COMPLEXITY REDUCTION IN FIR FILTERS USING INTEGER LINEAR PROGRAMMING TECHNIQUES

By:
SAIF UR TAIMUR

(2006-NUST-MSC-PhD-ComE-10)

Supervised By:
Dr SHOAB A KHAN

This Thesis is submitted in partial fulfillment of requirements for the degree of MS in Computer Engineering

Department of Computer Engineering
ACADEMIC STUDIES GROUP, COLLEGE OF E&ME
NATIONAL UNIVERSITY OF SCIENCE AND TECHNOLOGY
2008

Space for Attachment of TH-4 Form

ACKNOWLEDGEMENTS

I Thank Allah Almighty for giving me the ability to be become what I am today……

My sincere Appreciation and gratitude goes to my thesis adviser, Dr Shoab Ahmad Khan, for his professional guidance through out my thesis and for his moral support and encouragement during my hopeless hours in this work. I am also grateful to my thesis committee members Dr Muhammad Younis Javed, (Head of the Computer Department), Dr Assia Khanum and Mr Kaleem for their guidance and devotion.

I also appreciate and acknowledge the time given to me by Dr Sohail Sadiq during his efforts to acquaint me with the basics of Linear Programming and filter design using a Linear Program, Miss Ambreen Sheikh and Mr. Peter Notebart (moderator LP Yahoo Groups) for their help in introduction and in teaching me concepts of Linear Programming respectively.

I extend profound thanks to my friends and family members. Without their help and cooperation, I would not be able to complete my work.

DEDICATIONS

I dedicate my work to my parents and my wife…..

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

Complexity reduction in FIR filters implies to design techniques used to optimize performance of an FIR filter circuit by reducing its structural complexity. These optimizations improve performance parameters like power consumption, total area occupied by the circuit, when fabricated on a chip and its processing speed. The process involves reducing the complexity of the filter design and consequently that of the hardware which makes it up. Filters fabricated with these techniques show improved performance parameters as compared to conventionally designed ones.

This thesis covers the design of FIR filters using the Common Sub-Expression Elimination technique, further enhanced by using Linear Programming to automatically produce filter coefficients in CSD format, using algorithms which maximize commonality of sub-expressions in filter coefficients. This maximizes reduction in filter design and fabrication complexity.

CHAPTER 1

1. INTRODUCTION

With increasing trends in the use of mobile devices running more complex applications, need for more powerful mobile processors with still-higher processing speed has become imminent. Mobile devices using digital signal processing (DSP) cores immensely use finite-impulse-response (FIR) filters due to their numerous advantages. Need for increasing processing speeds with decreasing power consumptions and on-chip-area of FIR filters is becoming more and more demanding. Whereas processing speed can partially be increased by using faster performing chip technology, power consumption and area-on-chip cannot be reduced by this method alone. Optimization in all three areas can be achieved by reducing complexity in the structure of FIR filters.

1.1. Problem Statement

This thesis emphasizes on achieving reduction in structural complexity of FIR filters by designing the filter using a Linear Program (LP) and obtaining its coefficients directly in CSD format. Although work has been done in this field, the addition proposed and simulated in this thesis is designing an algorithm in the LP to automatically maximize common sub-expressions in filter coefficients, thereby facilitating common sub-expression elimination process. Maximizing common sub-expression elimination is a known way of reducing filter structural complexity, as the number of full-adders required to sum up the partial products generated in the FIR filter are reduced. Adders take up a lot of space on the chip during the fabrication process and consume power during operation and increase processing time also. Reducing adders improves processing time, reduces energy consumption and area-on-chip for the circuit.

1.2. Scope of This Thesis The scope of this thesis is to design and simulate an algorithm using the power of an LP which designs the coefficients of a Linear Phase, low Pass FIR filter directly in CSD format in a way that maximizes commonality of Sub-Expressions within each coefficient automatically. The algorithm has been designed and simulated successfully. Results show up to 34 % reduction in the number of adders required to sum partial products.

1.3. Application of This Thesis This thesis can be used to achieve area reductions in FIR filter on-chip fabrication using the flexibility of an LP. Filter characteristics such as cutoff frequency, allowable pass-band and stop-band ripple, filter order, and number of bits used to define filter coefficients are programmed into the LP. Filter coefficients are generated in CSD format with a defined common sub-expression in each coefficient. The Algorithm has been designed in the Linear Programming toolbox of Matlab.

1.4. **Organization of This Thesis** The **first chapter** revises some of the general concepts of the FIR filters, defines complexity reduction in FIR filters and some general methods to reduce filter design complexities.

1.4.1. **Second chapter** describes in detail, the common sub-expression elimination technique for complexity reduction that forms part of this thesis.

1.4.2. **Third chapter** describes the Linear Programming and how it can be used to provide a flexible platform to design FIR filters.

1.4.3. **Fourth chapter** covers mathematical modeling of the filter in an ILP.

1.4.4. **Fifth chapter** describes the algorithm which maximizes commonality of sub-expressions among filter coefficients, thereby achieving maximum complexity reduction.

1.4.5. **Sixth chapter** compares results of complexity reduction achieved by introducing common sub-expressions in filter coefficients. This is followed by **conclusions.**

2. LITERATURE REVIEW

FIR filter Structure: The FIR filter is a digital circuit that is made up of three basic elements. A delay circuit (represented in Fig 1 as Z^{-1}), a Digital multiplier circuit (Circle with an X in it) and an Adder circuit (Circle with + in it). The basic FIR filter circuit is shown in figure 1. The filtering process in digital domain is actually the convolution addition of the incoming digitized signal X and the filter coefficients h[N], where N is the order of the filter. The output Y is the digital representation of the filtered signal. Generally, the greater the value of N the better is 'fidelity' of the output filtered signal. However increasing the order of the filter means more taps required which increases the area on chip of the filter. A compromise has to be achieved between the size of the filter and the quality of the output signal required. Sample rate of the input and the output remain the same. All three types of filters i.e. low-pass, band-pass and hi-pass filters can be designed by the same basic structure. A typical 6 tap FIR filter is shown in Figure 1:

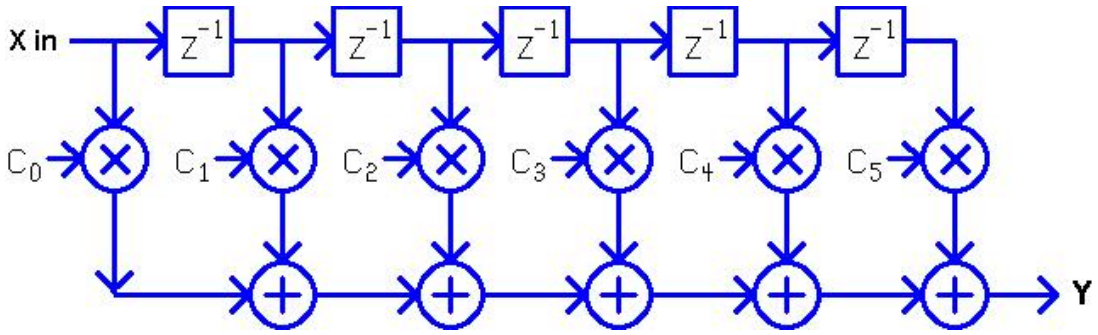

Figure 1: A typical 6-tap FIR filter circuit

There are two kinds of digital filters, the FIR or finite Impulse Response and the IIR or the Infinite Impulse response filter. The advantages and disadvantages of FIR filters over the IIR filters [13] are as follows:

2.1. Advantages:

2.1.1. They have linear phase and are easily designed.

2.1.2. They are inherently stable.

2.1.3. Excellent design methods are already available for various specifications.

2.1.4. Output noise due to multiplication round offs and sensitivity to filter coefficient variations is low.

2.2. Disadvantage:

2.2.1. They require a lot of arithmetic operations, thereby increasing size-on-chip and power requirements. This gets worse for tighter requirements of the transition band.

3. *Types Of Filters:*

There are 4 types of FIR filters [13]:

3.1. **Type I:** Order of filter (N-1) is Even and h[n] = h[N - n]. The filter is even about n=0 and has a repetition period of 2π, as shown in Figure 2.

3.2. **Type II:** Order of filter (N-1) is Odd and h[n] = h[N/2 - n]. The filter is even about n=0 and has a repetition period of 2π, as shown in Figure 3.

3.3. **Type III:** Order of filter (N-1) is Even and h[n] = - h[N - n]. The filter is not even about n=0 and has a repetition period of 4π, as shown in Figure 4.

3.4. **Type IV:** Order of filter (N-1) is Odd and h[n] = - h[N/2 - n]. The filter is even about n=0 and has a repetition period of 4π, as shown in Figure 5.

'n' is a design variable and varies from 0 to N-1. h[n] are the coefficients of the filter's impulse response and are usually represented as normalized values so that their highest value i.e. h[0], is <= 1. various types of filters are shown in Figure 2.

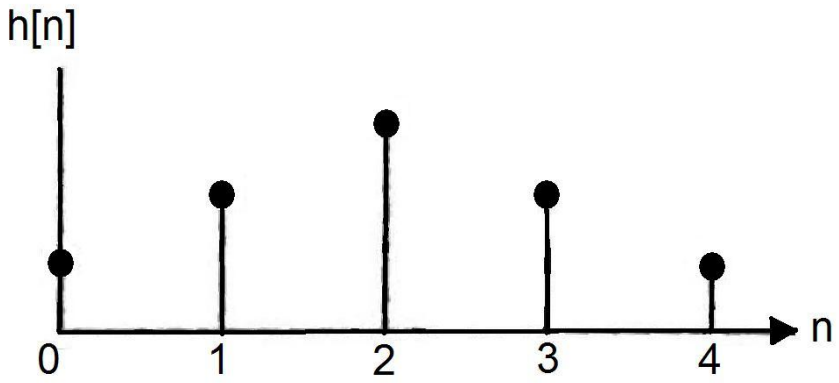

Figure 2: Type 1 filter

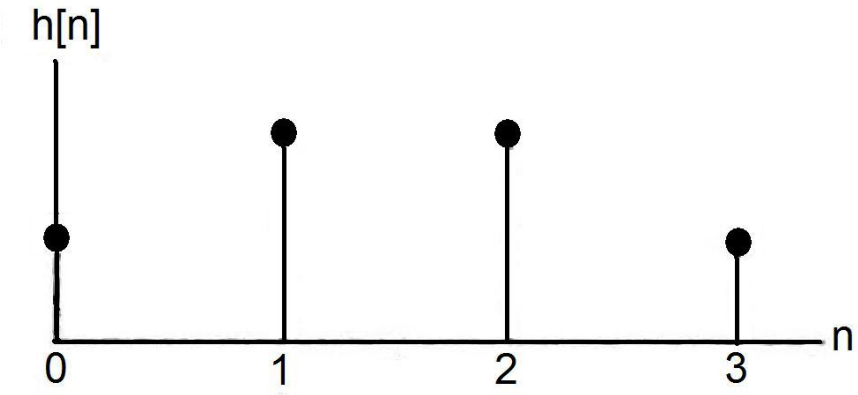

Figure 3: Type 2 filter

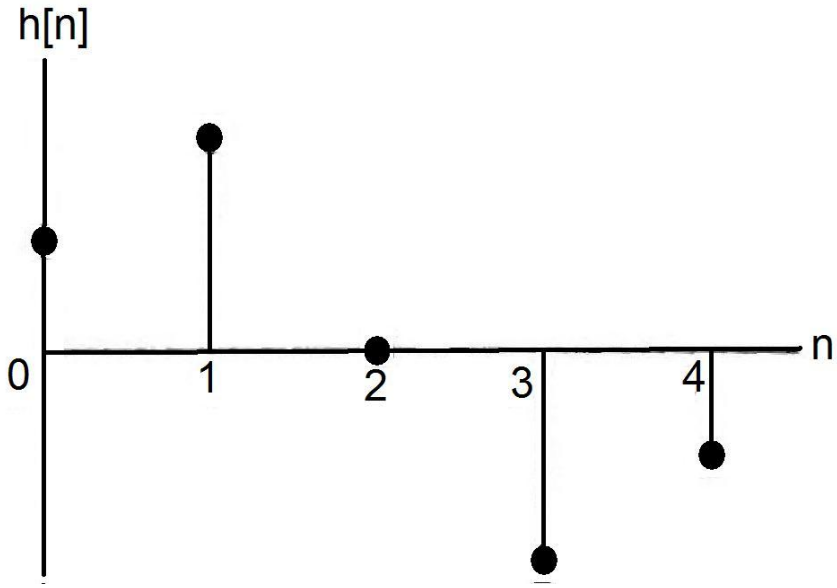

Figure 4: Type 3 filter

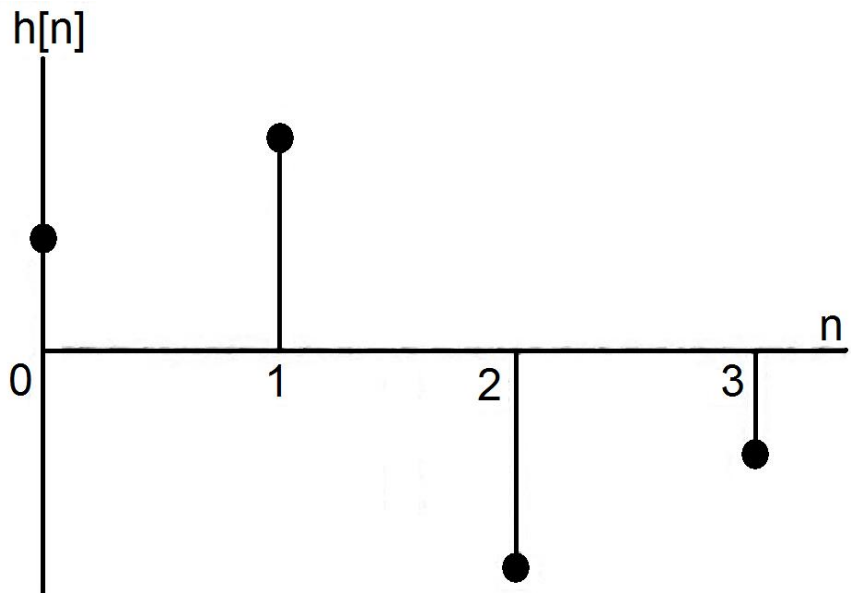

Figure 5: Type 4 filter

4. ***Filter Design Methods:***

FIR filter design techniques have matured and evolved to a stage where a lot of options for filter design have been found and perfected. Each method optimizes a particular characteristic of the filter. Choosing the design criteria is dictated by the nature of employment of the filter i.e. the characteristic for which the filter is to be designed, should be achieved most effectively. Following are the more common techniques of filter design. In some cases combination of these techniques is also used, e.g. the pass band may be designed using one technique and the stop band with another. [13]:

4.1. **Minimax Error Design Method:** One of the main advantages of FIR filter over their IIR counterparts is that there exists an efficient algorithm for optimizing in similar technique for the IIR filter is time consuming and convergence of the best solution is not always guaranteed. If the maximum deviation from the desired response is required to be minimized then it is preferred to use filters designed in the minimax sense. This design method optimizes the coefficients of the filter so that the error between the approximating response and the desired response is minimized or maximized. The solution minimizing the maximum error is called the Chebyshev approximation. Generally, increasing the filter order minimizes this error, so finding the minimum filter order that will just bring the error within the desired tolerances is the target of this method. The maximum error is mathematically represented by Eq 1. The frequency 'ω' at which the difference or error between the approximating response and the desired response becomes a maximum is taken which is kept within tolerances. This way ensures that minimum order of the filter is used to bring the filter's characteristics within tolerances or the approximating response follows the

desired response with their maximum differences limited within the specified tolerance limits.

$$\varepsilon = \max | E(\omega) | \quad \text{---- Eq 1}$$

One of the most important design method used is the Remez Multiple Exchange Method.

4.2. **Least-Squared Design Method:** In some cases, the square of the error between the approximating response and the desired response needs to be minimized. In this case the problem is to find the filter coefficients that minimize the error, E_2 [13] This is mathematically represented by Eq 2:

$$E_2 = \int_x [W(\omega)[|H(e^{j\omega})| - D(\omega)]]^2 d\omega \qquad \text{---- Eq 2}$$

Here x is the frequency band containing both the pass-band and the stop-bands, $W(\omega)$ is the weight of response at the frequency ' ω ', $|H(e^{j\omega})|$ is the magnitude of the achieved frequency response at ' ω ' and $D(\omega)$ is the desired frequency response at ' ω '. If $D(\omega)$ and $W(\omega)$ are sampled at very close intervals $\omega_1, \omega_2, \omega_3 \ldots .. \omega_k$ on x, then minimization of the error can be achieved by minimizing E_2 as shown in Eq 3:

$$E_2 = \sum_{k=1}^{K} [W(\omega_k)[H(\omega_k) - D(\omega_k)]]^2 \quad \text{---- Eq 3}$$

4.3. **Maximally Flat Approximation Method:** In this method the filter is designed with maximally flat responses at ω=0 and $\omega=\pi$. This gives the advantage that the design becomes extremely simple. This is useful in applications where signal is required to be preserved with very small errors at ω=0. The approximating response in this method is obtained, based on a Taylor Series approximation to the desired response at a given point, which is generally a frequency point. In some cases the Taylor Approximation is applied to two points, one in the Pass-Band and the other in the Stop-band. This is used in the Butterworth design and produces a maximally flat response. This type of design method usually produces the least ripples in the approximating response but at the cost of a large filter order. This is used in cases where fidelity over-rules expensive design and large chip sizes and power consumptions. The frequency response $H(\omega)$ for a Type I filter designed by this method is given by Eq 4.

$$H(\omega)=\sum_{n=0}^{2M}\alpha[n]\cos^{n}(\omega) \qquad \text{---- Eq 4}$$

The resulting $H(\omega)$ is characterized by the fact that it achieves the value of 1 at ω=0 and achieves the value of 0 at $\omega=\pi$.

4.4. **Windowing Method:** as this thesis uses the windowing method, it is covered in greater detail. As per reference [13], ideally, design of filters by windowing method begins by specifying zero-phase frequency response of an ideal low-pass filter $H_{id}(\omega)$, for a type I filter, this is even about ω=0, with a repetition period of 2π. This can be expanded in a Fourier series as Eq A and B:

$$H_{id}(\omega) = h_{id}[0] + 2\sum_{n=1}^{\infty} h_{id}[n]\cos n\omega \text{ ---- Eq -A}$$

$$\text{where } h_{id}[n] = \frac{1}{2\pi}\int_{-\pi}^{\pi} H_{id}(\omega)\cos(n\omega)d\omega \text{ ---- Eq -B}$$

the frequency response of the filter continue indefinitely towards $\pm\infty$ and is even about n=0. This is shown in Figure 6.

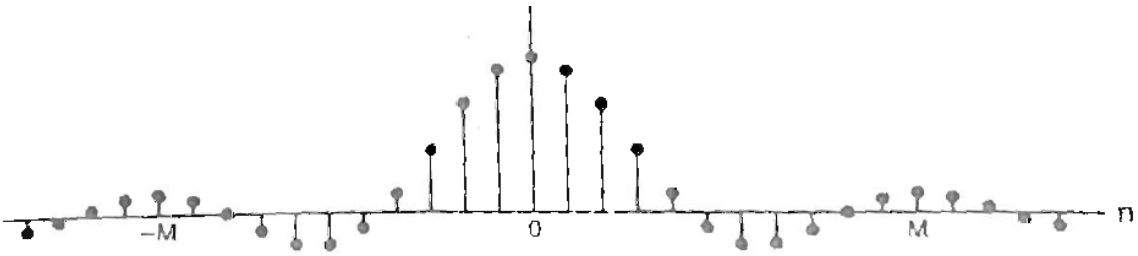

Figure 6: Impulse response of an Ideal filter

Implementing this practically is impossible so a windowing function is used. The window function for example is a rectangular with height as unity and exits between n = $\pm$ M. where M is the half of filter order N for a type I filter. i.e.

Magnitude of window = 1 { when n > -M till n < +M}

Magnitude of window = 0 { when n < -M and n > +M}

The window is multiplied with the Fourier series of the ideal filter response, the result is truncation of the response at a specified interval i.e. at n= $\pm$ M. This is shown in Figure 7.

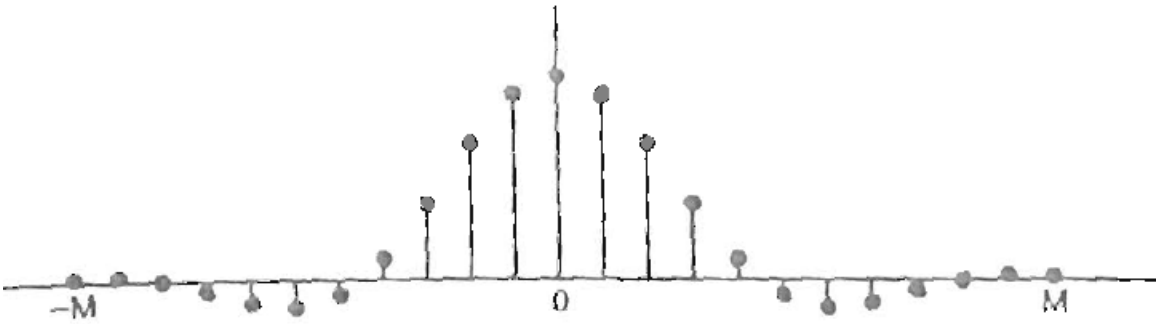

Figure 7: Impulse response truncated

A rectangular window truncates the impulse response suddenly. Since multiplication in time domain is equivalent to convolution in the frequency domain, truncation of impulse response is actually a convolution of the frequency response of the ideal filter and a rectangular function with its edges at $\pm\ \omega_c$. where ω_c is the cut off frequency of the filter. The convolution process is shown stepwise in Figure 8 till Figure 11.

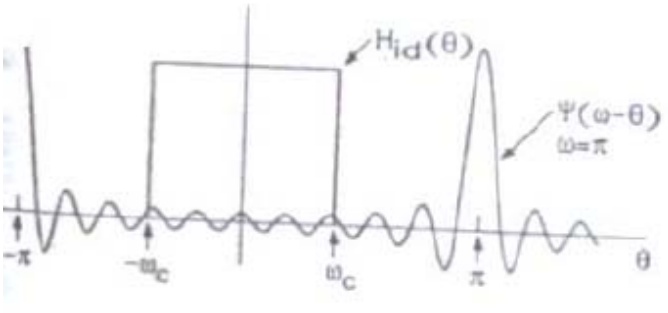

Figure 8: Convolution of ideal response and windowing function 1

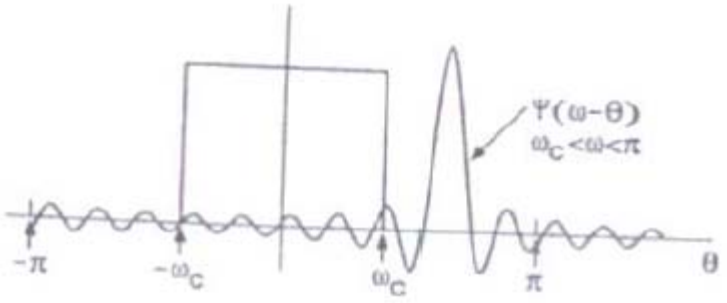

Figure 9: Convolution of ideal response and windowing function 2

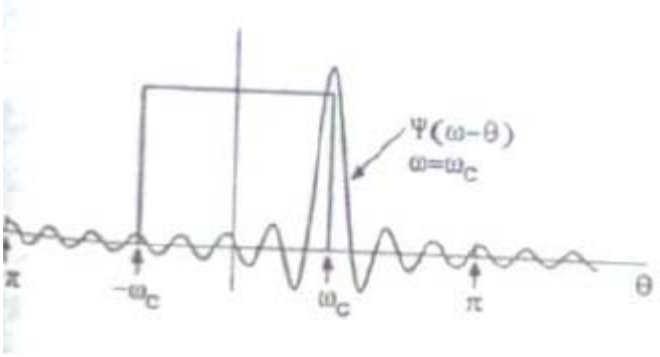

Figure 10: Convolution of ideal response and windowing function 3

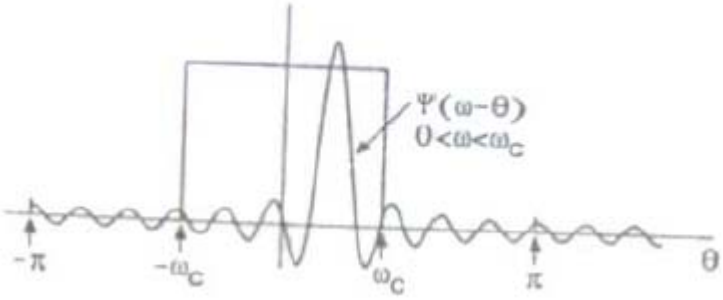

Figure 11: Convolution of ideal response and windowing function 4

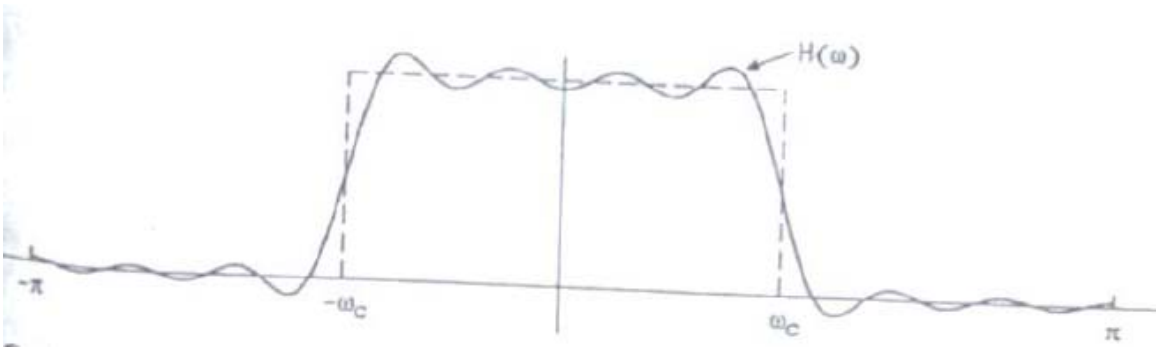

Figure 12: result of Truncation of the Ideal response

It can be seen from Figure 12 that the result of the convolution is the actual frequency response of the designed filter (also known as the approximating response). The side lobes or ripples in the ideal filter frequency response are higher in magnitude near to the main lobe and decrease as we move away from the main lobe. This means if the truncation is quite near the main lobe (narrow window), the resulting filter response will have higher magnitude of ripples than the case where the ideal frequency response is truncated at a point further away from the main lobe (wider window). Wider window is more accurate an approximation to the ideal behavior than the narrow window and hence closer to the ideal response, thereby displaying lower ripples. Using a rectangle window causes a sudden truncation of the ideal frequency response. These ripples in the frequency response are explained by the Gibbs phenomenon [13]. This can be seen in Figure 12 also. Similarly increasing the order of the filter causes the number of ripples to increase in approximating frequency response while keeping their magnitude of the ripple the same. This is shown in Figure 13. Here a filter with ω_c=0.4π with N=20 (solid line) and N=60 (dotted line) is shown.

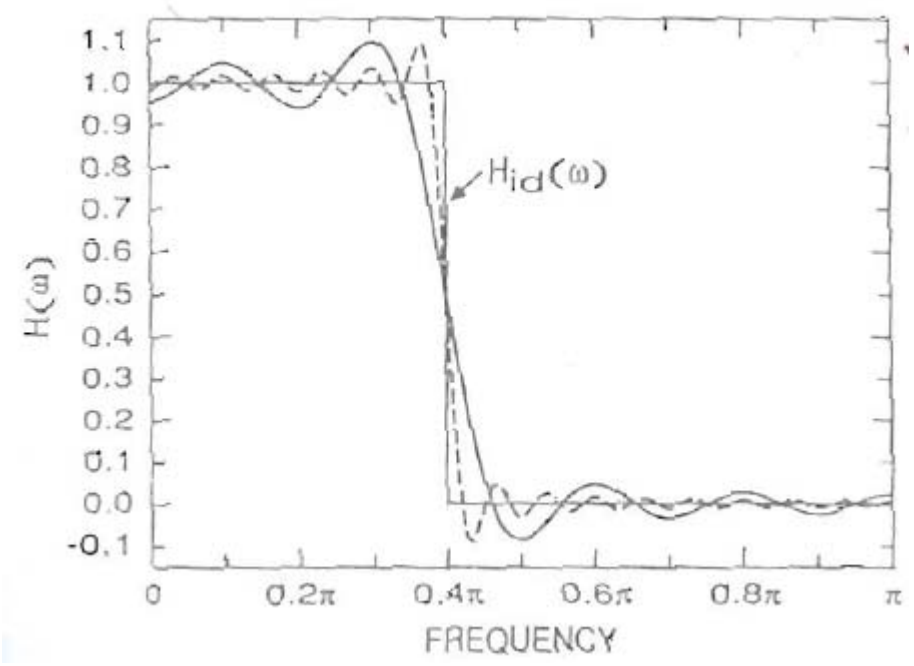

Figure 13: Ripple frequency Vs Filter Order.

These ripples are an undesirable effect and are therefore smoothened using a gradually decreasing windowing instead of an abruptly truncating rectangular window. Gradually decreasing window such as a raised cosine window does not truncate the ideal response abruptly and therefore does not produce ripples. Windows can be of different shapes, such as rectangular, triangular, raised cosine etc, each with its particular desired and undesired effects. Some of the popularly used windows are as below:

4.4.1. Rectangular Window.

4.4.2. Bartlett Window

4.4.3. Hann Window

4.4.4. Hamming Window

4.4.5. Blackman Window, Etc

4.4.6. Each of these windows has its own mathematical function with its own positive and negative characteristics. These characteristics are exploited as per the usage of the window.

5. ***Transposed Direct Form.*** A filter is a complex circuit and therefore a means must be devised to represent a filter as easy-to-represent symbols. These symbols represent various parts of the filter and can be drawn to show various details of the sub-components of the filter Following are the common representation forms. A digital circuit can be written in its direct form or transposed-direct form. Representing a circuit in any of these forms does not change the working of the circuit in any way. In case of FIR filters its transposed direct form is mostly used as it makes easy, the understanding of the working of the filter. A working-method to obtain the transpose direct form of the FIR filter is to reverse the direction of all arrows in the circuit and to interchange the input with the output. The basic structure of FIR filter and a working-method to obtain it's transpose-direct form is shown from Figure 14 till Figure 17:

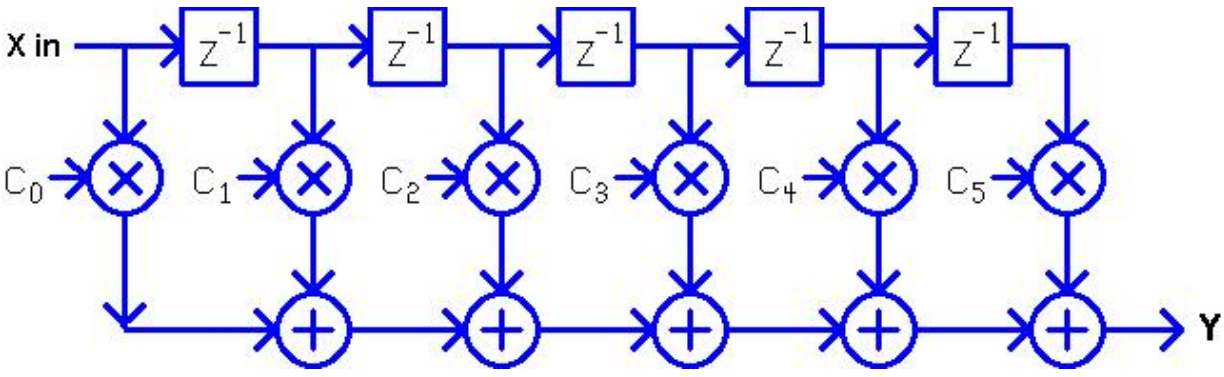

Figure 14: FIR filter drawn in conventional symbols

Figure 14 shows a typical arrangement of components in an FIR filter. A Z^{-1} block shows a delay element, the circles with '+' in them shows adders and circles with 'x' in them shows multipliers. The same circuit is redrawn below using a more easy-to-draw format. A Z^{-1} next to an arrow mark now represents the delay element. C0, C1, C2 etc next to an arrow represent multipliers with these constants are its multiplicands and a point where two arrows meet, represent an adder. This representation is the direct form of the FIR filter and is easy to draw and manipulate for reasons of analysis and solving. This is shown in Figure 15.

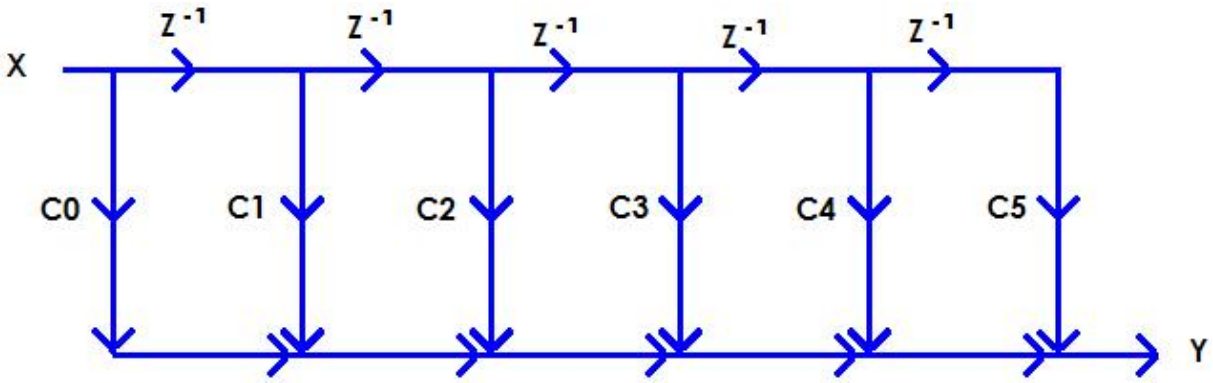

Figure 15: FIR filter drawn in easy-to-draw format

To obtain the transpose of this circuit we reverse the direction of the arrows and interchanging the inputs with outputs. This is shown in Figure 16.

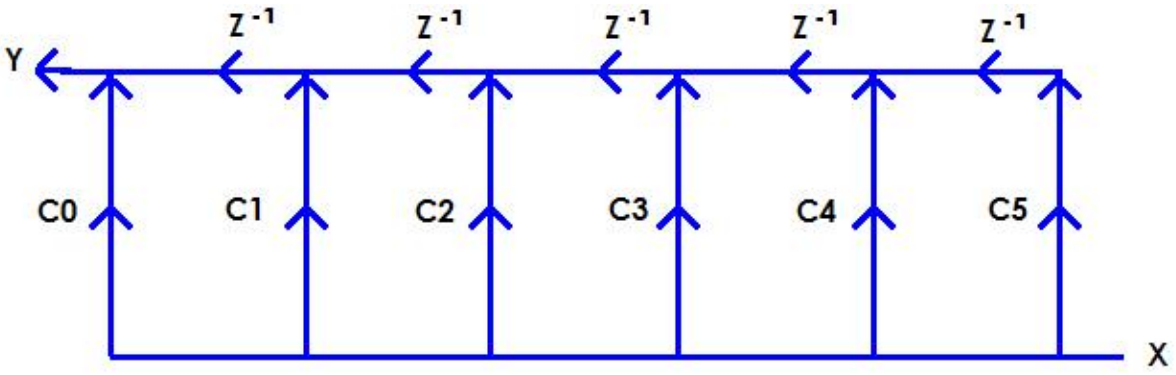

Figure 16: Taking Transpose of the structure

This circuit is oriented differently and may be difficult to understand, so to have a better reference the circuit is rotated clockwise 180 degrees. This is shown in Figure 17.

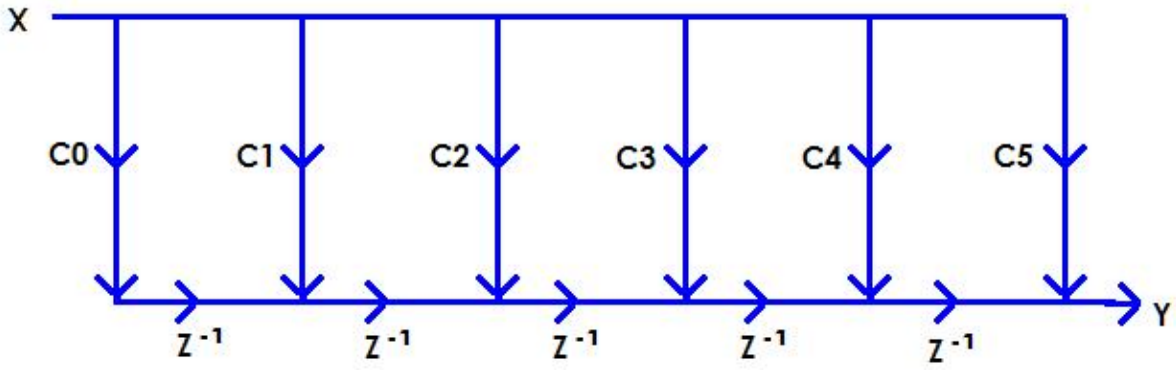

Figure 17: Typical representation in Transpose direct form

The type of filter structure in Fig 17 is known as transposed direct-form or data-broadcast. The property of this structure is that input value is fed to all the multipliers at the same instant. This results in all multiplications being applied to the single input as soon as it becomes available. The advantage here is purely from a solution point-of-view. All calculation on the single input data is completed before the arrival of the

next data. This helps in sharing of common sub-results between multipliers and therefore it becomes possible to apply sub-expression elimination and computation sharing procedures on the data. [9],[4].

CHAPTER 2

1. *COMPLEXITY REDUCTION:*

The basic structure of an FIR filter is very complex and occupies a prohibitively large area on the chip [1]. As such it cannot be fabricated on chip. This means that need to reduce the size of the circuit is imminent. Size reduction without compromising on the function of the circuit is the basic goal of fabrication. Complexity reduction in FIR filters refers to techniques by which the basic structure of the FIR filter can be modified or made simple to accommodate on chip. This enables us to overcome the only disadvantage of the FIR filter mentioned above. i.e. bigger size. These techniques also help optimize processing speed and power consumption of the filter. Since multipliers and adders are the most expensive as regards fabrication operation and power consumption in the FIR filter, simplifying the structure of or reducing the number multipliers is highly desirable for complexity reduction [4]. Many methods have been devised to achieve this and are continually being optimized, with varying degrees of success. Summaries of a few such approaches are discussed below:

1.1. **Structure Folding:** Coefficients of FIR filters are inherently symmetric [4] e.g. for a type I FIR filter, h[N-n] = h[n]. (**First** coefficient is same as the last, **second** one is same as second-last and so on). Because of this property of FIR filters, the results of multiplications produced by the first half set of multipliers of the filter can be re-used in its second half. From fig 14, we see that since C0 = C5, therefore, X.C0 is the same as X.C5; X.C1 is the same as X.C4 etc. This means that the multiplier at C5 can be removed and the result X.C0 can be shared between the first and the last tap. The same can be done for multipliers used in second and the second-last tap and similarly for multipliers in other taps. A folded structure gives the same result as the original and is shown in figure 18. In this folded structure, three

multipliers out of the total of 6 have been eliminated. This gives almost half reduction in area-on-chip during fabrication process. In actual practice the filter order may be around 100 or more, this method can reduce the number of multipliers to half and result in substantial savings in on-chip-area and power consumption of the circuit. It can be seen here that the number of adders and delay elements remain the same as before.

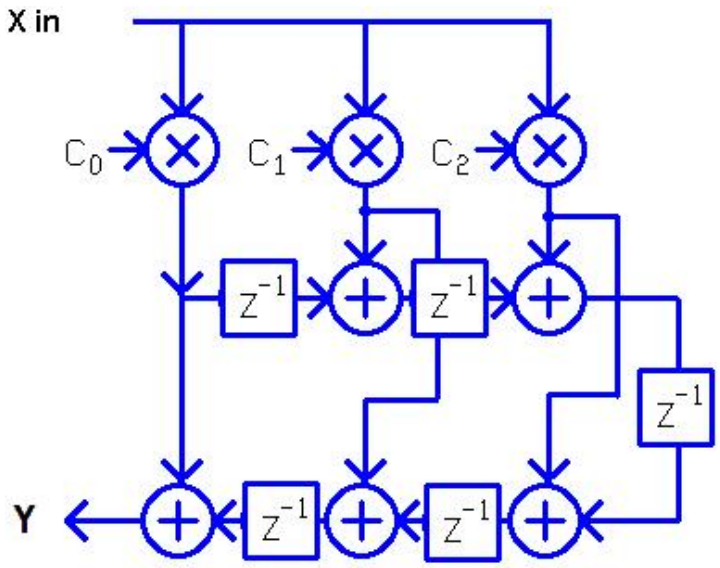

Figure 18: Folded Structure of a 6 tap FIR filter

1.2. **CSD Representation of Filter Coefficients:** In this method, the coefficients of the filter, which are constants, are represented in the CSD format. This process exploits the property of the CSD number which represents a binary number with the lease possible non-zero elements. This reduces the number of non-zero elements in the coefficient to a minimum, as the number of partial products generated in the multiplier is the same as the number of non-zero bits in the multiplicand,

reducing the number of non-zero bits in the multiplicand will reduce the number of partial products in the multiplier. This is a big achievement as this reduces the number partial-product adders in the multiplier circuit which is a very expensive component in terms of processing-time and area-on-chip. Another property of CSD format is that accuracy of result is not really compromised even if the number of non-zero bits in a coefficient are truncated to the 4 most significant non-zeros only. This will cause the number of partial products in each multiplier to be exactly four, thereby reducing the number of full-adder rows to add the excessive partial products. As an example, multiplication by conventional means and then by representing the coefficients in CSD format is given in Figure 19, for comparison:

Data Input (X):	3	0011	0011
Filter Coefficient:	7	0111	$100\bar{1}$
		0011	11101
		0110	11000
		1100	
Answer	21	10101	10101
		Conventional multiplication	CSD multiplication

Figure 19: Showing Conventional and CSD multiplier

In the above example, the top row represents the digitized data (x) input to a filter i.e. 3. The second row gives the magnitude of one of the filter coefficients, i.e. 7. Two

examples show a conventional multiplier (left) and a CSD multiplier (right) calculate the result. In a conventional multiplier, 3 partial products are generated because the multiplicand consists of 3 non-zero bits. Three partial products require 2 rows of full adders and the result is obtained after a 2-full-adder delay. In the CSD multiplier, the multiplicand has 2 non-sero bits and so 2 partial products are generated, requiring 1 row of full adders. The result is obtained after 1 full-adder delay. This example shows that by using CSD format, the hardware used and processing delay penalty is substantially reduced. Reduction in hardware also results in reduced power consumption, which is another desired factor.

1.3. **Perturbing Filter Coefficients:** When multiplying a number with another number which is an exact power of 2, multiplication becomes only a shift operation [4]. In case of the FIR filters if all coefficients are exact powers of 2, the input data, when multiplied to these coefficients is just shifted by the required number of bits to get the answer. This type of shifting is called 'hardwired shift' and is done by attaching the required number of ground wires as the least significant bits (LSB). The same is shown in the following example where a variable X is multiplied by a constant 16, which is a multiple of 2:

X . 16 => shift X left, 4 times only or append 4 zeros as LSB.

If X = 7, 7 x 16 = 112

In binary, (111) x (10000) = 1110000

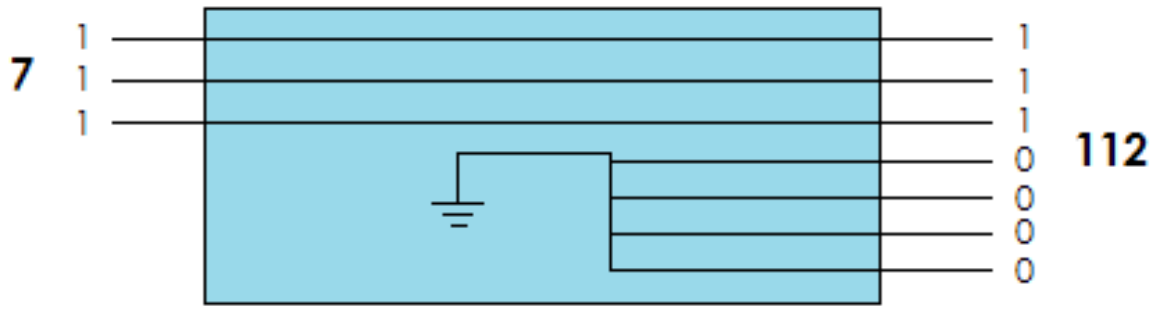

Figure 20: A Multiplication-by-shift structure

This is shown in Figure 20 where the input 7, (binary 111) is multiplied by 16; (binary '10000') the multiplier circuit merely appends 4 ground wires (representing 4 zeros) as LSBs to the input. This gives a real time multiplier circuit which multiplies every input number with 16.

The disadvantage of using this method is that if all filter coefficients cannot be perturbed to exact-powers-of-2 else it will introduce a lot of errors in the output results. A reasonable compromise might be that coefficients within an acceptable 'range' from exact-power-of-2s can be perturbed thus reducing structural complexity at the cost of inducing quantization error in the filter output.

1.4. **Computation Sharing:** FIR filters have symmetric structures and so many of its calculations are repeated. As such a lot of redundancy of computation is inherently present in it. These redundancies can be effectively eliminated by identifying the common computations among the coefficients of the FIR filter and finding their result only once. These common results are reused wherever required. Computation sharing is a way to reduce redundant blocks in the hardware, thereby

reducing area-on-chip which also contributes towards reduced power consumption of the circuit. There are many ways to share computations; the basic concept of the process can be understood by studying the method called ‘Factorization of Perturbed coefficients’ [4]. By this method complexity can be reduced to a considerable degree but cost paid here is again, the introduction of quantization error. However coefficient perturbation by this method introduces relatively less quantization noise than that of the method mentioned above. Reference [4] describes this method in which FIR filters are designed with coefficients that have been perturbed or changed in a way that they can be ‘re-constructed’ using first seven prime numbers as their only prime factors. The data input to the filter is multiplied simultaneously by the first 7 prime numbers (2, 3, 5, 7, 11, 13 and 17) and the results stored for re-use. The spirit behind choosing only the first 7 prime factors is that although increasing the number of prime factors beyond 7 will help reduce complexity of the filter to an even greater degree, but frequency of sharing of each prime factor will reduce thereby reducing its ‘re-use’. Alternately, reducing prime factors below 7 will increase ‘quantization’ error as each coefficient cannot be ‘re-created’ accurately/ efficiently with so less factors. The choice of ‘7’ offers a good compromise between accuracy and frequency of computation re-use. Multiplications by prime factors are generated not by actual multiplier circuits but by a multiplier less process of shift-and-add. The results are stored and re-used for multiplication for all the coefficients of the filter by adding the required stored results to complete the multiplication process. Consider the following example:

Consider 4 filter coefficients 35, 28, 21 and 5. These are being multiplied to data ‘x’ input to a filter. Multiplication with first 7 prime factors is carried out and stored these values are: 2x , 3x , 5x , 7x , 11x, 13x, 17x.

The coefficients are re-created from their prime factors as follows:

$$35x = 17x + 13x + 5x$$

$$28x = 17x + 7x + 5x$$

$$21x = 17x + 5x$$

$$5x = 5x$$

in the above example, pre-computed numbers 17x and 5x have been re-used thrice and 4 times respectively, thereby achieving reduction in hardware, however; in the second case 17x + 7x + 5x = 29x, and in the third case 17x + 5x = 22x which causes perturbation from the original value of 28x and 21x. In this way hardware reduction is achieved while accepting slight compromise on filter behavior. This method becomes more efficient with increase in the number of coefficients of the filter i.e. increasing the number of taps of the filter. With filter taps varying from 20 to 200, reduction in complexity from 35% to 50% have been calculated [4].

2. ***COMMON SUB-EXPRESSION ELIMINATION* (CSE)**. As per ref [9] when one variable is being multiplied by a set of constants (like input data multiplied by the set of filter coefficients in an FIR filter), usually there is a lot of redundant computations in the process. Since the filter coefficients are constants, the redundancies in these numbers are fixed for every filter. These can be identified and eliminated during the filter design process. In the CSE method the coefficients/ constants are broken up into smaller “sub-

constants" or "sub-expressions" that are recurring in most of the other coefficients. For example the decimal number 2709 is expressed in CSD format as follows:

$$1\ \ 0\ \ 1\ \ 0\ \ 1\ \ 0\ \ 0\ \ 0\ \ \bar{1}\ \ 0\ \ 1\ \ 0\ \ 1$$

analyzing this CSD number we see that it contains a CSE of '101' occurring twice in it. That is:

$$(1\ \ 0\ \ 1)\ \ 0\ \ 1\ \ 0\ \ 0\ \ 0\ \ \bar{1}\ \ 0\ \ (1\ \ 0\ \ 1)$$

in this case the number '101' is the sub-expression. Therefore; expressing a constant through sub-expressions leads to reduced number of arithmetic operations. This is because redundant computations for recurring sub-expressions can be effectively eliminated. This method for simplification of multiplication by constants is called Sub-expression elimination. This leads to efficient hardware implementation in terms of area-on-chip, processing speed and power consumption of the circuit. The larger the size of the common sub-expression the higher the complexity reduction achieved.

The main point of sub-expression elimination is to find common structures in the set of filter coefficients and to reuse them. For example, consider that an input to a filter is the data word X. It is to be multiplied by filter coefficients, a= 13 and b=27. The constants 'a' and 'b' are shown as binary numbers, (table 1) it can be seen that 1st and 4th bits are the same for both constants i.e. the common sub-expression in these two constants is 1001, as this is present in both constants. Multiplication with 1001 can be done once, and the result reused where-ever required.

Constant	Decimal	Binary
a	13	001101
b	27	011011

Table 1: Binary representation of a & b.

It can be noticed that the common term (1001) occurs in constant b twice: once with 1 left shift and another with the no shift, i.e. b x X = (X x 1001) << 1+ (X x 1001). Where '<<1' means one-shift in the left direction.

In the similar manner multiplication a becomes: a x X = X x 1001 + X x 0100. If the above given algorithm is used, the required number of shifts is 3 and required number of additions is also 3 (one for the CSE and two for the above expressions). If conventional multiplication is used then the total number of addition is 5 shift and 5 adds. As and alternative, just by inspecting constants a & b in either decimal or binary form, that b=2a +1. This scheme also gives total number of 3 additions and 3 shifts.

The basic algorithm consists of 5 steps implemented in the design phase.

Step 1. All the constants are expressed in CSD format to reduce the number of non-zeros.

Step 2. Common sub-expressions are determined between all constants.

Step 3. The best match, i.e. with the maximum length of common sub-expression, is chosen.

Step 4. The identical bits or so-called 'redundancy' are removed from all constants.

Step 5. Continue with the step 2, for the new set of constants, until no improvement is achieved.

2.1. **An Example.** An example of the algorithm [9], consider a variable X being multiplied by filter coefficients. Three coefficients {237, 182 and 93} are considered as an example. The constants are expressed in binary (table 2). The process is executed in the above steps in an iterative process as follows:

2.1.1. **First Iteration.** We can see that the greatest match is between constants a & c. Therefore this match '01001101' is being removed from the constants a & c

Constant	Decimal	Binary
a	237	11101101
b	182	10110110
c	93	01011101

Table 2: Binary representation of numbers

After removing the CSE in the first iteration, the new constants are denoted by a' and c'. a'=11101101-01001101 = 10100000 and c' = 01011101-01001101 = 00010000. These results are displayed in table 3.

2.1.2. **Second iteration**. By inspecting the table it is seen, that there are still 2 matches left among a' and b. This match '10100000' is removed in the second iteration.

Constant	Binary
a '	10100000
b	10110110
c'	00010000

Table 3: After extraction of first CSE

Redundancy in a & c. i.e. 01001101 has been removed

After removing CSEs in the second iteration the constants a '' and b ' look like this: a '' =10100000-10100000 = 00000000 and b ' = 10110110-10100000 = 00010110

The results are as in Table 4. There is one match between b' and c but since one match does not decrease number of operations, therefore the end of sub-expression elimination has been reached. All three constants may now be redefined as:

Constant a = redun(a,c) + redun(a,b)

Constant b = redun (a,b) + 00010110

Constant c = redun (a,c) + 00010000

Constant	Binary

a ’ ’	00000000
b ’	00010110
c'	00010000

Table 4: After extraction of second CSE

redun(a,c) 01001101

redun(a,b) 10100000

These two sub-expressions occur 2 times each.

2.2. **Comparison.** The comparison between implementing the 3 multiplications in conventional method and then with CSE method is shown in table 5.

Constant	without CSE	with CSE
a	5 shifts + 5 adds	1 add
b	5 shifts + 4 adds	3 shifts + 3 add
c	4 shifts + 4 adds	1 shift + 1 add

Table 5: Comparison before and after removing CSE

Total: without CSE is 14 shifts + 13 adds and with CSE is 9 shifts + 9 adds

As a result of the application of sub-expression elimination 5 shifts and 4 additions have been saved, this gives a saving of more than 25%.

CHAPTER 3

1. *LINEAR PROGRAMMING*

1.1. **Problem in Conventional Design methodology:** the main problem with designing a FIR filter for a DSP application is that during the design phase of the FIR filters, their coefficients are first calculated as real numbers. These numbers when converted to binary and then to fixed point numbers for use in the DSP or ASIC result in quantization errors being introduced in the coefficients [1]. These quantization errors change the behavior of the filter significantly. Similarly converting the calculated real coefficients to CSD format introduce non-uniform quantization errors in the frequency response of the filter [1]. To avoid the introduction of the quantization error and the non-uniform quantization, the preferred methodology is to design the filter coefficients directly in CSD format using an Integer Linear Program (ILP).

1.2. **The Linear Program (LP):** It is a computer software tool for solving general purpose problems containing many variables, which are linearly related to each other (hence the name linear program). The software, generally known as an Linear problem solver or just 'LP Solver' is a general purpose program that calculates optimized values for the problem variables in a way so as to maximize or minimize the objective to be achieved by the problem, in the face of some constraints that may exist in the problem. Each of these terms is explained in the following paragraphs.

The LP uses various algorithms in an iterative process to reach the solution. During each iteration, an arbitrary value for each variable is assumed; the solution is calculated and compared against the result or the previous iteration. In each successive

iteration, the variables are modified in a way so as to minimize or maximize the objective function as required in the program. The algorithm used for modifying the variable during each iteration converges to the solution in the least possible number of iterations. The variables of the problem can be real numbers, integers or Boolean. When the variables are integers, the LP is called Integer Linear Program (ILP); Boolean variables also form part of an ILP [1].

1.3. **Parts of an ILP problem:** A problem to be solved by LP, is to be suitable modeled mathematically as equalities or in-equalities, into the following parts:

1.3.1. **Objective Function:** This is the main expression of the problem. The results desired in the problem are written as a suitable mathematical expression. The objective function can either be maximized or minimized. For example, in a problem optimizing financial income of a business venture, the total profits from the commercial unit are always desired to be maximized, whereas in a problem optimizing a manufacturing process, total material used in a manufacturing process is always desired to be minimized, etc. the objective function, therefore is a mathematical expression of a real life problem that models the results, so desired.

1.3.2. **Constraints:** Constraints define the limitations that have to be observed while finding solution to the problem. For example, in a factory warehouse, there is a maximum limit on the quantity of raw material that can be stored and delivered per day to the manufacturing unit, when trying to maximize total sale of finished products. Alternatively, Although making a chair from wood, it is desirable to minimize the quantity of wood used per chair, however, quality assurance

department requires that the chair must withstand a certain amount of rough usage and more wood required to make it rugged. The quality therefore constraints the chair manufacturer to abstain from using the bare minimum wood for the chair. The constraints, therefore dictate the solution of the problem to divert from the best possible or most profitable scenarios.

1.4. **An Example:** To understand the concept of finding solutions using a LP, consider the example of a wood-works shop. The sale of table-chair sets is required to be maximized in the face of constraints as follows:

1.4.1. **Objective function**: Maximize per-day sale of the wood-work shop from fabricating chair-table sets.

Constraint #1	Table costs Rs 100/- & requires 2 cu ft wood.
Constraint #2	Chair costs Rs 75/- & requires 1 cu ft wood.
Constraint #3	Max total wood that can be supplied per day =100 cu ft
Constraint #4	Four chairs for every table to make the set.

Explaination of Terms

Objective function: maximize {sale price}

Sale price = no of tables (T) * 175 + no of chairs (C) * 100 i.e. total chairs and tables produced multiplied by their sale prices would give the total sale cost earned. This is desired to be maximized.

Constraint # 1 $1 * C + 2 * T <= 100$

This constraint controls the wood consumption of the problem. 1 cubic foot of wood multiplied by no of chairs gives total wood used to make chairs. 2 cubic foot of wood multiplied by no of tables gives wood used to make tables. These

two when added should not exceed 100 cubic foot of wood which is tha maximum capacity of the wood store.

Constraint #2 4 * T = C

This constraint controls the set making between tables and chairs. It says that the number of chairs produced should be 4 times the number of tables produced. Thus the production of table-chair sets is ensured.

Constraint #3 Integer Declaration of Variables T & C

Variables T & C should be integers, since number of tables and chairs cannot be a fraction.

It can be seen that the above in-equality, equation and expression define all the constraints stated above and the mathematical model is complete for solution.

Solution: max Sale per day = Rs 6400/-

No of tables produced per day = 16

No of chairs produced per day = 64

Total wood used (2 x 16 + 1 x 64) = 96 cu ft (4 cu-ft wood left over)

The variables in this problem are 2, i.e. the number of chairs and number of tables. Both variables are integers. After the solution, 96 cu ft of wood is used and 4 cu ft wood is left over, from which 4 more chairs or 2 more tables can be made which can further increase profits but as per constraint #4 (sets of 4-chairs-to-a-table) it is not allowed since 6 cu ft wood is required for the set which if not available, production cannot proceed. Also as per constraint # 3, max wood used per day < 100 cu ft. The solution is therefore optimized while following all the constraints.

1.5. **Using an ILP to Design FIR filter:** An ILP can be used not only to calculate optimized filter coefficients but to calculate them directly in CSD format. The coefficients be so designed that these produce the desired filter response with introduction of minimal quantization errors, while remaining within the defined

constraints such as the pass and stop bands attenuations, frequency response, allowable pass and stop-band ripple magnitudes, cutoff frequency etc. The ILP can also be used to design constraints so that the coefficients should be calculated having maximum common sub-expressions and still produce the desired filter response. Increasing the number of common sub-expressions will increase computation sharing and will optimize the filter fabrication in terms of area-on-chip, power consumption and speed of processing. This method can be employed to design FIR filters which are optimized to a maximum.

Like any ILP, first, a mathematical model of a filter is to be developed. The objective function, which minimizes the number of non-zero bits in the coefficients to minimize partial product adders, followed by constraints which are to be observed while calculating these coefficients. Like keep the pass-band and stop-band ripples less than the tolerance limits or minimizes difference between the approximating and the ideal frequency response to minimize frequency response deviation from ideal filter behavior. The constraints mainly force the coefficients to be designed directly in CSD format to minimize errors, to keep the difference between the approximating and ideal filter response within the specified tolerances, and to introduce same sub-expressions in every coefficient of the filter to reduce complexity of the multiplier size and to limit pass-band and stop-band ripples below a certain level etc.

2. *DESIGN OF FIR FILTER DIRECTLY IN CSD FORMAT*

Reference [1] describes a discrete approach for generating an optimal digital design of a CSD filter using ILP technique. The objective function is written so as to minimize two

characteristics. One, the deviation of the frequency-response of the filter form the ideal response and two, the number of non-zero digits used in the CSD representation of filter coefficients. FIR filter design demonstrated here is for type-I FIR filters only. After the ILP generates all the 15-bit coefficients of the filter in CSD format, these are fed to the filter structure. Each position of the non-zero bit in a coefficient corresponds to a created in the multiplication process. Each partial product, which is actually the word that is input to the filter, is shifted by a certain amount. The hardware of the filter is shown in Figure 21.

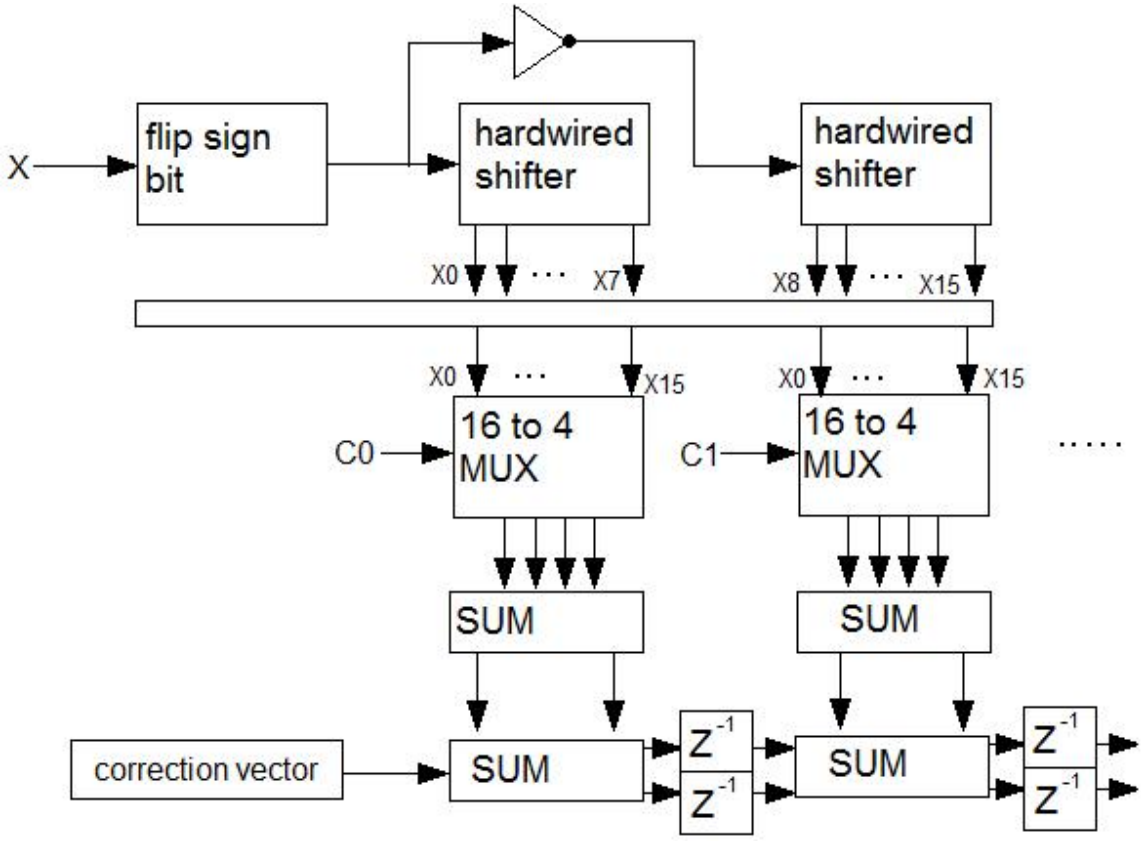

Figure 21: Block diagram of proposed FIR filter

The word ‘X’, input to the filter is to be multiplied by each coefficient. This is done by first converting the filter structure in its transposed direct form. Each

multiplication is accomplished by a shift-and-add operation according to the position and sign of each of the four non-zero bits in the CSD representation of the coefficient. The fastest way of accomplishing it is to shift each input through all the 15 bits to represent all the possible partial products. This is also accompanied by taking 2's complement of each shifted word to simulate partial product generated by negative non-zero bits of the coefficients. This creates all the 30 possible partial products.

Each coefficient is input to a 16 x 4 multiplexer. At the input to the mux the shifted words are fed. The mux selects 4 of these 16 words as 4 partial-products each for a non-zero in the coefficient. These partial-products are added together using a carry-sum adder to obtain a partial-sum and a carry vector. Each tap of the filter produces a partial-sum and a carry vector, these are delayed by the required clock cycles using delay elements (see Figure 9) and added together with the global carry vector to complete the convolution addition process and obtain the output.

CHAPTER 4

1. MATHEMATICAL REPRESENTATION:

1.1. **Basic Equation of the FIR filter:** As per reference [13], FIR filter Frequency response for a type-I filter and explanation of its variables is given by its Fourier Transform equation as below :

$$H(\omega) = \sum_{n=0}^{M} h[n]\cos(n\omega) \quad \text{when n = 0,} \qquad \text{---- Eq 3}$$

and

$$H(\omega) = \sum_{n=0}^{M} h[n]2\cos(n\omega) \quad \text{when n} \neq 0 \qquad \text{---- Eq 4}$$

M is half the filter order. i.e. M=(N/2) or (N-1)/2, when N is even or odd respectively. And h[n] is the nth coefficient of the impulse response,

This is the basic equation for the frequency response of the FIR filter and will be used for modeling in the ILP solver.

1.2. **Modeling for the ILP Solver:** ILP Solver requires a suitable mathematical model for producing the filter coefficients. The basic filter frequency response equation at paragraph 12 above is used. This equation is suitably modified [1].

1.3. **Model for Filter Coefficients in CSD format**: The filter coefficients, if designed directly in CSD format will reduce the quantization errors which occur when coefficients are designed as decimal and converted to fixed point numbers and then to CSD for use in a typical DSP [1]. The coefficients are therefore designed

directly in CSD format by imposing suitable constraints on the ILP solver. To design filter coefficients directly in CSD format equation 5 is used.

$$h[n] = \sum_{j=0}^{L-1} 2^{-j} (a_{nj} - b_{nj}) \qquad \text{---- Eq 5}$$

The representation defines the filter coefficient h[n] bit by bit, starting from the MSB towards the LSB as j varies from 0 to L-1. The variable L is the number of bits used to define h. In [1] filter coefficient "h" is fixed to 16 bits therefore j varies from 0 to 15. The term 2^{-j} defines the weight or "position" of each bit in h according to the binary number specifications. The variables a & b are Boolean i.e. they can have a value of 1 or 0 only. Together these variables define each bit of h[n] in terms of 0, +1 or -1 as required in the CSD format. When a=1 and b=1, (a – b) will be 0 and there will be a "0" at the bit position defined by 2^{-j}. Similarly, a=1 and b=0 defines a "1", (positive non-zero bit) and finally a=0 and b=1 defines a "-1" (negative non-zero bit) at the specified bit position.

1.4. **Expression for Objective Function**: This expression defines the objective function of the FIR filter. The function is desired to be minimized so that the filter coefficients are formed using the least number of non-zeros.

$$\text{Minimize} \quad \sum_{n=0}^{M-1} \sum_{j=0}^{L-1} (a_{nj} + b_{nj}) \qquad \text{---- Eq 6}$$

In the above expression, N is the filter order and M=half filter order. It is desired to be minimized so that the desired filter response can be achieved with the minimum possible order of the FIR filter.

L is the same as above i.e. the number of bits to which h has been fixed (16 in this case). This is also to be minimized.

δ1 is the maximum allowed ripple amplitude and required to be minimized.

The expression after the addition sign i.e. the double summation is for minimizing the number of non-zero elements used to represent all the filter coefficients, h[n].

1.5. **Model for Constraints of the ILP**: Four types of constraints are introduced to define filter coefficients as per requirements. These constraints are defined below:

1.5.1. **Frequency Response constraints**: These are used to define the filter shape as a function of frequency "ω" and are given by equation 7.

$$H(\omega) = \sum_{n=0}^{M} h[n]\cos(n\omega) - \delta_1 \leq D(\omega) \qquad \text{---- Eq 7}$$

The above equation is a derivation of the basic filter equations 3 and 4

δ_1 is the maximum allowable band ripple. I.e. when the equation is being used in the pass-band δ_1 is the pass-band ripple and in the stop band it is the stop-band ripple. D(ω) is the ideal low-pass filter response at the frequency "ω".

D(ω) = 1 in the pass-band

D(ω) = 0 in the stop-band

This constraint model limits the magnitude of H(ω) at the frequency "ω" to vary at the most by δ1 from the ideal behavior "D(ω)".

Since h[n] is required to be in CSD format, its value defined by Eq 5 is used in the above equation. The CSD-format model for the pass-band is given by equations 8 and 9.

$$\sum_{n=0}^{M-1}\sum_{j=0}^{L-1} 2^{-j}(a_{nj} - b_{nj})2\cos(n\omega) \le 1 + \delta_1 \qquad \text{---- Eq 8}$$

and

$$\sum_{n=0}^{M-1}\sum_{j=0}^{L-1} 2^{-j}(a_{nj} - b_{nj})2\cos(n\omega) \ge 1 - \delta_1 \qquad \text{---- Eq 9}$$

ω varies from to 0 to ωc (cut off frequency) and δ1 is the maximum allowable ripple in the pass band. For the stop-band the madel is given by equation 10 and 11.

$$\sum_{n=0}^{M-1}\sum_{j=0}^{L-1} 2^{-j}(a_{nj} - b_{nj})2\cos(n\omega) \le +\delta_2 \qquad \text{---- Eq 10}$$

and

$$\sum_{n=0}^{M-1}\sum_{j=0}^{L-1} 2^{-j}(a_{nj} - b_{nj})2\cos(n\omega) \ge -\delta_2 \qquad \text{---- Eq 11}$$

Here ω varies from to ωc to π and δ2 represents the maximum allowable ripple in the stop-band.

In Eq 8,9, 10 and 11 the term 2 cos $(n\omega)$ is used. This condition exixts so long as when n > 0. When n = 0, this term is replaced by unity.

1.5.2. **Four Non-Zero Bits Constraint**: A useful property of the CSD format is that even by considering a maximum of 4 MSB non-zero bits in the entire coefficient, rounding-off error is still very low. In this model we will constrain each coefficient to have a maximum of 4 non-negative bits. This constraint is modeled as per equation 12.

$$\sum_{j=0}^{L-1}(a_{nj}+b_{nj})=4 \qquad \text{---- Eq 12}$$

This constraint defines that the sum of all bits of h[n] should be less that or equal to 4. Here n defines the nth coefficient of the filter.

1.5.3. **CSD Format Constraint**: Since in CSD format maximum number of non-zero bits can only occur at alternate locations within the CSD word, a constraint has to be defined to place non-zero bits in a CSD Coefficient at alternate locations. The variable a and b are Boolean so the model for this constraint is given by equation 13.

$$a_{nj}+b_{nj}+a_{nj+1}+b_{nj+1}\le 1 \qquad \text{---- Eq 13}$$

this expression defines that for all n and j sum of two consecutive bits should be less than or equal to 1, so if one bit is 1, the next cannot be 1. This ensures that non-zero bits occur at alternate locations.

1.5.4. **Boolean variables constraint**: The variables a and b are Boolean. These are defined in an ILP as Boolean integers. Mathematically this is represented as:

$$a_{nj} \quad b_{nj} \quad \in \{0,1\}$$

2. ***DESIGN IMPROVEMENT INTRODUCED*:**

Reference [1] gives the details of the design of an FIR filter directly in CSD format using an ILP. It has been established in [15] that if the set of coefficients of the FIR filter have more CSE, more reduction in its complexity is achieved. An effort has been carried

out in this thesis to further constrain the ILP such that the FIR filter coefficients are designed directly in CSD in a way that maximizes commonality of sub-expressions found in the set of coefficients. This algorithm further reduces the redundancy and complexity of the filter structure. This is expected to further improve hardware implementation of the circuit in terms of area-on-chip, power consumption and speed of operation by reducing the number of adders required to reduce partial-products [15].

3. ***ALGORITHM ADAPTED FOR INTRODUCING COMMONALITY*:**

Statistically, horizontal common sub-expressions, 101, $10\bar{1}$, 1001, and $100\bar{1}$ occur more frequently in the CSD form of LPFIR filters and hence these sub-expressions are the most common horizontal sub-expressions [15].

The ILP calculates the filter coefficients directly in CSD format optimized to minimum number of non-zeros; it is also ensures that the coefficients are designed so that the above sub-expressions occur most frequently in them. These results in an algorithm that calculates FIR filter coefficients so that the filter produces the desired response as well as the filter coefficients exhibit maximum CSEs. The algorithm is introduced in the ILP as a set of constraints; mathematically these constraints are expressed as per Equation 14.

$$\sum_{j=0}^{L-3} a_{nj} + b_{nj} + a_{nj+2} + b_{nj+2} = 2 \qquad \text{---- Eq 14}$$

In the above expression, ‘a’ and ‘b’ are the variables of the ILP. Two sets of variables at location j and j+2 are selected within a CSD coefficient. This algorithm intentionally leaves the in-between location i.e. j+1, the ILP chooses these 4 variables such that their

sum is 2 i.e. bits at location j and j+2 become positive non-zeros, leaving the bit at location j+1 as zero. This expression can introduce the sub-expressions of '1 0 1', '1 0 $\bar{1}$', '$\bar{1}$ 0 1' and '$\bar{1}$ 0 $\bar{1}$' in each coefficient. The location of the sub-expression is not fixed and the ILP 'places' this sub-expression at a location which optimizes the coefficient, keeping in view the other constraints. In a similar way to introduce the sub-expression of '1001', '1 0 0 $\bar{1}$', '$\bar{1}$ 0 0 1' and '$\bar{1}$ 0 0 $\bar{1}$' in the coefficients equation 15 is used.

$$\sum_{j=0}^{L-4} a_{nj} + b_{nj} + a_{nj+3} + b_{nj+3} = 2 \qquad \text{---- Eq 15}$$

CHPATER 5

1. *COMAPRISONS:*

1.1. Comparing Filter Design by Conventional Methods

To demonstrate how conventional filter design techniques mentioned in chapter 2 help achieve complexity reduction, a comparison has been carried out by designing a particular filter by these techniques and then comparing the results. The criteria for complexity reduction will be the number of adders required for partial product reduction. The lesser the number of partial-product reduction adders used in an FIR filter, the higher the degree of complexity reduction achieved. Using this analogy, a Linear Phase, Low Pass Filter of the following specifications have been designed.

Cut off Freq: 0.4π

Filter order: 10

The impulse response of the filter is shown in Figure 22.

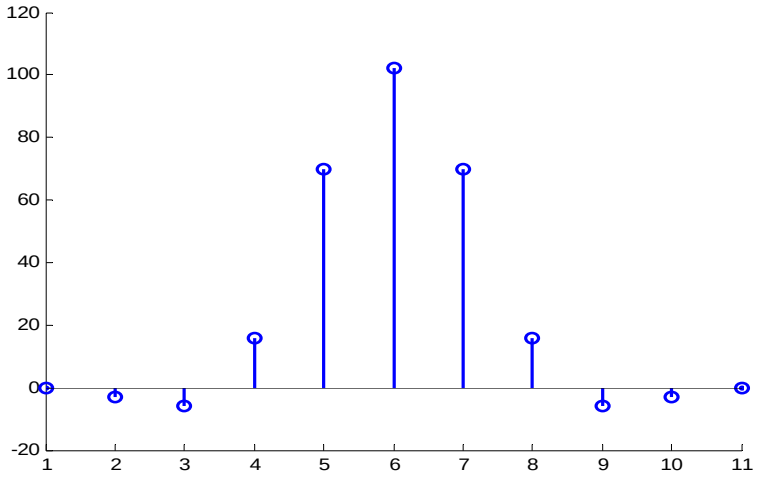

Figure 22: Impulse response of the simple-binary filter

1.2. Design by MATLAB (Binary):

This is the simplest method of filter design. In this method the coefficients of the filter are produced in MATLAB. These are presented as real, floating point, decimal integers. For design procedure, these decimal fractions are changed from floating point to fixed point decimal numbers which can be used on a Digital Signal Processor (DSP). The fixed point numbers are changed to Binary digits which are then used as multiplicands in the multipliers of the FIR filters. an example of this process is as follows:

Matlab calculates coefficients of the above filter as:

0, -0.0126, -0.0247, 0.0635, 0.2748, 0.3981, 0.2748, 0.0635, -0.0247, -0.0126, 0

These fractions are fixed to fixed point numbers fixed for an 8 bit DSP processor. The fixed values are:

0, -3.236, -6.3212, 16.2573, 70.3482, 101.904, 70.348, 16.2573, -6.3212, -3.236, 0

These fixed point numbers are then converted to integers as follows:

0 -3 -6 16 70 102 70 16 -6 -3 0

Now these integers are converted to binary numbers that can be finally used as coefficients in the FIR filter. These binary coefficients are as below:

h[0] = 0000000
h[1] = 0000011
h[3] = 0000110
h[4] = 0010000
h[5] = 1000110
h[6] = 1100110

h[7] = 1000110
h[8] = 0010000
h[9] = 0000110
h[10] = 0000011
h[11] = 0000000

As per reference [4], the number of adders required to implement partial produce reduction are N_b-1 = **19 Adders**. Where N_b is the number of non-zero bits in all filter coefficients.

1.3. **Structure folding*:*** In this technique, symmetry property of the FIR filter is exploited. In this case the last 5 multipliers are removed and the results of the first 5 multipliers are re-used in the last 5 taps. This reduces N_b to 12 and the number of adders required to **11 Adders.**

1.4. **CSD Method*:*** Here the coefficients of the filter are expressed as CSD format the changes the coefficients to:

h[0] = 0000000
h[1] = $000010\bar{1}$
h[3] = $00010\bar{1}0$
h[4] = 0010000
h[5] = $10010\bar{1}0$
h[6] = $10\bar{1}010\bar{1}0$
h[7] = $10010\bar{1}0$
h[8] = 0010000
h[9] = $00010\bar{1}0$
h[10] = $000010\bar{1}$
h[11] = 0000000

In this case the number of non-zero bits i.e. N_b=20 and the number of adders required are N_b -1 = **19 Adders required**. This is a special case where the nature of the coefficients is such that the coefficients even if represented in CSD format will not make any difference from the coefficients represented in binary method.

1.5. **Perturbing Coefficients*:*** In this method we change coefficients so that they become an exact power of 2. Although this will introduce deviation in the approximated frequency response of the filter, it will reduce the structural complexity. After defining our permissible error tolerances etc we perturb the coefficients which are near to the exact powers of 2. the perturbed coefficients become,

h[0] =	0000000
h[1] =	0000010
h[3] =	0001000
h[4] =	0010000
h[5] =	1000000
h[6] =	1100000
h[7] =	1000000
h[8] =	0010000
h[9] =	0001000
h[10] =	0000010
h[11] =	0000000

The number of non-zero bits become 10 and N_b = **9 Adders required.**

The maximum error introduced is 20% since the maximum change is in h[6] (perturbed from 120 to 96), which is a very high error.

1.6. **Computation Sharing**: In this method, the coefficients are generated using the first 7 prime numbers. These prime numbers are multiplied with the digitized data input to the FIR filter; the results are stored and re-used.

h[0] =	0	=	0
h[1] =	3	=	3
h[3] =	6	=	5 + 1
h[4] =	16	=	13 + 3
h[5] =	70	=	17 + 17 + 17 + 17 + 2
h[6] =	102	=	17 + 17 + 17 + 17 + 17 + 17
h[7] =	70	=	17 + 17 + 17 + 17 + 2
h[8] =	16	=	13 + 3
h[9] =	6	=	5 + 1
h[10] =	3	=	3
h[11] =	0	=	0

Total Adders required = Adders required prime factors + Addition the factors

= 5 Adders + 17 Adders = **22 Adders Required**

The above methods have been elaborately explained in chapter 2. Each method yield different results in complexity reduction achievement. These are summarized in the following table:

Ser No	Design Method	Adders required
1	Matlab	19
2	Structure Folding	11
3	CSD Format	19

4	Perturbing Coefficients	9
5	Computation Sharing	22

Table 6: Comparison of Complexity Reduction Achieved

A comparison of the methods from the above table reveals that the best method in this particular case is the design by perturbing the coefficients. This method can achieve high reductions in structure complexity of filters if the errors introduced remain within the permissible tolerances (20% in this case). However, if an error of 20 % is acceptable for any application, then this will be the best available method. Otherwise for an accurate approximating response, structure folding technique is the best. The computation sharing method usually achieves good results in filter coefficients with more that 8-bits.

Additionally, these methods can be used in combination also. E.g. Structure folding method can be used in conjunction with CSD format or with computation sharing to achieve compounded reduction in structural complexities.

2. *COMPARING FILTER DESIGN BY LP METHODS*

Using the algorithm stated in chapter 5, FIR filters with different N and L were designed in Matlab by installing its linear program toolbox. Initially all the above constraints except those given by Eq 11 and 12 were imposed and filter coefficients obtained, next constraints given by Eq 11 and 12 were also introduced and the coefficients re calculated. Resulting filter response is given in Figure 3. It can be seen that the two

curves are almost identical, the curve of the filter with introduced CSE shows lower side lobes than the one with out CSE, which demonstrates that the frequency response of the filter almost remains the same or becomes better, before and after introducing the common sub-expressions. Number of adders required to reduce the partial products of the filter with $\omega c = 0.2\pi$, $\delta 1 = 0.1$, $\delta 2 = 0.05$ were calculated as per procedure in [4]. The results are given in Table 1. The "Saving Expected" column in Table 1 gives the savings in adders expected by introducing CSE. In some cases the ILP, while calculating the rest of the non-zeros in the coefficients uses the same combination of non-zeros as that of the CSE introduced, this causes the number of sub expressions to become more than those actually introduced, thereby achieving greater savings. This is shown in the "Actual Savings achieved" column. This can be seen in case of coefficients h[0] and h[2] (In the CSD Coefficients with introduction of CSE '1 0 1') given below. In this case, the CSE 1 0 1 was introduced in every coefficient. In h[0] and h[2], the ILP re-used this in the coefficient once more, thereby increasing the number of CSEs from 5 to 7.

Adders required without CSE => Nb – 1

Where Nb => Total no of non-zeros in filter matrix

Adders required with CSE => Na = (Nb – 1) – 2 Ns + Nas

Where Na => Total no of adders for partial product reduction by CSE introduction.

Ns => no of CSE in half of the coefficient matrix

Nas => no of adders required to calculate the CSE

Sr No	Filter Order 'N'	Coefficient length 'L' bits	Adders required without CSE introduction	Adders required with CSE introduction	Savings Expected	Actual Savings Achieved
1	10	14	39	26	25.6 %	33.3%
2	14	10	41	30	26.8%	34.14%
3	16	10	47	32	31.9%	31.9%

Table 7: Showing complexity reduction achieved.

For the filter at serial 1 in Table 6, coefficients of FIR filter with and without introduction of the CSE are shown below. (Due to symmetry, only 5 coefficients of the 10th order filter are shown)

Coefficients without introducing CSE in CSD format:

h[0] = 0 0 1 0 1 0 0 1 0 $\bar{1}$ 0 0 0 0

h[1] = 0 1 0 0 1 0 0 $\bar{1}$ 0 $\bar{1}$ 0 0 0 0

h[2] = 0 1 0 $\bar{1}$ 0 $\bar{1}$ 0 1 0 0 0 0 0 0

h[3] = 0 0 0 1 0 $\bar{1}$ 0 $\bar{1}$ 0 1 0 0 0 0

h[4] = 0 0 0 $\bar{1}$ 0 0 1 0 0 0 $\bar{1}$ 0 $\bar{1}$ 0

Coefficients in decimal:

-0.0553 0.0439 0.1758 0.2764 0.1592 0.1592 0.2764 0.1758 0.0439 -0.0553

Coefficients with introduction of CSE ‘1 0 1’ in CSD format:

h[0]= 0 0 1 0 1 0 0 0 0 0 0 (1 0 1)

h[1]= 0 1 0 0 0 0 0 $\bar{1}$ 0 0 (1 0 1) 0

h[2]= 0 0 1 0 1 0 0 0 0 (1 0 1) 0 0

h[3]= 0 0 0 1 0 0 0 0 (1 0 1) 0 $\bar{1}$ 0

h[4]= 0 0 0 0 $\bar{1}$ 0 0 (1 0 1) 0 0 0 $\bar{1}$

Coefficients with introduction of CSE ‘1 0 1’ in decimal:

-0.0264 0.0648 0.1575 0.2467 0.1566 0.1566 0.2467 0.1575 0.0648 -0.0264

In Figure 23, frequency response of filter at ser no.1 of Table 1 is plotted. The blue curve shows filter designed without introducing CSE and the red curve shows filter design with CSE introduced. Change in the response is negligible.

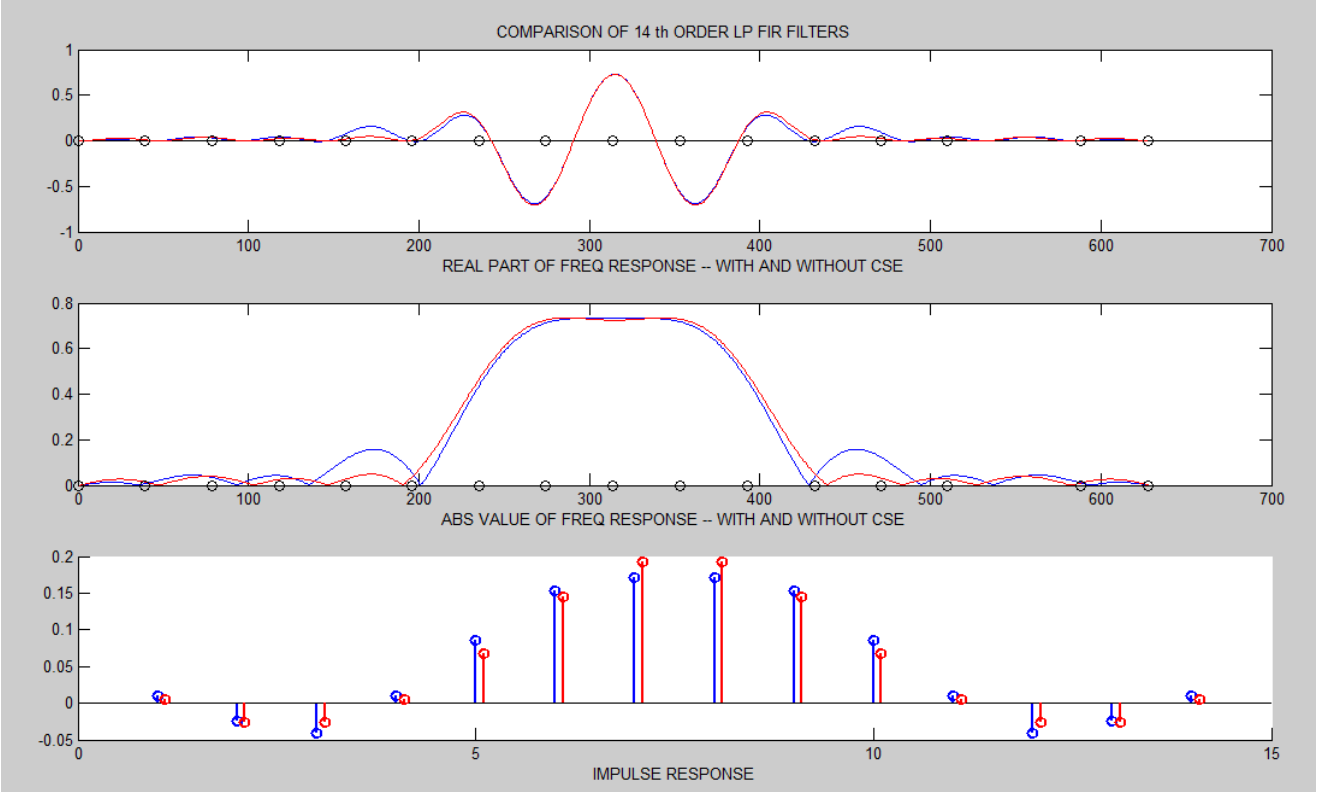

Figure 23: Comparison of filter response

CHPATER 6
CONCLUSIONS

1. Overview.

The present is a time with lifestyle trends pointing towards using micro-sized electronic stand-alone units using embedded digital hardware and running complex multipurpose applications using micro-sized batteries. These devices invariably use digital signal processors with embedded digital hardware running audio, video and other multimedia applications. Cell phone is a classic example of one of the most-used stand-alone device in the world. Since all such portable devices are oriented towards power savings, it becomes essential to design all sub-components of these devices so that they are power efficient.

FIR filters being a size-intensive component, reducing its size pays rich dividends in chip size and power consumption reduction. Many attempts have been carried out to achieve Increase in processing speeds of an FIR filter with simultaneous reduction in area occupied on chip and reduction in power consumption; some of these have been covered in this work. The option of using an Integer Linear Program (ILP), being the core issue in this work, has been covered in relatively more detail. It has been shown that the flexibility of an ILP can be exploited in the design of coefficients of an FIR filter. The ILP directly calculates the coefficients of the FIR filter in CSD format. This reduces the quantization errors which are parasitic in procedures which involve designing the filter coefficients as real numbers in the first step, converting these real numbers into fixed point numbers for use on a DSP and then by converting them into CSD format. Parameters of FIR filter such as ω_p, ω_s, N, L, δ_1, and δ_2 are specified as inputs to the ILP. The output of the ILP are

coefficients that are designed directly in CSD format which minimizes the maximum error between the frequency response of the filter and the desired frequency response while adhering to the constraints specified in the inputs. An algorithm has been devised which constraints the ILP to maximize commonality of the sub-expressions automatically, thereby reducing the number of adders required for convolution-summation in FIR filters by almost a third. This results in substantial saving in hardware which results in reduction in area-on-chip, power consumption and improvements in processing time of the filter.

2. Limitation of the Approach.

This thesis provides a new idea of using an ILP to maximize CSEs in and FIR filter. All work is based on research in the area of using CSE for filter performance and size improvements. Algorithm devised to maximize commonality of sub-expressions in filter coefficients is simple in approach and achieves an approximate reduction of 30% in the number of adders used in the filter. This algorithm a limitation that CSE which will effectively maximize commonality within the coefficient matrix of the filter has to be manually devised and introduced into all the coefficients as a constraint on the ILP. The calculations in reduction of hardware used in this work are based on methods specified in reference [4] and not by the implementation of the designed filter in actual hardware. This analysis gives relative improvements achieved between various designs and not absolute values of these parameters.

3. Future Work.

The efficiency of the ILP can be enhanced by re-designing the algorithm for maximizing commonality. Intelligent algorithms can be designed to automatically analyze

the filter coefficient matrix, design the CSE which will maximize the commonality and automatically write it as a constraint in the ILP.

Further improvements in the analysis part of the work can be carried out by actually implementing the designed filter on an FPGA or similar hardware and then calculating the actual results of the occupied area on-chip, speed of operation and power consumption. this will achieve a concrete analysis of the actual results.

REFRENCES

1. Optimal Design and Implementation of CSD FIR Filter, M. Sohail Sadiq and Shoab A. Khan.
2. Understanding Multiplier Design, using "Overturned-Stairs" Adder Trees, Magnus Karlsson, Oscar Gustafsson, J Jacob Wikner, Thomas Johansson, Weidong Li, Magnus Hörlin, Hans Ekberg, Department of Electrical Engineering, S-58183 Linköping University, Sweden.
3. FAST 32-BIT DIGITAL MULTIPLIER, Koomran Roohemifor, Mojid Ahmodi a, 'Electrical Engineering Department, University of Windsor, Windsor, Ontario, Canada.
4. Low Complexity FIR Filters Using Factorization of Perturbed Coefficients' Cassondra Neau, Khurram Muhammad, and Kaushik Roy.
5. CSDC, a New Complexity Reduction Technique for Multiplier less Implementation of Digital FIR Filters, Yongtao Wang and Kaushik Roy, IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS—I: REGULAR PAPERS, VOL. 52, NO. 9, SEPTEMBER 2005 1845.
6. A New Parallel Multiplication Algorithm And Its VLSI Implementation, Bhabani P Sinha and Pradip K Srimani.
7. Common Subexpression Elimination Involving Multiple Variables for Linear DSP Synthesis, Anup Hosangadi, Farzan Fallah, Ryan Kastner, 15th IEEE International Conference on Application-Specific Systems, Architectures and Processors.
8. Exploiting General Coefficient Representation for the Optimal Sharing of Partial Products in MCMs Eduardo Costa Paulo Flores Jos_e Monteiro.
9. Lecture on Numeric strength reduction by Giedrius ZAVADSKIS.

10. Optimization Method for Broadband Modem FIR Filter Design using Common Sub-expression Elimination Robert Pasko, Patrick Schaumont , Veerle Derudder, Daniela Durackova.
11. Wikipedia website for literature on ‘Linear Programming’.
12. Constrained FIR Filter Design by the Method of Vector Space Projections, Khalil C. Haddad, Henry Stark, Fellow, IEEE, and Nikolas P. Galatsanos.
13. Handbook of Digital Signal Processing By Sanjit K Mitra.
14. Handbook of Digital Signal Processing by Douglas F Elliot.
15. FIR filters implementation by efficient sharing of horizontal and vertical common sub-expressions, A.P.Vinod, E.M-K.Lai, A.B.Premkumar, and C.