LOW POWER AUDIO CODEC

by

Edwin W. Chan

Memorandum No. UCB/ERL M96/48

22 August 1996

LOW POWER AUDIO CODEC

by

Edwin W. Chan

Memorandum No. UCB/ERL M96/48

22 August 1996

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

Abstract

An audio codec (coder and decoder) is the interface between the analog domain and the digital domain in a talking device. The reason why the analog audio signals are converted to digital signals is that it is easier to manipulate and store digital rather than analog data.

The goal of this project is to cut down the power dissipation of the current audio codec in the InfoPad Project as much as possible and to increase the signal bandwidth. This report shows that by building a custom chip for the InfoPad, it is possible to cut down the power dissipation of the codec by an order of magnitude, from 40 mW to 3 mW.

Both the encoding and the decoding path of the codec are described, particularly the design details of the low power, linear phase decimation filter in the encoding path. The filter has a signal bandwidth of 7 kHz and an out-of-band attenuation of 60 dB. The power estimation is less than 0.4 mW. It consists of four stages. The first stage is a cascade of four comb filters. The second and third stage are halfband filters. The last stage is a droop-correction filter. Several different approaches to lower the power dissipation of the filter will be discussed in this report.

Acknowledgments

I would like to extend my sincere appreciation to Professor Bernhard Boser for his guidance and support over the past two years. The knowledge that I treasure from him is beyond what I can learn from textbooks. Also I would like to thank Professor Robert Brodersen for giving insightful directions to the project, and reading this thesis.

Besides, I am grateful to the help of the research group of Professor Brodersen. In particular, many thanks go to Dr. Andy Burstein, Dr. Sam Sheng, Tom Burd and Ian O'Donnell for solving a lot of CAD tool problems that I encountered.

In addition, I would like to thank my fellow officemates in my own group. Tom Wongkomet, Darrin Young, Colby Boles, Monico Ortiz, Eugene Cheung, Mark Lemkin and I have shared many happy and depressing moments together.

Also I would like to thank Martin Tsai and Wai Lau for beating me in Risk, Pramote Piriyapoksombut, Keng Fong and Tony Lin for beating me in Chess, George Chien and Jeff Ou for beating me in Bridge, Li Lin for beating me in Go, as well as all those who beat me on the basketball court.

Special thanks go to Steve Lo for working out with me in early mornings and helping me find a job without a referral bonus.

Last but not the least, I am indebted to my parents, my brother and my sister for their continuing financial and moral support. Emailing with my brother and my sister at 4 a.m. in the morning helped me work on my project without falling asleep.

This research was supported by the Advanced Research Projects Agency.

CHAPTER 1

Introduction

1.1 Goal and Motivation

In a portable communication device, a low power design is very important to increase the battery life. In the InfoPad Project [25], [26], the existing audio interface employs a general purpose codec from Motorola (MC145554), which has a power dissipation of 40mW. This is not desirable, especially when compared to the power dissipation of the other InfoPad custom chips, like the Protocol Chip (1 mW) [28], and the video module (< 2mW) [29]. The existing codec has a signal bandwidth of 3 kHz and has a SNDR of 35 dB with a dynamic range of 75 dB (using mu-255 law).

This work discusses the design of a codec with more than an order of magnitude reduced power dissipation. The new codec cuts down the power dissipation from 40 mW to less than 3 mW, and it increases the signal bandwidth from 3kHz to 7kHz for better audio recognition. Table 1.1 summaries the key specifications of the new codec and the existing codec.

Table 1.1 : Key Specifications Of The Proposed And The Existing Codec

	Existing Codec (MC145554)	Proposed Codec
Nyquist Rate	8 kHz	16 kHz
Sampling Rate	8 kHz	64 x 16 kHz (1.024 MHz)
Bandwidth	3 kHz	7 kHz
Dynamic Range	~75 dB (mu-255 law)	>72 dB
SNDR	~35 dB	>72 dB
Power Dissipation (active)	40 mW	<3 mW

Figure 1.1 shows the block diagram of the audio codec. In the encoding path, there are an anti-alias filter, an analog ΣΔ modulator and a decimation filter. It takes the analog signal from the microphone of a headset and gives a corresponding digital code. The digital code will then be sent to the audio chip of the InfoPad for storage or further computation. In the decoding path, there are an interpolation filter, a digital ΣΔ demodulator and a smoothing filter. The codec picks up the digital data from the audio chip and outputs an analog signal to the earphones of a headset.

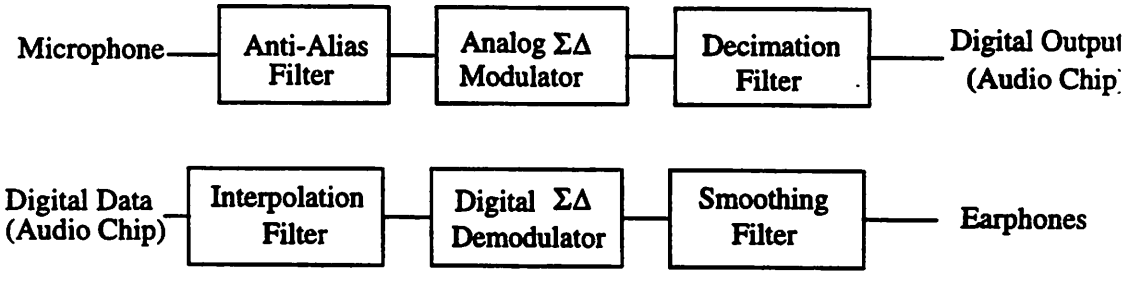


Figure 1.1: The Block Diagram Of An Audio Codec

In the encoding path, an oversampling ΣΔ modulator is used to move the quantization noise to a higher frequency band. This is why oversampling is a very attractive technique for low frequency applications. Since we are using an oversampling modulator, the requirements for the anti-alias filter in the front end are not stringent. It can be as simple as a two or three pole Butterworth lowpass filter, implemented with an active RC network.

A decimation filter is then used to transform the digitally modulated signal from short words occurring at a high sampling rate to longer words at the Nyquist rate. Hence a steep cutoff characteristic is required to avoid aliasing. With a Nyquist rate of 16kHz, the transition band of the filter is permitted to extend from 7kHz to 9kHz.

Besides attenuating the quantization noise above the signal band from the analog ΣΔ modulator, the decimation filter also suppresses other out-of-band noise within the signal path, such as noise from the power supplies, substrate coupling or high frequency signal components. We will ensure that the out-of-band noise will be attenuated by at least 60dB before being aliased into the signal band (0-7kHz).

Other requirements on the decimation filter include the in-band ripples and phase distortion. We would like to limit the ripples to be no more than 0.1 dB. To avoid phase distortion, we will make use of alinear-phase filter design.

Table 1.2 summarizes the design specifications of the decimation filter. This report will detail the whole design process to meet these specifications.

Table 1.2: Specifications Of The Decimation Filter

Transition band	7 kHz - 9 kHz
out-of-band attenuation	60 dB
in-band ripple	+/- 0.1 dB
phase distortion	linear phase

In the decoding path, an interpolation filter is used to raise the input word rate to a frequency well above the Nyquist rate. The output words are then truncated and converted to the analog form at the high sample rate by the oversampled demodulator. The digital demodulator, like its analog counterpart, also shapes the quantization noise to a higher frequency band. Hence a smoothing filter is used to remove the out-of-band noise, as well as to avoid distortion in the power amplifier and ear phones.

1.2 Thesis organization

This thesis is organized into five chapters. Chapter 2 discusses the function and the architecture of the building blocks in the encoding path, the anti-alias filter, the analog $\Sigma\Delta$ oversampling modulator and the decimation filter. Chapter 3 describes the actual implementation and the approaches to reduce the power dissipation of the decimation filter. Time domain verification and the layout of the filter are also included. In Chapter 4, some of the design issues of the interpolation filter, the digital $\Sigma\Delta$ demodulator and the smoothing filter in the decoding path of the audio codec are discussed. Finally, Chapter 5 draws conclusions.

In addition, Appendix A lists the actual coefficients of different stages in the decimation filter, the allocation of the RAM spaces to the different stages and the ROM control codes.

CHAPTER 2

Encoding Path

2.1 Overview

In this chapter, we will discuss the function and architectures of each building block in the encoding path. We first begin with an anti-alias filter. A 3rd order lowpass active RC filter is given as an example, and the power dissipation of the filter is estimated. Then the structure and the performance of a 3rd order analog $\Sigma\Delta$ modulator satisfying the requirements in Table 1.1 will be presented. Finally, a 4-stage decimation filter will be shown to be suitable in our application. The implementation and the power estimation of the decimation filter will be discussed in the next chapter.

2.2 Anti-Alias Filter

An anti-alias filter is a lowpass continuous-time analog filter. It prevents the high frequency noises from folding back to the signal bandwidth due to sampling. Only the part of the input spectrum which extends beyond ($f_{sample}/2 - f_{signal}$) is responsible for aliasing. Consequently, an anti-alias filter bears the characteristics of passing the baseband signal with the minimal distortion while attenuating the noise outside ($f_{sample}/2 - f_{signal}$), as shown in Figure 2.1.

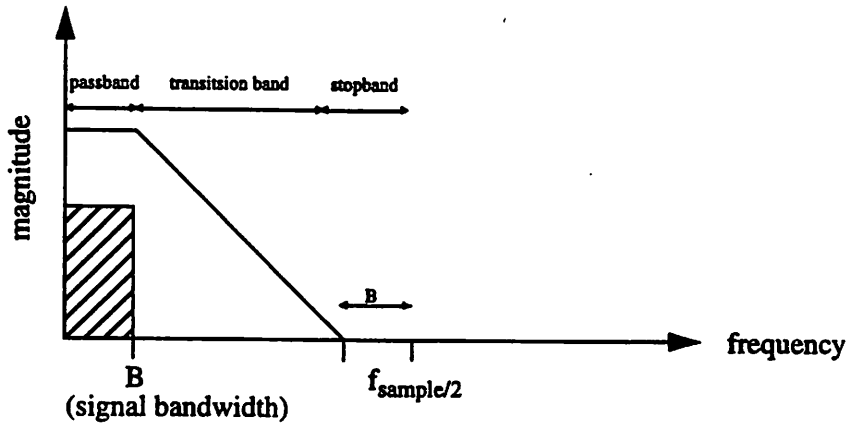


Figure 2.1: Characteristics Of An Anti-Alias Filter

Oversampling filters usually have a sampling frequency much higher than Nyquist rate samplers, so the transition band of the anti-alias filter for an oversampler can be much wider. Since the complexity of a lowpass filter is a strong function of the ratio of the width of the transition band to the width of the passband, the anti-alias filter for an oversampler is also much simpler.

In our applications, the sampling frequency of the analog ΣΔ modulator is 1.024 MHz while the signal bandwidth is 7 kHz. A suitable filter can be a 3rd order Butterworth lowpass filter [20] with a half-power frequency of 63 kHz. It gives a 72 dB attenuation at 505 kHz ($f_{sample/2}$ - B), where B is the signal bandwidth. It introduces less than 0.05 degree phase distortion and less 0.05 dB droop in the signal band.

The anti-alias filter can be implemented using an active RC filter, containing thin-oxide capacitors and polysilicon resistors. Figure 2.2 shows a 3rd order lowpass filter structure using a cascade of a 2nd Order lowpass filter and first order section [20].

In Figure 2.2, two opamps are needed in the 3rd order lowpass active RC filter. A 1.5 V class AB CMOS opamp described in [21] has an average power dissipation of 0.135 mW, using a 2μm CMOS technology. In this project, a better fabrication technology, 0.8 μm CMOS technology, is used, so the power dissipation of the anti-alias filter is expected to be lower than 0.3 mW.

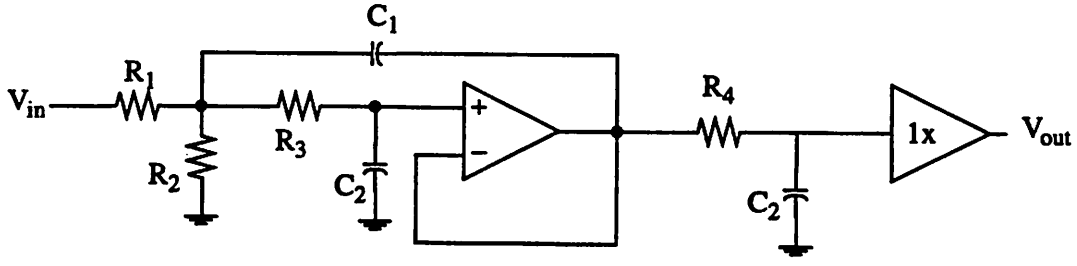


Figure 2.2: Structure of A 3rd Order Lowpass Active RC Filter

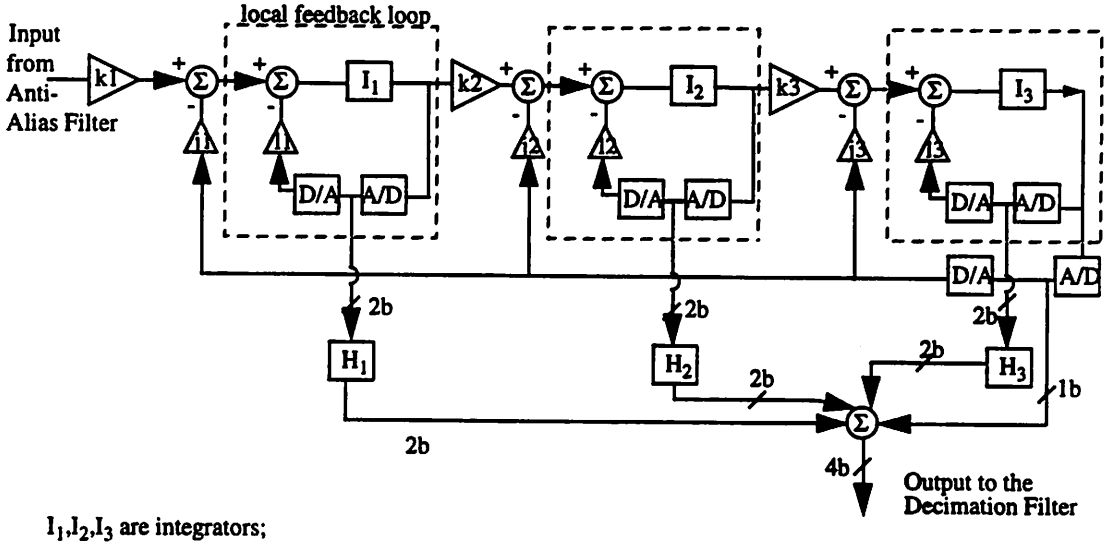
2.3 Analog Oversampling Sigma-Delta Modulator

An analog oversampling ΣΔ modulator takes the output of the anti-alias filter and performs a coarse amplitude quantization at a frequency much higher than the Nyquist rate and outputs a PCM (pulse code modulation) code. It trades the resolution in time for resolution in amplitude, so less precise analog circuits may be used. The output quantization noise spectrum will be shaped, that is, the quantization noise energy is moved towards higher frequencies, which will then be eliminated by the following digital decimation filter.

Two important parameters in designing an analog oversampling ΣΔ modulator are the order of the modulator and the oversampling ratio (OSR), which is the ratio of the half of the sampling frequency to the Nyquist rate (f_{sample} / (2 * Nyquist Rate)). A higher order modulator requires a lower OSR, but increases the complexity of the modulator structure. There are two advantages of keeping a low OSR. One of them is that the requirements of the opamp can be made less stringent, because the modulator is running at a lower frequencies. The other advantage is that it can help reduce the power dissipation of the following digital decimation filter.

With a first or second order modulator, stability is almost guaranteed. However, for a higher order modulator, stability is maintained either by using a multi-bit quantizer or using a multistage architecture (MASH) [12]. In the first case, the design challenge comes from maintaining the speed and linearity of the quantizer. In the latter case, the challenge comes from the matching accuracy between the noise transfer function of the analog modulator and the digital differentiator.

It is found that a 3rd order oversampled modulator with local feedback loops [11] gives a reasonable compromise between the order and the complexity of the modulator. The structure of the modulator is show in Figure 2.3. It is a single stage architecture with three feedback loops. Stability is maintained by the local feedback loops. The output is a 4-bit digital output code. From [11], with a rail-to-rail voltage of 2.5V, and an OSR of 64, or a sampling frequency of 1.024 MHz, the modulator achieves 73 dB for a signal bandwidth of 8 kHz. The input range is from -0.3 V to 0.3V and dissipates 0.83 mW.



I_1, I_2, I_3 are integrators;
j,k,l are scaling factors;
H_1, H_2, H_3 are digital cancellation circuits for the local feedback loops.

Figure 2.3: 3rd Order ΣΔ Modulator With Local Feedback Loops

2.4 Decimation Filter

The main function of a decimation filter is to transform the modulated signal from short words at a high sampling rate to longer words at the Nyquist rate. It is easy to argue that a sinc^k filter is very efficient in removing the out-of-band noise of a $(k-1)^{th}$ order ΣΔ modulator [5].

In our case, the decimation filter is designed for a 3rd order ΣΔ modulator, so a 4^{th} order sinc filter is used in our design. However the in-band droop of the $sinc^4$ filter is intolerable if the decimation is only done by the $sinc^4$ filter. This leads us to split the decimation process to more than one stage. Figure 2.4 shows that in order to decimate an input signal by a factor of 64, the signal is first decimated by (64/N) times in the $sinc^4$ filter, and further decimated by N times in an additional digital filter.

A droop-compensation filter is added to the architecture to compensate as much in-band droop produced by the 4^{th} order sinc filter as possible.

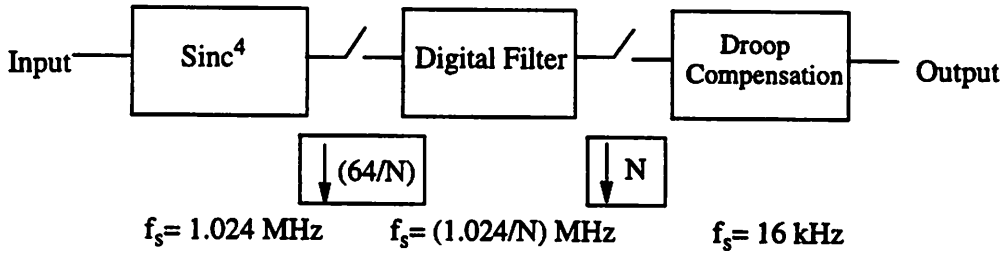


Figure 2.4: The Basic Structure Of The Decimation Filter

In the next two sub-sections, we will contrast the performance and complexity of two architectures. One of them is a 3-stage design, in which the second decimation factor, N, of the digital filter is 2. The other one is a 4-stage design, in which N is 4.

2.4.1 Three-Stage Decimation Filter

The first architecture that we look into is to have the second decimation factor, N, equal to 2. Halfband filters are particularly suitable for this 2-to-1 decimation [6], because they have the the characteristics of passing signals up to a half of the Nyquist frquency, or a quarter of the Nyquist rate. The implementation of halfband filters will be discussed more in details in Chapter 3. Consequently, we come up with the architecture as shown in Figure 2.5. The halfband filter is designed by using the design "trick" as illustrated in [8] and the impulse response coefficients of this filter are listed in Appendix A. Figure 2.6 shows the performance of the three-stage decimation filter.

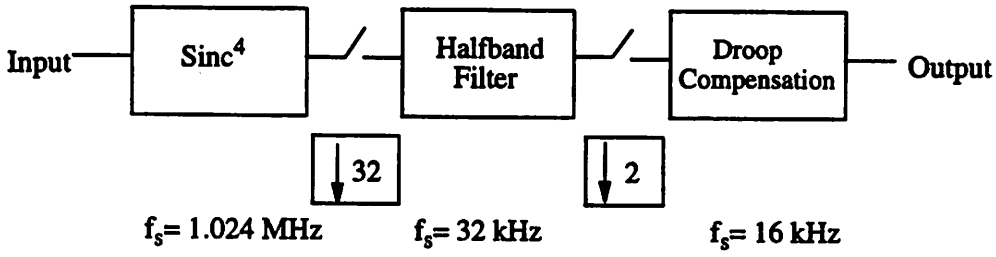


Figure 2.5: The 3-Stage Decimation Filter

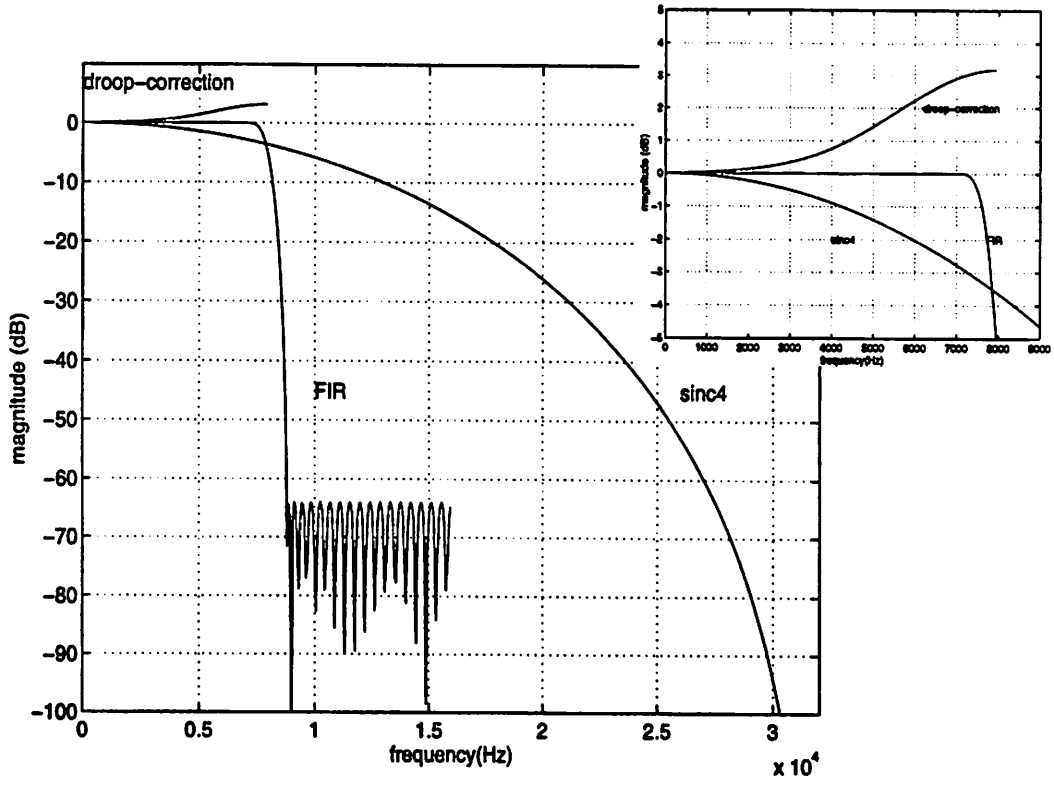


Figure 2.6: The Performance Of The 3 Stages Of The First Architecture

The transition band is from 7 kHz to 9 kHz. The in-band droop after the halfband filter is 2.8 dB. With the 15-tap droop-compensation filter, which consists of 8 non-zero taps, the in-band droop can be corrected up to +/- 0.2 dB, which is still not very desirable.

2.4.2 Four-Stage Decimation Filter

Another architecture we look into is a 4-stage decimation filter. It has the same basic structure as in Figure 2.4, with the second decimation factor of 4. Instead of designing a sharp cutoff digital filter with a decimation ratio of 4, we make use of two halfband filter stages, as shown in Figure 2.7.

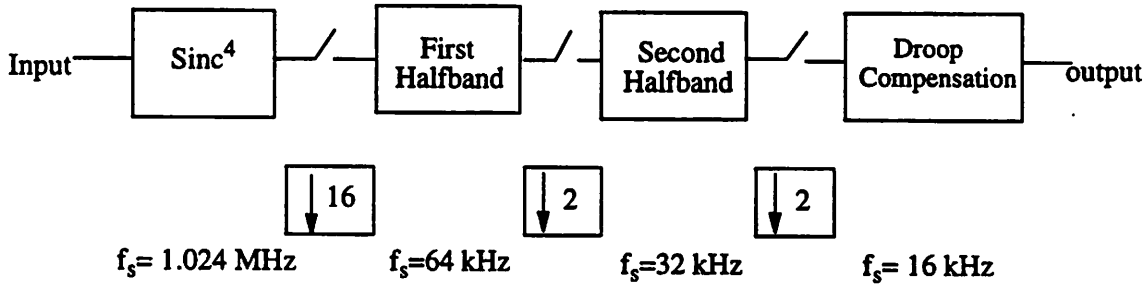


Figure 2.7: The 4-Stage Decimation Filter

The transition bandwidth can be less stringent in the first half-band filter. The F8 half-band filter given in [6], with a scaling factor, is used in our design. There are only 5 non-zero coefficients in this filter. The second half-band filter is the same as the one in section 2.4.2. The impulse response coefficients are also listed in Appendix A. Figure 2.8 shows the performance of the four stages of the filter.

The in-band droop before the droop-compensation filter is 0.68 dB. With a 7-tap droop-compensation filter, that is, 4 non-zero coefficients, the droop can be reduced to +/-0.02dB, which is well below the 0.1 dB specification.

We can see that both architectures have the same transition bandwidth, but the in-band ripples in the second architecture, +/- 0.02 dB, is much better than that of the first one, +/- 0.2 dB. Also,

the additional filter needed in the second architecture, which has only 5 non-zero coefficients, is somewhat compensated by the simpler droop-correction filter (4 coefficients less). Hence, the second architecture is adopted. Figure 2.9 shows the overall performance of the decimation filter, including the sinc filter, two halfband filters and the droop-compensation filter.

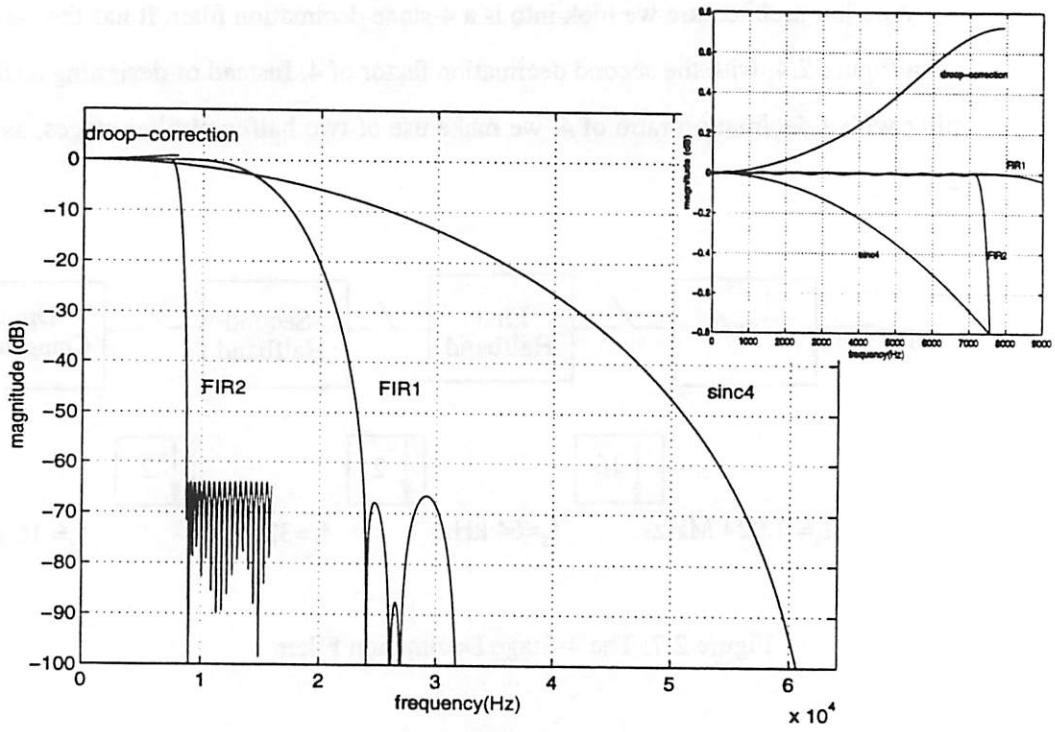


Figure 2.8: The Performance Of The 4 Stages Of The Second Architecture

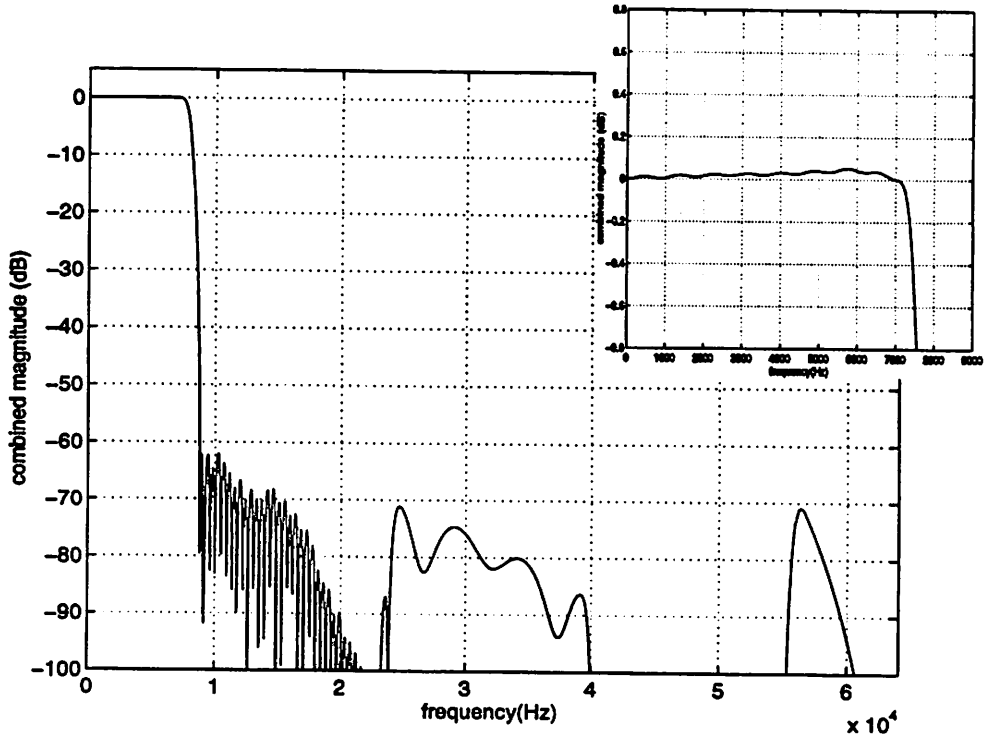


Figure 2.9: The Performance Of The Overall Filter Performance

2.5 Summary

Both the three-stage and four-stage decimation filters are considered. The four-stage decimation filter has less drooping effect in the baseband, but the number of non-zero coefficients for both filters are about the same. However, as we will cover the multirate addressing scheme in Section 3.4.2, the additional stage will result a more complex scheme. Nevertheless, the increase in complexity in the addressing scheme is justified by the better performance, so the four-stage architecture is more appropriate for the decimation filter. The implementation and the power dissipation of the decimation filter will be discussed in the next chapter.

Besides the functions of the other building blocks of the encoding path are discussed. The total power dissipation of the anti-alias filter and the analog $\Sigma\Delta$ modulator is found to be less than 1.1mW.

CHAPTER 3

Decimation Filter -- Implementation

3.1 Overview

In Chapter 2, we discussed two different architectures of the decimation filter. In this chapter, we will look at the actual implementation of the filter. We will discuss how to implement the $sinc^4$ function, using a cascade of 4 comb filters, and the halfband filters. As mentioned in Chapter 2, we have two halfband filter stages, each of which running at a different frequency. Section 3.3 will introduce a multirate addressing scheme, which only needs a single RAM and a single ROM to store all the required taps for both the halfband filters and the droop-compensation filter. Simulation results and the power estimation will be presented in Section 3.4.

3.2 Comb Filter

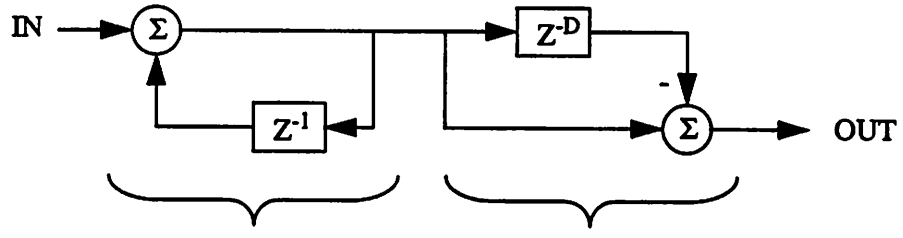
A comb filter of length D is an FIR filter with all D coefficients equal to one. It has a transfer function of

$$H(z) = \frac{1 - z^{-D}}{1 - z^{-1}} \qquad (3.1)$$

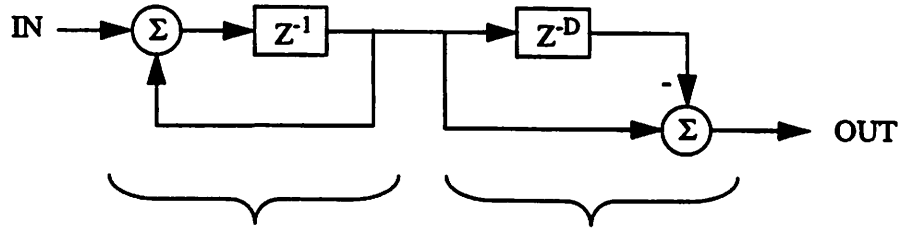
In the frequency domain,

$$H(e^{j\omega\tau}) = e^{-j\omega\tau \cdot \frac{(D-1)}{2}} \cdot \left(\frac{\sin\left(\frac{D\omega\tau}{2}\right)}{\sin\left(\frac{\omega\tau}{2}\right)} \right) \tag{3.2}$$

This will give the first order sinc function. From (3.1), the denominator can be implemented by a delay-free integrator (DFI), while the numerator can be implemented by a differentiator, as shown in Figure 3.1a. A delay block, z^{-1}, is simply a register. However, in the actual implementation, we would like to use a discrete digital integrator (DDI), that is, moving the delay block to the forward path (Figure 3.1b). Only a delay term will be added to the transfer function in (3.1). This serves as a pipeline action and cuts the critical path between the input and the output. It is particularly important in the higher order cases.



(a)



(b)

Figure 3.1: Single Stage Comb Filter

To obtain a 4th order sinc function, we just need to cascade 4 comb filters together, see Figure 3.2.

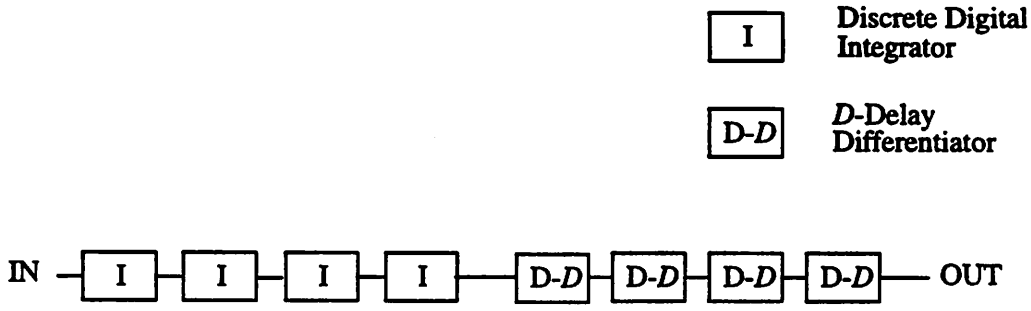


Figure 3.2: Implementation Of A 4th Order Sinc Function

Two more design issues have to be considered. One of them is the process of decimation and the other is the overflow problem in the integrators.

The most straight forward way to decimate the $sinc^4$ output by a factor of D is to subsample the output every D cycles. In other words, many output data will be thrown away. This will waste a lot of computational efforts. In order to only calculate the samples that we need, we would do the decimation between the integrator section and the differentiator section. To maintain the same transfer function, all the D-delay blocks in the differentiators are replaced by the unity delay block, as shown in Figure 3.3.

The data overflow problem in the integrator section can be overcome by using a two's complementation calculation, which allows wrap-around from the most positive number to the most negative. However, a minimum wordlength of each integrator has to be met. From [18],

$$\text{minimum wordlength} = N \log_2 D + B_{in} \qquad (3.3)$$

where N is the number of integrator stages, D is the decimation ratio and B_{in} is the number of input bits. In our design, N = 4, D = 16 and B_{in} = 4, the minimum wordlength is then 20 bits. The complete $sinc^4$ block is illustrated in Figure 3.3.

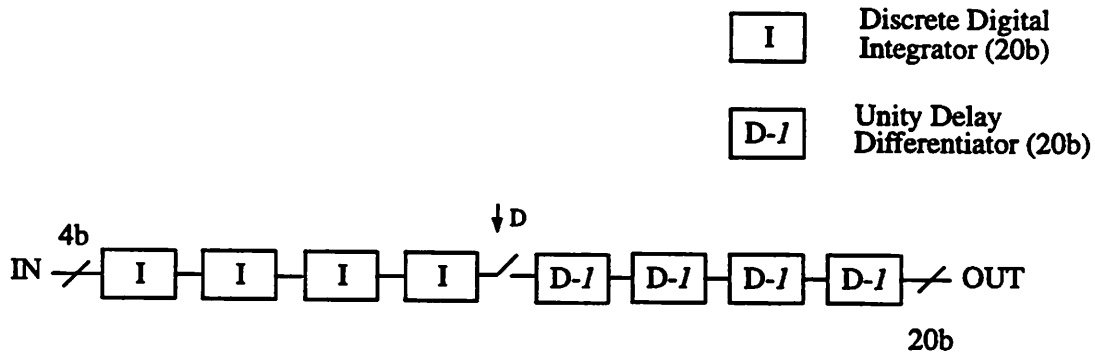


Figure 3.3: Implementation Of The Complete Sinc^4 Block

3.3 Direct Implementation Of Halfband Filters

A halfband filter is a linear-phase lowpass FIR filter, having the the characteristics of passing signals up to a half of the Nyquist frquency, or a quarter of the Nyquist rate. Therefore it is particularly attractive in decimating signal frequency by 2. The technique for designing a halfband filter (H) whose number of taps (N) is always odd is described in [8]. If we label the first coefficient, H(0), we find that all odd coefficients, except the middle one, H((N-1)/2), are zero.

As mentioned in the last chapter, there are two halfband filters. The first halfband filter has 5 non-zero coefficients while the second one has 19. A direct implementation of the first halfband filter is shown in Figure 3.4. However, to compute one output, the data stored in the registers need to be circularly shifted to a common point where they are multiplied by the appropriate coefficients. The large amount of data transfer not only has a considerable power dissipation, but also wastes a lot of area. In addition, in case of a cascade of several filters, one multiplier is needed for each filter.

In our example, X is a 22-bit, 64 kHz input. From Figure 3.4, there are 16 22b registers, 1 22b MUX, 1 22b adder, 1 22b multiplier, and a 15 word x 22 bit ROM. There are 15 registers in the tap line, so the filter needs to run at a frequency 15 times faster than the input frequency, that is, 960 kHz. Similarly, the second halfband filter needs 72 22b registers, 1 22b MUX, 1 22b adder, 1 22b

multiplier and a 71 word X 22 bit ROM. The input frequency of the second halfband filter is 32 kHz and the filter runs at a frequency 71 times faster than the input frequency.

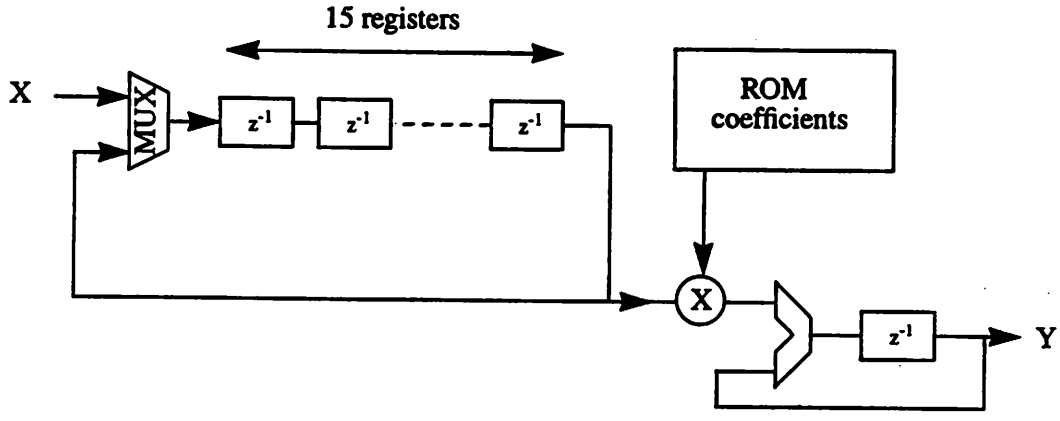


Figure 3.4: Direct Implementation Of The First Halfband Filter

We use a new tool developed in the University of California, Berkeley, PowerPlay, to estimate the power dissipation of this implementation. PowerPlay estimate the power dissipation of a component by calculating its effective capacitance ($P = C V^2 f$). The effective capacitance of each digital component is modelled as a set of formulas, based on measured data. The data for ROM are not available yet. We estimate the effective capacitance for ROM by using half of the capacitance for RAM.

Table 3.1 shows the power dissipation of the two halfband filters. For an input data rate of 64 kHz and a volatge supply of 1 V, the power estimation is about 870 μW. The high level of power dissipation is mainly due to the large amount of data shift, so direction implementation is definitely not a good soluation. In the next section, we will discuss some approaches that reduce the power dissipation down to below 300 μW.

Table 3.1: Power Dissipation of Two Halfband Filters

Description	Quantity	Effective Capacitance (pF)	Operating Frequency (kHz)	Power Dissipation (μW)
First Halfband Filter:				
15 word x 22 bit ROM	1	19[a]	960	18.2
22b register	16	2.1	960	32.3
22b adder	1	5.9	960	5.7
22b MUX	1	0.9	960	0.9
22b multiplier	1	122.5	960	117.6
Second Halfband Filter:				
71word x 22 bit ROM	1	25.7[b]	2272	58.4
22b register	72	2.1	2272	343.5
22b adder	1	5.9	2272	13.4
22b MUX	1	0.9	2272	2
22b multiplier	1	122.5	2272	278.3
			TOTAL	870.3

a. Estimation from 1/2 of the capacitance of RAM
b. Estimation from 1/2 of the capacitance of RAM

3.4 Approaches To Reduce Power Dissipation

One of the most efficient ways to reduce power dissipation is to lower the supply voltage. In our application, the decimation filter is running at around 2 MHz, which is slow enough to reduce the supply voltage to 1 V to 1.2 V. As the supply voltage is already very low, parallelism may not help in this case. In this section, four approaches to further reduce power dissipation are discussed. They are using a polyphase structure, employing a multirate addressing scheme, using a multiplier-free arithmetic & logic unit and making using of nested multiplication computation.

3.4.1 Polyphase Structure

Without loss of generosity, only the first halfband filter will be discussed here. The filter consists of 5 non-zero coefficients. A polyphase structure [2] shown in Figure 3.5 is used to implement the halfband filter.

The alternate switch at the input X, or commutator, directs the even input data to the upper branch and the odd to the lower branch. Since the halfband filter is linear phase, the impulse response coefficients are symmetrical, hence both branches are folded backwards. The splitting of the data into the even and odd branches is to take advantage of the fact that all but one odd coefficients are zero. This makes the implementation of the odd branch considerably simpler. All elements shown in broken lines in Figure 3.5 need not be implemented. This reduces the computational complexity by nearly 50 %. An output is only calculated when an even datum is input, that is, the input rate is twice the output rate. This automatically achieves the decimation process by a factor of two.

The second halfband filter is implemented in a similar fashion

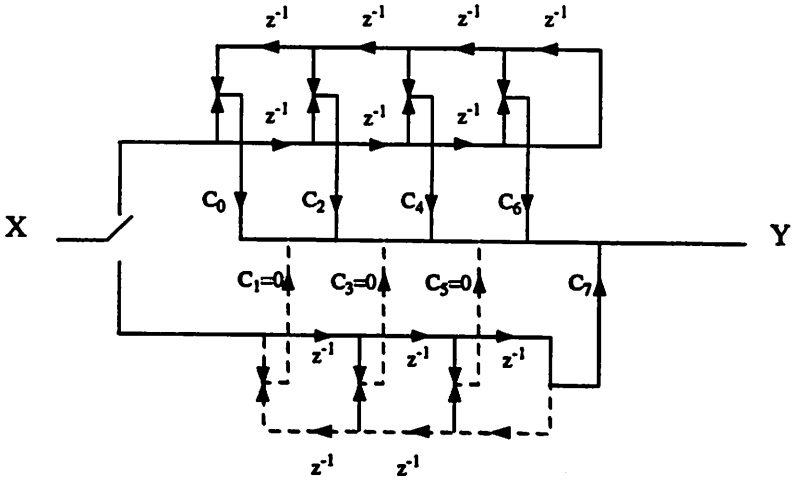


Figure 3.5: Polyphase Implementation Of The First Halfband Filter

3.4.2 Multirate Addressing Scheme

As we can see in Section 3.3, the filter tap line is implemented in a long line of registers. Alternatively, the filter input data can be stored in a RAM. Whenever a multiplication is to be done, only the corresponding tap is accessed. By using this method, data no longer need to be transferred in a circular fashion. The number of words in a RAM will equal the length of a tapped line. Each new word (datum) is stored in the next higher location in the RAM, compared to its previous word. After storing the last input in the last location of the RAM, the next input will be stored back in the 0th location of the RAM, hence overwriting the old datum.

The way we determine the correct address is shown in Figure 3.6. The correct address is the sum of a base address and a virtual address. The base address records the most recent input datum. It is incremented by one whenever a new datum is received. It is basically a counter. The virtual address records the delayed tap address, assuming the most recent datum is stored in 0^{th} location of the RAM. It is stored in a ROM. In other words, the locations in the RAM can be considered as the tapped line.

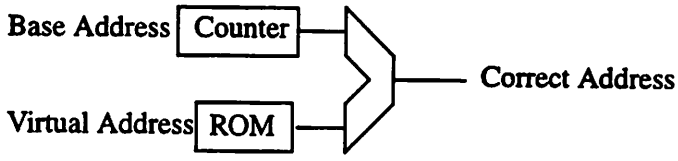


Figure 3.6: Implementation of the correct address of the RAM

To implement more than one filter, we can still use only one single RAM, provided the number of words of the RAM equals to the sum of the lengths of the tapped lines. Different base address counters are used to keep track of the most recent datum location for different filters.

However, the above scheme only works well for single rate filters. To implement several filters with different sampling rates, there will be a problem. For example, Figure 3.6 shows that 2 three-tapped filters, A and B, are implemented in a RAM, with A samples twice as fast as B. After a certain number of cycles, all the delayed tap values of B are overwritten.

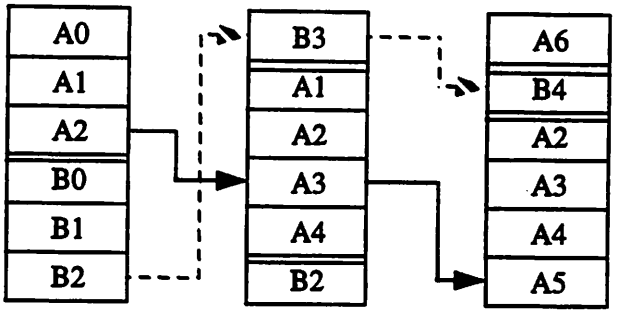


Figure 3.7: Problem of multirate filters in a single RAM

A multirate addressing scheme [2] is proposed to solve the problem. If a RAM is used to implement two multirate filters, say filter A samples n times faster than filter B, filter A data should be stored in forms of n sections. For example, filter A samples twice as fast as filter B, so filter A data are stored in two sections. Figure 3.8 shows how the multirate addressing scheme works. We can see that old data (compared to the delayed tap line) of filter A are overwritten by its own new data, instead of the "useful" delayed tap values of filter B. This idea can be extended to accommodate any number of stages of different sampling rates. However, the more number of stages is, the more complex of the scheme will be.

The actual allocations of the locations of the RAM to the last three filters of the four-stage decimation filter are listed in Appendix A.

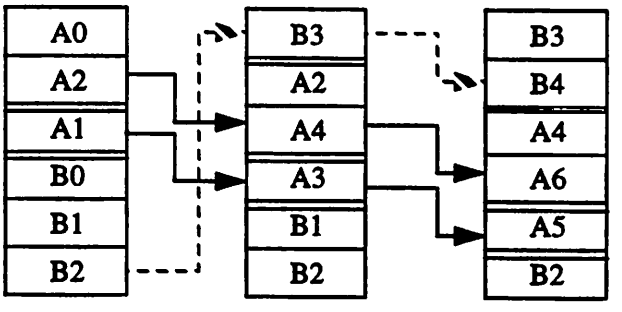


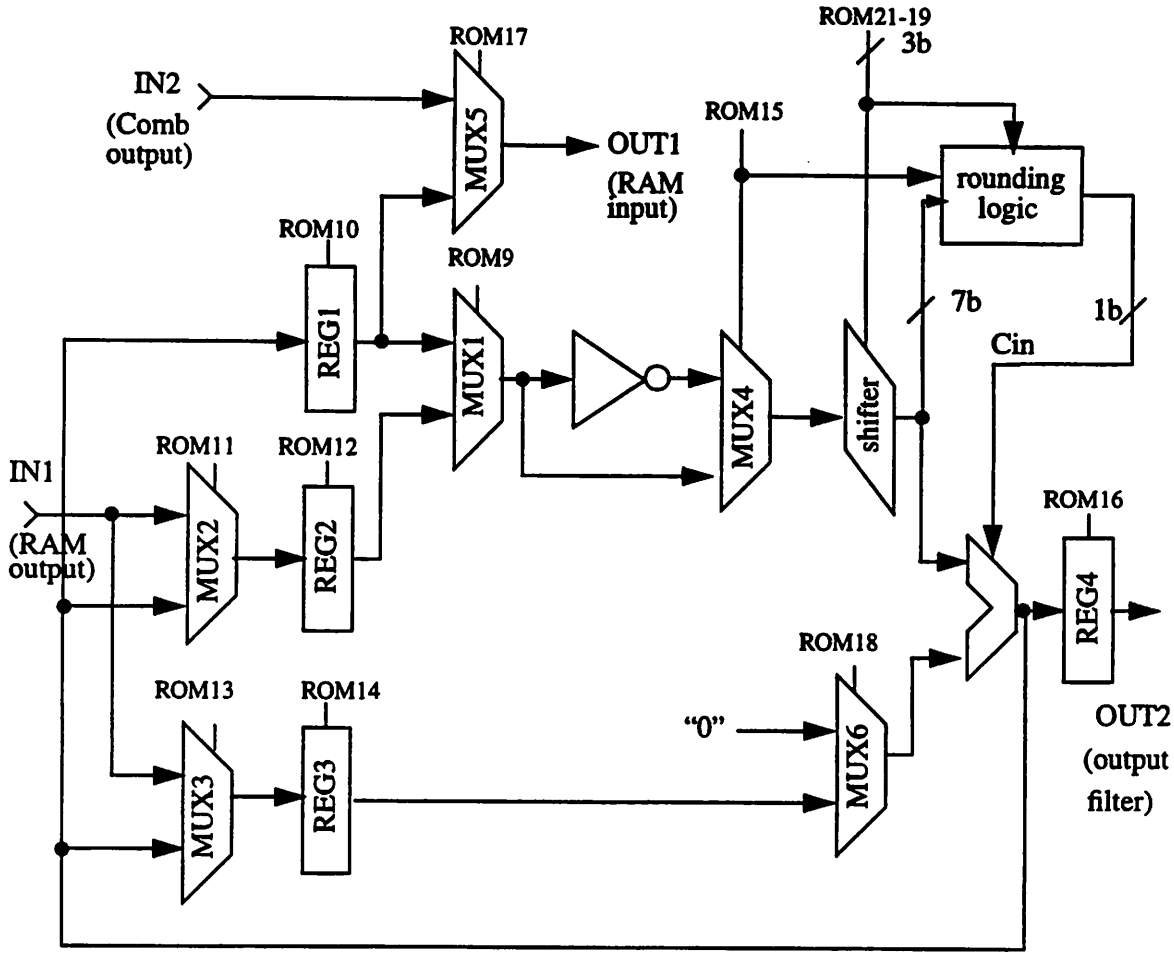
Figure 3.8: Multirate Addressing Scheme In A Single RAM

With this multirate addressing scheme, we get rid of long lines of registers and combine the tap lines of several filters into a single RAM. This help cutting down the power dissipation and the chip area.

3.4.3 Multiplier-Free Arithmetic Logic Unit

All the computations of the two halfband filters and the droop-compensation filter are done in 2's complement. The computations basically are the multiplication of a delayed tap by its corresponding coefficients and the additions of the products. For audio applications, since the output frequency is relatively low, it is possible to use a sequence of additions and shifts to perform the multiplications, so we do not need to implement the multiplier in the arithmetic unit. We can do so by carefully selecting our filter coefficients and quantizing them to some CSD (canonic signed-digit) codes [4], which are numbers representable as sums or differences of powers of two[4],[7]. All the coefficients of the two halfband filters and the droop-compensation filter can be represented in 128 adding and shifting sequences. No multiplier is needed. This will greatly reduce the complexity of the decimation filter.

Figure 3.9 shows a possible structure of the arithmetic logic unit. It consists of 4 registers, 6 multiplexors, a shifter, an adder and a rounding logic block. The ROM is not storing the coefficients directly, but the sequence of adding and shifting of the corresponding coefficients. 13 bits of the 22-bit ROM are used to control of the registers, the shifter and the multiplexors, which determine the sequences of adding and shifting. The other 9 bits of the ROM are used to control the read/write operation of the RAM, update the base address of each filter and store the virtual addresses of all delayed taps.



ROM9-21: controlling bits
ROM8: update base address
ROM7: RAM write
ROM0-6: virtual address

Figure 3.9: The 22-bit Arithmetic Logic Unit

The shifter is a logarithmic shifter, which is capable of shifting a number by 7 bit positions, i.e. having 3 controlling pins. The rounding unit, as shown in Figure 3.10, is used to ensure the rounding adder output to the nearest least significant bit. The arithmetic unit receives two input, one from the cascade of comb filters and the other from the RAM output. The $sinc^4$ filter output is the input data stream for the first halfband filter. The RAM output is the delayed tap to be multiplied by its corresponding multiplier. Whenever there is a new $sinc^4$ filter output, the arithmetic unit will

store (or write) that value into the RAM through MUX5. Then the unit will read the two delayed tap values

that have the same corresponding coefficient, to REG2 and REG3. Remember that all the filters are linear phase, so their impulse response taps are symmetrical. The sum of the two values are then stored in REG2 and REG3. The value in REG2 can then be shifted and added back to REG3. By repeating this adding and shifting sequence, the delayed tap value becomes the product of the delayed tap value and its corresponding coefficient. This value is then stored in REG1 temporarily. When the product of the other pair delayed tap values is calculated, it will be added to REG1 and then stored back in REG1. After repeating the similar procedure to every delayed tap in a filter, REG1 will store the output of that filter. The output will then be stored in the RAM.

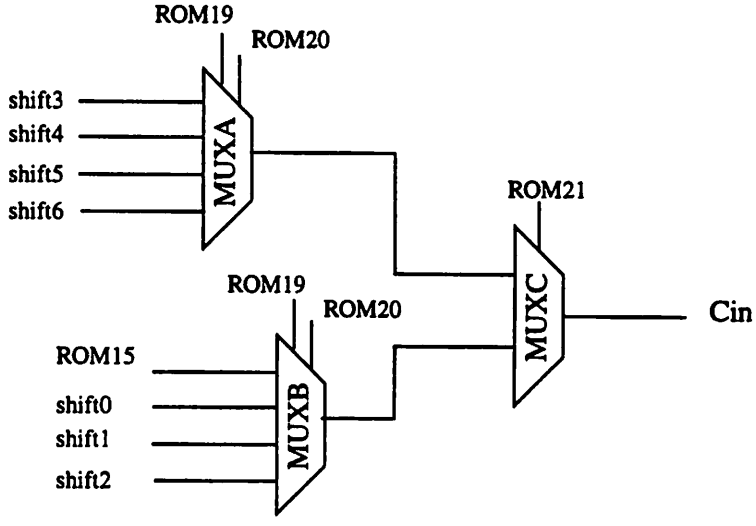


Figure 3.10: Rounding Logic

3.4.4 Nested Multiplication

An additional means of reducing the power dissipation is to employ the Horner's multiplication scheme, or nested multiplication in computation. Partial products, rather than the multipli-

cand, are shifted and added. For example, a normal multiplication of the multiplicand x by the CSD coefficient 3/8 is performed as:

$$\frac{3}{8}x = \frac{1}{4}x + \frac{1}{8}x \tag{3.1}$$

The corresponding nested multiplication is performed as:

$$\frac{3}{8}x = \frac{1}{4}\cdot\left(x + \frac{1}{2}x\right) \tag{3.2}$$

This reduces the number of bits required to be shifted in the shifter. In our example, a shifter capable of shifting an input by 7 bit positions is enough. The nested multiplication also help reduce the quantization error. To prevent any overflow problems during the nested multiplication operations, two more bits are added on top of the 20-bit sinc^4 filter output, resulting in a 22-bit arithmetic logic unit.

The actual code stored in the ROM is listed in Appendix A.

3.5 Complete Structure

The complete structure of the decimation is shown in Figure 3.11. The input frequency (from the modulator output) is 1.024 MHz. The clock frequency is only 2.048 MHz to maintain a 16 kHz output rate.

The actual 12-bit output is taken from bit 8 to bit 19 of the 22-bit ALU output, i.e. bit 0 to bit 7, bit 20 and bit 21 of the ALU are dropped.

Some designs [2] would like to take advantage of the multirate addressing scheme and use the ROM to implement the integrator stage of the Comb filter, which operate at a relatively low data rate. This can help reduce the chip area. This is not done here because of two reasons. Firstly, all the coefficients are nicely packed in the 128-word ROM, leaving no more room for another filter. Only a simple 7-b counter is used to address each of the ROM location. Secondly, with an output rate of 16 kHz and an operating frequency of 2.048 MHz, we have at most 128 cycles to perform one output, which are already used up by the 128-word ROM.

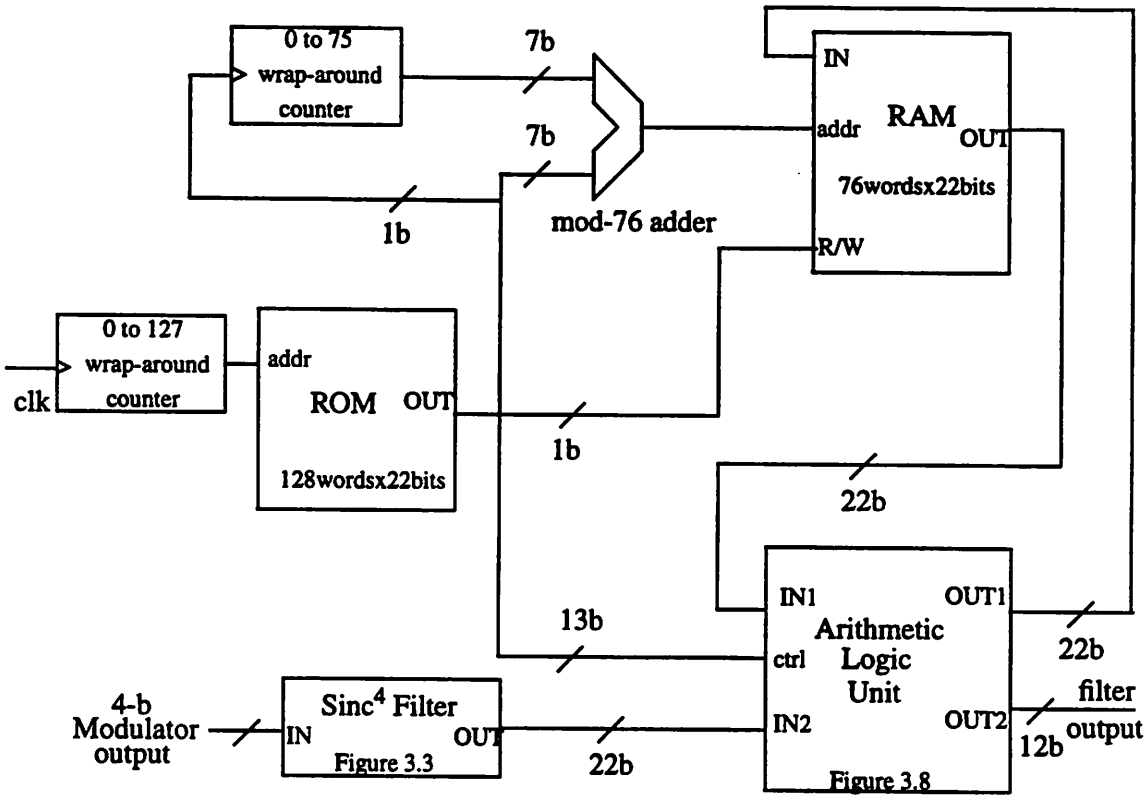


Figure 3.11: Complete Structure

3.6 Power Estimation

The power dissipation of the entire decimation filter is estimated by using PowerPlay. Table 3.2 shows the effective capcitance of each building block of the decimation filter. The data for the shifter is not available yet. Since we are using a logarithmic shifter with maximal shift width of 7 bits to the right [22], there are three levels of pass transistor logic, we estimate the effective capacitance of our shifter to be 5 times the capacitance of a register. For an input data rate of 1.024 MHz and a volatge supply of 1 V, the power estimation is about 270 μW. With a voltage supply of 1.2 V, the power estimation is about 390 μW. Hence the decimation filter should not dissipate more than 0.4 mW.

Table 3.2: Power Dissipation Of The Entire Decimation Filter

Description	Quantity	Effective Capacitance (pF)	Operating Frequency (kHz)	Power Dissipation (μW)
Sinc Integrator:				
20b adder	4	5.4	2048	44.2
20b register	4	1.9	2048	15.6
Sinc Differentiator:				
20b adder	4	5.4	128	2.8
20b register	5	1.9	128	1.2
ROM_addr counter:				
7b counter	1	1.1	2048	2.3
RAM_addr counter:				
7b counter	1	1.1	2048	2.3
7b adder	2	1.9	2048	7.8
7b MUX	1	0.3	2048	0.6
ROM:				
128 word x 22 bit	1	32.6[a]	2048	66.8
RAM:				
76 word x 22 bit	1	52.7	1216	64.1
ALU:				
22b register	3	2.1	2048	12.9
12b register	1	1.1	2048	2.3
22b MUX	6	0.9	2048	11.1
22b shifter (shift by 7 bit positions)	1	10.5[b]	2048	21.5
22b adder	1	5.9	2048	12.1
22b inverter	1	2.1	2048	4.3
			TOTAL	272

a. Estimation from 1/2 of capacitance of a RAM
b. Estimation from 5 times of capacitance of a register

3.7 Time Domain Analysis

Viewdraw program, under the Viewlogic tool sets, is used to enter the schematic diagram of the decimation filter. We then use the Viewsim program, another Viewlogic program, to verify the performance of the decimation filter in the time domain. We first use MATLAB to generate the modulator output data, which are fed into the Viewsim program as the input of the decimation filter. The Viewsim output data are then extracted using UNIX commands and plotted in MATLAB.

Figure 3.12, Figure 3.13 and Figure 3.14 show the filter output for an input of 1 kHz, 7 kHz and 9 kHz sine wave respectively. The actual 12-bit filter output is extracted from the 3rd MSB of the ALU output and onwards. The first two MSBs are excluded becasue they are just additional bits used to prevent overflowing during the computation process. The sampling frequency of the filter is 16 kHz.

In Figure 3.12, a smooth 1 kHz sine wave is obtained at the output. In Figure 3.13, the rugged output waveform represents the sampling version of the 7 kHz. Since the sampling frequency is 16 kHz, there are only 2.29 samples for each cycle of the sine wave, which accounts for the ruggedness of the waveform. In Figure 3.13, the uneven magnitude of the comb filter output is due to printer aliasing while the spike in the output waveform is due to the initial transient effect only. The ripples in the tail is due to the LSB round-off error of the 12-bit output.

From the three figures, we can see that the transition band lies from 7 kHz to 9 kHz and confirms the frequency response of the decimation filter in Figure 2.9.

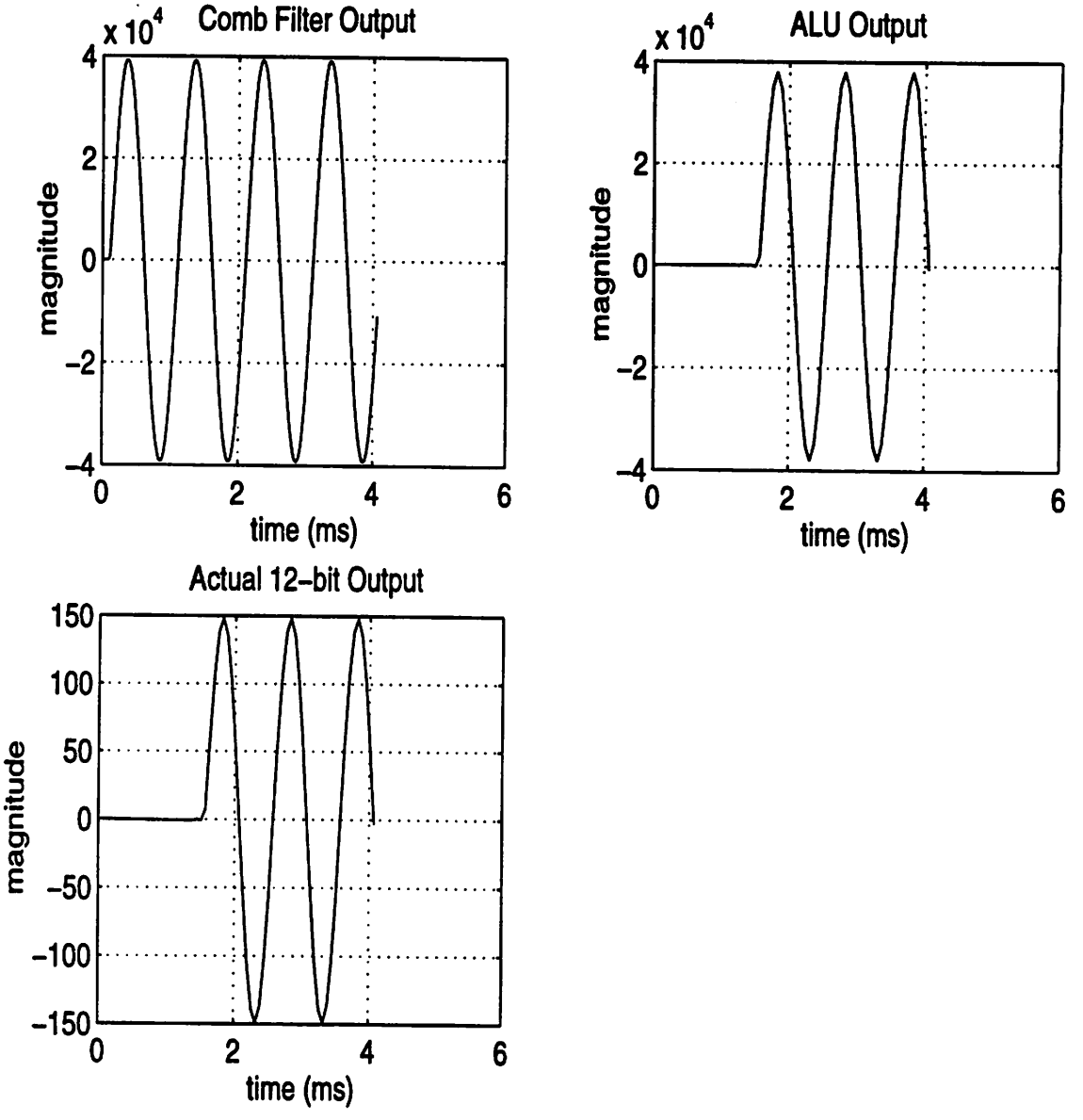


Figure 3.12: Viewsim outputs for a 1 kHz sine wave input (f_{sample} = 16 kHz)

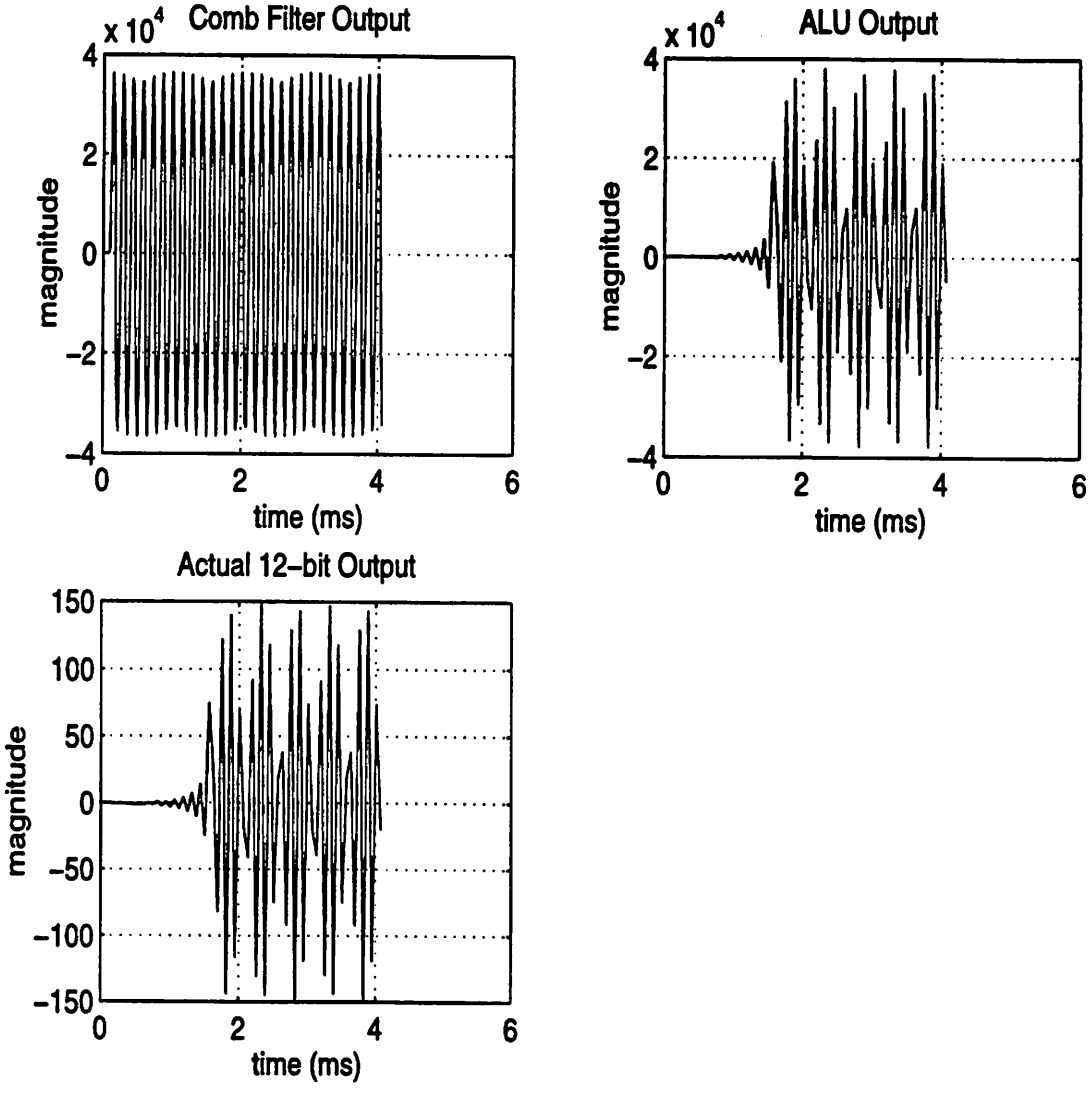


Figure 3.13: Viewsim outputs for a 7 kHz sine wave input (f_{sample} = 16 kHz)

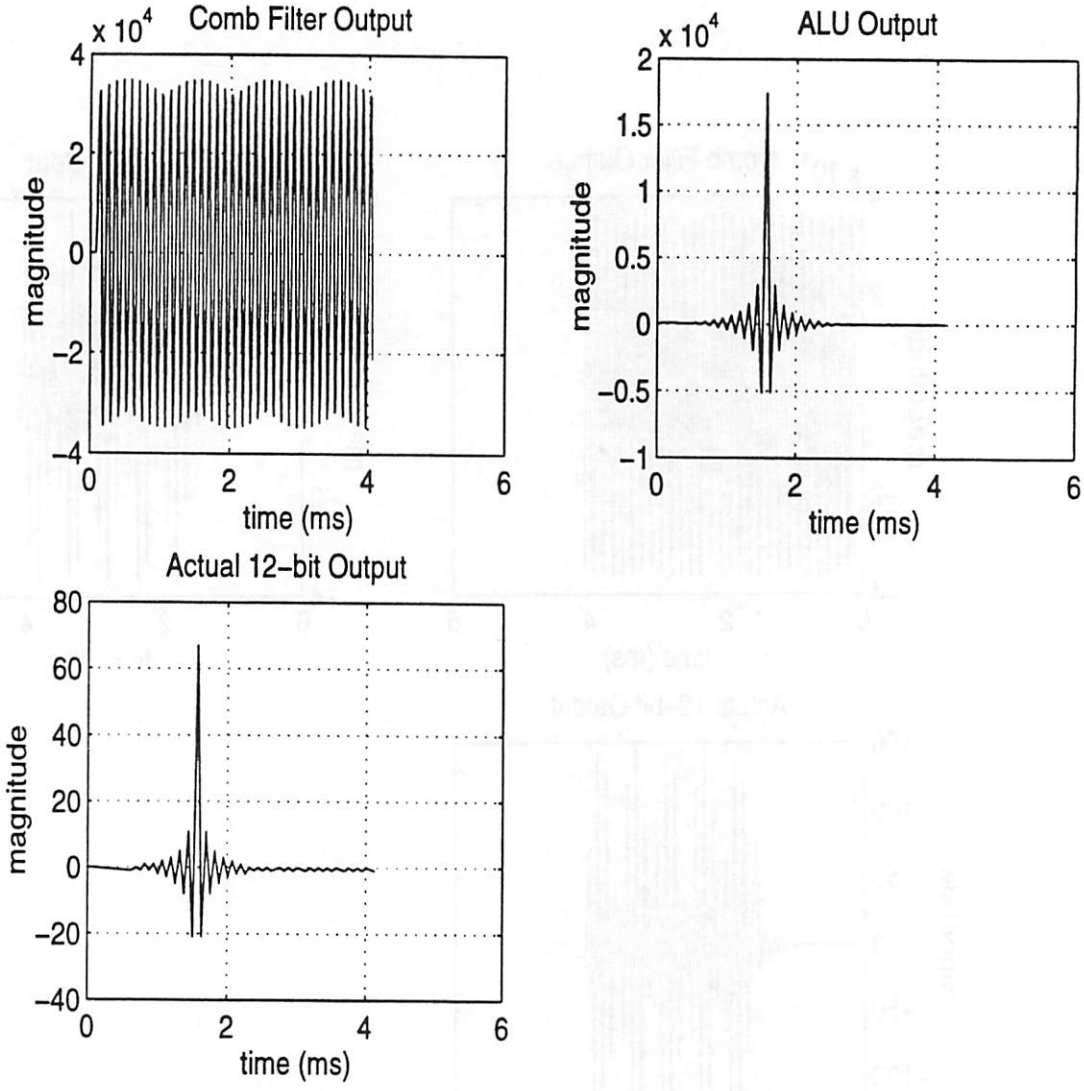


Figure 3.14: Viewsim outputs for a 9 kHz sine wave input (f_{sample} = 16 kHz)

3.8 Layout

Figure 3.14 shows the layout of the decimation filter, generated by the automated layout tool Lager. The area is 32.45 x 10^6 λ^2 , which is 20.7 mm^2 if a 0.8 μm technology is used. It is obviously not optimized.

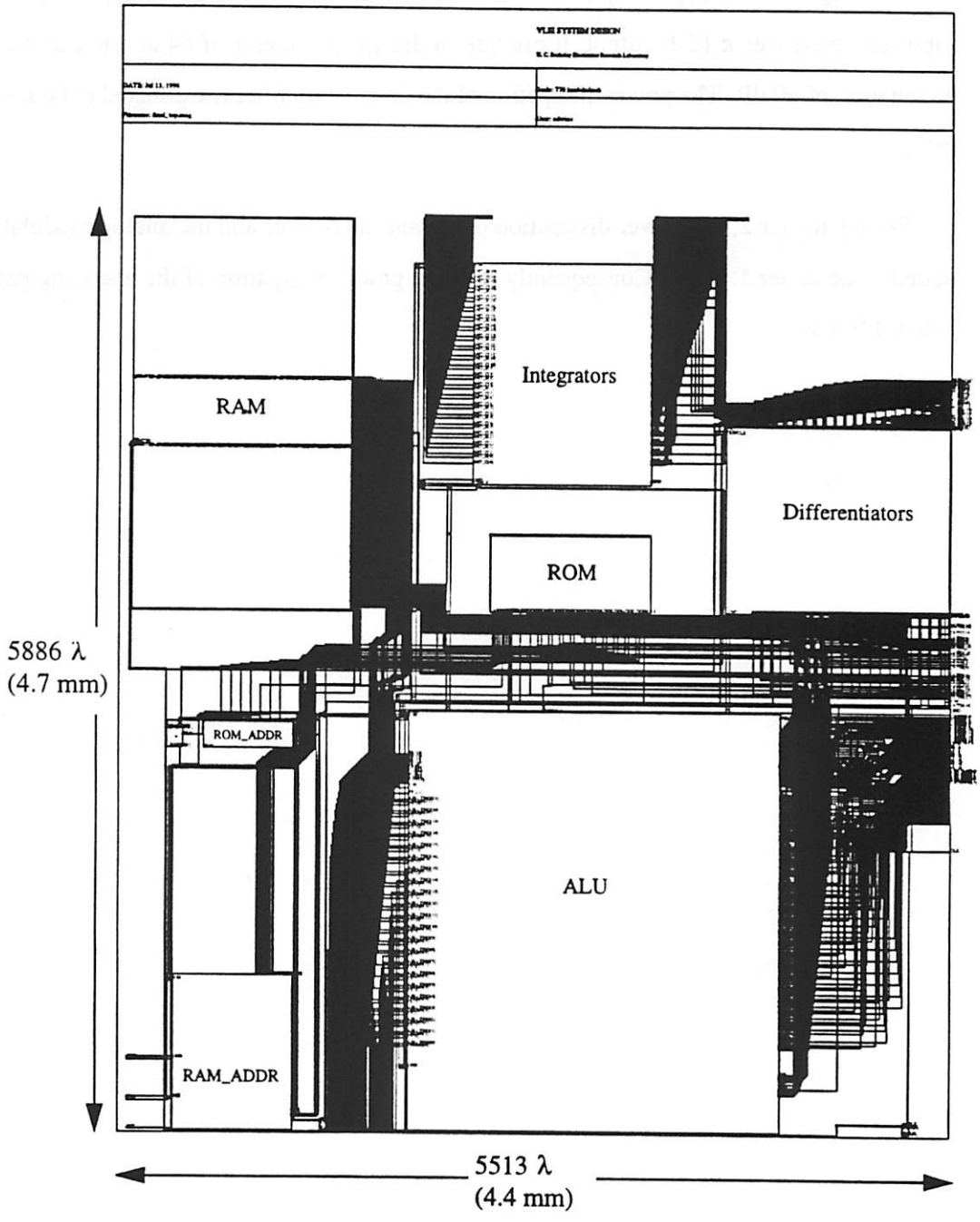


Figure 3.14: Magic Layout Of The Decimation Filter

3.9 Summary

A very area-efficient, low power design of the decimation filter is proposed. It consists of 4 stages. The first stage is a cascade of 4 comb filters, which is then followed by 2 halfband filters. The last stage is the droop-compensation filter. The overall filter is linear-phase. It receives a 4-bit input and produces a 12-b output. It provides a decimation factor of 64 and has an out-of-band attenuation of 60 dB. The power dissipation of the decimation filter is estimated to be less than 0.4 mW.

From Chapter 2, the power dissipation of the anti-alias filter and the analog modulator is estimated to be under 1.1 mW. Consequently the total power dissipation of the encoding path will be below 1.5 mW.

CHAPTER 4

Decoding Path

4.1 Overview

After discussing the decimation filter in previous chapters, which completes the encoding path, we are going to look into some of the design issues in the decoding path in this chapter. We will first investigate the interpolation filter, then the digital $\Sigma\Delta$ demodulator and finally the smoothing filter.

4.2 Interpolation Filter

An interpolation filter is used to raise the input word rate to a frequency well above the Nyquist rate, so that a demodulator can be used to truncate the words and convert the signal to the analog form at the high sample rate. The requirements for attenuating the out-of-band images of the signal spectrum in the interpolation filter are less stringent than those in the decimation filter. Consequently, we can safely use the specifications for the decimation filter here.

The structure of the interpolation filter, as shown in Figure 4.1, is very similar to a decimation filter (Figure 2.4). The order of the building blocks in the interpolation filter is just the reverse of that of the decimation filter, except the cascade of the comb filters are replaced by a zero-order-hold stage. This has two major advantages. Firstly, the comb filters, which are running at a rela-

tively high frequency, burns a considerable portion of power in the decimation filter. They also take up a lot of die area. Replacing the comb filters with a zero-order-hold stage will save both power dissipation and area. Secondly, the droop in the base band due the sinc^4 function (0.68 dB) is much bigger than that in a zero-order-hold stage, which is only 0.17dB. As a result, either the droop-compensation filter can be simpler or the in-band droop can be improved.

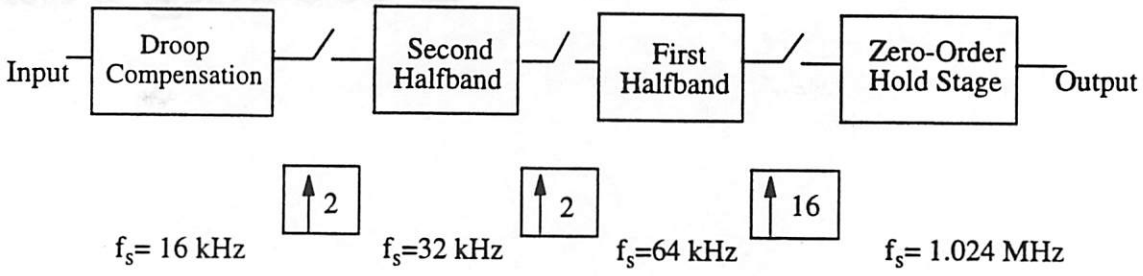


Figure 4.1: The 4-stage Interpolation Filter

Figure 4.2 shows the performance of the interpolation filter.

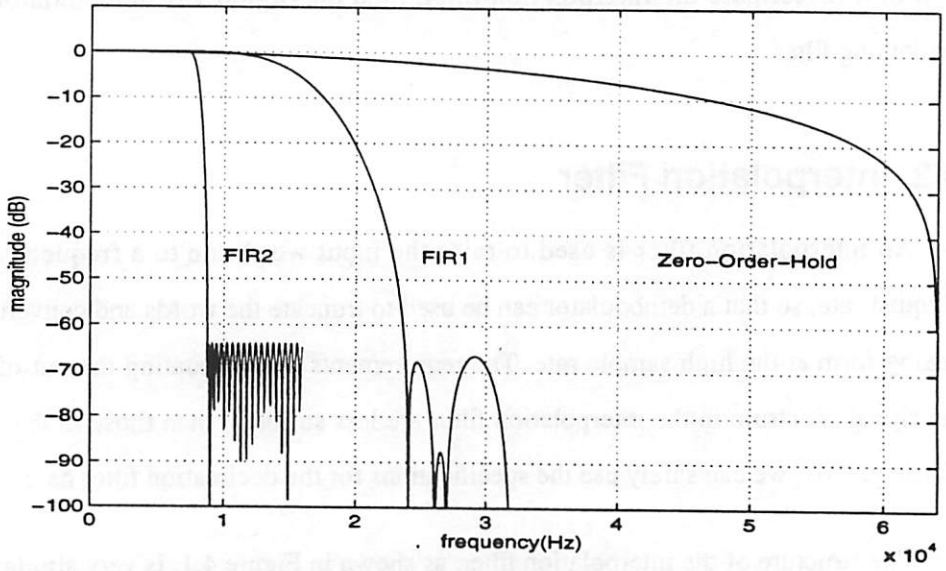


Figure 4.2: Performance Of The Interpolation Filter

4.3 Digital ΣΔ Demodulator

As mentioned in the previous section, the digital ΣΔ demodulator is used to truncate the interpolated words and convert them to the analog form at the high sample rate.

When designing a digital ΣΔ demodulator, there are some design issues that are quite different from those in designing the analog modulator.

Firstly, when we design an analog modulator, we need to consider the quantization noise, thermal noise (kT/C) and op-amp noise. However, only quantization noise affects the performance of a digital demodulator. This means that the specifications for the digital demodulator is less severe.

Secondly, in an analog modulator, gain blocks are implemented by using the ratios of capacitances, which is very flexible. However, in a digital demodulator, a gain of powers of 2 is achieved by shifting the data bits left or right. Therefore, no additional building blocks are needed. For a gain other than powers of 2, either a multiplier or additional shifting and adding sequences are needed. This increases the design complexity drastically.

Thirdly, in an analog modulator, the output of the quantizer is simply fed back to the input summation node. In a digital demodulator, no quantizers are needed. The most significant bit of the output will be like the output of a quantizer. This single bit must be fed back to the multi-bit input summation node in such a way that it represents the most positive or negative number.

After knowing that the specifications for the digital demodulator is less stringent, and the fact that gain blocks other than powers of 2 drastically increase the design complexity, we find that higher order topologies do not help reduce power consumption for low frequency applications, like the audio codec. Using simple structures, like the one in Figure 4.3, is the key solution.

Figure 4.4 shows the performance of the demodulator in Figure 4.3. We can see that using an oversampling ratio (M) of 128 is more than enough for the specifications of the audio codec. The power dissipation will only be in the μW range, which is still insignificant when compared to the power dissipation of the interpolation filter or the decimation filter.

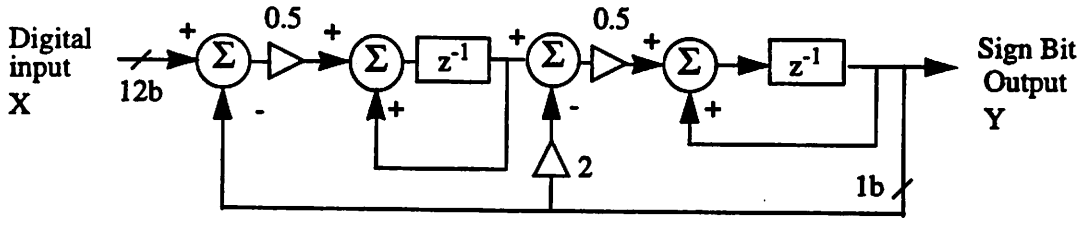


Figure 4.3: 2nd Order Single-bit Sigma Delta Demodulator

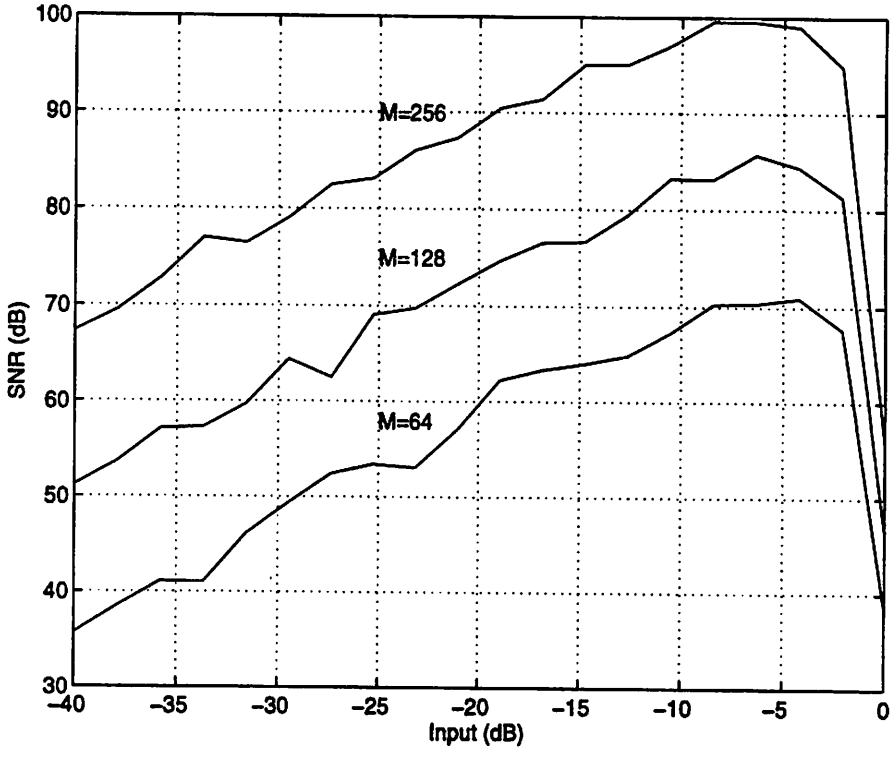


Figure 4.4: Performance Of A Second Order Sigma Delta Modulator

4.4 Smoothing Filter

A smoothing filter is simply a low pass filter, which removes all the high frequency components of the demodulator output. With a 2nd Order $\Sigma\Delta$ demodulator, the smoothing filter must be

at least of the third order. This can be implemented as the anti-alias filter, or by using a cascade of biquad filters. The power dissipation of the smoothing filter should be similar to that of the anti-alias filter.

4.5 Summary

The design issues of the building blocks of the decoding path have been discussed. We can see that the design complexity of the decoding path is less stringent than that in the encoding path. The decoding path is going to take up less area and burn less power than the encoding path From Chapter 3, the encoding path is found to dissipate less than 1.5 mW, so we can conclude that the entire codec will dissipate less than 3 mW.

CHAPTER 5

Conclusion

5.1 Conclusion

This report shows that it is possible to cut down the power dissipation of the audio interface by an order of magnitude, from 40 mW to 3 mW. The approach uses a custom chip to replace the existing audio general-purpose codec.

Several techniques are used to minimize the power dissipation of the decimation filter. A 128 word-ROM is used to achieve the desired specifications. It is also shown that the number of RAM words is not rerstricted to be a power of two. In our case, we use the minimum number of words, 76 words instead of 128 words, to store the delayed tap lines of the two halfband filters and the droop-compensation filter. A slightly more complex structure is then required for the RAM address counter, but this is justified by the reduction in size of the RAM.

However, one drawback of using the multiplier-free Arithemetic & Logic Unit (ALU) is that it requires additional efforts to quantized the filter coefficients to the canonic signed digit (CSD) code. This in turn makes the decimation filter less portable. Whenever there is a change in the design specifications, the entire process of quantization needs to be repeated.

Appendix A

A.1 First Halfband Filter Coefficients, F1

All coefficients are zeros unless stated otherwise.

Table A.3: First Halfband Filter Coefficients

Tap	Coefficients Before Rounding	Canonic Signed Digits	Rounded Coefficients
F1(0), F1(14)	-0.003741	$-2^{-8}+2^{-12}$	-0.003662
F1(2), F1(12)	0.020574	$2^{-6}+2^{-8}+2^{-10}$	0.020508
F1(4), F1(10)	-0.072319	$-2^{-4}-2^{-7}-2^{-9}$	-0.072266
F1(6), F1(8)	0.305486	$2^{-2}+2^{-4}-2^{-7}$	0.305664
F1(7)	0.5	2^{-1}	0.5

A.2 Second Halfband Filter Coefficients, F2

All coefficients are zeros unless stated otherwise.

Table A.4: First Halfband Filter Coefficients

Tap	Coefficients Before Rounding	Canonic Signed Digits	Rounded Coefficients
F2(0), F2(70)	-0.000589	$-2^{-11}-2^{-13}$	-0.000611
F2(2), F2(68)	0.000722	$2^{-10}-2^{-12}$	0.000732
F2(4), F2(66)	-0.001148	$-2^{-10}-2^{-12}+2^{-14}$	-0.001160
F2(6), F2(64)	0.001722	$2^{-9}-2^{-12}$	0.001709
F2(8), F2(62)	-0.002478	$-2^{-9}-2^{-11}$	-0.002441
F2(10), F2(60)	0.003453	$2^{-8}-2^{-11}$	0.003418
F2(12), F2(58)	-0.004691	$-2^{-8}-2^{-10}+2^{-12}-2^{-14}$	-0.004700
F2(14), F2(56)	0.006251	$2^{-7}-2^{-9}+2^{-11}-2^{-13}$	0.006226
F2(16), F2(54)	-0.008203	$-2^{-7}-2^{-11}+2^{-13}$	-0.008179
F2(18), F2(52)	0.010652	$2^{-6}-2^{-8}-2^{-10}-2^{-13}$	0.010620
F2(20), F2(50)	-0.013746	$-2^{-6}+2^{-9}-2^{-14}$	-0.013733
F2(22), F2(48)	0.017726	$2^{-6}+2^{-9}+2^{-13}$	0.017700
F2(24), F2(46)	-0.023004	$-2^{-5}+2^{-7}+2^{-11}-2^{-14}$	-0.023010
F2(26), F2(44)	0.030362	$2^{-5}-2^{-10}+2^{-13}$	0.030396
F2(28), F2(42)	-0.041482	$-2^{-5}-2^{-7}-2^{-9}-2^{-11}$	-0.041504
F2(30), F2(40)	0.060758	$2^{-4}-2^{-9}+2^{-12}$	0.060791
F2(32), F2(38)	-0.104339	$-2^{-3}+2^{-6}+2^{-8}+2^{-10}+2^{-13}$	-0.104370
F2(34), F2(36)	0.317718	$2^{-2}+2^{-4}+2^{-7}-2^{-9}-2^{-11}-2^{-13}$	0.317749
F1(35)	0.5	2^{-1}	0.5

A.3 Droop-Compensation Filter Coefficients, FD

All coefficients are zeros unless stated otherwise.

Table A.5: Droop Filter Coefficients

Tap	Coefficients Before Rounding	Canonic Signed Digits	Rounded Coefficients
FD(0), FD(6)	-0.000874	$-2^{-10}+2^{-13}$	-0.000856
FD(1), FD(5)	0.003739	$2^{-8}+2^{-12}$	0.003664
FD(2), FD(4)	-0.020277	$-2^{-6}-2^{-8}-2^{-10}+2^{-12}$	-0.020264
FD(3)	1	2^{0}	1

A.4 Allocation Of RAM Spaces For The Three Filters

Table A.4 shows how the 76 RAM spaces are allocated to the three filters: first halfband filter,F1(N), second halfband filter, F2(T), droop-compensation filter, FD(Z).

In the first halfband filter, according to the multirate addressing scheme, the most recent 14 tap values are stored into 4 sections, two of which record the odd tap values and the other two record the even tap values. More spaces are allocated to store the even tap values, but it is not very significant in this case.

In the second halfband filter, 53 most recent tap values are stored into 2 sections, one of which records odd tap values and the other records the even tap values. In this filter, we can clearly see that the number of spaces allocated to store the even tap values nearly doubles that for the odd tap values.

The droop-compensation filter is not a halfband filter, so all 7 tap values are stored in one section.

When the first datum, F1(N), to the first halfband filter arrives, it replaces the old tap value fo the first halfband filter, F1(N-11). When the output of the first halfband filter, or the first datum to the second halfband filter, F2(T), is available, it replaces the old tap value of the droop-compensation filter, FD(Z-7). While the calculation in the second halfband filter is going on, the first halfband filter takes up two more data, with F1(N+1) replacing F1(N-14) and F1(N+2) replacing the old tap value of the second halfband filter, F2(N-37). When the output of the second halfband filter is ready, FD(Z), F1(N-9) is replaced. After that the fourth datum to the first halfband filter, F1(N+3), arrives and replaces F1(N-16). A second F1 output, F2(T+1), will take the place of F2(T-73). Finally, FD will generate the filter output.

For every output datum FD generates, F2 receives two more data while F1 recieves four, which corresponds to a decimation factor of 4.

Table A.6: Allocation Of RAM Spaces For The Three Filters

Tap line	RAM space	Access Order	Tap line	RAM space	Acces Order
F1:			F2:		
F1(N-9)	0	5th: FD(Z)	F2(T-62)	38	
F1(N-5)	1		F2(T-60)	39	
F1(N-1)	2		F2(T-58)	40	
F1(N-16)	3	6th: F1(N+3)	F2(T-56)	41	
F1(N-12)	4		F2(T-54)	42	
F1(N-8)	5		F2(T-52)	43	
F1(N-4)	6		F2(T-50)	44	
F1(N-11)	7	1st: F1(N)	F2(T-48)	45	
F1(N-7)	8		F2(T-46)	46	
F1(N-3)	9		F2(T-44)	47	
F1(N-14)	10	3rd: F1(N+1)	F2(T-42)	48	
F1(N-10)	11		F2(T-40)	49	
F1(N-6)	12		F2(T-38)	50	
F1(N-2)	13		F2(T-36)	51	
F2:			F2(T-34)	52	
F2(T-37)	14	4th: F1(N+2)	F2(T-32)	53	
F2(T-35)	15		F2(T-30)	54	
F2(T-33)	16		F2(T-28)	55	
F2(T-31)	17		F2(T-26)	56	
F2(T-29)	18		F2(T-24)	57	
F2(T-27)	19		F2(T-22)	58	
F2(T-25)	20		F2(T-20)	59	
F2(T-23)	21		F2(T-18)	60	
F2(T-21)	22		F2(T-16)	61	

Table A.6: Allocation Of RAM Spaces For The Three Filters

Tap line	RAM space	Access Order	Tap line	RAM space	Acces Order
F2(T-19)	23		F2(T-14)	62	
F2(T-17)	24		F2(T-12)	63	
F2(T-15)	25		F2(T-10)	64	
F2(T-13)	26		F2(T-8)	65	
F2(T-11)	27		F2(T-6)	66	
F2(T-9)	28		F2(T-4)	67	
F2(T-7)	29		F2(T-2)	68	
F2(T-5)	30		FD:		
F2(T-3)	31		FD(Z-7)	69	2nd: F2(T)
F2(T-1)	32		FD(Z-6)	70	
F2(T-72)	33	7th: F2(T+1)	FD(Z-5)	71	
F2(T-70)	34		FD(Z-4)	72	
F2(T-68)	35		FD(Z-3)	73	
F2(T-66)	36		FD(Z-2)	74	
F2(T-64)	37		FD(Z-1)	75	

A.5 ROM Coefficients

The following list shows the 128 operations and the corresponding 22b code. "reg1", "reg2", "reg3" refer to the three registers in the ALU, Figure 3.9. "#7" refers to the RAM position #7.

ROM operations	22b Code
0. store #7 (F1(N))	000 11 10 11 11 10 00 0000111
1. reg2 = read #7	000 10 10 11 01 10 01 0000111
2. reg3, reg1 = reg3 + reg2 reg2 = read #4	000 10 10 00 01 00 01 0000100
3. reg1 = reg3 - (reg1 >> 4) reg3 = read #13	100 10 11 01 11 01 01 0001101
4. reg3, reg2 = reg3 + reg2	000 10 10 00 00 10 01 0001101
5. reg2 = reg3 + (reg2 >> 2)	010 10 10 10 00 10 01 0001101
6. reg3 = reg3 + (reg2 >> 2) reg2 = read #6	010 10 10 00 01 10 01 0000110
7. reg1 = reg3 - (reg1 >> 2) reg3 = read #11	010 10 11 01 11 01 01 0001011
8. reg3, reg2 = reg3 + reg2	000 10 10 00 00 10 01 0001011
9. reg2 = reg3 + (reg2 >> 2)	010 10 10 10 00 10 01 0001011
10. reg3 = reg3 + (reg2 >> 3) reg2 = read #5	011 10 10 00 01 10 01 0000101
11. reg1 = reg3 - (reg1 >> 2) reg3 = read #12	010 10 11 01 11 01 01 0001100
12. reg3, reg2 = reg3 + reg2	000 10 10 00 00 10 01 0001100
13. reg2 = reg3 - (reg2 >> 3)	011 10 11 10 00 10 01 0001100
14. reg2 = reg3 - (reg2 >> 3)	011 10 11 10 00 10 01 0001100
15. reg3 = reg3 + (reg2 >> 2)	010 10 10 00 10 10 01 0001100
16. reg1 = reg3 - (reg1 >> 2) reg3 = read #8	010 10 11 01 10 01 01 0001000
17. reg1 = reg3 + (reg1 >> 1) reg2 = read #34	001 10 10 11 01 01 01 0100010
18. reg1, reg3 = reg1 >> 1	001 00 10 00 11 01 01 0100010
19. store reg1 (F2(T)) into #69	000 10 10 10 11 10 00 1000101
20. reg3, reg1 = reg3 + reg2 reg2 = read #35	000 10 10 00 01 00 01 0100011
21. reg1 = reg3 + (reg1 >> 2) reg3 = read #68	010 10 10 01 11 01 01 1000100
22. reg3, reg2 = reg3 + reg2	000 10 10 00 00 10 01 1000100
23. reg3 = reg3 - (reg2 >> 2) reg2 = read #36	010 10 11 00 01 10 01 0100100
24. reg1 = reg3 - (reg1 >> 1) reg3 = read #67	001 10 11 01 11 01 01 1000011
25. reg3, reg2 = reg3 + reg2	000 10 10 00 00 10 01 1000011
26. reg2 = reg3 - (reg2 >> 2)	010 10 11 10 00 10 01 1000011
27. reg3 = reg3 + (reg2 >> 2) reg2 = read #37	010 10 10 00 01 10 01 0100101
28. reg1 = reg3 - reg1 reg3 = read #66	000 10 11 01 11 01 01 1000010
29. reg3, reg2 = reg3 + reg2	000 10 10 00 00 10 01 1000010
30. reg3 = reg3 - (reg2 >> 3) reg2 = read #38	011 10 11 00 01 10 01 0100110
31. reg1 = reg3 - (reg1 >> 1) r eg3 = read #65	001 10 11 01 11 01 01 1000001
32. reg3, reg2 = reg3 + reg2 store #10 (F1(N+1))	000 11 10 00 00 10 00 0001010
33. reg3 = reg3 + (reg2 >> 2)	010 10 10 00 01 10 01 0100111

reg2 = read #39	
34. reg1 = reg3 - reg1 reg3 = read #64	000 10 11 01 11 01 01 1000000
35. reg3, reg2 = reg3 + reg2	000 10 10 00 00 10 01 1000000
36. reg3 = reg3 - (reg2 >> 3)	011 10 11 00 01 10 01 0101000
reg2 = read #40	
37. reg1 = reg3 - (reg1 >> 1) reg3 = read #63	001 10 11 01 11 01 01 0111111
38. reg3, reg2 = reg3 + reg2	000 10 10 00 00 10 01 0111111
39. reg2 = reg3 - (reg2 >> 2)	010 10 11 10 00 10 01 0111111
40. reg3 = reg3 + (reg2 >> 2)	010 10 10 00 01 10 01 0101001
reg2 = read #41	
41. reg1 = reg3 - reg1 reg3 = read #62	000 10 11 01 11 01 01 0111110
42. reg3, reg2 = reg3 + reg2	000 10 10 00 00 10 01 0111110
43. reg2 = reg3 - (reg2 >> 2)	010 10 11 10 00 10 01 0111110
44. reg2 = reg3 - (reg2 >> 2)	010 10 11 10 00 10 01 0111110
45. reg3 = reg3 - (reg2 >> 2)	010 10 11 00 01 10 01 0101010
reg2 = read #42	
46. reg1 = reg3 - (reg1 >> 1) reg3 = read #61	001 10 11 01 10 01 01 0111101
47. reg3, reg2 = reg3 + reg2	000 10 10 00 00 10 01 0111101
48. reg2 = reg3 - (reg2 >> 2)	010 10 11 10 00 10 01 0111101
49. reg3 = reg3 + (reg2 >> 4)	100 10 10 00 01 10 01 0101011
reg2 = read #43	
50. reg1 = reg3 - reg1 reg3 = read #60	000 10 11 01 11 01 01 0111100
51. reg3, reg2 = reg3 + reg2	000 10 10 00 00 10 01 0111100
52. reg2 = reg3 + (reg2 >> 3)	011 10 10 10 00 10 01 0111100
53. reg2 = reg3 + (reg2 >> 2)	010 10 10 10 00 10 01 0111100
54. reg3 = reg3 - (reg2 >> 2)	010 10 11 00 01 10 01 0101100
reg2 = read #44	
55. reg1 = reg3 - (reg1 >> 1) reg3 = read #59	001 10 11 01 11 01 01 0111011
56. reg3, reg2 = reg3 + reg2	000 10 10 00 00 10 01 0111011
57. reg2 = reg3 - (reg2 >> 5)	101 10 11 10 00 10 01 0111011
58. reg3 = reg3 - (reg2 >> 3)	011 10 11 00 01 10 01 0101101
reg2 = read #45	
59. reg1 = reg3 - reg1 reg3 = read #58	000 10 11 01 11 01 01 0111010
60. reg3, reg2 = reg3 + reg2	000 10 10 00 00 10 01 0111010
61. reg2 = reg3 + (reg2 >> 4)	100 10 10 10 00 10 01 0111010
62. reg3 = reg3 + (reg2 >> 3)	011 10 10 00 01 10 01 0101110
reg2 = read #46	
63. reg1 = reg3 - reg1 reg3 = read #57	000 10 11 01 11 01 01 0111001
64. store #14 (F1(N+2))	000 11 10 11 11 11 00 0001110
65. reg3, reg2 = reg3 + reg2	000 10 10 00 00 10 01 0001110
66. reg2 = reg3 - (reg2 >> 3)	011 10 11 10 00 10 01 0001110
67. reg2 = reg3 + (reg2 >> 4)	100 10 10 10 00 10 01 0001110
68. reg3 = reg3 - (reg2 >> 2)	010 10 11 00 01 10 01 0101111
reg2 = read #47	
69. reg1 = reg3 - (reg1 >> 1) reg3 = read #56	001 10 11 01 11 01 01 0111000
70. reg3, reg2 = reg3 + reg2	000 10 10 00 00 10 01 0111000
71. reg2 = reg3 - (reg2 >> 3)	011 10 11 10 00 10 01 0111000
72. reg3 = reg3 - (reg2 >> 5)	101 10 11 00 01 10 01 0110000
reg2 = read #48	
73. reg1 = reg3 - reg1 reg3 = read #55	000 10 11 01 11 01 01 0110111
74. reg3, reg2 = reg3 + reg2	000 10 10 00 00 10 01 0110111
75. reg2 = reg3 + (reg2 >> 2)	010 10 10 10 00 10 01 0110111
76. reg2 = reg3 + (reg2 >> 2)	010 10 10 10 00 10 01 0110111
77. reg3 = reg3 + (reg2 >> 2)	010 10 10 00 01 10 01 0110001
reg2 = read #49	
78. reg1 = reg3 - reg1 reg3 = read #54	000 10 11 01 11 01 01 0110110

79. reg3, reg2 = reg3 + reg2	000 10 10 00 00 10 01 0110110
80. reg2 = reg3 - (reg2 >> 3)	011 10 11 10 00 10 01 0110110
81. reg3 = reg3 - (reg2 >> 5)	101 10 11 00 01 10 01 0110010
reg2 = read #50	
82. reg1 = reg3 - (reg1 >> 1)	001 10 11 01 11 01 01 0110101
reg3 = read #53	
83. reg3, reg2 = reg3 + reg2	000 10 10 00 00 10 01 0110101
84. reg2 = reg3 + (reg2 >> 3)	011 10 10 10 00 10 01 0110101
85. reg2 = reg3 + (reg2 >> 2)	010 10 10 10 00 10 01 0110101
86. reg2 = reg3 + (reg2 >> 2)	010 10 10 10 00 10 01 0110101
87. reg3 = reg3 - (reg2 >> 3)	011 10 11 00 01 10 01 0110011
reg2 = read #51	
88. reg1 = reg3 - (reg1 >> 1)	001 10 11 01 11 01 01 0110100
reg3 = read #52	
89. reg3, reg2 = reg3 + reg2	000 10 10 00 00 10 01 0110011
90. reg2 = reg3 + (reg2 >> 2)	010 10 10 10 00 10 01 0110011
91. reg2 = reg3 + (reg2 >> 2)	010 10 10 10 00 10 01 0110011
92. reg2 = reg3 - (reg2 >> 2)	010 10 11 10 00 10 01 0110011
93. reg2 = reg3 + (reg2 >> 3)	011 10 10 10 00 10 01 0110011
94. reg3 = reg3 + (reg2 >> 2)	010 10 10 00 01 10 01 0000100
reg2 = read #4	
95. reg1 = reg3 - (reg1 >> 1)	001 10 11 01 10 01 01 0001111
reg3 = read #15	
96. reg1 = reg3 + (reg1 >> 1)	001 11 10 11 10 01 00 0000011
store #3 (Sinc #4)	
97. reg1 = reg1 >> 1	001 00 10 01 11 01 01 0001110
reg3 = read #14	

---

98. store reg1(FD(Z)) into #0	000 10 10 11 11 10 00 0000000
99. reg3, reg1 = reg3 + reg2	000 10 10 00 01 00 01 0000111
reg2 = read #7	
00. reg1 = reg3 - (reg1 >> 4)	100 10 11 01 11 01 01 0001011
reg3 = read #11	
01. reg3, reg2 = reg3 + reg2	000 10 10 00 00 10 01 0001011
02. reg2 = reg3 + (reg2 >> 2)	010 10 10 10 00 10 01 0001011
03. reg3 = reg3 + (reg2 >> 2)	010 10 10 00 01 10 01 0000101
reg2 = read #5	
04. reg1 = reg3 - (reg1 >> 2)	010 10 11 01 11 01 01 0001101
reg3 = read #13	
05. reg3, reg2 = reg3 + reg2	000 10 10 00 00 10 01 0001101
06. reg2 = reg3 + (reg2 >> 2)	010 10 10 10 00 10 01 0001101
07. reg3 = reg3 + (reg2 >> 3)	011 10 10 00 01 10 01 0000110
reg2 = read #6	
08. reg1 = reg3 - (reg1 >> 2)	010 10 11 01 11 01 01 0001100
reg3 = read #12	
09. reg3, reg2 = reg3 + reg2	000 10 10 00 00 10 01 0001100
10. reg2 = reg3 - (reg2 >> 3)	011 10 11 10 00 10 01 0001100
11. reg2 = reg3 - (reg2 >> 3)	011 10 11 10 00 10 01 0001100
12. reg3 = reg3 + (reg2 >> 2)	010 10 10 00 10 10 01 0001100
13. reg1 = reg3 - (reg1 >> 2)	010 10 11 01 10 01 01 0000001
reg3 = read #1	
14. reg1 = reg3 + (reg1 >> 1)	001 10 10 11 01 01 01 1000110
reg2 = read #70	
15. reg1 = reg1 >> 1	001 00 10 01 11 01 01 0000000
r eg3 = read #0	

---

16. store reg1(F2(T+1)) into #33	000 10 10 10 11 10 00 0100001
17. reg3, reg1 = reg3 + reg2	000 10 10 00 01 00 01 1000111
reg2 = read #71	
18. reg1 = reg3 - (reg1 >> 3)	011 10 11 01 11 01 01 1001011
reg3 = read #75	
19. reg3, reg2 = reg3 + reg2	000 10 10 00 00 10 01 1001011
20. reg3 = reg3 - (reg2 >> 4)	100 10 11 00 01 10 01 1001000
reg2 = read #72	
21. reg1 = reg3 - (reg1 >> 2)	010 10 11 01 11 01 01 1001010
reg3 = read #74	
22. reg3, reg2 = reg3 + reg2	000 10 10 00 00 10 01 1001010
23. reg2 = reg3 - (reg2 >> 2)	010 10 11 10 00 10 01 1001010

24. reg2 = reg3 + (reg2 >> 2)	010 10 10 10 00 10 01 1001010
25. reg3 = reg3 + (reg2 >> 2)	010 10 10 00 10 10 01 1001010
26. reg1 = reg3 - (reg1 >> 2)	010 10 11 01 10 01 01 1001001
reg3 = read #73	
27. reg1 = reg3 - (reg1 >> 6)	110 10 01 01 11 01 11 0001010
reg3 = read #10	

References

[1] B. P. Brandt and B. A. Wooley, "A Low-Power, Area-Efficient Digital Filter for Decimation and Interpolation," IEEE Journal of Solid State Circuits, vol. 29, no. 6, June 1994, 1995, pp. 679-687.

[2] B. P. Brant, "Oversampled Analog-To-Digital Conversion," PhD thesis, Stanford University, August 1991.

[3] A. Oppenheim and R. Schafer, Digital Signal Processing, Prentice Hall, 1975.

[4] H. Samueli, "An Improved Search Algorithm for the Design of Multiplierless FIR Filters with Powers-of-Two Coefficients," IEEE Trans. on Circuits and Systems, vol. 36, no. 7, July 1989.

[5] J. C. Candy, "Decimation for Sigma Delta Modulation," IEEE Trans. on Communications, vol. COM-34, no. 1, January 1986.

[6] D. J. Goodman and M. J. Carey, "Nine Digital Filters for Decimation and Interpolation," IEEE Trans. on Acoustics, Speech, and Signal Processing, vol. ASSP-25, no. 2, April 1977.

[7] K. Hwang, Computer Arithmetic, Principles, Architecture, and Design. New York: Wiley, 1979.

[8] P. P. Vaidyanathan and T. Q. Nguyen, "A "trick" for the Design of FIR Half-Band Filters," IEEE Trans. on Circuits and Systems, vol. CAS-34, no. 3, March 1987.

[9] S. Chu and C. S. Burrus, "Multirate Filter Designs Using Comb Filters," IEEE Trans. on Circuits and Systerms, vol. CAS-31, no. 11, Novermber 1984.

[10] B. Leung, "Design of Low Power Oversampled Sigma-Delta Modulators," Technical Report, Electrical and Computer Engineering Department, U. of Waterloo, June 1995.

[11] J. C. Candy and G. C. Temes, "Oversampling Methods for A/D and D/A Conversion," Oversampling Delta-Sigma Data Converters, IEEE Press, 1991.

[12] R. W. Brodersen, Anatomy of a Silicon Compiler, Kluwer Academic Publishers, 1992.

[13] Q. Zhao and Y. Tadokoro, "A Simple Design of FIR Filters with Powers-of-Two Coefficients," IEEE Trans. on Circuits and Systerms, vol. 35, no. 5, May 1988.

[14] A. B. Carlson, Communication Systems, McGraw-Hill, 1988.

[15] J. M. Rabaey, S. P. Pope, R. W. Brodersen, "An Integrated Automated Layout Generation System for DSP Circuits," IEEE Trans. on Computer-Aided Design, vol. CAD-4, no. 3, July 1985.

[16] R. W. Brodersen, A. Chandrakasan, S. Sheng, "Design Techniques for Portable Systems," ISSCC Dig. Tech. Papers, pp. 168-169, February 1993.

[17] E. B. Hogenauer, "An Economical Class of Digital Filters for Decimation and Interpolation," IEEE Trans. on Acoustics, Speech,, and Signal Processing, vol. ASSP-29, no. 2, April 1981.

[18] M. G. Bellanger, J. L. Daguet and G. P. Lepagnol, "Interpolation, Extrpolation, and Reduction of Computation Speed in Digital Filters," IEEE Trans. on Acoustics, Speech, and Signal Pro-

cessing, vol. ASSP-22, no. 4, August 1974.

[19] R. Gregorian and G. C. Temes, Analog MOS Integrated Circuits For Signal Processing, John Wiley & Sons, Inc, 1986.

[20] G. S. Moschytz and P. Horn, Active Filter Design Handbook, John Wiley & Sons, Inc, 1981.

[21] R. Dongen, V. Rikkink, M. Degrauwe, "A 1.5 V class AB CMOS Biffer Amplifier for Driving Low-Resistance Loads," ISSCC 1995 Digest, pp. 48-49, February 1995.

[22] J. M. Rabaey, Digital Integrated Circuits, Prentice Hall, 1996.

[23] R. E. Crochiere and L. R. Rabiner, Multirate Digital Signal Processing, Prentice Hall, 1983.

[24] L. R. Rabiner and G. Gold, Theory And Application Of Digital Signal Processing, Prentice Hall, 1975.

[25] URL: http://infopad.EECS.Berkeley.EDU

[26] B. Richards and R. Brodersen, "InfoPad: The design of a portable multmedia terminal," 2nd International Workshop on Mobile Multi-Media Communications Proceedings, April 1995, p.S2/1/1-6.

[27] Motorola Telecommunications Device Data, pp. 2.627 - 2.640.

[28] A. Chandrakasan, A. Burstein and R. Brodersen, "A Low Power Chipset for Portable Multimedia Applications," ISSCC 1994 Digest, pp. 82-83, February 1994.

[29] I. Donnell and A. Chandrakasan, ARAP Report, October 1994 - April 1995, URL: /http://infopad.EECS.Berkeley.EDU/infopad-ftp/reports/A.Multimedia.Communication.System.Providing.Wireless.Access/