

OPTIMIZING FIR FILTER COEFFICIENTS
USING CSD REPRESENTATION
AND DM TECHNIQUE

By

WEN FUNG LEONG

Bachelor of Science
Oklahoma State University
Stillwater, Oklahoma
2000

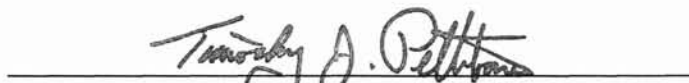
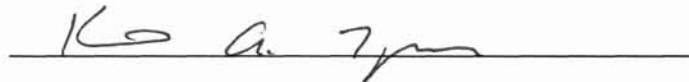
Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
In partial fulfillment of
The requirements for
The Degree of
MASTER OF SCIENCE
May, 2002

OPTIMIZING FIR FILTER COEFFICIENTS
USING CSD REPRESENTATION
AND DM TECHNIQUE

Thesis Approved:



Thesis Adviser



Dean of Graduate College

Preface

Hardware saving criteria is one of the research fields in digital signal processing. Particularly on hardware saving in digital filter designs, which are due to the usefulness of digital filters in various fields especially in engineering fields. The main advantages in hardware savings are reduce hardware complexity and reduce the usage of hardware resources. One particular interest is to write a CAD program that can optimize the require hardware for the filter design. Many researchers have developed various CAD software programs for digital filter design with hardware realization capability. Some of the researchers include the hardware reduction as the main theme of the CAD program.

This whole thesis is related to the second-generation public domain CAD program and the main priority is to generate a set of FIR filter coefficients that requires little hardware. The program is to allow users to design any transpose direct form linear phase FIR filter with the least hardware needed and to be user friendly. However, the main direction of this thesis is to discuss the techniques and algorithms that are incorporated into the program with the main goal of meeting the filter specifications configured by the user. One of the sections discusses the implementation of Canonical Signed Digit (CSD) representation to the filter coefficients. The CSD coefficients are further optimized by the three optimization algorithms, which contribute in more hardware savings for the filter.

Another section introduces another representation technique that provides better results compared to CSD representation. This technique is called as the Dempster and Macleod (DM) technique. The Signal Noise Ratio is also included to determine the structural adder or delay size of the filter. The thesis also covers the details of the CAD programs such as the instructions to use the program and the block diagrams that construct the program.

Acknowledgements

I sincerely thank my research advisor Professor Michael A. Soderstrand, who provide the advice, patience and support through my master program. Under his guidance, I have not only learned a lot from him but also learn to understand the basics of DSP, which I had studied during my undergraduate program. I also acknowledge Professors Louis G. Johnson and Keith A. Teach for reviewing my thesis and serving as my thesis committee. I thank all my Digital Signal Processing and Communications (DSPC) research colleagues for being cooperative and provide a friendly environment for our lab. In addition, I thank all the faculties and staffs from Electrical and Computer Engineering Department of Oklahoma State University. My special thank to Professors H. Jack Allison and Gary G. Yen who not only help me a lot while pursuing my master degree but for their inspiration, support and encouragement. I express my deepest gratitude to my family for their love, encouragement, support and patience. Moreover, I express my appreciation to my parents who have supported and provided me the opportunity to study in Oklahoma State University. Finally, a particular thanks to Professor Joan Barrick for proofread the thesis.

Table of Contents

1. Introduction.....	1
1.1 Introduction.....	1
1.2 Objective.....	2
1.3 Finite Impulse Response Filter (FIR)	3
1.4 Fixed Point Representation.....	5
1.5 Thesis Outline	6
 2. Background.....	 9
2.1 Other Filter Design Programs	9
2.2 Cost Definition.....	10
2.3 Order-Wordlength Tradeoff.....	11
2.4 Scaling.....	13
2.5 Adder Extraction.....	13
2.6 Roundoff Error Analysis.....	14
 3. Cost Analysis of Binary, CSD and DM Implementation	 15
3.1 Introduction.....	15
3.2 Number Representations.....	16
3.2.1 Binary Representation.....	16
3.2.2 CSD Representation.....	17
3.2.3 DM Representation	18
3.3 Cost Analysis of Binary, CSD and DM Representation	19

4. CSD Optimization Techniques	27
4.1 Introduction.....	27
4.2 Order-Wordlength Tradeoff.....	27
4.2.1 Procedure for Determining Optimum Order and Wordlength.....	28
4.2.2 Function Introduction.....	31
4.3 Scaling.....	35
4.3.1 Theory	35
4.3.2 Procedure of Determining Minimum Order and Wordlength.....	35
4.3.2 Function Introduction.....	37
4.4 Adder Extraction	40
4.4.1 Procedure of Adder Extraction	40
4.4.2 Function Introduction.....	41
4.5 Example Result of Three Optimization Techniques.....	44
 5. Dempster and Macleod Implementation	 47
5.1 DM Optimization Introduction	47
5.2 DM Optimization Procedure.....	48
5.3 Function Introduction.....	49
5.4 Implement DM technique after Optimization.....	50
 6. Signal to Noise Ratio.....	 54
6.1 Introduction and Approach	54
6.2 SNR Computation.....	55
6.3 Pseudocode For SNR Computation	58
6.4 Example Simulation Result.....	60
 7. Program Review.....	 64
7.1 GUI Overview.....	64
7.2 Program Overview	67
7.3 Technology and Cost Review	70

8. Conclusion and Future Work	72
8.1 Conclusion	72
8.2 Future Works	74
BIBLIOGRAPHY	76
Appendix A: Binary to CSD representation Conversion Algorithm.....	80
Appendix B: CSD and DM FIR Filter Design Program.....	82
B.1 Introductory Menu	82
B.2 File Chart	84

List of Figures

Figure 1.1	General Transpose Direct Form FIR filter	5
Figure 2.1	Graph of Cost (Nb) vs b	12
Figure 3.1	(a) Binary Implementation of Multiply by 93	16
	(b) CSD Implementation of Multiply by 93	16
	(c) DM Implementation Multiply by 93	16
Figure 3.2	Histogram of Number of adders Vs Number of Combinations for 8-bits wordlength	21
Figure 3.3	Histogram of Number of adders Vs Number of Combinations for 16-bits wordlength	24
Figure 3.4	Plot of Average Adder for Coefficient Multipliers with Wordlength Range from 4 bits to 20 bits	26
Figure 4.1	Optimum Order, <i>Nopt</i> Algorithm Block Diagram	32
Figure 4.2	Optimum wordlength, <i>Bopt</i> Algorithm Block Diagram	33
Figure 4.3	<i>newcsd_fir</i> Subroutine Block Diagram	34
Figure 4.4	Scaling Algorithm Block Diagram	38
Figure 4.5	<i>newcsd_firsc</i> Subroutine Algorithm Block Diagram	39
Figure 4.6	Adder Extraction Optimization General Routine Block Diagram	42
Figure 4.7	Adder Extraction Algorithm Block Diagram	43
Figure 5.1	DM Optimization Algorithm Block Diagram	53
Figure 6.1	Transpose Direct Form FIR Filter with Noise Model	56
Figure 6.2	Pseudocode of SNR subroutine	59
Figure 7.1	CSD and DM FIR Filter GUI	65
Figure 7.2	An Example Summary Box	66
Figure 7.3	Summary Data in Work Window Command.	66
Figure 7.4	General Program Flow	68
Figure 7.5	An Example CSD <i>params.vhd</i> File generated by Matlab	68
Figure 7.6	Program Flow Chart	69
Figure B-1	CSD and DM FIR filter design program.	83
Figure B-2	Top Level of Program File	84
Figure B-3	File Chart for the Optimization Process	85

Section One

Introduction

1.1 Introduction

Digital Signal Processing (DSP) has grown widely during the past decades, due to the technological advancement in computers. DSP has revealed more advantages in terms of flexibility, cost reduction and performance compared to conventional analog signal processing. Additionally, its advantages also cover versatility and stability. In the industry, areas that require DSP applications such as medical applications, time series analysis, and communication areas use digital signal processors with specialized hardware, instructions and mathematical computations instructions for processing digital signal projects.

DSP operations usually have two basic categories, which are signal analysis and digital filtering. Implementation of new structures using digital filters is very popular due to the flexibility of various applications of digital filters such as noise elimination and frequency band separations. In general, a digital filter is a discrete time convolver since the output of the filter is actually the sum of the weighted previous input and output.

Therefore, a digital filter is actually formed by three basic hardware elements, which are adders, multipliers and registers.

There are numerous automated software programs available for engineers to design any type of filter with reduced hardware requirements and the ability to implement the design in different technologies. Designing a digital filter that has less hardware complexity is important because reduced complexity can contribute to a reduction in power, size and cost while increasing speed. The main contribution to complexity for digital filtering is the coefficient multipliers, which are usually implemented using binary arithmetic. There are many ways to reduce hardware complexity that range from Hartley[10] numerical reduction method to direct building (Finite Impulse Response) FIR filter using components available in the hardware realization software such as KCM multipliers [20]. With new techniques implemented to improve the digital filter design with hardware reduction as a goal in mind, both Canonical Signed Digit (CSD), and Dempster and Macleod (DM) techniques appear very promising for filter design purposes.

1.2 Objective

Previously, Husinga [1] developed an automated software program to design highpass and lowpass FIR filters by using the CSD technique to minimize hardware complexity. This Computer Aided Design Program is designed for linear phase transpose direct form Finite Impulse Response (FIR) filters with minimal hardware complexity. There are three optimization techniques that are used to obtain better results with the

CSD technique. The three techniques used are order-wordlength tradeoff, scaling and adder extraction. These are the three objectives in this thesis. Firstly, Husinga's work is revised, identifying some errors and limitations occurring during the optimization process. Then, the program is expanded by adding more features and selections. Finally, a new hardware saving technique based upon the Dempster and Macleod (DM) implementation [6-7] is introduced into the program. This program is aimed at automatically generating a hardware layout for different targeted technologies such as Xilinx FPGA's and MOSIS CMOS processes. This paper will not cover the automation of hardware layout. However, a script file called the *params.vhd* is mentioned since it is included in the GUI program as one of the functions. The program does offer choices for different targeted technologies that can be added in the future when needed.

1.3 Finite Impulse Response Filter (FIR)

According to many textbooks and articles [1-2],[16-19], FIR filters are simple to design, flexible and there is no possibility of limit cycles. In addition, the reason that FIR filters are also stable is because it is guaranteed to be a bounded-input bounded-output (BIBO) system. Moreover, FIR filters with symmetrical characteristics will guarantee a linear phase frequency response as linear phase response produces constant amounts of delays, which make the design problem somewhat simpler. Therefore, the design of FIR filters is the focus for this thesis.

The transfer function of an FIR filter in the z-domain is

$$H(z) = \sum_{k=0}^{M-1} b[k]z^{-k} \quad (1.1)$$

Hence a general difference equation for causal FIR system is represented as

$$y[n] = \sum_{k=0}^{M-1} b[k]x[n-k] \quad (1.2)$$

This equation shows that if it is subjected to an impulse, the output will equal zero after the impulse has passed through all the summation. This operation is equivalent to convolution of input data samples with the desired unit impulse response of the filter, which is defined as Finite Impulse Response (FIR).

The transfer function from Equation (1.1) can be implemented using a variety of equivalent structures, which are equivalent in terms of transfer function but differ in hardware complexity. These structures, which are equivalent in terms of transfer function have different characteristics depending on quantization results due to finite wordlength limitation. Mitra [18] does mention a few equivalent FIR filter structures such as the direct form FIR filter structure, cascade form FIR filter structure, linear phase FIR structure and polyphase FIR structure. For this thesis, the linear phase form FIR filter is selected because of its wide applications. A characteristic of the linear phase form is that the coefficients are in symmetrical form and this can be exploited to reduce the number of filter coefficients by almost one half. In addition, the filter is selected for its transpose direct form because it supports the linear phase form FIR structure better in terms of hardware requirements for the filter coefficients. Transpose direct form also has a better timing characteristic since the input is directly connected to the filter coefficient

multiplication. Figure 1.1 shows an example of a general transpose direct form of a N -order linear phase FIR filter. However, there is a disadvantage of using transpose direct form, which is the size of the delay which must be a reasonable size to prevent quantization and this problem does not occur in direct form structure because all the delays are at the input bit size. In fact direct form structure will result in less hardware requirements compared to transpose direct form. In transpose direct form, half of the total number of adders are connected at the input bit size whereas the rest of the adders are connected at the output after the multipliers with the wordlength of input bit size plus the coefficients bit size. Another interesting point is if the sharing block in the CSD or DM realization is adapted to both the transpose direct form and direct form structures, the hardware requirement is much lower. Anyway, this thesis focuses on the transpose direct form FIR filters with no sharing block in CSD or DM realization, which is presented in Figure 1.1.

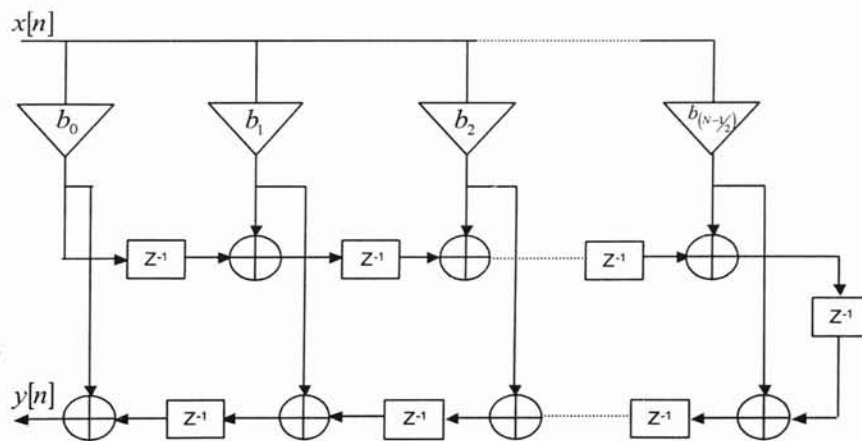


Figure 1.1 General Transpose Direct Form FIR filter

1.4 Fixed-Point Representation

Fixed-point format is used to represent the coefficients since this format is flexible both in DSP processing and VLSI implementations. This is because the binary point is fixed at a certain location of a binary representation of a coefficient. We use this representation to represent fractional numbers. A general fixed-point format for the fractional number with a dynamic range of -1 to approximate 1 is shown below:

$$b_0 \bullet b_1 b_2 b_3 \cdots b_{Bit} \quad (1.3)$$

where ' \bullet ' is the binary (radix) point.

According to Mitra [18], when the binary point is fixed, arithmetic operations are simpler to implement and the positive fixed-point number is easy to represent. In general, there are three formats to represent a fixed-point negative number. If we consider binary representation as the main basis to represent the fractions, the three formats are sign magnitude format, ones' complement format and twos' complement format. These formats do apply to signed radix-2 fixed-point numbers. Note that the binary point plays an important part in fixed-point representation.

1.5 Thesis Outline

Section 2 reviews previous work and research areas that are related to this thesis and it will include an overview of the CAD design. Other main topics included are cost calculations and optimization techniques that reduce the hardware requirement.

The following section will discuss the cost analysis of Binary, Canonical Signed Digit (CSD) and Dempster and Macleod (DM) techniques. A simple explanation for

these representations is to produce hardware savings. The simulated results are presented using the Matlab program for the ease of comparing each technique.

Section 4 describes and reviews the CSD optimization techniques that were implemented in this thesis taken from previous research work done by other authors. Basically it contains the procedure of all techniques with block diagrams and definition function, while integrating the algorithm step by step and translating them into block diagrams for future reference. The example results presented are also compared for robustness in difference techniques.

Section 5 will introduce the new method, Dempster and Macleod (DM) technique into the CAD as an added option for users and also to minimize hardware complexity. The DM optimized algorithm for this technique is also presented with a block diagram. When the filter coefficients are found, the coefficients will undergo optimization techniques and then they are implemented only with the DM technique. The reasoning behind this idea is discussed and example results from the Matlab program are presented.

Section 6 describes the computation of signal to noise ratio (SNR) in both cases. The first case is one in which the user does not specify a SNR requirement while the other case is to find the SNR value that meets the SNR requirement specified by the user. Both the procedures are discussed and a pseudocode and example results are included in this section.

Section 7 explains the routine of the CAD program. The features and block diagrams are provided to simplify the explanation. This chapter includes the introduction

of the program and the procedure to run the program. It also discusses the reason that the new cost function is introduced to the program.

The last section concludes this thesis based on the results discussed from the previous sections. Future work is also recommended to future researchers who are interested in improving the work done.

Section 2

Background

2.1 Other Filter Design Programs

Many programs have been written to construct a constant coefficient FIR filter based on various user-defined parameters. For example, Matlab version 6-demo toolbox for signal processing has a program that allows the user to compute a FIR filter with many options. Intensive research into writing software programs that includes a hardware implementation option has also been done by other researchers in the past. Research done by Jain et al. [11] produced a FIRGEN program that automatically generated the architecture and floorplan for integrated circuit fixed point FIR filters that achieves a high sample rate with compact layout. Another example is the construction of a FIR filter using the graphical toolbox available in Simulink toolbox provided by Mathworks in Matlab version 6. The design is then translated into hardware programming language, which produces hardware in Register-Transfer-Level (RTL) in VHDL that maps into a FPGA using CAD tools. The paper written by Haldar et al. [12] uses this technique. The authors presented their MATCH compiler that takes the Matlab input and implements the digital filter design in a FPGA chip. While Husinga and Darren [1-2] wrote a Matlab

program which does takes certain technological specification into consideration. The results from the program can be implemented in Xilinx FPGA's or other targeted technologies.

2.2 Cost Definition

Cost measurement is essential in order to evaluate the cost needed for a digital filter design while performing any optimization routines. Most papers define the cost based on the number of adders/subtractors and delay elements of a filter. Even so, cost function is not a unique function. In Dempster [8] the cost is defined as a cost equation for CSD representation, which is based on fewer adders needed for the multipliers compared to the binary representation. The Dempster cost equation is represented in the form of,

$$Cost = M \times A(w) + A + D + \alpha \quad (2.1)$$

where

M is the number of multipliers or coefficients; $A(w)$ is the average number of adders expected from graph multiplier of wordlength, w ; A is the number of structural adders; D is the number of structural delays and α is the weighting factor, equivalent to the number of adders in a delay.

In previous work done by Husinga and Darren[1-2] the cost function is defined based on specific target technology. Therefore, different technology has a different fundamental unit size and this will result in a different cost value for the different technologies. Both Husinga and Darren estimated the cost using lookup tables, which

contain a list of fundamental cost units that represent a particular size of adders and delay elements of the specific targeted technology. The following shows the general cost function used by Husinga and Darren:

$$Cost = \alpha(w)A + \beta(w)D \quad (2.2)$$

Both the $\alpha(w)$ and $\beta(w)$ are technology dependent cost units of each adder and delay of wordlength, w . The A and D are the number of adders and delays. The cost function is just an estimated number to describe the hardware complexity. Cost function will vary within different filter structures, hence cost is generalized as a function evaluation to aid the optimization algorithms.

2.3 Order- Wordlength Tradeoff

There are different ways to compute the minimum wordlength required for a digital filter. One approach is the use of the coefficients sensitivity function to compute the required wordlength for each coefficient. Debrunner [21] uses this technique for Infinite Impulse Response (IIR) filter design implementation. However, both Husinga and Daren use the order and wordlength tradeoff technique introduced by Kodek [9]. Kodek introduced the construction of a graph where the cost product of order and wordlength, $Cost(Nb)$ versus the wordlength, b , is shown in Figure 2.1. Kodek found out that at a certain point there is an optimum point that optimizes both the order and the wordlength. The optimum point is referred to as a global minimum. He also noticed that, as the wordlength increases further than the optimum wordlength, the curve becomes a constant positive slope. On the other hand, when the wordlength decreases from

optimum wordlength towards zero, the wordlength will be too small to represent the filter coefficient. Therefore, the filter specification will not be met. By observing the characteristic of the graph, which is shown in Figure 2.1, Kodek drew some conclusions and derived an equation that can compute the minimum wordlength from minimum order. This method is very useful because when the order of the filter is increased without changing the filter specifications, it will result in the decrease of wordlength needed in the filter. This tradeoff provides the ability to find the best quantized coefficients that will reduce the hardware area such as the number of CLBs for Xilinx's FPGA's. Section 4 explains the algorithm used for this method and for future reference the notation N and b refer to order and wordlength respectively.

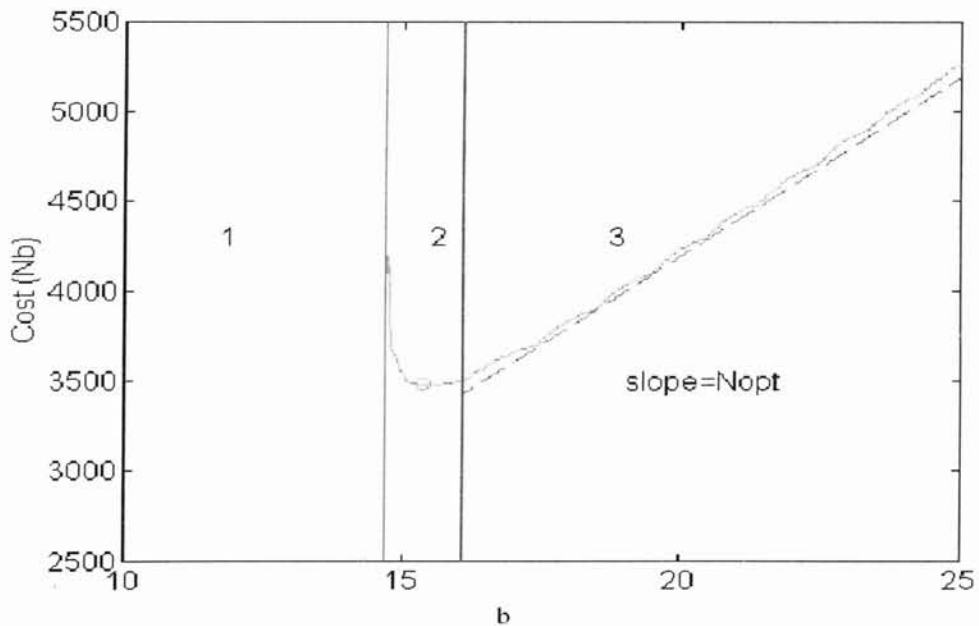


Figure 2.1: Graph of Cost (Nb) vs b

Source: D. L. Husinga, *Digital of Optimized Filter Using CSD Coefficient Representation* [1]. Master's Thesis, University of California, Davis, CA.1995.

2.4 Scaling

Though scaling has been the oldest technique, it is still the best in DSP or hardware applications. The drawback of using fixed point arithmetic for every addition or subtraction operation in each stage of the filter is that overflows may occur. DSP textbooks [17-19] use the norm and bound conditions to describe the scaling equation. While Darren's [2] approach is by scaling both the numerator coefficients and denominator coefficients (for IIR filter only) separately, and Husinga [1] scales down the inverse sum of the coefficients since the filter design is FIR filter. In previous papers, there are several techniques such as using scale factor to reduce the number of nonzero digits suggested by Serna [14] and using minimum weight representation (MWR) approach by Yagyu [22], which modifies the filter coefficients with CSD representation by using a scale factor approach. Another advantage of using scaling is to find the coefficients with minimum hardware implementation cost. This thesis will use scaling introduced by Husinga to overcome overflow and reduce the hardware complexity. Section 4.3 explains the search method to find the optimum scale factor that will result in the minimum hardware requirement.

2.5 Adder Extraction

The adder extraction technique was based on the work done by both Soderstrand and Serna [15]. In previous work, Darren, Husinga and Balasubramanian [1-3] used this method to further minimize the coefficients hardware requirement cost. The idea is to reduce the number of adders by replacing the non-zero digit with zero and

check for specifications response of the new filter to make sure that the specifications are still met; both Daren and Husinga included this technique in their programs.

2.6 Roundoff Error Analysis

Rounding and truncation after multiplication is often needed since all devices only have finite wordlength. Roundoff error is modeled as noise, while roundoff error can be avoided in FIR filters by making sure that the wordlengths are sufficient to accommodate all mathematical operations. This leads to excessive hardware. Signal to Noise ratio (SNR) is evaluated to make sure that the filter has a decent output performance. Usually, for FIR filter the SNR is relatively large due to the zero feedback, which means that it is easier to maintain the precision of the FIR filter coefficients. Hartley's [10] method of calculating SNR is used because the coefficients are represented in CSD representation. Section 6 will explain the procedure for computing SNR using Hartley's method.

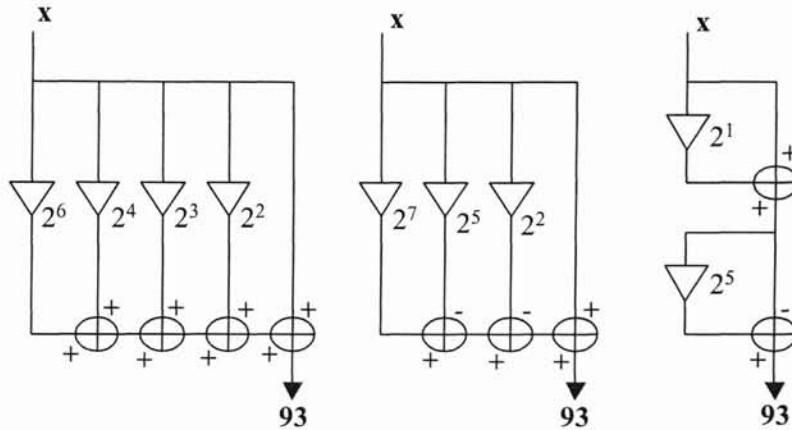
Section 3

Cost Analysis of Binary, CSD and DM Implementation

3.1 Introduction

In digital signal processing theory, the coefficient multipliers are represented in the real number system with infinite precision. However, in order to implement these coefficients in the digital hardware, a digital number system with finite length coefficients that are compatible with the digital hardware is required. Since the ideal coefficients are expressed in infinite precision, the hardware implementation of the digital processing systems may not be able to represent the coefficients very precisely or accurately. There are various types of digital number representations and usually digital hardware, such as DSP chips and digital computers, use a form of the binary number system or representation. Therefore, in this thesis, the digital number system or representation will represent the coefficient multipliers of the FIR filter. Finally, the digital number representations play an important role in hardware reduction of the filter so Section 3.3 will demonstrate the hardware cost reduction using three different digital number representations.

3.2 Number Representations



**Figure 3.1: (a) Binary Implementation of Multiply by 93
(b) CSD Implementation of Multiply by 93
(c) DM Implementation Multiply by 93**

3.2.1 Binary Representation

The binary number system is mainly used in modern digital systems and is actually the combination of the digits $\{1,0\}$. In fact binary representation is favored by digital applications because it is simple to represent the inherent on or off nature of digital hardware. Fractional numbers can also be represented by radix-2 binary number system by using a binary point to separate the integer part and the fractional part. That is how binary representation can represent the coefficient multiplications of the filter. For example, taking a simple example of a multiplier with the value 93, it can be represented as

$$93 = 2^6 + 2^4 + 2^3 + 2^2 + 2^0 \quad (3.1)$$

From this representation, the number 93 can be written as an eight bit wordlength, which is 01011101_2 . Figure 3.1(a) also shows the equivalent model of a multiplier of the value of 93 in a binary representation. Also note that there would be four elements multiplied by the power of two and four adders. Since the elements are just multiplied by the power of two, this represents shifts of number that can be accomplished by appropriate hardware wiring. Hence, multiplying by positive or negative integer powers of two does not require real hardware. In the case of Figure 3.1(a) the four adders are considered the real hardware while the multipliers are just hard wiring.

3.2.2 CSD Representation

A “signed digit” (SD) number system has the combination of $\{\bar{1}, 0, 1\}$, where $\bar{1}$ represents a negative one. The signed digit number system is useful in designing high-speed arithmetic machines. Since this is a radix-2 number system with three possible digits (ternary), there are redundancies to represent a simple decimal number, which make SD representation not unique. However, by selecting the SD representation with the fewest non-zero bits from the list of redundant representations, we can minimize the hardware required to represent a binary number. Canonical signed digit is a unique representation of signed digit numbers such that there are no adjacent nonzero digits.

CSD format has a maximum of $\frac{(N+2)}{2}$ nonzero bits, which is roughly one half that of

binary representation. Taking the same example as above, the CSD representation is

$$93 = 2^7 - 2^5 - 2^2 + 2^0 \quad (3.2)$$

The implementation of multiplying by 93 is shown in Figure 3.1(b). Comparing this to the binary implementation, CSD only requires three adders or subtractors. Based on this example, there is a 25% hardware saving over binary implementation. In addition, using signed digit for fixed-point format will allowed negative fractions to be represented in fixed-point format. The algorithm (described by Hwang [5]) that converts binary representation of a constant number to canonical signed digit is presented in Appendix A.

3.2.3 DM Representation

The Dempster and Macleod (DM) [6-7] technique is another technique to represent the multipliers in digital applications. The basic idea proposed by Dempster is a multiplier that is factorized into a set of prime numbers, where the constraint is that the coefficient must not be a prime number. To use the same example discussed previously, the factored numbers are 3 and 31, which now can be two module multipliers. Notice that the numbers are now prime numbers, therefore each factored number or module will be represented by a new number system, which is the best way is to represent the factored numbers in CSD representation. The following demonstrates that example in a mathematical form.

$$93 = 3 \times 31 = (2^1 + 2^0) \times (2^5 - 2^0) \quad (3.3)$$

Figure 3.1(c) is the equivalent model of the Equation 3.3. The figure shows that the multiplier is connected in a cascading style, so for this example, only two adders or subtractors are required. In this case, there will be a 50% and 33% hardware savings over binary and CSD implementation respectively.

3.3 Cost Analysis of Binary, CSD and DM Representation

Section 3.2 describes the three digital number representations with an example multiplier that results in the reduction of hardware complexity. As the wordlength represented by the multipliers becomes smaller, the chances of hardware savings by using Dempster and Macleod (DM) technique over CSD technique are reduced. Table 3.1 shows the average number of adders needed for all combinations with the positive dynamic range of four bits for each of the digital number representations mentioned in Section 3.2. In other words, for a four bit wordlength, there are 0 to 2^4-1 combinations, which means there are combinations that range from 0 to 15. The phrase *number of combinations* represents the number of multipliers. These multipliers are also used to calculate the average number of adders by taking the sum of the adders for all the combinations and dividing by the total number of combinations. For Table 3.1, total number of combinations is 16.

Adder	Number Of Combinations			Total Adders		
	Binary	CSD	DM	Binary	CSD	DM
0	5	5	5	0	0	0
1	6	9	9	6	9	9
2	4	2	2	8	4	4
3	1	0	0	3	0	0
Total	16	16	16	17	13	13
Average	1	1	1	1.0625	0.8125	0.8125

Table 3.1 Average Number of Adders for 4-bit Number of Combinations

By examining Table 3.1, the *ADDER* column means the number of adders. The *Number of combinations* column gives the number of combinations while it is related to a specific number of adders by referring to the *ADDER* column. The *Total ADDERS* column displays the total number of adders of the required number of combinations. Observing the highlighted row of the table on the previous page, it can be seen that under the *Number of combinations* heading, there is only one combination for binary representation and zero combinations for both CSD and DM representations. Therefore, the total adders required are three for binary representation and zero for both CSD and DM representation. The average number of adders produced by CSD and DM representation is 0.8125. Nevertheless, both CSD and DM techniques show a 23.53% savings of adders required over binary representation. For large numbers, which are determined by their available wordlength, if they are represented in CSD representation, an additional bit is required. For example, the number '14' in a four bit word length is 1110_2 in binary representation, but in CSD representation, it is represented as $100\bar{1}0_2$.

The earlier case shows the average numbers for DM and CSD is the same due to the fact that the wordlength is small. However with larger wordlength, the chances of hardware savings using DM technique over CSD technique are much higher. For discussion, observe Table 3.2 with eight bit dynamic range, which means that the numbers range from 0 to 2^8-1 . The average numbers of adders for Binary, CSD and DM representation are 3.0038, 2.1133 and 2.0078 respectively. The percentage savings for CSD and DM over Binary representations are 29.65% and 33.16% respectively.

Adders	Number Of Combinations			Total Adders		
	Binary	CSD	DM	Binary	CSD	DM
0	9	9	9	0	0	0
1	28	49	49	28	49	49
2	56	110	133	112	220	266
3	70	80	61	210	240	183
4	56	8	4	224	32	16
5	28	0	0	140	0	0
6	8	0	0	48	0	0
7	1	0	0	7	0	0
Total	256	256	256	769	541	514
Average	1	1	1	3.0039	2.1133	2.0078

Table 3.2 Average Number of Adders for 8-bit Number of Combinations

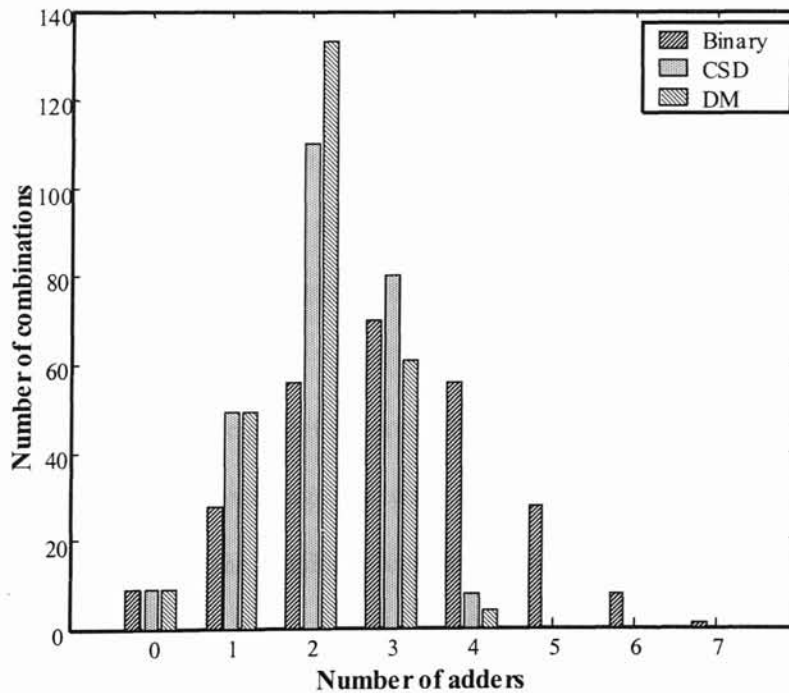


Figure 3.2: Histogram of Number of adders Vs Number of Combinations for 8-bit wordlength

The histogram presented in Figure 3.2 is based on the data from Table 3.2. The vertical axis represents the number of combinations that are required by the number of adders, while the horizontal axis represents the number of adders. From this histogram, notice that for the DM technique there are no combinations available starting from five adders onwards which is the same for CSD representation. When the number of adders is two, both CSD and DM technique achieve the highest number of combinations.

Another result is presented in Table 3.3 and the result is the average adder number for all the positive combinations of sixteen bit wordlength. The average number of adders for each combination is 7, 4.7778 and 4.3261 for binary, CSD and DM representations respectively. Hence, in this case, the percentage savings for CSD representation over binary representation is much higher, which is 31.45%. Similarly, the percentage savings for DM technique over Binary representation is 38.2%. The results show a higher savings for sixteen bit wordlength due to the number of combinations is significantly large compared to the case for eight bit wordlength. If taking the consideration of comparing DM technique over CSD representation, the percentage savings for eight bit and sixteen bit wordlength are 4.98% and 9.45% respectively. The savings is increased by approximately 5% between eight bit to sixteen bit wordlength. Relatively speaking, as the wordlength increases the percentage hardware savings will also increase. The histogram presented in Figure 3.3 is the graphical representation of the results from Table 3.3. By observing the histogram in Figure 3.3, the number of adders at the highest number of combinations for CSD representation is five while for DM technique, the number of adders where the highest number of combination is achieved is four. Also notice that for

both the CSD and DM representations, there are no more combinations left starting from nine adders onwards.

Adders	Number Of Combinations			Total Adders		
	Binary	CSD	DM	Binary	CSD	DM
0	17	17	17	0	0	0
1	120	225	225	120	225	225
2	560	1638	2133	1120	3276	4266
3	1820	6864	10140	5460	20592	30420
4	4368	16632	24631	17472	66528	98524
5	8008	22176	21072	40040	110880	105360
6	11440	14400	6522	68640	86400	39132
7	12870	3456	778	90090	24192	5446
8	11440	128	18	91520	1024	144
9	8008	0	0	72072	0	0
10	4368	0	0	43680	0	0
11	1820	0	0	20020	0	0
12	560	0	0	6720	0	0
13	120	0	0	1560	0	0
14	16	0	0	224	0	0
15	1	0	0	15	0	0
Total	65536	65536	65536	458753	313117	283517
Average	1	1	1	7	4.7778	4.3261

Table 3.3 Average Number of Adders for 16-bit Number of Combinations

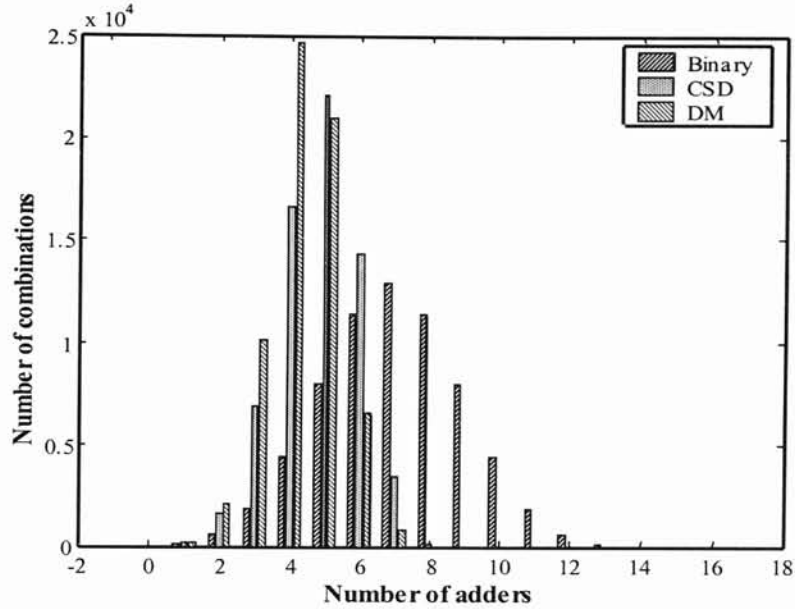


Figure 3.3: Histogram of Number of adders Vs Number of Combinations for 16-bit Wordlength

Table 3.4 shows the average adders for each combination for three different number representations from four bit to twenty bit wordlength. Basically, Table 3.4 is obtained by finding the average adder number for each combination for every wordlength, which ranges from four bits to twenty bits. Figure 3.4 is plotted based upon the data presented in Table 3.4. The plot illustrates the three different curves that explain the adders savings using the three different techniques described by the legend. Overall, the performances of DM and CSD representations are better than Binary representation. By comparing DM and CSD representation, DM representation shows a promising result in hardware reduction as the wordlength increases, but note that at wordlength from 4 bits to approximately 5.5 bits both CSD and DM implementation produce the same average

number of adders. By observing Figure 3.4, the slope computed for the binary, CSD and DM technique are 0.498, 0.333 and 0.29 respectively. The slope for binary representation is steeper than that for DM and CSD representation. Also note that the slope for CSD representation is steeper than that for DM representation. This pattern shows a distinct result that more adders are saved as the wordlength increases for both CSD and DM representation. By taking Figure 3.4 as a reference to obtain the average number of adder information, $A(w)$, the cost function derived by Demspster shown in Section 2 can be put into full use.

Wordlength	Average Adder Number for each Combination		
	Binary	CSD	DM
4	1.0625	0.8125	0.8125
5	1.5313	1.125	0.125
6	2.0156	1.4531	1.4188
7	2.5078	1.7813	1.7109
8	3.0039	2.1133	2.0078
9	3.5020	2.4453	2.3008
10	4.0010	2.7783	2.5986
11	4.5005	3.1113	2.8911
12	5.0002	3.4446	3.1785
13	5.5001	3.7778	3.4644
14	6.0000	4.1111	3.7525
15	6.5000	4.4445	4.0391
16	7.0000	4.7778	4.3261
17	7.5000	5.1111	4.6128
18	8.0000	5.4444	4.8999
19	8.5000	5.7778	5.1873
20	9.0000	6.1111	5.4751

Table 3.4: Average Adder Number for each Implementation

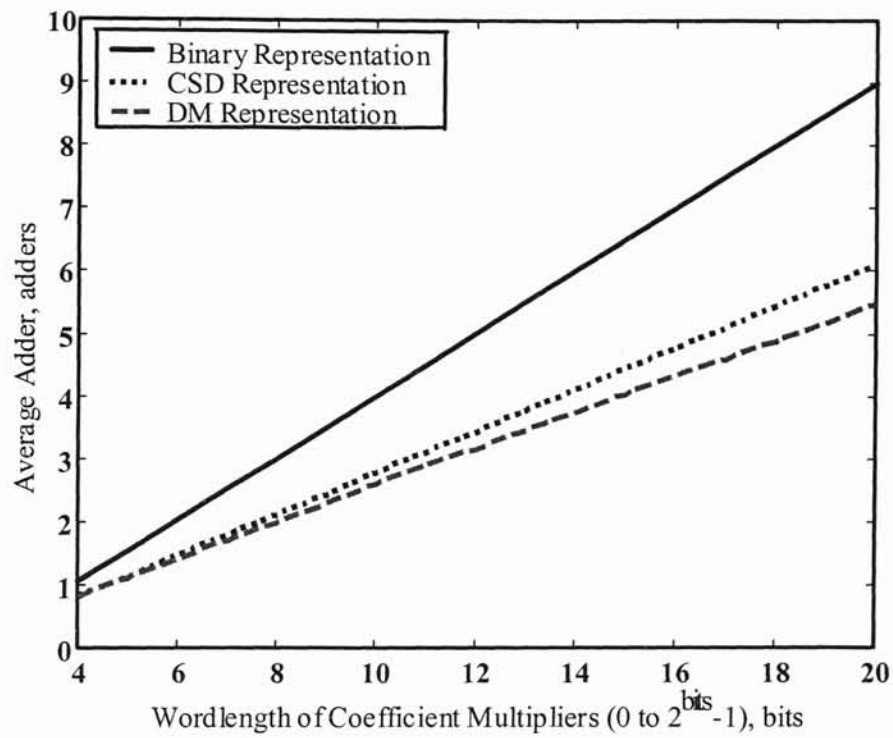


Figure 3.4: Plot of Average Adder for Coefficient Multipliers with Wordlength Range from 4-bits to 20-bits

Section 4

CSD Optimization Techniques

4.1 Introduction

In previous work, Husinga [1] has demonstrated her ingenuity by incorporating the three optimization techniques to find a set of optimum coefficients represented in CSD representation. It is worthwhile to mention and revise all these techniques. Recall that the three optimization techniques are order-wordlength tradeoff, scaling and adder extraction. The purpose of optimization is to eliminate the adders' requirement of the filter coefficients. The restriction for each optimization process is that the specification of the FIR filter must be maintained. The following subsections describe the three optimization techniques and the results analysis which results in a reduction of adders.

4.2 Order- Wordlength Tradeoff

Matlab program is used to determine the coefficients and the order of a FIR filter with required filter specifications. The coefficients are in infinite precision so in order to determine the optimum wordlength that represents the coefficients, the optimum order filter is computed. This optimum wordlength and order will be the guideline for the next

optimization technique, and it is the primary step of the three optimization process. This optimization technique is the implementation of Kodek's [9] idea discussed in Section 2. The idea is to determine an optimum filter order that meets the specifications with optimum wordlength to represent the filter coefficients. This will result in the minimized coefficients wordlength and consequently reduce hardware requirement.

4.2.1 Procedure for Determining Optimum Order and Wordlength

Remezord and *Remez* function available in Matlab will determine the overall order and coefficients for any type of FIR filter. First of all *remezord* function will produce an equivalent filter order and a set of coefficients that meets the filter specifications. By taking these parameters as a guideline, *remez* function is used again to determine the optimum filter order. This is done by reducing the overall order one at a time and for each order we apply to the *remez* function again to generate a new set of coefficients. These coefficients are feed to the *freqz* function to obtain the frequency response data. This data will be evaluated to check if the filter specifications are met. This process will repeat itself until an optimum order is found. The objective function that describes the task of obtaining the optimum order is shown below:

$$N_{opt} = \arg \min_{order} \{remez \text{ function that meet filter specifications}\} \quad (4.1)$$

where N_{opt} is the optimum order.

Since there are four types of filters, which are lowpass filter, highpass filter, bandpass filter and bandreject filter, the specifications are classified as the constraints for all the

objective functions. The notation used to represent the filter specifications for all filter types is presented as follow:

- F_{samp} (f_s) – Sampling Frequency (Hz)
- F_{p1} and F_{p2} - Passband Frequency (Hz)
- F_{s1} and F_{s2} – Stopband Frequency (Hz)
- R_p – Passband Ripple (dB)
- R_s – Stopband Ripple (dB)

These frequencies and ripple parameters give the characteristic of the frequency response of the filter. To determine if the filter design meets the specifications constraint, the specific magnitude frequency response, $H(e^{j\omega})$ is evaluated. There are two inequality constraints and they are determined based on the criteria and the filter type prompt by the user. The following are the two general inequality constraints.

$$\max[20\log_{10}|H(e^{j\omega})|] - \min[20\log_{10}|H(e^{j\omega})|] \leq R_p \quad (4.2)$$

$$\max 20\log_{10}|H(e^{j\omega})| \leq R_s \quad (4.3)$$

Filter Type	Frequency Range for Passband Ripple, R_p	Frequency Range for Stopband Ripple, R_p
Lowpass	$0 \leq \omega \leq \omega_p$	$\omega_s \leq \omega \leq \omega_N$
Highpass	$\omega_s \leq \omega \leq \omega_N$	$0 \leq \omega \leq \omega_p$
Bandpass	$\omega_{p1} \leq \omega \leq \omega_{p2}$	$0 \leq \omega \leq \omega_{s1}, \omega_{s2} \leq \omega \leq \omega_N$
Band reject	$0 \leq \omega \leq \omega_{p1}, \omega_{p2} \leq \omega \leq \omega_N$	$\omega_{s1} \leq \omega \leq \omega_{s2}$

Table 4.1: Frequencies Range For Passband Ripple and Stopband Ripple of All Filter Types.

Figure 4.1 presents the frequencies range corresponds to the two inequalities constrains shown in Equation 4.2 and Equation 4.3. The purpose of inequality constraint is to meet the passband and stopband ripples criteria at the right frequency ranges. As long as the inequality constraints are met, the original order is decreased by one or vice versa. This procedure is repeated until the optimum order, N_{opt} is found.

The next step is to determine the optimum coefficients wordlength, b_{opt} using the equation defined by Kodek [9], and the following equation proposed is:

$$\delta_r(N, b) \cong 2^{-(wordlength-1)} \sqrt{\frac{(2 \times order) - 1}{3}} \quad (4.4)$$

where $\delta_r = \delta_T - \delta_A$

Note that δ_T is the total deviation, which is the stopband deviation set by the user; δ_A is the approximation error of stopband deviation due to infinite precision coefficients; and δ_r is the round off error. Once the wordlength is determined using Equation 4.4, the filter specifications are checked. If the filter specifications are not met, the wordlength will increase by one, and this process is repeated until an optimum wordlength is found. Figure 4.1- 4.3 display flowcharts that explain this optimization algorithm. Figure 4.1 describes the algorithm for finding the optimum order; N_{opt} while Figure 4.2 presents the block diagram for obtaining the optimum wordlength, b_{opt} . In b_{opt} algorithm there is a small subroutine named *newcsd_fir*, which is presented in Figure 4.3. This subroutine is to check the filter specification status.

4.2.2 Function Introduction

The function to determine the optimum order, N_{opt} is get_N_{opt} . The function is shown as follows:

$$[N_{opt}, Fo, Mo, B_{opt}, csdRs] = get_N_{opt} \quad (4.5)$$

Where the function returns the following parameters:

- N_{opt} – Infinite precision optimum order
- Fo – The cutoff frequencies vector for *remez* function
- Mo – Magnitude description of frequency response vector for *remez* function
- B_{opt} – Optimum set of coefficients (infinite precision)
- $csdRs$ – Stopband Ripple Peak

The optimum wordlength function is get_b_{opt} and it is shown as follows:

$$[b_{opt}] = get_b_{opt}(B_{opt}, N_{opt}, csdRs) \quad (4.6)$$

Where the B_{opt} , N_{opt} and $csdRs$ are the input of the function and return the optimum wordlength value, b_{opt} . In get_b_{opt} function, there is another function called *newcsd_fir* that checks for filter specifications and returns the optimum wordlength, b_{opt} , that meets the specifications. The following shows the *newcsd_fir* function:

$$[torf] = newcsd_fir(B, N, b) \quad (4.7)$$

where the inputs are infinite precision coefficient, B ; optimum order, N and wordlength, b . The function returns the status value, $torf$, where if the filter specifications are met, the status value is equal to one.

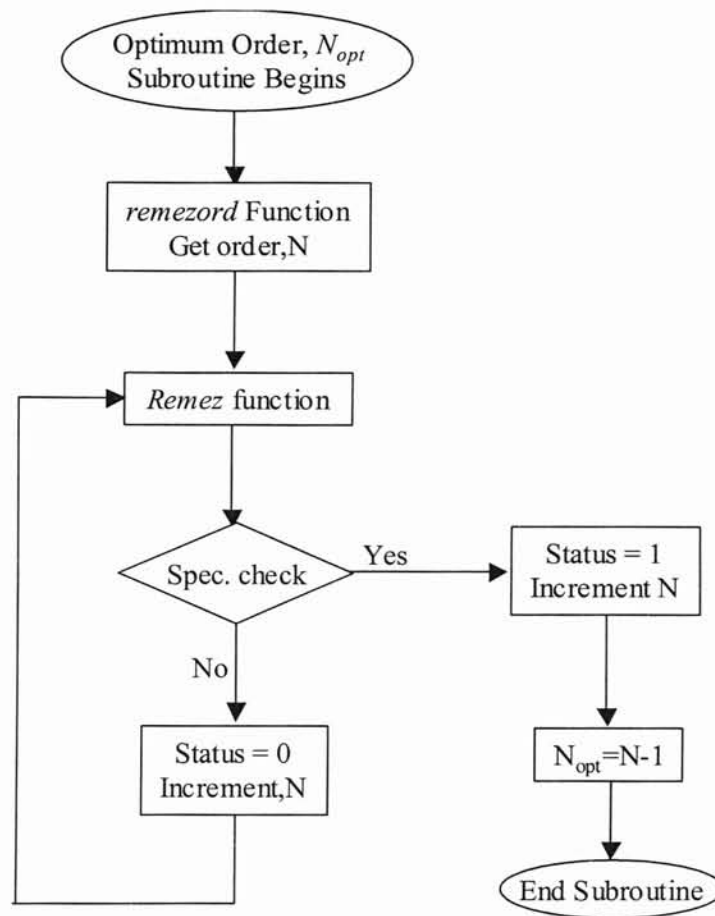


Figure 4.1: Optimum Order, N_{opt} Algorithm Block Diagram

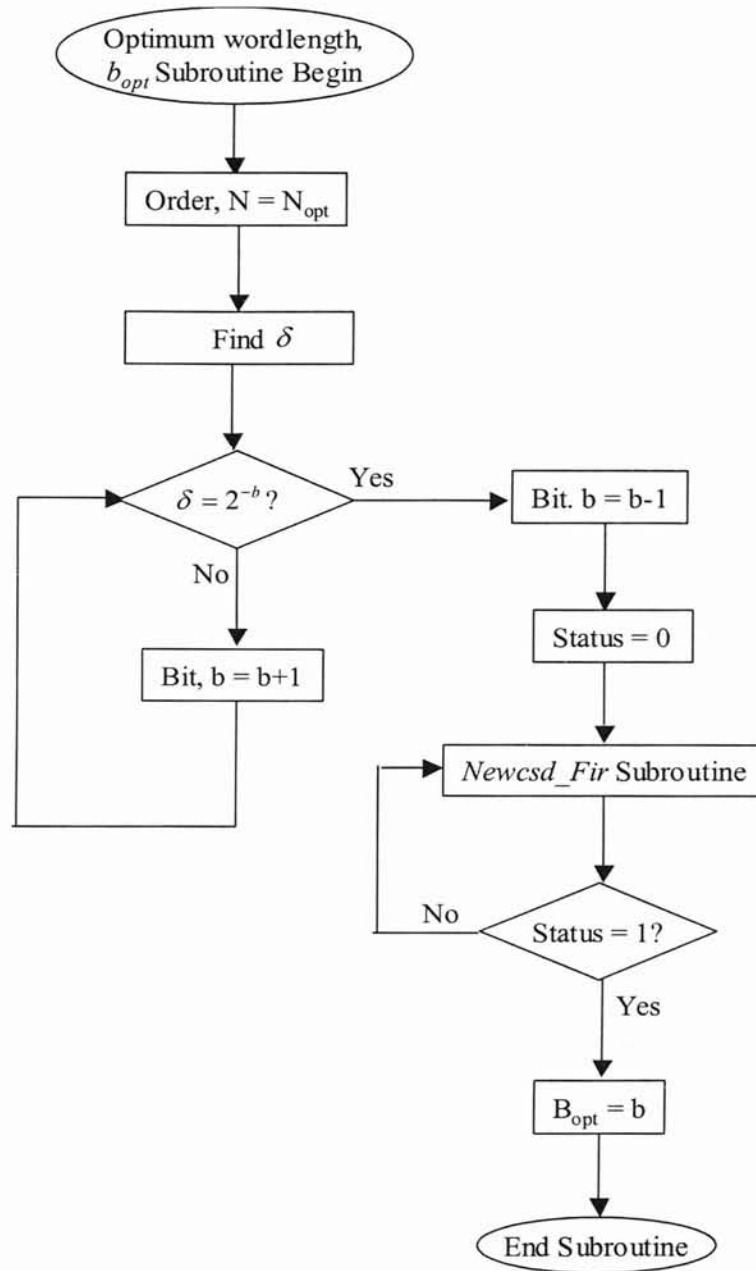


Figure 4.2: Optimum wordlength, B_{opt} Algorithm Block Diagram

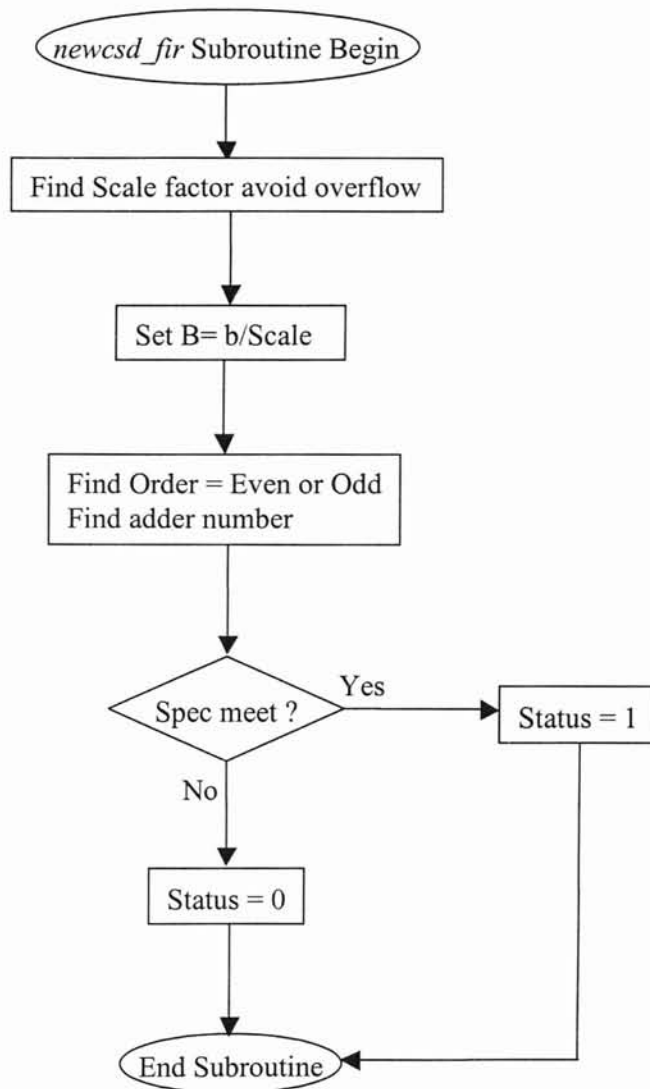


Figure 4.3: *newcsd_fir* Subroutine Block Diagram

4.3 Scaling

4.3.1 Theory

Because we are implementing the digital filter in fixed-point arithmetic, overflow may occur at certain internal nodes. Scaling is normally used to minimize the probability of overflow occurring in the filter. To pick a scale factor, first of all the maximum gain of the transfer function, K_{\max} must be determined. The maximum scaling factor is the inverse of the maximum gain that is $1/K_{\max}$. In our case, however, we will use the scaling factor to reduce the number of adders in the coefficients. This is done by searching possible scale factors between the ranges of the minimum gain factor, which is $1/2K_{\max}$ to the maximum gain factor, $1/K_{\max}$. The optimum scale factor will result in the best set CSD coefficients which in turn results in more hardware savings. Also notice that the minimum scaled factor is scaled by half, which is the power of two and is implemented simply by shift operation. Scaling down the coefficients will result in a slight decrease in SNR and precision.

4.3.2 Procedure For Determining Minimum Order and Wordlength

To determine both the minimum order and wordlength, cost function is incorporated in the program where the cost function is technology dependent. First the search region is defined for both order and wordlength, hence they are noted as

$\mathbf{N} = \{N_{opt} \leq N \leq 8 + N_{opt}\}$ and $\mathbf{B} = \{b_{opt} * 0.5 \leq b \leq b_{opt}\}$ respectively. The search regions of both order and wordlength are chosen to be wide enough to search for the minimum order and wordlength that provide the least hardware requirement for the specific FIR filter design. Each element from set \mathbf{N} is evaluated with all the elements from set \mathbf{B} . A new search region with different scale factors is determined by different sets of coefficients and the step size of each scale factor is incremented by 0.1. Each scale factor is computed using the formula described below:

$$\text{Scale factor, } S = \frac{K}{(1 + 0.1I)} \quad I = 0, 1, \dots, 9 \quad (4.8)$$

Equation 4.8 shows the region of the scaling factor as ranging from $\frac{1}{1.9}$ to 1. It now forms a set, S , where the elements are the scale factors varying from $\frac{1}{1.9}$ to 1. Every coefficient within the search region will multiply by a single scale factor obtained from the set, S , to generate a new set of coefficients. The new set of coefficients is evaluated where the filter's frequency response must meet the filter specification constraints. If it meets the filter specifications, then the particular wordlength, order and scale factor are recorded. So, for every three sets of costs, scale factors and wordlength that meet the filter specifications will be stored in a new matrix. From this matrix, the least cost and scale factor are preserved while the rest of the unimportant data will be eliminated. This procedure will repeat until the search region meets the last order and wordlength of the search regions. As the procedure ends, the most minimized order, N_{min} and wordlength, b_{min} will be determined from a list of corresponding minimum cost table and the minimum scale factor. The flowcharts presented in Figure 4.4 and Figure 4.5 give a

general overview of the algorithm described. Figure 4.4 illustrates the role of the scaling algorithm in the main program and Figure 4.5 illustrates the procedure steps of the scaling algorithm.

4.3.2 Function Introduction

The function of optimizing the coefficients using the scaling technique is as follows:

$$[addcnt, addcef, spcsmet, scaleopt] = newcsd_fisc(IC, Nvec(i), bvec(k)) \quad (4.9)$$

Where the inputs are

- *IC* – Infinite precision coefficients
- *Nvec* – Filter order obtain from the search region
- *Bvec* – Coefficient wordlength from the search region

And the outputs are

- *addcnt* – Total number of structural adders
- *addcef* – Total number of adders of the coefficients
- *spcsmet* – Status of meeting the specifications
- *scaleopt* – The optimum scale factor from the search region

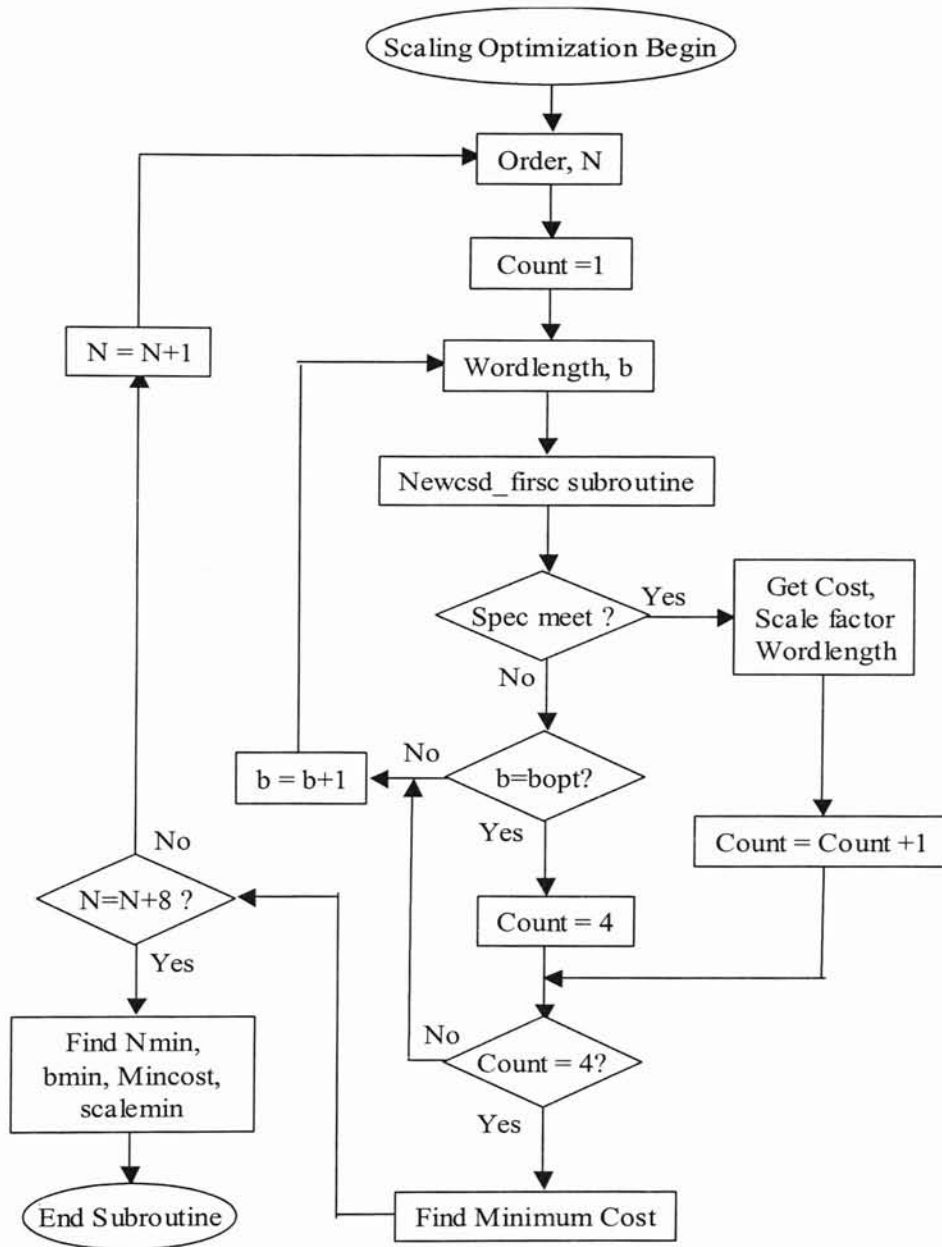


Figure 4.4: Scaling Algorithm Block Diagram

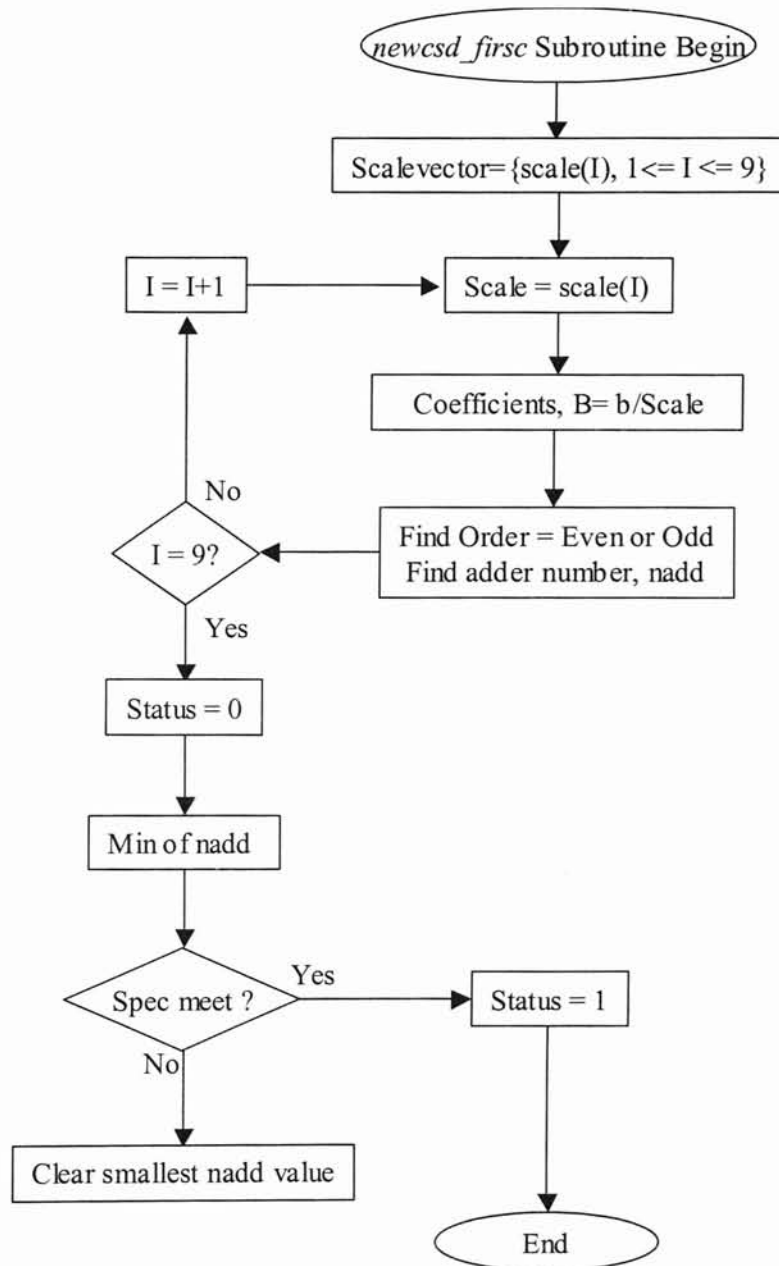


Figure 4.5: *newcsd_firsc* Subroutine Algorithm Block Diagram

4.4 Adder Extraction

The adder extraction technique removes non-zero bits from the coefficient table in binary numbers and replaces them with zero digits. Thus, a new set of coefficients will be generated and the frequency response is checked to make sure the filter specifications are met. If the specifications are met, then the coefficient table is updated. Hence a new minimum cost is computed. This technique will usually work if the coefficients are represented by a large wordlength such as 16 bits and 20 bits with many nonzero digits. The process usually takes place on the least significant bit from strings of binary numbers.

4.4.1 Procedure of Adder Extraction

Once the new set of coefficients is determined from the scaling optimization, these coefficients are implemented using the adder extraction process, which is the last optimization. This algorithm uses exhaustive search because each binary number obtained from each coefficient is evaluated for each iteration. Every time this process is done the filter specification constraints are checked to make sure that the filter specification meets the requirement. If the filter specifications are met, each coefficient is updated and this process continues for every coefficient for all the non-zero bits. Figure 4.7 shows a detailed explanation of adder extraction algorithm. Note that since it is a linear phase FIR filter, the coefficients are in symmetry so there is a very high chance that two coefficients will be affected. Once all the coefficients have been evaluated, the adder extraction subroutine will produce sufficient output to the main program for further

evaluation. Figure 4.6 presents the role of adder extraction process in the main program. If the optimization of the coefficients has occurred, a status describing the outcome of the adder extraction will occur. If there are any nonzero bits that are extracted, the status will return a true statement. At this point, a new minimized cost is recalculated and the new coefficients replace the old coefficients. If no coefficients are minimized from adder extraction subroutine, then all the results will be sustained.

4.4.2 Function Introduction

The *redcadd* function operates the adder extraction optimization and returns the updated result.

$$[rmvd, Hrmvd, newtable, newaddcefmin, newRpl] = (scalemin, table, Nmin, bmin, addcefmin) \quad (4.10)$$

Where the inputs are

- *scalemin* – Optimized scale factor
- *table* – A set of optimized coefficients in CSD representation
- *Nmin* – Minimum order
- *bmin* – Minimum wordlength
- *addcefmin* – Total number of adders of the coefficients

Where the outputs are

- *rmvd* – Adder extraction status
- *Hrmvd* – Complex frequency response after adder extraction process
- *newtable* – A set of optimized coefficients in CSD representation

- *newaddcefmin* – Total number of adders of the coefficients
- *newRpl* – New set of ripples information in dB.

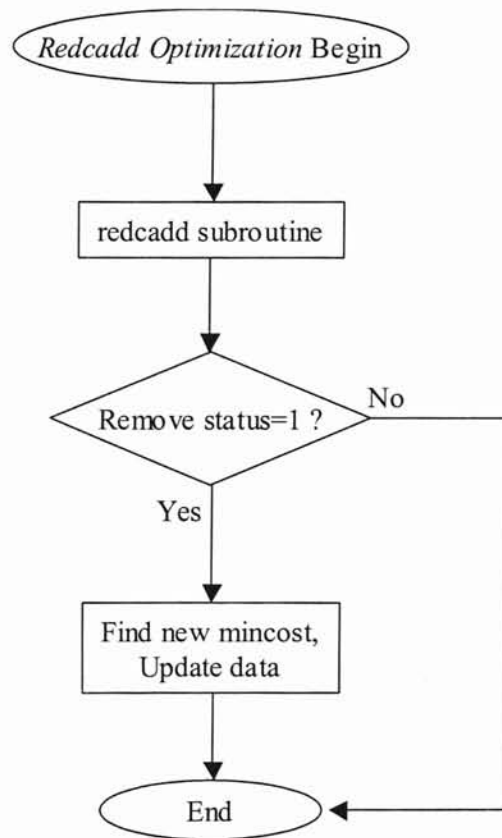


Figure 4.6: Adder Extraction Optimization General Routine Block Diagram

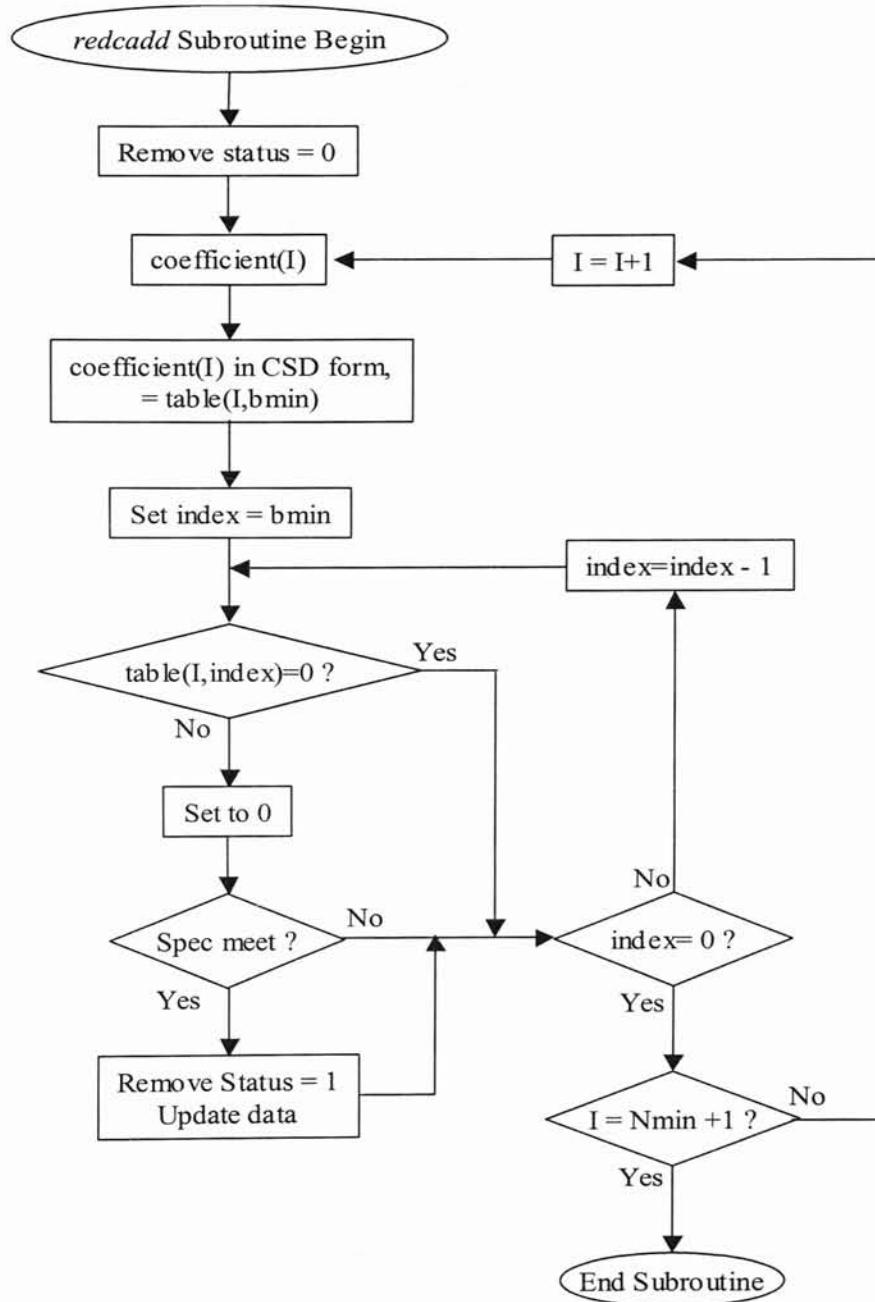


Figure 4.7: Adder Extraction Algorithm Block Diagram

4.5 Example Result of Three Optimization Techniques

The following tables show the example cost results of the four types of filters. Each filter has its own fixed filter specifications except the passband ripple, R_p is varied ranging from 5.5dB to 0.1dB. The sampling frequency is fixed on 24KHz and the input wordlength is set at 8 bit. The filter type and its filter specification profile are listed on the heading of each table. Referring to Tables 4.2 to 4.5, the cost results of each filter are recorded under three main headings, which are *Order-wordlength tradeoff*, *Scaling* and *Adder Extraction*. Additionally, the heading *% Reduction (1)* means the percentage cost savings computed from the difference between the first technique, *Order-wordlength tradeoff* over the second technique, *Scaling*. Similarly, the heading *% Reduction (2)* displays the percentage cost savings for the second technique, *Scaling* and the third technique, *adder extraction*. Also note that the cost is computed based on the cost equation, which will be discussed in Section 7.3. From the observation, the percentage savings of the second technique over the first technique range from 3% to 45%. On the other hand, the percentage reduction obtained from the second and third techniques is extremely low. The reduction range is 0 to 1.8 percent only. The reason is the large saving occurs during the scaling optimization. Therefore, if there is no saving shown, this means that the hardware representation of the coefficients is truly minimized. In short, adder extraction optimization's role is to make sure that the hardware required for the coefficients is minimized. Overall, the average hardware savings for scaling technique over order-wordlength tradeoff is about 20% and the average hardware savings for adder extraction technique over scaling technique is approximately 0.3%.

Lowpass Filter: Frequency = [5000 6000]Hz Stopband Ripple, Rs = 40dB					
Rp	Order- Wordlength Tradeoff	Scaling	%Reduction (1)	Adder Extraction	%Reduction (2)
5.5	1872	1572	16.03	1560	0.76
5	1956	1614	17.48	1614	0.00
4.5	2093	1685	19.50	1685	0.00
4	2022	1806	10.68	1794	0.66
3.5	2494	1880	24.62	1869	0.59
3	2144	1906	11.10	1894	0.63
2.5	2431	2078	14.52	2078	0.00
2	2878	2068	28.14	2068	0.00
1.5	2860	2304	19.44	23.04	0.00
1.0	3402	2548	25.10	2548	0.00
0.5	5112	2798	45.27	2798	0.00
0.2	4603	3532	23.27	3493	1.10
0.1	5488	3826	30.28	3826	0.00
Average (1)			21.96	Average (2)	0.29

Table 4.2: Percentage Cost reduction for Lowpass Filter Example

Highpass Filter: Frequency = [5000 6000]Hz Stopband Ripple, Rs = 40dB					
Rp	Order- Wordlength Tradeoff	Scaling	%Reduction (1)	Adder Extraction	%Reduction (2)
5.5	1728	1567	9.32	1567	0.00
5	1900	1762	7.26	1762	0.00
4.5	1888	1726	8.58	1726	0.00
4	2145	1795	16.32	1795	0.00
3.5	2156	1700	21.15	1700	0.00
3	27.16	1773	34.72	1773	0.00
2.5	2698	1992	26.17	1992	0.00
2	2918	2105	27.86	2094	0.52
1.5	2795	2088	25.30	2088	0.00
1.0	3591	2536	29.38	2488	1.89
0.5	3172	2552	19.55	2552	0.00
0.2	4394	3332	24.15	3320	0.36
0.1	4833	3696	23.53	3696	0.00
Average (1)			21.02	Average (2)	0.21

Table 4.3: Percentage Cost reduction for Highpass Filter Example

Bandpass Filter: Frequency = [3000 4000 5000 6000]Hz Stopband Ripple, Rs = 40dB					
Rp	Order- Wordlength Tradeoff	Scaling	%Reduction (1)	Adder Extraction	%Reduction (2)
5.5	2236	1830	18.16	1818	0.66
5	2184	1906	12.73	1884	1.15
4.5	2210	1878	15.02	1878	0.00
4	2210	1987	10.09	1987	0.00
3.5	2550	2016	20.94	2016	0.00
3	2682	2158	19.54	2132	1.20
2.5	2994	2560	14.50	2548	0.47
2	3583	2726	23.92	2690	1.32
1.5	4266	2750	35.54	2726	0.87
1.0	3640	2868	21.21	2868	0.00
0.5	2990	2894	3.21	2882	0.41
0.2	6648	3600	45.85	3588	0.33
0.1	5033	3664	27.20	3652	0.33
Average (1)			20.61	Average (2)	0.52

Table 4.4: Percentage Cost reduction for Bandpass Filter Example

Band Reject Filter: Frequency = [3000 4000 5000 6000]Hz Stopband Ripple, Rs = 40dB					
Rp	Order- Wordlength Tradeoff	Scaling	%Reduction (1)	Adder Extraction	%Reduction (2)
5.5	1752	1344	23.29	1344	0.00
5	1699	1412	16.89	1412	0.00
4.5	1876	1421	24.25	1421	0.00
4	1699	1385	18.48	1385	0.00
3.5	2000	1385	30.75	1385	0.00
3	2012	1385	31.16	1385	0.00
2.5	2136	1630	23.70	1630	0.00
2	2368	1884	20.44	1884	0.00
1.5	2215	2105	4.97	2083	1.05
1.0	2925	2223	24	2212	0.50
0.5	3406	2873	15.65	2847	0.91
0.2	4200	3592	14.48	3566	0.72
0.1	4522	4288	5.17	4223	1.52
Average (1)			19.48	Average (2)	0.36

Table 4.5: Percentage Cost reduction for Band Reject Filter Example

Section 5

Dempster and Macleod Implementation

5.1 DM Optimization Introduction

The DM approach is based upon saving adders in coefficient multiplier implementation by factoring the coefficient into numbers that require fewer adders to implement the cascading of the factored coefficient multipliers. The initial step is to convert the coefficients into decimal format and then factor the decimal formed coefficients using only the prime numbers. The hope is that factoring the coefficient numbers with prime numbers will be a better choice compared to choosing a vast range of numbers for factorization. For some cases, factoring the coefficients with prime numbers may not result in a savings in hardware. Thus, other factoring numbers may need to be considered, and will result in many possible ways of factoring the coefficients. Many combinations that represent the coefficients are produced and usually many combinations are redundant. For a simple example, consider the coefficient with multiplier 297, the computation of prime factors is as below:

$$297 = 3 \times 3 \times 3 \times 11 = (2^2 - 1) \times (2^2 - 1) \times (2^2 - 1) \times (2^4 - 2^2 - 1) \quad (5.1)$$

Notice that factors 3 and 11 are prime numbers. Equation 5.1 shows that the multiplier 297 needs 5 adders. But there is another alternative result, shown here:

$$297 = 9 \times 33 = (2^3 + 1) \times (2^5 + 1) \quad (5.2)$$

By just taking the factors of 9 and 11, now only two adders are needed. This result requires fewer numbers of adders compared to Equation 5.1. Notice that for this example the computation result has fewer adder requirements compared to CSD representation, which require the total of three adders. As the value of the multiplier increases, the number of possible combinations that represent the multiplier value increases. The only exception is where no combinations are generated when the multiplier itself is a prime number and then there is no need to perform DM optimization.

5.2 DM Optimization Procedure

Initially, all the coefficients are converted into decimal numbers based on the minimum wordlength obtained earlier. These multipliers will be evaluated one by one to determine the number of adders needed. First of all, each multiplier is tested to see if they are the result of the power of two or a prime number itself. If either of the conditions is true, then the number of adders will not be evaluated but will immediately state the number of adders based on the number of adders obtain from CSD representation. If the conditions are false, then all the possible combinations of the factored coefficients will be stored in a matrix and each combination is evaluated by counting the number of adders available. Note that if any of the factors is equal to two, it is ignored because no adders are required for this case. Another problem occurs is when a single number is a power of

Notice that factors 3 and 11 are prime numbers. Equation 5.1 shows that the multiplier 297 needs 5 adders. But there is another alternative result, shown here:

$$297 = 9 \times 33 = (2^3 + 1) \times (2^5 + 1) \quad (5.2)$$

By just taking the factors of 9 and 11, now only two adders are needed. This result requires fewer numbers of adders compared to Equation 5.1. Notice that for this example the computation result has fewer adder requirements compared to CSD representation, which require the total of three adders. As the value of the multiplier increases, the number of possible combinations that represent the multiplier value increases. The only exception is where no combinations are generated when the multiplier itself is a prime number and then there is no need to perform DM optimization.

5.2 DM Optimization Procedure

Initially, all the coefficients are converted into decimal numbers based on the minimum wordlength obtained earlier. These multipliers will be evaluated one by one to determine the number of adders needed. First of all, each multiplier is tested to see if they are the result of the power of two or a prime number itself. If either of the conditions is true, then the number of adders will not be evaluated but will immediately state the number of adders based on the number of adders obtain from CSD representation. If the conditions are false, then all the possible combinations of the factored coefficients will be stored in a matrix and each combination is evaluated by counting the number of adders available. Note that if any of the factors is equal to two, it is ignored because no adders are required for this case. Another problem occurs is when a single number is a power of

two and has a negative value. One adder is required for this case to maintain the negativity of the coefficients in the hardware perspective. All the number of adders gathered by evaluating all the combinations for a single multiplier and the calculated values are stored in a matrix. Then from this matrix, the number of minimum adders is sorted out and the results will be preserved. In addition, the best combination that gives the least number of adders is also preserved as well for future use. Figure 5.1 is the flowchart that explains the DM optimization algorithm in graphical view.

5.3 Function Introduction

In the program the function representing the DM optimization algorithm is presented as below:

$$[dm, dmnumopt] = dmnum(csdnum, newtable, bmin, ac) \quad (5.3)$$

From the function above, *csdnum* is the decimal number representation of the optimized coefficients and *newtable* is the CSD representation of the coefficients. Both are useful to find the factorized coefficients and identify the power of two including the sign evaluation. *bmin* is the minimum wordlength of the coefficients and *ac* is a list of number of adders for all coefficients computed from CSD representation. For the outputs, *dm* is a list with the minimum number of adders of all coefficients and *dmnumopt* stores the best combinations that provide the least number of adders needed. Another function that is included in *dmnum* function is to find all the possible combinations for every multiplier. This operation is done by heuristically defining all the possible combinations.

two and has a negative value. One adder is required for this case to maintain the negativity of the coefficients in the hardware perspective. All the number of adders gathered by evaluating all the combinations for a single multiplier and the calculated values are stored in a matrix. Then from this matrix, the number of minimum adders is sorted out and the results will be preserved. In addition, the best combination that gives the least number of adders is also preserved as well for future use. Figure 5.1 is the flowchart that explains the DM optimization algorithm in graphical view.

5.3 Function Introduction

In the program the function representing the DM optimization algorithm is presented as below:

$$[dm, dmnumopt] = dmnum(csdnum, newtable, bmin, ac) \quad (5.3)$$

From the function above, *csdnum* is the decimal number representation of the optimized coefficients and *newtable* is the CSD representation of the coefficients. Both are useful to find the factorized coefficients and identify the power of two including the sign evaluation. *bmin* is the minimum wordlength of the coefficients and *ac* is a list of number of adders for all coefficients computed from CSD representation. For the outputs, *dm* is a list with the minimum number of adders of all coefficients and *dmnumopt* stores the best combinations that provide the least number of adders needed. Another function that is included in *dmnum* function is to find all the possible combinations for every multiplier. This operation is done by heuristically defining all the possible combinations.

This function is presented as follows:

$$[posib] = pnum(k, j, p) \quad (5.4)$$

where k is the index of prime factors which are not equal to two; j is the last index of prime factors; and p is the remaining prime factors. Notice that the k , j and p are the indexes for a specific row and column of the matrix. It is for programming purposes. The function returns a set of possible combinations of each multiplier.

5.4 Implement DM technique after Optimization

There are arguments concerning the reason to obtain the optimized coefficients by DM implementation after CSD optimization. This is called the CSD/DM technique. Originally, this technique is done by taking a set of coefficients after the *remez* function and implementing them using pure DM technique which results in fewer adders required but the number of adders is not minimized. So to prove this case, an experimental program is written to find the optimized number of adders using pure DM technique after obtaining the minimum wordlength and order of the filter. The reason that the order and wordlength optimization technique is included in this experimental program is because the fixed coefficients wordlength is difficult to determine by observation. The following shows the comparison of hardware saving between the implementation of DM technique with the combination of the three CSD optimization techniques and direct implementation of pure DM technique after *order-wordlength tradeoff* procedure. The following Tables 5.1 – 5.3 indicate the cost results of direct application of pure DM

implementation and the implementation of combining DM with the three CSD optimization techniques. The tables indicate the cost result of the simulations by varying the passband ripple (R_p), stopband ripple (R_s) and transition band (ΔF) over lowpass filter. From the experimental cost results, notice that more hardware is saved by the combined DM optimization implementation, when compared to just implementation of pure DM technique without going through the optimization process. The results are presented into three tables showing the cost reduction varying from 12% to 53%. Notice that at certain specifications, the cost is higher due to the number of coefficients needed to satisfy the specification constraint. Overall, the approximate average savings obtained is about 25%. In conclusion, more hardware savings is expected if implementing the combination DM optimization technique.

Lowpass Filter: $F_{\text{sampling}} = 20000\text{Hz}$ Transition Band = 0.2 [5000 7000]Hz Stopband Ripple, $R_s = 40\text{dB}$ Input wordlength = 8 bit			
R_p	Cost		% Reduction
	Pure DM	Combine DM	
5.5	724	512	29.28
5	796	512	35.58
4.5	770	582	24.42
4	770	582	24.42
3.5	794	642	19.14
3	888	636	28.38
2.5	939	712	23.93
2	1066	766	28.14
1.5	962	840	12.68
1	1152	844	26.74
0.5	1692	1061	39.95
0.1	1764	1518	13.95
Average			25.56

Table 5.1: Average cost reduction for varying R_p with $\Delta F=0.2$ and $R_s=40\text{dB}$

implementation and the implementation of combining DM with the three CSD optimization techniques. The tables indicate the cost result of the simulations by varying the passband ripple (R_p), stopband ripple (R_s) and transition band (ΔF) over lowpass filter. From the experimental cost results, notice that more hardware is saved by the combined DM optimization implementation, when compared to just implementation of pure DM technique without going through the optimization process. The results are presented into three tables showing the cost reduction varying from 12% to 53%. Notice that at certain specifications, the cost is higher due to the number of coefficients needed to satisfy the specification constraint. Overall, the approximate average savings obtained is about 25%. In conclusion, more hardware savings is expected if implementing the combination DM optimization technique.

Lowpass Filter: $F_{\text{sampling}} = 20000\text{Hz}$ Transition Band = 0.2 [5000 7000]Hz Stopband Ripple, $R_s = 40\text{dB}$ Input wordlength = 8 bit			
R_p	Cost		% Reduction
	Pure DM	Combine DM	
5.5	724	512	29.28
5	796	512	35.58
4.5	770	582	24.42
4	770	582	24.42
3.5	794	642	19.14
3	888	636	28.38
2.5	939	712	23.93
2	1066	766	28.14
1.5	962	840	12.68
1	1152	844	26.74
0.5	1692	1061	39.95
0.1	1764	1518	13.95
Average			25.56

Table 5.1: Average cost reduction for varying R_p with $\Delta F=0.2$ and $R_s=40\text{dB}$

Lowpass Filter: $F_{\text{sampling}} = 20000\text{Hz}$ Transition Band = 0.2 [5000 7000]Hz Passband Ripple, $R_p = 3\text{dB}$ Input Wordlength = 8 bit			
R_s	Cost		% Reduction
	Pure DM	Combine DM	
20	372	314	15.60
30	486	406	16.46
40	888	636	28.38
50	1416	976	31.07
60	1982	1355	31.63
70	2584	1860	28.02
80	3104	2382	23.26
Average			24.92

Table 5.2: Average cost reduction for varying R_s with $\Delta F=0.2$ and $R_p=3\text{dB}$

Lowpass Filter: $F_{\text{sampling}} = 20000\text{Hz}$ Passband Ripple, $R_p = 3\text{dB}$ Stopband Ripple, $R_s = 40\text{dB}$ Input wordlength = 8 bit			
Transition Band , ΔF	Cost		% Reduction
	Pure DM	Combine DM	
0.3	735	342	53.47
0.28	478	388	18.83
0.26	582	450	22.68
0.24	640	490	23.44
0.22	808	550	31.93
0.2	888	636	28.38
0.18	1032	840	18.60
0.16	1344	932	30.65
0.14	1563	1036	33.72
0.12	1440	1094	24.03
0.1	1846	1434	22.32
Average			28.00

Table 5.3: Average cost reduction for varying ΔF with $R_p=3\text{dB}$ and $R_s=40\text{dB}$

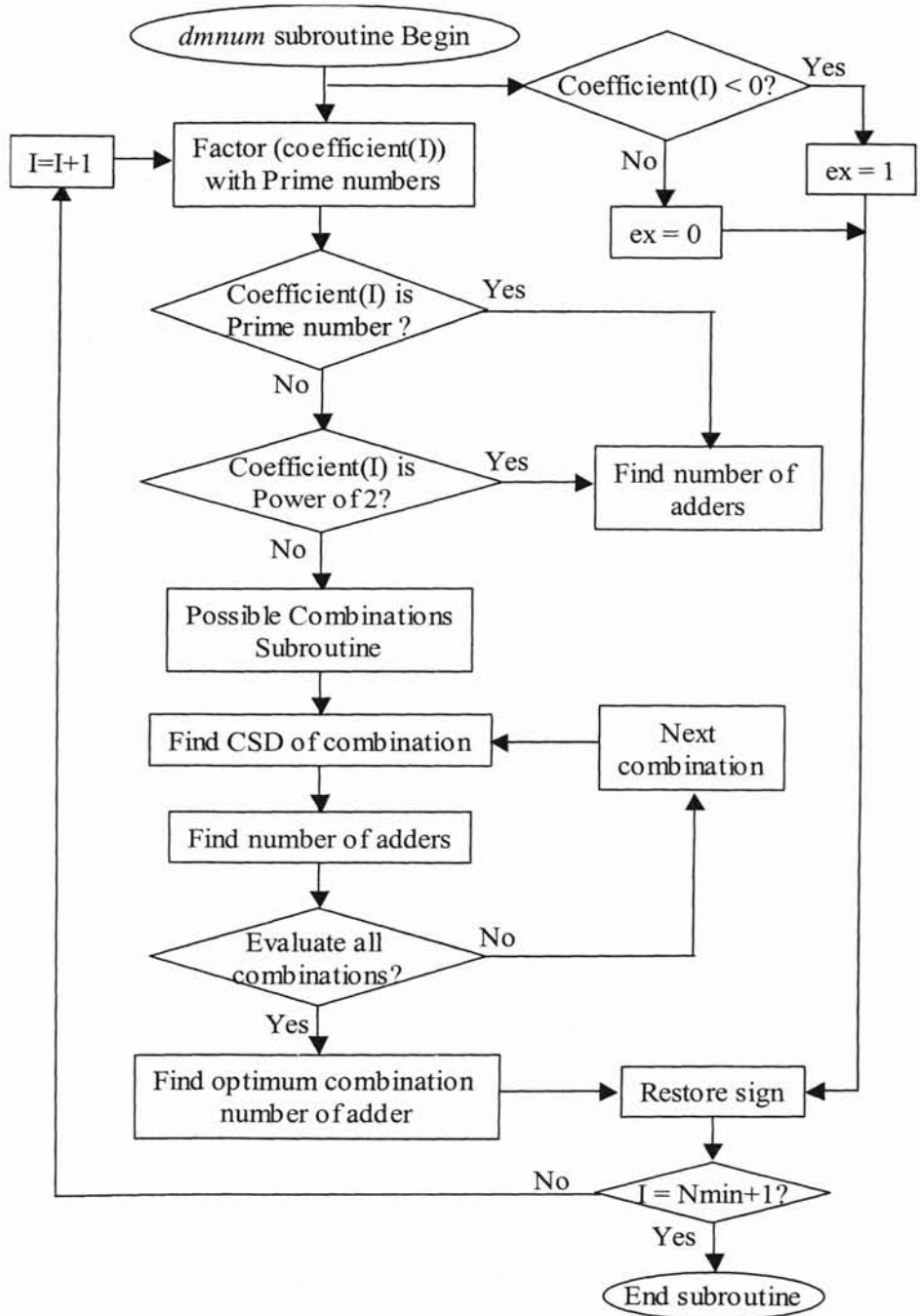


Figure 5.1: DM Optimization Algorithm Block Diagram

Section 6

Signal to Noise Ratio

6.1 Introduction and Approach

The limited precision of digital hardware has caused it to be difficult to implement the coefficients exactly. This is the oldest problem encountered by all DSP experts. The oldest and most effective solution to this problem is either truncate, roundoff or otherwise quantize the coefficients into a certain wordlength that can be implemented into hardware realization. Rounding off or truncating the coefficients, will cause additional noise to occur in the system. However, there are assumptions that need to be considered that result in existing noise in the filter. If the filter coefficients are truncated or rounded off before implementing them into the hardware, it will not result in the production of noise in the filter. However the quantization of the coefficients will just change the location of the FIR filter's zeros [16-17]. Also, noise will be introduced if the coefficients are rounded off or truncated after a shift operation, or an addition or subtraction operation, which occurs between each stage.

Previously, Husinga [1] used the SNR equation defined by Hartley [10] to compute the SNR of the FIR filter because Hartley's equation is applicable to CSD

representation. This idea is to further implement the coefficients that use the Dempster and Macleod technique. In fact this approach is reasonable because DM technique only optimizes the structure of the CSD expression of a multiplier. Also, remember that the term SNR is the ratio of the output signal variance to noise variance. In Husinga's thesis, she mentions using the Xilinx technology definition to fix structural adder size in order to approximate the structural adder and delay size. This idea is not adopted in this thesis. The structural adder size is determined by the input wordlength and the maximum shifted wordlength of the coefficients. The main purpose of considering the maximum shifted wordlength of the coefficients is to maintain the precision of the optimized coefficients and to eliminate the redundant zero bits.

6.2 SNR Computation

Computing estimated SNR for FIR filters is the last subroutine of the program. To compute SNR for the filter, first the structural adders or delays size are determined. The main idea to find a suitable structural adder size with a reasonable SNR value and noise is generated as the roundoff operation is done after each multiplier of every stage. The noise presents in the filter is assumed to be only white noise. This concept can be equivalent to injecting additive noise to every node of the filter before entering the adder or subtractor [17-19] and the noise. Figure 6.1 shows the noise model concept graphically with highlighted structural adders and delays. The noise from each node is summed together as a single noise source. Also, due to the hardware equivalent, FIR filter is written in hardware language, so the size of the adder is more flexible.

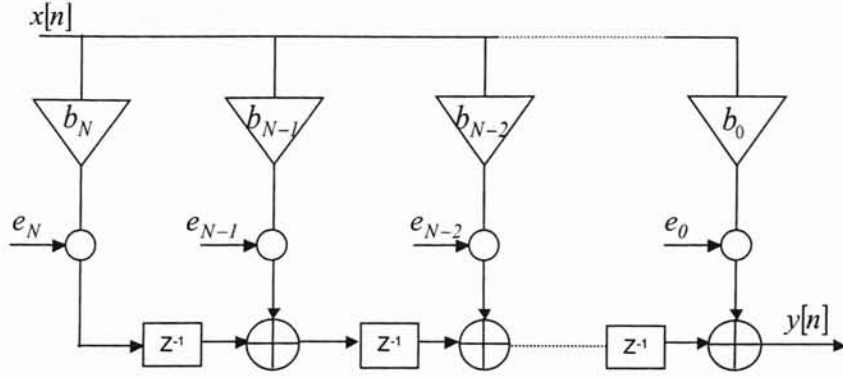


Figure 6.1: Transpose Direct Form FIR Filter with Noise Model

The size of the structural adder is estimated as large as possible to preserve the coefficients. The structural adder or delay size is stated as $badd$ and is computed as follows:

$$badd = \begin{cases} b_{in} + b_{max_shift} & ; \text{User specify input bit, } b_{in} \\ b_{coefficient} + b_{max_shift} & ; \text{Otherwise} \end{cases} \quad (6.1)$$

For the first case, if the user specifies the input bit, b_{in} the structural adder size, $badd$ is computed by adding the input bit to the maximum shifted wordlength of the coefficients, b_{max_shift} . The second case is by considering if the user does not specify the input bit size, then the input bit size is assumed to be the wordlength of the coefficients, $b_{coefficient}$. Using the structural adder size computed from Equation 6.1, the SNR is estimated using Hartley's [10] equation.

The general error variance for the FIR filter for both CSD and DM representation is

$$\delta_{\text{error}}^2 = \sum_{i=0}^{\text{order}} \frac{\Delta_i^2}{12} \quad (6.2)$$

where the i^{th} coefficient's maximum error introduced by Hartley is

$$\Delta_i = \sum_{j+k \geq \text{badd}} x_j b_{ik} 2^{-(j+k)} \quad (6.3)$$

Note that Equation 6.3 is the product of the input, x and the i^{th} coefficient, b_i . The notation j is the index for input x and the notation k is the index for b_i . The equation shows that the maximum error is calculated from the summation of the errors occurring at the wordlength, which is the total wordlength of input bit size and coefficient bit size, that is equal or more than the computed structural adder size, badd .

Also know that the output signal variance is

$$\delta_{\text{output}}^2 = \frac{1}{2} \sum_{i=0}^{\text{order}} (\text{coefficients})^2 \quad (6.4)$$

as the one half is actually the input variance, which is a sinusoid input. Hence, SNR is computed simply by substituting both Equation 6.2 and 6.4 in the following equation:

$$\text{SNR} = 10 * \log_{10} \left(\frac{\delta_{\text{output}}^2}{\delta_{\text{error}}^2} \right) \quad (6.5)$$

For the exceptional case if the user wants to find the minimum SNR, SNR_{min} for the filter that satisfying the required SNR requirement, $\text{SNR}_{\text{requirement}}$, which is set by the same user, the program will go to the different conditional loop. A set of SNR values will be computed based on the range from minimum coefficient wordlength to the structural adder size, badd and stored in a matrix. Hence, the set of the SNR corresponds to the

different size of the structural adder. Every element from the matrix is compared with the specified $SNR_{requirement}$ and once a higher or equal value is found, the search process is stopped. The new estimated SNR_{min} and the corresponding new minimum structural adder size will be displayed. This specifies that as long as the minimum adder size is implemented in hardware construction of the FIR filter, the $SNR_{requirement}$ value will be satisfied. Also, note that the program computed SNR value is the estimated value, which means that the SNR value obtained from the experimental data may appear different from the SNR computed by the program.

6.3 Pseudocode For SNR Computation

The procedure will start by determining the structural adder size, $badd$. Then the CSD coefficients table, $newtable$; minimum wordlength and order; input bit, bin ; adder size, the quantized coefficients, QC and the status of setting the SNR requirement, are fed into the $calcerr$ function subroutine, which is defined as computing the SNR. The function is shown in Equation 6.5:

$$[SNR, newbadd] = calcerr(newtable, bmin, bin, badd, Nmin, QC, SNRset) \quad (6.5)$$

First of all, the subroutine will calculate the output variance. The subroutine has two conditions that will be determined by the SNR status, where the SNR status is the constraint set by the user. If the SNR status is on, then a set of SNRs will be calculated based on a range starting from coefficient wordlength, $bmin$ to the overall adder size, $badd$. By evaluating the computed SNR values, the minimum SNR value that meets the SNR requirement will be determined, and the new structural adder size, $newbadd$ will be

estimated. On the other hand, if the SNR status is off, then SNR value will be calculated based on the overall adder size obtained from the calculation. Finally, a new SNR cost is evaluated and the SNR cost is compared to the initial minimum cost obtained from the optimization routine. Also, note that if the program is unable to find the SNR value that meets the SNR requirement, then the program will only output the largest SNR value that it can find.

Signal-to-Noise ratio (*calcerr*) Subroutine

```

Calculate output signal variance
If SNRset is off
    Loop                                % size of Nmin
        calculate all the coefficients error for adder size, badd
        sum the absolute of maximum error
        calculate the noise variance for single coefficient
    End loop
Compute SNR
Else
    Loop                                % range from bmin to badd
    Loop                                % size of Nmin
        calculate all the coefficients error for adder size, badd
        sum the absolute of maximum error
        calculate the noise variance for single coefficient
    End Loop
    Compute SNR
End Loop
Test for minimum SNR to meet SNRset (constraint)
Endif

```

Figure 6.2: Pseudocode of SNR subroutine

6.4 Example Simulation Result

Four sets of cost results are calculated and presented in four tables shown as Table 6.1 to Table 6.4. The tested filter is an example lowpass filter with specifications of sampling frequency at 2000Hz, passband ripple at 3dB, stopband ripple at 40dB and the input wordlength is set to eight bits. The transition band is varied from a ratio of 0.1 to 0.3. For a FIR filter to achieve infinite SNR, the structural adder size must be relatively large to preserve both the precision of the coefficients, which is the summation of maximum shift wordlength of the coefficients and the input data. Table 6.1 presents the cost results obtained from the *DM with optimization* column of Table 5.3 in which the cost results are equivalent to the cost results of the infinite SNR.

Lowpass Filter: F_{sampling} = 2000Hz Passband Ripple, R_p = 3dB Stopband Ripple, R_s = 40dB Input Wordlength = 8 bit SNR Requirement = Infinite dB			
Transition Band, ΔF	Maximum Shift of Coefficients	Original Structural Adder Size	SNR cost (DM with Optimization)
0.3	6	14	342
0.28	7	15	388
0.26	8	16	450
0.24	9	17	490
0.22	10	18	550
0.2	9	17	636
0.18	10	18	840
0.16	11	19	932
0.14	10	18	1036
0.12	11	19	1094
0.1	11	19	1434
Average			744.73

Table 6.1: Average Cost of Infinite SNR

Lowpass Filter: F_{sampling} = 20000Hz Passband Ripple, R_p = 3dB Stopband Ripple, R_s = 40dB Input Wordlength = 8 bit SNR Requirement = 80dB					
Transition Band ΔF	Maximum Shift of Coefficients	Original Structural Adder Size	Minimum Structural Adder Size	Minimum SNR (dB)	New Cost
0.3	6	14	14	77.02	342
0.28	7	15	15	82.01	388
0.26	8	16	16	87.12	450
0.24	9	17	16	81.62	470
0.22	10	18	17	88.82	530
0.2	9	17	16	81.82	610
0.18	10	18	17	83.50	810
0.16	11	19	17	84.26	872
0.14	10	18	17	87.45	998
0.12	11	19	17	84.66	1018
0.1	11	19	17	82.87	1342
				Average	711.82

Table 6.2: Minimum SNR, Minimum Structural Adder Size, Average Cost for Varying Bandwidth, ΔF with SNR Requirement = 80dB

Table 6.2 to Table 6.4 present the cost results using the set of specifications mentioned earlier with the SNR requirements of 80dB, 60dB and 40dB respectively. Taking the case to achieve SNR of 40dB, the minimum SNR computed must be 40dB and above. Table 6.2 to Table 6.4 show the result of the minimum SNR and the new structural adder size that are able to satisfy the SNR requirements. However, in Table 6.2, the filter with transition band of 0.3 does not meet the SNR of 80dB because the structural adder size is at the maximum wordlength of fourteen bits. So the maximum SNR that this filter can provide is only about 77dB. Also, the average cost values are computed for each table, which are 744.73 for infinite SNR; 711.82 for SNR of 80dB; 644 for SNR of 60dB and

560 for SNR of 40dB. Observe the results in the tables starting from Table 6.1 to Table 6.4. The structural adder sizes are getting smaller as the SNR requirements are reduced. Hence, comparing the cost average of Table 6.1 (infinite SNR) and the cost average of Table 6.2 (SNR of 80dB), the average percentage of hardware reduction is 4.42%. Hardware reduction is further improved a SNR of 60dB is allowed rather than SNR of 80dB. Compare to the average cost for infinite SNR; by reduce the SNR requirement from 80dB (Table 6.3) to 40dB (Table 6.4), the hardware cost savings increases from 4.42% to 24.8%.

Lowpass Filter: F_{sampling} = 2000Hz Passband Ripple, R_p = 3dB Stopband Ripple, R_s = 40dB Input Wordlength = 8 bit SNR Requirement = 60 dB					
Transition Band ΔF	Maximum Shift of Coefficients	Original Structural Adder Size	Minimum Structural Adder Size	Minimum SNR (dB)	New Cost
0.3	6	14	13	67.02	324
0.28	7	15	13	62.72	348
0.26	8	16	13	60.51	384
0.24	9	17	14	64.40	430
0.22	10	18	14	63.18	470
0.2	9	17	14	64.90	558
0.18	10	18	15	66.34	750
0.16	11	19	14	61.60	782
0.14	10	18	14	61.46	884
0.12	11	19	14	62.23	904
0.1	11	19	15	65.58	1250
				Average	644

Table 6.3: Minimum SNR, Minimum Structural Adder Size, Average Cost for Varying Bandwidth, ΔF with SNR Requirement = 60dB

Lowpass Filter: F_{sampling} = 2000Hz Passband Ripple, R_p = 3dB Stopband Ripple, R_s = 40dB Input Wordlength = 8 bit SNR Requirement = 40dB					
Transition Band ΔF	Maximum Shift of Coefficients	Original Structural Adder Size	Minimum Structural Adder Size	Minimum SNR (dB)	New Cost
0.3	6	14	10	44.00	270
0.28	7	15	10	42.03	288
0.26	8	16	10	40.01	318
0.24	9	17	11	43.69	370
0.22	10	18	11	42.64	410
0.2	9	17	11	43.63	480
0.18	10	18	12	44.68	660
0.16	11	19	11	43.40	692
0.14	10	18	11	41.40	770
0.12	11	19	11	43.84	790
0.1	11	19	12	44.66	1112
				Average	560

Table 6.4: Minimum SNR, Minimum Structural Adder Size, Average Cost for Varying Bandwidth, ΔF with SNR Requirement = 40dB

Section 7

Program Review

7.1 GUI Overview

The graphical User Interface(GUI) of the computer aided design software; *CSD and DM FIR Filter design program* contains several features that aid users in FIR filter design. The GUI is shown in Figure 7.1. Now by referring to the figure on the left of the GUI program, there is a magnitude plot area for the user to have a glance at the frequency response of the FIR filter design. On the right side, notice a large blue box, which has a menu with selection options. First, there is a ***help*** button that provides general information on how to use program. Consider the ***help*** button as the reference position. Below it there is a ***technique selection*** button that allows the user to select the desired implementation techniques. Beside the ***help*** button on the right, is a ***technology*** button that consists of several targeted technologies. Then, below the technology button, there is a ***filter type selection*** button. The filter types include *lowpass*, *highpass*, *bandpass* and *bandreject* filters. Once the type of filter is selected, the ***specification*** text boxes will appear based on the type of filter chosen. The next section of the GUI contains three buttons. The user can enter the desired wordlength or set the ***Auto*** button. Additionally,

the user can enter the desired Signal to Noise Ratio by turning on the selected **SNR (dB)** button and entering the desired SNR value on the text box beside the **SNR** button. Finally, there are two execution buttons, which are **Design** and **Layout** buttons. To run the program, the user must press the **Design** button. When the program finishes its job, a new frequency response plot is displayed on the GUI program and a summary box will appear. The summary box displays the parameter results regarding the filter design such as the coefficients, order, number of adders required, scale factor and the SNR value. An example summary box and the summary data in the work command window are displayed in Figure 7.2 and Figure 7.3 respectively. The user can press the **Layout** button for hardware realization. Appendix B1 provides the introductory menu for running the program.

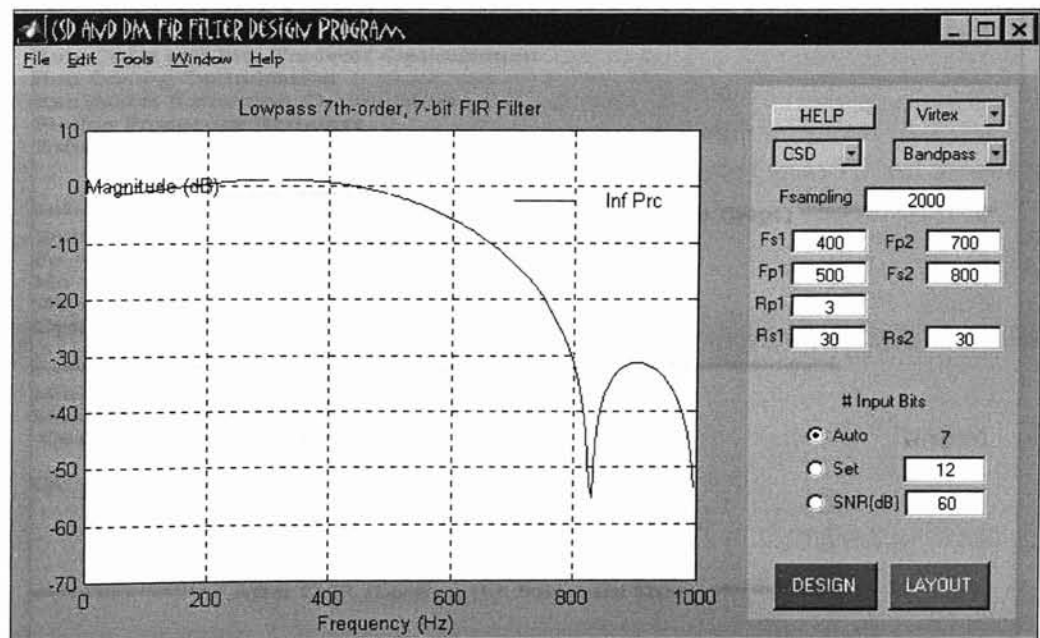


Figure 7.1: CSD and DM FIR Filter GUI

CSD and DM FIR Filter Design Program
File Edit View Insert Tools Window Help

FILTER DESIGN SUMMARY SHEET

16-Mar-2002

Enter Design Name

Technology = Xilinx **Technique =** CSD

Specifications: **Lowpass Filter**

Fsamp = 2000 Hz
Frequency Spec = 500 800 (Hz)
Rpass = 3 (dB)
Rstop = 30 (dB)

Results:

Order = 5	User set Input bits = 8
SNR= 82.1788	Coef. Wordlength = 6
Minimum Structural Adder/Delay Size = 14	Scale factor = 1.65
Total Number of adders = 7	

** Please refer to the Workspace for more data.

Enter file name

Figure 7.2: An Example Summary Box

```

Getting Specifications.....
Loading Technology Parameters.....
Clear Figure.....
Run Order and Bits Tradeoff Optimization .....
Run Scaling Optimization .....
Run Adder Extraction Optimization .....
Plotting Frequency Response .....
Estimating SNR cost and adder size .....

***** Optimum Order (Nopt) and Wordlength (bopt) *****
Optimum Order = 6
Fo = 0      0.5      0.8      1
Mo = 1 1 0 0
Wt = 1      5.961
Optimum Wordlength = 12

***** After Scaling and Adder Extraction *****
Minimum Order = 5
Minimum Wordlength = 7
Quantized Coefficients (QC)

QC =
-0.0625  0.0313  0.3125  0.3125  0.0313 -0.0625

***** After SNR (Look at the Summary Sheet)*****

```

Figure 7.3: Summary Data in Work Window Command

7.2 Program Overview

There are three main function files in the main routine. The operation of *csddesign2.m* is to generate a GUI menu as an initial step in the design process. The GUI menu consists of two options, which are **Design** (*csdoptimized.m*) and **Layout** (*layout.m*) options. Figure 7.4 presents the hierarchy of the main program flow. If the **Design** option is chosen, then the program will perform the optimization process. A detailed program flow of the whole optimization process is shown in Figure 7.6. To view more details of the program file chart, please refer to Appendix B2. Note that if no filter coefficients are initially optimized, the **Design** option must be implemented first to obtain some output results for hardware realization. When the optimization process is completed, the output result will be summarized. Then if the **Layout** option is selected, Matlab will generate a script file for hardware realization. This script file is named *params.vhd*. The script file contains coded information, which describes the characteristic of the coefficients and the information is supplied to the VHDL package files that will be programmed into a Xilinx FPGA chip for experiment. An example script file is shown in Figure 7.5 and it describes the optimized coefficients, which are in CSD representation.

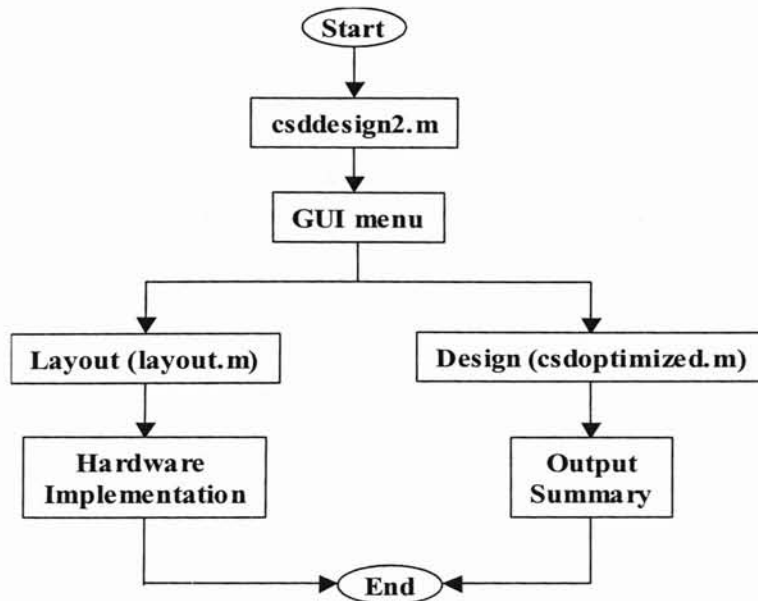


Figure 7.4: General Program Flow

```

1 Library IEEE;
2 use IEEE.std_logic_1164.all;
3 use IEEE.std_logic_unsigned.all;
4
5 PACKAGE params IS
6
7 TYPE int_arr1 IS ARRAY(natural range <>) OF INTEGER;
8 TYPE int_arr2 IS ARRAY(natural range <>) OF INTEGER;
9 TYPE int_arr3 IS ARRAY(natural range <>) OF INTEGER;
10
11 CONSTANT N: INTEGER:= 9;
12 CONSTANT b: INTEGER:= 9;
13 CONSTANT b_stage: int_arr1(0 to 9):=( 1, 1, 1, 1, 1, 1, 1, 1, 1, 1);
14 CONSTANT n_per_row: int_arr2(0 to 9):=( 1, 1, 3, 2, 1, 1, 2, 3, 1, 1);
15 CONSTANT coeff_vec: int_arr3(0 to 15):=(-8, 6, 4, -6, 8, 3, -5, 3, 3, 3, -5, 4);
16 CONSTANT zero_con: INTEGER:= 9;
17 CONSTANT bshift: INTEGER:= 18;
18 CONSTANT yex: INTEGER:= 2;
19 CONSTANT NN: INTEGER:= 4;
20 CONSTANT ODD: INTEGER:= 1;
21
22 END package;
  
```

Figure 7.5: An Example CSD *params.vhd* File generated by Matlab

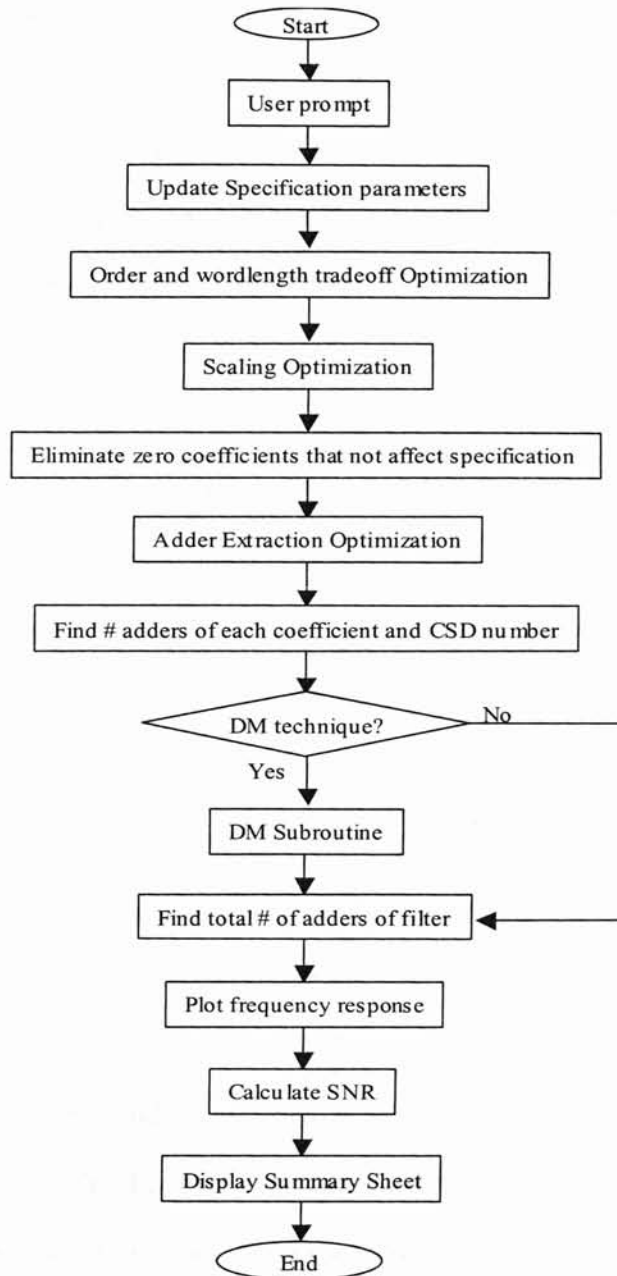


Figure 7.6: Program Flow Chart

7.3 Technology and Cost Review

Initial work demonstrates that the cost function is technology dependent. The cost function is the heart of evaluating the hardware requirement and also exhibits the flexibility regarding the targeted technology. Nowadays the synthesis tools for FPGAs are able to optimize the area and the speed of the circuit, which are compatible to specific technology. So the cost function must be defined only to be technology independent and can compute the estimated value that measures the amount of hardware area needed for the filter design. In this thesis both Synplify Pro and FPGA Express are considered the main logic synthesis tools for Xilinx FPGAs. The cost function of this thesis is redefined to be more general and technology independent since our primarily target technology is Xilinx FPGA. The general cost function is incorporated in the program and it is derived based on the worst-case scenario. The general cost function is shown as follows:

$$\text{cost} = \text{adders}(\text{coefficient}) \times \text{wordlength} + 2 \times \text{order} \times ((\text{Input bits} + \text{wordlength}) + \lceil \log_2 \text{order} \rceil) \quad (7.1)$$

This cost function is defined based on the combination of the general fixed point summation rule for FIR filters [23] and the cost function defined by Husinga [1]. The general fixed point summation rule for FIR filters states that for unknown coefficients of a FIR filter, the general adder wordlength is estimated based on the filter order, input and coefficient wordlength. The adder wordlength equation is displayed as follows:

For a N tap FIR filter, L bit input, M bit coefficients

$$\text{adder wordlength} = L + M + \log_2(N) \quad (7.2)$$

Equation 7.2 is derived to maximize the Signal Noise Ratio, to maintain the precision of the coefficients and to avoid any overflow occurring, which covers the worst case scenario. Also the old cost function introduced by Husinga is maintained in the program, which is defined for the MOSIS 1.2um CMOS technology. In the future, other target technologies can be added to the program such as the 0.25um Silicon On Insulator (SOI) CMOS or other technologies that are applicable to FIR filter design.

Section 8

Conclusion and Future Work

8.1 Conclusion

In this thesis, the three algorithms are examined carefully to optimize the coefficients using both CSD representations. Other algorithms are also discussed such as optimizing DM technique and SNR constraint. Knowing the fact that Dempster and Macleod (DM) technique is efficiently used to reduce the hardware cost for the fixed coefficients FIR filter, at this point DM technique can be stated as the minimal structure representation of CSD representation. Hence, the optimal DM technique can be obtained from the optimum CSD coefficients. One observation shows that among all these optimization techniques, scaling technique provides the most substantial hardware savings result.

In theory, DM implementation provides more savings compared to CSD representation. However, the optimization of CSD coefficients often results in a very small amount of nonzero bits, which means fewer adders are required. Consequently, the DM technique does not always show a substantial savings over optimized CSD representation. Table 8.1 shows a list of results generated by the program for Lowpass

filter by fixing the stopband ripple equal to 40dB and varies the sampling frequency, passband frequency, stopband frequency and the passband ripple. The last two columns on the right show the number of total adders required for the whole filter implementing both CSD and DM techniques. It shows that for some filter specifications there are savings in the hardware area between CSD and DM techniques. However, the truth is, most of the time both optimized CSD and DM results in the same number of adders. For this case, other factors may distinguish the best of either technique. Moreover the choice for choosing either CSD or DM implementation is up to user's preference. The main idea of this thesis is to produce a computer program that allows the user to design any type of FIR filter using either CSD or DM techniques, which will speed up the design process.

Sampling Frequency (Hz)	Passband Frequency (Hz)	Stopband Frequency (Hz)	Passband Ripple (dB)	Filter Order	Number of Adders	
					CSD	DM
20000	4000	4500	1	62	102	100
20000	5000	5300	1	105	171	170
20000	5000	8000	1	11	17	17
48000	9000	12750	3	17	28	28
48000	12000	15000	3	23	29	29
48000	12500	15500	3	21	34	34
48000	13000	15000	3	29	43	43
48000	15000	15500	3	120	187	186
48000	15000	18250	3	18	30	29

Table 8.1: Results Generated with different example Lowpass Filter Specifications with Stopband Ripple fix at 40dB

Also note that the hardware requirement for the minimum SNR that meets the SNR constraint set by the user is smaller compared to the hardware needed for both optimized

CSD and DM techniques. With the CAD program, a FIR filter with less hardware requirements can be designed and the optimum output filter design, which is generated in VHDL description, is implemented in the Xilinx FPGA.

8.2 Future Work

In the future, direct form Impulse Infinite Response (IIR) filter design can be added to the program since IIR filters has its own advantages just as FIR filters do. For instance, one advantage is IIR filters need less order to meet the filter specifications that are similar to FIR filters. Another advantage is the digital filters, which are difficult to design using FIR filters, but are easy to design using IIR filters. Future researchers can also expand the program by adding more design options that can design either FIR filters or IIR filters with different equivalent filter structures such as cascade structure and lattice structure. Different equivalent filter structures may results in different needs of the hardware requirements.

The Residue Number System (RNS) arithmetic is another technique that can be implemented in FIR and IIR filter design [24], which results in hardware savings. RNS arithmetic is a true integer number system. Hence the RNS system does not encounter round off or truncation errors. Unless the coefficients are rounded off without exceeding the dynamic range before implementing to the RNS arithmetic, no noise will occur in the filter. The main thing is this technique has sufficient dynamic range to accommodate the coefficient multipliers. In the future the RNS arithmetic can be added to the CAD where

there are options for the user to enter the modulus, the word length of the coefficient multipliers and order of the FIR filter needed for the design.

Another idea is to implement the idea introduced by Hartley [10] and Dempster [6-7], who designed non-fixed FIR structure to minimize the hardware utilization. Their approach is difficult to implement because it needs to investigate different approaches to find and optimize the partial sums across the coefficients. The algorithm introduced by Hartley is to calculate the number of adders using sub-expression method and to define a new FIR structure based on the calculations. Another author, Benyamin [25] implements Dempster's idea for filter design with optimized hardware area. This paper presents a method called the Multiple Constant Multiplier Trees (MGMTs), which relies heavily on common subexpressions elimination (CSE). The heart of MCMT algorithm is the representation of common subexpressions contained in constant data patterns, which provide several optimization capability strategies. There are other optimization algorithms that may result in the least hardware requirement for the filter coefficients, which are worthwhile to study. One example of the new optimization algorithm is introduced by Persson [26], which is called the multimode mean field annealing (MM-MFA). Another approach is to replace adder extraction technique with Genetic Algorithm (GA) since GA may allow higher possibilities for finding coefficients that will result in additional hardware savings. There is a new idea of representing the coefficients such as the minimal signed digit (MSD), which is currently being introduced by Park and Kang [27] and may result in more hardware area savings compared to CSD and DM techniques.

Bibliography

- [1] D. L. Husinga, *Digital of Optimized Filter Using CSD Coefficient Representation*. Master's Thesis, University of California, Davis, CA.1995.
- [2] Darren S. Jue, *Optimal Design of Canonic Signed Digit IIR Digital Filter*. Master's Thesis, University of California, Davis, CA 1996.
- [3] Naren B. Balasubramanian, *Optimal Design of Digital Filter using CSD Coefficients*. Master's Thesis, University of California, Davis, CA 1997.
- [4] Choi, Hangsuk, *Implementation of DSP part of modulator systems*. Master's Thesis, University of California, Davis, CA 1995.
- [5] Kai Hwang, *Computer Arithmetic: Principle, Architecture, And Design*. John Wiley & Son, Inc. 1979
- [6] Andrew G. Dempster and Malcolm D. Macleod, "Use of minimum-adder multiplier blocks in FIR digital filters," *IEEE Trans. Circuits and Systems*, vol. 42, no 9, pp. 407-413, October 1994.
- [7] Andrew G. Dempster and Malcolm D. Macleod, "Constant integer multiplication using minimum adders," *IEE Proceedings CircuitsDevices Systems*, vol. 141, no 5, pp. 569-576, September 1995.

- [8] A.G. Dempster and M.D. Macleod, "Comparison of fixed-point FIR digital filter design techniques," *IEEE Trans. Circuits and Systems*, vol. 44, no. 7, pp. 591-593, July 1997.
- [9] D. Kodek and K. Steiglitz, "Filter-length word-length trade-offs in FIR digital filter design," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. 28, pp. 739-744, December 1980.
- [10] Richard Hartley, "Optimization of canonical signed digit multipliers for filter design," *Proceedings IEEE International Symposium on Circuits and Systems*, vol. 4, pp. 485-488, 1994.
- [11] R. Jain, P.T. Yang, T. Yoshino, "FIRGEN: A Computer Aided Design system for high performance FIR filter Integrated Circuit," *IEEE Transaction on Signal Processing*, vol. 39, pp. 1655-1668, July 1991.
- [12] M. Halder, A. Nayak, Nagrajshenoy, A. Choudhary and P. Banerjee, "FPGA hardware synthesis from MATLAB," *VLSI Design, Fourteenth International Conference*, pp 299-304, 2001.
- [13] Soderstrand, M.A., Johnson, L.G., Arichanthiran, H., Hoque, M.D., Elangovan, R, "Reducing hardware requirement in FIR filter design" *IEEE Trans. Acoust. Speech, Signal Processing*, vol.6, pp. 3275 –3278, 2000.
- [14] E. de la Serna and M. A. Soderstrand. "Trade-off between FPGA resource utilization and roundoff error in optimized CSD FIR digital filters," *Proceedings of 28th Asilomar Conference on Signals, Systems and Computers*, Pacific Grove, CA, pp. 187-191, November 1994.

- [15] M. A. Soderstrand, N. Balasubramanian, D. Husinga and M. Potharlanka. "An optimal automated implementation of FIR filters on field programmable gate arrays," *Proceedings of the International Conference on Signal Processing Applications and Technology*, Boston, MA, October 1995.
- [16] Litwin. L . "FIR and IIR digital filter," *IEEE Potential*, Volume 19, Issue 4, pp. 28-31, Oct.- Nov. 2000..
- [17] A. V. Oppenheim and R. W. Schafer, *Discrete - Time Signal Processing*. Prentice Hall, Englewood Cliffs, NJ. 1989.
- [18] Sanjit K. Mitra, *Digital Signal Processing: A Computer-Based Approach*. McGraw- Hill Companies, Schaum's, New York 1998.
- [19] Boaz Porat, *A Course in Digital Signal Processing*. John Wiley, New York 1997.
- [20] V. Pasham, A. Miller and K. Chapman, *Transposed Form FIR Filters*, Xilinx Application Note, XAP219 (v1.1), Jan 10, 2001. (<http://www.xilinx.com/xapp/xapp219.pdf>)
- [21] L.S. DeBrunner, V. DeBrunner, P. Pinault, "Variable wordlength IIR filter implementations for reduced space designs," *Signal Processing System*, pp. 326-339, 2000.
- [22] M. Yagyu, T. Yoshida, A. Nishihara, N. Fujii. "Design of FIR digital filters with minimum weight representation," *Proceedings IEEE International Symposium on Circuits and Systems*, pp. I/227-230, 1995.
- [23] Randy Yates, *Practical Considerations in Fixed-Point FIR Filter Implementations*. Digital Sound Lab, Digital Audio Signal Processing, March 2001.

- [24] M.A. Soderstrand, K. Al-Marayati "VLSI Implementation of very-high-order FIR filters," *Proceedings IEEE International Symposium on Circuits and Systems*, Vol. 2, pp. 1436-1439, 1995.
- [25] D. benyamin, W. Luk, J. Villasenor, "Optimizing FPGA-based vector designs," *Field-Programmable Custom Computing Machines, Proceedings Seventh Annual IEEE Symposium*, pp. 188-197, 1999.
- [26] Persson. P, Nordebo. S and Claesson, I. "Hardware efficient digital filter design by multimode mean field annealing," *IEEE Signal Processing Letters*, Vol. 8, No. 7, July 2001.
- [27] Park. I-C, Kang. H-J. "Digital filter synthesis based on minimal signed digit representation," *Design Automation Conference*, Las Vegas, Nevada, pp. 468-473, June 2001.

Appendix A

Binary to CSD representation Conversion Algorithm

Hwang [5] procedure of converting binary numbers to CSD representation is implemented to the program. The function is named as *CSD* and the command is presented as following:

$$[C] = \text{csd}(x, k1, k2) \quad (\text{A-1})$$

The input x can be a decimal scalar or binary vector. $k1$ and $k2$ is denoted as the precision of input x in CSD representation, where $k1$ is the wordlength on the left side of the binary point and $k2$ is the wordlength on the right side of the binary point. Note that summation of $k1$ and $k2$ will be the total wordlength needed to represent input x . Finally, C is the CSD representation of input x .

The following will simply explain Hwang's method of computing CSD number. Lets consider that any binary number as B and let b_i be the binary bit, the binary number is represented as below:

$$B = \sum_{i=0}^n b_i 2^i \quad (\text{A-2})$$

Taking the binary number and represent them in CSD number, the equivalent CSD number is express as:

$$C = \sum_{i=0}^n d_i 2^i \quad (\text{A-3})$$

where d_i is the element from the set $\{-1 \ 0 \ 1\}$.

The following is Hwang algorithm to convert into binary number to CSD representation:

1. For a new binary number:
 - Set index, $i = 0$, where it represents the least Significant Bit (LSB) of the Binary number, B .
 - Set the initial carry, $c_0 = 0$.
2. Then find the next carry using this equation $c_{i+1} = ib_i + b_{i+1} + c_i$.
3. Using the following equation to generate d_i for vector d from Equation (A-3):

$$d_i = b_i + c_i - 2c_{i+1}$$

4. If $i < n$: $i = i + 1$ and go to *Step 2*.

If $i = n$: Go to *Step 1* for the next binary number until it reaches the

Appendix B

CSD and DM FIR Filter Design Program

B.1 Introductory Menu

1. Launch the Matlab software. You will see a window command. Now set the desired directory. For example *C:\MATLABR11\work\new_csdoptimized1*.
2. Type *csddesign2* and press ENTER. A GUI labeled **CSD and DM FIR Filter Design Program** will appear. The GUI will look like the following Figure B-1.
3. First select the desired technology from the *technology* button. You can click on the *help* button to read the help files.
4. Then click on the *technique* button to select desired technique.
5. Select the *filter type* and enter the desired *filter parameters (specifications)* according to the listed text boxes.
6. Now enter the wordlength after triggering the *set* button if you know the desired input bits of the filter. Or else if you do not know the input wordlength, trigger on the *Auto* button. The program will define the input wordlength.

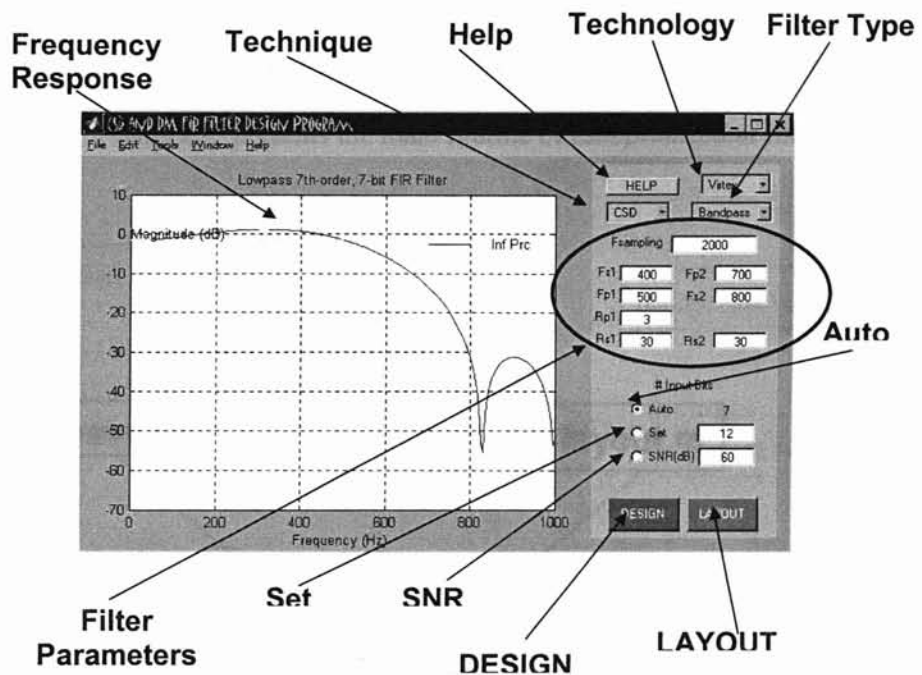


Figure B-1: CSD and DM FIR filter design program

7. If you wish to set the SNR constraint, trigger the *SNR* button and enter the desired SNR value in dB.
8. Click on the *DESIGN* button to run the program. As the program finishes its task, you will see a new and the old frequency responses on the plot on the left of the selection buttons. The summary will appear on the window and some results will display on the command window.
9. Click on the *LAYOUT* button for hardware implementation.

B.2 File Chart

The file chart presents the files of the program. Figure B-2 shows the main top level files of the program. Figure B-3 presents the main routine of *csdoptimized.m* file.

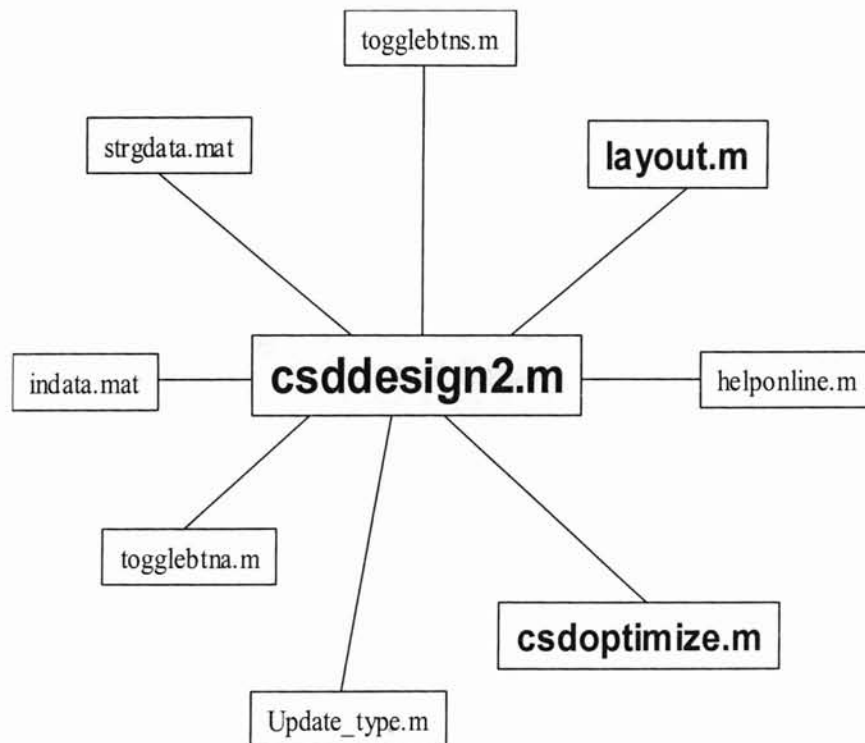


Figure B-2: Top Level of Program File

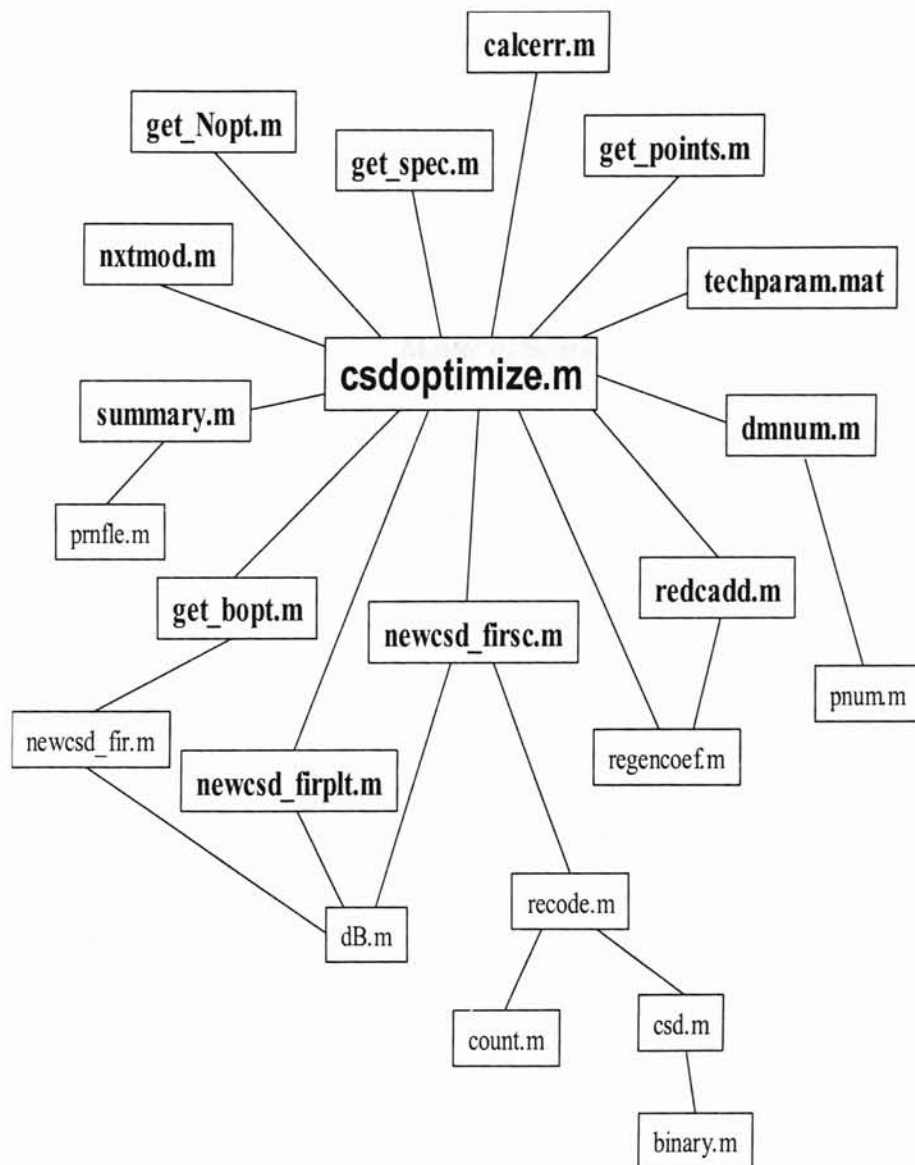


Figure B-3: File Chart for the Optimization Process

Ø

VITA

Wen Fung Leong

Candidate for Degree of

Master of Science

Thesis: OPTIMIZING FIR FILTER COEFFICIENTS USING CSD
REPRESENTATION AND DM TECHNIQUE

Major Field: Electrical Engineering

Biographical:

Education: Graduate from High School in Port Dickson, Malaysia in 1995; received a Bachelor of Science in Electrical Engineering from Oklahoma State University, Stillwater, Oklahoma in July 2000. Completed the requirements for Master of Science degree with a major in Electrical Engineering at Oklahoma State University in May, 2002.

Experience: Work as research assistant and teaching assistant while pursuing a Master degree with Department of Electrical Engineering at Oklahoma State University, 2000 to present.

Professional Membership: Institute of Electrical and Electronics Engineers, Inc (IEEE)