Library Rights Statement

In presenting the thesis *Digital Decimating Filter for a Monolithic Sonar Receiver* in partial fulfillment of the requirements for an advanced degree at the University of Rhode Island, I agree that the Library shall make it freely available for inspection. I further agree that permission for copying as provided for by the Copyright Law of the U.S. (Title 17, U.S. Code) of this thesis for scholarly purposes may be granted by the Librarian. It is understood that any copying or publication of this thesis for financial gain shall not be allowed without my written permission.

I hereby grant permission to the URI Library to copy my thesis for scholarly purposes.

Roger S. Meier

Date

DIGITAL DECIMATING FILTER FOR A MONOLITHIC SONAR RECEIVER

BY

ROGER S. MEIER

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE
IN
ELECTRICAL ENGINEERING

UNIVERSITY OF RHODE ISLAND
2000

MASTER OF SCIENCE THESIS

OF

ROGER S. MEIER

APPROVED:

Thesis Committee

Major Professor

_____

_____

_____

_____

DEAN OF THE GRADUATE SCHOOL

UNIVERSITY OF RHODE ISLAND

2000

Abstract

This thesis describes the synthesis and the implementation of a digital decimating filter for a monolithic sonar receiver which uses a delta-sigma modulator to acquire the received acoustic signal. Various MATLAB® scripts (m-files) have been written to aid filter design and to analyze functionality and performance. A hardware simulation for the digital decimating filter is introduced and its performance is verified by a dedicated MATLAB® simulation. The system is implemented by FPGA technology (XILINX XC4000 series), and a behavioral as well as a structural timing simulation confirm the system's performance.

Acknowledgments

I would like to thank the members of my committee, Dr. Peter F. Swaszek, Dr. Conrad W. Recksieck, and the chairperson of my defense, Dr. David R. Heskett, for their time and concern.

I am grateful to my major Professor, Dr. Godi Fischer, for what I have learned at URI, the guidance through my studies, and the time he has spent reviewing the contents of this thesis.

Special thanks to my project team members, Dr. Alan J. Davis and Xiaodan Wang, for helpful discussions and sharing their experience with me.

For the understanding, patience, love and support, and sharing the experience of studying in a foreign country with me, I am grateful to my wife Rosalinda. May we be able to continue to share all our undertakings, however large or small they may be.

Contents

List of Tables

List of Figures

To Rosalinda

Chapter 1

Introduction

Sonar has been used for object detection, including fish finding, for a long time. There are many target-like noise sources in the ocean, such as human-made and biologic noise. In order to be able to determine target direction and shape, a large amount of data is necessary to be generated and processed by a multi-beam sonar system, which heavily relies on the fast development of digital signal processing and related semiconductor technologies today.

A typical multi-beam sonar system consists of a large sensor array followed by a digital beamformer that spatially analyzes the received data. Figure 1.1 shows the block diagram of such a system. In order to process the sensors' data values digitally, the signals are first converted into the digital domain. Before the received signals can be digitized, they have to be amplified and lowpass-filtered in order to avoid

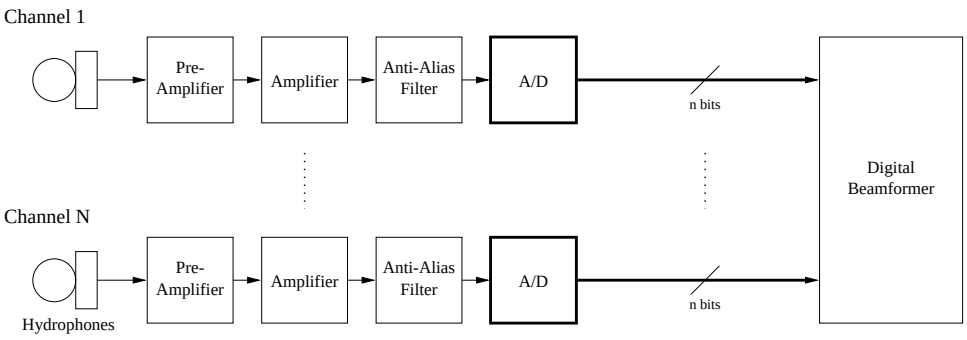


Figure 1.1: Conventional approach of a sonar receiver array.

aliasing.

In the past, the conversion to the digital domain has been accomplished by off-the-shelf analog-to-digital (A/D) converters with multi-bit outputs, which subsequently have been fed into the digital beamformer. For systems with a higher number of channels (say more than 50), issues like hardware cost, dimensionality, and electrical problems gain importance.

A more cost and space effective way of implementing the analog-to-digital conversion is to use Delta-Sigma ($\Delta\Sigma$) modulators that are specially designed to acquire the sensors' output signals. The modulators' pulse density modulated (PDM) output comprises the oversampled input signals and wideband noise (off-band), which stems from the quantization process. In order to obtain wideband output signals, which can be further processed by either a beamformer or additional filters for narrowband beamforming, the $\Delta\Sigma$ modulator output signals have to be digitally low-pass filtered and decimated. Figure 1.2 shows one channel of such a system in form of a block diagram.

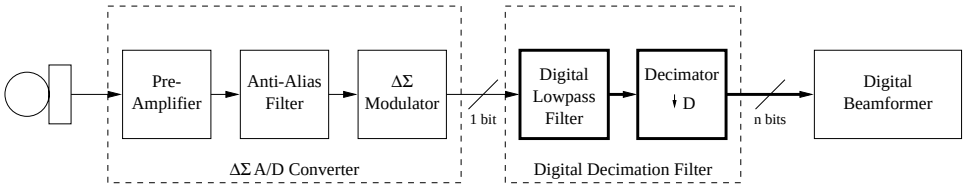


Figure 1.2: Sonar receiver employing a $\Delta\Sigma$ modulator and a digital decimation filter.

The purpose of this thesis is to design and implement a digital decimation filter for a monolithic sonar receiver.

1.1 Organization

This thesis is split into five chapters. In the next chapter, the necessary background is reviewed. We provide the underlying theory behind $\Delta\Sigma$ modulators, and fundamentals of digital filter design. The considerations are focused on realization

concepts for linear-phase FIR filters, and the impact of coefficient quantization on the performance of these filters.

In Chapter 3, we will discuss the design of a linear-phase FIR filter based on specifications derived from the requirements of the sonar receiver. After designing the filter using the Signal Processing Toolbox [1], we will verify the performance with MATLAB® simulations. We will then develop a basic hardware concept for the filter. We will show that, because of the 1-bit format of the input signal and the fact that the output signal is decimated, the hardware effort can be significantly reduced. With a script written in MATLAB®, we will perform a behavioral simulation of the filter impulse response based on the developed hardware concept.

In Chapter 4, an FPGA implementation of the digital decimation filter is proposed. The basic building blocks are explored in detail, and simulations are performed to observe the behavior of the implementation. By simulating the impulse response of the filter and comparing the results to the simulations obtained in Chapter 3, we will be able to verify the correct function of the implementation.

Chapter 5 will summarize the resulting work and explain the steps toward a commercialized version of the digital decimation filter.

Chapter 2

Theoretical Background

2.1 $\Delta\Sigma$ Modulators

2.1.1 Introduction

A $\Delta\Sigma$ modulator is an efficient oversampling quantizer used to convert signals from the analog to the digital domain. Figure 2.1 shows (a) a block diagram of a first-order $\Delta\Sigma$ modulator and (b) its discrete-time linearized equivalent. The input to the circuit in Figure 2.1a is brought to the quantizer via an integrator, and the quantized output is fed back and subtracted from the input. This feedback forces the average value of the quantized signal to track the average value of the input. Any difference between these two signals accumulates in the integrator and eventually corrects itself. Figure 2.2 illustrates the response of the circuit to a sinusoidal input; it shows how the quantized signal $y(t)$ oscillates between two levels that are adjacent to the input in such a manner that its local average equals the average value of the input. This modulated output, which is quantized to one bit, represents a pulse density modulated (PDM) version of the input signal (in this case a sine function). The time granularity at the output is provided by a certain oversampling ratio (OSR), defined as half the sampling rate divided by the signal bandwidth.

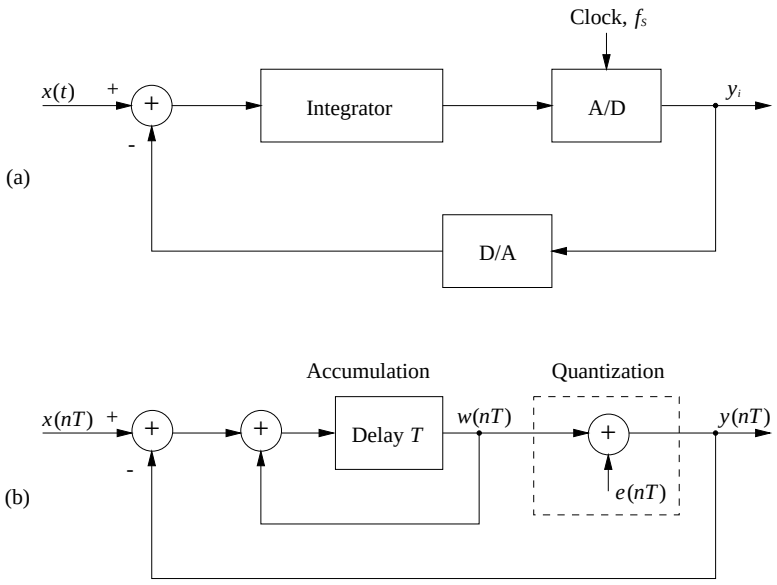


Figure 2.1: (a) Block diagram of a first-order $\Delta\Sigma$ modulator and (b) its discrete-time linearized equivalent circuit.

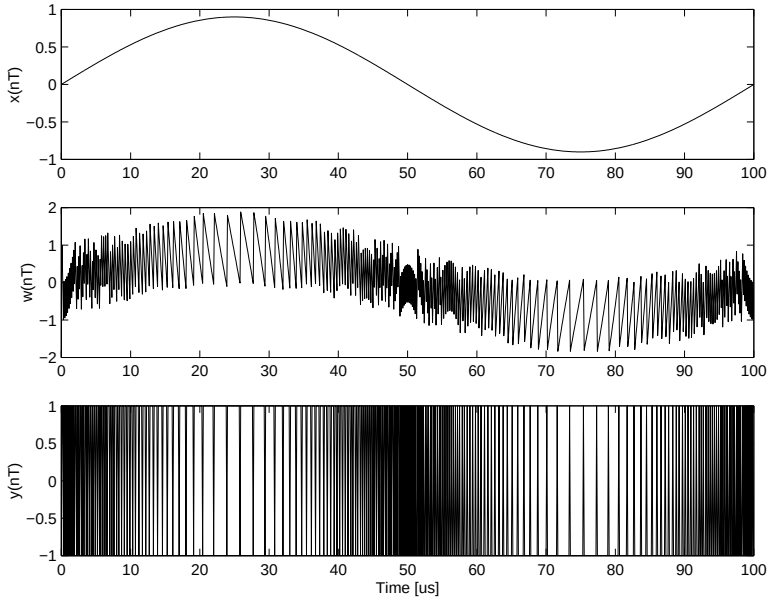


Figure 2.2: Time domain response of a first-order $\Delta\Sigma$ modulator with one bit quantization to a 10 kHz sinusoidal input signal (MATLAB simulation with $f_s = 10$ MHz).

2.1.2 Noise Shaping

The most prominent feature of a $\Delta\Sigma$ modulator is its capability to shape the quantization noise by employing oversampling. This can be analyzed using the linearized first order modulator shown in Figure 2.1b, where the quantizer has been replaced by an additive noise source and the integration is represented by a discrete-time accumulation. However, the linear model breaks down when we ponder questions related to stability or the exact role of quantization noise. The output of the accumulator and the quantized signal can be described by the following difference equations

$$w[n+1] = w[n] + x[n] - y[n] \tag{2.1}$$

$$y[n] = w[n] + e[n], \tag{2.2}$$

where the cycle time of the sampling frequency is equated to unity. Transforming these equations into the z-domain yields:

$$W(z) = [W(z) + X(z) - Y(z)]z^{-1} \tag{2.3}$$

$$Y(z) = W(z) + E(z). \tag{2.4}$$

By eliminating W(z) and solving for Y(z), we obtain

$$Y(z) = X(z)z^{-1} + E(z)(1 - z^{-1}). \tag{2.5}$$

According to this result, the modulator digitally differentiates the quantization error while leaving the signal unchanged, except for a delay. Thus, the modulator can be characterized by two transfer functions: a signal transfer function (STF) defined as the ratio Y/X (in the absence of noise) and a noise transfer function (NTF) defined as Y/N (setting $X = 0$).

2.1.3 Output Noise Power

We next want to calculate the effective resolution of the $\Delta\Sigma$ modulator. The NTF can be written after substituting the variable z by its frequency domain equivalent $e^{j2\pi f/f_s}$ and inserting Euler's formula for the sine function by

$$|NTF(f)| = \left|1 - e^{j2\pi f/f_s}\right| = 2\ sin\left(\frac{\pi f}{f_s}\right). \tag{2.6}$$

If the quantization error represents the dominant system noise source, the total mean square in-band noise at the output of the modulator loop can be computed as

$$n_0{}^2 = e_q{}^2 \frac{2}{f_s} \int_0^{BW} |NTF|^2 \, df, \tag{2.7}$$

where BW represents the effective bandwidth, and e_q^2 corresponds to the input noise power. When we replace the signal bandwidth BW by the ratio $\frac{f_s}{2\ OSR}$, the expression is more practical and depends on three parameters only. Furthermore, since $f/f_s \ll 1$ in the passband region of the modulator, we can approximate the sine function by its argument $(\pi f/f_s)$. The in-band noise power present at the output can now be written as

$$n_0{}^2 = e_q{}^2 \frac{2}{f_s} \int_0^{f} |NTF(f)|^2 \, df \approx e_q{}^2 \frac{\pi^2}{3} \left(\frac{2f}{f_s}\right)^3, \tag{2.8}$$

and its rms value is

$$n_0 = e_q \frac{\pi}{\sqrt{3}} \left(\frac{2f}{f_s}\right)^{3/2}. \tag{2.9}$$

As we see from (2.9), the output noise power strongly depends on the OSR, beyond the quantization error. Each doubling of the oversampling ratio reduces the output noise power level by 9 dB and provides 1.5 bits of extra resolution. If we increase the order of the system by cascading m identical cells, the output noise power can even be further reduced [2].

2.1.4 Application: Monolithic Sonar Receiver

Several circuit arrangements are possible to improve the modulator noise shaping characteristic. The most frequently applied can be divided into two categories, namely multi-stage and single-stage modulators. To find a reasonable architecture for a specified problem many trade-offs have to be considered. The monolithic sonar receiver, for which the decimation filter discussed in this thesis is built, employs an IFLF5 (inverse follow-the-leader feedback) structure. The IFLF5 is a fifth-order modulator with 1-bit quantization where the output is fed back to every integrator input [2]. Its characteristics already have been simulated using DelSi [3], and thus

its description models could easily be reused for further simulations in MATLAB®. Figure 2.3 shows its simulated spectral response in presence of a 10 kHz sinusoidal input signal. The displayed spectrum has been computed by averaging four FFT's obtained from 2^{18} output samples, each shaped by a Kaiser-Bessel window. In this example the achieved signal-to-noise ratio (SNR) was 90 dB. It can be clearly seen that high noise levels only occur at out-of-band frequencies and that the signal band includes a harmonic.

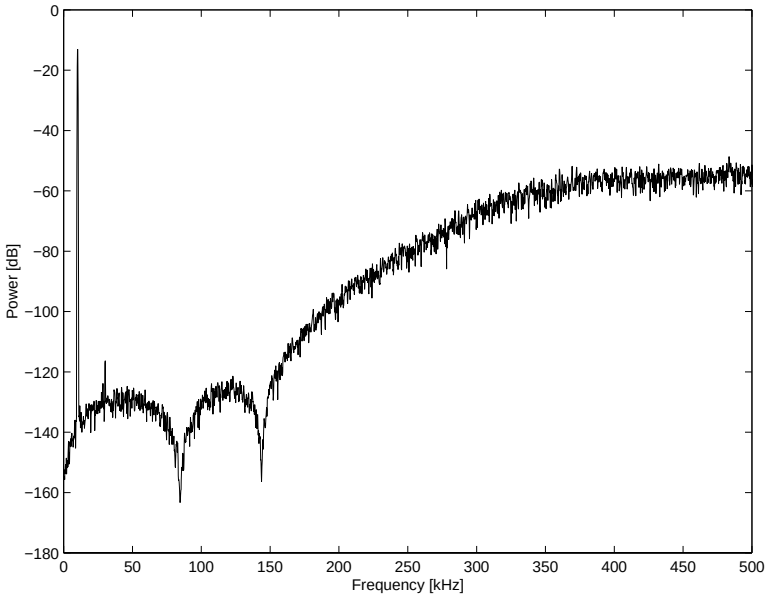


Figure 2.3: Spectral response of a IFLF5 modulator in presence of a 10 kHz sinusoidal input signal (DelSi simulation with $f_s = 10$ MHz).

2.2 Digital Decimation Filters

2.2.1 Introduction

The output of the modulator represents the input signal of the decimator including its out-of-band components, modulation noise, circuit noise, and interference. In order to have the signal available for further digital processing, the out-of-band energy of

this signal will be attenuated by a digital filter. Thereafter, it may be resampled at Nyquist rate without incurring a significant noise penalty because of aliasing. Figure

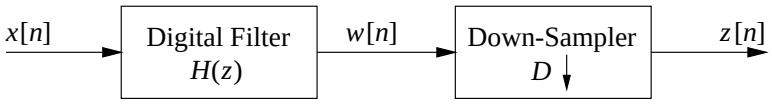


Figure 2.4: Block diagram of digital decimation filter with sampling rate reduction by a factor of D.

2.4 shows a block diagram of a digital decimation filter, sometimes referred to as a down-sampling filter. First, the signal is fed into a digital low-pass filter which approximates the ideal characteristic

$$H(e^{jw}) = \begin{cases} 1, & |w| \leq 2\pi f_D T/2 = \pi/D \\ 0, & \text{otherwise} \end{cases}, \tag{2.10}$$

where f_d denotes the new sampling rate f/D. The sampling rate reduction can then be achieved by forming the sequence $y[m]$ by extracting every Dth sample of the filtered output. If we denote the actual low-pass filter impulse response as $h[n]$ the filtered output $w[n]$ can be written as

$$w[n] = \sum_{k=-\infty}^{\infty} h[k]x[n-k], \tag{2.11}$$

and the final output $y[m]$ is

$$y[m] = w[Dm]. \tag{2.12}$$

Combining (2.11) and (2.12) leads to

$$y[m] = \sum_{k=-\infty}^{\infty} h[k]x[mD-k]. \tag{2.13}$$

Due to the out-of-band components of the modulated signal, abrupt low-pass filters are needed. Such filters are expensive to build for elevated sampling rates and thus the implementation has to be analyzed carefully. Kusch [4] investigates several filter structures and shows their trade-offs.

Since it is crucial in beamforming applications to preserve the phase information of received signals, the decimation filter has to yield a constant group delay for all frequencies, i.e., linear phase. Due to their nonrecursive structure FIR filters are always stable and, if the coefficient sets are symmetric, provide linear phase.

2.2.2 Properties of Linear-Phase FIR Filters

In general a digital filter is completely characterized by its difference equation which describes the relationship between input and output. The difference equation of a generic digital filter is given by

$$\begin{aligned} y[n] &= a_0x[n] + a_1x[n-1] + a_2x[n-2] + \cdots + a_Lx[n-L] \\ &\quad -b_1y[n-1] - b_2y[n-1] - \cdots - b_My[n-M]. \end{aligned} \tag{2.14}$$

Since a FIR filter is a nonrecursive system it comprises no feedback terms. It is characterized by only its impulse response $h[n]$. Hence, the difference equation for an N-th order FIR filter is given by

$$y[n] = h[0]x[n] + h[1]x[n-1] + h[2]x[n-2] + \cdots + h[N]x[n-N], \tag{2.15}$$

which can also be written as a finite convolution of the input signal $x[n]$ and the impulse response $h[n]$

$$y[n] = \sum_{k=0}^{N} h[k]x[n-k]. \tag{2.16}$$

The transfer function (2.17) can be obtained by taking the z-transform of the impulse response $h[n]$

$$H(z) = \sum_{n=0}^{N} h[n]z^{-n}. \tag{2.17}$$

To produce a linear phase response the constraint is simply that the finite-duration impulse response has conjugate-even or conjugate-odd symmetry about its midpoint. To see that this constraint ensures linear phase, consider the FIR system function (2.17) with

$$h[n] = \pm h^*[N-n] = |h[n]|e^{j\phi_n}.$$

If N is even, the coefficient $h[\frac{N}{2}]$ is real and corresponds to the center of symmetry of $h[n]$, while if N is odd, there is no central coefficient. These four cases are illustrated in Figure 2.5 for $h[n]$ real, we then have

$$h[n] = \pm h[N - n].$$

Note that type I and II filters have even symmetry about their midpoints, while type III and IV filters have odd symmetry. The type I and III filters have even order (N), while types II and IV have odd order.

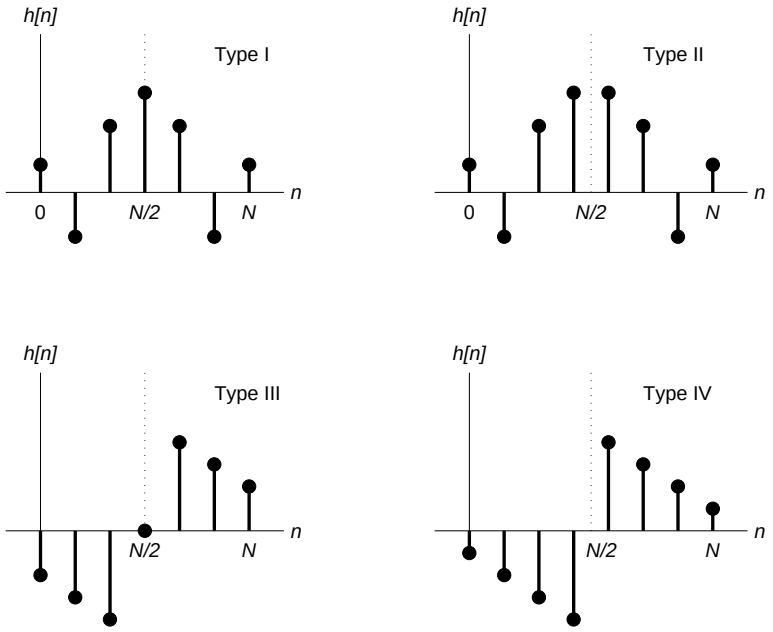


Figure 2.5: Four cases of even/odd symmetry about $\frac{N}{2}$ for real-valued, linear-phase FIR filters with even or odd order (N).

By applying a constant input $x[n] = 1$ to these filters, we observe that odd symmetry implies zero output for all DC inputs

$$y[n] = \sum_{k=0}^{N} h[k] = H(1) = 0 \qquad \text{(for types III and IV)}.$$

Hence, $H(z)$ must have a zero at $z = 1$ for types III and IV filters. On the other hand, by applying a Nyquist-frequency sequence $x[n] = (-1)^n$ to these filters, types

II and III will have zero outputs because

$$y[n] = \sum_{k=0}^{N} h[k](-1)^{n-k} = H(-1) = 0 \qquad \text{(for types II and III)},$$

and thus $H(z)$ has a zero at $z = -1$. These constrained zeros at $z = \pm 1$ for real-values linear-phase FIR filters are shown in Figure 2.6. Because of the given zero

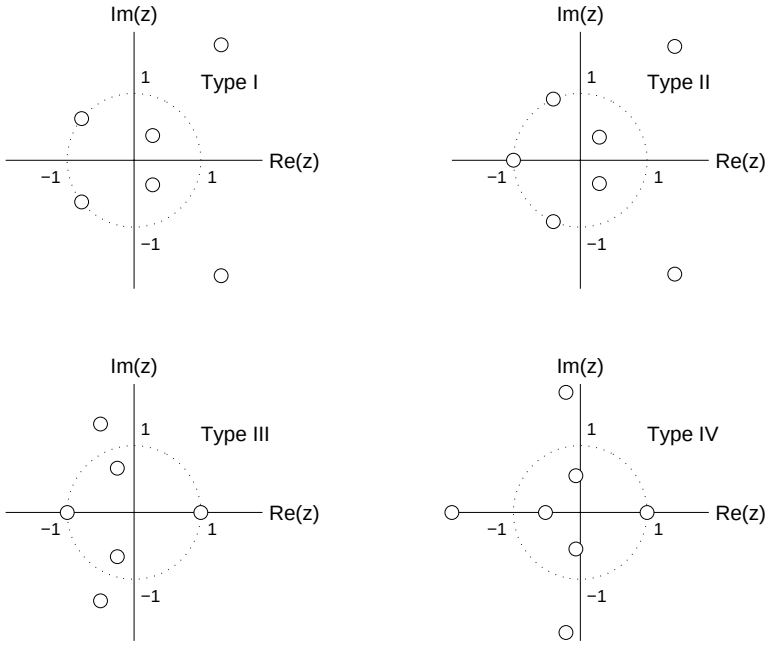


Figure 2.6: Zero locations for the type I, II, III, and IV linear-phase filters from Figure 2.5 showing required zeros at $z = \pm 1$.

locations it is not possible for all filter types to have an arbitrary magnitude response. While only type I filters can be used to realize all-pass filters, type II filter would be employed to synthesize low-pass filters. Because both type III and IV filters have zero output for DC-inputs, they can not be used to implement low-pass functions. Since type III filters also have a zero at the Nyquist-frequency, they are suitable to build band-pass filters, while type IV filters can be used to realize high-pass filters.

A further effect of the linear-phase constraint on the zeros of $H(z)$ is seen by

noting from (2.17) that

$$z^N H(z) = \pm H^*(\frac{1}{z^*}) \tag{2.18}$$

because of the assumed symmetry in the impulse response $h[n]$. Equation (2.18) implies that the zeros of $H(z)$ must also be zeros of $H^*(1/z^*)$, which means that if z_n is a zero of $H(z)$, then $1/z_n$ is also. Therefore, the zeros of a linear-phase filter either must lie on the unit circle or must occur in pairs with reciprocal radii. For $h[n]$ real, the zeros must also occur as complex conjugates. Hence, the zeros not lying on the unit circle or on the real axis will actually occur in quadruples, as shown in Figure 2.6.

To verify the linear phase-response let us consider the type I case with real coefficients of even symmetry and N even. We then may rewrite (2.17) as

$$\begin{aligned} H(z) &= z^{-\frac{N}{2}} \left(\sum_{n=0}^{N} h[n] z^{-n+\frac{N}{2}} \right) \\ &= z^{-\frac{N}{2}} \left(h[\frac{N}{2}] + \sum_{n=0}^{\frac{N}{2}-1} \left(h[n] z^{-n+\frac{N}{2}} + h[N-n] z^{n-\frac{N}{2}} \right) \right). \end{aligned} \tag{2.19}$$

Substituting $z = e^{j\omega}$ and taking into account that $h[n] = h[N-n]$, we find that the frequency response is given by

$$\begin{aligned} H(\omega) &= e^{-j\omega\frac{N}{2}} \left(h[\frac{N}{2}] + \sum_{n=0}^{\frac{N}{2}-1} 2h[n] \cos\left(\left(\frac{N}{2} - n \right) \omega \right) \right) \\ &= e^{-j\omega\frac{N}{2}} R(\omega). \end{aligned} \tag{2.20}$$

where $R(\omega)$ is purely real. If $R(\omega)$ is of constant sign for all ω, then $R(\omega) = \pm|H(\omega)|$ and we indeed have the linear phase response

$$\angle H(\omega) = -\omega\frac{N}{2} + C,$$

where $C = 0$ or π. If however, there are sign changes in $R(\omega)$, there are corresponding 180° phase shifts in $\angle H(\omega)$, and $\angle H(\omega)$ is only piecewise linear. In common practice, the filter is still referred to as having linear phase. This terminology is reasonable

since we are actually trying to constrain the filter's group delay $D(\omega)$ to be constant, and since

$$D(\omega) = -\frac{d}{d\omega}\angle H(\omega),$$

we have

$$D(\omega) = \frac{N}{2} \tag{2.21}$$

except at those frequencies where $R(\omega)$ changes sign. But since $R(\omega) = \pm|H(\omega)| = 0$ at those frequencies, there is no output contribution. Thus, the group delay is simply the delay to the midpoint of $h[n]$ (see Figure 2.5).

The more general proof that type I, II, III, and IV filters yield phase linearity for complex $h[n]$ can be found in [5].

2.2.3 Design of Linear-Phase FIR Filters using the Remez Exchange Algorithm

In [5] it is shown that filter design of lowest order satisfying a classical frequency-selective specification is a design, which has uniform ripples or extrema in the pass-band and the stopband. Such designs are known as *equiripple* filters. Since a closed-form solution is not available for these filters, an iterative design method needs to be applied. The Remez exchange algorithm is one of these methods.

As for other FIR design methods, the specification on the passband and stopband is assumed to be

$$\begin{aligned} 1 + \delta_1 \geq |H(\omega)| \geq 1 - \delta_1 \quad &, \quad 0 \leq \omega \leq \omega_c \\ |H(\omega)| \leq \delta_2 \quad &, \quad \omega_r \leq \omega \leq \pi \end{aligned} \tag{2.22}$$

as illustrated in Figure 2.7. For convenience let us assume that a type I filter impulse response is made symmetric about $n = 0$. Hence, $h[n] = h[-n]$, and we obtain the following frequency response

$$\begin{aligned} H(\omega) &= \sum_{n=-\frac{N}{2}}^{\frac{N}{2}} h[n]e^{-j\omega n} \\ &= h[0] + \sum_{n=1}^{\frac{N}{2}} 2h[n]\cos(n\omega). \end{aligned} \tag{2.23}$$

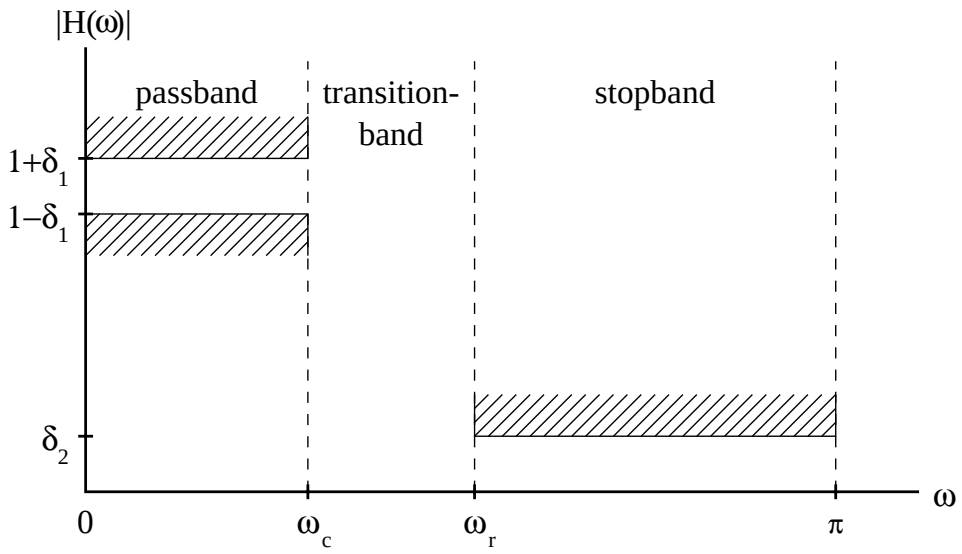


Figure 2.7: Magnitude response specification for FIR filter designs.

Thus, $H(\omega)$ is a real and even function, expressed as a finite Fourier cosine series with $h[n]$ as its corresponding coefficients. By writing $\cos(n\omega)$ as a trigonometric polynomial in $\cos(\omega)$ of order n, we obtain a sum of polynomials for $H(\omega)$, which is also a polynomial in $\cos(\omega)$ of order n

$$H(\omega) = \sum_{n=0}^{\frac{N}{2}} a_n(\cos(\omega))^n. \tag{2.24}$$

An important parameter in the Remez exchange algorithm is the total number of extrema in the Nyquist interval $[0, \pi]$. To determine this number we take the derivative of $H(\omega)$ and set it to zero. We then have

$$\frac{d}{d\omega}H(\omega) = 0 = -\sin(\omega)\sum_{n=1}^{\frac{N}{2}} n a_n(\cos(\omega))^{n-1}, \tag{2.25}$$

which consists of a trigonometric polynomial of order $\frac{N}{2} - 1$, and thus has at most $\frac{N}{2} - 1$ distinct roots. The factor $sin(\omega)$ implies solutions at 0 and π. Hence, there are $\frac{N}{2} - 1 + 2 = \frac{N}{2} + 1$ possible extrema of zero slope. Adding the two extrema at the band edges ω_c and ω_r, we have a maximum total number of possible extrema

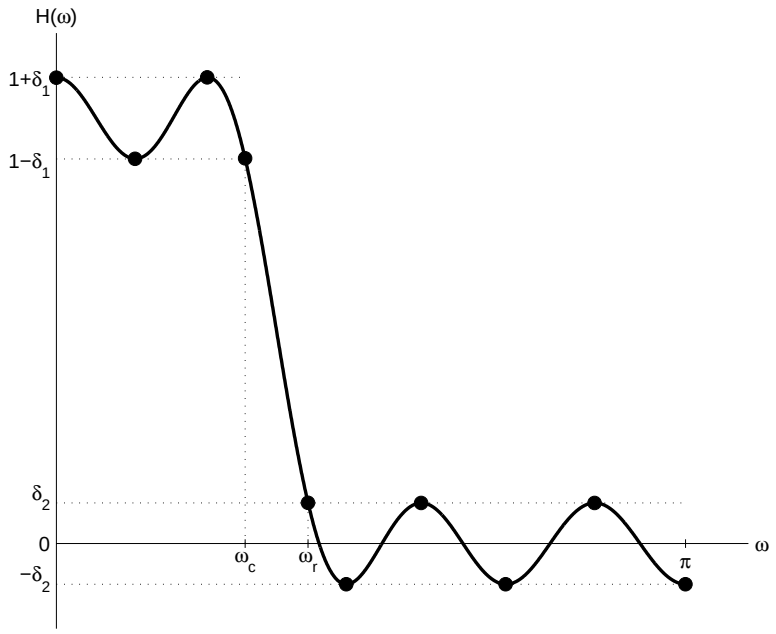


Figure 2.8: Sample equiripple design with dots (•) indicating the alternations (for $N = 16$).

of $\frac{N}{2} + 3$. A theorem from the theory of approximation states that the design is optimized for minimum ripple if, and only if, there are at least $\frac{N}{2} + 2$ extrema of equal (weighted) amplitudes and alternating signs in the pass and stop bands. Such extrema are known as *alternations*. Figure 2.8 shows a sample design for $N = 16$.

Since CAD software is readily available in MATLAB® [1] to design equiripple filters, we will only outline the general design formulations and the Remez exchange algorithm. Even though it can be closely estimated, we do not know a priori what minimum ripple amplitudes δ_1 and δ_2 can be achieved for a given order N. By applying a weight K [6] to the stopband specification, a single unknown $K\delta_2 = \delta_1 = \delta$ is produced, which is determined iteratively along with the filter coefficients. If the specifications can not be met, the order has to be increased and the algorithm repeated.

To describe the Remez exchange algorithm, let ω_1^i, ω_2^i, ... be the estimates at the ith iteration of the frequencies at which alternations of zero slope will occur, and let

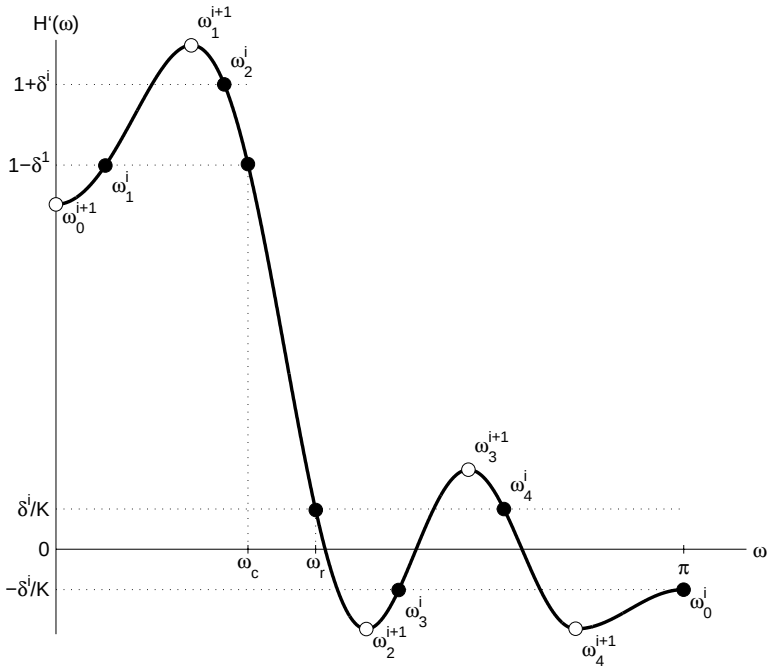


Figure 2.9: Illustration of the ith iteration of the Remez exchange algorithm.

δ^i be the estimate of δ. In addition, we know that there are alternations at ω_c and ω_r and at either 0 or π or both (let ω_0^i our current estimate of which one). These estimated alternation points are indicated by dots ($\bullet$) in Figure 2.9. A trigonometric polynomial of form (2.24) is passed through the points $1 \pm \delta^i$ in the passband and $\pm\frac{\delta^i}{K}$ in the stopband at these frequencies with alternating sign such that $1-\delta^i$ occurs at ω_c and $+\frac{\delta^i}{K}$ at ω_r, as depicted in Figure 2.9. This new estimate of $H(\omega)$ is then evaluated with high resolution to locate the frequencies ω_0^{i+1}, ω_1^{i+1}, ..., at which extrema actually occur (choosing ω_0^{i+1} by whether the extremum at 0 or π is larger or using both if needed) as indicated by circles ($\circ$) in Figure 2.9. These frequencies plus a new estimate δ^{i+1} form the input to the $(i+1)$st iteration.

2.2.4 Realization of Linear-Phase FIR Filters

As previously shown in Chapter 2.2.2 the input-output relationship of any Nth order FIR filter can be written as a finite convolution of the input signal $x[n]$ and the

impulse response $h[n]$

$$y[n] = \sum_{k=0}^{N} h[k]x[n-k]. \tag{2.26}$$

The network structure that realizes the convolution in (2.26) is shown in Figure 2.10. This structure is often called the *direct form* because it is a direct implementation of the basic difference equation. Direct form structures, in general, realize the given

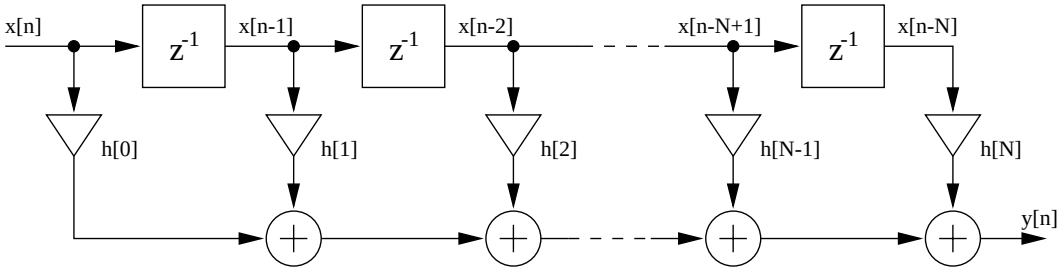


Figure 2.10: Direct form FIR filter network structure.

system function with the smallest possible number of delays, adders, and multipliers. The number of each of these components required is as follows:

$$\begin{aligned} \text{number of delays} &= N \\ \text{number of adders} &= N \\ \text{number of multipliers} &= N+1 \\ \text{number of coefficients} &= N+1 \end{aligned} \tag{2.27}$$

By *adders* we mean two-input adders, and thus a summation node with S inputs implies $S-1$ adders in the implementation.

Since real-valued linear-phase FIR filters of order N do not consist of $N+1$ distinct coefficients, but of $\frac{N+1}{2}$ pairs of equal or complementary coefficients (plus one middle coefficient for N even), the direct form network structure can be implemented more efficiently. To explain the benefit of having coefficient symmetry, let us consider a type II filter with even symmetry and N odd. By splitting the sum in (2.26) at the midpoint we obtain

$$y[n] = \sum_{k=0}^{\frac{N-1}{2}} h[k]x[n-k] + \sum_{k=\frac{N+1}{2}}^{N} h[k]x[n-k]. \tag{2.28}$$

After reversing the order of the summation on the right-hand side and shifting the limits we get

$$y[n] = \sum_{k=0}^{\frac{N-1}{2}} h[k]x[n-k] + \sum_{k=0}^{\frac{N-1}{2}} h[N-k]x[n-N+k]. \tag{2.29}$$

Since for linear-phase filters $h[k] = h[N-k]$ we can simplify (2.29) and thus obtain

$$y[n] = \sum_{k=0}^{\frac{N-1}{2}} h[k]\left(x[n-k] + x[n-N+k]\right). \tag{2.30}$$

Hence, by adding $x[n-k]$ and $x[n-N+k]$ prior to multiplying them with the corresponding coefficients, only $\frac{N+1}{2}$ multiplications have to be performed, and a summation node with only $\frac{N+1}{2}$ inputs is needed to generate the output signal $y[n]$. Furthermore, only $\frac{N+1}{2}$ coefficients have to be stored. The modified network structure implementing this simplified system is depicted in Figure 2.11. The total number of

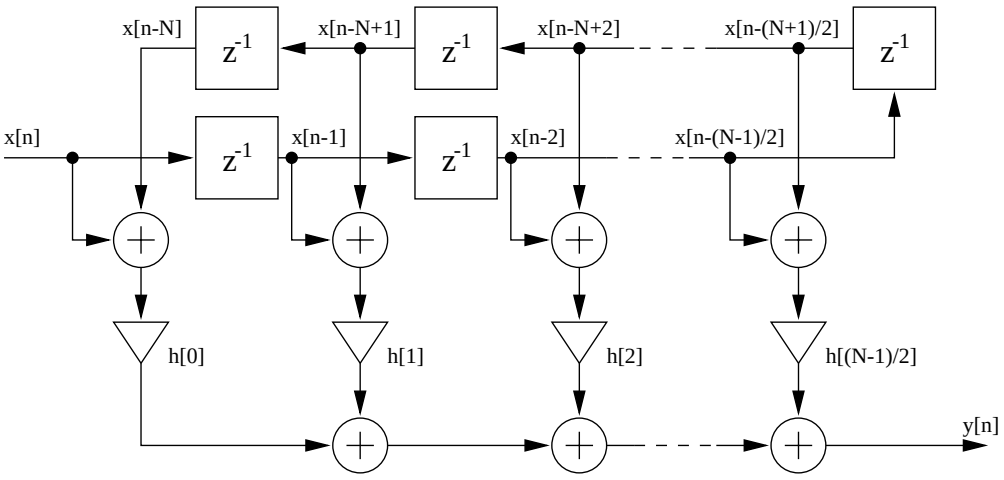


Figure 2.11: Modified direct form structure for linear-phase FIR filters with even coefficient symmetry and N odd.

two-port adders needed amounts to $\frac{N+1}{2}$ adders to create the multiplier input signals, plus $\frac{N-1}{2}$ adders for the summation node. The number of required components to implement both type I and II filters are listed in Table 2.1. Note that for N odd, every

Component	N odd	N even
delays	N	N
adders	N	N
multipliers	$\frac{N+1}{2}$	$\frac{N}{2}+1$
coefficients	$\frac{N+1}{2}$	$\frac{N}{2}+1$

Table 2.1: Required components for the implementation of linear-phase FIR filters.

coefficient requires one adder, one multiplier, and one input of the output summation node. For N even, the extra middle coefficient only requires one multiplier and one summation input.

The numbers in Table 2.1 also apply to type III and IV filters. But because of the odd coefficient symmetry, each signal $x[n-k]$ has to be added to $-x[n-N+k]$. This subtraction can be achieved by inserting a sign-inverting component into the delay line immediately after the midpoint. This results in $-x[n]$ being shifted through the second half of the delay line.

2.2.5 Effects of Coefficient Quantization

As for all digital signal processing applications, the coefficients $h[n]$ of an FIR filter can only be stored with limited precision. This rounding to a quantization step size of q results in an additive error $e[n]$ for each coefficient. Thus, the impulse response of an FIR filter with quantized coefficients can be written as

$$\hat{h}[n] = h[n] + e[n], \tag{2.31}$$

where $e[n]$ for each n is a number that satisfies $|e[n]| \leq \frac{q}{2}$. For a given number of bits B, to which $h[n]$ is rounded, the quantization step size is $q = \frac{\Delta}{2^B-1}$, where Δ denotes the maximum quantization range. For future reference we will assume $\Delta = 1$, since $|h[n]| \leq 1$ for all unity-gain FIR filters. Hence, $q = \frac{1}{2^B-1}$.

By taking the z-transform of (2.31) we obtain

$$\hat{H}(z) = \sum_{n=0}^{N} \hat{h}[n]z^{-n} = \sum_{n=0}^{N} h[n]z^{-n} + \sum_{n=0}^{N} e[n]z^{-n} = H(z) + E(z). \tag{2.32}$$

Therefore, a filter with quantized coefficients can be represented as the parallel connection of the "infinite-precision" version of that filter with a filter whose transfer function is $E(z)$. The block diagram for this system is depicted in Figure 2.12.

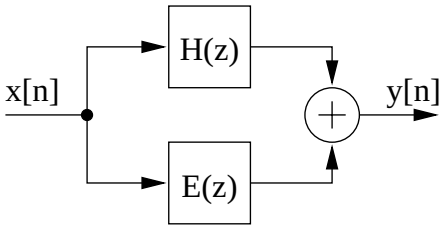


Figure 2.12: Block diagram for an FIR filter with quantized coefficients.

As explained in Chapter 2.2.4 only half of the coefficients have to be stored for the implementation of a linear-phase FIR filter. Because of the way a such a system is realized, $\hat{h}[n]$ and $\hat{h}[N-n]$ correspond to the same stored coefficient. Hence, the impulse response of a linear-phase filter with quantized coefficients is symmetric around its midpoint. Since the ideal impulse response $h[n]$ is symmetric, $e[n]$ is also symmetric. Consequently, *the linear-phase property is not affected be the quantization of the coefficients.* Because of their nonrecursive structure, FIR filters do not suffer from limit-cycle effects or instability due to coefficient quantization. In fact, *FIR filters with quantized coefficients are always stable.*

Figure 2.13 shows the effect of quantization on the magnitude response of a linear-phase FIR lowpass filter of order $N = 64$ with coefficients that have been rounded to a word length of $B = 9$ bits. The dashed line indicates the magnitude response of the ideal filter, while the solid line shows the actual response of the filter with quantized coefficients. It is obvious that 9 bits do not provide a precise enough representation of the coefficients for this filter. The distortion introduced by the process of quantization is strongest in the stopband. It can be seen in Figure 2.13 that at some frequencies the stopband attenuation dropped by more than 20dB. To minimize the error caused by coefficient quantization the number of resolution bits has to be increased. Within this Chapter we will introduce the means to estimate the minimum number of bits required to ensure a given minimum attenuation.

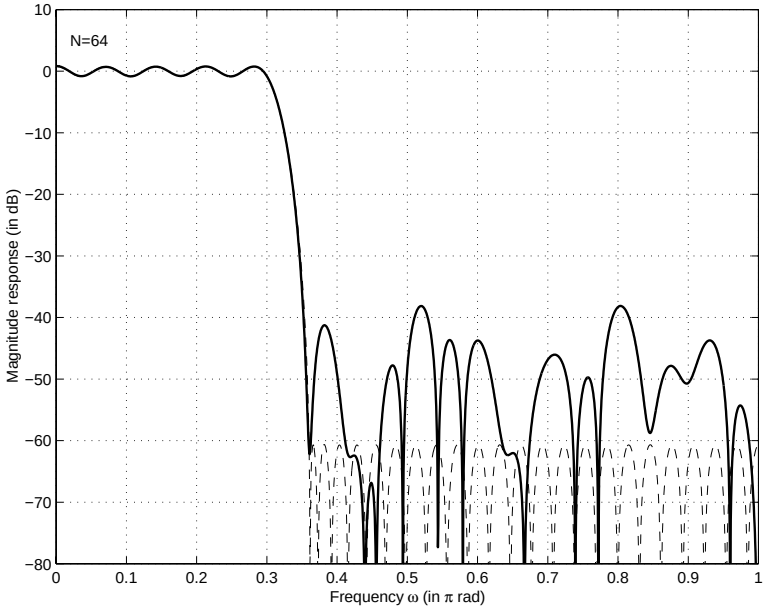


Figure 2.13: Magnitude response of a linear-phase FIR filter with quantized coefficients of bit depth 9 (for $N = 64$).

The frequency response $\hat{H}(\omega)$ can be obtained by substituting z by $e^{j\omega}$ in (2.32)

$$\hat{H}(\omega) = \sum_{n=0}^{N} h[n]e^{-j\omega n} + \sum_{n=0}^{N} e[n]e^{-j\omega n} = H(\omega) + E(\omega). \tag{2.33}$$

To determine the amount of distortion caused by $e[n]$, it is necessary to analyze the filter response error $E(\omega)$. Since the values for $e[n]$ are unpredictable, $E(\omega)$ is not a deterministic function, and thus, only a qualitative analysis can be conducted. Because $|e[n]| \leq \frac{q}{2}$ it is possible to compute a bound on $E(\omega)$ that is independent of the elements of $e[n]$. The derivation is as follows

$$|E(\omega)| = \left| \sum_{n=0}^{N} e[n]e^{-j\omega n} \right| \leq \sum_{n=0}^{N} |e[n]| \underbrace{|e^{-j\omega}|}_{=1} \leq \frac{q}{2}(N+1) \tag{2.34}$$

$$\breve{E}_{dB} = 20 \log_{10} \left(\frac{q}{2}(N+1) \right), \tag{2.35}$$

where $\breve{E}_{dB}$ denotes the upper bound of $|H(\omega)|$ in decibels. Let us apply the bound to the system in the previous example. The error bound for a system of order

$N = 64$ with coefficients quantized to 9 bits is $\breve{E}_{dB} = -23.93$dB. Figure 2.14 shows the error $E_{dB}(\omega)$ in dB of the filter response depicted in Figure 2.13 obtained by subtracting the magnitude response $H(\omega)$ of the ideal filter from the actual response $\hat{H}(\omega)$. The dot-dashed line $(\cdot -)$ indicates the previously calculated error bound.

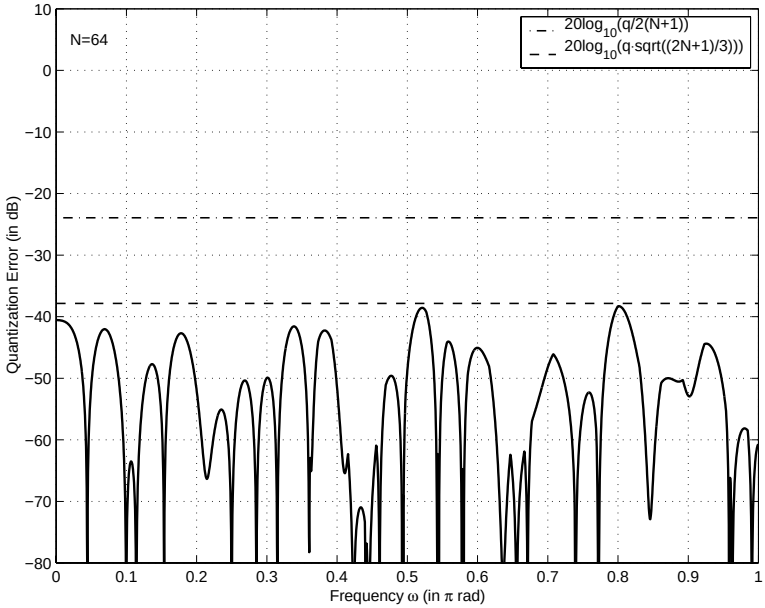


Figure 2.14: Frequency response error $E(\omega)$ for a filter of order $N = 64$ with 9-bit coefficient quantization.

The comparison shows that the deviation introduced by the quantization errors is less than the corresponding bound in all cases. $\breve{E}_{dB}$ is a pessimistic bound, since all of the quantization errors would have to be of the same sign and equal to $\frac{q}{2}$ to be as large as the bound. The probability of this occurring is very small.

Because of the inherently hard-to-predict nature of quantization errors, a statistical analysis [7] of the effect of coefficient quantization is appropriate, even though for a given filter the quantization process is performed only once, after which the filter response is exactly determined. The statistical model to be used is a very reasonable one that assumes the errors due to the quantization of different coefficients to be independent and uniformly distributed between -$\frac{q}{2}$ and $\frac{q}{2}$, as depicted in Figure 2.15. It can be seen in the Figure that the probability distribution function $f_e(e)$ has a

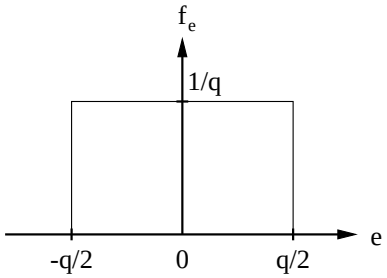


Figure 2.15: Uniform probability distribution of the quantization errors $e[n]$.

mean $\mu_e = 0$. The variance σ_e^2 for each $e[n]$ can be calculated as follows

$$\sigma_e^2[n] = \int_{-\infty}^{\infty} e^2[n] f_e(e) de = \frac{1}{q} \int_{-\frac{q}{2}}^{\frac{q}{2}} e^2[n] de = \frac{q^2}{12} = \sigma_e^2. \tag{2.36}$$

By applying the manipulations previously used in Chapter 2.2.2 we obtain the following expression for the filter error response $E(\omega)$

$$\begin{aligned} E(\omega) &= e^{-j\omega\frac{N}{2}} \left(e[\frac{N}{2}] + \sum_{n=0}^{\frac{N}{2}-1} 2e[n] \cos\left(\left(\frac{N}{2} - n \right) \omega \right) \right) \\ &= e^{-j\omega\frac{N}{2}} R(\omega). \end{aligned} \tag{2.37}$$

$E(\omega)$ consists of a linear-phase term and the real-valued error response $R(\omega)$, which for all ω is a linear combination of independent uniform random variables. The linear-phase term $e^{-j\omega\frac{N}{2}}$ in (2.37) simply represents a pure delay of an integer number of samples that is not affected by quantization. Therefore, this factor need not be considered, and only the change in the real function $R(\omega)$ due to coefficient quantization is studied. Since the probability density functions $f_e(e)$ for all $e[n]$ vanish outside some finite interval, the sum in (2.37) satisfies the Lindeberg condition of the central limit theorem. Thus, for sufficiently large N the underlying probability of $R(\omega)$ will converge to a Gaussian distribution. Because of this tendency of the errors in the frequency domain to be Gaussian, their mean and variance alone provide an excellent description of their statistical behavior. The mean $\mu(\omega)$ and variance

$\sigma(\omega)$ are computed as follows

$$\begin{aligned}\mu(\omega) &= \mu_e[\frac{N}{2}] + \sum_{n=0}^{\frac{N}{2}-1} 2\mu_e[n]\cos\left(\left(\frac{N}{2}-n\right)\omega\right) = 0 \\ \sigma^2(\omega) &= \sigma_e^2[\frac{N}{2}] + \sum_{n=0}^{\frac{N}{2}-1} 2\sigma_e^2[n]\cos^2\left(\left(\frac{N}{2}-n\right)\omega\right).\end{aligned} \tag{2.38}$$

$\sigma^2(\omega)$ may be expressed in closed form by summing the series in (2.38)

$$\sigma^2(\omega) = \sigma_e^2\left(N + \frac{\sin\Big((N+1)\omega\Big)}{\sin(\omega)}\right). \tag{2.39}$$

After inserting $\sigma_e^2 = \frac{q^2}{12}$ and normalizing the expression in parentheses, we obtain

$$\sigma^2(\omega) = \frac{q^2}{12}(2N+1)\frac{1}{2N+1}\left(N + \frac{\sin\Big((N+1)\omega\Big)}{\sin(\omega)}\right). \tag{2.40}$$

Hence we get the following standard deviation

$$\begin{aligned}\sigma(\omega) &= \frac{q}{2}\sqrt{\frac{2N+1}{3}}\cdot\sqrt{\frac{1}{2N+1}\left(N + \frac{\sin\Big(\Big((N+1)\omega\Big)}{\sin(\omega)}\right)} \\ &= \sigma_{max}\cdot W(\omega),\end{aligned} \tag{2.41}$$

where $W(\omega)$, a normalized weighting function that is not a function of the quantization step size q, is the only frequency-dependent term. Figure 2.16 shows $W(\omega)$ for a filters of order $N = 8$ and $N = 64$. The behavior of $W(\omega)$ in the limit of large N is readily seen from (2.41). In the range $0 < \omega < \pi$,

$$\lim_{N\to\infty} W(\omega) = \frac{1}{\sqrt{2}} \tag{2.42}$$

whereas for $\omega = 0, \pm\pi, \pm 2\pi, \ldots,$

$$W(\omega) = 1, \qquad \text{all } N. \tag{2.43}$$

The convergence of (2.42) is not uniform over $(0, \pi)$. However, in the interval $[\epsilon, \pi - \epsilon]$ for any $\epsilon > 0$ convergence is uniform.

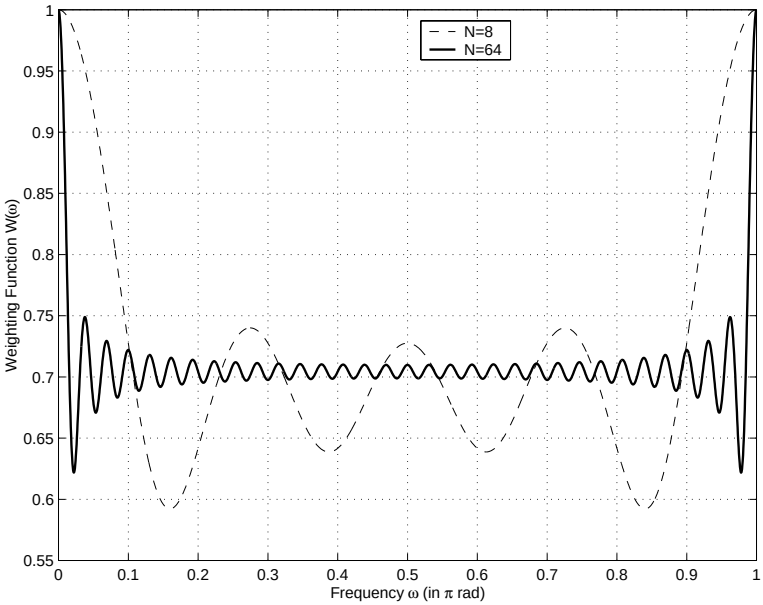


Figure 2.16: Weighting function $W(\omega)$ for filter orders $N = 8$ and $N = 64$.

Equation (2.41) shows $W(0) = W(\pi) = 1$, and $0 < W(\omega) \leq 1$ for all N. Thus

$$\sigma(\omega) \leq \sigma_{max} = \frac{q}{2}\sqrt{\frac{2N+1}{3}}, \tag{2.44}$$

where σ_{max} represents the minimum upper bound of $\sigma(\omega)$. The significance of the plots in Figure 2.16 and (2.42) is that they show $\sigma(\omega)$ to be well described by its bound in (2.44). Since we know that the error response $E(\omega)$ yields a Gaussian distribution, we can say that with a high probability it is bound to two or three times its standard deviation. In fact, the probability of a Gaussian random variable to be smaller or equal to twice its standard deviation is 0.954. Therefore, the probability of $|E(\omega|)$ to be smaller than or equal to twice the upper bound σ_{max} is even higher. Thus

$$|E(\omega)| \tilde{\leq} 2\sigma_{max} = q\sqrt{\frac{2N+1}{3}}, \tag{2.45}$$

where $\tilde{\leq}$ denotes "is with high probability less than or equal to". The resulting upper bound in dB then is

$$\breve{E}_{dB} = 20\log_{10}\left(\frac{q}{2}(N+1)\right). \tag{2.46}$$

When applying this bound to the filter in the previous example of order $N = 64$ and 9-bit coefficients, we find that $\breve{E}_{dB} = -37.83dB$. This new bound is indicated by a dashed line in Figure 2.14. It can be seen that (2.46) provides a much sharper bound than (2.35).

To investigate the distortion of the frequency response caused by coefficient quantization, let $L(\omega)$ be some real, ideal band-select function to be approximated by the frequency response $H(\omega)$ of a linear-phase FIR filter. The usual design specifications consist of a set of disjoint frequency bands $\Omega_k \subset [0, \pi], k = 1, \ldots, M$, and a set of corresponding in-band bounds $\delta_k > 0$, such that for each k, $L(\omega)$ is approximated by $H(\omega)$ within an error of δ_k for all $\omega \in \Omega_k$. Hence

$$\max_{\omega \in \Omega_k} |H(\omega) - L(\omega)| = \delta_k, \qquad k = 1, \ldots, M. \tag{2.47}$$

The frequency bands Ω_k are separated by transition bands where the frequency response is unconstrained. Quantizing the coefficients of a filter designed to meet these specifications can increase the approximation error to be beyond the specified bounds δ_k. Clearly, the new upper bound for all ω is

$$\begin{aligned} |\hat{H}(\omega) - L(\omega)| &\leq |\hat{H}(\omega) - H(\omega)| + |H(\omega) - L(\omega)| \\ &\leq |E(\omega)| + |H(\omega) - L(\omega)|, \end{aligned} \tag{2.48}$$

where $\hat{H}(\omega)$ denotes the frequency response of the filter with quantized coefficients. By only considering the bands of interest Ω_k we obtain

$$\begin{aligned} |\hat{H}(\omega) - L(\omega)| &\leq \max_{\omega \in \Omega_k} |E(\omega)| + \delta_k \\ &\tilde{\leq} \; q\sqrt{\frac{2N+1}{3}} + \delta k \\ &\tilde{\leq} \; \underbrace{\frac{1}{2^B - 1}\sqrt{\frac{2N+1}{3}} + \delta k}_{\epsilon_k}, \qquad \text{all } \omega \in \Omega_k, \end{aligned} \tag{2.49}$$

which means that with high probability, $L(\omega)$ can be approximated on Ω_k by the frequency response of an N-point linear-phase FIR filter, with B-bit coefficients, to an error bounded by ϵ_k, if the frequency response of the filter with infinite-precision coefficients $H(\omega)$ can be designed such that for all $\omega \in \Omega_k$

$$|H(\omega) - L(\omega)| \leq \delta_k. \tag{2.50}$$

Equations (2.49) and (2.50) are further investigated be rephrasing them in terms of *in-band attenuation* in decibels. We define the in-band attenuation on Ω_k of the ideal filter as

$$A_k = -20\log_{10}(\delta_k). \tag{2.51}$$

Then from (2.49) the lower bound on the in-band attenuation for the filter with quantized coefficients is

$$\hat{A}_k(N, B, A_k) \tilde{\geq} \; -20\log_{10}\left(10^{-\frac{A_k}{20}} + \frac{1}{2^B - 1}\sqrt{\frac{2N+1}{3}}\right). \tag{2.52}$$

Accordingly for the previous example, a filter of order $N = 64$ with 9-bit coefficients, the stopband attenuation will, with a high probability, be at least 37.19dB. By comparing this result with the plot in Figure 2.13 we see that the frequency response of the quantized filter is indeed as expected. The lower bound in (2.52) is plotted in Figure 2.17 as a function of A_k for $N = 64$ and several values of B.

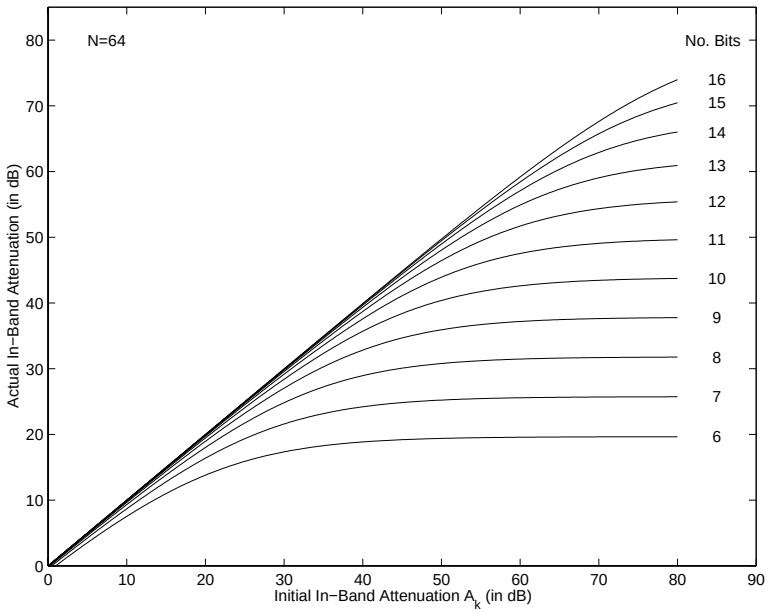


Figure 2.17: Lower bounds on in-band attenuation obtained after coefficient quantization as a function of initial attenuation (for $N = 64$).

In the process of filter design it is sometimes more useful to know what minimum number of bits are required for a given maximum deviation from the initial in-band attenuation, rather than what maximum attenuation can be achieved for a given number of bits. Let us put a constraint on $\hat{A}_k$ by defining Δ_k to be the maximum change of in-band attenuation allowed for a given ideal filter, hence $\hat{A}_k \geq A_k - \Delta_k$. Then, by substituting $\hat{A}_k$ in (2.52) with this constraint and solving for B we obtain

$$B(N, A_k, \Delta_k) \geq \log_2 \left(1 + \frac{10^{\frac{A_k}{20}}}{10^{\frac{\Delta_k}{20}} - 1} \cdot \sqrt{\frac{2N+1}{3}} \right). \tag{2.53}$$

Figure 2.18 shows the minimum number of bits required for a maximum deviation of 6dB as a function of the filter order N for several given initial in-band attenuations. When applying (2.53) to the previous example, we find that we need at least 13 quantization bits for a maximum stopband deviation of 6dB.

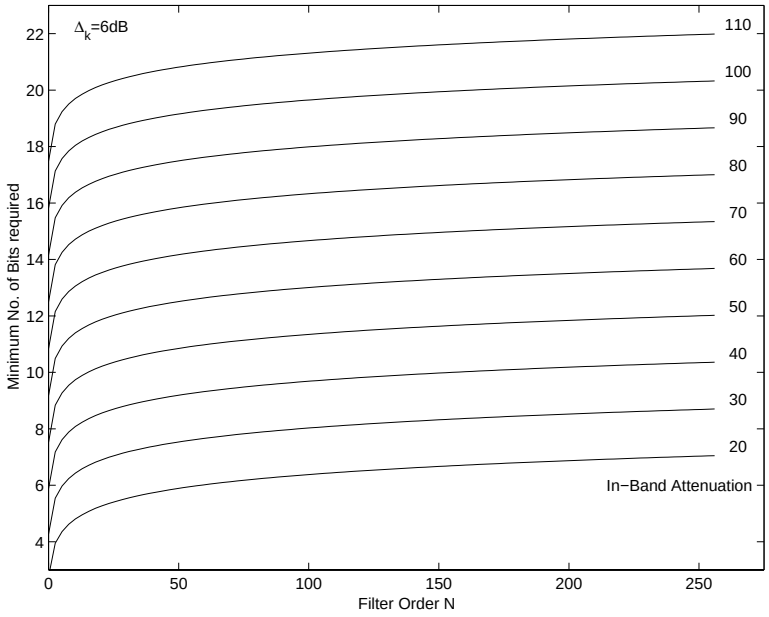


Figure 2.18: Minimum number of bits required for a given minimum in-band attenuation (for $N = 64$).

Sometimes the maximum number of bits that can be used to quantize the filter coefficients is limited by a given hardware setup, i.e., a digital signal processor with

fixed point arithmetic and a fixed data bus width. In this case it would be helpful to have a formula that offers the possibility of investigating the maximum decrease of in-band attenuation caused by coefficient quantization to a given number of bits. By negating (2.52) and adding A_k on both sides we obtain

$$A_k - \hat{A}_k \lesssim 20 \log_{10} \left(1 + \frac{10^{\frac{A_k}{20}}}{2^B - 1} \cdot \sqrt{\frac{2N+1}{3}} \right). \tag{2.54}$$

This relationship empowers the filter designer to estimate the influence of having non-ideal hardware to implement the filter, and to take the necessary measures if the quantized filter does not meet the specifications. The free parameters in (2.54) are the filter order N and the initial in-band attenuation A_k. Figure 2.19 depicts

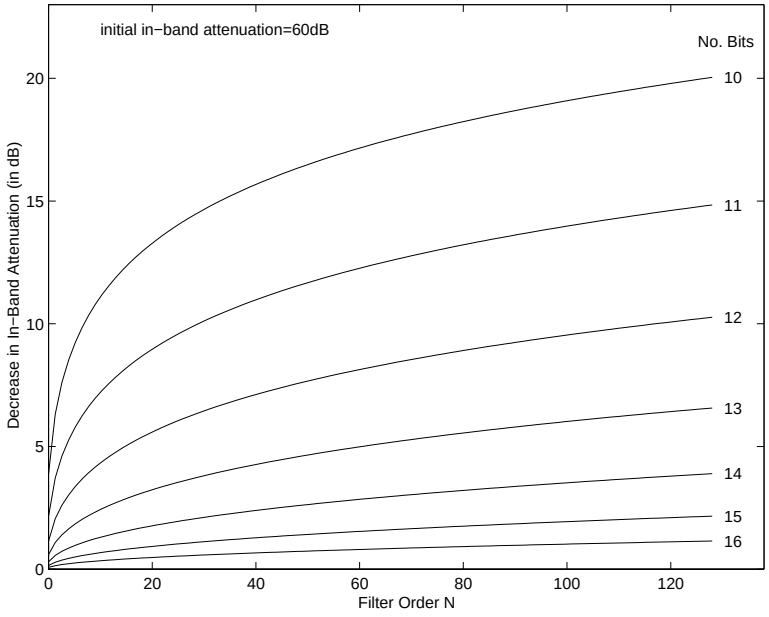


Figure 2.19: Bounds on the change of in-band attenuation due to coefficient quantization.

the effect on the in-band attenuation of linear-phase FIR filters with an initial in-band attenuation of 60dB versus filter order N for a given number of bits used to represent the filter coefficients. Let us consider the previous example, a filter of order $N = 64$ and an initial stopband attenuation of 60dB. By using 2.54 we find that the

quantization of the coefficients to a length of 9 bits results in a drop of stopband attenuation of up to 22.82dB, which can be seen in Figure 2.13.

Equations (2.52), (2.53), and (2.54) provide a very helpful toolbox which enables us to study the impact of coefficient quantization on a filter designed with conventional design techniques. Let us apply this toolbox to the example first introduced at the beginning of this section. Recall that the coefficients of a linear-phase FIR lowpass filter of order $N = 64$ and an initial stopband attenuation of 60dB were quantized to a length of 9 bits. It can be seen in Figure 2.13 that 9 bits are clearly not enough to guarantee a sufficient rejection in the stopband. By allowing a maximum decrease of 6dB in the stopband, we find that we need a minimum of 13 bits to represent the coefficients. The solid line in Figure 2.20 indicates the the Magnitude response of the filter with 13-bit coefficients. It can be seen that the filter now meets the required specifications.

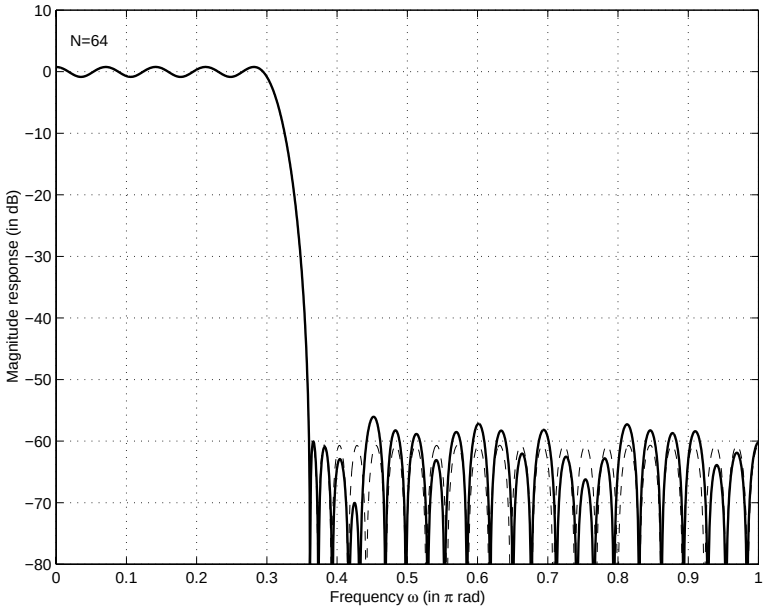


Figure 2.20: Magnitude response of a linear-phase FIR filter with 13-bit coefficients (for $N = 64$).

Chapter 3

FIR Decimation Filter Design using MATLAB®

In the early years of digital signal processing, discrete-time filters were designed by calculating appropriate continuous-time filters and then transforming them into discrete-time filters. This approach is reasonable when we can take advantage of continuous-time designs that have closed-form formulas or when we can use extensive existing design tables. Filters for which this is possible are frequency-selective circuits such as Butterworth, Chebyshev, or elliptic filters. In general, however, analytical formulas do not exist for the design of discrete time filters to match arbitrary specifications. In these more general cases, design procedures have been developed that are algorithmic, generally relying on the use of dedicated computer software to solve sets of linear or nonlinear equations. The *Signal Processing Toolbox* [1] for MATLAB® comprises an excellent collection of tools for all kinds of digital signal processing applications, including the design of digital decimation filters.

3.1 Defining the Filter Specifications

Before a digital decimation filter can be designed, the desired specifications for the filter have to be known. Normally, they do not only depend on the process following the filter and the downsampling ratio D, but also on the characteristics of the input signal. Furthermore, the limitations of the hardware used to implement the filter

have to be considered. Typically, the number of taps is limited since the physical size of a filter grows with its order, and also, the maximum number of bits available for coefficient memory is usually bounded by a certain value.

3.1.1 Input Signal Characteristics

The fifth-order IFLF $\Delta\Sigma$ Modulator used in the sonar receiver system, for which the decimation filter has to be designed, was introduced by Fischer [8]. It employs an oversampling ratio of $OSR = 30$ at a maximum sampling rate of $f_s = 6.4$MHz. Therefore, the maximum frequency of the input signal $x(t)$ can be computed as follows

$$f_{max} = \frac{f_s}{2OSR} = 106.7\text{kHz}. \tag{3.1}$$

In order to avoid aliasing, the $\Delta\Sigma$ Modulator is preceded by an RC analog lowpass filter. Due to the high oversampling ratio, the order of this filter can be kept fairly low since the transition band can be as wide as $f_T = f_s - f_{max} = 6.293$MHz.

To ensure the stability of the $\Delta\Sigma$ Modulator its input is limited to the range $-\frac{1}{2}V_{ref} \ldots +\frac{1}{2}V_{ref}$, i.e., half of the maximum possible output range. Figure 3.1 shows the output spectrum of the fifth-order IFLF $\Delta\Sigma$ Modulator for an input signal $x(t)$ with frequency $f_x = 102$kHz and amplitude $V_x = \frac{1}{2}V_{ref}$.

3.1.2 Output Signal Requirements

The requirements for the filter-output signal are given by our application. The filtered signal has to be suitable for wideband beamforming, i.e., the signal must be downsampled by a factor of $D = 16$ to the output sampling rate of $f_s/D = f_d = 400$kHz. To avoid aliasing, all the spectral components of the filtered signal must be below -100dB for all frequencies $f \geq f_d - f_{max} = f_a = 293.3$kHz.

Since the input signal amplitude will always be within the interval $[-\frac{1}{2}, +\frac{1}{2}]$ with respect to V_{ref}, the $\Delta\Sigma$ Modulator output signal can be amplified by a factor 1.5 ($\hat{=}$3.52dB) without causing any overflow. The range of the output signal component will then be $[-\frac{3}{4}, +\frac{3}{4}]$ with respect to V_{ref}. The remaining 25% of the output range serve as a reserve to accommodate the noise.

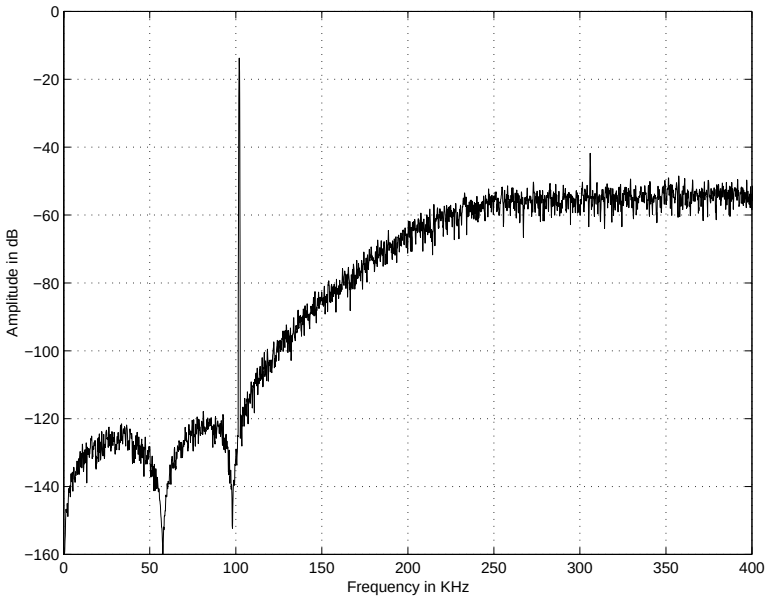


Figure 3.1: $\Delta\Sigma$ Modulator output spectrum for an input signal $x(t)$ with $f_x = 102\text{kHz}$ and $V_x = \frac{1}{2}V_{ref}$.

3.1.3 Filter Specifications

One of the most important requirements for a digital filter used in beamforming applications is that the filter must yield a constant group delay. Hence, a linear-phase FIR filter is the most suitable choice, since it guarantees a linear phase response for all frequencies.

The edges of the passband and the stopband are defined by the maximum input signal frequency f_{max} and its decimated alias $f_a = f_d - f_{max}$. Thus, the filter has to approximate the following characteristic

$$H(f) = \begin{cases} 1, & f \leq f_c = f_{max} = 106.7\text{kHz} \\ 0, & f \geq f_r = f_a = 293.3\text{kHz} \end{cases} . \tag{3.2}$$

To ensure that all spectral components for $f \geq f_a$ are less than -100dB in amplitude, the stopband attenuation has to be at least 100dB. The required gain of 1.5 can easily be realized by scaling all the coefficients by 1.5. Note that by doing so, we increase not only the passband gain but also the stopband gain by 3.52dB.

Consequently, to maintain the minimum stopband attenuation of 100dB for the filter with the scaled coefficients, the stopband ripple of the unscaled filter has to be $\delta_s = -103.52$dB or less. The last constraint on the frequency response of the filter is the maximum allowable error in the passband, i.e., the passband ripple δ_p. It is important in beamforming applications that the gain does not fluctuate more than ± 0.1dB over the width of the passband.

By considering all the requirements mentioned above we define a first set of filter specifications in Table 3.1. Figure 3.2 graphically displays these specifications.

Cut-off frequency	106.7kHz
Stopband edge	293.3kHz
Passband ripple	0.1dB
Stopband ripple	-104dB

Table 3.1: Specifications for the digital decimation filter.

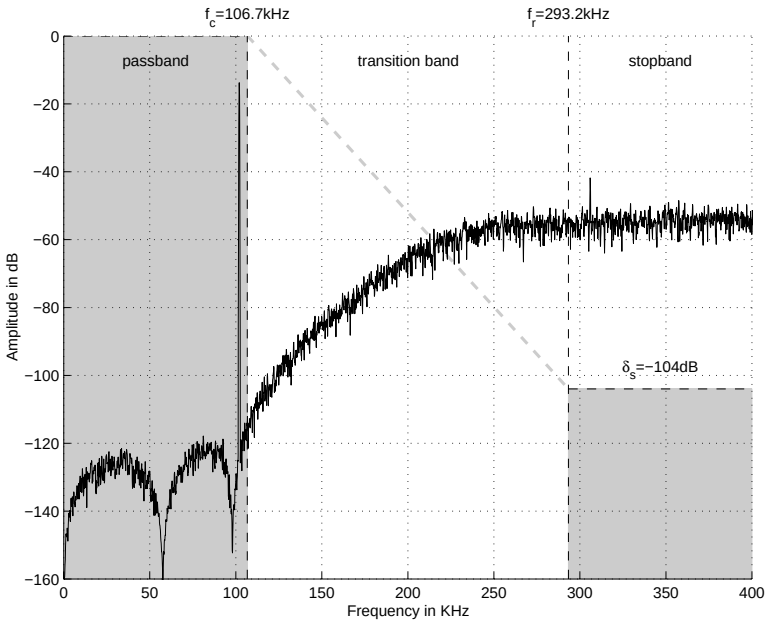


Figure 3.2: Specifications of the digital decimation filter.

3.2 Filter Design

The design of a digital filter using CAD software is usually an iterative process. In most cases the specifications defined prior to designing the filter provide a good set of parameters for filter design algorithms. However, since the ideal filter can only be approximated, certain trade-offs have to be considered. Often it is necessary to adjust the specifications in order to achieve satisfactory performance.

3.2.1 Determining the Filter Order

Before the Remez Exchange Algorithm can be used to calculate the coefficients, the filter order N needs to be known. The *Signal Processing Toolbox* command `[N,Fo,Ao,W] = remezord(F,A,DEV,fs)` finds the approximate order N, the normalized frequency band edges F_o, the frequency band magnitudes A_o, and the weights W to be used by the `remez` function. The resulting filter will approximately meet the specifications given by the input parameters F, A, and DEV. F is a vector of cutoff frequencies in Hz, in ascending order between 0 and half the sampling frequency f_s. A is a vector specifying the desired function's amplitude on the bands defined by F, and DEV is a vector of maximum ripples allowable for each band. In our case these vectors are as follows

$$F = \begin{bmatrix} f_p \\ f_r \end{bmatrix}, \quad A = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \quad DEV = \begin{bmatrix} 10^{\delta_p/20} - 1 \\ 10^{\delta_s/20} \end{bmatrix}. \tag{3.3}$$

For the values listed in Table 3.1, the function `remezord` returns the value $N = 131$. As we will see in Chapter 3.5, it is beneficial to build digital FIR decimation filters with 2^n coefficients where n is an integer number. Therefore, the next higher-order filter for which this is true has 256 coefficients, hence, $N = 255$. It is obvious that this is much more than would be necessary in order to meet the specifications, but as we will show in Chapter 3.5, the filter implementation does not become much more complex.

Choosing a filter order that is higher than necessary leaves the possibility to improve the filter characteristic. The minimum order required to realize an arbitrary filter characteristic is generally determined by the relative width of the transition

bands with respect to the sampling frequency and the maximum allowable error in the individual frequency bands. For our application, the ripples previously defined are sufficient, but by reducing the transition bandwidth, we can significantly improve the filter's performance. It can be seen in Figure 3.2 that immediately above the cut-off frequency f_c, the quantization noise power increases drastically. By narrowing the transition band, the noise rejection between f_c and f_r can be enhanced.

We can use the function `remezord` to estimate the minimum required filter order for a given transition bandwidth. Figure 3.3 depicts the required filter order for our given set of specifications with a variable transition bandwidth. According to these

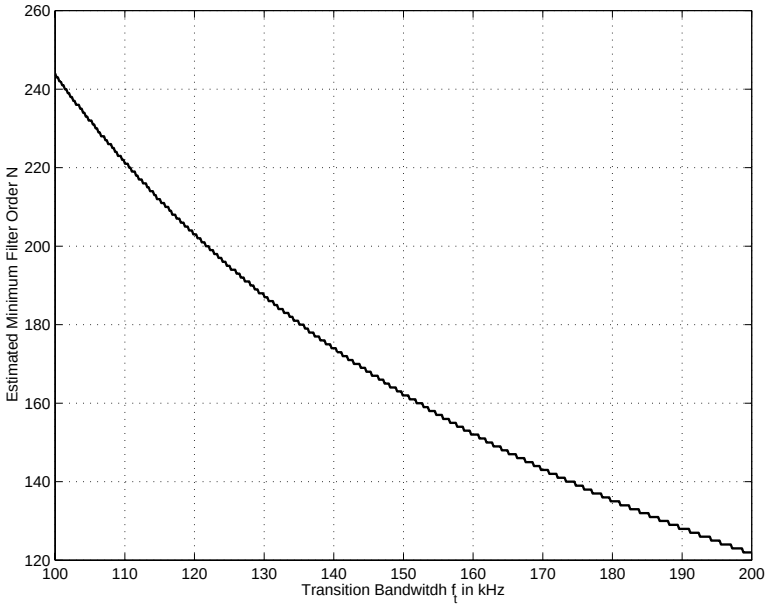


Figure 3.3: Estimated minimum filter order versus transition bandwidth f_t.

results, the transition bandwidth can be chosen as low as 100kHz. But knowing that `remezord` only *estimates* the filter order, we make a more conservative choice for the new transition bandwidth. By setting the new value for f_t to 115kHz, we obtain the improved set of filter specifications listed in Table 3.2.

The estimated minimum filter order for these specifications is $N = 212$.

Cut-off frequency	106.7kHz
Stopband edge	221.7kHz
Passband ripple	0.1dB
Stopband ripple	-104dB
Filter order	255

Table 3.2: Improved specifications for the digital decimation filter.

3.2.2 Calculating the Filter Coefficients

The MATLAB® command `h = remez(N,Fo,Ao,W)` returns a length $N+1$ real linear-phase FIR filter which has the best approximation to the desired frequency response described by F_0 and A_0 in the minimax sense. F_0 is a vector of normalized frequency band edges in pairs, in ascending order between 0 and 1. 1 corresponds to the Nyquist frequency or half the sampling frequency f_s. A_0 is a real vector of the same size as F_0, which specifies the desired amplitude of the frequency response of the resultant filter h. `remez` uses the values in W to weight the error. W has one entry per band which tells the Remez Exchange Algorithm how much emphasis to put on minimizing the error in each band relative to the other bands. The vectors A_0, F_0 and W are obtained by calling the function `remezord` with the specifications listed in Table 3.2. The returned values are as follows

$$F_0 = \begin{bmatrix} 0 \\ 0.03334 \\ 0.06928 \\ 1 \end{bmatrix}, \quad A_0 = \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \end{bmatrix}, \quad W = \begin{bmatrix} 1 \\ 1835.22 \end{bmatrix}. \tag{3.4}$$

Executing `remez` with the values in (3.4) and $N = 255$ returns the linear-phase FIR filter impulse response h of length 256 depicted in Figure 3.4. Note that the impulse response h is symmetric around its midpoint, which is typical for linear-phase FIR filters.

The frequency response $H(f)$ can be calculated by taking the Fourier-Transform of h. The MATLAB® function `H = fft(h,M)` computes the M-point FFT of h, padded with zeros if h has less than M points and truncated if it has more. The complex

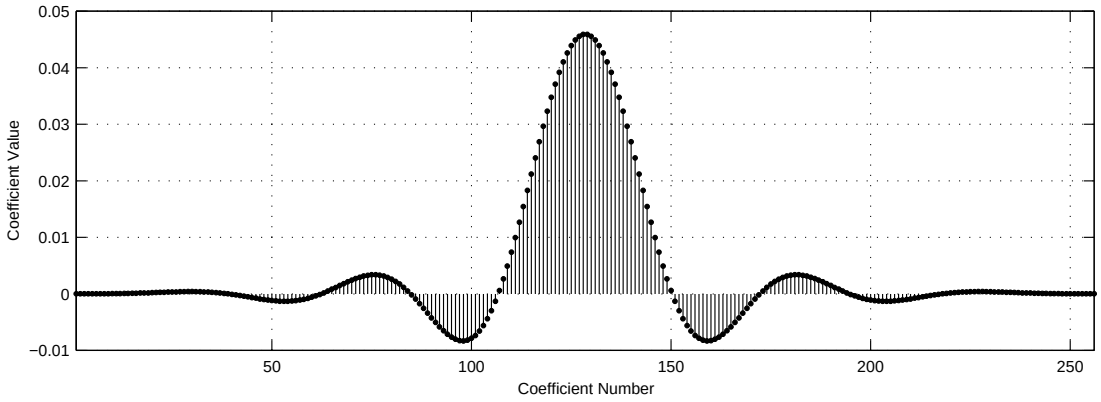


Figure 3.4: Filter coefficients.

resultant vector H represents the frequency response of the FIR filter with impulse response h. Note that choosing M higher than the length of h results in a larger set of points to represent $H(f)$, and hence a smoother plot. Figure 3.5 shows the

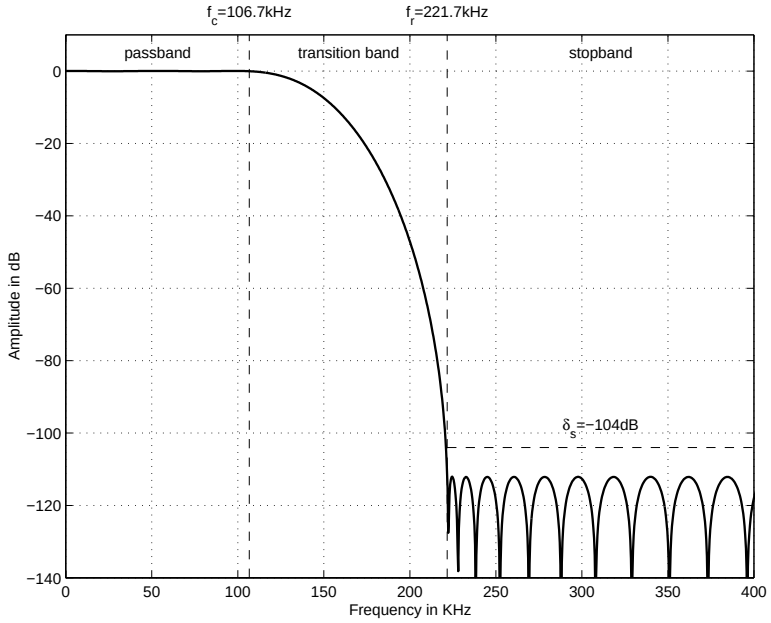


Figure 3.5: Magnitude response of the 256-tap FIR filter.

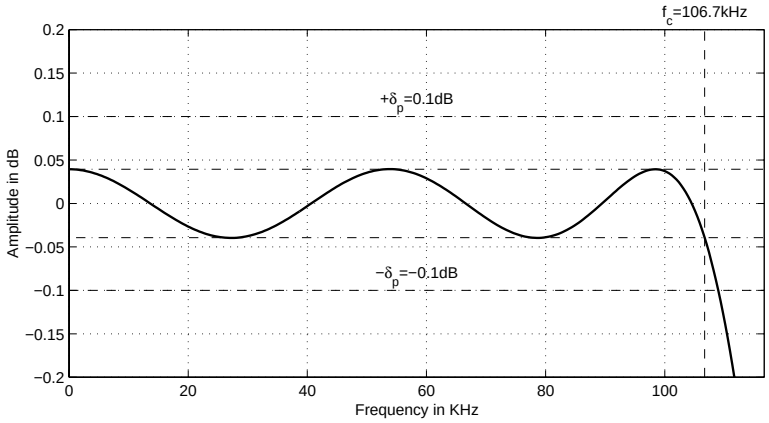


Figure 3.6: Magnitude response in the passband.

magnitude response $|H(f)|$ of our filter for $M = 2^{14}$. It can be seen that the stopband attenuation is about 8dB higher than specified. This is due to running the Remez Exchange Algorithm for more filter taps than necessary. By observing the magnitude response within $[0, f_c]$ depicted in Figure 3.6, we see that choosing $N = 255$ also results in a better approximation in the passband. The ripple is 0.6dB less than previously specified.

We expect our linear-phase filter to have a constant group delay for all frequencies. This can easily be verified by computing the slope of the phase response. From (2.21) we get

$$D(f) = -\frac{d}{df}\angle H(f)) = \frac{N}{2}T = 19.92\mu\text{s}, \tag{3.5}$$

where $T = 1/f_s$ denotes the delay time of a single delay block. Both the phase response and the group delay of our filter are shown in Figure 3.7 for the frequency range $0 \leq f \leq 400$kHz. I can be seen that the phase response is indeed linear, and that the filter yields a constant group delay D. This is true for all frequencies except at those places where $|H(f) = 0|$. At each of these zero locations the phase jumps by π because the sign of $R(f)$, the real-valued part of $H(f)$, changes its sign. But since for these frequencies there is no output contribution the filter is still referred to as having linear phase. See equation (2.20) for further reference.

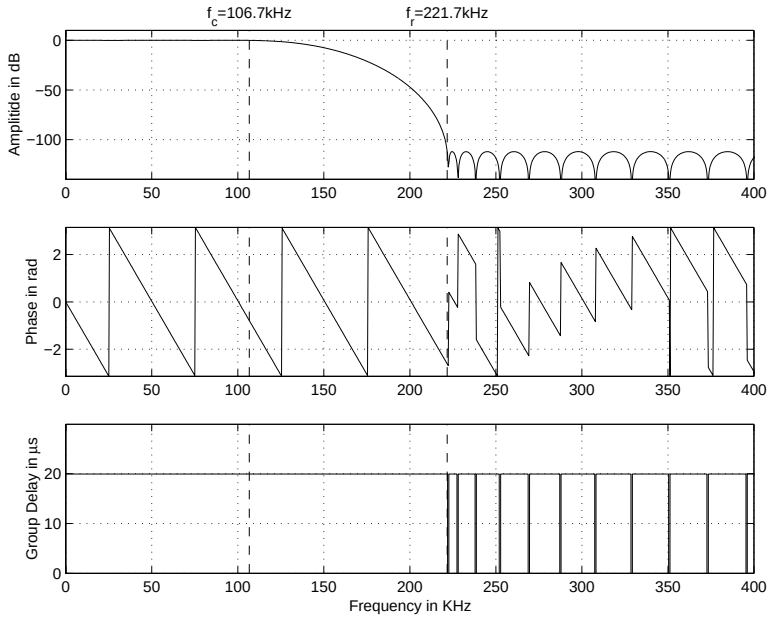


Figure 3.7: Phase response and group delay of the FIR filter.

3.2.3 Simulation of the FIR filter

Once the coefficients for a digital filter are available, its performance can by simulated by the MATLAB® function `y = filter(a,b,x)`, which filters the data in vector x with the filter described by the vectors a and b. The filter is a *Direct Form Transposed* implementation of the standard difference equation (2.14). Hence, the vectors a and b correspond to numerator and denominator of the filter's z-domain transfer function. For an FIR filter the denominator is 1. Recall, that we specified the DC gain of our filter to be 1.5, thus prior to filtering the input signal x, the coefficients h have to be scaled accordingly. The filtered signal is calculated by the following MATLAB® statement:

```
y = filter(1.5*h,1,x);
```

The $\Delta\Sigma$ Modulator output signal x can be generated by using the appropriate functions comprised in the *DelSi Toolbox* [9].

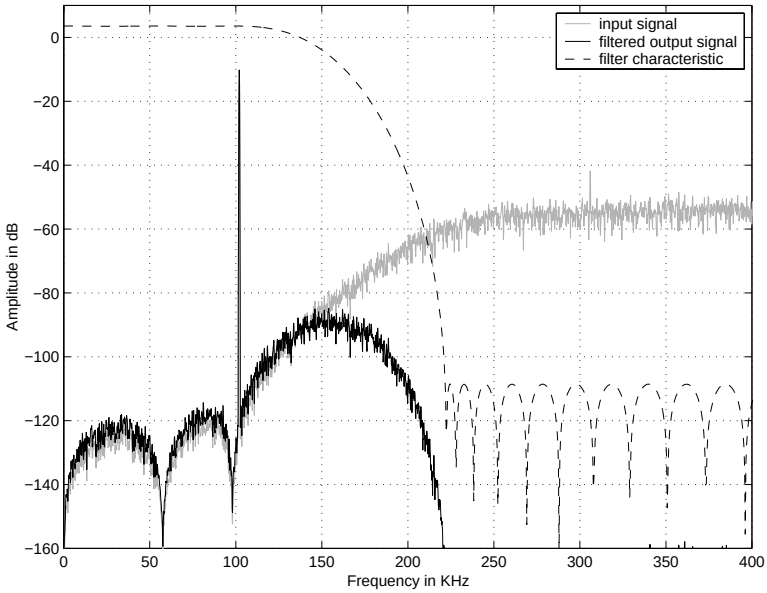


Figure 3.8: Comparison of input signal and filtered output signal.

The performance of the FIR filter is best verified by observing the spectrum of its output signal y, which is obtained by simply computing the FFT of a windowed version of y. Figure 3.8 depicts the comparison of the unfiltered input signal x and the filter output signal y, where the dashed line indicates the magnitude response of the scaled FIR filter. It can be seen that the scaling of the coefficients results in a 3.52dB amplification of all frequency components.

3.3 Quantizing the Filter Coefficients

As shown in Chapter 2.2.5, quantizing the filter coefficients causes a deviation from the unquantized filter response, especially in frequency bands with a high attenuation. The stopband attenuation A_s of our filter is 108dB. As originally specified, we want our filter to have at least 100dB attenuation in the stopband. Consequently, we have a reserve Δ_s of 8dB which determines the maximum allowable deviation in the stopband caused by coefficient quantization. Equation (2.53) describes the lower bound on the minimum number of quantization bits required to guarantee a certain

maximum distortion Δ_s for a given attenuation A_s of the unquantized filter. By plugging the above values into (2.53), we obtain

$$B(N, A_s, \Delta_s) \geq \log_2 \left(1 + \frac{10^{\frac{A_s}{20}}}{10^{\frac{\Delta_s}{20}} - 1} \cdot \sqrt{\frac{2N+1}{3}} \right) = 21.05. \tag{3.6}$$

Hence, by choosing $B = 22$bits, we can guarantee a stopband attenuation of at least 100dB for all frequencies.

The quantization of the filter coefficients stored in the vector h with a quantization range of $\Delta = 1$ can easily be simulated inMATLAB® by the following statement:

```
hq = round(2^B*h)/2^B;
```

While the coefficients in h are stored with full precision, the values stored in h_q are rounded to a precision of B bits. Figure 3.9 shows the magnitude response $|H_q(f)|$ of our filter with coefficients quantized to length of $B = 22$bits . The dashed curve shows the response before coefficient quantization. A lower bound on the stopband

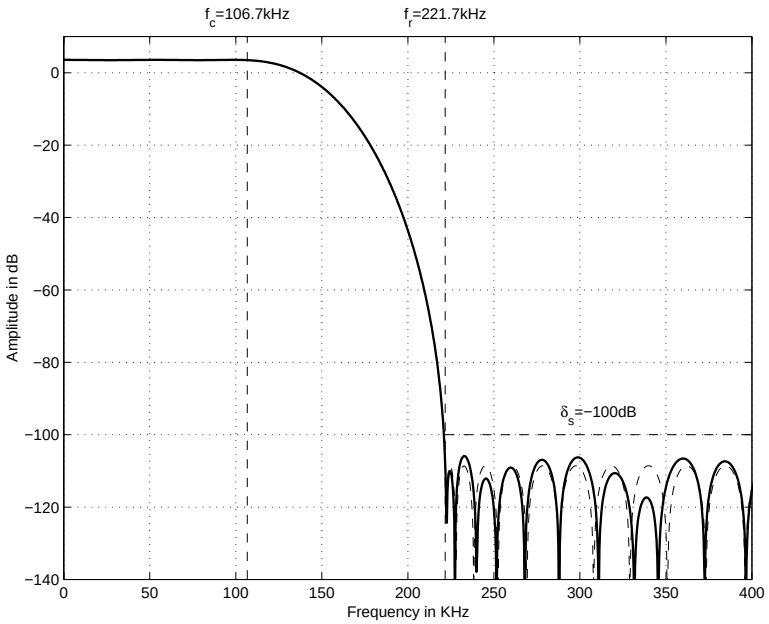


Figure 3.9: Magnitude response of the FIR filter with 22-bit coefficient quantization.

attenuation can be computed by inserting the corresponding values into (2.52). We then get

$$\hat{A}_s(N, B, A_s) \tilde{\geq} -20 \log_{10} \left(10^{-\frac{A_s}{20}} + \frac{1}{2^B - 1} \sqrt{\frac{2N+1}{3}} \right) = 102.98\text{dB}. \tag{3.7}$$

Thus, the filter with 22-bit coefficients still meets the requirement listed in Table 3.2. Equation (2.54) allows the computation of an upper bound of the deviation in the passband. For a given passband gain $A_p = 3.52$db, we obtain

$$A_p - \hat{A}_p \tilde{\leq} 20 \log_{10} \left(1 + \frac{10^{\frac{A_p}{20}}}{2^B - 1} \cdot \sqrt{\frac{2N+1}{3}} \right) = 4.04 \cdot 10^{-5}\text{dB}. \tag{3.8}$$

This value is very small. The plot in Figure 3.10 shows that quantizing the coefficients has indeed no noticeable impact on the filter performance in the passband. It can be seen that the ripple is still 0.4dB, just as it is for the unquantized filter.

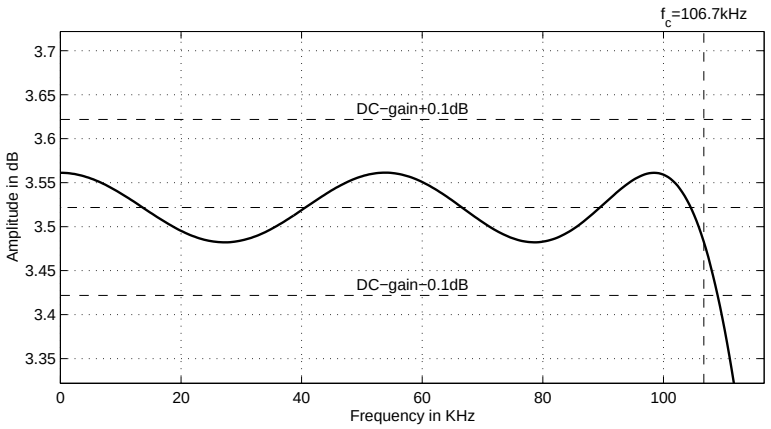


Figure 3.10: Magnitude response in the passband for the filter with 22-bit coefficients.

3.4 Decimation

Once the input signal has been properly filtered to avoid aliasing, it can be decimated by the downsampling factor D to the new sampling frequency $f_d = f_s/D$. The

sampling rate reduction is achieved by forming the sequence $y_d[m]$ by extracting every Dth sample of the filtered output signal $y[n]$. This operation can be written as

$$y_d[m] = y[Dm]. \tag{3.9}$$

In MATLAB® this decimation by D can be performed by the following statement:

```
d  = 1:D:length(y);
yd = y(d);
```

The first command line creates a vector d containing the indices of every Dth sample of y. The second command extracts these samples by copying them in to a new vector y_d. This new vector contains the decimated version of the original signal y. The stem plot in Figure 3.11 shows the decimated impulse response of our filter with an initial sampling frequency of $f_s = 6.4$MHz and a downsampling ratio of $D = 16$. The dashed line indicates the impulse response before decimation. The original signal was represented by 256 samples, the decimated version only consists of 16 samples.

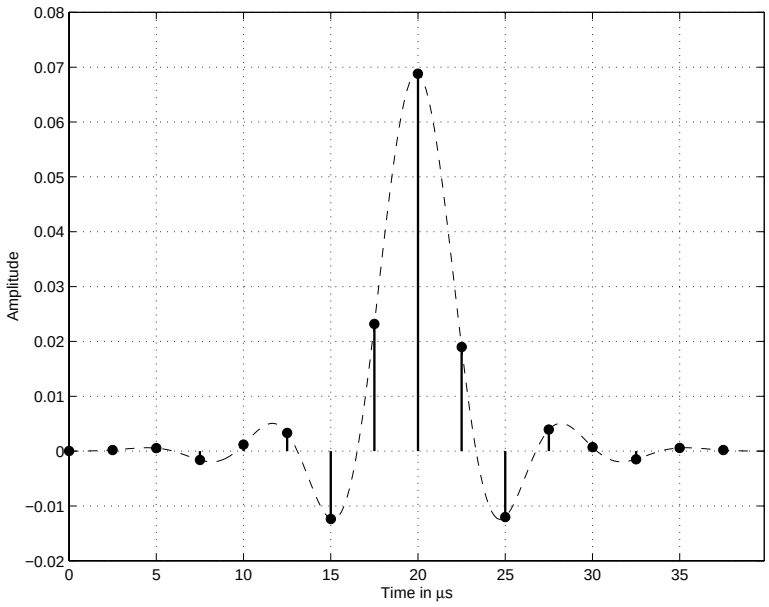


Figure 3.11: Decimated impulse response.

By applying the decimation operation to the filtered $\Delta\Sigma$ Modulator output signal, we obtain a signal with the spectrum depicted in Figure 3.12. It can be seen that the sampling rate of the decimation filter output signal is indeed $f_d = f_s/D = 400\text{kHz}$, because its spectrum is symmetric around the new Nyquist-frequency $f_d/2$.

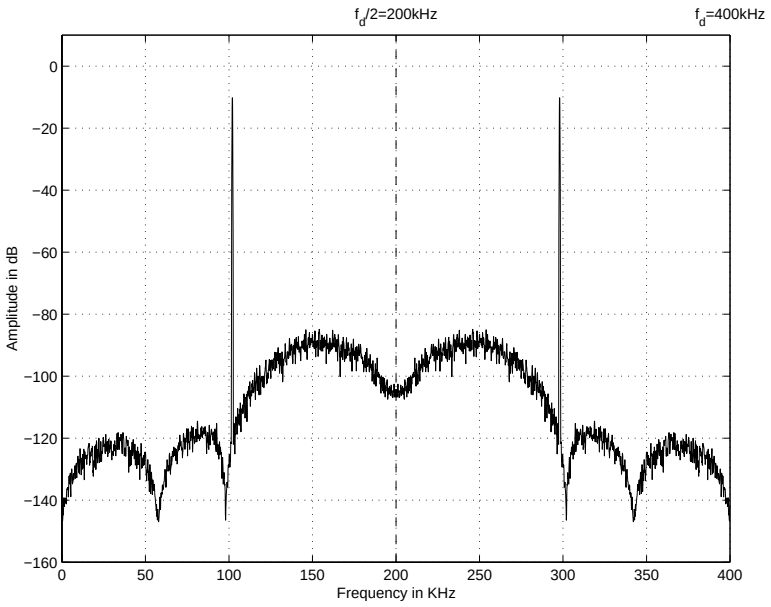


Figure 3.12: Spectrum of the decimation filter output signal for a $\Delta\Sigma$ modulated input signal.

3.5 Digital Hardware Concept

In practice, the design of a digital filter not only involves the computation of filter coefficients but also the development of a hardware based concept to be physically implemented in a later stage. In general, the simulation of a digital circuit is only as good as its underlying model. Thus, it is crucial to find a description as close as possible to the physical implementation. In this section, we will develop a hardware concept which will be implemented in Chapter 4. Doing so enables us to simulate the behavior of the design in MATLAB®İt offers the possibility to simulate the behavior of each functional block within the design structure. It is essential to simulate the

concept as thorough as possible in order to be able to prove the correctness of the performance of the final implementation.

3.5.1 Topology

As shown in Chapter 2.2.4, the most effective topology for the realization of a linear-phase FIR filter is the modified direct form structure depicted in Figure 2.11. Because of the symmetrical impulse response, only half of the coefficients have to be stored. By adding $x[n-k]$ to $x[n-N+k]$ prior to multiplying by the coefficient $h[k]$, a major reduction of components can be achieved. Figure 3.13 shows the direct form structure for our previously designed linear-phase FIR filter of order $N = 255$.

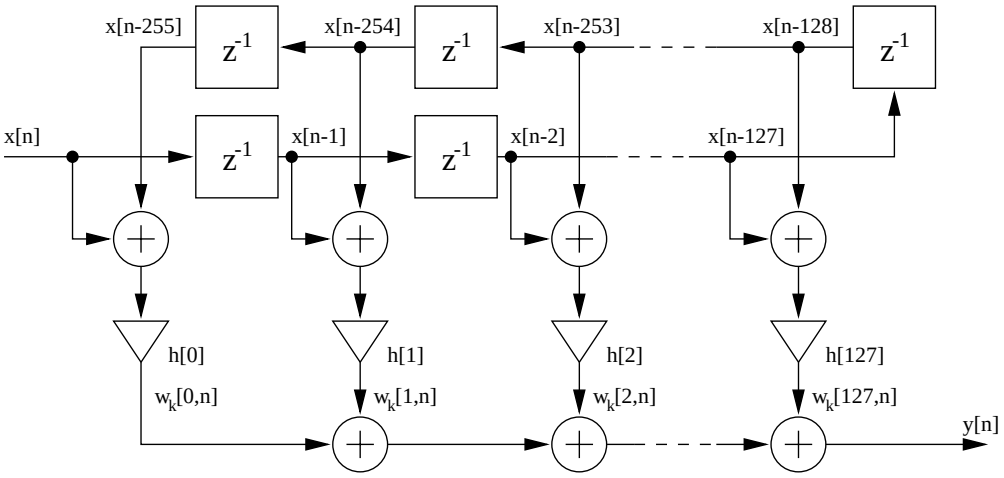


Figure 3.13: Modified direct form structure for the linear-phase FIR filter of order $N = 255$.

The input signal $x[n]$ is the 1-bit quantized output signal of a $\Delta\Sigma$ Modulator. As discussed in Chapter 2.1, the signal $x[n]$ can only assume the binary values 0 and 1, which represent the voltages $+V_{ref}$ and $-V_{ref}$, respectively. Hence, the values $w_k[n]$ at the output of the multipliers are limited to the set $\{+2h[k], 0, -2h[k]\}$, corresponding to the voltages $\{+2V_{ref}h[k], 0, -2V_{ref}h[k]\}$. Table 3.3 lists these values and its conditions on the input signal.

It can be seen that $w_k[n] \neq 0$ only if $x[n-k] = x[n-N+k]$, otherwise $w_k[n]$

$x[n-k]$	$x[n-N+k]$	$w_k[n]$
0	0	$-2h[k]$
0	1	0
1	0	0
1	1	$+2h[k]$

Table 3.3: Truth table for the multiplier outputs $w_k[n]$.

will always be 0. Since the contribution to the output signal is limited to $+2h[k]$, $-2h[k]$, or 0, this operation can be implemented without the use of multipliers. For $x[n-k] \neq x[n-N+k]$, there is no contribution, and hence, no action is required. For $x[n-k] = x[n-N+k]$, the value $2h[k]$ has to be either added to or subtracted from the final output value, depending on the value of $x[n-k]$ and $x[n-N+k]$. Figure 3.14 shows the multiplier-free implementation of the FIR filter.

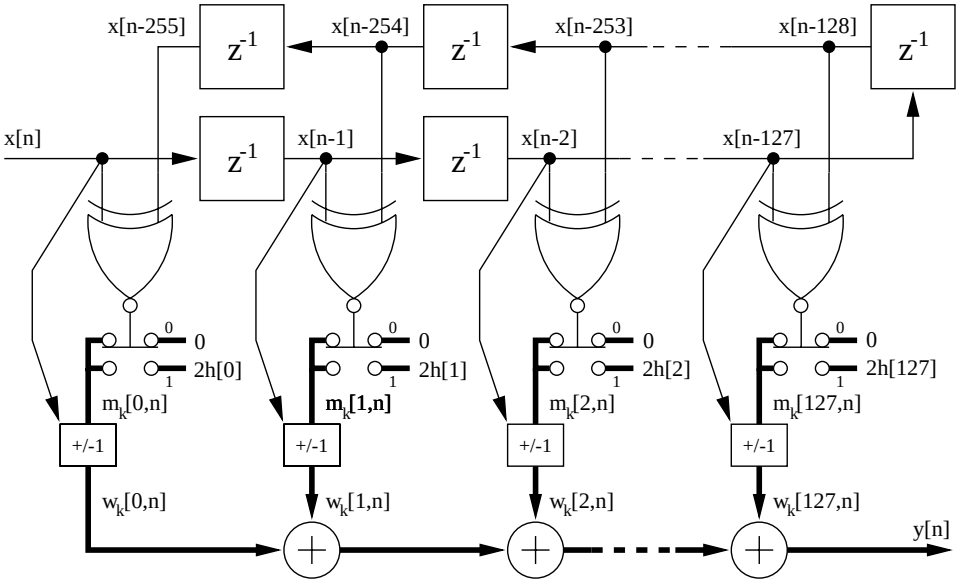


Figure 3.14: Modified direct form structure for the linear-phase FIR filter for single-bit input signals.

Note that the thin lines represent single-bit signals while the thicker lines indicate

multi-bit signals. Since the input signal $x[n]$ shifted trough the N delays is a single-bit signal, the delay-line for our filter can simply be implemented by a 255-bit shift register with serial input and parallel output. The decision, whether $x[n-k]$ and $x[n-N+k]$ are equal or different, is realized by an exclusive-nor logic gate. The output value of this gate then determines whether $m_k[n] = 2h[k]$ or $m_k[n] = 0$ is fed into a sign-inverter. The sign-inverter negates its input value $m_k[n]$ if $x[n-k] = 0$ or passes it trough as is if $x[n-k] = 1$. Table 3.4 lists the states of the signals involved for the multiplier-free computation of $w_k[n]$.

$x[n-k]$	$x[n-N+k]$	$\overline{x[n-k] \oplus x[n-N+k]}$	$m_k[n]$	$w_k[n]$
0	0	1	$2h[k]$	$-2h[k]$
0	1	0	0	0
1	0	0	0	0
1	1	1	$2h[k]$	$+2h[k]$

Table 3.4: Extended truth table for the multiplier outputs $w_k[n]$.

The signal $m_k[n]$ is obtained by either loading $2h[k]$ from the coefficient memory or setting it to 0. An easy way of realizing this with logic gates is presented in Chapter 4. To avoid multiplying the coefficient $h[k]$ by 2, we store $2h[k]$ in the coefficient memory. This requires one additional memory bit for each coefficient. The topology of the sign-inverter depends on the number format employed to store the coefficients, which we will discuss later in this section. We can then express $w_k[n]$ as

$$w_k[n] = \overline{x[n-k] \oplus x[n-N+k]} \cdot 2h[k] \cdot s(x[n-k]), \tag{3.10}$$

where $s(x[n-k])$ denotes the sign-inversion

$$s(x[n-k]) = \begin{cases} +1, & x[x-k] = 1 \\ -1, & x[n-k] = 0 \end{cases} . \tag{3.11}$$

Consequently, the expression for the filter output $y[n]$ is

$$y[n] \quad = \quad \sum_{k=0}^{\frac{N-1}{2}} \overline{x[n-k] \oplus x[n-N+k]} \cdot 2h[k] \cdot s(x[n-k])$$

$$= \sum_{k=0}^{\frac{N-1}{2}} w_k[n]. \tag{3.12}$$

The entire circuit operates at the input sampling rate $f_s = 6.4$MHz, i.e., for each new input sample $x[n]$ being shifted into the delay-line, a new output sample $y[n]$ is generated.

Decimation by D

Instead of extracting every Dth sample from the undecimated filter-output signal, a decimation by D can also be achieved by only calculating one result every D clock cycles. The sampling rate f_d of the downsampled output signal $y[m]$ is then $f_d = f_s/D$, and the expression for $y[m]$ is

$$y[Dm] = \sum_{k=0}^{\frac{N-1}{2}} \overline{x[Dm-k] \oplus x[Dm-N+k]} \cdot 2h[k] \cdot s(x[Dm-k]). \tag{3.13}$$

Unfortunately, this reduction of computations does not lead to a reduction of required computing speed or hardware effort. Because the status of the shift register only remains constant for one period $T = 1/f_s$ of the input sampling clock, the summation involving $\frac{N-1}{2}$ two-port adders still has to be performed within that time. However, a significant reduction of hardware effort can be achieved by replacing groups of D adjacent two-port adders by one accumulator, which adds D numbers within D cycles of the input sampling clock f_s The accumulator thus generates results at the decimated clock rate f_d. It is beneficial to choose powers of 2 for both the number of coefficients $(N+1)$ and the downsampling ratio D. This guarantees that the originally needed $\frac{N-1}{2}$ adders can be replaced by exactly $L = \frac{N+1}{2D}$ accumulators, each adding exactly D numbers. Furthermore, it allows the simple derivation of all possible needed clock signals from the master clock of frequency f_s by ordinary binary counters, as we will see in Chapter 4.

The result $v_l[Dm]$ of the lth accumulator at the time $Dm \cdot T$ can be written as

$$v_l[Dm] = \sum_{k=D(l-1)}^{Dl-1} \overline{x[Dm-k] \oplus x[Dm-N+k]} \cdot 2h[k] \cdot s(x[Dm-k]) \tag{3.14}$$

for $l = 1 \ldots L$. The final output signal $y[Dm]$ is then computed by adding the results of the L accumulators

$$y[Dm] = \sum_{l=1}^{L} v_l[Dm]. \tag{3.15}$$

Since each of the L accumulators only accumulates one summand per master clock cycle, it has to be taken into account that the samples $x[Dm-k]$ and $x[Dm-N+k]$, needed to compute each of the D summands, are shifted to the next delay after one clock cycle. For the analysis of the accumulation procedure, let us consider the first accumulator of our filter with $N = 255$ and $D = 16$. It performs the following calculation:

$$v_1[16m] = \sum_{k=0}^{15} \overline{x[16m-k] \oplus x[16m-255+k]} \cdot 2h[k] \cdot s(x[16m-k]), \tag{3.16}$$

Accordingly, the samples needed for the computation of $v_1[16m]$ are

$$x[16m], \ldots, x[16m-15] \text{ and } x[16m-240], \ldots, x[16m-255].$$

Let $d_0, \ldots, d_{255}$ be the labels for the 256 taps of our delay-line. At the time $16m \cdot T$, when the final result $v_1[16m]$ of the accumulation has been generated, the samples $x[16m], \ldots, x[16m-255]$ are available at the taps $d_0, \ldots, d_{255}$, respectively. By using the tap numbers, equation (3.16) can be written as

$$v_1[16m] = \sum_{k=0}^{15} \overline{d_k \oplus d_{255-k}} \cdot 2h[k] \cdot s(d_k). \tag{3.17}$$

Since the accumulator can only accumulate one summand per clock cycle, the accumulation requires 16 clock cycles, of which $(16m-15)$ is the first, and $16m$ is the last. During the first cycle $(16m-15)$, the sample $x[16m-15]$ is available at d_0 and the corresponding sample $x[16m-240]$ is available at d_{225}. Thus, the first summand to be accumulated is $\overline{d_0 \oplus d_{225}} \cdot 2h[15] \cdot s(d_0)$. During the next clock cycle $(16m-14)$, the samples $x[16m-14]$ and $x[16m-241]$, which are then available at the taps d_0 and d_{227}, are used together with the coefficient $2h[14]$ to calculate the next summand to be added to the previous result. This is done for all the 16 summands. It is obvious, that due to the shift of the samples at each clock cycle, the first of the two samples

connected to the exclusive-nor is always available at the delay tap d_0, and that the index for the tap storing the second sample has to be increased by 2 for each cycle. During the last cycle $16m$, the last summand is accumulated and the final result is stored in an output register to make it available to the adder block that sums the final results of all $L = \frac{N+1}{2D} = 8$ accumulators. Table 3.5 shows the cycle-by-cycle operation of the first accumulator of our filter.

Cycle	Operation
$16m-15$	Clear Accumulator add $\overline{d_0 \oplus d_{225}} \cdot s(d_0) \cdot 2h[15]$
$16m-14$	add $\overline{d_0 \oplus d_{227}} \cdot s(d_0) \cdot 2h[14]$
$16m-13$	add $\overline{d_0 \oplus d_{229}} \cdot s(d_0) \cdot 2h[13]$
$16m-12$	add $\overline{d_0 \oplus d_{231}} \cdot s(d_0) \cdot 2h[12]$
$16m-11$	add $\overline{d_0 \oplus d_{233}} \cdot s(d_0) \cdot 2h[11]$
$16m-10$	add $\overline{d_0 \oplus d_{235}} \cdot s(d_0) \cdot 2h[10]$
$16m-9$	add $\overline{d_0 \oplus d_{237}} \cdot s(d_0) \cdot 2h[9]$
$16m-8$	add $\overline{d_0 \oplus d_{239}} \cdot s(d_0) \cdot 2h[8]$
$16m-7$	add $\overline{d_0 \oplus d_{241}} \cdot s(d_0) \cdot 2h[7]$
$16m-6$	add $\overline{d_0 \oplus d_{243}} \cdot s(d_0) \cdot 2h[6]$
$16m-5$	add $\overline{d_0 \oplus d_{245}} \cdot s(d_0) \cdot 2h[5]$
$16m-4$	add $\overline{d_0 \oplus d_{247}} \cdot s(d_0) \cdot 2h[4]$
$16m-3$	add $\overline{d_0 \oplus d_{249}} \cdot s(d_0) \cdot 2h[3]$
$16m-2$	add $\overline{d_0 \oplus d_{251}} \cdot s(d_0) \cdot 2h[2]$
$16m-1$	add $\overline{d_0 \oplus d_{253}} \cdot s(d_0) \cdot 2h[1]$
$16m$	add $\overline{d_0 \oplus d_{255}} \cdot s(d_0) \cdot 2h[0]$ Store result in output register

Table 3.5: Cycle-by-cycle operation of the first accumulator.

Consequently, the operation of the first accumulator can be written as

$$v_1[16m] = \sum_{k=-15}^{0} \overline{d_0 \oplus d_{255+2k}} \cdot 2h[-k] \cdot s(d_0). \tag{3.18}$$

Note that (3.18) is only valid if the samples stored in the delay-line are shifted by one tap after each increment of the summation index k.

The fact that the tap containing the first of the two samples used to generate a summand remains the same during all the clock cycles allows the use of a single exclusive-nor gate. The second sample can then be picked out of the delay-line by a simple D-bit multiplexer. Figure 3.15 depicts the hardware concept for the first accumulator section of our filter. Both the 16 coefficient memory cells and the 16

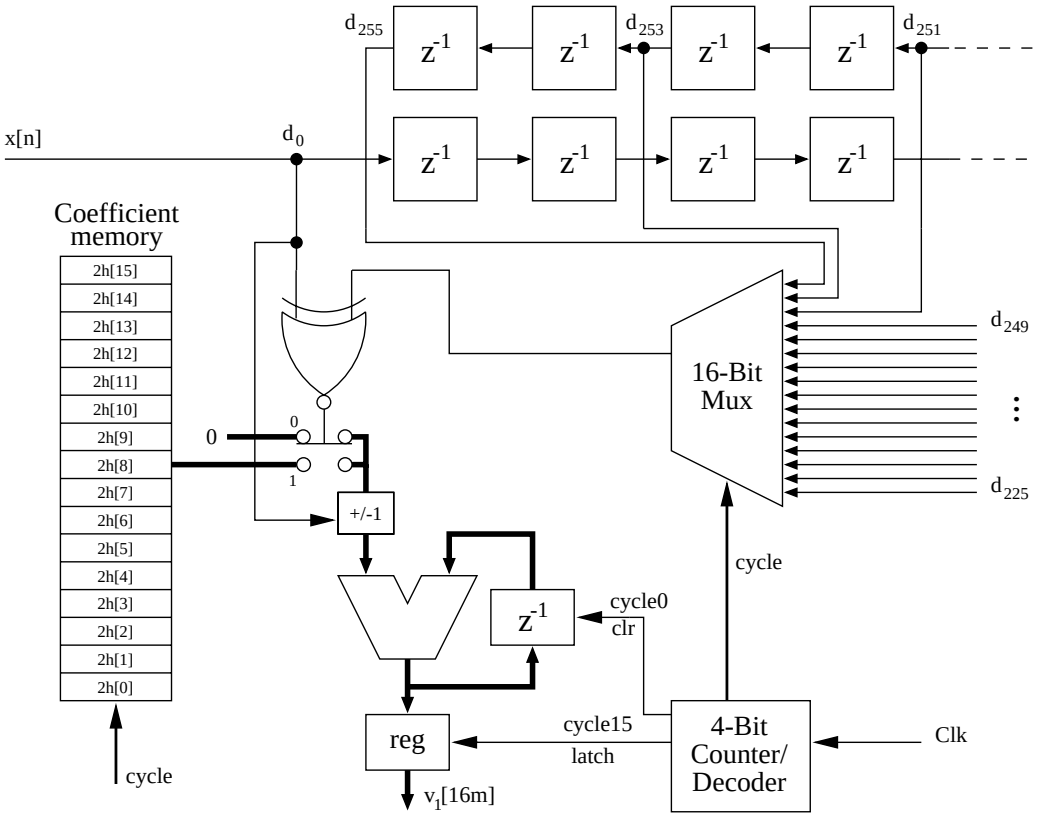


Figure 3.15: Direct form structure for the first accumulator stage of our filter.

inputs of the multiplexer are addressed by the output signals of a 4-bit counter, which counts the clock cycles of the master clock. Because of the order in which the individual summands are computed, the coefficients have to be stored in descending order, i.e., $2h[15]$ is stored in the first and $2h[0]$ is stored in the last memory cell.

At the beginning of the accumulation, during the first cycle, the delay register of the accumulator has to be cleared. During the last cycle, once the final result is accumulated, it is latched into the output register.

From (3.18) we can derive the general case. The lth accumulator performs the following computation

$$v_l[Dm] = \sum_{k=-Dl+1}^{-D(l-1)} \overline{d_{D(l-1)} \oplus d_{N+2k+D(l-1)}} \cdot 2h[-k] \cdot s(d_{D(l-1)}). \tag{3.19}$$

The only requirement on N and D for a hardware efficient implementation is that $N+1 = 2^n$ and $D = 2^d$, i.e., integer powers of 2. The filter structure can then be separated into $L = \frac{N+1}{2D}$ accumulation sections that perform D additions each. Figure 3.16 shows the functional block representation of the lth accumulation section. Each of these sections has one input (in_0) connected to a fixed tap of the delay line,

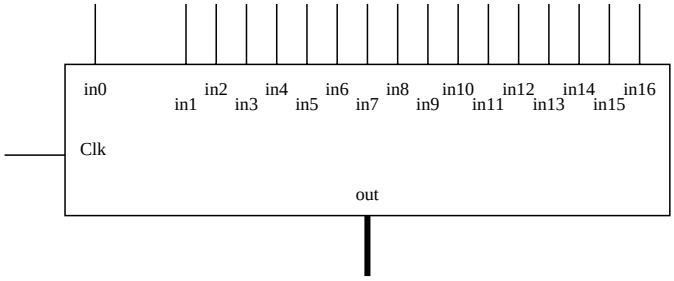


Figure 3.16: Functional block for the lth accumulation section.

and D multiplexer inputs ($in_1 \ldots in_D$), all connected to specific filter taps separated by two delays. The outputs of all L sections are summed by a summation node to generate the final filter output signal $y[Dm]$. Since every section latches its output result, it will be available at the output for D clock cycles. Hence, this addition can be realized by a simple accumulator that loads the individual results from the accumulation sections, and sums them sequentially. This accumulator should also be followed by a data register which is updated every D cycles with a new final result.

In the process of implementing the hardware concept, it is crucial to know, how the delay-line taps and the inputs of the individual accumulation sections have to be

connected. The number of the fixed tap, denoted d_{n_0}, to which the input in_0 of the l-th accumulation section has to be connected, can be determined as follows:

$$n_0 = D(l-1), \qquad l = 1 \ldots L \tag{3.20}$$

The number of other delay-line taps, denoted $d_{n_1} \ldots d_{n_D}$, to which the D multiplexer inputs $in_1 \ldots in_D$ of the lth section have to be connected, is obtained by the following computation

$$n_k = N - D(l+1) + 2k, \qquad l = 1 \ldots L, \quad k = 1 \ldots D. \tag{3.21}$$

Table 3.6 lists the inputs and the corresponding delay-line taps for all the 8 accumulation sections of our filter with $N = 255$ and $D = 16$.

	1	2	3	4	5	6	7	8
in_0	d_0	d_{16}	d_{32}	d_{48}	d_{64}	d_{80}	d_{96}	d_{112}
in_1	d_{225}	d_{209}	d_{193}	d_{177}	d_{161}	d_{145}	d_{129}	d_{113}
in_2	d_{227}	d_{211}	d_{195}	d_{179}	d_{163}	d_{147}	d_{131}	d_{115}
in_3	d_{229}	d_{213}	d_{197}	d_{181}	d_{165}	d_{149}	d_{133}	d_{117}
in_4	d_{231}	d_{215}	d_{199}	d_{183}	d_{167}	d_{151}	d_{135}	d_{119}
in_5	d_{233}	d_{217}	d_{201}	d_{185}	d_{169}	d_{153}	d_{137}	d_{121}
in_6	d_{235}	d_{219}	d_{203}	d_{187}	d_{171}	d_{155}	d_{139}	d_{123}
in_7	d_{237}	d_{221}	d_{205}	d_{189}	d_{173}	d_{157}	d_{141}	d_{125}
in_8	d_{239}	d_{223}	d_{207}	d_{191}	d_{175}	d_{159}	d_{143}	d_{127}
in_9	d_{241}	d_{225}	d_{209}	d_{193}	d_{177}	d_{161}	d_{145}	d_{129}
in_{10}	d_{243}	d_{227}	d_{211}	d_{195}	d_{179}	d_{163}	d_{147}	d_{131}
in_{11}	d_{245}	d_{229}	d_{213}	d_{197}	d_{181}	d_{165}	d_{149}	d_{133}
in_{12}	d_{247}	d_{231}	d_{215}	d_{199}	d_{183}	d_{167}	d_{151}	d_{135}
in_{13}	d_{249}	d_{233}	d_{217}	d_{201}	d_{185}	d_{169}	d_{153}	d_{137}
in_{14}	d_{251}	d_{235}	d_{219}	d_{203}	d_{187}	d_{171}	d_{155}	d_{139}
in_{15}	d_{253}	d_{237}	d_{221}	d_{205}	d_{189}	d_{173}	d_{157}	d_{141}
in_{16}	d_{255}	d_{239}	d_{223}	d_{207}	d_{191}	d_{175}	d_{159}	d_{143}

Table 3.6: List of inputs and corresponding delay-line taps for our filter.

Coefficient Memory

Due to the separation of the filter architecture into L sections, the $\frac{N+1}{2}$ coefficients that have to be stored are divided into L memory blocks, each containing D coefficients. Each block is assigned to the corresponding accumulation section. Figure 3.17 shows the coefficient set for our filter. Note that these coefficients are already

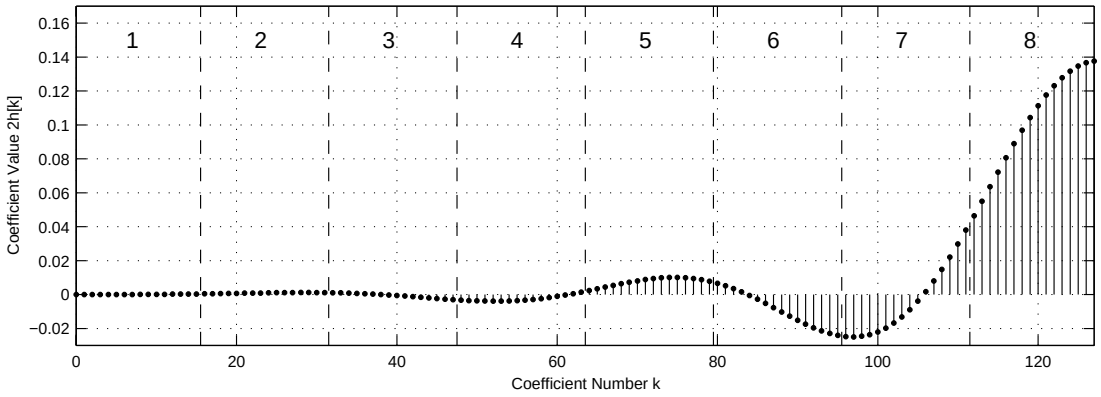


Figure 3.17: First half of the filter coefficients $2h[n]$ to be stored.

scaled by 1.5 (see section 3.1.3 for reference), quantized, and multiplied by 2. The dashed lines indicate the separation into 8 sections. Figures 3.4 and 3.17 show that the values of the coefficients closer to the middle of the impulse response are larger than the ones that are further away. The values in section 1 are much smaller than the values in section 8. Consequently, less memory-bits are needed to store the coefficients of section 1 compared to section 8.

Assuming a quantization range of $\Delta = 1$ and a quantization bit-depth of B, the number b_k of bits required to store the magnitude of the coefficient $2h[k]$ is

$$b_k = \lceil \log_2(|2h[k]| \cdot 2^B) \rceil, \tag{3.22}$$

where the delimiters $\lceil\,\rceil$ denote *next higher integer*. Equation (3.22) shows that the full resolution $b_k = B$ is only needed if $|2h[k] = 1|$. If $|2h[k]| < 1$ then $b_k < B$.

With (3.22) the required number of bits b_k can be calculated for every value $|2h[k]|$. If the coefficient set consists of both positive and negative numbers, an

additional bit is needed to store the sign of the corresponding coefficient. In order to maintain simplicity of the implementation, all the coefficients within one section are stored with the same memory word-length. If they all have the same sign, no additional sign-bit is required. Instead, the sign-information can be 'hard-wired' inside the accumulation section.

Figure 3.18 depicts a $\log_2$-plot of the coefficient magnitudes $|2h[k]|$ of our filter. The numbers on top of the graph represent the number of memory-bits chosen for

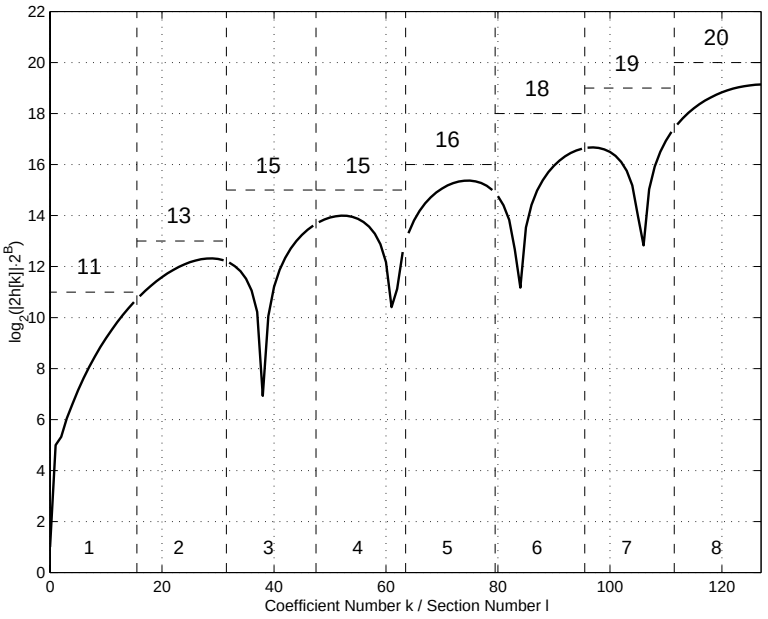


Figure 3.18: Number of memory bits needed to store the filter coefficient for each section.

the coefficient memories of the individual sections. The notches between the lobes of the $\log_2$-plot indicate that the sign of the coefficients changes at these positions. If such a sign-inversion occurs within a section, an additional bit is needed to represent the sign of the coefficients. This is the case for the sections 3,4,6 and 7, while the coefficients of the other sections are strictly positive.

One of the most frequently used number formats in the field of digital signal processing is the *two's complement* format, because it allows the addition of both

positive and negative binary numbers with simple binary adders. Also, the complement $-n$ for a given number n can easily be computed by inverting all the bits of the binary representation of n and adding 1. To illustrate this, let us consider the number range $-8, \ldots, 7$ which can be represented by the 4-bit two's complement format, as shown in Table 3.7. The positive numbers $0 \ldots 7$ can be represented by just 3 bits. By extending the range to negative numbers, an additional bit is needed. The sign-bit in the two's complement format is the MSB (most significant bit), which is 0 for positive and 1 for negative numbers. To show the computation of the complement $-n$ for a given number n, let us consider $n = 7$. The binary representation of 7 is 0111. This number can be inverted by simply inverting all the bits of 0111 and adding 1:

n	binary
7	0111
$\vdots$	$\vdots$
1	0001
0	0000
-1	1111
$\vdots$	$\vdots$
-8	1000

Table 3.7: Two's complement representation of the number range $-8, \ldots, 7$.

$$-7 \hat{=} \overline{0111} + 1 = 1000 + 1 = 1001$$

In MATLAB® the binary representation of a number can be computed by the command `Cbin = dec2bin(C,b)` which converts the decimal integer C to a binary string with at least b bits, where C must be a non-negative integer smaller than 2^{52}. Therefore, our quantized coefficients first have to be converted into integer values. This is accomplished by the command `H = 2*h*2^B`. Since all coefficients h have been quantized to $B = 22$bits, the result H will be an integer number. Because `dec2bin` can only convert positive integers, the negative coefficients have to be handled differently. The following MATLAB® routine converts the coefficient $2h[k]$

to its corresponding two's complement representation H_{bin} with a word-length of b_l bits:

```
if h(k) >= 0
    Hbin = dec2bin(2*h(k)*2^B,b(l));
else
    Hbin = dec2bin(1+bitcmp(abs(2*h(k)*2^B),b(l)),b(l));
end;
```

The function `Cc = bitcmp(C,b)` returns the bit-wise complement of C as a b-bit non-negative integer. By adding 1 to C_c and then converting it to a binary number, we obtain the binary b-bit representation of $-C$.

The two's complement coefficients for our filter are listed in appendix A. Note that the bit-depth b_l chosen for the conversion is the same for all coefficients within the lth accumulation section.

Once the coefficients are converted to the binary format, they have to be assigned to the corresponding sections in the correct order. The number $n_{l,i}$ of the ith coefficient in the lth section is

$$n_{l,i} = Dl - 1 - i, \qquad l = 1 \ldots L, \quad i = 0 \ldots D - 1. \tag{3.23}$$

Table 3.8 shows the assignment of the coefficients $h[n_{l,i}]$ to the coefficient memories for our filter. Recall that because of the order in which the accumulations are performed, the coefficients are stored in reverse order.

Accumulator Word Length

The accumulators in each of the L sections sum exactly D numbers. As discussed earlier in this section, these numbers can only assume the values $+2h[k]$, 0, or $-2h[k]$. It is important that the word-length of the adders and the delay registers comprising the accumulators are chosen such that no overflow can occur. The output result $v_l[Dm]$ of the lth accumulator can be upper-bounded by

$$\begin{aligned} |v_l[Dm]| &= \left| \sum_{k=D(l-1)}^{Dl-1} \overline{x[Dm-k] \oplus x[Dm-N+k]} \cdot 2h[k] \cdot s(x[Dm-k]) \right| \\ &\leq \sum_{k=D(l-1)}^{Dl-1} |2h[k]|. \end{aligned} \tag{3.24}$$

l	1	2	3	4	5	6	7	8
b_l	11	13	15	15	16	18	19	20
$i = 0$	$h[15]$	$h[31]$	$h[47]$	$h[63]$	$h[79]$	$h[95]$	$h[111]$	$h[127]$
1	$h[14]$	$h[30]$	$h[46]$	$h[62]$	$h[78]$	$h[94]$	$h[110]$	$h[126]$
2	$h[13]$	$h[29]$	$h[45]$	$h[61]$	$h[77]$	$h[93]$	$h[109]$	$h[125]$
3	$h[12]$	$h[28]$	$h[44]$	$h[60]$	$h[76]$	$h[92]$	$h[108]$	$h[124]$
4	$h[11]$	$h[27]$	$h[43]$	$h[59]$	$h[75]$	$h[91]$	$h[107]$	$h[123]$
5	$h[10]$	$h[26]$	$h[42]$	$h[58]$	$h[74]$	$h[90]$	$h[106]$	$h[122]$
6	$h[9]$	$h[25]$	$h[41]$	$h[57]$	$h[73]$	$h[89]$	$h[105]$	$h[121$
7	$h[8]$	$h[24]$	$h[40]$	$h[56]$	$h[72]$	$h[88]$	$h[104]$	$h[120]$
8	$h[7]$	$h[23]$	$h[39]$	$h[55]$	$h[71]$	$h[87]$	$h[103]$	$h[119]$
9	$h[6]$	$h[22]$	$h[38]$	$h[54]$	$h[70]$	$h[86]$	$h[102]$	$h[118]$
10	$h[5]$	$h[21]$	$h[37]$	$h[53]$	$h[69]$	$h[85]$	$h[101]$	$h[117]$
11	$h[4]$	$h[20]$	$h[36]$	$h[52]$	$h[68]$	$h[84]$	$h[100]$	$h[116]$
12	$h[3]$	$h[19]$	$h[35]$	$h[51]$	$h[67]$	$h[83]$	$h[99]$	$h[115]$
13	$h[2]$	$h[18]$	$h[34]$	$h[50]$	$h[66]$	$h[82]$	$h[98]$	$h[114]$
14	$h[1]$	$h[17]$	$h[33]$	$h[49]$	$h[65]$	$h[81]$	$h[97]$	$h[113]$
15	$h[0]$	$h[16]$	$h[32]$	$h[48]$	$h[64]$	$h[80]$	$h[96]$	$h[112]$

Table 3.8: Assignment of the coefficients to the coefficient memories.

Therefore, the maximum number of bits needed for an accumulation can be obtained by computing the number of bits required to represent the sum of all $|2h[k]|$ of an accumulation section. Note that not only a positive, but also a negative maximum can occur, in case all summands are negative. This requires an additional bit. Hence, the number of bits needed to prevent overflow in the lth accumulator is computed as follows

$$a_l = 1 + \left\lceil \log_2 \left(2^B \cdot \sum_{k=D(l-1)}^{Dl-1} |2h[k]| \right) \right\rceil . \tag{3.25}$$

Figure 3.19 depicts a $\log_2$-plot of the accumulations performed in each section. The segments represent the growth of each accumulator sum during the accumulation process for the case that all the summands are positive. The last and highest value

in each segment is the highest number that can occur, i.e., the term in parentheses in (3.25). The numbers on top of the graph represent the adder-width chosen for the accumulator of the corresponding section.

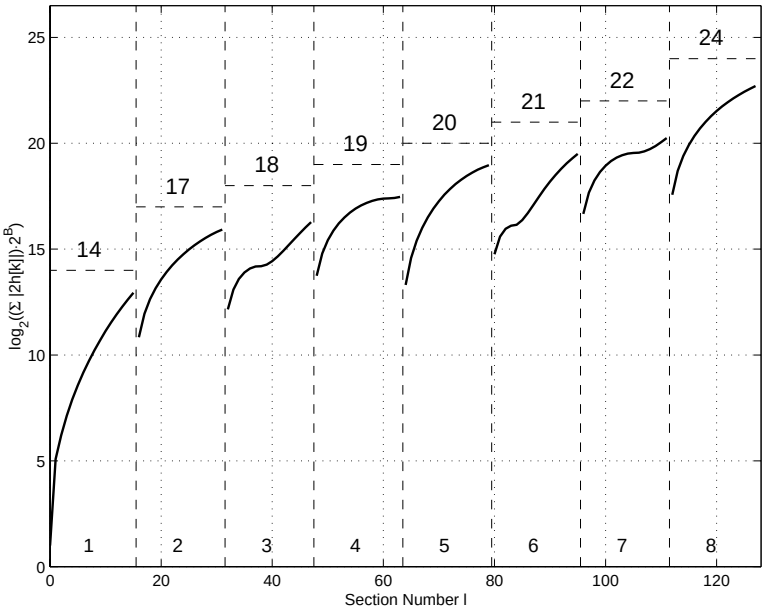


Figure 3.19: Maximum number of bits needed for the accumulation in individual sections.

Finally, the adder-width for the accumulator that adds the results $v_l[Dm]$ of the L sections has to be determined. Just as for the other accumulators, the upper bound for $y[Dm]$ could be computed and used to calculate the number of bit required to avoid overflow. But due to the nature of the input signal $x[n]$ and the fact that it is limited to $\pm\frac{1}{2}V_{ref}$, we are able to compute a less pessimistic bound of the output signal. Assuming an input signal $x[n]$ to be equal to 1 for all n, which corresponds to a DC signal of $+V_{ref}$, we obtain

$$y[Dm] = \sum_{k=0}^{N} 2h[k] = 1.5068 \tag{3.26}$$

for the output signal, which is also a DC signal. Note that because of the coefficient scaling by 1.5, the DC gain for our filter is also 1.5, as originally specified. The

error of 0.0068 is caused by coefficient quantization. Consequently, the number of adder-bits required for the final accumulator is

$$a_{out} = 1 + \lceil \log_2(2^B \cdot 1.5068) \rceil = 1 + \lceil 22.59 \rceil = 24. \tag{3.27}$$

Truncation of the Output Signal

Once the final result is computed, the resolution of the result can be reduced to a more reasonable word-length, before it is fed into a digital beamformer or an additional filter stage. The quantization to the new word-length $\tilde{a}_{out}$ is simply realized in MATLAB® by the statement

```
y_t = floor(y*2^(a_out_t-1))/2^(a_out_t-1);
```

where `a_out_t` denotes $\tilde{a}_{out}$. This command reduces the precision of the magnitude of y to $\tilde{a}_{out} - 1$ bits. Since y can assume both positive and negative values, the effective precision of y is $\tilde{a}_{out}$ bits. The choice of $\tilde{a}_{out}$ is determined by the desired dynamic range of the signal of interest. According to the specifications [8], the $\Delta\Sigma$ modulator used in our application has a maximum narrow-band signal-to-noise ratio (SNR) of 97dB, which is equivalent to a precision of 16 bits, hence $\tilde{a}_{out} = 16$.

3.5.2 Behavioral Simulation

A behavioral simulation of the filter can be accomplished by converting the functional description of the hardware concept derived in the last section into equivalent MATLAB® scripts. The following sub-sections describe this transition for the general case of an Nth order linear-phase FIR filter with single-bit input.

Shift Register

In general, a data register such as an N-bit shift register can be represented in MATLAB® by a vector with N elements, each containing one sample of the input signal. Since the sample present at the filter tap d_0 at the input of the shift register has to be stored as well, an additional element is required. Hence, we need a vector with $N + 1$ elements. This vector is generated by the following command:

```
d = zeros(1,N+1);
```

This statement initializes the vector d by setting all $N+1$ elements to 0. This corresponds to their initial value. Note that these elements can only be addressed by their number, i.e., $d_0 =$ `d(1)`, $\ldots d_k =$ `d(k+1)`. In order to be able to address the filter taps d_{n_0} and d_{n_k} needed for the computations, the tap numbers n_0 and n_k have to be computed and stored. By employing (3.20), we can compute the tap numbers n_0 for all L sections. The following statement creates a vector with L elements holding the corresponding tap numbers:

```
l   = 1:L;
n_0 = D*(l-1)+1;
```

Accordingly, a matrix storing the D numbers of the filter taps serving as multiplexer inputs in each of the L sections can be generated by employing (3.21) as follows:

```
[k,l] = meshgrid(1:D,1:L);
n_k   = N-D*(l+1)+2*k+1;
```

When performing the calculations, the currently needed samples at the k cycle in the lth section are `d(n_0(l))` and `d(n_k(k,l))`. Since these samples can be obtained directly from the shift register, a script implementing the D-bit multiplexer is not necessary.

Coefficient Memory

The $M = \frac{N+1}{2}$ coefficients H_{bin} generated by the function `dec2bin` are stored as character strings. In order to make them suitable for calculations, they have to be converted to signed integer numbers representing these binary values. Furthermore, by storing them in a $L \times D$ matrix they can be addressed easily. The following script converts the two's complement binary coefficients stored in the character array `Hbin` into unsigned integer numbers, and stores them in the matrix `coeff_rom`:

```
dec = bin2dec(Hbin);
for k = 1:M
    if sig(k) == -1
        dec(k) = -bitcmp(dec(k)-1,b(k));
```

```
    end
end
coeff_rom = zeros(L,D);
for l = 1:L
    coeff_rom(l,:) = dec(l*D:-1:(l-1)*D+1)';
end
```

The function `bin2dec` converts a binary number into an equivalent positive integer. Because of that, the sign (stored in `sig`) and word-length of each coefficient (stored in `b`) has to be known, so that in case a coefficient is negative, the positive integer generated by `bin2dec` can be converted to the correct negative integer.

Accumulator Registers

The states of the L accumulators can be stored in a vector with L elements, which is created as follows:

```
accu = zeros(1,L);
```

Storing the values of all accumulators in a single vector allows the simple calculation of the final output result by the command `sum(accu)`, which adds the elements of `accu`, and hence, makes the implementation of an accumulation script obsolete.

Main Loop

As discussed in Chapter 3.5, the accumulations in the individual sections are performed within D cycles of the master clock. After each cycle, the samples stored in the shift register are shifted by one delay block. The D operations necessary to compute the final result $y[Dm]$ are listed in Table 3.5. It can be seen that the operations performed during each cycle are the same, with the exception that different filter taps and coefficients are used in each cycle. This allows the implementation of a simple main loop which repeats itself after each cycle. Within this loop, a variable c that counts the clock cycles has to be increased by 1 if $c \leq D$, and reset to 1 if $c > D$. During the cycle $c = 1$, the accumulators have to be cleared, and during the cycle $c = D$, the final output result has to be computed and stored in a vector containing the samples of the output signal. The following MATLAB® script computes

the output signal `Y` for a given input signal stored in the vector `u` by employing a hardware-based simulation of a linear-phase FIR filter with a single-bit input.

```
for k = 1:length(u)
    c = mod(k-1,D)+1;
    d = [u(k) d(1:end-1)]; % Shifting the shift register
    if c == 1
        accu = accu*0; % clear accumulator
    end
    for l = 1:L % Do the computations section by section
        if d(n_0(l)) == d(n_k(l,c))
            s = 2*d(n_0(l))-1; % sign s(x[n-k])
            accu(l) = accu(l)+s*coeff_rom(l,c);
        end
    end
    if c == D
        result = sum(accu);
        resultout = floor(result/2^tr); % truncate by 'tr' bits
        Y = [Y resultout];
    end
end
```

The total number of computations that are performed by this script is determined by the length of the input vector `u`. The samples of `u` have to be either 0 or 1, corresponding to the voltages $-V_{ref}$ and $+V_{ref}$, respectively. Note that the values in `Y` are signed integers with a precision that has been reduced by $\texttt{tr} = a_{out} - \tilde{a}_{out}$ bits. In order to obtain the real values `Y`, has to be scaled properly.

3.5.3 Behavioral Simulation of the Impulse Response

The routine printed in appendix B allows the hardware-based simulation of any linear-phase FIR filter with single-bit input for an arbitrary input signal.

The simulation of the filter's impulse response requires an input signal of the form $u[n] = \delta[n]V_{ref}$. Since $u[n]$ is the output signal of a $\Delta\Sigma$ modulator, its samples can only assume the values $\pm V_{ref}$ and hence, the impulse response can not be simulated. However, it is possible to generate the signal

$$u[n] = \begin{cases} +V_{ref}, & n = 0 \\ -V_{ref}, & n \neq 0 \end{cases} = V_{ref}(2\delta[n] - 1), \tag{3.28}$$

which corresponds to the superposition of the DC voltage $-V_{ref}$ and an impulse of height $+2V_{ref}$. In MATLAB® the corresponding vector needed as input signal for the simulation can be generated by the statement

```
u = [1 zeros(1,I)];
```

where `I` denotes the numbers of zeros following the first sample. The number of iterations performed by the simulator is determined by the number of samples stored in the input vector `u`. Thus, in order to simulate the full impulse response of our filter, the input signal has to be at least 256 samples long. The decimated output vector `y` will then consist of 16 samples.

Figure 3.20 shows the response of our filter to the signal defined in (3.28) under the assumption, that the initial value of all the delays was 0, corresponding to a DC input voltage of $-V_{ref}$. The dashed line indicates the impulse response of our original filter as designed in Chapter 3.2. The hexadecimal values for the output signal as well as the accumulator output values of this simulation are listed in Appendix C.

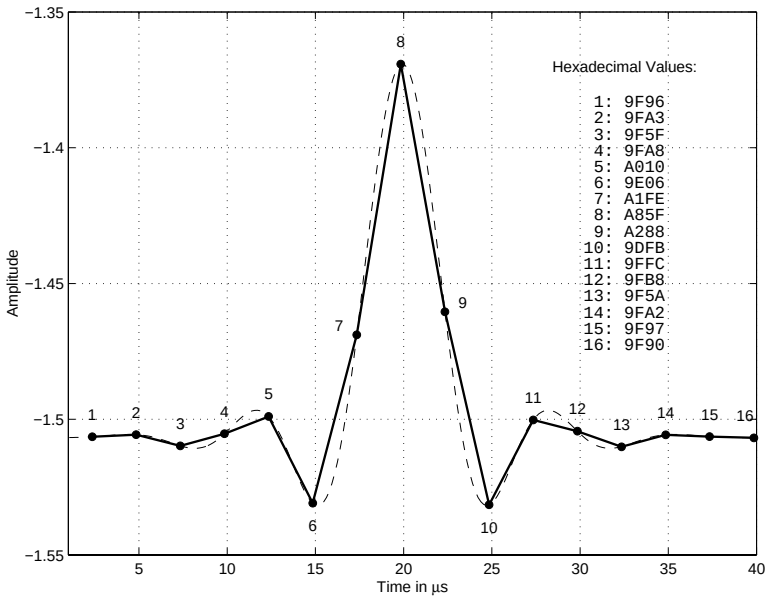


Figure 3.20: Behavioral simulation of the impulse response of our filter.

Chapter 4

Hardware Implementation

4.1 Introduction

There is a wide range of design options that my be used to implement a digital signal processing system design. As the investment made in any chip design is significant, designers search for ways to amortize the design effort over a large number of devices. This might result from a huge single market for one device, or multiple smaller markets for a more adaptive device. The larger the unit volume for a part, the lower its cost will be to the end user. Very low unit cost can be achieved by implementing the design as a custom IC which is, in general, less adaptive but usually produced in larger quantities.

Even though there will probably be a large market for our filter in the future, it is both more cost and time efficient to implement a first prototype using a programmable device. In CMOS, one may divide this spectrum of programmable devices into three areas:

- Chips with programmable logic structures
- Chips with programmable interconnects
- Chips with reprogrammable gate arrays

A very flexible, but also complex device is the programmable gate array. This approach may be further categorized into ad-hoc and structured arrays [10].

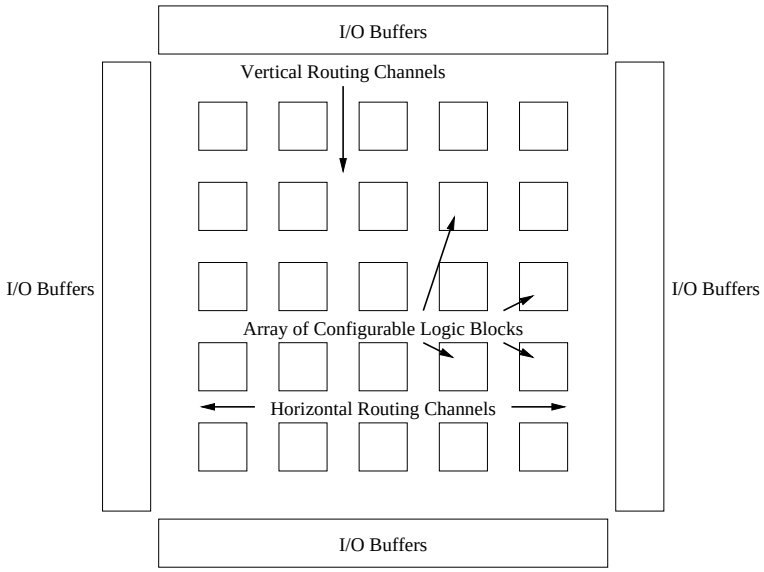


Figure 4.1: XILINX FPGA architecture.

4.1.1 The XILINX Field Programmable Gate Array (FPGA)

An example of an ad-hoc array is a set of products from the XILINX company. The architecture of the XC3000 series is depicted in Figure 4.1. An array of Configurable Logic Blocks® (CLBs) is embedded within a set of horizontal and vertical channels that contain routing, which can be personalized to interconnect CLBs. The configuration of the interconnects is achieved by turning on n-channel CMOS pass transistors. The state that determines a given interconnect pattern is held in static RAM cells distributed across the chip close to the controlled elements. The CLBs and routing channels are surrounded by a set of programmable input/output (I/O) buffers.

The detailed structure of a CLB is shown in Figure 4.2. It consists of two registers, a number of multiplexers, and a combinatorial function unit. The latter can generate two functions of four variables, any function of five variables, or a selection between two functions of four variables. The function bit and each multiplexer is controlled by a number of RAM state bits. More recent CLBs feature enhanced table lookup

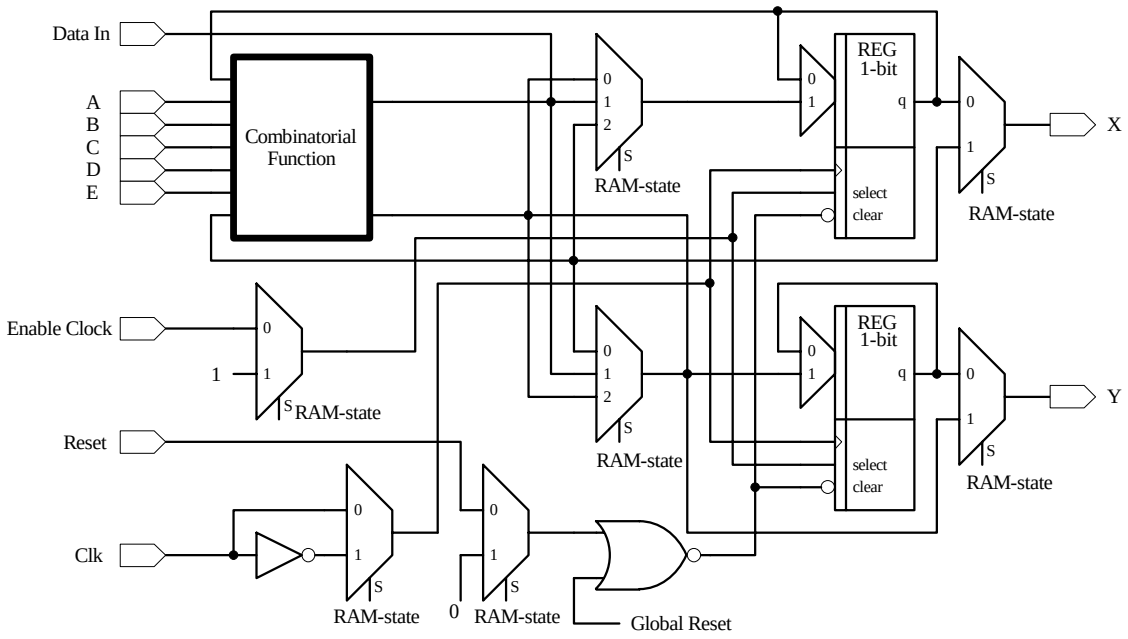


Figure 4.2: XILINX Configurable Logic Block® (CLB).

function generators, which can be used to build logic functions or serve as register storage. The XC4000 series has built-in support for carry chains, to conveniently build data paths. Each input and output on a CLB has a particular local interconnect pattern, which allows most local interconnection between adjacent CLBs to take place. At the junction of the horizontal and vertical routing channels, where the general-purpose interconnect runs, programmable switching matrices are employed to redirect routes. Figure 4.3 shows a typical CLB surrounded by switching matrices. They perform crossbar switching of the global interconnects, which run both vertically and horizontally. Programmable Interconnect Points (PIPs) interconnect the global routing to CLBs. Both PIPs and the switching matrices are implemented as n-channel pass gates controlled by 1-bit RAM cells. Extra specially long-distance interconnects are used to route important timing signals with low skew.

After capturing a circuit using CAD software, the design proceeds by mapping the logic to the CLBs. The software *places and routes* the CLBs by loading the internal state RAM with the codes needed to program the I/Os, the CLBs, and the

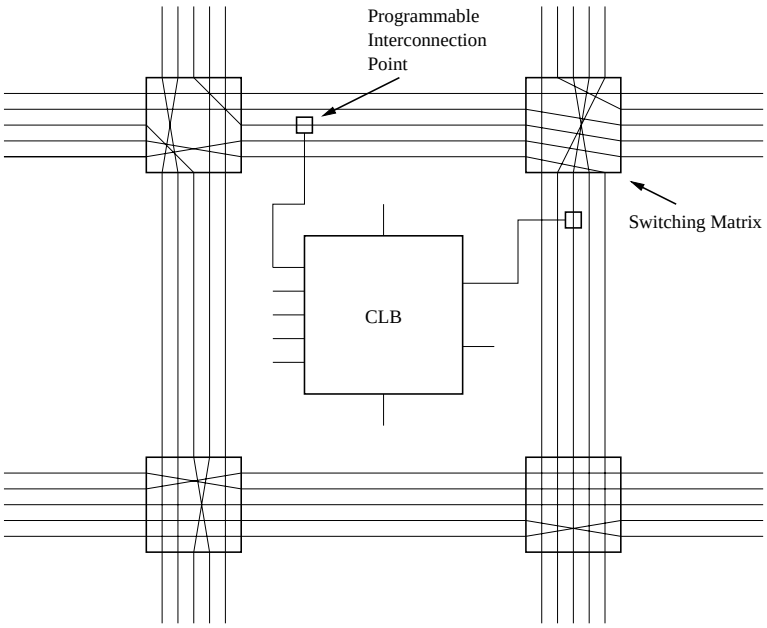


Figure 4.3: XILINX crossbar connect and CLB local connect example.

routing. The design is then ready to be tested or used.

The XC4000X series achieve high speed through advanced semiconductor technology and improved architecture. This series support system clock rates of up to 80MHz and internal performance in excess of 150MHz. Currently, the largest devices of this series contain 3,136 CLBs which corresponds to 85,000 logic gates, if none of the CLBs are used for RAM.

4.1.2 Objectives

The objective in this chapter is to realize the hardware concept developed in Chapter 3.5 with a XILINX FGPA device. The XILINX implementation has to be kept as transparent as possible since, provided that there will be a larger market for this application, the design will be transferred to a full custom IC at a later time, . Despite the fact that the XILINX Foundation® design tool suite offers the possibility to easily realize very sophisticated functions via LogiBLOX℠, only basic logic gates

and building blocks should be used to implement our FIR filter. The schematic-based design option offered by XILINX Foundation® [11] allows the capture of the system on a very basic level, which will simplify the transition to a full custom VLSI implementation at a later time.

4.2 Basic Concept

4.2.1 Overview

In Chapter 3.5 we developed the digital hardware concept for our digital decimation filter in order to be able to simulate the behavior of the system. Because the filter decimates the output signal by a factor of $D = 16$, the summation node was divided into 8 accumulation sections, each performing 16 additions. The input signal is fed in

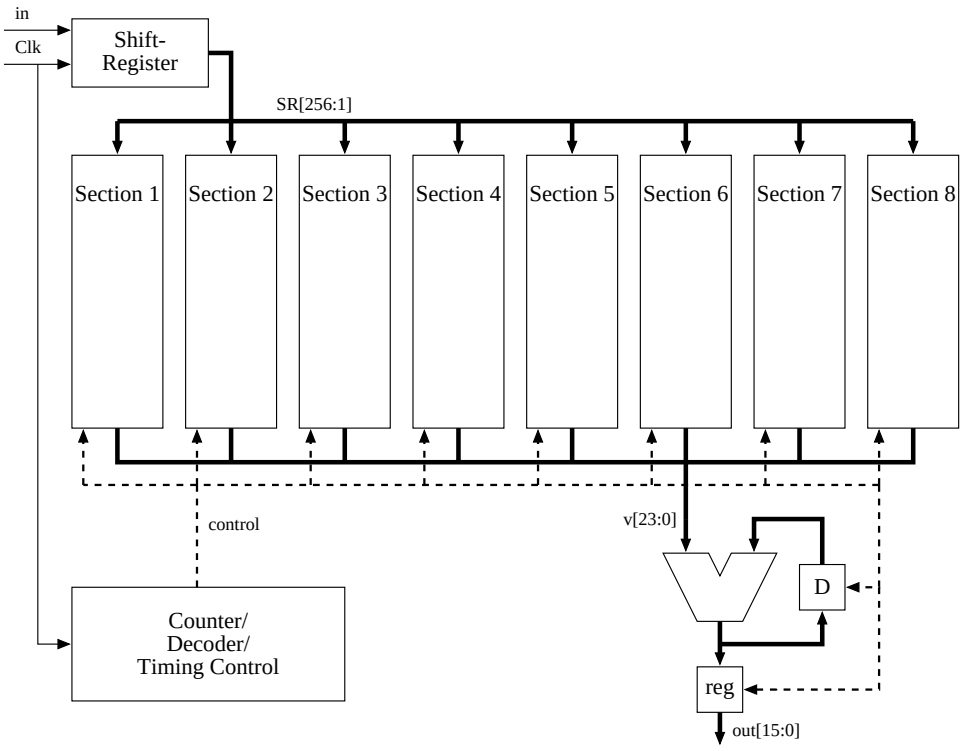


Figure 4.4: Block diagram of the digital decimation filter.

to a 256-bit shift register with serial input and parallel outputs. The latter are connected to the accumulation sections. The output signal is generated at the reduced sampling rate $f_s/16$ by an accumulator that sums the result of the 8 accumulation sections. This relatively simple basic concept defines the hierarchy of the hardware implementation of our filter. Figure 4.4 shows the major functional blocks of our FIR filte. Note that the outputs of all 8 accumulation sections are connected to the same 24-bit data bus `v[23:0]`, which is connected to the input of the final accumulator. Therefore, all the output registers of the accumulation sections have to be 24 bit wide tristate registers. These registers are controlled by the timing control circuit, which also generates the necessary control signals for the final accumulator.

4.2.2 General Timing

Before the hardware can be implemented, the exact timing of the system has to be defined. Figure 4.5 shows the general timing diagram for our filter hardware concept. Each accumulation section sums 16 numbers within 16 cycles. The accumulators

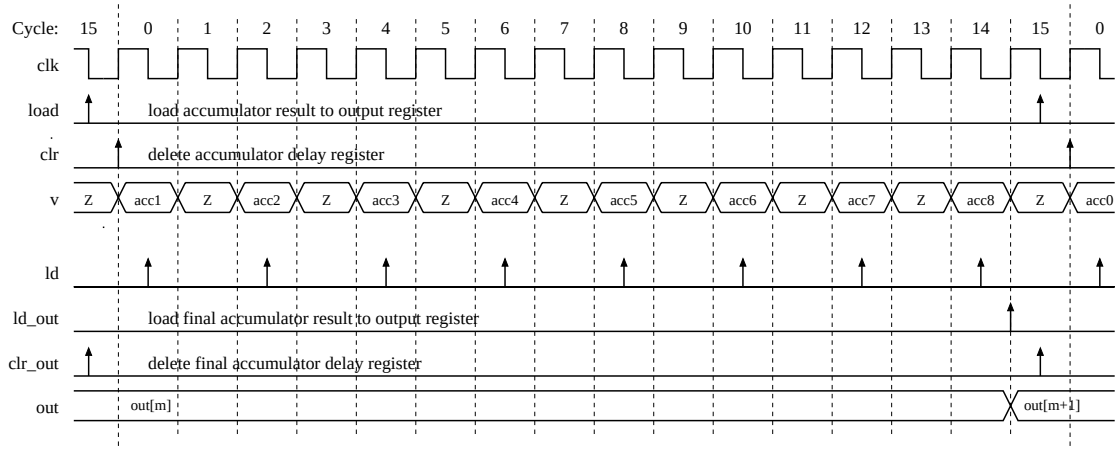


Figure 4.5: General timing diagram.

within these sections are clocked directly by the positive edge of the master clock `clk`. At the beginning of the accumulation, at cycle 0, the delay registers are cleared by the signal `clr`, and the 16 numbers are added during the following 16 cycles. The

results of the sections are available during cycle 15, when the last number is added to the sum. We define the signal `load`, which latches these results into the corresponding tristate data registers at the falling clock edge during the 15th cycle, where they will be stored for the next 16 cycles. The delay registers in the accumulation sections are cleared again one half cycle later. The control signals `clr` and `load` are generated by the timing control circuit.

The output samples are also generated every 16 cycles. Hence, the final accumulator sums the 8 results from the accumulation sections within the duration of 16 cycles of the master clock. When the tristate output register of an accumulation section is addressed by the the timing control circuit, it will write it's data to the data bus `v[23:0]`, which is connected to the input of the final accumulator. Thus, the timing control circuit is responsible for selecting the tristate registers and clocking the accumulator with the signal `ld` when the number to be added is available on the bus `v[23:0]`. Since only 8 numbers have to be summed, the accumulation can be performed at half the master clock rate. It can be seen in Figure 4.5 that the output registers of the accumulation sections are only selected during even cycles, while during odd cycles the registers are in tristate mode. Once the final result is calculated, it is latched into the output register by the signal `ld_out`. One half clock cycle later, at the beginning of cycle 15, the delay register of the final accumulator is cleared.

4.3 Capture of the linear-phase FIR Filter using Schematic-Based Design

During the process of implementation, it is important to maintain the design hierarchy previously defined. The separation of the design into major building blocks can be achieved in XILINX Foundation® by the creation of *macros*. A macro is a user-definable cell whose underlying function is described by either a schematic, an HDL script, a state machine, or LogiBLOX℠. In the schematic-based design, macros and primitive cells are treated in the same manner. Thus, instead of having a flat design, it is possible to define multiple hierarchy levels. In this chapter, we will create the

macros for the building blocks depicted in Figure 4.4 such that the building of the complete system is significantly simplified.

4.3.1 256-Bit Shift Register

As discussed in Chapter 2.2.4, a total number of N delays is required for the realization of an N-th order FIR filter. Hence, 255 delays are needed for our 256-tap filter, using the filter input as the first tap. Delays are realized in digital hardware by employing *Delay-Flip-Flops* (DFFs), which can be cascaded to build a shift register. By using an additional DFF at the input of the shift register, the integrity of the signal at the first tap is improved. The DFF *cleans* the signal by synchronizing it with the master clock and generating clear logic levels. Figure 4.6 shows the schematic of the 256-bit shift register used for our filter. Note that only 4 out of the total 256 DFFs are depicted.

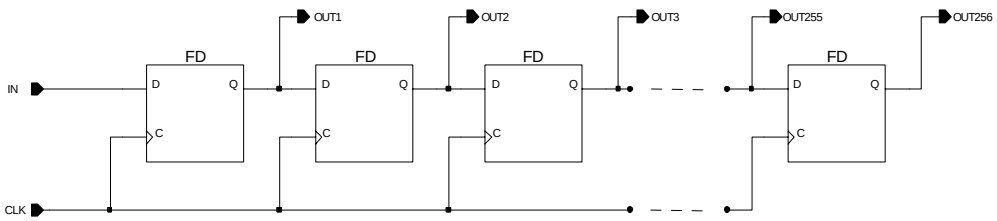


Figure 4.6: Schematic of the 256-bit shift register.

Since the shift register comprises 256 components, it is advantageous to define a macro in order to maintain a transparent design hierarchy. Figure 4.7 shows the macro symbol for our shift register with two inputs and a 256-bit output bus.

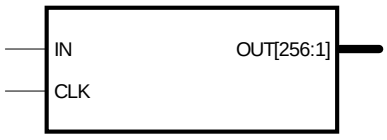


Figure 4.7: The 256-bit shift register macro.

4.3.2 Accumulation Sections

The basic hardware concept for the accumulation sections in linear-phase FIR decimation filters has already been developed in Chapter 3.5. The modified direct form structure of such a section is depicted in Figure 3.15. An accumulation section consists of a coefficient memory holding the coefficients $2h[k]$, a multiplexer, an accumulator, and a circuit that feeds either $+2h[k]$, $-2h[k]$, or 0 into the accumulator, depending on the signs of the samples of the input signal. The structures of the different accumulation sections of our filter only differ in the contents of the coefficient memories, the word length of the coefficients, and the data path width of the accumulator. Figure 4.8 shows the schematic of the first section of our filter. The schematics for all the other accumulation sections can be found in Appendix D.

Coefficient Memory

ROM modules can be created easily in Foundation® with the LogiBLOX℠ wizard [11]. By simply specifying the bus width, the memory depth, and a `.mem`-file containing the coefficients, a ROM building block can be generated. The ROM cell `COEFF_ROM1` of the first section of our filter is depicted in Figure 4.8. In case all the coefficients within a memory cell have the same sign, it is not necessary to store the sign bit. Instead, the sign bit signal is hardwired to the corresponding logic potential. For our filter, this is the case for the sections 1, 2, 5, and 8, where contain only positive coefficients. Thus, the sign bit signal is connected to the ground (GND) potential, which is equal to a logic '0'.

Because of the order in which the accumulation is performed, the coefficients have to be stored in reverse order (see Table 3.8). Within the framework of this thesis, a MATLAB® script was written which not only calculates the two's complement representation of filter coefficients, but also generates the necessary `.mem` files for the implementation of ROM cells in XILINX. The binary coefficients for each section are listed in Appendix A.

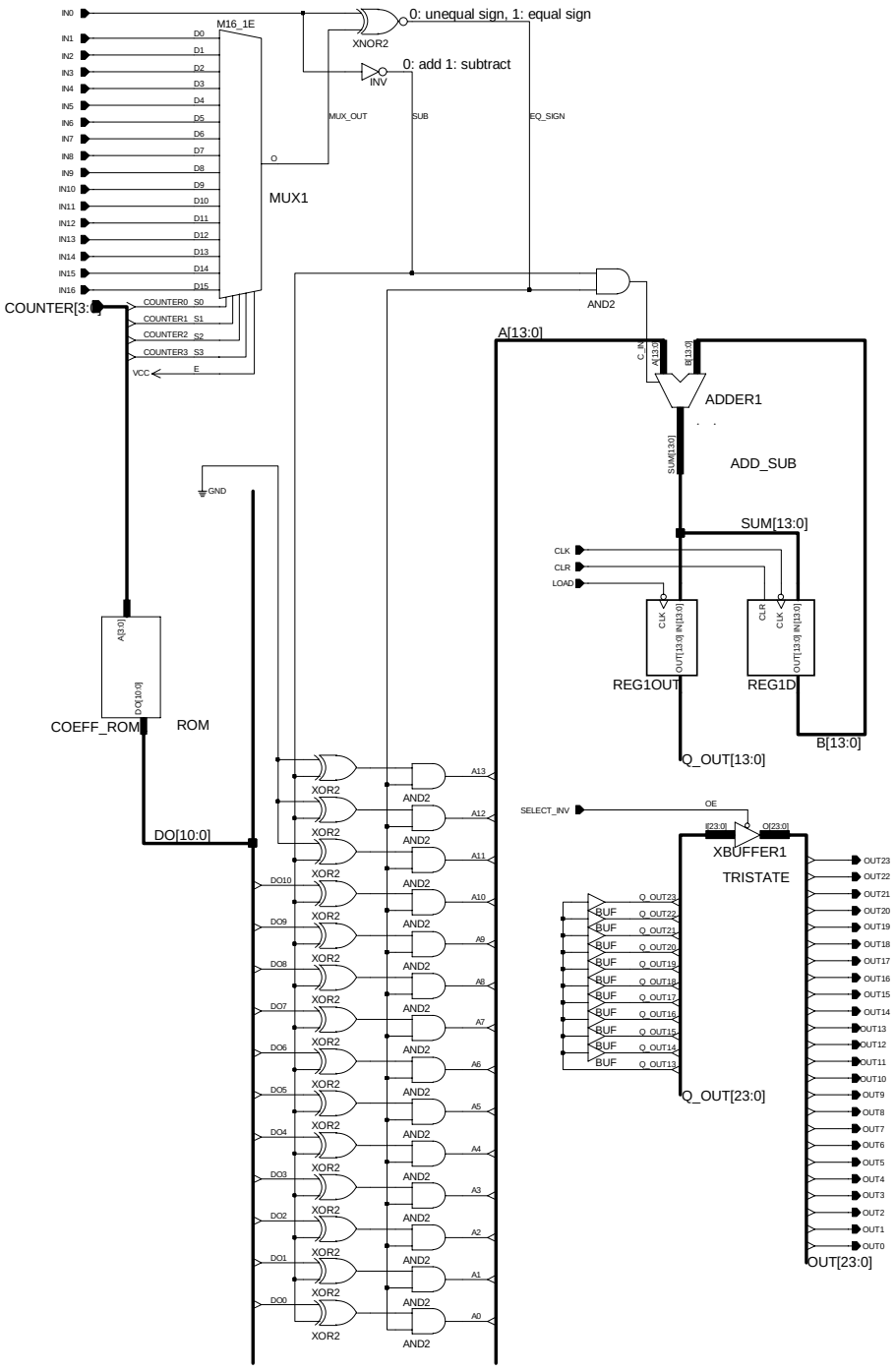


Figure 4.8: Schematic of the first accumulation section.

Sign Inverter

Depending on the signs of the samples of the input signal, the coefficients $2h[k]$ stored in the coefficient memory have to be either added or subtracted. In the case that the samples have opposite signs, no action is required (see Table 3.3). The subtraction of $2h[k]$ can simply be realized by feeding a sign-inverted version of $2h[k]$ into the accumulator. Since the coefficients are stored in two's complement format, the sign can be inverted by inverting all the bits and adding 1. A controlled bit-inversion can be realized by *exclusive-or* (XOR) gates, as depicted in Figure 4.8. If the input signal sample `IN0` is negative ($\hat{=}$ logic '0'), the signal `SUB` will be logic '1', which causes the XOR-gates to invert the bits of the coefficient. The addition of 1 is achieved by feeding the signal `SUB` into the carry-input of the adder used for the accumulator. In case the two samples are not equal, the signal `EQ_SIGN` will be logic '0', causing the AND-gates to set all the bits and the signal `SUB` to logic '0'. Thus, 0 will be added to the accumulator sum.

Accumulator

The accumulator consists of a multi-bit adder with carry-input and a delay-latch of the same bit-depth. The adder can be realized with the LogiBLOX^{SM}wizard, which offers a variety of different adder topologies. The most hardware efficient topology is the *ripple-carry adder* [10], which requires a minimum number of logic-gates. An n-bit ripple-carry adder is built by cascading n singe-bit adder stages. Because the carry-output of one stage is used to generate the sum of the next stage, the sum will be delayed with respect to the carry-input of that stage. Hence, the delay associated with the adder is $T_n = nT_c$, where T_n is the total add time, and T_c is the delay of one carry stage. Fortunately, the XILINX hardware guarantees very short gate-delay times, such that even in the case of a 24-bit adder the result is only delayed by a few dozen nano-seconds (the cycle time for our master clock of 6.4MHz is 156ns).

The delay-register used in this implementation consists of Flip-Flops that acquire the adder output-signal at the falling edge of the master clock, and feed it back into the adder at the rising edge, thus realizing a full clock cycle delay. The register can be cleared synchronously at the rising edge of the master clock by setting `CLR` to

logic '1' before the preceding falling clock edge. As specified in Chapter 4.2.2, this register has to be cleared at the beginning of clock cycle 0. This can be achieved by generating a signal `CLR` that is equal to logic '1' during cycle 15, and equal to logic '0' during all the other cycles.

Adjusting the Data Path Width

In all accumulation sections, the accumulator word-length is higher than the bit-depth of the coefficients. Therefore, the coefficients have to be converted to the word-length of the accumulator before they can be accumulated. For numbers in two's complement representation, this conversion is achieved by simply expanding the binary word by the necessary number of bits on the side of the *most significant bit* (MSB) and filling them with the value of the sign-bit. For example, the 4-bit number 1001 ($\hat{=} -7$) has to be extended to a 7-bit number:

$$\begin{array}{rcl} 1001 & \hat{=} & \text{-7} \\ \downarrow & & \\ 1111001 & \hat{=} & \text{-7} \end{array}$$

The new number 1111001 still represents -7, just with more bits. In terms of hardware this data path expansion is realized by wiring the additional bits to the MSB of the original multi-bit signal. Note that if all coefficients of a memory section have the same, sign no sign bit is stored. In that case, the additional bits have to be connected to the corresponding logic potential (sections 1, 2, 5, and 8 of our filter).

Output Register and Tristate Buffer

As specified in Chapter 4.2.2, the accumulator output result has to be latched into the output register in the middle of master clock cycle 15. The register used in our application latches its input signal on the falling edge of the signal `LD`.

All output registers of the accumulation sections are connected to a 24-bit data bus via 24-bit tristate buffers. Since the word length of the output register is generally less than 24 bits, it has to be expanded by wiring the additional input signals of the buffer to the MSB of the output signal of the register (see Figure 4.8). The tristate

buffers are negative-level triggered, i.e., the outputs can be made active by setting the signal SELECT_INV to logic '0'.

Creating XILINX Macros for the Accumulation Sections

In order to maintain the design hierarchy specified in Chapter 4.2.1, it is necessary to create XILINX macros for all accumulation sections. The Symbol Editor in Foundation® is used to create the symbols that will associated with the macro [11]. Once the symbol is created, it is linked to the schematic of the macro which can then be used as single component in other schematics. Figure 4.9 shows the macro symbol for an accumulation section.

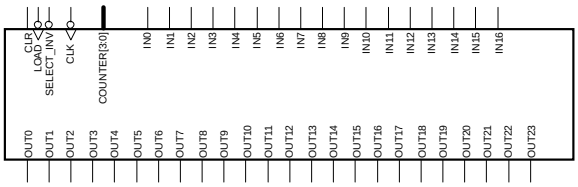


Figure 4.9: XILINX macro symbol for an accumulation section.

4.3.3 Final Accumulator and Output Register

The final accumulator and the output register are realized in the same manner as the accumulators and registers used in the accumulation sections. Both the accumulator and the output register are 24 bits wide. According to the timing specifications defined in Chapter 4.2.2, the accumulator is clocked by the signal LD_OUT in the middle of all even master clock cycles. The final output result is available during cycle 14. Therefore, at the end of cycle 14, it has to be latched. Immediately after that, the delay register has to be cleared by the signal CLR_OUT.

As specified in Chapter 3.5 the output signal has to be 16 bits wide. Hence, the 8 *least significant bits* (LSBs) are not connected to the output buffers.

4.3.4 Timing Control

It is the responsibility of the timing control circuits to generate all the necessary control signals for our filter. Figure 4.10 shows the schematic of these circuits. They are divided into 5 different segments. The 4-bit synchronous counter generates the

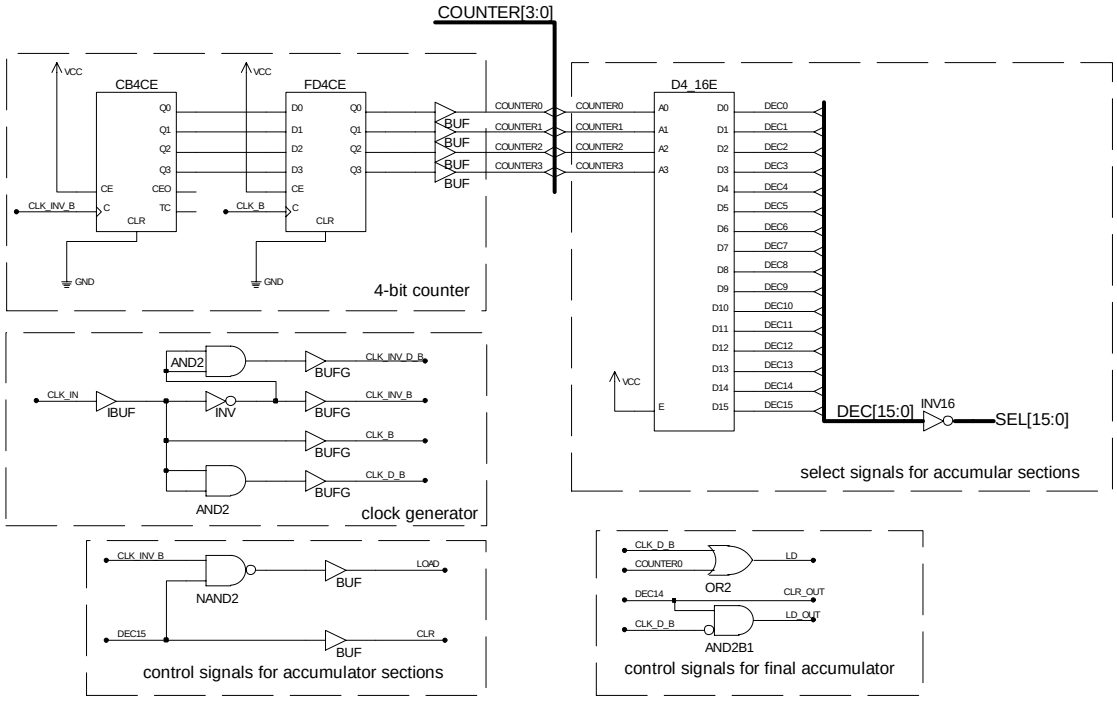


Figure 4.10: Schematic of the timing control circuits.

counter signals used in the accumulations to address the coefficients in the coefficient memories and to select the input signal samples via the 16-bit multiplexer. Furthermore, the counter signals are fed into a 4-to-16 bit decoder which generates the signals `DEC[15:0]` and their complement `SEL[15:0]`. The latter are used to control the tristate buffers of the accumulation sections. The two circuits on the bottom of Figure 4.10 generate the signals `LD`, `CLR`, `LD_OUT`, and `CLR_OUT`, which are needed to control the accumulators and output registers in the accumulator sections and the final accumulator. The timing diagram for these control circuits are depicted in Figure 4.11.

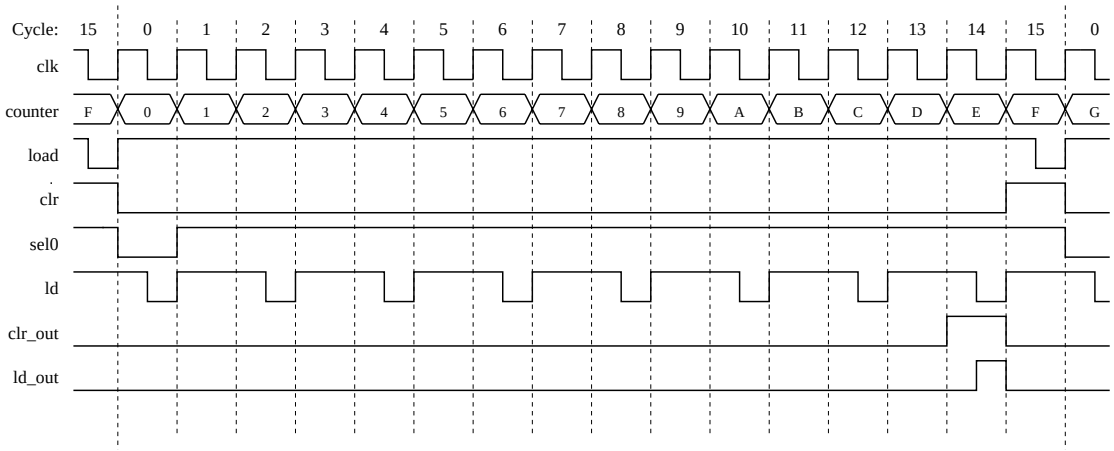


Figure 4.11: Timing diagram for the timing control circuits.

4.3.5 Complete System

Once the macros for all building blocks are created, the implementation of the complete FIR filter is reduced to the correct wiring of the macros. A connection diagram for the shift register and the accumulation sections can be obtained from Table 3.6.

Figure 4.12 shows the top-level schematic of the complete FIR filter.

4.4 Functional Simulation

XILINX Foundation® provides a *Logic Simulator*, which is a gate-level simulator. It is capable of performing a functional simulation immediately after a design is captured in the schematic capture tool. This makes it possible to verify that the logic which was created is correct, before the design is implemented in the FPGA architecture. It is beneficial to use the simulator often during the design process on different parts of the design in order to reduce the occurrence of errors. For example, the correct function of the coefficient memory macros in our design can be verified before they are used to build the accumulation sections. Once it is certain that a macro works correctly, its behavior does not have to be simulated on a higher level of hierarchy, only the signals connected to that macro have to be observed.

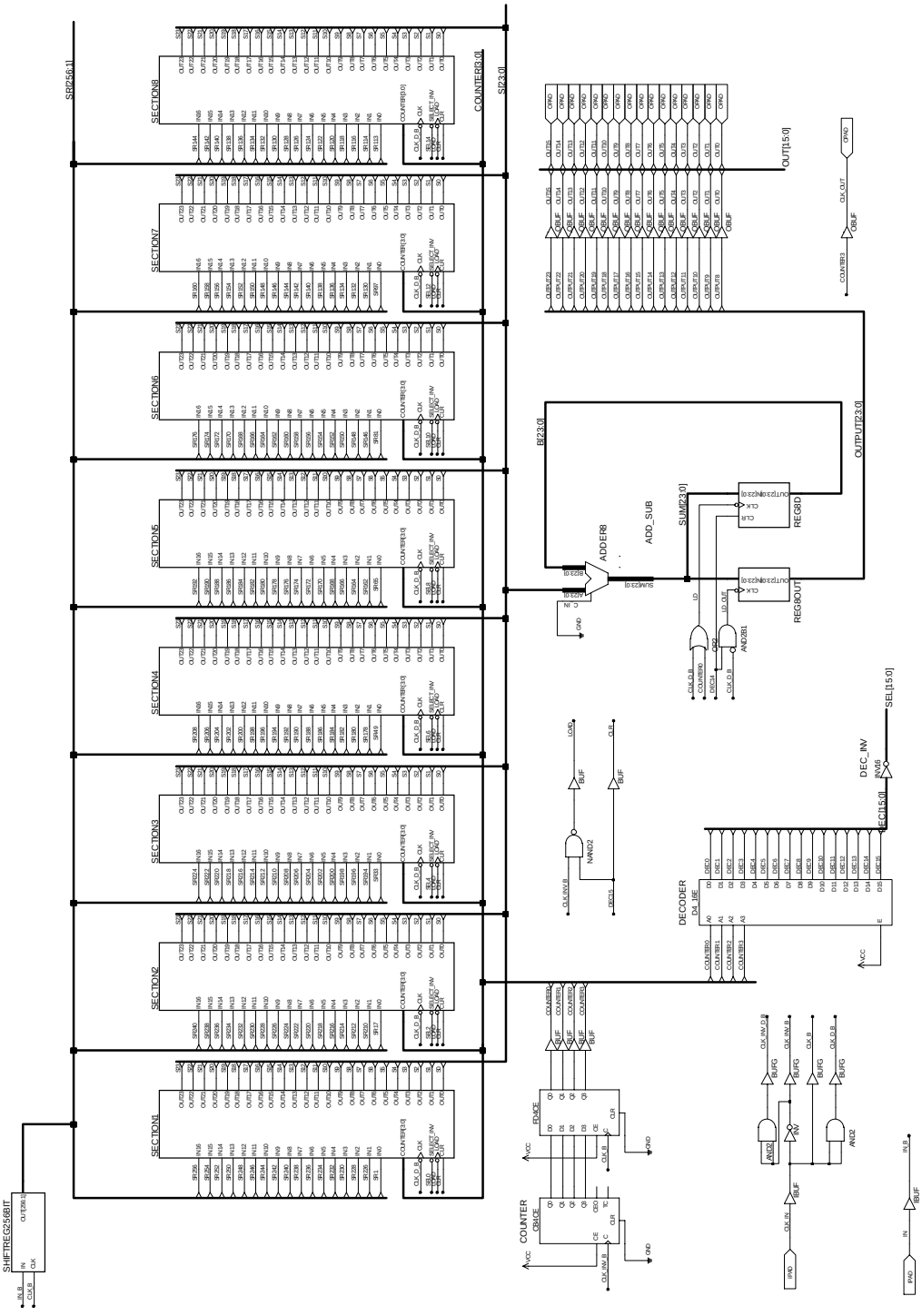


Figure 4.12: Top level schematic of the complete system.

The *Probe Tool* within the schematic capture toolbox allows the simple addition of signals to the waveform viewer tool of the simulator. By just clicking on a signal name in the schematic, the corresponding signal is added to the list of signals to be analyzed. With the *Stimulus Tool*, it is possible to define stimuli by means of custom formulae, internal binary counter outputs, stimulator state selectors, script files, waveform files, and simple keystrokes.

4.4.1 Simulating the Impulse Response

The most convenient way to verify the correct function of our filter is to simulate its impulse response. Results in hexadecimal form are already available from the behavioral simulation in Chapter 3.5.3. Hence, by performing the same simulation, the correctness of the performance can easily be verified by comparing the results of the behavioral and functional simulation.

Only 2 stimuli have to be generated for the simulation of the impulse response, namely the master clock and the input signal. We can define both stimuli with custom formulae:

```
C1:  L78nsH78ns
F0:  H200nsL45000ns
```

The first formula defines the signal `C1`, which corresponds to a clock signal with 50% duty-cycle and a period of 156ns ($\hat{=}$6.41MHz). The second signal defines the input signal, which is logic '1' for the first 200ns and logic '0' for the following 45μs, i.e., the input signal is only high during the first rising edge of the master clock. Hence, the input signal corresponds to a single pulse.

Signals of interest for the simulation on the top level hierarchy are the master clock, the counter output signals, the input signal, important timing control signals, the data signals in the final accumulator, and the output signal. Figure 4.13 shows an example simulation in the waveform viewer of the logic simulator. It depicts the states of important signals of our design between 19.9μs and 21.5μs.

To simulate the entire impulse response, it is necessary to run the simulation for at least 256 clock cycles of the master clock `CLK_IN`, i.e., 39.94μs. The resulting

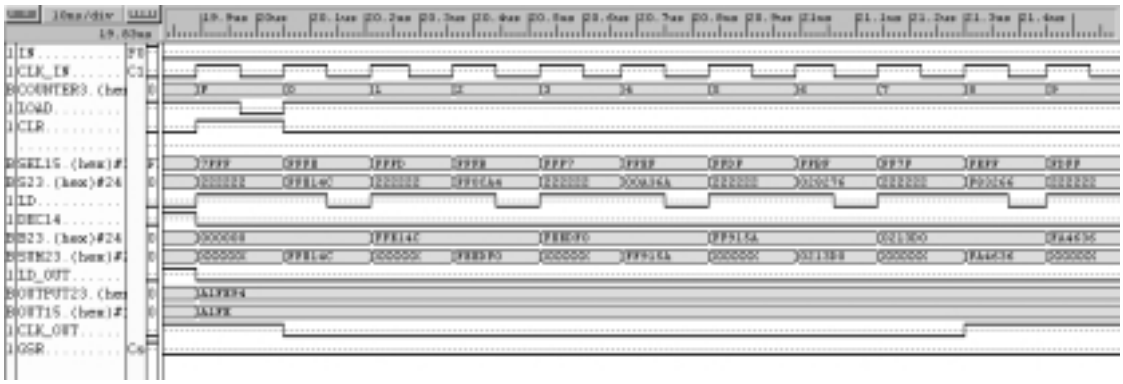


Figure 4.13: Waveform simulator in XILINX Foundation® showing an excerpt of the functional simulation of the impulse response.

output signal will then consist of 16 samples, generated at the decimated clock frequency `CLK_OUT` of 400kHz. Figure 4.14 shows the plot of the functional simulation of the impulse response of our filter. By comparing these simulation results to the

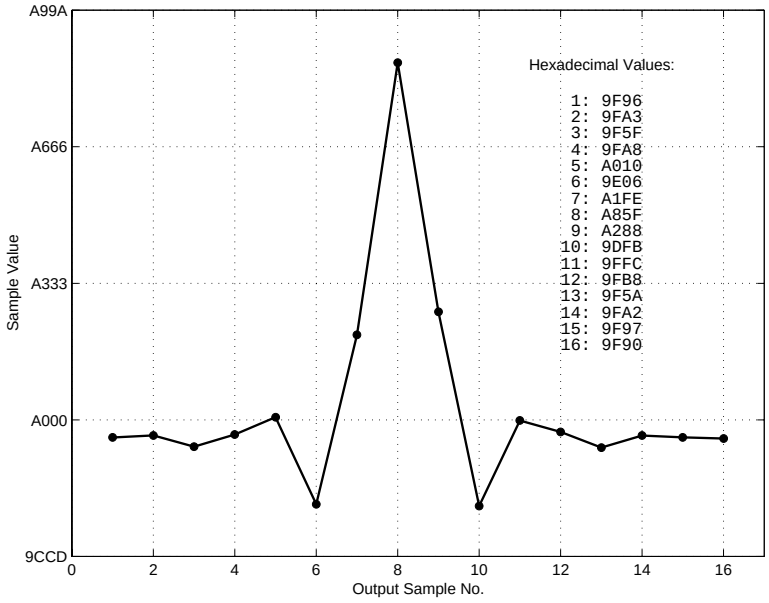


Figure 4.14: Functional simulation of the impulse response.

results obtained with the behavioral simulation in Chapter 3.5.3, we see that they are identical. Hence, the design works properly.

4.5 Design Implementation

Once a design has been captured and its performance has been verified, the *Project Manager* in Foundation® can be used to implement the design in a specific XILINX FPGA device. The smallest FPGA suitable for our filter is the XC4013XL, which consists of an array of 24×24 CLBs and 192 *I/O Buffers* (IOBs).

When implementing the design, the software first translates the design of the system into a description that can later be mapped onto the array of CLBs and IOBs. The software then places and routes these blocks in order to optimize the performance of the device. Table 4.1 shows an excerpt of the device utilization report generated by the design implementation tool in Foundation®. It can be seen that all CLBs on the device are used for logic functions.

Number of External IOBs	19	out of	192	9%
Flops:	0			
Latches:	0			
Number of CLBs	576	out of	576	100%
Total Latches:	0	out of	1152	0%
Total CLB Flops:	793	out of	1152	68%
4 input LUTs:	700	out of	1152	60%
3 input LUTs:	453	out of	576	78%
Number of BUFGLSs	3	out of	8	37%
Number of TBUFs	192	out of	1248	15%

Table 4.1: Device utilization report for the XC4013XL.

Once a design is successfully implemented, the software creates a `.bin` file containing the interconnect-pattern for the CLBs. This file can then be loaded into the static configuration RAM of the XILINX FPGA, and the device is ready to be used.

4.5.1 Structural Simulation

The simulation results from the functional simulation are helpful during the process of capturing the design, but they are based on a strictly logical description of the system. Therefore, a *structural simulation* is performed after the design has been placed and routed. The simulator used for structural simulation is the same one used for functional simulation. The only difference is that the design which is loaded into the simulator for structural simulation contains worst-case routing delays based on the actual placed and routed design. Hence, structural simulation gives a more accurate assessment of the behavior of the circuit.

Figure 4.15 depicts the structural simulation of our filter between 19.9μs and 20.7μs. It shows the timing behavior of the accumulation section 1 (`DO0`, `B0`, and `SUM0`), and the output signal of the filter (`OUT15`). Unlike the functional simulation, this simulation shows the delays caused by simple gate-delays and routing. It can be seen that, depending on the value of the bits at the input, the sum of the ripple-carry adders is delayed up to 30ns in some cases. But since the period time of the master clock is 156ns this does not cause a malfunction of the system.

After performing a thorough structural simulation of the impulse response, it can be said that our design is robust and works properly for clock frequencies up to 10Mhz. The results obtained from the structural simulation are identical to the ones from the behavior simulation with MATLAB® and the functional simulation in Foundation®.

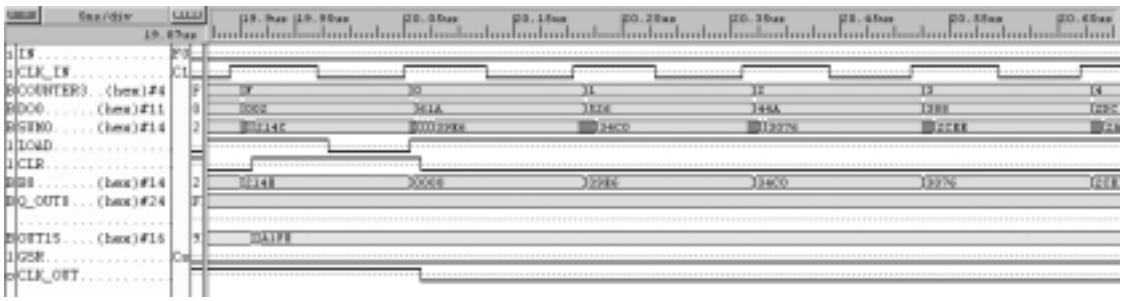


Figure 4.15: Structural simulation.

Chapter 5

Conclusions

In this thesis, the design and the implementation of a digital decimation FIR filter have been presented to be used in the development of a monolithic sonar receiver. A 1-bit output IFLF5 $\Delta\Sigma$ modulator was employed in the front end to obtain high dynamic range and ease of VLSI implementation.

The design began with the requirements on the sonar receiver that were used to derive a set of specifications for the digital decimation filter. The linear-phase FIR filter was designed using the Remez-Exchange Algorithm from the Signal Processing Toolbox [1] of MATLAB®. The obtained filter coefficients were quantized to a word-length of 22 bits, and the frequency response of the filter was simulated and verified by MATLAB® routines.

A digital hardware concept for the decimation filter was developed, which takes advantage of the 1-bit input signal from the $\Delta\Sigma$ modulator. Due to the decimation, the hardware effort could be reduced significantly. A concept was developed that generates the output samples directly at the decimated sampling rate. A simulator in MATLAB®, based on the hardware concept, was programmed and used to verify the impulse response of the digital decimation filter.

As to the hardware implementation, the digital decimation filter was realized on an XILINX XC4013XL *Field Programmable Gate Array* (FPGA). The design was captured entirely with a schematic-based design entry tool, and simulated with the logic simulator provided by the XILINX Foundation® tool suite. The *placed and routed* design was simulated with the same simulator, with the only difference that

worst-case routing delays were included in the simulation. Comparisons of the simulation of the actual design and the results obtained from the behavioral MATLAB® simulator showed that the implemented digital decimation filter was working properly up to input sampling frequencies of up to 10MHz.

As a next step toward a commercialized product, the digital decimation filter could be realized as a custom IC. This would significantly reduce the cost the device, given that there is a large market for it. Since the design was captured entirely with the schematic capture tool, and a strict design hierarchy was maintained, the design is very well suited for a VLSI implementation. Because only basic logic building blocks were used in the capture of the design, the transition to VLSI layout using a basic logic cell-library is significantly simplified.

Appendix A

Filter Coefficients

The following is a list of the 128 filter coefficients for the 256-tap linear-phase FIR filter described in this thesis. These coefficients, which have to be stored in digital memory, have been divided into 8 sections of 16 coefficients each. The list contains the 'analog' value and the two's complement binary and hexadecimal representation for each of the coefficients. Furthermore, the list shows the number of memory-bits selected for each section and the data-path width chosen for the accumulator of the corresponding section.

This list was generated by a MATLAB® script written for this purpose during the design process of the filter discussed in this thesis. Besides computing the two's complement representation for any given set of coefficients, this script also generates the necessary `.mem` files needed for the implementation of ROM cells with XILINX FPGA design tools, as discussed in Chapter 4. Furthermore, it generates the layout cells to be used in Magic, a part of the UC Berkeley VLSI Tool Suite.

SECTION 1
 Coefficient resolution: 11 Bits
 Maximum no. of bits needed for accumulation: 14

No.	orig.coeff.(2x)	two's complement	
1	0.00000048	00000000010	0002
2	0.00000763	00000100000	0020
3	0.00000954	00000101000	0028
4	0.00001574	00001000010	0042
5	0.00002337	00001100010	0062
6	0.00003386	00010001110	008E
7	0.00004721	00011000110	00C6
8	0.00006390	00100001100	010C
9	0.00008440	00101100010	0162
10	0.00010967	00111001100	01CC
11	0.00013971	01001001010	024A
12	0.00017452	01011011100	02DC
13	0.00021553	01110001000	0388
14	0.00026178	10001001010	044A
15	0.00031424	10100100110	0526
16	0.00037241	11000011010	061A

SECTION 2
 Coefficient resolution: 13 Bits
 Maximum no. of bits needed for accumulation: 17

No.	orig.coeff.(2x)	two's complement	
17	0.00043583	0011100100100	00724
18	0.00050497	0100001000110	00846
19	0.00057793	0100101111000	00978
20	0.00065470	0101010111010	00ABA
21	0.00073338	0110000000100	00C04
22	0.00081301	0110101010010	00D52
23	0.00089169	0111010011100	00E9C
24	0.00096750	0111111011010	00FDA
25	0.00103760	1000100000000	01100
26	0.00110054	1001000001000	01208
27	0.00115252	1001011100010	012E2
28	0.00119162	1001110000110	01386
29	0.00121450	1001111100110	013E6
30	0.00121880	1001111111000	013F8
31	0.00120068	1001110101100	013AC
32	0.00115824	1001011111010	012FA

SECTION 3
 Coefficient resolution: 15 Bits
 Maximum no. of bits needed for accumulation: 18

No.	orig.coeff.(2x)	two's complement	
33	0.00108910	001000111011000	011D8
34	0.00099039	001000000111010	0103A
35	0.00086164	000111000011110	00E1E
36	0.00070095	000101101111100	00B7C
37	0.00050831	000100001010100	00854
38	0.00028419	000010010101000	004A8
39	0.00002909	000000001111010	0007A
40	-0.00025415	111101111010110	07BD6
41	-0.00056267	111011011001000	076C8
42	-0.00089264	111000101100000	07160
43	-0.00123882	110101110110100	06BB4
44	-0.00159550	110010111011100	065DC
45	-0.00195551	101111111110110	05FF6
46	-0.00231123	101101000100010	05A22
47	-0.00265360	101010010000110	05486
48	-0.00297356	100111101001000	04F48

SECTION 4
 Coefficient resolution: 15 Bits
 Maximum no. of bits needed for accumulation: 19

No.	orig.coeff.(2x)	two's complement	
49	-0.00326157	100101010010000	04A90
50	-0.00350761	100011010001000	04688
51	-0.00370169	100001101011010	0435A
52	-0.00383425	100000100101110	0412E
53	-0.00389624	100000000101010	0402A
54	-0.00387812	100000001110110	04076
55	-0.00377321	100001000101110	0422E
56	-0.00357533	100010101101100	0456C
57	-0.00327969	100101001000100	04A44
58	-0.00288343	101000011000010	050C2
59	-0.00238705	101100011100100	058E4
60	-0.00179148	110001010100110	062A6
61	-0.00110149	110110111110100	06DF4
62	-0.00032425	111101010110000	07AB0
63	0.00053120	000100010110100	008B4
64	0.00145149	001011111001000	017C8

```
SECTION 5
 Coefficient resolution: 16 Bits
 Maximum no. of bits needed for accumulation: 20
 No.   orig.coeff.(2x)   two's complement
-------------------------------------------------
  65   0.00242138        0010011110101100   027AC
  66   0.00342369        0011100000011000   03818
  67   0.00443840        0100100010111000   048B8
  68   0.00544262        0101100100101100   0592C
  69   0.00641346        0110100100010100   06914
  70   0.00732470        0111100000000010   07802
  71   0.00815058        1000010110001010   0858A
  72   0.00886536        1001000101000000   09140
  73   0.00944233        1001101010110100   09AB4
  74   0.00985718        1010000110000000   0A180
  75   0.01008606        1010010101000000   0A540
  76   0.01010752        1010010110011010   0A59A
  77   0.00990391        1010001001000100   0A244
  78   0.00946045        1001101100000000   09B00
  79   0.00876760        1000111110100110   08FA6
  80   0.00781870        1000000000011010   0801A

SECTION 6
 Coefficient resolution: 18 Bits
 Maximum no. of bits needed for accumulation: 21
 No.   orig.coeff.(2x)   two's complement
-------------------------------------------------
  81   0.00661469      000110110001100000   006C60
  82   0.00516081      000101010010001110   00548E
  83   0.00346899      000011100011010110   0038D6
  84   0.00155687      000001100110000010   001982
  85  -0.00055027      111111011011111100   03F6FC
  86  -0.00282240      111101000111000010   03D1C2
  87  -0.00522232      111010101001110000   03AA70
  88  -0.00770664      111000000110111100   0381BC
  89  -0.01022720      110101100001110000   035870
  90  -0.01273155      110010111101101000   032F68
  91  -0.01516199      110000011110010110   030796
  92  -0.01745892      101110000111110100   02E1F4
  93  -0.01955938      101011111110001010   02BF8A
  94  -0.02139997      101010000101100010   02A162
  95  -0.02291727      101000100010000110   028886
  96  -0.02404928      100111010111111010   0275FA
```

```
SECTION 7
 Coefficient resolution: 19 Bits
 Maximum no. of bits needed for accumulation: 22
 No.   orig.coeff.(2x)   two's complement
-------------------------------------------------
  97  -0.02473640     1100110101010111000   066AB8
  98  -0.02492237     1100110011110101100   0667AC
  99  -0.02455759     1100110110110100110   066DA6
 100  -0.02359629     1100111110101100110   067D66
 101  -0.02200317     1101001011110000000   069780
 102  -0.01974821     1101011110001110010   06BC72
 103  -0.01681376     1101110110010000110   06EC86
 104  -0.01319027     1110010011111100100   0727E4
 105  -0.00887966     1110110111010000100   076E84
 106  -0.00389576     1111100000000101100   07C02C
 107   0.00173712     0000001110001110110   001C76
 108   0.00798225     0001000001011001000   0082C8
 109   0.01479244     0001111001001011100   00F25C
 110   0.02210903     0010110101000111100   016A3C
 111   0.02986383     0011110100101001010   01E94A
 112   0.03797770     0100110111000111010   026E3A

SECTION 8
 Coefficient resolution: 20 Bits
 Maximum no. of bits needed for accumulation: 24
 No.   orig.coeff.(2x)   two's complement
-------------------------------------------------
 113   0.04636526    00101111011110100110  02F7A6
 114   0.05493164    00111000010000000000  038400
 115   0.06357813    01000001000110101010  0411AA
 116   0.07220078    01001001111011110000  049EF0
 117   0.08069324    01010010101000010100  052A14
 118   0.08894873    01011011000101010110  05B156
 119   0.09685993    01100011001011110100  0632F4
 120   0.10432434    01101010110101000000  06AD40
 121   0.11124134    01110001111010010100  071E94
 122   0.11751842    01111000010101101100  07856C
 123   0.12307024    01111110000001100010  07E062
 124   0.12781954    10000010111000110010  082E32
 125   0.13170147    10000110110111001100  086DCC
 126   0.13466167    10001001111001001100  089E4C
 127   0.13665819    10001011111100000010  08BF02
 128   0.13766384    10001100111101111100  08CF7C

Maximum no. of bits used for final addition: 24
```

Appendix B

Hardware-based Simulator for MATLAB®

The following is the MATLAB® function `simulfilt.m`, which performs a behavioral simulation for a given linear-phase FIR filter with single-bit input. It is called as follows:

```
[y,yhex] = simulfilt(u,Hbin,sig,b,a,a_out,a_out_t,D,B,textfile);
```

The vector `u` contains the samples of the input signal, which must be either 0 or 1, corresponding to the reference voltages $-V_{ref}$ and $+V_{ref}$, respectively. `Hbin` is a character array containing the two's complement equivalent of the filter coefficients, one coefficient per row. The vector `sig` contains the signs for each coefficient in `Hbin`. The elements of `sig` must be either -1 or 1, and the length must be equal to the number of coefficients stored in `Hbin`. The vector `b` and `a` contain the memory bits for each coefficient and the accumulator word-length of the section to which the coefficient belongs, respectively. Both `b` and `a` must have the same length as `sig`, i.e., they must have one entry per coefficient. The variables `a_out` and `a_out_t` represent the word-length of the final accumulator, and the word-length, to which the output signal `y` is truncated, respectively. `D` is the downsampling ratio, and `B` is the number of bits to which the coefficients have originally been quantized. `textfile` is a string containing the name of the output text-file to which the simulation results will be written. After the execution of `simulfilt`, the output signal will be stored in the vectors `y` and `yhex`.

```
function [y,yhex] = simulfilt(u,Hbin,sig,b,a,a_out,a_out_t,D,B,textfile);


% [y,yhex] = simulfilt(u,Hbin,sig,b,a,a_out,a_out_t,D,B,textfile);
%
% Simulates the behavior of the implemented decimating FIR filter
%
% u:        input signal
% Hbin:     vector containing the binary coefficients (2's complement)
% sig:      vector containing the signs of the coefficients
% b:        vector containing the wordlength (in bits) of the coefficients)
% a:        vector containing the necessary accumulator width of a
%           particular section
% a_out:    wordlength of the output signal
% a_out_t:  wordlength og the truncated output signal
% D:        section size (no. of coefficients per section)
% B:        quantization width depth of the coefficients
% textfile: Name of the textfile containing the hecadecimal output values
%
% y:        Output signal, real values
% yhex:     Output signal, hexadecimal values
%
%  Roger Meier, 7/99
%  Dept. of Electrical Engineering
%  University of Rhode Island



M = length(sig);    % M: no. of coefficients
L = M/D;            % L: no. of sections
N = 2*M-1;          % N: Filter order

% Create (N+1)-bit shift register
% -------------------------------
d = zeros(1,N+1);

% Assigning in0 pins for each section
l = 1:L;
n_0 = D*(l-1)+1;

% Assigning multiplexed input pins for each section
[k,l] = meshgrid(1:D,1:L);
n_k = N-D*(l+1)+2*k+1;

% Create Coefficient ROM
% ----------------------
dec = bin2dec(Hbin);
for k = 1:M
    if sig(k) == -1
```

```
        dec(k) = -bitcmp(dec(k)-1,b(k));
    end
end
coeff_rom = zeros(L,D);
for l = 1:L
    coeff_rom(l,:) = dec(l*D:-1:(l-1)*D+1)';
end

% Create Accumulator Registers
accu = zeros(1,L);

% Truncation of the output signal
trunc = a_out-a_out_t;



% Main Loop
% ---------
delete(textfile)
echo off
diary(textfile);

if length(u) < 32
    u = [u zeros(1,32-length(u))];
end
y = [];
for k = 1:length(u)
    cycle = mod(k-1,D)+1;
    % Shifting the shift register
    d = [u(k)  d(1:end-1)];
    if cycle == 1
        accu = accu*0; % clear accumulator
    end
    % Do the accumulation section by section
    for l = 1:L
        input1 = n_0(l);
        input2 = n_k(l,cycle);
        if d(input1) == d(input2)
            opsign = 2*d(input1)-1;
            accu(l) = accu(l)+opsign*coeff_rom(l,cycle);
        end
    end
    if cycle == D
        disp(['Output ' num2str(k/D)]);
        result = sum(accu);
        resultout = floor(result/2^trunc); % truncate outputsignal
        y = [y resultout];
        % ------------------------------------------------------
        % everything between the dashed lines is used for displaying
```

```
        % the hexadecimal results
            for p = 1:L
                if sign(accu(p)) == -1
                    hexstring = ...
                    dec2hex(1+bitcmp(abs(accu(p)),a_out),ceil(a_out/4));
                else
                    hexstring = ...
                    dec2hex(abs(accu(p)),ceil(a_out/4));
                end
                disp(['Accumulator ' num2str(p) ': ' hexstring])
        end
    if sign(result) == -1
        yfull = dec2hex(1+bitcmp(abs(result),a_out),ceil(a_out/4));
        yhex(k/D,:) = dec2hex(1+bitcmp(abs(resultout),a_out_t),ceil(a_out_t/4));
    else
        yfull = dec2hex(abs(result),ceil(a_out/4));
        yhex(k/D,:) = dec2hex(abs(resultout),ceil(a_out_t/4));
    end
    disp(['Sum: ' yfull]);
    disp(['Output: ' yhex(k/D,:)]);
    disp('----------------------')
    % ------------------------------------------------------------------
end
end
diary off


% calculating the real value of the output signal
y = y/2^(B-trunc);

% plotting the output signal
figure
clf reset
subplot(2,1,1)
plot(y)
v = axis;
axis([1 length(y) v(3) v(4)])
grid on

% plotting the output signal, scaling the y axis with the maximum output range
subplot(2,1,2)
plot(y)
axis([1 length(y) -2^(a_out_t-(B-trunc)-1) 2^(a_out_t-(B-trunc)-1)])
grid on
```

Appendix C

MATLAB® Simulation Results

The following is the text-file `output.txt` generated by the MATLAB® function `simulfilt`, which is described in Appendix B. These results are the response of the linear-phase FIR filter discussed in this thesis to the signal described in equation (3.28). For our filter the simulation is initiated as follows:

```
[y,yhex] = simulfilt(u,Hbin,sig,b,a,24,16,16,22,'output.txt');
```

where the input signal is generated by the following statement:

```
u = [1 zeros(1,255)];
```

The result is a quasi-impulse response, which is depicted in Figure 3.20.

```
Output 1
Accumulator 1: FFE766
Accumulator 2: FF0CA4
Accumulator 3: 00A36A
Accumulator 4: 028276
Accumulator 5: F83266
Accumulator 6: 092702
Accumulator 7: 04582A
Accumulator 8: 97CAF8
Sum: 9F9674
Output: 9F96
--------------------
Output 2
Accumulator 1: FFE14C
Accumulator 2: FF1F9E
Accumulator 3: 00A36A
Accumulator 4: 028276
Accumulator 5: F83266
Accumulator 6: 092702
Accumulator 7: 04582A
Accumulator 8: 97CAF8
Sum: 9FA354
Output: 9FA3
--------------------
Output 3
Accumulator 1: FFE14C
Accumulator 2: FF0CA4
Accumulator 3: 0072B2
Accumulator 4: 028276
Accumulator 5: F83266
Accumulator 6: 092702
Accumulator 7: 04582A
Accumulator 8: 97CAF8
Sum: 9F5FA2
Output: 9F5F
--------------------
Output 4
Accumulator 1: FFE14C
Accumulator 2: FF0CA4
Accumulator 3: 00A36A
Accumulator 4: 029A3E
Accumulator 5: F83266
Accumulator 6: 092702
Accumulator 7: 04582A
Accumulator 8: 97CAF8
Sum: 9FA822
Output: 9FA8
--------------------
```

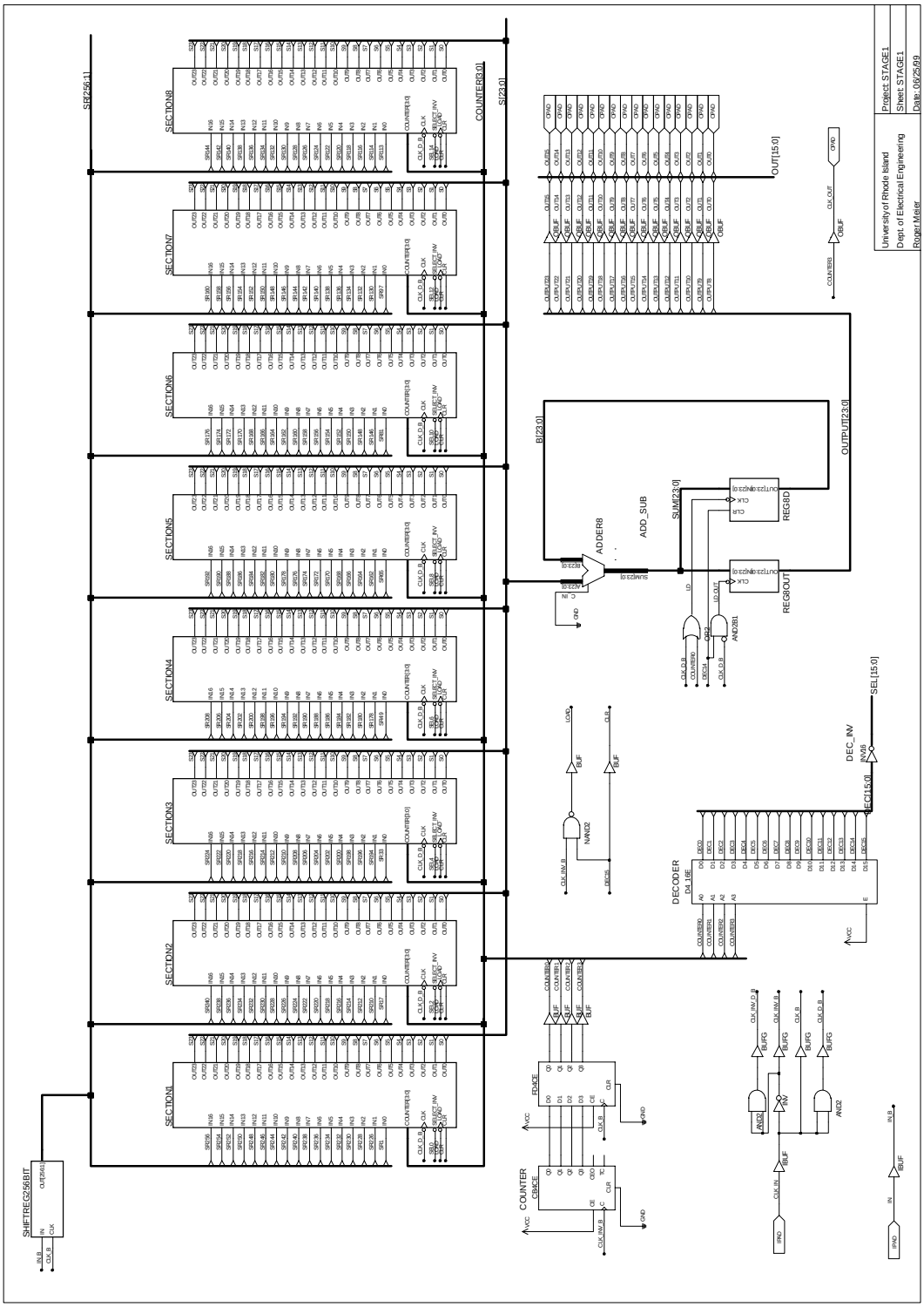
```
Output 5
Accumulator 1: FFE14C
Accumulator 2: FF0CA4
Accumulator 3: 00A36A
Accumulator 4: 028276
Accumulator 5: F8B280
Accumulator 6: 092702
Accumulator 7: 04582A
Accumulator 8: 97CAF8
Sum: A01074
Output: A010
--------------------
Output 6
Accumulator 1: FFE14C
Accumulator 2: FF0CA4
Accumulator 3: 00A36A
Accumulator 4: 028276
Accumulator 5: F83266
Accumulator 6: 079CFC
Accumulator 7: 04582A
Accumulator 8: 97CAF8
Sum: 9E0654
Output: 9E06
--------------------
Output 7
Accumulator 1: FFE14C
Accumulator 2: FF0CA4
Accumulator 3: 00A36A
Accumulator 4: 028276
Accumulator 5: F83266
Accumulator 6: 092702
Accumulator 7: 06C664
Accumulator 8: 97CAF8
Sum: A1FE94
Output: A1FE
--------------------
Output 8
Accumulator 1: FFE14C
Accumulator 2: FF0CA4
Accumulator 3: 00A36A
Accumulator 4: 028276
Accumulator 5: F83266
Accumulator 6: 092702
Accumulator 7: 04582A
Accumulator 8: A09A74
Sum: A85FD6
Output: A85F
--------------------
```

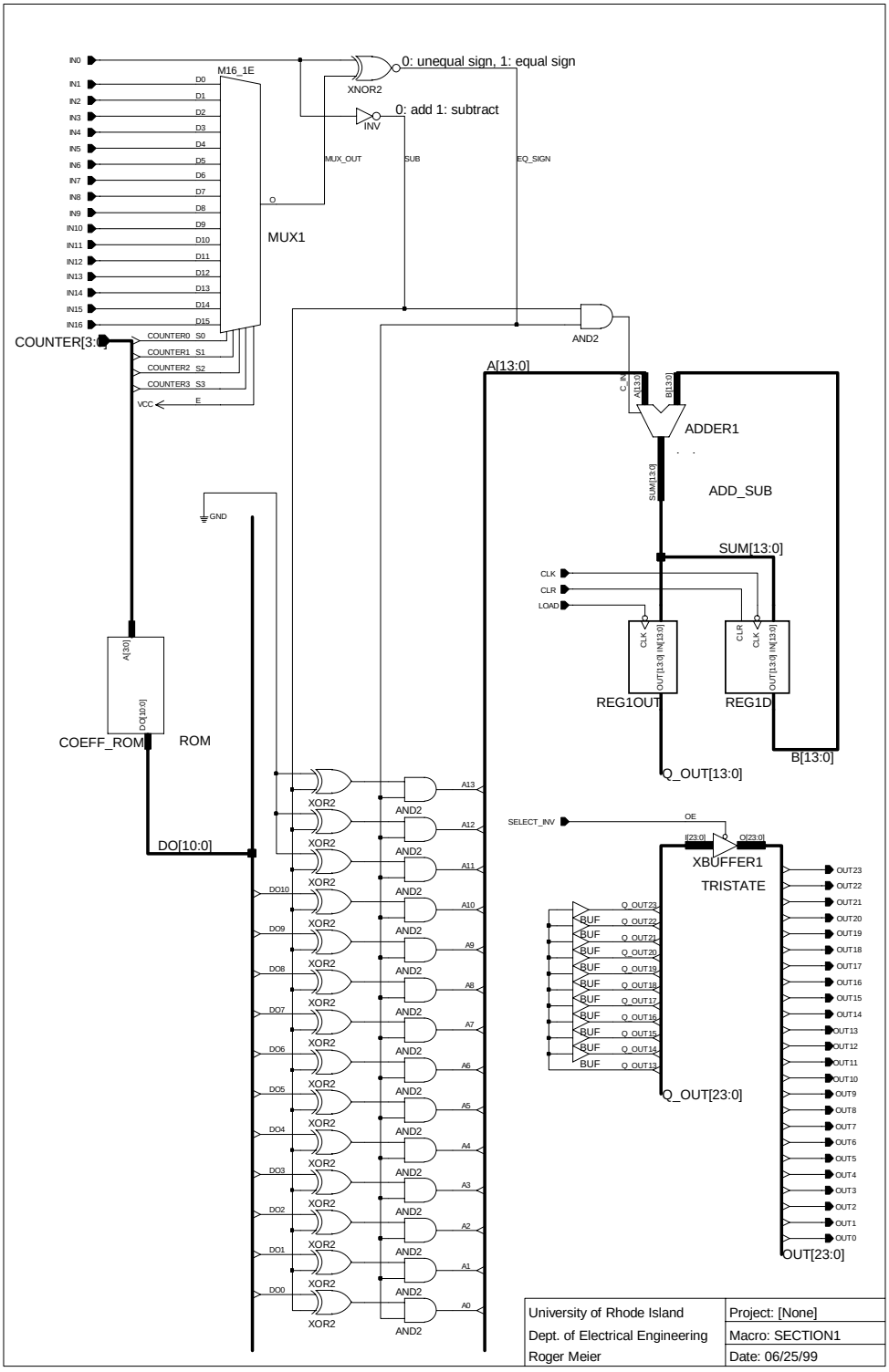
```
Output 9
Accumulator 1: FFE14C
Accumulator 2: FF0CA4
Accumulator 3: 00A36A
Accumulator 4: 028276
Accumulator 5: F83266
Accumulator 6: 092702
Accumulator 7: 04582A
Accumulator 8: 9AC29E
Sum: A28800
Output: A288
--------------------
Output 10
Accumulator 1: FFE14C
Accumulator 2: FF0CA4
Accumulator 3: 00A36A
Accumulator 4: 028276
Accumulator 5: F83266
Accumulator 6: 092702
Accumulator 7: 02C2E2
Accumulator 8: 97CAF8
Sum: 9DFB12
Output: 9DFB
--------------------
Output 11
Accumulator 1: FFE14C
Accumulator 2: FF0CA4
Accumulator 3: 00A36A
Accumulator 4: 028276
Accumulator 5: F83266
Accumulator 6: 099362
Accumulator 7: 04582A
Accumulator 8: 97CAF8
Sum: 9FFCBA
Output: 9FFC
--------------------
Output 12
Accumulator 1: FFE14C
Accumulator 2: FF0CA4
Accumulator 3: 00A36A
Accumulator 4: 028276
Accumulator 5: F85A12
Accumulator 6: 092702
Accumulator 7: 04582A
Accumulator 8: 97CAF8
Sum: 9FB806
Output: 9FB8
--------------------
Output 13
Accumulator 1: FFE14C
Accumulator 2: FF0CA4
Accumulator 3: 00A36A
Accumulator 4: 024D06
Accumulator 5: F83266
Accumulator 6: 092702
Accumulator 7: 04582A
Accumulator 8: 97CAF8
Sum: 9F5AEA
Output: 9F5A
--------------------
Output 14
Accumulator 1: FFE14C
Accumulator 2: FF0CA4
Accumulator 3: 00B542
Accumulator 4: 028276
Accumulator 5: F83266
Accumulator 6: 092702
Accumulator 7: 04582A
Accumulator 8: 97CAF8
Sum: 9FA232
Output: 9FA2
--------------------
Output 15
Accumulator 1: FFE14C
Accumulator 2: FF13C8
Accumulator 3: 00A36A
Accumulator 4: 028276
Accumulator 5: F83266
Accumulator 6: 092702
Accumulator 7: 04582A
Accumulator 8: 97CAF8
Sum: 9F977E
Output: 9F97
--------------------
Output 16
Accumulator 1: FFE14E
Accumulator 2: FF0CA4
Accumulator 3: 00A36A
Accumulator 4: 028276
Accumulator 5: F83266
Accumulator 6: 092702
Accumulator 7: 04582A
Accumulator 8: 97CAF8
Sum: 9F905C
Output: 9F90
--------------------
```

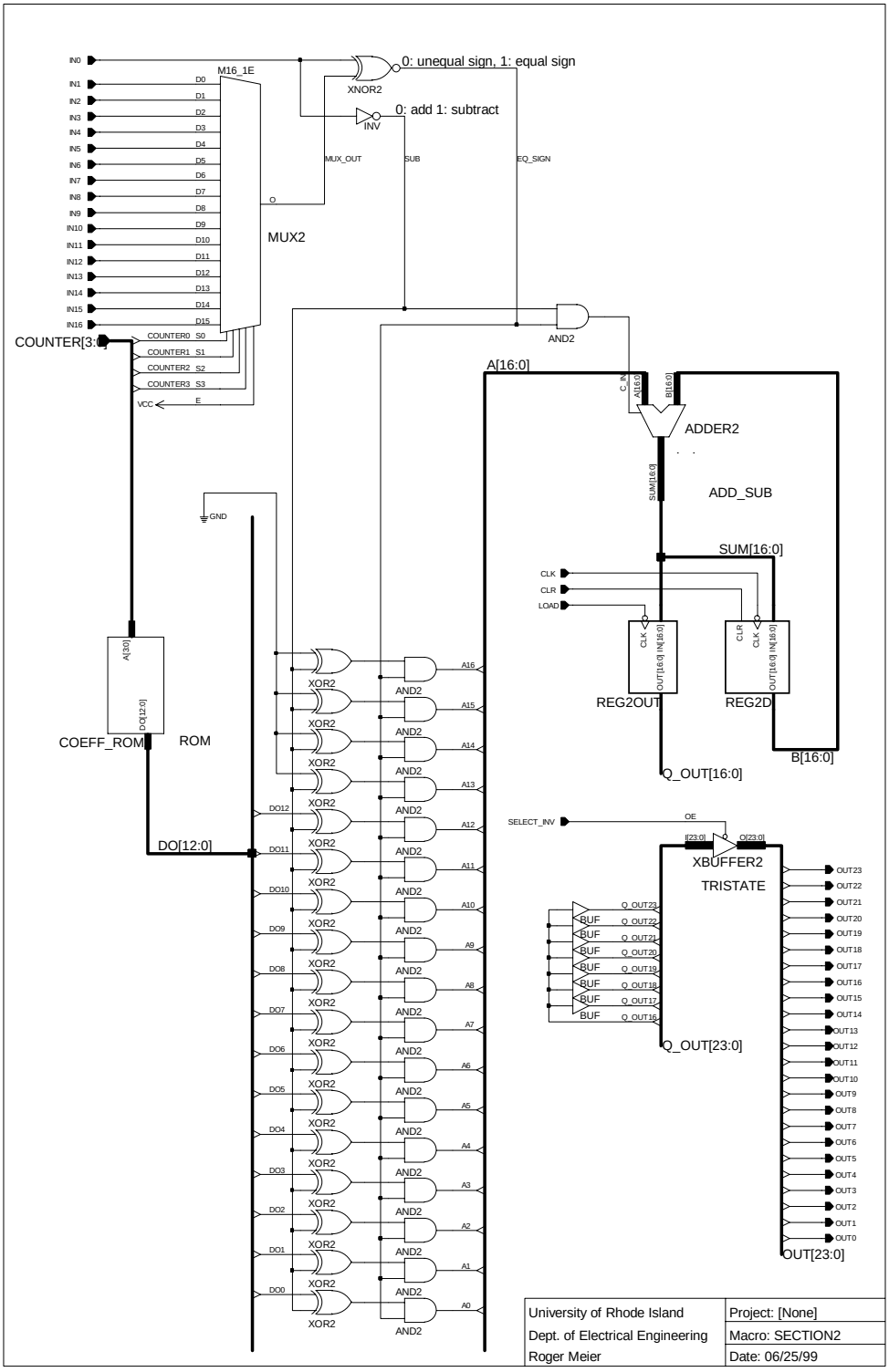
Appendix D

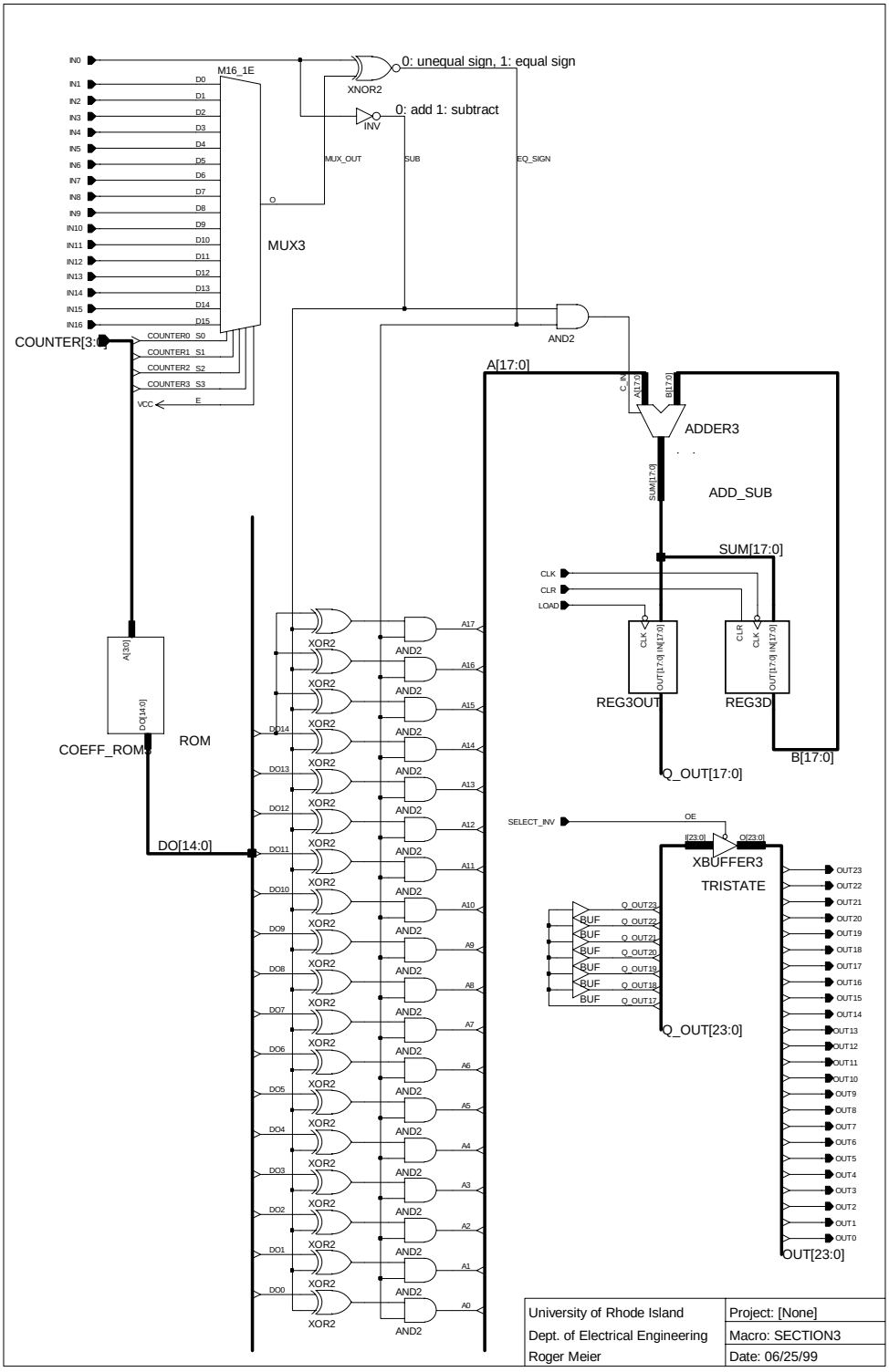
XILINX Design Schematics

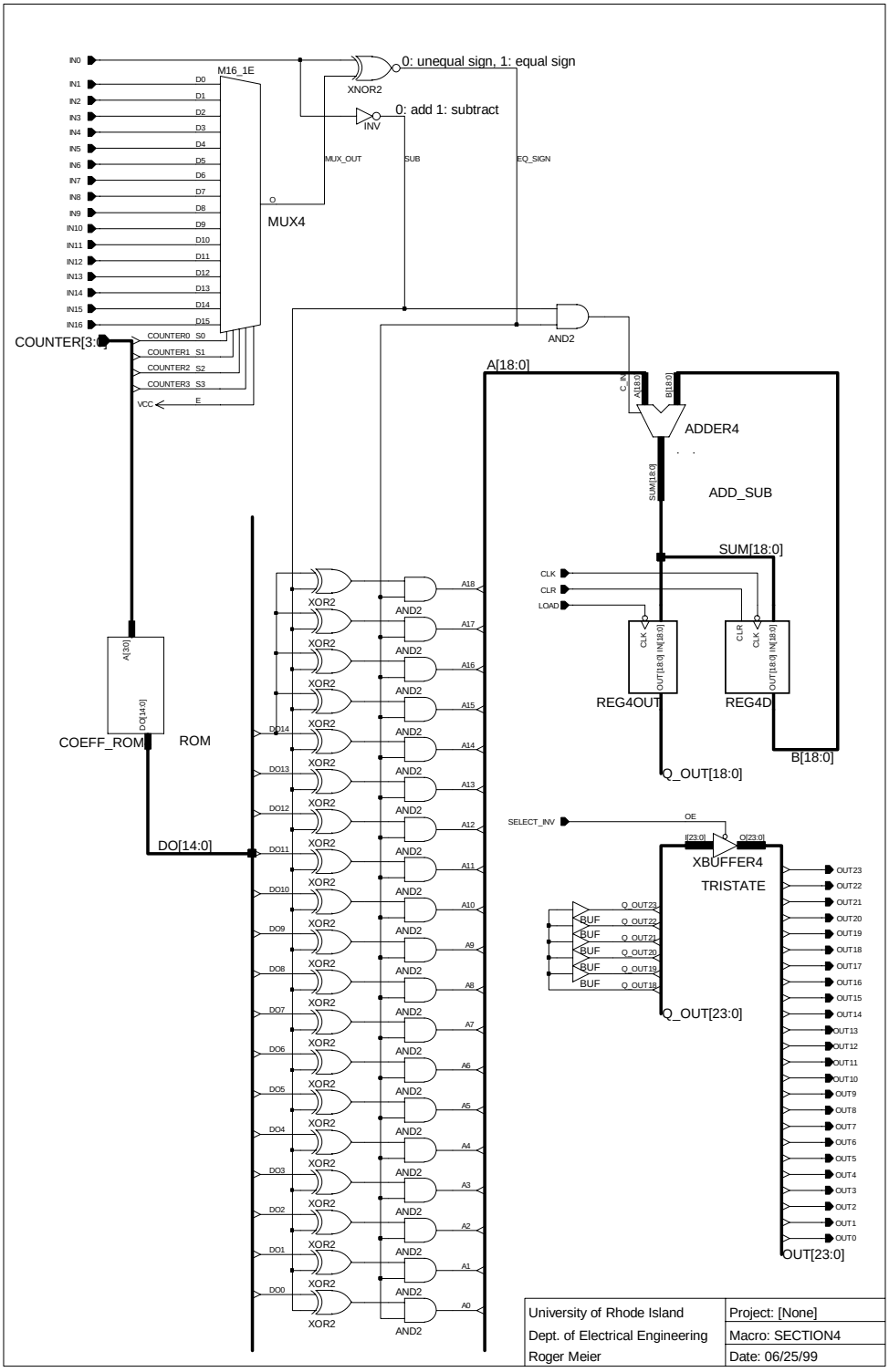
The following are the schematics of the XILINX implementation of the digital decimation filter. The first page shows the top hierarchy level schematic of the system. The schematics for the accumulator sections are printed on the following 8 pages.

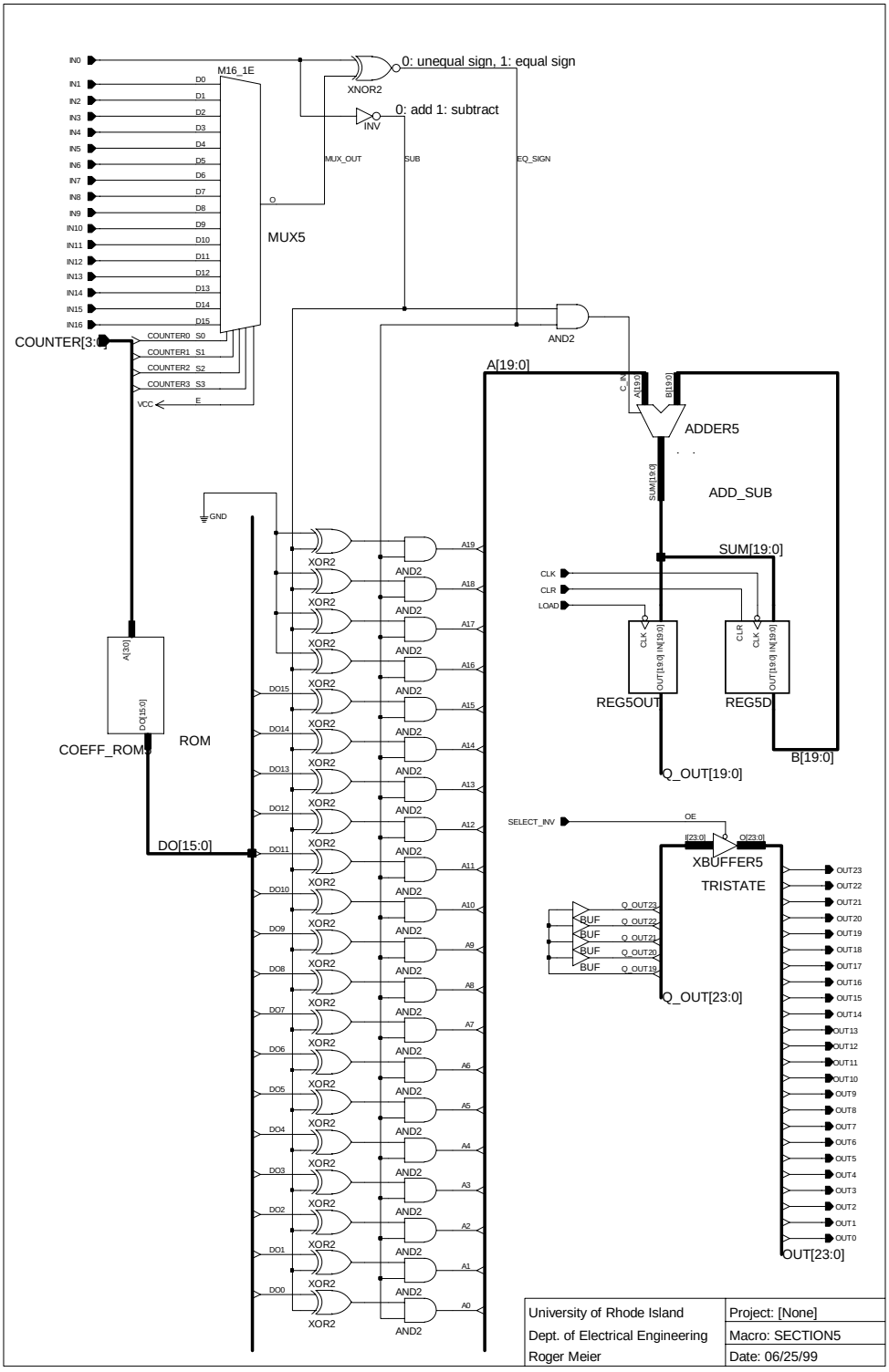

SHIFTREG256BIT
SR[256:1]
COUNTER[3:0]
SI[23:0]
SECTION1
SECTION2
SECTION3
SECTION4
SECTION5
SECTION6
SECTION7
SECTION8
B[23:0]
ADDER8
ADD_SUB
SUM[23:0]
REG8A
REG8B
OUTPUT[23:0]
OUT[15:0]
REG8OUT
AND2B1
DECODER
DI_16E
DEC[15:0]
DEC_INV
SEL[15:0]
NAND2
COUNTER
CB4CE
FD4CE
BUFG
INV
AND2
Universityof Rhode Island
Dept. of Electrical Engineering
Roger Meier
Project: STAGE1
Sheet: STAGE1
Date: 09/25/09

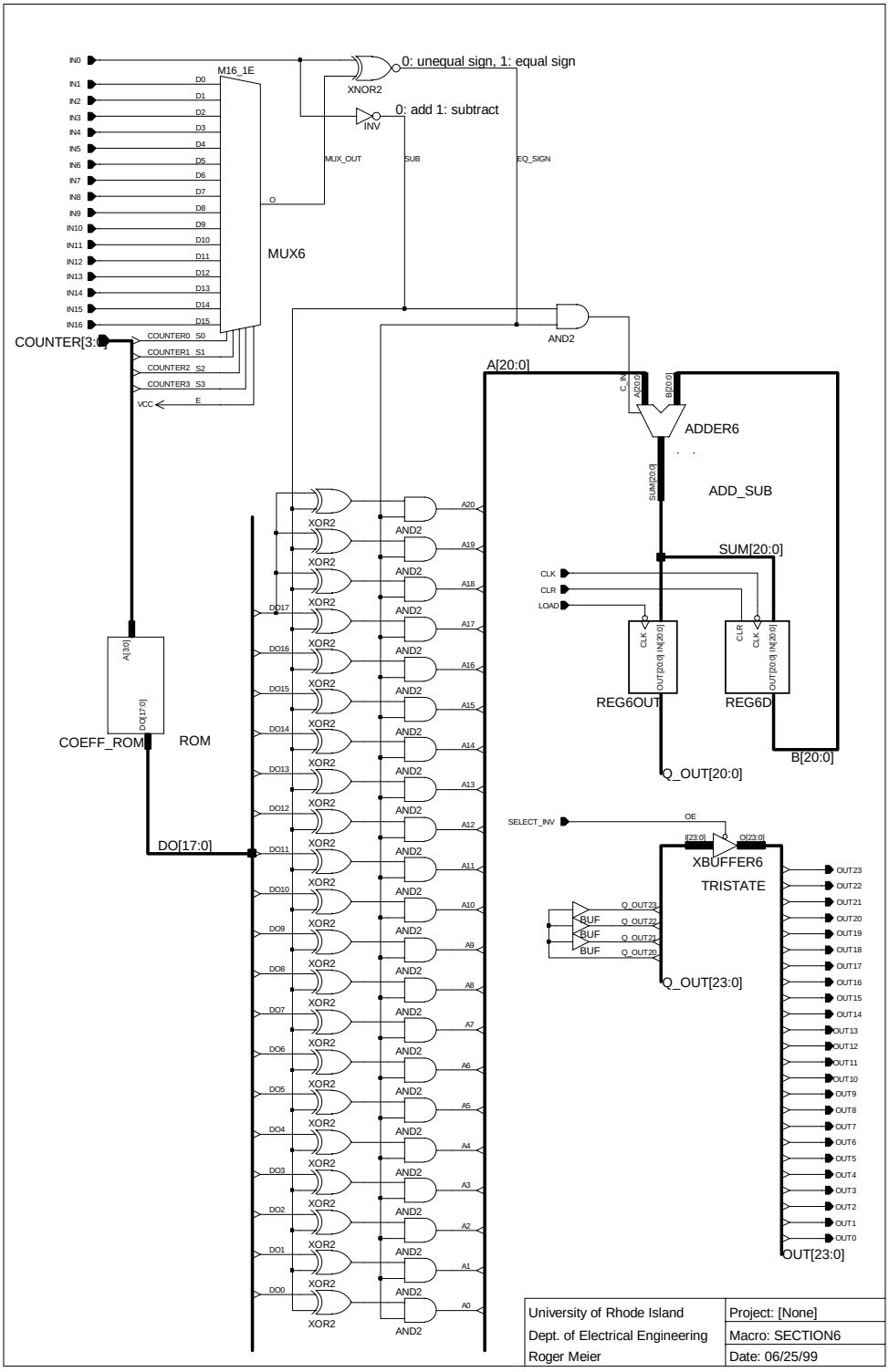

0: unequal sign, 1: equal sign
XNOR2
0: add 1: subtract
INV
M16_1E
MUX1
MUX_OUT
SUB
EQ_SIGN
AND2
COUNTER[3:0]
A[13:0]
ADDER1
ADD_SUB
SUM[13:0]
GND
CLK
CLR
LOAD
REG1OUT
REG1D
B[13:0]
Q_OUT[13:0]
COEFF_ROM
ROM
DO[10:0]
XOR2
SELECT_INV
OE
XBUFFER1
TRISTATE
BUF
Q_OUT[23:0]
OUT[23:0]
University of Rhode Island
Dept. of Electrical Engineering
Roger Meier
Project: [None]
Macro: SECTION1
Date: 06/25/99

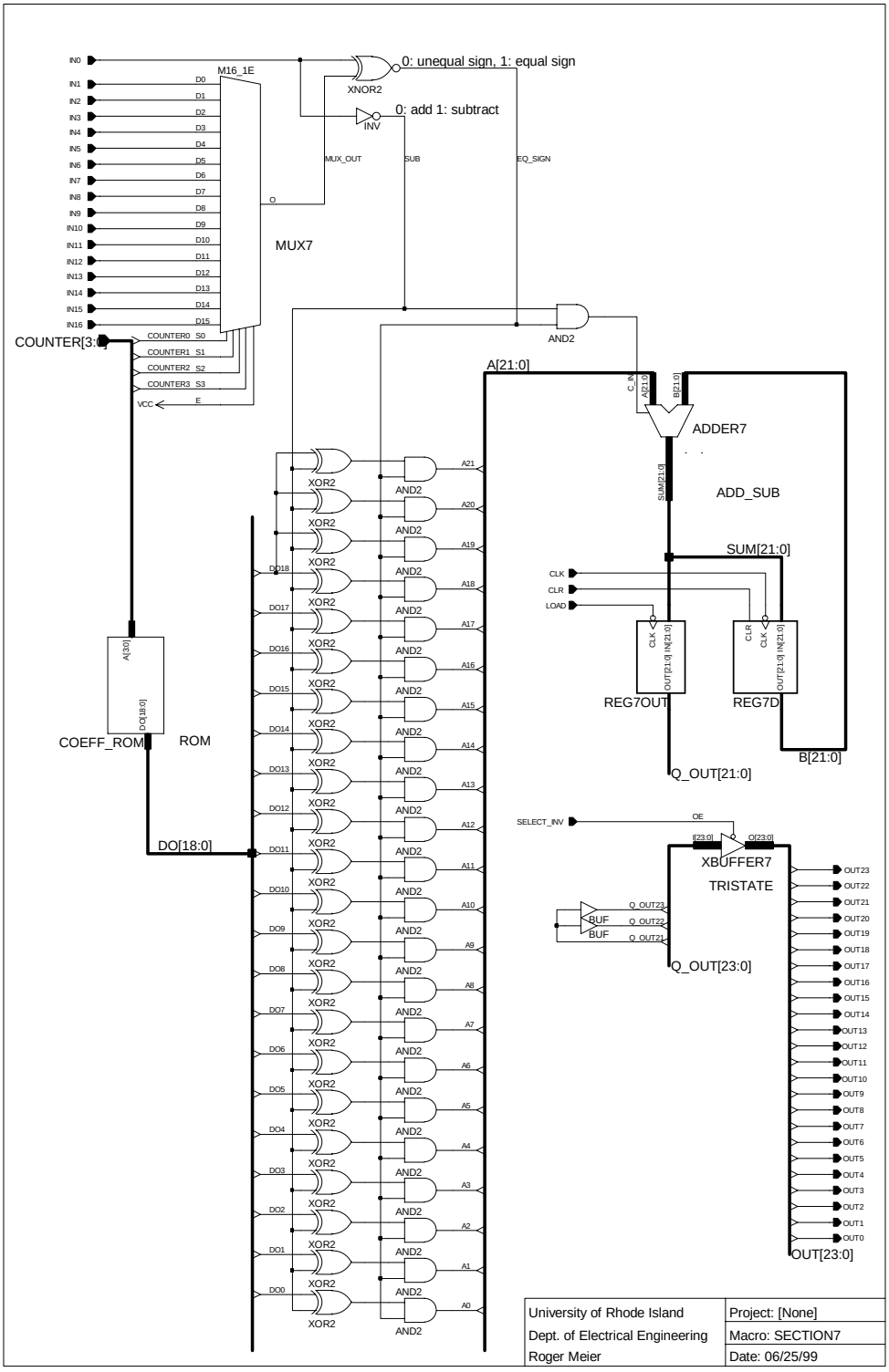

0: unequal sign, 1: equal sign
XNOR2
0: add 1: subtract
INV
M16_1E
MUX2
MUX_OUT
SUB
EQ_SIGN
AND2
COUNTER[3:0]
A[16:0]
ADDER2
ADD_SUB
SUM[16:0]
GND
CLK
CLR
LOAD
REG2OUT
REG2D
B[16:0]
Q_OUT[16:0]
COEFF_ROM
ROM
DO[12:0]
XOR2
AND2
SELECT_INV
OE
XBUFFER2
TRISTATE
Q_OUT[23:0]
OUT[23:0]
University of Rhode Island
Dept. of Electrical Engineering
Roger Meier
Project: [None]
Macro: SECTION2
Date: 06/25/99


0: unequal sign, 1: equal sign
XNOR2
0: add 1: subtract
INV
M16_1E
MUX3
MUX_OUT
SUB
EQ_SIGN
AND2
COUNTER[3:0]
A[17:0]
ADDER3
ADD_SUB
SUM[17:0]
CLK
CLR
LOAD
REG3OUT
REG3D
B[17:0]
Q_OUT[17:0]
COEFF_ROM3
ROM
DO[14:0]
XOR2
AND2
SELECT_INV
OE
XBUFFER3
TRISTATE
BUF
Q_OUT[23:0]
OUT[23:0]
University of Rhode Island
Dept. of Electrical Engineering
Roger Meier
Project: [None]
Macro: SECTION3
Date: 06/25/99

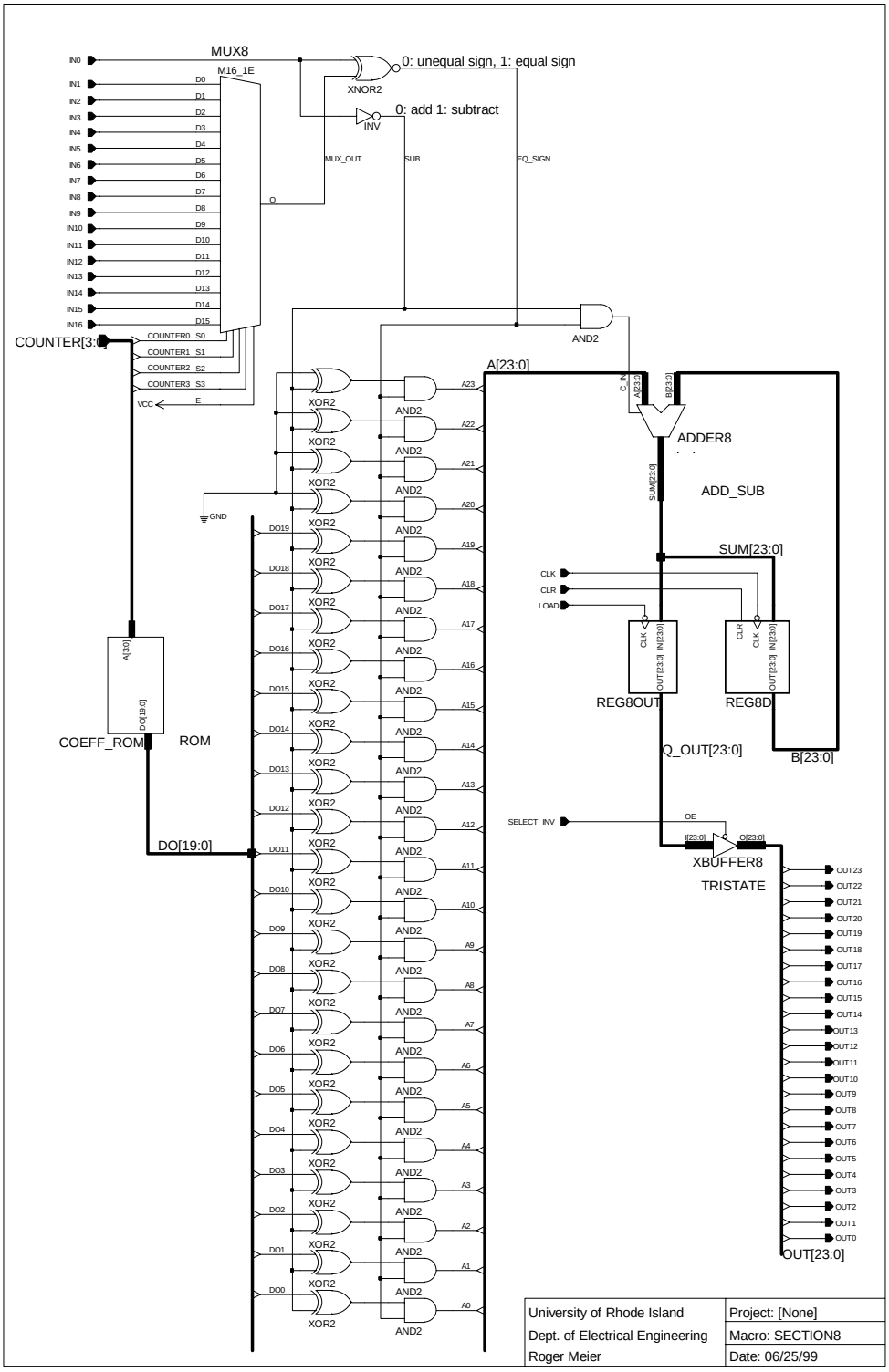

M16_1E
MUX4
XNOR2
0: unequal sign, 1: equal sign
INV
0: add 1: subtract
MUX_OUT
SUB
EQ_SIGN
AND2
COUNTER[3:0]
A[18:0]
ADDER4
ADD_SUB
SUM[18:0]
REG4OUT
REG4D
Q_OUT[18:0]
B[18:0]
COEFF_ROM
ROM
DO[14:0]
XOR2
AND2
SELECT_INV
XBUFFER4
TRISTATE
Q_OUT[23:0]
OUT[23:0]
University of Rhode Island
Dept. of Electrical Engineering
Roger Meier
Project: [None]
Macro: SECTION4
Date: 06/25/99


0: unequal sign, 1: equal sign
XNOR2
0: add 1: subtract
INV
M16_1E
MUX5
MUX_OUT
SUB
EQ_SIGN
AND2
A[19:0]
ADDER5
ADD_SUB
SUM[19:0]
GND
XOR2
AND2
CLK
CLR
LOAD
REG5OUT
REG5D
B[19:0]
Q_OUT[19:0]
COUNTER[3:0]
COUNTER0
COUNTER1
COUNTER2
COUNTER3
VCC
COEFF_ROM5
ROM
DO[15:0]
SELECT_INV
OE
XBUFFER5
TRISTATE
BUF
Q_OUT[23:0]
OUT[23:0]
University of Rhode Island
Dept. of Electrical Engineering
Roger Meier
Project: [None]
Macro: SECTION5
Date: 06/25/99


0: unequal sign, 1: equal sign
XNOR2
0: add 1: subtract
INV
M16_1E
MUX6
MUX_OUT
SUB
EQ_SIGN
COUNTER[3:0]
AND2
A[20:0]
ADDER6
ADD_SUB
SUM[20:0]
CLK
CLR
LOAD
REG6OUT
REG6D
B[20:0]
Q_OUT[20:0]
COEFF_ROM
ROM
DO[17:0]
SELECT_INV
OE
XBUFFER6
TRISTATE
Q_OUT[23:0]
OUT[23:0]
XOR2
AND2
BUF
University of Rhode Island
Dept. of Electrical Engineering
Roger Meier
Project: [None]
Macro: SECTION6
Date: 06/25/99



University of Rhode Island	Project: [None]
Dept. of Electrical Engineering	Macro: SECTION7
Roger Meier	Date: 06/25/99


MUX8
M16_1E
0: unequal sign, 1: equal sign
XNOR2
0: add 1: subtract
INV
MUX_OUT
SUB
EQ_SIGN
COUNTER[3:0]
AND2
A[23:0]
ADDER8
ADD_SUB
SUM[23:0]
GND
VCC
CLK
CLR
LOAD
REG8OUT
REG8D
COEFF_ROM
ROM
DO[19:0]
Q_OUT[23:0]
B[23:0]
SELECT_INV
OE
XBUFFER8
TRISTATE
OUT[23:0]
XOR2
AND2
University of Rhode Island
Dept. of Electrical Engineering
Roger Meier
Project: [None]
Macro: SECTION8
Date: 06/25/99

References

[1] L. S. Thomas P. Krauss and J. N. Little, *Signal Processing Toolbox*. The MathWorks, Inc., 1994.

[2] G. Fischer and A. J. Davis, "Delta-sigma modulation," in *Wiley Encyclopedia of Electrical and Electronics Engineering* (J. G. Webster, ed.), vol. 19, pp. 244–254, New York: John Wiley & Sons, Inc., 1999.

[3] A. J. Davis and G. Fischer, "Behavioral modeling of delta-sigma modulators," *Journal of Computer Standards and Interfaces*, vol. 19, pp. 189–203, September 1998.

[4] R. Kusch, "Decimation lowpass filters for delta-sigma modulators - a comparative study," tech. rep., University of Rhode Island, Department of Electrical Engineering, 1998.

[5] L. B. Jackson, *Digital Filters and Signal Processing*. Kluwer Academic Publishers, 1995. ISBN 0-7923-9559-X.

[6] T. W. Parks and J. H. McClellan, "Chebyshev approximation for nonrecursive digital filters with linear phase," *IEEE Transactions on Circuit Theory*, vol. CT-19, pp. 189–194, March 1972.

[7] D. S. K. Chan and L. Rabiner, "Analysis of quantization errors in the direct form for finite impulse response digital filters," *IEEE Transactions on Audio Electroacoustics*, vol. AU-21, pp. 189–194, August 1973.

[8] G. Fischer and A. J. Davis, "Vlsi implementation of a wide-band sonar receiver," in *Proceedings of the ISCAS'00*, (Geneva), 28-31 May 2000.

[9] A. J. Davis and G. Fischer, "Delsi - A design and simulation tool for delta-sigma modulators," in *IMEKO* 2^{nd} *International Workshop on ADC Modeling and Testing*, (Tampere, Finland), pp. 213–218, 1-3 June 1997.

[10] N. H. E. Weste and K. Eshraghian, *Priciples of CMOS VLSI Design*. Addison Wesley, second ed., 1993. ISBN 0-201-53376.

[11] Xilinx, Inc., San Jose, CA, *Foundation Series Quick Start Guide 1.5*.

Bibliography

Chan, D. S. K. and Rabiner, L., "Analysis of quantization errors in the direct form for finite impulse response digital filters," *IEEE Transactions on Audio Electroacoustics*, vol. AU-21, pp. 189–194, August 1973.

Davis, A. J. and Fischer, G., "Behavioral modeling of delta-sigma modulators," *Journal of Computer Standards and Interfaces*, vol. 19, pp. 189–203, September 1998.

Davis, A. J. and Fischer, G., "Delsi - A design and simulation tool for delta-sigma modulators," in *IMEKO* 2^{nd} *International Workshop on ADC Modeling and Testing*, (Tampere, Finland), pp. 213–218, 1-3 June 1997.

Fischer, G. and Davis, A. J., "Delta-sigma modulation," in *Wiley Encyclopedia of Electrical and Electronics Engineering*, (Webster, J. G., ed.), pp. 244–254, New York: John Wiley & Sons, Inc., 1999.

Fischer, G. and Davis, A. J., "Vlsi implementation of a wide-band sonar receiver," in *Proceedings of the ISCAS'00*, (Geneva), 28-31 May 2000.

Jackson, L. B., *Digital Filters and Signal Processing*. Kluwer Academic Publishers, 1995. ISBN 0-7923-9559-X.

Kusch, R., "Decimation lowpass filters for delta-sigma modulators - a comparative study," Technical report, University of Rhode Island, Department of Electrical Engineering, 1998.

Parks, T. W. and McClellan, J. H., "Chebyshev approximation for nonrecursive digital filters with linear phase," *IEEE Transactions on Circuit Theory*, vol. CT-19, pp. 189–194, March 1972.

Thomas P. Krauss, L. S. and Little, J. N., *Signal Processing Toolbox*. The MathWorks, Inc., 1994.

Weste, N. H. E. and Eshraghian, K., *Priciples of CMOS VLSI Design*. Addison Wesley, second ed., 1993. ISBN 0-201-53376.

Xilinx, Inc., San Jose, CA, *Foundation Series Quick Start Guide 1.5*.