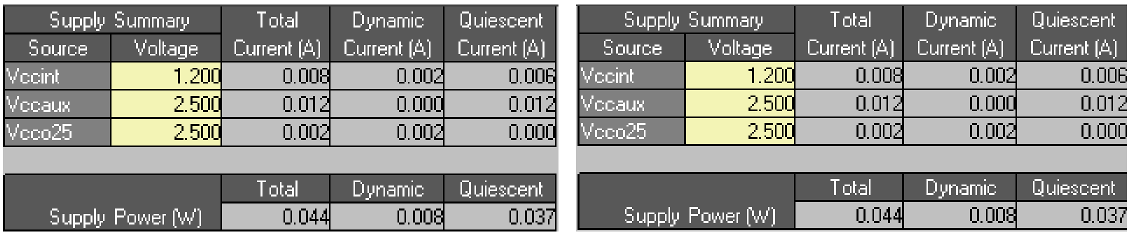
After the design and the measurements, a comparison between the new filter design and the original filter that was previously used in the back-stage will be done. The properties of both filters can be seen in Table 5.1. The original filter specifications for the attenuation in the stop-band were -80dB but it was found that with -60dB the images that were enough suppressed so the extra -20dB suppression was not needed. This reduction of the attenuation specified will lead to a less sharp filter which will mean that the hardware required to implement it will be lower. This change in the specifications of the attenuation in the stop-band will also leads to a reduction of the order of the filter from 100 to 74 hence the number of coefficients of the half-band filter would be reduced from 101 to 75. In both cases half of the coefficients were zero, which were the ones stored in the ROM memory. Also, in the original filter the symmetric property of the filter coefficients was not taken in account which means that both symmetrical parts were stored in the ROM memory. In the new design, just one of the symmetrical parts is stored, which will decrease the number of coefficients stored to the half, and due to that, the size of the ROM memory will also be halved. The overall reduction of coefficients stored in the ROM memory is from 51 to 19, which is significant diminution of a 37,25%.

	Original filter	**New filter designed**
Attenuation of the stop-band specifications	-80dB	-60dB
Filter order	100	74
Number of coefficients	101	75
Coefficients stored in the ROM memory	51	19
Number of taps	51	38

Table 5.1: Original and new filter properties.

A part from the ROM memory size reduction other hardware savings can be achieved. The FPGA is a versatile device, which is an advantage for the design, but it also means that the components used to implement a design will not be specifically designed for its function. Therefore, it is possible that the same versatile component is used for both designs, order 100 and 74, even though the lower order design could use a more specific and more hardware saving component. This fact makes a full and precise description of the hardware savings with this order reduction not achievable with the FPGA design. For that purpose the designs should be imported to ASIC. The ASIC designs would only contain the components designed specifically for this purpose, which means that any single hardware saving would be contemplated with the comparison of both.

The current consumption of the Spartan 6 board can also be found. This current consumption can't be used to know the current consumption of the filter itself since it includes the consumption of the whole board including the FPGA but also other blocks needed for the design like the digital clock manager (DCM) or the oscillator of the board. Even with that, the current consumption of the board with the original filter implemented and the new filter implemented can be compared since the amount of current not used in the FPGA will be similar in both cases. The difference between the two values will mostly come from the consumption of the FPGA since, so an idea of the current reduction can be obtained. The current consumption of the original filter and the new filter can be seen in Figure 5.1. As it is shown, the current consumption does not vary from order 100 to order 74. This means that if some current consumption is saved with the attenuation reduction, the magnitude is negligible compared to the current consumption of the FPGA itself, which leads to an equal overall current consumptions for both cases. After this design, the filter will be implemented with ASIC, which will allow precise current consumption measurements since not any additional components will be contained in the microchip. In this case the current consumption that will be measured will be strictly the one used for the operation of the filter.



Supply Summary		Total	Dynamic	Quiescent
Source	Voltage	Current (A)	Current (A)	Current (A)
Vccint	1.200	0.008	0.002	0.006
Vccaux	2.500	0.012	0.000	0.012
Vcco25	2.500	0.002	0.002	0.000

	Total	Dynamic	Quiescent
Supply Power (W)	0.044	0.008	0.037

(a)

Supply Summary		Total	Dynamic	Quiescent
Source	Voltage	Current (A)	Current (A)	Current (A)
Vccint	1.200	0.008	0.002	0.006
Vccaux	2.500	0.012	0.000	0.012
Vcco25	2.500	0.002	0.002	0.000

	Total	Dynamic	Quiescent
Supply Power (W)	0.044	0.008	0.037

(b)

Figure 5.1: Current consumption of the: (a) original filter, order=100; (b) the new filter, order=74.

Chapter 6

6. Future work

6.1 Reducing the number of multiplications

The first idea for the future work that comes into mind is to save half of the multiplications by taking out the common factor of the input samples with the same filter coefficient multiplying them. As it can be seen in Eq.(6.1) the mathematical idea is that if two sample inputs are multiplied by the same filter coefficient, taking out the filter coefficient as a common factor reduces the number of multiplications from two to one. Since the filter coefficients are symmetrical, this process will be applied to all the input samples reducing the number of multiplications to the half, from 38 to 19. It's obvious that the number of adders will also increase, but the hardware demand of an adder is much lower than a multiplier, so hardware will be saved anyway. The new structure of the filter would be the structure in Figure 6.1.

$$h(0) \cdot x(n) + h(0) \cdot x(0) = h(0) \cdot [x(n) + x(0)] \tag{6.1}$$

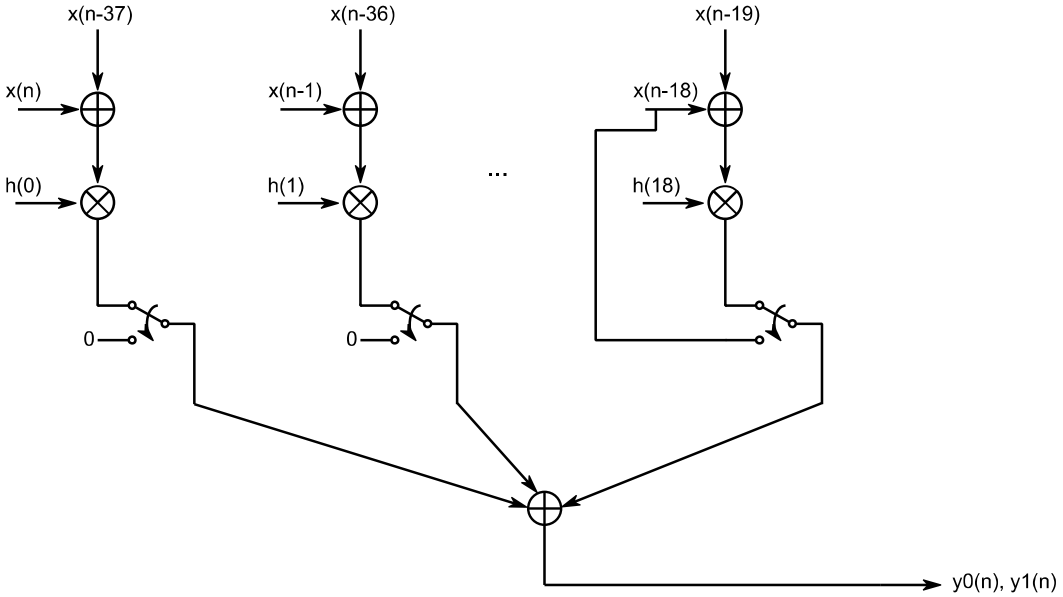


Figure 6.1: New structure with the common factor approach.

6.2 Reducing the number of output bits

Another further improvement that should be done to the filter is the reduction of output bits. The original design had to suppress -80dB, so the maximum quantization achievable that maintained the SQNR=98dB at the output was to 2 integer bits and 20 fractional bits. With the new specifications of -60dB these maximum quantization might change. Starting by the last quantization applied (2 integer bits and 20 fractional bits) more bits should be quantized step by step until the maximum reduction of bits that satisfy the output SQNR specification is found.

In Figure 6.2 the starting point quantization can be seen, in where the noise level of the pass band is still -125dB. The next quantizations will reduce the number of fractional bits to 19 and 18, giving as a result the outputs in Figure 6.3 and Figure 6.4 respectively. The level of the pass band noise is still maintained to -125dB which means that further quantization should be tried. Going down to 17 fractional bits the Figure 6.5 is obtained. It seems that now the noise level in the pass band has increased significantly, which means that probably the SQNR will start getting below the 98dB specifications. Checking the values of the SQNR with a 2 integer bits and 17 fractional bits the result obtained is 96.2515 dB which is clearly below the 98dB specifications. This will lead to the disregarding of this last quantization. The SQNR obtained with the immediate higher quantization, 18 fractional bits, is SQNR=97.8 dB which perfectly satisfies the input specifications. It is worth to keep in mind that that 0.2dB difference between 97.8dB and 98dB comes from the 0.2dB attenuation of the input signal.

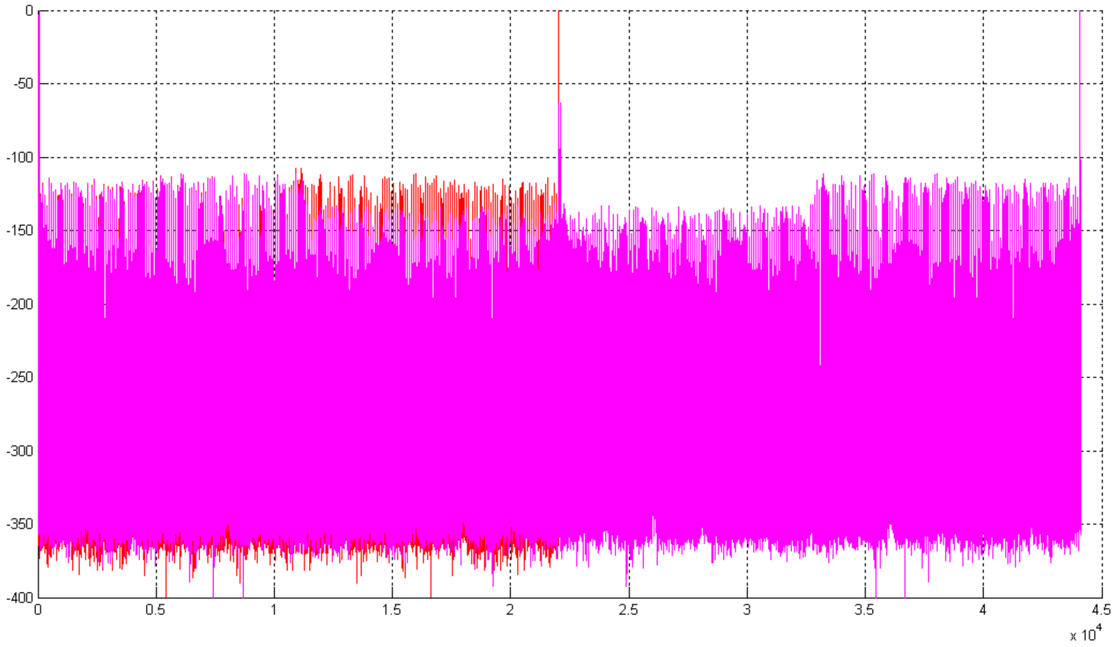


Figure 6.2: Output quantized to 2 integer bits and 20 fractional bits.

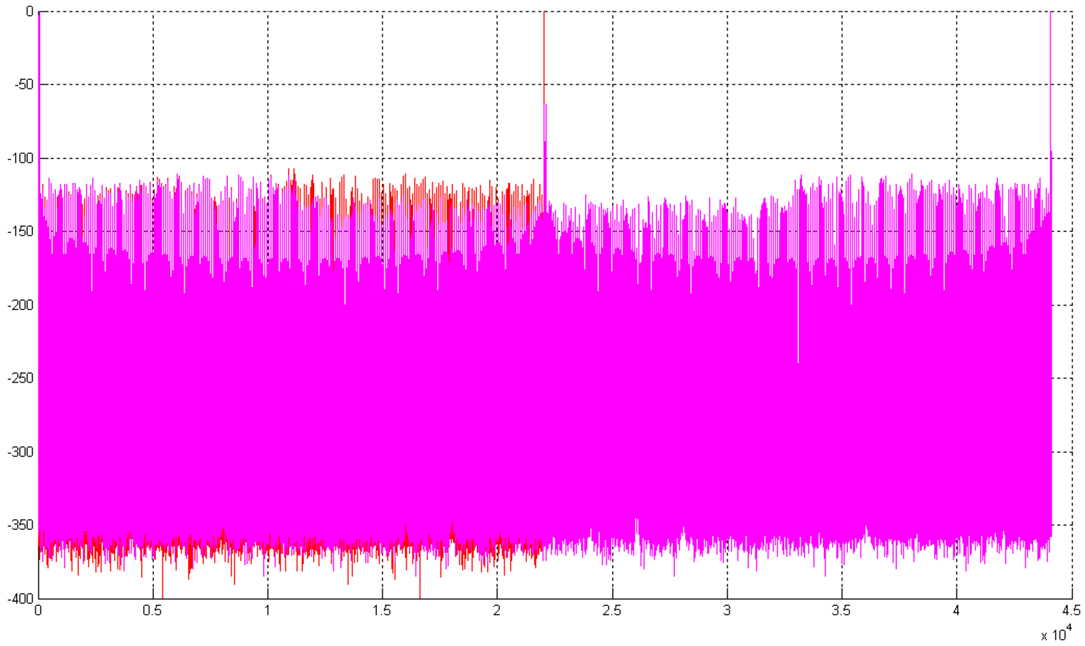


Figure 6.3: Output quantized to 2 integer bits and 19 fractional bits.

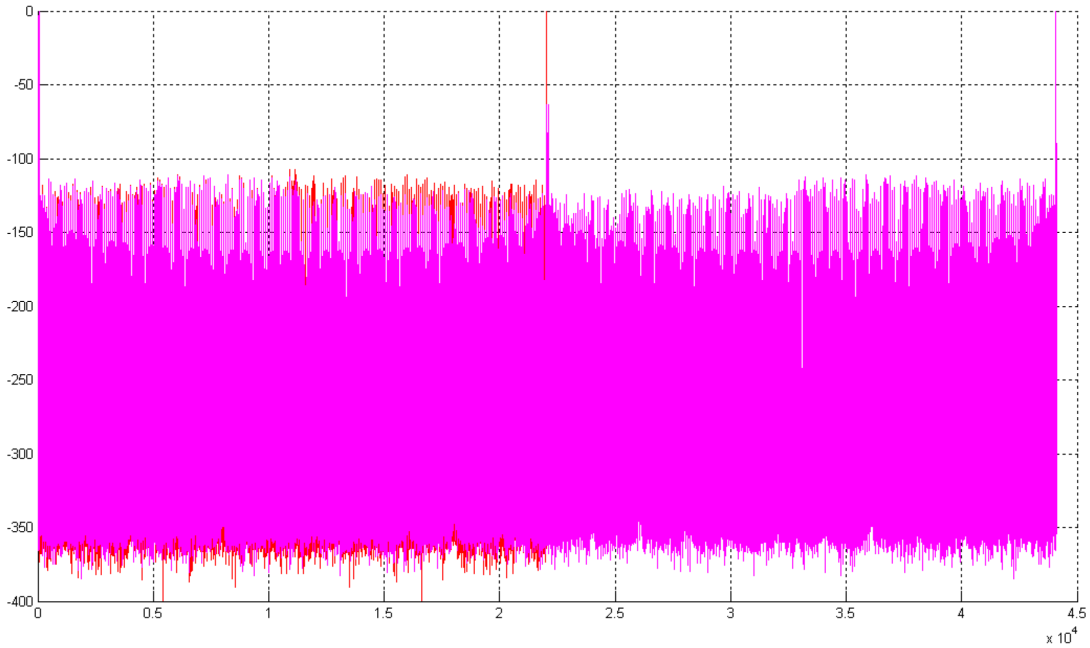


Figure 6.4: Output quantized to 2 integer bits and 18 fractional bits.

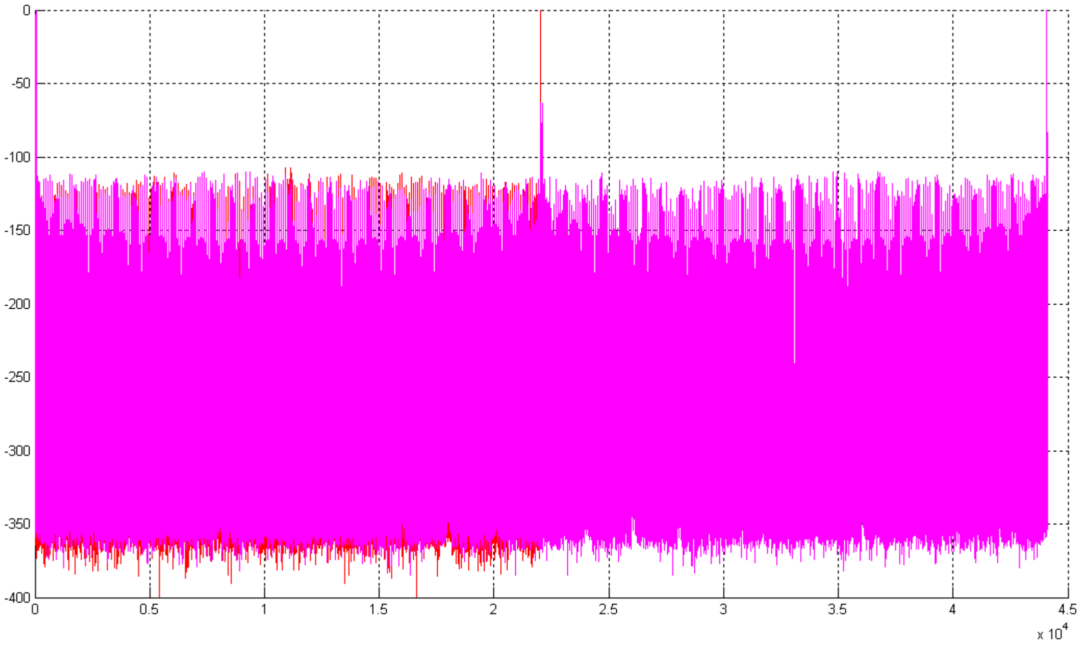
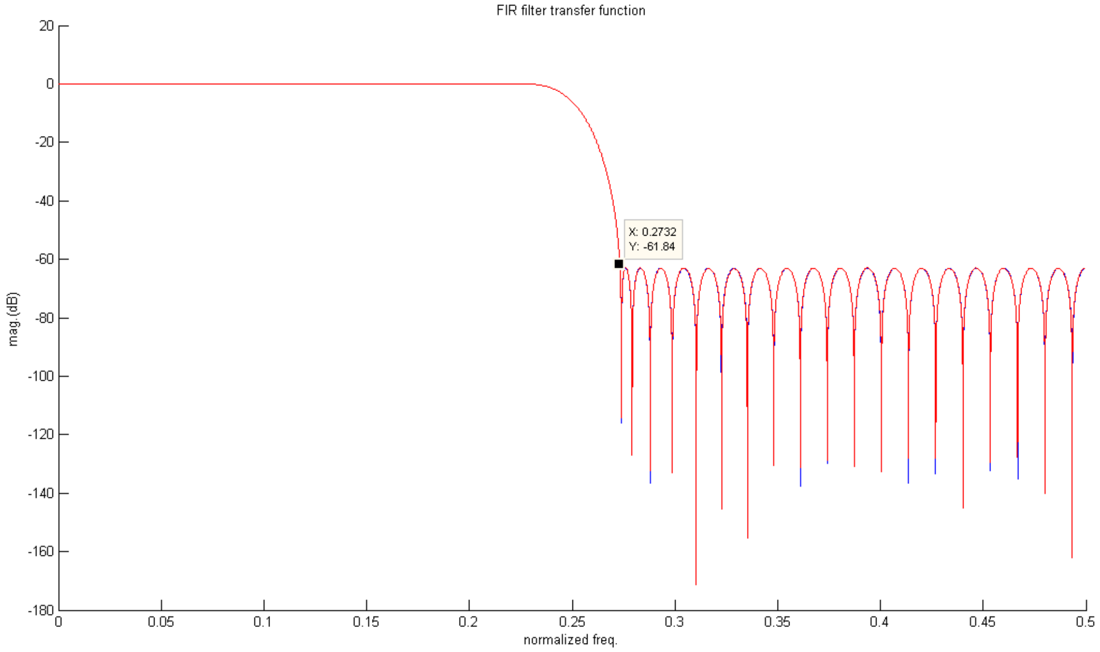


Figure 6.5: Output quantized to 2 integer bits and 17 fractional bits.

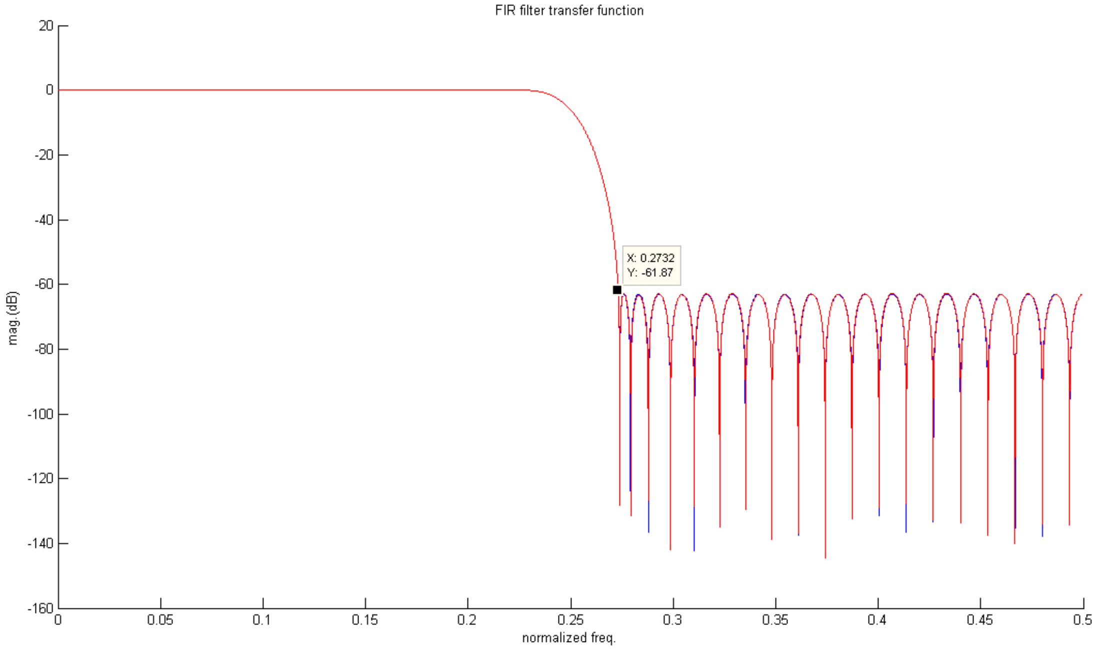
6.3 Filter coefficients quantization

The quantization of the filter coefficients used in this paper was set to 18 bits since it is the edge in where the transfer function starts to differ significantly from the floating point precision transfer function. Also, since the prototype of this design is implemented with the FPGA and the maximum number of bits at the input of the multiplier inside the FPGA is 18, even if the coefficients were quantized to a lower number of bits, the hardware needed for the implementation would remain the same. However, after the prototype, the design should be imported to ASIC where this multiplier could be customized to the number of bits needed. In consequence, if the number of bits of the coefficients of the filter is reduced, even if in the prototype in the FPGA there will not be any improvement, in the final design in ASIC hardware will be saved. With that idea in mind, the filter coefficient quantization should be analyzed further.

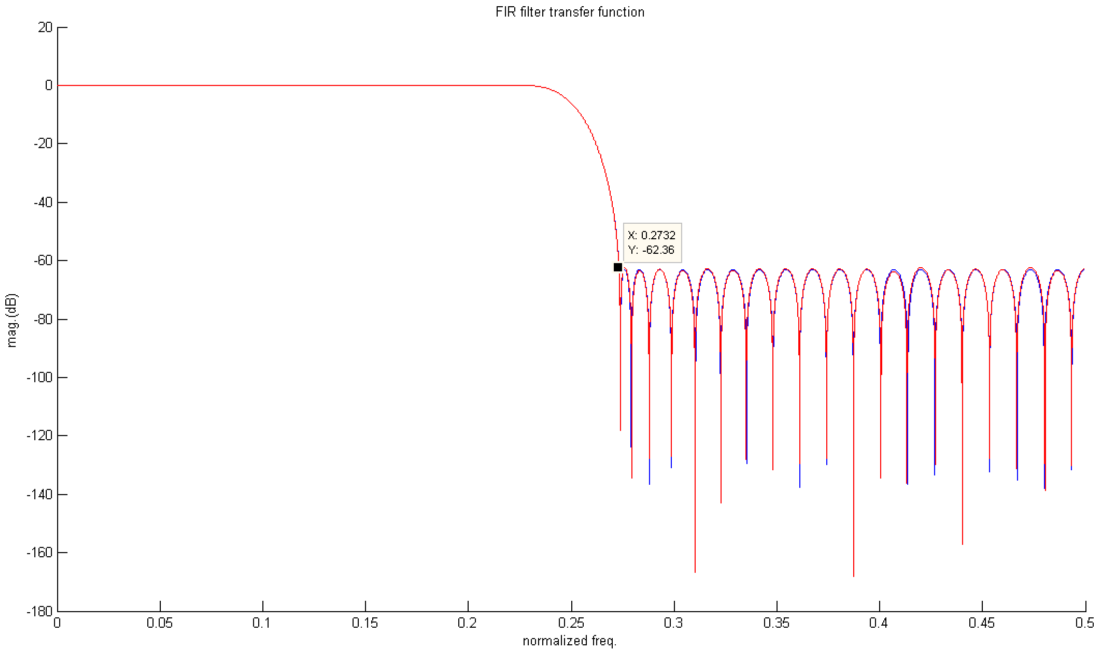
As it was said, a quantization that lowers the number of bits below 18 make the transfer function of the quantized filter differ significantly from the non-quantized one. However, even if the transfer function differs, it is possible that the attenuation in the stop band is still below -60dB. In this case, a further quantization could be used since the specifications would still be satisfied. Using that idea, and checking the attenuation with different quantizations the Figure 6.6 is found. As it can be seen by quantizing down to 18, 17, 16, 15 and 14 bits, the attenuation of the stop band is still over -60dB, but when the quantization is done further down to 13 bits, there is a peak in the stop band that is attenuated less than -60dB. In consequence the quantization of the coefficients of the filter that should be used in the future for this filter could go down to 14 bits.


FIR filter transfer function
20
0
-20
-40
-60
-80
-100
-120
-140
-160
-180
mag.(dB)
X: 0.2732
Y: -61.84
0
0.05
0.1
0.15
0.2
0.25
0.3
0.35
0.4
0.45
0.5
normalized freq.

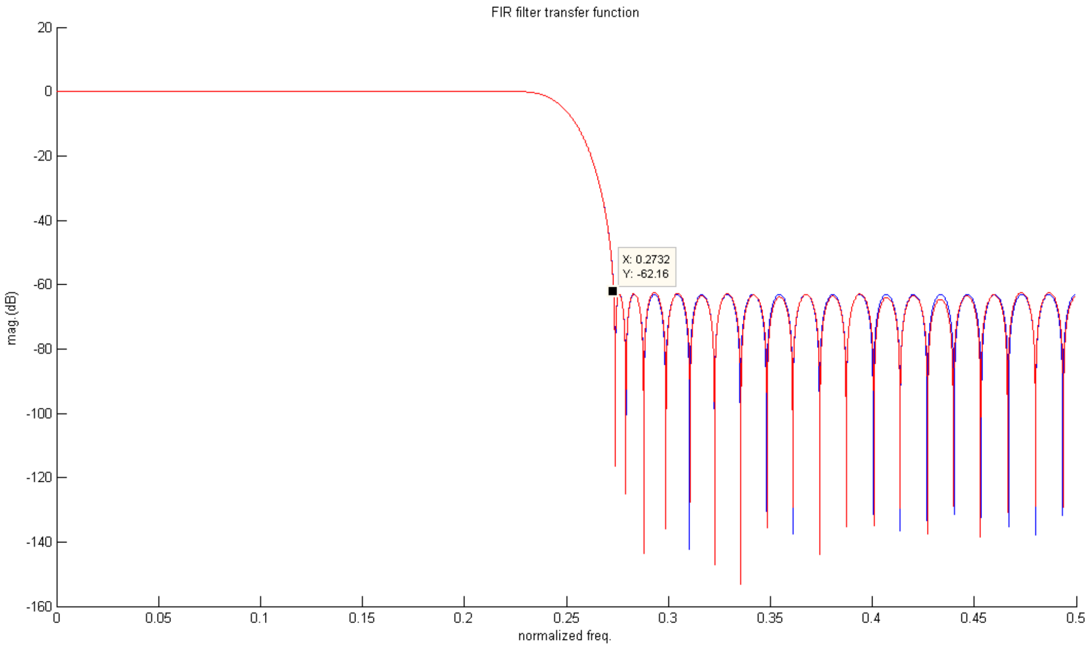
(a)


FIR filter transfer function
20
0
-20
-40
-60
-80
-100
-120
-140
-160
mag.(dB)
X: 0.2732
Y: -61.87
0
0.05
0.1
0.15
0.2
0.25
0.3
0.35
0.4
0.45
0.5
normalized freq.

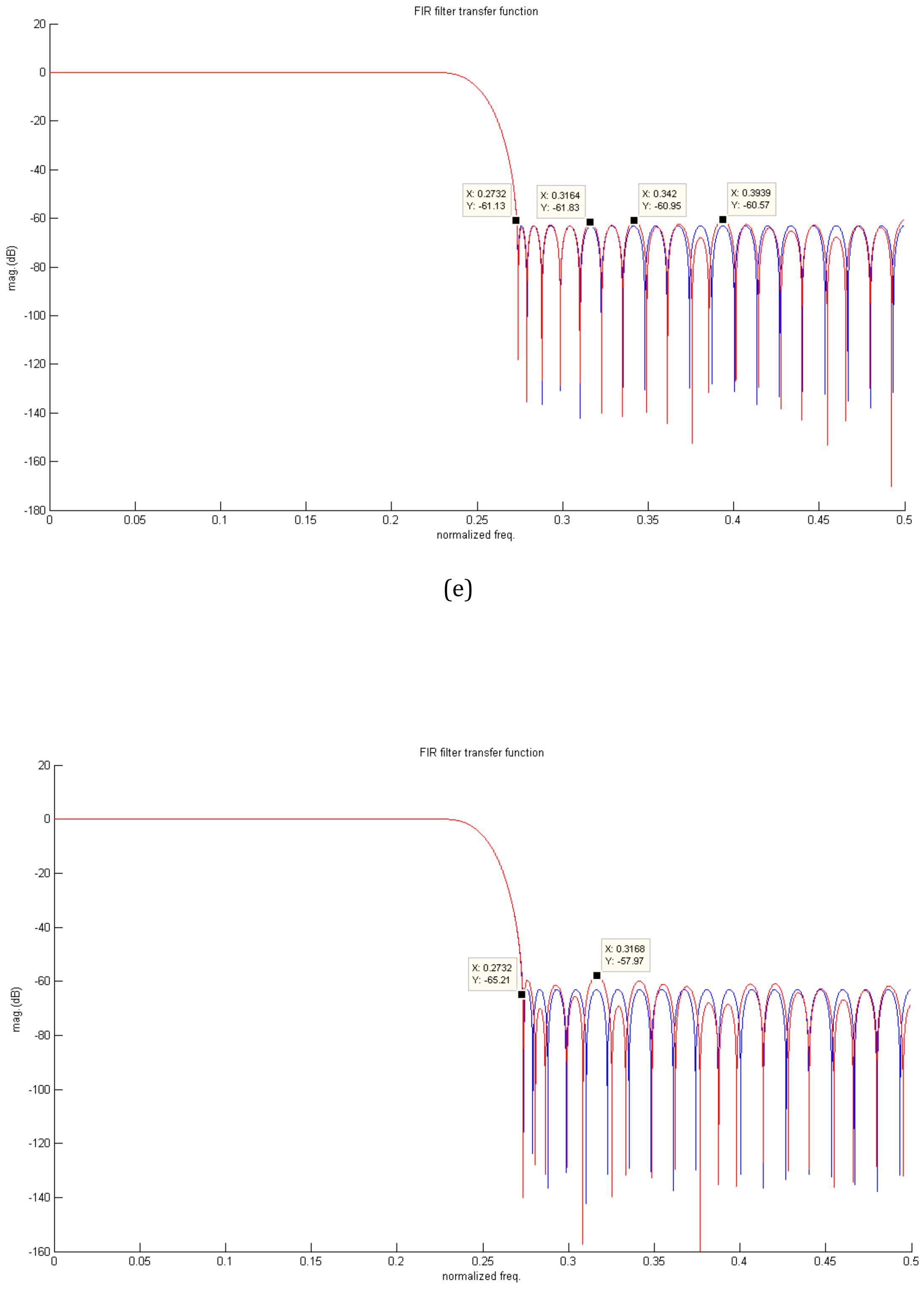
(b)


FIR filter transfer function
20
0
-20
-40
-60
-80
-100
-120
-140
-160
-180
mag.(dB)
X: 0.2732
Y: -62.36
0
0.05
0.1
0.15
0.2
0.25
0.3
0.35
0.4
0.45
0.5
normalized freq.

(c)


FIR filter transfer function
20
0
-20
-40
-60
-80
-100
-120
-140
-160
mag.(dB)
X: 0.2732
Y: -62.16
0
0.05
0.1
0.15
0.2
0.25
0.3
0.35
0.4
0.45
0.5
normalized freq.

(d)



(e)

(f)

Figure 6.6: Transfer function of the FIR filter high precision (blue) and with different quantizations(red) with number of bits: (a)18; (b)17; (c)16; (d)15; (e)14; (f)13.

6.4 Multiplier-less filter

A part from this multiplication reduction, and the reduction of the number of bits of the output and the coefficients, even a more important goal should be achieved. As it is explained in [9] the multipliers are the components in FIR filters that consume maximum power, and they are also very high hardware demanding. For this reason if the multiplier could be replaced, the microchip could be much smaller, and the hardware needed for its implementation would also be lower. With the simple delay line approach that was used in this design it's unfeasible to get rid of the multiplier, but other approaches might be able to do so. In consequence, the next step of this project will be to use a different approach to the design to try to find a way to obtain a multiplier-less filter.

6.4.1 Subfilters approach

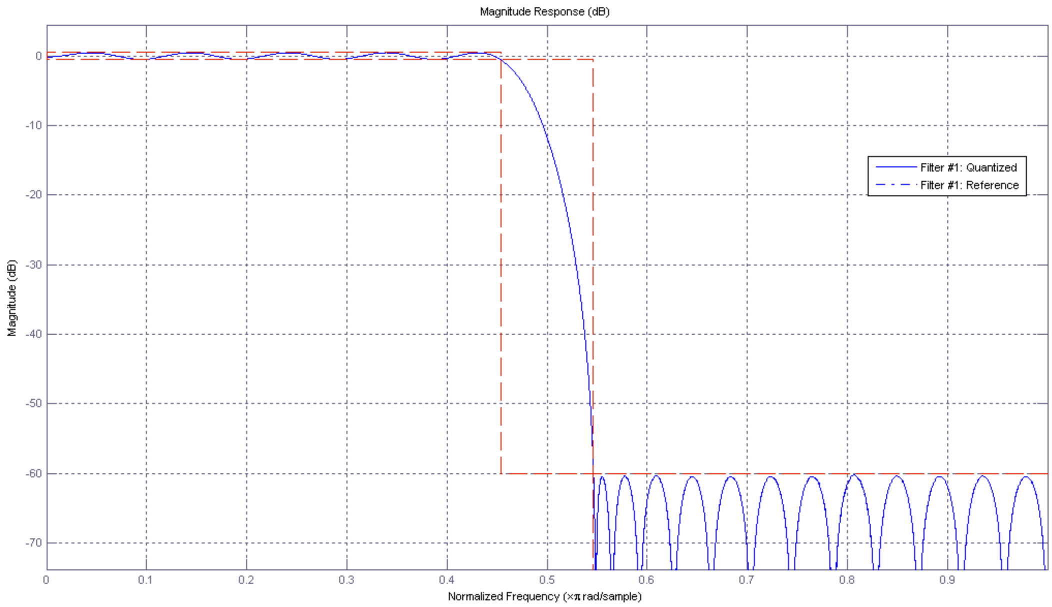
The first idea to get rid of the multiplier would be to use the subfilters approach used in the second stage of the interpolation filter proposed by Saramäki[1]. As it was mentioned before a filter designed with this approach will be structured as a combination of identical repetitions of a subfilter which will be simple with low hardware requirements. This will allow to get rid of the multiplier and will save hardware, reduce the current consumption and reduce the area needed in the microchip. However, there is always a tradeoff between how ideal the frequency response of a filter is and how hardware saving this filter is. As it was explained the second stage of the filter has to suppress the second repetition of the frequency spectrum of the input which is situated at 2fs, but the first stage of the filter has to suppress the first repetition situated in fs. This means that the first filter has to be sharper, and for that reason more hardware demanding than the second stage. Therefore, it is possible that the hardware efficient filter proposed by Saramäki, despite being sharp enough to suppress the second image, it cannot suppress the first image because it is too close to the original frequency spectrum of the input. This approach should definitely be tested, but in case that the specifications cannot be satisfied with it, other solutions should be considered.

6.4.2 IIR filter

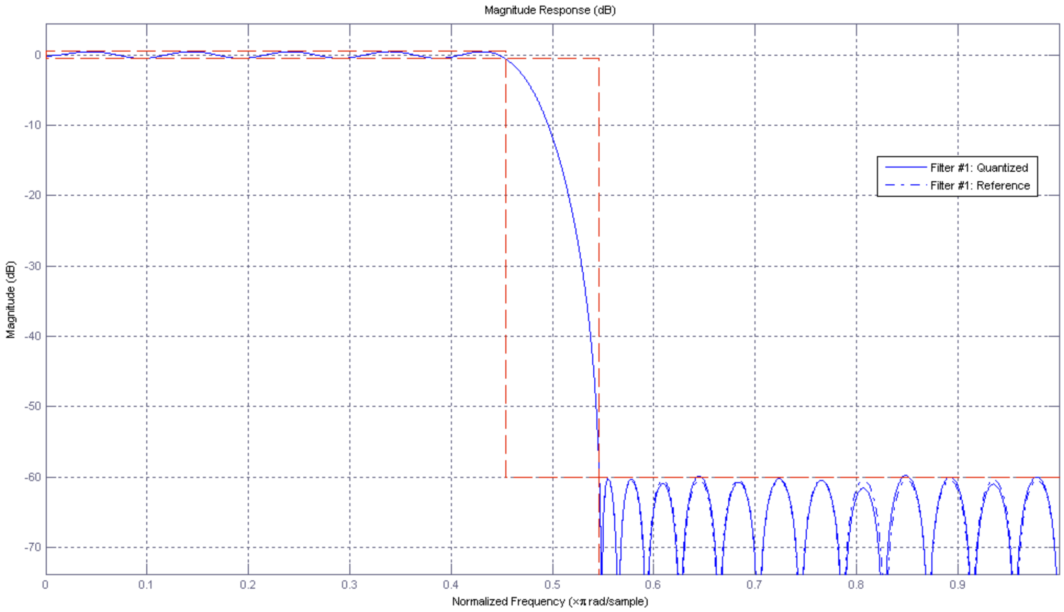
Another approach that could be used would be to design an IIR filter. Using IIR filters the quantization of the filter coefficients can be lowered to even less bits than FIR filters. This fact can easily be seen with the example. In Figure 6.7 a FIR low pass filters with the following characteristics can be seen:

- Passband cutoff frequency: 0.4536 (normalized to fs)
- Stopband cutoff frequency: 0.5464 (normalized to fs)
- Attenuation in the stopband: -60dB

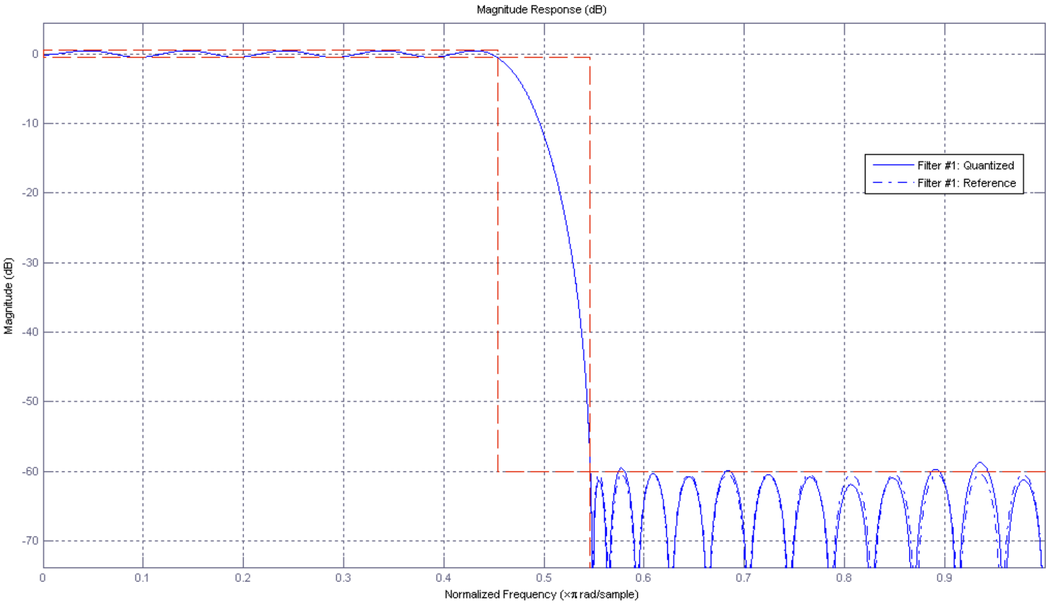
An IIR filter with the same characteristics can be seen in Figure 6.8. Different output quantizations have been used in order to see the effect on both FIR and IIR filter. As it can be seen, the transfer function of the FIR filter starts differing quickly from the full precision transfer function, while the transfer function of the IIR filter remains mostly exactly until the number of bits in the output is significantly reduced. The difference between the full precision and quantized transfer function of the FIR filter starts to be noticeable when the output number of bits is around 13. In the IIR filter case, the quantization can go down to 5 bits, and the effect on the transfer function will still be minimal.



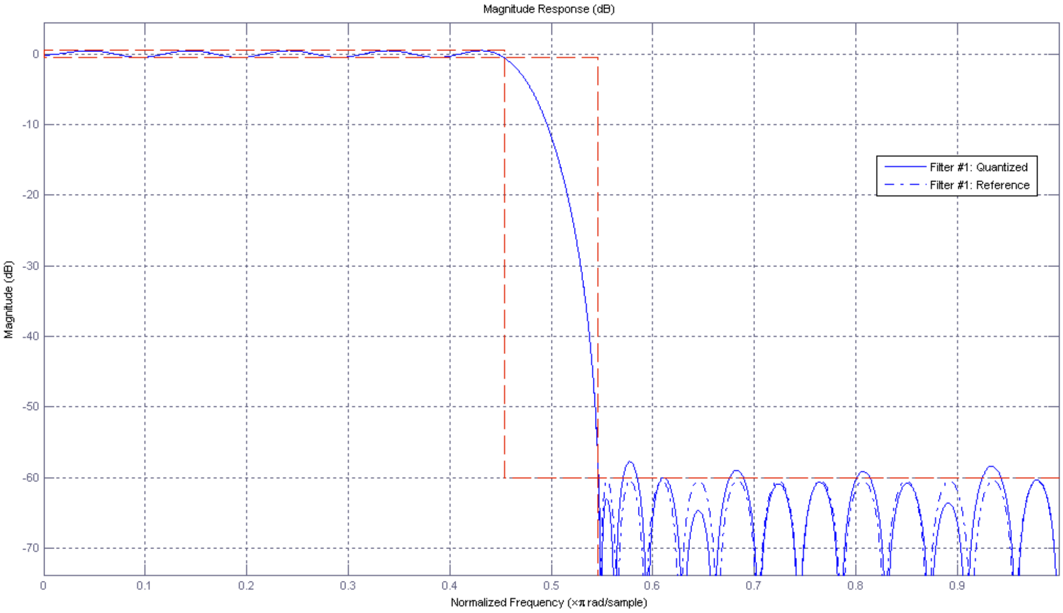
(a)



(b)

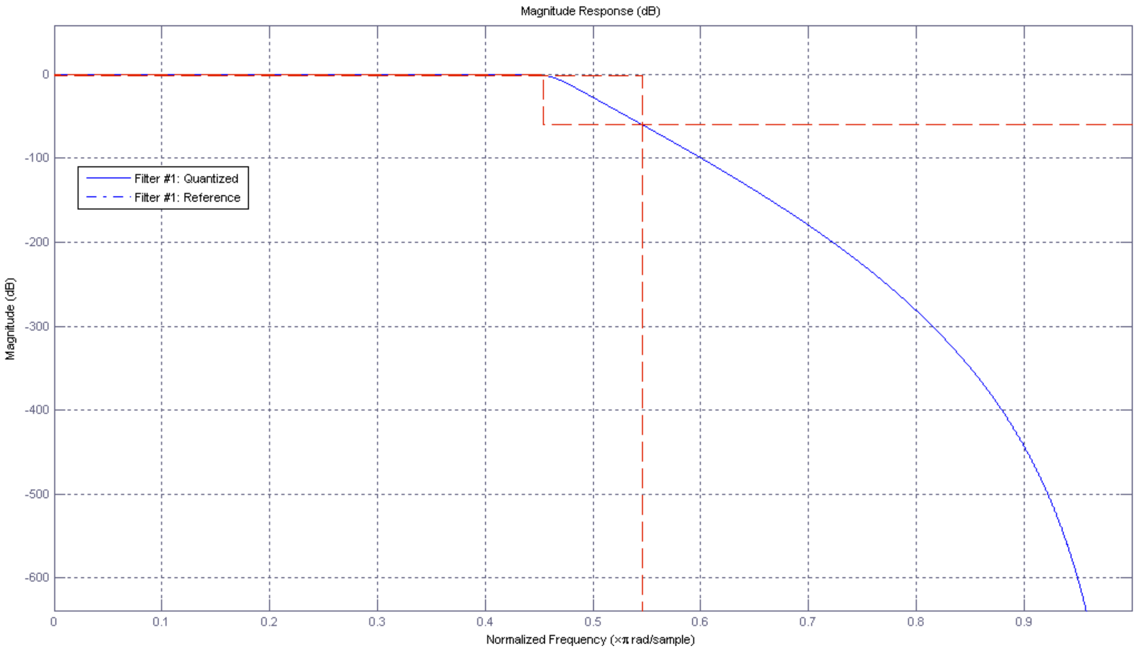


(c)

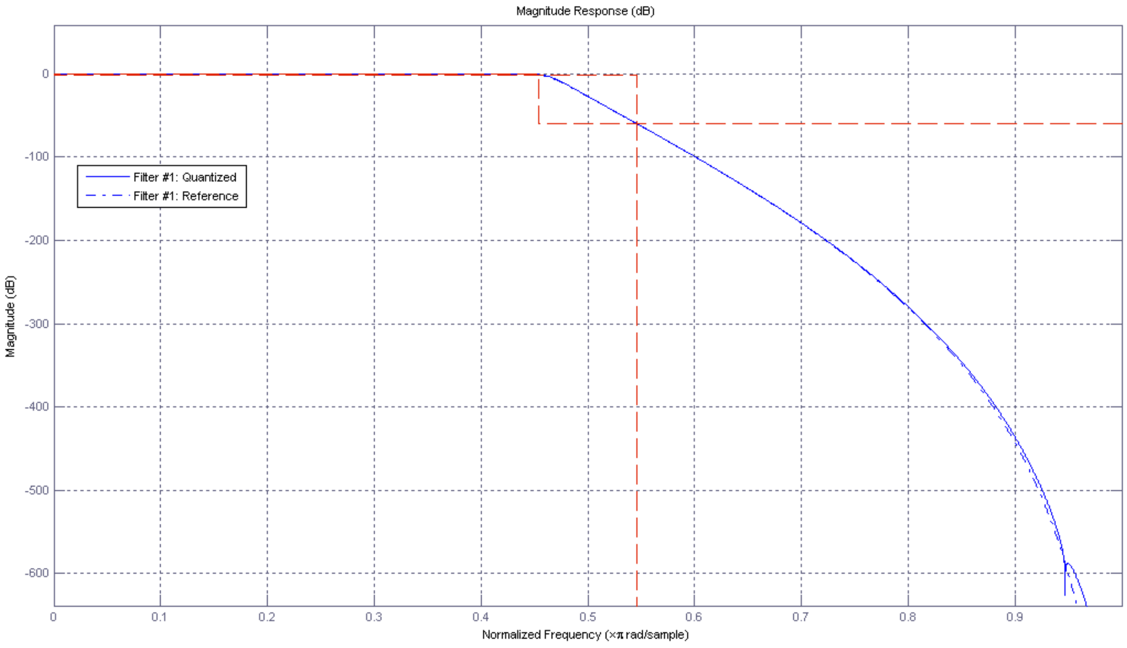


(d)

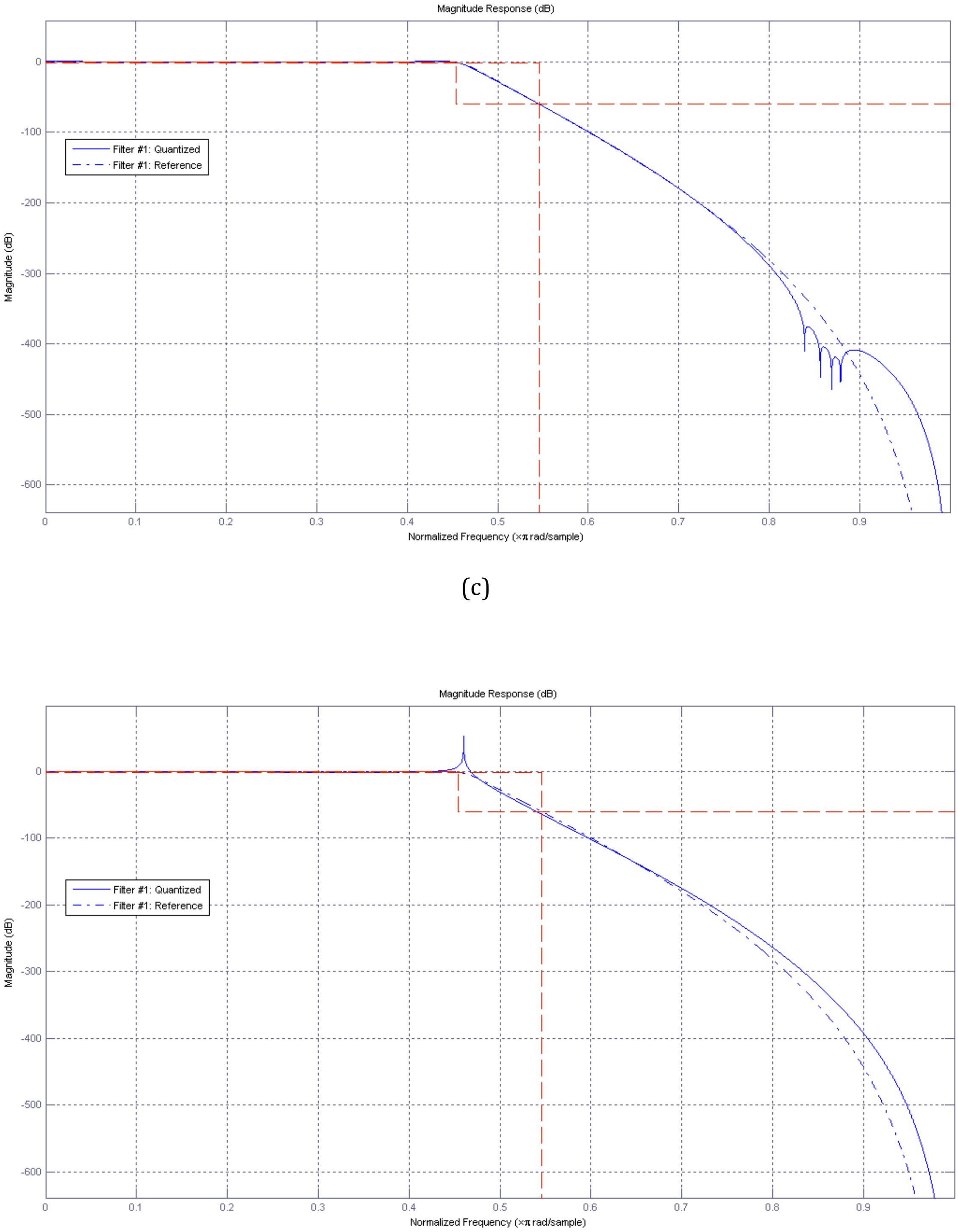
Figure 6.7: FIR low pass filter with an output quantization down to: (a) 18 bits; (b) 15 bits; (c) 14 bits; (d) 13 bits.


Magnitude Response (dB)
Magnitude (dB)
0
-100
-200
-300
-400
-500
-600
Filter #1: Quantized
Filter #1: Reference
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
Normalized Frequency (×π rad/sample)

(a)


Magnitude Response (dB)
Magnitude (dB)
0
-100
-200
-300
-400
-500
-600
Filter #1: Quantized
Filter #1: Reference
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
Normalized Frequency (×π rad/sample)

(b)



(c)

(d)

Figure 6.8: IIR low pass filter with the output quantization down to: (a) 18 bits; (b) 8 bits; (c) 5 bits; (d) 4 bits.

For that reason, if a IIR filter was used in the design, the filter coefficients quantization that was discussed before could even go down to less bits. With the number of bits reduced, then, the multiplier would not be needed anymore since a multiplication can be easily implemented as an addition together with shifts. The example of the multiplication of 4·7 will be used to show the process. Expressing the 4·7 multiplication as an addition of 4 multiplied by 2 powered numbers the Eq.(6.2) can be found. As it is known, in binary, a multiplication by 2^n is equivalent as shifting the number n times to the left, which means that each of the multiplications in Eq.(6.2) can be expressed as the addition of the number 4 shifted a certain amount of times. In this example the overall multiplication can be expressed by just adding 4, 4 shifted one time to the left, and 4 shifted two times to the left Eq.(6.3). The result can be seen in Eq.(6.4), and it is effectively 28, the original result.

$$4 \cdot 7 = 4 \cdot (1 + 2 + 4) = 4 \cdot 1 + 4 \cdot 2 + 4 \cdot 4 = 4 \cdot 2^0 + 4 \cdot 2^1 + 4 \cdot 2^2 = 28 \qquad (6.2)$$

$$4 \cdot 2^0 + 4 \cdot 2^1 + 4 \cdot 2^2 = shift_0(4) + shift_1(4) + shift_2(4) = 00100 + 01000 + 10000 \quad (6.3)$$

$$00100 + 01000 + 1000 = 111000 = 28 \qquad (6.4)$$

Obviously this process could also be done with a higher amount of bits like in the FIR filter designed (18 bits for the coefficients and 16 bits for the input samples) but then the implementation becomes too complicated to use this approach. This forces the FIR filter to have a multiplier. As it was said, since the IIR filters are less sensitive to quantization the number of bits used for the coefficients could be reduced enough to use this multiplier-less approach.

Using IIR filters would also make the design more complicated since the IIR filters are less stable than FIR filters but nowadays there is enough knowledge about IIR filters that simplify the stability control.

It is also worth to consider the fact that IIR filters have non-linear phase even though a good approximation of linear phase can be achieved. However the subjective effect of the linear phase and the approximated linear phase in human hearing should be tested in order to find out if it is a critical factor. Two interpolation filters with the same specifications, one with finite impulse response and another with infinite impulse response, would be designed and tested with real human subjects. As it was said before the IIR filters will be more hardware saving, so if this study shows that the non-linearity of the IIR filters is not hearable for humans, the filter should definitely be approached as an IIR filter.

6.5 CSD coefficients

This last approach of expressing the multiplication as shifts and additions can even be simplified if the numbers are expressed in canonic signed digits (CSD)]10]. Assuming 5 bits, the binary number representation of number 15 is 01111 as it can be seen in Eq.(6.5). Each bit x_i (i=0, 1, 2, 3, 4) determines if the 2^i factor will be added in the overall number. The CSD representation allows the bits x_i to represent -1, 0 or 1, which basically allows the subtraction of the factors 2i. Using CSD representation the number 15 can be represented as $1000\overline{1}$, Eq.(6.6). The number of additions has been reduced from four to two, which means that the expression of the number of 15 has been simplified.

$$15 = 0 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = 2^3 + 2^2 + 2^1 + 2^0 \qquad (6.5)$$

$$15 = 1 \cdot 2^4 + 0 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + (-1) \cdot 2^0 = 2^3 + (-2^0) \qquad (6.6)$$

Now, an example will be used to show the CSD representation implications in the multiplications that the filter designed needs to perform. The multiplication 10·15 can be expressed with:

$$10 \cdot 7 = 10 \cdot 2^0 + 10 \cdot 2^1 + 10 \cdot 2^2 + 10 \cdot 2^3$$

Generically, it could be said that the multiplication of 10 by a five bits number $x = x_4 x_2 x_1 x_0$ can be expressed as:

$$10 \cdot x = x_0 \cdot (10 \cdot 2^0) + x_1 \cdot (10 \cdot 2^1) + x_2 \cdot (10 \cdot 2^0) + x_3 \cdot (10 \cdot 2^3) + x_4 \cdot (10 \cdot 2^4)$$

All the bits x_i $(i = 0,1,2,3,4)$ are 0 or 1, but with CSD representation those bits could be -1, 0 or 1. This means that the multiplication of 10·15 could be expressed like:

$$10 \cdot 15 = -1 \cdot (10 \cdot 2^0) + 0 \cdot (10 \cdot 2^1) + 0 \cdot (10 \cdot 2^0) + 0 \cdot (10 \cdot 2^3) + 1 \cdot (10 \cdot 2^4) =$$

$$= -1 \cdot (10 \cdot 2^0) + 1 \cdot (10 \cdot 2^4)$$

The implications with this new representation regarding the implementation is that the multiplications by zero don’t require any component to be implemented. Therefore, just the multiplications by 1 or -1 will require hardware in its implementation. This means that by reducing the number of ones in the representation of the number 15, hardware will be saved. That’s the aim of the CSD representation, to reduce the number of ones in the expression of a number. It’s worth to notice that the multiplication by -1 is equal to a multiplication by 1 and then the subtraction of the value.

By reducing the number of multiplications by one, also the number of additions needed for the calculation of the result of the multiplication is decreased. The original number of additions was three, and now the number of additions is just two. The CSD representation has saved, in this case, the performance of one addition.

The idea will be to apply this CSD representation to the filter coefficients. This way, the implementation of the multiplication of the coefficients by the input samples will have fewer nonzero multiplications, and also fewer adders. The filter that was designed performs 38 multiplication and stores 19 coefficients, therefore the hardware saving effect generated by the expression of the filter coefficients in CSD will be even amplified.

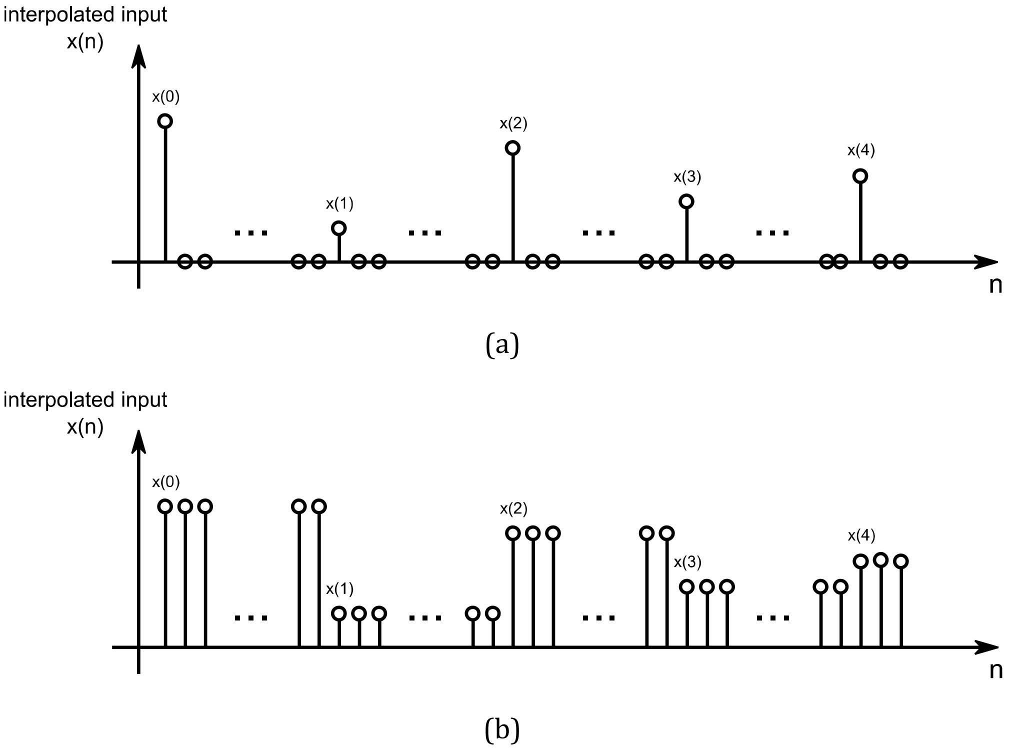
6.6 Discussion of the image suppression

Another interesting point that should be considered and discussed is the image suppression. The whole interpolation filter is designed to multiply the input sample frequency and to suppress all the 32 frequency spectrum repetitions generated by the sample frequency increment. These frequency spectrum repetitions are situated in the frequencies shown in Table 6.1.

Number of repetition	Frequency
1	fs=22.05 kHz
2	2·fs= 44.1 kHz
3	3·fs= 66.15 kHz
...	
31	31·fs= 683.55 kHz
32	32·fs= 705.6 kHz

Table 6.1: Frequencies where the images are situated.

The frequency range of human hearing is up to 20 kHz which means that theoretically talking none of these frequency spectrum repetitions is hearable, so if the interpolation filter is used for hearing aids, the attenuation of these images seems to be pointless. Even if the images are not filtered the frequency still has to be incremented 64 times, which means that more samples should be interpolated between the original ones. The two most obvious options are to interpolate zero samples Figure 6.9(a), or to interpolate the last sample input value Figure 6.9(b).



(a)

(b)

Figure 6.9: (a) Zero samples interpolated; (b) Last input sample input value interpolated.

If the signal that should be heard was directly the output of the interpolation filter, filtering or not filtering the images wouldn't really matter since as it was said they wouldn't be hearable. However, after the interpolation filter there is a sigma-delta modulator, a digital pulse width modulation a class-D output stage, a feedback chain and an output low pass filter. The non suppressed images will also have to be processed by all those stages and it is possible that part of those non images will be thrown to the hearable band and will distort the normal hearing. One of the problems that might be expected is the downfold of a high frequency component generated by the nonlinearities of the class-D amplifier. The downfold will occur if a high frequency component is added by one of those posterior stages of the back end. Since it is a discrete signal, this high frequency component will also be repeated in all the bands, including the band of interest. This high frequency component will distort the sinusoidal input signal, and will be hearable since it will be contained in the hearable human band.

Both interpolations proposed in Figure 6.9 should be tested and should be studied since in case that not any part of the images is thrown into the hearable band width by the sigma-delta modulator or the class-D end stage, the hardware reduction would be very substantial and the current consumption would be reduced as well. The most reasonable assumption here is that probably just the first image should be attenuated since it is situated at fs=22.05 kHz which is quite close to 20 kHz. In that case just the first step of the interpolation filter would be needed.

References

[1] Richard G. Lyons, 'Understanding digital signal processing – 3rd ed.', Prentice Hall, Pearson Education, April 2011.

[2] Pong P. Chu, 'RTL Hardware design using VHDL', John Wiley & Sons, 2006.

[3] R. Schreier, G. C. Temes, "Understanding Delta-Sigma Data Converters,"Wiley-IEEE Press, Prentice Hall, November 2010.

[4] Pong P. Chu, 'FPGA Prototyping by VHDL examples, Xilinx Spartan – 3 version', John Wiley&Sons, 2008.

[5] Tapio Saramäki, 'Design of FIR Filters as a Tapped Cascaded Interconnection of Identical Subfilters,' IEEE Transactions on Circuits and Systems, vol-cas 34, no. 9, September 1987.

[6] B. A. Shenoi, 'Introduction to digital signal processing and filter design', John Wiley&Sons, 2006.

[7] Julius O. Smith III, 'Introduction to digital filters with audio applications', W3K Publishing, October 2007.

[8] R. G. Lyons, 'Introduction to Digital Signal Processing', Prentice Hall, November 2010.

[9] M. Garg, R. Kumar Bansal, S. Bansal, 'Reducing Power Dissipation in FIR Filters: An Analysis', Signal processing: An International Journal (SPIJ), Volume(4): Issue(1).

[10] T. Zhangwen, 'A high-speed, programmable, CSD coefficient FIR filter', Costumer Electronics, IEEE Transactions on, November 2002.

List of Figures

List of Tables