20 MHz Bandwidth Decimation Filter for a fourth order 1 GS/s $\Delta\Sigma$ Modulator

A Project Report

submitted by

JISHNU C

in partial fulfilment of the requirements
for the award of the dual degree of

BACHELOR OF TECHNOLOGY
and
MASTER OF TECHNOLOGY

DEPARTMENT OF ELECTRICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY, MADRAS.

May 2010

THESIS CERTIFICATE

This is to certify that the thesis titled **20 MHz Bandwidth Decimation Filter for a fourth order 1 GS/s $\Delta\Sigma$ Modulator**, submitted by **Jishnu C**, to the Indian Institute of Technology Madras, for the award of the degree of **Bachelor of Technology** and **Master of Technology**, is a bona fide record of the work done by him under my supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Dr. Nagendra Krishnapura
Advisor,
Assistant Professor,
Dept. of Electrical Engineering,
IIT-Madras, 600 036

Place: Chennai
Date: 25 May 2010

ACKNOWLEDGEMENTS

First of all, I would like to express deepest sense of gratitude to my advisor Dr. Nagendra Krishnapura whose energy, enthusiasm and constant support inspired me from the beginning till the end of the project. I am indebted to him for his ideas, valuable guidance and the encouragement throughout.

I would like to thank various faculty members of IIT Madras from whom I have benefited as a student. I especially thank Dr. Shanthi Pavan whose stimulating and inspiring lectures motivated me to work in Analog and Mixed-Signal design.

I express my sincere gratitude to Shankar whose support had proved invaluable during the course of the project. I would not have been able to finish in time if not for his help with design tools and the concepts of decimation filter. I would like to thank Vikas for his help with Sigma-Delta modulation and Current Mode Logic. I would also like to thank Abhishek, lab supporting staff Mrs. Sumathy and Mrs. Janaki for their help and support.

Last but not least, I would like to thank Radha, Lokesh, Ankur, Sameer, Ankesh, Sandeep and Timir, my friends in the VLSI lab for their support and the friendly atmosphere in lab. I would also like to thank my friends Arvind, Karthikeyan and Rohit who helped me let go of the stress of the working hours by joining me in the tea shop.

Finally, I dedicate this thesis to my parents and sister who make my any little success meaningful.

ABSTRACT

This project involves the design of a Decimation filter for a High speed $\Delta\Sigma$ Analog to Digital converter with a sampling frequency of 1 GHz. Decimation by a factor 25 is implemented in a cascade of two stages, each stage decimating by a factor 5. Polyphase decomposition reduces the maximum operating frequency to 200 MHz so that the entire design can be implemented with 0.18 μm standard cell library. Appropriate grouping of partial products reduces the number of explicit additions/mutliplications and minimizes the power consumption. The design is implemented in 0.18 μm CMOS process from UMC. It occupies an area of $\approx 758 \times 650$. The design consumes a total power of about 18.887 mW (mostly dynamic power) from a 1.8 V supply. The simulated SNR for the Decimation filter is 93.0482 dB when fed from the output of an ideal $\Delta\Sigma$ modulator.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS i

ABSTRACT ii

LIST OF TABLES vi

LIST OF FIGURES ix

1 Introduction 1

1.1 Motivation 1

1.2 Organization 2

2 Decimation Filter Design and Architecture 4

2.1 Decimation filter fundamentals 4

2.1.1 Time domain characteristics 4

2.1.2 Frequency domain characteristics 6

2.1.3 Cascade equivalence of decimation filters 7

2.2 $\Delta\Sigma$ modulator specifications 9

2.3 Multi-stage decimation filter 12

2.3.1 Single stage implementation 13

2.3.2 Two stage implementation 15

2.4 Sinc filter characteristics 16

2.5 FIR filter characteristics 21

2.5.1 Determining the FIR filter coefficients 21

2.5.2 Sinc and FIR filter frequency responses 25

2.5.3 64^{th} order FIR filter 26

2.5.4 Finite precision of the FIR filter coefficients 27

2.6 Architecture 28

2.6.1 Hogenauer structure for sinc filter 30

2.7 Decimation filter using polyphase decomposition 32

2.7.1 Polyphase decomposition of sinc filter 33

2.7.2 Polyphase decomposition of the FIR filter 34

3 Implementation of the Sinc Filter 38

3.1 Implementation of decomposed filters 38

3.1.1 $E_0(z)$ and $E_1(z)$ 39

3.1.2 $E_2(z)$ and $E_4(z)$ 39

3.1.3 $E_3(z)$ 42

3.2 Partial product generation 43

3.3 Wallace tree addition 43

3.3.1 Reduction of the number of partial products 47

3.3.2 Minimizing the number of adders 48

3.4 Vector Merging Adder (VMA) 50

3.4.1 Generate and Propagate signals 51

3.4.2 Dot operation 51

3.4.3 Sum Generation 54

3.4.4 14-bit Kogge-Stone adder 54

3.5 The complete sinc filter 55

4 FIR Filter Implementation 57

4.1 Implementation of decomposed filters 57

4.1.1 Implementation of $G_4(z)$ 59

4.1.2 Implementation of $G_0(z)$ and $G_3(z)$ 61

4.1.3 Implementation of $G_1(z)$ and $G_2(z)$ 61

4.1.4 Optimizing the wallace tree 67

4.2 The complete FIR filter 67

5 Decimators and Clock Dividers 71

5.1 Decimator design 71

5.1.1 Basic decimator circuit 71

5.1.2 Low power decimator circuit 71

5.2 Clock dividers 73

5.2.1 Counter as a state machine 75

5.2.2 Ring counter and Johnson's counter 75

6 Decimation Filter with Minimal Passband Droop 78

6.1 Five tap fourth order sinc filter 78

6.2 FIR filter for the second stage 79

6.2.1 Finite precision of the FIR filter coefficients 81

6.2.2 Polyphase decomposition of the FIR filter 83

6.3 Equalizer 87

6.4 Conclusion 89

7 Results and Conclusion 90

7.1 Simulation and results 90

7.2 Conclusion 92

A Decimation Filter Implementation using CML in First Stage 94

A.1 CIC implementation using Carry Save Accumulators 94

A.1.1 CSA for the first accumulator in CIC 94

A.1.2 CSA for the later stage accumulators in CIC 97

A.2 Implementation in CML 101

A.2.1 Adders 101

A.2.2 Registers and bias generation 103

A.3 Results and conclusion 103

A Asynchronous reset 106

LIST OF TABLES

2.1 Assumed LPF parameters for the single stage implementation . 13

2.2 Variation of maximum droop and worst alias attenuation with the value of r 18

2.3 Parameters for the FIR filter $H_{pc}(z)$ 22

2.4 Finite precision FIR filter coefficients 29

2.5 $G_k(z)$ definitions 36

3.1 Partial products of the sinc filter 44

4.1 Filter coefficients and partial products 58

4.2 Filter coefficients of $G_4(z)$ and partial products 60

4.3 Filter coefficients of $G_3(z), G_0(z)$ and partial products 64

4.4 Filter coefficients of $G_2(z)$,$G_1(z)$ and partial products 66

6.1 Parameters for the FIR filter $\hat{H}_{pc}(z)$ 80

6.2 Finite precision FIR filter coefficients 82

6.3 $\hat{G}_k(z)$ definitions 83

6.4 Finite precision equalizer coefficients 88

7.1 Power report of the decimation filter 91

7.2 Decimation filter design summary 92

A.1 Static power consumption for XOR, AND and OR gates 103

A.2 Static power consumption of the D-Flip flop and the power generation circuit 103

A.3 Static power and timing analysis of a single bit accumulator . . 105

LIST OF FIGURES

1.1 Decimation filter for a $\Delta\Sigma$ modulator 2

2.1 Block diagram of a down-sampler 5

2.2 Down-sampling a sine wave 5

2.3 Effect of down-sampling in frequency domain 7

2.4 Cascade equivalence of decimation filter 7

2.5 Block diagram of a decimation filter for the $\Delta\Sigma$ modulator . . . 9

2.6 Magnitude response of the NTF 11

2.7 Output of the $\Delta\Sigma$ modulator 11

2.8 Single stage and Multi-stage implementation of a decimation filter 12

2.9 Single stage vs Two stage implementation of decimation filter. . 14

2.10 Block diagram of the two stage decimation filter. 16

2.11 Frequency response of sinc filter 17

2.12 Comparison of frequency responses of sinc filters with five and ten taps 19

2.13 Passband droop and alias rejection in sinc 20

2.14 FFT of the output of the sinc decimation filter 20

2.15 Normalized frequency response of the FIR filter $H_{pc}(z)$ 22

2.16 Normalized frequency response of FIR filter- closer look 23

2.17 Droop and ripple in passband of $H_{pc}(z)$ 24

2.18 FIR filter and SINC filter's frequency response superimposed . . 25

2.19 64^{th} order FIR filter 27

2.20 FFT of the signal after FIR filter and decimator 30

2.21 Decomposition of sinc decimator for CIC implementation 31

2.22 Sinc decimation in Hogenauer or CIC architecture 31

2.23 Polyphase decomposition of sinc filter 35

2.24 Polyphase decomposition of sinc filter 35

2.25 Polyphase decomposition of FIR the filter 37

2.26 Polyphase decomposition of the FIR filter 37

3.1 Circuit implementation of $E_0(z)$ and $E_1(z)$ 40

3.2 Circuit implementation of $E_2(z)$ and $E_4(z)$ 41

3.3 Circuit implementation of $E_3(z)$ 42

3.4 Wallace adder schematic 45

3.5 Wallace adder RCA 46

3.6 Reduction of partial products 47

3.7 Partial product reduction block 48

3.8 Wallace tree addition of partial products 49

3.9 Careful grouping of partial products can decrease the number of adders 50

3.10 Basic operations involved in Kogge-Stone adder 53

3.11 Associative property of dot operation 53

3.12 Schematic of 14-bit Kogge-Stone logarithmic look ahead adder . 54

3.13 Complete sinc filter schematic 56

4.1 Circuit implementation of $G_4(z)$ 60

4.2 Circuit implementation of $G_0(z)$ and $G_3(z)$ 63

4.3 Circuit implementation of $G_1(z)$ and $G_2(z)$ 65

4.4 Minimizing the number of adders 68

4.5 Addition of partial products in FIR filter 70

5.1 Basic decimator structure and timing diagram 72

5.2 Low power decimator structure and timing diagram 74

5.3 Ring counter 76

5.4 Johnson's counter 76

6.1 Passband droop and alias rejection in 5-tap sinc filter 79

6.2 Normalized frequency response of the FIR filter $\hat{H}_{pc}(z)$ 80

6.3 FIR filter and Sinc filter's frequency response superimposed . . 81
6.4 Circuit implementation of $\hat{G}_4(z)$ 84
6.5 Circuit implementation of $\hat{G}_0(z)$ and $\hat{G}_3(z)$ 85
6.6 Circuit implementation of $\hat{G}_1(z)$ and $\hat{G}_2(z)$ 86
6.7 Frequency response of the equalizer 88

7.1 Block diagram of the design flow 90
7.2 Layout 93

A.1 Pipelining an accumulator based on RCA 95
A.2 First accumulator in CIC implemented using CSA 97
A.3 Second accumulator onwards in CIC using CSA 98
A.4 Pipelining the 2I2O accumulator 99
A.5 Retiming the 2I2O accumulator 99
A.6 Second accumulator onwards in CIC using pipelined retimed CSA 100
A.7 Full adder circuit 101
A.8 XOR gate in CML 102
A.9 AND and OR gate in CML 102
A.10 D-Flipflop in CML 104
A.11 Bias generator circuit 104

A.1 Reset synchronizer circuit 106

ADC	Analog to Digital Converter
CIC	Cascaded Integrator Comb
CML	Current Mode Logic
CPD	Critical Path Delay
CLA	Carry Look ahead Adder
CSA	Carry Save Accumulator
FA	Full Adder
FFT	Fast Fourier Transform
FIR	Finite Impulse Response
LPF	Low pass filter
LSB	Least Significant Bit
MSA	Maximum Stable Amplitude
MSB	Most Significant Bit
NTF	Noise Transfer Function
RCA	Ripple Carry Adder
SDM	Sigma Delta Modulator
SNR	Signal to Noise Ratio
SoC	System on Chip
VMA	Vector Merging Adder

CHAPTER 1

Introduction

1.1 Motivation

Even though all the real world signals are inherently analog, we convert them into digital form since digital signals can be processed using robust and reliable digital signal processing (DSP) techniques. Also digital signals are less susceptible to noise and are easier for computations. Therefore, analog-to-digital conversion becomes a crucial part in today's *Systems-on-Chip*(SoCs).

Oversampling $\Delta\Sigma$ ADCs have found increasing use in modern electronic components. Oversampling of signal greatly relaxes the anti-aliasing filter and the reconstruction filter requirements. Also with its noise-shaping technique shown in Figure 1.1, the $\Delta\Sigma$ modulation gives high resolution for modern day ADCs[1].

Once the quantization noise is shaped out of the passband as shown in Figure 1.1, a low pass filter(LPF) is required to remove the out of band components and a decimator is needed to the bring the sampling rate back to the Nyquist rate from the oversampled rate. Both these tasks are performed by a decimation filter, which as its name suggests, low pass filters and decimates (brings down the sampling rate) the input signal.

The aim of this project is to design a low power decimation filter for a high speed $\Delta\Sigma$ modulator. It becomes clear that for a decimation filter to work in the Gigahertz range, the conventional architectures of the decimation filter cannot be used. So the issues involved in the design of high speed decimation filters are considered before going to the details of the design. Since the low pass filtering and the decimation are performed on the output signal of the $\Delta\Sigma$ modulator, the entire decimation filter system works in the digital domain.

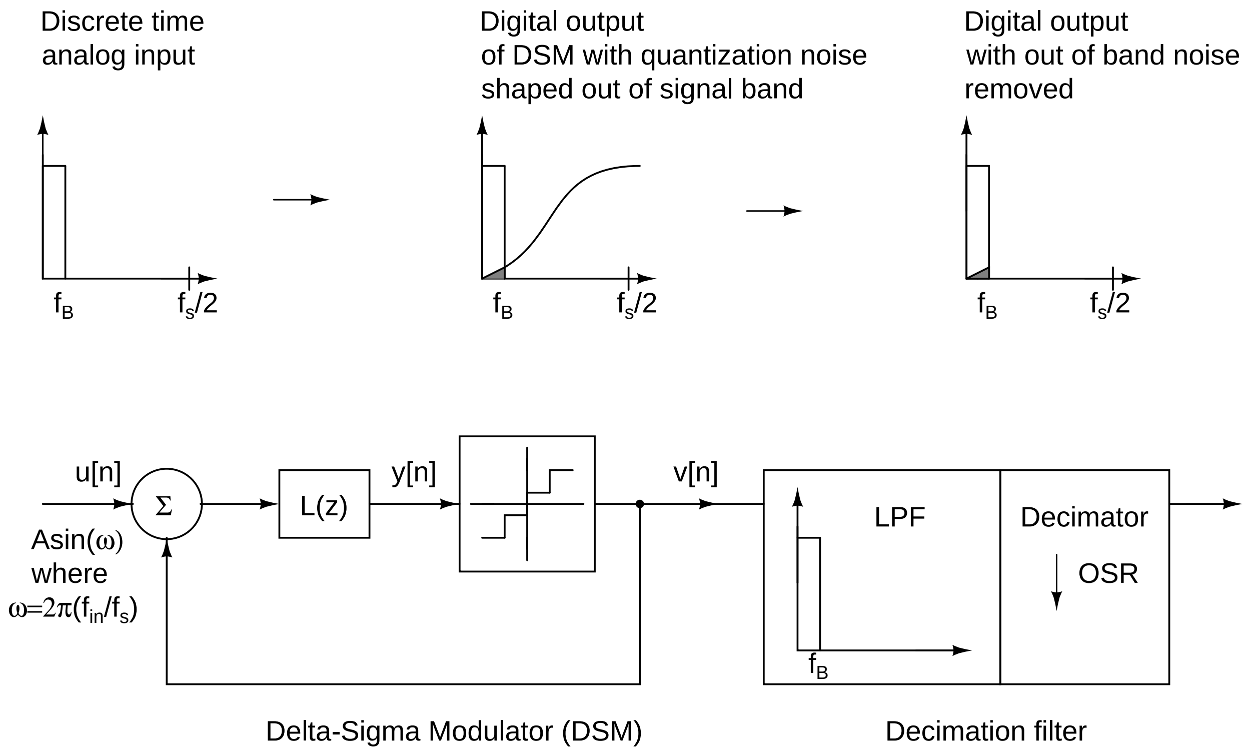

Discrete time analog input
Digital output of DSM with quantization noise shaped out of signal band
Digital output with out of band noise removed
f_B
$f_s/2$
f_B
$f_s/2$
f_B
$f_s/2$
u[n]
Σ
L(z)
y[n]
v[n]
LPF
Decimator
OSR
Asin(ω) where ω=2π(f_{in}/f_s)
f_B
Delta-Sigma Modulator (DSM)
Decimation filter

Figure 1.1: Block diagram of decimation filter for a ΔΣ modulator. f_B is the maximum inband frequency. OSR = Oversampling ratio.

1.2 Organization

Chapter 2 introduces the basics of Delta Sigma Modulation(DSM), decimation and decimation filter design. Specifications of the ΔΣ modulator and decimation filter are considered and a final two stage design of decimation filter is derived. Possible architectures for decimation filters are also discussed.

Chapter 3 gives the details of the circuit implementation of the first stage of the decimation filter involving a sinc filter. Challenges involved in implementing the sinc filter in polyphase architecture are discussed in detail.

Chapter 4 discusses the design details of of the second stage FIR filter of the decimation filter.

Chapter 5 discusses the design and implementation of the decimators and clock dividers used in the decimation filter.

Chapter 6 explains the design of a decimation filter with minimal passband droop.

Chapter 7 concludes the thesis with simulation results after synthesis, placement and routing.

CHAPTER 2

Decimation Filter Design and Architecture

2.1 Decimation filter fundamentals

There are many applications where the signal at a given sampling rate needs to be converted to another signal with a different sampling rate. Discrete-time systems with unequal sampling rates at various parts of the system are called *multirate* systems. The process of reducing the sampling rate is generally called *decimation*, and the multirate structure used for decimation is called *decimator*. Accompanying a decimator is usually a low pass filter, which along with the decimator makes a *decimation filter*. In this chapter, the basic theory of decimation filters and their application in $\Delta\Sigma$ modulation are discussed.

2.1.1 Time domain characteristics

To change the sampling rate of a digital signal, multirate digital signal processing uses an *up-sampler* or *down-sampler*. These are two basic sampling rate alteration devices apart from adders, multipliers, delay elements required in multirate systems. To reduce the sampling rate or to decimate a signal, we need a down-sampler, the time domain definition of which is given in (2.1).

$$y[n] = x[nM] \tag{2.1}$$

where M is the down-sampling or decimation factor. In English, (2.1) means that output $y[n]$ is one out of every M samples of $x[n]$ or every M^{th} sample of $x[n]$. The block diagram representing the process of down-sampling is given in Figure 2.1. Down-sampling is further illustrated in Figure 2.2 for a sinusoidal signal with

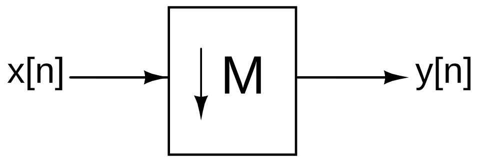


Figure 2.1: Block diagram of a down-sampler or decimator with down-sampling or decimation factor M

amplitude 1. Here the value of $M = 2$ and the process of down-sampling is accomplished by selecting every $2^{nd}\,(M^{th})$ sample.

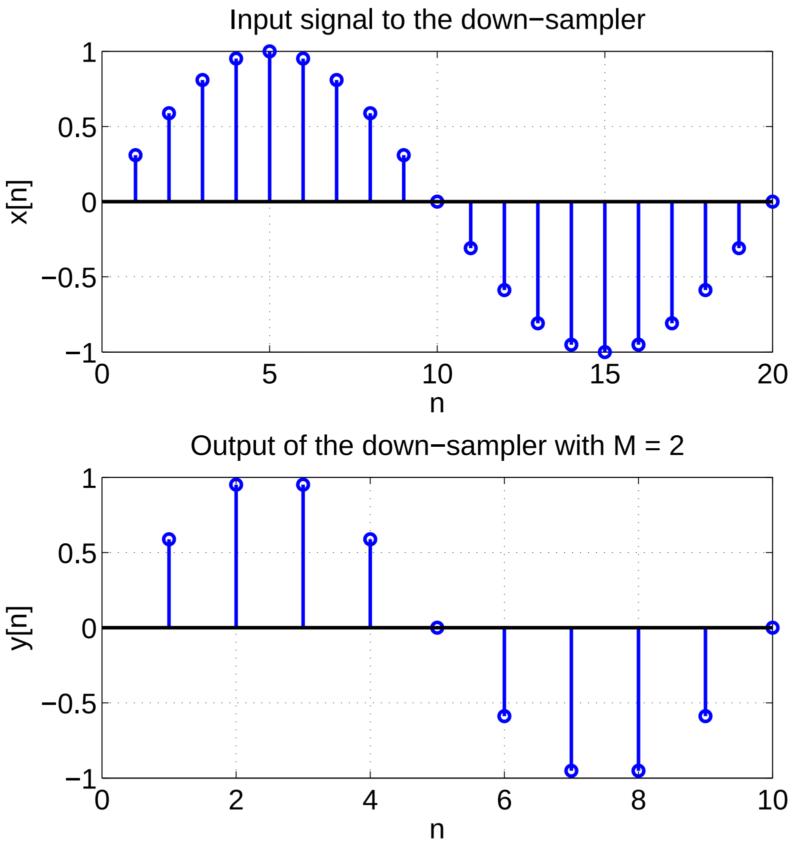


Figure 2.2: Down-sampling a discrete-time sine wave by a factor, $M = 2$

2.1.2 Frequency domain characteristics

Applying z-transform to the input-output relation given in (2.1), we get

$$Y(z) = \sum_{n=-\infty}^{\infty} x[nM] z^{-n} \tag{2.2}$$

Expression $Y(z)$ in terms of $X(z)$ is derived to be [2]

$$Y(z) = \frac{1}{M} \sum_{k=0}^{M-1} X(z^{1/M} e^{j2\pi k/M}) \tag{2.3}$$

Despite the complicated expression of $Y(z)$ in (2.4), the effect of down-sampling in the frequency domain can be understood easily by considering the simple case of $M = 2$. For $M = 2$, $z = e^{j\omega}$, we get

$$Y(e^{j\omega}) = \frac{1}{2}\Big(X(e^{j\omega/2}) + X(-e^{j\omega/2})\Big) \tag{2.4}$$

Equation (2.4) shows that the frequency response of the output of the down-sampler with $M = 2$ consists of two terms. First term $X(e^{j\omega/2})$ is the frequency response of the input signal, $X(e^{j\omega})$ stretched along ω-axis by a factor of two and the second term $X(-e^{j\omega/2})$ is simply the first term shifted by 2π along ω axis (Figure 2.3) i.e.

$$X(-e^{j\omega/2}) = X(e^{j(\omega+2\pi)/2}) \tag{2.5}$$

From Figure 2.3, it is clear that aliasing can occur if down-sampling results in a sampling rate less than the Nyquist rate. In the figure, input has a sampling rate twice the Nyquist rate and down-sampling results in Nyquist rate. Thus if you down-sample by any factor greater than 2, it will cause aliasing. For a general value of M, $Y(e^{j\omega})$ is a sum of stretched and shifted versions of $X(e^{j\omega})$ scaled by a factor M.

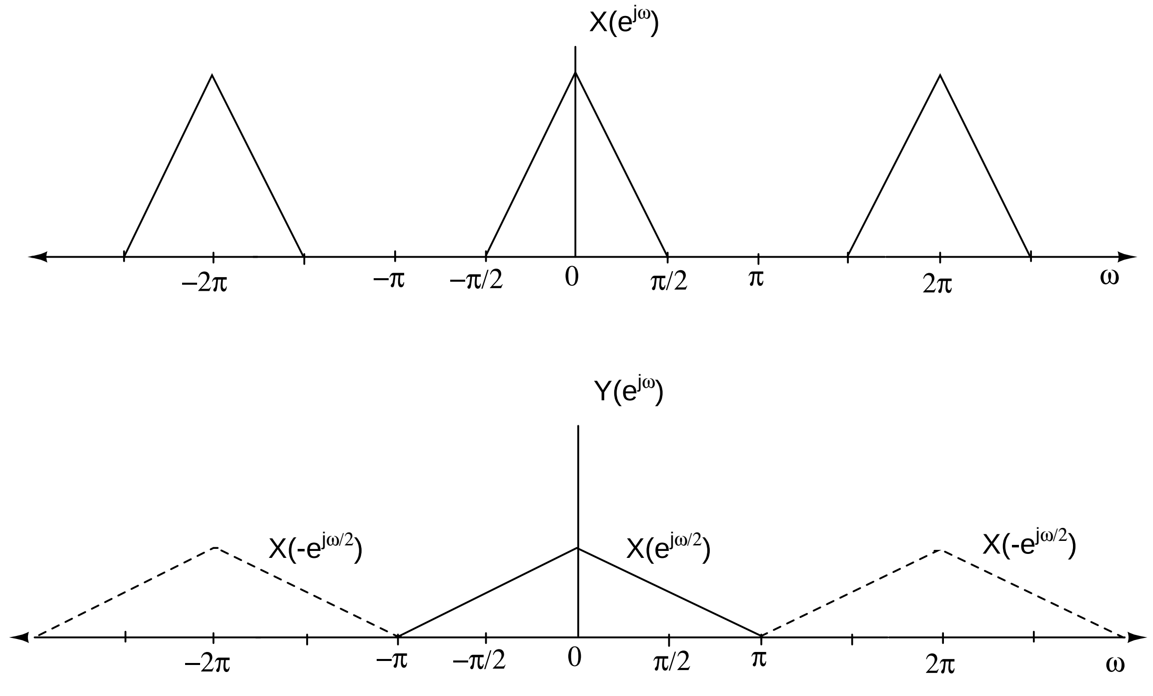


Figure 2.3: Effect of down-sampling by a factor M=2 in frequency domain

2.1.3 Cascade equivalence of decimation filters

Basic sampling rate alteration devices like down-samplers and components of LTI digital filters forms multirate systems. In most applications, these two appear in cascade as shown in Figure 2.4. An interchange in the position in a cascade can sometimes lead to computationally efficient realizations. A particularly useful cascade equivalence property for our design is shown in Figure 2.4 and the equivalence can be proved by showing that the z-domain transfer function of $y_1[n]$ and $y_2[n]$ are identical.

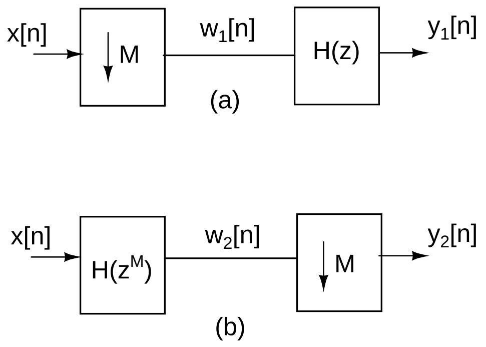


Figure 2.4: Cascade equivalence of decimation filter

Two decimators in Figure 2.4 give

$$w_1[n] = x[nM] \tag{2.6}$$

$$y_2[n] = w_2[nM] \tag{2.7}$$

Recalling the input-output relation of a down-sampler in (2.4), the cascade in Figure 2.4(a) in z-domain gives

$$W_1(z) = \frac{1}{M}\sum_{k=0}^{M-1} X(z^{1/M}e^{j2\pi k/M}) \tag{2.8}$$

$$Y_1(z) = W_1(z)H(z) \tag{2.9}$$

Cascade in Figure 2.4(b) in z-domain gives

$$W_2(z) = X(z)H(z^M) \tag{2.10}$$

$$Y_2(z) = \frac{1}{M}\sum_{k=0}^{M-1} W_2(z^{1/M}e^{j2\pi k/M}) \tag{2.11}$$

Substituting the expression for $W_2(z)$ in that of $Y_2(z)$, we get

$$Y_2(z) = \frac{1}{M}\sum_{k=0}^{M-1} X(z^{1/M}e^{j2\pi k/M})H(ze^{j2\pi k}) \tag{2.12}$$

$$= \frac{1}{M}\sum_{k=0}^{M-1} X(z^{1/M}e^{j2\pi k/M})H(z) \tag{2.13}$$

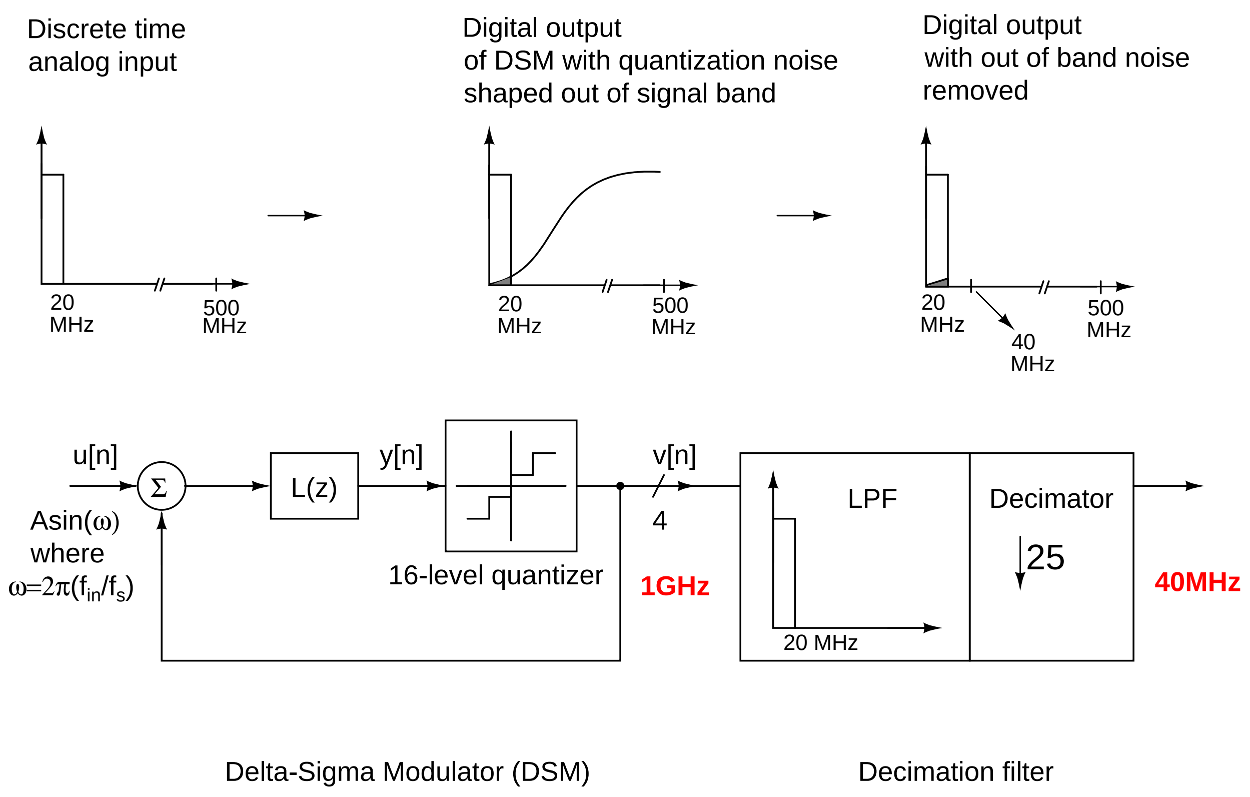
$$= W_1(z)H(z) \tag{2.14}$$

$$= Y_1(z) \tag{2.15}$$

Thus the equivalence of $y_1[n]$ and $y_2[n]$ are proved[2]. As we will see later, this cascade equivalence property is crucial in implementing the decimation filter using polyphase decomposition.

2.2 ΔΣ modulator specifications

As we have stated in chapter 1, the goal of this project is to design a decimation filter for a ΔΣ modulator. So before we set about designing the decimation filter, the specifications of the ΔΣ modulator which has already been designed, taped out and tested has to be stated and understood. The output of the ΔΣ modulator which has to be filtered and decimated should be generated from a model since it is this same output that forms the input to the required decimation filter.



Discrete time analog input
20 MHz
500 MHz
Digital output of DSM with quantization noise shaped out of signal band
20 MHz
500 MHz
Digital output with out of band noise removed
20 MHz
40 MHz
500 MHz
u[n]
Σ
Asin(ω) where ω=2π(f_in/f_s)
L(z)
y[n]
16-level quantizer
v[n]
4
1GHz
LPF
20 MHz
Decimator
25
40MHz
Delta-Sigma Modulator (DSM)
Decimation filter

Figure 2.5: Block diagram of a decimation filter for the ΔΣ modulator with $OSR = 25$, $f_s = 1\,\text{GHz}$ f_B=20 MHz and 4-bit output.

The ΔΣ modulator for which the decimation filter has to be designed has a sampling frequency, $f_s = 1\,\text{GHz}$ and OSR 25 with a 4-bit output. The general block diagram of a ΔΣ modulator and its decimation filter shown in Figure 1.1 is redrawn here in Figure 2.5 for our particular case. The ΔΣ modulator has an $OSR = 25$ and gives a 4-bit output. The Over-Sampling Ratio or $OSR = 25$

means that the sampling rate is 25 times the Nyquist frequency. So to bring the system back to Nyquist rate after decimation, we have to decimate by a factor 25. The maximum frequency component present in the input signal or the input bandwidth, f_B can be calculated as shown below. Since $f_s = OSR \times$ Nyquist rate,

$$f_s = \text{OSR} \times 2f_B \tag{2.16}$$

$$\implies f_B = \frac{f_s}{2 \times \text{OSR}} = \frac{1\,\text{GHz}}{50} = 20\,\text{MHz} \tag{2.17}$$

After the $\Delta\Sigma$ modulation, the quantization noise will be shaped out of the signal band and thus the decimation to Nyquist rate should be accompanied by low pass filtering which removes the out of band noise components. Since the input bandwidth is 20 MHz, we essentially need a structure with a low pass filter with passband 20 MHz and a decimator with decimating factor 25. The output of the decimation filter will be a digital signal with sampling rate 1 GHz/25 =40 MHz which is the Nyquist rate.

The noise-shaping of the modulator is determined by the Noise-Transfer Function or NTF of the $\Delta\Sigma$ modulator. The NTF of the $\Delta\Sigma$ modulator in question is given in (2.18) and the magnitude response of this NTF is shown in Figure 2.6.

$$NTF(z) = \frac{(1+1.352z^{-1})(1-1.998z^{-1}+z^{-2})(1-1.988z^{-1}+z^{-2})}{(1-1.204z^{-1}+0.3771z^{-2})(1-1.43z^{-1}+0.6585z^{-2})} \tag{2.18}$$

To generate the input signal for the decimation filter, the $\Delta\Sigma$ modulator with the NTF defined in (2.18), OSR 25 and a 16-level quantizer is modelled and simulated in MATLAB using Schreier's Delta Sigma Toolbox[3]. An input signal $A\sin(2\pi(f_{in}/f_s)n)$ is given to the modulator with the value of A chosen to be slightly less than the Maximum Stable Amplitude or MSA of the modulator. Considering that the OSR is 25, which means the decimation factor is also 25, it is better to have the number of input points and FFT bins of the form 5^n. Thus

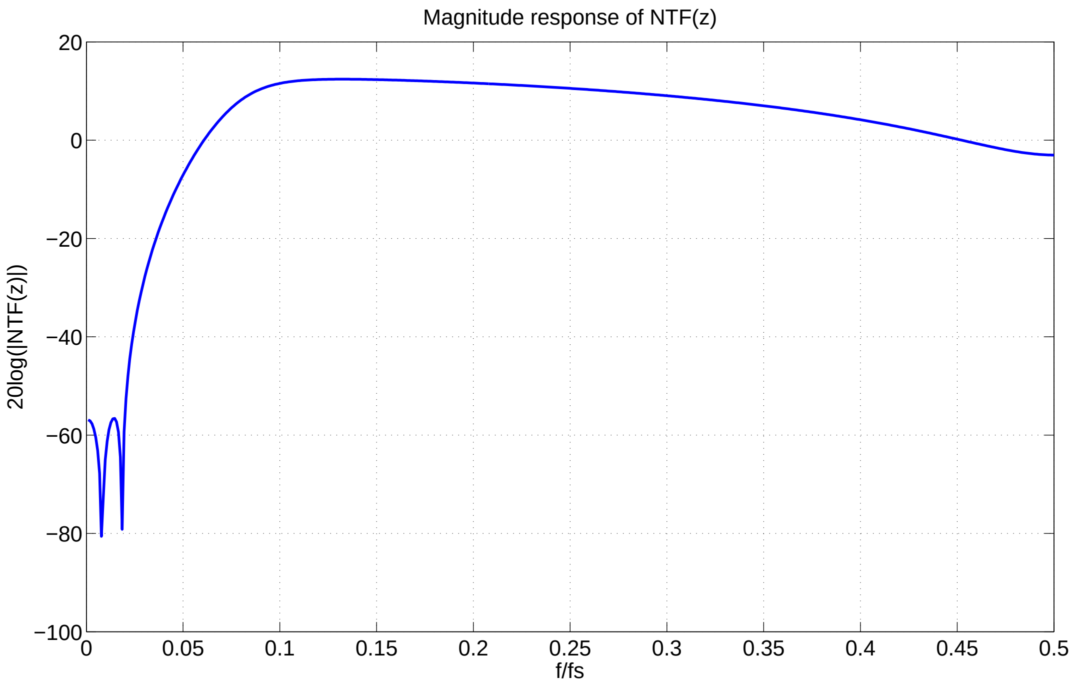


Figure 2.6: Magnitude response of the NTF of the ΔΣ modulator.

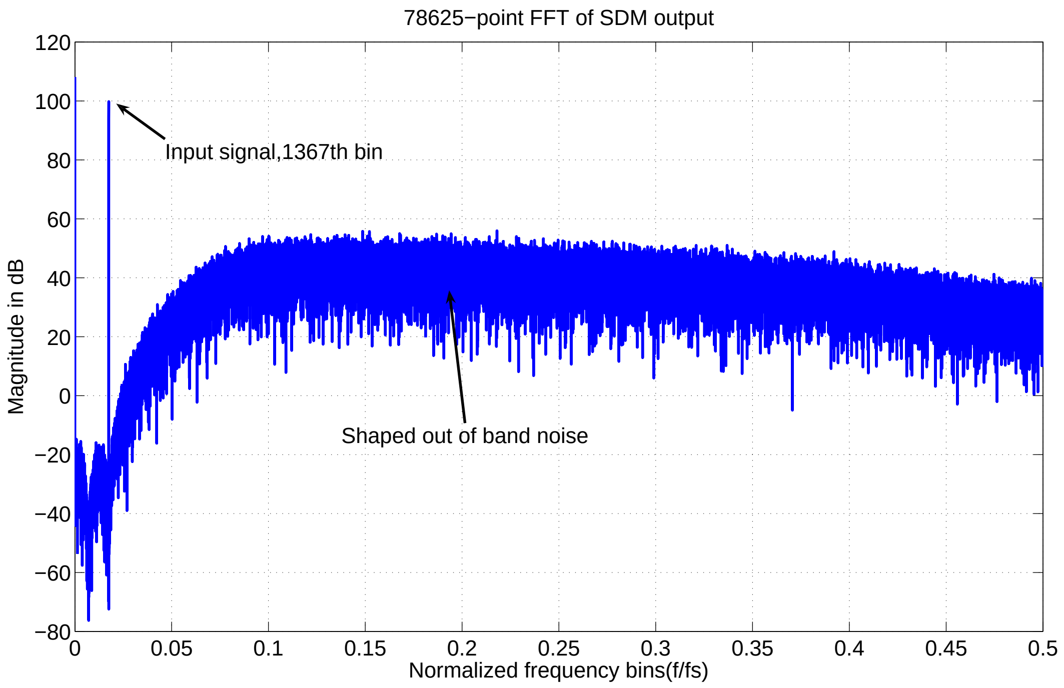


Figure 2.7: Output of the ΔΣ modulator or equivalently the input of the Decimation filter.

a 5^7-point FFT of the output of the $\Delta\Sigma$ modulator is shown in Figure 2.7.

For this design of the $\Delta\Sigma$ modulator, the maximum SNR after decimation and low pass filtering can be calculated. Assuming an ideal low pass filter with sharp response, SNR can be calculated using the *simulateSNR* command in the Delta-Sigma toolbox. The maximum possible SNR we can get after decimation filter is calculated to be 96 dB.

2.3 Multi-stage decimation filter

A general decimation filter that we have considered so far, as shown in Figure 2.8(a) is a single stage structure where the basic scheme for sampling alteration involves a single filter $H(z)$ and a single decimator of factor M. Now consider the case where M can be factorized into $M_1 \times M_2 \times ... \times M_n$. Then we can implement the same structure in n stages as shown in Figure 2.8(b). This is called the multi-stage implementation of decimation filter[4].

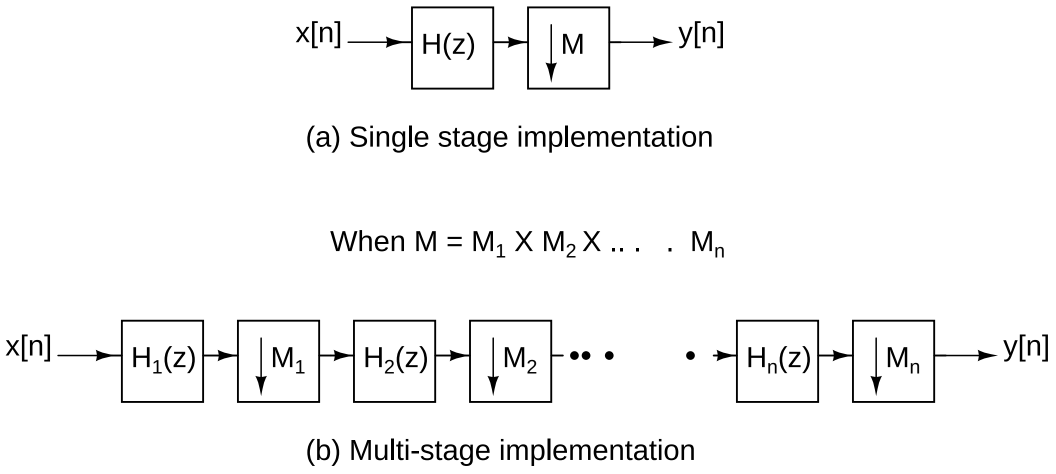

x[n]
H(z)
M
y[n]
(a) Single stage implementation
When M = M1 X M2 X .. . . Mn
x[n]
H1(z)
M1
H2(z)
M2
Hn(z)
Mn
y[n]
(b) Multi-stage implementation

Figure 2.8: Single stage and Multi-stage implementation of a decimation filter with decimating factor $M = M_1 \times M_2 \times$ .. . M_n.

It is shown that it is computationally efficient to implement the decimation filter in multiple stages than in a single stage. This is true despite the increase in the number of filters in multi-stage implementation because in Figure 2.8(b) the frequency of operation decreases as we go from $H_1(z)$ to $H_n(z)$ and the power dis-

Table 2.1: Assumed parameters for the LPF in the single stage implementation shown in Figure 2.9 to estimate order, N

Filter parameter	value
Passband ripple, δ_p	1 dB
Stopband ripple, δ_s	1 dB
Passband edge, f_p	19 MHz
Stopband edge, f_s	21 MHz

sipation of the digital filters is directly proportional to the frequency of operation. Apart from this, it is also possible to exploit the fact that steep filter responses are not necessary in the initial stages resulting in lower order filters operating at higher frequencies and higher order filters operating at lower frequencies. This makes the overall decimation filter better off.

Since the required decimation factor 25 can be written as 5×5, it is possible to implement the decimation filter in two stages. In this section, we discuss the advantages of implementing the filter in two stages over a single stage.

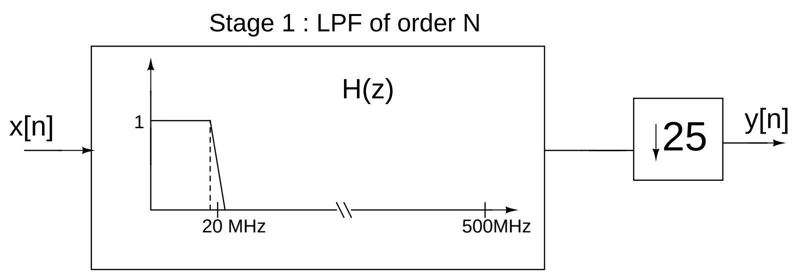
2.3.1 Single stage implementation

Consider the single stage decimation filter shown in Figure 2.9(a). A simple formula to estimate the order of an FIR filter is Kaiser's formula [2] given in (2.19). The assumed parameters for the filter $H(z)$ are given in Table 2.1.

$$N \cong \frac{-20\log_{10}(\sqrt{\delta_p \delta_s}) - 13}{14.6(\omega_s - \omega_p)/2\pi} \tag{2.19}$$

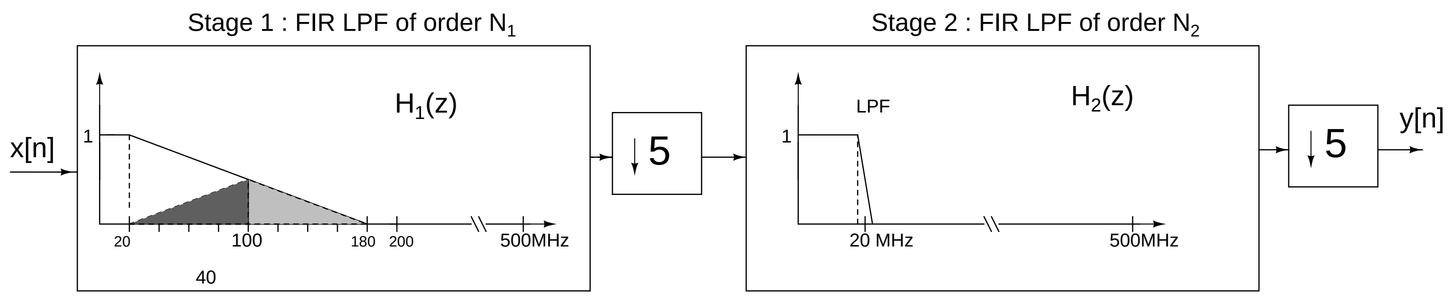
For the assumed parameters given in Table 2.1, Kaiser's formula gives a value of the order of the filter $N \approx 240$. Thus the approximate number of multiplications per second R_1, when this filter is implemented is

$$R_1 \approx \frac{N}{2} \times 40\,\text{MHz} = 4.8 \times 10^9\,s^{-1} \tag{2.20}$$



(a) Single stage decimation filter

Since 25 = 5 X 5


Stage 1 : FIR LPF of order N_1
$H_1(z)$
x[n]
1
20
100
180
200
500MHz
40
↓5
Stage 2 : FIR LPF of order N_2
LPF
$H_2(z)$
1
20 MHz
500MHz
↓5
y[n]

(b) Two-stage decimation filter

Figure 2.9: Single stage vs Two stage implementation of decimation filter.

Note the division by 2 in calculating R_1, which is due to the symmetry of the linear phase FIR filter coefficients.

2.3.2 Two stage implementation

The only way the decimation factor 25 can be factorized into integers is 5×5. With this in mind, consider the two stage structure shown in Figure 2.9(b). The first stage will bring down the sampling rate to 200 MHz and second stage to 40 MHz. Unlike the single stage structure, where the transition of the LPF from passband to stopband needs to be really sharp to avoid aliasing to the passband, in a two stage structure, only the transition of second stage FIR filter needs to be sharp. As shown in Figure 2.9, the transition band of the first stage filter can extend from 20 MHz to as large as 180 MHz and still there will not be any aliasing to the passband. By exploiting this fact, we can greatly reduce the computational complexity of the decimation filters[4].

Assuming the same ripple values in Table 2.1 for FIR filters in two stage also, the approximate orders, N_1 and N_2 for the first and second stage FIR filters can be calculated using Kaiser's formula to be 3 and 47 respectively. The order of the first stage is as low as three because of the large transition band extending from 20 MHz to 180 MHz. The total number of multiplications per second, R_2 in two stage implementation thus becomes

$$\begin{aligned} R_2 &= \frac{N_1}{2} \times (200\,\text{MHz}) + \frac{N_2}{2} \times (40\,\text{MHz}) && (2.21) \\ &= \frac{3}{2} \times (200\,\text{MHz}) + \frac{47}{2} \times (40\,\text{MHz}) && (2.22) \\ &= 1.24 \times 10^9\, s^{-1} && (2.23) \end{aligned}$$

This preliminary analysis shows that number of multiplications per second required for two stage implementation is much lower(less by a factor $R_1/R_2 \approx 3.9$) than that required for a single stage implementation i.e. it is computationally efficient to implement the required decimation filter in two stages. Due to myriad reasons, it is conventional to use a sinc filter in the first stage of a multi-stage decimation filter[5][6]. Thus in this design also we will use a sinc filter for the first stage.

The entire decimation filter has now become a two stage system with a sinc filter in the first stage and an FIR filter in the second. For both stages, decimating factor is 5 (Figure 2.10).

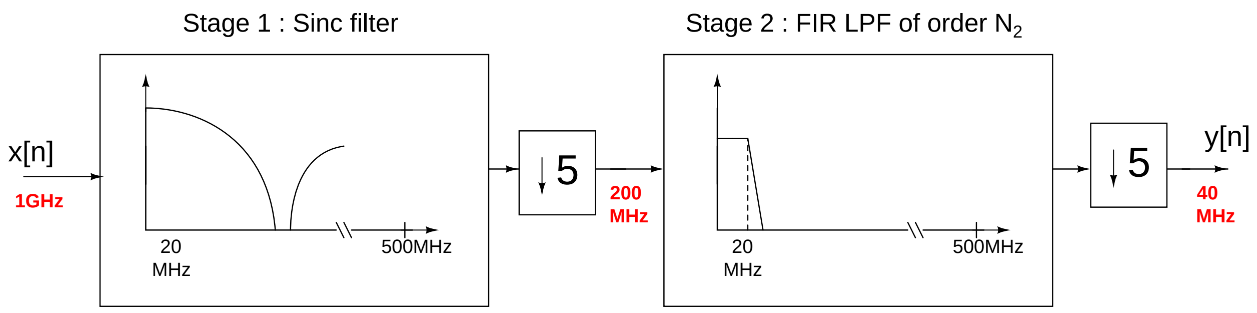


Figure 2.10: Block diagram of the two stage decimation filter.

As we will see later on, one of the main reasons to use a sinc filter in first stage, the simple architecture, breaks down due to the high speed requirement(1 GHz) of this design. Thus we are actually free to use any FIR filter for the first stage.

2.4 Sinc filter characteristics

Sinc filter is proven to be very efficient for the first stage in a multi-stage decimator and can be used to decimate to a rate four times the Nyquist rate[5][6]. Since OSR is 25, the fact that a sinc filter can be used to decimate to four times the Nyquist rate means that the decimation factor in first stage, M_1 can be as high as $25/4 = 6.25$. i.e. M_1 should be ≤ 6.25. Since both $M_1 = M_2 = 5$ in our case, this condition is met. Thus a sinc filter can be used as the first stage FIR filter in the two stage implementation scheme and it will decimate the sampling rate to five times the Nyquist rate. A general sinc filter $H_s(z)$ of order K with $(r \times M)$

taps is of the form

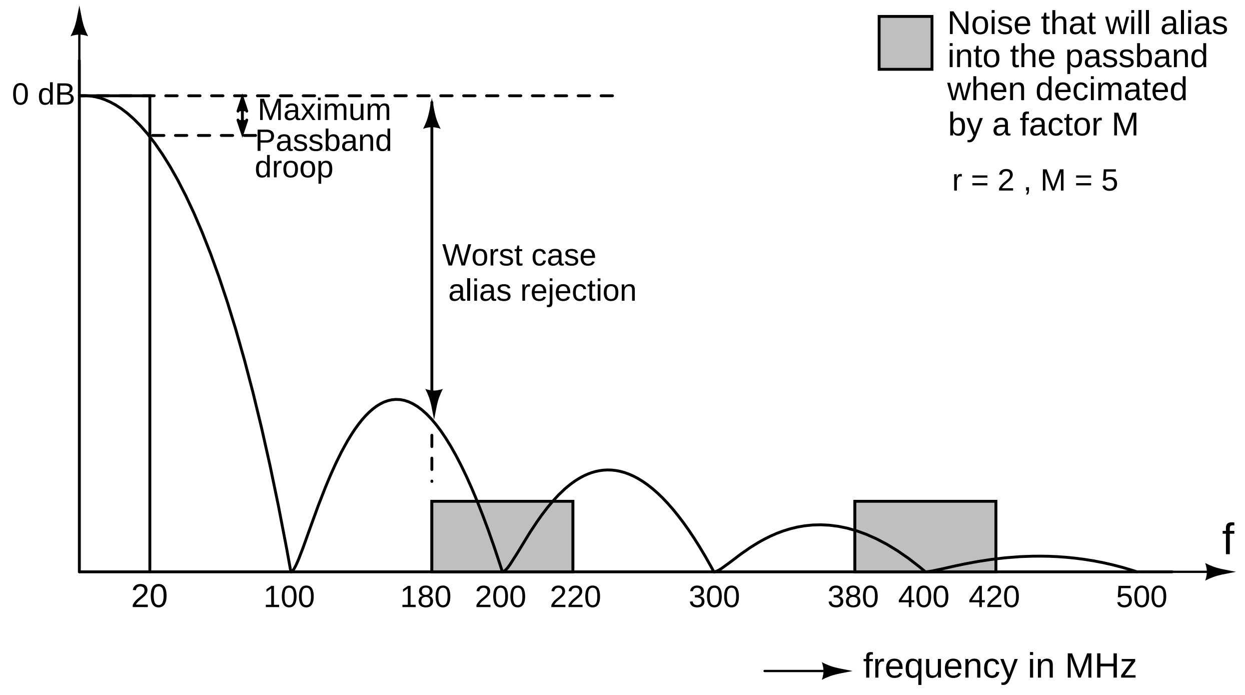
$$H_s(z) = \left(\frac{1}{rM} \left(\frac{1 - z^{-rM}}{1 - z^{-1}} \right) \right)^K \tag{2.24}$$

where the nulls are at $\pm l(f_s/rM)$ where l = 1, 2, . . rM-1 as shown in Figure 2.11. Note that

$$\left(\frac{1 - z^{-rM}}{1 - z^{-1}} \right) = 1 + z^{-1} + z^{-2} \ldots z^{-(rM-1)} \tag{2.25}$$

Thus a sinc filter of order K is simply K moving average filters in cascade.

Now consider the input signal of bandwidth 20 MHz in Figure 2.11. From section 2.1.2, we know that when decimated by a factor of 5, noise at the frequencies in the range $(200 - 20)$ MHz $< f <$ $(200 + 20)$ MHz, $(400 - 20)$ MHz $< f <$ $(400 + 20)$ MHz will alias into the passband (± 20 MHz of f_s/M,$2f_s/M \ldots$ because signal bandwidth is only 20 MHz).



Noise that will alias into the passband when decimated by a factor M
r = 2 , M = 5
0 dB
Maximum Passband droop
Worst case alias rejection
f
20
100
180
200
220
300
380
400
420
500
frequency in MHz

Figure 2.11: Frequency response of sinc filter showing noise aliasing to the passband. Diagram is not to scale.

Table 2.2: Variation of maximum droop and worst alias attenuation with the value of r (K=4, M=5).

Value of r	Maximum passband droop at f_B	Worst alias rejection at $(f_s/M - f_B)$
1	-0.5507 dB	-75.04 dB
2	-2.2950 dB	-76.79 dB
3	-5.2840 dB	-79.78 dB
4	-9.6600 dB	-84.15 dB

The decimation factor, M of the decimator which follows the sinc filter is fixed to be 5. Value of r has to be chosen after considering the passband droop of sinc filter into account. Droop is the deviation of the sinc filter response from the ideal value of unity (0 dB) in the passband. The maximum passband droop occurs at $f_B = 20\,\text{MHz}$. The worst case alias rejection is the attenuation at the frequency $(200 - 20)\,\text{MHz} = 180\,\text{MHz}$. Passband droops and worst case alias rejection for different values of r (for $K = 4$, $M = 5$) are shown in Table 2.2. The plots of fourth order sinc filter frequency responses for $r = 1$, $M = 5$ (5 taps) and $r = 2$, $M = 5$ (10 taps) are compared in Figure 2.12.

Usually a maximum passband droop of 3dB or above is difficult to correct using an equalizer[7, p. 15]. So from Table 2.2, it is clear that we can choose r to be one or two. The value of r is chosen to be 2 since this will greatly relax the requirements of the second stage FIR filter. For $r = 2$, passband droop is about 2.4 dB as shown in Figure 2.13. However if a lower droop is required, a five tap filter has to be chosen, the details of which are discussed in chapter 6. Alias rejection and droop increase as K increases. A fourth order sinc filter is found to be sufficient to meet the SNR requirement in the final stage. So K is chosen to be 4. Thus the final transfer function of the sinc filter becomes

$$H_s(z) = \left(\frac{1}{10} \left(\frac{1 - z^{-10}}{1 - z^{-1}} \right) \right)^4 \tag{2.26}$$

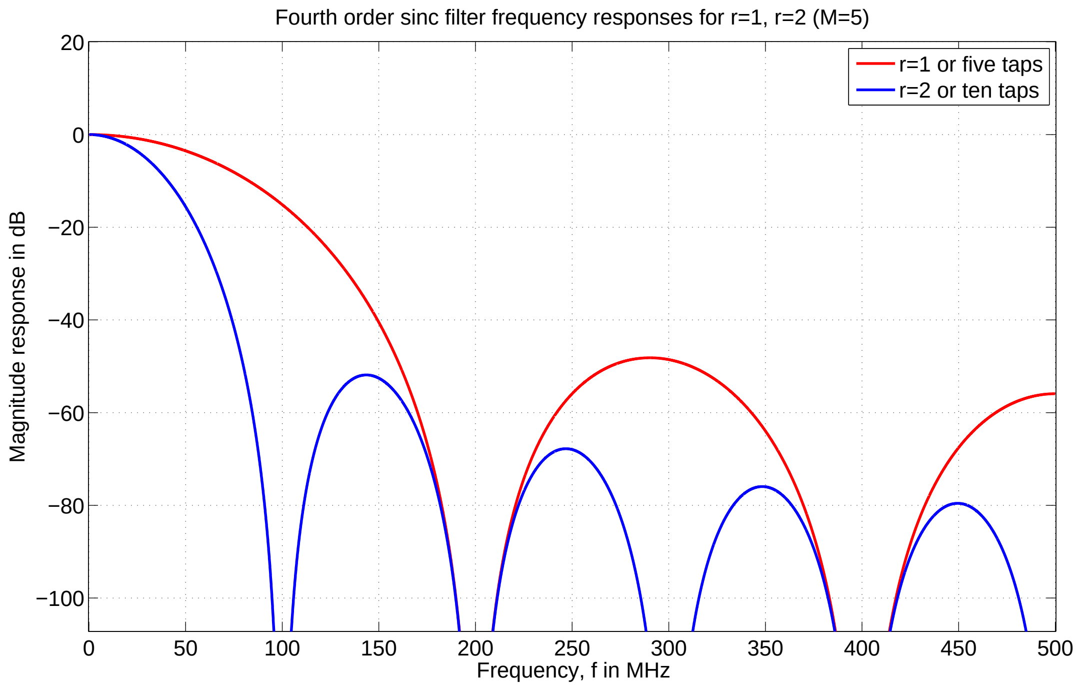


Figure 2.12: Frequency responses of sinc filter as given in (2.26) with $r = 1$ and $r = 2$. For both responses, $K = 4$, $M = 5$. Note that for the 10-tap filter alias rejection is better but passband droop is higher.

For $z = e^{j\omega}$,

$$H_s(e^{jw}) = \left(\frac{1}{10} \left(\frac{\sin(10\omega/2)}{\sin(\omega/2)} \right) \right)^4 \tag{2.27}$$

The frequency response of the sinc filter given in (2.27) is plotted in Figure 2.12. The maximum passband droop and worst case alias rejection for the chosen sinc filter are shown in Figure 2.13.

As we have mentioned before, the input to the sinc filter is the output from the DSM. Since we have designed the sinc filter for a 4-bit unsigned input, the output from the DSM has to be scaled and offset. The output of DSM corresponding to an input signal $A\sin(2\pi(f_{in}/f_s)n)$ is given as the input to the sinc filter. The output signal after the sinc filtering and decimation is shown in Figure 2.14.

The SNR after sinc decimation is calculated to be 54.69dB. This 14-bit output forms the input to the second stage of the decimation filter.

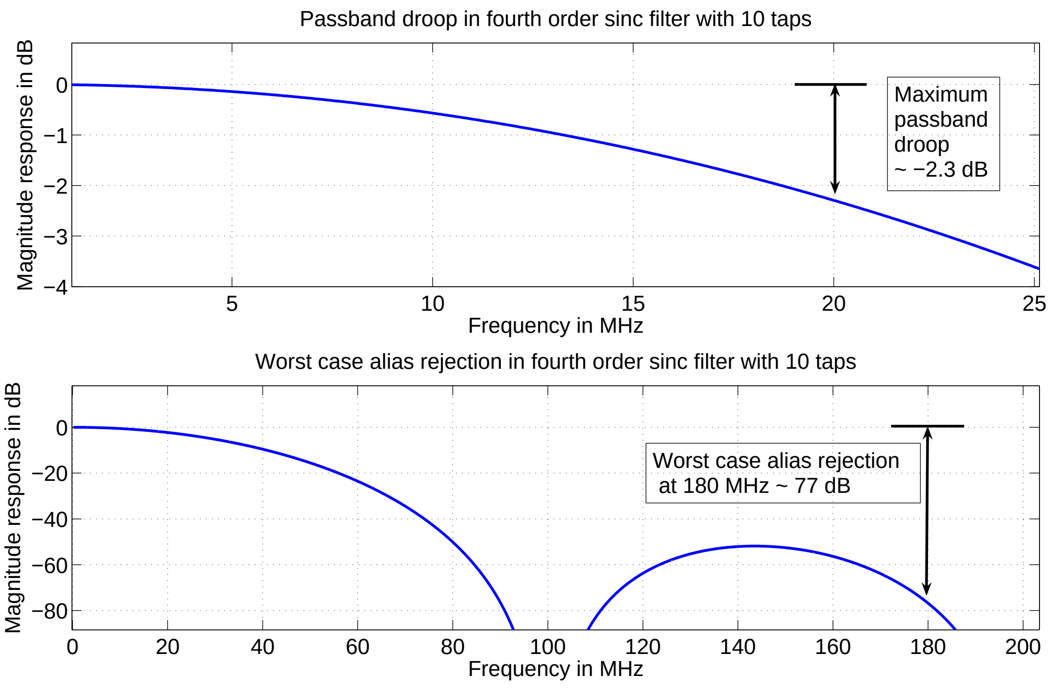


Figure 2.13: Maximum passband droop and worst case alias rejection for sinc filter with $K = 4$, $r = 2$, $M = 5$ given in (2.26).

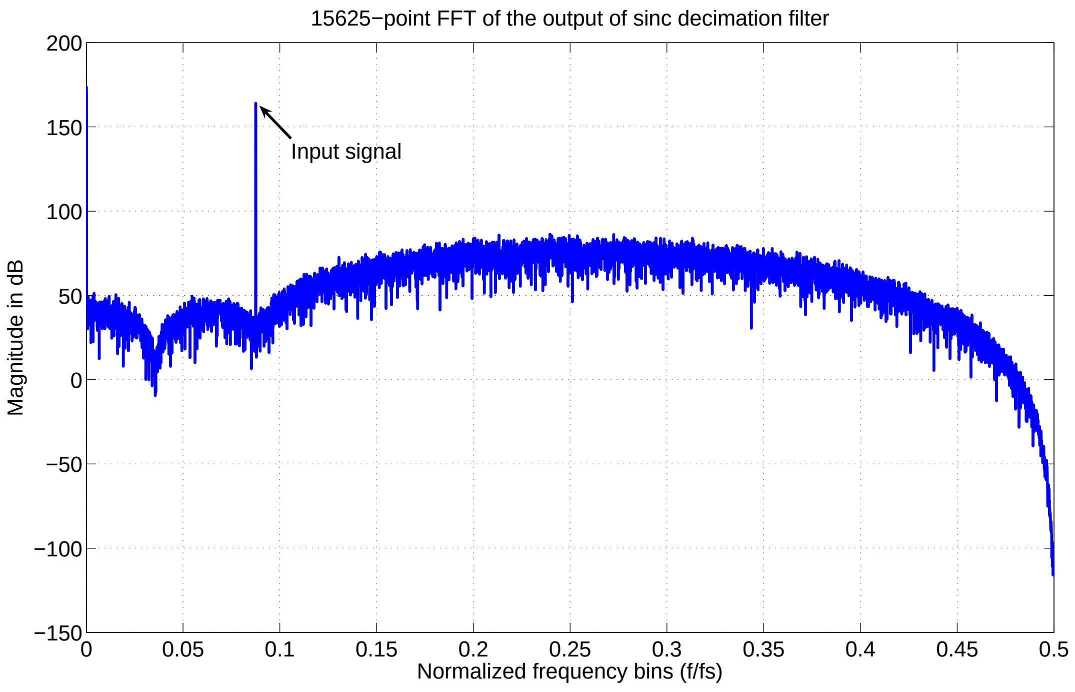


Figure 2.14: 5^6-point FFT of the output of the fourth order 10-tap sinc decimation filter. Normalized frequency 1 after decimation corresponds to 200 MHz. Thus the input signal which is 1367^{th} bin is ≈ 17.5 MHz.

2.5 FIR filter characteristics

The ideal FIR filter response should be as shown in Figure 2.10. It is required that after the second stage FIR filter, the decimation filter should meet the specification of 93 dB SNR at the output. The input of the FIR filter is the output of the first stage sinc filter. This output is derived for an input sinusoidal signal of frequency $\approx$ 17.5 MHz (1367th bin) and is plotted in Figure 2.14. This signal when passed through the stage 2 FIR filter and decimator should give a signal of 93 dB SNR. Considering all this, we can set about to determine the FIR filter coefficients.

2.5.1 Determining the FIR filter coefficients

FIR filter coefficients are determined using Parks-McClellan optimal FIR filter design algorithm. This can be done in MATLAB using two commands *firpmord* and *firpm*. This command uses the Remez exchange algorithm and Chebyshev approximation theory to design filters with an optimal fit between the desired and actual frequency responses. The filters are optimal because the maximum error between the desired frequency response and the actual frequency response is minimized.

Our aim is to use the Parks-McClellan algorithm to obtain an FIR filter $H_{pc}(z)$ of minimum order which meets the SNR requirement. The basic filter parameters used to design the filter $H_{pc}(z)$ are given in Table 2.3. Since the order of an FIR filter increases with lesser passband ripple, steeper transition and higher stopband attenuation, the values in Table 2.3 are found using trial and error so that the order of $H_{pc}(z)$ is as low as possible while still meeting the SNR requirement at the output.

MATLAB commands *firpmord* and *firpm* returns an FIR filter of order 60. The frequency response of the FIR filter, $H_{pc}(z)$ is shown in Figure 2.15 and Figure 2.16.

From Figure 2.16, it can be seen that the stop band attenuation is around

Table 2.3: Parameters for the FIR filter $H_{pc}(z)$

Passband ripple, δ_p	$< 1\,\text{dB}$
Stopband attenuation, δ_s	$50\,\text{dB}$
Passband edge, f_p	$18.25\,\text{MHz}$
Stopband edge, f_s	$24\,\text{MHz}$
Sampling frequency, f_s	$200\,\text{MHz}$

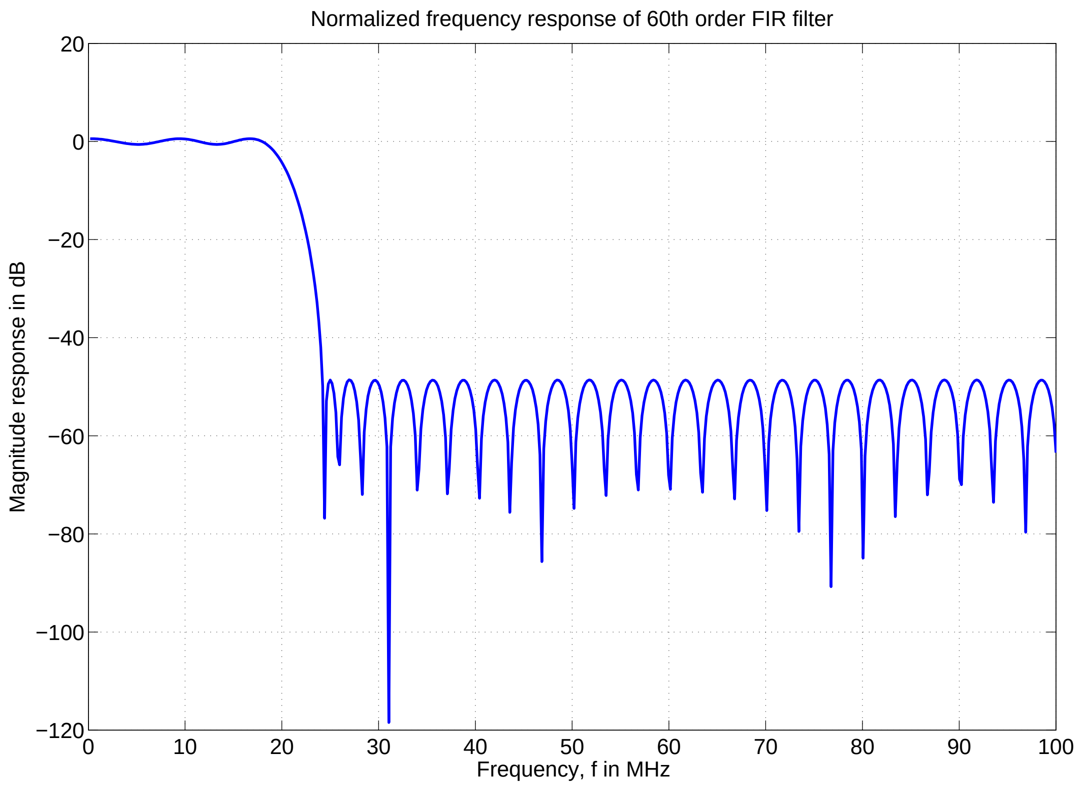


Figure 2.15: Normalized frequency response of the FIR filter $H_{pc}(z)$ obtained using MATLAB.

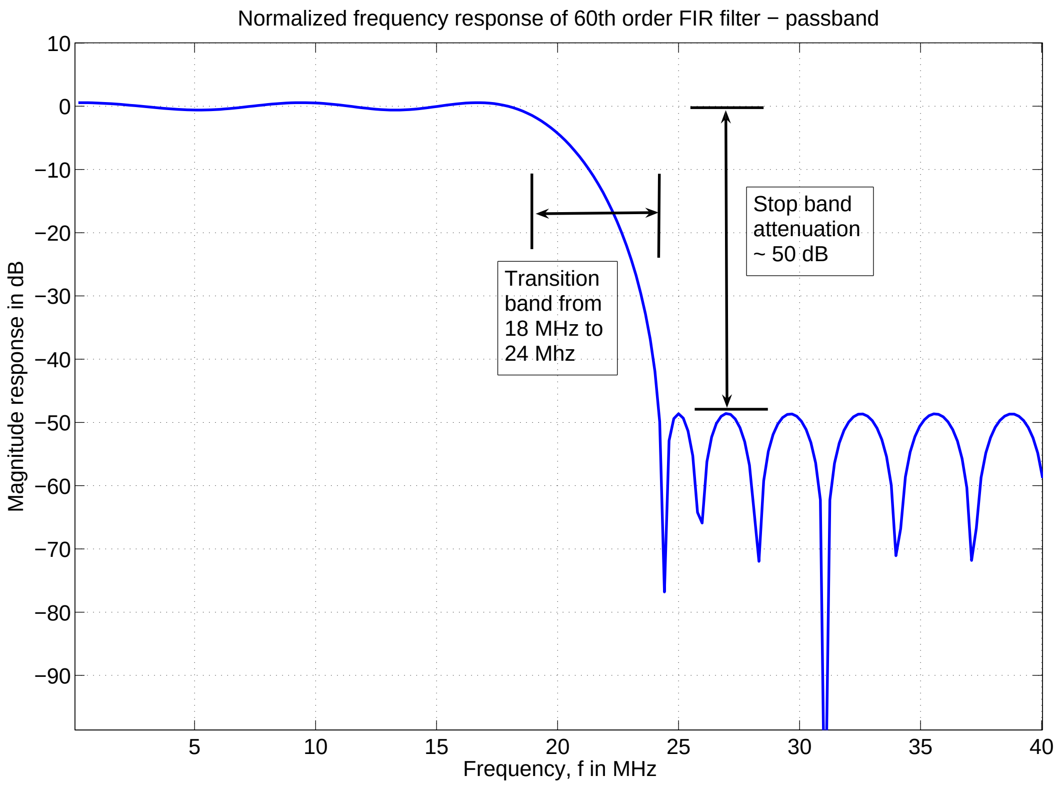


Figure 2.16: Normalized frequency response of the FIR filter $H_{pc}(z)$- a closer look. Note that the stop band attenuation is around 50 dB and the transition band is from 18 MHz to 24 MHz.

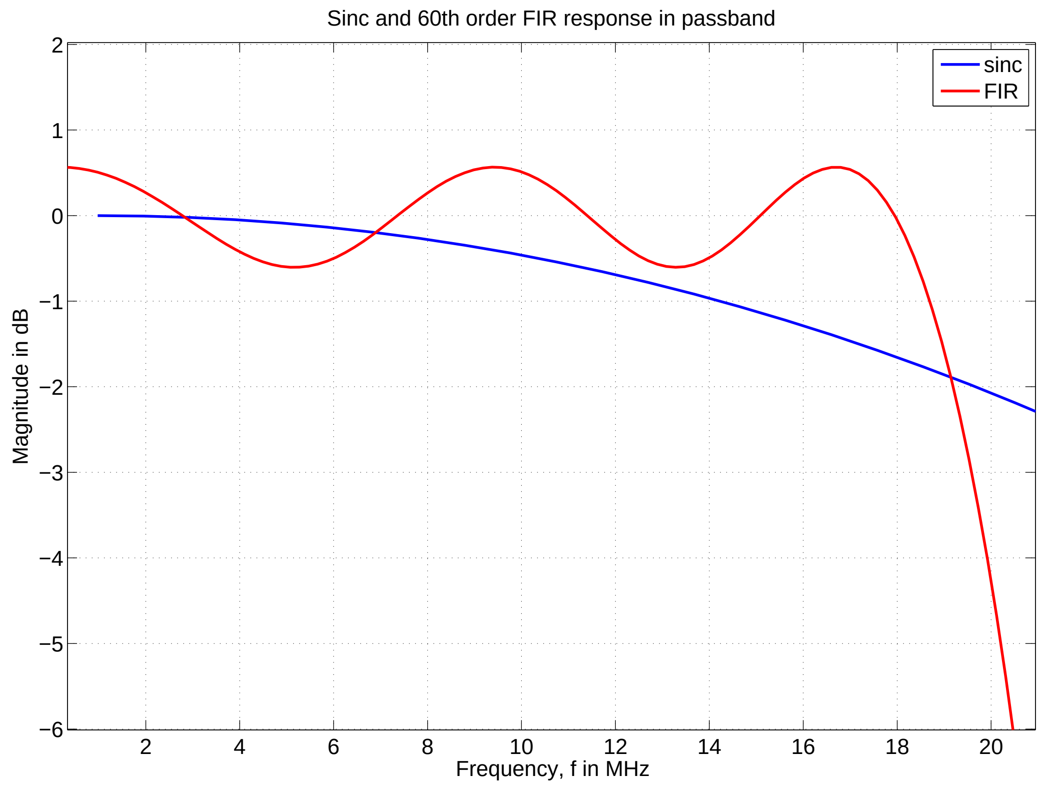


Figure 2.17: Droop of 4^{th} order 10-tap sinc filter and FIR ripple within passband 20 MHz. Note that the FIR filter ripple is within 1 dB.

50 dB, transition band is between 18 MHz and 24 MHz. A closer look of the passband frequency response of both FIR filter in second stage and sinc filter in first stage given in Figure 2.17 reveals that the passband ripple of FIR filter along with the passband droop of the sinc filter deteriorates the response for frequencies closer to 20 MHz.

2.5.2 Sinc and FIR filter frequency responses

To put the entire decimation filter in perspective, the normalized frequency responses of both the FIR filter and the sinc filter are plotted in the same graph in Figure 2.18. Note that FIR filter transition is much more steeper than that of the sinc filter. The output of the decimation filter after the second stage FIR filtering and decimation for an input sinusoidal signal of frequency ≈ 17.5 MHz(1367^{th} bin) is shown in Figure 2.20.

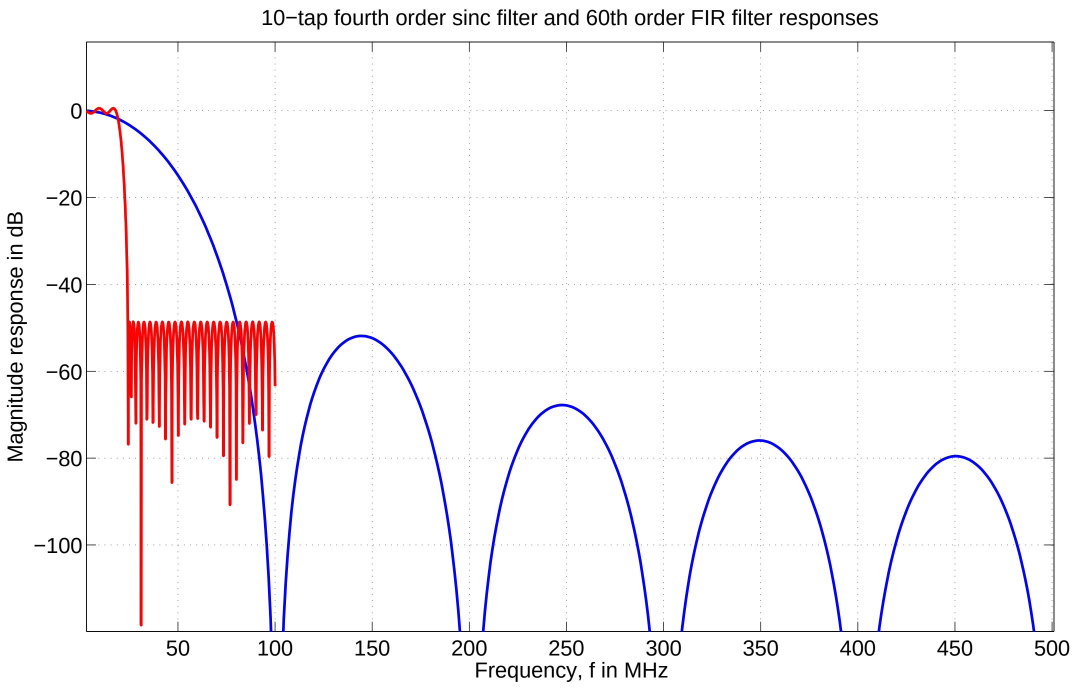


Figure 2.18: Frequency response of the 60^{th} order FIR filter $H_{pc}(z)$ and the 4^{th} order 10-tap sinc filter super imposed. Note that half the sampling rate is 100 MHz and 500 MHz for FIR and sinc respectively.

2.5.3 64th order FIR filter

Note that the sinc filter in (2.26) can be written as

$$H_s(z) = \left(\frac{1}{10}\left(\frac{1-z^{-10}}{1-z^{-1}}\right)\right)^4 \tag{2.28}$$

$$= \left(\frac{1}{10}\left(\frac{(1-z^{-5})(1+z^{-5})}{1-z^{-1}}\right)\right)^4 \tag{2.29}$$

This factorization of $(1-z^{-10})$ into $(1-z^{-5})(1+z^{-5})$ helps us to exploit the cascade equivalence discussed in section 2.1.3 and the factor $(1+z^{-5})^4$ can thus be pushed to the lower frequency side(200 MHz) as shown in Figure 2.19 to become $(1+z^{-1})^4$. This leaves only a scaled fourth order sinc filter with 5 taps operating at 1 GHz.

The 60th order FIR filter obtained using Parks-McClellan algorithm, $H_{pc}(z)$ can be multiplied with the factor $(1+z^{-1})^4$ pushed from the sinc stage. This makes the order of the final FIR filter, $H_f(z)$ in (2.30) 64. This is illustrated in Figure 2.19. The combined frequency response of two stages of the decimation filter is a fourth order sinc filter with 10 taps in cascade with 60th order $H_{pc}(z)$ or equivalently a fourth order sinc filter with 5 taps in cascade with 64th order $H_f(z)$.

$$H_f(z) = H_{pc}(z) \times (1+z^{-1})^4 \tag{2.30}$$

$$= \sum_{k=0}^{k=63} hf_k z^{-k} \tag{2.31}$$

If we choose to implement $(1+z^{-1})^4$ and $H_{pc}(z)$ separately, $(1+z^{-1})^4$ has to operate at 200 MHz, on the other hand, if $H_{pc}(z)$ and $(1+z^{-1})^4$ are clubbed together to form a single filter $H_f(z)$, all the computations can be done at a lower rate of 40 MHz as we will see in the later sections. Thus it is advantageous to implement $(1+z^{-1})^4$ and $H_{pc}(z)$ as a single FIR filter $H_f(z)$.

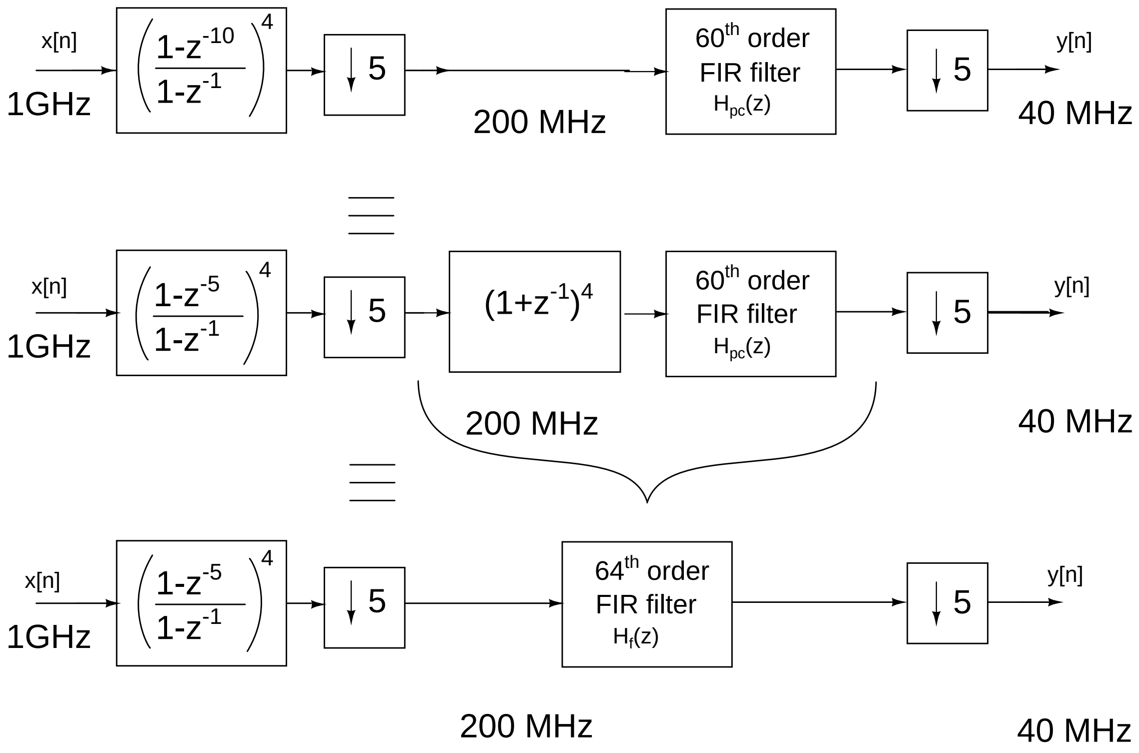

x[n]
1GHz
$\left(\frac{1-z^{-10}}{1-z^{-1}}\right)^4$
↓5
200 MHz
60th order FIR filter $H_{pc}(z)$
↓5
y[n]
40 MHz
x[n]
1GHz
$\left(\frac{1-z^{-5}}{1-z^{-1}}\right)^4$
↓5
$(1+z^{-1})^4$
60th order FIR filter $H_{pc}(z)$
↓5
y[n]
200 MHz
40 MHz
x[n]
1GHz
$\left(\frac{1-z^{-5}}{1-z^{-1}}\right)^4$
↓5
64th order FIR filter $H_f(z)$
↓5
y[n]
200 MHz
40 MHz

Figure 2.19: The factor $(1+z^{-1})^4$ pushed to the low frequency side of the sinc filter is combined with the 60^{th} order FIR filter to get a 64^{th} order FIR filter

2.5.4 Finite precision of the FIR filter coefficients

Since we are implementing the combined 64^{th} order FIR filter $H_f(z)$, the coefficients are obtained by performing the multiplication $H_{pc}(z) \times (1+z^{-1})^4$ shown in (2.30). The 64 coefficients in (2.31) given in Table 2.4 are then scaled such that $|hf_k| \leq 1$. i.e. the scaling factor $F_1 = 1/(\max(|hf_k|))$.

$$hf'_k = hf_k \times F_1 \tag{2.32}$$

$$= hf_k \times \frac{1}{max(|hf_k|)} \tag{2.33}$$

$$= hf_k \times \frac{1}{3.0035} \tag{2.34}$$

Even though MATLAB gives filter coefficients to 64-bit precision, fortunately it is not necessary to do 64-bit multiplications. A finite precision much lower than 64 bits is actually sufficient to meet the SNR requirement. To find out the required number of bits to represent the FIR filter coefficients,

1. Scale the coefficients hf'_k again by a scaling factor $F_2 = 2^n$.
2. Round $(hf'_k \times 2^n)$ to the nearest integer.
3. Calculate SNR.

Repeat this process for $n = 1, 2, \ldots$. After a few iterations it is found that the SNR criterion of the decimation filter is met only if $n \geq 10$. Thus the value of n is chosen to be 10 or in other words, the FIR filter coefficients can be represented in atmost 10 bits. Scaling of the filter coefficients is summarized below.

$$hf''_k = hf'_k \times F_2 \tag{2.35}$$

$$= hf_k \times \frac{2^{10}}{3.0035} \tag{2.36}$$

$$= hf_k \times 340.9366 \tag{2.37}$$

The new scaled coefficients are given in Table 2.4. Note that the division by $\max(|hf_k|)$ while scaling makes sure that the largest FIR filter coefficient after scaling will be the largest possible number that can be represented in 10 bits (i.e. 1024). The FFT of the decimation filter output with 10-bit precision FIR filter coefficients is given in Figure 2.20. SNR calculated from this FFT for an input signal at 1367^{th} bin is 93 dB.

2.6 Architecture

The conventional architecture used to implement the sinc filter in the first stage of a decimation filter is the Cascade Integrator Comb(CIC) or Hogenauer structure. In this section the limitations of CIC implementation at high speeds is discussed. Since the second stage FIR filter works at a lower sampling rate, its architecture is less crucial and a direct form implementation of FIR filters usually suffices.

Table 2.4: Finite precision FIR filter coefficients. Only 32 filter coefficients are shown because of the symmetry $h_k = h_{63-k}$, $k = 0, 1, \dots 63$

hf_k	Value of hf_k	$hf''_k = hf_k \times 340.9366$	round(hf''_k)
hf_0	0.003453128632484	1.1773	1
hf_1	0.017459621385645	5.9526	6
hf_2	0.039133310612196	13.3419	13
hf_3	0.053315357077549	18.1771	18
hf_4	0.049194153841569	16.7720	17
hf_5	0.022826019450836	7.7822	8
hf_6	-0.026726821644420	-9.1121	-9
hf_7	-0.091379259526725	-31.1545	-31
hf_8	-0.152588887407698	-52.0231	-52
hf_9	-0.187513355588177	-63.9302	-64
hf_{10}	-0.177614450723896	-60.5553	-61
hf_{11}	-0.117651625377535	-40.1117	-40
hf_{12}	-0.021138111557846	-7.2068	-7
hf_{13}	0.080997947545488	27.6152	28
hf_{14}	0.149548722242695	50.9866	51
hf_{15}	0.151726477066279	51.7291	52
hf_{16}	0.076271160792871	26.0036	26
hf_{17}	-0.057406812101084	-19.5721	-20
hf_{18}	-0.201111667924139	-68.5663	-69
hf_{19}	-0.292186843342283	-99.6172	-100
hf_{20}	-0.276829023603923	-94.3811	-94
hf_{21}	-0.134665635427954	-45.9124	-46
hf_{22}	0.104952878889345	35.7823	36
hf_{23}	0.362737737201257	123.6706	124
hf_{24}	0.528036189800425	180.0269	180
hf_{25}	0.492153423628284	167.7931	168
hf_{26}	0.186382819150577	63.5447	64
hf_{27}	-0.388185540468529	-132.3467	-132
hf_{28}	-1.150402802168458	-392.2144	-392
hf_{29}	-1.953771295922935	-666.1122	-666
hf_{30}	-2.623248549904323	-894.3615	-894
hf_{31}	-3.003490924493843	-1024.0000	-1024

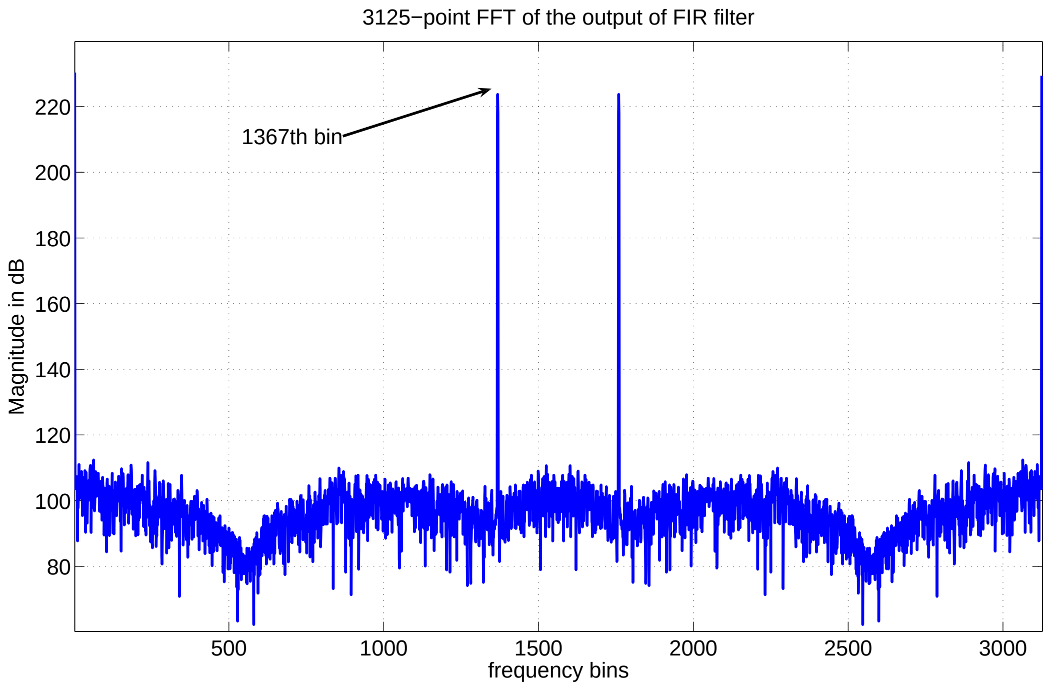


Figure 2.20: FFT of the output signal after FIR filter and decimator

2.6.1 Hogenauer structure for sinc filter

Consider the transfer function of the sinc filter obtained in (2.26). It can be decomposed as shown in Figure 2.21 into accumulators(or integrators) $1/(1-z^{-1})$ and differentiators(or comb filters) $(1-z^{-2})$. The accumulator $1/(1-z^{-1})$ can be pipelined and retimed to get $z^{-1}/(1-z^{-1})$. This leads to the classic Hogenauer structure[6] or CIC(Cascaded Integrator Comb) structure of the sinc filter as shown in Figure 2.22.

The popularity of the sinc filter as the first stage of the decimation filter has much to do with this elegant architecture. As you can see, the whole filter can be implemented using two repetitive blocks. Accumulators and differentiators. The minimum width of the accumulator adder, B_A, so that there is no loss of information at the output is calculated using the formula[6]

$$B_A = B_{in} + K \log_2 M \tag{2.38}$$

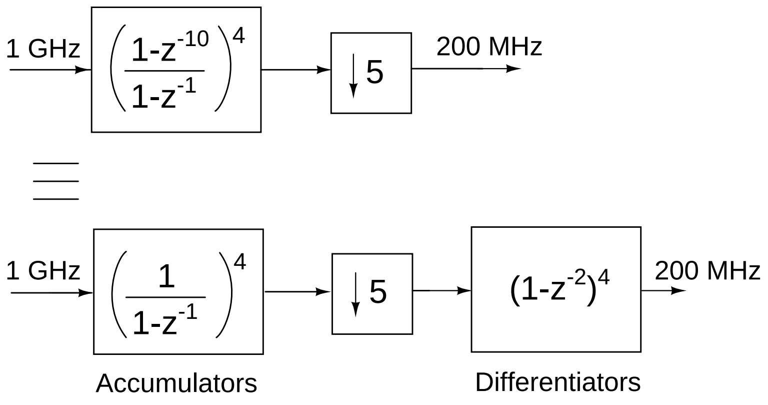

1 GHz
$\left(\frac{1-z^{-10}}{1-z^{-1}}\right)^4$
5
200 MHz
1 GHz
$\left(\frac{1}{1-z^{-1}}\right)^4$
5
$(1-z^{-2})^4$
200 MHz
Accumulators
Differentiators

Figure 2.21: Decomposition of sinc decimation filter which leads to CIC implementation. Note that there is a cascade of 4 accumulators $1/(1-z^{-1})$ at 1 GHz and a cascade of 4 differentiators $(1-z^{-2})$ at 200 MHz.

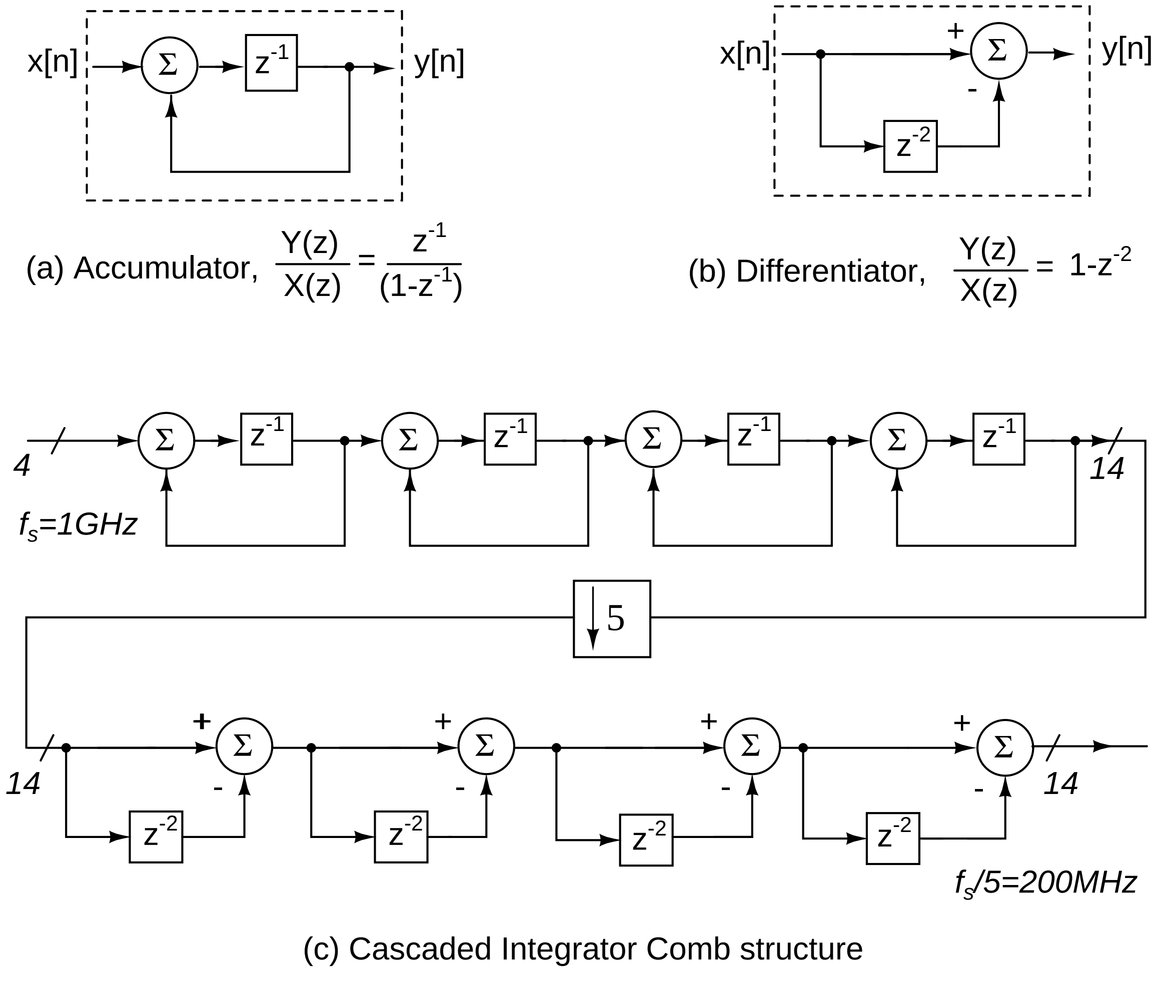

x[n]
Σ
z^{-1}
y[n]
(a) Accumulator, $\frac{Y(z)}{X(z)} = \frac{z^{-1}}{(1-z^{-1})}$
x[n]
+
-
Σ
z^{-2}
y[n]
(b) Differentiator, $\frac{Y(z)}{X(z)} = 1-z^{-2}$
4
f_s=1GHz
z^{-1}
14
5
14
+
-
z^{-2}
14
$f_s/5$=200MHz
(c) Cascaded Integrator Comb structure

Figure 2.22: Sinc decimation in Hogenauer or CIC architecture

where B_{in} is the number of bits at the input, K the order of the sinc filter and M, the decimation factor of sinc filter. Substituting the values, $B_{in} = 4, M = 5, K = 4$, we get $B_A = 14$.

From the CIC structure, it is clear that the speed at which the sinc filter can work is limited by the adder in the accumulator block. The fastest N-bit adder architecture has its delay proportional to $log_2 N$. (Carry look ahead adder is not practical for N>4). So for high sampling rates, we can hardly go for accumulators implemented using conventional N-bit adders. An architectural technique suitable for high speed decimation filters called the polyphase decomposition is discussed in the next section.

2.7 Decimation filter using polyphase decomposition

In section 2.1.3, the cascade equivalence property of decimation filters is proven. The property literally means that a filter $H(z^M)$ before a decimator with decimation factor M is equivalent to a filter $H(z)$ if it comes after the decimator. This property is significant due to the fact that a filter after decimator works at a lower frequency and thus has less stringent timing constraints and lower power consumption. A practical application of the cascade equivalence can be best seen in polyphase decomposition which will be discussed in this section.

2.7.1 Polyphase decomposition of sinc filter

The transfer function of the sinc filter with 10 taps obtained in (2.26) can be expanded as

$$\begin{aligned} H_s(z) &= \left(\frac{1-z^{10}}{1-z^{-1}}\right)^4 && (2.39)\\ &= \left(\frac{(1-z^{-5})(1+z^{-5})}{1-z^{-1}}\right)^4 && (2.40)\\ &= (1+z^{-1}+z^{-2}+z^{-3}+z^{-4})^4(1+z^{-5})^4 && (2.41)\\ &= H_1(z)(1+z^{-5})^4 && (2.42)\end{aligned}$$

As we have seen, the factor $(1+z^{-5})^4$ can be pushed to the low frequency side of the decimator to give $(1+z^{-1})^4$ (Figure 2.19) which is later be combined with the FIR filter in the second stage of the decimation filter. By moving even a part of the sinc filter computation to the low frequency side, we can considerably decrease power consumption. Now the transfer function to be implemented in the first stage (at 1 GHz) is only $H_1(z)$. This can be expanded as shown below.

$$H_1(z) = (1+z^{-1}+z^{-2}+z^{-3}+z^{-4})^4 = \sum_{n=0}^{16} h(n)z^{-n} \qquad (2.43)$$

$$\begin{aligned} &= 1+4z^{-1}+10z^{-2}+20z^{-3}+35z^{-4}+52z^{-5}+68z^{-6}+80z^{-7}+85z^{-8}+80z^{-9}\\ &\quad +68z^{-10}+52z^{-11}+35z^{-12}+20z^{-13}+10z^{-14}+4z^{-15}+z^{-16} \qquad (2.44)\end{aligned}$$

Which can be written as

$$\begin{aligned} H_1(z) = \quad & (1+52z^{-5}+68z^{-10}+4z^{-15})\\ &+z^{-1}(4+68z^{-5}+52z^{-10}+z^{-15})\\ &+z^{-2}(10+80z^{-5}+35z^{-10})\\ &+z^{-3}(20+85z^{-5}+20z^{-10})\\ &+z^{-4}(35+80z^{-5}+10z^{-10}) \qquad (2.45)\end{aligned}$$

That is

$$H_1(z) = E_0(z^5) + z^{-1}E_1(z^5) + z^{-2}E_2(z^5) + z^{-3}E_4(z^5) + z^{-4}E_4(z^5) \quad (2.46)$$

where

$$E_0(z) = (1 + 52z^{-1} + 68z^{-2} + 4z^{-3}) \quad (2.47)$$

$$E_1(z) = (4 + 68z^{-1} + 52z^{-2} + z^{-3}) \quad (2.48)$$

$$E_2(z) = (10 + 80z^{-1} + 35z^{-2}) \quad (2.49)$$

$$E_3(z) = (20 + 85z^{-1} + 20z^{-2}) \quad (2.50)$$

$$E_4(z) = (35 + 80z^{-1} + 10z^{-2}) \quad (2.51)$$

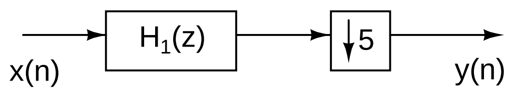
The above decomposition of the sinc filter $H_1(z)$ into $E_0(z)$ to $E_4(z)$ is called polyphase decomposition (Figure 2.23 and Figure 2.24). Note that all the multiplications with filter coefficients now need to be carried out only at 200 MHz. This greatly relaxes the timing constraints and lowers the power consumption. The actual circuit implementation of this architecture is described in detail in the next chapter.

2.7.2 Polyphase decomposition of the FIR filter

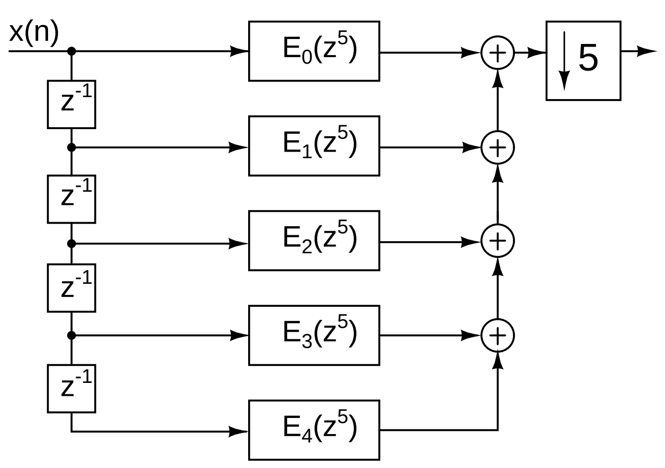
The transfer function of the 64^{th} order FIR filter obtained in Table 2.4 with 10-bit precision coefficients can be written as

$$H_f(z) = \sum_{k=0}^{k=63} h_k z^{-k} \quad (2.52)$$

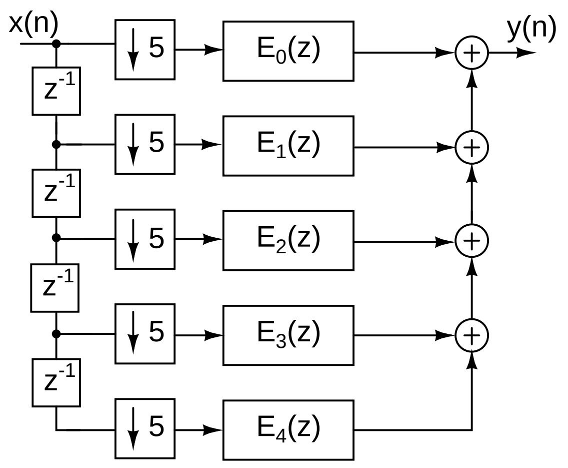
where h_k, k = 0,1..63 are the coefficients. Just like in the case of sinc filter, the FIR filter can also be split into five filters as shown in (2.53). Figure 2.25 and Figure 2.26 illustrate the decomposition. Note that this is quite similar to the



(a) Sinc decimator



(b) Polyphase decomposition of $H_1(z)$



(c) Calculations being done at a lower frequency

Figure 2.23: Polyphase decomposition of sinc filter

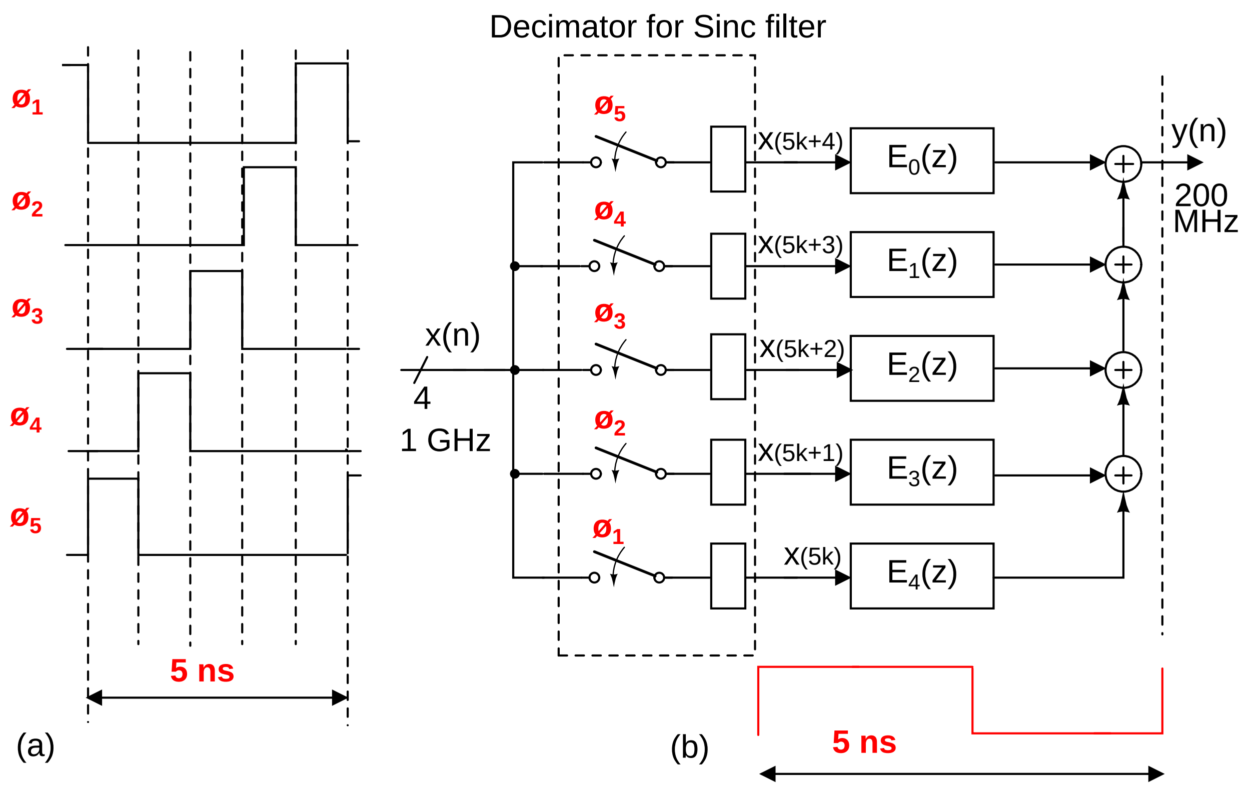


Figure 2.24: Polyphase decomposition of sinc filter. The filter is split into a decimator and five FIR filters

Table 2.5: $G_k(z)$ definitions

$G_k(z)$	Definition
$G_0(z)$	$h_0 + h_5 z^{-1} + h_{10} z^{-2} + h_{15} z^{-3} + h_{20} z^{-4} + h_{25} z^{-5} + h_{30} z^{-6} + h_{28} z^{-7} + h_{23} z^{-8} + h_{18} z^{-9} + h_{13} z^{-10} + h_8 z^{-11} + h_3 z^{-12}$
$G_1(z)$	$h_1 + h_6 z^{-1} + h_{11} z^{-2} + h_{16} z^{-3} + h_{21} z^{-4} + h_{26} z^{-5} + h_{31} z^{-6} + h_{27} z^{-7} + h_{22} z^{-8} + h_{17} z^{-9} + h_{12} z^{-10} + h_7 z^{-11} + h_2 z^{-12}$
$G_2(z)$	$h_2 + h_7 z^{-1} + h_{12} z^{-2} + h_{17} z^{-3} + h_{22} z^{-4} + h_{27} z^{-5} + h_{31} z^{-6} + h_{26} z^{-7} + h_{21} z^{-8} + h_{16} z^{-9} + h_{11} z^{-10} + h_6 z^{-11} + h_1 z^{-12}$
$G_3(z)$	$h_3 + h_8 z^{-1} + h_{13} z^{-2} + h_{18} z^{-3} + h_{23} z^{-4} + h_{28} z^{-5} + h_{30} z^{-6} + h_{25} z^{-7} + h_{20 z^{-8}} + h_{15 z^{-9}} + h_{10} z^{-10} + h_5 z^{-11} + h_0 z^{-12}$
$G_4(z)$	$h_4 + h_9 z^{-1} + h_{14} z^{-2} + h_{19} z^{-3} + h_{24} z^{-4} + h_{29} z^{-5} + h_{29} + z^{-6} h_{24} z^{-7} + h_{19} z^{-8} + h_{14} z^{-9} + h_9 z^{-10} + h_4 z^{-11}$

decomposition done for sinc filter case.

$$H_f(z) = G_0(z^5) + z^{-1}G_1(z^5) + z^{-2}G_2(z^5) + z^{-3}G_4(z^5) + z^{-4}G_4(z^5) \qquad (2.53)$$

The forms of $G_k(z)$ are given in Table 2.5. The polyphase decomposition of the FIR filter $H_f(z)$ into $G_0(z)$ to $G_4(z)$ facilitates computations to be carried out in a lower frequency (40 MHz) instead of 200 MHz (Figure 2.25, Figure 2.26). It is assumed that the five streams of 14-bit data $y(5k)$ to $y(5k + 4)$ will appear at the positive edge of a 25 ns clock and will stay available throughout the period.

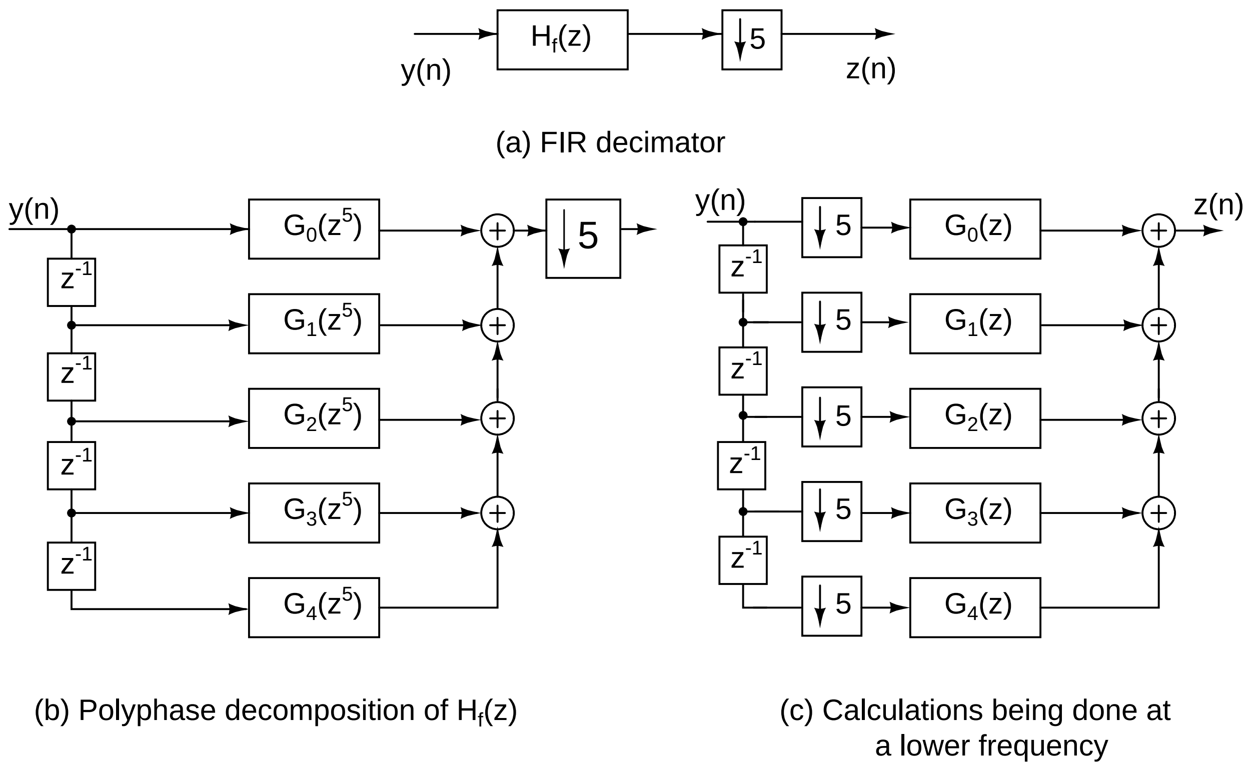


Figure 2.25: Polyphase decomposition of FIR the filter

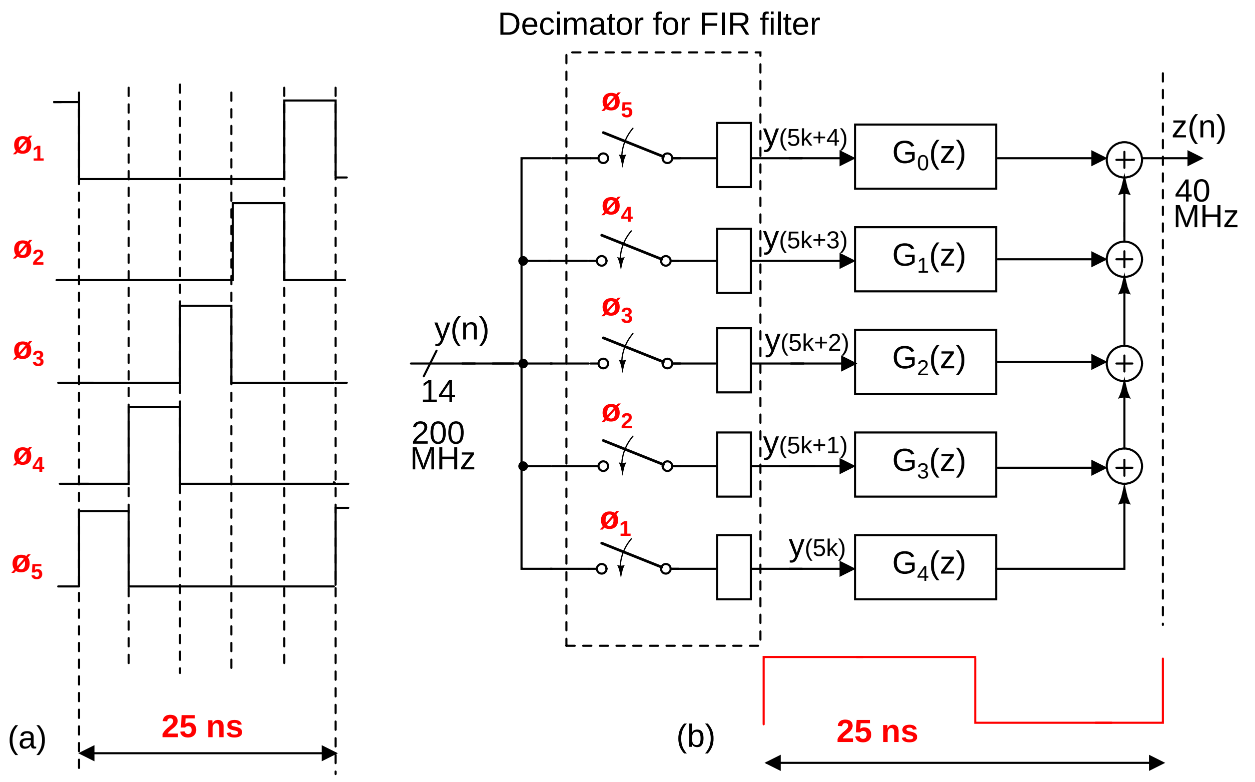


Figure 2.26: Polyphase decomposition of the FIR filter

CHAPTER 3

Implementation of the Sinc Filter

In the earlier chapter, a two stage design of a decimation filter that meets the SNR requirement is derived. It is known that the input to the decimation filter from the $\Delta\Sigma$ modulator is a 4-bit digital signal. This requires all the filtering and decimation to be done in digital domain, making the entire decimation filter a digital circuit. Polyphase decomposition is used so that the circuit has to work only at the decimated rate thereby greatly relaxing the timing constraints. By decreasing the maximum frequency of operation to 200 MHz, the whole digital circuit can now be implemented using $0.18\,\mu$m standard cell library.

The challenge hereafter is to implement the decimation filter as efficiently as possible so that the power consumption and area are minimum. The digital design is modelled using the Hardware Description Language (HDL), *Verilog* and is synthesized and routed using *Synopsis Design Vision* and *Cadence Encounter* respectively. In this chapter, the circuit details of the implementation of the decimation filter in polyphase architecture are given.

3.1 Implementation of decomposed filters

From the Figure 2.24, it is clear that the sinc filter implementation using polyphase decomposition involves the design of a decimator with decimation factor 5 and five FIR filters, $E_0(z)$ to $E_4(z)$. The design of the decimator is discussed in detail in a later chapter. For now, assume that the inputs to the five FIR filters, $x(5k)$ to $x(5k+4)$ are available at the positive edge of a 5 ns clock (200 MHz). Also assume that all these five inputs stay available for one full time period (5 ns).

3.1.1 $E_0(z)$ and $E_1(z)$

Recall that the transfer functions of $E_0(z)$ and $E_1(z)$ are

$$E_0(z) = (1 + 52z^{-1} + 68z^{-2} + 4z^{-3}) \tag{3.1}$$

$$E_1(z) = (4 + 68z^{-1} + 52z^{-2} + z^{-3}) \tag{3.2}$$

From (3.1) and (3.2), it can be seen that the usual symmetry of the FIR filter coefficients that gives a linear phase response is absent in $E_0(z)$ to $E_4(z)$. So it may appear that in polyphase decomposition, the number of multiplications with filter coefficients doubles, since the symmetry of the coefficients is lost in each filter. But note that even though each filter has asymmetric coefficients, the coefficients are symmetric between pairs of two filters. For example, note that the filter coefficients of $E_0(z)$ and $E_1(z)$ given in (3.1) and (3.2) are the same and are mirror images of each other. Making use of this symmetry between filters, two FIR filters $E_0(z)$ and $E_1(z)$ can be implemented together as shown in Figure 3.1[2, p. 433]. The outputs of $E_0(z)$ and $E_1(z)$ have to be added later.

3.1.2 $E_2(z)$ and $E_4(z)$

The z-domain transfer functions of $E_2(z)$ and $E_4(z)$ are

$$E_2(z) \quad = (10 + 80z^{-1} + 35z^{-2}) \tag{3.3}$$

$$E_4(z) \quad = (35 + 80z^{-1} + 10z^{-2}) \tag{3.4}$$

Similar to $E_0(z)$ and $E_1(z)$, the symmetry of filter coefficients between $E_2(z)$ and $E_4(z)$ can also be exploited to get an efficient implementation as shown in Figure 3.2.

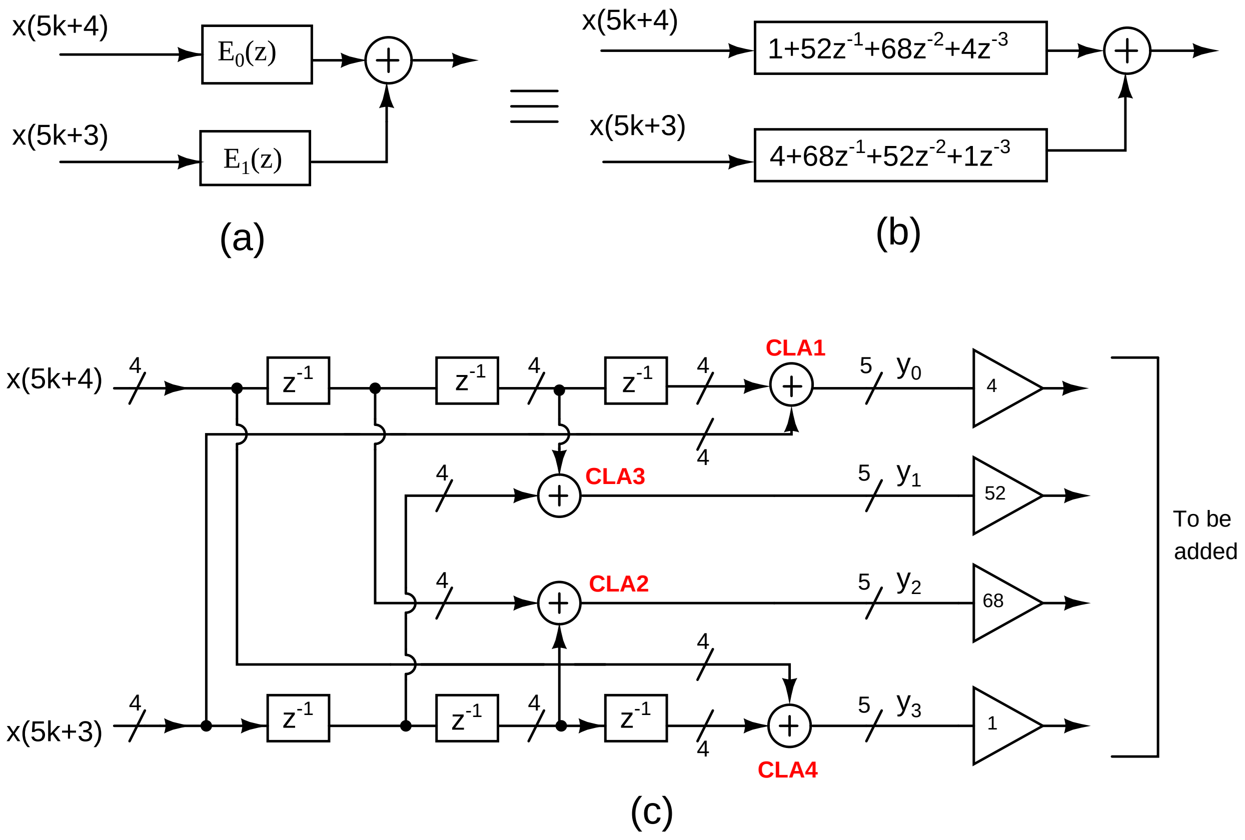


Figure 3.1: The combined circuit implementation of $E_0(z)$ and $E_1(z)$. The terms y_0 to y_3 get multiplied with the filter coefficients to form partial products which have to be added later.

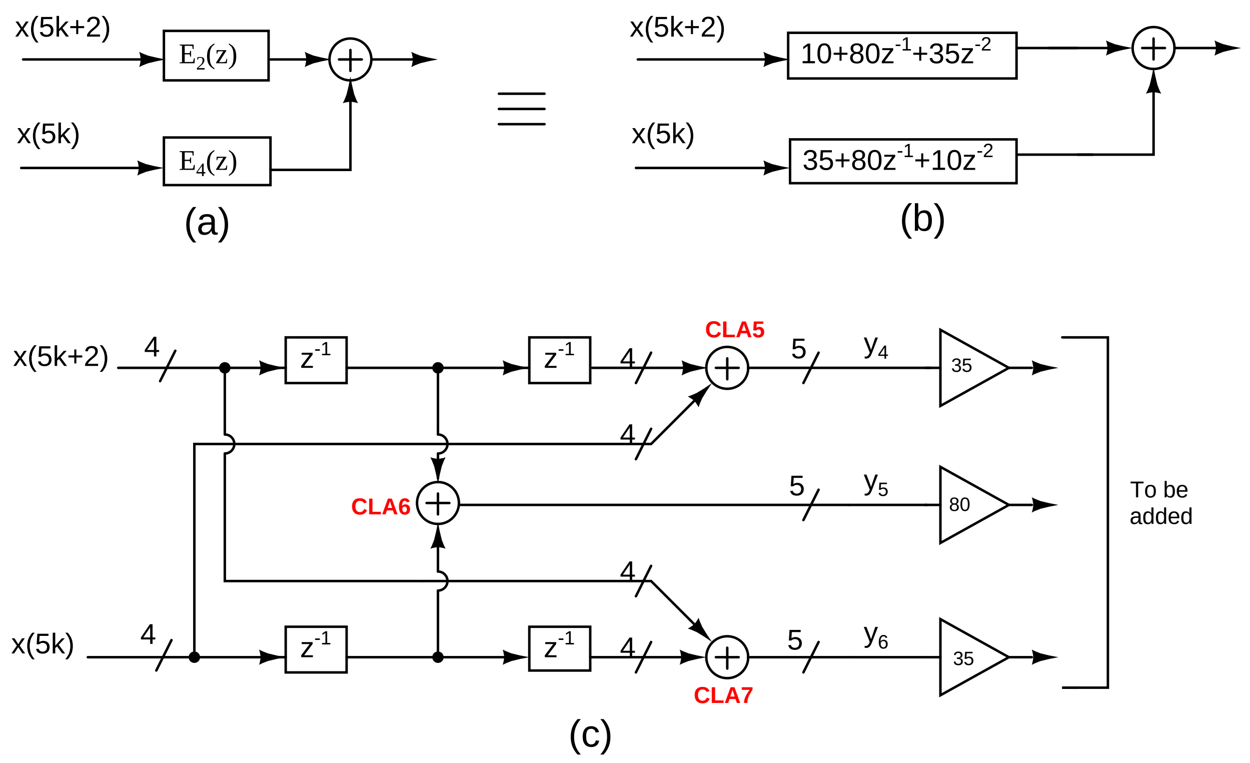


Figure 3.2: Combined circuit implementation of $E_2(z)$ and $E_4(z)$. The terms y_4 to y_6 get multiplied with the filter coefficients to form the partial products which have to added later.

3.1.3 $E_3(z)$

From the transfer function of $E_3(z)$ given in (3.5), it can be seen that the filter $E_3(z)$ is symmetric with a linear phase. The direct form implementation of $E_3(z)$ is shown in Figure 3.3.

$$E_3(z) = (20 + 85z^{-1} + 20z^{-2}) \tag{3.5}$$

From Figures 3.1, 3.2 and 3.3, it can be seen that inputs $x(5k)$ to $x(5k+4)$ are shifted and added using eight standard 4-bit carry look ahead adders (CLA1 to CLA8) taking care to exploit the symmetry of coefficients between and within filters. The sums y_0 to y_8 are multiplied with the filter coefficients and should be added to get the final output of the sinc decimator. Since multiplications are nothing but additions of partial products, all we need to do is to generate all the partial products and add them up.

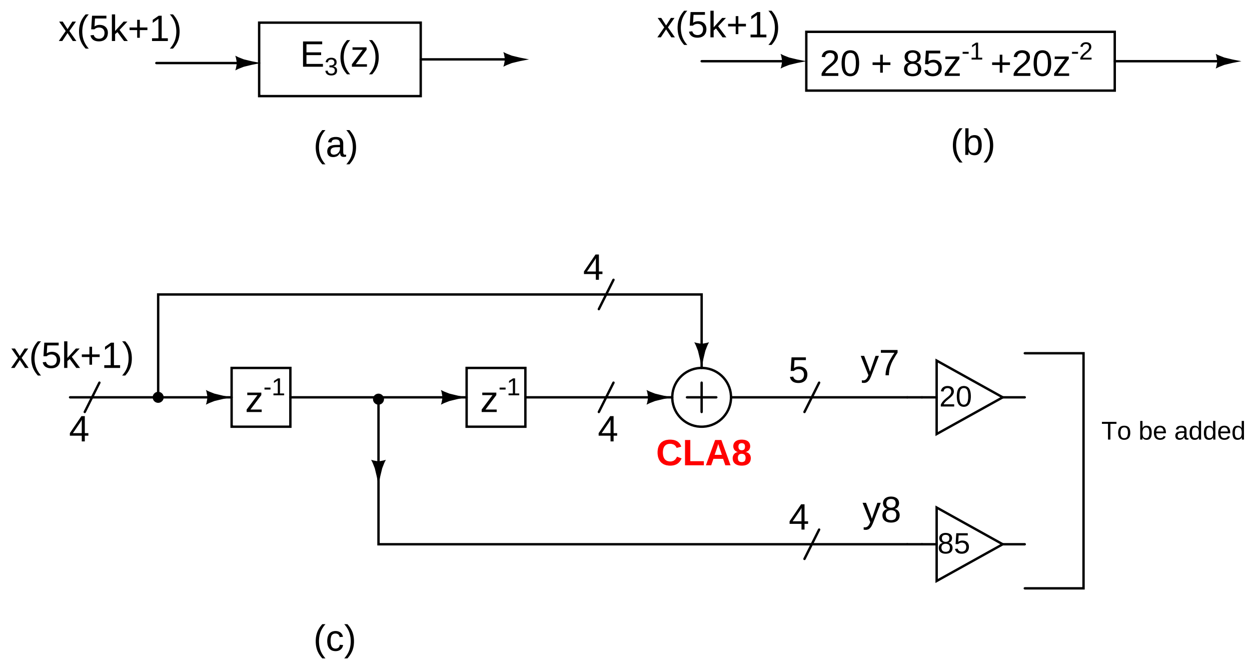


Figure 3.3: Circuit implementation of $E_3(z)$. The terms y_7 and y_8 get multiplied with the filter coefficients to form the partial products which have to be added later.

3.2 Partial product generation

In the previous section, it is seen that to get the final output from the sinc filter, nine multiplications of the eight 5-bit numbers y_0 to y_7 and one 4-bit number y_8 with different filter coefficients need to be done and all the products have to be added. Performing multiplication with each sinc filter coefficient separately and adding the products at the end may not be the best way since some of the multiplications can take longer time than the other. Faster multiplications finish sooner and stay idle till the more time consuming multiplications get over.

Since the whole sinc filter has to work at a speed of 200 MHz which gives a critical path delay (CPD) of 5 ns, we may not be able to afford the simple design where nine multiplications with y_0 to y_8 are carried out individually and addition of products is done at the end. Thus all the partial products of multiplication are generated at the beginning itself to add it together in a wallace tree formation[8], [9], [10].

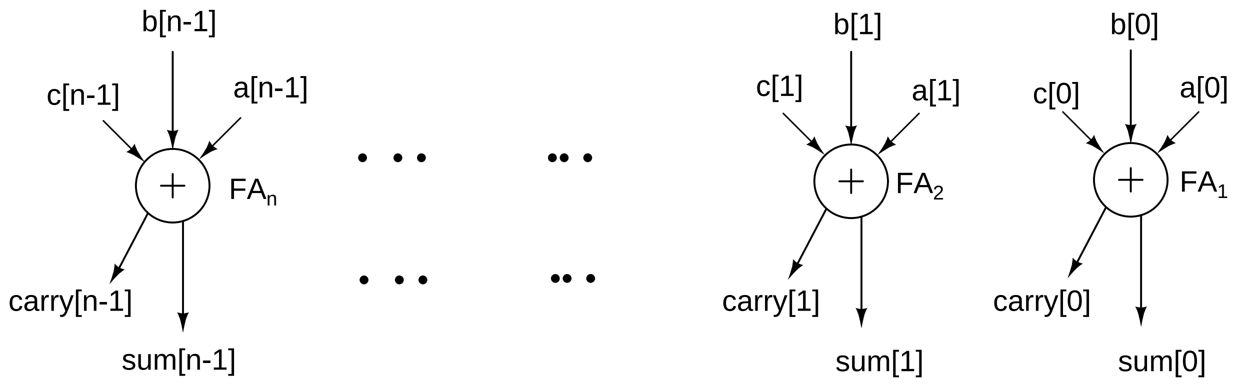
All sinc filter coefficients are integers and hence each of them can be written as a sum of powers of 2 as shown in Table 3.1. In binary representation, a multiplication by two is simply padding a zero to the LSB or equivalently left shifting by one. Thus all the partial products essentially are the zero-padded (or left shifted) versions of y_0, $y_1 \ldots y_8$. The twenty partial products generated by the multiplication of the shifted and added inputs y_0 to y_8 with sinc filter coefficients are given in Table 3.1. The sum of these partial products form the final output of the sinc filter.

3.3 Wallace tree addition

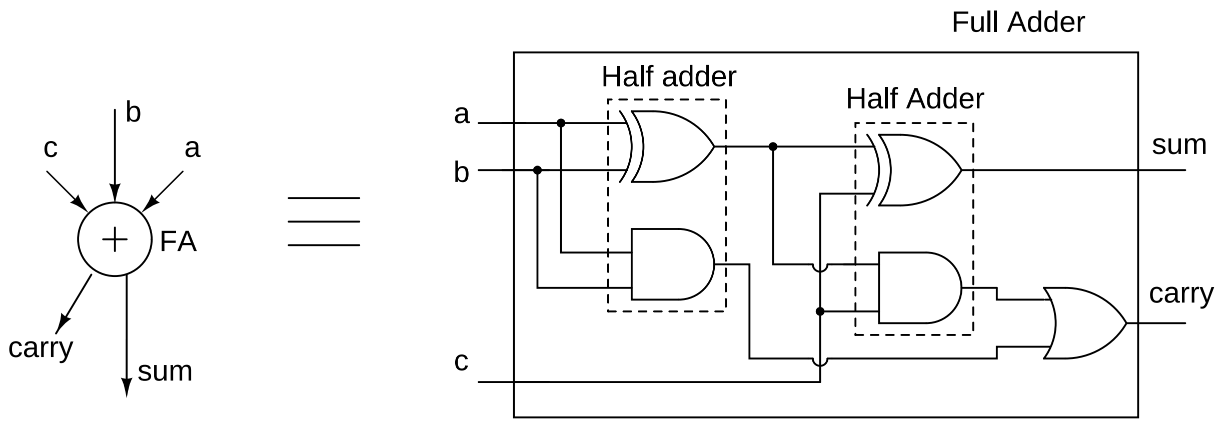
We cannot hope to meet the timing constraint if we use conventional n-bit adders to add up the twenty partial products pp_1 to pp_{20}. A suitable method to add up a large number of vectors is to use the Wallace tree structure[9].

Table 3.1: Partial products of the sinc filter to be added. y_0 to y_7 are the 5-bit outputs of the eight CLAs. y_8 is a 4-bit number.

Shifted and added inputs y_k	Sinc filter coefficient to which y_k is to be multiplied	Partial products
y_0	$4 = 2^2$	$pp_1 = 2^2 y_0$
y_1	$52 = 2^5 + 2^4 + 2^2$	$pp_2 = 2^5 y_1$, $pp_3 = 2^4 y_1$, $pp_4 = 2^2 y_1$
y_2	$68 = 2^6 + 2^2$	$pp_5 = 2^6 y_2$, $pp_6 = 2^2 y_2$
y_3	$1 = 2^0$	$pp_7 = y_3$
y_4	$35 = 2^5 + 2 + 1$	$pp_8 = 2^5 y_4$, $pp_9 = 2y_4$, $pp_{10} = y_4$
y_5	$80 = 2^6 + 2^4$	$pp_{11} = 2^6 y_5$, $pp_{12} = 2^4 y_5$
y_6	$10 = 2^3 + 2$	$pp_{13} = 2^3 y_6$, $pp_{14} = 2y_6$
y_7	$20 = 2^4 + 2^2$	$pp_{15} = 2^4 y_7$, $pp_{16} = 2^2 y_7$
y_8	$85 = 2^6 + 2^4 + 2^2 + 1$	$pp_{17} = 2^6 y_8$, $pp_{18} = 2^4 y_8$, $pp_{19} = 2^2 y_8$, $pp_{20} = y_8$



(a) n-bit Wallace Adder



(b) A full adder

Figure 3.4: Circuit implementation of an n-bit wallace adder. Three input vectors $\bar{a}$, $\bar{b}$ and $\bar{c}$ can be added in 1 full adder delay to get a sum and carry vector.

An n-bit Wallace adder is simply an array of n full adders as shown in Figure 3.4. In Figure 3.4, $\bar{a}$, $\bar{b}$ and $\bar{c}$ are three n-bit numbers (partial products) to be added. By giving the k^{th} bit of $\bar{a}$, $\bar{b}$ and $\bar{c}$ to the k^{th} full adder, we can generate an array of sums and carrys each of length n, denoted by $\overline{sum}$ and $\overline{carry}$. The actual sum of the three input vectors

$$\bar{a} + \bar{b} + \bar{c} = 2\overline{carry} + \overline{sum} \tag{3.6}$$

Note that $\overline{carry}$ is to be shifted by one bit before adding to $\overline{sum}$ to get the final sum thus making the maximum length of the final vector n+2.

Thus a wallace adder can be used to add three n-bit vectors to generate a sum vector and carry vector. Like a ripple carry adder (RCA), wallace adder also decreases the number of addends by one. The difference is that if a wallace adder takes only one full adder delay (irrespective of n) to reduce three addends to two, a RCA needs a delay proportional to n to reduce two addends to one final sum vector. This is illustrated in Figure 3.5.

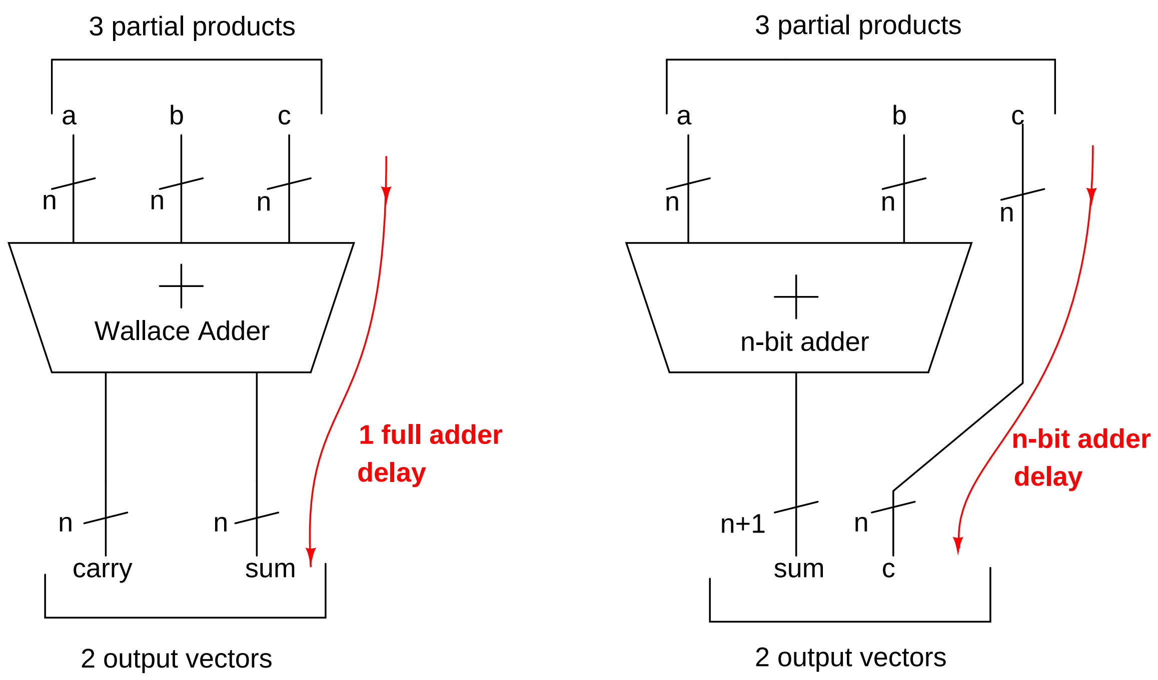


Figure 3.5: Comparison of wallace adder with an n-bit RCA in adding up partial products.

3.3.1 Reduction of the number of partial products

From Table 3.1, it is seen that there are 20 partial products to be added (pp_1 to pp_{20}) to form the final sinc filter output. A careful observation of these partial products helps us to reduce their number. This is possible because certain additions of partial products can be done without actually using any adders. For

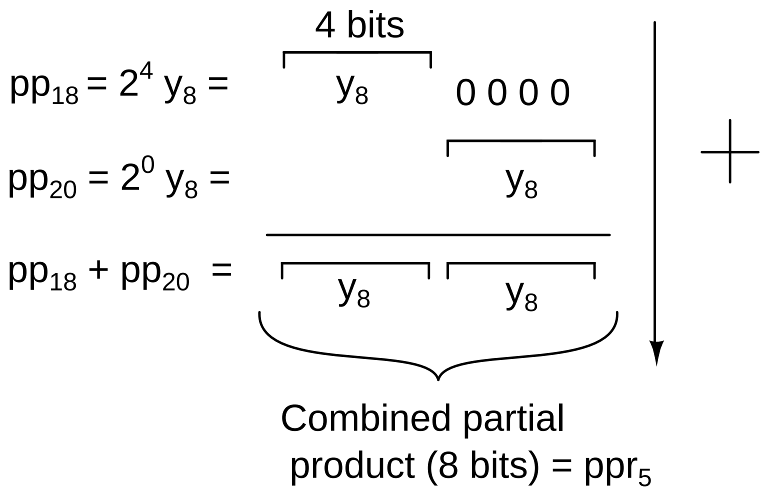


Figure 3.6: Reduction of partial products with no hardware. Partial products $pp_{18} = 2^4 y_8$ and $pp_{20} = y_8$ are added by simply placing y_8 in the last four bit positions of $2^4 y_8$.

example, consider the partial products pp_{18} and pp_{20}. $pp_{18} = 2^4 y_8$ which means the least significant four bits of pp_{18} are 0. $pp_{20} = y_8$ on the other hand is simply a 4-bit number. Addition of pp_{18} and pp_{20} is simply placing the four bits of pp_{20} in the least significant four bits of pp_{18}. Clearly this does not require any hardware, but only appropriate wiring as shown in Figure 3.6.

This grouping of partial products that can be added without any hardware results in a lower number of vectors to be added. This in turn results in a less complex wallace tree (lesser number of stages in the tree) and better performance. Similarly pp_{17} and pp_{19} can be combined. Also pp_{11} and pp_{14}, pp_8 and pp_{10}, pp_5 and pp_9, pp_2 and pp_7 form combined partial products. As shown in Figure 3.7, now we have only fourteen partial products to be added in the tree instead of the earlier twenty and this can be done in just 6 stages of wallace adders. The number of partial products can be reduced from fourteen to two in six full adder delays.

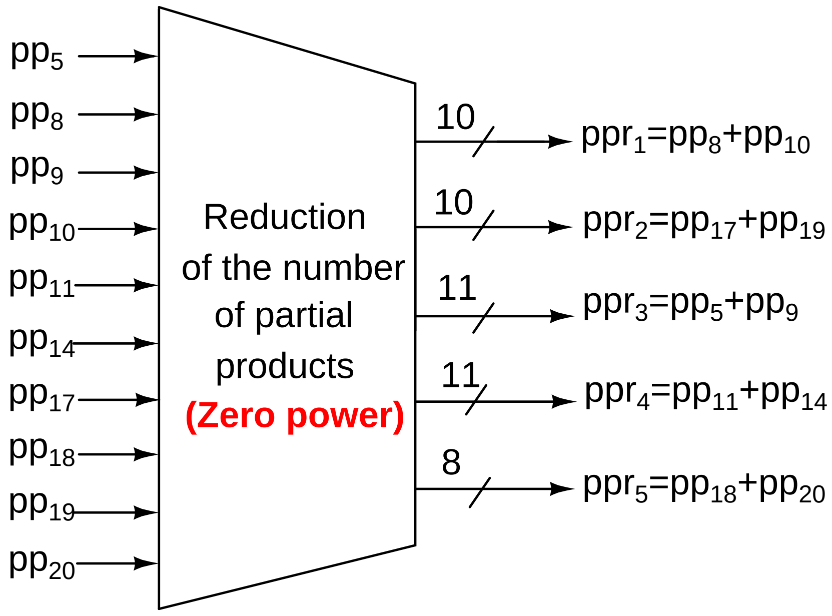


Figure 3.7: Ten out of twenty original partial products are reduced to five reduced partial products ppr_1 to ppr_5 by simple rewiring.

The complete wallace tree[9], [10] implementing this is shown in Figure 3.8.

3.3.2 Minimizing the number of adders

Apart from minimizing the number of partial products, we can further reduce the number of adders required in the wallace tree by appropriate grouping of similar partial products. To get the final output, it is only required to add all the partial products somehow. There is no rule stipulating which partial product has to be added with which one and in what stage of the wallace tree.

Consider the 7-bit partial products pp_1 and pp_4. It can be added with any of the twelve other partial products (after reduction). If we choose to add pp_1 and pp_4 with, say the 10-bit reduced partial product ppr_1, we need a 10-bit wallace adder. Instead if we choose to add pp_1 and pp_4 with another 7-bit partial product with similar zero-padding, say pp_6, only a five bit wallace adder is required as shown in Figure 3.9. Similarly we can group pp_3, pp_{12} and pp_{15} etc to reduce the number of adders. The same technique can be continued in the later stages of

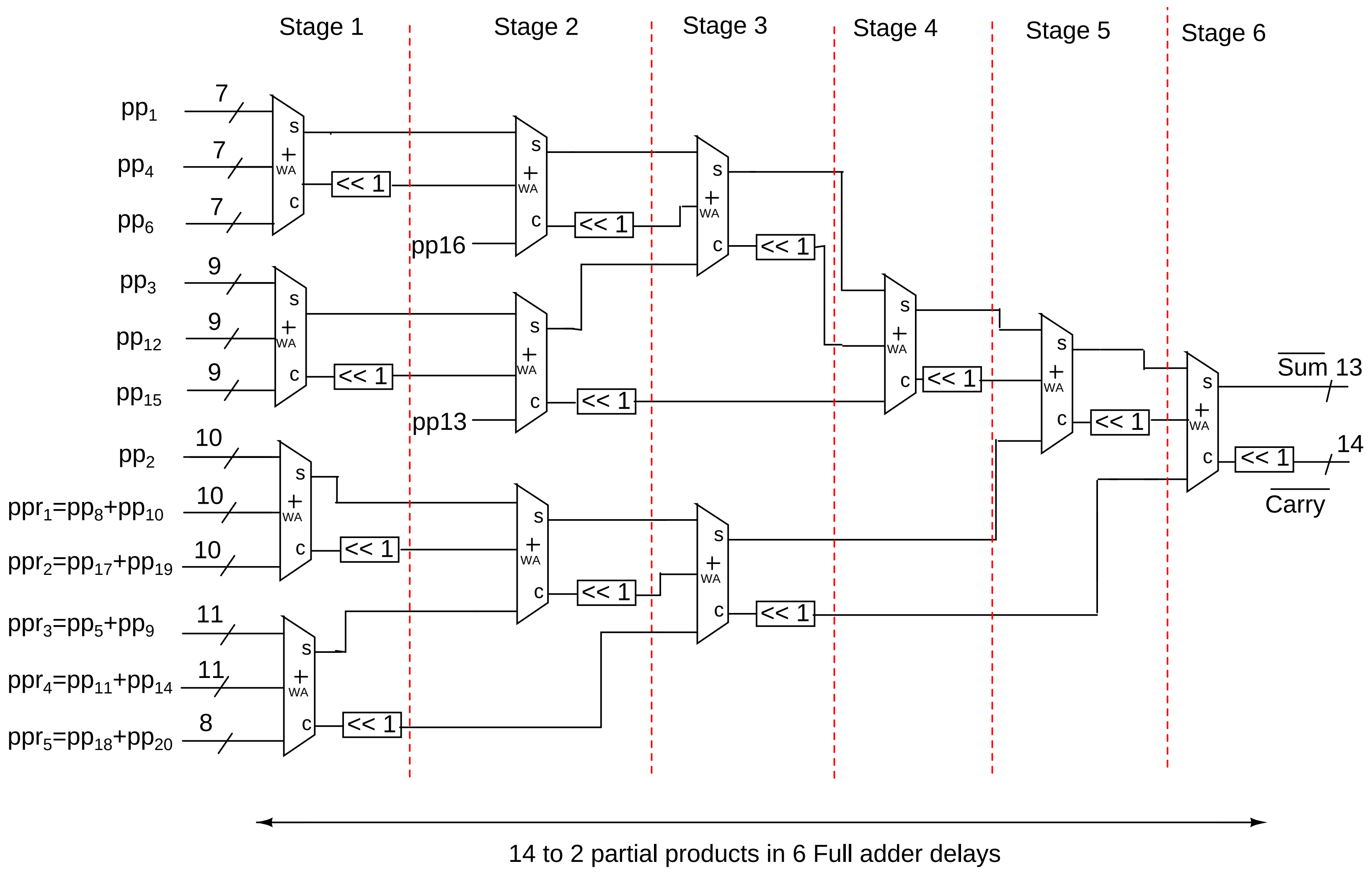


Figure 3.8: Six stage wallace tree addition of nine original partial products and the five reduced partial products. Note the one bit left shift performed at the carry output of each wallace adder.

wallace tree by identifying partial products with identical zero paddings.

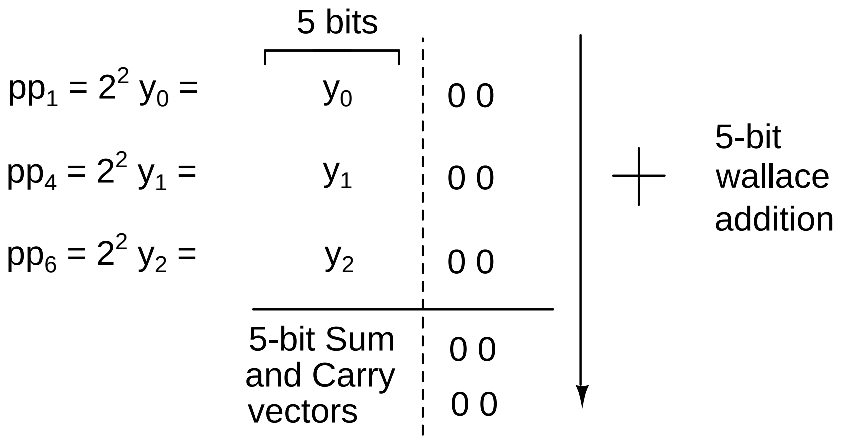


Figure 3.9: Careful grouping of partial products can decrease the number of adders

3.4 Vector Merging Adder (VMA)

In Figure 3.8, it is found that the wallace addition in its final stage gives two vectors: a 14-bit $\overline{\textbf{carry}}$ and a 13-bit $\overline{\textbf{sum}}$. In order to get the final output from the sinc filter, it is necessary to add up these two vectors. Since carry look ahead addition is not usually practical for adding vectors of length more than four bits, to build fast adders, logarithmic look ahead adders are used[11, p.579]. Thus the final Vector Merging Adder (VMA) in the sinc filter circuit is implemented using a Kogge-Stone logarithmic look ahead adder[12], the concept and details of which are explained in this section.

Kogge-Stone adder works by decomposing the the process of carry propagation in a carry look ahead adder into subgroups. The final output carry of an N-bit adder is generated in $O(\log_2 N)$ time and hence the name logarithmic adder. The working of Kogge-Stone adder can be split into three operations:

1. Creation of *generate* and *propagate* signals
2. Dot operation
3. Generation of sum

3.4.1 Generate and Propagate signals

Assume that the 14-bit numbers to be added are $A_{0:13}$ and $B_{0:13}$. The definitions of *generate* G and *propagate* P given in (3.7) and (3.8) are the same as that of a standard carry look ahead adder. See Figure 3.10(a).

$$G_k = A_k B_k \tag{3.7}$$

$$P_k = A_k \oplus B_k \tag{3.8}$$

where $k = 0, 1, \ldots 13$ for a 14-bit adder. The carry dependencies in a carry look ahead topology for the first four bits for an initial input carry C_i are given below. C_k is the k^{th} output carry.

$$C_0 = G_0 + P_0 C_i \tag{3.9}$$

$$C_1 = G_1 + P_1 G_0 + P_1 P_0 C_i \tag{3.10}$$

$$C_2 = G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_i \tag{3.11}$$

$$C_3 = G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0 + P_3 P_2 P_1 P_0 C_i \tag{3.12}$$

3.4.2 Dot operation

A dot operation is defined as[11, p. 580]

$$(G, P).(G', P') = (G + PG', PP') \tag{3.13}$$

This applied to a particular case is

$$(G_1, P_1).(G_0, P_0) = (G_1 + P_1 G_0, P_1 P_0) \tag{3.14}$$

Keeping this definition in mind, the boolean expressions for carrys in a CLA given in (3.9) to (3.12) can be written as

$$C_0 = G_0 + P_0 C_i \tag{3.15}$$

$$\begin{aligned} C_1 &= (G_1 + P_1 G_0) + (P_1 P_0) C_i \\ &= G_{1:0} + P_{1:0} C_i \end{aligned} \tag{3.16}$$

$$\begin{aligned} C_2 &= G_2 + P_2 (G_1 + P_1 G_0 + P_1 P_0 C_i) \\ &= G_2 + P_2 C_1 \end{aligned} \tag{3.17}$$

$$\begin{aligned} C_3 &= (G_3 + P_3 G_2) + (P_3 P_2)(G_1 + P_1 G_0 + P_1 P_0 C_i) \\ &= G_{3:2} + P_{3:2} C_1 \end{aligned} \tag{3.18}$$

where $(G_{3:2}, P_{3:2}) = (G_3, P_3).(G_2, P_2)$. $G_{3:2}$ is one only when a carry is generated at bit position 3 or is generated at 2 and is propgated. $P_{3:2}$ is one only when a carry propagates through both bit positions 2 and 3. Similarly for $(G_{1:0}, P_{1:0}) = (G_1, P_1).(G_0, P_0)$. The pictorial representation of the dot operation is given in Figure 3.10(b).

Associative property of dot operation

Simple boolean analysis will show that the dot operation is not commutative but is associative[11, p. 580]. A result of the associative property is given in (3.19) and is illustrated in Figure 3.19.

$$\begin{aligned} (G_{3:2} P_{3:2}).(G_{1:0} P_{1:0}) &= \Big[(G_3, P_3).(G_2, P_2)\Big].\Big[(G_1, P_1).(G_0, P_0)\Big] \\ &= \Big[G_3 + P_3 G_2, P_3 P_2\Big].\Big[G_1 + P_1 G_0, P_1 P_0\Big] \\ &= \Big[G_3 + P_3 G_2 + P_3 P_2 (G_1 + P_1 G_0), P_3 P_2 P_1 P_0\Big] \\ &= \Big[G_{3:0}, P_{3:0}\Big] \end{aligned} \tag{3.19}$$

From (3.19) and (3.12), we get the fourth carry, $C_3 = G_{3:0} + P_{3:0} C_i$.

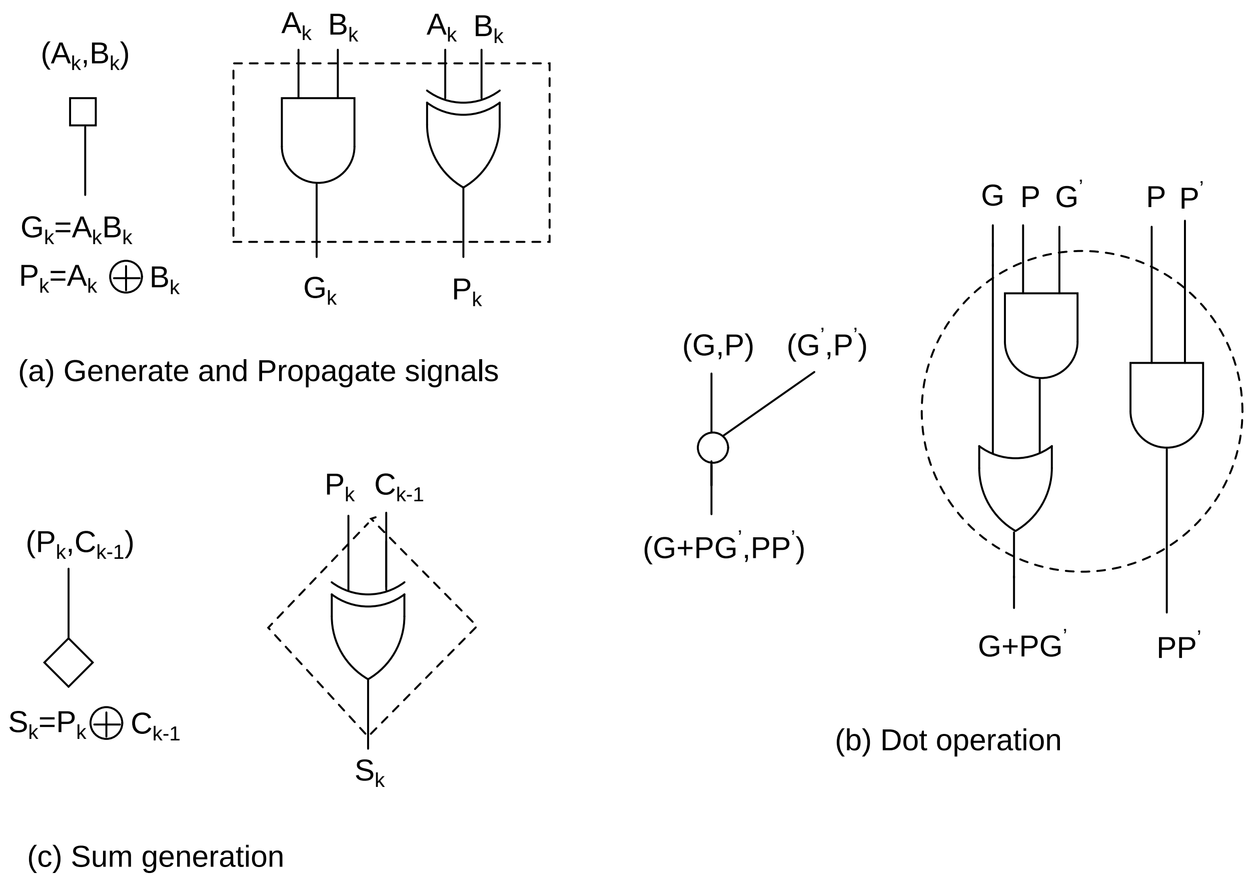


Figure 3.10: The basic operations involved in Kogge-Stone adder. Creation of Pand G signals for each input bit, dot operation and final sum generation.

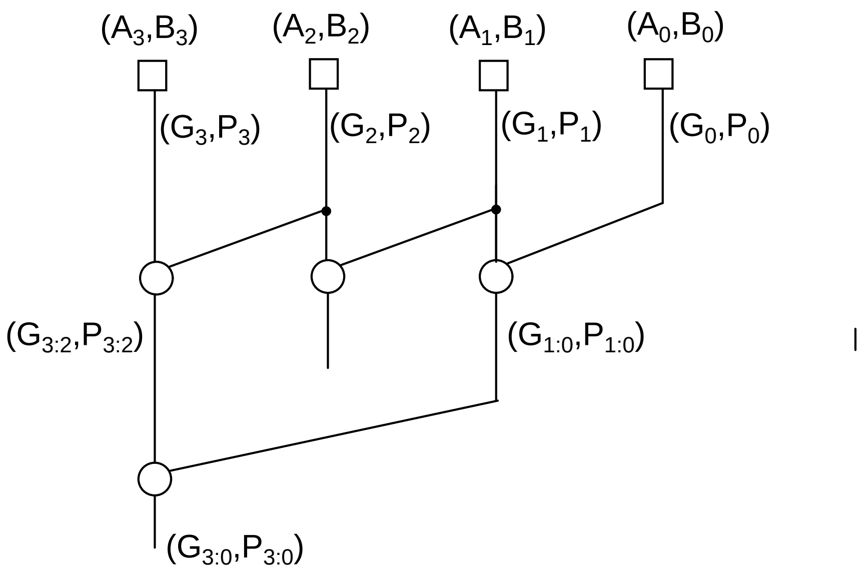


Figure 3.11: Associative property of dot operation.

3.4.3 Sum Generation

Once all the carrys are generated in a Kogge-Stone tree, the final sum has to be generated. For this, the sum generation blocks in the Figure 3.10(c) are used. For $k = 1, 2 \ldots 13$, k^{th} bit of the sum S_k, is given by

$$S_k = P_k \oplus C_{k-1} \tag{3.20}$$

$$\tag{3.21}$$

S_0 is given by $P_0 \oplus C_i$. Since $C_i = 0$ in our case, S_0 is just equal to P_0. The complete schematic diagram of the 14-bit Kogge-Stone adder is shown in Figure 3.12.

3.4.4 14-bit Kogge-Stone adder

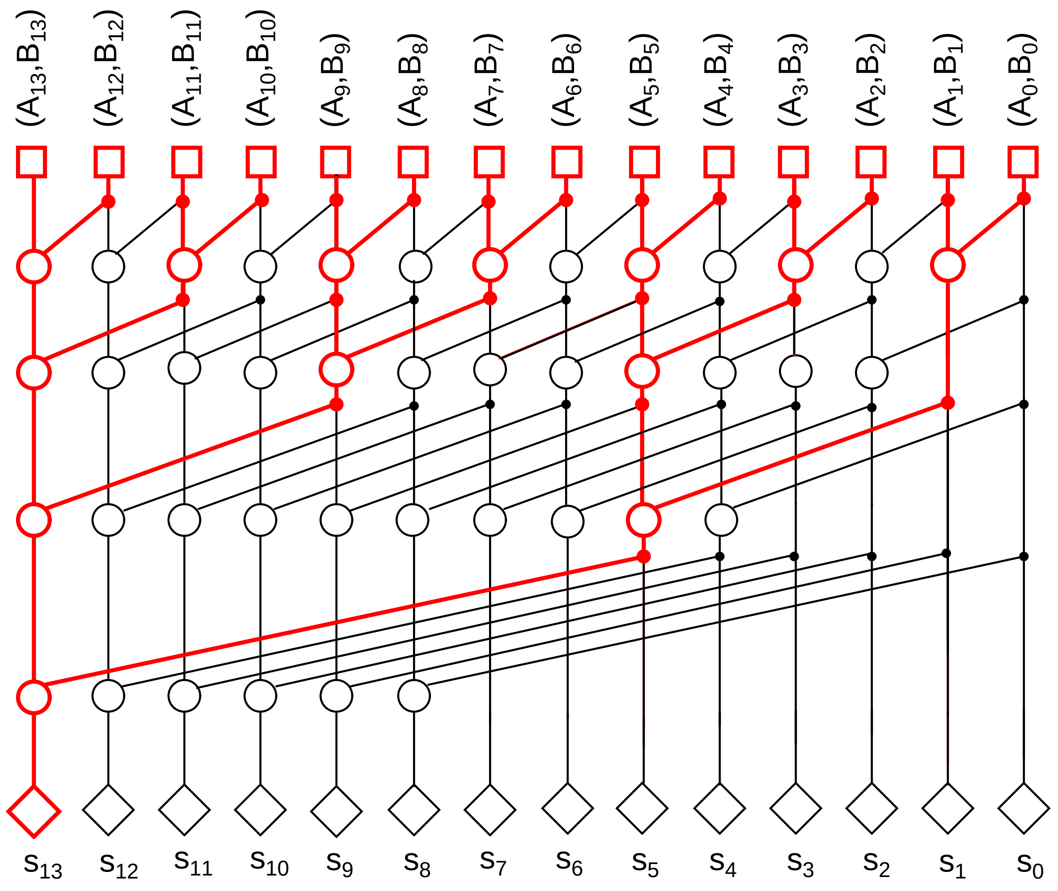


Figure 3.12: Schematic of 14-bit Kogge-Stone logarithmic look ahead adder. The path that generates the fourteenth carry C_{13} is highlighted.

3.5 The complete sinc filter

The complete sinc filter implementation is shown in Figure 3.13. The output from the wallace tree, $\overline{sum}$ and $\overline{carry}$ is added together using the vector merging Kogge-Stone adder. We have considered all the components of sinc filter except the intial block that splits the input data at 1 GHz into five streams. The design of this decimator and clock divider will be discussed in Chapter 5.

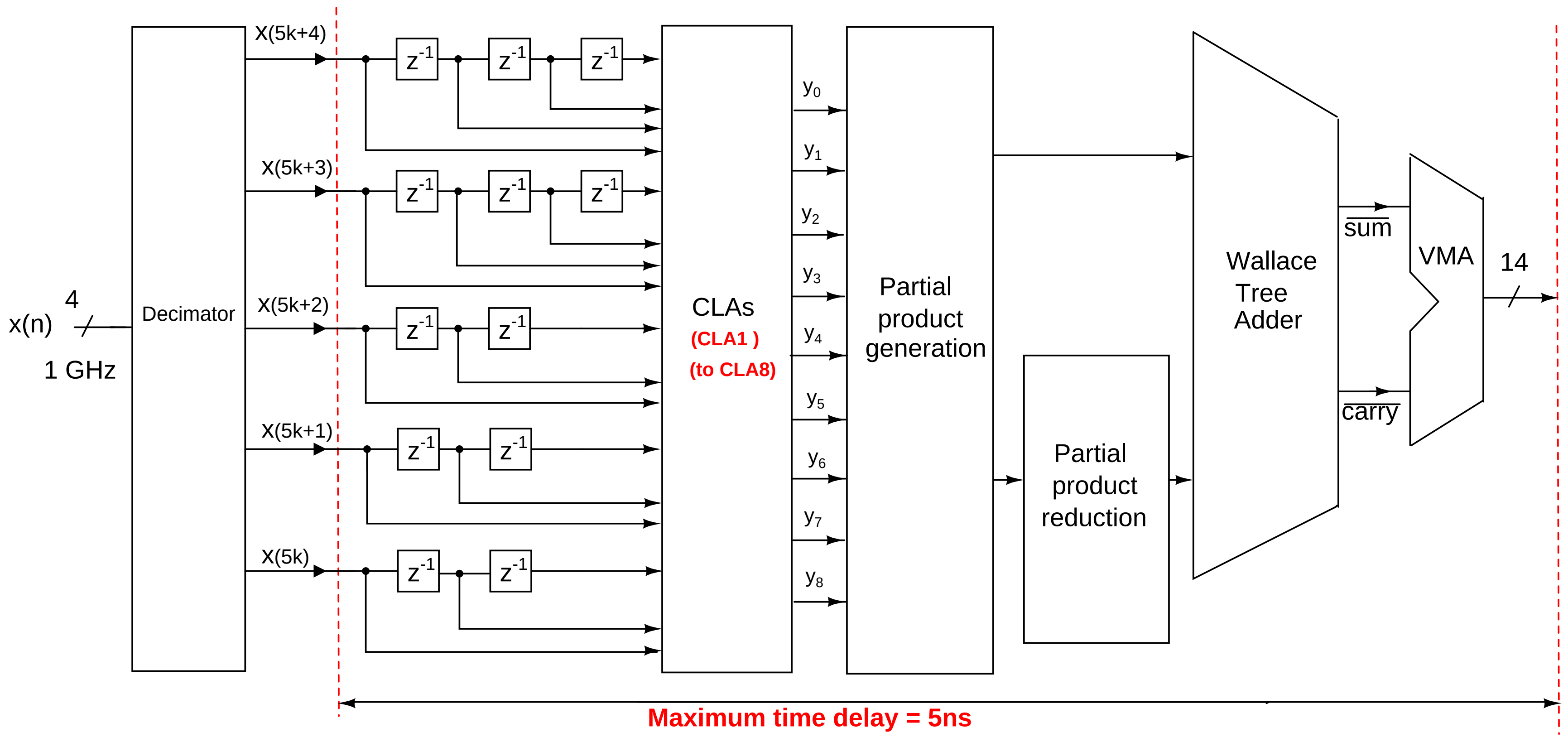


Figure 3.13: Complete sinc filter schematic. CLA: Carry Lookahead Adder, VMA: Vector Merging Adder.

CHAPTER 4

FIR Filter Implementation

In the earlier chapters, we have seen that it is beneficial to implement the decimation filter in two stages: A sinc filter in first stage and an FIR filter in the second. We have also seen that the polyphase decomposition of the sinc filter has taken the whole operation of the circuit to a lower rate which apart from relaxing the timing constraints, lowers power consumption too. For the same reasons, we use polyphase decomposition for implementing the second stage FIR filter also. The decomposition of the FIR filter to five filters $G_0(z)$ to $G_4(z)$ is discussed in section 2.7.2.

The finite precision FIR filter coefficients are obtained in Table 2.4. The input to the FIR filter is the 14-bit output of the sinc filter. Unlike sinc filter, the coefficients can be either positive or negative for the FIR filter. So the partial products can also be positive or negative. Each filter coefficient gets multiplied by the shifted and summed input signals in a direct form FIR implementation. Considering that multiplication is just addition of partial products, a rough idea about the complexity of the circuit can be obtained by counting the number of partial products. Table 4.1 lists the filter coefficients, their expansion and the number of partial products each coefficient will yield.

4.1 Implementation of decomposed filters

From the Figure 2.25 and Figure 2.26, we know that the FIR filter implementation using polyphase decomposition involves the design of a decimator with decimation factor 5 and five FIR filters, $G_0(z)$ to $G_4(z)$. The design of the decimator is discussed in detail in chapter 5. For now, assume that the inputs to the five FIR

Table 4.1: Filter coefficients and partial products

h_k	Expansion	Number of partial products
$h_0 = 1$	1	1
$h_1 = 6$	$(2^2 + 2)$	2
$h_2 = 13$	$(2^3 + 2^2 + 1)$	3
$h_3 = 18$	$(2^4 + 2)$	2
$h_4 = 17$	$(2^4 + 1)$	2
$h_5 = 8$	(2^3)	1
$h_6 = -9$	$-(2^3 + 1)$	2
$h_7 = -31$	$-(2^4 + 2^3 + 2^2 + 2 + 1)$	5
$h_8 = -52$	$-(2^5 + 2^4 + 2^2)$	3
$h_9 = -64$	$-(2^6)$	1
$h_{10} = -61$	$-(2^5 + 2^4 + 2^3 + 2^2 + 1)$	5
$h_{11} = -40$	$-(2^5 + 2^3)$	2
$h_{12} = -7$	$-(2^2 + 2 + 1)$	3
$h_{13} = 28$	$(2^4 + 2^3 + 2^2)$	3
$h_{14} = 51$	$(2^5 + 2^4 + 2 + 1)$	4
$h_{15} = 52$	$(2^5 + 2^4 + 2^2)$	3
$h_{16} = 26$	$(2^4 + 2^3 + 2)$	3
$h_{17} = -20$	$-(2^4 + 2^2)$	2
$h_{18} = -69$	$-(2^6 + 2^2 + 1)$	3
$h_{19} = -100$	$-(2^6 + 2^5 + 2^2)$	3
$h_{20} = -94$	$-(2^6 + 2^4 + 2^3 + 2^2 + 2)$	5
$h_{21} = -46$	$-(2^5 + 2^3 + 2^2 + 2)$	4
$h_{22} = 36$	$(2^5 + 2^2)$	2
$h_{23} = 124$	$(2^6 + 2^5 + 2^4 + 2^3 + 2^2)$	5
$h_{24} = 180$	$(2^7 + 2^5 + 2^4 + 2^2)$	4
$h_{25} = 168$	$(2^7 + 2^5 + 2^3)$	3
$h_{26} = 64$	(2^6)	1
$h_{27} = -132$	$-(2^7 + 2^4)$	2
$h_{28} = -392$	$-(2^8 + 2^7 + 2^3)$	3
$h_{29} = -666$	$-(2^9 + 2^7 + 2^4 + 2^3 + 2)$	5
$h_{30} = -894$	$-(2^9+2^8+2^6+2^5+2^4+2^3+2^2+2)$	8
$h_{31} = -1024$	-2^{10}	1
	Total =	96

filters, $y(5k)$ to $y(5k{+}4)$ are available at the positive edge of a 25 ns clock (40 MHz). Also assume that all these five inputs stay available for one full time period (25 ns).

4.1.1 Implementation of $G_4(z)$

The definition of $G_4(z)$ is

$$\begin{aligned} G_4(z) = h_4 + h_9 z^{-1} + h_{19} z^{-2} + h_{24} z^{-3} + h_{29} z^{-4} + h_{29} z^{-5} \\ + h_{24} z^{-6} + h_{19} z^{-7} + h_{14} z^{-8} + h_9 z^{-9} + h_4 z^{-10} \end{aligned} \tag{4.1}$$

Note that the filter $G_4(z)$ is symmetric and the direct form implementation of this FIR filter is shown in Figure 4.1. The input $y(5k)$ is shifted and added to generate 15-bit multiplicands a_0 to a_5. The 14-bit addition of the shifted inputs can be performed by simple ripple carry adders(RCAs) unlike in a sinc filter where carry look ahead adders(CLAs) are used. This is because the timing constraint is much more relaxed (by a factor of 5 to be precise) for the FIR filter making the use of simpler and less bulky adders feasible.

Table (4.2) shows all the negative and positive partial products obtained by the multiplication of filter coefficients, h_4, h_9, h_{14}, h_{19}, h_{24} and h_{29} with 15-bit numbers a_0 to a_5 respectively. k^{th} positive and negative partial products are denoted pp_k^+ and pp_k^- respectively. From Table 4.2, it can be seen that $G_4(z)$ generates 10 positive partial products (pp_1^+ to pp_{10}^+) and 9 negative partial products (pp_1^- to pp_9^-).

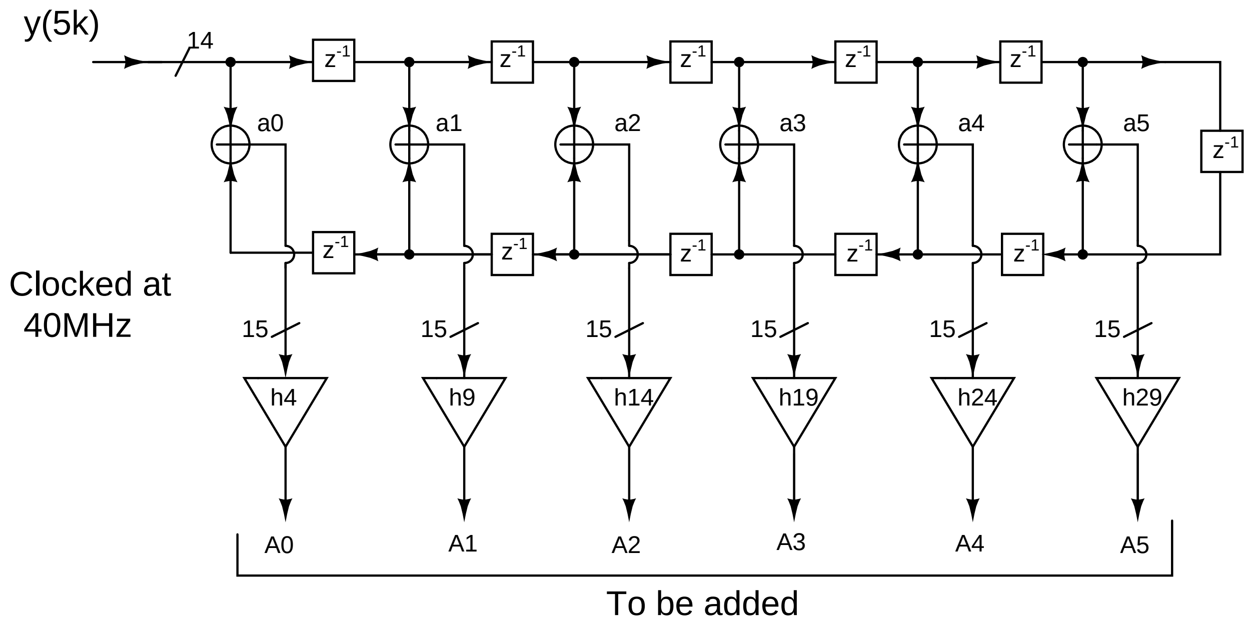


Figure 4.1: Circuit implementation of $G_4(z)$. The shifted and added input terms a_0 to a_5 get multiplied with the filter coefficients to form partial products A_0 to A_5.

Table 4.2: Filter coefficients of $G_4(z)$ and partial products

A_k	$a_k \times h$	Positive and negative partial products
A_0	$a_0 \times h_4 = 17a_0$	$pp_1^+ = 2^4 a_0$, $pp_2^+ = a_0$
A_1	$a_1 \times h_9 = -64a_1$	$pp_1^- = 2^4 a_1$
A_2	$a_2 \times h_{14} = 51a_2$	$pp_3^+ = 2^5 a_2$, $pp_4^+ = 2^4 a_2$, $pp_5^+ = 2a_2$, $pp_6^+ = a_2$
A_3	$a_3 \times h_{19} = -100a_3$	$pp_2^- = 2^6 a_3$, $pp_3^- = 2^5 a_3$, $pp_4^- = 2^3 a_3$
A_4	$a_4 \times h_{24} = 180a_4$	$pp_7^+ = 2^7 a_4$, $pp_8^+ = 2^5 a_4$, $pp_9^+ = 2^4 a_4$, $pp_{10}^+ = 2^2 a_4$
A_5	$a_5 \times h_{29} = -666a_5$	$pp_5^- = 2^9 a_5$, $pp_6^- = 2^7 a_5$, $pp_7^- = 2^4 a_5$, $pp_8^- = 2^3 a_5$, $pp_9^- = 2a_5$

4.1.2 Implementation of $G_0(z)$ and $G_3(z)$

The definitions of $G_0(z)$ and $G_3(z)$ are

$$\begin{aligned} G_3(z) &= h_3 + h_8 z^{-1} + h_{13} z^{-2} + h_{18} z^{-3} + h_{23} z^{-4} + h_{28} z^{-5} + h_{30} z^{-6} && (4.2)\\ &\quad + h_{25} z^{-7} + h_{20z^{-8}} + h_{15z^{-9}} + h_{10} z^{-10} + h_5 z^{-11} + h_0 z^{-12} \\ G_0(z) &= h_0 + h_5 z^{-1} + h_{10} z^{-2} + h_{15} z^{-3} + h_{20} z^{-4} + h_{25} z^{-5} + h_{30} z^{-6} && (4.3)\\ &\quad + h_{28} z^{-7} + h_{23} z^{-8} + h_{18} z^{-9} + h_{13} z^{-10} + h_8 z^{-11} + h_3 z^{-12} \end{aligned}$$

The symmetry of the filter coefficients between the filters is exploited to get the FIR filter implementation as shown in Figure 4.2. The 14-bit inputs $y(5k+1)$ and $y(5k+4)$ are shifted and added to generate 15-bit multiplicands b_0 to b_{12}. The 14-bit additions of the shifted inputs are performed by simple ripple carry adders(RCAs).

Table 4.3 shows all the negative and positive partial products obtained by the multiplication of filter coefficients of $G_0(z)$ and $G_3(z)$ with 15-bit numbers b_0 to b_{12} respectively. From Table 4.3, it can be seen that $G_0(z)$ and $G_3(z)$ generates 18 positive partial products $(pp_{11}^{+}$ to $pp_{28}^{+})$ and 27 negative partial products $(pp_{10}^{-}$ to $pp_{36}^{-})$.

4.1.3 Implementation of $G_1(z)$ and $G_2(z)$

The definitions of $G_1(z)$ and $G_2(z)$ are

$$\begin{aligned} G_1(z) &= h_1 + h_6 z^{-1} + h_{11} z^{-2} + h_{16} z^{-3} + h_{21} z^{-4} + h_{26} z^{-5} + h_{31} z^{-6} && (4.4)\\ &\quad + h_{27} z^{-7} + h_{22} z^{-8} + h_{17} z^{-9} + h_{12} z^{-10} + h_7 z^{-11} + h_2 z^{-12} \\ G_2(z) &= h_2 + h_7 z^{-1} + h_{12} z^{-2} + h_{17} z^{-3} + h_{22} z^{-4} + h_{27} z^{-5} + h_{31} z^{-6} && (4.5)\\ &\quad + h_{26} z^{-7} + h_{21} z^{-8} + h_{16} z^{-9} + h_{11} z^{-10} + h_6 z^{-11} + h_1 z^{-12} \end{aligned}$$

Again, by exploiting the symmetry of coefficients between the filters, the implementation is shown in Figure 4.3. The inputs $y(5k+2)$ and $y(5k+1)$ are shifted and added to generate 15-bit multiplicands c_0 to c_{12}. The 14-bit additions of the shifted inputs are performed by ripple carry adders(RCAs).

Table 4.4 shows all the negative and positive partial products obtained by the multiplication of filter coefficients of $G_1(z)$ and $G_2(z)$ with 15-bit numbers c_0 to c_{12} respectively. From Table 4.4, it can also be seen that $G_1(z)$ and $G_2(z)$ generates 11 positive partial products $(pp^+_{29}$ to $pp^+_{39})$ and 21 negative partial products $(pp^-_{37}$ to $pp^-_{57})$.

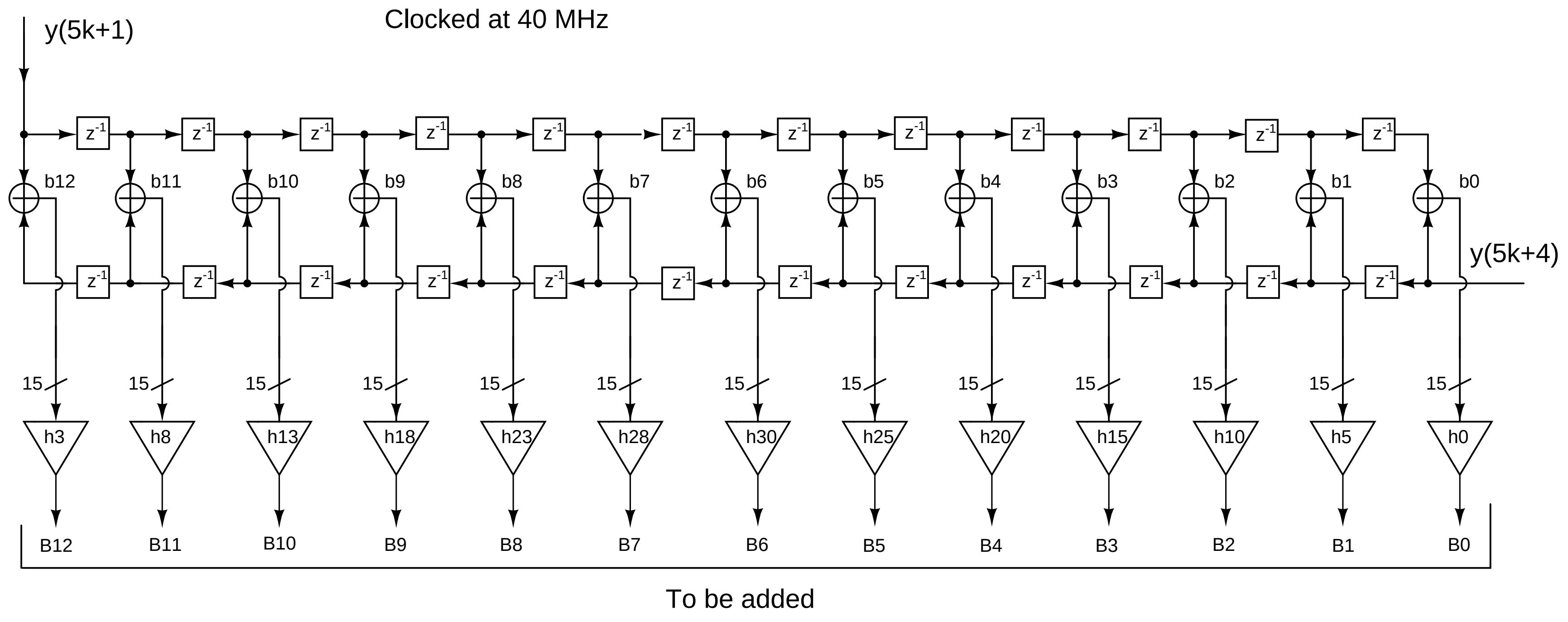


Figure 4.2: Circuit implementation of $G_0(z)$ and $G_3(z)$. The terms b_0 to b_{12} gets multiplied with the filter coefficients to form the partial products B_0 to B_{12}.

Table 4.3: Filter coefficients of $G_3(z), G_0(z)$ and partial products

B_k	$b_k \times h$	Positive and negative partial products
B_0	$b_0 \times h_0 = b_0$	$pp_{11}^{+} = b_0,$
B_1	$b_1 \times h_5 = 8b_1$	$pp_{12}^{+} = 2^3 b_1,$
B_2	$b_2 \times h_{10} = -61b_2$	$pp_{10}^{-} = 2^5 b_2,$ $pp_{11}^{-} = 2^4 b_2$ $pp_{12}^{-} = 2^3 b_2$ $pp_{13}^{-} = 2^3 b_2$ $pp_{14}^{-} = b_2$
B_3	$b_3 \times h_{15} = 52b_3$	$pp_{13}^{+} = 2^5 b_3,$ $pp_{14}^{+} = 2^4 b_3$ $pp_{15}^{+} = 2^2 b_3$
B_4	$b_4 \times h_{20} = -94b_4$	$pp_{15}^{-} = 2^6 b_4,$ $pp_{16}^{-} = 2^4 b_4$ $pp_{17}^{-} = 2^3 b_4$ $pp_{18}^{-} = 2^2 b_4$ $pp_{19}^{-} = 2b_4$
B_5	$b_5 \times h_{25} = 168b_5$	$pp_{16}^{+} = 2^7 b_5,$ $pp_{17}^{+} = 2^5 b_5$ $pp_{18}^{+} = 2^3 b_5$
B_6	$b_6 \times h_{30} = -894b_6$	$pp_{20}^{-} = 2^9 b_6,$ $pp_{21}^{-} = 2^8 b_6$ $pp_{22}^{-} = 2^6 b_6$ $pp_{23}^{-} = 2^5 b_6$ $pp_{24}^{-} = 2^4 b_6$ $pp_{25}^{-} = 2^3 b_6$ $pp_{26}^{-} = 2^2 b_6$ $pp_{27}^{-} = 2b_6$
B_7	$b_7 \times h_{28} = -392b_7$	$pp_{28}^{-} = 2^8 b_7,$ $pp_{29}^{-} = 2^7 b_7$ $pp_{30}^{-} = 2^3 b_7$
B_8	$b_8 \times h_{23} = 124b_8$	$pp_{19}^{+} = 2^6 b_8,$ $pp_{20}^{+} = 2^5 b_8$ $pp_{21}^{+} = 2^4 b_8$ $pp_{22}^{+} = 2^3 b_8$ $pp_{23}^{+} = 2^2 b_8$
B_9	$b_9 \times h_{18} = -69b_9$	$pp_{31}^{-} = 2^6 b_9,$ $pp_{32}^{-} = 2^2 b_9$ $pp_{33}^{-} = b_9$
B_{10}	$b_{10} \times h_{13} = 28b_{10}$	$pp_{24}^{+} = 2^4 b_{10},$ $pp_{25}^{+} = 2^3 b_{10}$ $pp_{26}^{+} = 2^2 b_{10}$
B_{11}	$b_{11} \times h_8 = -52b_{11}$	$pp_{34}^{-} = 2^5 b_{11},$ $pp_{35}^{-} = 2^4 b_{11}$ $pp_{36}^{-} = 2^2 b_{11}$
B_{12}	$b_{12} \times h_3 = 18b_{12}$	$pp_{27}^{+} = 2^4 b_{12},$ $pp_{28}^{+} = 2^2 b_{12}$

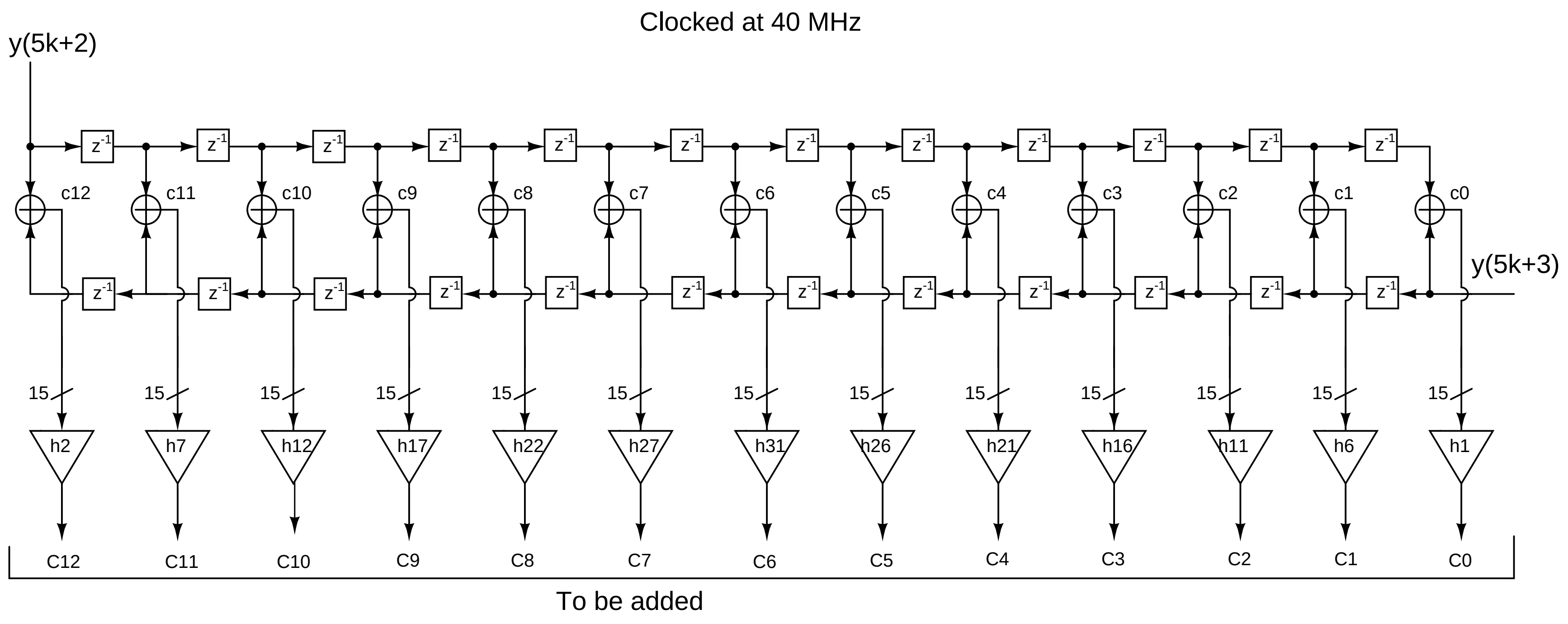


Figure 4.3: Circuit implementation of $G_1(z)$ and $G_2(z)$. The terms c_0 to c_{12} gets multiplied with the filter coefficients to form the partial products C_0 to C_{12}

Table 4.4: Filter coefficients of $G_2(z)$,$G_1(z)$ and partial products

C_0	$c_0 \times h_1 = 6c_0$	$pp^+_{29} = 2^2c_0$, $pp^+_{30} = 2c_0$
C_1	$c_1 \times h_6 = -9c_1$	$pp^-_{37} = 2^3c_1$, $pp^-_{38} = c_1$
C_2	$c_2 \times h_{11} = -40c_2$	$pp^-_{39} = 2^5c_2$, $pp^-_{40} = 2^3c_2$
C_3	$c_3 \times h_{16} = 26c_3$	$pp^+_{31} = 2^4c_3$, $pp^+_{32} = 2^3c_3$ $pp^+_{33} = 2c_3$
C_4	$c_4 \times h_{21} = -46c_4$	$pp^-_{41} = 2^5c_4$, $pp^-_{42} = 2^3c_4$ $pp^-_{43} = 2^2c_4$ $pp^-_{44} = 2c_4$
C_5	$c_5 \times h_{26} = 64c_5$	$pp^+_{34} = 2^6c_5$,
C_6	$c_6 \times h_{31} = -1024c_6$	$pp^-_{45} = 2^10c_6$,
C_7	$c_7 \times h_{27} = -132c_7$	$pp^-_{46} = 2^7c_7$, $pp^-_{47} = 2^2c_7$
C_8	$c_8 \times h_{22} = 36c_8$	$pp^+_{35} = 2^5c_8$, $pp^+_{36} = 2^2c_8$
C_9	$c_9 \times h_{17} = -20c_9$	$pp^-_{48} = 2^4c_9$, $pp^-_{49} = 2^2c_9$
C_{10}	$c_{10} \times h_{12} = -7c_{10}$	$pp^-_{50} = 2^2c_{10}$, $pp^-_{51} = 2c_{10}$ $pp^-_{52} = c_{10}$
C_{11}	$c_{11} \times h_7 = -31c_{11}$	$pp^-_{53} = 2^4c_{11}$, $pp^-_{54} = 2^3c_{11}$ $pp^-_{55} = 2^2c_{11}$ $pp^-_{56} = 2c_{11}$ $pp^-_{57} = c_{11}$
C_{12}	$c_{12} \times h_2 = 13c_{12}$	$pp^+_{37} = 2^3c_{12}$, $pp^+_{38} = 2^3c_{12}$ $pp^+_{39} = c_{12}$

4.1.4 Optimizing the wallace tree

So far, we have seen that to implement the FIR filter, we should implement the five sub-filters. The implemention of these five filters boils down to adding the partial products pp_1^+ to pp_{39}^+ and pp_1^- to pp_{57}^-. This addition can be performed in a wallace adder tree similar to that of sinc filter.

Before we blindly add up the partial products, it is a good idea to carefully observe the nature of partial products. As we have seen in the case of sinc filters, by grouping together suitable partial products we can perform some additions without any hardware at all. Also, as in the case of sinc filter, by grouping the partial products with similar zero-padding, the number of adders in the wallace tree can be minimized.

As shown in Figure 4.4, consider the partial products emerging from the multiplication of a_4 with $h_{19} = 180$.

$$A_4 = a_4 \times h_{24} = 180a_4 \tag{4.6}$$

$$= a_4 \times (2^7 + 2^5 + 2^4 + 2^2) \tag{4.7}$$

Thus calculation of A_4 appears to require three additions with atleast two stages. Instead, A_4 can also be written as

$$A_4 = (a_4 2^2 + a_4)2^5 + (a_4 2^2 + a_4)2^2 \tag{4.8}$$

This splitting requires just two adders, one 15-bit adder that gives $(a_4 2^2 + a_4)$ and one 17-bit adder which gives the final result A_4.

4.2 The complete FIR filter

Unlike in the sinc filter, where all partial products are positive (because all filter coefficients of sinc filter are positive and the 4-bit input from the $\Delta\Sigma$ is DC off-

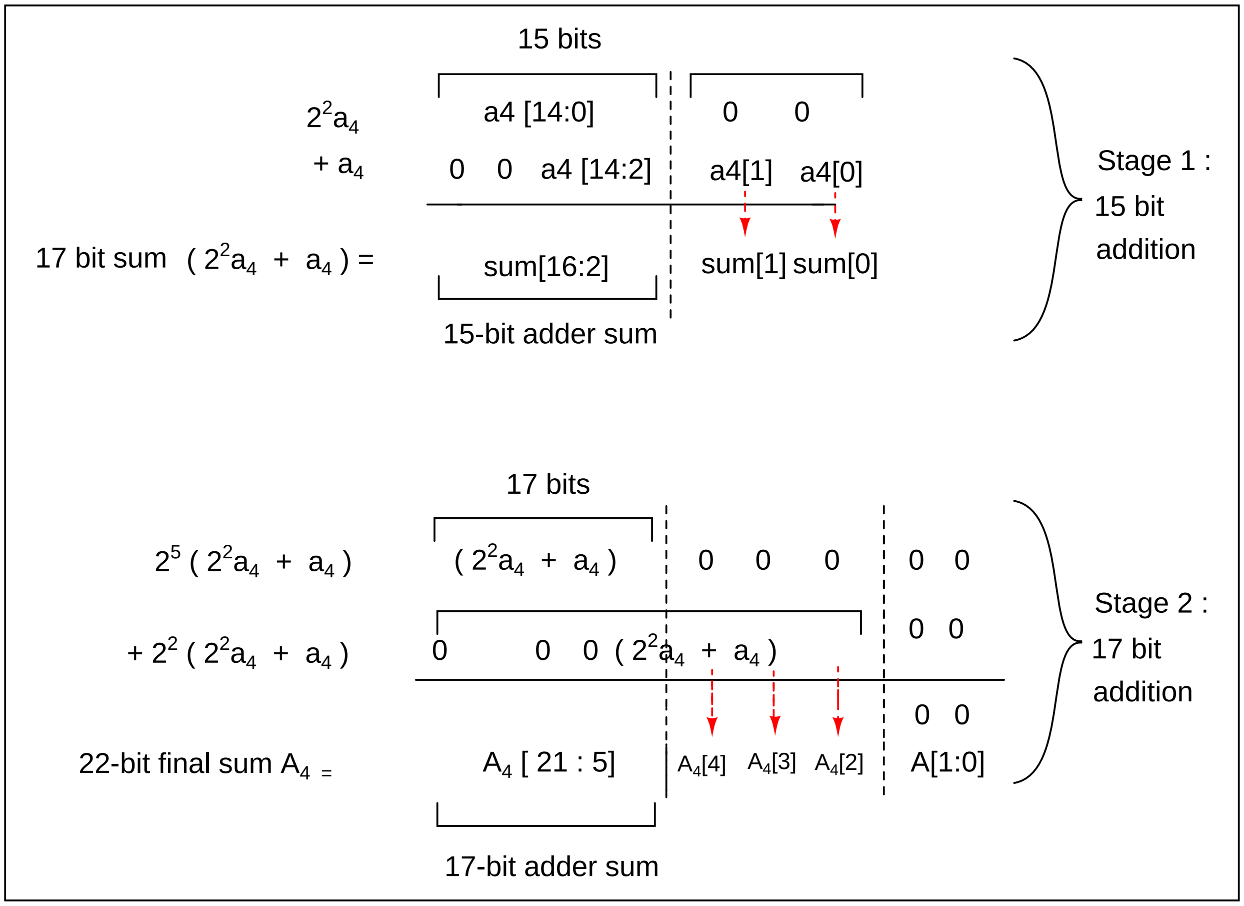


Figure 4.4: Addition of partial products using minimum number of adders

set and scaled to be positive), in the case of FIR filter, there are negative filter coefficients too. This makes the partial products negative eventhough the 14-bit input to the FIR filter is an unsigned positive integer. To make matters simple, we follow the following strategy. Group all the positive partial products and negative partial products separately. Treat them as positive numbers and add up to get two sums. Then finally subtract the sum of negative partial products from that of positive partial products. Refer Figure 4.5.

Note that the for the final subtraction, the sum of postive partial products has to be added to the 2's complement of the sum of negative partial products. This is done by inverting the sum and carry vector from the negative partial product wallace tree and also by giving a C_{in} value one.

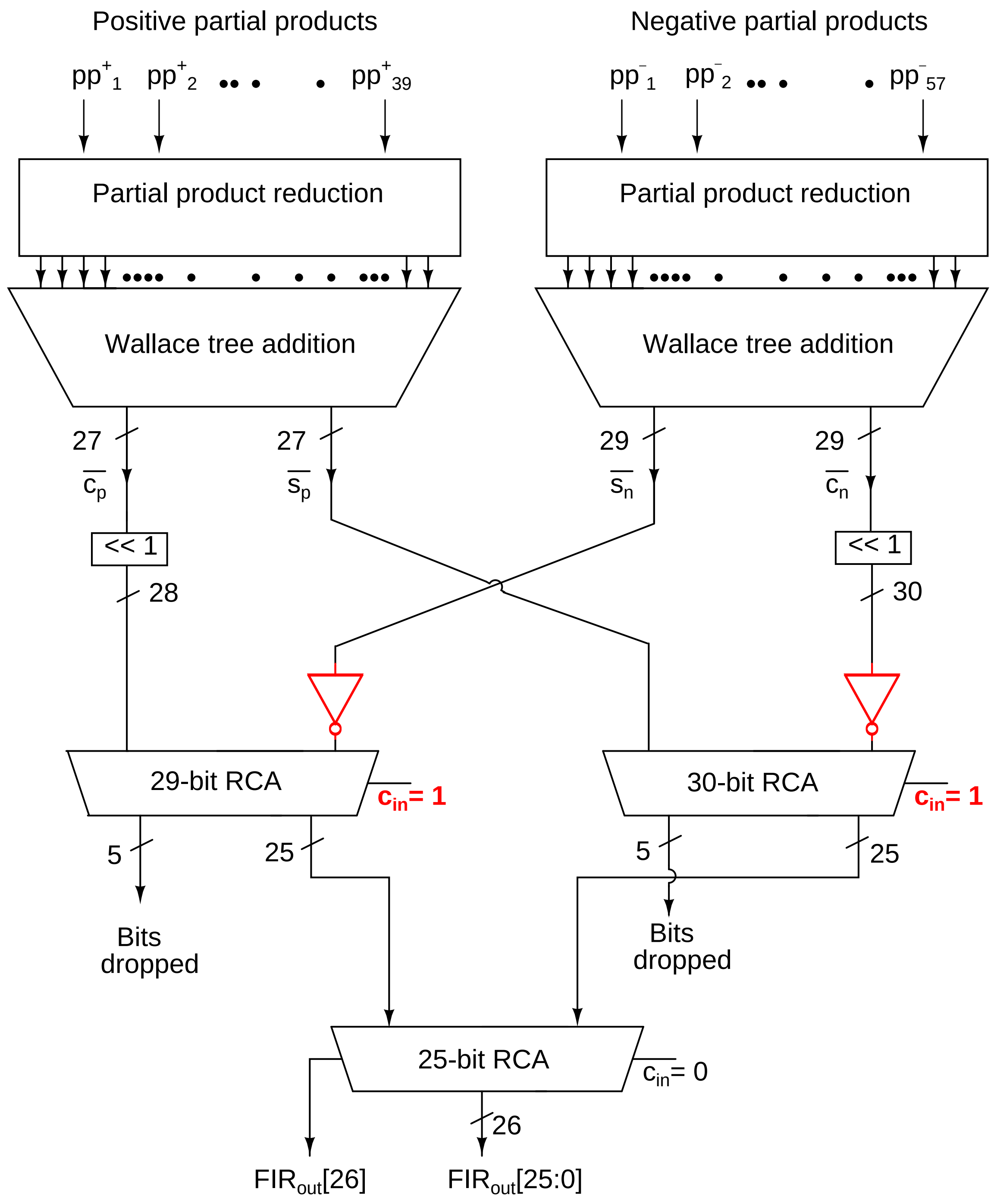


Figure 4.5: Addition of partial products in FIR filter

CHAPTER 5

Decimators and Clock Dividers

5.1 Decimator design

Throughout our discussion, we have assumed that the five parallel streams of data $x(5k)$ to $x(5k+1)$ or $y(5k)$ to $y(5k+1)$ are available at the positive edge of a clock and remains available for one time period which is 5 ns and 25 ns respectively for the sinc and FIR filter. In this chapter, we discuss this decimating process where a single input stream at a particular rate is split into five streams.

5.1.1 Basic decimator circuit

If we have a clock divider that gives one pulse of clk5 for every five pulses of clk1, then the basic structure of a decimator is quite simple as shown in Figure 5.1. For the case of the sinc filter, in Figure 5.1 the clk1 is a 1 GHz or 1 ns clock and clk5 is 200 MHz or 5 ns clock. Thus input is shifted through a chain of registers by 1 ns clock and once in every 5 ns, clk5 registers the shifted values $x[n]$ to $x[n-4]$ to $x[5k+4]$ to $x[5k]$.

Thus five consecutive inputs are provided to the five sub-filters $E_0(z)$ to $E_4(z)$ of the sinc filter in 5 ns. The same basic circuit will hold for FIR filter as well except that clk1 becomes 200 MHz (5 ns) and clk5 becomes 40 MHz (25 ns).

5.1.2 Low power decimator circuit

In the basic decimating structure, we can see that for the sinc filter, we need at least four 4-bit registers working at 1 GHz and five 4-bit registers working at

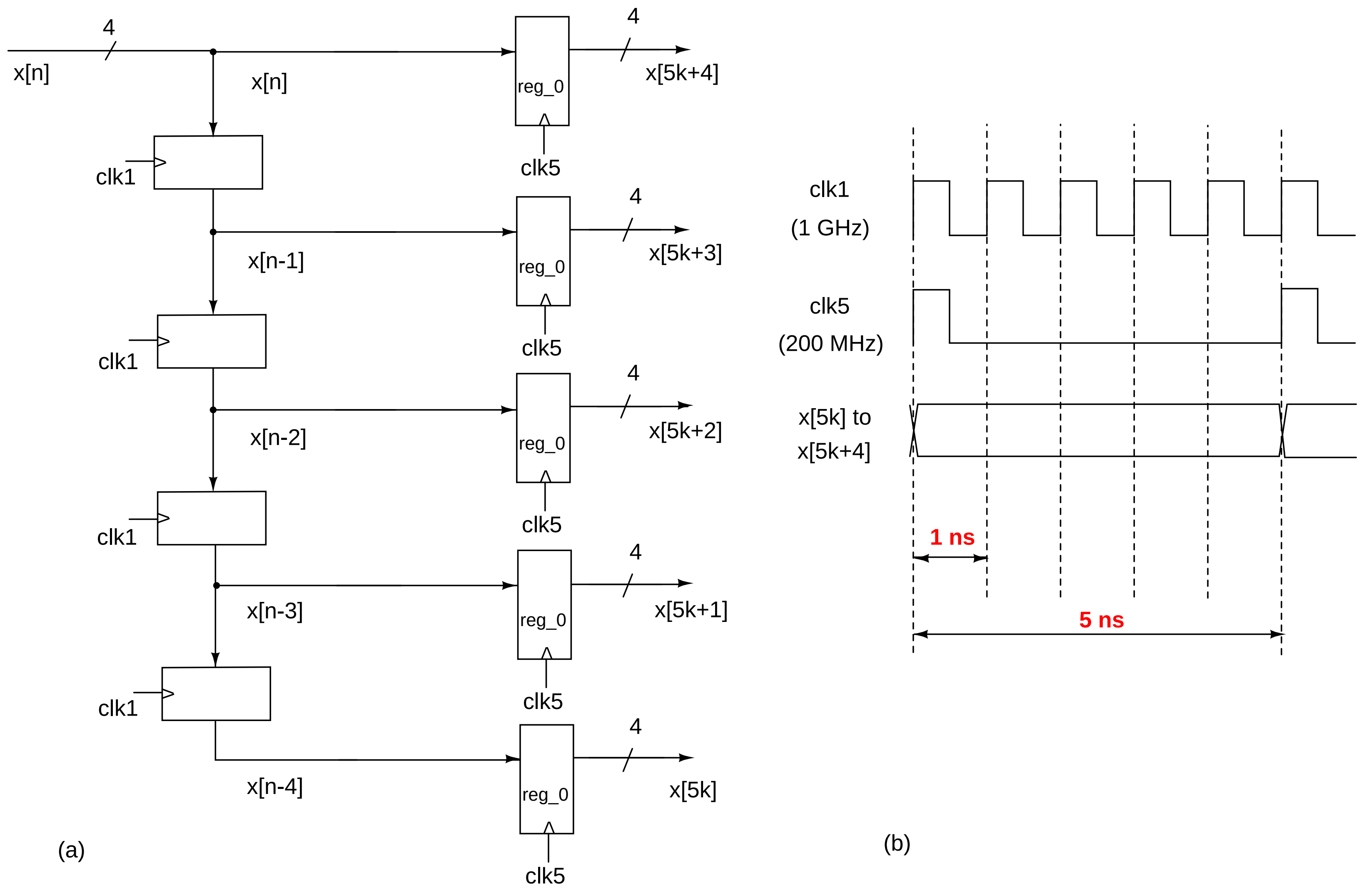


Figure 5.1: (a)Basic decimator structure. (b)The timing diagram. The clocks correspond to the decimator for the sinc filter. For the FIR filter, structure is same except that clk1 becomes 200 MHz and clk5 becomes 40 MHz.

200 MHz. Consider the dynamic power metric CV_{dd}^2f, where C is the capacitance, V_{dd} the supply voltage and f the switching frequency. Assuming a single bit register causes 1 unit of C, the power metric for the basic circuit

$$\begin{aligned} P_1 &= \Big((4\times 4)CV_{dd}^2\Big)\times 1\,\text{GHz} + \Big((5\times 4)CV_{dd}^2\Big)\times 0.2\,\text{GHz} \\ &= 20CV_{dd}^2\times 10^9\text{ units} \end{aligned} \tag{5.1}$$

Now consider a second circuit shown in Figure 5.2(a). Here five phase shifted clocks are used to choose the five consecutive inputs. At the end of five cycles, a common clock registers the five streams together. Detailed timing diagram is shown in Figure 5.2(b). It can be seen that a particular input is ready at least 0.5 ns (with the negative edge of the master clock, clk1) before the corresponding phase shifted clock registers it with a positive clock edge.

In this decimator, there are five 4-bit registers reg_a to reg_e working with the phase shifted 5 ns clocks and five 4-bit registers reg_0 to reg_4 clocked at 5 ns clock. Altogether we have 40 registers working at 200MHz. This makes the power metric

$$\begin{aligned} P_2 &= 40CV_{dd}^2\times 0.2\,\text{GHz} \\ &= 8CV_{dd}^2\times 10^9\text{ units} \end{aligned} \tag{5.2}$$

Comparing P_1 and P_2, note that the power consumption can be reduced 2.5 times using the second circuit with the use of phase shifted clocks. Same argument holds for the second stage decimator before the FIR filter.

5.2 Clock dividers

For both the decimator circuits in the previous section, we assumed a master clock clk1 and a divided-by-5 clock clk5. For the power efficient design of the decimator,

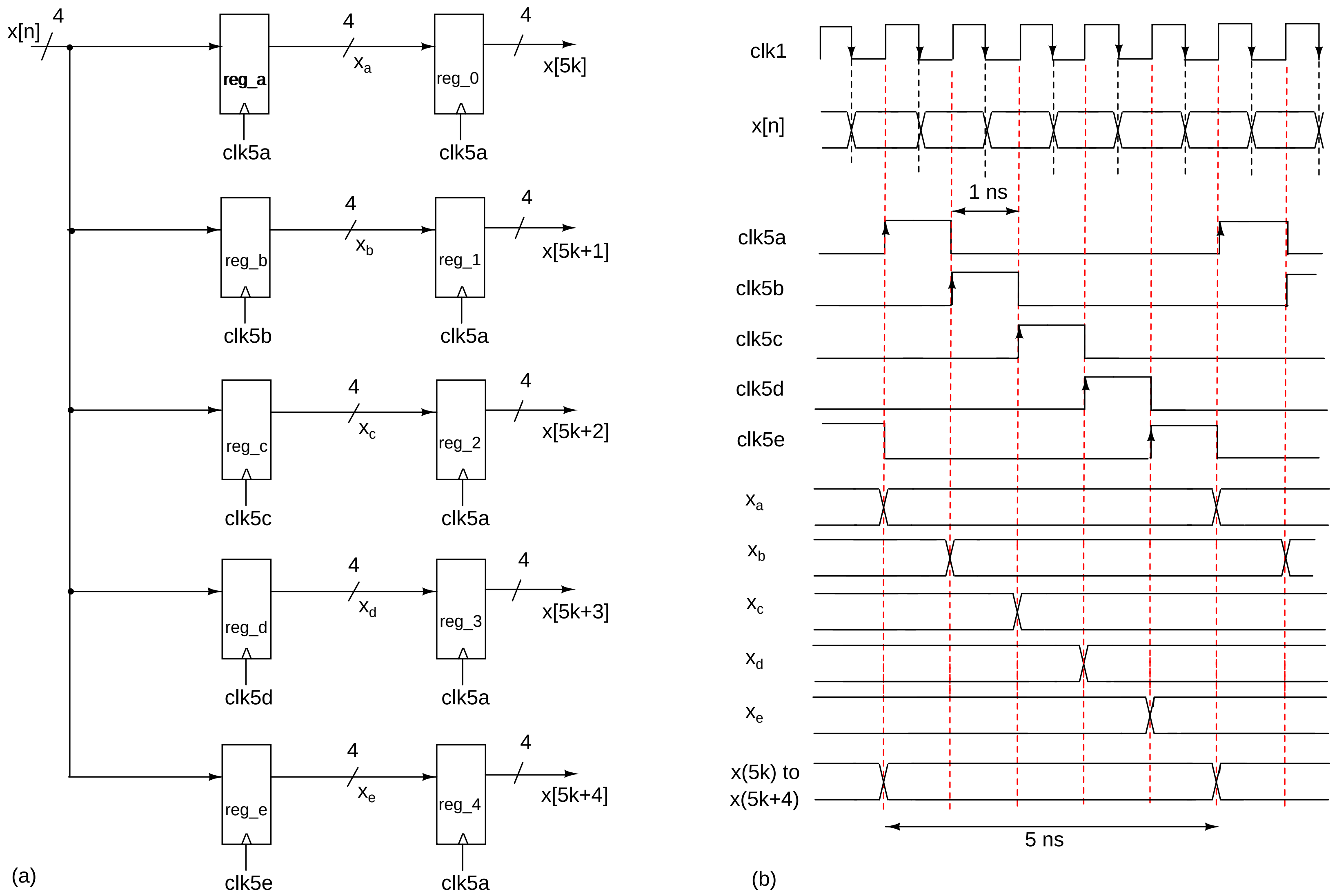


Figure 5.2: (a)Low power decimator structure (b)Timing diagram of the power efficient decimator. clk1 has time period 1 ns and clk5a to clk5e are five phase shifted clocks with time period 5 ns. For the FIR filter decimator, the timing diagram is same except that clk1 is at 200 MHz and clk5a to clk5e are at 40 MHz.

we assumed five phase shifted clocks. In this section we discuss the generation of these phase shifted clocks from the master clock. All the discussions are done for the stage 1 clock divider which derives 5 ns clock from 1 ns clock. Same principles apply for the stage 2 clock divider which generates 25 ns clock from 5 ns clock. Due to the stricter timing constraints, clock divider in the first stage is more challenging.

5.2.1 Counter as a state machine

To divide a clock by 5, the simplest technique is to use a counting state machine that counts from 0 to 4 and then repeats. This creates five states each of which gives rise to the five phase shifted clocks required for the decimator. This is a simple method and requires only three registers working at 1 GHz. Bu the combinational delay between two state transitions is about 3 gate delays and thus it cannot be synthesized using standard cell libraries.

Using trial and error, it has been found that to synthesize a sequential circuit at 1 GHz, the combinational delay in the feedback loop must be less than or equal to two gate delays. So alternative methods for clock generation like *Ring counter* and *Johnson counter* are considered even though these counters require more registers working at 1 GHz (Hence more power).

5.2.2 Ring counter and Johnson's counter

In a Ring counter (Figure 5.3), the $\overline{reset}$ signal sets the state to 10000 and the *locked* '1' shifts between the registers creating five states for five phase shifted clocks.

In a Johnson counter (Figure 5.4), $\overline{reset}$ brings the state to 00000 and as the clock ticks, it passes through ten states, which can be used to generate five phase shifted clocks by coupling states that are five clock cycles apart.

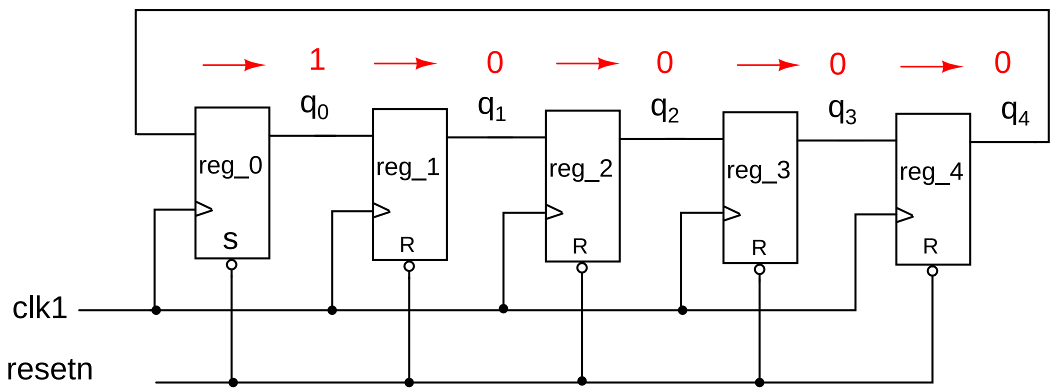


Figure 5.3: Ring counter

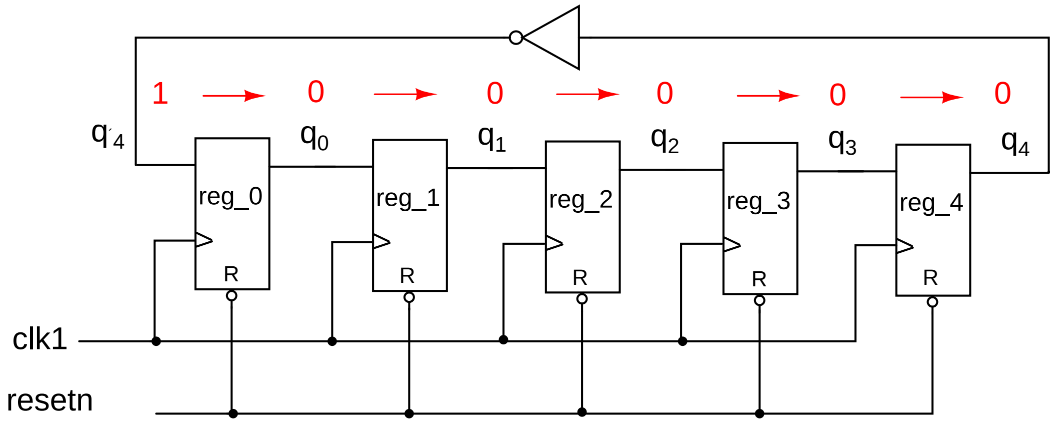


Figure 5.4: Johnson's counter

Note that in both Ring and Johnson counter, there are five 1-bit registers working at 1 GHz and the combinational loop delay ≤ 2 gate delays. Thus any one of these two can be used to divide the clocks. For simplicity, a Ring counter is used in our design.

CHAPTER 6

Decimation Filter with Minimal Passband Droop

So far we have discussed the design and circuit implementation of a two stage decimation filter with a fourth order 10-tap sinc filter in the first stage and a 60th order FIR filter in the second stage. In Chapter 2, it is seen that the passband droop of the 10-tap sinc filter is as high as 2.4 dB. Despite this fairly high droop, a 10-tap filter was chosen instead of a 5-tap one in order to reduce the complexity of the second stage FIR filter. Also, recall that the simulated SNR for a 17.5 MHz input signal was meeting the output SNR requirement of 93 dB.

However, for the purpose of completeness, the case of 5-tap sinc filter followed by an FIR filter should also be considered. This is useful where a passband droop of 2.4 dB is not desirable.

6.1 Five tap fourth order sinc filter

Recall that a general sinc filter is given by

$$H_s(z) = \left(\frac{1}{rM}\left(\frac{1-z^{-rM}}{1-z^{-1}}\right)\right)^K \tag{6.1}$$

For a five tap filter, the value of r is one. The values of K and M remains unchanged (4 and 5 respectively). i.e.

$$H_s(z) = \left(\frac{1}{5}\left(\frac{1-z^{-5}}{1-z^{-1}}\right)\right)^4 \tag{6.2}$$

The passband droop and the worst case alias rejection of this filter is shown in Figure 6.1. The implementation of this filter using polyphase decomposition is

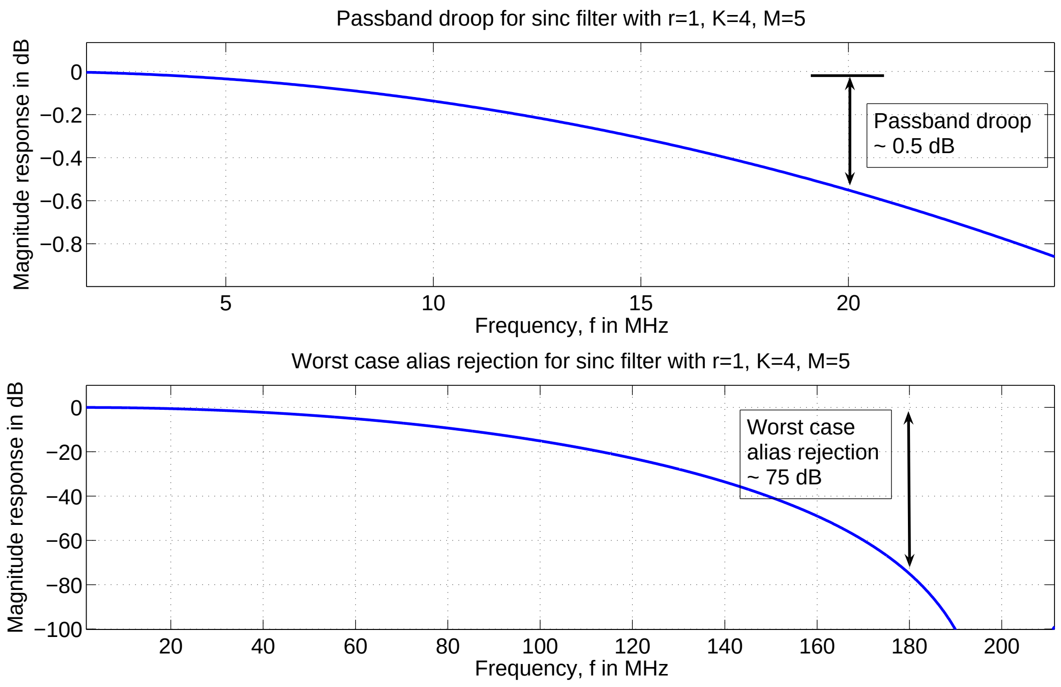


Figure 6.1: Maximum passband droop and worst case alias rejection for sinc filter with $r = 1$, $K = 4$, $M = 5$.

discussed in detail in Chapter 3 (Since the factor $(1 + z^{-5})^4$ was pushed to the low frequency side, the filter discussed in Chapter 3 is a five tap fourth order sinc filter).

6.2 FIR filter for the second stage

From Figure 2.18, it is clear that a 10-tap sinc filter has a much sharper response. Thus if we choose a 5-tap sinc filter, the stopband attenuation of the FIR filter in the second stage must be much higher to get the same SNR. If we simply use a 5-tap sinc filter in cascade with the 60^{th} order FIR filter $H_{pc}(z)$ derived in Chapter 2, the output SNR will be only 84 dB. Thus we need a higher order FIR filter in second stage with a higher stopband attenuation if we were to use 5-tap sinc filter in first stage. As shown in Table 6.1, all the parameters of the FIR filter are kept the same as the earlier except stopband attenuation. With the higher stopband attenuation of 64 dB, the order of the new filter $\hat{H}_{pc}(z)$ is 74.

Table 6.1: Parameters for the FIR filter $\hat{H}_{pc}(z)$

Passband ripple, δ_p	< 1 dB
Stopband attenuation, δ_s	64 dB
Passband edge, f_p	18.25 MHz
Stopband edge, f_s	24 MHz
Sampling frequency, f_s	200 MHz

The frequency response of the 74^{th} order FIR filter is shown in Figure 6.2. The frequency responses of the 5-tap sinc filter and the FIR filter $\hat{H}_{pc}(z)$ are shown together in Figure 6.3.

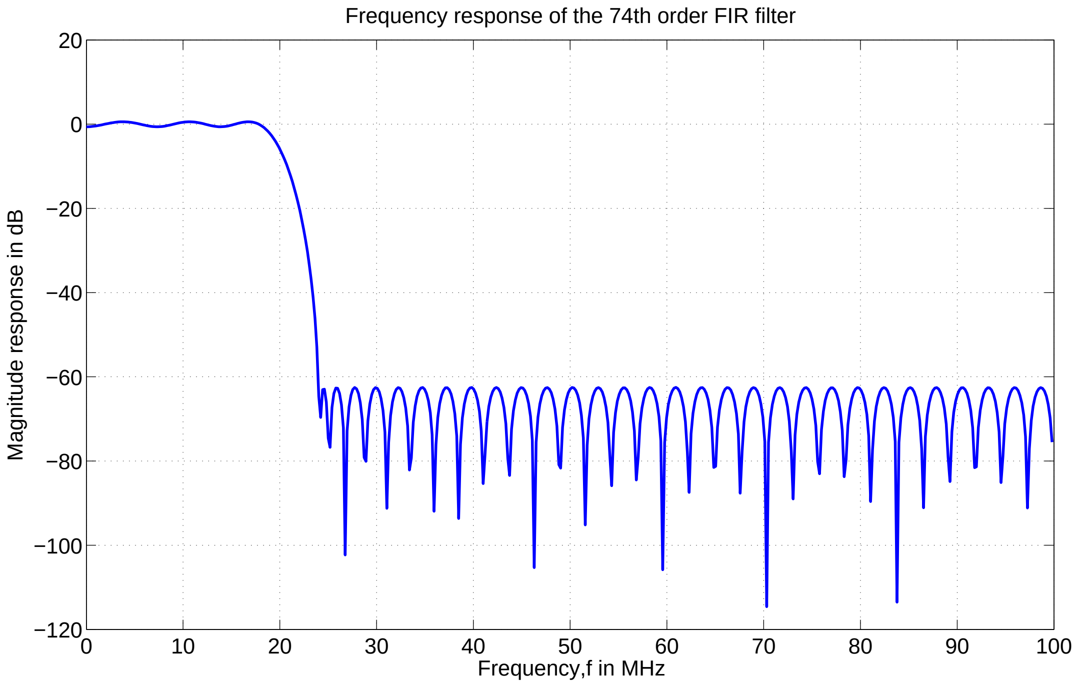


Figure 6.2: Normalized frequency response of the 74^{th} order FIR filter $\hat{H}_{pc}(z)$. Note that the attenuation is around 64 dB, much higher compared to that of $H_{pc}(z)$ obtained in Chapter 2.

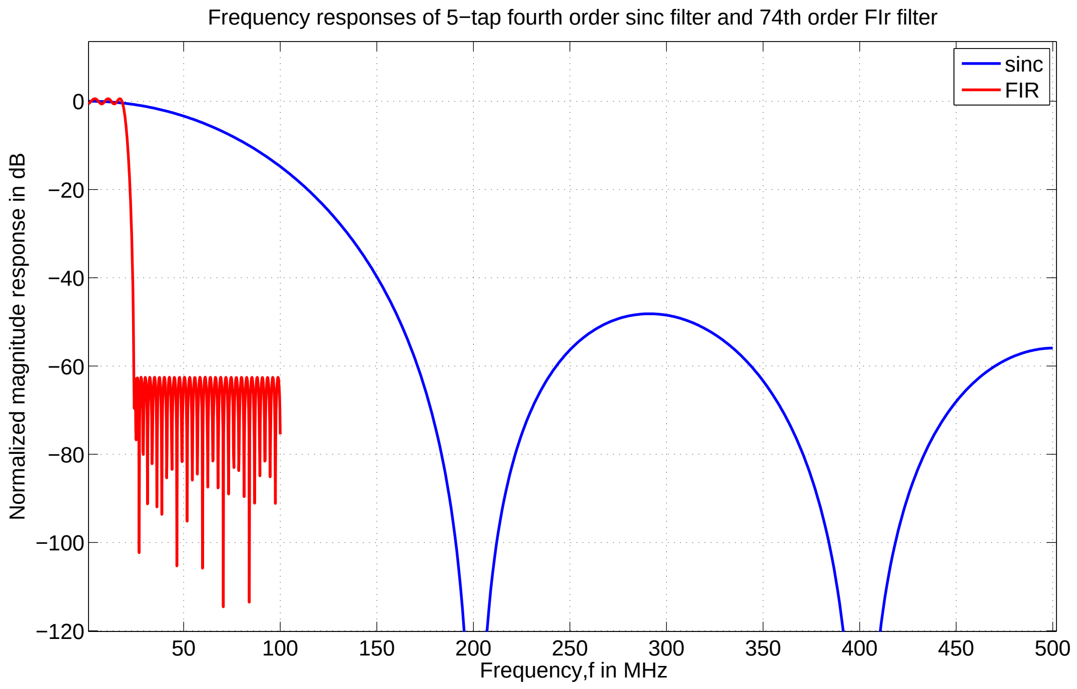


Figure 6.3: Frequency response of the 74th order FIR filter $\hat{H}_{pc}(z)$ and the 4th order 5-tap sinc filter super imposed. Note that half the sampling rate is 100 MHz and 500 MHz for FIR and sinc respectively.

6.2.1 Finite precision of the FIR filter coefficients

The FIR filter $\hat{H}_{pc}(z)$ can be written as

$$\hat{H}_{pc}(z) = \sum_{k=0}^{k=73} h_k z^{-k} \tag{6.3}$$

The coefficients h_k can be scaled just as in the earlier case of $H_{pc}(z)$. It is found that the minimum number of bits required for the largest coefficient is 13 to still obtain the required SNR.

$$h''_k = h_k \times \frac{2^{13}}{0.1984} \tag{6.4}$$

$$= h_k \times 41290.93 \tag{6.5}$$

This new scaled coefficients h''_k are rounded to the nearest integer. The final values of the filter coefficients are given in Table 6.2.

Table 6.2: Finite precision FIR filter coefficients. Only 37 filter coefficients are shown because of the symmetry $h_k = h_{73-k}$, $k = 0, 1, \ldots 73$

$h_k(z)$	Value of h_k	$h''_k = h_k \times$	round(h''_k)
h_0	0.000952031066569	39.3102	39
h_1	0.001546729111274	63.8659	64
h_2	0.002163482919353	89.3322	89
h_3	0.002224222544432	91.8402	92
h_4	0.001302324163622	53.7742	54
h_5	-0.000813321751279	-33.5828	-34
h_6	-0.003928026427016	-162.1919	-162
h_7	-0.007369251224740	-304.2832	-304
h_8	-0.010100012422552	-417.0389	-417
h_9	-0.011030359826897	-455.4538	-455
h_{10}	-0.009446253768217	-390.0446	-390
h_{11}	-0.005394856624708	-222.7586	-223
h_{12}	0.000149333848588	6.1661	6
h_{13}	0.005461515331918	225.5110	226
h_{14}	0.008595073979538	354.8986	355
h_{15}	0.008115157726663	335.0824	335
h_{16}	0.003770477274373	155.6865	156
h_{17}	-0.003187334122221	-131.6080	-132
h_{18}	-0.010238119573370	-422.7415	-423
h_{19}	-0.014393107050753	-594.3048	-594
h_{20}	-0.013336133492576	-550.6613	-551
h_{21}	-0.006498535961441	-268.3306	-268
h_{22}	0.004381881524485	180.9320	181
h_{23}	0.015535244222601	641.4647	641
h_{24}	0.022305822943639	921.0281	921
h_{25}	0.020868642279401	861.6856	862
h_{26}	0.009943471111150	410.5752	411
h_{27}	-0.008151226951782	-336.5717	-337
h_{28}	-0.027604489539197	-1139.8150	-1140
h_{29}	-0.040512878708929	-1672.8143	-1673
h_{30}	-0.039248482202936	-1620.6062	-1621
h_{31}	-0.019081036923807	-787.8737	-788
h_{32}	0.019825223708830	818.6018	819
h_{33}	0.071783498167021	2964.0073	2964
h_{34}	0.126657200403193	5229.7934	5230
h_{35}	0.172409483162752	7118.9476	7119
h_{36}	0.198397087839156	8192	8192

Table 6.3: $\hat{G}_k(z)$ definitions

$\hat{G}_k(z)$	Definition
$\hat{G}_0(z)$	$h_0 + h_5 z^{-1} + h_{10} z^{-2} + h_{15} z^{-3} + h_{20} z^{-4} + h_{25} z^{-5} + h_{30} z^{-6} + h_{35} z^{-7} + h_{33} z^{-8} + h_{28} z^{-9} + h_{23} z^{-10} + h_{18} z^{-11} + h_{13} z^{-12} + h_8 z^{-13} + h_3 z^{-14}$
$\hat{G}_1(z)$	$h_1 + h_6 z^{-1} + h_{11} z^{-2} + h_{16} z^{-3} + h_{21} z^{-4} + h_{26} z^{-5} + h_{31} z^{-6} + h_{36} z^{-7} + h_{32} z^{-8} + h_{27} z^{-9} + h_{22} z^{-10} + h_{17} z^{-11} + h_{12} z^{-12} + h_7 z^{-13} + h_2 z^{-14}$
$\hat{G}_2(z)$	$h_2 + h_7 z^{-1} + h_{12} z^{-2} + h_{17} z^{-3} + h_{22} z^{-4} + h_{27} z^{-5} + h_{32} z^{-6} + h_{36} z^{-7} + h_{31} z^{-8} + h_{26} z^{-9} + h_{21} z^{-10} + h_{16} z^{-11} + h_{11} z^{-12} + h_6 z^{-13} + h_1 z^{-14}$
$\hat{G}_3(z)$	$h_3 + h_8 z^{-1} + h_{13} z^{-2} + h_{18} z^{-3} + h_{23} z^{-4} + h_{28} z^{-5} + h_{33} z^{-6} + h_{35} z^{-7} + h_{30} z^{-8} + h_{25} z^{-9} + h_{20} z^{-10} + h_{15} z^{-11} + h_{10} z^{-12} + h_5 z^{-13} + h_0 z^{-14}$
$\hat{G}_4(z)$	$h_4 + h_9 z^{-1} + h_{14} z^{-2} + h_{19} z^{-3} + h_{24} z^{-4} + h_{29} z^{-5} + h_{34} z^{-6} + h_{34} z^{-7} + h_{29} z^{-8} + h_{24} z^{-9} + h_{19} z^{-10} + h_{14} z^{-11} + h_9 z^{-12} + h_4 z^{-13}$

6.2.2 Polyphase decomposition of the FIR filter

Similar to the decomposition of $H_f(z)$ in section, $\hat{H}_{pc}(z)$ can be decomposed as follows.

$$\hat{H}_{pc}(z) = \hat{G}_0(z^5) + z^{-1}\hat{G}_1(z^5) + z^{-2}\hat{G}_2(z^5) + z^{-3}\hat{G}_4(z^5) + z^{-4}\hat{G}_4(z^5) \qquad (6.6)$$

The five sub filters $\hat{G}_0$ to $\hat{G}_4$ are defined in Table 6.3. All the five filters can be implemented as shown in Figure 6.4, Figure 6.5 and Figure 6.6.

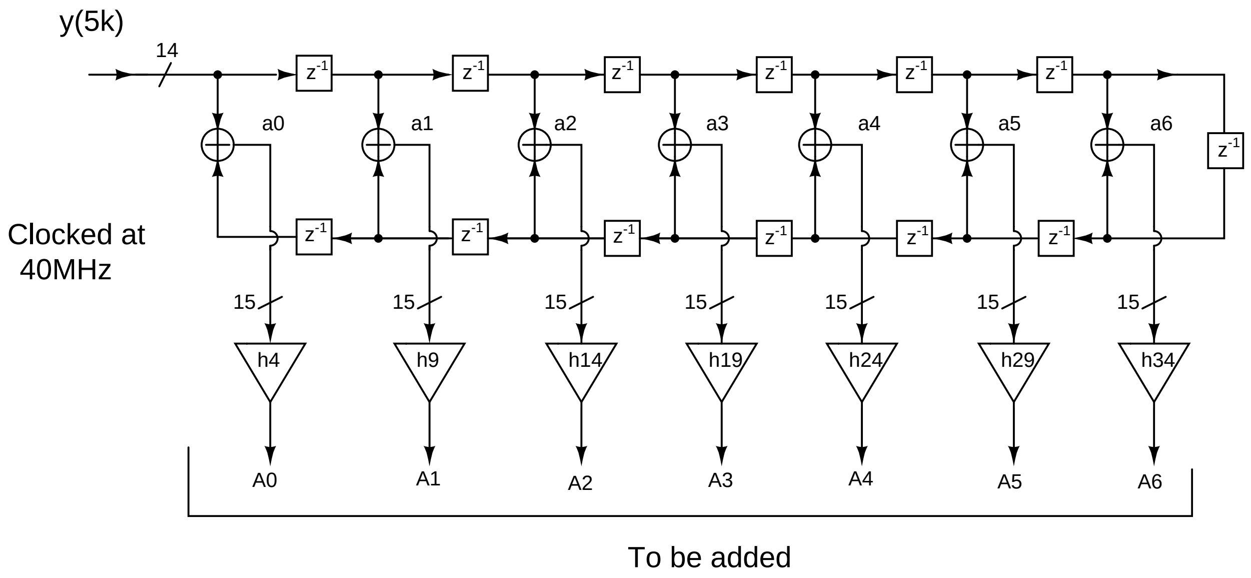


Figure 6.4: Circuit implementation of $\hat{G}_4(z)$. The shifted and added input terms a_0 to a_6 get multiplied with the filter coefficients to form partial products A_0 to A_6.

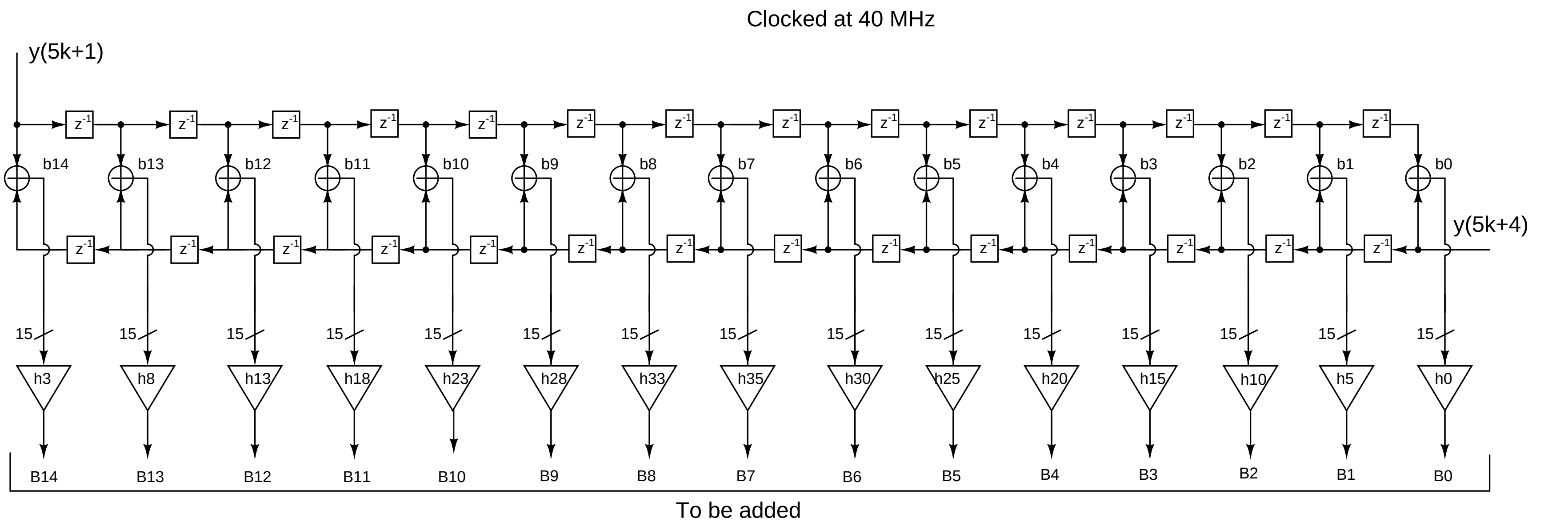


Figure 6.5: Circuit implementation of $\hat{G}_0(z)$ and $\hat{G}_3(z)$. The terms b_0 to b_{14} gets multiplied with the filter coefficients to form the partial products B_0 to B_{14}.

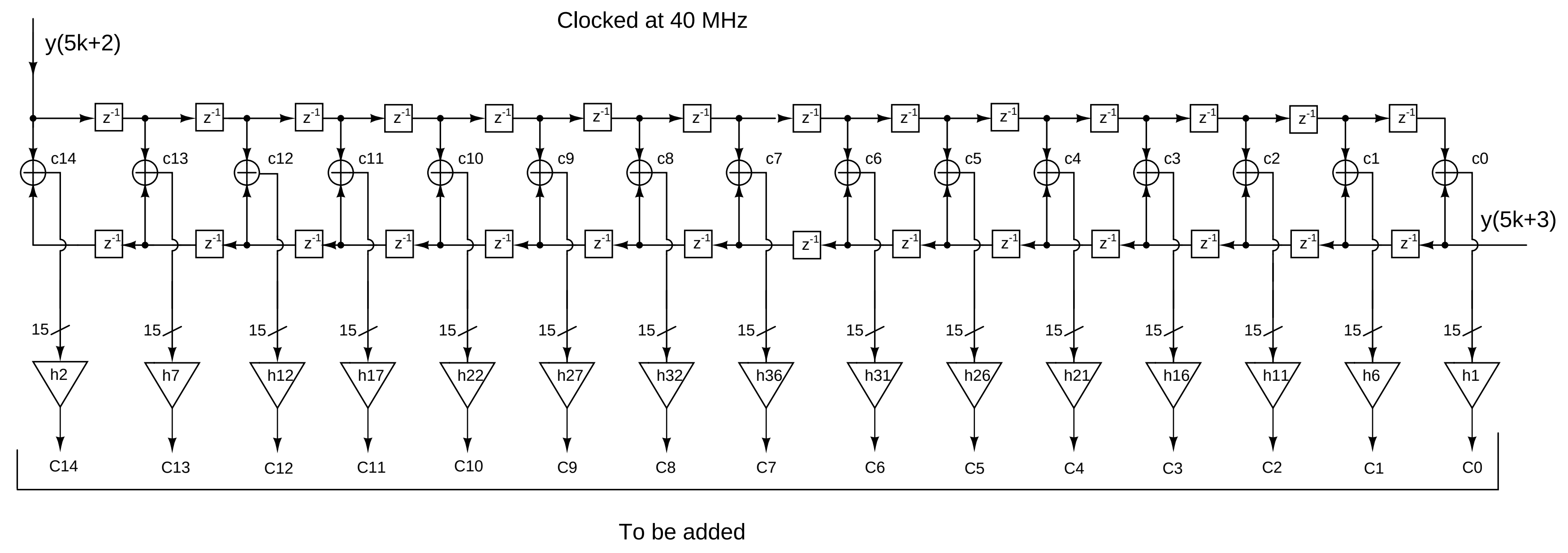


Figure 6.6: Circuit implementation of $\hat{G}_1(z)$ and $\hat{G}_2(z)$. The terms c_0 to c_{14} gets multiplied with the filter coefficients to form the partial products C_0 to C_{14}

6.3 Equalizer

Another way to reduce passband droop in a decimation filter is to add an equalizer at the end. An equalizer should have an inverse sinc response in the signal band which nullifies the droop caused by the sinc filter. The equalizer response $|H_{eq}(e^{j\omega})|$ should be $1/|H_s(e^{j\omega})|$. Consider the implemented design of the decimation filter where the 10-tap fourth order sinc response is given by

$$H_s(e^{jw}) = \left(\frac{1}{10}\left(\frac{\sin(10\omega/2)}{\sin(\omega/2)}\right)\right)^4 \tag{6.7}$$

Thus equalizer should have a response

$$H_{eq}(e^{jw}) = \left(\frac{1}{10}\left(\frac{\sin(10\omega/2)}{\sin(\omega/2)}\right)\right)^{-4} \tag{6.8}$$

$$= \left(\left(10\frac{\sin(\omega/2)}{\sin(10\omega/2)}\right)\right)^4 \tag{6.9}$$

The combined response of equalizer and sinc filter in the signal band is given by $H_s(e^{jw}) \times H_{eq}(e^{jw})$ which ideally is unity. An FIR filter that approximates equalizer's inverse sinc response in (6.8) can be obtained from MATLAB using the command *fir2*. *fir2* designs frequency sampling-based digital FIR filters with arbitrarily shaped frequency response[2, p. 462-468].

A 10^{th} order equalizer is obtained using *fir2* that minimizes the passband droop to the level of 5-tap sinc filter (i.e. $\sim -0.5dB$). The filter coefficients are given in Table 6.4. Coefficients are scaled such that the maximum number of bits required is eight (Scaling factor $= 2^8/\max(heq_k) = 233.7784$). After rounding, certain coefficients disappear, making the order of the equalizer 7. The inverse sinc response and the final equalized response are shown in Figure 6.7.

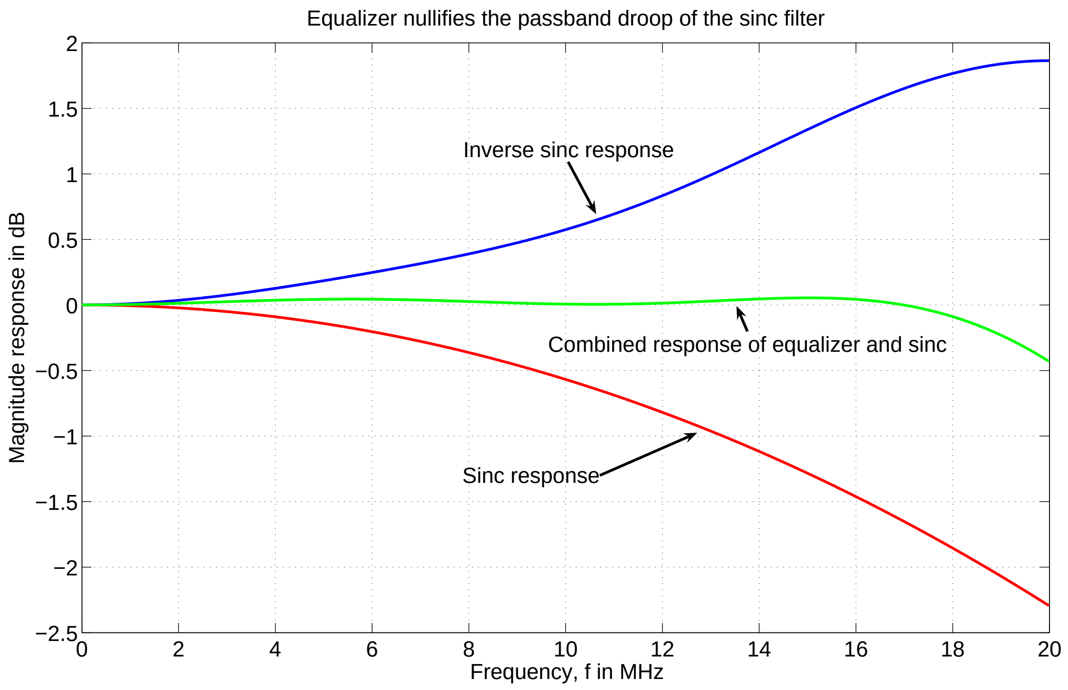


Figure 6.7: The inverse sinc response of a 7^{th} order equalizer compensates the passband droop of 2.4 dB caused by the fourth order 10-tap sinc filter ($r = 2$, $K = 4$ and $M = 5$). Note that the passband droop is minimized to the level of 5-tap sinc filter. Since the equalizer is added after the second stage FIR filter, $f_s/2 = 20\,\text{MHz}$.

Table 6.4: Finite precision equalizer coefficients. Only 6 filter coefficients are shown because of the symmetry $h_k = h_{10} - k$, $k = 0, 1, \ldots 10$

$heq_k(z)$	Value of heq_k	$heq''_k = heq_k \times 233.7784$	round(heq''_k)
heq_0	-0.000224456013123	-0.0524	0
heq_1	0.000729824579280	0.1706	0
heq_2	-0.003050302562090	-0.7130	-1
heq_3	0.011414266357615	2.668	3
heq_4	-0.054046950045288	-12.6350	-13
heq_5	1.095054038012276	256	256

6.4 Conclusion

After simulations, it is found that the FIR filter in second stage with 25 ns clock has a huge positive slack. Thus in retrospect, it is obvious that even if a 74^{th} order FIR filter is implemented, timing constraints would have met. This would have made the passband droop of the decimation filter only -0.5 dB.

Another way to reduce the droop is to add an equalizer after the 60^{th} order FIR filter described in chapters 2 and 4. As seen in the previous section, a 7^{th} order equalizer working at 20 MHz is sufficient to reduce the droop. On balance, this seems be a better alternative to using a sinc filter with r=1 followed by a 74^{th} order FIR filter.

CHAPTER 7

Results and Conclusion

7.1 Simulation and results

The design and implementation details of the decimation filter in two stages using polyphase decomposition are discussed in chapters 2, 3, 4 and 5. The actual circuit is modelled using the Hardware Description Language, Verilog and functional verification is done in Modelsim by writing appropriate test benches. Once this ideal circuit gives the same SNR obtained in Matlab for the same input stream, we can go ahead to the final steps of synthesizing and routing the circuit.

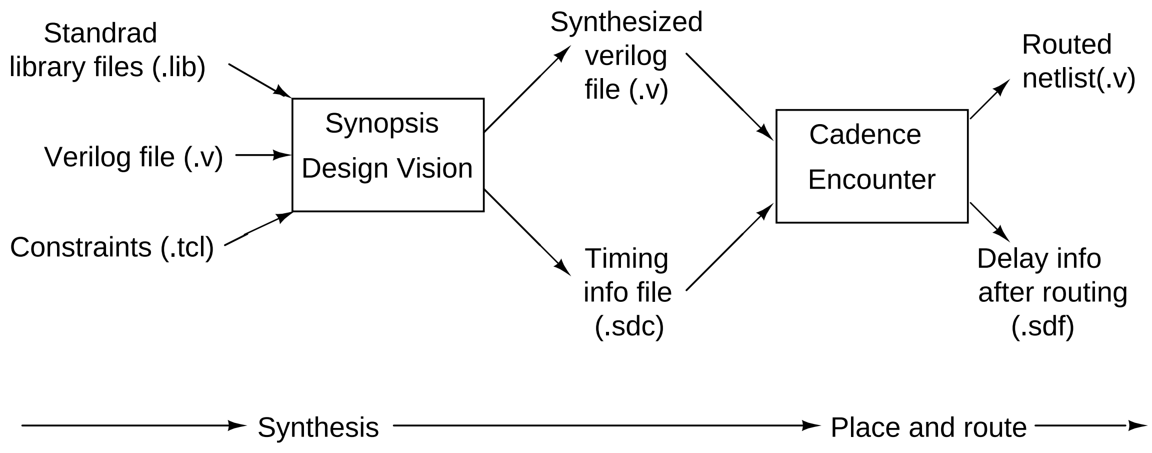


Figure 7.1: Block diagram of the design flow

The digital circuitry behaviorally modelled in verilog is to be mapped into actual circuit components based on standard cell libraries and constraints. This procedure is called synthesis and is performed using the Synopsis synthesizing tool *Design Vision*. Design vision gives the synthesized verilog file as output by reading three inputs (Figure 7.1).

1. Verilog code for the circuit

Table 7.1: Power report of the decimation filter(10 tap sinc filter + 60th order FIR filter) obtained using Synopsis design vision(Typical case)

Module	Clock	Power	% Power
Sinc filter	200 MHz (5 ns)	4.887mW	25.9
FIR filter	40 MHz (25 ns)	10.367mW	54.9
Decimator for sinc	1 GHz (1 ns)	1.122mW	5.9
Decimator for FIR	200 MHz (5 ns)	0.784mW	4.2
Reset synchronizer	1 GHz (1 ns)	0.166mW	0.9
Total		18.887 mW	100%

2. Standard library files
3. Constraints

The verilog code is a behavioural description of the circuit. The standard cell libraries from UMC provide standard building blocks of the circuit in CMOS 180 nm technology. The constraints are given to the design vision using a .tcl script file. There are mainly three constraints for the decimation filter circuit.

1. Meet all the timing conditions i.e. the delay should be less than 5 ns and 25 ns for the first and second stage of decimation filter respectively.
2. Minimize the area of the circuit
3. Minimize the power consumption of the circuit

Once the three input files (verilog, libraries and constraints) are ready, design vision compiles the input and creates a synthesized verilog file which is now based on standard building blocks. Design vision also gives a timing information file (.sdc). These files can be read using *Cadence encounter* to place and route (Figure 7.1). After placement and routing, the routed netlist can be saved as a verilog file and the timing information after routing can be extracted into a .sdf file both of which can be used for the post-route simulations.

The final simulation results are summarized in Table 7.1 and 7.2. From Table 7.1, it can be seen that most of the power is dissipated in the second stage FIR filter. After simulations, it is found that the FIR filter has a huge positive slack.

Table 7.2: Decimation filter design summary

Technology	0.18μm CMOS
Supply Voltage	1.8 V
Total power	18.887 mW
Area	$\approx 758 \times 650$
Passband droop	-2.4 dB at 20 MHz
Simulated SNR	93.0482 dB

This means that the power consumption the FIR filter alone can be reduced much further by decreasing the power supply voltage for FIR filter block alone.

7.2 Conclusion

The total power consumed by the entire decimation filter is 18.887 mW. Thus polyphase implementation is proved to be very useful in high speed low power decimation filters. The low power of the decimation filter is attributed mainly to the decrease in the operating frequency of the circuit in polyphase. The price paid for the low power are the higher area in polyphase compared to the CIC implementation of sinc and the increased design complexity in both the sinc and FIR filters.

The continous time $\Delta\Sigma$ modulator for which the decimation filter is designed consumes about 50 mW power. Low power decimation filter described in this thesis can be used to turn the continous time $\Delta\Sigma$ modulator into a complete Analog to Digital Converter system with only 38% additional power. Amoeba view of the final layout is given in Figure 7.2.

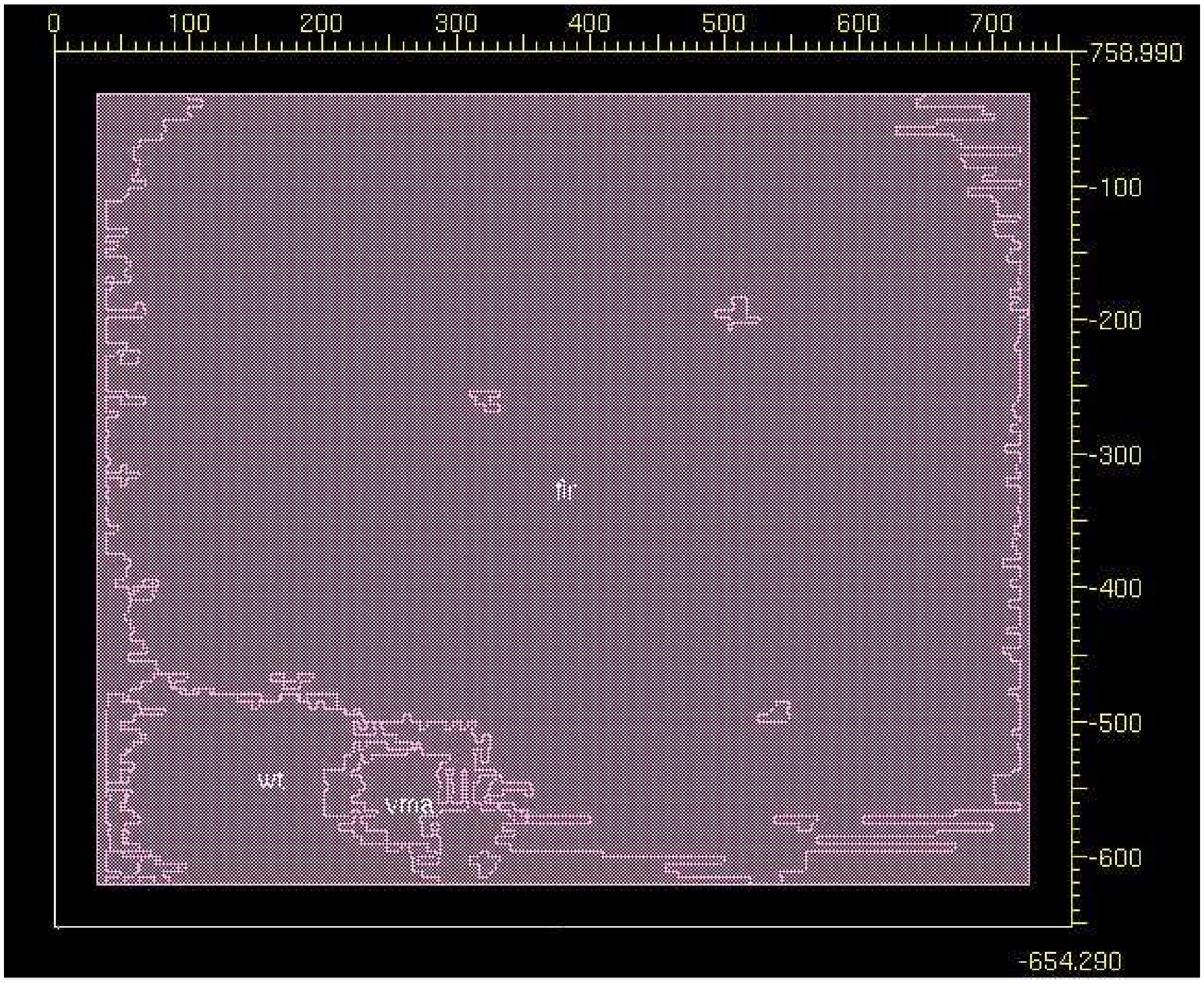


Figure 7.2: Amoeba view of the circuit after layout in encounter

APPENDIX A

Decimation Filter Implementation using CML in First Stage

It is seen in chapter 2 that it is not possible to implement the sinc filter in CIC form using CMOS standard cell libraries due to the timing condition which requires a 14-bit addition to be performed within 1 ns. Recall that we adopted the polyphase architecture to counter this problem even though it is more complicated and time consuming for the designer. Here, we discuss a different method explored for implementing the decimation filter.

A.1 CIC implementation using Carry Save Accumulators

The challenge in the design is the requirement that the N-bit addition in the accumulators in CIC sinc filter needs to be performed in just 1 ns. While it is true that this is unachievable using conventional N-bit adders, there are other accumulator structures which can perform the accumulation with a fixed delay independent of the number of bits N. One such structure is Carry Save Accumulator (CSA) which will be discussed in the following section.

A.1.1 CSA for the first accumulator in CIC

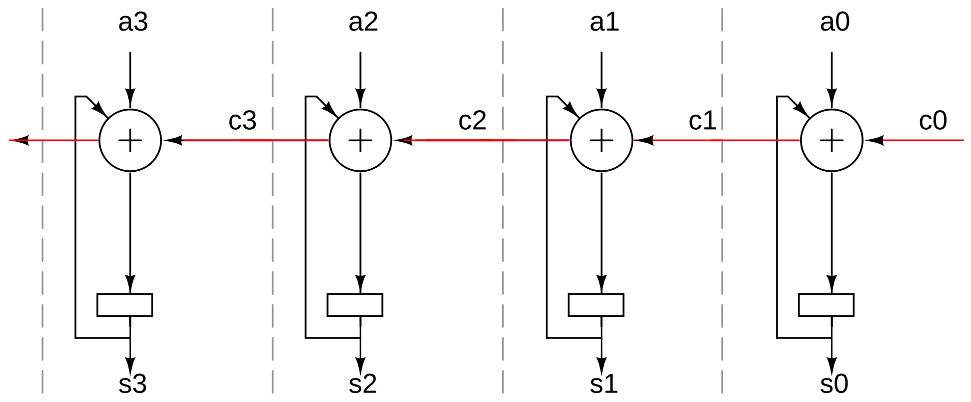
The simplest accumulator one can think of is the one in which input is added with the previous sum with an N-bit Ripple Carry Adder (RCA) as shown in

Figure A.1(a). In such a structure, for the i^{th} fulladder (i=0,1,..N-1),

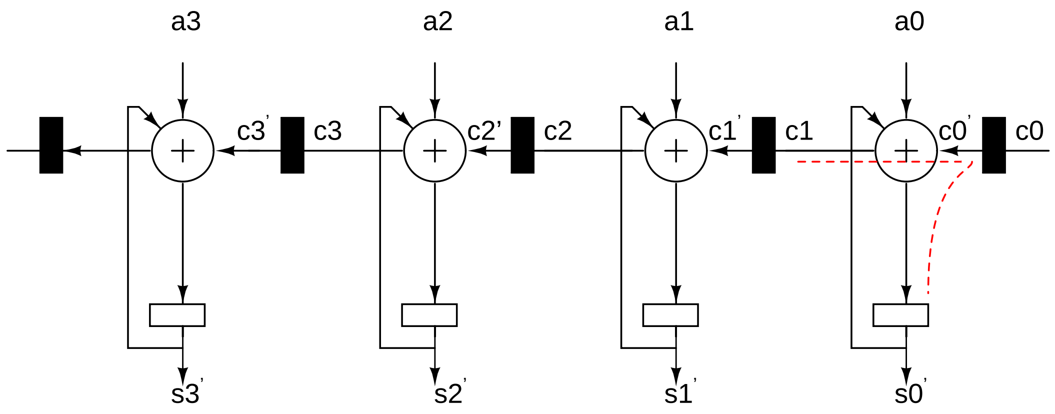
$$(a_i + c_i + s_i)z^{-1} = s_i \tag{A.1}$$

$$\Rightarrow s_i = a_i(\frac{z^{-1}}{1 - z^{-1}}) + c_i(\frac{z^{-1}}{1 - z^{-1}}) \tag{A.2}$$

Here, due to carry propagation in the RCA, the Critical Path Delay (CPD) is proportional to the number of input bits N. To decrease the CPD, the feedforward path of the carry (marked red in Figure A.1(a)) can be pipelined as shown in Figure A.1(b).



(a) Accumulator with ripple carry adder.



(b) Pipelined accumulator

Figure A.1: A simple accumulator based on ripple carry adder is pipelined to reduce the critical path delay (CPD). After pipelining, the delay is only 1 fulladder delay + 1 register delay.

For the pipelined accumulator in Figure A.1(b), the equations of additions are

$$(a_i + c_i' + s_i')z^{-1} = s_i' \tag{A.3}$$

$$\Rightarrow s_i' = a_i(\frac{z^{-1}}{1-z^{-1}}) + c_i'(\frac{z^{-1}}{1-z^{-1}}) \tag{A.4}$$

Since $c_i' = c_i z^{-1}$,

$$s_i' = a_i(\frac{z^{-1}}{1-z^{-1}}) + c_i(\frac{z^{-2}}{1-z^{-1}}) \tag{A.5}$$

From (A.2) and (A.5), note that $(s_i - s_i') = c_i z^{-1}$ which is equal to c_i'. This means that from the pipelined circuit, in order to get the original sum s_i, we have to add up s_i' and c_i'. i.e.

$$s_i = s_i' + c_i' \tag{A.6}$$

The important fact is that the first accumulator in CIC sinc can be implemented at a much less delay by pipelining the carry path. These high speed pipelined accumulators are called Carry Save Accumulators (CSAs)[13]. If the first accumulator is implemented using CSA, the Critical Path Delay (CPD) is just one fulladder delay (neglecting register delays). Note that the delay of the CSA is independent of the number of input bits N, making CSA useful for high speed accumulators.

A CSA used as the first stage of the sinc filter is shown in Figure A.2. Note that Figure A.2 is simply Figure A.1(b) redrawn. Even though the input to the CSA shown can be 14 bits (x_0 to x_{13}), the output from the $\Delta\Sigma$ modulator has only 4 bits. Thus x_4 to x_{13} is logic '0' if we assume the input in unsigned integer format.

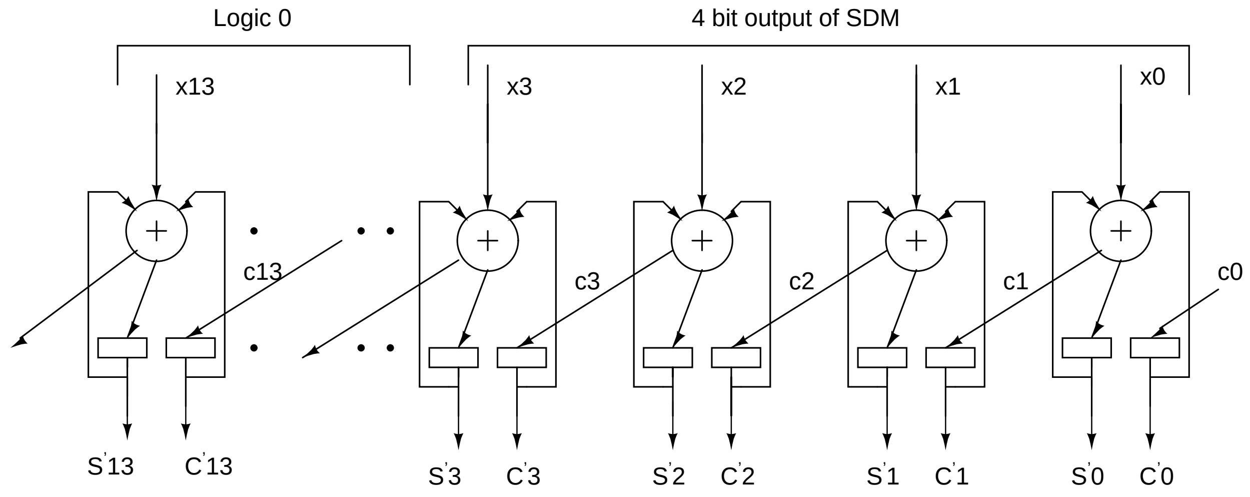


Figure A.2: First accumulator in CIC structure implemented using Carry Save Accumulator. Critical Path Delay, CPD = 1 fulladder delay + 1 register delay.

A.1.2 CSA for the later stage accumulators in CIC

The only problem with using CSA in the first accumulator of CIC is that instead of getting a single sum (as in a conventional accumulator with an N-bit RCA), CSA gives two vectors, a sum vector and a carry vector as outputs ($\bar{s'}$ and $\bar{c'}$), the sum of which is the actual accumulated output as given in (A.6). Due to this, from the second accumulator onwards, we need a different kind of carry save accumulators which takes two vectors as inputs. These two-input two-output(2I2O) CSA is shown in Figure A.3 [13].

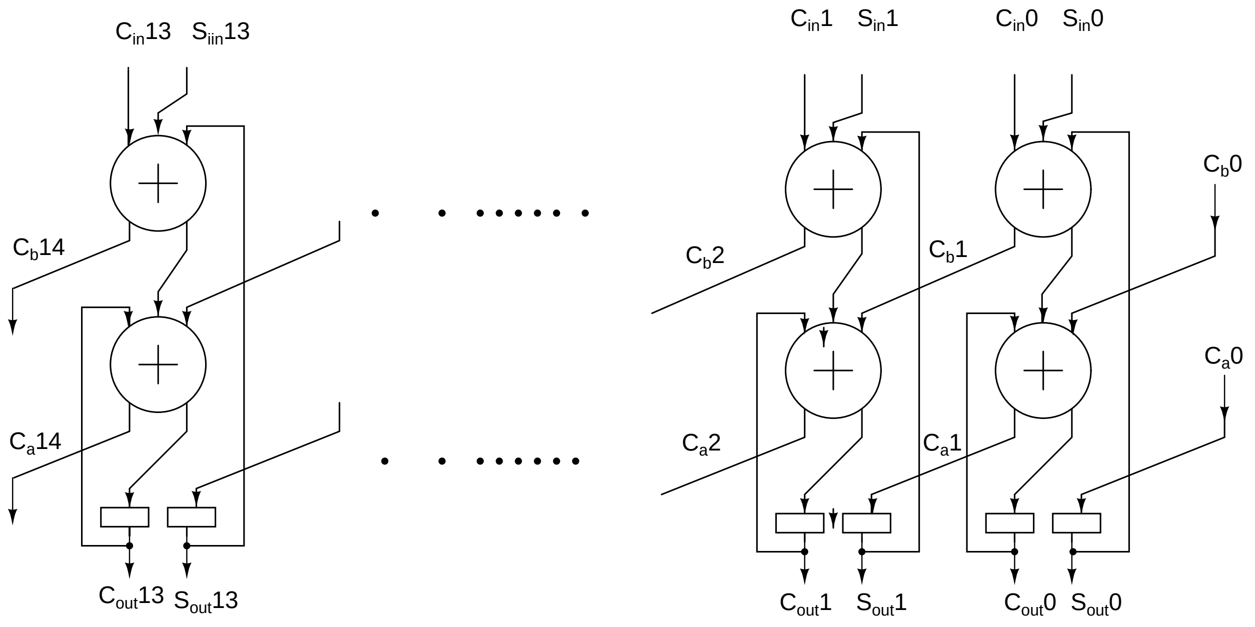
The equations of addition for the 2I2O CSA in Figure A.3can be written as

For the i^{th} bit position,

$$(s_{in}i + c_{in}i + c_{out}i + c_bi + s_{out}i)z^{-1} = s_{out}i \tag{A.7}$$

$$\Rightarrow s'_{out}i = (s_{in}1 + c_{in}i + c_{out}i + c_bi)(\frac{z^{-1}}{1-z^{-1}}) \tag{A.8}$$

(A.9)



C_{in}13 S_{iin}13
C_{in}1 S_{in}1
C_{in}0 S_{in}0
C_b0
C_b14
C_b2
C_b1
C_a0
C_a14
C_a2
C_a1
C_{out}13 S_{out}13
C_{out}1 S_{out}1
C_{out}0 S_{out}0

Figure A.3: Implementation of second accumulator onwards in CIC structure using CSA. Critical Path Delay, CPD = 2 fulladder delays + 1 register delay.

Also

$$c_{out}i = c_a i z^{-1} \tag{A.10}$$

Again, the actual accumulated sum is $\overline{s_{out}} + \overline{c_{out}}$. This circuit can be further pipelined and retimed as shown in Figure . Then the equations become

$$\text{For the} i^{th} \text{bit position,} (s_{in} i z^{-1} + c_{in} i z^{-1} + c'_{out} i + c'_b i + s'_{out} i) z^{-1} = s'_{out} i \tag{A.11}$$

$$\tag{A.12}$$

Substituting $c'_{out}i = c_a i z^{-2}$ and $c'_b i = c_b i z^{-1}$ in the above equation, we get

$$s'_{out} i = (s_{in} i + c_{in} i + c_a i z^{-1} + c_b i)(\frac{z^{-2}}{1 - z^{-1}}) \tag{A.13}$$

which is exactly one time unit delayed than the non-pipelined output. i.e. $s'_{out} = s_{out} z^{-1}$. Since $c'_{out} = c_{out} z^{-1}$, the final output of the pipelined circuit is just one time unit delayed version of the original circuit. Advantage is that we have reduced the critical path delay to just one fulladder delay plus one register delay.

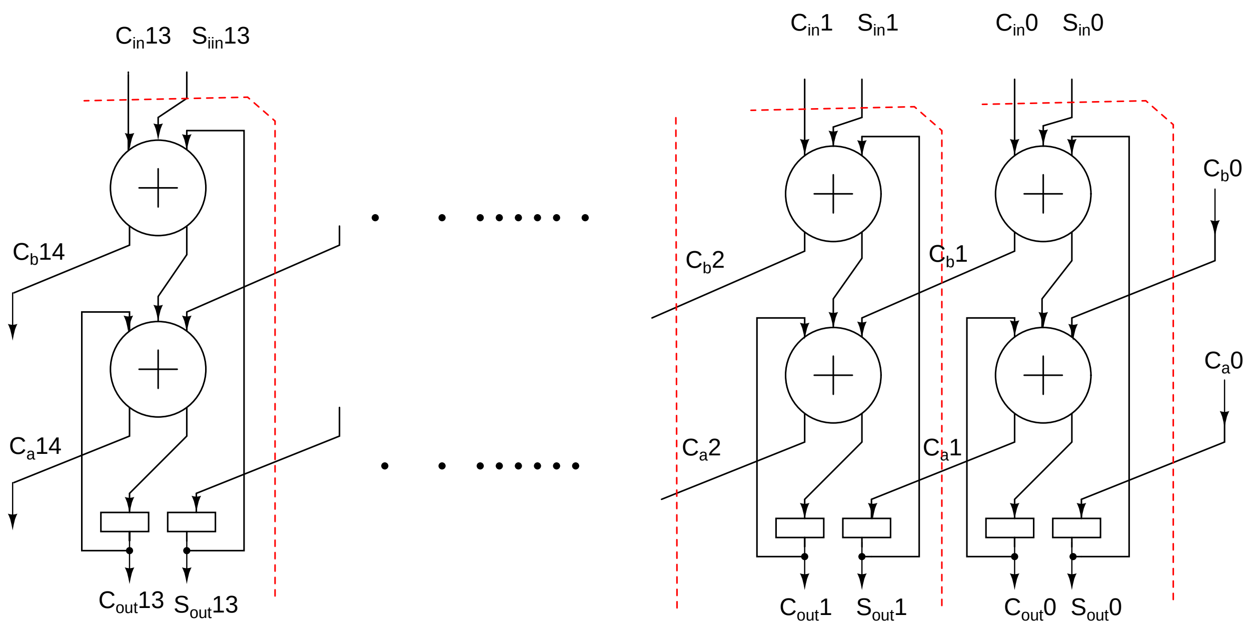


Figure A.4: Pipelining the 2I2O accumulator in the second and later stages of CIC

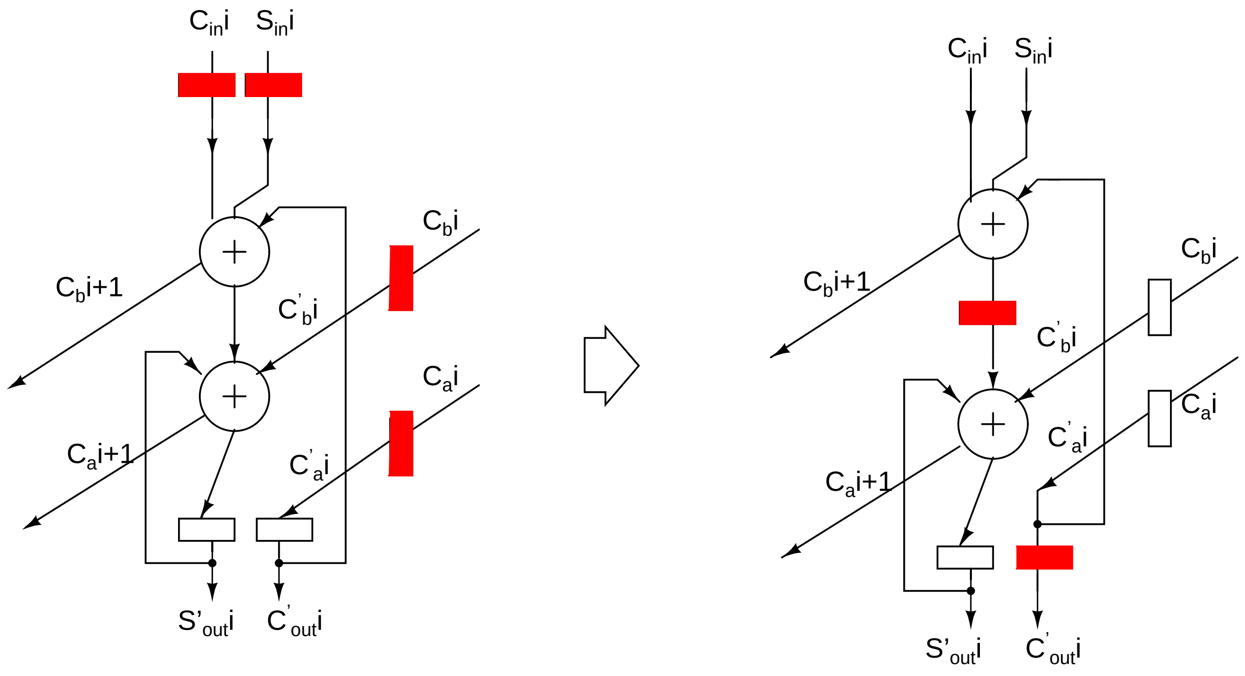


Figure A.5: Retiming the 2I2O accumulator in the second and later stages of CIC

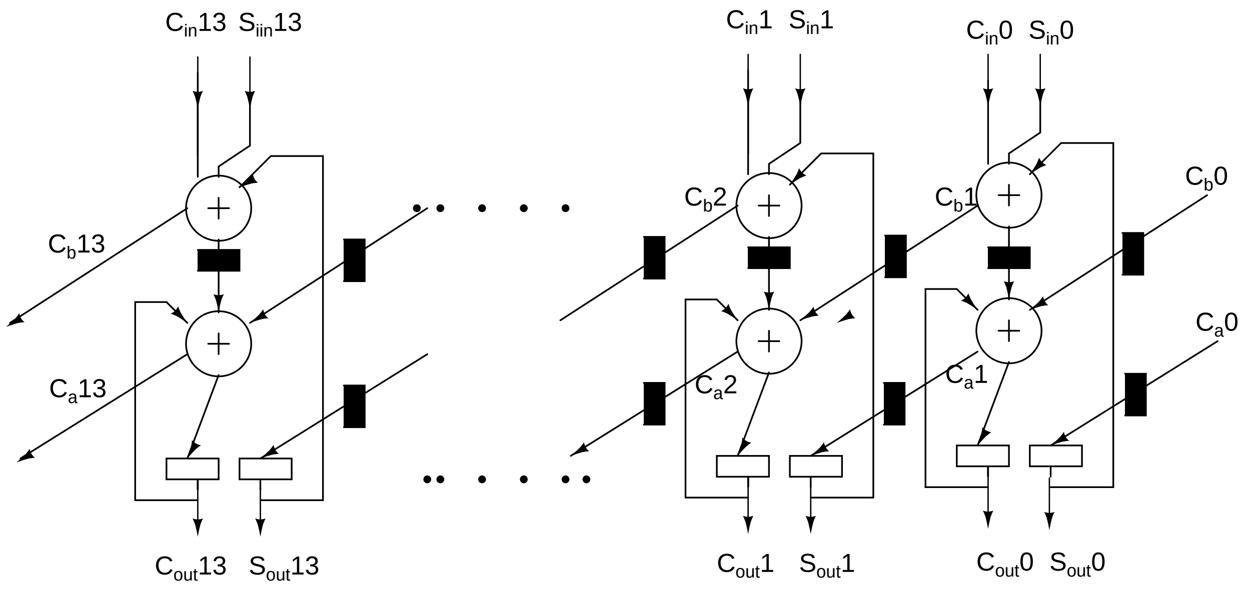


Figure A.6: Carry Save Accumulator from second stage onwards in sinc filter. Pipelined and retimed so that Critical Path Delay, CPD = 1 Fulladder delay + 1 Register delay.

The 2I2O CSA has a CPD of two fulladder delays (neglecting register delays). This critical path delay can be further reduced to 1 fulladder delay by pipelining and retiming as shown in Figure A.6. Thus using CSAs, the simple CIC architecture of the sinc filter can be implemented as long as the sampling time period is less than the sum of two fulladder delays and one register delay. The output at the end of integrator chain consists of two vectors $\overline{S_{out}}$ and $\overline{C_{out}}$, the sum of which is the actual output of the accumulator cascade. This final addition of $\overline{S_{out}}$ and $\overline{C_{out}}$ can be performed later in the low frequency region.

A conventional fulladder will have three gate delays. This means, in order to use the CIC architecture for sinc filter, we need to have six gate delays plus one register delay to be less than 1 ns. Even this is very stringent timing condition in CMOS technology and the synthesized circuit in *Synopsis Design Vision* will not meet the timing constraints. Thus we have two options: Either go for a faster technology or logic style or abandon the simple CIC architecture in search for a better structure.

In the earlier chapters, it is seen that to implement the decimation filter, we did abandon the CIC form in favor of polyphase decomposition. Sticking to 180 nm, another option available is to implement the CIC form using carry save accumulators using a faster logic style. An example of a faster logic style is the Current Mode Logic (CML) where we trade off power for performance. Thus if the CIC structure of the sinc filter is implemented using CSAs made of gates in current mode logic, it will meet the timing constraints by consuming higher power. A preliminary analysis of implementation of sinc filter in CML is discussed in the subsequent sections.

A.2 Implementation in CML

To implement the accumulators, we need two building blocks. Full adders and registers. Circuit details of these blocks are given in the following sections.

A.2.1 Adders

A conventional fulladder is made of two half adders and an OR gate as shown in Figure A.7. The XOR, AND and OR logic gates in the fulladder are made in CML as shown in Figure A.8 and Figure A.9.

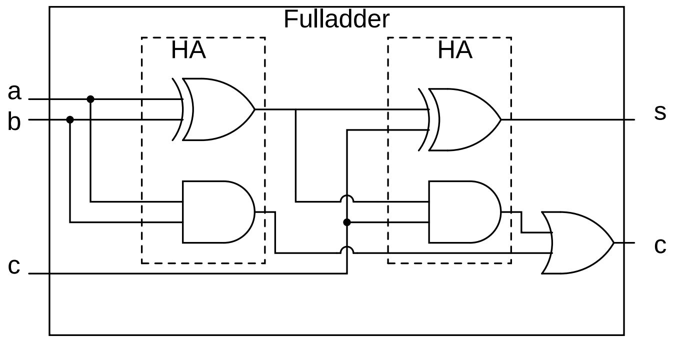


Figure A.7: A fulladder implementation using two half adders and an OR gate

The transistor level implementations of logic gates given in Figures A.8 and A.9 are simulated in cadence and the static power consumption from a 1.8 V supply for the three corners of resistances are given in Table A.1. The static current varies with the value of R, hence the variation in power. All simulations are done using ss-transistors at 80^o Celsius. The value $2I_0R$ is chosen to be 0.8 V.

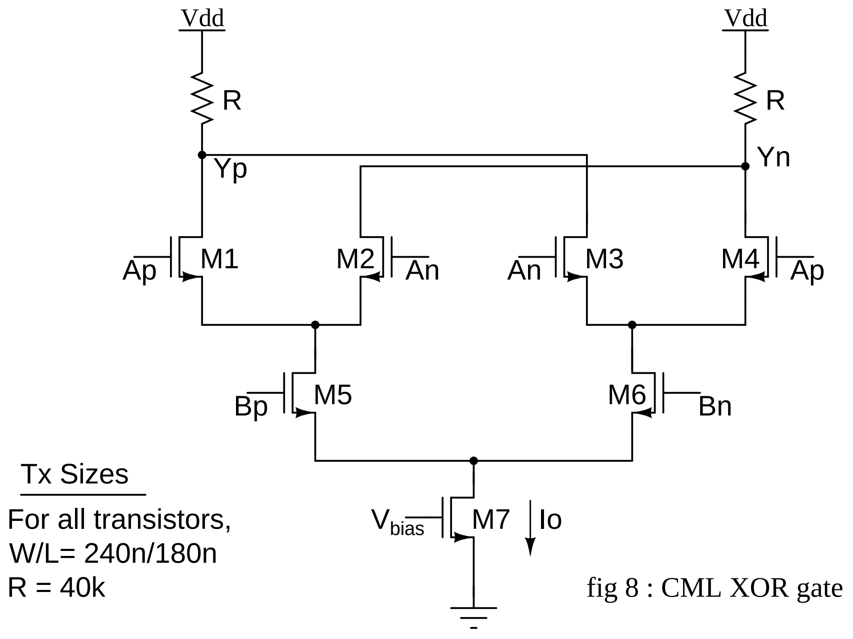


Figure A.8: XOR gate implementation in CML

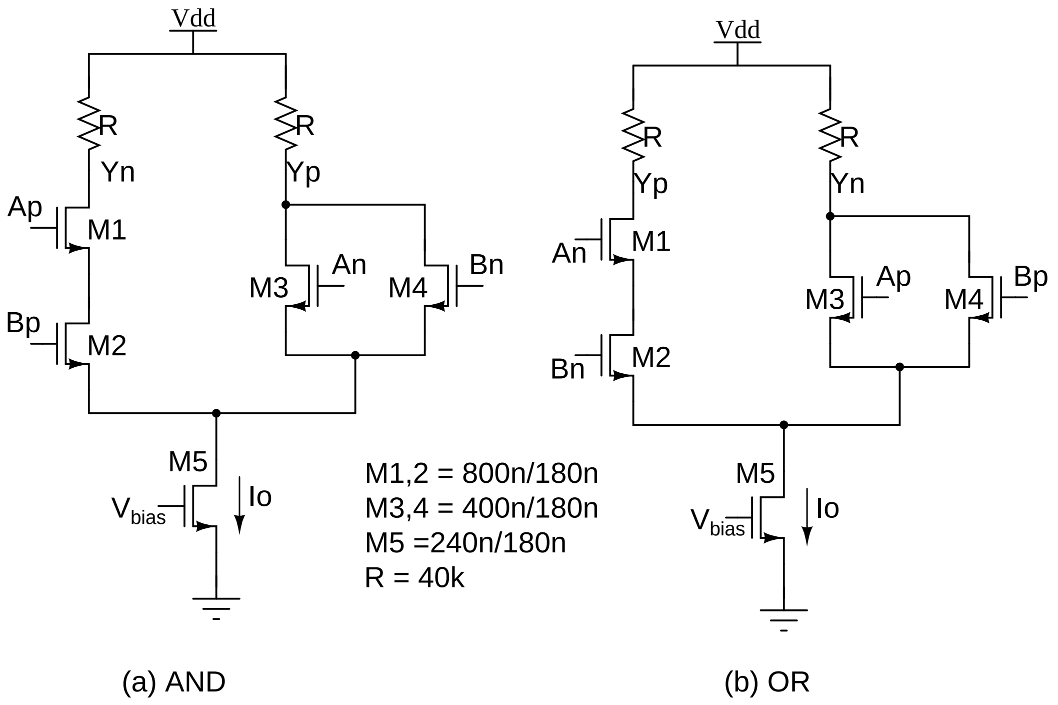


Figure A.9: AND and OR gate implementation in CML

Table A.1: Static power consumption for XOR, AND and OR gates

Static power consumption	res_max	res_typ	res_min
XOR	14.41 μW	17.4 μW	22.1 μW
ANDOR	15.21 μW	18.35 μW	23.45 μW

Table A.2: Static power consumption of the D-Flip flop and the power generation circuit

Static power Consumption	res_max	res_typ	res_min
D-Flipflop	50.11 μW	63.3 μW	85.5 μW
Bias Generation Circuit	98.99 μW	120 μW	156.4 μW

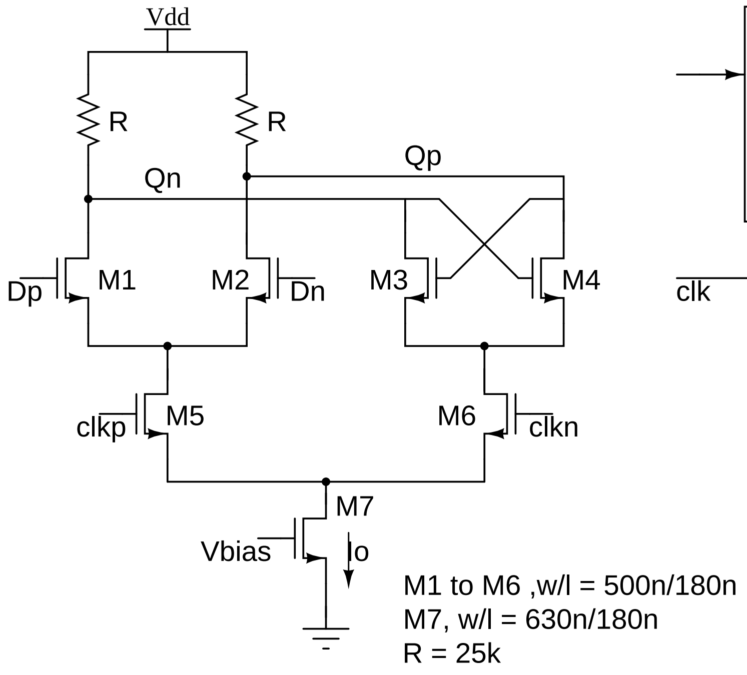
A.2.2 Registers and bias generation

A one bit register required in the accumulator is made using two latches in Master-Slave configuration. The circuit given in Figure A.10 is simulated for static power consumption and the results are given in Table A.2. Again the simulations are done for the power supply $V_{dd} = 1.8\,\mathrm{V}$, ss-transistors at 80^o Celsius. The differential output swing $2I_0R$ of the CML is chosen to be 0.8 V.

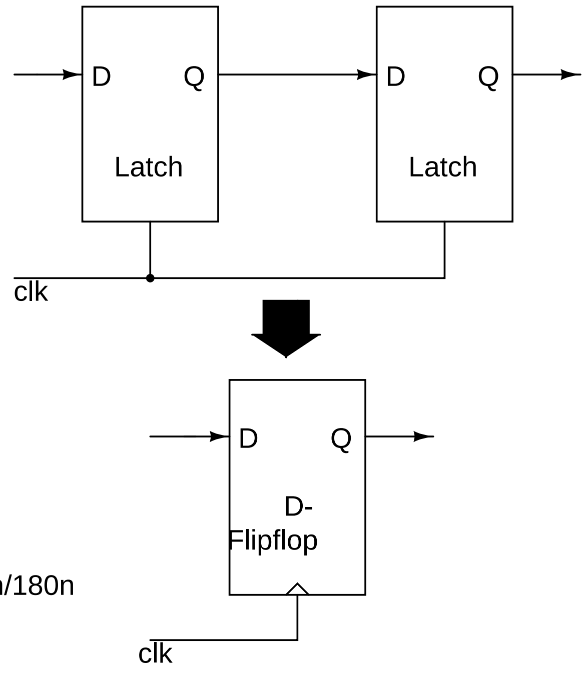
To keep the differential output swing of the CML gates $(2I_0R)$ independent of the variations in R, a bias generation circuit shown in Figure A.11 is used. The current $(V_{dd} - V_{cm})/R$ generated by this bias generation circuit is mirrored to the tail transistors of other gates. This makes I_0 inversely proportional to R and $2I_0R$ is 0.8 V independent of R variations. This also explains lower power for maximum R. I_0 decreases as R increases which in turn decreases static power $(V_{dd} \times I_0)$.

A.3 Results and conclusion

A single bit accumulator to be used in the first accumulator of CIC is built using the building blocks discussed in the previous sections and the performance of the

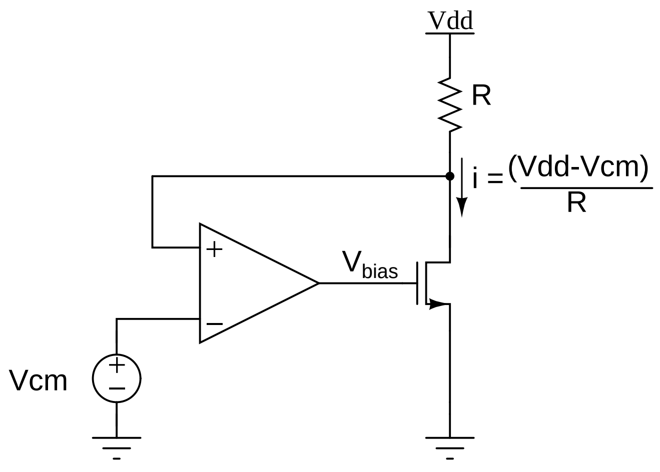


(a) CML implementation of a latch



(b) Master-slave D flipflop

Figure A.10: The CML implementation of a latch and the implementation of D-Flipflop using two latches in Mater-Slave fashion.



Bias generation which keeps output swing independent of R variations

Figure A.11: Bias generator circuit

Table A.3: Static power and timing analysis of a single bit accumulator in the first accumulator of CIC

Single bit accumulator	res_max	res_typ	res_min
Static power	321.5 μW	397.4 μW	521.5 μW
CPD (1 fulladder) + 1 register delay)	633 ps	518 ps	447 ps

circuit is summarized in Table A.3. The critical path delay(CPD) shown in the table is the time required to reach 90% of the final value. Note that even the worst case delay of 633 ps is much lower than 1 ns. Thus the accumulator can be made to work easily at 1 GHz.

Using these data, the static power consumption for the entire fourth order sinc filter working at 1 GHz can be estimated. This is found to be greater than 23 mW, 28 mW, and 37 mW for the maximum, typical and minimum resistance corners respectively. Note that this is only for the first stage of the decimation filter. As we have seen in Chapter 7, this power consumption number is much higher than that obtained using polyphase decomposition thereby justifying the choice of higher design complexity in polyphase over simple CIC design for low power.

APPENDIX A

Asynchronous reset

It is seen in chapter 5 that for clock dividers, an asynchronous reset is used. To avoid metastability problems while de-asserting the reset, a standard reset synhronizer circuit as shown in Figure A.1 is used for both the ring counters. Also a condition that reset should be asserted during the negative edge of the 1 ns clock is also specified to ensure robust performance.

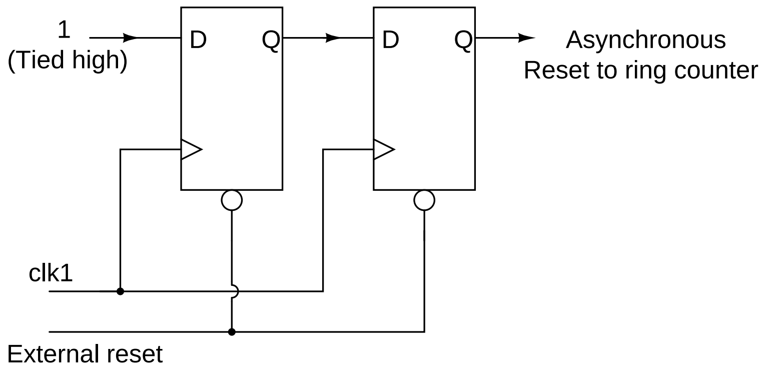


Figure A.1: Reset synchronizer circuit used to ensure proper asynchronous reset. For the second ring counter used for FIR filter, a similiar reset synchronizer circuit is required with clk5 (5 ns clock) instead of clk1 (1 ns clock).

REFERENCES

[1] R. Schreier and G. C. Temes, *Understanding Delta-Sigma Data Converters.* John Wiley and Sons Inc, 2005.

[2] S. K. Mitra, *Digital Signal Processing.* 3^{rd} ed.

[3] R. Schreier, *The Delta-Sigma Toolbox Version 7.3.* July 2009.

[4] R. E. Crochiere and L. R. Rabiner, "Interpolation and decimation of digital signals - a tutorial review," in *Proceedings of the IEEE*, vol. 69, pp. 300–331, Mar 1981.

[5] J. C. Candy, "Decimation for sigma delta modulation," in *IEEE Trans. on Communications*, vol. COM-34, pp. 72–76, Jan 1986.

[6] E. B. Hogenauer, "An economical class of digital filters for decimation and interpolation," in *IEEE Trans. on Acoust., Speech and Signal Processing*, vol. ASSP-29, pp. 155–162, April 1981.

[7] N. R. Doppalapudi, "Decimator, interpolator, and dem techniques for oversampling $\Delta\Sigma$ data converters," Master's thesis, IIT Madras, 2006.

[8] O. Gustafsson and H. Ohlsson, "A low power decimation filter architecture for high-speed single-bit sigma-delta modulation," in *IEEE Inter. Symp. on Circ. and Sys.*, vol. 2, pp. 1453–1456, 2005.

[9] C. S. Wallace, "A suggestion for a fast multipler," in *IEEE Trans. Electron. Comput.*, vol. EC-13, pp. 14–17, 1964.

[10] H. Aboushady, Y. Dumonteix, M.-M. Louerat, and H. Mehrez, "Efficient polyphase decomposition of comb decimation filters in $\delta\sigma$ analog-to-digital converters," in *IEEE Trans. on Circuits and Systems*, vol. 48, pp. 898–903, Oct 2001.

[11] J. M. Rabaey, A. Chandrakasan, and B. Nikolic, *Digital Integrated Circuits.* 2^{nd} ed.

[12] P. M. Kogge and H. S. Stone, "A parallel algorithm for the efficient solution of a general class of recurrence equations," in *IEEE Trans. on Computers*, vol. C-22, pp. 783–791, 1973.

[13] K.-Y. Khoo, Z. Yu, and J. Alan N. Wilson, "Efficient high-speed cic decimation filter," in *Proceedings of the IEEE, ASIC Conference*, pp. 251–254, Mar 1998.