THE IMPLEMENTATION COMPLEXITY OF FINITE IMPULSE RESPONSE DIGITAL FILTERS UNDER DIFFERENT COEFFICIENT QUANTIZATION SCHEMES AND REALIZATION STRUCTURES

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

SEFA AKYÜREK

IN PARTIAL FULLFILMENT OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE
IN THE DEPARTMENT OF ELECTRICAL AND ELECTRONICS ENGINEERING

DECEMBER 2004

Approval of the Graduate School of Natural and Applied Sciences

Prof. Dr. Canan ÖZGEN
Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science.

Prof. Dr. İsmet ERKMEN
Head of Department

This is to certify that we have read this thesis and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science.

Assoc. Prof. Dr. Tolga ÇİLOĞLU
Supervisor

Examining Committee Members (first name belongs to the chairperson of the jury and the second name belongs to supervisor)

Prof. Dr. Zafer ÜNVER	(METU, EE)
Assoc. Prof. Dr. Tolga ÇİLOĞLU	(METU, EE)
Assoc. Prof. Dr. Engin TUNCER	(METU, EE)
Alper ÜLKÜ	(ASELSAN)
A. Umut CİNİVİZ	(ASELSAN)

PLAGIARISM

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last name:

Signature:

ABSTRACT

THE IMPLEMENTATION COMPLEXITY OF FINITE IMPULSE RESPONSE DIGITAL FILTERS UNDER DIFFERENT COEFFICIENT QUANTIZATION SCHEMES AND REALIZATION STRUCTURES

AKYÜREK, Sefa
M.S., The Department of Electrical and Electronics Engineering
Supervisor: Assoc. Prof. Dr. Tolga ÇİLOĞLU
December 2004, 86 Pages

It has been aimed to investigate the complexity of discrete-coefficient FIR filters when they are implemented in transposed form and the coefficient redundancy is removed by the n-Dimensional Reduced Adder Graph (RAG-n) approach. Filters with coefficients represented by different quantization schemes have been designed or selected from the literture; their transposed form implemetations after RAG-n process have been compared in terms of complexity. A Genetic Algorithm (GA) based design algorithm has been implemented and used for the design of integer coefficient filters. Algorithms for the realization of filter coefficients in Canonic Signed Digit (CSD) form and realization of n-Dimensional Reduced Adder Graph (RAG-n) have also been implemented. Filter performance is measured as Normalized Peak Ripple Magnitude and implementation complexity as the number of adders used to implement filter coefficients. Number of adders used to implement filter coefficients is calculated by using two different methods: CSD and RAG-n. RAG-n method has been applied to FIR digital filter design methods that don't

consider reduction of implementation complexity via RAG-n with transposed direct form filter structure. For implementation complexity, it is concluded that "RAG-n algorithm with transposed direct form filter structure" provides better results over the "CSD, SPT coefficient design followed by transposed direct form filter structure" in terms of number of adders used in the implementation.

Keywords: Discrete Coefficient Filter, Powers-of-two coefficient, Filter Optimization, FIR Digital Filter, Genetic Algorithm, Quantization.

ÖZ

SONLU DÜRTÜ YANITLI SAYISAL SÜZGEÇLERİN FARKLI KATSAYI NİCEMLEMELERİ VE SÜZGEÇ YAPILARI İLE GERÇEKLEŞTİRİM KARMAŞIKLIĞI

AKYÜREK, Sefa
Yüksek Lisans, Elektrik ve Elektronik Mühendisliği Bölümü
Tez Yöneticisi: Doç. Dr. Tolga ÇİLOĞLU
Aralık 2004, 86 Sayfa

Ayrık katsayılı, sonlu dürtü yanıtlı sayısal süzgeçlerin gerçekleştirim karmaşıklığının, çevrilmiş süzgeç yapısı kullanıldığında ve katsayı fazlalığı n-Boyutlu İndirgenmiş Toplayıcı Çizgesi (RAG-n) yaklaşımı kullanılarak azaltıldığında incelenmesi amaçlanmıştır. Katsayıları farklı nicemleme yöntemleri ile gösterilen süzgeçler tasarlanmış veya literatürden alınmış; RAG-n işleminden sonraki çevrilmiş süzgeç yapıları gerçekleştirim karmaşıklığı açısından karşılaştırılmıştır. Genetik algoritma (GA) tabanlı tasarım algoritması geliştirilmiş ve katsayıları tam sayı olan süzgeçlerin tasarımında kullanılmıştır. Süzgeç katsayılarının Kurallı İşaretli Sayı (CSD) formu ve RAG-n ile gerçekleştirilmesi için gerekli algoritmalar da geliştirilmiştir. Süzgeç başarımı normalize edilmiş tepecik büyüklüğü olarak ve gerçekleştirim karmaşıklığı süzgeç katsayılarını oluşturmak için gerekli toplayıcı sayısı olarak alınmıştır. Süzgeç katsayılarını oluşturmak için gerekli toplayıcı sayısı CSD ve RAG-n yöntemleri kullanılarak bulunmuştur. Ayrıca süzgeç gerçekleştirim karmaşıklığını indirgemede çevrilmiş süzgeç yapısını kullanan ve RAG-n yöntemini dikkate almayan ayrık katsayılı sonlu dürtü yanıtlı sayısal süzgeç gerçekleştirim yöntemlerine de bu yöntem uygulanmıştır. Çevrilmiş süzgeç yapısını kullanan RAG-n yöntemi, süzgeç

katsayılarını oluşturmak için gerekli toplayıcı sayısı açısından, çevrilmiş direk süzgeç yapısını kullanan CSD ve işaretli ikinin kuvveti yöntemlerinden daha iyi sonuçlar vermiştir.

Anahtar Sözcükler: Ayrık katsayılı süzgeç, ikinin kuvveti katsayı, süzgeç eniyileme, FIR sayısal süzgeç, genetik algoritma, katsayı nicemleme.

To My Family!

ACKNOWLEDGEMENTS

I would like to express my deepest gratitude to my supervisor Assoc. Prof. Dr. Tolga ÇİLOĞLU his guidance, advice, criticism, encouragements and insight throughout my research.

TABLE OF CONTENTS

LIST OF FIGURES

FIGURE

LIST OF TABLES

TABLE

LIST OF ACRONYMS

ASIC: Application Specific Integrated Circuit

CSD: Canonic Signed Digit

DSP: Digital Signal Processing

FIR: Finite Impulse Response

GA: Genetic Algorithm

LMS: Least-Mean Squared Error

MAX: Maximum

MILP: Mixed Integer Linear Programming

MIN: Minimum

MSE: Mean Squared Error

NPRM: Normalized Peak Ripple Magnitude

PTA: Polynomial Time Algorithm

RAG-n: n-Dimensional Reduced Adder Graph

SA: Simulated Annealing

SPT: Signed Powers-of-Two

CHAPTER 1

INTRODUCTION

Finite Impulse Response (FIR) digital filters are frequently used in digital signal processing because of their inherent stability and the possibility of perfect linear phase. Designing FIR digital filters has received a great interest because of their wide usage in consumer and military DSP applications such as video and image processing, target tracking, radar processing. However, Digital Signal Processing (DSP) cores are not always suitable for DSP applications requiring high processing performance and low power consumption such as coding, image and video processing that require high performance filters operating at high data rates. Application specific FIR digital filters are frequently designed to meet the constraints of processing performance and power consumption of such DSP applications. FIR digital filter implementation may also suffer from a large number of multiplications, leading to excessive chip area and power consumption in Application Specific Integrated Circuits (ASIC).

Fast fixed-point arithmetic is required for realization of low-cost and high-speed DSP hardware. Because of this requirement, designing FIR digital filters with very coarsely quantized coefficient values are valuable. Design of FIR digital filters with discrete valued coefficients has been a research area where considerable effort has been spent. The reasons for necessity of quantization of filter coefficients are as follows:

- Finite wordlength constraint in microprocessors
- Demand for multiplier-less/efficient/low cost VLSI implementation and/or high speed processing

A major problem in discrete coefficient FIR digital filter design is the determination of the optimal quantized coefficient values that satisfy the given filter design specifications. Multimodal behavior of cost function over a discrete domain

has been a fundamental difficulty in the development of efficient methods for discrete coefficient FIR digital filter design. Various FIR digital filter design methods have been proposed to solve this problem and to guarantee the optimal discrete coefficient solution for linear phase FIR digital filters such as Mixed Integer Linear Programming (MILP) [3], [6], [11], and [16], Simulated Annealing (SA) [5], [14], and [23], Genetic Algorithms (GA) [2], [15], [17], [19], [21], and [24] and Local Search [1], [18], and [25] and Free Allocation algorithms [4], [7], [20], and [22].

Beside the filter coefficient quantization problem, implementation of quantized filter coefficients is also an important task in FIR digital filter design for low cost and high performance filtering. Complexity of FIR digital filters is determined by the number of multiplications in filters. Many researches have studied the minimization the complexity of multiplier blocks required in filtering. Multiplications can be eliminated by decomposing them into simple operations such as addition, subtraction and shifting. For a filter with constant coefficients, the decomposition into addition, subtraction and shifting is more efficient than employing multipliers. The complexity of FIR digital filters in this case is dominated by the number of adders/subtractors used to implement the coefficient multiplications. To reduce the complexity, the coefficients can be restricted, for example, to two signed-powers-of-two (SPT) or expressed in Canonic Signed Digit (CSD) code [1], [3], [4], [6], [7], [11], [13], [14], [16], [18], [20], [22]. CSD representation is the unique minimum representation for which no two SPT terms are adjacent. CSD representation produces better results than SPT representation in terms of number of adders used. CSD and SPT approaches deal with each coefficient individually while realizing the filter coefficients.

Another way to reduce filter complexity is the graph representation [8], [9], [12], [19], [21]. Graph representation approach arises in the context of transposed direct form (Fig. 1.1) and considers all filter coefficients a whole. As shown in Figure 1.2, the hardware block called a multiplier block is used to implement all coefficient multiplications. The multiplier block is significant in terms of area and power because some adders and shifters can be shared among different multiplications. RAG-n algorithm [8] is one of the methods that reduces complexity

of this multiplier block by exploiting the redundancy in filter coefficients. Transposed direct form filter structure is a necessity to use RAG-n in the implementation of FIR digital filters since all filter coefficients are multiplied with the input signal at the beginning and delayed later. So there is a possibility to synthesize all filter coefficients before multiplication and delay.

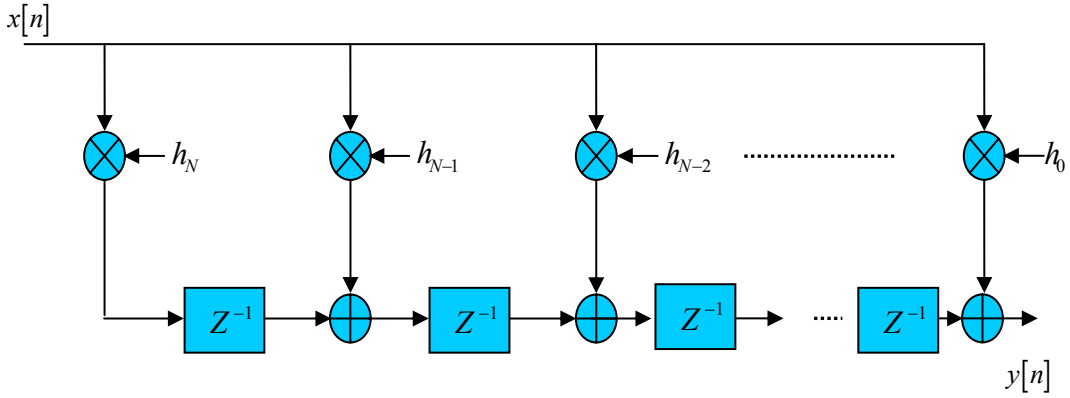


Figure 1.1 Transposed Direct Form

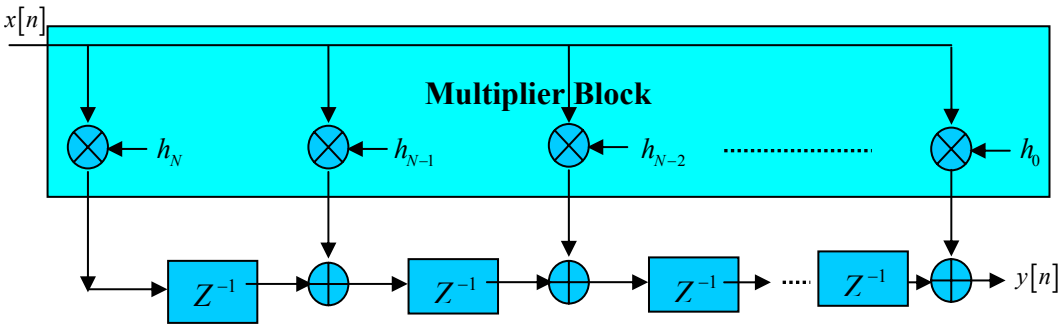


Figure 1.2 Multiplier Block

1.1. Coefficient Quantization Methods

Coefficient Quantization methods can be classified as follows:

Uniform Quantization: The filter coefficients are represented by b-bit binary numbers including the sign bit. Figure 1.3 shows a uniform quantization grid for two coefficients. Design of filters with uniform quantization has been considered in [3] by MILP, in [23] by simulated annealing, in [2], [19], [21] by GAs.

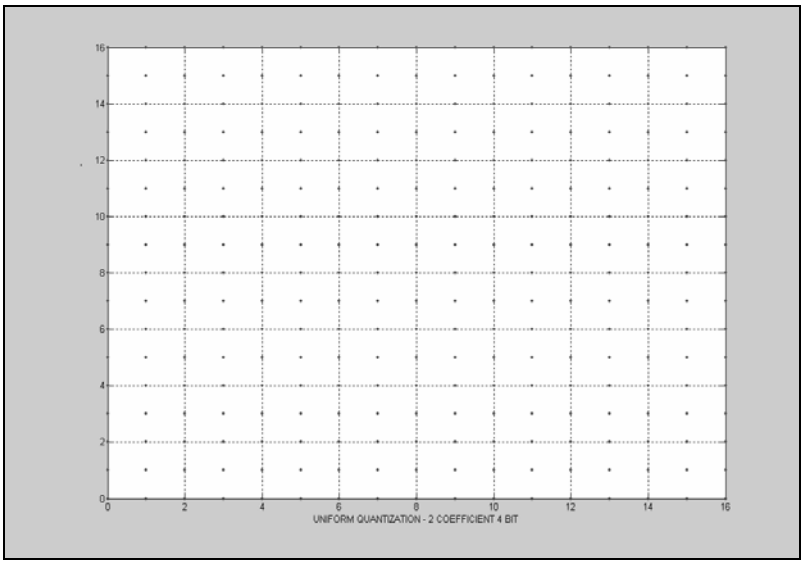


Figure 1.3 Uniform Quantization

Non-Uniform Quantization: Distribution of values of filter coefficients is nonuniform. For example, if two SPT are assigned per filter coefficient, then this is a nonuniform quantization. In this case domain of filter coefficients is as follows:

$$D = \left\{ \alpha : \alpha = \sum_{k=1}^{2} c_k \, 2^{-g_k} \right\} \tag{1.1}$$

$$c_k \in \{-1,0,1\} \text{ and } g_k \in \{0,1,2,3,.....,B\} \tag{1.2}$$

B is an integer that represents maximum number of shifts that can be performed on the filter input signal.

Figure 1.4 shows nonuniform quantization grid for two coefficients for B=3. Design of filters with nonuniform quantization has been considered in [5], [14] by SA, in [11] by MILP

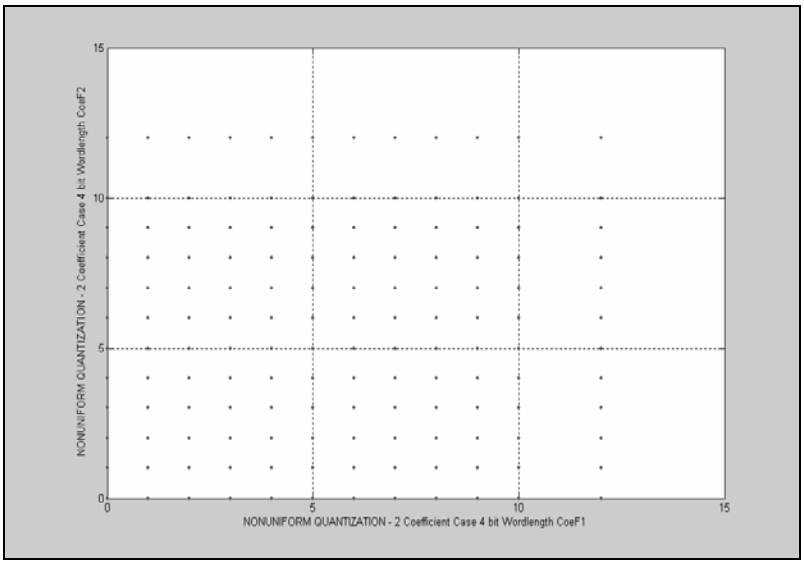


Figure 1.4 Nonuniform Quantization

Free allocation: In this method, total number of SPT terms is fixed. These SPT terms are distributed to filter coefficients via an optimization technique. Actually, this method is also a nonuniform quantization. Figure 1.5 shows free allocation quantization grid for two coefficients with 2^0, 2^1 as SPT terms and a total of two SPT terms. Design of filters with free allocation has been considered in [7] by a rounding algorithm, in [4] by a trellis search algorithm and in [20] by a local search algorithm.

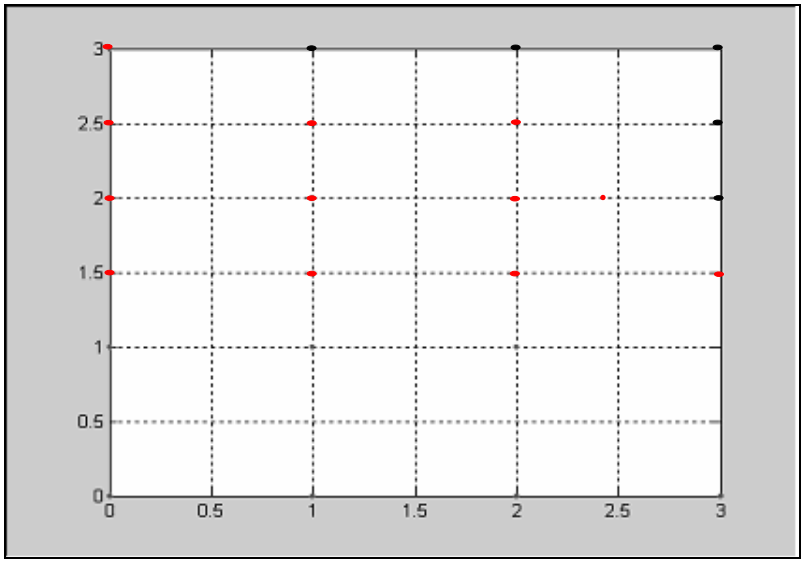


Figure 1.5 Free Allocation

1.2. The Aim and Organization of the Thesis

This thesis aims to compare FIR digital filter design methods that use different quantization techniques for filter coefficients. Comparisons have been made in terms of both filter performance and implementation complexity. For filter design, different filter design methods have been studied and specifically a filter design method based on genetic algorithm has been implemented. For filter implementation, transposed direct form filter structure and an algorithm [8] that reduces the number of adders required in this structure (implementation complexity) have been studied. This structure and algorithm [8] have been applied to filter design methods [1-7], [11], [13-16], [22] to reduce the implementation complexity since this has not been studied before in literature. Results drawn from the comparisons are dramatic in terms of number of adders required in the implementation. It has been seen that implementation complexity has been reduced approximately 50% by using this structure and algorithm [8] in filter design methods [1]-[7], [11], [13]-[16], [22]. It is concluded that filters with integer or SPT coefficients implemented by RAG-n with transposed direct form filter structure are less complex than a CSD implementation.

Thesis is organized in the following manner: Chapter 2 deals with the basic concepts of discrete coefficient FIR digital filter design methods. Three major approaches to the discrete coefficient FIR digital filter design problem are given. Chapter 3 is based on the details of the discrete coefficient FIR digital filter design method implemented in this thesis. Chapter 4 is based on the details of realization algorithm for filter coefficients namely RAG-n [8] implemented in this thesis and gives the results drawn from the research and finally in Chapter 5, conclusion and possible directions for future work are given.

CHAPTER 2

A SURVEY ON DISCRETE COEFFICIENT FIR DIGITAL FILTER DESIGN METHODS

2.1. Introduction

The filter design problem can be stated as follows: obtain a set of filter coefficients such that $H(f)$ (designed filter frequency response) is the best approximation to some desired function $D(f)$ (desired filter frequency response), over a given frequency range, with respect to some optimality criterion. Filter design methods commonly optimize the following cost function:

$$E(\vec{x}) = \max\{W(f)\left|H(f) - D(f)\right|\} \tag{2.1}$$

where $W(f)$ is a positive error weighting function and $\vec{x}$ is the vector of filter coefficients. This kind of optimization is a difficult combinatorial problem. A variety of algorithms are available for designing filters with discrete coefficient values and to guarantee the optimal discrete coefficient solution for FIR digital filters.

MILP [3], [6], [11], [13], [16] has been applied to design FIR digital filters according to $E(\vec{x})$. One of the drawbacks of MILP is that computation time for long filters is very high and also large amount of memory is needed. Computational load is exponentially dependent on the filter length. The optimum finite word length solution obtained by MILP saves only a few bits in coefficient word length when compared to the simple coefficient rounding.

GAs have been used for the problem of discrete coefficient FIR digital filter design [2], [19], [21], [24]. GAs are search algorithms based on genetic and natural selection paradigm and can be successfully employed for minimizing a cost function

such as the difference between the desired filter frequency response and the designed filter frequency response. They can handle arbitrary constraints on coefficient values and filter specifications. High computation time because of many functions evaluations, the need for multiple trials to get satisfactory solutions because of random selections made inside the algorithm, and the need for experience to adjust the algorithm parameters are drawbacks of GAs.

SA is a very powerful optimization method that has been applied to solve the problem of discrete coefficient FIR digital filter design [5], [14], [23]. There are no limitations on the cost function and constraints of the problem. These features make SA a valuable tool in filter design problem. Like GAs, high computation time because of many functions evaluations, the need for multiple trials to get satisfactory solutions because of random selections made inside the algorithm, and the need for experience to adjust the algorithm parameters are drawbacks of SA.

Local search techniques have been employed to solve the problem of discrete coefficient FIR digital filter design [1], [18], [25]. Local search techniques take the rounded solution of infinite precision design as a start point and look for improved solutions by partially examining some neighborhood of the best discrete solution. Multiple trials with different sequence or contents of coefficient groups are needed to get the best solution. This increases the computation time spent by the algorithm.

Some methods focus on the number of SPT terms since they determine filter hardware complexity. Free allocation methods [4], [7], [20], [22] allocate SPT terms to most deserving filter coefficients but they fix the number of SPT terms at the beginning. The coefficients are found by minimizing the distance between the coefficients of infinite wordlength design and those of the sum of the SPT terms. Initially all quantized coefficient values are set to zero. Choosing one SPT term at a time and allocating it to the most deserving filter coefficient minimize the distance mentioned above.

2.2. Optimization Methods for FIR Digital Filter Design

The methods that have been used to design FIR digital filters under uniform and nonuniform quantization are MILP, SA, GA and local search methods. They are going to be described briefly in this section.

2.2.1 Mixed Integer Linear Programming

MILP technique can be used in discrete coefficient FIR digital filter design [3], [6], [11], [16]. This technique can be used to minimize the ripple subject to hardware specifications (wordlength and filter order) or to minimize the hardware cost (number of SPT terms). The equation in 2.1 can be used as a cost function and evaluating this cost function on a dense frequency grid allows the optimization to be formulated as a linear programming problem. The optimization reduces to determining the set of integer values for filter coefficients which satisfy the following conditions

$$W(f)\left|H(f)-D(f)\right| \le \delta \tag{2.2}$$

$$W(f)\left|H(f)-D(f)\right| \ge -\delta \tag{2.3}$$

with a minimum value of δ. This is an integer linear programming when the coefficients are restricted to integer values.

MILP with a suitable branch-and-bound algorithm enables to design filters with any discrete coefficient value. Branch-and-bound algorithm starts with obtaining a continuous coefficient value (i.e., infinite precision coefficient value). Let this problem be P_0. The next step is to select a coefficient whose value is not a desired value. Let this coefficient be $h(n)$. If $\lfloor h(n) \rfloor$ and $\lceil h(n) \rceil$ two consecutive discrete levels such that

$$\lfloor h(n) \rfloor \le h(n) \le \lceil h(n) \rceil \tag{2.4}$$

then, since the discrete value of $h(n)$ can not fall between $\lfloor h(n) \rfloor$ and $\lceil h(n) \rceil$, two mathematical programming problems P_1 and P_2 are generated by adding the constraints.

$$h(n) \le \lfloor h(n) \rfloor \tag{2.5}$$

$$h(n) \ge \lceil h(n) \rceil \tag{2.6}$$

P_1 and P_2 are solved individually. Further branching may be performed on P_1 and P_2 to produce the subproblems P_3, P_4, P_5, P_6. If it is predicted that an improved solution cannot be obtained for a branch then this branch can be removed to reduce number of branchings.

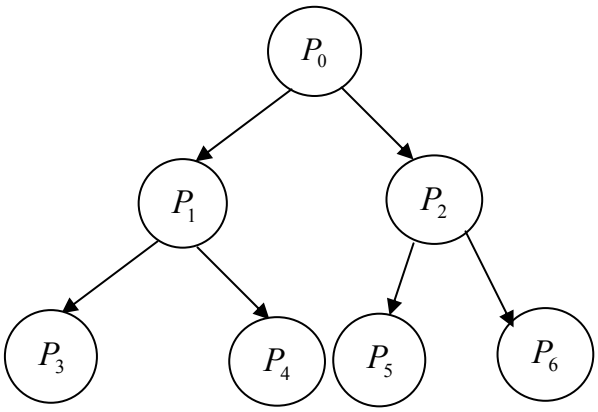


Figure 2.1 A Branch and Bound Tree

There are several branch-and-bound algorithms. Two of them are as follows:

1. *Depth-first branch-and-bound*: The algorithm is as follows: after solving P_0, speculate on P_1 and P_2. If a decision is made to solve P_1, then P_2 is saved for later solution. After solving P_1, speculate on P_3 and P_4; solve one of them and save the other. This procedure continues until a suboptimum discrete solution is obtained. After obtaining a discrete solution, reinitiate the depth-first search process from an unsolved problem. This process continues until the entire problem is solved.
2. *Isocost branch-and-bound*: The algorithm is as follows: after solving P_1 and P_2, they are compared and the better one is selected for branching. If P_1 is better one, then after solving for P_3 and P_4; P_3, P_4 and P_2 are compared and the best one is selected. The procedure continues by always selecting the best subproblem for further branching.

2.2.2 Simulated Annealing

One of the well-known powerful global optimization algorithms is SA that is introduced in combinatorial optimization and used in discrete coefficient FIR digital filter design [5], [14], [23]. In simulated annealing, a trial solution is chosen and the effects of taking a small random step from this position are tested. If the step results in a reduction in the cost function, it replaces the previous solution as the current

trial solution. If it does not result in a cost saving, the solution still has a probability P of being accepted as the new trial solution given by:

$$P = \exp(-\Delta E / T) \tag{2.7}$$

Here, ΔE is the increase in the cost function that would result from the step. The temperature T is simply a numerical value that determines the stability of a trial solution. If T is high, new trial solutions will be generated continually. If T is low, the trial solution will move to a local or global cost minimum — if it is not there already — and will remain there. The value of T is initially set high and is periodically reduced according to a cooling schedule. A commonly used simple cooling schedule is:

$$T_{t+1} = \alpha T_t \tag{2.8}$$

where T_t is the temperature at step number t and α is a constant close to, but below, 1. When T is high, the optimization routine is free to accept many varied solutions, but as it drops, this freedom diminishes.

The success of the technique is dependent upon values chosen for starting temperature, the size and frequency of the temperature decrement, and the size of perturbations applied to the trial solutions.

In the design of FIR digital filters, the minimax objective function in equation 2.1 can be used. The starting point of the algorithm is the rounded infinite precision optimum solution. The random moves are made in the neighborhood of this infinite precision solution with proper step sizes. This is followed by a decision mechanism and cooling schedule is applied for the new move and solution is accepted if it is better from the previous and algorithm reinitiates from this new point.

2.2.3 Genetic Algorithms

The basic principle of a genetic algorithm is to randomly generate an initial population (consists of a collection of chromosomes) of solutions, each solution having an objective value or ‘fitness’. Pairs of individual solutions are then combined in an attempt to produce better (fitter) solutions. After each round of combinations, the least fit individuals are excluded from further combination to keep the number of individuals in the ‘breeding’ pool constant. In this way it is hoped

that, after many breeding cycles, the pool will contain a population of solutions with much improved objective values from which the best can be chosen. This class of techniques is appealing since it mimics the natural selection process that drives evolution.

GA based design techniques are widely proposed for discrete coefficient filter designs [2], [15], [17], [19], [21], [24]. In GA, chromosomes consisting of integer genes can be used to represent the filter coefficients. The fitness function that can be used in selection process may be Mean Squared Error (MSE), Least-Mean Squared Error (LMS), Normalized Peak Ripple Magnitude (NPRM), number of adders used in the implementation (adder cost) or may consist of a combination of NPRM and adder cost. The starting point of the algorithm is the rounded infinite precision optimum solution. New populations (new set of filter coefficients) are generated with mutation and crossover operations and a fitness function is evaluated for this population and better ones are selected for next GA cycle.

2.2.4 Local Search Methods

Local search type algorithms [1], [18], [25] start from the rounded and scaled solution of the infinite precision solution and seek for improved solutions by partially examining some neighborhood of the best discrete solution at the instant. Equation 2.1 can be used as a cost function. The starting solution can be

$$h'_i = [2^{b-1} h_i], \quad i = 0,1,2\ldots\text{N-1} \tag{2.9}$$

where h_i is the infinite precision coefficient and the brackets denote rounding to the nearest integer. Coefficients are represented by b bits and N is filter length. The simplest neighborhood of a solution is defined by examining the perturbations of each coefficient, taken one-at-a-time, by ± 1. This set is called as "1-change" neighborhood. "2-change" neighborhood is defined as union of "1-changes" and the set of perturbations of two coefficients at a time. There are two main strategies for searching the neighborhood. The first one accepts the first improved solution found. The second one searches the entire neighborhood and selects the best.

2.3. Free Allocation Methods

Some of the FIR digital filter design methods that use free allocation method

are [4], [7], [20], and [25]. The total number of SPT terms determines the realization cost (adder cost) of a digital filter. Distribution of SPT terms among the filter coefficients does not affect the realization cost. Therefore, the number of SPT terms for each coefficient is not necessarily limited to a fixed number. Instead, they should be allowed to vary subject to a given number of total SPT terms for the filter. This provides the possibility of finding a better set of coefficients.

2.4.1 Polynomial Time Algorithm [7]

This algorithm designs digital filters with SPT coefficients subject to a prescribed total number of SPT terms. The algorithm finds the coefficients by minimizing the L^{∞} norm of the difference between the coefficients of infinite wordlength and those of sums of SPT terms.

Basic polynomial algorithm outline is as follows:

Step 1. Start with initializing all the quantized coefficient values to zero.

Step 2. Choose one SPT term at a time and allocate it to the currently most deserving coefficient to minimize the L^{∞} distance between the SPT coefficients and their corresponding infinite wordlength values. This process of allocating the SPT terms for the filter is equal to a prescribed number.

2.4.2 Trellis Search Algorithm [4]

Trellis search algorithm also uses free allocation method for discrete coefficient filter design. FIR digital filters with coefficients implemented as sums of signed-powers-of-two terms can be designed with this algorithm in two stages:

Stage 1: Using a fast time domain approximation, a prototype filter is designed.

Stage 2: Formulating the design problem as a dynamic-programming-like recursive optimization problem, filters are designed by performing a trellis search that is similar to the viterbi algorithm. Inherent in the proposed trellis search algorithm is an iterative procedure that designs filters with gradually growing number of SPT terms, providing a means to control the filter's implementation complexity.

CHAPTER 3

DESIGN OF INTEGER COEFFICIENT FILTERS BY GENETIC ALGORITHM

The aim of this chapter is to introduce the genetic algorithm that is developed for the discrete coefficient FIR digital filter design in this thesis. The chapter is organized as follows:

- Basics of genetic algorithms
- GA for the design of the integer coefficient filters.

3.1. Genetic Algorithms

3.1.1 Introduction

GA is a tool for searching and optimizing methodology. GAs have been inspired by natural evolution, the process by which successive generations of animals and plants are modified so as to approach an optimum form. Each offspring has different features from its parents, i.e., it is not a perfect copy of parents. If the new characteristics are favorable, the offspring is more likely to flourish and pass its characteristics to the next generation. However, an offspring with unfavorable characteristics is likely to die without reproducing. It works on the Darwinian principle of natural selection where stronger individuals are likely the winners in a competing environment. These ideas have been applied to mathematical optimization, where a population of candidate solutions "evolves" toward an optimum. Each cell of a living organism contains a set of chromosomes that define the organism's characteristics. The chromosomes are made up of genes, where each gene determines a particular trait such as eye color. The complete set of genetic material is referred to as the genome, and a particular set of gene values constitutes a genotype. The resulting set of traits is described as the phenotype. Each individual in

the population of candidate solutions is graded according to its fitness. The higher the fitness of a candidate solution, the greater are its chances of reproducing and passing its characteristics to the next generation. In order to implement a GA, the following design decisions need to be made:

- how to use sequences of numbers, known as chromosomes, to represent the candidate solutions;
- the size of the population;
- how to evaluate the fitness of each member of the population;
- how to select individuals for reproduction using fitness information (conversely, how to determine which less-fit individuals will not reproduce);
- how to reproduce candidates, i.e., how to create a new generation of candidate solutions from the existing population;
- when to stop the evolutionary process.

3.1.2. Basic Genetic Algorithm

GA uses a direct analogy of such natural evolution. Solution of any problem can be represented by a set of parameters. According to the GA, the solution is viewed as a chromosome and the corresponding parameters are regarded as the genes of that chromosome. Also the effect of the environment substitutes with a fitness (objective) function, which is used to reflect the degree of "goodness" of the chromosome. Throughout a genetic evolution, the fitter chromosome has a tendency to yield good quality offspring, which means a better solution to any problem. In a GA application, a population pool, whose size varies from one problem to another, of chromosomes has to be initialized randomly. In each cycle of genetic operation, termed as an evolving process, a subsequent generation is created from the current chromosomes in the population.

3.1.2.1 Chromosomes

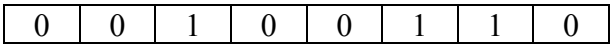
Each point in the search space can be represented as a unique chromosome, made up of genes. Suppose, for example, it is aimed to find the maximum value of a fitness function, f(x, y). In this example, the search space variables, x and y, are constrained to the 16 integer values in the range 0–15. A chromosome corresponding

to any point in the search space can be represented by two genes:

Figure 3.1 GA Chromosome

Thus the point (2, 6) in search space would be represented by the following chromosome:

The possible values for the genes are called alleles, so there are 16 alleles for each gene in this example. Each position along the chromosome is known as a locus; there are two loci in the above example. The loci are usually constrained to hold only binary values. (The term evolutionary algorithm describes the more general case where this constraint is relaxed.) The chromosome could therefore be represented by eight loci comprising the binary numbers 0010 and 0110, which represent the two genes:



Although there are still 16 alleles for the genes, there are now only two possible values (0 and 1) for the loci. The chromosome can be made as long as necessary for problems involving many variables, or where many loci are required for a single gene. In general, there are 2^N alleles for a binary-encoded gene that is N bits wide.

3.1.2.2 Basic GA Structure

A flow chart for the basic GA is shown in Figure 3.2. In the basic algorithm, the following assumptions have been made:

- The initial population is randomly generated.
- Individuals are evaluated according to the fitness function.
- Individuals are selected for reproduction on the basis of fitness; the fitter an individual, the more likely it is to be selected

• Reproduction of chromosomes to produce the next generation is achieved by "breeding" between pairs of chromosomes using the crossover operator and then applying a mutation operator to each of the offspring.

Initial population consists of a collection of chromosomes that represent a set of solutions for the problem. The chromosome producing the minimum error function value represents the best solution to the given problem. The chromosomes that give better fitness function value are selected and sent to the crossover operator. Two new chromosomes are created from the two selected chromosomes existing in the population by choosing a common point in the selected chromosomes and swapping their corresponding digits. In the mutation operation, values of the chromosomes mutate randomly. These GA operators are explained in detail below.

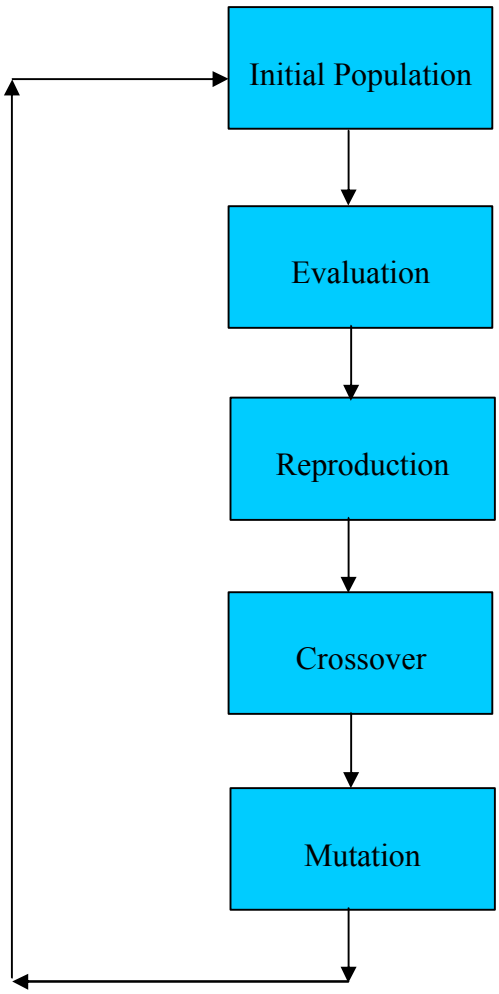


Figure 3.2 Basic GA Cycle

3.1.2.3 Crossover

Child chromosomes are produced by aligning two parents, picking a random position along their length, and swapping the tails, known as the crossover probability. An example for an eight-loci chromosome, where the mother and father genes are represented by m_i and f_i respectively, would be:

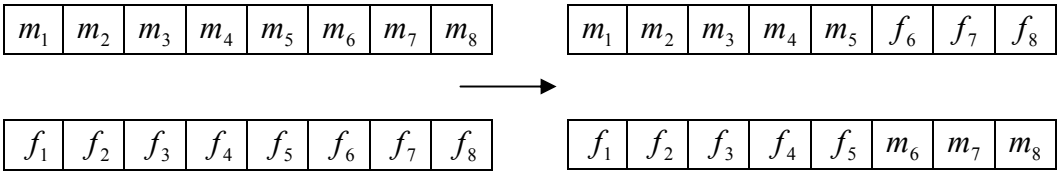


Figure 3.3 GA Single-point Crossover

This is known as single-point crossover, as only one position is specified for separating the swapped and unswapped loci. In fact this is a misnomer, as a second crossover position is always required. In single-point crossover the second crossover position is assumed to be the end of the chromosome. This can be made clearer by considering two-point crossover, where the chromosomes are treated as though they were circular, i.e., m_1 and m_8 are neighboring loci:

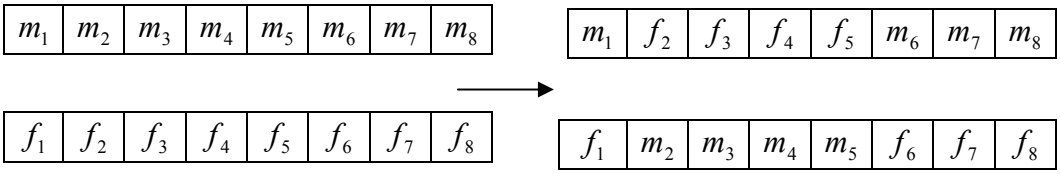


Figure 3.4 GA Two-point Crossover

In general, multipoint crossover is also possible, provided there are an even number of crossover points:

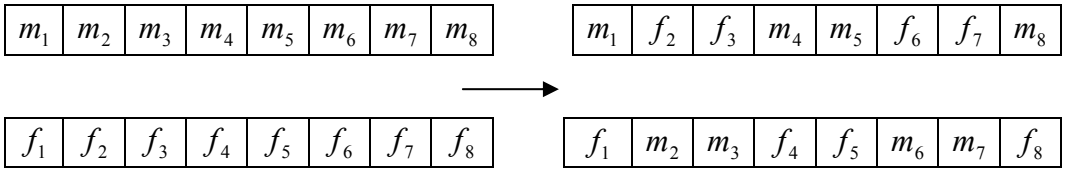


Figure 3.5 GA Multi-point Crossover

In the extreme case, each locus is considered for crossover, independently of the rest. This is known as uniform crossover.

3.1.2.4 Mutation

Unlike crossover, mutation involves altering the values of one or more loci. This creates new possibilities for gene combinations that can be generated by crossover. Mutation can be carried out in either of two ways:

- The value of a randomly selected gene can be replaced by a randomly generated allele. This works for both binary and nonbinary chromosomes.
- In a binary chromosome, randomly selected loci can be toggled, i.e., 1 becomes 0 and 0 becomes 1.

Individuals are selected randomly for mutation. The main advantage of mutation is that it puts variety into the gene pool, enabling the GA to explore potentially beneficial regions of the search space that might otherwise be missed.

3.1.3. Selection

It has already been stated that individuals are selected for reproduction on the basis of their fitness, i.e., the fittest chromosomes have the highest likelihood of reproducing. Selection determines not only which individuals will reproduce, but how many offspring they will have. The selection method can have an important impact on the effectiveness of a GA.

Selection is said to be strong if the fittest individuals have a much greater probability of reproducing than less fit ones. Selection is said to be weak if the fittest individuals have only a slightly greater probability of reproducing than the less fit ones. If the selection method is too strong, the genes of the fittest individuals may dominate the next generation population even though they may be suboptimal. This is known as premature convergence, i.e., the exploitation of a small region of the search space before a thorough exploration of the whole space has been achieved. On the other hand, if the selection method is too weak, less fit individuals are given too much opportunity to reproduce and evolution may become too slow. This can be a particular problem during the latter stages of evolution, when the whole population may have congregated within a smooth and fairly flat region of the search space. All

individuals in such a region would have similar, relatively high fitnesses and, thus, it may be difficult to select among them. This can result in stalled evolution, i.e., there is insufficient variance in fitness across the population to drive further evolution.

3.2. Genetic Algorithm in the Design of FIR Filters

3.2.1 Genetic Algorithm and Operators

A GA has been implemented for the design of discrete coefficient FIR filter design in this thesis. GA consists of basic genetic operators such as mutation, crossover and evaluation. Each filter is represented as a set of integer filter coefficients $\{h_1, h_2, h_3, ... h_n\}$. Filter performance is measured using Normalized Peak Ripple Magnitude (NPRM). Filter complexity that is number of adders required to implement filter coefficients is measured using RAG-n and CSD algorithm. The advantage of using integer coefficient representation is that it is a more natural way in which to describe filters. Chromosomes consisting of integer genes have been used in GA to represent the filter coefficients rather than a binary coding. This provides a more natural implementation, which facilitates the use of customized genetic operators. Generation of the initial population and the genetic operators that have been used in GA in this thesis are as follows:

Initial Population: The set of infinite precision filter coefficients is determined by using the Park-McClellan algorithm. These coefficients are rounded to the nearest finite wordlength number. An initial population that consists of a set of filter coefficients is created by randomly adding an integer between ± 2 to each rounded filter coefficient.

Crossover: The crossover operator in GA consists of uniform and single-point crossover on the integer genes, such that two new chromosomes generated have a mixture of coefficients from each parent.

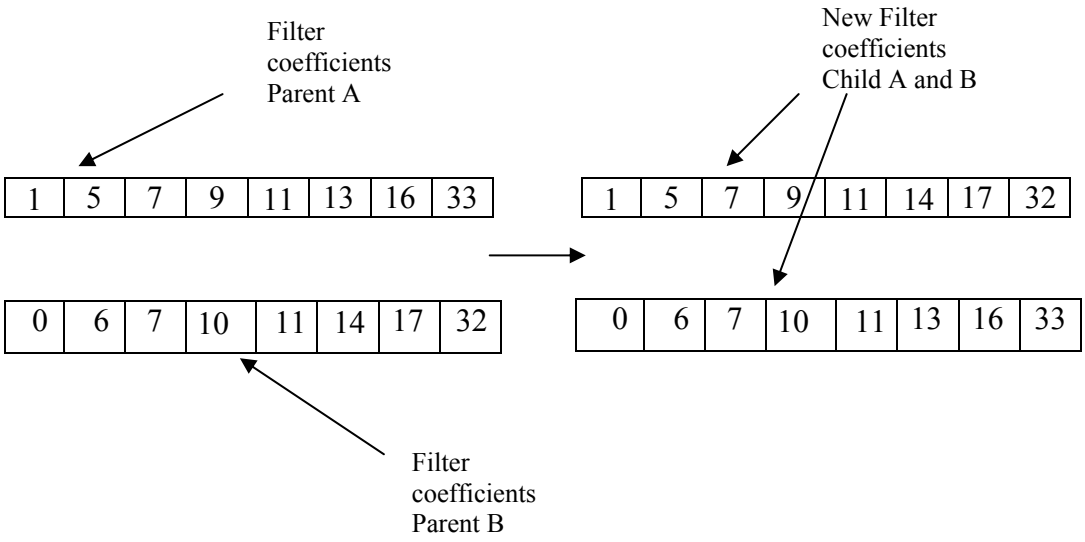


Figure 3.6 GA Single-point Crossover

Mutation 1: Addition/subtraction of a small random power-of-two value to the filter coefficients is the main mutation technique.

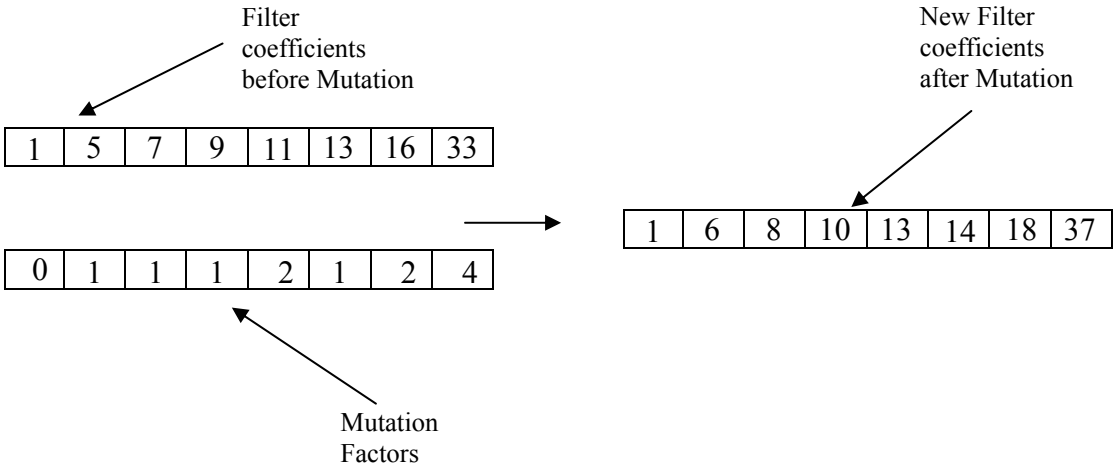


Figure 3.7 GA Mutation 1

Mutation2: Scaling (and quantizing) all the filter coefficients within a chromosome is the second mutation technique that is used. This allows a change to the overall gain of the filter with little change to the filter performance.

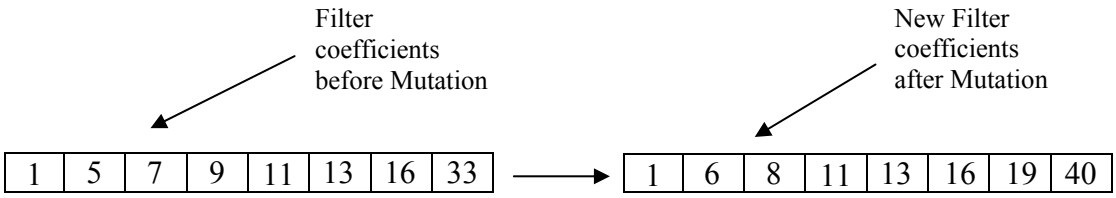


Figure 3.8 GA Mutation 2

3.2.2 Fitness Function in GA

The fitness function used in GA is the NPRM [10]. NPRM has been a convenient design criterion for discrete coefficient FIR filters. For many filter implementations the absolute value of the passband gain is of less importance. Instead the relative attenuation between the passband and stopband is of interest. This is referred to as NPRM.

Let h_i, $i = 0,1,2, ...,M,$ denote the impulse response sequence of a linear phase FIR filter. The amplitude of the frequency response of the designed filter is expressed as

$$H(w) = \sum_{i=0}^{M} a_i T(w, i) \tag{3.1}$$

$$N = \frac{M}{2} - 1 \text{ if } M \text{ is even} \tag{3.2}$$

$$N = \frac{M-1}{2} \text{ if } M \text{ is odd} \tag{3.3}$$

where a_i's are related to h_i's and *T(w,i)*'s are sinusoidal functions. *T(w,i)* can be *cos(wi), cos(w(i+0.5)), sin(wi)* or *sin(w(i+0.5))* depending on the type of the symmetry of the impulse response and, the type of *M* as even or odd. Let *W(w)* be a positive weighting function and $H_d(w)$ be the amplitude of the frequency response of the desired filter. Then, NPRM is defined as δ / g, such that

$$\begin{aligned} \delta &= \max\left|E(w)\right| = \left\|E(w)\right\|_\infty \\ F &= \left\{w \middle| w \in [0,\pi] - \{transitionbands\}\right\} \end{aligned} \tag{3.4}$$

where $E(w)$, the error function is given by

$$E(w) = W(w)(gH_d(w) - H(w)) \tag{3.5}$$

and g is the "passband gain" or "filter gain". The minimization of NPRM is to be carried out with respect to h_i's and g subject to h_i's take quantized values. According to this problem definition, filter gain has the maximum value that minimizes NPRM for a given set of filter coefficients. An explicit expression for g has not been provided in the literature in this sense. In particular, g has been set as

$$g = \frac{P_{\max} + P_{\min}}{2} \tag{3.6}$$

where $P_{\max}$ and $P_{\min}$ are the maximum and minimum values, respectively of $H(w)$ in the passband. The above g definition is not general since it attempts to minimize the maximum passband deviation without considering the stopband deviation.

Let $S_{\max}$ denote the maximum value of $|H(w)|$ in the stopband. Then, for a given set of filter coefficients the filter gain that minimizes δ / g is given by

$$g = \frac{P_{\max} + P_{\min}}{2} \quad \text{if} \quad \frac{P_{\max} - P_{\min}}{2} > S_{\max} \tag{3.7}$$

$$g = P_{\min} + S_{\max} \quad \text{if} \quad \frac{P_{\max} - P_{\min}}{2} < S_{\max} \tag{3.8}$$

In setting this equality, the following assumptions have been made:

- Band specifications of the desired amplitude response, $H_d(w)$, are given as constant levels.
- Without loss of generality, $H_d(w) = 1$ in the passband and $H_d(w) = 0$ in the stopband.

3.2.4 GA Algorithm Flowchart

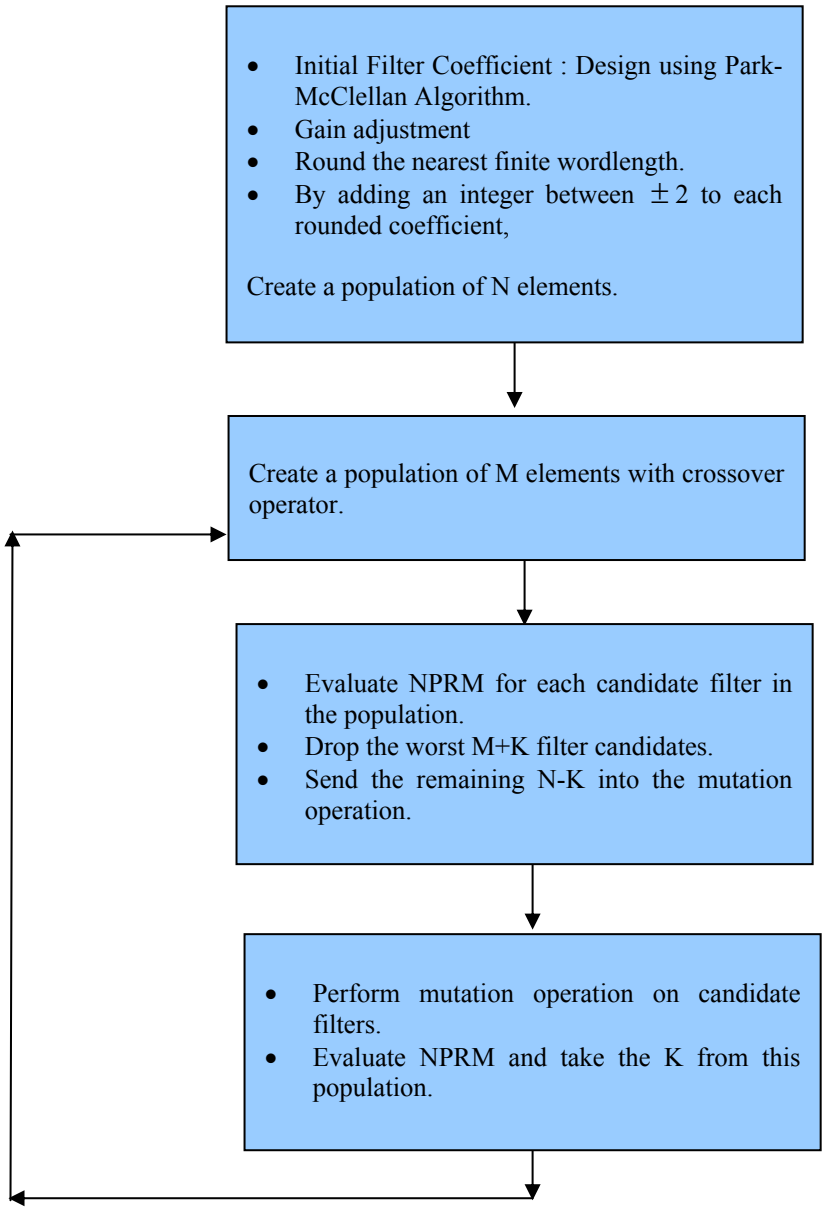


Figure 3.9 GA Flowchart

CHAPTER 4

COMPARISON OF FILTER DESIGN METHODS IN TERMS OF FILTER PERFORMANCE AND COMPLEXITY

First, an algorithm that reduces the number of adders used in the multiplier block of FIR digital filters namely RAG-n [8] is explained. Then, different filters designed under uniform and nonuniform quantization constraints are implemented in transposed direct form. In these implementations, implementation complexity is reduced by using RAG-n algorithm. The results are compared in terms of performance and implementation complexity. Implementation complexity is the number of adders required to implement the filter coefficients and calculated via RAG-n [8] and CSD algorithm. It has been seen from the results that RAG-n [8] has provided 50% cost saving over the CSD and SPT design.

4.1. Digital FIR Filter Implementation

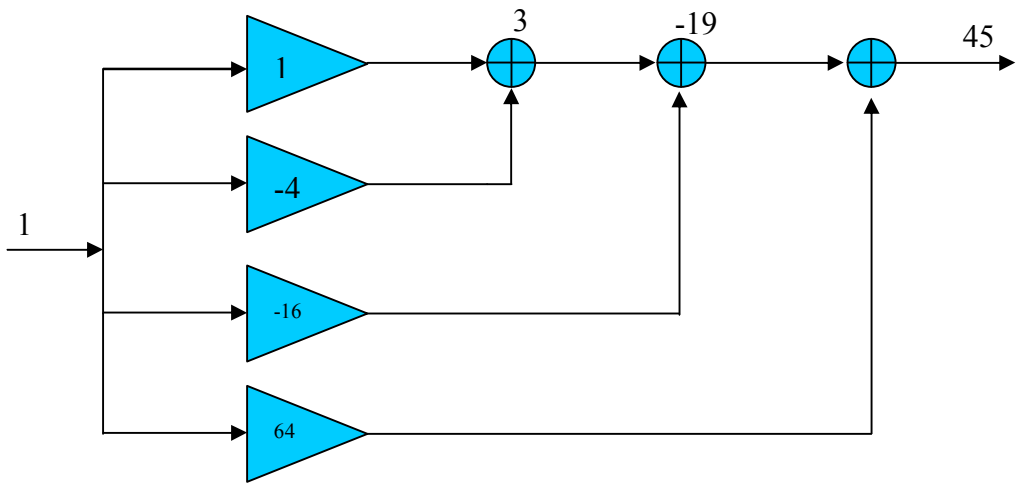
4.1.1 Introduction

It has been shown that implementation complexity of FIR digital filters with integer coefficients can be reduced by using multiplier blocks to exploit redundancy across the coefficients [8]. This approach uses the transposed form FIR filter structure (See Figure 1.1) and the redundancy in the multiplication process. It replaces multiplications by decomposing them into simple operations such as additions, subtractions and shift and tries to reduce the number of simple operations. The multiplier block (See Figure 1.2) in the transposed form FIR filter structure is significant since some adders and shifters in this multiplier block can be shared among different multiplications. RAG-n algorithm [8] focuses on this multiplier block and tries to make it as simple as possible using less adders and subtractors.

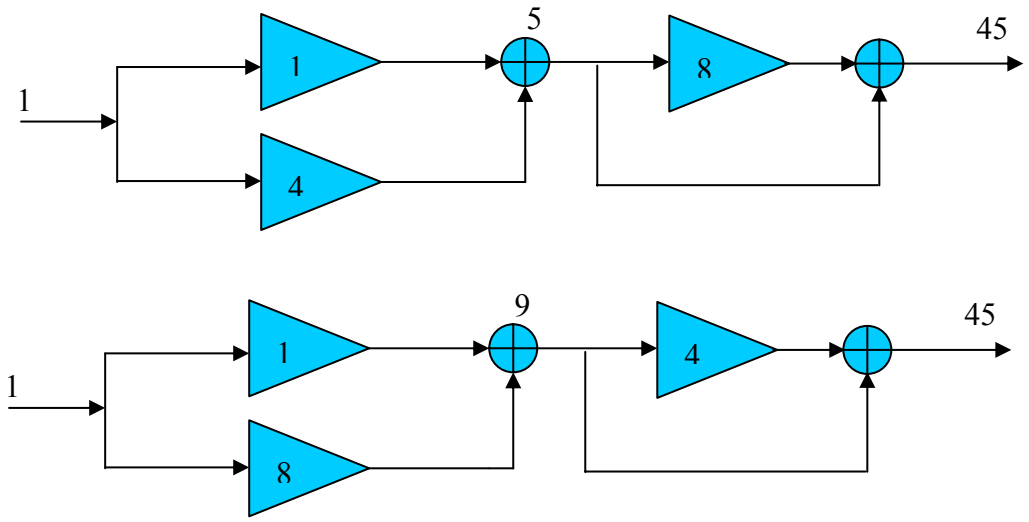
The filter design methods [1]-[7], [11], [13]-[16], [18], [20], [22]-[23] that use different quantization techniques mentioned in Chapter 1 and constrain the filter coefficients to SPT terms have never been thought using RAG-n after designing filter to reduce the implementation complexity. This algorithm has been applied to these filter design methods and significant cost savings have been achieved in filter implementation.

4.1.3 Graph Representation for Multiplier Block

The graph representation technique is based on the concept of replacing the discrete multipliers in the multiplier block of the transposed form FIR filter structure with a graph which exactly preserves the input-output transfer function for all signals. It is a graphical method to represent multiplication by a constant integer, where each vertex except the initial and terminal vertices means an adder, and each edge is associated with a value to be multiplied with the left vertex of the edge. The value is a positive or negative constant of a power of two. The initial vertex is assigned to 1 and the result of the multiplication is obtained from the terminal vertex. Graph representation utilizes redundancy in the coefficient set in the design of FIR digital filters. An example of multiplier graph is shown Figure 4.1, where the number 45 is synthesized using CSD and the technique in RAG-n. Each vertex of the graph represents an adder, and each edge represents multiplication by a power of two, which can be executed with minimal complexity; in hardware it is wired as a simple shift.



(a)



(b)

Figure 4.1 (a) CSD Representation (b) Represented with fewer Adders than CSD

Number of adders used to implement the filter coefficients determines the complexity of the graph that replaces the multiplier block. Binary shifts incur no cost. For fixed-point filter coefficients with least significant bit 2^{-M}, scaling by 2^{M} produces integer coefficients. The required output can be regained at zero cost by rescaling by 2^{-M}. Therefore fixed-point coefficient problem can be considered to be an integer coefficient problem. The followings are definitions for terms used in the RAG-n [8] algorithm.

Adder Cost: Number of adders and subtractors required to implement filter coefficients in the multiplier block.

Fundamentals: The values assigned to vertices in the graphs used to represent multiplications.

Equivalent Graphs: The graphs that have the same topology, and can be derived one from the other by scaling inputs to a vertex by 2^{x}, and the fundamental at that vertex 2^{x}, and output edges from that vertex by 2^{-x}.

Odd Fundamental Graph: The unique graph among equivalent graphs that has only odd fundamentals.

4.1.4 The n-Dimensional Reduced Adder Graph (RAG-n) Algorithm [8]

A flowchart of the n-Dimensional Reduced Adder Graph algorithm is shown in Figure 4.6 (a) and (b). The two parts of the algorithm is as follows:

- Optimal Part: Filter coefficients that are cost-1 (i.e. implementing this filter coefficient requires 1 adder) or producible by other cost-1 coefficients are realized by this part of the algorithm. Minimum adder cost is assured if the set of filter coefficients is completely realized by this part of the algorithm.
- Heuristic Part: Filter coefficients that are not realized in the optimal part are realized in this part of the algorithm. This part of the algorithm uses the filter coefficients realized in the optimal part to implement the higher cost filter coefficients. Minimum adder cost is not assured if this part of the algorithm executes.

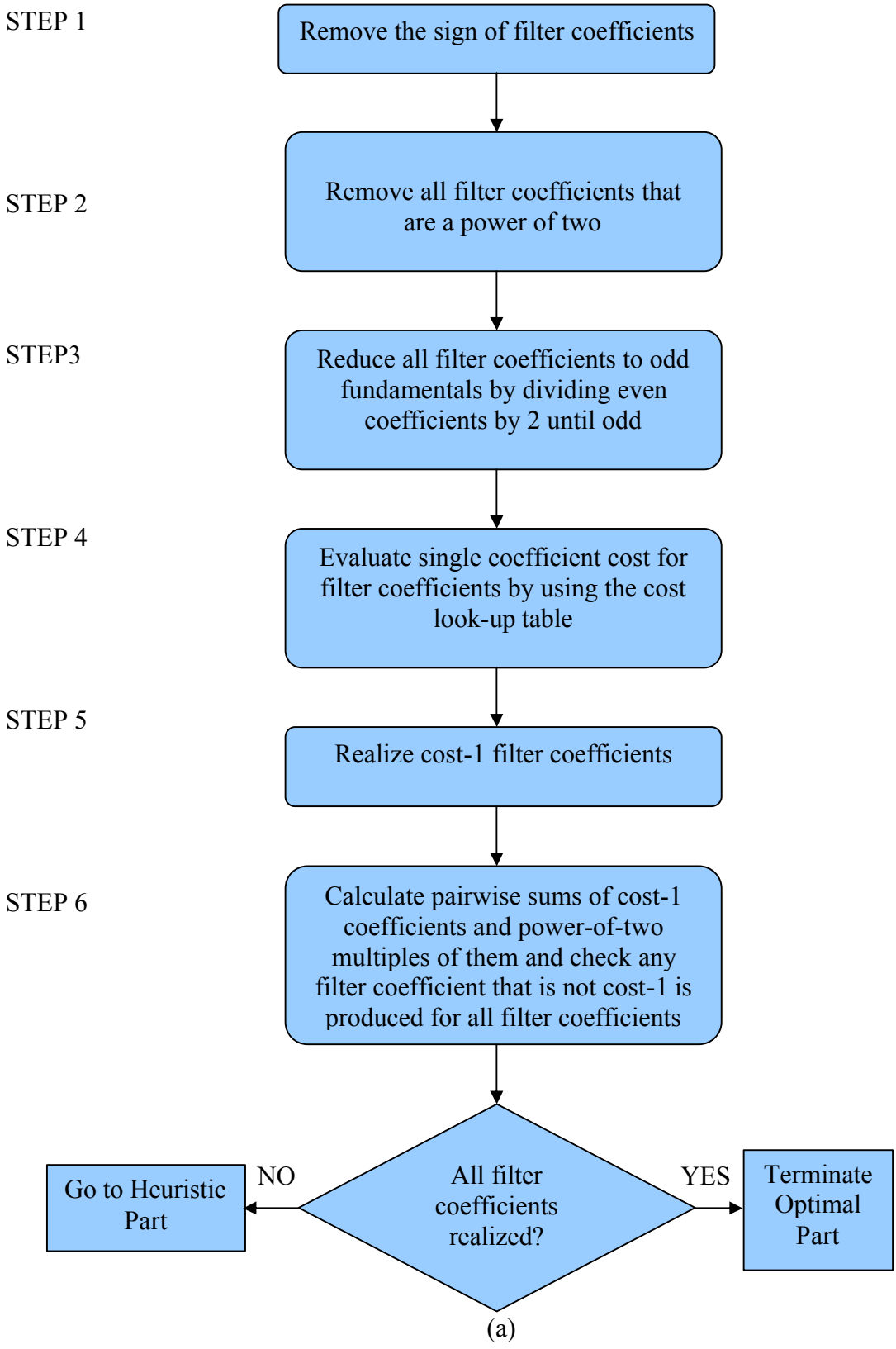
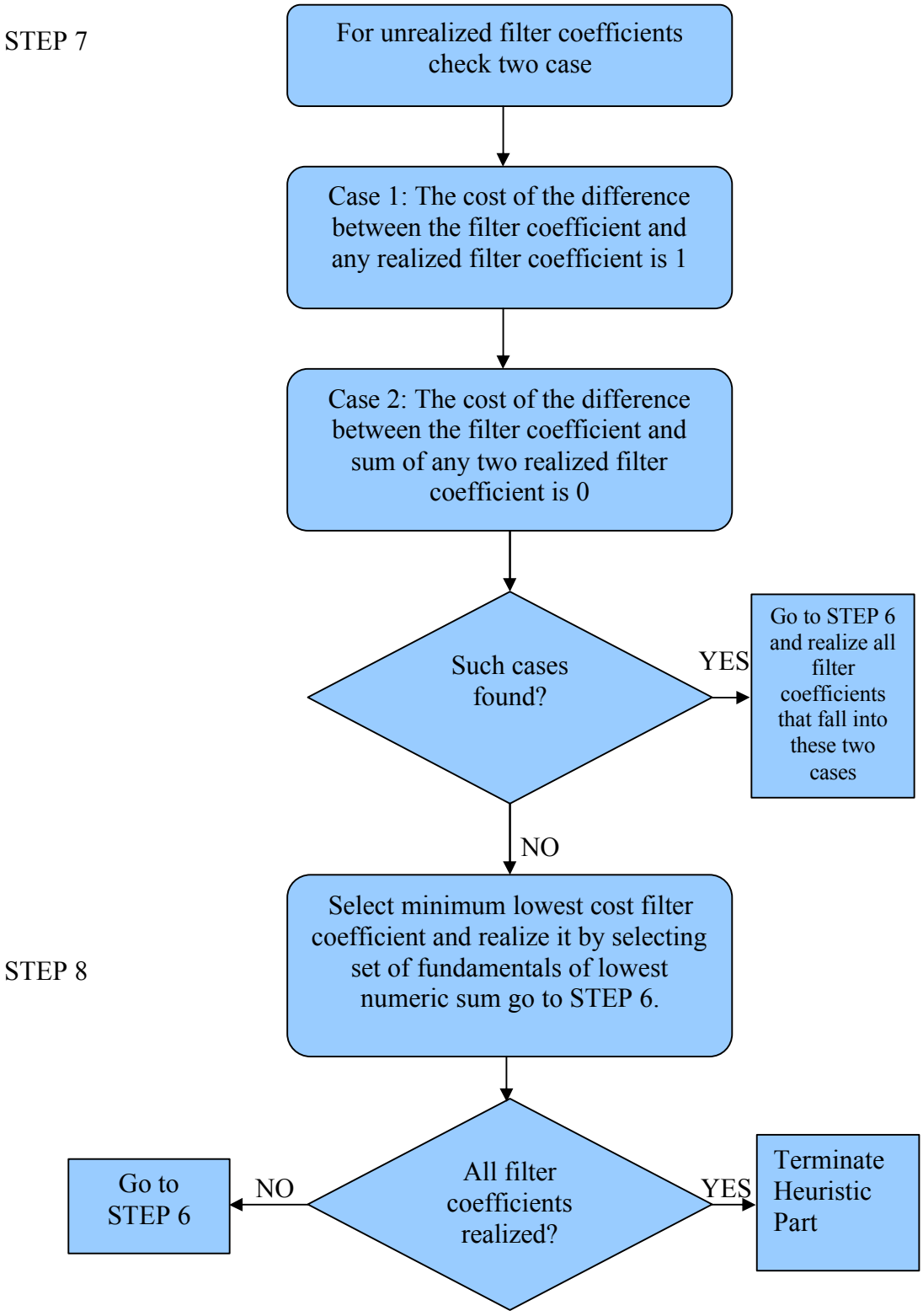


Figure 4.2 (a) Flowchart of the Optimal Part of the RAG-n Algorithm



(b)

Figure 4.2 (b) Flowchart of the Heuristic Part of the RAG-n Algorithm

The algorithm is explained in detail below:

a. Remove the sign of the filter coefficients.

b. Remove all of the filter coefficients that are a power-of-two since they incur no cost other than a hardwired data shift.

c. Reduce all of the filter coefficients to odd fundamentals by dividing 2 until they become odd. Create a set named as "unrealized filter coefficient set" and store the odd valued filter coefficients into this set.

d. Determine the cost for each entry in the unrealized filter coefficient set by using the cost lookup table. This cost gives the number of adders required to implement each filter coefficient reduced to odd value in the "unrealized filter coefficient set".

e. Create the "realized filter coefficient set" for the storage of realized filter coefficients. Into this set, enter all cost-1 filter coefficients and remove these from the "unrealized filter coefficient set".

f. Examine pairwise sums of filter coefficients in the "realized filter coefficient set" with power-of-two multiples of these same filter coefficients. If any of the filter coefficients in the "unrealized filter coefficient set" is produced, remove them from the "unrealized filter coefficient set" and put them in the "realized filter coefficient set".

g. Repeat *f* until no more filter coefficients are added to the "realized filter coefficient set".

If at any time, all of the filter coefficients are realized, the RAG-n algorithm terminates. After *g*, if multiplier block in the transposed form FIR filter structure is completely implemented, then this multiplier block is guaranteed to be optimal that means minimum number of adders and subtractors used in the implementation. The following theorems explain optimality criteria.

Theorem 1. [8] A set of *n* nonrepeated cost-1 or more odd filter coefficients can not be realized using fewer than *n* adders.

Theorem 2. [8] For a set described in theorem 1 to incur an adder cost of *n*, at least one cost-1 must appear in the set.

Steps *a* to *g* form the optimal part of the RAG-n algorithm. After the execution of the optimal part, if any entry exists in the "unrealized filter coefficient set" then execution of the heuristic part of the RAG-n algorithm begins. "Adder

distance" definition is used in the heuristic part of the RAG-n algorithm. The adder distance of a new vertex from an exiting graph is the number of extra adders that are needed to reach new vertex. The aim of the heuristic part of the algorithm is to find the minimum adder distance between the graph created by the optimal part of the algorithm and the remaining terminating vertices, which corresponds to the coefficients not yet realized.

The heuristic part of the algorithm continues as follows:

h. Check each of the "unrealized filter coefficient set" for two different instances of distance 2 vertices:

- The case where the cost of the difference between the filter coefficient of interest in the "unrealized filter coefficient set" and any realized filter coefficient in the "realized filter coefficient set" is 1.
- The case where the cost of the difference between the filter coefficient of interest in the "unrealized filter coefficient set" and the sum of any two realized filter coefficients in the "realized filter coefficient set" is 0.

In both cases, two new fundamentals will be created, the new realized filter coefficient and one other. In the first case, this latter fundamental is the cost-1 difference which was calculated. In the second case it is the sum of the two realized filter coefficients. If such a case is found, add the two fundamentals to the "realized filter coefficient set", and remove the realized filter coefficient from the "unrealized filter coefficient set".

i. Repeat *g* and *h* until no new distance 1 or 2 fundamentals are found.

j. If this point is reached, there are filter coefficients that are at a greater distance than 2 from the existing graph, or at a distance 2 with a topology which is not covered by the two examples in *h*. Therefore an arbitrary choice of fundamentals to add to the "realized filter coefficient set" must be made. Of the "unrealized filter coefficient set", select the minimum fundamental of lowest single-coefficient cost. Realize it by selecting the set of fundamentals which has the lowest numerical sum. Add these

fundamentals to the "realized filter coefficient set" and remove the filter coefficient from the "unrealized filter coefficient set".

k. Repeat steps *g* to *j* until all filter coefficients are realized.

4.1.5 Example Filter Implementations with RAG-n

Example filters will be implemented with RAG-n.

EXAMPLE 1

Filter coefficients are as follows:

$h(0) = 7,\ h(1) = 9,\ h(2) = 14,\ h(3) = 21,\ h(4) = 33,\ h(5) = 36,\ h(6) = 44.$

a. No sign in filter coefficients. $h(0) = 7,\ h(1) = 9,\ h(2) = 14,\ h(3) = 21,$ $h(4) = 33,\ h(5) = 36,\ h(6) = 44.$

b. No power-of-two coefficients.

c. Unrealized filter coefficient set = $[\,h(0) = 7,\ h(1) = 9,\ h(2) = 7,$ $h(3) = 21,\ h(4) = 33,\ h(5) = 9,\ h(6) = 11\,]$ (Filter coefficients divided by 2 until odd)

d. Cost($h(0) = 7$) = 1, Cost($h(1) = 9$) = 1, Cost($h(2) = 7$) = 1, Cost($h(3) = 21$) = 2, Cost($h(4) = 33$) = 1, Cost($h(5) = 9$) = 1, Cost($h(6) = 11$) = 2.

e. Realized filter coefficient set = $[\,h(0) = 7, h(1) = 9, h(2) = 7, h(4) = 33,$ $h(5) = 9\,]$

Unrealized filter coefficient set = $[\,h(3) = 21, h(6) = 11\,]$

f. $h(3) = 2 * h(0) + h(0)$ (Realized)

$h(6) = h(1) + 2$ (Realized)

g. Unrealized filter coefficient set = [ ] so terminate.

Implemented filter coefficients are shown in Figure 4.7.

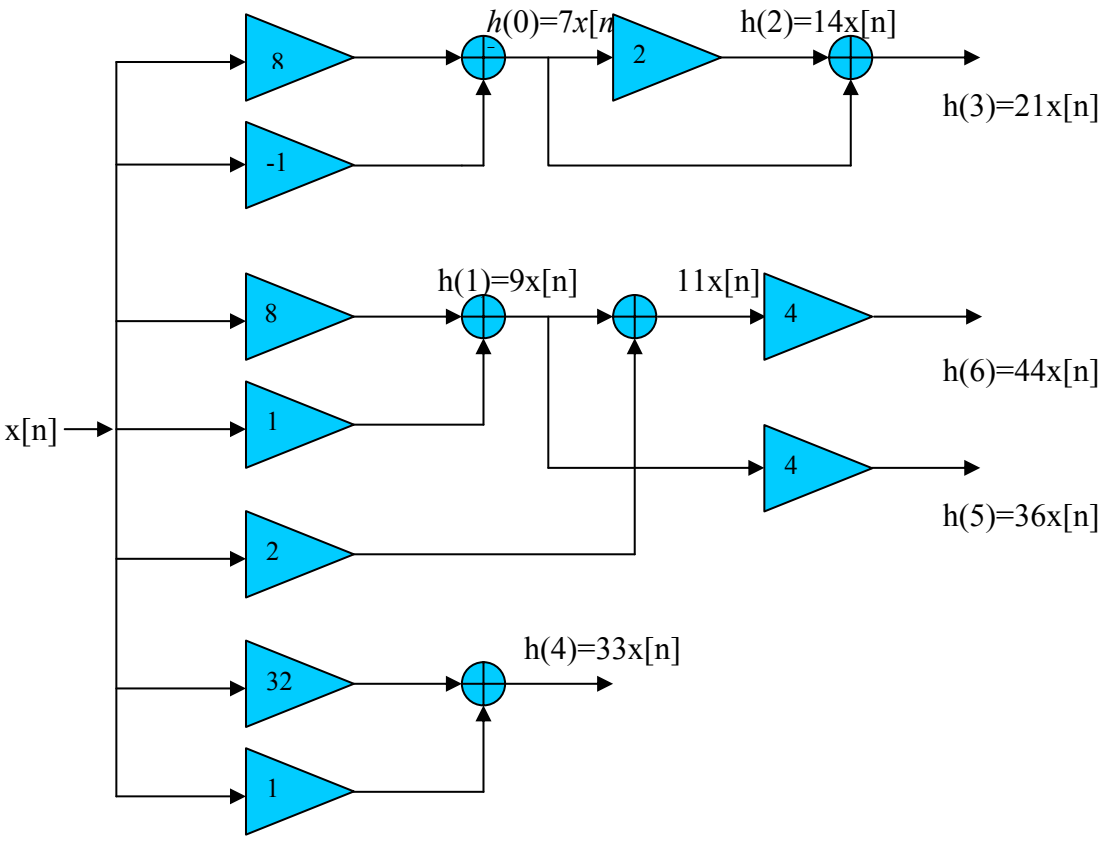


Figure 4.3 RAG-n Implementation of Example Filter 1

Total number of adders used to implement filter coefficients is 5 (adder cost). If CSD algorithm is used for implementation then

$h(0) = 7 = 100\bar{1}_{csd}$, $h(1) = 9 = 1001_{csd}$, $h(2) = 14 = 100\bar{1}0_{csd}$, $h(3) = 21 = 10101_{csd}$,

$h(4) = 33 = 100001_{csd}$, $h(5) = 36 = 100100_{csd}$, $h(6) = 44 = 10\bar{1}0\bar{1}00_{csd}$.

$h(0) = 7 = 8 - 1$, $h(1) = 9 = 8 + 1$, $h(2) = 14 = 16 - 2$, $h(3) = 21 = 16 + 4 + 1$,

$h(4) = 33 = 32 + 1$, $h(5) = 36 = 32 + 4$, $h(6) = 44 = 64 - 16 - 4$.

Total adder cost is 9 adders when CSD is used for filter implementation.

EXAMPLE 2

Filter coefficients are as follows:

$h(0) = 9$, $h(1) = -44$, $h(2) = 208$, $h(3) = 346$.

a. Remove sign in filter coefficients. $h(0) = 9$, $h(1) = 44$, $h(2) = 208$, $h(3) = 346$.

b. No power-of-two coefficients.

c. Unrealized filter coefficient set =[$h(0) = 9$, $h(1) = 11$, $h(2) = 13$, $h(3) = 173$] (Filter coefficients divided by 2 until odd)

d. Cost ($h(0) = 9$) = 1, Cost ($h(1) = 11$) = 2, Cost ($h(2) = 13$) = 2, Cost ($h(3) = 173$) = 3.

e. Realized filter coefficient set = [$h(0) = 9$]

Unrealized filter coefficient set = [$h(1) = 11$, $h(2) = 13$, $h(3) = 173$]

f. $h(1) = h(0) + 2$ (Realized)

$h(2) = h(0) + 4$ (Realized)

g. Realized filter coefficient set = [9 11 13] Unrealized filter coefficient set = [173] not empty so go to heuristic part.

h. $h(3) = 173 = 128 + 45$

$45 = 4 * h(0) + h(0)$ Cost-1 so create 45.

i. $h(newfundamenal) = 45 = 4 * h(0) + h(0)$

$h(3) = 173 = 128 + h(newfundamenal)$

Unrealized filter coefficient set = [ ] so terminate.

Implemented filter coefficients are shown in Figure 4.8.

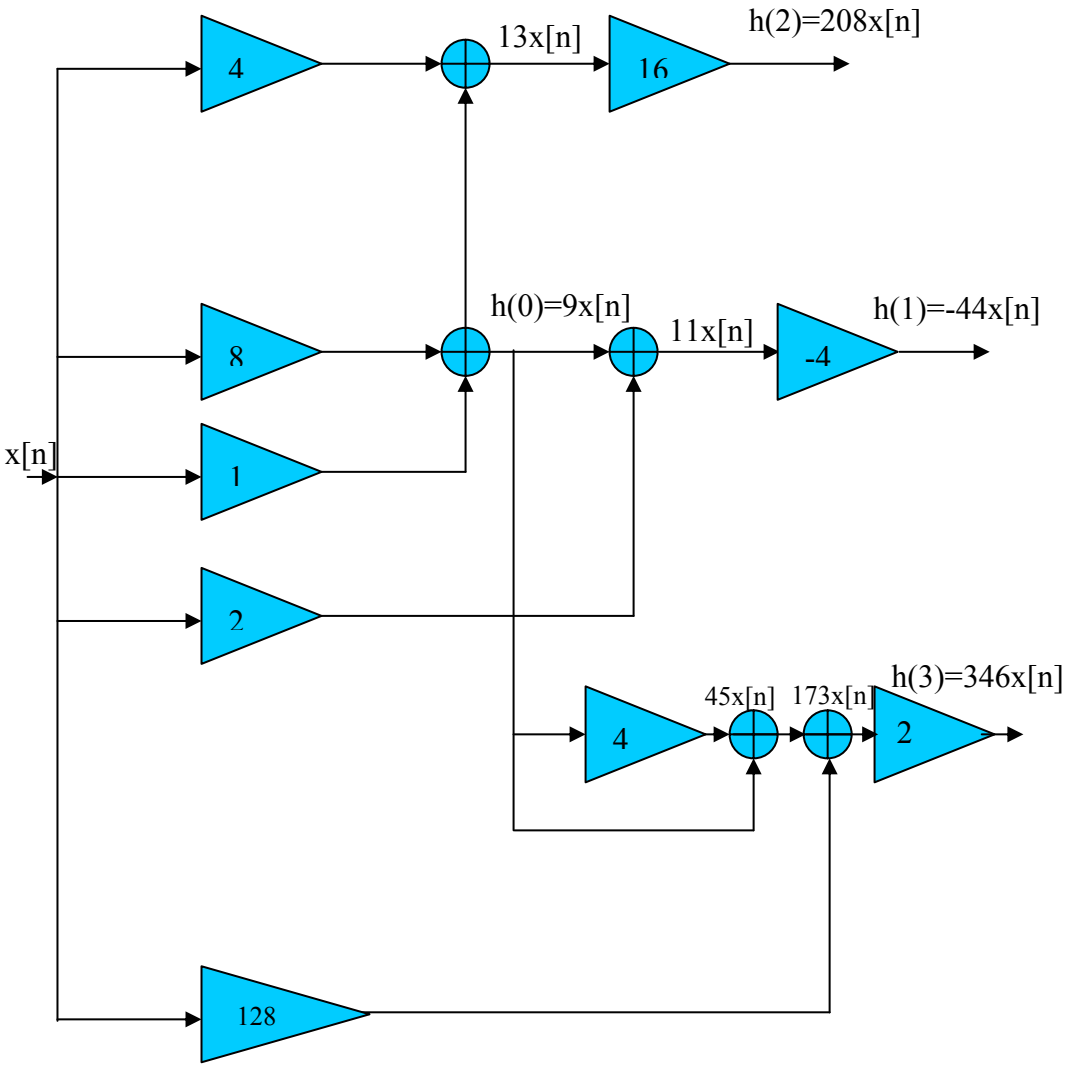


Figure 4.4 RAG-n Implementation of Example Filter 2

Total number of adders used to implement filter coefficients is 5 when RAG-n is used. If CSD algorithm is used for implementation then

$h(0) = 9 = 1001_{csd}$, $h(1) = -44 = \bar{1}010100_{csd}$, $h(2) = 208 = 10\bar{1}010000_{csd}$,

$h(3) = 346 = 10\bar{1}0\bar{1}0\bar{1}010_{csd}$.

$h(0) = 9 = 8 + 1$, $h(1) = -44 = 64 + 16 + 4$, $h(2) = 208 = 256 - 64 + 16$,

$h(3) = 346 = 512 - 128 - 32 - 8 + 2$.

Total adder cost is 9 adders when CSD is used for filter implementation. As it is seen RAG-n provides near 45% reduction in the number of adders used to implement filter coefficients than CSD. This reduction enables to design low cost and high performance digital FIR filters in ASICs.

4.2 Experimental Results

Test Case 1

The two example filters of "An Improved Search Algorithm for the Design of Multiplierless FIR filters with Powers-of-Two Coefficients [1]" by H. Samueli are used for this test case.

Filter (1) Requirements are as follows:

(a) Symmetrical Impulse Response; Low pass

(b) Passband edges at 0.0 and 0.15 sampling frequency

(c) Stopband edges at 0.25and 0.50 sampling frequency

(d) Weights in passband and stopband 1.

(e) Wordlength 9 bit.

Table 4.1 Summary of Filter (1) Design in Test Case 1

Filter Length N = 25 Infinite Wordlength NPRM = -46.03 dB				
Filter Design Method	**Quant. Type**	**NPRM (dB)**	**Complexity (CSD) (Adders)**	**Complexity (RAG-n) (Adders)**
Samueli [1]	2-3 SPT	-42.18	11	6
GA	Integer	-43.74	14	8
GA	Integer	-43.93	13	6

Table 4.1 summarizes the results for the given filter specifications. As it is seen from the results, the filter designed by GA has slightly improved attenuation than the filter designed by Samueli's [1] local search method. For the implementation of the filter, it is very clear that a RAG-n [8] design of filter uses less number of adders than a CSD design for the same attenuation by exploiting the redundancy in the filter coefficient set and use of transposed form filter structure. Multiple runs of GA have found filters with improved or degraded attenuations. Adder cost also changes with multiple runs of GA. This shows the need for the multiple trials to get improved attenuations and reduced adder cost since random selections are made

inside algorithm. This is a disadvantage of GA. Figure 4.5 and 4.6 show frequency responses of filters designed by GA (red) and Parks-McClellan (black) algorithm for the first filter in test case 1.

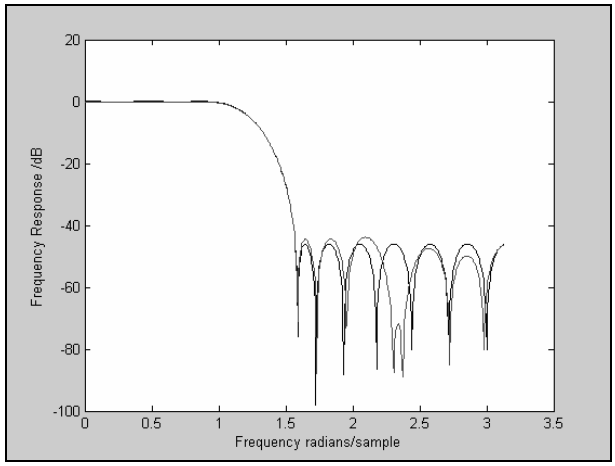


Figure 4.5 Frequency Response (-43.74 dB) (Red: GA Implementation; Black: Remez Implementation for the Filter (1) in Test Case 1)

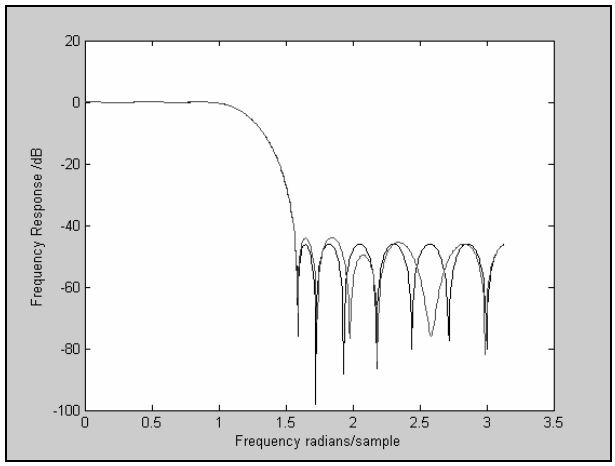


Figure 4.6 Frequency Response (-43.93 dB) (Red: GA Implementation; Black: Remez Implementation for the Filter (1) in Test Case 1)

Filter (2) Requirements are as follows:

(a) Symmetrical Impulse Response; Low pass

(b) Passband edges at 0.0 and 0.021 sampling frequency

(c) Stopband edges at 0.07and 0.50 sampling frequency

(d) Weights in passband and stopband 1.

(e) Wordlength 14 bit.

Table 4.2 Summary of Filter (2) Design in Test Case 1

Filter Length N = 60 Infinite Wordlength NPRM = -55.47 dB				
Filter Design Method	**Quant. Type**	**NPRM (dB)**	**Complexity (CSD) (Adders)**	**Complexity (RAG-n) (Adders)**
Samueli [1]	3-4 SPT	-38.75	57	25
GA	Integer	-54.91	61	28

Table 4.2 summarizes the results for the given filter specifications. NPRM of the filter designed by GA is much better than the filter designed by Samueli's [1] local search method. For the implementation of the filter, it is clear that RAG-n [8] design uses less number of adders than CSD design for the same NPRM value. RAG-n [8] provides approximately 55% cost saving in the implementation over the CSD resulting in low cost ASICs implementations. Since most of the coefficients of the given filter are cost-1, cost-2 and producible from each other, RAG-n implements the filter with the optimal adder cost than CSD. Figure 4.7 shows frequency responses of filters designed by GA (red) and Parks-McClellan (black) algorithm for the second filter in test case 1.

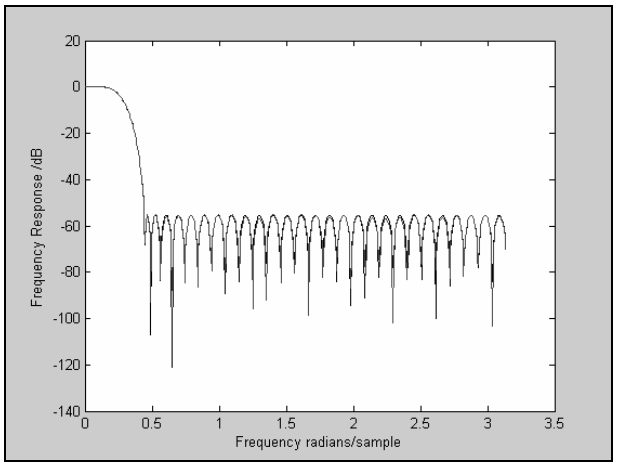


Figure 4.7 Frequency Response (Red: GA Implementation; Black: Remez Implementation for the Filter (2) in Test Case 1)

Test Case 2

The example filter of "Design of Optimal Paralel Filter Using a Parallel Genetic Algorithm [2]" by D. J. Xu and M. L. Daley is used for the test case.

Filter Requirements are as follows:

(a) Symmetrical Impulse Response; Low pass

(b) Passband edges at 0.0 and 0.20 sampling frequency

(c) Stopband edges at 0.25and 0.50 sampling frequency

(d) Weights in passband and stopband 1.

(e) Wordlength 10 bit.

Table 4.3 Summary of Filter Design in Test Case 2

Filter Length N = 40 Infinite Wordlength NPRM = -39.38 dB				
Filter Design Method	**Quant. Type**	**NPRM (dB)**	**Complexity (CSD) (Adders)**	**Complexity (RAG-n) (Adders)**
D.J. Xu M.L. Daley [2]	Integer	-35.89	17	8

Table 4.3 Summary of Filter Design in Test Case 2 (Con't)

GA	Integer	-36.81	18	9
GA	Integer	-37.82	24	12
GA	Integer	-38.58	24	12

Table 4.3 summarizes the results for the given filter specifications. The filters designed by GA have slightly better attenuation values than the filter designed by [2]'s but with increased adder cost when RAG-n [8] is used realize filter coefficients. For the implementation of the filter, RAG-n design uses less number of adders than CSD design for the same NPRM value. RAG-n adder cost is approximately 50% of the CSD adder cost. Figure 4.8, 4.9, and 4.10 show frequency responses of filters designed by GA (red) and Parks-McClellan (black) algorithm.

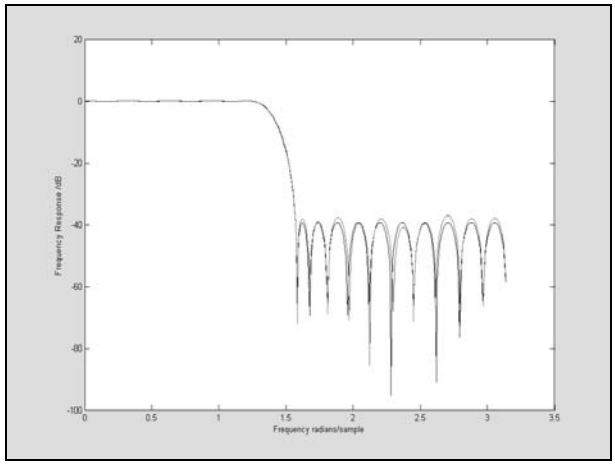


Figure 4.8 Frequency Response (-36.81 dB) (Red: GA Implementation; Black: Remez Implementation for the Filter in Test Case 2)

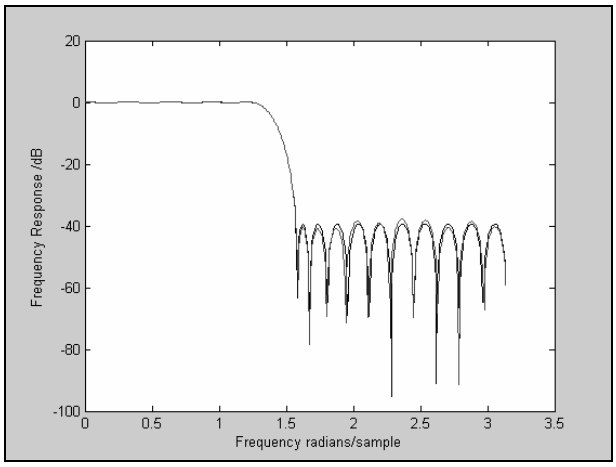


Figure 4.9 Frequency Response (-37.82 dB) (Red: GA Implementation; Black: Remez Implementation for the Filter in Test Case 2)

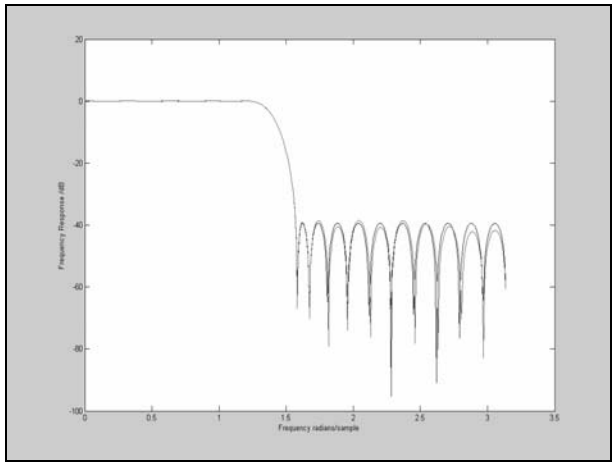


Figure 4.10 Frequency Response (-38.58 dB) (Red: GA Implementation; Black: Remez Implementation for the Filter in Test Case 2)

Test Case 3

The example filter of "Design of Optimal Finite Wordlength FIR Digital Filters Using Integer Programming Techniques [3]" by D. M. Kodek is used for the test case.

Filter Requirements are as follows:

(a) Symmetrical Impulse Response; Low pass

(b) Passband edges at 0.0 and 0.20 sampling frequency

(c) Stopband edges at 0.25and 0.50 sampling frequency

(d) Weights in passband and stopband 1.

Table 4.4 Summary of Filter Design in Test Case 3

Filter Length N = 21 Wordlength = 7 Infinite Wordlength NPRM = -25.21 dB				
Filter Design Method	**Quant. Type**	**NPRM (dB)**	**Complexity (CSD) (Adders)**	**Complexity (RAG-n) (Adders)**
Kodek [3]	Integer	-23.86	6	3
GA	Integer	-23.86	6	3
GA	Integer	-24.06	9	4
GA	Integer	-24.38	11	6

Table 4.5 Summary of Filter Design in Test Case 3

Filter Length N = 40 Wordlength = 10 Infinite Wordlength NPRM = -39.38 dB				
Filter Design Method	**Quant. Type**	**NPRM (dB)**	**Complexity (CSD) (Adders)**	**Complexity (RAG-n) (Adders)**
Kodek [3]	Integer	-36.07	18	9
GA	Integer	-36.81	18	9
GA	Integer	-37.82	24	12
GA	Integer	-38.58	24	12

Table 4.4 and 4.5 summarizes the results for the given filter specifications. GA designed the same or improved performance filters with multiple runs with increased adder cost than [3]'s integer programming method [3]. For the

implementation of the filter, RAG-n [8] design uses less number of adders than CSD design for the same NPRM value. RAG-n adder cost is approximately 50% of the CSD adder cost. Figure 4.11, 4.12, 4.13, 4.14, 4.15, and 4.16 show frequency responses of filters designed by GA (red) and Parks-McClellan (black) algorithm.

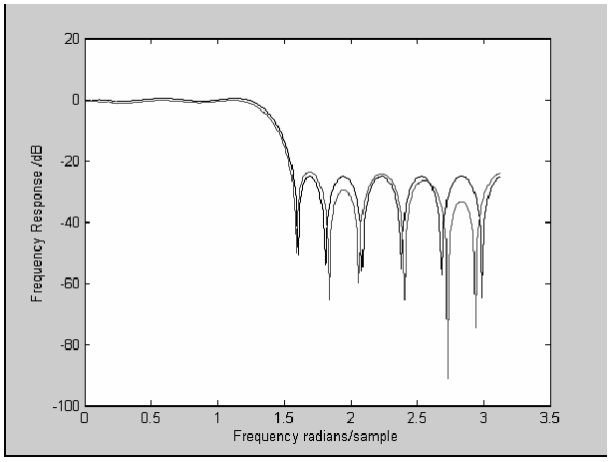


Figure 4.11 Frequency Response (-23.86 dB)(Red: GA Implementation; Black: Remez Implementation for the Filter (Filter Length 21, Wordlength 7 Bit) in Test Case 3)

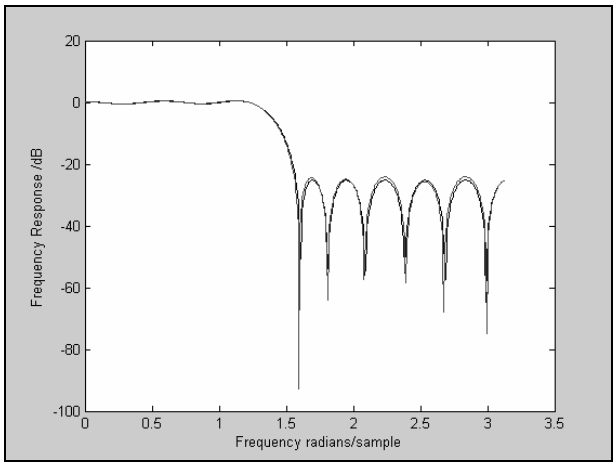


Figure 4.12 Frequency Response (-24.06 dB) (Red: GA Implementation; Black: Remez Implementation for the Filter (Filter Length 21, Wordlength 7 Bit) in Test Case 3)

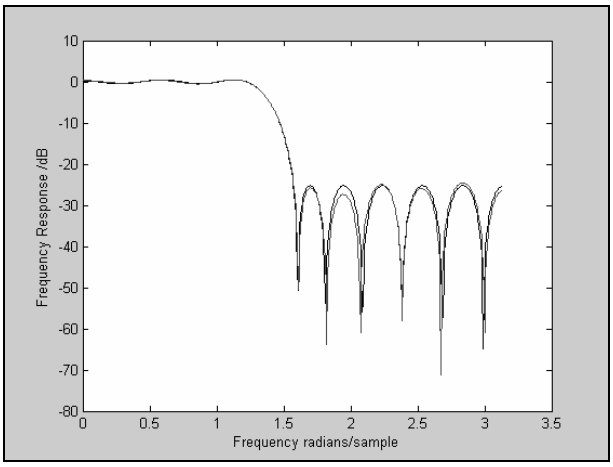


Figure 4.13 Frequency Response (-24.38 dB) (Red: GA Implementation; Black: Remez Implementation for the Filter (Filter Length 21, Wordlength 7 Bit) in Test Case 3)

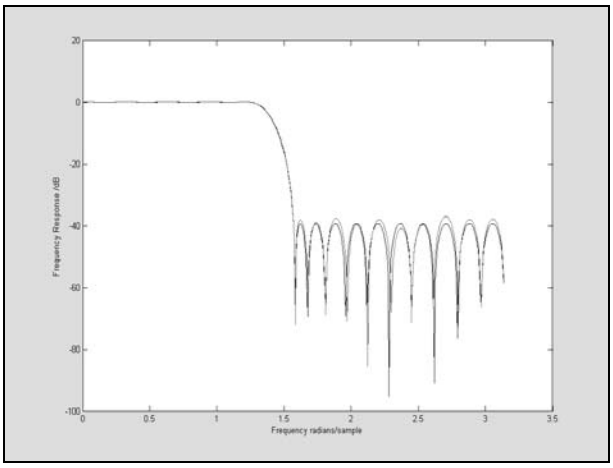


Figure 4.14 Frequency Response (-36.81 dB) (Red: GA Implementation; Black: Remez Implementation for the Filter (Filter Length 40, Wordlength 10 Bit) in Test Case 3)

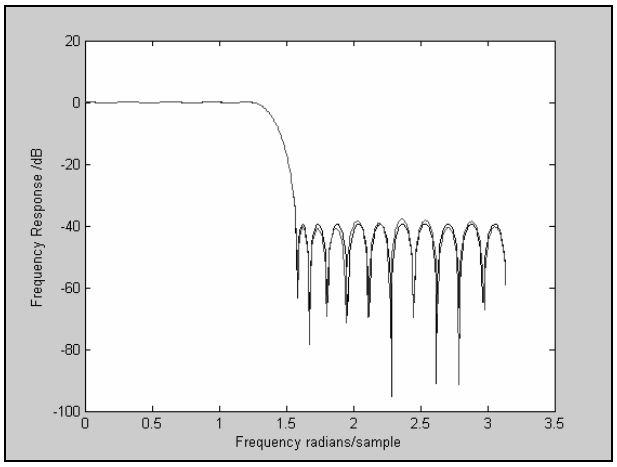


Figure 4.15 Frequency Response (-37.82 dB) (Red: GA Implementation; Black: Remez Implementation for the Filter (Filter Length 40, Wordlength 10 Bit) in Test Case 3)

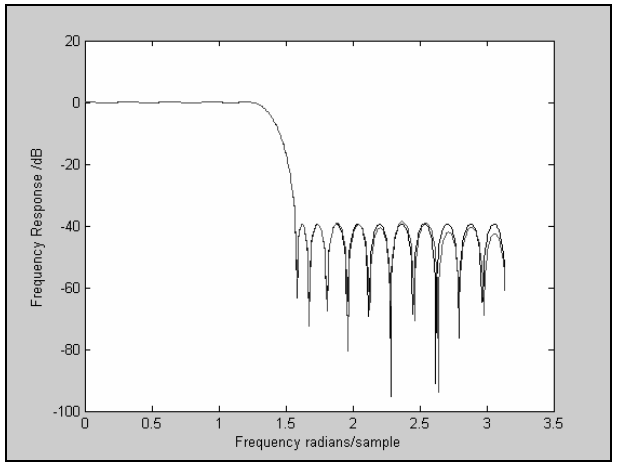


Figure 4.16 Frequency Response (-38.58 dB) (Red: GA Implementation; Black: Remez Implementation for the Filter (Filter Length 40, Wordlength 10 Bit) in Test Case 3)

Test Case 4

The example filter of "A Trellis Search Algorithm for the Design of FIR filters with Signed-Powers-of-Two Coefficients [4]" by C. L. Chen and A. N. Willson is used for the test case.

Filter Requirements are as follows:

(a) Symmetrical Impulse Response; Low pass
(b) Passband edges at 0.0 and 0.15 sampling frequency
(c) Stopband edges at 0.25and 0.50 sampling frequency
(d) Weights in passband and stopband 1.
(e) Wordlength 12 bit.

Table 4.6 Summary of Filter Design in Test Case 4

Filter Length N = 28 Infinite Wordlength NPRM = -51.42 dB				
Filter Design Method	**Quant. Type**	**NPRM (dB)**	**Complexity (CSD) (Adders)**	**Complexity (RAG-n) (Adders)**
Trellis Search [4]	Free Allocation	-50.07	17	9
GA	Integer	-50.85	24	12

Table 4.6 summarizes the results for the given filter specifications. The filter designed by GA has a slightly improved attenuation than Trellis Search's filter with increased adder cost in terms of RAG-n implementation. Trellis Search uses free allocation method for filter coefficient quantization and designs filter with coefficients expressible as Signed-Power-of-Two (SPT) terms to reduce implementation complexity. The filter designed by GA and implemented with RAG-n uses less number of adders than the one designed by Trellis Search and implemented by CSD. Results show considerable reduction in number of adders required for the implementation when RAG-n is used. Figure 4.17 shows frequency responses of filters designed by GA (red) and Parks-McClellan (black) algorithm.

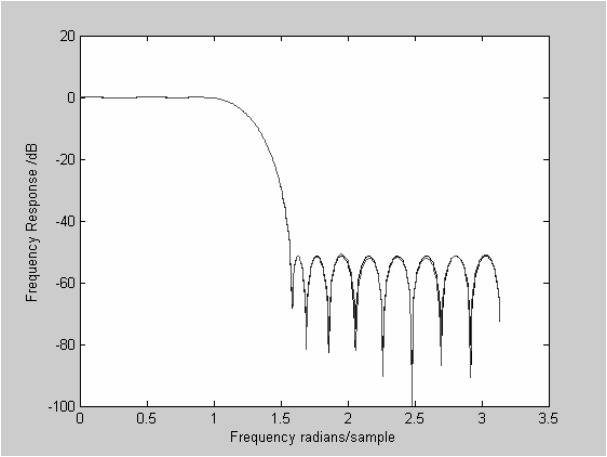


Figure 4.17 Frequency Response (Red: GA Implementation; Black: Remez Implementation for the Filter in Test Case 4)

Test Case 5

The example filter of "A new approach to Discrete Coefficient FIR Digital Filter Design by Simulated Annealing [5]" by T. Çiloğlu and Z. Ünver is used for the test case.

Filter Requirements are as follows:

(a) Symmetrical Impulse Response; Low pass

(b) Passband edges at 0.0 and 0.20 sampling frequency

(c) Stopband edges at 0.25 and 0.50 sampling frequency

(d) Weight in passband 1.

Table 4.7 Summary of Filter Design in Test Case 5

Filter Length N = 15 Wordlength = 5 Stopband Weight = 1 Infinite Wordlength NPRM = -18.46 dB				
Filter Design Method	**Quant. Type**	**NPRM (dB)**	**Complexity (CSD) (Adders)**	**Complexity (RAG-n) (Adders)**
SA [5]	2 SPT	-17.64	4	3
GA	Integer	-16.66	3	3
GA	Integer	-17.80	5	3
Filter Length N = 25 Wordlength = 5 Stopband Weight = 1 Infinite Wordlength NPRM = -28.04 dB				
SA [5]	2 SPT	-23.41	2	2
GA	Integer	-23.41	2	2
Filter Length N = 15 Wordlength = 7 Stopband Weight = 10 Infinite Wordlength NPRM = -11.09 dB				
SA [5]	2 SPT	-10.37	7	3
GA	Integer	-10.48	9	4
Filter Length N = 25 Wordlength = 7 Stopband Weight = 10 Infinite Wordlength NPRM = -18.21 dB				
SA [5]	2 SPT	-16.14	9	3
GA	Integer	-15.85	8	5
GA	Integer	-16.14	9	3

Table 4.7 Summary of Filter Design in Test Case 5 (Con't)

Filter Length N = 35 Wordlength = 7 Stopband Weight = 10 Infinite Wordlength NPRM = -25.57 dB				
Filter Design Method	**Quant. Type**	**NPRM (dB)**	**Complexity (CSD) (Adders)**	**Complexity (RAG-n) (Adders)**
SA [5]	2 SPT	-17.58	8	3
GA	Integer	-17.58	8	3
GA	Integer	-17.59	8	5

Filter Requirements are as follows:

(a) Symmetrical Impulse Response; Low pass

(b) Passband edges at 0.0 and 0.1 sampling frequency

(c) Stopband edges at 0.14 and 0.50 sampling frequency

(d) Weights in passband and stopband 1.

(e) Wordlength 13 bit.

Table 4.8 Summary of Filter Design in Test Case 5

Filter Length N = 63 Infinite Wordlength NPRM = -48.05 dB				
Filter Design Method	**Quant. Type**	**NPRM (dB)**	**Complexity (CSD) (Adders)**	**Complexity (RAG-n) (Adders)**
SA [5]	2 SPT	-39.52	27	7
GA	Integer	-47.63	61	25

Table 4.7 and 4.8 summarize the results for the given filter specifications. The filter designed by GA has nearly same (for table 4.6) and better (for table 4.7) attenuation than SA's [5] filters with same or increased adder cost in terms of RAG-

n. The filters designed by GA and implemented with RAG-n uses less number of adders than the one designed by SA and implemented with CSD. The need for multiple trials for GA and SA to get the satisfactory results and the need for the experience to adjust the algorithm parameters are the major disadvantages GA and SA. Figure 4.18-26 show frequency responses of filters designed by GA (red) and Parks-McClellan (black) algorithm.

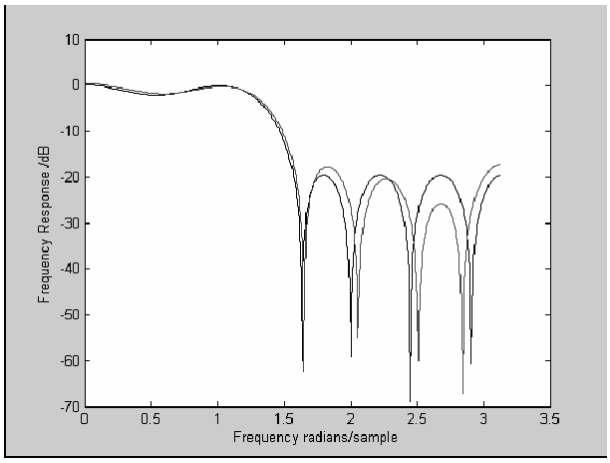


Figure 4.18 Frequency Response (-16.66 dB) (Red: GA Implementation; Black: Remez Implementation for the Filter (Filter Length 15, Wordlength 5 Bit) in Test Case 5)

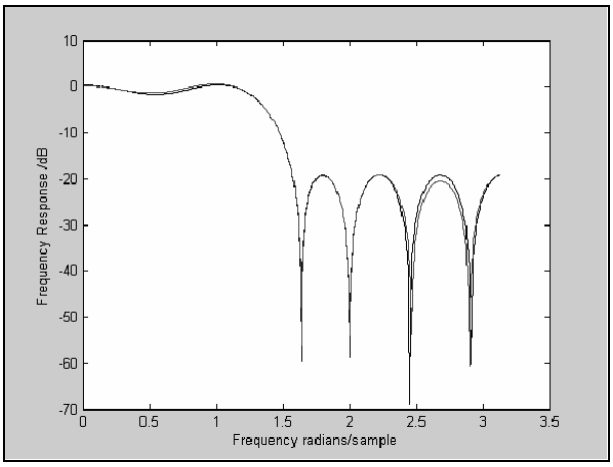


Figure 4.19 Frequency Response (-17.80 dB) (Red: GA Implementation; Black: Remez Implementation for the Filter (Filter Length 15, Wordlength 5 Bit) in Test Case 5)

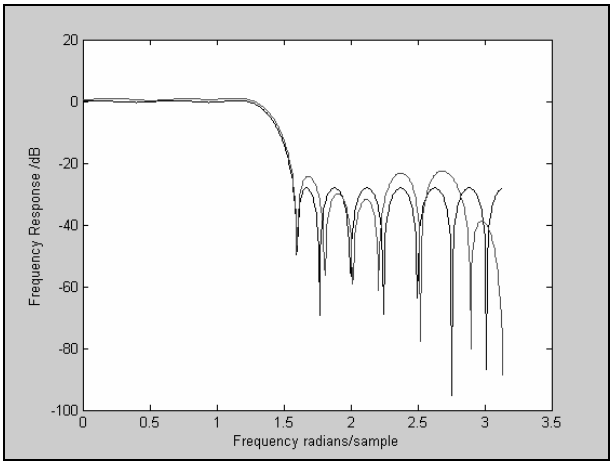


Figure 4.20 Frequency Response (Red: GA Implementation; Black: Remez Implementation for the Filter (Filter Length 25, Wordlength 5 Bit) in Test Case 5)

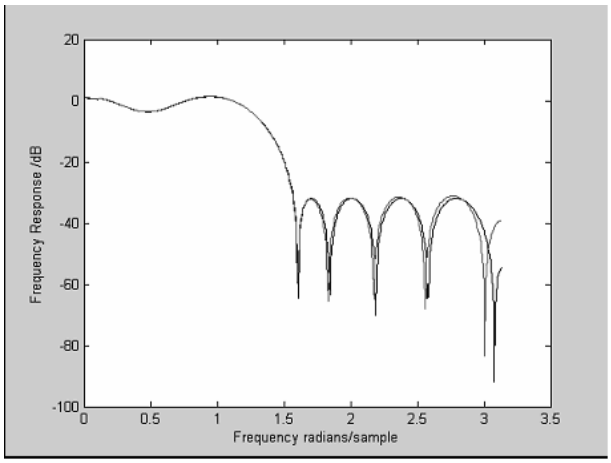


Figure 4.21 Frequency Response (Red: GA Implementation; Black: Remez Implementation for the Filter (Filter Length 15, Wordlength 7 Bit, Stopband Weight 10) in Test Case 5)

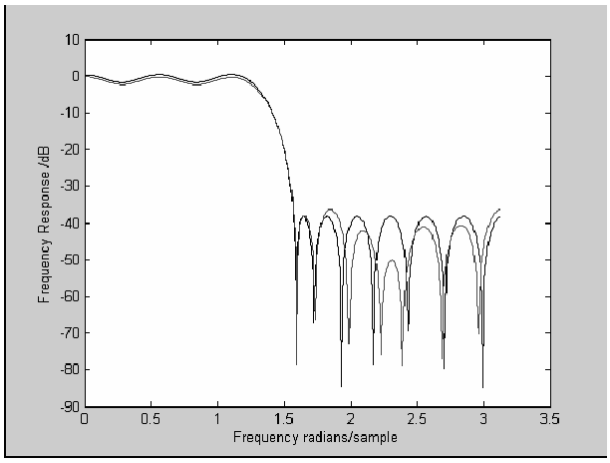


Figure 4.22 Frequency Response (-15.85 dB) (Red: GA Implementation; Black: Remez Implementation for the Filter (Filter Length 25, Wordlength 7 Bit, Stopband Weight 10) in Test Case 5)

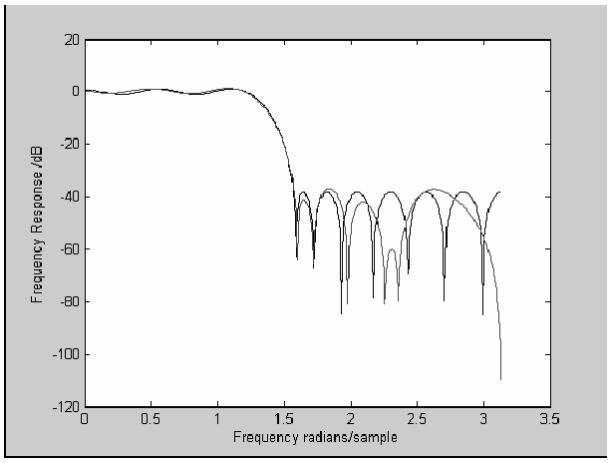


Figure 4.23 Frequency Response (-16.14 dB) (Red: GA Implementation; Black: Remez Implementation for the Filter (Filter Length 25, Wordlength 7 Bit, Stopband Weight 10) in Test Case 5)

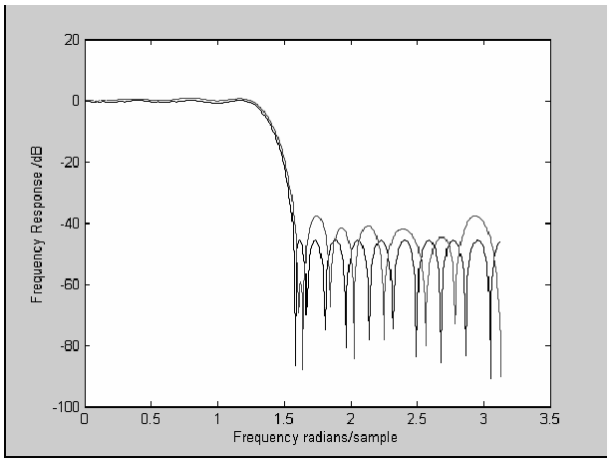


Figure 4.24 Frequency Response (-17.59 dB) (Red: GA Implementation; Black: Remez Implementation for the Filter (Filter Length 35, Wordlength 7 Bit, Stopband Weight 10) in Test Case 5)

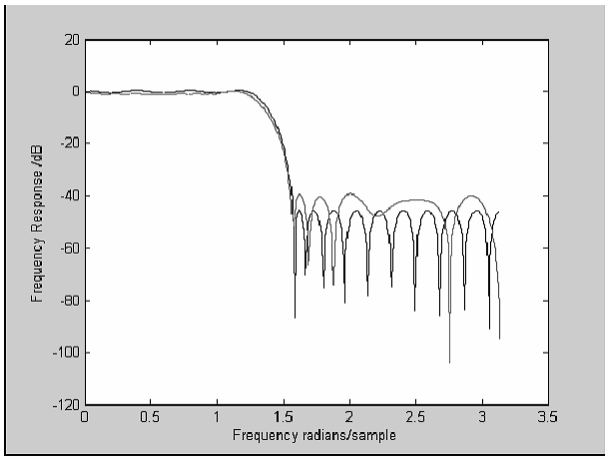


Figure 4.25 Frequency Response (-17.58 dB) (Red: GA Implementation; Black: Remez Implementation for the Filter (Filter Length 35, Wordlength 7 Bit, Stopband Weight 10) in Test Case 5)

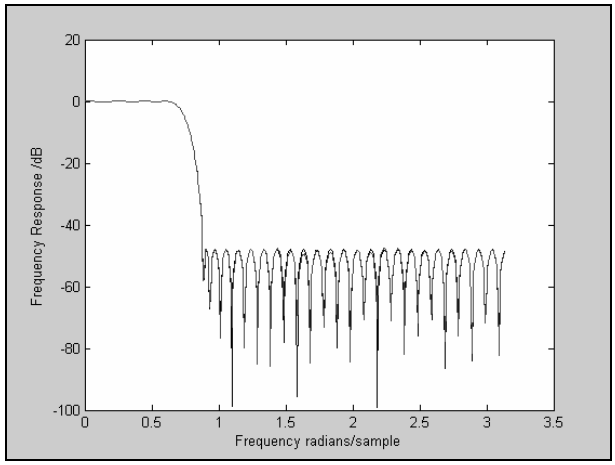


Figure 4.26 Frequency Response (Red: GA Implementation; Black: Remez Implementation for the Filter (Filter Length 63, Wordlength 13 Bit) in Test Case 5)

Test Case 6

The example filter of "Design of Discrete-Coefficient-Value Linear Phase FIR Filters with Optimum Normalized Peak Ripple Magnitude [6]" by Y.C. Lim is used for the test case.

Filter Requirements are as follows:

(a) Symmetrical Impulse Response; Low pass

(b) Passband edges at 0.0 and 0.15 sampling frequency

(c) Stopband edges at 0.25 and 0.50 sampling frequency

(d) Weight in passband and stopband 1.

(e) Worlength 9 bit.

Table 4.9 Summary of Filter Design in Test Case 6

Filter Length N = 35 Infinite Wordlength NPRM = - 61.38 dB				
Filter Design Method	**Quant. Type**	**NPRM (dB)**	**Complexity (CSD) (Adders)**	**Complexity (RAG-n) (Adders)**
MILP [6]	2 SPT	-39.12	12	5
GA	Integer	-48.34	17	8

Table 4.9 summarizes the results for the given filter. The filter designed by GA has a better attenuation than MILP's [6] filter with increased adder cost in terms of RAG-n. The filter designed by GA and implemented with RAG-n uses less number of adders than the one designed by MILP and implemented with CSD. Results show considerable reduction in the number of adders required for implementation when RAG-n is used. Figure 4.27 shows frequency responses of filters designed by GA (red) and Parks-McClellan (black) algorithm.

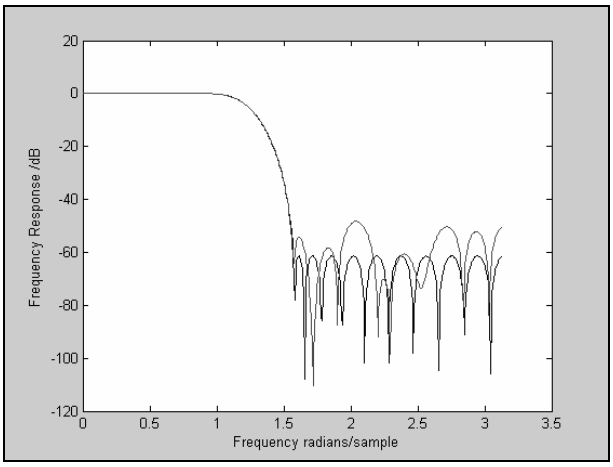


Figure 4.27 Frequency Response (Red: GA Implementation; Black: Remez Implementation for the Filter (in Test Case 6)

Test Case 7

The example filter of "An Improved Polynomial Time Algorithm for Designing Digital Filters with Power-of-Two Coefficients [7]" by C.L. Chen, K.Y. Khoo and A.N. Willson is used for the test case.

Filter Requirements are as follows:

(a) Symmetrical Impulse Response; Low pass

(b) Passband edges at 0.0 and 0.15 sampling frequency

(c) Stopband edges at 0.25 and 0.50 sampling frequency

(d) Weight in passband and stopband 1.

(e) Worlength 9 bit.

Table 4.10 Summary of Filter Design in Test Case 7

Filter Length N = 25 Infinite Wordlength NPRM = -46.03 dB				
Filter Design Method	**Quant. Type**	**NPRM (dB)**	**Complexity (CSD) (Adders)**	**Complexity (RAG-n) (Adders)**
PTA[7]	Free Allocation	-43.97	13	8
GA	Integer	-43.93	13	6
GA	Integer	-44.20	15	9

Table 4.10 summarizes the results for the given filter. The filter designed by GA has slightly better attenuation than PTA's filter with increased adder cost in terms of RAG-n implementation. PTA is a filter design method that uses free allocation for coefficient quantization and expresses filter coefficients as SPT terms to reduce implementation complexity. All the filters designed by GA and implemented by RAG-n uses less number of adders than the filter designed by PTA with CSD implementation. The implementation complexity of PTA with RAG-n is less than the complexity with CSD. Better results are obtained by applying RAG-n to PTA instead of CSD implementation. Figure 4.28-29 show frequency responses of filters designed by GA (red) and Parks-McClellan (black) algorithm

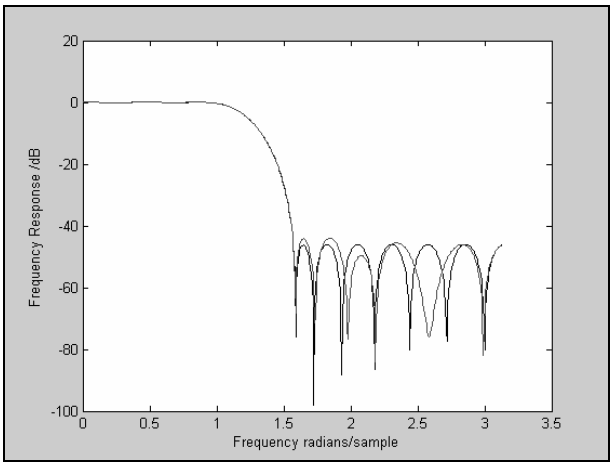


Figure 4.28 Frequency Response (-43.93 dB) (Red: GA Implementation; Black: Remez Implementation for the Filter in Test Case 7)

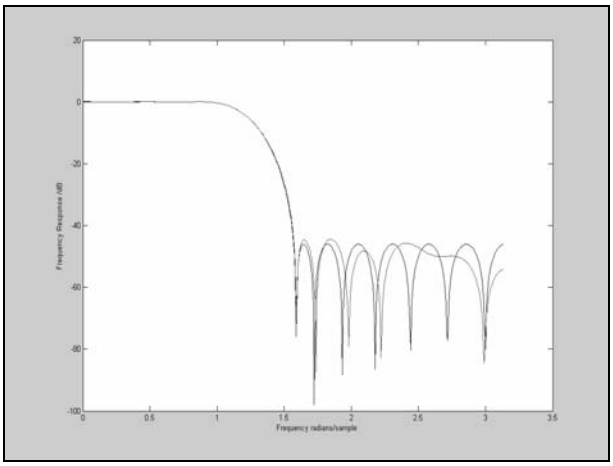


Figure 4.29 Frequency Response (-44.20 dB) (Red: GA Implementation; Black: Remez Implementation for the Filter in Test Case 7)

CHAPTER 5

CONCLUSION

The work in this thesis is composed of two stages namely filter design stage and filter implementation stage. In the filter design stage, a survey has been made on discrete coefficient FIR digital filters design methods that use different quantization techniques for filter coefficients in literature and comparisons have been made with the filter design method based on genetic algorithm that is implemented for this thesis in terms of filter performance. The main focus of the thesis is the filter implementation stage. In this stage, algorithms and filter structures that reduce implementation complexity have been searched, implemented and applied to these filter design methods and comparisons have been made in terms of filter implementation complexity of architectures based on the use of CSD and RAG-n.

Mixed integer linear programming [3], [6], [11], [16], simulated annealing [5], [14], and [23], genetic algorithms [2], [15], [17], [19], [21], and [24], local search [1], [18] and [25], and free allocation algorithms [4], [7], [20], and [22] have been used for the design of discrete coefficient FIR digital filters in this thesis.

As a conclusion for filter design methods, test results have showed that there is no best method for discrete coefficient filter design. Filter designer should use all the methods if possible in the design process.

Implementation complexity of the filters, that is number of adders required to implement filter coefficients, is as important as filter design. Some of the algorithms mentioned above have focused on designing filters with coefficients expressible as SPT terms in order to reduce this complexity and but it can be said that filter implementation complexity is not taken into consideration completely. In this thesis, implementation complexity is also considered and these design methods have been compared. SPT or CSD and RAG-n algorithms have been used for the realization of the filter coefficients since they reduce the required number of adders. RAG-n algorithm has been used in filter design methods that limit the filter coefficients to SPT terms [1], [4]-[7], [11], [14], [16], [22]. For RAG-n algorithm, transposed form

FIR filter structure has been used since it allows multiple uses of the repeated and producible coefficients. Test results have showed that filter implementations that use RAG-n have used nearly %50 adders less than a CSD design even if filter is designed with coefficients expressible as SPT terms since RAG-n algorithm exploits the redundancy in the filter coefficient set by use of the transposed form filter structure. It can be concluded that constraining the filter coefficients to SPT terms does not produce less complex implementations. Instead filter coefficients can be designed using integer values instead of SPT terms, other algorithms can be used to reduce the implementation complexity.

As a future work, the improvement of the above algorithms with the inclusion of implementation complexity in the cost function in addition to filter performance can be considered so that optimization of both performance and implementation complexity can be achieved.

REFERENCES

[1] H. Samueli, "An Improved Search Algorithm for the Design of Multiplierless FIR filters with Powers-of-Two Coefficients," IEEE Trans. Circuits Syst., vol. 36, pp. 1044-1047, July 1989.

[2] D. J. Xu and M. L. Daley, "Design of Optimal Paralel Filter Using a Parallel Genetic Algorithm," IEEE Trans. Circuits Syst., vol. 42, pp. 673-675, October 1995.

[3] D.M. Kodek, "Design of Optimal Finite Wordlength FIR Digital Filters Using Integer Programming Techniques," IEEE Trans. Acoustics, Speech, Signal Processing, vol. ASSP-28, pp. 304-308, June 1980.

[4] C. L. Chen and A. N. Willson, "A Trellis Search Algorithm for the Design of FIR filters with Signed-Powers-of-Two Coefficients," IEEE Trans. Circuits Syst., vol. 46, pp. 29-39, January 1999.

[5] T. Çiloğlu and Z. Ünver, "A new approach to Discrete Coefficient FIR Digital Filter Design by Simulated Annealing," Proc. Of IEEE Int. Con. On Acoustics, Speech and Sig. Proc., Minneapolis, pp.101-104, 1993.

[6] Y. C. Lim and S. R. Parker, "Design of Discrete-Coefficient-Value Linear Phase FIR Filters with Optimum Normalized Peak Ripple Magnitude," IEEE Trans. Circuits Syst., vol. 37, pp. 1480-1486, Dec. 1990.

[7] C. L. Chen, K. Y. Khoo and A.N. Willson Jr., "An Improved Polynomial Time Algorithm for Designing Digital Filters with Power-of-Two Coefficients," in Proc. IEEE Int. Symp. Circuits Systems, Seattle, WA, May 1995, vol. 1, pp. 223-226.

[8] A.G. Dempster and M.D. Macleod, "Use Of Minimum-Adder Multiplier Blocks In FIR Digital Filters," IEEE Trans. Circuits Syst., vol. 42, pp. 569-577, Sep. 1995.

[9] A.G. Dempster and M.D. Macleod, "Constant Integer Multiplication Using Minimum Adders," IEE Proc. –Circuits, Devices, Syts., vol. 141, pp. 407-413, Oct.1994.

[10] T. Çiloğlu, "Normalized Peak Ripple Magnitude as an Objective Function in Discrete Coefficient FIR Filter Design," IEEE Midwest Symposium on Circuits and Systems, August 2001, Dayton, Ohio, USA.

[11] Y. C. Lim and S. R. Parker, "FIR Filter Design over A Discrete Powers-Of-Two Coefficient Space," IEEE Trans. Acoust., Speech, Signal Processing, vol. ASSP-31, pp. 583-591, June 1983.

[12] D. R. Bull and D. H. Horrocks, "Primitive Operator Digital Filters," IEE Proc. G, vol. 138, pp. 401-412, June 1991.

[13] D.M. Kodek and K. Steiglitz, "Comparison of Optimal and Local Search Methods for Designing Finite Wordlength FIR Digital Filters," IEEE Trans. Circuits Syst., vol. 28, pp. 28-32, Jan. 1981.

[14] N. Benvenuto, M. Marchesi, and A. Uncini, "Applications of Simulated Annealing for the Design of Special Digital Filters," IEEE Trans. Signal Process. vol. 40, pp. 323-332, Feb. 1992.

[15] A. Lee, M. Ahmadi, G.A. Jullien, and R. S. Lashkari, W.C. Miller "Design of 1-D FIR Filters with Genetic Algorithms," Proceedings of 1999 IEEE Int. Symp. On Circuits and Systems, vol. 3, Orlando, USA, pp. 295-298, 1999.

[16] Y. C. Lim and S. R. Parker, "Discrete Coefficient FIR Digital Filter Design Based Upon an LMS Criteria," IEEE Trans. Circuits Syst., vol. CAS-30, pp. 723-739, Oct. 1983.

[17] G. Wade, A. Roberts and G. Williams, "Multiplier-less FIR filter design using a genetic algorithm," IEE Proc., Vision, Image Signal Process., 81, pp.175-180, 141 (3) June 1994.

[18] Q. Zhao and Y. Tadokoro, "A Simple Design of FIR Filters with Power-of-Two Coefficients," IEEE Trans. Circuits Syst., vol. 35, pp. 566-570, May. 1988.

[19] D.W. Redmill and D.R. Bull, "Automated Design of Low Complexity FIR Filters," Proceedings of 1998 IEEE Int. Symp. On Circuits and Systems, 1998.

[20] T. Çiloğlu, "Design of FIR Filters for low implementation complexity," Electronics Letters, vol.35, No.7, pp.529-530, April 1999.

[21] D.W. Redmill, D.R. Bull, and E. Dagless, "Genetic synthesis of reduced complexity filters and filter banks using primitive operator directed graphs,"

IEE Circuits, Devices and Systems, IEE Proceedings, Volume: 147 , Issue: 5 , pp.303 – 310, Oct. 2000.

[22] D. Li, Y. C. Lim, Y. Lian, and J. Song, "A Polynomial-Time Algorithm for Designing FIR Filters with Power-of-Two Coefficients," Signal Processing, IEEE Transactions on Acoustics, Speech, and Signal Processing, IEEE Transactions on] , Vol. 50 , Issue: 8, pp.1935 – 1941, Aug. 2002

[23] N. Benvenuto and M. Marchesi, "Digital Filters Design by Simulated Annealing," IEEE Transactions on Circuits and Systems, Vol. 36, No. 3, pp. 459-460, March 1989.

[24] D. Suckley, "Genetic Algorithm in the design of FIR filters," IEE Proc. G., Vol. 138, pp. 234-238, April 1991.

[25] T. Çiloğlu, "An efficient local search method guided by gradient information for discrete coefficient filter design", Signal Processing, vol. 82, no.10, pp. 1337-1350, Oct. 2002.

APPENDIX

FILTER COEFFICIENTS FOR TEST CASES

Test Case 1

Filter (1) Coefficients for H. Samueli [1]:

$h(0) = 2$, $h(1) = 6$, $h(2) = -2$, $h(3) = -16$, $h(4) = -14$, $h(5) = 20$, $h(6) = 40$,
$h(7) = -2$, $h(8) = -80$, $h(9) = -68$, $h(10) = 112$, $h(11) = 368$, $h(12) = 492$

Filter (1) Coefficients for GA (43.74 dB):

$h(0) = 1$, $h(1) = 4$, $h(2) = 0$, $h(3) = -8$, $h(4) = -7$, $h(5) = 10$, $h(6) = 21$, $h(7) = 0$,
$h(8) = -39$, $h(9) = -35$, $h(10) = 55$, $h(11) = 184$, $h(12) = 245$

Filter (1) Coefficients for GA (43.93 dB):

$h(0) = 1$, $h(1) = 4$, $h(2) = 0$, $h(3) = -8$, $h(4) = -7$, $h(5) = 10$, $h(6) = 21$, $h(7) = 0$,
$h(8) = -40$, $h(9) = -35$, $h(10) = 56$, $h(11) = 186$, $h(12) = 249$

Filter (2) Coefficients for H. Samueli [1]:

$h(0) = 62$, $h(1) = 56$, $h(2) = 58$, $h(3) = 44$, $h(4) = 16$, $h(5) = -34$, $h(6) = -118$,
$h(7) = -232$, $h(8) = -376$, $h(9) = -536$, $h(10) = -704$, $h(11) = -864$,
$h(12) = -1000$, $h(13) = -1064$, $h(14) = -1058$, $h(15) = -928$, $h(16) = -672$,
$h(17) = -258$, $h(18) = 316$, $h(19) = 1052$, $h(20) = 1928$, $h(21) = 2944$,
$h(22) = 4016$, $h(23) = 5152$, $h(24) = 6272$, $h(25) = 7296$, $h(26) = 8220$,
$h(27) = 8956$, $h(28) = 9474$, $h(29) = 9736$

Filter (2) Coefficients for GA:

$h(0) = 22$, $h(1) = 21$, $h(2) = 28$, $h(3) = 33$, $h(4) = 36$, $h(5) = 34$, $h(6) = 26$,
$h(7) = 12$, $h(8) = -10$, $h(9) = -39$, $h(10) = -73$, $h(11) = -110$, $h(12) = -145$,
$h(13) = -174$, $h(14) = -193$, $h(15) = -194$, $h(16) = -174$, $h(17) = -128$,
$h(18) = -53$, $h(19) = -51$, $h(20) = 183$, $h(21) = 339$, $h(22) = 514$, $h(23) = 699$,
$h(24) = 884$, $h(25) = 1060$, $h(26) = 1216$, $h(27) = 1343$, $h(28) = 1432$,
$h(29) = 1478$

Test Case 2

Filter Coefficients for D.J. Xu and M. L. Daley [2]:

$h(0) = 2,\ h(1) = 3,\ h(2) = -1,\ h(3) = -4,\ h(4) = 1,\ h(5) = 6,\ h(6) = 1,\ h(7) = -7,$
$h(8) = -5,\ h(9) = 9, h(10) = 10,\ h(11) = -8,\ h(12) = -16,\ h(13) = 5,\ h(14) = 27,$
$h(15) = 2,\ h(16) = -44,\ h(17) = -24,\ h(18) = 92,\ h(19) = 211$

Filter Coefficients for GA (-36.81dB):

$h(0) = 1,\ h(1) = 4,\ h(2) = -2,\ h(3) = -4,\ h(4) = 0,\ h(5) = 6,\ h(6) = 2,\ h(7) = -7,$
$h(8) = -5,\ h(9) = 8, h(10) = 10,\ h(11) = -8,\ h(12) = -17,\ h(13) = 5,\ h(14) = 27,$
$h(15) = 3,\ h(16) = -44,\ h(17) = -25,\ h(18) = 92,\ h(19) = 212$

Filter Coefficients for GA (-37.82dB):

$h(0) = 3,\ h(1) = 8,\ h(2) = -4,\ h(3) = -7,\ h(4) = 1,\ h(5) = 11,\ h(6) = 3,\ h(7) = -15,$
$h(8) = -10,\ h(9) = 17,\ h(10) = 20,\ h(11) = -16,\ h(12) = -34,\ h(13) = 10,$
$h(14) = 54,\ h(15) = 5,\ h(16) = -87,\ h(17) = -49,\ h(18) = 184,\ h(19) = 423$

Filter Coefficients for GA (-38.58dB):

$h(0) = 3,\ h(1) = 9,\ h(2) = -4,\ h(3) = -9,\ h(4) = 1,\ h(5) = 13,\ h(6) = 4,$
$h(7) = -17,\ h(8) = -11,\ h(9) = 20,\ h(10) = 23,\ h(11) = -19,\ h(12) = -40,$
$h(13) = 12,\ h(14) = 64,\ h(15) = 6,\ h(16) = -103,\ h(17) = -58,\ h(18) = 217,$
$h(19) = 499$

Test Case 3

Filter (Filter Length 21, Wordlength 7 Bit) Coefficients for Kodek [3]:

$h(0) = 2,\ h(1) = 0,\ h(2) = -2,\ h(3) = -1,\ h(4) = 2,\ h(5) = 3,\ h(6) = -3,\ h(7) = -6,$
$h(8) = 3,\ h(9) = 20,\ h(10) = 28$

Filter (Filter Length 21, Wordlength 7 Bit) Coefficients for GA (-23.86 dB):

$h(0) = 2,\ h(1) = 0,\ h(2) = -2,\ h(3) = -1,\ h(4) = 2,\ h(5) = 3,\ h(6) = -3,\ h(7) = -6,$
$h(8) = 3,\ h(9) = 20, h(10) = 28$

Filter (Filter Length 21, Wordlength 7 Bit) Coefficients for GA (-24.06 dB):

$h(0) = 5,\ h(1) = 0,\ h(2) = -4,\ h(3) = -2,\ h(4) = 5,\ h(5) = 5,\ h(6) = -6,$
$h(7) = -12,\ h(8) = 6,\ h(9) = 40,\ h(10) = 57$

Filter (Filter Length 21, Wordlength 7 Bit) Coefficients for GA (-24.38 dB):

$h(0)=6,\quad h(1)=0,\quad h(2)=-5,\quad h(3)=-3,\quad h(4)=6,\quad h(5)=7,\quad h(6)=-7,$
$h(7)=-15,\ h(8)=8,\ h(9)=52,\ h(10)=74$

Filter (Filter Length 40, Wordlength 10 Bit) Coefficients for Kodek [3]:

$h(0)=2,\ h(1)=2,\ h(2)=-1,\ h(3)=-4,\ h(4)=0,\ h(5)=6,\ h(6)=1,\ h(7)=-8,$
$h(8)=-5,\ h(9)=8,h(10)=10,\ h(11)=-9,\ h(12)=-17,\ h(13)=5,\ h(14)=27,$
$h(15)=2,\ h(16)=-43,\ h(17)=-25,\ h(18)=92,\ h(19)=211$

Filter Coefficients for GA (-36.81dB):

$h(0)=1,\ h(1)=4,\ h(2)=-2,\ h(3)=-4,\ h(4)=0,\ h(5)=6,\ h(6)=2,\ h(7)=-7,$
$h(8)=-5,\ h(9)=8,h(10)=10,\ h(11)=-8,\ h(12)=-17,\ h(13)=5,\ h(14)=27,$
$h(15)=3,\ h(16)=-44,\ h(17)=-25,\ h(18)=92,\ h(19)=212$

Filter Coefficients for GA (-37.82dB):

$h(0)=3,\ h(1)=8,\ h(2)=-4,\ h(3)=-7,\ h(4)=1,\ h(5)=11,\ h(6)=3,\ h(7)=-15,$
$h(8)=-10,\quad h(9)=17,\quad h(10)=20,\quad h(11)=-16,\quad h(12)=-34,\quad h(13)=10,$
$h(14)=54,\ h(15)=5,\ h(16)=-87,\ h(17)=-49,\ h(18)=184,\ h(19)=423$

Filter Coefficients for GA (-38.58dB):

$h(0)=3,\quad h(1)=9,\quad h(2)=-4,\quad h(3)=-9,\quad h(4)=1,\quad h(5)=13,\quad h(6)=4,$
$h(7)=-17,\quad h(8)=-11,\quad h(9)=20,\quad h(10)=23,\quad h(11)=-19,\quad h(12)=-40,$
$h(13)=12,\quad h(14)=64,\quad h(15)=6,\quad h(16)=-103,\quad h(17)=-58,\quad h(18)=217,$
$h(19)=499$

Test Case 4

Filter Coefficients for Trellis Search:

$h(0)=-14,\ h(1)=0,\ h(2)=36,\ h(3)=32,\ h(4)=-42,\ h(5)=-97,\ h(6)=0,$
$h(7)=172,\ h(8)=144,\ h(9)=-192,\ h(10)=-432,\ h(11)=-1,\ h(12)=1104,$
$h(13)=2088$

Filter Coefficients for GA:

$h(0)=-12,\ h(1)=1,\ h(2)=29,\ h(3)=27,\ h(4)=-34,\ h(5)=-78,\ h(6)=-1,$
$h(7)=140,\ h(8)=117,\ h(9)=-154,\ h(10)=-350,\ h(11)=-2,\ h(12)=895,$
$h(13)=1693$

Test Case 5

Filter (Filter Length 15, Wordlength 5 Bit, Stopband Weight 1) Coefficients for SA [5]:

$h(0) = -1,\quad h(1) = 3,\quad h(2) = 2,\quad h(3) = -2,\quad h(4) = -5,\quad h(5) = 2,\quad h(6) = 14,$
$h(7) = 20$

Filter (Filter Length 15, Wordlength 5 Bit, Stopband Weight 1) Coefficients for GA (-16.66 dB):

$h(0) = -1,\ h(1) = 2,\ h(2) = 2,\ h(3) = -1,\ h(4) = -3,\ h(5) = 2,\ h(6) = 10,\ h(7) = 15$

Filter (Filter Length 15, Wordlength 5 Bit, Stopband Weight 1) Coefficients for GA (-17.80 dB):

$h(0) = -1,\ h(1) = 3,\ h(2) = 2,\ h(3) = -2,\ h(4) = -4,\ h(5) = 2,\ h(6) = 13,\ h(7) = 19$

Filter (Filter Length 25, Wordlength 5 Bit, Stopband Weight 1) Coefficients for SA [5]:

$h(0) = -1,\quad h(1) = 0,\ h(2) = 1,\ h(3) = 0,\ h(4) = -1,\ h(5) = -1,\ h(6) = 2,\ h(7) = 2,$
$h(8) = -2,\ h(9) = -4,\ h(10) = 2,\ h(11) = 14,\ h(12) = 20$

Filter (Filter Length 25, Wordlength 5 Bit, Stopband Weight 1) Coefficients for GA:

$h(0) = -1,\ h(1) = 0,\ h(2) = 1,\ h(3) = 0,\ h(4) = -1,\ h(5) = -1,\ h(6) = 2,\ h(7) = 2,$
$h(8) = -2,\ h(9) = -4,\ h(10) = 2,\ h(11) = 14,\ h(12) = 20$

Filter (Filter Length 15, Wordlength 7 Bit, Stopband Weight 10) Coefficients for SA [5]:

$h(0) = 6,\quad h(1) = 10,\quad h(2) = 4,\quad h(3) = -10,\quad h(4) = -14,\quad h(5) = 6,\quad h(6) = 40,$
$h(7) = 56$

Filter (Filter Length 15, Wordlength 7 Bit, Stopband Weight 10) Coefficients for GA:

$h(0) = 9,\quad h(1) = 15,\quad h(2) = 8,\quad h(3) = -9,\quad h(4) = -15,\quad h(5) = 10,\quad h(6) = 51,$
$h(7) = 72$

Filter (Filter Length 25, Wordlength 7 Bit, Stopband Weight 10) Coefficients for SA [5]:

$h(0) = 1,\ h(1) = 3,\ h(2) = 2,\ h(3) = -3,\ h(4) = -6,\ h(5) = -2,\ h(6) = 5,\ h(7) = 3,$
$h(8) = -8,\ h(9) = -12,\ h(10) = 7,\ h(11) = 40,\ h(12) = 56$

Filter (Filter Length 25, Wordlength 7 Bit, Stopband Weight 10) Coefficients for GA (-15.85 dB):

$h(0)=1,\ h(1)=4,\ h(2)=4,\ h(3)=0,\ h(4)=-4,\ h(5)=-1,\ h(6)=5,\ h(7)=4,$
$h(8)=-6,\ h(9)=-10,\ h(10)=7,\ h(11)=37,\ h(12)=52$

Filter (Filter Length 25, Wordlength 7 Bit, Stopband Weight 10) Coefficients for GA (-16.14 dB):

$h(0)=1,\ h(1)=3,\ h(2)=2,\ h(3)=-3,\ h(4)=-6,\ h(5)=-2,\ h(6)=5,\ h(7)=3,$
$h(8)=-8,\ h(9)=-12,\ h(10)=7,\ h(11)=40,\ h(12)=56$

Filter (Filter Length 35, Wordlength 7 Bit, Stopband Weight 10) Coefficients for SA [5]:

$h(0)=0,\ h(1)=0,\ h(2)=0,\ h(3)=-1,\ h(4)=-2,\ h(5)=-1,\ h(6)=2,\ h(7)=2,$
$h(8)=-2,\ h(9)=-5,\ h(10)=-1,\ h(11)=6,\ h(12)=5,\ h(13)=-5,\ h(14)=-10,$
$h(15)=6,\ h(16)=34,\ h(17)=48$

Filter (Filter Length 35, Wordlength 7 Bit, Stopband Weight 10) Coefficients for GA (-17.59 dB):

$h(0)=0,\ h(1)=1,\ h(2)=2,\ h(3)=1,\ h(4)=-2,\ h(5)=-2,\ h(6)=1,\ h(7)=3,$
$h(8)=0,\ h(9)=-4,\ h(10)=-1,\ h(11)=6,\ h(12)=5,\ h(13)=-7,\ h(14)=-12,$
$h(15)=8,\ h(16)=44,\ h(17)=62$

Filter (Filter Length 35, Wordlength 7 Bit, Stopband Weight 10) Coefficients for GA (-17.58 dB):

$h(0)=0,\ h(1)=0,\ h(2)=0,\ h(3)=-1,\ h(4)=-2,\ h(5)=-1,\ h(6)=2,\ h(7)=2,$
$h(8)=-2,\ h(9)=-5,\ h(10)=-1,\ h(11)=6,\ h(12)=5,\ h(13)=-5,\ h(14)=-10,$
$h(15)=6,\ h(16)=34,\ h(17)=48$

Filter (2) Coefficients for SA [5]:

$h(0)=-6,\ h(1)=-4,\ h(2)=1,\ h(3)=7,\ h(4)=8,\ h(5)=10,\ h(6)=0,\ h(7)=-9,$
$h(8)=-24,\ h(9)=-20,\ h(10)=-4,\ h(11)=24,\ h(12)=40,\ h(13)=40,\ h(14)=9,$
$h(15)=-28,\ h(16)=-68,\ h(17)=-72,\ h(18)=-30,\ h(19)=34,\ h(20)=96,$
$h(21)=120,\ h(22)=72,\ h(23)=-36,\ h(24)=-160,\ h(25)=-240,\ h(26)=-160,$
$h(27)=48,\ h(28)=384,\ h(29)=768,\ h(30)=1040,\ h(31)=1152$

Filter (2) Coefficients for GA:

$h(0) = -23$, $h(1) = -7$, $h(2) = 6$, $h(3) = 22$, $h(4) = 32$, $h(5) = 26$, $h(6) = 2$, $h(7) = -30$, $h(8) = -52$, $h(9) = -46$, $h(10) = -8$, $h(11) = 46$, $h(12) = 87$, $h(13) = 85$, $h(14) = 29$, $h(15) = -60$, $h(16) = -135$, $h(17) = -145$, $h(18) = -67$, $h(19) = 75$, $h(20) = 209$, $h(21) = 251$, $h(22) = 146$, $h(23) = -86$, $h(24) = -345$, $h(25) = -479$, $h(26) = -350$, $h(27) = 94$, $h(28) = 783$, $h(29) = 1535$, $h(30) = 2116$, $h(31) = 2334$

Test Case 6

Filter Coefficients for MILP [6] (-33.09 dB):

$h(0) = -1$, $h(1) = 2$, $h(2) = 3$, $h(3) = 0$, $h(4) = -6$, $h(5) = -4$, $h(6) = 8$, $h(7) = 10$, $h(8) = -5$, $h(9) = -18$, $h(10) = -2$, $h(11) = 24$, $h(12) = 18$, $h(13) = -28$, $h(14) = -48$, $h(15) = 32$, $h(16) = 160$, $h(17) = 224$

Filter Coefficients for MILP [6] (-39.09 dB):

$h(0) = 1$, $h(1) = 1$, $h(2) = -1$, $h(3) = -3$, $h(4) = 0$, $h(5) = 4$, $h(6) = 4$, $h(7) = -3$, $h(8) = -9$, $h(9) = -3$, $h(10) = 12$, $h(11) = 14$, $h(12) = -6$, $h(13) = -28$, $h(14) = -17$, $h(15) = 40$, $h(16) = 112$, $h(17) = 144$

Filter Coefficients for GA:

$h(0) = 0$, $h(1) = 1$, $h(2) = 0$, $h(3) = -2$, $h(4) = -2$, $h(5) = 3$, $h(6) = 6$, $h(7) = 0$, $h(8) = -11$, $h(9) = -9$, $h(10) = 12$, $h(11) = 25$, $h(12) = 0$, $h(13) = -44$, $h(14) = -38$, $h(15) = 60$, $h(16) = 198$, $h(17) = 264$

Test Case 7

Filter Coefficients for PTA:

$h(0) = 2$, $h(1) = 8$, $h(2) = 0$, $h(3) = -18$, $h(4) = -14$, $h(5) = 20$, $h(6) = 44$, $h(7) = 0$, $h(8) = -82$, $h(9) = -72$, $h(10) = 116$, $h(11) = 386$, $h(12) = 514$

Filter Coefficients for GA (-43.93dB):

$h(0) = 1$, $h(1) = 4$, $h(2) = 0$, $h(3) = -8$, $h(4) = -7$, $h(5) = 10$, $h(6) = 21$, $h(7) = 0$, $h(8) = -40$, $h(9) = -35$, $h(10) = 56$, $h(11) = 186$, $h(12) = 249$

Filter Coefficients for GA (-44.20dB):

$h(0) = 1$, $h(1) = 3$, $h(2) = 0$, $h(3) = -7$, $h(4) = -6$, $h(5) = 8$, $h(6) = 17$, $h(7) = 0$, $h(8) = -33$, $h(9) = -29$, $h(10) = 46$, $h(11) = 153$, $h(12) = 205$