Design of IIR filters with canonical signed-digit (CSD) coefficients using genetic algorithms.

Design of IIR Filters with
Canonical Signed-Digit (CSD) Coefficients
Using Genetic Algorithms

by

Li Liang

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

Windsor, Ontario, Canada
2003

©Copyright Li Liang 2003
The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

L'auteur a accordé une licence non-exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

L'auteur conserve la propriété du droit d'auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.
ABSTRACT

In this thesis, an optimization method is used in the design of 1-D IIR digital filters, doubly complementary filter pairs and 2-D IIR digital filters. This method uses genetic algorithm (GA) to minimize the mean square error between the desired and the designed filter responses in order to calculate the coefficients of the filter’s transfer function.

The 1-D IIR filters are designed using cascade structure, and constraints are imposed on the coefficients of the denominator polynomials to ensure stability. In the one-dimensional case, the method is also used to design doubly complementary filter pairs, in which case two filters that are both all-pass complementary and power complementary are designed at the cost of one filter. Based on the same stability criterion, an optimization method is presented for the design of 2-D IIR digital filters with non-separable numerator and separable denominator transfer functions.

The advantage of the proposed method is that it produces filters with Canonical Signed-Digit (CSD) coefficients, which not only eliminates the quantization process in digital filter design but also make the designed filter more efficient for high speed DSP applications. Design examples of 1-D IIR filters, doubly complementary filter pairs and 2-D IIR filters are provided in order to demonstrate the usefulness of the presented method.
To My Husband
ACKNOWLEDGEMENTS

I wish to express my sincere appreciation to Dr. M. Ahmadi. His patience, encouragement and innovative ideas enable me to successfully complete this thesis. His full support of my graduate study, his guidance over the course of this research and his appreciation of my work make my school life meaningful at the University of Windsor. I would also like to give my sincere appreciation to Dr. M.A.Sid-Ahmed. His sincere support and encouragement enable me to successfully finish this Master program. His invaluable recommendations and suggestions have broaden my eyes and been a great help of my work and this thesis. I would also like to thank my committee members, Dr. B.B.Budkowska and Dr. S. Erfani for their invaluable comments and evaluations.

At last I wish to extend my sincere thanks to my parent and my husband for their support through my academic career. They are always there to support me, give me the best they can give and encourage me out of difficulties. Without their moral support, this work would not be successfully completed.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABSTRACT</td>
<td>iii</td>
</tr>
<tr>
<td>DEDICATION</td>
<td>iv</td>
</tr>
<tr>
<td>ACKNOWLEDGEMENTS</td>
<td>v</td>
</tr>
<tr>
<td>LIST OF TABLES</td>
<td>ix</td>
</tr>
<tr>
<td>LIST OF FIGURES</td>
<td>x</td>
</tr>
<tr>
<td>CHAPTER 1. INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>1.1 Introduction</td>
<td>1</td>
</tr>
<tr>
<td>1.2 Classification and Characterization of Digital Filters</td>
<td>2</td>
</tr>
<tr>
<td>1.2.1 Difference Equation</td>
<td>2</td>
</tr>
<tr>
<td>1.2.1.1 One-Dimensional Filters</td>
<td>2</td>
</tr>
<tr>
<td>1.2.1.2 Two-Dimensional Filters</td>
<td>3</td>
</tr>
<tr>
<td>1.2.2 Transfer Function</td>
<td>3</td>
</tr>
<tr>
<td>1.2.2.1 One-Dimensional Filters</td>
<td>3</td>
</tr>
<tr>
<td>1.2.2.2 Two-Dimensional Filters</td>
<td>4</td>
</tr>
<tr>
<td>1.2.3 Magnitude and Phase Responses</td>
<td>6</td>
</tr>
<tr>
<td>1.2.3.1 One-Dimensional Filters</td>
<td>6</td>
</tr>
<tr>
<td>1.2.3.2 Two-Dimensional Filters</td>
<td>7</td>
</tr>
<tr>
<td>1.2.4 Stability</td>
<td>8</td>
</tr>
<tr>
<td>1.2.4.1 One-Dimensional Filters</td>
<td>8</td>
</tr>
<tr>
<td>1.2.4.2 Two-Dimensional Filters</td>
<td>9</td>
</tr>
<tr>
<td>1.2.5 Some Comparisons of IIR and FIR Digital Filters</td>
<td>10</td>
</tr>
<tr>
<td>1.3 Summary of Previous Work</td>
<td>10</td>
</tr>
<tr>
<td>1.4 Organization of Thesis</td>
<td>13</td>
</tr>
<tr>
<td>CHAPTER 2. GENETIC ALGORITHM</td>
<td>15</td>
</tr>
<tr>
<td>2.1 Introduction</td>
<td>15</td>
</tr>
<tr>
<td>2.2 Genetic Algorithm Cycle</td>
<td>15</td>
</tr>
<tr>
<td>2.2.1 Encoding Scheme</td>
<td>17</td>
</tr>
<tr>
<td>2.2.2 Reproduction</td>
<td>17</td>
</tr>
<tr>
<td>2.2.3 Crossover</td>
<td>19</td>
</tr>
<tr>
<td>2.2.4 Mutation</td>
<td>20</td>
</tr>
<tr>
<td>2.2.5 Fitness Evaluation</td>
<td>21</td>
</tr>
<tr>
<td>2.2.6 Replacement Strategy</td>
<td>21</td>
</tr>
</tbody>
</table>
2.3 Schema Theorem
   2.3.1 Schema Properties
   2.3.2 Effect of Reproduction
   2.3.3 Effect of Crossover
   2.3.4 Effect of Mutation
   2.3.5 Schema Growth Equation

2.4 An Example

CHAPTER 3. CANONICAL SIGNED-DIGIT (CSD) COEFFICIENTS

3.1 Representation of Filter Coefficients as CSD Number
   3.1.1 Canonical Signed-Digit Representation
   3.1.2 Advantage of CSD Representation

3.2 Problems of Genetic Operators with CSD Coefficients

3.3 Strategies of Previous Literature

3.4 New CSD Coefficients Restoration Technique
   3.4.1 Pseudo Code of the Restoration Technique
   3.4.2 Explanation
   3.4.3 Test Result of the Restoration Technique

CHAPTER 4. DESIGN OF 1-D IIR DIGITAL FILTERS

4.1 IIR Filter Structures

4.2 Cascade Structure

4.3 Stability Consideration

4.4 Design Problem Formulation

4.5 Modified Genetic Algorithm

4.6 Design Examples
   4.6.1 Design of 1-D IIR Filters without Constant Group Delay Characteristic
      4.6.1.1 Lowpass IIR Filters
      4.6.1.2 Bandpass IIR Filters
      4.6.1.3 Highpass IIR Filters
   4.6.2 Design of 1-D IIR Filters with Constant Group Delay Characteristic

CHAPTER 5. DESIGN OF DOUBLY COMPLEMENTARY FILTER PAIRS

5.1 Allpass Transfer Function and Its Properties
   5.1.1 Allpass Transfer Function
   5.1.2 Properties

5.2 Doubly Complementary Filter Pair
   5.2.1 Allpass Complementary and Power Complementary
# LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table 2.1</td>
<td>Subtotal of chromosomes' fitness</td>
<td>29</td>
</tr>
<tr>
<td>Table 2.3</td>
<td>Mutation Operation</td>
<td>30</td>
</tr>
<tr>
<td>Table 3.1</td>
<td>Test result of the conversion technique</td>
<td>42</td>
</tr>
<tr>
<td>Table 4.1</td>
<td>CSD coefficients of the 6th order lowpass IIR filter</td>
<td>54</td>
</tr>
<tr>
<td>Table 4.2</td>
<td>CSD coefficients of the 9th order lowpass IIR filter</td>
<td>57</td>
</tr>
<tr>
<td>Table 4.3</td>
<td>CSD coefficients of the 8th order bandpass IIR filter</td>
<td>61</td>
</tr>
<tr>
<td>Table 4.4</td>
<td>CSD coefficients of the 6th order bandpass IIR filter</td>
<td>65</td>
</tr>
<tr>
<td>Table 4.5</td>
<td>CSD coefficients of the 5th order highpass IIR filter</td>
<td>68</td>
</tr>
<tr>
<td>Table 4.6</td>
<td>CSD coefficients of the 7th order highpass IIR filter</td>
<td>72</td>
</tr>
<tr>
<td>Table 4.7</td>
<td>CSD coefficients of the 4th order lowpass IIR filter</td>
<td>76</td>
</tr>
<tr>
<td>Table 4.8</td>
<td>CSD coefficients of the 8th order bandpass IIR filter</td>
<td>81</td>
</tr>
<tr>
<td>Table 5.1</td>
<td>CSD coefficients of the two allpass filters</td>
<td>96</td>
</tr>
<tr>
<td>Table 5.2</td>
<td>CSD coefficients of the two allpass filters</td>
<td>102</td>
</tr>
<tr>
<td>Table 6.1</td>
<td>CSD coefficients of the 2nd order 2-D filter</td>
<td>111</td>
</tr>
<tr>
<td>Table 6.2</td>
<td>CSD coefficients of the 4th order 2-D filter</td>
<td>112</td>
</tr>
<tr>
<td>Fig. 2.1</td>
<td>GA cycle</td>
<td>Page 16</td>
</tr>
<tr>
<td>--------</td>
<td>-----------</td>
<td>---------</td>
</tr>
<tr>
<td>Fig. 2.2</td>
<td>Roulette Wheel Selection</td>
<td>18</td>
</tr>
<tr>
<td>Fig. 2.3</td>
<td>Example of one-point crossover</td>
<td>20</td>
</tr>
<tr>
<td>Fig. 2.4</td>
<td>Example of mutation operation</td>
<td>21</td>
</tr>
<tr>
<td>Fig. 2.5</td>
<td>Example of a schema contained in both chromosomes</td>
<td>23</td>
</tr>
<tr>
<td>Fig. 3.1</td>
<td>Crossover operation generating invalid offspring</td>
<td>34</td>
</tr>
<tr>
<td>Fig. 3.2</td>
<td>Mutation operation generating invalid offspring</td>
<td>34</td>
</tr>
<tr>
<td>Fig. 3.3</td>
<td>Restore the invalid offspring to its CSD representation</td>
<td>37</td>
</tr>
<tr>
<td>Fig. 4.1</td>
<td>Cascade realization of H(z)</td>
<td>45</td>
</tr>
<tr>
<td>Fig. 4.2</td>
<td>Direct realization of second-order sections</td>
<td>45</td>
</tr>
<tr>
<td>Fig. 4.3</td>
<td>Stability triangle</td>
<td>46</td>
</tr>
<tr>
<td>Fig. 4.4</td>
<td>Modified genetic algorithm</td>
<td>50</td>
</tr>
<tr>
<td>Fig. 4.5</td>
<td>Zeros and poles location of the 4th order lowpass filter</td>
<td>54</td>
</tr>
<tr>
<td>Fig. 4.6</td>
<td>Magnitude response of the 4th order lowpass filter</td>
<td>55</td>
</tr>
<tr>
<td>Fig. 4.7</td>
<td>Zeros and poles location of the 9th order lowpass filter</td>
<td>58</td>
</tr>
<tr>
<td>Fig. 4.8</td>
<td>Magnitude response of the 9th order lowpass filter</td>
<td>59</td>
</tr>
<tr>
<td>Fig. 4.9</td>
<td>Zeros and poles location of the 8th order bandpass filter</td>
<td>62</td>
</tr>
<tr>
<td>Fig. 4.10</td>
<td>Magnitude response of the 8th order bandpass filter</td>
<td>63</td>
</tr>
<tr>
<td>Fig. 4.11</td>
<td>Zeros and poles location of the 6th order bandpass filter</td>
<td>66</td>
</tr>
<tr>
<td>Fig. 4.12</td>
<td>Magnitude response of the 6th order bandpass filter</td>
<td>67</td>
</tr>
<tr>
<td>Fig. 4.13</td>
<td>Zeros and poles location of the 5th order highpass filter</td>
<td>69</td>
</tr>
<tr>
<td>Fig. 4.14</td>
<td>Magnitude response of the 5th order highpass filter</td>
<td>70</td>
</tr>
<tr>
<td>Fig.</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>------</td>
<td>-----------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>4.15</td>
<td>Zeros and poles location of the 7\textsuperscript{th} order highpass filter</td>
<td>73</td>
</tr>
<tr>
<td>4.16</td>
<td>Magnitude response of the 7\textsuperscript{th} order highpass filter</td>
<td>74</td>
</tr>
<tr>
<td>4.17</td>
<td>Zeros and poles location of the 4\textsuperscript{th} order lowpass filter</td>
<td>77</td>
</tr>
<tr>
<td>4.18</td>
<td>Magnitude response of the 4\textsuperscript{th} order lowpass filter</td>
<td>78</td>
</tr>
<tr>
<td>4.19</td>
<td>Group delay response of the 4\textsuperscript{th} order lowpass filter</td>
<td>79</td>
</tr>
<tr>
<td>4.20</td>
<td>Zeros and poles location of the 8\textsuperscript{th} order bandpass filter</td>
<td>82</td>
</tr>
<tr>
<td>4.21</td>
<td>Magnitude response of the 8\textsuperscript{th} order bandpass filter</td>
<td>83</td>
</tr>
<tr>
<td>4.22</td>
<td>Group delay response of the 8\textsuperscript{th} order bandpass filter</td>
<td>84</td>
</tr>
<tr>
<td>5.1</td>
<td>Implementation of the doubly complementary filter pair</td>
<td>89</td>
</tr>
<tr>
<td>5.2</td>
<td>Zeros and poles location of the lowpass filter</td>
<td>97</td>
</tr>
<tr>
<td>5.3</td>
<td>Magnitude response of the lowpass-highpass filter pair</td>
<td>98</td>
</tr>
<tr>
<td>5.4</td>
<td>Sum of the lowpass-highpass filter pair</td>
<td>99</td>
</tr>
<tr>
<td>5.5</td>
<td>Sum of power of the lowpass-highpass filter pair</td>
<td>100</td>
</tr>
<tr>
<td>5.6</td>
<td>Zeros and poles location of the bandstop filter</td>
<td>102</td>
</tr>
<tr>
<td>5.7</td>
<td>Magnitude response of the bandpass-bandstop filter pair</td>
<td>103</td>
</tr>
<tr>
<td>5.8</td>
<td>Sum of the bandpass-bandstop filter pair</td>
<td>104</td>
</tr>
<tr>
<td>5.9</td>
<td>Sum of power of the bandpass-bandstop filter pair</td>
<td>105</td>
</tr>
<tr>
<td>6.1</td>
<td>Computational cost of a 2-D filter with central symmetry</td>
<td>107</td>
</tr>
<tr>
<td>6.2</td>
<td>Computational cost of a 2-D filter with quadrantral symmetry</td>
<td>108</td>
</tr>
<tr>
<td>6.3</td>
<td>Computational cost of a 2-D filter with octagonal symmetry</td>
<td>109</td>
</tr>
<tr>
<td>6.4</td>
<td>Magnitude response of the 2\textsuperscript{nd} order 2-D filter</td>
<td>111</td>
</tr>
<tr>
<td>6.5</td>
<td>Magnitude response of the 4\textsuperscript{th} order 2-D filter</td>
<td>113</td>
</tr>
</tbody>
</table>
1.1 Introduction

During the past three decades, advances in digital computer technology and software development have fueled the growth of digital signal processing (DSP). DSP has become the basis of many areas of technology, from mobile phones to modems and multimedia PCs. Digital signal processing is concerned with representing signals by discrete sequences of numbers and processing these sequences in order to estimate characteristic parameters of a signal or to transform a signal into a desirable form.

An important branch of digital signal processing is digital filtering. Digital filtering is a computational process that transforms an input array of samples to an output array of samples according to a prescribed rule. The arrays can be of one-dimensional or multi-dimensional. Typical one-dimensional (1-D) signals are speech signals. Seismic and image signals are typical examples of two-dimensional (2-D) signals. The problem of designing a 1-D or 2-D filter is one of finding filter coefficients such that the filter’s frequency response approximates a desired behavior.

Digital filters can be implemented as software on a general-purpose computer such as a PC, or as a dedicated hardware on a specialized DSP chip. Advances in DSP applications has demanded high throughput rate, low power and low implementation cost of digital filters. Since multipliers in digital filters consume most of power and implementation cost, it is desirable to reduce the complexity of the multipliers. One popular method is to
reduce the number of partial products being summed by constraining the filters to have power-of-two (POT) ([1]-[4]) or Canonical Signed-Digit (CSD) ([5]-[7]) coefficients. In this thesis, an optimization method will be presented to design 1-D and 2-D infinite impulse response (IIR) digital filters with CSD coefficients.

1.2 Classification and Characterization of Digital Filters

Like other systems, digital filters can be classified as time-invariant or time-dependent, causal or noncausal, and linear or nonlinear. This thesis will be concerned exclusively with linear, time-invariant and causal digital filters. Digital filters can be generally classified into two types, namely finite impulse response (FIR) filters if the impulse response of a filter is of finite duration, and infinite impulse response (IIR) filters if the filter’s impulse response contains infinite duration.

1.2.1 Difference Equation

1.2.1.1 One-Dimensional Filters

Digital filters are characterized in terms of difference equations. A linear, time-invariant and casual FIR filter has a difference equation of the form

\[ y(n) = \sum_{i=0}^{N} a(i)x(n-i) \]  \hspace{1cm} (1.1)

From Eq. (1.1) it is clear that the output sequence of a FIR filter is a function of input sequence alone. A 1-D linear, time-invariant and casual IIR filter has a difference equation of the form

\[ y(n) = \sum_{i=0}^{N} a(i)x(n-i) - \sum_{i=1}^{M} b(i)y(n-i) \]  \hspace{1cm} (1.2)
The response of an IIR filter is a function of input sequence as well as the output sequence.

1.2.1.2 Two-Dimensional Filters

A 2-D linear, time-invariant and causal FIR filter can be described by difference equation:

\[ y(m,n) = \sum_{i=0}^{M} \sum_{j=0}^{N} a(i,j)x(m-i,n-j) \]  \hspace{1cm} (1.3)

Similarly, the difference equation that describes a 2-D linear, time-invariant and causal IIR filter can be written as:

\[ y(m,n) = \sum_{i=0}^{M_1} \sum_{j=0}^{N_1} a(i,j)x(m-i,n-j) - \sum_{i=0}^{M_2} \sum_{j=0}^{N_2} b(i,j)y(m-i,n-j) \]  \hspace{1cm} (1.4)

1.2.2 Transfer Function

1.2.2.1 One-Dimensional Filters

Using z-transform, a digital filter can be also characterized by a discrete-time transfer function, which plays a key role in frequency domain analysis. The transfer function of a digital filter is defined as the ratio of the z transform of the output sequence to the z transform of the input sequence, i.e. for a linear, time-invariant digital filter, we have

\[ Y(z) = H(z)X(z) \]  \hspace{1cm} (1.5)

In effect, the transfer function of a digital filter is the z-transform of the impulse response. The one-dimensional z-transform of a sequence \( x(n) \) is defined as:

\[ X(z) = \sum_{n=-\infty}^{\infty} x(n)z^{-n}, \quad z = e^{j\omega T} \]  \hspace{1cm} (1.6)
where $\omega$ is the continuous frequency variable and $T$ is the sampling period. Therefore, the transfer function of a FIR filter is

$$H(z) = \sum_{n=0}^{N} h(n)z^{-n}$$ (1.7)

Alternatively, the transfer function of an IIR filter is

$$H(z) = \sum_{n=0}^{\infty} h(n)z^{-n} = \frac{\sum_{i=0}^{N} a(i)z^{-i}}{\sum_{i=0}^{M} b(i)z^{-i}} = \frac{N(z)}{D(z)}$$ (1.8)

### 1.2.2.2 Two-Dimensional Filters

The two-dimensional z-transform can be defined as:

$$X(z_1,z_2) = \sum_{n=\infty}^{\infty} \sum_{m=\infty}^{\infty} x(n,m)z_1^{-n}z_2^{-m}$$ (1.9)

where $z_i = e^{j\omega_i T_i}$, $i=1,2$. $T_1$ and $T_2$ are sampling period in $z_1$ and $z_2$ respectively. The sampling period in both dimensions are assumed to be equal. The corresponding 2-D transfer function of a linear, time-invariant and casual FIR filter can be written as:

$$H(z_1,z_2) = \sum_{i=0}^{M} \sum_{j=0}^{N} h(i,j)z_1^{-i}z_2^{-j}$$ (1.10)

The transfer function of a 2-D linear, time-invariant and casual IIR filter is:

$$H(z_1,z_2) = \frac{\sum_{i=0}^{M_1} \sum_{j=0}^{N_1} a(i,j)z_1^{-i}z_2^{-j}}{\sum_{i=0}^{M_2} \sum_{j=0}^{N_2} b(i,j)z_1^{-i}z_2^{-j}} = \frac{N(z_1,z_2)}{D(z_1,z_2)}$$ (1.11)

where $z_i = e^{j\omega_i T_i}$, $i=1,2$. 


Assuming that factorization of \( N(z_1, z_2) \) or \( D(z_1, z_2) \) is possible, transfer function of Eq.(1.11) can take three sub-classes of form ([8]).

1. Separable product transfer function

\[
H(z_1, z_2) = H(z_1)H(z_2) = \left[ \sum_{i=0}^{M_1} a_1(i)z_1^{-i} \right] \left[ \sum_{j=0}^{N_2} b_2(j)z_2^{-j} \right] \left[ \sum_{j=0}^{N_1} b_1(i)z_1^{-i} \right] \left[ \sum_{j=0}^{N_2} b_2(j)z_2^{-j} \right]
\]

(1.12)

This sub-class of 2-D transfer function has two advantages. First, the problem of 2-D filter design is reduced to that of 1-D filter design. Second, the stability problem is reduced to that of the 1-D filter. The disadvantage of this sub-class is that the shape of the cutoff boundary is restricted to a rectangular one.

2. Separable numerator non-separable denominator transfer function

\[
H(z_1, z_2) = \frac{\sum_{i=0}^{M_1} a_1(i)z_1^{-i}}{\sum_{i=0}^{M_1} \sum_{j=0}^{N_2} b(i, j)z_1^{-i}z_2^{-j}} \left[ \sum_{j=0}^{N_2} b_2(j)z_2^{-j} \right]
\]

(1.13)

This sub-class of transfer function presents the same difficulties regarding the stability problem as of the general form of Eq.(1.11). Since there is no advantage in using them this sub-class is seldom used.

3. Non-separable numerator separable denominator transfer function

\[
H(z_1, z_2) = \frac{\sum_{i=0}^{M_1} \sum_{j=0}^{N_1} a(i, j)z_1^{-i}z_2^{-j}}{\sum_{i=0}^{M_1} b_1(i)z_1^{-i} \sum_{j=0}^{N_2} b_2(j)z_2^{-j}}
\]

(1.14)

This sub-class of transfer function has the advantages of the separable product filters but not their disadvantage. So circular, elliptical and in general non-symmetrical cutoff
boundary 2-D filters can be designed using this sub-class of transfer function. Therefore this sub-class of transfer functions will be considered in this thesis.

1.2.3 Magnitude and Phase Response

1.2.3.1 One-Dimensional Filters

A digital filter, like an analog filter, can be represented in frequency-domain by a magnitude response and a phase response. In this case, the discrete-time transfer function $H(z)$ is evaluated on the unit circle $|z| = 1$ of the $z$ plane, i.e.

$$H(e^{j\omega}) = H(z)|_{z = e^{j\omega}} = H(e^{j\omega})e^{j\theta(\omega)}$$  \hspace{1cm} (1.12)

where $|H(e^{j\omega})|$ represents the gain of the filter at frequency $\omega$.

$$\theta(\omega) = \arg H(e^{j\omega})$$ represents the phase shift of the filter at frequency $\omega$.

The magnitude response of the filter is defined as

$$|H(e^{j\omega})| = \sqrt{\text{Re}[H(z)]^2 + \text{Im}[H(z)]^2}|_{z = e^{j\omega}}$$  \hspace{1cm} (1.13)

and the phase response is defined as

$$\theta(\omega) = \tan^{-1}\left(\frac{\text{Im}[H(z)]}{\text{Re}[H(z)]}\right)|_{z = e^{j\omega}}$$  \hspace{1cm} (1.14)

The group delay of a filter is a measure of the average delay of the filter as a function of frequency and is defined as

$$\tau(\omega) = -\frac{d\theta(\omega)}{d\omega} = -j\omega \frac{d\theta}{dz}|_{z = e^{j\omega}}$$  \hspace{1cm} (1.15)

For a 1-D IIR filter described by its transfer function Eq. (1.8)
\[ H(z) = \frac{\sum_{i=0}^{N} a(i)z^{-i}}{\sum_{i=0}^{M} b(i)z^{-i}} = \frac{N(z)}{D(z)} \]

\( \tau(\omega) \) can be written as

\[
\tau(\omega) = - \text{Re} \left[ \frac{dH(z)}{dz} \frac{1}{H(z)} \right]_{z=e^{j\omega}} = \text{Re} \left[ \frac{D'(z)}{D(z)} - z \frac{N'(z)}{N(z)} \right]_{z=e^{j\omega}} \tag{1.16}
\]

where \( D'(z) = \frac{dD(z)}{dz} \) and \( N'(z) = \frac{dN(z)}{dz} \).

A desirable group delay characteristic is one that approximately constant over the passband of the filter, which indicates that the phase response is a linear function of \( \omega \). In effect, if the magnitude response of a filter is flat and its phase response is linear in the passband, then the output signal is a delayed replica of the input signal except that a gain and a phase shift are introduced, and we will be able to reconstruct the signal from its Fourier components. If the group delay is not constant in the passband, we have what is known as delay distortion. Delay distortion is quite tolerable in some applications while in others the delay characteristic is required to be fairly flat. Therefore the design of digital filters that approximate both magnitude response and group delay characteristics is of great importance in some cases in signal processing.

### 1.2.3.2 Two-Dimensional Filters

As an extension of the 1-D case, a 2-D digital filter transfer function of Eq. (1.11) can be written as:
\[ H(e^{j\omega_1 T}, e^{j\omega_2 T}) = H(z_1, z_2)|_{z_j = e^{j\omega_j T}, j=1,2} \]
\[ = |H(e^{j\omega_1 T}, e^{j\omega_2 T})| e^{j\theta(\omega_1, \omega_2)} \]  
(1.17)

where \( |H(e^{j\omega_1 T}, e^{j\omega_2 T})| \) is the magnitude response of \( H(z_1, z_2) \).

\( \theta(\omega_1, \omega_2) \) is the phase response of \( H(z_1, z_2) \).

### 1.2.4 Stability

#### 1.2.4.1 One-Dimensional Filters

A digital filter is said to be bounded-input, bounded-output (BIBO) stable if and only if any bounded input results in a bounded output. For a linear and time-invariant digital filter, the response of the filter can be expressed in convolution summation, i.e.

\[ y(n) = \sum_{k=-\infty}^{\infty} h(k)x(n-k) \]  
(1.18)

Eq.(1.18) gives

\[ |y(n)| \leq \sum_{k=-\infty}^{\infty} |h(k)||x(n-k)| \]  
(1.19)

if \( |x(n)| \leq M < \infty \) for all \( n \)

Eq. (1.19) becomes

\[ |y(n)| \leq M \sum_{k=-\infty}^{\infty} |h(k)| < \infty \]

So if \( \sum_{k=-\infty}^{\infty} |h(k)| < \infty \)  
(1.20)

then \( |y(n)| < \infty \) for all \( n \)
Therefore the sufficient condition for a stable system is that its impulse response to be absolutely summable. Since the impulse response of a FIR filter is only defined over a bounded limit, they are always stable. But for IIR filters stability is not always guaranteed.

As in analog systems, the poles of the transfer function of a filter determine whether the filter is stable or not. For an IIR filter described by its transfer function Eq. (1.8)

\[ H(z) = \frac{\sum_{i=0}^{N} a(i)z^{-i}}{\sum_{i=0}^{M} b(i)z^{-i}} = \frac{N(z)}{D(z)} \]

to be stable, the denominator polynomial \( D(z) \) must satisfy the following constraint ([9])

\[ D(z) \neq 0 \quad \forall \quad |z| \geq 1 \]

de in., all the poles of the 1-D transfer function are located inside the unit circle of \( z \) plane.

### 1.2.4.2 Two-Dimensional Filters

Similar to the 1-D case, for a 2-D digital filter described by Eq.(1.11)

\[ H(z_1, z_2) = \frac{\sum_{i=0}^{M_1} \sum_{j=0}^{N_1} a(i, j)z_1^{-i}z_2^{-j}}{\sum_{i=0}^{M_2} \sum_{j=0}^{N_2} b(i, j)z_1^{-i}z_2^{-j}} = \frac{N(z_1, z_2)}{D(z_1, z_2)} \]

the filter is stable if and only if

\[ D(z_1, z_2) \neq 0 \quad \bigcap_{i=1}^{2} |z_i| \geq 1 \quad i = 1, 2 \]

This stability test problem can be reduced to that of 1-D stability problem by constraining the 2-D transfer function to be the sub-class that has separable denominator polynomials.
in $z_1$ and $z_2$, as described in Eq. (1.14). This way, the stability of the designed 2-D filter can be guaranteed using any of the 1-D design technique ([8], [10], [11]).

1.2.5 Some Comparisons of IIR and FIR Digital Filters

Both FIR and IIR filters have their own advantages and disadvantages. In IIR filter, the poles of the transfer function can be placed anywhere inside the unit circle. A consequence of this degree of freedom is that high frequency selectivity can easily be achieved with lower order transfer functions. While in FIR filters, with the poles all located at the origin of the $z$ plane, high selectivity can only be achieved by using higher order transfer function. For the same specification, IIR filters are much more efficient than FIR filters in achieving the given specification on magnitude response.

On the other hand, FIR filters can be readily designed to possess constant group delay over the entire passband, which indicates a linear phase response. In contrast, the group delay of IIR filters can only be approximated. Besides FIR filters are inherently stable while IIR filters suffer from stability problem. The requirement of stabilizing the designed filter and approximating a constant group delay makes the design of IIR filter more complicated.

1.3 Summary of Previous Work

Generally there are two approaches to the design of 1-D IIR digital filters, namely indirect method and direct method ([12], [13]). In indirect method, a prototype normalized analog lowpass filter is first designed through one of the classical analog filter
approximations, such as Butterworth, Chebyshev, Elliptic, etc. Then the desired digital filter is obtained by digitizing the analog filter through the use of one of the mapping procedures such as impulse invariant transformation, matched z-transform or bilinear transformation ([9]). While in direct method, the desired discrete-time transfer function is obtained directly from the given specifications through the use of iterative optimization methods based on linear or nonlinear programming ([12], [13]). In these methods, a discrete-time transfer function is assumed and an error function is formulated on the basis of some desired magnitude or phase response. A norm of the error function is then minimized with respect to the transfer function coefficients. As the value of the norm approaches zero, the resulting magnitude or phase response approaches the desired ones.

Similarly, optimization techniques can be used to design 2-D IIR filters. In [8], [10] and [11], the authors have used linear or non-linear programming technique to design 2-D IIR filters with non-separable numerator and separable denominator polynomials. The stability of the designed filters is guaranteed by forcing the denominator polynomial to have zeros inside the unit circle through some suitable transformation of the filter coefficients.

The design techniques mentioned above all generate filter coefficients with high precision. But in actual implementation of a digital filter, either in software or hardware, the filter coefficients can only be stored in finite length registers. Consequently the coefficients must be quantized before they can be stored. If coefficient quantization is applied, the frequency response of the resulting filter may differ appreciably from the desired
response. If the quantization step is coarse, the filter may even fail to meet the desired specifications. For this reason, it is desirable to design digital filters with finite precision coefficients, and four types of iterative optimization techniques are often employed. They are branch and bound optimization ([14]), discretization and re-optimization ([15]), simulated annealing ([16]) and genetic algorithm ([2]–[7], [17]–[32]).

Genetic algorithm (GA) is a stochastic search method that mimics the process of natural selection and evolution. GA possesses many important features such as parallelism, multiple objectives and multi-modality, which make GA different from other optimization techniques. Filters designed by GA have the potential of obtaining near global optimum solution. During the past decade, GA has been successfully used to design digital filters. At the same time, the design of power-of-two coefficient (POT) or canonical signed-digit coefficient (CSD) filter has received widespread attention in the literature. The utilization of POT or CSD coefficients has the advantage of minimizing the implementation cost of digital filters.

In 1-D case, various techniques have been proposed to design FIR filters with POT or CSD coefficients ([1]–[7]), however, methods of using genetic algorithm to design IIR filters with CSD coefficients has not yet been reported. In [1], IIR filters with POT coefficients are designed with mixed integer linear programming. In [25]–[29], the authors have developed GA-based methods to design IIR filters. But the coefficients are only finite-precision coefficients but not in CSD format, making them inefficient for IIR filter realization. In addition, both methods initialize the GA with coefficients generated
either from MATLAB signal processing package or a bilinear transformation design, which actually introduces an extra step in the GA-based filter design.

Similarly, in 2-D case, using genetic algorithm to design IIR filters with CSD coefficients has received limited attention. In [3] and [7], GA has been used to design 2-D FIR filters. In [30], GA is used to design 2-D IIR filters with finite wordlength. In [4], GA is used to design 2-D IIR filters with POT coefficients.

Complementary filter pairs find applications in many signal processing systems where different frequency bands are to be processed separately. In the past, various techniques have been proposed to design complementary filter pairs ([41]–[51]), but all of them generate high-precision coefficients, not CSD coefficients.

In this thesis, genetic algorithm will be used to design 1-D and 2-D IIR filters and complementary filter pairs with CSD coefficients, thus minimizing implementation cost of IIR digital filters and making them more efficient in high throughput rate DSP application as well.

1.4 Organization of Thesis

Chapter 2 presents genetic algorithm and its fundamental theory. In Chapter 3 we first introduce canonical signed-digit number system. Then obstacles from applying GA to CSD number system are described. Finally the strategy used in this thesis is presented. Chapter 4 presents a GA-based optimization method and its application to 1-D IIR filter
design. In Chapter 5 the proposed method is used to design doubly complementary filter pairs. In Chapter 6 the proposed method is extended to design 2-D IIR filters. In each chapter some design examples are given to demonstrate the usefulness of the proposed method. Chapter 7 is the concluding remarks.
CHAPTER 2. GENETIC ALGORITHM

2.1 Introduction

Genetic algorithm (GA) is a stochastic optimization algorithm that was originally motivated by the mechanisms of natural selection and the principle of survival of the fittest. It was originally developed by John Holland at the University of Michigan in 1975 ([33]) and was further improved by Goldberg ([34]) and others. Compared with other traditional optimization techniques such as calculus-based technique and enumerative technique, genetic algorithm operates in an entirely different optimization procedure, and it may be differentiated from those conventional techniques by the following characteristics ([34]):

- Direct manipulation of the encoded representation of parameters, rather than the parameters themselves.
- Search from a large number of points rather than from a single point, thus reducing the possibility of reaching local optimum
- Blind search by sampling, ignoring all auxiliary information
- Use stochastic rather than deterministic operations.

These characteristics provide GA with further flexibility and robustness in searching large, noisy, multimodal and discrete problem space. Ever since it was invented, GA has been proven to be effective and robust in a wide variety of applications.

2.2 Genetic Algorithm Cycle

Genetic algorithm is a searching process that mimics the process of natural selection and
evolution. Usually a simple GA consists of three operations: Reproduction, Crossover Operation and Mutation Operation. Fig. 2.1 illustrates a typical GA cycle.

The population is a collection of chromosomes. Each chromosome is an encoded trial solution to the problem to be solved. The fitness value of each chromosome is evaluated by calculating a pre-specified objective function. Reproduction operation selects chromosomes to reproduce offspring through genetic operations—crossover and mutation. The current population is then replaced by the offspring based on a certain replacement strategy. Such a GA cycle is repeated until a desired termination criterion is reached. The chromosome that has the highest fitness is the solution to the target problem.

A genetic algorithm, in its canonical form, operates in the following step-by-step procedure:

- Step1. Randomly generate an initial population;
- Step 2. Evaluate the fitness of each chromosome;
- Step 3. Select a group of chromosomes as mating parents;
- Step 4. Create offspring by applying crossover and mutation operation;
- Step 5. Setup a new population using a certain replacement strategy;
• Step 6. Go to step 2 if the termination criterion is not reached, otherwise, stop and return the best chromosome.

In the following sections, detailed introduction of each step involved in the above procedure will be described.

2.2.1 Encoding Scheme

In genetic algorithm, parameters of the target problem are encoded to form chromosomes. Each chromosome represents a possible solution to the problem. It is usually expressed in a string of variables. The variable can be binary, real number or other forms. Bit-sting encoding is the most classical approach used in GA because of its simplicity.

Encoding of the parameters is a crucial issue in any GA because it is an important link to the problem being solved. An inappropriate encoding scheme will severely limit the searching space and the resultant solution may not be optimal. Until now, there is no general theory or method to find an optimal encoding scheme. The rule of thumb is that the encoded chromosome must well represent the characteristic of the target problem and enhance the GA to explore new solution spaces. In addition, the length of the chromosome should be kept as small as possible, because shorter chromosome length tends to give GA a faster convergence rate.

2.2.2 Reproduction

Reproduction emulates the survival-of-the fittest mechanism in nature. In reproduction operation, chromosomes are selected with a probability proportional to their individual
fitness. It is expected that fitter chromosomes have higher chances to be selected. Then in the subsequent generations, fitter chromosomes will begin to dominate and eradicate the weaker ones from the population, and thus having higher chances of survival.

There are many different types of reproduction operators including rank selection, fitness proportionate selection (FPS) and tournament selection, but the key idea is to give preference to fitter individuals. Roulette Wheel Selection is a commonly used scheme to implement FPS, and it executes in the following procedure:

- Step 1. Sum the fitness of all the chromosomes;
- Step 2. Randomly generate a number between 0 and the total fitness;
- Step 3. Return the first chromosome whose fitness added to the fitness of the preceding chromosomes is greater than or equal to the randomly generated number;
- Step 4. Repeat Step 1 to Step 3 until the population size is reached.

To look more clearly at this method, consider the roulette wheel below, assuming that we have three chromosomes: 01001, 01110 and 10000 to be selected and their fitness value is 19%, 35% and 46% of the total fitness respectively.

![Roulette Wheel Selection](image.png)

Fig. 2.2 Roulette Wheel Selection
When selecting the mating parents, the roulette wheel is spun three times. It is obvious from the above wheel that chromosome 10000 have higher chance to be selected and may be selected more than once.

2.2.3 Crossover

Crossover is the main operator for GA to explore new searching space. It is a recombination operator that combines subparts of two parent chromosomes, thus creating offspring that contain both parents’ genes. A probability rate, namely crossover probability, is set to control the operation rate. Crossover probability, denoted as $p_c$, is defined as the ratio of the number of offspring produced in each generation to the population size ([36]). A higher crossover probability allows GA to explore more diversity of the solution space, thus reducing the chance of stuck in local optimum. But if the crossover probability is too high, the chromosomes with high fitness may be destroyed.

A variety of crossover operation has been used including one-point crossover, two-point crossover, uniform crossover etc. One-point crossover operation is performed in the following steps:

- Step 1. Randomly select two chromosomes as mating parents;
- Step 2. Randomly generate a number between 0 and 1;
- Step 3. If the number is greater than $p_c$, the two chromosomes are cloned to the next generation;
- Step 4. Otherwise, randomly select a crossover point and the portions of the two chromosomes beyond this point are exchanged to form two offspring;
- Step 5. Repeat Step 1 to Step 4 until the whole population has gone through the above procedure.

Fig. 2.3 illustrates a one-point crossover.

```
crossover point
0 0 1 0 1 1 0 1 1 0 0 0 0 0 1 0 1 0 1 0 1 0 1
  1 0 1 1 0 1 0 1 0 1 0 1
Parents
    0 0 1 0 1 1 0 1 0 1 0 1 0 1 0 1
    1 0 1 1 0 0 1 0 1 0 0 0
    0 0 1 0 1 1 0 1 0 1 0 1
    1 0 1 1 0 0 1 0 1 0 0 0
Offspring
```

Fig. 2.3 Example of one-point crossover

### 2.2.4 Mutation

Crossover operation alone can obviously generate a large variety of offspring. However, depending on the initial population chosen, there may not be enough variety to ensure the GA searches the entire parameter space, then GA may converge to a local optimum rather than a global optimum. The introduction of mutation operator fills in this gap and makes the entire searching space reachable.

The mutation operation usually occurs at a low probability rate $p_m$, namely mutation probability. Mutation probability is defined as the probability of mutating each gene. This operator results in a random walk in the parameter space and introduces new information into the evolution process. Although happened at a very low rate, mutation is the key...
operation to maintain diversity in GA. It prevents GA from being stuck at local optimal solution.

In binary encoding mutation is just a bit flip. During mutation operation, each bit position is first applied a probability test. If it passes the probability test, the bit is flipped from ‘1’ to ‘0’ or ‘0’ to ‘1’. If it failed the test, the bit remains unchanged. An example of bit mutation is shown in Fig. 2.4.

<table>
<thead>
<tr>
<th>Original Chromosome</th>
<th>1 0 0 1 1 0 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>New Chromosome</td>
<td>1 0 0 0 1 0 0</td>
</tr>
</tbody>
</table>

Fig. 2.4 Example of mutation operation

2.2.5 Fitness Evaluation

Except for encoding, fitness evaluation is another important link between GA and the problem to be solved. The fitness value is usually evaluated through an objective function or cost function, which contains all the information about the problem. The objective function is a main source to measure the performance of each possible solution. The better the solution encoded by a chromosome, the higher the fitness.

2.2.6 Replacement Strategy

After generating the offspring, two representative strategies can be applied for old population replacement, which is steady-state replacement and generational replacement. In steady-state replacement, only a few chromosomes are replaced. Usually the worst
chromosomes are replaced by one or two new chromosomes. While in generational replacement, an equal number of new chromosomes are created to replace the entire old population. Using this strategy it is possible that the best chromosomes of the old population be replaced and fail to reproduce offspring in the next generation. In some extreme cases, the highest fitness in the offspring may be even lower than that of the old population, which actually means that the GA is getting worse as it evolves. For this reason generational replacement is usually combined with an elitist strategy where one or a few of the best chromosomes are copied to the succeeding generation. On the other hand, the elitist strategy may increase the speed of domination of a population by a super chromosome, which will decrease the diversity of the population. But on balance it improves the performance of genetic algorithm.

2.3 Schema Theorem

To understand the mechanism of genetic algorithm, the notion of schemata has to be introduced in [34]. Since every bit in the binary encoding of a possible solution captures some information about the problem, high performance chromosomes are similar in the sense that they contain 1's and 0's in particular position. These similarities encode the high performance features. It is stated that schemata are sets of strings that have one or more features in common.

A schema is built by introducing a “don’t care” symbol, “*”, into the encoding alphabet. In binary encoding, a schema is built by alphabet \{0, 1, *\}. The “*” symbol can represent either 0 or 1 at the position it stands. A schema represents all strings which match it on
all positions other than "*". For example, the schema in Fig. 2.5 is contained in both chromosome 1 and chromosome 2.

Chromosome 1: 0 0 1 0 1 1 0
Chromosome 2: 1 0 1 1 0 1 1 0
Schema: * 0 1 * * 1 1 0

Fig. 2.5 Example of a schema contained in both chromosomes

Assume that \( n \) is the number of don’t care symbol, it is clear that each schema matches exactly \( 2^n \) chromosomes.

In the following sections, we will discuss the effects of the genetic operations, i.e. reproduction operation, crossover operation and mutation operation, on a schema’s survival. We will introduce the Schema Theorem or Fundamental Theorem of genetic algorithm. We begin with the introduction of two properties of the schema.

2.3.1 Schema Properties

Two important properties are used to describe schema:

- Order of a schema \( S \)

The order of a schema \( S \), denoted by \( o(S) \), is defined as the number of fixed positions in a given schema, i.e. 0 or 1 in binary encoding. For example the order of the above schema is \( o(*01**110)=5 \).

- Length of a schema \( S \)
The length of a schema \( S \), denoted by \( \delta(S) \), is defined as the distance between the first and the last fixed positions in a given schema. For example the length of the above schema is \( \delta(*01**110)=6 \).

### 2.3.2 Effect of Reproduction

Let \( f(S,t) \) represents the average fitness of a schema \( S \), \( \zeta(S,t) \) be the number of chromosomes matched by schema \( S \) in the current generation. If fitness proportionate selection is used as the reproduction operator, we can estimate the number of chromosomes that match schema \( S \) in the next generation. The number is:

\[
\zeta(S,t+1) = \zeta(S,t) \times \frac{f(S,t)}{F(t)}
\]

where \( F(t) \) is the average fitness of the current generation.

Let

\[
\varepsilon = \frac{f(S,t) - F(t)}{F(t)}
\]

If \( \varepsilon > 0 \), it means that the schema has an above-average fitness in the current generation.

Substitute Eq.(2.2) into Eq.(2.1), we have

\[
\zeta(S,t+1) = \zeta(S,t) \times (1 + \varepsilon)
\]

Starting at \( t=0 \) and assuming a stationary value of \( \varepsilon \), we obtain the equation

\[
\zeta(S,t+1) = \zeta(S,0) \times (1 + \varepsilon)^t
\]

Eq.(2.4) shows that above-average schemata receive an exponentially increasing number of offspring in the next generation after reproduction operation, and below-average schemata die off.
2.3.3 Effect of Crossover

Assume that the length of the chromosome is \( L \), and \( \delta(S) \) is the order of a schema \( S \). A schema survives the crossover operation if the crossover point falls outside its defining length \( \delta(S) \). Usually the crossover point is selected uniformly among \( L-1 \) positions. Now if one-point crossover is applied, the probability a schema \( S \) being destroyed is:

\[
p_c(S) = \frac{\delta(S)}{L-1}
\]  

(2.5)

so the probability of the survival of schema \( S \) is:

\[
p_s(S) = 1 - \frac{\delta(S)}{L-1}
\]  

(2.6)

Assuming the crossover probability is \( p_c \), the probability of a schema’s survival is:

\[
p_{sc}(S) \geq 1 - p_c \frac{\delta(S)}{L-1}
\]  

(2.7)

Eq.(2.7) shows that the longer the schema, the more likely it will be destroyed by crossover operation.

2.3.4 Effect of Mutation

Assume that the mutation probability in a bit mutation operation is \( p_m \), then the probability of a single bit to survive is \( 1-p_m \). A schema survives the mutation operation if all of the positions in the schema remain unchanged. Let \( o(S) \) denotes the order of a schema \( S \). We can derive that the probability of a schema \( S \) surviving the bit mutation operation is:

\[
p_s(S) = (1 - p_m)^{o(S)}
\]  

(2.8)

Since usually \( p_m << 1 \), Eq.(2.6) can be approximated by:
\[ p_{sm}(S) = 1 - p_m \cdot o(S) \]  \hfill (2.9)

Eq.(2.9) shows that the higher the order of a schema, the more likely it will be destroyed by mutation operation.

### 2.3.5 Schema Growth Equation

Combining the effects discussed above, we obtain a schema growth equation:

\[
\zeta(S, t+1) = \zeta(S, t) \times \frac{f(S, t)}{F(t)} \times p_{sc} \times p_{sm}
\]

\[
\approx \zeta(S, t) \times \frac{f(S, t)}{F(t)} \times \left[ 1 - \frac{p_c \cdot \delta(S)}{L-1} - p_m \cdot o(S) \right] \hfill (2.10)
\]

Eq.(2.10) shows that short, lower-order, above average schemata receive exponentially increasing number of representatives in the subsequent generations of a GA. For this reason, encoding scheme should be chosen so that short, lower-order schemata are relevant to the underlying problem.

### 2.4 An Example

Having introduced genetic algorithm and its fundamental theorem, we now present an example to show how genetic algorithm works. This example is an unconstraint optimization problem:

**PROBLEM:**

Maximize function \( f(x_1, x_2) = 21.5 + x_1 \sin(4\pi x_1) + x_2 \sin(4\pi x_2) \), where \(-3.0 \leq x_2 \leq 12.1\) and \(4.1 \leq x_2 \leq 5.8\).
GENETIC ALGORITHM:

STEP 1. Variable Encoding

In this problem, we encode the two variables $x_1$ and $x_2$ as binary bit strings, and chromosomes are formed by concatenating the two binary strings. The length of the bit string is determined by the dynamic range required. In this example a dynamic range of $10^{-5}$ is required for $x_1$ and $x_2$, then $x_1$ will need a 18-bit binary string and $x_2$ will need a 15-bit binary string. So the length of the chromosome is $18+15=33$ bits. For example we have a chromosome which is obtained by randomly generating either a '0' or a '1' for each bit:

```
000001010100101001 101111011111110
```

Then the corresponding decimal value of $x_1$ and $x_2$ respectively are:

$x_1=-2.687969$

$x_2=5.361653$

STEP 2. Initialization

In this step, a population size of chromosomes are randomly generated to initialize the population. For simplicity we use 4 as the population size in this example. Assume that the initialization population is as follows:

$c_1=[0011101011100110000000010101010010000]$

$c_2=[100110110101101000100000010111001]$

$c_3=[1111100010111011000111101000111101]$

$c_4=[111101001111010101000001010110101010]

Their corresponding decimal values are:
\[ c_1 = [x_1, x_2] = [0.474101, 4.170144] \]
\[ c_2 = [x_1, x_2] = [6.159951, 4.109598] \]
\[ c_3 = [x_1, x_2] = [11.671267, 4.873501] \]
\[ c_4 = [x_1, x_2] = [11.446273, 4.171908] \]

STEP 3. Fitness Evaluation

We use the function value as the fitness value of GA. So

\[ \text{fitness} = f(x_1, x_2) = 21.5 + x_1 \sin(4\pi x_1) + x_2 \sin(4\pi x_2). \]

\[ \text{fitness} (c_1) = f(0.474101, 4.170144) = 17.370896 \]
\[ \text{fitness} (c_2) = f(6.159951, 4.109598) = 29.406122 \]
\[ \text{fitness} (c_3) = f(11.671267, 4.873501) = 26.401669 \]
\[ \text{fitness} (c_4) = f(11.446273, 4.171908) = 10.252480 \]

STEP 4. Reproduction

In this example we use Roulette Wheel Selection as the reproduction operation.

1. Calculate the total fitness.

\[ F = \sum_{i=1}^{4} \text{fitness}(c_i) = 83.431167 \]

2. Randomly generate a number in the range [0, 83.431167].

   Assume that the number generated in this step is 76.178609.

3. Select a chromosome based on the randomly generated number and the subtotal of chromosomes’ fitness.

   Table 2.1 shows the subtotal of chromosomes’ fitness in this example.
<table>
<thead>
<tr>
<th>Chromosome</th>
<th>Sum of the Preceding Fitness</th>
</tr>
</thead>
<tbody>
<tr>
<td>fitness ((c_1))= 17.370896</td>
<td>17.370896</td>
</tr>
<tr>
<td>fitness ((c_2))= 29.406122</td>
<td>17.370896+29.406122=46.777018</td>
</tr>
<tr>
<td>fitness ((c_3))= 26.401669</td>
<td>46.777018+26.401669=73.178687</td>
</tr>
<tr>
<td>fitness ((c_4))= 10.252480</td>
<td>73.178687+10.252480=83.431167</td>
</tr>
</tbody>
</table>

Table 2.1 Subtotal of chromosomes' fitness

According to Table 2.1 and the number 76.178609, chromosome \(c_3\) is selected out.

4. Repeat Step 2 and Step 3 to select other three chromosomes as parents.

Assume that the four selected chromosomes are as following.

\[
c_1' (c_3) = [1111100010111011100011101000111101]
\]

\[
c_2' (c_2) = [100110110100101101010000001011101]
\]

\[
c_3' (c_1) = [00111010110011000000010101001000]
\]

\[
c_4' (c_2) = [1001101101001011010000000010111011]
\]

STEP 5. Crossover

Now we use one-point crossover operation to create offspring. Assume that the crossover probability \(p_c\) is 50%.

1. Randomly select two chromosomes.

Chromosomes \(c_2'\) and \(c_3'\) are selected as mating parents.

2. Randomly generate a number between [0,1].

Number 0.163275 is generated.

3. Since 0.163275< \(p_c\), crossover operation is performed. Randomly select the crossover point between [1,32]. The crossover point selected is 1. \(c_2'\) and \(c_3'\) exchange genes after the crossover point.
4. Repeat Step 1–3 until the population size of offspring are created. In this case, 4 offspring are created.

After crossover operation, 4 new chromosomes are created as following:

\[
\begin{align*}
  c_2'' &\ 1011101101110011000000010101001000 \\
  c_3'' &\ 0001101001100101101000000010111001 \\
  c_1' &\ 11111000101110110001110100011101 \\
  c_4' &\ 100110111010110101100000010111001
\end{align*}
\]

STEP 6. Mutation

Bit mutation with a probability of 0.01 is used in this example. The first step in mutation operation is to perform probability test as introduced in Section 2.2.4. In this example, 3 genes passed the test, shown in Table 2.3.

<table>
<thead>
<tr>
<th>Chromosomes</th>
<th>Gene</th>
<th>Random Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>(c_2'')</td>
<td>32</td>
<td>0.003113</td>
</tr>
<tr>
<td>(c_3'')</td>
<td>1</td>
<td>0.000946</td>
</tr>
<tr>
<td>(c_1')</td>
<td>6</td>
<td>0.009857</td>
</tr>
</tbody>
</table>

Table 2.3 Mutation operation

After mutation operation, we get a new generation of chromosomes:

\[
  n_1 \quad 10110101110011000000010101001010
\]
\[
\begin{align*}
n_2 &= 100110110100101101000000010111001 \\
n_3 &= 1111100101110110011101000111101 \\
n_4 &= 100110110100101101000000010111001
\end{align*}
\]

STEP 7. Decode the chromosomes and calculate the fitness values.

\[
\begin{align*}
\textit{fitness} (n_1) &= 19.91025 \\
\textit{fitness} (n_2) &= 29.406122 \\
\textit{fitness} (n_3) &= 5.702781 \\
\textit{fitness} (n_4) &= 29.406122
\end{align*}
\]

Up to this point, we have finished the first iteration of genetic algorithm. To perform subsequent iterations, we just repeat STEP 4 to STEP 7 until the maximum iteration is reached. The best chromosome of the last iteration then becomes the highly evolved solution.
CHAPTER 3. CANONICAL SIGNED-DIGIT (CSD) COEFFICIENTS

3.1 Representation of Filter Coefficients as CSD Number

Traditionally the design of digital IIR filters is a multi-stage process ([9], [29]). First filter coefficients with infinite precision are produced using various approaches ([8]–[13]). Then these coefficients are quantized. At this stage finite wordlength analysis must be performed because coefficient quantization may have undesirable effects on the designed filter’s behavior. For example, degradation of the frequency response, alternation of the poles-zeros position, and in extreme case, the filter even become unstable. Sometimes the filter’s performance degrades so much that the designer has to go back to the first stage. Therefore it is desirable to design IIR filters with finite wordlength coefficients, and a suitable number system to represent the filter coefficients needs to be chosen first.

Usually filter coefficients are stored using their binary representation. However, in recent years, advances in DSP application demand low complexity, low power and high throughput digital filters. One commonly approach to achieve this is to use power-of-two (POT) or canonical signed-digit (CSD) number system to represent filter coefficients.

3.1.1 Canonical Signed-Digit Representation

In CSD number system, a fractional number \( x \) is represented using the form of:

\[
x = \sum_{k=1}^{L} s_k 2^{-p_k}
\]
where $s_k$ are ternary digits, $s_k \in \{1, 0, 1\}$, and $\bar{1}$ is defined as $-1$.

$M$ is a pre-specified wordlength, and $p_k \in \{0, 1, \ldots, M\}$.

$L$ is a pre-specified maximum number of non-zero digits.

In addition, a CSD representation cannot contain adjacent non-zero digits, i.e.

$$s_k \times s_{k+1} = 0, \text{ for all } s_k \in \{0,1,\cdots,M-1\}$$

Take coefficient 0.421875 for instance, if we limit $M=8$ and $L=3$, it is represented as $01001010$. It is clear that all the non-zero digits are separated by at least one zero digit.

In CSD representation, the most significant bit is located on the left handside and is decoded as $2^0$. So,

$$0.421875 = 2^{-1} - 2^{-4} - 2^{-6}$$

### 3.1.2 Advantage of CSD Representation

Compared with conventional binary representation, CSD representation contains fewer non-zero digits, which means less partial product in multiplications. For example, the binary representation of 15 is $(1111)_2$, and there are four non-zero digits. On the other hand in CSD representation 15 is $1000\bar{1}$, and only two non-zero digits are needed. Nowadays, multipliers in digital filters are realized with wired shifters, adders and subtractors. The use of CSD representation can reduce the number of adders and subtractors, thus reducing the complexity of multipliers. Therefore digital filters having CSD coefficients will have faster response time and consume less power in DSP chip.
3.2 Problems of Genetic Operators with CSD Coefficients

The unique encoding scheme makes genetic algorithms quite suitable for handling discrete number space. However GA cannot be directly applied to the case of CSD coefficients. This is due to the complication arising from the direct operations of crossover and mutation operations to CSD number. Direct crossover and mutation may cause the resulting offspring coefficients cease to conform to CSD format.

Fig. 3.1 shows that crossover operation may create invalid offspring. In this case, the crossover operation causes two consecutive non-zero digits become adjacent in one of the offsprings, which is not allowed in CSD representation.

D₁ (CSD parent) 1 0 1 0 1 0 1 0
D₂ (CSD parent) 0 1 0 1 0 1 0 0

\[ \text{crossover point} \]

D₁' (invalid offspring) 1 0 1 0 1 1 0 0
D₂' (CSD offspring) 0 1 0 1 0 0 1 0

Fig. 3.1 Crossover operation generating invalid offspring

Fig. 3.2 shows that the same problem exists in mutation operation.

D₁ (CSD parent) 0 1 0 1 0 1 0 0

\[ \text{bit mutation} \]

D₁' (invalid offspring) 0 1 0 1 1 1 0 0

Fig. 3.2 Mutation operation generating invalid offspring

The problems illustrated above make the design of filters with CSD coefficients difficult, and special strategies have to be developed.
3.3 Strategies of Previous Literature

Different strategies have been proposed to address this problem. Generally they fall in two categories. One is to use special encoding scheme ([3]–[5]). The other is to use CSD number restoration technique to restore those invalid offspring to their CSD number representations ([6], [23]).

1. Special Encoding Scheme

The strategy here is to use special encoding scheme to make sure that only the pre-specified maximum number of non-zero digits will appear in each coefficient. Take coefficient $0.421875 = 2^{-1} - 2^{-4} - 2^{-6}$ for instance. When encoding it into a GA chromosome, it is not directly encoded in its CSD representation, which is 01001010. Instead, it is encoded in the following format:

```
  1 0 0 1 | 1 1 0 0 | 1 1 0
```

Here a signed digit, which can be $\bar{1}$ or 1, is followed by a binary representation of the exponent, which can only be 0 or 1. For example in the last four digits "1110", $\bar{1}$ is the signed digit and demonstrate that the exponent, represented by "110", is negative. So the first four digits represent $2^{-1}$. The next four digits represent $-2^{-4}$. The last four digits represent $-2^{-6}$. This way, it is guaranteed that only three or less non-zero digits will appear in each coefficient.

But there are some drawbacks with this method. First, it only limits the maximum number of non-zero digit but cannot prevent two non-zero digits from being adjacent.
Since in genetic algorithm, crossover and mutation all happen in random manner, it is possible that two exponents become adjacent or even equal after a certain crossover or mutation operation. In the above example,

\[
\begin{array}{ccccccc}
1 & 0 & 0 & 1 & 1 & 1 & 0 \\
\end{array}
\]

It is possible that a certain random mutation make "1100" become "1110" or "1101". For this reason, some other strategies have to be developed to eliminate this problem. In [5] the authors choose to discard the offspring having adjacent non-zero digits and regenerate a new offspring.

Second, in some cases, this encoding scheme makes chromosomes too long. Assume that \( M=16/L=4 \) CSD coefficients are desired, to represent a maximum exponent of 16, 4 binary digits are required. So each coefficient will be 20-bit long, which is four-bit longer than the original CSD representation. This may cause some problems for genetic algorithm. Just as we have explained in Section 2.3, short, lower-order, above average schemata receive increase exponentially in the subsequent generations. Usually when the chromosomes are too long, it becomes hard for the GA to find the best schemata. So the convergence rate of GA will become slow, and the performance also degrades.

Third, the decoding of the coefficients’ value is not straightforward in this method. It is clear that the exponent values have to be decoded first. The execution speed of the program may slow down when the filter to be designed have many coefficients.
Other special encoding schemes are also available. In [3] and [4], POT coefficients have also been encoded using integer encoding. But the authors have pointed out that the results of integer encoding are not as good as that of binary encoding.

2. CSD Number Restoration Technique

In [18], [19], the authors use a CSD number restoration technique to restore invalid offspring to their CSD representations after each crossover and mutation operation. The authors proposed three theorems to eliminate consecutive non-zero digits and to find the occurrence of out-of-range situation. Fig. 3.3 shows an example of this technique.

\[ D_1 \text{ (invalid offspring)} \quad \overline{1 0 1 0 1 0 1 0} \]
\[ D_1' \text{ (restored coefficient)} \quad 0 1 0 1 0 1 0 0 \]

Fig. 3.3 Restore the invalid offspring to its CSD representation

This strategy also has some drawbacks. First, this technique only targets on consecutive-nonzero-digit problem, but cannot limit the maximum number of non-zero digits. Using the theorems proposed, consecutive 1/-1 can be eliminated as shown in Fig. 3.3, but the maximum number of non-zero digits depends on the specific case and cannot be controlled.

Second, it is possible that the result of genetic operations leads to a number beyond the pre-specified wordlength. In these cases, using the restoration technique proposed will end up with a CSD number that is not the original coefficient. So the judgment of out-of-range situation is necessary. In this technique the judgment of out-of-range and the restoration operate bit by bit, and may lower the execution speed of the program.
Third, the judgment of out-of-range and the restoration may have to be carried out more than one time for one coefficient, because in some cases, the restoration may result in another two consecutive non-zero digits. This will also slow down the speed of the program.

3.4 New CSD Coefficients Restoration Technique

To address the above problems, a new CSD number restoration technique is proposed in this thesis. Different from the CSD restoration technique proposed in [6] and [23], this technique operates on decimal number of the coefficients. Using this technique, if any coefficient is found violating CSD representation, the coefficient will first be decoded to its decimal number, and then the decimal number will be resorted to its nearest CSD number. This technique targets both the consecutive 1/-1 problem and the limit on maximum number of non-zero digits. The judge of the out-of-range situation and the restoration operation is faster. For each invalid coefficient, the restoration operation needs to be carried out for only one time.

Assume that \textit{dec} is the decimal number of a violated coefficient, and we want to convert it to a $M=16/L=4$ CSD number, denoted as \{\textit{d}_0, \textit{d}_1, \textit{d}_2, \ldots, \textit{d}_{13}\}. The pseudo-code and the detailed explanation of the conversion procedure are given in Section 3.4.1 and 3.4.2 respectively. Then Section 3.4.3 shows the test result of the restoration technique.

3.4.1 Pseudo Code of the Restoration Technique

void DecimalToCSD(){
if (dec is out-of-range AND dec>0) { 
    d_0 = d_2 = d_4 = d_6 = 1, conversion completed
}

elseif (dec is out-of-range AND dec<0)
    { 
    d_0 = d_2 = d_4 = d_6 = -1, conversion completed 
}

/*start from d_0 until d_15, find whether a -1, 0 or 1 should be put on d_i */

else{
    for (let variable i change from 0 to 15, the step is 1) {
        if (counter ≥ 4) the conversion is completed
        elseif (dec - dec \(d_{2j-1} - d_i\)) = 0) { d_i = 1, conversion completed}
        elseif (dec + dec \(d_{2j-1} - d_i\)) = 0) { d_i = -1, conversion completed}
        elseif (\(|dec - dec \(d_{2j-1} - d_i\)| ≤ max dec \(d_{i+1} - d_{15}\)) { d_i = 1, d_{i+1} = 0, counter+1}
        elseif (\(|dec + dec \(d_{2j-1} - d_i\)| ≤ max dec \(d_{i+1} - d_{15}\)) { d_i = -1, d_{i+1} = 0, counter+1}
        else { d_i = 0 }
    }
}

3.4.2 Explanations

1. During the conversion, a variable named counter is used to record how many non-zero digits have been found out. In this case, counter should be less than 4. When counter reaches 4, the conversion procedure is completed and all the other positions after the 4th non-zero digit are assigned ‘0’. This way, the least significant non-zero bits in the coefficients has actually been truncated.
2. Since the maximum decimal number can be represented by a $M=16/L=4$ CSD number is 1.328125 in which case $d_0 = d_2 = d_4 = d_6 = 1$, if a decimal number is greater than 1.328125 or less than $-1.328125$, the genetic operations must have resulted in a coefficient that is out-of-range. The resulting coefficient cannot be restored to an $M=16/L=4$ CSD number. Under this circumstance, 1.328125 or $-1.328125$ is the nearest CSD number that can represent $dec$, so the program will assign $d_0 = d_2 = d_4 = d_6 = 1$ or $d_0 = d_2 = d_4 = d_6 = -1$.

3. Assume that we have found $d_1 = -1$ and $d_0 = 0, d_2 = 0, d_3 = 0, d_4 = 0$. Now we are about to find whether $d_5$ deserves $-1, 0$ or $1$. Under this circumstance, the meaning of $dec_{(d_0\cdots d_1)}$ and $\max dec_{(d_0\cdots d_5)}$ is as following:

$$dec_{(d_0\cdots d_1)} = -2^{-1} + 2^{-5}$$
$$\max dec_{(d_0\cdots d_5)} = 2^{-7} + 2^{-9}$$

If $|dec - dec_{(d_0\cdots d_1)}| \leq \max dec_{(d_0\cdots d_5)}$, it means that $dec - dec_{(d_0\cdots d_1)}$ can be represented by the CSD number formed by the rest of the bits $\{d_7, d_8, d_9, \cdots d_{15}\}$, so a ‘1’ can be assigned to $d_4$.

### 3.4.3 Test Result of the Restoration Technique

The proposed restoration technique can be used to convert any decimal number to its nearest CSD number, since we know that not every decimal number has a corresponding CSD representation. To test the accuracy of such conversion, the new restoration
technique has been used to convert some high precision multiplier coefficients. These coefficients were used in [37] as the starting point of a local search algorithm. The conversion results and their corresponding errors are shown in Table 3.1. The mean error of the 30 conversions is 0.445%, which shows that the conversion is very efficient. When the restoration technique is used in the GA-based filter design method to restore coefficients violating CSD format, the design procedure converges very fast.

The execution speed of the proposed restoration technique has also been tested. It is used to convert 30,000 decimal numbers. The CPU time is only 6 seconds. This test is on a Pentium III 733MHz computer.
<table>
<thead>
<tr>
<th>Infinite-Precision Multiplier Coefficients (M=12/L=3 CSD number)</th>
<th>Conversion Result</th>
<th>Decimal Number of the CSD Number</th>
<th>Conversion Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.738409</td>
<td>$2^0-2^{-2}-2^{-6}$</td>
<td>0.734375</td>
<td>0.546%</td>
</tr>
<tr>
<td>0.960374</td>
<td>$2^0-2^3-2^{-7}$</td>
<td>0.9609375</td>
<td>0.059%</td>
</tr>
<tr>
<td>0.629449</td>
<td>$2^1+2^3+2^{-8}$</td>
<td>0.62890625</td>
<td>0.086%</td>
</tr>
<tr>
<td>1.116458</td>
<td>$2^0+2^3-2^{-7}$</td>
<td>1.1171875</td>
<td>0.065%</td>
</tr>
<tr>
<td>0.605182</td>
<td>$2^1+2^3-2^{-6}$</td>
<td>0.609375</td>
<td>0.693%</td>
</tr>
<tr>
<td>1.173078</td>
<td>$2^0+2^2-2^{-4}$</td>
<td>1.1875</td>
<td>1.229%</td>
</tr>
<tr>
<td>-0.850835</td>
<td>$-2^6+2^3+2^{-5}$</td>
<td>-0.84375</td>
<td>0.833%</td>
</tr>
<tr>
<td>-0.860000</td>
<td>$-2^0+2^3+2^{-6}$</td>
<td>-0.859375</td>
<td>0.073%</td>
</tr>
<tr>
<td>-0.931460</td>
<td>$-2^4+2^4+2^{-7}$</td>
<td>-0.9296875</td>
<td>0.190%</td>
</tr>
<tr>
<td>-0.940429</td>
<td>$-2^0+2^3+2^{-8}$</td>
<td>-0.94140625</td>
<td>0.104%</td>
</tr>
<tr>
<td>-0.983693</td>
<td>$-2^0+2^6+2^{-10}$</td>
<td>-0.983398438</td>
<td>0.030%</td>
</tr>
<tr>
<td>-0.986166</td>
<td>$-2^0+2^6-2^{-9}$</td>
<td>-0.986328125</td>
<td>0.016%</td>
</tr>
<tr>
<td>0.135843</td>
<td>$2^3+2^6-2^{-8}$</td>
<td>0.13671875</td>
<td>0.645%</td>
</tr>
<tr>
<td>0.278901</td>
<td>$2^2+2^5-2^{-9}$</td>
<td>0.279296875</td>
<td>0.142%</td>
</tr>
<tr>
<td>0.535773</td>
<td>$2^1+2^3+2^{-8}$</td>
<td>0.53515625</td>
<td>0.115%</td>
</tr>
<tr>
<td>0.697447</td>
<td>$2^0+2^2-2^{-4}$</td>
<td>0.6875</td>
<td>1.426%</td>
</tr>
<tr>
<td>0.773093</td>
<td>$2^0-2^2+2^{-5}$</td>
<td>0.78125</td>
<td>1.055%</td>
</tr>
<tr>
<td>0.917937</td>
<td>$2^0-2^{-4}-2^{-6}$</td>
<td>0.921875</td>
<td>0.429%</td>
</tr>
<tr>
<td>0.026265</td>
<td>$2^5-2^8-2^{-10}$</td>
<td>0.026367188</td>
<td>0.388%</td>
</tr>
<tr>
<td>-0.444500</td>
<td>$-2^1+2^4-2^{-7}$</td>
<td>-0.4453125</td>
<td>0.183%</td>
</tr>
<tr>
<td>-0.249249</td>
<td>$-2^2+2^{-10}$</td>
<td>-0.249023438</td>
<td>0.091%</td>
</tr>
<tr>
<td>-0.891543</td>
<td>$-2^0+2^3-2^{-6}$</td>
<td>-0.890125</td>
<td>0.103%</td>
</tr>
<tr>
<td>-0.425920</td>
<td>$-2^1+2^4+2^{-6}$</td>
<td>-0.421875</td>
<td>0.950%</td>
</tr>
<tr>
<td>-1.122226</td>
<td>$-2^0-2^3+2^{-8}$</td>
<td>-1.12109375</td>
<td>0.101%</td>
</tr>
<tr>
<td>0.135843</td>
<td>$2^3+2^6-2^{-8}$</td>
<td>0.13671875</td>
<td>0.645%</td>
</tr>
<tr>
<td>0.278901</td>
<td>$2^2+2^5-2^{-9}$</td>
<td>0.279296875</td>
<td>0.142%</td>
</tr>
<tr>
<td>0.535773</td>
<td>$2^1+2^5+2^{-8}$</td>
<td>0.53515625</td>
<td>0.115%</td>
</tr>
<tr>
<td>0.697447</td>
<td>$2^0-2^2-2^{-4}$</td>
<td>0.6875</td>
<td>1.426%</td>
</tr>
<tr>
<td>0.773093</td>
<td>$2^0-2^2+2^{-5}$</td>
<td>0.78125</td>
<td>1.055%</td>
</tr>
<tr>
<td>0.917937</td>
<td>$2^0-2^{-4}-2^{-6}$</td>
<td>0.921875</td>
<td>0.429%</td>
</tr>
</tbody>
</table>

Table 3.1 Test result of the conversion technique
CHAPTER 4. DESIGN OF 1-D IIR DIGITAL FILTERS

In this chapter, we will propose a 1-D IIR filter design method. The proposed method uses genetic algorithm as the iterative optimization technique. It produces stable IIR filters with canonical signed-digit coefficients. The design is based on cascade structure of IIR filter. We begin our discussion with a brief introduction of 1-D IIR filter structures.

4.1 IIR Filter Structures

1-D IIR digital filters can be realized in several structures including direct forms, cascade structure, parallel structure, state-space structure, parallel all-pass realization ([38]) etc. Different filter structure can differ quite significantly with respect to complexity, number of elements and properties. One structure might be economical in terms of elements but is relatively sensitive to coefficient quantization, and another structure might require a large number of multipliers but is more robust under finite wordlength constraints. In selecting a filter structure, both computational complexity of a filter and its performance under finite wordlength constraints should be taken into consideration. The computational complexity of a digital filter structure is given by the total number of multipliers and the total number of two-input adders required for its implementation. Computational complexity provides a rough indication of the structure's implementation cost.

Direct forms require less multipliers than cascade structure and parallel structure, but are the most sensitive to coefficient quantization error. Cascade and parallel structures are far
less sensitive than direct forms and is the most often used structure in 1-D IIR filter
design. State-space structure is computationally the most complex realization, but it is
more robust under finite wordlength constraints. Parallel all-pass realization is the most
computationally efficient structure and has low passband sensitivity as well. Using
parallel all-pass realization, it is possible to design two IIR filters at one filter’s cost,
which is known as doubly complementary filter pair.

In this chapter, we consider the design of IIR filter based on cascade structure, and in the
next chapter we will design doubly complementary filter pairs using parallel all-pass
realization.

### 4.2 Cascade Structure

When an \(N^{\text{th}}\) order IIR digital filter is realized in direct forms, the sensitivity of the
structure to coefficient quantization increases rapidly with \(N\) ([9]). Consequently, small
errors introduced by coefficient quantization give rise to large errors in the magnitude and
phase responses. This problem can be partly solved by using cascade structure. An
arbitrary transfer function in the form of:

\[
H(z) = \frac{\sum_{i=0}^{N} a(i)z^{-i}}{\sum_{i=0}^{M} b(i)z^{-i}} = \frac{N(z)}{D(z)}
\]

can be decomposed into product of first and second order transfer function as

\[
H(z) = \prod_{i=1}^{K} H_i(z) = \prod_{i=1}^{K} \frac{a_{0i} + a_{1i}z^{-1} + a_{2i}z^{-2}}{1 + b_{1i}z^{-1} + b_{2i}z^{-2}}
\] (4.1)

44
with \( a_{2i} = b_{2i} = 0 \) for a first order transfer function.

Hence
\[
Y(z) = \left[ H_1(z)X(z) \right] H_2(z) \cdots H_K(z)
\]
\[
= \left[ H_1(z)Y_1(z) \right] H_2(z) \cdots H_K(z)
\]

\[
\ldots \ldots \ldots \ldots
\]
\[
= H_K(z)Y_{K-1}(z)
\]

(4.2)

where \( Y_i(z) = H_i(z)Y_{i-1}(z) \). This way, \( H(z) \) can be realized using the cascade structure of Fig. 4.1.

![Cascade realization of \( H(z) \)](image)

Fig. 4.1 Cascade realization of \( H(z) \)

Each individual first/second order section can be realized by a direct form, as shown in Fig. 4.2.

![Direct realization of second-order section](image)

Fig. 4.2 Direct realization of second-order section

### 4.3 Stability Consideration

In Section 1.3.3 we have shown that the sufficient and necessary condition for an IIR transfer function stable is that all its poles lie inside the unit circle. For very high order
transfer functions, it is difficult to determine the pole locations analytically, and the use of some type of root finding computer program is necessary. But for a second order transfer function, the stability can be checked easily by examining its denominator coefficients. Let

\[ D(z) = 1 + d_1 z^{-1} + d_2 z^{-2} \]

denotes the denominator of the transfer function. For the stability of the transfer function, coefficients \(d_1\) and \(d_2\) must follow:

\[ |d_2| < 1 \]

\[ |d_1| < 1 + d_2 \]  \hspace{1cm} (4.3)

The region of the \((d_1, d_2)\) plane where two coefficient conditions of Eq. (4.3) are satisfied is a triangle and is known as the stability triangle ([38]), illustrated in Fig. 4.3.

![Stability triangle](image)

**Fig. 4.3** Stability triangle

When using cascade structure, the stability of the designed filter can be guaranteed by making each second order section stable.

4.4 **Design Problem Formulation**

The design problem here is to use GA-based optimization technique to search for coefficients of the filter transfer function such that the difference between the designed
filter and the desired filter is minimized. This difference is measured by least square error
criterion, which is also known as $l_2$ norm.

Let $|H_D(e^{j\omega T})|$ denotes the magnitude response of the filter to be designed, $|H_I(e^{j\omega T})|$ be
the desired magnitude response. Let $\tau_o$ denotes the group delay response of the designed
filter and is calculated using Eq.(1.16), and $\tau_g$ be a constant representing the ideal group
delay response. The value of $\tau_g$ is usually chosen equal to the order of the designed filter
([12], [13]). In order to formulate the design problem, two cases will be considered in this
thesis.

1. For the approximation of the magnitude response only

In this case, the least square error criterion is used in the following relationship:

$$E_{ls}(j\omega_m) = \sum_{m \in I_{ps}} E_{Mag}^2(j\omega_m)$$  \hspace{1cm} (4.4)

and $E_{Mag}(j\omega_m) = |H_I(\exp(j\omega_m T))| - |H_D(\exp(j\omega_m T))|$

where $I_{ps}$ is the set of all discrete frequency points along $\omega$ axis in the passband and
stopband.

2. For the approximation of both the magnitude response and group delay response.

In this case, the least square error criterion is used in the following manner:

$$E_{ls}(j\omega_m) = \alpha \sum_{m \in I_{ps}} E_{Mag}^2(j\omega_m) + (1 - \alpha) \sum_{m \in I_p} E_{\tau}^2(j\omega_m) \hspace{1cm} (4.5)$$

and $E_{Mag}(j\omega_m) = |H_I(\exp(j\omega_m T))| - |H_D(\exp(j\omega_m T))|$

$$E_{\tau}(j\omega_m) = \tau_g T - \tau_D(j\omega_m T)$$
where \( I_\omega \) is the set of frequency points along \( \omega \) axis in the passband, and \( \alpha \) is a weighting factor to emphasize the magnitude or group delay characteristics.

For a given specification, the design procedure is divided into four stages:

Stage 1. Filter order estimation

Determine the order of the filter to be designed using MATLAB order estimation function \texttt{cheb1ord}.

Stage 2. Choose parameters of genetic algorithm, including:

1. Maximum iteration or target fitness value:

   Maximum iteration depends on the complexity of the problem to be solved, the length of the chromosomes. The higher the order of the filter to be designed, the more iterations are needed. In this thesis, the maximum iteration ranges from 500 to 1000 iterations.

2. Population size \( N \):

   Population size is usually chosen to be 1.5 to 2 times the chromosome’s length.

3. Crossover probability \( p_c \):

   Crossover probability is usually chosen between 50\% to 100\%. In this thesis, \( p_c \) is set to be 70\%. This value is obtained through an exhaustive search.

4. Mutation probability \( p_m \):

   Mutation probability is usually chosen between 1\% to 10\%. In this thesis, \( p_m \) is set to be 1\%. This value is also obtained through an exhaustive search.

Stage 3. Use GA-based design method to design filters.
Stage 4. Use MATLAB tools to analysis the zeros and poles position, magnitude response and group delay of the designed filter.

4.5 Modified Genetic Algorithm

The genetic algorithm introduced in Chapter 2 has been modified in order to produce CSD filter coefficients and to guarantee that the designed filter is stable, as shown in Fig. 4.4.
Fig. 4.4 Modified genetic algorithm
The modified GA is executed in the following steps:

1. Initialization

In the step, a population size $N$ of chromosomes randomly generated. Each chromosome is constructed by concatenating all the coefficients in transfer function Eq. (4.1). Thus for a IIR filter with the order of $2K$, consisting of $K$ second order sections, is represented as:

$$\{a_{01}, a_{11}, a_{21}, b_{11}, b_{21}, a_{02}, a_{12}, \ldots, a_{0K}, a_{1K}, a_{2K}, b_{1K}, b_{2K}\}$$

Each coefficient is encoded in its CSD number representation. The wordlength $M$ and the maximum number of non-zero digits $L$ can be set at the beginning of the program.

2. Fitness Evaluation

The fitness evaluation is a two-step process. The first step is to check the stability of the filter encoded in each chromosome by examining the denominator coefficients of each second order section. If coefficients $b_{11}$ and $b_{12}$ in the denominator fail the stability constraint introduced in Section 4.3, the chromosome will be assigned a fitness value 0. Otherwise, the total error is first calculated using least squares error function $E_{\omega} (j\omega_n)$. Then the fitness value is calculated as the reciprocal of the total error, i.e.

$$fitness = \frac{1}{E_{\omega} (j\omega_n)}$$

3. Reproduction

Roulette Wheel Selection is used as the reproduction operator introduced in Section 2.2.2.

4. Crossover and Restoration

One-point crossover is used in this design flow. After each crossover operation, the coefficient where the crossover point lies in will be checked upon CSD format. If the
coefficient is found violated, it will be restored to its nearest CSD number, using the restoration technique discussed in Chapter 3.

5. Mutation and Restoration

Mutation operator is the simple single bit flip. After mutation, each coefficient in the offspring is checked upon CSD format. Any coefficient violating CSD representation will be restored to its nearest CSD number.

6. Fitness Evaluation

The stability and fitness values of the offspring are evaluated.

7. Replacement Strategy

Elitist strategy is applied for old generation replacement. After reproduction the best chromosome (with maximum fitness) and the worst chromosome (with the minimum fitness) in the offspring are found out. If the best chromosome in the offspring is worse than the best chromosome in the parent generation, the best chromosome of the parent generation will be copied to the offspring to replace the worst chromosome in the offspring.

After replacement, step 3 to step 7 are repeated if the pre-specified maximum iteration is not reached or the pre-specified target fitness value is not obtained. When either of the above stop criteria is reached, the design is completed and the best chromosome of the last iteration is returned as the final result. Decoding the chromosome will give the filter coefficients \( \{ a_0, a_1, \ldots, a_{2K}, b_1, b_{2K} \} \).
4.6 Design Examples

To illustrate the usefulness of the proposed design method, 1-D IIR filters with different specifications and characteristics have been designed. The designed filters’ coefficients, zeros and poles location, magnitude response and group delay response will be shown in this section.

4.6.1 Design of 1-D IIR Filters without Constant Group Delay Characteristic

4.6.1.1 Lowpass IIR Filters

Example 1

A 4th order lowpass IIR filter is designed to satisfy the following specification:

\[ |H(e^{j\omega})| = \begin{cases} 1 & 0 \leq \omega \leq 1.0 \\ 0 & 2.0 \leq \omega \leq \frac{\omega_s}{2} \end{cases} \]

where \( \omega_s = 2\pi \text{ rad/sec} \) and \( T = 1 \text{ sec} \). In the filter order estimation, the passband ripple and stopband attenuation are set to be 1dB and 40dB respectively, i.e., \( R_p = 1 \text{dB}, R_s = 40 \text{dB} \).

The result of `cheb1ord` function is \( N = 4 \), so the transfer function is of the form:

\[
H(z) = \prod_{i=1}^{2} \frac{a_{qi} + a_{1i}z^{-1} + a_{2i}z^{-2}}{1 + b_{1i}z^{-1} + b_{2i}z^{-2}}
\]

The coefficient wordlength is restricted to be 16 bits, and the maximum number of non-zero digits is restricted to be 4. Table 4.1 shows the canonical signed-digit coefficients of the designed filter. Fig. 4.5 shows its zeros and poles location, and it can be seen that the designed filter is stable. Fig. 4.6 is the corresponding magnitude response, and the stopband attenuation is above 45dB.

53
<table>
<thead>
<tr>
<th>Coefficients</th>
<th>CSD Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_{01}$</td>
<td>$-2^{-2} + 2^{-5} - 2^{-7} + 2^{-9}$</td>
</tr>
<tr>
<td>$a_{11}$</td>
<td>$-2^{-1} + 2^{-3} - 2^{-6} - 2^{-8}$</td>
</tr>
<tr>
<td>$a_{21}$</td>
<td>$-2^{-2} + 2^{-5} + 2^{-11} + 2^{-13}$</td>
</tr>
<tr>
<td>$b_{11}$</td>
<td>$-2^{-1} - 2^{-4} + 2^{-6} - 2^{-10}$</td>
</tr>
<tr>
<td>$b_{21}$</td>
<td>$2^0 - 2^2 - 2^{-4} - 2^{-6}$</td>
</tr>
<tr>
<td>$a_{02}$</td>
<td>$2^{-2} - 2^{-6} - 2^{-8} - 2^{-10}$</td>
</tr>
<tr>
<td>$a_{12}$</td>
<td>$2^{-2} - 2^{-5} - 2^{-12} + 2^{-14}$</td>
</tr>
<tr>
<td>$a_{22}$</td>
<td>$2^{-2} - 2^{-5} + 2^{-7} + 2^{-9}$</td>
</tr>
<tr>
<td>$b_{12}$</td>
<td>$-2^0 + 2^{-2} + 2^{-4} + 2^{-9}$</td>
</tr>
<tr>
<td>$b_{22}$</td>
<td>$2^{-2} - 2^{-4} + 2^{-6}$</td>
</tr>
</tbody>
</table>

Table 4.1 CSD coefficients of the 4th order lowpass filter

![Fig.4.5 Zeros and poles location of the 4th order lowpass filter](image_url)
Fig.4.6 Magnitude response of the 4th order lowpass filter
Example 2

A 9th order lowpass IIR filter with the following specification:

\[
|H(e^{j\omega})| = \begin{cases} 
1 & 0 \leq \omega \leq 1.5 \\
0 & 1.8 \leq \omega \leq \frac{\omega_0}{2}
\end{cases}
\]

where \( \omega_0 = 2\pi \text{ rad/sec} \) and \( T = 1 \text{ sec} \). In filter order estimation, \( R_p = 1 \text{ dB}, R_s = 50 \text{ dB} \). The coefficient wordlength is restricted to be 16 bits, and the maximum number of non-zero digits is restricted to be 4. The transfer function is of the form:

\[
H(z) = \frac{a_0 + a_1z^{-1}}{1 + b_1z^{-1}} \prod_{i=2}^{5} \frac{a_0 + a_1z^{-1} + a_2z^{-2}}{1 + b_1z^{-1} + b_2z^{-2}}
\]

Table 4.2 shows the canonical signed-digit coefficients of the designed filter. Fig. 4.7 shows its zeros and poles location and Fig. 4.8 shows the corresponding magnitude response. It can be seen that the stopband attenuation is above 42dB, and does not reach 50dB. The degrade performance is due to the long chromosome length. In this example, the filter order is 9, so the filter to be designed has 23 coefficients. Since the wordlength is 16 bit, each chromosome is 368-bit long. While in genetic algorithm, when the chromosomes are too long, it will become very hard for the GA to find the best schemata.
<table>
<thead>
<tr>
<th>Coefficients</th>
<th>CSD Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_{01}$</td>
<td>$-2^{-1} + 2^{-6} + 2^{-8} + 2^{-10}$</td>
</tr>
<tr>
<td>$a_{11}$</td>
<td>$-2^{-1} + 2^{-6} + 2^{-8} - 2^{-10}$</td>
</tr>
<tr>
<td>$b_{11}$</td>
<td>$2^{-4} - 2^{-6} - 2^{-8} + 2^{-15}$</td>
</tr>
<tr>
<td>$a_{02}$</td>
<td>$-2^{0} + 2^{-2} + 2^{-9} + 2^{-15}$</td>
</tr>
<tr>
<td>$a_{12}$</td>
<td>$-2^{0} + 2^{-2} + 2^{-5} - 2^{-8}$</td>
</tr>
<tr>
<td>$a_{22}$</td>
<td>$-2^{0} + 2^{-2} + 2^{-7} - 2^{-11}$</td>
</tr>
<tr>
<td>$b_{12}$</td>
<td>$2^{-7} + 2^{-11} + 2^{-14}$</td>
</tr>
<tr>
<td>$b_{22}$</td>
<td>$2^{0} - 2^{-2} + 2^{-4} + 2^{-7}$</td>
</tr>
<tr>
<td>$a_{03}$</td>
<td>$2^{0} - 2^{-6} + 2^{-8} - 2^{-11}$</td>
</tr>
<tr>
<td>$a_{13}$</td>
<td>$2^{-1} - 2^{-3} + 2^{-5} - 2^{-8}$</td>
</tr>
<tr>
<td>$a_{23}$</td>
<td>$2^{-2} + 2^{-4} + 2^{-9} - 2^{-12}$</td>
</tr>
<tr>
<td>$b_{13}$</td>
<td>$2^{0} - 2^{-2} + 2^{-4} + 2^{-7}$</td>
</tr>
<tr>
<td>$b_{23}$</td>
<td>$2^{-2} + 2^{-5} + 2^{-10} + 2^{-13}$</td>
</tr>
<tr>
<td>$a_{04}$</td>
<td>$2^{-1} - 2^{-4} - 2^{-6} - 2^{-9}$</td>
</tr>
<tr>
<td>$a_{14}$</td>
<td>$2^{0} - 2^{-2} - 2^{-4} - 2^{-8}$</td>
</tr>
<tr>
<td>$a_{24}$</td>
<td>$2^{-1} - 2^{-4} - 2^{-8} - 2^{-12}$</td>
</tr>
<tr>
<td>$b_{14}$</td>
<td>$2^{-2} - 2^{-7} + 2^{-9} - 2^{-12}$</td>
</tr>
<tr>
<td>$b_{24}$</td>
<td>$2^{-4} - 2^{-6} + 2^{-8} + 2^{-14}$</td>
</tr>
<tr>
<td>$a_{05}$</td>
<td>$2^{0} + 2^{-3} - 2^{-5} - 2^{-10}$</td>
</tr>
<tr>
<td>$a_{15}$</td>
<td>$2^{-1} + 2^{-3} - 2^{-5} - 2^{-7}$</td>
</tr>
<tr>
<td>$a_{25}$</td>
<td>$2^{0} + 2^{-3} - 2^{-5} - 2^{-7}$</td>
</tr>
<tr>
<td>$b_{15}$</td>
<td>$-2^{-4} + 2^{-6} + 2^{-12} - 2^{-14}$</td>
</tr>
<tr>
<td>$b_{25}$</td>
<td>$2^{-1} - 2^{-3} + 2^{-7} - 2^{-9}$</td>
</tr>
</tbody>
</table>

Table 4.2 CSD coefficients of the 9th order lowpass filter
Fig. 4.7 Zeros and poles location of the 9th order lowpass filter
Fig. 4.8 Magnitude response of the 9th order lowpass filter
4.6.1.2 Bandpass IIR Filters

Example 1

A 8th order bandpass IIR filter with the following specification:

\[
|H(e^{j\omega T})| = \begin{cases} 
0 & 0 \leq \omega \leq 0.7 \\
1 & 1.2 \leq \omega \leq 1.9 \\
0 & 2.4 \leq \omega \leq \frac{\omega_p}{2}
\end{cases}
\]

where \( \omega_p = 2\pi \text{ rad/sec} \) and \( T = 1 \text{ sec} \). In filter order estimation, \( R_p = 1 \text{dB}, R_s = 40 \text{dB} \). The coefficient wordlength is restricted to be 16 bits, and the maximum number of non-zero digits is restricted to be 4. The transfer function is of the form:

\[
H(z) = \prod_{i=1}^{4} \frac{a_{wi} + a_{zi} z^{-1} + a_{zi} z^{-2}}{1 + b_{wi} z^{-1} + b_{zi} z^{-2}}
\]

Table 4.3 shows the canonical signed-digit coefficients of the designed filter. Fig. 4.9 shows its zeros and poles location and Fig. 4.10 shows the corresponding magnitude response.
<table>
<thead>
<tr>
<th>Coefficients</th>
<th>CSD Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_{01}$</td>
<td>$2^{-1} + 2^{-5} + 2^{-11}$</td>
</tr>
<tr>
<td>$a_{11}$</td>
<td>$2^{0} - 2^{-3} + 2^{-8} + 2^{-10}$</td>
</tr>
<tr>
<td>$a_{21}$</td>
<td>$2^{-1} + 2^{-5} + 2^{-8} - 2^{-11}$</td>
</tr>
<tr>
<td>$b_{11}$</td>
<td>$2^{0} - 2^{-2} - 2^{-4} + 2^{-8}$</td>
</tr>
<tr>
<td>$b_{21}$</td>
<td>$2^{0} - 2^{-2} + 2^{-5} - 2^{-7}$</td>
</tr>
<tr>
<td>$a_{02}$</td>
<td>$-2^{-3} + 2^{-6} + 2^{-8} - 2^{-10}$</td>
</tr>
<tr>
<td>$a_{12}$</td>
<td>$-2^{-2} - 2^{-4} - 2^{-6} - 2^{-8}$</td>
</tr>
<tr>
<td>$a_{22}$</td>
<td>$-2^{-2} + 2^{-6} - 2^{-9} - 2^{-11}$</td>
</tr>
<tr>
<td>$b_{12}$</td>
<td>$-2^{0} + 2^{-2} - 2^{-4} + 2^{-8}$</td>
</tr>
<tr>
<td>$b_{22}$</td>
<td>$2^{0} - 2^{-2} + 2^{-4} - 2^{-8}$</td>
</tr>
<tr>
<td>$a_{03}$</td>
<td>$-2^{-1} + 2^{-5} - 2^{-8} - 2^{-11}$</td>
</tr>
<tr>
<td>$a_{13}$</td>
<td>$2^{0} - 2^{-2} + 2^{-5} - 2^{-9}$</td>
</tr>
<tr>
<td>$a_{23}$</td>
<td>$-2^{-1} + 2^{-5}$</td>
</tr>
<tr>
<td>$b_{13}$</td>
<td>$2^{2} - 2^{-9} - 2^{-11}$</td>
</tr>
<tr>
<td>$b_{23}$</td>
<td>$2^{-1} - 2^{-4} - 2^{-7} - 2^{-9}$</td>
</tr>
<tr>
<td>$a_{04}$</td>
<td>$2^{-1} - 2^{-5} + 2^{-8}$</td>
</tr>
<tr>
<td>$a_{14}$</td>
<td>$-2^{0} + 2^{-4} - 2^{-6} - 2^{-9}$</td>
</tr>
<tr>
<td>$a_{24}$</td>
<td>$2^{-1} + 2^{-6} + 2^{-13}$</td>
</tr>
<tr>
<td>$b_{14}$</td>
<td>$-2^{-2} - 2^{-4} - 2^{-6} - 2^{-8}$</td>
</tr>
<tr>
<td>$b_{24}$</td>
<td>$2^{-1} - 2^{-5} + 2^{-7} - 2^{-13}$</td>
</tr>
</tbody>
</table>

Table 4.3 CSD coefficients of the 8th order bandpass filter
Fig. 4.9 Zeros and poles location of the 8th order bandpass filter
Fig. 4.10 Magnitude response of the 8th order bandpass filter
Example 2

A 6\textsuperscript{th} order bandpass IIR filter is designed with the following specification:

\[ |H(e^{j\omega})| = \begin{cases} 
0 & 0 \leq \omega \leq 0.1\pi \\
1 & 0.4\pi \leq \omega \leq 0.6\pi \\
0 & 0.9\pi \leq \omega \leq \frac{\omega_0}{2}
\end{cases} \]

where \( \omega_0 = 2\pi \) rad/sec and \( T = 1 \) sec. In filter order estimation, \( R_p = 1 \text{dB} \), \( R_s = 40 \text{dB} \).

The coefficient wordlength is restricted to be 16 bits, and the maximum number of non-zero digits is restricted to be 4. The transfer function is of the form:

\[ H(z) = \prod_{i=1}^{3} \frac{a_{ai} + a_{i1}z^{-1} + a_{i2}z^{-2}}{1 + b_{i1}z^{-1} + b_{i2}z^{-2}} \]

Table 4.4 is the CSD coefficients of the designed filter. Fig. 4.11 and Fig. 4.12 shows its zeros and poles location and magnitude response respectively.
<table>
<thead>
<tr>
<th>Coefficients</th>
<th>CSD Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_{01}$</td>
<td>$-2^{-2} - 2^{-4} - 2^{-7} + 2^{-10}$</td>
</tr>
<tr>
<td>$a_{11}$</td>
<td>$2^{-1} + 2^{-3} - 2^{-7} + 2^{-11}$</td>
</tr>
<tr>
<td>$a_{21}$</td>
<td>$2^{-2} - 2^{-4} - 2^{-7} + 2^{-10}$</td>
</tr>
<tr>
<td>$b_{11}$</td>
<td>$2^{0} - 2^{-2} - 2^{-6} + 2^{-8}$</td>
</tr>
<tr>
<td>$b_{21}$</td>
<td>$2^{-1} + 2^{-8} + 2^{-10} - 2^{-13}$</td>
</tr>
<tr>
<td>$a_{02}$</td>
<td>$2^{0} - 2^{-2} - 2^{-8} - 2^{-10}$</td>
</tr>
<tr>
<td>$a_{12}$</td>
<td>$-2^{-7} + 2^{-9} + 2^{-11}$</td>
</tr>
<tr>
<td>$a_{22}$</td>
<td>$2^{0} - 2^{-2} + 2^{-7} - 2^{-9}$</td>
</tr>
<tr>
<td>$b_{12}$</td>
<td>$-2^{-6} - 2^{-9} + 2^{-13} + 2^{-15}$</td>
</tr>
<tr>
<td>$b_{22}$</td>
<td>$-2^{-2} + 2^{-5} - 2^{-7} + 2^{-10}$</td>
</tr>
<tr>
<td>$a_{03}$</td>
<td>$2^{-1} + 2^{-4} + 2^{-9} + 2^{-11}$</td>
</tr>
<tr>
<td>$a_{13}$</td>
<td>$2^{0} + 2^{-3} - 2^{-5} - 2^{-12}$</td>
</tr>
<tr>
<td>$a_{23}$</td>
<td>$2^{-1} + 2^{-4} + 2^{-8} - 2^{-10}$</td>
</tr>
<tr>
<td>$b_{13}$</td>
<td>$-2^{0} + 2^{-2} - 2^{-4} + 2^{-7}$</td>
</tr>
<tr>
<td>$b_{23}$</td>
<td>$2^{-1} + 2^{-6} + 2^{-9} - 2^{-11}$</td>
</tr>
</tbody>
</table>

Table 4.4 CSD coefficients of the 6th order bandpass filter
Fig. 4.11 Zeros and poles location of the 6\textsuperscript{th} order bandpass filter
Fig. 4.12 Magnitude response of the 6th order bandpass filter
4.6.1.3 Highpass IIR Filters

Example 1

A 5th order highpass IIR filter is designed with the following specification:

\[
|H(e^{j\omega})| = \begin{cases} 
0 & 0 \leq \omega \leq 0.2\pi \\
1 & 0.4\pi \leq \omega \leq \frac{\omega_3}{2}
\end{cases}
\]

where \(\omega_3=2\pi \text{ rad/sec}\) and \(T=1\) sec. In filter order estimation, \(R_p=1\)dB, \(R_s=50\)dB. The coefficient wordlength is restricted to be 16 bits, and the maximum number of non-zero digits is restricted to be 4. The transfer function is of the form:

\[
H(z) = \frac{a_{0l} + a_{11}z^{-1}}{1 + b_{11}z^{-1}} \prod_{i=2}^{3} \frac{a_{0i} + a_{1i}z^{-1} + a_{2i}z^{-2}}{1 + b_{1i}z^{-1} + b_{2i}z^{-2}}
\]

Table 4.5 is the CSD coefficients of the designed filter. Fig. 4.13 shows its zeros and poles location and Fig. 4.14 shows the corresponding magnitude response.

<table>
<thead>
<tr>
<th>Coefficients</th>
<th>CSD Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a_{01})</td>
<td>(-2^0 + 2^{-2} + 2^{-6} + 2^{-9})</td>
</tr>
<tr>
<td>(a_{11})</td>
<td>(2^0 - 2^{-2} - 2^{-4} + 2^{-6})</td>
</tr>
<tr>
<td>(b_{11})</td>
<td>(-2^{-3} + 2^{-6} - 2^{-10} - 2^{-12})</td>
</tr>
<tr>
<td>(a_{02})</td>
<td>(2^{-1} - 2^{-4} + 2^{-8} + 2^{-11})</td>
</tr>
<tr>
<td>(a_{12})</td>
<td>(-2^0 + 2^{-2} - 2^{-7} + 2^{-11})</td>
</tr>
<tr>
<td>(a_{22})</td>
<td>(2^{-1} - 2^{-4} + 2^{-9} - 2^{-14})</td>
</tr>
<tr>
<td>(b_{12})</td>
<td>(-2^{-1} - 2^{-5} + 2^{-7} + 2^{-10})</td>
</tr>
<tr>
<td>(b_{22})</td>
<td>(2^{-2} + 2^{-7} + 2^{-9} + 2^{-11})</td>
</tr>
<tr>
<td>(a_{03})</td>
<td>(-2^0 + 2^{-2} + 2^{-4} + 2^{-6})</td>
</tr>
<tr>
<td>(a_{13})</td>
<td>(2^0 + 2^{-4} - 2^{-6} - 2^{-10})</td>
</tr>
<tr>
<td>(a_{23})</td>
<td>(-2^{-1} + 2^{-4} + 2^{-6} - 2^{-8})</td>
</tr>
<tr>
<td>(b_{13})</td>
<td>(-2^0 + 2^{-3} + 2^{-5} - 2^{-7})</td>
</tr>
<tr>
<td>(b_{23})</td>
<td>(2^0 - 2^{-2} - 2^{-4} + 2^{-8})</td>
</tr>
</tbody>
</table>

Table 4.5 CSD coefficients of the 5th order highpass filter

68
Fig. 4.13 Zeros and poles location of the $5^{th}$ order highpass filter
Fig. 4.14 Magnitude response of the 5th order highpass filter
Example 2

A 7\textsuperscript{th} order highpass IIR filter is designed with the following specification:

\[ |H(e^{j\omega T})| = \begin{cases} 
0 & 0 \leq \omega \leq 1.4 \\
1 & 1.8 \leq \omega \leq \frac{\omega_3}{2} 
\end{cases} \]

where \( \omega_3 = 2\pi \text{ rad/sec} \) and \( T = 1 \text{ sec} \). In filter order estimation, \( R_p = 1 \text{dB}, \) \( R_s = 40 \text{dB} \). The coefficient wordlength is restricted to be 16 bits, and the maximum number of non-zero digits is restricted to be 4. The transfer function is of the form:

\[ H(z) = \frac{a_{01} + a_{11}z^{-1}}{1 + b_{11}z^{-1}} \prod_{i=2}^{4} \frac{a_{0i} + a_{ii}z^{-1} + a_{2i}z^{-2}}{1 + b_{ii}z^{-1} + b_{2i}z^{-2}} \]

Table 4.6 shows the canonical signed-digit coefficients of the designed filter. Fig. 4.15 shows its zeros and poles location and Fig. 4.16 shows the corresponding magnitude response.
<table>
<thead>
<tr>
<th>Coefficients</th>
<th>CSD Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_{01}$</td>
<td>$-2^0 + 2^{-2} + 2^{-5} + 2^{-7}$</td>
</tr>
<tr>
<td>$a_{11}$</td>
<td>$2^{-1} + 2^{-3} + 2^{-6} + 2^{-9}$</td>
</tr>
<tr>
<td>$b_{11}$</td>
<td>$-2^{-2} + 2^{-4} - 2^{-6} - 2^{-8}$</td>
</tr>
<tr>
<td>$a_{02}$</td>
<td>$2^0 - 2^{-2} + 2^{-5} + 2^{-7}$</td>
</tr>
<tr>
<td>$a_{12}$</td>
<td>$-2^{-1} + 2^{-3} - 2^{-7} + 2^{-9}$</td>
</tr>
<tr>
<td>$a_{22}$</td>
<td>$2^0 - 2^{-2} + 2^{-5} + 2^{-7}$</td>
</tr>
<tr>
<td>$b_{12}$</td>
<td>$2^{-2} - 2^{-7} + 2^{-9}$</td>
</tr>
<tr>
<td>$b_{22}$</td>
<td>$2^0 - 2^{-2} + 2^{-6} - 2^{-8}$</td>
</tr>
<tr>
<td>$a_{03}$</td>
<td>$2^{-2} + 2^{-4} + 2^{-8} + 2^{10}$</td>
</tr>
<tr>
<td>$a_{13}$</td>
<td>$-2^{-1} - 2^{-4} + 2^{-8} + 2^{-10}$</td>
</tr>
<tr>
<td>$a_{23}$</td>
<td>$2^{-1} - 2^{-3} - 2^{-5} - 2^{-10}$</td>
</tr>
<tr>
<td>$b_{13}$</td>
<td>$2^{-2} - 2^{-4} + 2^{-6} + 2^{-8}$</td>
</tr>
<tr>
<td>$b_{23}$</td>
<td>$2^{-2} - 2^{-5} - 2^{-8} - 2^{-11}$</td>
</tr>
<tr>
<td>$a_{04}$</td>
<td>$2^{-2} + 2^{-5} + 2^{-9} - 2^{-11}$</td>
</tr>
<tr>
<td>$a_{14}$</td>
<td>$-2^{-2} - 2^{-4} + 2^{-6} - 2^{-8}$</td>
</tr>
<tr>
<td>$a_{24}$</td>
<td>$2^{-2} + 2^{-5} + 2^{-13} - 2^{-15}$</td>
</tr>
<tr>
<td>$b_{14}$</td>
<td>$-2^{-1} + 2^{-3} - 2^{-8} - 2^{-11}$</td>
</tr>
<tr>
<td>$b_{24}$</td>
<td>$2^{-3} + 2^{-6} - 2^{-9} - 2^{-11}$</td>
</tr>
</tbody>
</table>

Table 4.6 CSD coefficients of the 7th order highpass filter
Fig. 4.15 Zeros and poles location of the 7th order highpass filter
Fig. 4.16 Magnitude response of the 7th order highpass filter
4.6.2 Design of 1-D IIR Filters with Constant Group Delay Characteristic

Genetic algorithm is a powerful optimization technique and has the ability of multi-objective optimization. In this section, we will explore this feature to optimize both the magnitude response and the group delay response of IIR filters. For the convenience of comparison, Example 1 of Section 4.6.1.1 and Example 2 of Section 4.6.1.2 have been re-designed in this section.

Example 1

The 4^{th} order lowpass IIR filter of Section 4.6.1.1 with the following specification:

\[ |H(e^{j\omega})| = \begin{cases} 
1 & 0 \leq \omega \leq 1.0 \\
0 & 2.0 \leq \omega \leq \frac{\omega_c}{2}
\end{cases} \]

where \( \omega_c = 2\pi \text{ rad/sec} \) and \( T = 1 \text{ sec} \). The coefficient wordlength is restricted to be 16 bits, and the maximum number of non-zero digits is restricted to be 4. The transfer function is of the form:

\[ H(z) = \prod_{i=1}^{2} \frac{a_{a_i} + a_{b_i}z^{-1} + a_{c_i}z^{-2}}{1 + b_{a_i}z^{-1} + b_{b_i}z^{-2}} \]

As we have introduced in Section 4.4, Eq. (4.5) is used as the error function. In this example \( \tau \) is set to equal to 4, and \( \alpha \) is chosen to be 0.7. Table 4.7 shows the canonical signed-digit coefficients of the designed filter. Fig. 4.17 shows its zeros and poles location and Fig. 4.18 shows the corresponding magnitude response. Fig. 4.19 shows the group delay response of the filter. It can be seen that the group delay is constant in the passband. It can be noticed that in the former design, the stopband attenuation can reach
45dB. But in this example, the $R_s$ is above 25dB. Just as the authors have pointed out in [43], there is a tradeoff between the magnitude and group delay errors.

<table>
<thead>
<tr>
<th>Coefficients</th>
<th>CSD Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_{01}$</td>
<td>$2^{-3} - 2^{-7} - 2^{-10} + 2^{-12}$</td>
</tr>
<tr>
<td>$a_{11}$</td>
<td>$-2^{-5} + 2^{-8} + 2^{-11}$</td>
</tr>
<tr>
<td>$a_{21}$</td>
<td>$-2^{-1} - 2^{-4} - 2^{-6}$</td>
</tr>
<tr>
<td>$b_{11}$</td>
<td>$-2^{-1} + 2^{-3} - 2^{-5} + 2^{-10}$</td>
</tr>
<tr>
<td>$b_{21}$</td>
<td>$2^{-1} - 2^{-3} + 2^{-5} - 2^{-7}$</td>
</tr>
<tr>
<td>$a_{02}$</td>
<td>$-2^{-2} + 2^{-4} - 2^{-6} - 2^{-9}$</td>
</tr>
<tr>
<td>$a_{12}$</td>
<td>$2^{-2} - 2^{-4} - 2^{-6} + 2^{-8}$</td>
</tr>
<tr>
<td>$a_{22}$</td>
<td>$-2^{-2} + 2^{-7} + 2^{-10} - 2^{-12}$</td>
</tr>
<tr>
<td>$b_{12}$</td>
<td>$-2^{0} + 2^{-3} + 2^{-6} + 2^{-8}$</td>
</tr>
<tr>
<td>$b_{22}$</td>
<td>$2^{-2} - 2^{-9} - 2^{-13} + 2^{-15}$</td>
</tr>
</tbody>
</table>

Table 4.7 CSD coefficients of the 4th order lowpass filter
Fig.4.17 Zeros and poles location of the 4th order lowpass filter
Fig. 4.18 Magnitude response of the 4\textsuperscript{th} order lowpass filter
Fig. 4.19 Group delay response of the 4th order lowpass filter
Example 2

The 8th order bandpass IIR filter of Section 4.6.1.2 with the following specification:

\[
|H(e^{j\omega T})| = \begin{cases} 
0 & 0 \leq \omega \leq 0.7 \\
1 & 1.2 \leq \omega \leq 1.9 \\
0 & 2.4 \leq \omega \leq \frac{\omega_s}{2}
\end{cases}
\]

where \(\omega_s = 2\pi \text{ rad/sec}\) and \(T = 1\) sec. The coefficient wordlength is restricted to be 16 bits, and the maximum number of non-zero digits is restricted to be 4. The transfer function is of the form:

\[
H(z) = \prod_{i=1}^{4} \frac{a_{0i} + a_{1i}z^{-1} + a_{2i}z^{-2}}{1 + b_{0i}z^{-1} + b_{2i}z^{-2}}
\]

In this example \(\tau\) is set to equal to 8, and \(\alpha\) is chosen to be 0.7. Table 4.8 shows the canonical signed-digit coefficients of the designed filter. Fig. 4.20 shows its zeros and poles location and Fig. 4.21 shows the corresponding magnitude response. Fig. 4.22 shows the group delay response of the filter.
<table>
<thead>
<tr>
<th>Coefficients</th>
<th>CSD Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_{01}$</td>
<td>$2^{-1} + 2^{-6} + 2^{-10} - 2^{-13}$</td>
</tr>
<tr>
<td>$a_{11}$</td>
<td>$-2^0 + 2^{-3} + 2^{-6} - 2^{-8}$</td>
</tr>
<tr>
<td>$a_{21}$</td>
<td>$2^{-1} - 2^{-11} - 2^{-13}$</td>
</tr>
<tr>
<td>$b_{11}$</td>
<td>$2^{-1} + 2^{-5} - 2^{-7} + 2^{-9}$</td>
</tr>
<tr>
<td>$b_{21}$</td>
<td>$2^{-1} - 2^{-3} - 2^{-6} + 2^{-11}$</td>
</tr>
<tr>
<td>$a_{02}$</td>
<td>$2^{-1} + 2^{-4} + 2^{-8} + 2^{-10}$</td>
</tr>
<tr>
<td>$a_{12}$</td>
<td>$-2^{-8} - 2^{-10}$</td>
</tr>
<tr>
<td>$a_{22}$</td>
<td>$-2^{-1} - 2^{-5} + 2^{-8} + 2^{-10}$</td>
</tr>
<tr>
<td>$b_{12}$</td>
<td>$2^{-1} + 2^{-6} - 2^{-8} - 2^{-11}$</td>
</tr>
<tr>
<td>$b_{22}$</td>
<td>$2^{-1} + 2^{-7} + 2^{-15}$</td>
</tr>
<tr>
<td>$a_{03}$</td>
<td>$2^{-1} + 2^{-3} - 2^{-5} - 2^{-7}$</td>
</tr>
<tr>
<td>$a_{13}$</td>
<td>$-2^{-5} - 2^{-9} + 2^{-11} + 2^{-13}$</td>
</tr>
<tr>
<td>$a_{23}$</td>
<td>$2^0 + 2^{-2} + 2^{-4} + 2^{-6}$</td>
</tr>
<tr>
<td>$b_{13}$</td>
<td>$-2^{-1} + 2^{-3} + 2^{-5} + 2^{-7}$</td>
</tr>
<tr>
<td>$b_{23}$</td>
<td>$2^{-1} - 2^{-4} - 2^{-6} - 2^{-8}$</td>
</tr>
<tr>
<td>$a_{04}$</td>
<td>$2^{-1} + 2^{-5} + 2^{-8} - 2^{-13}$</td>
</tr>
<tr>
<td>$a_{14}$</td>
<td>$2^{-1} - 2^{-5} + 2^{-8} + 2^{-12}$</td>
</tr>
<tr>
<td>$a_{24}$</td>
<td>$2^{-2} + 2^{-5} - 2^{-8} - 2^{-11}$</td>
</tr>
<tr>
<td>$b_{14}$</td>
<td>$-2^0 + 2^{-2} + 2^{-6}$</td>
</tr>
<tr>
<td>$b_{24}$</td>
<td>$2^{-1} + 2^{-4} - 2^{-15}$</td>
</tr>
</tbody>
</table>

Table 4.8 CSD coefficients of the 8th order bandpass filter
Fig. 4.20 Zeros and poles location of the 8th order bandpass filter.
Fig. 4.21 Magnitude response of the 8th order bandpass filter
Fig. 4.22 Group delay response of the 8th order bandpass filter
CHAPTER 5. DESIGN OF DOUBLY COMPLEMENTARY FILTER PAIRS

Complementary filters find wide applications in signal processing systems where different frequency bands are to be processed separately to measure signal strengths in each band or to achieve compression or noise reduction. In this chapter, we will design doubly complementary filter pairs by the parallel of two allpass filters. The modified GA-based design flow proposed in Chapter 4 will produce filter pairs with canonical signed-digit coefficients. We begin the discussion with a brief introduction of allpass transfer function.

5.1 Allpass Transfer Function and Its Properties

5.1.1 Allpass Transfer Function

The digital allpass filter is a very useful building block in digital signal processing ([38], [39]). A typical application is that it is often used as a delay equalizer. Another important application is in the efficient implementation of a set of transfer functions satisfying certain complementary properties.

An IIR transfer function \( A(z) \) with unity magnitude response for all frequencies, i.e.

\[
|A(e^{j\omega})|^2 = 1, \quad \text{for all } \omega
\]

(5.1)

is called an allpass transfer function. An \( M^{th} \) order causal real coefficient allpass transfer function has the form of
\[ A_M(z) = \frac{d_M + d_{M-1}z^{-1} + \cdots + d_{M-M+1}z^{-M} + z^{-M}}{1 + d_1z^{-1} + \cdots + d_{M-1}z^{-M+1} + d_Mz^{-M}} \]  

(5.2)

\[ A_M(z) \text{ can also be written as} \]

\[ A_M(z) = \frac{z^{-M}D_M(z^{-1})}{D_M(z)} \]  

(5.3)

where \( D_M(z) = 1 + d_1z^{-1} + \cdots + d_{M-1}z^{-M+1} + d_Mz^{-M} \) is the denominator polynomial of the allpass transfer function \( A_M(z) \). In effect, the numerator polynomial can be obtained from the denominator polynomial by reversing the order of the coefficients. For example, for a second order allpass transfer function, it follows

\[ A(z) = \frac{d_2 + d_1z^{-1} + z^{-2}}{1 + d_1z^{-1}d_2z^{-2}} \]  

(5.4)

In this case, the numerator and denominator polynomials are said to form a mirror-image pair, and the poles and zeros of a real coefficient allpass function exhibit mirror-image symmetry in the \( z \)-plane.

### 5.1.2 Properties

A causal stable allpass function have three very useful and important properties ([38], [39]).

**Property 1.** A causal stable real coefficient allpass transfer function is a lossless bounded real (LBR) transfer function, i.e.

\[ \sum_{n=-\infty}^{\infty} \left| y(n) \right|^2 = \sum_{n=-\infty}^{\infty} \left| u(n) \right|^2 \]  

(5.5)
where the two sides of Eq. (5.5) represent the output energy and input energy of the
digital filter respectively. Thus an allpass filter is a lossless structure since the output
energy equals the input energy for all finite energy inputs.

**Property 2.** The second property is concerned with the magnitude of a stable allpass
function \( A(z) \), i.e.

\[
|A(z)| = \begin{cases} 
< 1 & \text{for } |z| > 1 \\
= 1 & \text{for } |z| = 1 \\
> 1 & \text{for } |z| < 1
\end{cases}
\]  

(5.6)

**Property 3.** The last property is with regard to the change in phase for a real stable allpass
function over the frequency range \( \omega = 0 \) to \( \omega = \pi \). Let \( \tau(\omega) \) denotes the group delay function
of an allpass filter \( A(z) \), i.e.

\[
\tau(\omega) = -\frac{d}{d\omega} \arg(e^{i\omega})
\]

where \( \theta(\omega) \) is the unwrapped form of the phase response. Then for an \( M^{th} \) order stable
real allpass transfer function, it follows

\[
\int_0^\pi \tau(\omega)d\omega = M\pi
\]  

(5.7)

Or in the other words, the change in the phase of an \( M^{th} \) order allpass function as \( \omega \) goes
from 0 to \( \pi \) is \( M\pi \) radians.

**5.2 Doubly Complementary Filter Pair**

A set of digital transfer functions with certain complementary characteristics are often
useful in practice, such as efficient realization of transfer functions, low sensitivity
realization filter bank design, etc. Those complementary characteristics include delay
complementary, allpass complementary, power complementary and magnitude complementary ([38]).

5.2.1 Allpass Complementary and Power Complementary

A set of \( M \) digital filters \( \{H_i(z)\}, 0 \leq i \leq M - 1 \) are defined to be allpass complementary of each other, if the sum of their transfer functions is equal to an allpass function \( A(z) \), i.e.

\[
\sum_{i=0}^{M-1} H_i(z) = A(z), \quad \text{for all } \omega.
\]  

(5.8)

A set of \( M \) digital filters \( \{H_i(z)\}, 0 \leq i \leq M - 1 \) are defined to be power complementary of each other, if the sum of the squares of their magnitude responses is equal to one, i.e.

\[
\sum_{i=0}^{M-1} |H_i(e^{j\omega})|^2 = 1, \quad \text{for all } \omega.
\]  

(5.9)

5.2.2 Doubly Complementary Pair

A pair of transfer functions that satisfies both allpass complementary and power complementary properties is called a doubly complementary filter pair, i.e.

\[
|H_1(e^{j\omega}) + H_2(e^{j\omega})| = 1, \quad \text{for all } \omega.
\]  

(5.10)

\[
|H_1(e^{j\omega})|^2 + |H_2(e^{j\omega})|^2 = 1, \quad \text{for all } \omega.
\]  

(5.11)

Now let us consider two transfer functions described by

\[
H_1(z) = \frac{1}{2} \left[ A_0(z) + A_1(z) \right]
\]

\[
H_2(z) = \frac{1}{2} \left[ A_0(z) - A_1(z) \right]
\]  

(5.12)
where $A_0(z)$ and $A_1(z)$ are stable allpass transfer functions. It follows from the above definition that the sum of the two transfer functions $H_1(z)$ and $H_2(z)$ is an allpass transfer function, i.e. $A_0(z)$. So $H_1(z)$ and $H_2(z)$ is an allpass complementary pair. It can be easily shown that the two transfer functions also form a power complementary pair. Therefore, transfer functions described by Eq. (5.12) is a doubly complementary filter pair.

A doubly complementary pair can be easily realized in the form of the sum and difference of two allpass filters, as shown in Fig. 5.1.

![Diagram](image)

**Fig. 5.1 Implementation of the doubly complementary filter pair**

Here we have:

\[
H_1(z) = \frac{Y_1(z)}{X(z)}, \quad H_2(z) = \frac{Y_2(z)}{X(z)} \tag{5.13}
\]

$H_1(z)$ is the sum of two allpass functions, and the passband occurs at frequencies where the two allpass functions are in phase. $H_2(z)$ is the difference of the two allpass filters, and in accordance with Eq. (5.11), $H_2(e^{io})$ has a stopband where $H_1(e^{io})$ has a passband, and vice versa.

### 5.2.3 Design Considerations

Let $A_1(z)$ be an $M1^{th}$ order allpass function and $A_2(z)$ be an $M2^{th}$ order allpass function, and in phasor notation let
\[ A_1(e^{j\omega}) = e^{j\theta_1(\omega)} \]
\[ A_2(e^{j\omega}) = e^{j\theta_2(\omega)} \]  \hfill (5.14)

where \( \theta_1(\omega) \) and \( \theta_2(\omega) \) are monotonically decreasing functions of \( \omega \), with \( \theta_1(0) = \theta_2(0) = 0 \), \( \theta_1(\pi) = -M1\pi \), \( \theta_2(\pi) = -M2\pi \). From Eq. (5.12), we obtain

\[ |H_1(e^{j\omega})| = \frac{1}{2} |e^{j[\theta_1(\omega) - \theta_2(\omega)]} + 1| \]
\[ |H_2(e^{j\omega})| = \frac{1}{2} |e^{j[\theta_1(\omega) - \theta_2(\omega)]} - 1| \]  \hfill (5.15)

In particular, at \( \omega = \pi \)

\[ |H_1(e^{j\omega})| = \frac{1}{2} |e^{j[M1-M2]\pi} + 1| \]
\[ |H_2(e^{j\omega})| = \frac{1}{2} |e^{j[M1-M2]\pi} - 1| \]  \hfill (5.16)

Eq. (5.16) shows that a lowpass-highpass complementary pair requires

\[ M1 - M2 = 2m \pm 1, \quad m = \text{integer} \]  \hfill (5.17)

Worthy of mention is that when \( m \) is nonzero, this leads to multiple passbands. In this thesis, we only consider the case of single passband. So to design a lowpass-highpass pair, we need to have

\[ M1 - M2 = \pm 1 \]  \hfill (5.18)

From Eq. (5.12), it is clear that \( H_1(z) \) and \( H_2(z) \) will have the poles of both \( A_1(z) \) and \( A_2(z) \). So their order will be \( M1+M2 \). Therefore \( M1+M2 \) must be an odd integer if a lowpass-highpass pair is to be designed.

Similarly, for a bandpass-bandstop pair, we need to have

\[ M1 - M2 = \pm 2 \]  \hfill (5.19)

\( M1+M2 \) must be an even integer if a bandpass-bandstop pair is to be designed.
5.2.4 Advantages of Parallel Allpass Realization

The parallel allpass realization possesses a number of very attractive properties from the implementation point of view.

First, parallel allpass realization is the most computationally efficient structure in realizing IIR filters. By exploiting the mirror-image symmetry relation between the numerator and denominator polynomials of an allpass transfer function, it is possible to realize an arbitrary $M^{th}$ order allpass transfer function using only $M$ multiplications. Therefore, the realization of an $N^{th}$ order IIR filter based on the allpass decomposition of Eq. (5.12) requires only $N$ multipliers. While a direct form realization of an $N^{th}$ order IIR filter requires $2N+1$ multipliers.

Second, as indicated in Fig. 5.1, just by change a sign, we can easily implement the power complementary filter as well, also of $N^{th}$ order. Therefore two IIR filters can be realized at the cost of only one IIR filter.

Third, this realization is also famous for its low passband sensitivity property ([38], [39]). From Eq. (5.12), it is clear that $|H_i(e^{j\omega})|$ can never exceed unity for any value of $\omega$ because the allpass filters remain allpass property regardless of parameter quantization due to the mirror-image relation. So the magnitude response $|H_i(e^{j\omega})|$ is structurally bounded by one. Suppose that $H_i(z)$ is designed such that at specific frequencies $\omega_k$, the passband amplitude achieves the upper bound of unity. Regardless of the sign of any multiplier perturbation the magnitude of the transfer function can only decrease. We can
thus apply Orchard's argument ([40]) at these frequencies to establish the low passband sensitivity behavior.

Because of these advantages, during the past decades, parallel allpass realization has received wide attention and has been used to design IIR filters or complementary filter pairs with infinite precision coefficients ([41]–[51]). But the method of using genetic algorithm to design complementary filters with CSD coefficients to our knowledge has not yet been reported. So in the following sections we will consider this problem. Thus by combining the advantages of parallel allpass structure and that of CSD coefficients, IIR filter pairs that have lower implementation cost and faster response time can be designed.

5.3 Design Problem Formulation

The design method here involves the application of the GA-based optimization technique presented in Chapter 4 to search for coefficients of two allpass transfer functions such that a filter pair that satisfy doubly complementary property can be obtained.

Let $A_0(z)$ and $A_f(z)$ represent two allpass transfer functions, $H_1(z)$ and $H_2(z)$ represent the pair of filters to be designed. For stability consideration, $A_0(z)$ and $A_f(z)$ are realized by the cascade of first order or second order sections, i.e.

$$A(z) = \frac{d_{01} + z^{-1} \prod_{i=2}^{K} d_{0i} + d_{1}z^{-1} + z^{-2}}{1 + d_{01}z^{-1} + d_{1}z^{-1} + d_{0i}z^{-2}}$$  \hspace{0.5cm} (5.20)
where \(d_{ll}\) and \(d_{oi}\) satisfy the constraint of Stability Triangle. In this thesis, two types of doubly complementary filter pairs will be designed.

1. Lowpass-highpass doubly complementary filter pairs

As introduced in the Section 5.2.3, to design a lowpass-highpass doubly complementary filter pair, we need to have

\[
M1 - M2 = \pm 1 \quad \& \quad M1 + M2 = \text{odd integer}
\]

where \(M1\) and \(M2\) are the orders of \(A_0(z)\) and \(A_1(z)\) respectively. In this case, the lowpass filter is of the form:

\[
H_1(z) = \frac{1}{2} \left[ A_0(z) + A_1(z) \right] \tag{5.21}
\]

and the highpass filter is of the form:

\[
H_2(z) = \frac{1}{2} \left[ A_0(z) - A_1(z) \right] \tag{5.22}
\]

In the course of optimization, we use least square error criterion as the error function. The least square error criterion is used in the following relationship:

\[
E_{l_1}(j\omega_m) = \sum_{m \in I_{ps}} E_{Mag}^2(j\omega_m) \tag{5.23}
\]

and \(E_{Mag}(j\omega_m) = |H_1(\exp(j\omega_m T))| - |H_1(\exp(j\omega_m T))|

where \(|H_1(e^{j\omega T})|\) denotes the desired magnitude response.

\(|H_1(e^{j\omega T})|\) denotes the magnitude response of the designed lowpass filter of the form Eq. (5.21).

\(I_{ps}\) is the set of all discrete frequency points along \(\omega\) axis in the passband and stopband.
2. Bandpass-bandstop doubly complementary filter pairs

To design a bandpass-bandstop doubly complementary filter pair, we need to have

\[ M_1 - M_2 = \pm 2 \quad \& \quad M_1 + M_2 = \text{even} \quad \text{integer} \]

where \( M_1 \) and \( M_2 \) are the orders of \( A_0(z) \) and \( A_1(z) \) respectively. In this case, the bandpass filter is of the form:

\[ H_1(z) = \frac{1}{2} [A_0(z) - A_1(z)] \quad (5.24) \]

and the bandstop filter is of the form:

\[ H_2(z) = \frac{1}{2} [A_0(z) + A_1(z)] \quad (5.25) \]

The least square error criterion is used in the following relationship:

\[ E_{j\omega_m} = \sum_{m \in I_{ps}} E_{\text{Mag}} (j\omega_m) \quad (5.26) \]

and \( E_{\text{Mag}} (j\omega_m) = |H_1(\exp(j\omega_m T)) - |H_1(\exp(j\omega_m T))| \)

where \( |H_1(e^{j\omega T})| \) denotes the desired magnitude response.

\[ |H_1(e^{j\omega T})| \] denotes the magnitude response of the designed bandpass filter of the form Eq. (5.24).

\( I_{ps} \) is the set of all discrete frequency points along \( \omega \) axis in the passband and stopband.

Based on the above formulation, the modified GA presented in Chapter 4 is used to design lowpass-highpass doubly complementary filter pairs and bandpass-bandstop doubly complementary filter pairs.
5.4 Design Examples

Based on the above formulation, the modified GA presented in Chapter 4 has been used to design lowpass-highpass doubly complementary filter pairs and bandpass-bandstop doubly complementary filter pairs. The designed filters’ coefficients, zeros and poles location, their magnitude response and group delay response are shown in this section.

5.4.1 Design of Lowpass-Highpass Pairs

A lowpass-highpass pair with the following lowpass filter specification:

\[ |H_1(e^{j\omega})| = \begin{cases} 
1 & 0 \leq \omega \leq 0.52\pi \\
0 & 0.58\pi \leq \omega \leq \frac{0.5}{2} 
\end{cases} \]

where \( \omega = 2\pi \text{ rad/sec} \) and \( T = 1 \text{ sec} \). The order of \( H_1(z) \) and \( H_2(z) \) is chosen to be 11, thus \( A_0(z) \) and \( A_1(z) \) is 6th order and 5th order allpass filter respectively. Filter coefficient wordlength is restricted to be 16 bits, and the maximum number of non-zero digits is restricted to be 4. The transfer function of the two allpass filters are in the form of:

\[ A_0(z) = \prod_{i=1}^{3} \frac{d_{0i} + d_{1i}z^{-1} + z^{-2}}{1 + d_{1i}z^{-1} + d_{0i}z^{-2}} \]

\[ A_1(z) = \frac{d_{01} + z^{-1}}{1 + d_{01}z^{-1}} \prod_{i=2}^{3} \frac{d_{0i} + d_{1i}z^{-1} + z^{-2}}{1 + d_{1i}z^{-1} + d_{0i}z^{-2}} \]

Table 5.1 shows the canonical signed-digit coefficients of the two allpass filters. Fig. 5.2 shows the zeros and poles location of the lowpass filter \( H_1(z) \). It can be seen that \( H_1(z) \) is stable. Since \( H_2(z) \) and \( H_1(z) \) have the same poles, \( H_2(z) \) is also stable. Fig. 5.3 shows the corresponding magnitude response of the filter pair. Fig. 5.4 shows the sum of the filter pair, from which it is clear that they satisfy allpass complementary. Fig. 5.5 shows the
sum of power of the filter pair, which indicates that the filter pair is also power complementary.

<table>
<thead>
<tr>
<th>Coefficients</th>
<th>CSD Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \bar{d}_{01} )</td>
<td>( 2^{-3} + 2^{-5} + 2^{-7} - 2^{-10} )</td>
</tr>
<tr>
<td>( \bar{d}_{11} )</td>
<td>( 2^0 - 2^{-3} + 2^{-5} - 2^{-7} )</td>
</tr>
<tr>
<td>( \bar{d}_{02} )</td>
<td>( -2^{-1} + 2^{-5} + 2^{-7} + 2^{-9} )</td>
</tr>
<tr>
<td>( \bar{d}_{12} )</td>
<td>( 2^{-1} + 2^{-5} - 2^{-7} + 2^{-10} )</td>
</tr>
<tr>
<td>( \bar{d}_{03} )</td>
<td>( -2^{-1} + 2^{-3} + 2^{-6} )</td>
</tr>
<tr>
<td>( \bar{d}_{13} )</td>
<td>( 2^{-3} + 2^{-5} - 2^{-9} + 2^{-11} )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Coefficients</th>
<th>CSD Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \bar{d}_{01} )</td>
<td>( -2^{-3} + 2^{-5} - 2^{-9} )</td>
</tr>
<tr>
<td>( \bar{d}_{02} )</td>
<td>( -2^{-1} - 2^{-3} - 2^{-6} - 2^{-9} )</td>
</tr>
<tr>
<td>( \bar{d}_{12} )</td>
<td>( 2^{-1} - 2^{-4} + 2^{-7} - 2^{-9} )</td>
</tr>
<tr>
<td>( \bar{d}_{03} )</td>
<td>( -2^{-6} + 2^{-12} )</td>
</tr>
<tr>
<td>( \bar{d}_{13} )</td>
<td>( 2^{-1} + 2^{-3} + 2^{-5} + 2^{-7} )</td>
</tr>
</tbody>
</table>

Table 5.1 CSD coefficients of the two allpass filters
Fig.5.2 Zeros and poles location of the lowpass filter
Fig. 5.3 Magnitude response of the lowpass-highpass filter pair
Fig. 5.4 Sum of the lowpass-highpass filter pair
Fig. 5.5 Sum of power of the lowpass-highpass filter pair
5.4.2 Design of Bandpass-Bandstop Pairs

A bandpass-bandstop pair with the following bandpass filter specification:

\[
\left| H_1(e^{j\omega T}) \right| = \begin{cases} 
0 & 0 \leq \omega \leq 0.2\pi \\
1 & 0.3\pi \leq \omega \leq 0.6\pi \\
0 & 0.7\pi \leq \omega \leq \frac{\omega_T}{2}
\end{cases}
\]

where \(\omega_T = 2\pi \text{ rad/sec}\) and \(T = 1\) sec. The order of \(H_1(z)\) and \(H_2(z)\) is chosen to be 10, thus \(A_0(z)\) and \(A_1(z)\) is 6th order and 4th order allpass filter respectively. Filter coefficient wordlength is restricted to be 16 bits, and the maximum number of non-zero digits is restricted to be 4. The transfer function of the two allpass filters are in the form of:

\[
A_0(z) = \prod_{i=1}^{3} \frac{d_{0i} + d_{1i}z^{-1} + z^{-2}}{1 + d_{1i}z^{-1} + d_{0i}z^{-2}}
\]

\[
A_1(z) = \prod_{i=1}^{2} \frac{d_{0i} + d_{1i}z^{-1} + z^{-2}}{1 + d_{1i}z^{-1} + d_{0i}z^{-2}}
\]

Table 5.2 shows the canonical signed-digit coefficients of the two allpass filters. Fig. 5.6 shows the zeros and poles location of the bandstop filter. It can be seen that it is stable, so the bandpass filter is also stable. Fig. 5.7 shows the corresponding magnitude response of the pair. Fig. 5.8 shows the sum of the filter pair, from which it is clear that they satisfy allpass complementary. Fig. 5.9 shows the sum of power of the filter pair, which indicates that the filter pair is also power complementary.
<table>
<thead>
<tr>
<th>Coefficients</th>
<th>CSD Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>$d_{01}$</td>
<td>$2^{-1} + 2^{-3} - 2^{-7} + 2^{12}$</td>
</tr>
<tr>
<td>$d_{11}$</td>
<td>$2^0 - 2^{-3} - 2^{-6} - 2^{-8}$</td>
</tr>
<tr>
<td>$d_{02}$</td>
<td>$-2^{-2} - 2^{-4} - 2^{-6} - 2^{-8}$</td>
</tr>
<tr>
<td>$d_{12}$</td>
<td>$2^{-1} - 2^{-3} + 2^{-5}$</td>
</tr>
<tr>
<td>$d_{03}$</td>
<td>$-2^0 + 2^{-2} + 2^{-10} + 2^{-15}$</td>
</tr>
<tr>
<td>$d_{13}$</td>
<td>$2^0 - 2^{-3} + 2^{-7} - 2^{-9}$</td>
</tr>
</tbody>
</table>

### Table 5.2 CSD coefficients of the two allpass filters

![Graph showing zeros and poles location of the bandstop filter](image)

Fig. 5.6 Zeros and poles location of the bandstop filter
Fig. 5.7 Magnitude response of the bandpass-bandstop filter pair
Fig. 5.8 Sum of the bandpass-bandstop filter pair
Fig. 5.9 Sum of the power of the bandpass-bandstop filter pair
CHAPTER 6. DESIGN OF 2-D IIR DIGITAL FILTERS

Similar to 1-D digital filter design, the implementation of 2-D digital filters with infinite-precision coefficients introduces quantization procedure, which may have undesirable effects on the designed filter's behavior. Design of 2-D filters with discrete coefficients has the advantage of eliminating this problem. In this chapter, a GA-based optimization technique will be presented for the design of 2-D digital filters with canonical signed-digit coefficients. The utilization of CSD coefficients makes the designed filters more efficient for high throughput rate applications. Examples will be provided to show the efficiency of the proposed technique.

6.1 Design 2-D IIR Filters Using Separable Denominator

The approximation of 2-D IIR digital filters can be carried out in digital ($z$) domain by using any suitable optimization techniques, but the difficulties associated with this approximation are:

- Stability
- Extensive amount of computational cost required

We have shown in Section 1.2.2 that the stability problem can be simplified if non-separable numerator and separable denominator transfer function is used. It will be shown later that the computational cost can be decreased by constraining the designed filter to have certain symmetric property. In addition, using this sub-class of transfer
function, more conventional 2-D filters with circular symmetric frequency response can be designed.

A non-separable numerator and separable denominator transfer function has the form of ([53]):

\[ H(z_1, z_2) = \frac{\sum_{i=0}^{M_1} \sum_{j=0}^{N_1} a(i, j) z_1^{-i} z_2^{-j}}{\sum_{i=0}^{M_2} b_1(i) z_1^{-i} \sum_{j=0}^{N_2} b_2(j) z_2^{-j}} = \frac{A(z_1, z_2)}{B_1(z_1) B_2(z_2)} \] (6.1)

where \( z_i = e^{\omega_i T}, i=1,2 \). Without any loss of generality, we can assume

\[ M_1 = M_2 = N_1 = N_2 = N \]

The transfer function of Eq. (6.1) represents a central symmetry. This means that the magnitude response of the 2-D filter is the same in the 1\(^{st}\) and 3\(^{rd}\) quadrants, and the same in the 2\(^{nd}\) and 4\(^{th}\) quadrants. So in the design procedure, the approximation has to be carried out in the 1\(^{st}\) and 2\(^{nd}\) quadrants, as shown in Fig. 6.1.

![Fig. 6.1 Computational cost of a 2-D filter with central symmetry](image-url)
A quadrantally symmetric can be obtained if the following constraints are imposed on the transfer function defined in Eq. (6.1) ([52]).

\[ a_{ij} = a_{N-i,j} = a_{i,N-j} = a_{N-i,N-j} \quad (6.2) \]

\[ b_{ij} = b_{2j} \quad (6.3) \]

So Eq. (6.1) can be written as

\[
H(z_1, z_2) = \frac{z_1^{-N/2} z_2^{-N/2} \sum_{i=0}^{N/2} \sum_{j=0}^{N/2} a^{'}(i,j) \cos(i \omega_1 T) \cos(j \omega_2 T)}{\left( \sum_{i=0}^{N} b_1 z_1^{-i} \right) \left( \sum_{j=0}^{N} b_2 z_2^{-j} \right)} = \frac{A(z_1, z_2)}{B_1(z_1) B_2(z_2)} \quad (6.4)
\]

In this case, the magnitude response of the 2-D filter is the same in all four quadrants, and the computational cost can be reduced by half, as shown in Fig. (6.2).

![Fig. 6.2 Computational cost of a 2-D filter with quadrantral symmetry](image)

If in addition to the above constraints Eq. (6.2) and Eq. (6.3), we include an additional constraint

\[ a_{ij} = a_{ji} \quad (6.5) \]
an octagonal symmetric 2-D filter is realized. The magnitude response of this subclass of 2-D filters is similar in all octants. Computational cost can be further reduced, shown in Fig. 6.3.

![Diagram showing octagonal symmetry in 2-D frequency space](image)

**Fig. 6.3** Computational cost of a 2-D filter with octagonal symmetry

Among the many advantages of non-separable numerator and separable denominator transfer functions, the most attractive one is that the stability problem of 2-D filters can be reduced to that of 1-D filters, i.e., the stability of the two 1-D polynomials of $z_1$ and $z_2$ in the denominator. So the stability of the designed 2-D filter can be guaranteed using any of the techniques used in 1-D filter design, as presented in [8], [10] and [11].

In this thesis, the two 1-D all pole filters of the denominator, i.e. $B_1(z_1)$ and $B_2(z_2)$, are designed using cascade structure. $B_1(z_1)$ and $B_2(z_2)$ has the form of

$$B_i(z_i) = \prod_{k=1}^{N/2} \left(1 + b_{ik}z_i^{-1} + b_{2k}z_i^{-2}\right), \; i=1,2$$  \hspace{1cm} (6.6)

The stability of $B_1(z_1)$ and $B_2(z_2)$ is guaranteed by constraining the coefficients of each second order section in the stability triangle, i.e.
\[
\begin{align*}
|b_{2k}| &< 1 \\
|b_{1k}| &< 1 + b_{2k}
\end{align*}
\] (6.7)

6.2 Design Problem Formulation

The design method used here involves the application of a GA-based optimization technique in the course of searching for coefficients of the filter transfer function such that it approximates the desired specification. The least square error criterion is used as the error function in the optimization.

Let \( |H_D(e^{j\omega_1 T}, e^{j\omega_2 T})| \) denote the magnitude response of the filter to be designed, \( |H_I(e^{j\omega_1 T}, e^{j\omega_2 T})| \) be the desired magnitude response. The least square error function is used in the following relationship:

\[
E_M(j\omega_{1m}, j\omega_{2n}) = \sum_{m,n} \sum_{\beta} \left[ |H_I(e^{j\omega_{1\beta}}, e^{j\omega_{2\beta}})| - |H_D(e^{j\omega_{1\beta}}, e^{j\omega_{2\beta}})| \right]^2
\]

where \( I_{\beta} \) is the set of all discrete frequency points along \( \omega_1 \) and \( \omega_2 \) axis in the passband and stopband.

6.3 Design Examples

To test the utility of the design method presented above, octagonal symmetric 2-D IIR filters has been designed.

Example 1

A 2\textsuperscript{nd} order octagonal symmetric lowpass filter with the following desired magnitude response:
\[ |H_i(e^{j\omega_1}, e^{j\omega_2})| = \begin{cases} 
1 & 0 \leq \sqrt{\omega_1^2 + \omega_2^2} \leq 1.0 \left( \text{rad/ sec} \right) \\
0 & 2.5 \leq \sqrt{\omega_1^2 + \omega_2^2} \leq \frac{\omega_s}{2} 
\end{cases} \]

where \( \omega_s = 10 \left( \text{rad/ sec} \right) \) and \( T = \frac{2\pi}{10} \left( \text{sec} \right) \). The designed filter’s CSD coefficients are shown in Table 6.1, and Fig. 6.4 shows its magnitude response.

<table>
<thead>
<tr>
<th>Coefficients</th>
<th>CSD Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>( a_{00} )</td>
<td>( 2^{-8} + 2^{-13} - 2^{-15} )</td>
</tr>
<tr>
<td>( a_{01} = a_{10} )</td>
<td>( 2^{-5} - 2^{-8} - 2^{-10} - 2^{-14} )</td>
</tr>
<tr>
<td>( a_{11} )</td>
<td>( 2^{-5} + 2^{-14} )</td>
</tr>
<tr>
<td>( b_1 )</td>
<td>( -2^0 - 2^{-3} - 2^{-5} - 2^{-15} )</td>
</tr>
<tr>
<td>( b_2 )</td>
<td>( 2^{-1} - 2^{-4} + 2^{-6} - 2^{-15} )</td>
</tr>
</tbody>
</table>

Table 6.1. CSD coefficients of the 2\(^{nd}\) order 2-D filter

![Magnitude Response](image)

Fig. 6.4 Magnitude Response of the 2\(^{nd}\) order 2-D filter
Example 2

A 4\textsuperscript{th} order octagonal symmetric lowpass filter with the same desired magnitude response:

\[ |H_z(e^{j\omega z}, e^{j\omega y})| = \begin{cases} 1 & 0 \leq \sqrt{\omega_1^2 + \omega_2^2} \leq 1.0 \left( \frac{rad}{sec} \right) \\ 0 & 2.5 \leq \sqrt{\omega_1^2 + \omega_2^2} \leq \frac{\omega_2}{2} \end{cases} \]

Table 6.2 is the CSD coefficients of the designed filter and Fig. 6.5 shows the corresponding magnitude response.

<table>
<thead>
<tr>
<th>Coefficients</th>
<th>CSD Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>( a_{01} )</td>
<td>( 2^{-1} - 2^{-4} + 2^{-11} )</td>
</tr>
<tr>
<td>( a_{01} = a_{10} )</td>
<td>( 2^0 - 2^{-2} - 2^{-4} + 2^{-7} )</td>
</tr>
<tr>
<td>( a_{02} = a_{20} )</td>
<td>( 2^{-2} + 2^{-6} - 2^{-10} )</td>
</tr>
<tr>
<td>( a_{11} )</td>
<td>( 2^0 + 2^{-3} - 2^{-5} - 2^{-8} )</td>
</tr>
<tr>
<td>( a_{12} = a_{21} )</td>
<td>( 2^{-1} - 2^{-4} - 2^{-6} - 2^{-12} )</td>
</tr>
<tr>
<td>( a_{22} )</td>
<td>( 2^{-3} + 2^{-5} + 2^{-8} + 2^{-10} )</td>
</tr>
<tr>
<td>( b_{11} )</td>
<td>( 2^{-2} + 2^{-4} + 2^{-6} + 2^{-10} )</td>
</tr>
<tr>
<td>( b_{12} )</td>
<td>( -2^{-3} - 2^{-5} + 2^{-7} - 2^{-14} )</td>
</tr>
<tr>
<td>( b_{21} )</td>
<td>( 2^0 - 2^{-2} + 2^{-7} + 2^{-10} )</td>
</tr>
<tr>
<td>( b_{22} )</td>
<td>( -2^{-4} - 2^{-8} - 2^{-10} + 2^{-14} )</td>
</tr>
</tbody>
</table>

Table 6.2. CSD coefficients of the 4\textsuperscript{th} order 2-D filter
Fig. 6.5 Magnitude Response of the 4th order 2-D filter
CHAPTER 7. CONCLUSION

In this thesis, we have proposed an optimization method for the design of 1-D IIR filter, doubly complementary filter pairs and 2-D IIR filters to have canonical signed-digit (CSD) coefficients. Genetic algorithm (GA) is used as the optimization technique in this method. To solve the problems arising from applying genetic operators to CSD coefficients, a new CSD number restoration technique is proposed in this thesis.

Using the presented method, 1-D IIR filters are designed to satisfy a desired frequency response. Two cases are considered here, i.e. the optimization of magnitude response only and the optimization of both magnitude response and constant group delay characteristics. The filters are designed based on cascade structure, and the stability of the designed filter is ensured by constraining the filter coefficients of the denominator polynomials to be within stability triangle.

In turn, we presented the design method for doubly complementary filter pairs. The filter pairs are designed based on parallel allpass structure, where IIR filters are realized by the parallel of two allpass sub-filters. This structure has the advantage of requiring less multipliers and possessing low passband sensitivity. Based on the same stability criterion, the stabilities of the designed filter pairs are guaranteed.
Finally we presented a design method for a class of 2-D IIR filters. The transfer function has non-separable numerator and separable denominator polynomials in $z_1$ and $z_2$ plane. Using this sub-class of transfer function, the stability problem associated with 2-D filter can be simplified to 1-D case. In the proposed method, the same stability constraints are imposed on the coefficients of the two 1-D filters of the denominator. The usefulness of the proposed method in the design of 2-D IIR filters has been well justified.

The design examples show that the proposed method is suitable to design a wide variety of digital filters. The advantage of the proposed method is that it produces digital filters with CSD coefficients, which not only eliminate the quantization step in digital filter design, but also makes the designed filter very efficient in high speed digital signal processing. It is also worthy of mention that the design technique presented in this thesis has used unconstrained optimization procedure, thus having the flexibility of designing virtually any type of filter such as multi-band filter, except from standard lowpass, bandpass or highpass filters.
REFERENCES


APPENDIX

This Appendix is the source code of the modified genetic algorithm for the design of 1-D IIR filters with or without constant group delay characteristics.

The filter specification, the desired coefficient wordleng and the maximum number of non-zero digits are specified in the micro at the beginning of the program. The arguments required to run this program is as following:

    idiiir.exe iter popsize order pmut pcross

where

    iter:        maximum iteration
    popsize:     population size
    order:       filter order
    pmut:        mutation probability
    pcross:      crossover probability
/* This program is for 1-D IIR lowpass filter design. The maximum number of SOS can be 10. */

#include<stdio.h>
#include<math.h>
#include<stdlib.h>
#include<string.h>
#include<time.h>

#define NCSDdigit 4    /* Specify desired specification and CSD requirement. */
#define Nbit 16
#define T 1.0
#define maxFreq 3.1416
#define Npoints 80
#define stopbd 2.0
#define passbd 1.0

typedef struct{
    signed int bit;
} CSDbit;

typedef struct{
    CSDbit coef[50][Nbit];
    double fitness;
} CHMtype;

typedef struct{
    double re,im;
} cnumber;

double pmut,pcross;
int Nehrom,Norder,Ncoef;
CHMtype *chrom,*offsprg,bestChrom;
FILE *outptr;
int *ordering;
double *coeff;
int *csd;

cnumber cadd(double anumber, cnumber a, cnumber b);
cnumber cadd2(double anumber, cnumber a);
cnumber cmulti(double anumber, cnumber b);
cnumber cdivid(cnumber a, cnumber b);
cnumber cexpj(double a);
double cmag(cnumber a);
unsigned int ANUMBER(void);
double UNI(void);
void presetChrom();
void ArandChrom(int loc);
int randBit(int except);
int randBit();
void GETchromfit();
void GETbestChrom();
void Elitist(int maxi, double maxF, int mini, double miniF);
double decodeChrom(int chrom_loc, int coef_loc);
double fitness(double *coeff);
double MagErr(int Nbiq, double biqCoeff[10][5]);
double GrpErr(int Nbiq, double biqCoeff[10][5]);
void reproduce();
void CMCfix();
void preloadChrom();
void chrom2offsprg(int chrom_loc, int offsprg_loc);
void reloadGene(int a, int loc1, int loc2, int coefloc);
void DOcrossover(int loc1, int loc2, int coef_loc);
int DecToCSD(double decimal, int first);
void DOMutation(int loc, int coefloc);
int CSDcheck(int loc, int geneloc);
void report();
double decodeBest(int coef_loc);

void main(int argc, char* argv[]){
    int Citer, MaxIter;
srand((unsigned)time(0));

/*Obtain the maximum iteration, population size, filter order, mutation probability and crossover probability from the command line.

if (argc!=6) {
    printf("\nUsage:");
    printf("n1dirr MaxIter Nchrom Norder pmut pcross\n");
    exit(-1);
}

MaxIter=atoi(argv[1]);
Nchrom=atoi(argv[2]);
Norder=atoi(argv[3]);
pmut=(double)atoi(argv[4])/100;
pcross=(double)atoi(argv[5])/100;

/*Calculate the number of coefficients.

if ((Norder%2)!=0) Ncoef=5*(Norder/2)+3;
else Ncoef=5*(Norder/2);

chrom=new CHMtype[Nchrom];
if(chrom==NULL){
    printf("nNot enough memory for *chrom");
    exit(-1);
}

offsprg=new CHMtype[Nchrom];
if(offsprg==NULL){
    printf("n Not enough memory for *offsprg");
    exit(-1);
}

ordering=new int[Nchrom];
if(ordering==NULL){

126
printf("\nno memory for ordering");
exit(-1);
}

coeff=new double[Ncoef];
if(coeff==NULL){
    printf("\n Not enough memory for *coeff");
    exit(-1);
}

csd=new int[Nbit];
if(csd==NULL){
    printf("\n Not enough memory for *csd");
    exit(-1);
}

presetChrom();        /* Initialization
GETchromfit();        /*Fitness evaluation
GETbestChrom();       /*Find the best chromosome of the current generation.

Citer=0;
while(Citer<MaxIter){   /*GA cycle
    Citer++;

    reproduce();        /*Roulette wheel selection

    CMCfix();           /*Crossover and mutation

    GETchromfit();

    GETbestChrom();

    printf("\n%d  \%f",Citer,bestChrom.fitness);
}
printf("\n");
report(); /* Output result to file.

delete[] chrom;
delete[] offsprg;
delete[] ordering;
delete[] coeff;
delete[] csd;
}

cnumber cadd(double anumber, cnumber a, cnumber b){
cnumber c; /* Addition of complex number

c.re=a.re+b.re+anumber;
c.im=a.im+b.im;

return(c);
}

cnumber cadd2(double anumber, cnumber a){
cnumber c;

c.re=a.re+anumber;
c.im=a.im;

return(c);
}

cnumber cmulti(double anumber, cnumber a){ /* Multiplication of complex number

cnumber c;

c.re=a.re*anumber;
c.im=a.im*anumber;

return(c);
}
cnnumber cdivid(cnnumber a, cnnumber b){  
    cnnumber c;  
    double scalar;  
    
    scalar=(b.re)*(b.re)+(b.im)*(b.im);  
    if(scalar==0) scalar=1.0E-7;  
    c.re=(a.re*b.re+a.im*b.im)/scalar;  
    c.im=(a.im*b.re-a.re*b.im)/scalar;  
    
    return(c);  
}  

cnnumber cexpj(double a){  
    /* Deomposition of z^n */  
    cnnumber c;  
    
    c.re=cos(a);  
    c.im=(-1.0)*sin(a);  
    
    return(c);  
}  

double cmag(cnnumber a){  
    /* Calculate magnitude. */  
    double c;  
    
    c=(a.re)*(a.re)+(a.im)*(a.im);  
    c=sqrt(c);  
    
    return(c);  
}  

unsigned int ANUMBER(void){  
    /* Generate a random interger. */  
    int random_integer = rand();  
    return random_integer;  
}  

double UNI(void){  
    /* Generate a random number between between 0 and 1 */  
    return rand()/RAND_MAX;  
}
return(ANUMBER()/(RAND_MAX+1.0));
}

void presetChrom(){
    /* Initialize the population. */
    int j;
    for(j=0;j<Nchrom;j++){
        ArandChrom(j);
    }
}

/* Randomly generate a chromosome with the specified CSD requirement. */
void ArandChrom(int loc){
    int i,j,n,counter;
    for(j=0;j<Ncoef;j++){
        counter=0;
        for(i=0;i<Nbit;i++){
            if(i==0){
                chrom[loc].coef[j][i].bit=randBit();
                if(chrom[loc].coef[j][i].bit!=0) counter++;
            }
            else if(counter<NCSDDigit){
                if(chrom[loc].coef[j][i-1].bit==0){
                    chrom[loc].coef[j][i].bit=randBit();
                    if(chrom[loc].coef[j][i].bit!=0) counter++;
                }
                else chrom[loc].coef[j][i].bit=0;
            }
            else{
                for(n=i;n<Nbit;n++) chrom[loc].coef[j][n].bit=0;
            }
        }
    }
}
int randBit(){
    int numb;
    numb=ANUMBER()%3;
    numb=numb-1;
    return(numb);
}

int randBit(int except){ /*Bit flip.
    int numb=except;
    while(numb==except){
        numb=ANUMBER()%3;
        numb=numb-1;
    }
    return(numb);
}

void GETchromfit(){ /*Decode chromosomes and calculate fitness.
    int j,i;
    double val;
    for(j=0;j<Nchrom;j++){
        for(i=0;i<Ncoef;i++){
            coeff[i]=decodeChrom(j,i);
        }
        val=fitness(coeff);
        chrom[j].fitness=val;
    }
}

void GETbestChrom(){ /*Find the best chromosome
    int i,j,n,indexMax,indexMin;
    double minFit,maxFit;
    static int NumlIteration=0;
}
minFit=999.9;
maxFit=-10.0;
for(n=0;n<Nchrom;n++){
    if(minFit>chrom[n].fitness){minFit=chrom[n].fitness;indexMin=n;}
    if(maxFit<chrom[n].fitness){maxFit=chrom[n].fitness;indexMax=n;}
}

if(NumIteration!=0){
    Elitist(indexMax,maxFit,indexMin,minFit);
}
else{
    NumIteration++;
    for(j=0;j<Ncoef;j++){
        for(i=0;i<Nbit;i++){
            bestChrom.coef[j][i].bit=chrom[indexMax].coef[j][i].bit;
        }
    }
    bestChrom.fitness=maxFit;
}

void Elitist(int maxi,double maxF,int mini,double minF){     /* Elitist Strategy
    int i,j;
    if(maxF>bestChrom.fitness){
        for(j=0;j<Ncoef;j++){
            for(i=0;i<Nbit;i++){
                bestChrom.coef[j][i].bit=chrom[maxi].coef[j][i].bit;
            }
        }
        bestChrom.fitness=maxF;
    }
    else if(maxF<bestChrom.fitness){
        for(j=0;j<Ncoef;j++){
            for(i=0;i<Nbit;i++){

chrom[mini].coef[j][i].bit=bestChrom.coef[j][i].bit;
}
}
chrom[mini].fitness=bestChrom.fitness;
}
else{}
}

double decodeChrom(int chrom_loc,int coef_loc){      /*Decode chromosomes */
double anumber;
int n;

anumber=0;
for(n=0;n<Nbit;n++){
    if(chrom[chrom_loc].coef[coef_loc][n].bit==1)
        anumber=anumber+pow(2.0,-1.0*(n));
    else if(chrom[chrom_loc].coef[coef_loc][n].bit==-1)
        anumber=anumber-pow(2.0,-1.0*(n));
    else{}
}
return(anumber);
}

double fitness(double *coeff){                  /*Fitness evaluation */
double sosCoeff[10][5];
int Nsos,i,j,k,d1,d2,d3;
double magerror,delayererror,return_val;

Nsos=Norder/2;
if ((Norder%2)==0){                             /*For even order */
    for (i=0;i<Nsos;i++){
        /*Find coefficients for each SOS */
        for (j=0;j<5;j++){
            k=i*5+j;
            sosCoeff[i][j]=coeff[k];
        }
    }
}
for (i=0;i<Nsos;i++) { /* Stability check
    d2=(fabs(sosCoeff[i][4])<1.0)?1.0;
    d1=(fabs(sosCoeff[i][3])<1.0+sosCoeff[i][4])?1.0;
    if((d2&d1)==0) {
        return_val=0;
        goto BACK;
    }
}

magerror=MagErr(Nsos,sosCoeff);
delayerror=GrpErr(Nsos,sosCoeff);

return_val=1/(0.7*magerror+0.3*delayerror); /* Fitness
} else { /* For odd order
    for (i=0;i<Nsos;i++) {
        for (j=0;j<5;j++) {
            k=i*5+j;
            sosCoeff[i][j]=coeff[k];
        }
    }
    for (i=0;i<3;i++) {
        k=Nsos*5+i;
        sosCoeff[Nsos][i]=coeff[k];
    }
}

for (i=0;i<Nsos;i++) {
    d2=(fabs(sosCoeff[i][4])<1.0)?1.0;
    d1=(fabs(sosCoeff[i][3])<1.0+sosCoeff[i][4])?1.0;
    if((d2&d1)==0) {
        return_val=0;
        goto BACK;
    }
}

d3=(fabs(sosCoeff[Nsos][2])<1.0)?1.0;
if(d3==0) {
return_val=0;
goto BACK;
}
magerror=MagErr(Nsos,sosCoef);
delayerror=GrpErr(Nsos,sosCoef);

return_val=1/(0.5*magerror+0.5*delayerror);
}
BACK:return(return_val);
}
double MagErr(int Nbiq,double biqCoef[10][5])
{ /*Calculate magnitude error */
    int i;
    double w,delta_w,err,magH,Hpass,Hstop;
    cnumber z1,z2,temp1,temp2,temp3,temp4,num,den,tempH;
    cnumber tempFos1,tempFos2,numFos,denFos,tempFosH;

delta_w=maxFreq/Npoints;
err=0;

for(w=0;w<maxFreq+delta_w;w=w+delta_w){
    magH=1.0;
    z1=cexpj(T*w);
    z2=cexpj(2*T*w);

    for (i=0;i<Nbiq;i++){
        temp1=cmulti(biqCoef[i][1],z1);
        temp2=cmulti(biqCoef[i][2],z2);
        temp3=cmulti(biqCoef[i][3],z1);
        temp4=cmulti(biqCoef[i][4],z2);
        num=cadd(biqCoef[i][0],temp1,temp2);
        den=cadd(1.0,temp3,temp4);
        tempH=cdivid(num,den);
        magH=magH*emag(tempH);
    }
}
if(Norder%2!=0)  /*For odd order */
    tempFos1=cmulti(biqCoeff[Nbiq][1],z1);
    tempFos2=cmulti(biqCoeff[Nbiq][2],z1);
    numFos=cadd2(biqCoeff[Nbiq][0],tempFos1);
    denFos=cadd2(1.0,tempFos2);
    tempFosH=cdivid(numFos,denFos);
    magH=magH*cmag(tempFosH);
}

Hpass=1.0-fabs(magH);
Hstop=fabs(magH);

if(w<=passbd) err=err+Hpass*Hpass;
else if(w>=stopbd) err=err+Hstop*Hstop;
else {}
}
return(err);
}

double GrpErr(int Nbiq,double biqCoeff[10][5])   /*Calculate group delay error*/
{
    int i;
    double w,delta_w,err,tempVar,numDev,num,denDev,den,delay,grpdelay;
    double numDevFos,numFos,denDevFos,denFos,delayFos;

    delta_w=maxFreq/Npoints;
    err=0;

    for(w=0;w<maxFreq+delta_w;w=w+delta_w){
        grpdelay=0;
        tempVar=cos(T*w);

        for (i=0;i<Nbiq;i++){
            numDev=biqCoeff[i][0]*biqCoeff[i][0]-biqCoeff[i][2]*biqCoeff[i][2]+biqCoeff[i][1]*(biqCoeff[i][0]-biqCoeff[i][2])*tempVar;

            numDevFos=numDev*den;
            denDev=dnumDevFos*denFos;
            den=denDevFos+numDevFos;
            numDevFos=numDevFos+numDev;
            num=denDevFos+denDev;
            den+=denFos;
            numDevFos+=denDevFos;
            denDevFos+=numDevFos;
            num+=numDevFos;
            den+=denDevFos;
            numDevFos=numDevFos+numDev;
            denDevFos=denDevFos+denDev;
            num+=numDevFos;
            den+=denDevFos;
            numDevFos=numDevFos+numDev;
            denDevFos=denDevFos+denDev;
            num+=numDevFos;
            den+=denDevFos;
            numDevFos=numDevFos+numDev;
            denDevFos=denDevFos+denDev;
            num+=numDevFos;
            den+=denDevFos;
            numDevFos=numDevFos+numDev;
            denDevFos=denDevFos+denDev;
            num+=numDevFos;
            den+=denDevFos;
            numDevFos=numDevFos+numDev;
            denDevFos=denDevFos+denDev;
            num+=numDevFos;
            den+=denDevFos;
            numDevFos=numDevFos+numDev;
            denDevFos=denDevFos+denDev;
            num+=numDevFos;
            den+=denDevFos;
            numDevFos=numDevFos+numDev;
            denDevFos=denDevFos+denDev;
            num+=numDevFos;
            den+=denDevFos;
            numDevFos=numDevFos+numDev;
            denDevFos=denDevFos+denDev;
            num+=numDevFos;
            den+=denDevFos;
            numDevFos=numDevFos+numDev;
            denDevFos=denDevFos+denDev;
            num+=numDevFos;
            den+=denDevFos;
            numDevFos=numDevFos+numDev;
            denDevFos=denDevFos+denDev;
            num+=numDevFos;
            den+=denDevFos;
            numDevFos=numDevFos+numDev;
            denDevFos=denDevFos+denDev;
            num+=numDevFos;
            den+=denDevFos;
            numDevFos=numDevFos+numDev;
            denDevFos=denDevFos+denDev;
            num+=numDevFos;
            den+=denDevFos;
            numDevFos=numDevFos+numDev;
            denDevFos=denDevFos+denDev;
            num+=numDevFos;
            den+=denDevFos;
            numDevFos=numDevFos+numDev;
            denDevFos=denDevFos+denDev;
            num+=numDevFos;
            den+=denDevFos;
            numDevFos=numDevFos+numDev;
            denDevFos=denDevFos+denDev;
            num+=numDevFos;
            den+=denDevFos;
            numDevFos=numDevFos+numDev;
            denDevFos=denDevFos+denDev;
            num+=numDevFos;
            den+=denDevFos;
            numDevFos=numDevFos+numDev;
            denDevFos=denDevFos+denDev;
            num+=numDevFos;
            den+=denDevFos;
            numDevFos=numDevFos+numDev;
            denDevFos=denDevFos+denDev;
            num+=numDevFos;
            den+=denDevFos;
            numDevFos=numDevFos+numDev;
            denDevFos=denDevF0
num=(biqCoef[i][0]-biqCoef[i][2])*(biqCoef[i][0]-
+biqCoef[i][2]+biqCoef[i][1]*biqCoef[i][1]+2.0*biqCoef[i][1]*biqCoef[i][0]+biqCoef[i]
+2.0*biqCoef[i][2]*biqCoef[i][0]*tempVar*tempVar;
  denDev=1.0-biqCoef[i][4]*biqCoef[i][4]+biqCoef[i][3]*(1.0-
+biqCoef[i][4])*tempVar;
  den=(1.0-biqCoef[i][4])*(1.0-
+biqCoef[i][4]+biqCoef[i][3]*biqCoef[i][3]+2.0*biqCoef[i][3]*(1.0+biqCoef[i][4])*temp
  Var+4.0*biqCoef[i][4]*tempVar*tempVar;
  delay=T*(denDev/(denDev+1.0E-7))-T*(numDev/(numDev+1.0E-7));
  grpdelay=grpdelay+delay;
}

if((Norder%2)!=0) { /*For odd order

  numDevFos=biqCoef[Nbiq][0]*biqCoef[Nbiq][0]+biqCoef[Nbiq][1]*biqCoef[Nbiq][0]*tempVar;
  numFos=biqCoef[Nbiq][0]*biqCoef[Nbiq][0]+biqCoef[Nbiq][1]*biqCoef[Nbiq][1]*biqCoef[Nbiq][0]*tempVar;
  denDevFos=1+biqCoef[Nbiq][2]*tempVar;
  denFos=1+biqCoef[Nbiq][2]*biqCoef[Nbiq][2]+2.0*biqCoef[Nbiq][2]*tempVar;
  delayFos=T*(denDevFos/(denDevFos+1.0E-7))-T*(numDevFos/(numFos+1.0E-7));
  grpdelay=grpdelay+delayFos;
}

  grpdelay=fabs(grpdelay);

  if(w<=passbd) err=err+(Norder*T-grpdelay)-(Norder*T-grpdelay);
  else {}
}
return(err);
}

void reproduce(){ /*Roulette wheel selection

137
int i, count;
double rawfitness, total;
double sum;
total = 1.0E-7;
for (i = 0; i < Nchrom; i++) total = total + chrom[i].fitness;

for (i = 0; i < Nchrom; i++) {
    rawfitness = UNI() * total;
    sum = chrom[0].fitness;
    count = 0;
    while (sum < rawfitness) {
        count++;
        sum = sum + chrom[count].fitness;
    }
    ordering[i] = count;
}

void CMCfix() {
    /* Crossover and mutation */
    int j, coef_count, coefPosition;
    preloadChrom();

    for (j = 0; j < Nchrom; j++) {
        coefPosition = ANUMBER() % Ncoef;
        for (coef_count = 0; coef_count <= coefPosition; coef_count++) {
            reloadGene(j, ordering[j], ordering[j + 1], coef_count);
        }

        for (coef_count = coefPosition + 1; coef_count < Ncoef; coef_count++) {
            reloadGene(j, ordering[j + 1], ordering[j], coef_count);
        }
    }
    DOcrossover(j, j + 1, coefPosition);
}
for(coef_count=0;coef_count<Ncoef;coef_count++){
    DOmutation(j,coef_count);
    DOmutation(j+1,coef_count);
}
}

void preloadChrom(){
    int i;
    for(i=0;i<Nchrom;i++) chrom2offsprg(i,i);
}

void chrom2offsprg(int chrom_loc,int offsprg_loc){
    int i,j;

    for(j=0;j<Ncoef;j++){
        for(i=0;i<Nbit;i++)
            offsprg[offsprg_loc].coef[j][i].bit=chrom[chrom_loc].coef[j][i].bit;
        offsprg[offsprg_loc].fitness=chrom[chrom_loc].fitness;
    }
}

void reloadGene(int a,int loc1,int loc2,int coefloc){
    int i;

    for(i=0;i<Nbit;i++){
        chrom[a].coef[coefloc][i].bit=offsprg[loc1].coef[coefloc][i].bit;
        chrom[a+1].coef[coefloc][i].bit=offsprg[loc2].coef[coefloc][i].bit;
    }
}

void DOcrossover(int loc1,int loc2,int coef_loc){
    int apoint,i;
    int abit;
    int flag1,flag2;
    double decCoef1,decCoef2;
if(UNI()<pcross){
    apoint=ANUMBER()%Nbit-1;
    apoint++;

    for(i=apoint;i<Nbit;i++){
        /*Crossover*/
        abit=chrom[loc1].coef[coef_loc][i].bit;

        chrom[loc1].coef[coef_loc][i].bit=chrom[loc2].coef[coef_loc][i].bit;
        chrom[loc2].coef[coef_loc][i].bit=abit;
    }

    flag1=CSDcheck(loc1,coef_loc);  /*CSD check*/
    if(flag1!=0){  /*Restore invalid coefficients*/
        decCoeff1=decodeChrom(loc1,coef_loc);
        DecToCSD(decCoeff1,1);
        for(i=0;i<Nbit;i++){
            chrom[loc1].coef[coef_loc][i].bit=csd[i];
        }
    }

    flag2=CSDcheck(loc2,coef_loc);
    if(flag2!=0){
        decCoeff2=decodeChrom(loc2,coef_loc);
        DecToCSD(decCoeff2,1);
        for(i=0;i<Nbit;i++){
            chrom[loc2].coef[coef_loc][i].bit=csd[i];
        }
    }
}

/*Restore invalid coefficient to its nearest CSD number*/
int DecToCSD(double decimal,int first){
    static int counter;
    static int position;
    int finished=0;

    /*...*/
if (first==1) {
    counter=0;
    position=0;
    int i;
    double MaxDec=0;

    for (i=0;i<Nbit;i++)
        csd[i]=0;

    for (i=0;i<=(NCSDdigit-1)*2;i+=2)
        MaxDec+=pow(2,0-i);

    if (decimal>MaxDec)
        decimal=MaxDec;
    else if (decimal<0-MaxDec)
        decimal=0-MaxDec;
}

if (position>=Nbit) finished=1;

while (finished==0) {
    if (decimal-1/pow(2,position)==0) {
        csd[position]=1;
        finished=1;
    }
    else if (decimal+1/pow(2,position)==0) {
        csd[position]=-1;
        finished=1;
    }
    else if (fabs(decimal-1.0/pow(2.0,position))<1.0/pow(2.0,(position+2.0))/0.75) {

csd[position]=1;

if (position<=Nbit-2)
    csd[position+1]=0;

counter+=1;
if (counter>=NCSDdigit)
    finished=1;
else
{
    position+=2;
    finished=DecToCSD(decimal-1/pow(2,position-2),0);
}
else if (fabs(decimal+1.0/pow(2.0,position))<(1.0/pow(2.0,(position+2)))/0.75)
{
    csd[position]=-1;

    if (position<=Nbit-2)
        csd[position+1]=0;

    counter+=1;
    if (counter>=NCSDdigit)
        finished=1;
    else
    {
        position+=2;
        finished=DecToCSD(decimal+1/pow(2,position-2),0);
    }
}
else
{
    csd[position]=0;
    position+=1;
    finished=DecToCSD(decimal,0);
}
return finished;
}

void DOmutation(int loc,int coefloc){
    int i,temp,flag;
    double decCoef;

    for(i=0;i<Nbit;i++)
        /*Mutation*/
        if(UNI()<pmut){
            temp=chrom[loc].coef[coefloc][i].bit;
            chrom[loc].coef[coefloc][i].bit=randBit(temp);
        }
    }

    flag=CSDcheck(loc,coefloc);   /*CSD check*/
    if(flag!=0){ /*Restore invalid coefficients*/
        decCoef=decodeChrom(loc,coefloc);
        DecToCSD(decCoef,1);
        for(i=0;i<Nbit;i++)
            chrom[loc].coef[coefloc][i].bit=csd[i];
    }
}

int CSDcheck(int loc,int geneloc){
    int sumbit=0;
    int i,a,b,c,rvalue=1;

    for(i=0;i<Nbit;i++) sumbit=sumbit+abs(chrom[loc].coef[geneloc][i].bit);

    if(sumbit<=NCSDdigit){
        for(i=1;i<(Nbit-1);i++)
            a=abs(chrom[loc].coef[geneloc][i].bit);
        b=abs(chrom[loc].coef[geneloc][i-1].bit);
        c=abs(chrom[loc].coef[geneloc][i+1].bit);
if(a&b) goto JUMP;
if(a&c) goto JUMP;
}
rvalue=0;
}

JUMP: return(rvalue);
}

void report(){
    /* Output to file */
    int i,j;

    outptr=fopen("output.txt", "w+");  
    if(outptr==NULL){
        printf("cannot open output file");
        exit(-1);
    }

    for(i=0;i<Ncoef;i++){
        for(j=0;j<Nbit;j++) fprintf(outptr,"%3d",bestChrom.coef[i][j].bit);
        fprintf(outptr,"\n");
    }

    fprintf(outptr,"\n");
    for(i=0;i<Ncoef;i++){
        fprintf(outptr,"\n%18.15f",decodeBest(i));
    }

    fprintf(outptr,"\n\nThe best fitness is %f", bestChrom.fitness);
    fclose(outptr);
}

double decodeBest(int coef_loc){
    /* Decode the value of the best chromosome */
    double anumber;
    int n;
anumber=0;
for(n=0;n<Nbit;n++){
    if(bestChrom.coef[coef_loc][n].bit==1)
        anumber=anumber+pow(2.0,-1.0*(n));
    else if(bestChrom.coef[coef_loc][n].bit==-1)
        anumber=anumber-pow(2.0,-1.0*(n));
    else{}
}
return(anumber);
VITA AUCTORIS

Li Liang was born in Beijing, China. She received her Bachelor of Engineering degree from Northern Jiaotong University in Beijing, China in 1994. After graduation, she worked as an Electrical Engineer in Beijing. She immigrated to Canada in 2001. She is currently a candidate for the degree of Master of Applied Science at the University of Windsor.