2016

Linear-Phase FIR Digital Filter Design with Reduced Hardware Complexity using Discrete Differential Evolution

Muhammed Kunwar Rehan
University of Windsor

Linear-Phase FIR Digital Filter Design with Reduced Hardware Complexity using Discrete Differential Evolution

by

Kunwar Muhammed Rehan

A Thesis
Submitted to the Faculty of Graduate Studies
through the Department of Electrical and Computer Engineering
in Partial Fulfillment of the Requirements for
the Degree of Master of Applied Science
at the University of Windsor

Windsor, Ontario, Canada

2016

Linear-Phase FIR Digital Filter Design with Reduced Hardware Complexity using Discrete Differential Evolution

by

Kunwar Muhammed Rehan

APPROVED BY:

Dr. Guoqing Zhang
Department of Mechanical, Automotive and Materials Engineering

---

Dr. Huapeng Wu
Department of Electrical and Computer Engineering

Dr. Hon Keung Kwan, Advisor
Department of Electrical and Computer Engineering

March 31, 2016

Declaration of Originality

Abstract

Optimal design of fixed coefficient finite word length linear phase FIR digital filters for custom ICs has been the focus of research in the past decade. With the ever increasing demands for high throughput and low power circuits, the need to design filters with reduced hardware complexity has become more crucial. Multiplierless filters provide substantial saving in hardware by using a shift add network to generate the filter coefficients. In this thesis, the multiplierless filter design problem is modeled as combinatorial optimization problem and is solved using a discrete Differential Evolution algorithm. The Differential Evolution algorithm's population representation adapted for the finite word length filter design problem is developed and the mutation operator is redefined for discrete valued parameters. Experiments show that the method is able to design filters up to a length of 300 taps with reduced hardware and shorter design times.

Dedication

Dedicated to my Parents and my siblings: Reesha, Saaim and Hamzah

Acknowledgements

I would like to thank my supervisor Prof. H.K. Kwan for introducing me the set-based discrete Differential Evolution and for suggesting it for discrete-valued digital filter design as the project of my thesis. Also, his guidance and support helped me to achieve successes. I would also like to thank my committee members Dr. H. Wu and Dr. G. Zhang for their valuable inputs. A special thanks goes to my colleagues Manpreet Malhi and Mohamed Abdinur. They provided me with constant encouragement and the discussion that I had with them prompted me to develop new ideas and provide breakthroughs.

Table of Contents

List of Tables

List of Figures

List of Acronyms

ACO Ant Colony Optimization

ASIC Application Specific Integrated Circuit

CR Crossover Rate

CSD Canonic Signed Digit

CSE Common Subexpression Elimination

DE Differential Evolution

DEFDO Differential Evolution Filter Design Optimization

EA Evolutionary Algorithms

EWL Effective Word Length

F Mutation Factor

FA Full Adder

FIR Finite Impulse Response

GA Genetic Algorithm

GB Graph Based

HA Half Adder

IIR Infinite Impulse Response

LTI	linear time invariant
MAD	Maximum Adder Depth
MBA	Multiplier Block Adders
MCM	Multiple Constant Multiplication
MILP	Mixed Integer Linear Programming
NPRM	Normalized Peak Ripple Magnitude
PSO	Particle Swarm Optimization
RAG-*n*	Reduced Adder Graph-n
SA	Structural Adders
SAN	Shift and Add Network
TSP	Travelling Salesman Problem

Chapter 1

Introduction to Digital Filter Design

A digital filter is a system that alters an incoming signal in a desired way in order to extract useful information and discard undesirable components. Digital filters are used pervasively in wide ranging products. Some examples include:

- Communication Systems
- Digital Audio Systems
- Signal Processing Systems including applications in seismology, biology etc.
- Image Processing and enhancement systems
- Speech Synthesis
- Instrumentation and Control Systems

This chapter will introduce the digital filter design problem and the terminologies associated. The specification of the digital filter optimization problem and the solution techniques will also be discussed. Lastly, the motivation for the research work and the outline of the thesis are given.

1.1 Mathematical Representation of Digital Filters

The emergence of digital technology in the 1960s opened a new world of applications. It was realized that digital filters have various advantages over their analog counterparts as

- Digital filters did not suffer from components tolerances and their response was invariant to temperature and time.
- Digital filters could be programmed easily on digital hardware
- Digital filters were insensitive to electrical noise to a great extent
- Digital filters are very versatile in the desired responses they can produce

A digital filter can be characterized as a linear time invariant (LTI) discrete system. The LTI system can be described by a constant coefficient difference equation

$$y(n) = \sum_{k=0}^{N-1} a(k)x(n-k) - \sum_{k=1}^{M} b(k)y(n-k)$$

where $a(k)$ and $b(k)$ are the forward tap coefficients and feedback tap coefficients respectively. Taking the z Transform of the above equation, and rearranging, we obtain the transfer function of the system shown in Eq. 1.1.

$$H(z) = \frac{Y(z)}{X(z)} = \frac{\sum_{k=0}^{N-1} a(k)z^{-k}}{1 + \sum_{k=1}^{M} b(k)z^{-k}} \tag{1.1}$$

From the Eq. 1.1, two sub classes of digital filters can be defined: Finite Impulse Response (FIR) filters, also called non recursive filters and Infinite Impulse Response (IIR) filters, also called recursive filters. Mathematically, the distinction is made for the case when poles are non existent in the transfer function. Hence, the denominator terms vanishes and the transfer function can be written as

$$H(z) = \sum_{k=0}^{N-1} a(k)z^{-k}$$

The above transfer function exhibits a finite length impulse response and hence is called the FIR filter transfer function.

Physically, the distinction is based on whether a feedback from the output exists or not. Recursive filters have a feedback from the output and hence possess impulse responses that are infinite in duration. A non-recursive or finite impulse response (FIR) digital filter, on the other hand, exhibits a finite duration impulse response. For an FIR filter whose impulse response of length[1] N is given by $\mathbf{h} = [h_0, h_1, h_2 \cdots h_{N-1}]^T$ the transfer function is found using the Z transform given by Eq. 1.2.

$$H(z) = \sum_{n=0}^{N-1} h_n z^{-n} \tag{1.2}$$

The frequency response is defined as the transfer function evaluated at $z = e^{j\omega}$. Hence, the frequency response of a FIR digital filter can be written as in Eq. 1.3

$$H(e^{j\omega}) = \sum_{n=0}^{N-1} h_n e^{-j\omega n} \tag{1.3}$$

The frequency response can be re-written as

$$H(e^{j\omega}) = H_a(\omega)\theta(\omega) \tag{1.4}$$

where $H_a(\omega) = |H(e^{j\omega})|$ is called the magnitude response and and $\theta(\omega) = \angle H(e^{j\omega})$ is called the phase response. The group delay (τ_g) of the filter is defined as

$$\tau_g(\omega) = -\frac{\partial\theta(\omega)}{\partial\omega} \tag{1.5}$$

By introducing symmetry in the impulse response, a linear phase or constant group delay can be insured in an FIR filter. Based on the type of symmetry, four types of linear phase FIR filters can be categorized:

[1]The order of the filter is one less than the number of taps i.e. $(N - 1)$

1. Type 1: The impulse response has odd number of coefficients (order of filter being even) and the coefficients are symmetric with respect to the midpoint.
2. Type 2: The impulse response has even number of coefficients (order of filter being odd) and the coefficients are symmetric with respect to the midpoint (not an actual point).
3. Type 3: The impulse response has odd number of coefficients (order of filter being even) and the coefficients are anti-symmetric with respect to the midpoint.
4. Type 4: The impulse response has even number of coefficients (order of filter being odd) and the coefficients are anti-symmetric with respect to the midpoint (not an actual point).

Due to the symmetry property of the linear phase FIR filters, their frequency response can be characterized by $M+1$ unique coefficients,where $M = \lfloor \frac{N-1}{2} \rfloor$ for an N tap filter. Thus the amplitude response is given by Eq. 1.6

$$H_a(\omega) = a_0 + \sum_{lim_l}^{lim_u} a_n Trig(n\omega) \tag{1.6}$$

Table 1.1 shows the form of the amplitude response of linear phase FIR filters for each type of filter. The symmetric filters have a cosine term owing to the addition of the duplicate frequency response terms and the resulting cosine term due to the Euler's formula. Anti-symmetric terms have a sine term due to the subtraction of the duplicate terms.

1.2 Filter Design Methodology and Specifications

Finite Impulse Response filters are preferred over Infinite Impulse Response filter due to their linear phase and guaranteed stability (Table 1.2). The filter design problem can be

Table 1.1: Amplitude Response of Linear Phase FIR Filters

Parameter	Type 1	Type 2	Type 3	Type 4
a_0	$h(M)$	0	0	0
a_n	$2h(M-n)$	$2h(M-n)$	$2h(M-n)$	$2h(M-n)$
lim_l	$n=1$	$n=0$	$n=1$	$n=0$
lim_u	$n=M$	$n=M$	$n=M$	$n=M$
$Trig(\omega, n)$	$cos(n\omega)$	$cos(n\omega+0.5)$	$sin(n\omega)$	$sin(n\omega+0.5)$

specified by a 4-tuple *fspec* (Fig. 1.1):

$$fspec = (\omega_p, \omega_s, \delta_p, \delta_s)$$

where ω_p is the passband cutoff frequency, ω_s is the stopband cutoff frequency, δ_p is the maximum allowable passband ripple error and δ_s is the maximum allowable stopband ripple error. The error specifications can either be given in decibels or in absolute error. The cutoff frequency can be either given in radians per second or in normalized form. The normalization is done either by diving by π or 2π. If the frequency is given in hertz, than the sampling frequency must also be given. In that case the 2π radians/s correspond to the sampling frequency and π radians/s to half the sampling frequency. Also, due to the sampling theorem, frequencies up to half the sampling frequency can be completely recovered in a sampled signal. Thus, if the specifications require higher frequencies than half the sampling frequency, the sampling frequency must be increased to ensure all the frequencies that are to be dealt with are less than half the sampling frequency.

Upon defining the specification, the filter order is estimated. The MATLAB function "$firpmord.m$" gives a close approximation of the filter order. However, a few iterative designs have to be done till the specifications are met exactly and the filter order is not more than the minimum required for meeting the specifications.

Due to the automatic zeros of linear phase FIR filters, certain type of filters can not be used for designing either highpass or lowpass filters. Table 1.3 summarizes the location of the automatic zeros. Thus, high pass filters can not be designed using Type 2 and Type 3 filters and low pass filters can not be designed using Type 3 and Type 4 filters.

Table 1.2: FIR vs. IIR

Property	IIR Filter	FIR Filter
Phase or group delay	hard to control	linear phase always possible
Stability	Can exhibit unstable behavior and limit cycles	Always stable
Order Required	Small	Long
Implementation	No multirate or polyphase	can be multirate and have polyphase implementations

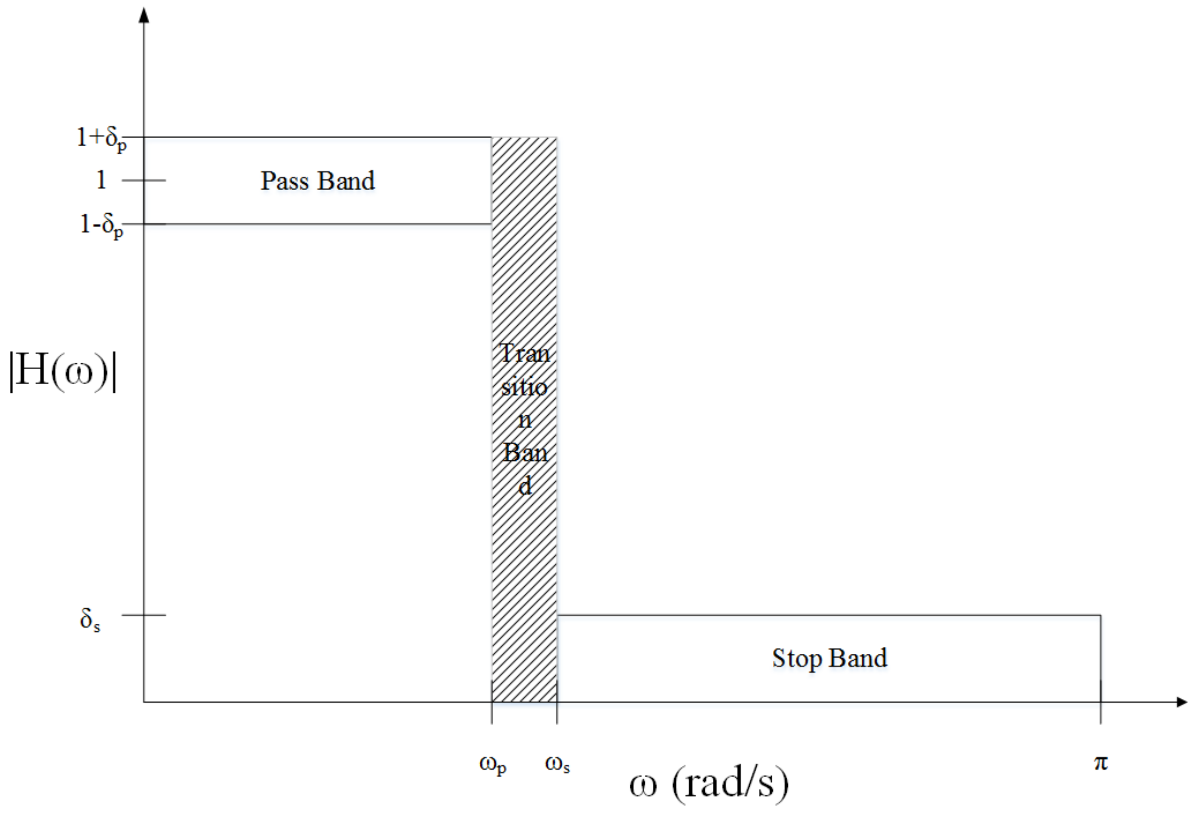


Figure 1.1: Filter Design Specifications

Table 1.3: Automatic Zeros of Linear Phase FIR Filters

Type	Type 1	Type 2	Type 3	Type 4
Zeros	None	$\omega = \pi$	$\omega = 0, \pi$	$\omega = 0$

The optimal filter design problem can be formulated in many ways depending on the objective function. Taking only the frequency into consideration and not the group delay the error can be defined as the L_p norm (Eq. 1.7) of the weighted difference between the amplitude response (Eq. 1.6) and the desired frequency response.

$$||\mathbf{x}||_p = (\sum_{i=1}^{n} |x_i|^p)^{1/p} \tag{1.7}$$

Thus, among the many possible L_p norms, the two cases that are most often used are when $p = 2$ and $p = \infty$. The Least Square design corresponds to $p = 2$ and its objective function is given in Eq. 1.8.

$$\epsilon = \int_0^{\pi} W(\omega)|H_a(\omega) - D(\omega)|^2 d\omega \tag{1.8}$$

where $W(\omega)$ is given in Eq. 1.9 and $D(\omega)$ is given in Eq. 1.10 and Ω_p,Ω_s are the passband and stopband frequency points respectively.

$$W(\omega) = \begin{cases} W_p & \text{if } \omega \in \Omega_p \\ W_s & \text{if } \omega \in \Omega_s \\ 0 & \text{otherwise} \end{cases} \tag{1.9}$$

$$D(\omega) = \begin{cases} 1 & \text{if } \omega \in \Omega_p \\ 0 & \text{if } \omega \in \Omega_s \\ \textit{any number} & \text{otherwise} \end{cases} \tag{1.10}$$

The least square design tries to minimize the energy difference between the desired response and the actual response.

The Minimax or Chebychev design corresponds to $p = \infty$ and its objective function is given by 1.11. As the L_∞ norm corresponds to the maximum value of the vector, the minimax design tries to reduce the maximum difference between the actual frequency response and the desired response (called ripple error).

$$\epsilon = \mathbf{max}\ [W(\omega)|H_a(\omega) - D(\omega)|] \tag{1.11}$$

For a detailed discussion of filter design techniques, the reader is referred to [1]. Also, the applications of digital filters and their implementation techniques are discussed in detail in [2].

1.3 Motivation and Outline of Thesis

Over the past decade portable electronic gadgets running on battery have become ubiquitous. Many new biomedical applications have emerged that require minimal power consumption. Thus, a new paradigm began in design of digital systems; one that accentuated low power design. This paradigm also made its way into digital filter design and recent researches have been focused on designing filters with low computational and hardware complexity.

Digital design techniques have different level of abstractions: system, algorithm, architecture, circuit and device. At each level, design techniques that underline low power consumption or high throughput can be incorporated. Starting from the top low system clocks can reduce power consumption and parallel processing and pipelining can be incorporated to increase throughput. At the other end of the spectrum, the device and circuit level, implementation technology choice, transistor sizing and supply voltage reduction are among many techniques employed for reduction of power consumption. For a DSP designer, the middle level of design i.e. the algorithm and architecture level is of primary concern. Minimizing the number of operations and the hardware required to carry out a given task is the main goal at the algorithmic level and architecture level. Multiplier less digital filters offer a substantial saving in power and area due to the elimination of dedicated multipliers and utilize shift and adds to generate the products.

The current finite word length filter design techniques that minimize hardware suffer from large design times or non optimal results. This thesis aims to develop a design algorithm that can generate a high level filter architecture such that the chip area, the computation cost and the power consumption are minimized. The proposed algorithm produces results competent with the best deterministic methods and also cuts down the run time of the algorithm.

The thesis is organized as follows. Chapter 2 gives the review of the literature present on the design of finite word length digital filters and hardware complexity reduction techniques. The techniques are broadly classified into multiplier less and with multipliers. The techniques with multipliers are briefly reviewed. The multiplier less techniques are reviewed in detail. The sum of power of two designs, the multiple constant multiplication algorithms and their application to filter synthesis and the state of the art techniques in minimal hardware filter designs are given.

Chapter 3 gives an overview of the optimization algorithms used throughout the thesis. The choice of selecting an optimization technique based on the objective is firstly given. Next, the linear programming algorithm and its setup is given. Subsequently, the Differential Evolution algorithm is given in detail along with the variations in the mutation techniques, adaptive control parameters and the discrete variant of the Differential Evolution algorithm proposed in this thesis.

Chapter 4 gives the proposed algorithm for the design of minimal hardware complexity finite word length digital filters. The algorithm is referred to as Differential Evolution Filter Design Optimization (DEFDO) algorithm. The problem is formulated in section 4.1 and the DEFDO algorithm is given. Section 4.2 gives a detailed mathematical and analytical discussion on the DEFDO algorithm. The various techniques employed at enhancing the run time and search of the Differential Evolution algorithm are also discussed.

Chapter 5 gives the design examples and results. Firstly, in section 5.1 the working of the discrete Differential Evolution is analyzed. The analysis is done for the control parameters and the variation in mutation strategy of the algorithm and the test objective function is the minimax error. In section 5.2, the joint optimization of minimax error and hardware complexity is carried out to show the working of the algorithm. Six filters from literature and two special filters have been implemented. An analysis of the filter orders and word length for implementation is given prior to the design examples. Section 5.3 gives the examples of the hardware synthesis at a architecture level of abstraction and for continuity of the design process the full adder counting technique is also discussed. Section 5.4 gives the comparison of the proposed algorithm's result with the state of the art methods present in literature. Lastly, in section 5.5 the analysis of the DEFDO algorithm is given with respect to the algorithm complexity, design time, and design scalability.

Chapter 2

Review of Filter Complexity Reduction Methods

Filtering operation is central to all digital signal processing systems. Design of optimal linear phase FIR filters has always been the focus of attention because of their marked superiority over IIR filters. The design that guaranteed optimality for infinite precision FIR filters was given by Parks-McClellan [3] which is to date the status quo in infinite precision techniques. However, with advances in communication, demands for filters with narrow transition width and high stopband attenuation requiring large orders prompted researchers to develop hardware reduction techniques. This chapter will first give an overview of the filter complexity reduction techniques. The multiplier less techniques will be discussed in detail. A brief overview of techniques that utilize multipliers such as sparse filter designs and frequency response masking approach would also be given.

2.1 Filter Complexity Reduction Technique

Among the notable techniques for reducing computational and hardware complexity of FIR digital filters are: sparse filter design, frequency response masking, multi-rate techniques, and multi stage decomposition. While sparse filters try to minimize the non-zero coefficients, the other techniques involve cascading filters and other circuitry to reduce the hardware complexity. However, for single stage filters, multiplier-less filters have proven to be very effective.

1. Complexity Reduction with Multipliers
    - Recursive running-sum prefilters
    - Cyclotomic polynomial prefilters
    - Interpolated FIR filters
    - Frequency-response masking technique
    - Multirate techniques
    - Multi-Stage decomposition
    - Sparse filter techniques
2. Multiplierless Filter Design

The most researched upon techniques in designs utilizing multipliers are sparse designs and frequency response masking technique designs. They are briefly discussed here and further information on these design techniques can be found in the references.

Sparse Filter Design

Sparse filter design techniques aim at minimizing the non-zero coefficients in a digital filter. A zero coefficient implies no computation cost and thus maximizing the number of zero coefficients or minimizing the number of non-zero coefficients reduce the computational complexity of the digital filter.

A sparse filter design is an l_0 minimization problem (Eq. 2.1).

$$\textbf{minimize} \quad \|\mathbf{x}\|_0 \tag{2.1}$$

$$\textbf{subject to:} \quad |H(\omega) - D(\omega)| \leq \Delta(\omega), \quad \forall\omega \in \Omega_I$$

where $\mathbf{x}$ is the vector containing the unique filter coefficients, $\Delta(\omega)$ is the tolerance in the frequency response and $D(\omega)$ is the desired frequency response, Ω_I is the set bands of interest in frequency and $H(\omega)$ is the amplitude response of the filter.

Among the notable methods for sparse filter design are linear program techniques given in [4] and a WLS relaxation approach is given in [5].

Frequency Response Masking

Lim, [6], introduced the frequency response masking technique for designing filters that had a narrow transition width. Because of sharp frequency characteristics, a single stage design required a huge number of taps for complying with commendable error constraints. In sparse filter design, the filter with is wide transition width is upsampled by replacing the delay element by M delay elements. Thus, the frequency response is a periodic and shrunk version of the initial frequency response curve. The ratio that the frequency response is shrunk by and the number of replicas that are generated depends on the factor M. Subsequently, a mask filter whose transition width is much larger is cascaded to filter out the unwanted replicas of the upscaled filter. Hence, a filter with a transition width of Δ/M is obtained where Δ was the original transition width. Even though the mask involves extra hardware, however, compared to the hardware needed to design the narrow transition width filter on its own there is a substantial saving in the hardware. For a detailed discussion of frequency response masking the reader is reffered to [6] and [7].

2.2 Multiplierless Filter Design

Digital filters implemented in hardware are limited by the finite word length used in its implementation. FIR filters have been the choice for implementation because of their stability in finite wordlength designs. The effects of finite word length on the accuracy of filtering operations has been extensively studied and it was shown that a substantial loss in accuracy takes place in going from infinite precision to finite precision by simple quantization to the nearest value. However, it was noted that the loss in accuracy can be mitigated to a great extent by formulating the optimization problem that took into account the discrete nature of the filter coefficients. Initially, two classes of optimization problems that handled finite word length filters emerged: exact and approximate [8]. The exact methods were based on search techniques that encompassed the entire space while the approximate techniques utilized local search algorithm in the neighborhood of continuous coefficients.

Many design techniques have evolved over time for designing discrete coefficient filters. Among the most notable ones are the signed power of two (referred to as SPT, SOPOT or POT), the Multiple Constant Multiplication and dynamically expanding subexpression space design techniques. The following sections discuss each of the following design techniques and review their strengths and weaknesses.

2.2.1 Design of Discrete Filters in Sum of Power of Two Space

Design of discrete filters was first proposed by Kodek, [9]. He utilized a integer programming package and upscaled the coefficients to fit into the integer subspace. The branch and bound technique was proven to be successful in designing the discrete coefficient filter. However, the gains in the error as compared to the simply quantized optimal infinite precision designs were not substantial. Thus, the amount of time required for design outweighed the gain in the error.

To justify such long design times Lim *et al*, [10] proposed the design of the discrete filter in the power of two subspace. Since a power of two did not require any hardware for its implementation as it could be generated simply by hardwired shifts, the substantial reduction in hardware complexity proved very attractive. In Lim's design each coefficient was represented as a sum of two powers of two (Eq. 2.2).

$$h(n) = \sum_{i=1}^{L} S_i \times 2^{g_i(n)} \tag{2.2}$$

where $L = 2$, $S_i(n) \in -1, 0, +1$ and $-9 \leq g_i(n) \leq 0$. The method was suitable for designing filter of length upto 40 using the computing resources available at that time. The paper paved the way for future research in discrete filter optimal designs.

In [11], the filter design objective was modified to the Normalized Peak Ripple Magnitude. This was owed to the fact that a filter's purpose is to alter an incoming signal to allow one set of frequencies while attenuating another set of frequencies. Thus, a gain of unity was inconsequential. However, a constraint had to be set on the passband gain. A high gain meant a better signal to noise ratio but it also caused overflow. In the discrete power of two space, the upper bound and lower bound of the passband gain differed by two i.e. the lower bound was always half the upper bound. This was because any other gain could always be represented in the given range by multiplying or diving by a suitable power of two and the NPRM would remain the same. To find a coefficient set with the optimal NPRM, passband gain sectioning was introduced. In the gain sectioning technique, the gain range was divided into many fine gain sections and a discrete filter was designed using the upper bound and lower bound constraints of the section on the gain. A simple technique was to section the range $[0.7, 1.4]$ and select the best solution found among all the sections. A trade off was met between design quality and design time as more sections led to a better design but with increased design time. A more involved technique based on elimination was also given.

Samueli, [12], used a Canonic Signed Digit representation to represent the discrete FIR filter coefficients. A canonic signed digit code a special representation of the non unique signed digit code whereby the number of non zero digits is minimal and no two adjacent digits are non zero. Since, an adder and a subtractor have the same hardware complexity, the CSD form provides the most efficient representation scheme exhibiting 33% fewer non zero terms than the 2-Complement representation. The CSD code for a number is unique. Due to the non uniform spreading of the CSD codes using L power of two terms (as opposed to the uniform 2s complement) and the fact that the density of the codes being much higher among the small valued coefficients, an extra non zero digit was used to represent the filter coefficients having a magnitude of 0.5 or greater. The justification given was that the addition of an extra term improved the frequency response considerably while only having

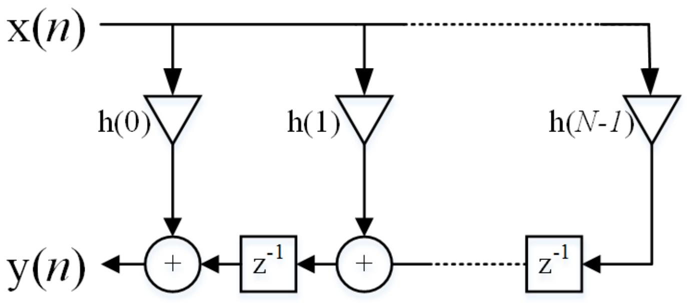


Figure 2.1: Transposed Direct Form

a mild affect on the hardware complexity. Also, since impulse responses of low pass filters exhibit a $sinx/x$ sort of response, the percentage of large coefficients was small. The design algorithm first determined the scaling factor and then ran a local bivariate neighborhood search around the scaled and rounded coefficients.

In [13], a SPT term allocation scheme was developed where each coefficient is allotted different number of SPT terms based on its sensitivity to the frequency response. After assigining the SPT terms, an integer-programming algorithm was used to optimize the coefficient values. This new technique produced designs that had a much better NPRM as more emphasis was laid on coefficients with larger magnitudes.

2.2.2 MCM Algorithms

Bull and Horrocks, [14], introduced filtering operation as a multiple constant multiplication problem of the form of Eq. 2.3 by using the transposed direct form of the FIR filter structure (Fig. 2.1) where each output $y(i)$ is the output of the corresponding multiplier. They argued that the multiplication is the bottleneck of the entire operation in terms of both chip area and throughput. They noted that an array multiplier has an area of complexity of the order of $\mathbb{O}(B^2)$ where B is the word length whereas an adder has an area complexity of the order of $\mathbb{O}(B)$. Thus reducing the implementation cost of the coefficient multiplier was a prime goal to reducing the overall implementation cost. For fixed coefficient filters, the redundancies among the coefficients could be eliminated and

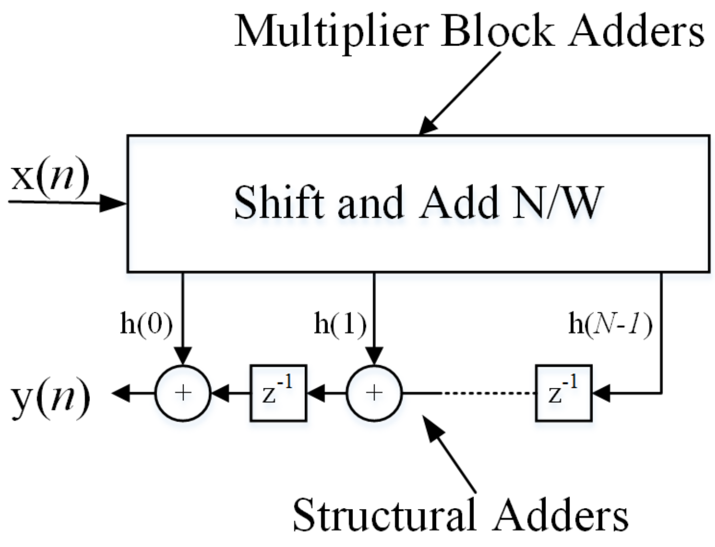


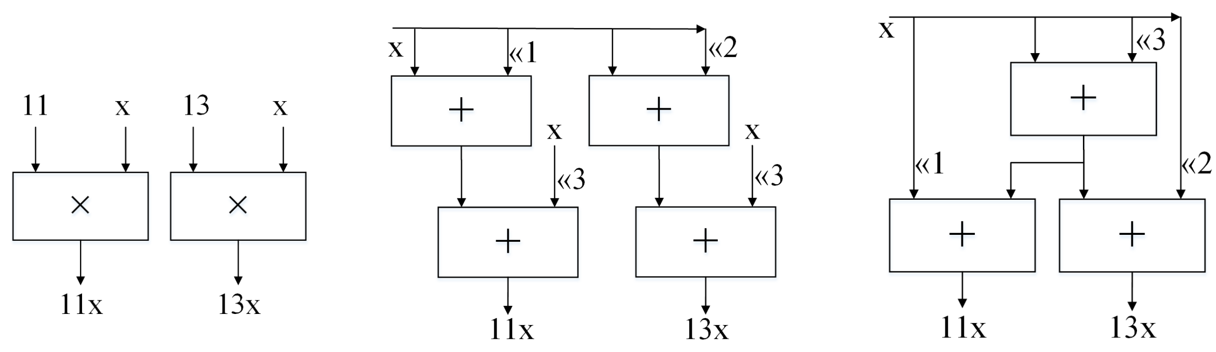
Figure 2.2: Shift and Add Network

the partial results generated could be utilized for multiple coefficients generation. This work paved the way for further research in Multiple Constant Multiplication algorithms as a means for implementing filtering and other DSP transforms.

$$y(i) = h(i)x \quad i = 0, 1 \cdots N \tag{2.3}$$

In defining the fixed coefficient filtering operation as a Multiple Constant Multiplication problem, the multipliers in the transposed direct form filter structure are replaced by a Shift and Add Network (SAN). The adders used for implementing the coefficients inside the SAN are referred to as Multiplier Block Adders (MBAs). The adders summing the delayed and weighted input are referred to as Structural Adders (SA). The MCM optimization problem has been widely researched and is proved to be NP-complete problem justifying the use of heuristic algorithms for solving the problem [15]. Figure 2.3 shows the comparison of the three approaches to implementing the filter coefficients.

Two classes of algorithms exist for solving the MCM problem: Common Subexpression Elimination (CSE) [16], [17] and Graph Based (GB) [18]. The Common Subexpression



(a) Multiplier Approach (b) SOPT Approach (c) MCM Approach

Figure 2.3: Filter Design Approach

Elimination algorithms first define the constants to be multiplied in a number representation e.g. Binary, CSD, or Minimal Signed Digit. After that the common subexpression present in the numbers are obtained. The most common subexpression is used for sharing among the coefficients. The graph based technique do not utilize any specific number representation and represent the adder network as a graph and construct child tree branches from the parent branches. The RAG-n algorithm [18] is the most notable graph based algorithm. It consists of 2 parts: exact and heuristic. The exact part of the algorithm outperforms all CSE algorithms, however its applicability to a given set of numbers is restricted. It can only synthesize constants that can be represented as a sum of two other coefficients using one adder and there exists at least one constant which is a cost one constant [19]. The CSE algorithms have the disadvantage that they are specific to a number representation and only search the common subexpression space. The filter design technique proposed in this thesis utilizes the exact part of RAG-n algorithm and is thus explained in detail.

Reduced Adder Graph-*n* Algorithm

The Reduced Adder Graph-n (RAG-n) algorithm, [18], given by Dempster *et al* is a graph based MCM algorithm. The following definitions are used in the paper:

Definition: *Adder Cost*
The adder cost of a set of constant integers is the number of adders and subtracters required to perform multiplication by all those constants.
Definition: *Fundamentals*
The intermediate or final odd values (the vertices of the graph) used in synthesizing the shift and add network.

The RAG-n algorithm utilizes the lookup tables generated by the MAG algorithm [19]. One lookup table gives the adder cost of multiplying by an integer and the other lookup table gives the different set of fundamentals that can be used to implement the constant multiplication optimally. The steps of the algorithm are enumerated for generating a shift and add network of constants with word length B:

1. Obtain the odd fundamentals from the set by taking the odd numbers and diving even numbers until an odd number results and delete the repeated numbers. Call the set "*incomplete set*"
2. Find among the "*incomplete set*", cost one fundamentals form the lookup table.
3. Add the cost one fundamentals to the *"graph set"* and remove them from the "*incomplete set*".
4. Examine pairwise sums of the form $(a\times 2^i \pm 1)$ or $(a\times 2^i \pm b)$ where $a, b \in$ "*incomplete set*" and i is an index that is varied from $0, 1, \cdots B$. If any coefficient from the "*incomplete set*" is found, remove it from the "*incomplete set*" set and add it to the *"graph set"*.
5. Repeat until no more coefficients remain in the "*incomplete set*".

The enumeration gives the exact part of the RAG-n algorithm. The following theorem were proved in the paper and also provide useful insight into applicability of the exact part of the algorithm:
Theorem 1: A set of n non repeated odd coefficients which each have a single coefficient cost can not be synthesized using fewer than n adders.

Theorem 2: For a set to incur the minimal adder cost of n, at least one cost-1 coefficient must be in the set.

The exact part of the algorithm may not always synthesize and a supplemental heuristic algorithm is given in the paper. However, since in this thesis only the exact part is utilized, the heuristic part is not explained.

2.2.3 State of the Art

The MCM problem was limited in the sense that it only optimized the already synthesized coefficients. However, a coefficient set complying with a particular design constraint is not unique. Thus, a coefficient set which in the representation sense was not minimal could possibly have a better hardware implementation than the one with the minimal bit representation. This problem was circumvented by including the synthesis of the shift and add network in the optimization of the discrete coefficient representation. The latest research in multiplierless design is focused on cutting the design time while maintain optimum performance level. Since the exact techniques, which employ mixed integer linear programming (MILP) branch and bound or tree search algorithms such as width recursive depth first search, have exponential growth in complexity with the filter order thus they cannot be used for designing higher order filters without proper pruning techniques.

A few new terms were defined in [20] for the discrete filter design problem. The concept of bases or fundamentals was already in use in MCM designs, but [20] extended the definition to include subexpression space. The definitions are repeated here and used throughout the thesis.

Definition: *Basis Set*

In the synthesis of the *SAN*, the intermediate or final odd values are called fundamentals or bases. The set of all the bases needed to synthesize the SAN is the basis set.

Definition: *Subexpression Space*

The set of all possible bases is called the subexpression space. Mathematically, a discrete subexpression space is defined as

$$n = \sum_{i=0}^{T-1} s_i 2^{m_i} \tag{2.4}$$

where m_i is a non-negative integer and $s_i \in S$ and S is the basis set. For the SOPT space, the basis set is $S = \{-1, 1\}$. Other examples of basis set are $S_1 = \{0, \pm 1, \pm 3, \pm 5\}$.

Definition: *Contiguous Basis Set*

A basis set if said to be contiguous if it contains all contiguous odd integers till the largest odd integer present.

Definition: *Order of Basis Set*

The order of a basis set is defined as the number of adders required to construct the basis set.

Branch and Bound MILP

The linear phase FIR filter design problem for continuous coefficients can be modeled as a linear program. Thus, it can be solved easily using polynomial time algorithms for linear programming. However, if the coefficients are forced to take discrete values than the problem becomes NP-complete. By upscaling the coefficients, the problem can be modeled as a Mixed Integer Linear Program. To tackle the problem, a linear relaxation approach is developed whereby the discrete constraint is dropped and the problem is solved as a linear program. However, simply quantizing the solution of the relaxed problem can result in the solution lying very far away from the optimal discrete solution. Thus, a branch and bound technique is developed to systematically solve relaxed linear programs and further branches are created or fathomed depending on the feasibility of the solution of the relaxed problem.

The branch and bound MILP is used in [20] to find the optimal coefficients from a fixed subexpression space. An upper cap on the number of adders is obtained and the frequency response ripple is minimized. Firstly, a continuous solution is obtained using linear programming. Among the coefficients a coefficient, say x_i, is selected for branching. If the optimal value of x_i is 5.4, than two sub-linear programs are created $L1$ and $L2$ with the following bounds on x_i are $x_i \geq 6$ and $x_i \leq 5$ respectively. The depth first search progresses by keeping $L2$ aside and exploring $L1$ further. Another coefficient, say x_j, is chosen and the bounds of $x_j \geq \lceil x_j \rceil$ and $x_j \leq \lfloor x_j \rfloor$ are imposed giving rise to subproblems $L3$ and $L4$. $L4$ is kept aside and $L3$ is further explored. The process continues till all the coefficients have been fixed and a discrete solution is obtained. Thus that branch is terminated and the algorithm back traces to solve the adjacent problem. A node is also terminated if it is seen that an optimal solution is not possible upon further exploration.

The algorithm suffered from the following problems:

1. *Prefixing the Basis Set:* The set of numbers and the space to search for the optimal filter coefficients was fixed *apriory*. Thus, the choice limited the search space and hence the optimal solution. A lot of effort had to be made to define a basis set that would best serve the problem at hand.

2. *Determining the closest approximation:* The closest number that can be represented in the subexpression space to serve as the bounds of the linear program was to be determined. An exhaustive search or a greedy algorithm was utilized which slowed down the process. A lookup table could be generated for static subexpression spaces which cut short the time consumed in the search process.

3. *Tradeoff Between Order of Basis Set and Number of Terms Per Coefficient:* A trade-off had to be met whereby choosing a small order basis set meant using more number of terms to construct a coefficient or choosing a large order basis set and using less number of bases for constructing the coefficients.

The use of MILP to solve the discrete coefficient linear phase FIR filter design problem was based on first defining a basis set based on which a sub space was created. Thus, an upper cap was done on the order of basis set and the number of terms used to construct the basis set. Based on this upper cap, the coefficients were found that met frequency response specifications. Hence, the hardware reduction problem was not dealt with in a direct manner and rather the upper cap on hardware was established. Also, the design procedure involved a lot of designer experience and design automation was marginalized.

Dynamically Expanding Subexpression Space Design

The optimality of the basis set can be obtained by dynamically expanding the basis set based upon the need for discretizing the coefficients starting from the trivial basis set $0, \pm 1$. In this method, the coefficients that are likely to result from the usage of less adders are discretized first. However, for solving the problem in a dynamically expanding subexpression space, the branch and bound MILP cannot be used and hence in[21], [22], [23] MILP with depth first width recursive search is used. In this method, a coefficient is selected for discretization and it is branched to L different branches with the L closest discrete values selected in each branch. With one coefficient fixed, the rest of the coefficients are again optimized and another coefficient is selected for discretization. Again, a set of L branches are formed. In the depth first search, only one among the L branches at each stage is selected for further branching. When, all the coefficients have been discretized, then the solution is stored and the algorithm back traces and solves the other branches at the next upper level. Also, for expediting the search process, the solution of a branch is compared with the current best obtained solution and if it cannot offer a better solution, it is not further explored.

The complexity of the above algorithm depends highly on the number of braches (L) that are created at each stage. In [21], L branches are created such that only one adder must be used in the synthesis of the discrete value from the already existing subexpression space and these L values are the closest to the continuous coefficient value. In [22], an exhaustive search is made where all the possible discrete values are selected for branching based upon a feasible range of that coefficients (calculated beforehand). Thus, the complexity of the depth first width recursive search is exponential with L.

In [24], a two-step optimization process is proposed. In this method, firstly a set of initial acceptable solutions is obtained for different passband gains using a polynomial time algorithm (setting $L = 1$). The coefficients are synthesized using an MCM algorithm [16]. In the second stage, the coefficients are grouped into small and large coefficients. The large group of coefficients is further optimized for low hardware cost using a dynamically expanding subexpression space search technique with the initial basis set being the set needed to synthesize the smaller group of coefficient.

Genetic Algorithm Based Filter Design

The paper [25] uses a genetic algorithm for designing the multiplier-less filter. In their algorithm, a reduced search space is created around a base solution. The base solution is obtained by discretizing a continuous solution for a corresponding gain. Also, they have encoded the difference between the possible values the GA can take to the base solution as the value the chromosomes of the GA represent leading to a shorter encoding scheme. Thus, they have managed to reduce the search space and expedite the search process of the GA. Also, the mutation and crossover operations have been modified and the rates made adaptive.

The number of adders in the implementation has been set as the objective to minimize. They make use of the RAG-n [18] algorithm for determining the number of adders. Since for each passband gain the search process is independent, they have cast each problem to a different machine and ran in parallel. However, since the RAG-n algorithm consumes a substantial time, they formulate a fitness function that inhibits the algorithm for running the RAG-n algorithm for solutions that do not meet the error constraints. For a filter order of 324, their design time is around 3h49m when casted onto 20 machines or 37h9m when completed on a single machine.

Summary

In this chapter the techniques that reduce the implementation complexity of digital filters are discussed. Techniques such a sparse designs and frequency response masking approach were briefly reviewed. The multiplier less techniques such a sum of power of two and MCM algorithms' utilization for filtering operation was discussed. The state of the art methods were also examined. Designs utilizing dynamically expanding subexpression space have seen a surge in recent literature on deterministic algorithms for finite word length digital filter designs. Also, heuristic algorithms such as genetic algorithm and local search which have been used for design of digital filters were reviewed.

Chapter 3

Optimization Methods

Optimization is the process of minimizing the cost or maximizing the gain of a process or function. Optimization is performed in every field from economics to engineering. The technique to approach a particular type of problem depends on the characteristics of the problem whether the function is linear or nonlinear, convex or non-convex, constrained or unconstrained. Also, a function maybe multi-objective where a number of objectives need to be optimized simultaneously or multimodal where multiple equally good solutions exists. The problem may be defined on the set of real number whereby the parameters are continuous, or on a discrete set of number where the parameters take discrete values or the problem may be mixed where some parameters can take continuous values while others can only take discrete values.

The filter design problem is also modeled as an optimization problem in numerous ways depending on the objective e.g. minimax design, least square design, least p-th design etc. The gist of all filter design optimization problems is to reduce the weighted ripple error (Eqs. 1.8, 1.11). In this chapter, the various optimization algorithms used in the proposed algorithm are discussed. Firstly an overview of the types of optimization algorithms available is discussed along with paramters to measure computational complexity. Next, the linear programming algorithm and its setup is given. After that the Differential Evolution Algorithm is discussed. Lastly, the variations of the Differential Evolution Algorithm are examined.

3.1 Selection of Optimization Method

The theory of continuous parameter optimization has been developed for centuries and is at present very mature. Efficient algorithms exist for linear problems with linear inequality and equality constraints. Quadratic unconstrained problems can be solved using gradient based methods such Quasi Newton algorithm. A detailed discussion on optimization can be found in [26].

While continuous parameter optimization is tractable however, situations arise when the parameters can take values only from a finite set. If the finite set is the set of integers, the problem is modeled as integer programming or if they belong to a mixed set consisting of real numbers and integers then mixed integer programming models are used provided the constraints are linear. However, if the constraints follow no general rule then the problem is generally classified as a combinatorial optimization problem. In this problem, the possible solution is a combination from a finite set of points with the objective function and the constraints taking any form. The problems in combinatorial optimization are ranked based on the computational complexity.

Definition: Computational Complexity of Algorithms
The complexity of an algorithm to solve a given problem is generally specified by the worst case scenario. The $\mathbb{O}$ notation is used to denote the complexity which is defined as

$$f(n) = \mathbb{O}(g(n)) \tag{3.1}$$

if there exist positive constants a and b such that $\forall n > a, f(n) \leq b.g(n)$

Among the set of problem classified based on order of complexity two special cases are highly discussed: polynomial time and exponential time. A polynomial complexity is defined as $\mathbb{O}(p(n))$ where $p(n)$ is a polynomial in n of degree k i.e. $\mathbb{O}(n^k)$. On the other hand, an exponential complexity is when the computation take the complexity of order

$\mathbb{O}(b^n)$,$b > 1$. Problems for which polynomial time algorithms exist for solving them are called tractable and if a polynomial time algorithm does not exist than they are intractable.

Two classes of algorithms exist for solving combinatorial optimization problems :

1. Exact Algorithms
2. Approximate Algorithms

The exact methods include branch and bound algorithms, graph based tree search (breadth first or depth first) while approximate methods include approximation algorithms, heuristics and metaheuristics. The choice of algorithm for approaching a certain problem is dependent on whether it is a P problem or not. A P problem is a problem for which a polynomial time algorithm exists for solving it. An NP problem is one whose solution can be verified in polynomial time. A $NP-Hard$ is problem which is as hard as the hardest NP problem. $NP-Hard$ problems which are also NP are known as $NP-Complete$ problems. Deterministic algorithms should be used if the problem is P and use of heuristics is unjustified. Also for $NP-Complete$ problems, the instance of the problem must be considered as small instances can solved easily by exact algorithms. The justifiable use of heuristics is in large instances of $NP-Complete$ problems. If, however, the polynomial time algorithm has high order index than use of heuristics might be favored.

3.2 Linear Programming

Linear Programming is a method for optimizing a linear functions subject to linear equalities and inequalities. The generalized linear programming problem can be defined as: Find $\mathbf{x} \in \mathbb{R}^N$ such that the function

$$func(\mathbf{x}) = \mathbf{f}^T\mathbf{x} = \sum_{i=1}^{N} f_i x_i$$

where $\mathbf{f} \in \mathbb{R}^N$, is minimized and is subject to M inequality constraints

$$a_{11}x_1 + a_{12}x_1 \cdots + a_{1N}x_N \leq b_1$$

$$a_{21}x_1 + a_{22}x_1 \cdots + a_{2N}x_N \leq b_2$$

$$\vdots$$

$$a_{M1}x_1 + a_{M2}x_1 \cdots + a_{MN}x_N \leq b_M$$

and P equality constraints

$$c_{11}x_1 + c_{11}x_1 \cdots + c_{1N}x_N \leq d_1$$

$$c_{21}x_1 + c_{22}x_1 \cdots + c_{2N}x_N \leq d_2$$

$$\vdots$$

$$c_{P1}x_1 + c_{P2}x_1 \cdots + c_{PN}x_N \leq d_P$$

and the following bounds on the variables

$$x_{1l} \leq x_1 \leq x_{1u}$$

$$x_{2l} \leq x_2 \leq x_{2u}$$

$$\vdots$$

$$x_{Nl} \leq x_N \leq x_{Nu}$$

Alternatively, a dual problem can also be defined which maximizes the function. The optimal point, however, is the same for both the problems. The minimization problem is explained as most available linear programming solvers utilize the minimization form.

Table 3.1: Linear Program Setup

$\min \quad \mathbf{f}^T\mathbf{x}$	
$\mathbf{A}_{ieq}\mathbf{x} \leq \mathbf{b}_{ieq}$	Inequality Constraint
$\mathbf{C}_{eq}\mathbf{x} = \mathbf{d}_{eq}$	Equality Constraint
$\mathbf{x}_l \leq \mathbf{x} \leq \mathbf{x}_{ieq}$	Bounds on Variables

Rewriting the constraints in matrix form

$$\mathbf{A}_{ieq} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1N} \\ a_{21} & a_{22} & \cdots & a_{2N} \\ \vdots & \vdots & \vdots & \vdots \\ a_{M1} & a_{M2} & \cdots & a_{MN} \end{bmatrix}_{M\times N}$$

$$\mathbf{b}_{ieq} = [b_1, b_2, \cdots b_M]^T$$

$$\mathbf{C}_{eq} = \begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1N} \\ c_{21} & c_{22} & \cdots & c_{2N} \\ \vdots & \vdots & \vdots & \vdots \\ c_{P1} & c_{P2} & \cdots & c_{PN} \end{bmatrix}_{P\times N}$$

$$\mathbf{d}_{eq} = [d_1, d_2, \cdots d_P]^T$$

Table 3.1 summarizes the linear program setup which is the most used format in most available solvers (e.g. MATLAB). A linear programming problem's constraints can be viewed as a polyhedron with the vertices being one of the basic feasible points for the optimal point. The simplex method is very popular in solving linear programs. However, for large problems, the interior point method is more efficient. The interior point method utilizes non-linear algorithms modified for the linear problems. The simplex method searches along the boundary of the polyhedron while interior point methods approach from inside or outside the feasibility polyhedron. Many authors have proposed techniques for solving problems.

The OPTI Toolbox, [27] is an open source collection of optimization algorithms made available. The CSDP Algorithm by Borchers, [28], is utilized in this thesis for solving filter design problems.

3.3 Differential Evolution Algorithm for Continuous Optimization

Among the class of global optimization algorithms, Evolutionary Algorithms (EA) are metaheuristic algorithms. The strategies employed in EAs are inspired by biological phenomenon of evolution and survival of the fittest. Among the notable techniques are Genetic Algorithm (GA) and Differential Evolution (DE). EAs create a random population of parameter vectors in the search space representing all the landscape. The population members interact with each other through operators such as crossover, mutation, and new generations are produced using selection. Closely related to EAs are algorithms utilizing swarm intelligence e.g. Particle Swarm Optimization (PSO), Ant Colony Optimization (ACO) etc.

Given an objective function with D parameters to be optimized. The Differential Evolution algorithm begins by creating a population of N_p D-dimensional random vectors. The simple DE employs three operators to evolve from one generation to the other: mutation, crossover and selection (Figure 3.1).

Initialization

The population is initialized by selecting the values of the parameters so that they cover the entire range. If the upper and lower bounds of each parameter are given in vector b_u

Figure 3.1: Pseudo Code for the Differential Evolution Algorithm

```
process DE
1   initialization
2       do
3           mutation
4           crossover
5           selection
6       while   (termination criteria)
end process
```

and b_l respectively, then the values are generated according to Eq. (3.2).

$$x_j^{p,1} = b_{l,j} + r_j(b_{u,j} - b_{l,j}) \tag{3.2}$$

where $r_j \in (0,1)$ is a random number and $j = 1, 2, D$. The first generation ($g = 1$) population is created by repeating the above process N_p times. Let each member of the population be represented as in Eq. (3.3).

$$\mathbf{x}^{p,g} = [x_1^{p,g}, x_2^{p,g}, ...x_D^{p,g}] \tag{3.3}$$

where $p = 1, 2, \rightarrow N_p$ and $g = 1, 2, g_{max}$. Subsequently, the population can be represented as a N_pD matrix (Eq. (3.4)).

$$\mathbf{P}^g = [x^{1,g}, x^{2,g}, ...x^{N_p,g}]^T \tag{3.4}$$

Mutation

The mutation operator produces a population of trial vectors. For each member of the population, called target vector of index p, the trial vector $\mathbf{u}^{p,g+1}$ is generated using Eq. (3.5).

$$\mathbf{u}^{p,g+1} = \mathbf{x}^{r1,g} + F.(\mathbf{x}^{r2,g} - \mathbf{x}^{r3,g}) \tag{3.5}$$

The indices $r1, r2, r3$ are all distinct and also different from the target index p. $F \in (0,1)$ is a scaling factor and is one of the control parameters.

Crossover

The crossover operator combines parameters from the target vector $\mathbf{x}^{p,g}$ and the trial vector $\mathbf{u}^{p,g+1}$ to generate the mutant vector $\mathbf{y}^{p,g+1}$. The parameters of the mutant are selected according to Eq. (3.6).

$$y_j^{p,g+1} = \begin{cases} u_j^{p,g+1} & \text{if } r \leq CR \text{ or } j = jrand \\ x_j^{p,g} & \text{if } r > CR \end{cases} \tag{3.6}$$

for $j = 1, 2, D$ and $r \in (0,1)$ is a random number and $jrand$ is a randomly selected integer between 1 and D (both including). $CR \in [0,1]$ is the crossover probability and it reflects the amount of information that is inherited from the trial population. A random number index, $jrand$, is also checked for to ensure that the target vector is not replicated.

Selection

The selection operator is responsible for creating new generation from among the trial vectors. For each target vector, if the objective function value is less corresponding to the mutant vector, then the mutant vector replaces the target vector in the population. Thus, selection can be done according to Eq. (3.7).

$$\mathbf{x}^{p,g+1} = \begin{cases} \mathbf{x}^{p,g} & \text{if } f(\mathbf{x}^{p,g}) \leq f(\mathbf{y}^{p,g+1}) \\ \mathbf{y}^{p,g+1} & \text{otherwise} \end{cases} \tag{3.7}$$

The termination criteria for the DE can be the tolerance in the objective function, or can be a maximum number of generations g_{max}.

3.4 Variations of Differential Evolution

3.4.1 Variation in Mutation

The DE algorithm described in section 3.3 is the most basic form. It is denoted as *DE/rand/1/bin* as the base vector $\mathbf{x}^{r1,g}$ in Eq. 3.5 is randomly chosen and there is only one differential term $F.(\mathbf{x}^{r2,g} - \mathbf{x}^{r3,g})$ and the mutation operator exhibits a binomial distribution. There is another form of mutation called exponential but for brevity its definition is omitted. Storm and Price, [29], have also given other variations which are described below. They all differ in how the mutation operator takes place. Thus, the mutation equation for each is given.

1. *DE/best/1/bin*

$$\mathbf{u}^{p,g+1} = \mathbf{x}^{best,g} + F.(\mathbf{x}^{r2,g} - \mathbf{x}^{r1,g}) \tag{3.8}$$

2. *DE/best/2/bin*

$$\mathbf{u}^{p,g+1} = \mathbf{x}^{best,g} + F.(\mathbf{x}^{r2,g} - \mathbf{x}^{r1,g}) + F.(\mathbf{x}^{r4,g} - \mathbf{x}^{r3,g}) \tag{3.9}$$

3. *DE/rand/2/bin*

$$\mathbf{u}^{p,g+1} = \mathbf{x}^{r1,g} + F.(\mathbf{x}^{r2,g} - \mathbf{x}^{r3,g}) + F.(\mathbf{x}^{r4,g} - \mathbf{x}^{r5,g}) \tag{3.10}$$

4. *DE/tar2best/1/bin*

$$\mathbf{u}^{p,g+1} = \mathbf{x}^{i,g} + F.(\mathbf{x}^{best,g} - \mathbf{x}^{i,g}) + F.(\mathbf{x}^{r2,g} - \mathbf{x}^{r1,g}) \tag{3.11}$$

3.4.2 Adaptive Control Parameters

The DE algorithm has 3 control parameters viz., the scaling factor F, the crossover probability CR, and the population size P. While the choice of P can affect the quality of the solution, it does not influence the convergence of the algorithm. The other two parameters influence the convergence and must be properly set with trial and error. Brest *et al*, [30], have proposed an adaptive control parameter adjustment technique. With slight modifications to their results, equations 3.12 and 3.13 are obtained to be used in the discrete DE.

$$F^{p,g+1} = \begin{cases} F_l + r_1 * (F_u - F_l) & \text{if } r_2 < \tau_1 \\ F^{p,g} & otherwise \end{cases} \tag{3.12}$$

$$CR^{p,g+1} = \begin{cases} CR_l + r_3 * (CR_u - CR_l) & \text{if } r_4 < \tau_2 \\ CR^{p,g} & otherwise \end{cases} \tag{3.13}$$

where $r_1, r_2, r_3, r_4 \in (0, 1)$ are random numbers, $\tau_1 = \tau_2 = 0.1$ and F_l, F_u and CR_l, CR_u are the lower and upper bounds of F and CR. The values used are $F_l = 0.4$, $F_u = 1$, $CR_l = 0.05$ and $CR_u = 0.2$.

3.5 Differential Evolution for Discrete Filter Optimization

The original Differential Evolution algorithm is used for real valued parameter optimization. Due to its superior convergence characteristics, researchers have attempted to utilize the differential approach for discrete parameter optimizations. Strategies based on truncation of the parameters for calculating the objective function value were proposed initially. Other techniques involved defining the population member as a permutation of numbers. However, the essence of the DE being the differential operator for mutation was not followed in these methods.

Liu *et al*, [31], have proposed a set based approach for solving the Travelling Salesman Problem using a discrete variant of the DE algorithm which is inspired by similar approach for the Particle Swarm Optimization (PSO) is given in [32]. The PSO utilizes a fuzzy set and crisp set technique to represent the velocity and position of a particle. The DE also uses sets to represent parameters. The DE implements the mutation operation by using four operators defined on the sets. In the crossover, the learn procedure is utilized whereby the trial learns either from the target or the trial vector to obtain the mutant vector. In both the papers a complex representation of the parameters for population generation and evolution is given. Also, the concept of Hamiltonian circuit constraint satisfiability further accentuates the complexity. However, such complex representation is not needed for the filter design problem as each parameter can have only a single value and is selected from a discrete set.

A new discrete Differential Evolution algorithm inspired by the set based algorithms is proposed for the finite word length filter design. The algorithm utilizes a bit string and integer population and a decode scheme is implemented for converting the one from the other. Consider a function $f(\mathbf{x})$ defined over the vector $\mathbf{x} = [x_1, x_2...x_D]^T \in \mathbb{Z}^D$ where D is the number of parameters to be optimized and $\mathbb{Z}$ is the set of integers. This form of the vector is employed in this thesis as the filter coefficients have been scaled up by multiplying with a suitable power of two such that they can be represented as integers. So, the minimization of the function is carried out in the integer subspace and each parameter represents the unfixed filter coefficient.

To find the optimal discrete value of the filter coefficients, the mutation, crossover and selection operators of the Differential Evolution are applied. For the crossover and selection operators the equations given in section 3.3 (Eq. 3.6, 3.7) can be applied directly and the resulting output would lie in the integer subspace. However, the mutation equation (Eq. 3.5) will yield a result that lies outside the integer subspace. Thus, for the mutation operator, each parameter is encoded into a bit string consisting of a total of N bits for the D parameters. The actual encoding and bit allocation scheme is implemented taking into account the problem to be solved. For the finite word length filter design problem it is discussed later with the proposed algorithm. To summarize the scheme, the mutation operation is performed on the N bits and is explained in the following paragraphs. The crossover operation is performed on the D parameters and is the same as discussed in section 3.3.

To obtain the trial vectors Eq. 3.5 is redefined and each of the N bit position (b_i) is determined according to Eqs. 3.14,3.15,3.16.

Figure 3.2: Pseudo Code for Discrete Differential Evolution

```
process discrete DE
1   generate population of P members $\in \mathbb{Z}$ with D parameters
2   encode each D parameter as a bit string
3   g=0
4   do {
5     for $i = 1 : P$
6       for $j = 1 : N$
7         mutation
8       end for
9       decode from bit to integer
10        for $j = 1 : D$
11          crossover
12        end for
13        for $j = 1 : D$
14          selection
15        end for
16      end for
17      g=g+1 }
18  while $(g < g_{max})$
output: best solution
end process
```

$$b_i^{\mathbf{d}} = \begin{cases} b_i^{x^{r2}} & \text{if } b_i^{\mathbf{x}^{r3}} = 0 \\ 0 & \text{if } b_i^{\mathbf{x}^{r3}} = 1 \end{cases} \tag{3.14}$$

$$b_i^{F.\mathbf{d}} = \begin{cases} b_i^d & \text{if } rand(0,1) < F \\ 0 & otherwise \end{cases} \tag{3.15}$$

$$b_i^{\mathbf{u}} = \begin{cases} 1 & \text{if } b_i^{\mathbf{x}^{r1}} = 1 \\ b_i^{F.d} & otherwise \end{cases} \tag{3.16}$$

where $i = 1, 2, ...N$ and $rand(0, 1) \in (0, 1)$ is a random number.

Figure 3.2 shows the pseudo code for the proposed discrete Differential Evolution algorithm. The algorithm is utilized along with adaptive control parameters (section 3.4.2) for designing finite word length filters (Chapter 4). The discrete Differential Evolution algorithm presented can also be used for optimizing functions where the coefficients are independent of each other and belong to the set of integers.

Summary

In this chapter, the selection of a optimization routine for a particular problem and the measures of computational complexity were discussed. After that the linear programming algorithm for solving linear objective function with linear inequality and equality constraints was given. Lastly, the Differential Evolution algorithm and its variations were given. Lastly, the proposed discrete Differential Evolution algorithm adapted for the finite word length filter design problem was given. The methods discussed in this chapter are utilized in the proposed algorithm.

Chapter 4

Proposed Algorithm

Chapter 2 discussed the various techniques for reducing the hardware complexity of finite word length digital filters. Among the multiplierless techniques, the sum of power of two design was successful in reducing the hardware complexity. However, large orders were needed to meet stringent error constraints. The multiple constant multiplication algorithms gave a new approach and focused on reducing the redundancy among the coefficients to reduce the hardware complexity. However, the MCM did not take into account the design procedure of the filter coefficients. The latest techniques jointly find the filter coefficients and reduce the hardware. However, among the state of the art algorithms present, the deterministic MILP based algorithms have an exponential computational complexity. The heuristic methods are fast but give out inferior designs.

The proposed algorithm aims at developing an algorithm which is as fast as the heuristic methods and also gives results comparable to the deterministic methods. For this, the discrete Differential Evolution algorithm combined with a population generating mechanism which utilizes linear programming is developed. In section 4.1, firstly the problem is formulated and the objective function for minimization is defined. Next, the entire design procedure is given. The algorithm is referred to as Differential Evolution Filter Design Optimization (DEFDO) Algorithm. Lastly, in section 4.2, each part of the DEFDO algorithm is explained in detail.

4.1 Problem Formulation

In this section, the proposed filter design method is discussed. The proposed technique involves a combination of linear programming and evolutionary algorithm. The evolutionary method is a hybrid Differential Evolution method discussed in Chapter 3 in sections 3.4.2 and 3.5. The adaptive control parameters have been used and for the mutation operator, a bit string method described in section 3.5 is implemented.

4.1.1 Minimax Error and Adder Cost Joint Objective Function Calculation

For the discrete filter design problem, the problem can be divided into two parts: subject and object (as refereed to in [33]). The object is the main aim of the problem i.e. minimize the hardware complexity while the subject are the constraints that have to be satisfied i.e. the maximum allowable ripple error. Hence, the optimization problem can be modeled as in Eq. 4.1.

$$\mathbf{minimize} \quad \sum_{i=0}^{M} Cost(x_i) \tag{4.1}$$

$$\mathbf{subject \quad to:}$$

$$\Delta \leq \delta_p \tag{4.2}$$

where $\mathbf{x}$ is defined as the vector $\mathbf{x} = [x_0, x_1 ... x_M]^T \in \mathbb{Z}^{M+1}$, Δ is the weighted minimax error (Eq. 4.3) and δ_p is the maximum allowable error in the passband as defined in the filter specifications (*fspec*). The values of vector $\mathbf{x}$ are related to the filter coefficients as $h_n = x_{M-n+1} = h_{N-n-1}$. The passband error ($\delta_p$) is considered for comparison as the weight of the pass band is fixed to be unity.

$$\Delta = \mathbf{max}[W(\omega)|\frac{H_a(\omega)}{G} - D(\omega)|] \tag{4.3}$$

where $H_a(\omega)$ is the amplitude response of the filter, $D(\omega)$ is the desired function, $W(\omega)$ is the weighting function and G is the passband gain. Eq. 4.4 and 4.5 give the expressions for the weighting function $W(\omega)$ and the desired function $D(\omega)$. The value of passband gain G is given by (4.6).

$$W(\omega) = \begin{cases} 1 & \text{if } \omega \in \Omega_p \\ \frac{\delta_p}{\delta_s} & \text{if } \omega \in \Omega_s \\ 0 & \text{otherwise} \end{cases} \tag{4.4}$$

$$D(\omega) = \begin{cases} 1 & \text{if } \omega \in \Omega_p \\ 0 & \text{if } \omega \in \Omega_s \\ \textit{any number} & \text{otherwise} \end{cases} \tag{4.5}$$

$$G = \begin{cases} \frac{G_{p,max}+G_{p,min}}{2} & \text{if } \frac{G_{p,max}-G_{p,min}}{2} > \frac{\delta_p}{\delta_s} G_{s,max} \\ G_{p,min} + \frac{\delta_p}{\delta_s} G_{s,max} & \text{if } \textit{otherwise} \end{cases} \tag{4.6}$$

where

$$G_{p,max} = \mathbf{max}|H_a(\omega)|_{\omega \in \Omega_p}$$

$$G_{p,min} = \mathbf{min}|H_a(\omega)|_{\omega \in \Omega_p}$$

$$G_{s,max} = \mathbf{max}|H_a(\omega)|_{\omega \in \Omega_s}$$

In Eqs. 4.4, 4.5, 4.6 Ω_p is the set of frequencies belonging to the pass band, Ω_s is the set of frequencies belonging to the stop band.

Since in Eq. 4.3, the coefficients are represented in the integer subspace and Normalized Peak Ripple Magnitude is used, hence the amplitude response is divided by the passband gain, given by Eq. 4.6, in order to determine the error constraints which are given for unity passband gain. Eq. 4.7 shows the expression for the amplitude response of a linear phase filter.

$$H_a(\omega) = \mathbf{v}_c^T \mathbf{x} \tag{4.7}$$

where $\mathbf{v}_c$ is a vector containing the appropriate trigonometric function values and depends on the type of filter. Eq. 4.8 gives the expression for Type I and Type II filters for $\mathbf{v}_c$. Similar expressions can be derived for Type III and Type IV filters.

$$\mathbf{v}_c = \begin{cases} [1, 2cos(\omega), ...2cos((M-1)\omega), 2cos(M\omega)]^T & \text{for Type I} \\ 2cos(0.5), 2cos(\omega+0.5)\cdots 2cos((M-1)\omega+0.5), 2cos(M\omega+0.5) & \text{for Type II} \end{cases} \tag{4.8}$$

4.1.2 The Differential Evolution Filter Design Optimization Algorithm

In the previous section (4.1.1), the objective of the problem was defined along with the constraints. In this section an algorithm that achieves the objective is given. The algorithm if referred to as Differential Evolution Filter Design Optimization (DEFDO). The method is summarized below and explained in detail in the next section.

1. The index of the coefficients that can be made zero, denoted as Z, is found first. For small to medium length filters, Z is found by finding the range using Eq. 4.9.

$$\textbf{minimize} \quad f = h(k) \quad f = -h(k) \tag{4.9}$$

$$\text{for } k = 0, 1 ... \lfloor \tfrac{N-1}{2} \rfloor$$

$$\begin{aligned} \textbf{subject to:} \quad & (1-\delta_p) \leq H_a(\omega) \leq (1+\delta_p) \quad \text{for} \quad \omega \in \Omega_p \\ & -\delta_s \leq H_a(\omega) \leq \delta_s \quad \text{for} \quad \omega \in \Omega_s \end{aligned}$$

where

$$H_a(\omega) = \sum_{n=0}^{\lfloor \frac{N-1}{2} \rfloor} h(n) Trig(\omega, n)$$

$Trig(\omega, n)$ is the trigonometric function based on the type of FIR filter for an N tap filter. Those coefficients whose range includes the zero crossing are possible candidates for Z. The optimal infinite precision coefficients are found by solving the linear program in Eq. 4.10 and setting $Z = \phi$.

$$\textbf{minimize} \quad f = \delta \tag{4.10}$$

$$\begin{aligned}
\textbf{subject to:} \quad (1-\delta) \leq H_a(\omega) \leq (1+\delta) \quad & \text{for} \quad \omega \in \Omega_p \\
-\delta\frac{\delta_p}{\delta_s} \leq H_a(\omega) \leq \delta\,\frac{\delta_p}{\delta_s} \quad & \text{for} \quad \omega \in \Omega_s
\end{aligned}$$

$$H_a(\omega) = \sum_{n=0, n\notin Z}^{\lfloor\frac{N-1}{2}\rfloor} h(n)Trig(\omega, n)$$

The coefficient, among the Z candidates, whose absolute value is the least is fixed as zero and the linear program is again solved to find the minimax error (δ). The process iterates by including the next candidate until the minimax error exceeds the required error. Sometimes, it is needed to back trace the steps and include lesser zeros in the coefficients than given by the algorithm in order to accommodate the loss in precision due to quantization of the coefficients. The last optimal coefficient values are denoted by the vector $\mathbf{h}_c$. The range is again calculated by fixing the zero coefficients. Let the range of each unique coefficient be denoted by $[r_l^i, r_u^i]$ for $i = 0, 1 \cdots M$. The expanded form and explanation of the linear programs is given in section 4.2.1.

2. For a given effective word length (EWL), it is needed that the greatest coefficient can be represented in the specified number of bits. So, if the coefficient set $\mathbf{h}_c$ has a gain of 1, then the maximum gain whose greatest coefficient can take a value up to 2^{EWL-1} is given by G^u, where G^u is defined as

$$G^u = \frac{2^{EWL} - 1}{\|\mathbf{h}_c\|_\infty} \tag{4.11}$$

Table 4.1: Range of Filter

Lower Limit (r_l^i)	Optimized Coefficient ($\mathbf{h}_c$)	Upper Limit (r_u^i)
0.3138	0.3300	0.3457
0.1964	0.2012	0.2051
0.0279	0.0430	0.0581
-0.0556	-0.0439	-0.0342
-0.0487	-0.0427	-0.0369
-0.0174	-0.0070	0.0026
0.0059	0.0124	0.0195
0.0020	0.0091	0.0147

$$G^l = \frac{G^u}{2} \tag{4.12}$$

The gain range, $[G^l, G^u]$, is divided into S equally spaced sections. For each section the lower range is denoted as $g_{s,l}$ and the upper range is denoted as $g_{s,u}$. The number S is taken according to the hardware capability and the time constraint of the design procedure. Thus, a trade-off is met between design time and design accuracy.

3. The optimization problem is solved by generating a population as explained in section 4.2.2 in the gain range and running a local search using the Differential Evolution algorithm described in Chapter 3. The variant of the DE used is *DE/rand/1/bin* along with adaptive control parameters. The algorithm also utilizes adaptive search space reduction (section 4.2.5) and the selection operator given in section 4.2.4. Techniques used for shortening the run time are also used (section 4.2.6). The synthesis of the SAN is carried out by modifying the RAG-n algorithm (section 4.2.3).

Table 4.1 shows an example of the range of the filter (G1 of Table 5.1). The first column shows the lower limit of each unique coefficient for unity gain while the third column shows the upper limit. In the second column, the optimal infinite precision filter coefficients are given. It is seen that the third last coefficient among them has the zero crossing in its range. Thus, this coefficient is a possible candidate for the fixing to zero. In fact, for

$EWL = 7$ design shown in Chapter 5 this coefficient has been fixed to zero. However, for $EWL = 6$, fixing this coefficient to zero renders the filter search space not to give out feasible solutions.

4.2 Algorithms Used In DEFDO

In this section, the DEFDO algorithm and adjustments in the Differential Evolution algorithm tailored for the filter design problem are discussed. Firstly, the linear programs used in section 4.1 are explained in detail. Next, the population generation mechanism for the DE algorithm is explained. After that the modified version of the RAG-n algorithm which gives the hardware cost is explained. Subsequently, the selection operator of the DE algorithm is explained. Lastly, enhancements for the DE algorithm are discussed which include adaptive sub space reduction. Next, methods to cut short the computation time are discussed. Lastly, the modified RAG-n algorithm for the synthesis of filters is given.

4.2.1 Linear Programs Used in Filter Design

The linear programs of Eqs. 4.9 and 4.10 are solved on a dense grid on frequencies in $[0, \pi]$. Let the frequency grid be represented as $\Omega = [\omega_1, \omega_2, ...\omega_n]$ and the filter coefficients as $\mathbf{h}_f = [h_0, h_1, \cdots h_{N-1}]$ for an N tap filter. Also let $M = \lfloor \frac{N-1}{2} \rfloor$. The inequality

$$W(\omega)|H_a(\omega) - D(\omega)| \leq \delta$$

can be written as

$$\begin{aligned} H_a(\omega) - \frac{\delta}{W(\omega)} &\leq D(\omega) \\ -H_a(\omega) - \frac{\delta}{W(\omega)} &\leq -D(\omega) \end{aligned}$$

where $W(\omega)$ and $D(\omega)$ are defined in Eqs. 4.4, 4.5 respectively. Since the linear program is evaluated on Ω, the above inequalities are written as

$$\begin{aligned} \mathbf{h} - \mathbf{w}\delta &\leq \mathbf{d} \\ -\mathbf{h} - \mathbf{w}\delta &\leq -\mathbf{d} \end{aligned}$$

where

$$\begin{aligned} \mathbf{h} &= [H(\omega_1), H(\omega_2), \cdots H(\omega_n)]^T \\ \mathbf{w} &= [1/W(\omega_1), 1/W(\omega_2), \cdots 1/W(\omega_n)]^T \\ \mathbf{d} &= [D(\omega_1), D(\omega_2), \cdots D(\omega_n)]^T \end{aligned}$$

The vector $\mathbf{h}$ can be written as

$$\mathbf{h} = \mathbf{C}\mathbf{g}$$

where $\mathbf{C}$ is defined for Type I filter as

$$\mathbf{C} = \begin{bmatrix} 1 & 2cos(\omega_1) & \cdots & 2cos(M\omega_1) \\ 1 & 2cos(\omega_2) & \cdots & 2cos(M\omega_1) \\ \vdots & \vdots & \vdots & \vdots \\ 1 & 2cos(\omega_n) & \cdots & 2cos(M\omega_n) \end{bmatrix}$$

$$\mathbf{g} = [h_M, h_{M-1}, \cdots h_0]^T$$

For the linear program of Eq. 4.9, the linear program setup as defined in Table 3.1 is given in Eq. 4.13.

$$\begin{aligned} \mathbf{x} &= \mathbf{g} \\ \mathbf{f} &= [0, 0, \cdots \pm 1, \cdots 0]^T_{M+1\times 1} \\ A_{ieq} &= \begin{bmatrix} \mathbf{C} \\ -\mathbf{C} \end{bmatrix} \\ b_{ieq} &= \begin{bmatrix} \mathbf{d}+\mathbf{w} \\ -\mathbf{d}+\mathbf{w} \end{bmatrix} \end{aligned} \tag{4.13}$$

No equality constraints and bounds on the values of $\mathbf{x}$ are needed and the value of $\mathbf{f}$ corresponding to the the k^{th} coefficient is made 1 for finding the lower limit and -1 for the upper limit. For the linear program of Eq. 4.10, the setup is given in Eq. 4.14. For setting the Z constraint, the column in matrix $\mathbf{C}$ corresponding to that index is made zero. Also, no equality constraints and bounds on variables are needed.

$$\begin{aligned} \mathbf{x} &= [\mathbf{g}^T, \delta]^T \\ \mathbf{f} &= [0, 0, \cdots 0, 1]^T_{M+2\times 1} \\ A_{ieq} &= \begin{bmatrix} \mathbf{C} & -\mathbf{w} \\ -\mathbf{C} & -\mathbf{w} \end{bmatrix} \\ b_{ieq} &= \begin{bmatrix} \mathbf{d} \\ -\mathbf{d} \end{bmatrix} \end{aligned} \tag{4.14}$$

4.2.2 Population Generation for Differential Evolution

For the Differential Evolution Algorithm, a bit string population is generated and for each unfixed coefficient the number of bits allocation is done according to its range (found using Eq. 4.9) scaled by the gain called quantized range. The unfixed coefficients are the coefficients that have not already been fixed to a value of zero using the sparse filter technique. For finding the quantized range, the lower range limit is multiplied by the lower gain limit and rounded to nearest integer to obtain the lower limit of the quantized range.

The upper limit is found in a similar fashion (Eqs. 4.15).

$$q^i_{l,s} = \lceil g_{s,l} \times r^i_l \rceil \\ q^i_{u,s} = \lfloor g_{s,u} \times r^i_u \rfloor \tag{4.15}$$

and the values are stored in vector $\mathbf{q}_{l,s} = [q^0_l, q^1_l \cdots q^M_l]$ and $\mathbf{q}_{u,s} = [q^0_u, q^1_u \cdots q^M_u]$. The maximum number of bits to represent each coefficient is given by Eq. 4.16.

$$b^i = log_2\lceil q^i_u - q^i_l + 1\rceil \tag{4.16}$$

The bit information is saved in the vector $\mathbf{b} = [b^0, b^1 \cdots b^M]$. The population is generated uniformly in the quantized coefficient range and the lower end of the range is encoded by string of zeros and the upper end is encoded accordingly. Thus, the actual number are not encoded but instead the position of numbers in the range are encoded in gray codes. The use of gray codes is done to minimize the offset created by the mutation operation for which a large jump might cause the filter to become infeasible and selection of such population member will not occur. For example, in the range $[50, 53]$, the coefficient 50 will be given the code "00", 51 be given "01" and so on. This way, the number of bits needed to represent the entire population is substantially reduced. Figure 4.1 shows the pseudo code for generating population for small filters. Also, Figure 4.2 shows the pseudo code for converting the population into decoded bits using differential encoding.

Table 4.2 shows the quantized range of the filter G1 as an example. This range has been calculated from the entries in Table 4.1 where $\|\mathbf{h}_c\|_\infty = 0.3300$ and $G^u = 190.8979$ and $G^l = 95.4489$. The range is divided into 10 sections and the section shown is the eight section.

For large filters, a polynomial time algorithm similar to that given in [34, 36] is run to get initial discrete solutions for each gain section. However, the algorithm is modified to keep running the loop for each gain section even if the filter constraints fail to meet. The algorithm is repeated here.

1. For each passband gain, set $i = 0$.

Table 4.2: Quantized Range of Filter

Lower Limit $q^i_{l,s}$	Optimized Coefficient	Upper Limit $q^i_{u,s}$	Bit Allocation (b^i)
50	56	60	4
31	34	36	3
4	7	10	3
-10	-7	-5	3
-8	-7	-6	2
-3	-1	1	3
0	2	4	3
0	1	3	3

2. If $i \notin Z$, find the range of the coefficient using Eq. 4.9 where

$$H_a(\omega) = \sum_{n=0, n\notin Z}^{i-1} h_f(n) Trig(\omega, n) + \sum_{n=i, n\notin Z}^{\lfloor \frac{N-1}{2} \rfloor} h(n) Trig(\omega, n)$$

where $h_f(n)$ are the fixed coefficients.

3. Find the midpoint of the range and set $h_f(i)$ to the integer closest to the midpoint.

4. $i = i + 1$ until all coefficients have been fixed ($[h_f(0), h_f(1), \cdots h_f(M)]^T = \mathbf{h}_{base}$).

For each gain section, the discrete solution that are within 15% away from meeting the specifications are saved: pseudo-viable ($\mathbf{h}_{base}$) solutions. Half the population is generated by perturbing a percentage of the base solution ($Prob_{pert}$) and the other half is generated uniformly in the quantized range. For high word lengths the search space is capped and the highest range coefficient is given the maximum number of bits and others are given according to their range extent and a neighborhood is created around the base solution. The decision of the maximum number of bits is a trade off between design time and design quality. Figure 4.3 shows the pseudo code for generating the population for large filters. Figure 4.4 shows the encoding scheme for the neighborhood of the base solution.

Figure 4.1: Pseudo Code for Population Generation(Small)

```
**process** popgensmall
if $filtord < 60$
1  find quant. range
2  find bits for each coeff
3     for $i = 1 : P$
4        for $j = 1 : D$
5           $m^i_j = q^i_l + rand(0,1) * [q^i_u - q^i_l]$
6        end for
7     end for
decode to bits
end process
```

Figure 4.2: Pseudo Code for Decoding to Bits

```
process decode to bits
1     for $i = 1 : P$
2        for $j = 1 : D$
3           $gene^i_j = dec2bin(m^i_j - q^i_l)$
4           if $length(gene^i_j) = b^i$ : goto step 6
5            else prefix zeros to make equal length
6           $chromosome^i = horzconcat(chromosome^i, gene^i_j)$
7        end for
8        $pop = vertconcat(pop, chromosome^i)$
9     end for
end process
```

Figure 4.3: Pseudo Code for Population Generation(Large)

```
process popgenlarge
if $filtord \geq 60$
1   decide bits for each coeff.
2   find quant. range
3       for $i = 1 : P/2$
4           for $j = 1 : D$
5               $m_j^i = q_l^i + rand(0,1) * [q_u^i - q_l^i]$
6           end for
7       end for
8       for $i = P/2 : P$
9           for $j = 1 : D$
10              if $rand(0,1) < Prob_{pert}$
11                  $m_j^i = q_l^i + rand(0,1) * [q_u^i - q_l^i]$
12              else $m_j^i = h_{base,j}$
13              end for
14          end for
decode to bits
end process
```

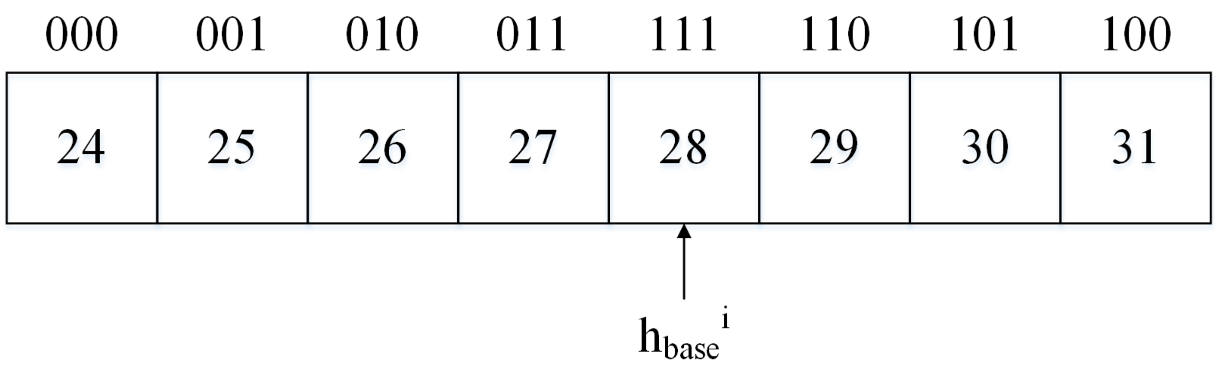


Figure 4.4: Gray Encoding for Base Solution Neighborhood

Figure 4.5: Pseudo Code for Synthesis Using RAG-n

```
process RAG-$n$ synthesis
input: coefficient set
1   generate fundamentals (F) from coefficient set
2   incomplete (I)=F, graphset(G) ={1}
3   generate all cost one numbers upto $2^{EWL}$
4   among incomplete find all costone (C1) numbers
5   $\mathbf{G} = \mathbf{G} \cup \mathbf{C1}$, adder=size{C1}
6   I=I - C1
7   do find $num \in \mathbf{I}$, $num = a \times 2^i \pm b$
     where $a, b \in \mathbf{G}$, $i = 0, 1 \cdots EWL$
8      if a match is found:$\mathbf{G} = \mathbf{G} \cup num$
9      I=I -num, adder=adder+1
10  while ($\mathbf{I} = \phi$ or all combinations exhausted)
output: adder, F, G, I
end process
```

4.2.3 Modified RAG-n Algorithm

It was observed in literature that the minimal adder filter coefficients always had adders equal to the number of fundamentals and thus required only one adders. The RAG-n algorithm [18] uses the MAG [19] algorithm for generating the lookup table which is valid up to a word length of 10 bits. We modify the RAG-n algorithm to not use the lookup table, but instead generate cost one adders up to the required number of bits and utilize those coefficients further synthesize the rest of the fundamentals. Figure 4.5 shows the pseudo code for the modified RAG-n algorithm used in the thesis to synthesize the shift and add network. If, however, the deterministic part of the RAG-n cannot synthesize all the fundamentals, the algorithm is terminated and the number of adders is estimated using Eq. 4.17. This estimate is all that is needed in directing the search process.

$$n_{add} = \|\mathbf{F}\|_0 + \|\mathbf{I}\|_0 \tag{4.17}$$

where **F** and **I** are vectors containing the fundamentals and incomplete (un-synthesized) fundamentals.

4.2.4 Selection Operator for Differential Evolution

In section 4.1.1, the objective is defined as the hardware cost to implement the filter. An effective measure of hardware cost is the number of adders needed to represent the shift and add network. The reason for this choice is that for a fixed word length and filter order, the number of structural adders is fixed and the width of each adder is also fixed. Thus after empirically determining the filter order and the word length for the design, the search is for the coefficients that have the least adder count. Eq. 4.18 gives the calculation of the objective function for driving the search for minimal adders as well as finding solutions that satisfy the error constraints.

$$f(\mathbf{x}) = \begin{cases} \Delta(\alpha - \frac{\beta}{n_{add}}) & \text{if } \Delta > \delta_p \\ \delta_p(\alpha - \frac{\beta}{n_{add}}) & \Delta \leq \delta_p \end{cases} \tag{4.18}$$

where δ_p is the error required by the filter specification $fspec$, and n_{add} is the number of adders needed to synthesize the filter coefficients obtained using the RAG-n algorithm described in section 4.2.3, and α, β are constants. The value of Δ is calculated according to Eq. 4.3 and the constants α, β are chosen to drive the filter search process. For large filters, $\alpha = \beta = 1$, but for small filters, different variations are tried.

To reduce the time for the objective function calculation, synthesis is not carried out at every iteration for every member. Fig. 4.6 shows the flowchart for calculating the objective function value. If the minimax error does not meet the specification, then the actual error Δ is used and the number of fundamentals is used in equation (4.18) as n_{add}. If, however, the minimax error meets the specification, then further reduction in minimax error is not needed. Thus, the filter is synthesized using the modified RAG-n algorithm and its number of adders is recorded. Also, at every iteration if the number of adders is found to be better than the current best, then only is it synthesized. If not, then the number of fundamentals is used as a count of the number of adders to drive the search process. The reason for

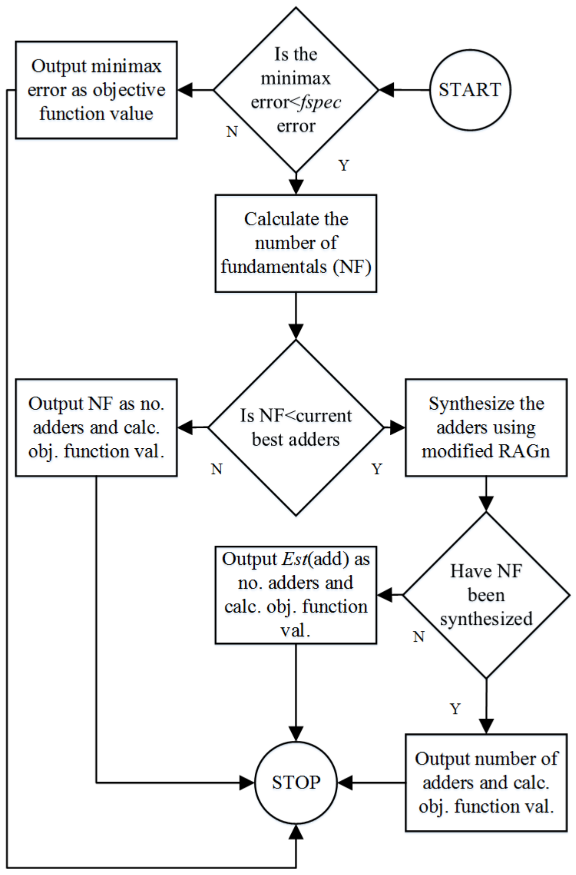

START
Is the minimax error<*fspec* error
N
Y
Output minimax error as objective function value
Calculate the number of fundamentals (NF)
Is NF<current best adders
N
Y
Output NF as no. adders and calc. obj. function val.
Synthesize the adders using modified RAGn
Have NF been synthesized
N
Y
Output *Est*(add) as no. adders and calc. obj. function val.
Output number of adders and calc. obj. function val.
STOP

Figure 4.6: Selection Operation Flowchart

using the count of the number of fundamentals as an indicative of the number of adders is because of the result given in [18] which states that the number of adders needed to implement n fundamentals can not be less than n.

4.2.5 Adaptive Search Space Reduction

It was observed that the values of some small filter coefficients converged to the same value in early generations for all the population members. Thus, the need to further optimized those coefficients was eliminated. Thus an adaptive search space reduction technique is implemented. The technique proceeds as follows. While running the algorithm, at every

Figure 4.7: Pseudo Code for Extremal Points Using Newton's Root Finding Algorithm

```
process Ex points
input: coefficient vector $h$
1  Find $\frac{dH(\omega)}{d\omega}$ at a grid of 3N points($\Omega$)
2  Find points where $\frac{dH(\omega)}{d\omega}$ changes sign: $SCP$
3  Compute midvalue of each SCP; save in $\mathbf{p}$
4  for $i = 1 : size(\mathbf{p})$
5      $\mathbf{x}_{out}(i)$=newton($\mathbf{p}$(i))
6  end for
output: $\mathbf{x}_{out}(i)$
end process
```

50th generation, the population is checked after decoding for each coefficient. If, for a certain coefficient, the members of the entire population have converged to a single value then that coefficient is not further optimized. The population is made to exclude the bits corresponding to that coefficient. However, in the objective function, the fixed value of that coefficient is considered. This way, the search space is reduced accelerating the search process.

4.2.6 Computation Cost Reduction

To reduce the computation cost, two methods are employed. Firstly, for the linear programming algorithms, taking a dense grid of frequencies amounts to large number of inequalities. The problem is aggravated for large filters. In [35], Samulei *et al* have found the zero crossing points of the stopband. In a similar fashion, the extremal points of ripples of the amplitude response can be found. The extremal points are usually of the order of half the filter length and if the filter meets the specification at the extremal points, it is bound to meet the specification elsewhere. Thus, the grid of frequencies for the linear program consists of the extremal points and the band edges which significantly reduces the number of inequalities.

For finding the extremal points of a Type I FIR filter with unique coefficients given by

Figure 4.8: Pseudo Code for Newton's Root Finding Algorithm

```
process newton
input: $x_0$
1  Compute $f_0 = \frac{dH(\omega)}{d\omega}|_{x_{in}}$
2  Compute $df_0 = \frac{d^2H(\omega)}{d\omega^2}|_{x_{in}}$
3  set $i = 0$, set tolerance=$tol$
4  do $x_{i+1} = x_i - \frac{f_i}{df_i}$
5      Compute $f_{i+1} = \frac{dH(\omega)}{d\omega}|_{x_{i+1}}$
6      Compute $df_{i+1} = \frac{d^2H(\omega)}{d^2\omega}|_{x_{i+1}}$
7      $er = |x_{i+1} - x_i|$
8      $i = i + 1$
9  while($er > tol$)
output: $x_i$
end process
```

vector $\mathbf{x} = [h_M, h_{M-1} \cdots h_0]^T$, the zero crossings of the derivative of the amplitude response are found. The amplitude response $H(\omega)$ is given by:

$$H(\omega) = [1, 2cos(\omega_1) \cdots 2cos(M\omega_1)]\mathbf{x}$$

Differentiating twice,

$$\frac{dH(\omega)}{d\omega} = [0, -2sin(\omega), -2 \times 2sin(2\omega) \cdots - 2 \times Msin(M\omega)]\mathbf{x}$$

$$\frac{d^2H(\omega)}{d\omega^2} = [0, -2cos(\omega), -2 \times 2^2cos(2\omega) \cdots - 2 \times M^2cos(M\omega)]\mathbf{x}$$

To find the extremal point (minima or maxima), the derivative of the frequency response is analyzed by Eq. 4.19.

$$sign(\frac{dH(\omega)}{d\omega}|_{\omega\in\Omega}) = [p_1, p_2 \cdots p_n] \tag{4.19}$$

where Ω is a grid of frequencies evenly spaced containing n points ($n = 3$ times the filter order) and *sign* is the signum function. The two points where the above equation changes sign are the points in between which the extremal point of the amplitude response lies. Thus, this midpoint value is given as an initial starting point for the Newtons root finding algorithm. Also, Newtons root finding algorithm is supplied with the function $f = (dH(\omega))/dw$ and the derivative $df = d^2H(\omega)/d\omega^2$. The above equations can similarly be modified for Type II, III and IV FIR filters. Figure 4.7 and figure 4.8 show the pseudo codes for finding the extremal points of the frequency response of an FIR filter using the Newton's root finding algorithm.

Secondly, for reducing the computation cost in calculating the minimax error for the Differential Evolution algorithm, the amplitude response of the filter corresponding extreme minima of the coefficient's quantized range is calculated and stored as reference.

$$H_{pre} = \mathbf{v}_c^T q_{l,s}^i \tag{4.20}$$

Since, the amplitude response is a linear function, the amplitude response of difference of the obtained filter coefficients from the reference is calculated and added to the stored value. This minimizes the computation cost by 5 times

Summary

This chapter introduced the DEFDO algorithm for the design of finite word length FIR filter with the aim of reducing the hardware cost of implementation. The objective of the problem was defined mathematically as a joint optimization of minimax error and adder cost. The supplemental techniques for adapting the optimization routines for the filter design problem such as the modified RAG-n algorithm, the difference based population generation mechanism, the selection operator for cutting short the objective function calculation time, the adaptive search space reduction and the extremal point utilization for linear programs were also given.

Figure 4.9: Pseudo Code for Differential Evolution with Adaptive Search Space Reduction and Pre-Calculated Objective Function

```
process DE
1   generate population
2   calc. $H_{pre}$, set g=1
3   do
4      mutation
5      decode bits to coeff.
6      crossover
7      selection
8      if g%50=0 check for convergence
9         if converged coeff found: $H_{pre} = H_{pre} + \mathbf{v}_c^T h_c$
10          remove bits of $h_c$ in pop.
11      end if end if
12      g=g+1
13   while ($g < g_{max}$)
end process
```

Chapter 5

Design Examples and Results

This chapter will show the effectiveness of the proposed Differential Evolution Filter Design Algorithm for the design of finite word length linear phase FIR filters with minimal hardware cost. Firstly, the convergence properties of the algorithm will be analyzed. Next, the design results for prototype filters will be given. The design is carried out for the joint optimization of minimax error and adder cost. Subsequently, the hardware synthesis examples will be given to show the reduction in hardware. Lastly, the obtained results are compared with the state of the art methods and an analysis of the algorithm complexity is given.

5.1 Comparison of Variations of Differential Evolution Algorithms

Figure 5.1 and figure 5.2 show the convergence curves of different values of the mutation factor F and crossover rate CR . On the x-axis is the generation number and on the y-axis is the objective function value of the best trial vector in that generation. To differentiate the curves, a median filter of window size 10 has been applied. The generation's best trial vector is an important characteristic of the Differential Evolution algorithm as it shows its

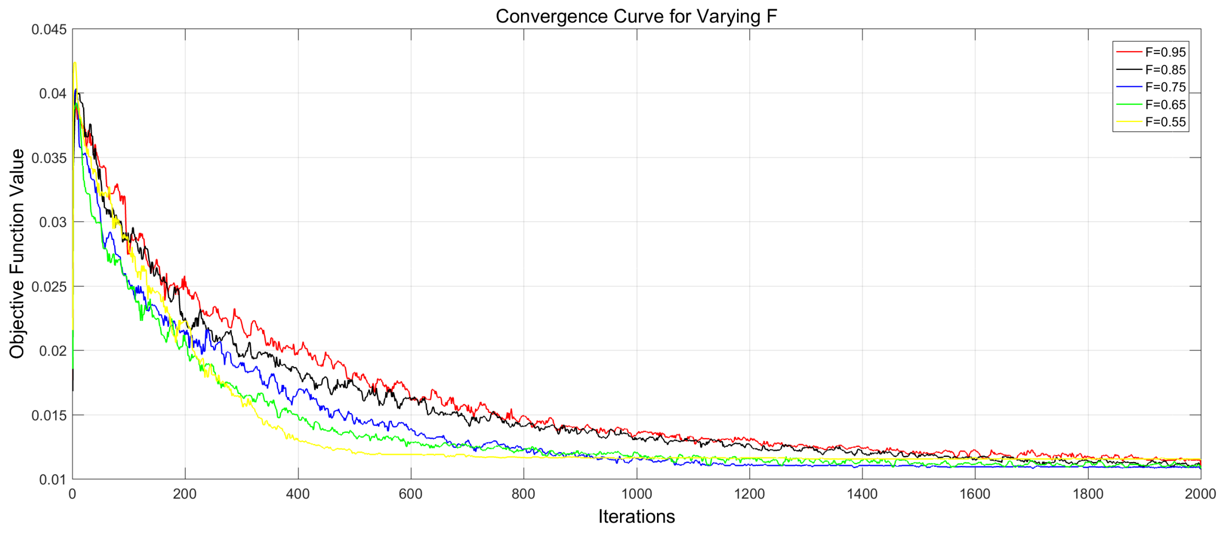


Figure 5.1: Convergence Curve for Different Value of F

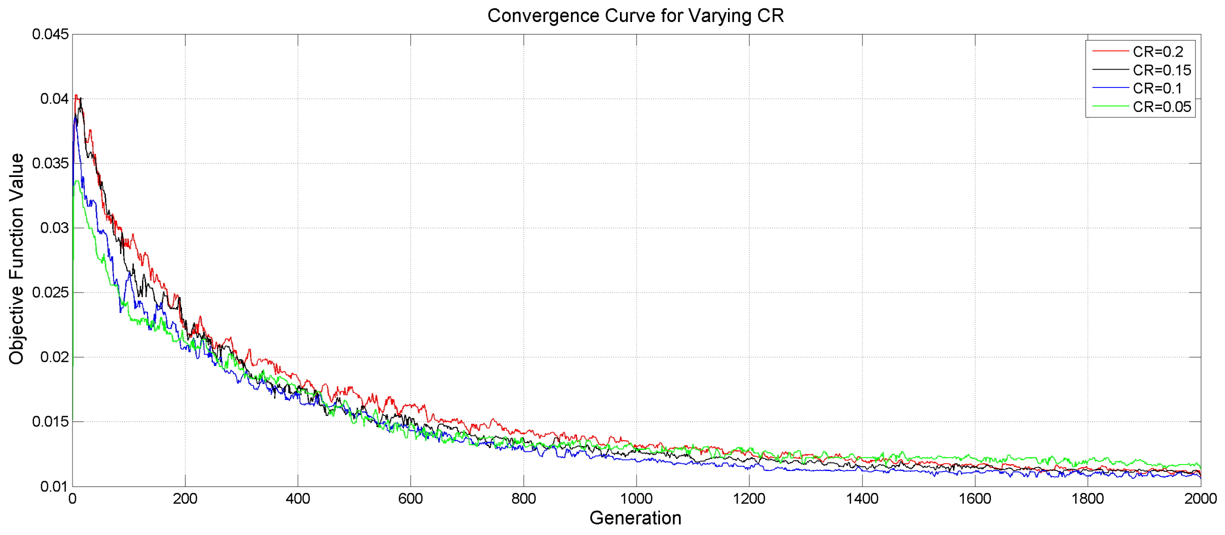


Figure 5.2: Convergence Curve for Different Value of CR

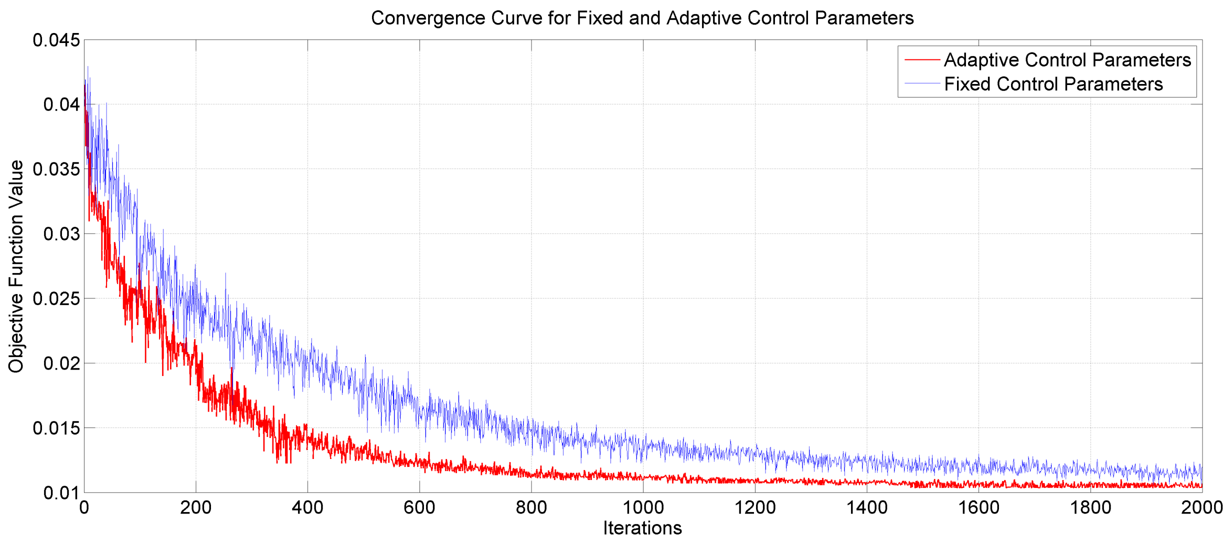


Figure 5.3: Convergence Curve for Adaptive and Fixed Control Parameters Before

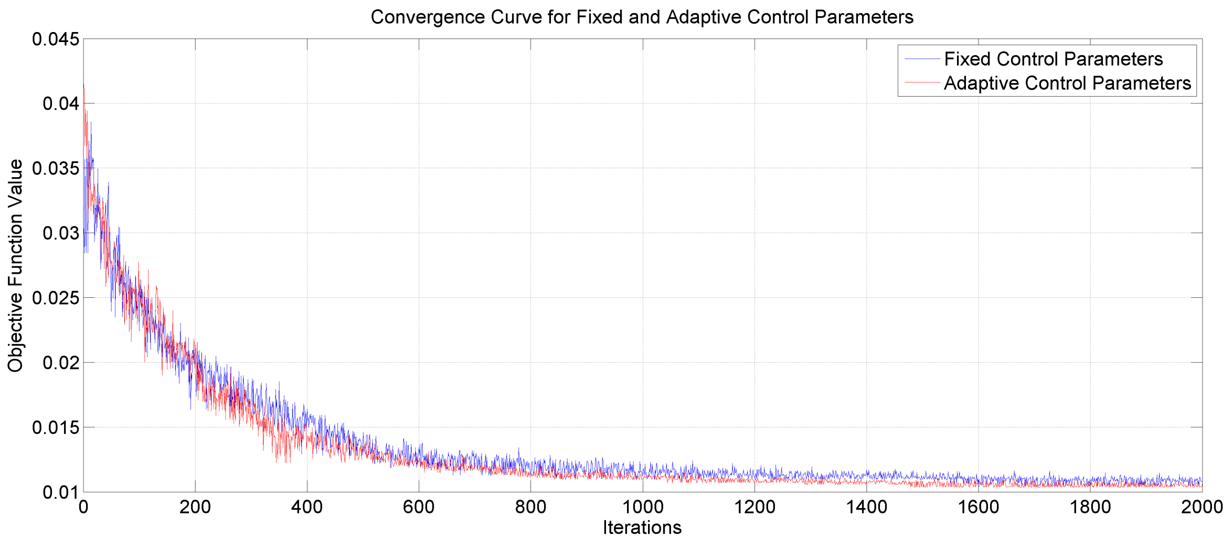


Figure 5.4: Convergence Curve for Adaptive and Fixed Control Parameters After

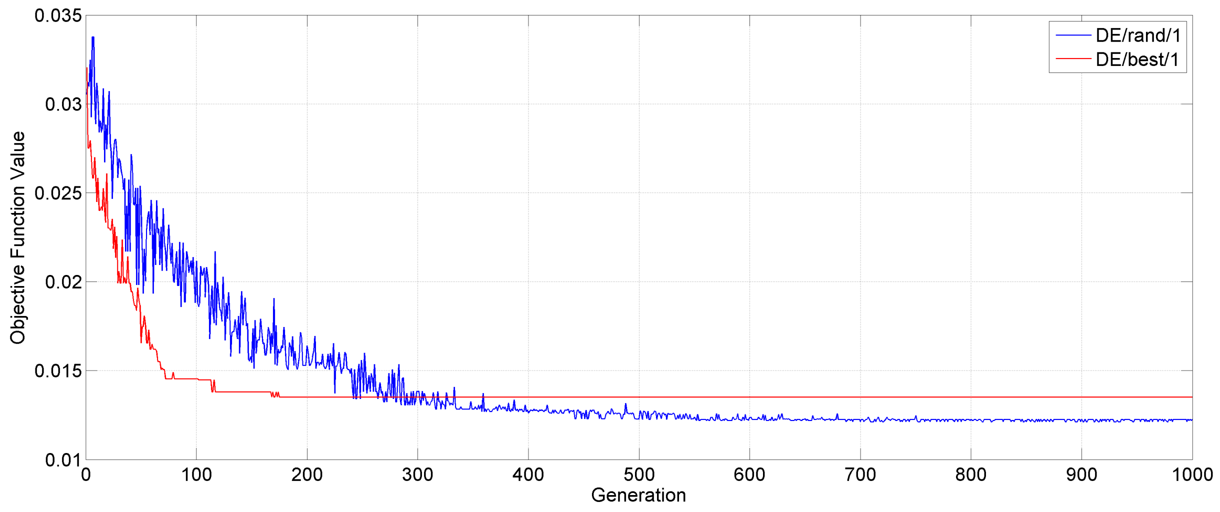


Figure 5.5: Convergence Curve for Different Variation of Mutation Operator

ability to search and come up with better trial vectors.

It can be seen in the figure 5.1 that the value of $F = 0.75$ shows better convergence properties than other values. The convergence for $F = 0.65$ is much faster initially but the terminal value is slightly higher than that of $F = 0.75$. In the figure 5.2 the value of $CR = 0.1$ shows better convergence properties than other values. Even though $CR = 0.05$ has the better convergence initially, it is seen that the final convergence value is higher for this case.

Figure 5.3 shows the convergence curve when fixed parameters of $F = 0.85$ and $CR = 0.2$ are used as compared to when adaptive parameters are used. The adaptive parameters have been defined in section 3.4.2. The values of CR and F have been taken from the recommended values in literature. It can be seen that a marked difference exists in the convergence properties of the fixed and adaptive control parameters. The adaptive control parameters show faster convergence and also converge to a better value than fixed parameters. Figure 5.4 shows the comparison again after obtaining the optimal values of CR and F by going through many iterations. It can be seen that adaptive control parameters still out perform fixed parameters slightly.

Figure 5.5 shows the convergence curve for the variation of the mutation parameter. The two variations plotted as $DE/rand/1$ and $DE/best/1$. It is seen from the figure that the $DE/best/1$ has a very fast convergence rate. However, it is very likely to converge prematurely. The $DE/rand/1$ version has a slower convergence rate but it converges to a better value. The reason for its better convergence is the presence of randomness in the base vector which causes many different trials vectors to be generated.

5.2 Joint Optimization of Minimax Error and Hardware Complexity

In this section, the results for the joint optimization of minimax error and adder cost for filter implementation will be given. Firstly, the empirical determination of the filter order and word length will be discussed. Once, the filter order and word length is fixed, an optimization routine can be performed aimed at minimizing the adder cost while keeping an upper tab on the maximum allowable ripple error as defined in section 4.1 of Chapter 4.

5.2.1 Empirical Determination of Filter Order and Wordlength

The filter design problem is given in Chapter 1, section 1.2 with a $fspec$ as shown in figure 1.1. For a infinite precision design, the filter order can be estimated using many techniques (e.g. firmpmord function of MATLAB). However, for a discrete design, the filter order estimation is a difficult task. In addition, the word length for the design also needs to be determined. In Table 5.1, the specifications of 10 filters found in literature are given. In an attempt to find the order for designing the discrete filter, Table 5.2 list the orders for various configurations. In column 2 of Table 5.2, the order given by the MATLAB function *firmpmord.m* is given for the filters whose specifications are given in Table 5.1. In column 3, the actual order required to obtain a feasible infinite precision design is given. Thus, in most cases the *firmpmord.m* function underestimates the filter order. In column 4 and

5, the filter orders required to satisfy a more stringent error constraint are given. Thus $0.9 \times \delta$ implies that the stringent error constraint is 0.9 multiplied by the initial maximum allowable error in the stop and passbands. In column 6, the actual filter order used in our design and also in the most state of the art examples found in literature is listed. Finally, in column 7, the effective word length of the filter coefficients corresponding to the order in column 6 is listed.

Figure 5.6 gives a bar graph of the orders listed in Table 5.2 normalized by the minimum EWL filter order. It can be seen from the figure that the filter order with the least possible effective word length lies very close to the order of $0.8 \times \delta$. The reason for this behavior is that some relaxation must be given in moving from infinite precision to finite precision. Also, the presence of sparsity further exacerbates the problem.

Figure 5.7, 5.8, 5.9 shows the results initially published in [36]. It is seen that an inverse relation exists between filter order (or filter length as shown in the figure) and the effective word length used for implementing the filter for a certain part of the curve. It is seen that increasing the word length decreases the filter order needed to implement. However, below a certain filter order, increasing the word length produces no feasible results. Similarly, by increasing the filter length, the minimum word length needed for a feasible result decreases. However, after a certain limit, no reduction in word length is achievable upon further increase of filter length. Simulations results of the ASIC design given in [36] show that the hardware complexity and power consumption are least for the minimum word length design. To corroborate the findings, the analysis that an larger word length amounts to a larger width of the adders and registers. Also, for the transposed form of the filter, the latency is not dependent on the filter order but instead on the maximum adder depth of the shift and add network. Thus, designs with the minimum word length are the most preferred designs. The conclusion from the above discussion is that there is no closed form formula to determine what is the minimum word length of implementation. However, a good starting point is to design the filter with an order that satisfies the error constraints between $0.8 \times \delta$ and $0.9 \times \delta$. An EWL=8 is taken as an initial guess and if the design process exhibits very far off error values, the filter order is incremented by 2-10 depending on the initial starting order. If, however, increasing the filter order has no effect

Table 5.1: Filter Specifications

Filter	ω_p	ω_s	δ_p	δ_s
G1	0.2π	0.5π	0.01	0.01
Y1	0.3π	0.5π	0.00316	0.00316
Y2	0.3π	0.5π	0.001	0.001
Y3	0.4π	0.51π	0.01	0.001
A	0.125π	0.225π	0.01	0.01
S2	0.042π	0.14π	0.012	0.001
L2	0.2π	0.28π	0.028	0.001
B	0.2π	0.24π	0.01	0.01
L1	0.8π	0.74π	0.0057	0.0001
C	0.125π	0.14π	0.005	0.005

than the EWL is increased by one and the process is repeated.

5.2.2 Design Examples

Design of Prototype Filters

In this section the effectiveness of the algorithm has been shown by designing benchmark filters available in literature. All the algorithms have been designed in MATLAB . The linear programs are solved using open source code available at [27]. Six filters (G1, Y1, A, B, L1, C) from Table 5.1 have been designed to show the working of the algorithm. The filters have been chosen in such a way that they cover all filter length ranges. The programs are run on a laptop using an i5 2nd generation processor with 4 GB RAM on a Windows 10 platform. The designs have been performed for adder minimization and no constraint on adder depth has been considered.

Table 5.3 and Table 5.4 show the design results for filter G1. Two designs have been done, one with $EWL = 6$ and another with $EWL = 7$. Design with $EWL = 6$ has a total of 17 adders (2 MBAs and 15 SAs) while the design with $EWL = 7$ has a total of 15 adders (2 MBAs and 13 SAs). The total number of adders can be calculated using Eq.

Table 5.2: Emperical Determination of Filter Order and Wordlength

Filter	MATLAB firpm Est.	Order Req.(δ)	$0.9 \times \delta$ Order Req.	$0.8 \times \delta$ Order Req.	Min. EWL Order	Min. EWL
G1	12	14	14	14	15	6
Y1	25	27	27	28	29	9
Y2	32	33	33	36	37	10
Y3	46	48	49	49	49	11
A	51	55	56	57	58	10
S2	51	57	59	61	59	10
L2	55	60	61	62	62	10
B	97	99	102	104	104	8
L1	112	116	118	120	120	14
C	311	311	321	331	324	10

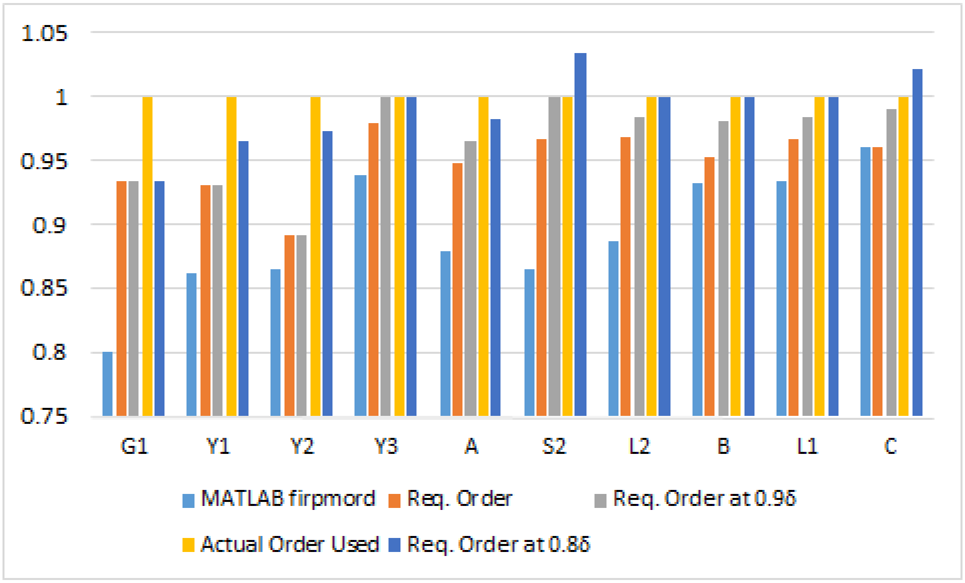


Figure 5.6: Bar Graph Showing Filter Orders

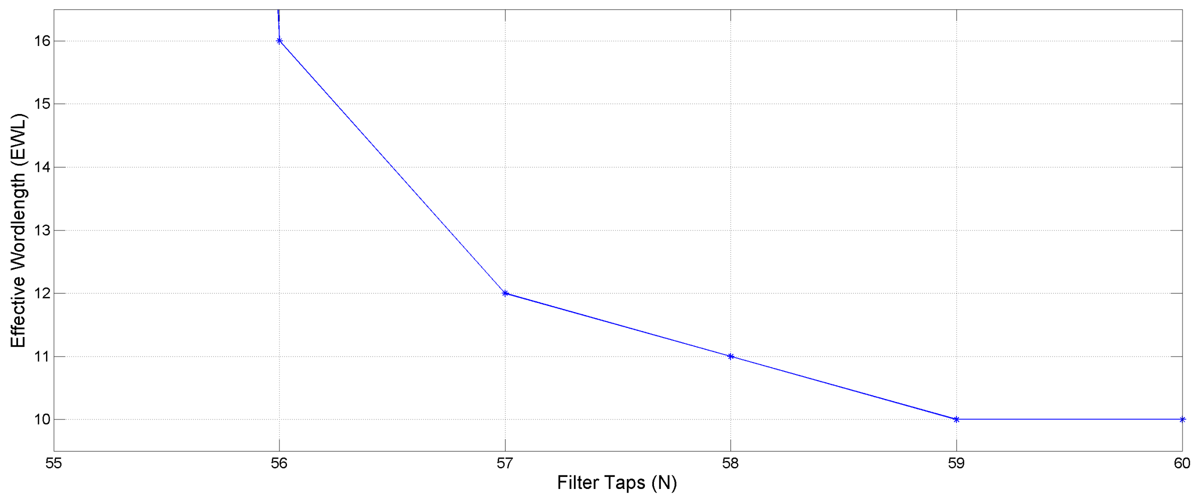


Figure 5.7: EWL vs. Filter Length for A

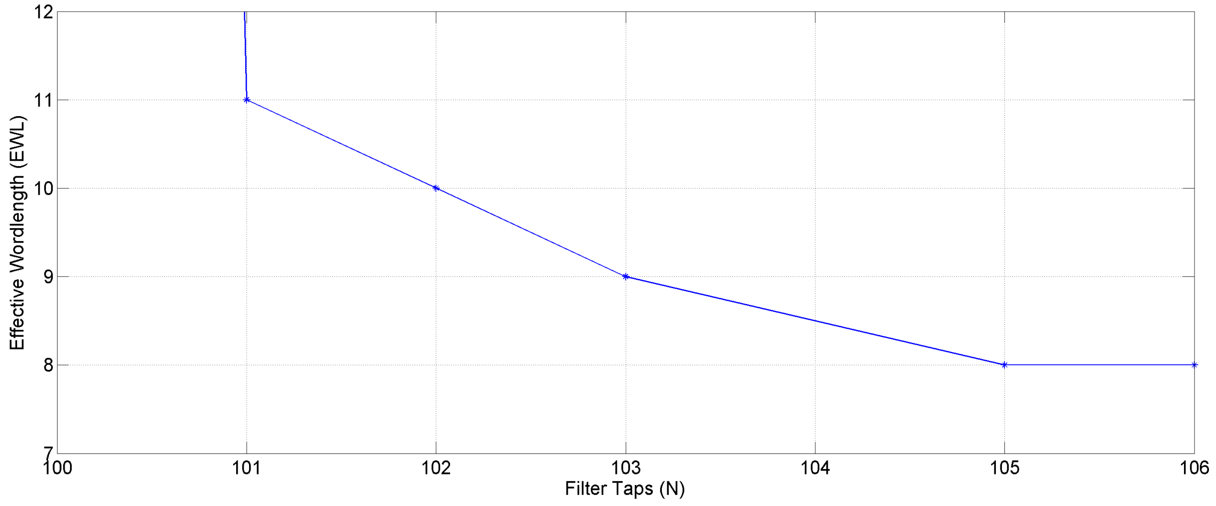


Figure 5.8: EWL vs. Filter Length for B

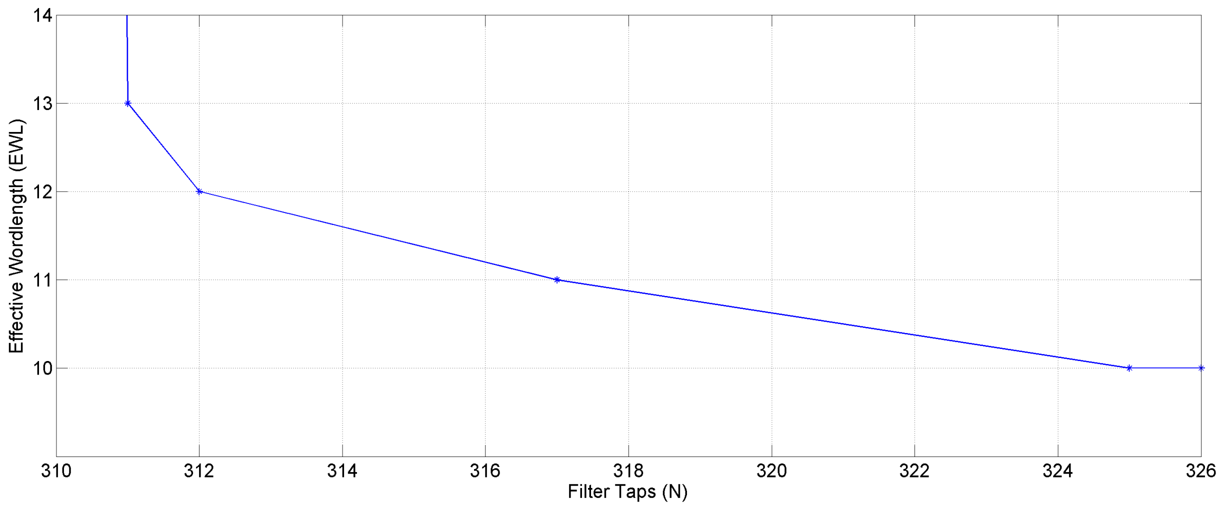


Figure 5.9: EWL vs. Filter Length for C

5.1.

$$C_A = (N - 1) + N_{MBA} - 2N_{zero} \tag{5.1}$$

where N_{MBA} is the number of MBAs, N_{zero} is the number of zero valued coefficients among the unique coefficients.

The reason for the two designs is that both the designs have comparable hardware complexity owing to the small number of filter coefficients. In the previous section (5.2.1) it was mentioned that a minimum EWL is almost always preferred because increasing the word length increases adder width and also the register lengths. However, in this case the higher EWL design has fewer total adders and since there are very few coefficients the two designs have comparable hardware complexity. To differentiate which design is better, an actual ASIC synthesis must be done.

In row 1 of Table 5.3 and 5.4, the passband gain of the filter is given. The passband gain has been calculated according to Eq. 4.6 and no downscaling of the coefficients has been done for the calculation. Row 1 also gives the EWL, maximum adder depth (MAD), the passband ripple error and the stopband ripple error of the design. Row 2 gives the unique filter coefficients and the basis set for constructing the filter coefficients. Finally,

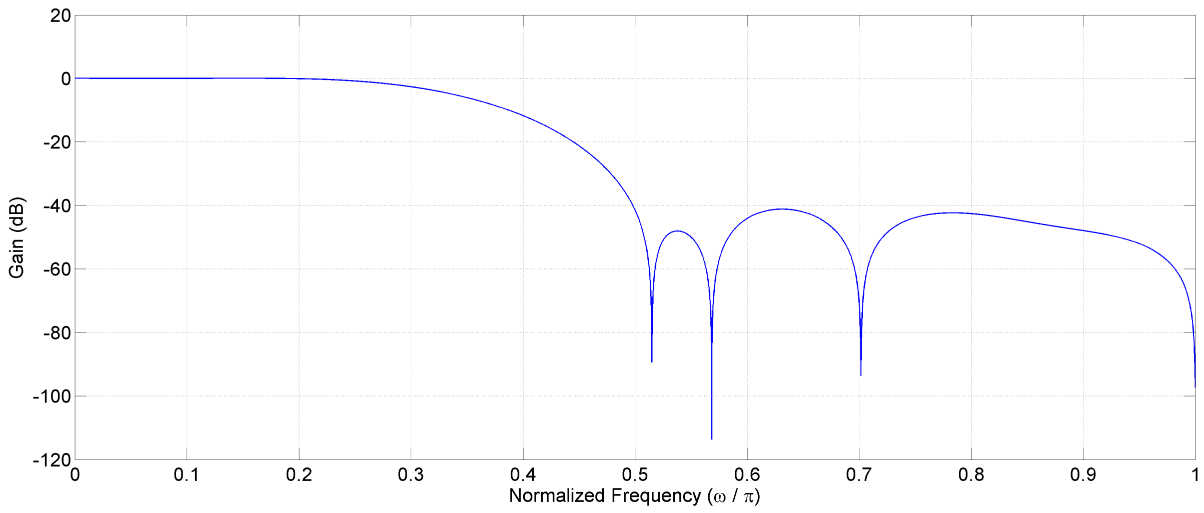


Figure 5.10: Amplitude Response of Filter G1 (EWL=6)

row 3 gives the synthesis of the basis set. Also, all the tables showing design results follow the same pattern.

Table 5.5 and Table 5.6 show the design results for filter Y1. Two designs have been done, one with $EWL = 9$ and another with $EWL = 10$. Similar to filter G1, the reason for the two designs is that both the designs have comparable hardware complexity owing to the small number of filter coefficients. Also, to differentiate which design is better, an actual ASIC synthesis must be done. Filter with $EWL = 9$ has 7 MBAs and 23 SAs while

Table 5.3: Result For Filter G1

Passband Gain=169.062841 EWL=6 MAD=1 $\delta_p = 0.00876$ $\delta_s = 0.00876$		
$h(n) = h(15-n)$ for $0 \leq n \leq 7$ $1, 2, -1, -7, -7, 7, 34, 56$ Basis Set= $\{7, 17\}$		
$7 = 1 \times 2^3 - 1$	$17 = 1 \times 2^4 + 1$	

Table 5.4: Result For Filter G1

Passband Gain=376.99733713731		
EWL=7 MAD=2 $\delta_p = 0.00998$ $\delta_s = 0.00998$		
$h(n) = h(15 - n)$ for $0 \leq n \leq 7$		
$3, 6, 0, -16, -19, 12, 76, 128$		
Basis Set= $\{3, 19\}$		
$3 = 1 \times 2^1 + 1$	$19 = 1 \times 2^4 + 3$	

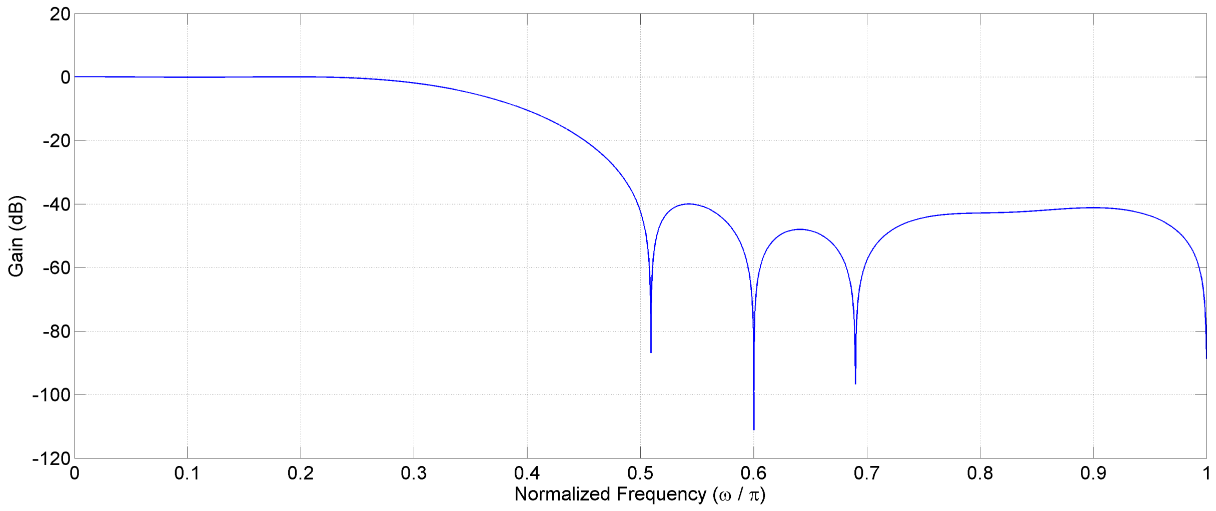


Figure 5.11: Amplitude Response of Filter G1 (EWL=7)

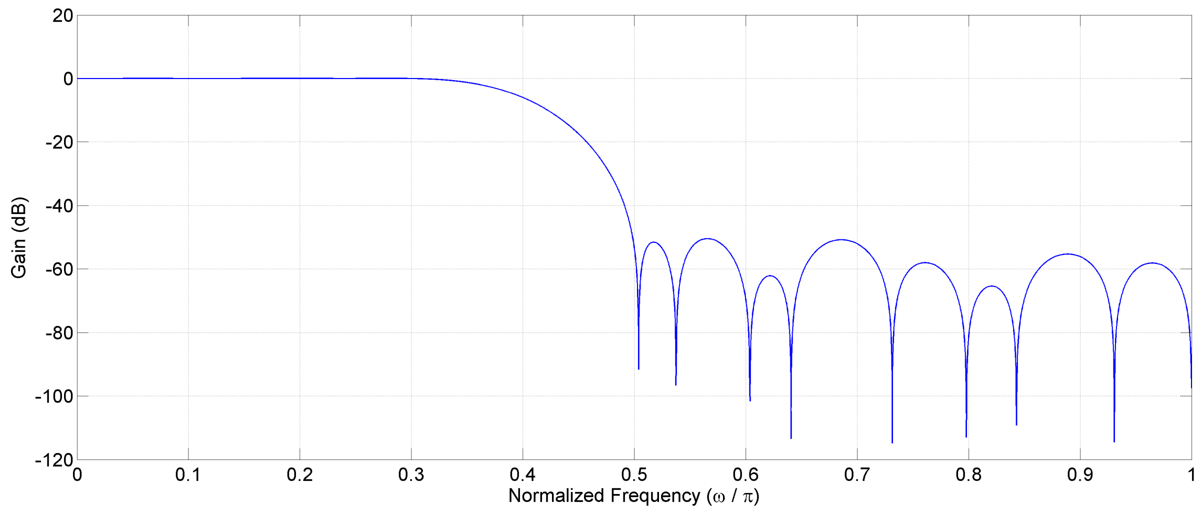


Figure 5.12: Amplitude Response of Filter Y1 (EWL=9)

the design with $EWL = 10$ has an equal number of SAs but has one less MBA. Table 5.7, 5.8, 5.9 and 5.10 show the design results for filter A, B, L1 and C respectively.

Table 5.5: Result For Filter Y1

Passband Gain=1372.11660988306		
EWL=9 MAD=3 $\delta_p = 0.0030002$ $\delta_s = 0.0030002$		
$h(n) = h(29 - n)$ for $0 \leq n \leq 14$		
$-1, -4, 0, 9, 8, -11, -24, 0, 43, 35, -48, -106, 0, 271, 512$		
Basis Set= $\{3, 9, 11, 35, 43, 53, 271\}$		
$3 = 1 \times 2^1 + 1$	$9 = 1 \times 2^3 + 1$	$11 = 1 \times 2^3 + 3$
$35 = 1 \times 2^5 + 3$	$43 = 1 \times 2^5 + 11$	$53 = 1 \times 2^6 - 11$
$271 = 35 \times 2^3 - 9$		

Table 5.6: Result For Filter Y1

Passband Gain=1400.6574034897		
EWL=10 MAD=3 $\delta_p = 0.003029$ $\delta_s = 0.003029$		
$h(n) = h(29-n)$ for $0 \leq n \leq 14$		
$-1, -4, 0, 9, 8, -11, -24, 0, 44, 36, -48, -108, 0, 277, 523$		
Basis Set= $\{3, 9, 11, 27, 523, 277\}$		
$3 = 1 \times 2^1 + 1$	$9 = 1 \times 2^3 + 1$	$11 = 1 \times 2^3 + 3$
$27 = 1 \times 2^4 + 11$	$523 = 1 \times 2^9 + 11$	$277 = 1 \times 2^5 - 11$

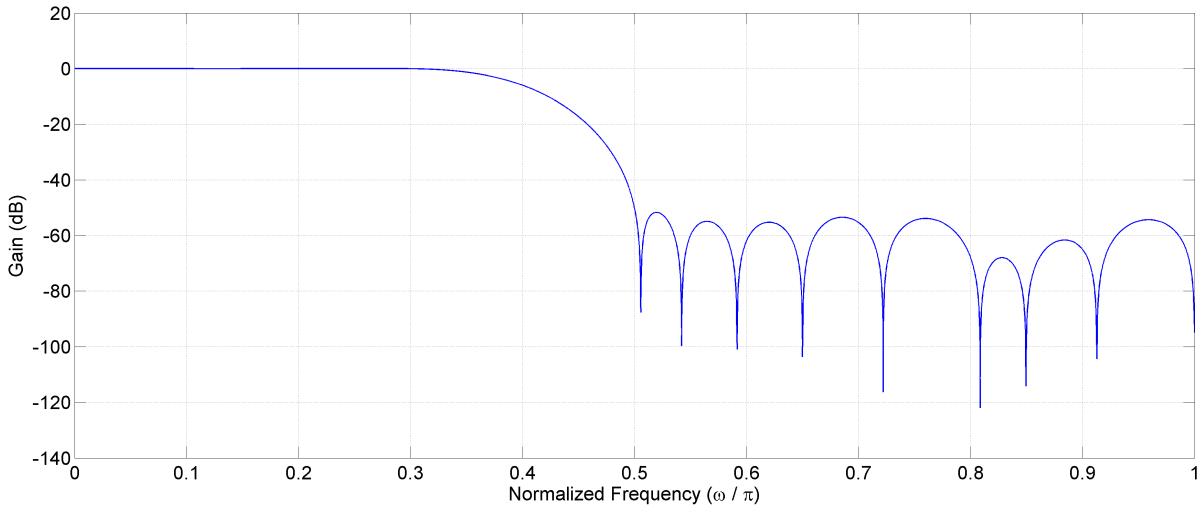


Figure 5.13: Amplitude Response of Filter Y1 (EWL=10)

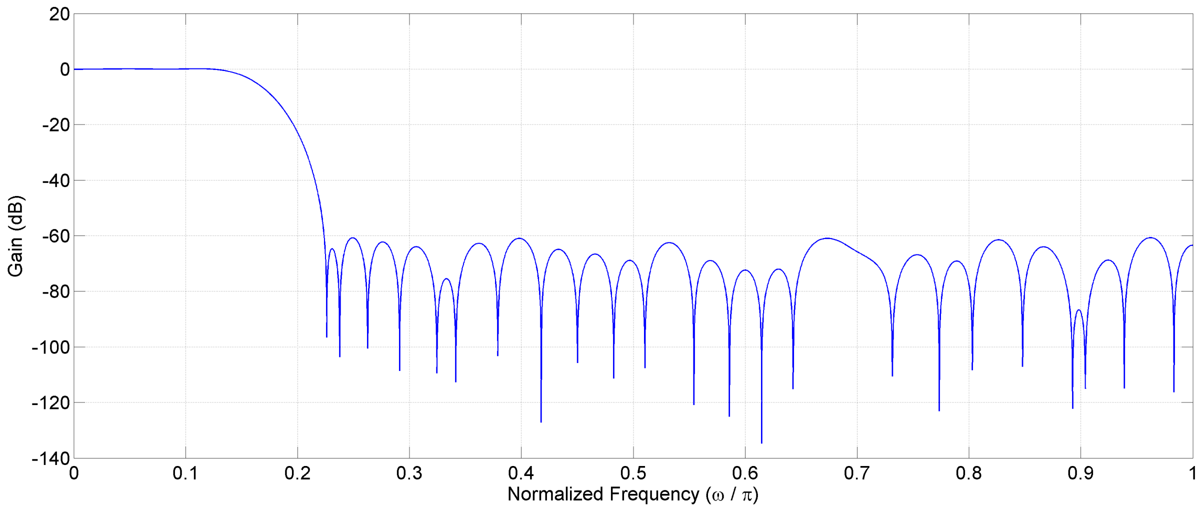


Figure 5.14: Amplitude Response of Filter A

Table 5.7: Result For Filter A

Passband Gain=5930.41818372911		
EWL=10 MAD=3 $\delta_p = 0.009851$ $\delta_s = 0.00009218$		
$h(n) = h(58 - n)$ for $n \leq n \leq 29$		
$3, 5, 7, 6, 3, -4, -14, -24, -31, -31, -20, 0, 28, 55, 73, 73, 49, 0, -65, -130, -176, -179$		
$-124, -4, 175, 392, 616, 812, 945, 992$		
Basis Set= $\{3, 5, 7, 31, 65, 11, 73, 77, 55, 179, 49, 203, 175, 945\}$		
$3 = 1 \times 2^1 + 1$	$5 = 1 \times 2^2 + 1$	$7 = 1 \times 2^3 - 1$
$31 = 1 \times 2^5 - 1$	$65 = 1 \times 2^6 + 1$	$11 = 1 \times 2^3 + 3$
$73 = 1 \times 2^3 + 65$	$77 = 1 \times 2^2 + 73$	$55 = 1 \times 2^6 - 73$
$179 = 1 \times 2^8 - 77$	$49 = 3 \times 2^4 + 1$	$203 = 3 \times 2^6 + 11$
$175 = 7 \times 2^5 - 49$	$945 = 7 \times 2^7 + 49$	

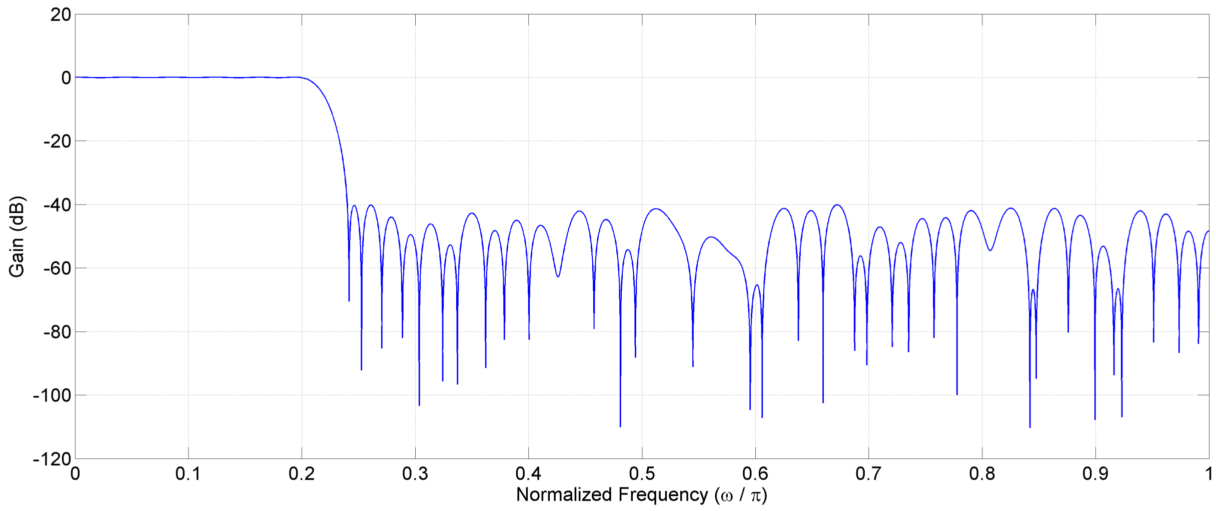


Figure 5.15: Amplitude Response of Filter B

Table 5.8: Result For Filter B

Passband Gain=1042.956759041683		
EWL=8 MAD=4 $\delta_p = 0.00999$ $\delta_s = 0.00989$		
$h(n) = h(104-n)$ for $0 \leq n \leq 52$		
$-2, 0, 0, 2, 3, 2, 1, 0, -2, -3, -2, 0, 2, 4, 4, 2, -1, -4, -6, -4, 0, 3, 7, 7, 4$		
$-2, -7, -10, -8, -2, 6, 12, 13, 8, -2, -12, -19, -16, -5, 10, 23, 27, 19, -2, -28, -46$		
$-46, -20, 30, 97, 163, 211, 230$		
Basis Set= $\{3, 5, 7, 15, 13, 23, 27, 97, 19, 115, 211, 163\}$		
$3 = 1 \times 2^1 + 1$	$5 = 1 \times 2^2 + 1$	$7 = 1 \times 2^3 - 1$
$9 = 1 \times 2^3 + 1$	$15 = 1 \times 2^4 - 1$	$13 = 3 \times 2^2 + 1$
$23 = 3 \times 2^3 - 1$	$27 = 3 \times 2^3 + 3$	$97 = 3 \times 2^5 + 1$
$115 = 3 \times 2^5 + 19$	$211 = 3 \times 2^6 + 19$	$163 = 3 \times 2^4 + 115$

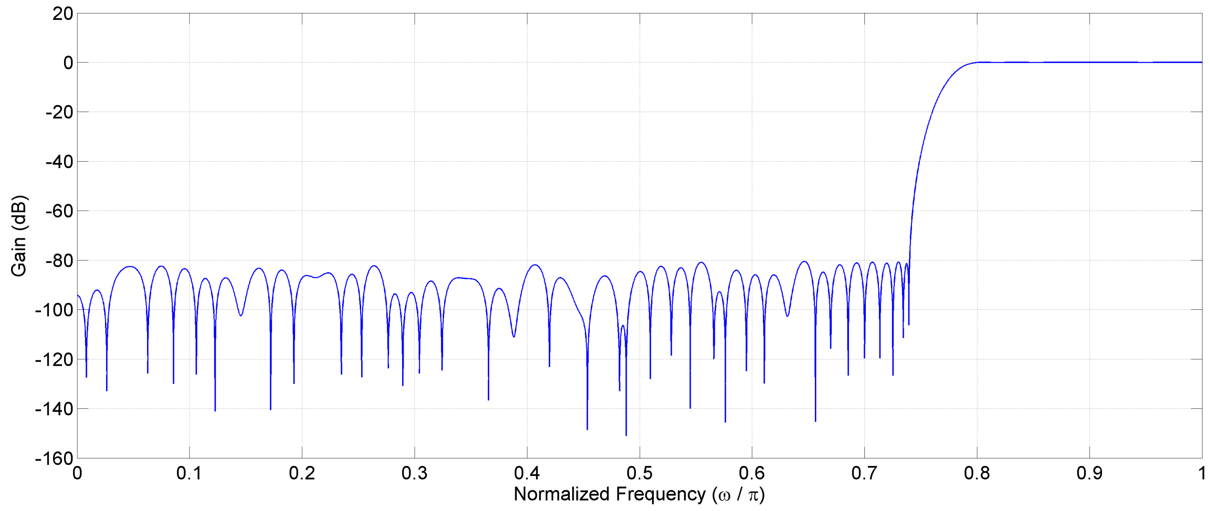


Figure 5.16: Amplitude Response of Filter L1

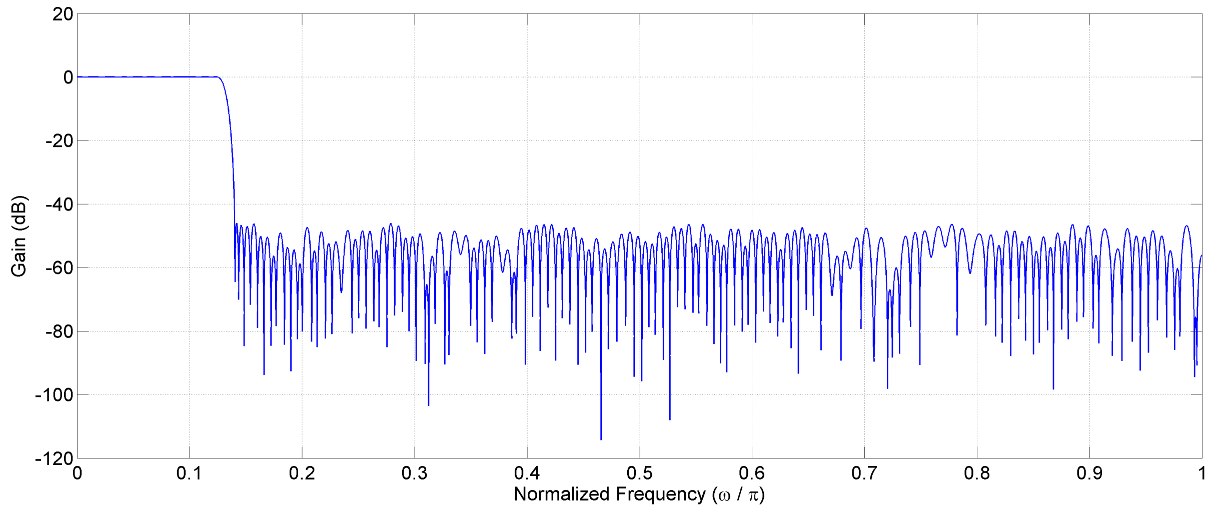


Figure 5.17: Amplitude Response of Filter C

Table 5.9: Result For Filter L1

Passband Gain=51380.59748058755

EWL=10 MAD=6 $\delta_p = 0.005368$ $\delta_s = 0.00009417598$

$h(n) = h(120 - n)$ for $0 \le n \le 60$

$3, -8, 14, -18, 16, -7, -9, 26, -36, 31, -9, -24, 54, -65, 47, 0, -58, 101, -104, 58, 24$
$-106, 147, -117, 20, 107, -203, 212, -115, -58, 230, -313, 249, -46, -214, 406, -421, 221$
$126, -466, 622, -484, 72, 449, -829, 846, -423, -307, 1028, -1363, 1049, -89, -1189, 2210$
$-2350, 1187, 1297, -4650, 8056, -10589, 11525$

Basis Set= $\{3, 5, 7, 9, 31, 63, 65, 257, 13, 29, 27, 23, 121, 249, 47, 449, 115, 101, 233$
$313, 117, 147, 53, 203, 89, 423, 221, 307, 311, 107, 421, 1189, 1175, 1187, 1105, 1297, 1007, 829$
$1049, 2325, 10589, 1363, 11525\}$

$3 = 121 + 1$	$5 = 122 + 1$	$7 = 123 - 1$
$9 = 123 + 1$	$31 = 125 - 1$	$63 = 126 - 1$
$65 = 126 + 1$	$257 = 128 + 1$	$13 = 124 - 3$
$29 = 125 - 3$	$27 = 125 - 5$	$23 = 124 + 7$
$121 = 127 - 7$	$249 = 128 - 7$	$47 = 124 + 31$
$449 = 129 - 63$	$115 = 127 - 13$	$101 = 127 - 27$
$233 = 128 - 23$	$313 = 126 + 249$	$117 = 121 + 115$
$147 = 125 + 115$	$53 = 324 + 5$	$203 = 128 - 53$
$89 = 325 - 7$	$423 = 129 - 89$	$221 = 326 + 29$
$307 = 326 + 115$	$311 = 122 + 307$	$107 = 321 + 101$
$421 = 526 + 101$	$1189 = 328 + 421$	$1175 = 927 + 23$
$1187 = 322 + 1175$	$1105 = 927 - 47$	$1297 = 326 + 1105$
$1007 = 928 - 1297$	$829 = 6325 - 1187$	$1049 = 6524 + 9$
$2325 = 25722 + 1297$	$10589 = 2322 + 1187$	$1363 = 4722 + 1175$
$11525 = 11723 + 10589$		

Table 5.10: Result For Filter C

Passband Gain=5057.21580045042 EWL=10 MAD=3 $\delta_p = 0.00498$ $\delta_s = 0.00498$		
$h(n) = h(324 - n)$ for $0 \leq n \leq 162$ $-4, -3, 0, 0, 1, 2, 3, 0, 3, 3, 0, 0, 0, 0, -3, -3, -3, 0, 0, 0, 2, 3, 4, 3, 2, 3, 0, 0, -3, -4, -4, -3$ $-30, 0, 3, 4, 5, 5, 4, 3, 0, 0, -4, -5, -6, -5, -4, -2, 0, 3, 5, 7, 7, 6, 4, 0, -2, -5, -7, -8, -8$ $-6, -3, 0, 4, 7, 9, 10, 8, 6, 2, -3, -7, -10, -12, -11, -9, -5, 0, 5, 10, 13, 14, 13, 9, 3, -3, -9$ $-14, -16, -16, -13, -8, 0, 7, 14, 18, 20, 18, 13, 5, -3, -12, -19, -23, -23, -20, -12, -2$ $8, 19, 26, 28, 27, 19, 9, -3, -17, -27, -34, -35, -29, -19, -4, 12, 27, 39, 44, 43, 32, 16, -4$ $-26, -43, -55, -59, -51, -35, -10, 19, 47, 71, 83, 82, 67, 36, -4, -50, -94, -128, -144,$ $-137, -100, -38, 52, 160, 280, 400, 509, 596, 651, 670$ Basis Set= $\{3, 5, 7, 9, 11, 13, 17, 19, 23, 25, 27, 29, 35, 39, 41, 43, 47, 51, 55, 59, 67, 71, 83, 137$ $149, 335, 509, 651\}$		
$3 = 1 \times 2^1 + 1$	$5 = 1 \times 2^2 + 1$	$7 = 1 \times 2^3 - 1$
$13 = 3 \times 2^2 + 1$	$9 = 1 \times 2^3 + 1$	$17 = 1 \times 2^4 + 1$
$23 = 3 \times 2^3 - 1$	$11 = 3 \times 2^2 - 1$	$25 = 3 \times 2^3 + 1$
$51 = 3 \times 2^4 + 3$	$27 = 3 \times 2^3 + 3$	$47 = 3 \times 2^4 - 1$
$43 = 3 \times 2^4 - 5$	$29 = 17 \times 2^1 + 115$	$19 = 3 \times 2^3 - 5$
$39 = 3 \times 2^4 - 9$	$55 = 3 \times 2^4 + 7$	$41 = 3 \times 2^4 - 7$
$59 = 3 \times 2^4 + 11$	$35 = 3 \times 2^4 - 13$	$83 = 59 \times 2^5 - 13$
$149 = 3 \times 2^6 - 43$	$71 = 3 \times 2^5 - 25$	$67 = 3 \times 2^5 - 25$
$335 = 9 \times 2^5 + 47$	$137 = 3 \times 2^6 - 55$	$651 = 5 \times 2^7 + 11$
$509 = 9 \times 2^6 - 67$		

Table 5.11: Band Pass and Band Stop Specifications

Filter	Ω_p	Ω_s	δ_p	δ_s
Band Pass	$[0.24\pi, 0.5\pi]$	$[0, 0.2\pi] \cup [0.55\pi, \pi]$	0.01	0.005, 0.005
Band Stop	$[0, 0.06\pi] \cup [0.25\pi, \pi]$	$[0.1\pi, 0.2\pi]$	0.01,0.01	0.01

Self Design Examples

In this subsection, two design examples are given to show the robustness of the algorithm. One example is a bandpass filter and the other is a bandstop filter. The specifications are given in Table 5.11 for the filters.

For the bandpass filter, the number of filter coefficients is 113 and the EWL=9. These order and word length were decided after going through many iterations of designs. It was observed no feasible solution existed for EWL=8. For EWL=10, filter lengths of 112 gave feasible solutions. However, since minimum EWL design is preferred, the filter was implemented with EWL=9 and filter length of 113. The number of MBAs= 16 and the number of SAs=104 as there are 4 zero valued coefficients. Thus, the total number of adders are 120.

For the bandstop filter, the number of filter coefficients is chosen to be 105 and the EWL=10. The number of MBAs=13 and the number of SAs=98 as 3 coefficients among the unique coefficients are zero valued. Thus, the total number of adders for the bandstop filter is 111.

Table 5.12: Result For Band Pass Filter

Passband Gain= 1617.61973197782
EWL=9 MAD=2 $\delta_p = 0.00100$ $\delta_s = 0.0050$

$h(n) = h(112 - n)$ for $0 \leq n \leq 56$
$-3, -2, 1, 2, 2, 3, 2, -3, -5, -2, 0, -2, 1, 8, 6, -3, -4, -1, -6, -10, 1, 13, 7, 0$
$6, 3, -16, -20, 0, 10, 0, 8, 27, 11, -24, -22, -1, -13, -21, 22, 55, 17, -20, 2, -3$
$-70, -66, 42, 80, 10, 32, 126, -3, -315, -296, 183, 495$

Basis Set= $\{1, 3, 5, 7, 17, 33, 63, 11, 13, 35, 21, 37, 27, 55, 183, 315, 495\}$

$3 = 1 \times 2^1 + 1$	$5 = 1 \times 2^2 + 1$	$7 = 1 \times 2^3 - 1$
$17 = 1 \times 2^4 + 1$	$33 = 1 \times 2^5 + 1$	$63 = 1 \times 2^6 - 1$
$11 = 1 \times 2^3 + 3$	$13 = 1 \times 2^4 - 3$	$35 = 1 \times 2^5 + 3$
$21 = 1 \times 2^4 + 5$	$37 = 1 \times 2^5 + 5$	$27 = 1 \times 2^5 - 5$
$55 = 3 \times 2^4 + 7$	$183 = 1 \times 2^7 + 55$	$315 = 5 \times 2^6 - 5$
$495 = 33 \times 2^4 - 33$		

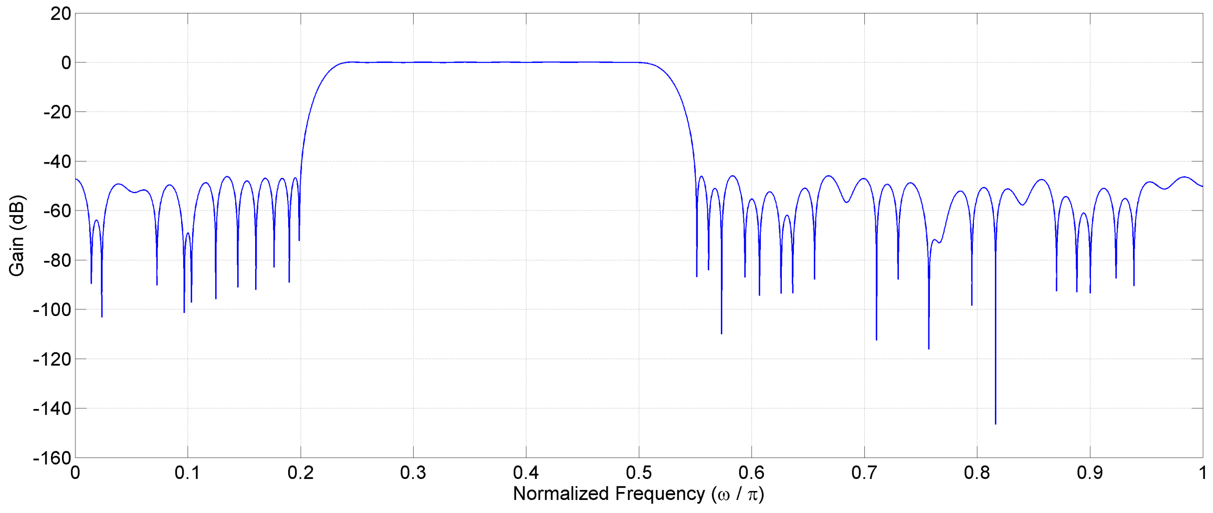


Figure 5.18: Amplitude Response of Band Pass Filter

Table 5.13: Result For Band Stop Filter

Passband Gain= 1125.913228724032
EWL=10 MAD=4 $\delta_p = 0.00979387$ $\delta_s = 0.00979387$

$h(n) = h(104 - n)$ for $0 \leq n \leq 52$
$3, 2, 1, 0, -2, -4, -5, -4, -2, -1, 0, -1, -2, -4, -5, -4, 0, 5, 9, 12, 11, 7, 2, -1, -1$
$2, 7, 10, 8, 1, -10, -22, -29, -28, -20, -7, 3, 7, 2, -9, -19, -19, -5, 26, 64, 98, 111, 95$
$48, -19, -90, -144, 962$

Basis Set= $\{1, 3, 5, 7, 9, 11, 19, 13, 29, 45, 49, 95, 111, 481\}$

$3 = 1 \times 2^1 + 1$	$5 = 1 \times 2^2 + 1$	$7 = 1 \times 2^3 - 1$
$9 = 1 \times 2^3 + 1$	$11 = 1 \times 2^3 + 3$	$19 = 1 \times 2^4 + 3$
$13 = 1 \times 2^4 - 3$	$29 = 1 \times 2^5 - 3$	$45 = 1 \times 2^6 - 19$
$49 = 1 \times 2^2 + 45$	$95 = 3 \times 2^5 - 1$	$111 = 1 \times 2^4 + 95$
$481 = 9 \times 2^6 - 95$		

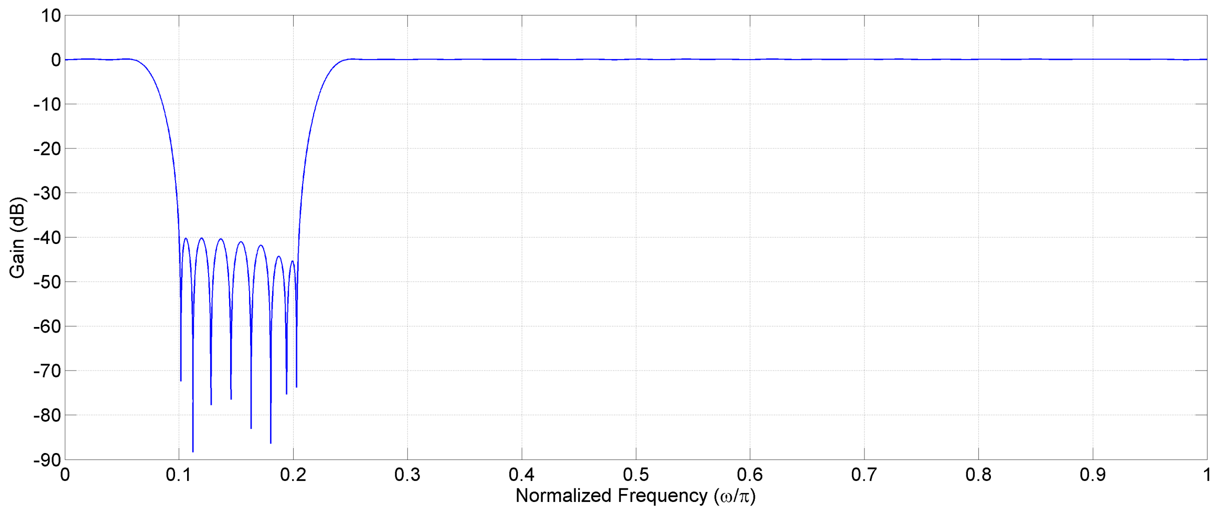


Figure 5.19: Amplitude Response of Band Stop Filter

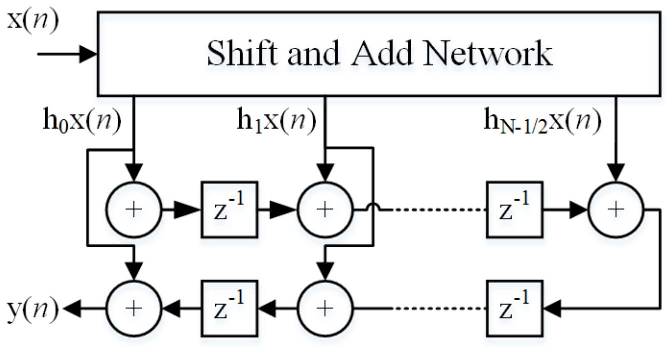


Figure 5.20: Transposed Direct Form of Linear Phase FIR Filter with Multipliers Replaced by Shift and Add Network

5.3 Hardware Synthesis

5.3.1 Structure of Filter and Shift Add Network

The hardware for the filters is synthesized in the transposed direct form. In the transposed direct form, the multiplier array is replaced by a shift and add network (Figure 5.20). Figure 5.21 shows the expanded form of the filter coefficients with the synthesis mechanism of filter G1 whose design is given in Table 5.4. Figure 5.22 shows the synthesis of the shift and add network. Similarly, figure 5.23 shows the synthesis of the shift and add network of filter Y1 whose design is given in Table 5.6. From the figures, it is seen that the latency of the filter is dependent on the maximum adder depth and not on the filter order. Thus the use of increasing the filter order in order to decrease the effective word length is justified.

5.3.2 Filter Adders' Topology

The circuit topology at a lower abstraction level is described in [38]. For generating the fundamentals, an multiplier block adders are employed as shown in figures 5.22 and 5.23. However, each adder has a different structure based on the fundamental and the generating

Figure 5.21: Expanded Form of Filter G1 Synthesis

$h(0)=$	$1 \times 2^1 + 1$	$=h(15)$
$h(1)=$	3×2^1	$=h(14)$
$h(2)=$	0	$=h(13)$
$h(3)=$	$-(1) \times 2^4$	$=h(12)$
$h(4)=$	$-(1 \times 2^4 + 3)$	$=h(11)$
$h(5)=$	3×2^2	$=h(10)$
$h(6)=$	19×2^2	$=h(9)$
$h(7)=$	1×2^8	$=h(8)$

mechanism. The width of output (B_{out}) is calculate according to equation 5.2.

$$B_{out} = B_{in} + \lceil log_2|f_j| \rceil \tag{5.2}$$

where B_{in} is the word length of the input and f_j is the fundamental being generated. Figure 5.24 shows the ripple carry adder topology for the case when the fundamental is generated in the form $(a \times 2^n + b)$. It is seen that in this case, the first n outputs are directly obtained from the non-scaled input. Thus in this case, the number of full adders (FA) is given by Eq. 5.3.

$$N^j_{FA,MBA} = B_{in} + \lceil log_2|f_j| \rceil - n^j \tag{5.3}$$

where n^j is the index of the power of two for the jth fundamental.

Figure 5.25 shows the ripple carry adder topology for the case when the fundamental is generated in the form $(a \times 2^n - b)$. It is seen that in this case, the first n outputs are obtained using a half adder and also each bit of the input b is inverted and a 1 is added a carry in to represent the number in 2s complement form. In this case, there is an additional hardware overhead in the form of n half adders (HA) and B_{out} NOT gates. To overcome the extra HAs, the fundamental can be implemented in negative form i.e. $(-a \times 2^n + b)$. The negative output can either be negated to obtain the actual fundamental or the next adder utilizing this fundamental can take into account its negation and form further fundamental accordingly. In either case, the cost of using HAs is eliminated and

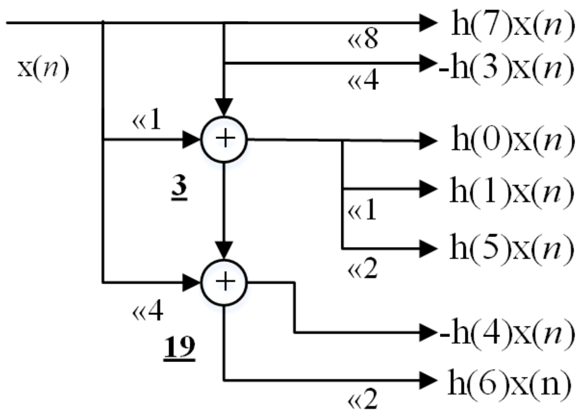


Figure 5.22: Hardware Synthesis for Filter G1 (EWL=7)

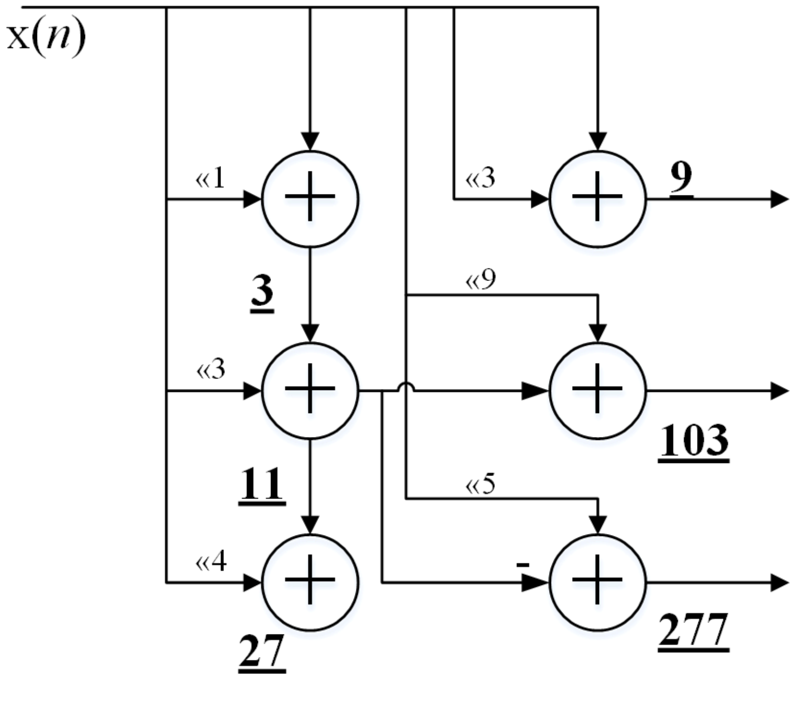


Figure 5.23: Synthesis of Shift Add Network for Filter Y1 (EWL=10)

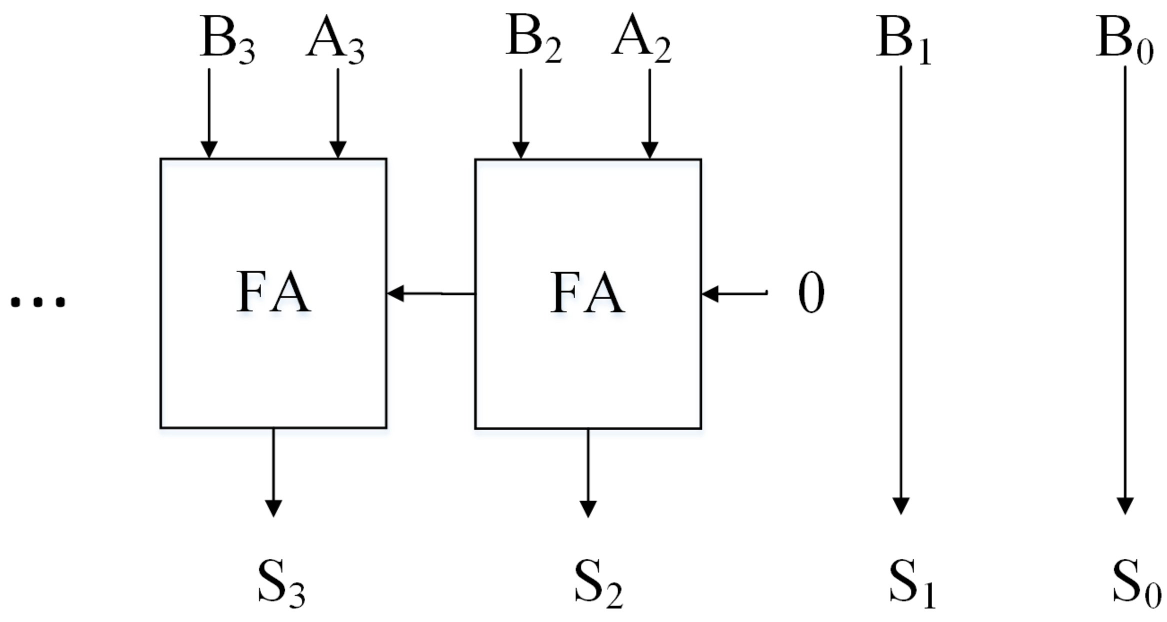


Figure 5.24: Ripple Carry Adder Topology for $(a \times 2^n + b)$ for $n = 2$

as estimate on the number of FAs for both the $(a \times 2^n + b)$ and $(a \times 2^n - b)$ topologies is given by Eq. 5.3. For the shift and add network given in Figure 5.22, the number of FAs for the MBAs is calculated as follows assuming an input bit width of 8.

$$N_{FA}^{3} = 8 + \lceil log_2(3) \rceil - 1 = 9$$

$$N_{FA}^{19} = 8 + \lceil log_2(19) \rceil - 4 = 9$$

The structural adders also employ ripple carry adders for accumulating the scaled and delayed inputs. For the SAs, at each stage, the width of the output increases. For each SA corresponding to a filter coefficient $h(k)$, the number of FAs needed is given by Eq. 5.4.

$$N_{FA,SA}^{k} = B_{in} + \lceil log_2 \sum_{i=0}^{k} |h(i)| \rceil - n^k \tag{5.4}$$

where n^k is the index of the power of two needed to generate the filter coefficient from the

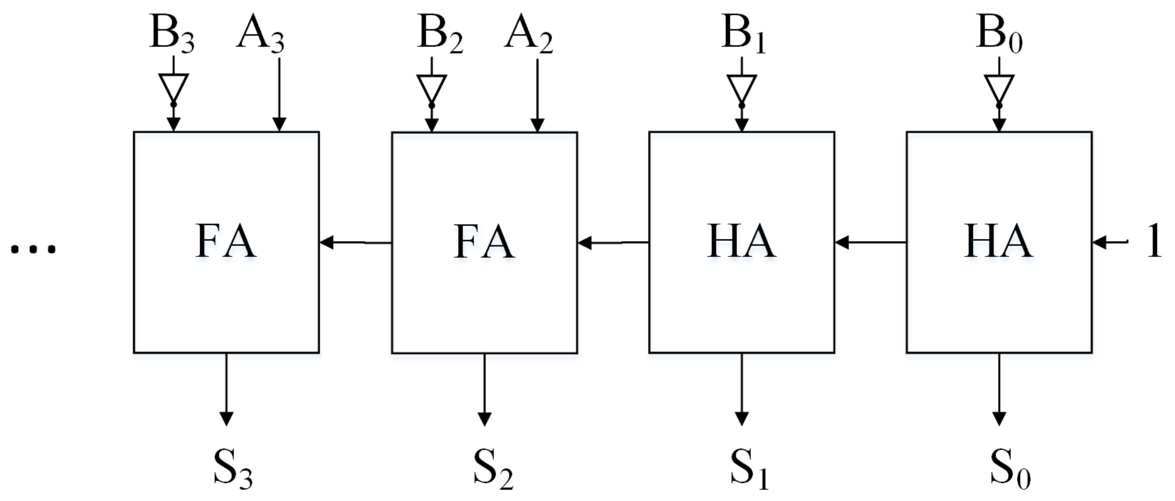


Figure 5.25: Ripple Carry Adder Topology for $(a \times 2^n - b)$ for $n = 2$

corresponding fundamental.

Table 5.14 gives the FA count for the designed examples from literature. $N_{FA,MBA}$ is the number of FAs used in all the MBAs, N_{MBA} is the number of MBAs in the shift and add network, $N_{FA,SA}$ is the number of the FAs used in building all the structural adders, N_{SA} is the count of the number of structural adders used, and $N_{FA,T}$ is the total number of FAs used in constructing the entire filter. It is to be noted that the first coefficient does not use a SA and zero valued coefficients also don't employ a SA.

From Table 5.14 it is seen that the length of MBA is independent of the length of the filter and depends on the input bit width (taken as 8). It is found that on average, the number of FAs per MBA is 9. For SAs, the length is heavily dependent on the position of the SA as the length progressively increases. The average length of each adder is highly dependent on the number of filter coefficients. For small filters (G1, Y1) it is found to average 14, for medium to large (A,B) it averages around 17 while for large filters (L1,c), it averages around 19. Thus the ratio of hardware complexity of a SA to an MBA is 1.5 for short filters, 1.8 for medium length filters and 2.0 for large length filters.

Table 5.14: Full Adder Count for Filters

Filter	EWL	$N_{FA,MBA}$	N_{MBA}	$N_{FA,SA}$	N_{SA}	$N_{FA,T}$
G1	6	17	2	205	15	222
G1	7	17	2	175	13	192
Y1	9	66	7	356	23	422
Y1	10	56	6	366	23	422
A	10	133	14	974	54	1107
B	8	107	12	1503	94	1610
L1	14	462	43	2509	118	2971
C	10	252	28	5219	282	5471

5.4 Result Comparison

In this section, the design results of section 5.2.2 are compared with the state of the art algorithms present in literature. Firstly a comparison with approximate algorithms is done. Next, a comparison with the latest deterministic methods is done.

Figure 5.26 shows the design results of filter A when compared to the infinite precision Parks-McClellan method while figure 5.27 shows the comparison with a simply quantized response. The quantization has been done such that the highest magnitude coefficient can be represented in the same amount of bits as the designed filter. It is observed that the amplitude response of the finite word length filter is very close to the infinite precision design. Also, simply quantizing the coefficients results in a filter that violates the filter specifications and a larger word length must be employed for the quantized design in order for it to meet the specification.

5.4.1 Adder Cost Comparison: Approximate Methods

The approximate methods that have been taken for comparison are the genetic algorithm based design (Ye1) found in [25], the NAIAD method described in [37], and the POTx method of [33]. The papers chosen for comparison are the latest available papers utilizing

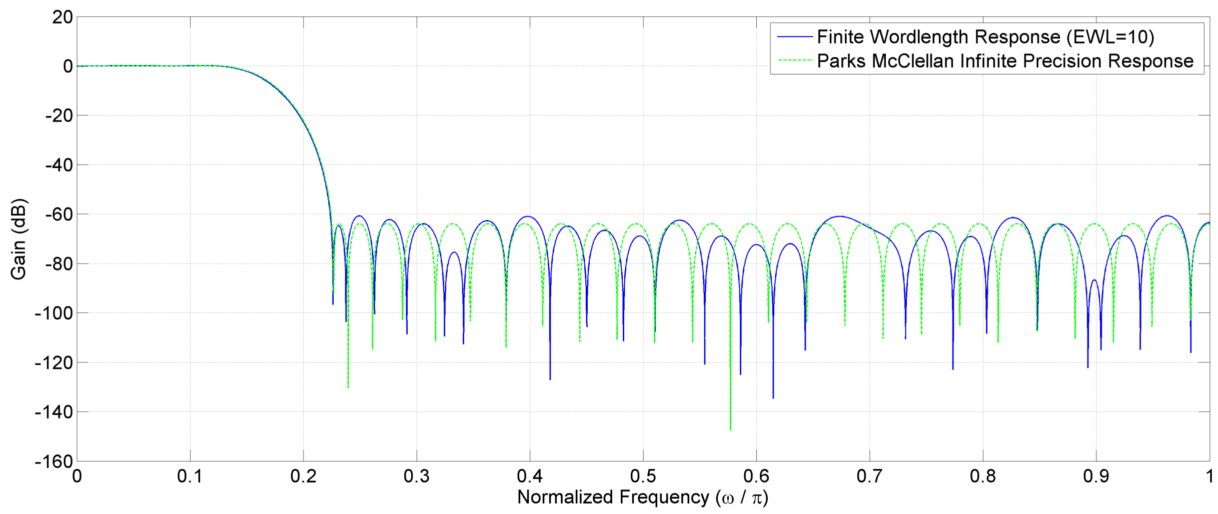


Figure 5.26: Comparison of Amplitude Response of Opitmal Finite Wordlength Design With Infinite Precision Parks McClellan Design for Filter A

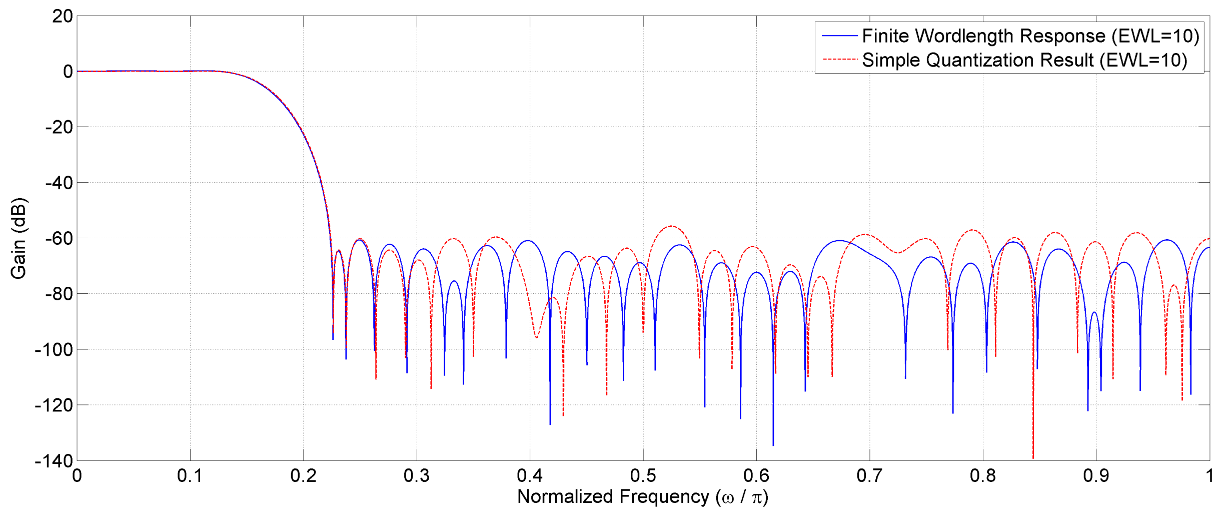


Figure 5.27: Comparison of Amplitude Response of Opitmal Finite Wordlength Design With Simply Quantized Design for Filter A

approximate algorithms. The POTx algorithm uses a heuristic method to locally search in the neighborhood of a quantized coefficient. A similar approach is given for the NAIAD algorithm. Since local search algorithms are more adept at designing filters with large number of filter taps, the comparison has been made for filters A, B, L1 and C. In Table 5.15, N is the filter length, EWL is effective word length (the number of bits needed to represent the filter coefficients excluding the sign bit), MBA is the number of multiplier block adders, SA is the number of structural adders. The last column shows the total time for running the algorithm as given in the respective publications. For the proposed algorithm, the time given is the time to run the Differential Evolution part of the algorithm for one gain section since each section can be optimized independently. It is to be noted that the hardware complexity of a SA is approximately 1.5-2.0 times higher than a MBA as mentioned in section 5.3.2 Thus, when comparing two designs, the higher weight of a SA must be taken into consideration. A more detailed discussion on the actual weights of SAs and MBAs was given in section 5.3.2.

As seen in Table 5.15, the proposed method outperforms all the compared methods in all the filter design examples. For filter B, the methods of Ye1 and POTx have lesser number of MBAs but since the proposed design has more zero valued coefficients and hence far fewer structural adder giving a total cost of hardware implementation to be better than all other methods. Same reasoning can be made for the design of filter C where the NAIAD and POTx methods have fewer MBAs but have larger structural adder count. Hence, the proposed method has a much lower hardware cost. However, for all the designs compared in Table 5.15, the proposed method has the least number of SAs (with the exception of L1). Even though the design of NAIAD for filter L1 has less number of total adders and structural adders, it can be seen that the EWL is greater than the proposed design. Since hardware complexity heavily depends on the EWL, thus the proposed design exhibits less hardware complexity. Thus, it is concluded that the inclusion of a sparse technique into the multiplierless design approach goes a long way into reducing the hardware complexity.

Table 5.15: Comparison With Approximate Methods

Filter	Method	N	EWL	MBA	SA	Time
A	**Prop.**	59	10	**14**	**54**	10m
	NAIAD	59	10	16	56	44m
	POTx	59	10	16	58	1.49m
B	**Prop.**	105	8	**12**	**94**	18m
	NAIAD	105	8	13	96	2h2m
	Ye1	105	8	10	98	16m24s
	POTx	105	8	10	99	1.13m
L1	**Prop.**	121	14	**43**	**118**	2h5m
	NAIAD	121	15	40	116	3h48m
	Ye1	121	14	43	120	26m41s
C	**Prop.**	325	10	**28**	**282**	1h15m
	NAIAD	325	10	25	286	4h
	Ye1	325	10	31	306	3h49m
	POTx	325	10	24	301	28.7m

5.4.2 Adder Cost Comparison: Deterministic Methods

The exact methods that have been taken for comparison are FIRGAM method found in [36], the SIREN method described in [37], the method by Shi [22], the method by Yao [23] and the Ye2 method of [24]. The papers chosen for comparison are the latest available papers and have a runtime of less than 24 hours [1]. In Table 5.16 N, EWL, MBA, SA and TA have the same meaning as in Table 5.15. Also, the time given is the timings reported in the respective publications.

As seen in 5.16, the proposed method is very competitive with the state of the art deterministic algorithms. In all cases except for filter L1 and B, the number of adders found is in comparison with the best available method. Since deterministic methods have a exponential time complexity, the proposed method is advantageous as it has non-exponential time

[1]A cap of 24hrs was made as deterministic graph based methods can always produce globally optimum solution provided enough time is given. Methods such as graph based depth first search (Shi [22]) and MILP ([20]) can take upto years to design filter such as example C.

Table 5.16: Comparison With Deterministic Methods

Filter	Method	N	EWL	MBA	SA	Time
G1	**Prop.**	16	7	**2**	**13**	**31s**
	Prop.	16	6	2	15	59s
	SIREN	16	6	**2**	**15**	<1s
	Shi	16	7	**2**	**13**	<1s
	Shi	16	6	**2**	**15**	<1s
Y1	**Prop.**	30	9	**7**	**23**	**1m**
	Prop.	30	10	6	23	2m
	SIREN	30	9	6	23	7m56s
	Shi	30	10	6	23	6s
	Shi	30	9	7	23	5m9s
A	**Prop.**	59	10	**14**	**54**	**10m**
	Ye2	59	10	14	54	8m
	Yao	59	10	14	54	2h
	FIRGAM	59	10	18	58	56m
B	**Prop.**	105	8	**12**	**94**	**18m**
	Ye2	105	8	12	94	31m
	Yao	105	8	11	94	>24h
	FIRGAM	105	8	11	100	1h41m
L1	**Prop.**	121	14	**43**	**118**	**2h5m**
	Ye2	121	14	42	118	33m
	Yao	121	14	41	120	>24h
	FIRGAM	121	14	47	120	56m
C	**Prop.**	325	10	**28**	**282**	**1h15m**
	Ye2	325	10	28	282	5h17m
	Yao	325	10	20	304	>24h
	FIRGAM	325	10	22	304	5h11m

complexity owing to the heuristic nature of the algorithm. The time complexity analysis is discussed more in detail in section 5.5.

5.5 Design Algorithm Complexity Analysis

As discussed briefly in chapter 3 section 3.1, in theoretical computer science, algorithms are compared on the basis of their worst case complexity and the $\mathbb{O}$-notation is frequently used. The comparison is done independent of the implementation language and the hardware used to run the algorithm.

The design procedure for small filters consists of finding the range of each coefficient for unity gain, upscaling that range to fit into the integer space, sectioning the passband gain and running a local search Differential Evolution algorithm in that section. Once, the results for each section are obtained, the best among them is the output of the design result.

Among the operators of Differential Evolution, the selection operator is the most time consuming and is the bottleneck of the algorithm. Thus, many strategies have been implemented to reduce the time consumption. Firstly, as discussed in Chapter 4 section 4.2.6, the amplitude response value of the coefficients corresponding to the lowest limit of the quantized range is stored. Since the amplitude response is a linear function, thus the amplitude response of the difference between the actual coefficient value is added to the lower range value to obtain the actual amplitude response. Thus, this avoids a series of repeated calculations and cuts short the objective function calculation time by a factor of almost 5. Secondly, the adaptive search space reduction technique as discussed in section 4.2.5 expedites the search process by excluding those coefficients whose entire population has converged to a single value. Table 5.17 shows the timings of short filters. T_b is the minimum time in which the result is obtained and the algorithm converged, T_{avg} is the average time of the result to converge, S is the number of gain sections taken, g_{max} is the maximum number of generations ran for each gain or the termination criteria for the Differential Evolution, and T_G is time for running the Differential Evolution for the maximum

Table 5.17: Time Analysis for Small Length Filters

Filter	EWL	T_b	T_{avg}	S	g_{max}	T_G
G1	6	1.76s	4.11s	10	500	31s
G1	7	6.6s	33.1s	10	1000	59s
Y1	9	12.1	45s	30	1500	1.7m
Y1	10	7.51	10.2s	20	1500	2.8m
A	10	-	-	30	5000	10m

generations (g_{max}). The averages have been taken from 30 runs of the program and the population has 100 members for each of the small filters.

For large filters, an extra step in the design procedure is introduced to obtain an initial guess of the solution (called base solution) in each gain section. The reason for this is that a population generated with total randomness for large filters is not able to search the entire space. Thus, an initial hint is given in the form of perturbing the parameters around the base solution to create the population. However, to preserve large random pool of population members, only half the population is generated by perturbing the base solution.

Table 5.18 shows the design statistics for the large filters and is divided into three parts. In the first part, the statistics of the linear program used for obtaining an initial base solution is listed. In column 2, S is the number of gains sections for which the linear program is run, while in column 3 n_{PV} is the number of pseudo-viable solutions(PV) tried. Once the base solution is obtained than a population around the pseudo viable base solution is created as explained in Chapter 4 section 4.2.2. In the second part of the table, the population statistics are given. Pop. is the number of members in the population, Max. bits is the maximum number of bits the neighborhood of the largest valued coefficient is alloted, Percent Pert. is the percentage of the parameters perturbed from the initial base solution found by the linear program to generate half the population. Also, $n_{excl.}$ is the number of coefficients excluded as a result of adaptive sub space reduction. Lastly, in the third part of the table the timings for the Differential Evolution algorithm are given. g_{max} is the maximum number of generations ran for each gain PV solution, T_G is the time taken

Table 5.18: Design Statistics for Large Filters

Filter	S	n_{PV}	Pop.	Max. bits	Percent Pert.	$n_{excl.}$	g_{max}	T_G
B	40	5	200	3	90%	10	10k	18m
L1	50	6	400	4	85%	0	25k	2h5m
C	40	7	200	3	90%	49	15k	1h15m

Table 5.19: Adder Saving From Base Solution

Filter	Base Solution MBAs	Final Solution MBAs	Adder Reduction
B	No F.S.	12	-
L1	45	43	2
C	37	28	9

for running the maximum generations.

From Table 5.18, it is seen that the adaptive search space reduction technique is successful in reducing the search space and expediting the search process. For filter L1, no reduction in subspace was possible because of the high word length and hence the coefficients range was large ever for small coefficients.

Table 5.19 shows the number of adders reduced as a result of the optimization method. For Filter B, no feasible solution exists among the base solution. Since, the SAs are fixed before optimization, they are not taken into consideration. At first glance it may seem like a wastage of resources to optimize at the expense of such high design times. However, since the algorithm is tailored for custom IC design, the high cost can be nullified in the recurring engineering cost. Hence, the use of the optimization algorithm is justified.

Summary

Firstly in this chapter the Differential Evolution algorithm's performance was analyzed with respect to its control parameters. Next, an analysis of the filter length and word length used for ensuring minimal hardware was given. After that the results of the prototype filters were tabulated and their amplitude responses were given to show conformity with the filter specifications. Afterwards, a comparison of the design examples was carried out with the state of the art methods present in literature. Next, the complexity of the algorithm was analyzed and the statistics for the various designs were presented. Finally, the for the purpose of continuity into a lower abstraction level, the hardware synthesis was explained and two examples of the hardware synthesis were given to demonstrate the construction of the hardware from the tabulated data.

Chapter 6

Conclusion and Future Scope

6.1 Conclusion

This thesis formulated a design procedure for fixed coefficient finite word length filters. These filters are suitable for custom IC designs for cutting down the recurring engineering cost owing to their minimal hardware complexity. The design procedure has the capability to design single stage filters having sharp transition widths and stringent error constraints requiring filter orders up to length 300. The approach implements a shift and add network to replace the multipliers needed for generating the scaled and delayed values used in non recursive filtering operation.

A novel Differential Evolution algorithm, which is a population based evolutionary meta heuristic optimization technique, is proposed for the filter design problem. The algorithm is a discrete variant of DE algorithm which redefines the mutation operator in terms of bit representation aimed at optimizing discrete or mixed valued problems. The algorithm has been tested for standalone error minimization and joint optimization of error and hardware cost.

The algorithm is dived among design for small length filters or large filters. For small length filter, the algorithm is able to converge to feasible solutions without the need for giving hints on the location of the solutions. However, for large filters, a linear program is utilized for generating initial base solutions around which the population is generated by perturbation. The entire filter design approach is named DEFDO and it consists of 3 basic steps: coefficient sparsity, coefficient range, optimization routine. Coefficient sparsity is introduced to maximize zero valued coefficients. As zero valued coefficients have no hardware presence, they reduce the hardware cost substantially. Coefficient range is used to create the bounds on the search space for the filter coefficients. Lastly, the optimization routine consists of the DE algorithm.

Eight designs of six filters taken from literature have been implemented to show the working of the algorithm. Also, two additional filters have been designed to showcase the robustness of the algorithm for designing bandpass and bandstop filters. The comparison with the state of the art algorithms show that the design procedure is able to produce better results than the best available evolutionary and local search algorithms. When compared to deterministic methods, the results are comparable. In six out of the eight cases, the result is equal to the best deterministic method present. From the design examples, it is seen that the algorithm works best for the filter whose constraints are such that many feasible solutions are available. In cases where very few feasible solutions are present, the algorithm is not able to jointly optimize the hardware complexity and satisfy the feasibility condition. The reason for this is that the solutions are present further apart from each other and once the algorithm enters the valley created, it is not able to escape from it.

6.2 Contribution of Thesis

The thesis proposed a novel application of Differential Evolution algorithm for the design of finite word length digital filters. The Differential Evolution algorithm has been successful at solving hard optimization problems with continuous parameters. Owing to its success, researchers have attempted to utilize the differential approach to solving combinatorial optimization problems. However, all attempts have been made at solving classical problems such the Traveling Salesman Problem or the Multiple Knapsack Problem. This thesis proposed a representation for the Differential Evolution algorithm's population adapted for the finite word length filter design problem. Due to the nature of the FIR filter's coefficients, adaptive search space reduction technique was developed and proved successful to enhance the algorithm's run time and efficiency. The complex abstraction used for representing the TSP was simplified for the problem at hand and the mutation operator was redefined.

On the other front, this thesis proposed a fast and efficient method to minimize the hardware cost of implementing an FIR filter in custom ICs. Compared to other approximate methods such a Genetic Algorithm and Local Search, the algorithm is able to provide better results in terms of hardware cost. Owing to its non exponential computational complexity the method is scalable and can be used for larger design problems. Also, the design results are at par when compared with the best deterministic methods available. In conclusion, the design procedure is suited for designing filters with stringent frequency specifications that require large filter orders.

6.3 Future Scope

The proposed Differential Evolution algorithm is able to reduce hardware complexity when the filter constraints are relaxed and many feasible solutions exist. The new escape approach can be developed whereby the search agent is able to escape a deep valley. One such scheme could be to develop Particle Swarm optimization inspired local and global best learning approach.

The current objective in the filter design procedure has been to reduce the adder cost of implementing the finite word length FIR filter. However, it is observed that the problem is multi modal. Thus, a new problem at a lower level of abstraction can be formulated taking into account the full adder cost of implementation. Also, the actual synthesis of the filter can carried out using tools such as Synopsys Design Compiler to make precise latency and power analysis.

The proposed Differential Evolution algorithm can also be used for other discrete problems. It can be used to design cascade form FIR filters. It can also be modified for solving mixed problems by modifying the population members to include floating point parameters and utilizing the original mutation scheme for these parameters.

References

[1] H. K. Kwan, *Digital Filters and Systems.* ISP Lab, University of Windsor, 2015. 8

[2] L. R. Rabiner and B. Gold, “Theory and application of digital signal processing,” *Englewood Cliffs, NJ, Prentice-Hall, Inc., 1975. 777 p.*, vol. 1, 1975. 8

[3] T. Parks and J. McClellan, “A program for the design of linear phase finite impulse response digital filters,” *Audio and Electroacoustics, IEEE Transactions on*, vol. 20, no. 3, pp. 195–199, Aug 1972. 11

[4] T. Baran, D. Wei, and A. Oppenheim, “Linear Programming Algorithms for Sparse Filter Design,” *Signal Processing, IEEE Transactions on*, vol. 58, no. 3, pp. 1605–1617, March 2010. 13

[5] A. Jiang and H. K. Kwan, “WLS Design of Sparse FIR Digital Filters,” *Circuits and Systems I: Regular Papers, IEEE Transactions on*, vol. 60, no. 1, pp. 125–135, Jan 2013. 13

[6] Y. C. Lim, “Frequency-response masking approach for the synthesis of sharp linear phase digital filters,” *Circuits and Systems, IEEE Transactions on*, vol. 33, no. 4, pp. 357–364, Apr 1986. 13

[7] Y. Lim and Y. Lian, “Frequency-response masking approach for digital filter design: complexity reduction via masking filter factorization,” *Circuits and Systems II: Analog and Digital Signal Processing, IEEE Transactions on*, vol. 41, no. 8, pp. 518–525, Aug 1994. 13

[8] D. Kodek and K. Steiglitz, "Comparison of optimal and local search methods for designing finite wordlength FIR digital filters," *Circuits and Systems, IEEE Transactions on*, vol. 28, no. 1, pp. 28–32, Jan 1981. 14

[9] D. Kodek, "Design of optimal finite wordlength FIR digital filters using integer programming techniques," *Acoustics, Speech and Signal Processing, IEEE Transactions on*, vol. 28, no. 3, pp. 304–308, Jun 1980. 14

[10] Y. Lim and S. Parker, "FIR filter design over a discrete powers-of-two coefficient space," *Acoustics, Speech and Signal Processing, IEEE Transactions on*, vol. 31, no. 3, pp. 583–591, Jun 1983. 14

[11] Y. Lim, "Design of discrete-coefficient-value linear phase FIR filters with optimum normalized peak ripple magnitude," *Circuits and Systems, IEEE Transactions on*, vol. 37, no. 12, pp. 1480–1486, Dec 1990. 15

[12] H. Samueli, "An improved search algorithm for the design of multiplierless FIR filters with powers-of-two coefficients," *Circuits and Systems, IEEE Transactions on*, vol. 36, no. 7, pp. 1044–1047, Jul 1989. 15

[13] Y. C. Lim, R. Yang, D. Li, and J. Song, "Signed power-of-two term allocation scheme for the design of digital filters," *Circuits and Systems II: Analog and Digital Signal Processing, IEEE Transactions on*, vol. 46, no. 5, pp. 577–584, May 1999. 16

[14] D. Bull and D. Horrocks, "Primitive operator digital filters," *Circuits, Devices and Systems, IEE Proceedings G*, vol. 138, no. 3, pp. 401–412, June 1991. 16

[15] P. Cappello and K. Steiglitz, "Some complexity issues in digital signal processing," *Acoustics, Speech and Signal Processing, IEEE Transactions on*, vol. 32, no. 5, pp. 1037–1041, Oct 1984. 17

[16] A. Yurdakul and G. Dundar, "Fast and efficient algorithm for the multiplierless realisation of linear DSP transforms," *Circuits, Devices and Systems, IEE Proceedings -*, vol. 149, no. 4, pp. 205–211, Aug 2002. 17, 23

[17] M. Potkonjak, M. Srivastava, and A. Chandrakasan, “Multiple constant multiplications: efficient and versatile framework and algorithms for exploring common subexpression elimination,” *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 15, no. 2, pp. 151–165, Feb 1996. 17

[18] A. Dempster and M. Macleod, “Use of minimum-adder multiplier blocks in FIR digital filters,” *Circuits and Systems II: Analog and Digital Signal Processing, IEEE Transactions on*, vol. 42, no. 9, pp. 569–577, Sep 1995. 17, 18, 24, 52, 54

[19] ——, “Multiplication by an integer using minimum adders,” in *Mathematical Aspects of Digital Signal Processing, IEE Colloquium on*, Feb 1994, pp. 11/1–11/4. 18, 19, 52

[20] Y. J. Yu and Y. C. Lim, “Design of Linear Phase FIR Filters in Subexpression Space Using Mixed Integer Linear Programming,” *Circuits and Systems I: Regular Papers, IEEE Transactions on*, vol. 54, no. 10, pp. 2330–2338, Oct 2007. 20, 21, 89

[21] Y. Yu and Y. Lim, “Optimization of Linear Phase FIR Filters in Dynamically Expanding Subexpression Space,” *Circuits, Systems and Signal Processing*, vol. 29, no. 1, pp. 65–80, 2010. 23

[22] D. Shi and Y. J. Yu, “Design of Linear Phase FIR Filters With High Probability of Achieving Minimum Number of Adders,” *Circuits and Systems I: Regular Papers, IEEE Transactions on*, vol. 58, no. 1, pp. 126–136, Jan 2011. 23, 89

[23] C.-Y. Yao, W.-C. Hsia, and Y.-H. Ho, “Designing Hardware-Efficient Fixed-Point FIR Filters in an Expanding Subexpression Space,” *Circuits and Systems I: Regular Papers, IEEE Transactions on*, vol. 61, no. 1, pp. 202–212, Jan 2014. 23, 89

[24] W. B. Ye and Y. J. Yu, “Two-Step Optimization Approach for the Design of Multiplierless Linear-Phase FIR Filters,” *Circuits and Systems I: Regular Papers, IEEE Transactions on*, vol. 62, no. 5, pp. 1279–1287, May 2015. 23, 89

[25] ——, “Single-Stage and Cascade Design of High Order Multiplierless Linear Phase FIR Filters Using Genetic Algorithm,” *Circuits and Systems I: Regular Papers, IEEE Transactions on*, vol. 60, no. 11, pp. 2987–2997, Nov 2013. 24, 86

[26] A. Antoniou and W.-S. Lu, *Practical optimization: algorithms and engineering applications.* Springer Science & Business Media, 2007. 27

[27] "OPTI Toolbox," accessed: 2016-01-02. [Online]. Available: http://www.i2c2.aut.ac.nz/Wiki/OPTI/index.php/ 31, 65

[28] B. Borchers, "CSDP, A C library for semidefinite programming," *Optimization Methods and Software*, vol. 11, no. 1-4, pp. 613–623, 1999. [Online]. Available: http://dx.doi.org/10.1080/10556789908805765 31

[29] K. Price, R. M. Storn, and J. A. Lampinen, *Differential Evolution: A Practical Approach to Global Optimization (Natural Computing Series).* Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2005. 34

[30] J. Brest, S. Greiner, B. Boskovic, M. Mernik, and V. Zumer, "Self-Adapting Control Parameters in Differential Evolution: A Comparative Study on Numerical Benchmark Problems," *Evolutionary Computation, IEEE Transactions on*, vol. 10, no. 6, pp. 646–657, Dec 2006. 35

[31] Y. Liu, W. neng Chen, Z. hui Zhan, Y. Lin, Y.-J. Gong, and J. Zhang, "A Set-Based Discrete Differential Evolution Algorithm," in *Systems, Man, and Cybernetics (SMC), 2013 IEEE International Conference on*, Oct 2013, pp. 1347–1352. 35

[32] W.-N. Chen, J. Zhang, H. Chung, W.-L. Zhong, W. gang Wu, and Y. hui Shi, "A Novel Set-Based Particle Swarm Optimization Method for Discrete Optimization Problems," *Evolutionary Computation, IEEE Transactions on*, vol. 14, no. 2, pp. 278–300, April 2010. 35

[33] A. Shahein, Q. Zhang, N. Lotze, and Y. Manoli, "A Novel Hybrid Monotonic Local Search Algorithm for FIR Filter Coefficients Optimization," *Circuits and Systems I: Regular Papers, IEEE Transactions on*, vol. 59, no. 3, pp. 616–627, March 2012. 40, 86

[34] W. B. Ye and Y. J. Yu, "A polynomial-time algorithm for the design of multiplierless linear-phase FIR filters with low hardware cost," in *Circuits and Systems (ISCAS), 2014 IEEE International Symposium on*, June 2014, pp. 970–973. 48

[35] H. Samueli, "On the design of FIR digital data transmission filters with arbitrary magnitude specifications," *Circuits and Systems, IEEE Transactions on*, vol. 38, no. 12, pp. 1563–1567, Dec 1991. 55

[36] M. Aktan, A. Yurdakul, and G. Dundar, "An Algorithm for the Design of Low-Power Hardware-Efficient FIR Filters," *Circuits and Systems I: Regular Papers, IEEE Transactions on*, vol. 55, no. 6, pp. 1536–1545, July 2008. 48, 64, 89

[37] L. Aksoy, P. Flores, and J. Monteiro, "Exact and Approximate Algorithms for the Filter Design Optimization Problem," *Signal Processing, IEEE Transactions on*, vol. 63, no. 1, pp. 142–154, Jan 2015. 86, 89

[38] K. Johansson, O. Gustafsson, and L. Wanhammar, "Bit-Level Optimization of Shift-and-Add Based FIR Filters," in *Electronics, Circuits and Systems, 2007. ICECS 2007. 14th IEEE International Conference on*, Dec 2007, pp. 713–716. 81

Vita Auctoris

Kunwar Rehan, born 1992 in India, is an MASc candidate at the University of Windsor, Windsor, Ontario, Canada. He received his Bachelor of Technology in Electronics Engineering from Aligarh Muslim University, Aligarh, India in 2014 with the highest honours.

In 2015, Kunwar Rehan was admitted as a member of the Golden Key Honour Society for his academic excellence. At the University of Windsor, he served as a graduate teaching assistant. He also served as a Industrial Relation Chair of the IEEE Student Branch.

Kunwar Rehan's research interests include VLSI Signal Processing and Optimization Methods.