UNIVERSITY OF LAGOS

AKOKA LAGOS

Design of Digital Filters
Using Incremental Changes in Processing

A Thesis submitted in fulfillment of the
requirements for the degree of Doctor of philosophy
in Electrical Engineering

By

Samuel Oluwole Falaki
B.Sc.(Eng.) M.Sc.(Eng.) M.Sc.(Computer Sciences)

1980

SUPERVISORS: Dr D.S. Dawoud
Prof. O.A. Seriki
Prof. F.S. Atiya.

UNIVERSITY OF LAGOS
THESIS APPROVED
BY SENATE
28TH DAY OF OCTOBER

(i)
# TABLE OF CONTENTS

LIST OF FIGURES xi
LIST OF TABLES x
ACKNOWLEDGEMENTS xi
PREFACE xii

## CHAPTER 1

1.1 Introduction 1
1.2 Digital filters: Definitions 3
1.3 Methods of designing Digital Filters 11
1.3.1 FIR Filter Design techniques 11
1.3.1.1 Design of FIR Filters using windows 12
1.3.1.2 Computer-Aided Design of FIR Filters 13
1.3.2 IIR Filters Design Technique 14
1.3.2.1 IIR Digital Filter Design from continuous time Filters 14
1.3.2.2 Direct Design of IIR Digital Filters 16
1.3.2.3 Optimization Methods for Designing IIR Filters 17
1.4 Sources of Errors in Digital Filters 21
1.4.1 Effects of Number Representation on Quantization 22
1.4.2 Effect of truncating or Rounding 23
1.4.3 Effect of coefficient Inaccuracy 27
1.4.5 Limit Cycle Oscillations 29
1.4.6 Overflow Oscillation 32
References 34
CHAPTER 2. MODULAR APPROACH TO THE DESIGN OF DIGITAL SIGNAL PROCESSOR

2.1 Introduction 36
2.2 Design Approach - Mapping of differentials 38
2.3 Hardware Implementation 41
2.3.1 The Use of Backward Difference 42
2.3.2 The Use of Forward Difference 42
2.3.3 The Use of Higher Order Difference 47
2.3.4 Difference Table Generation Method 49
2.4 Realization of Higher Order Digital Filters 49
2.5 Conclusions 54
References 57

CHAPTER 3. DIGITAL SIGNAL PROCESSING USING ONE-BIT WORD

3.1 Introduction 58
3.2 The Delta Modulation (Single-bit word) System 60
3.3 On the Possibility of using DM in processing 62
3.4 Direct-form Realization of One-bit word Non-recursive Digital Filter 68
3.4.1 The Use of Recirculating Shift Registers 70
3.4.2 Direct-form Realization of One-bit word Recursive Digital Filter 70
3.4.3 Other Methods of Implementing the One-bit word Digital Filter 76
3.5 Digital Filter Implementation Using Digital Incremental Machines 78
3.5.1 The Digital Integrator Unit 80
3.5.2 The Summing Integrator (Digital Servo) 80
3.5.3 The Scale Integrator (Multiplication by a Constant) 83
CHAPTER 4. A NEW APPROACH TO THE IMPLEMENTATION OF HIGH SPEED DIGITAL FILTERS USING CARRY-SAVE ARRAYS

4.1 Introduction 96
4.2 The Proposed Algorithm 97
4.3 Hardware Realization of the Proposed Algorithm 98
4.3.1 Calculation of the $S_{kj}$ Numbers 101
4.3.1.1 The Use of Asynchronous and Synchronous Counters 101
4.3.1.2 The Use of Parallel Counters 101
4.3.1.2.1 The Use of Two Level Gate Network 103
4.3.1.2.2 The Use of ROM 103
4.3.1.2.3 The Use of Full-adder Network 103
4.3.1.2.4 The Use of Full-Adder/Fast adder Array 105
4.3.1.2.5 The Use of Small Parallel Counters 107
4.3.1.2.6 The Use of Quasi-digital Techniques 107
4.3.1.2.7 The Use of Reordering Technique 107
4.3.2 Calculation of the $C_j$ Numbers 111
4.3.3 Calculation of the $y_n$ Number 122
4.4 Modified Design to allow the use of Negative Numbers 122

4.5 Realization of $M$th Order Non-Recursive Digital Filter 126

4.5.1 Hardware Requirements and Computation time 132

References 135

CHAPTER 5. CONCLUDING REMARKS. 138
<table>
<thead>
<tr>
<th>Figure</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Typical recursive and non-recursive realizations of digital filters</td>
<td>7</td>
</tr>
<tr>
<td>1.2</td>
<td>Recursive Second-order Section</td>
<td>9</td>
</tr>
<tr>
<td>1.3</td>
<td>Cascade Form</td>
<td>9</td>
</tr>
<tr>
<td>1.4</td>
<td>Parallel Form</td>
<td>10</td>
</tr>
<tr>
<td>1.5</td>
<td>Block Diagram of A/D Converter</td>
<td>25</td>
</tr>
<tr>
<td>1.6</td>
<td>Probability Density Function for Roundoff Error</td>
<td>25</td>
</tr>
<tr>
<td>1.8</td>
<td>Linear Model of Quantization Noise in an LTI System</td>
<td>27</td>
</tr>
<tr>
<td>2.1</td>
<td>The Module Using Backward Difference</td>
<td>43</td>
</tr>
<tr>
<td>2.2</td>
<td>The Module Using Forward Difference</td>
<td>46</td>
</tr>
<tr>
<td></td>
<td>(Second-order Prediction)</td>
<td></td>
</tr>
<tr>
<td>2.3</td>
<td>The Module Using Higher Order Difference</td>
<td>48</td>
</tr>
<tr>
<td>2.4</td>
<td>The Module in Case of Using table of Differences</td>
<td>50</td>
</tr>
<tr>
<td>2.5</td>
<td>The Hardware Realization of $N^{th}$ Order Digital Filter</td>
<td>52</td>
</tr>
<tr>
<td>2.6</td>
<td>Formation of the Difference table using Delay Elements and Adders</td>
<td>53</td>
</tr>
<tr>
<td>2.7</td>
<td>Generation of the First Row of the Difference Table</td>
<td>55</td>
</tr>
<tr>
<td>2.8</td>
<td>The Proposed Hardware Realization of an $N^{th}$ Order Filter</td>
<td>56</td>
</tr>
<tr>
<td>3.1</td>
<td>Delta Modulator and Demodulator</td>
<td>61</td>
</tr>
<tr>
<td>3.2</td>
<td>Realization using PCM</td>
<td>63</td>
</tr>
<tr>
<td>Section</td>
<td>Title</td>
<td>Page</td>
</tr>
<tr>
<td>---------</td>
<td>----------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>3.3</td>
<td>Realization using Delta Modulation</td>
<td>63</td>
</tr>
<tr>
<td>3.4</td>
<td>Effect of delaying the binary signal $w_n$ by $rT$ and Multiplying by a Constant $a_r$</td>
<td>66</td>
</tr>
<tr>
<td>3.5</td>
<td>Non-recursive Filter Realization using One-bit word</td>
<td>67</td>
</tr>
<tr>
<td>3.6</td>
<td>Programmable binary transversal filter having binary weighting coefficients</td>
<td>69</td>
</tr>
<tr>
<td>3.7</td>
<td>Programmable binary transversal filter with Serial Processing</td>
<td>71</td>
</tr>
<tr>
<td>3.8</td>
<td>DM Recursive Filter at Sampling Instants</td>
<td>72</td>
</tr>
<tr>
<td>3.9</td>
<td>DM Recursive Filter</td>
<td>74</td>
</tr>
<tr>
<td>3.10</td>
<td>Elementary Filter Section</td>
<td>75</td>
</tr>
<tr>
<td>3.11</td>
<td>Hardware realization of non-recursive Digital Filter using Delta Modulation</td>
<td>77</td>
</tr>
<tr>
<td>3.12</td>
<td>A very fast ROM realization of a non-recursive Digital Filter</td>
<td>79</td>
</tr>
<tr>
<td>3.13</td>
<td>The Digital Integrator</td>
<td>81</td>
</tr>
<tr>
<td>3.14</td>
<td>Special Connection for the Digital Integrator</td>
<td>81</td>
</tr>
<tr>
<td>3.15</td>
<td>Non-recursive Digital Filter</td>
<td>89</td>
</tr>
<tr>
<td>3.16</td>
<td>First and Second Order Recursive DF</td>
<td>92</td>
</tr>
<tr>
<td>3.17</td>
<td>Output Response of Low-pass DF</td>
<td>93</td>
</tr>
<tr>
<td>1.</td>
<td>Symbolic Representation of the proposed Algorithm</td>
<td>100</td>
</tr>
<tr>
<td>2.</td>
<td>Three-Stage binary ripple Counter</td>
<td>102</td>
</tr>
<tr>
<td>3.</td>
<td>Three-Stage binary Synchronous Counter</td>
<td>102</td>
</tr>
<tr>
<td>4.</td>
<td>A $(U,V)$ - Counter Schema</td>
<td>102</td>
</tr>
</tbody>
</table>
5. The Use of ROM to implement $S_{kj}$

6a. 7-input binary full adder network parallel counter

6b. Synthesis of a 15-input Counter with a full-adder network

7. (U.V.)-Counter Pyramid

8. Sequential Circuit for Reordering $x_{ji}$

8a. Calculation of $S_{kj}$ using Serial Counter

8b. Calculation of $S_{kj}$ using parallel Counter

9a. Array Multiplication of Positive Numbers

9b. Faster Array Structure for Multiplication

10. Tree Idea for Array Multiplier

11. Array Multiplier with Diagonal Carry and Tree Arrangement for Sums

12. Example of 3-to-2 Tree for 7-input Operands

13. Bit positions of $S_{kj}$ Numbers and $C_j$ Number

14. Realization of $C_j$ Number using full-adder Arrays

15. Calculation of $C_j$ Number using Carry-save Arrays

16. Realization of $C_j$ Number using CSA and CLA

17. Realization of $y_n$ using Carry-save Arrays

18. Three-bit up-down Synchronous Counter

19. Bit-position of the $S_{kj}$ Numbers (Arranged to obtain $C_j$ Number)

20. Implementation of $C_j$ Number using Carry-Save Arrays

(viii)
21. Implementation of $C_j$ Number using CSA and CLA

22. Realization of the Matrix of the bit product $a_{i,k} \cdot x_{n-i,j}$ using AND-gates

23. General Block Diagram of the DF
## LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Comparison between exact and Rounded Values of ( y(n) )</td>
<td>30</td>
</tr>
<tr>
<td>2.1</td>
<td>Relationship between ( k_i ) and ( L ) (order of difference)</td>
<td>47</td>
</tr>
<tr>
<td>2.2</td>
<td>Comparison of the various Implementation Methods</td>
<td>51</td>
</tr>
<tr>
<td>3.1</td>
<td>Values of ( S_n ) and ( C_n ) as function of the values ( X_n ), ( Y_n ) and ( C_{n-1} )</td>
<td>84</td>
</tr>
<tr>
<td>4.1</td>
<td>Speed of Operation</td>
<td>133</td>
</tr>
<tr>
<td>4.2</td>
<td>Hardware Requirements</td>
<td>134</td>
</tr>
</tbody>
</table>
ABSTRACT

Procedures are developed for designing high-speed, modular digital filters. The techniques presented here are based on the use of differences in the data signal in processing. The use of one-bit word, and the use of mapping of differentials in processing to produce a multiplication free, fast, and modular structured digital filters is considered in details in this thesis. A new algorithm for the digital filter is also proposed in the thesis which leads to the possibility of using, for the first time, the carry-save arrays in implementing a multiplication free digital filter.

The importance of these techniques "use of differences" arises from the wide-spread use of differential pulse code modulated (DPCM) signals, as well as the use of delta modulated (DM) signals in communication networks.

The principles of operations as well as the possible hardware realizations of such digital filters are considered in details.
ACKNOWLEDGEMENTS

The author is indebted to the many people who have contributed to the preparation of this thesis. I am appreciative of the guidance provided by my Supervisors, and I am especially indebted to Dr D.S. Dawoud whose encouragement and advice made this thesis possible. Special thanks are due to Professor O.A. Seriki for providing the environment and stimulation for this work, and also to Professor F.S. Atiya for introducing me to the work.

I am deeply indebted to my wife and children for their understanding and support. I am also thankful to the faculty members of the Department of Computer Sciences, University of Lagos, for their valuable assistance and moral support.

Lastly I am grateful to Mr R. Tayo Sami and E.C. Boko for the typing done in preparing the final manuscript. I am also indebted to Mr Ime J. Abatai (Engineering Draughtsman) for preparing the drawings.
PREFACE

The advantages of coding a signal digitally are well known and are widely discussed in the literature. Briefly, digital representation offers ruggedness, efficient signal regeneration, easy encryption, the possibility of combining transmission and switching functions, and the advantage of a uniform format for different types of signals. The price paid for these benefits is the need for increased bandwidth. Due to the wide-spread use of digital signal processing systems, it is important to build digital systems which have modular structures to minimize hardware and also increase computation speed.

The speed of computation of a digital Signal Processor is directly related to the number of multiplications required and also to the speed of carrying out a multiplication. While the modularity of a system is related to the finding of algorithms that can be used to convert the well known digital signal processor equations (recursion equations) to a form suitable for modular implementation. The number of multiplications is directly related to the order of the filters, which depends on the required characteristics. As a matter of fact, the use of recursive filters reduces the order of the filters for a given characteristics. But for a given order, the number of multiplications will be fixed. As a result, several authors have tried to develop very fast multipliers. Fast multipliers are generally characterized by their hardware complexity. This means that any reduction in the computation time due to the use of fast multipliers, will result in increased complexity of the hardware. It was then necessary to solve the problem of speed and hardware complexity by using a completely different strategy. This new strategy is based on building a multiplication free digital filter.
During the last few years, there were some proposals on this line. These proposals can be divided into three different groups:

1. the use of look-up tables;
2. the use of differences in processing;
3. the use of distributed Arithmetic.

The use of a look-up table (using ROM) is limited by the required storage capacity. For practical values of the coefficients and data word length (12 bits/word), the required storage capacity ($2^{24}$ words for the given wordlength) will be impracticable.

The use of differences to realize a multiplication free digital filter is the subject of this thesis. In this technique the difference in the data signal which generally has a less word-length than the full sampled value, is to be used for processing. The limiting case is the case of using One-bit word-length in processing, which results in a completely multiplication free digital filter. The importance of this technique, "use of differences", arises from the wide-spread use of differential pulse code modulated (DPCM) signals, as well as the use of delta modulated (DM) signals (case of One-bit word) in communication networks. The resultant digital filter has a modular structure beside high-speed of operations. The principles of operations as well as the possible hardware realization will be considered in details in the thesis.

The use of distributed arithmetic for implementing a multiplication free digital filter is out of the scope of this thesis. But a short ideal about it will be given during our proposals for the use of carry-save arrays in building digital filters.

Chapter I deals with the general concepts of digital signal processing (Synthesis and Analysis). An up-to-date and detailed introduction to the fundamentals of digital signal processing is treated here.
In Chapter 2, a new technique for analysis and synthesis of digital signal processor is proposed. This approach is modular, and is flexible. The technique is based on the use of a simple mapping (Mapping of differentials) procedure to transform a continuous-time filter to its corresponding digital filter. More than one design for the module is considered. A comparison between these different designs from the viewpoint of computational time and equipment needed is also given.

In Chapter 3, a new treatment of the completely digital addition, subtraction, and multiplication by a constant, of DM signals (One-bit word) is presented. Beside the theoretical treatment, the hardware implementation of these operations are given. Furthermore, with the suggested multipliers, it is possible to realize multiple-multipliers, which yield simultaneously the product of DM signals with many coefficients. The method suggested also provides information concerning the errors introduced by the operation. The realization of recursive digital filters using the proposed algorithm is also presented. The characteristics of the filter is calculated by simulating the proposed hardware on the IBM 370/145 digital computer. The results of the simulation are also given.

In Chapter 4, a short idea about the use of distributed Arithmetic to design an effectively multiplication free digital filter in spite of the use of full sample values of input variables in processing is presented. A new algorithm is considered which will convert the difference equation describing the filter to the sum of the weighted number $C_j \cdot 2^j$. The realization of $n^{th}$ order non-recursive digital filter is given. The realization of the filter needs only counters and carry-save arrays.

Conclusions and areas of further research are given in Chapter 5.
CHAPTER 1

GENERAL CONCEPTS OF DIGITAL SIGNAL PROCESSING

1.1 Introduction:

Digital Signal processing is concerned with the representation of signals by sequences of numbers or symbols and the processing of these sequences. The purpose of such processing may be to estimate characteristic parameters of a signal or to transform a signal into a form which is in some sense more desirable [3].

The major subdivisions of the field of digital signal processing are digital filtering and spectrum analysis. The field of digital filtering is further divided into finite impulse response (FIR) filters and infinite impulse response (IIR) filters. The field of spectrum analysis is broken into calculation of spectra via the discrete fourier transform (DFT) and via statistical techniques as in the case of random signals - e.g. quantization noise in a digital system [1]. The Fast Fourier transform (FFT) and the related area of fast convolution are almost exclusively used in practical spectrum analysis techniques.

The proliferation of the use of digital signal processing can be witnessed by its appearance in a variety of areas of scientific endeavour such as biomedical engineering, seismic and geophysical research, image processing and pattern recognition, radar and sonar detection and counter measures acoustics and speech research, and telecommunications.

The use of digital techniques for various signal processing applications has increased tremendously in the last few years, and this trend is expected to continue in the years ahead. Systems that previously consisted only of analog circuits now have incorporated digital circuits to
do many of the functions required. These developments are partly due to recent advances in integrated circuit technology. With the advent of large scale integration and the resulting reduction in cost and size of digital components, together with increasing speed, the class of applications of digital signal processing techniques is growing.

Digital systems have several common advantages when compared with their analog equivalents for signal processing. The first of these is that the digital system is inherently more stable. By this, it is meant that the systems are less susceptible to changes in environment. Typically, the digital system will have its accuracy specified as a single figure over its entire environmental operating range. The equivalent analog system, on the other hand, will require that its accuracy is only specified at a single temperature with a temperature coefficient, or is specified within wider limits to take account of variations in system sensitivity with temperature. Another way in which this stability manifests itself is in the effects of the aging of components. With the digital system, the effects of aging are almost totally insignificant. The second advantage of the digital system is an improved linearity. The sampling frequency can be controlled to a high degree of accuracy, to give an exceptional frequency linearity. The third advantage of the digital system is increased flexibility. For instance, in a digital filtering system, the same hardware can be made to represent a seemingly infinite number of different filter shapes, simply by programming in different filter coefficients [6].

The design of a digital filter is the task of determining a discrete transfer function or difference equation which will achieve a prescribed signal filtering requirement (i.e. Analysis of the digital filter). The implementation of the filter transfer function by digital hardware is the second phase of the design process (synthesis of the digital filter).
The aim of this chapter, is to introduce the concept of digital signal processing and the subjects related to the realization of the two phases of digital filter design i.e. the analysis phase and the synthesis phase.

For this, definitions for the different terminologies related to the field of signal processing are given in section 2. The first phase of designing a digital filter i.e. in the analysis phase is considered in details in section 3. In this section the various techniques that can be used to design FIR as well as IIR filters are considered. The digital processing of an originally analog signal results in some errors e.g. quantization error, round-off error ... etc. These errors are considered in section 4.

1.2 Digital Filters: Definitions:

A digital filter is a computational process or algorithm by which a digital signal or sequence of numbers (acting as input) is transformed into a second sequence of numbers termed the output digital signal. The numbers are limited to a finite precision. The algorithm may be implemented in software as a computersubroutine for a general-purpose machine or in hardware as a special-purpose computer. The term digital filter is then applied to the specific routine in execution or to the hardware [7].

The signals to be processed by a digital filter may originate in an analogue-to-digital converter in which case they represent some real-time analogue signal. Theoretically a digital signal can represent an analogue signal with full fidelity and without loss of information, though very careful design is necessary to keep the distortion within limits; a factor not to be forgotten when considering digital systems for processing analogue signals.

In case of processing analog signals digitally, digital filter is considered as a black box which must in principle function as an equivalent analogue filter. Analogue systems or continuous-time systems are described by a differential equation and by using the Laplace transformation the
transfer function can be found. It can be written in the form:

\[ H(S) = \frac{(s + r_1)(s + r_2) \ldots (s + r_M)}{(s + p_1)(s + p_2) \ldots (s + p_N)} \quad \ldots \quad (1.1) \]

where \( s \) is the Laplace Operator, \( r_1, r_2, \ldots \) are the zeroes and \( p_1, p_2, \ldots \) are the poles. Similarly, the digital systems or discrete time systems are described by difference equations and the \( z \)-transform plays in these a role analogous to that of the Laplace transform in continuous-time systems.

The difference equation of any digital filter may be written in the following form [1]:

\[ y(n) = \sum_{k=0}^{M} a_k x(n-k) - \sum_{k=1}^{N} b_k y(n-k) \quad \ldots \quad (1.2) \]

where \( \{x\} \) is the input sequence, \( \{y\} \) the output sequence and \( \{a_k\}, \{b_k\} \) are constants (Filter Coefficients). These coefficients define the filter response.

In the \( z \)-domain equation (1.2) can be represented by the discrete transfer function:

\[ H(z) = \frac{N(z)}{D(z)} = \frac{\sum_{k=0}^{M} a_k z^{-k}}{1 + \sum_{k=1}^{N} b_k z^{-k}} \quad \ldots \quad (1.3) \]

where \( N(z) \) is the numerator polynomial and \( D(z) \) is the denominator polynomial. The roots of \( N(z) \) and \( D(z) \) are, respectively, the zeros and poles of \( H(z) \). \( H(z) \), which is known as the transfer function, is a complex function and the values taken by \( H(z) \) when evaluated on the unit circle in the \( z \)-plane give the frequency response. Each point on the unit circle, characterized by its angle only, corresponds to a particular frequency. In most cases - especially for digital filters derived from
analog designs - M will be less than or equal to N, and there will be at least N - M zeros at z = 0. Systems of this type are called \( N^{th} \) order systems. When \( M > N \), the order of the system is no longer unambiguous. Here, N gives the order as the term used to characterize the mathematical properties of the difference equations. M gives the order used to characterize the complexity of a realization of the system. There is no general agreement as to which of \( M \) or \( N \) best characterizes the system when \( M > N \) [7].

The Fourier transform of the unit impulse response, \( H(e^{j\omega}) \), is called the frequency response. The digital filter of equation (1.2) calculates the \( n^{th} \) output sample \( y_n \) by using the previous N output samples \( y_{n-1}, y_{n-2}, \ldots, y_{n-N} \) as well as the current and the previous M input samples \( x_n, x_{n-1}, \ldots, x_{n-M} \). Since no future samples from either the input or the output are needed to calculate the current output sample, the filter is physically realizable or causal. In this case the unit sample impulse response \( h(n) = 0 \) for \( n < 0 \). A stable system is one for which every bounded input produces a bounded output. A necessary and sufficient condition for a linear shift-invariant system to be stable is

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

(1.4)

That is, \( h(n) \to 0 \) as \( n \to \infty \). So all the poles lie inside the unit circle.

Digital filters are classified both from the standpoint of the duration of the impulse response and from the standpoint of the type of realization. The impulse response duration characteristics can be divided into two broad classes:
(a) **Finite Impulse Response (FIR).** An FIR filter is one in which the impulse response \( h(n) \) is limited to a finite number of samples defined over the range \( n_1 \leq n \leq n_2 \), where \( n_1 \) and \( n_2 \) are both finite.

(b) **Infinite Impulse Response (IIR).** An IIR filter is one in which the impulse response \( h(n) \) has an infinite number of samples. Thus \( h(n) \) is non-zero at an infinite number of points in the range \( n_1 \leq n \leq \infty \).

The possible realization procedures can be divided into three broad classes:

(a) **Recursive Realization:** A recursive realization is one in which the present value of the output depends both on the input (present and/or past values) and previous values of the output. A recursive filter is usually recognized by the presence of both \( a_k \) and \( b_k \) terms in a realization of the form of (1.2).

(b) **Non recursive (or Direct convolution) Realization:** A non recursive or direct convolution realization is one in which the present value of the output depends only on the present and past values of the input. This usually means that all values of \( b_k = 0 \) in a realization of the form of (1.2).

(c) **Fast Fourier Transform (FFT) Realization:** This type of realization is achieved by transforming the input signal with the FFT, filtering the spectrum as desired, and performing an inverse transformation.

In general, an IIR filter is usually more easily implemented by a recursive realization, and an FIR filter is usually more easily implemented by either a non recursive realization or an FFT realization. However, both FIR and IIR functions may be implemented by either recursive or non recursive techniques.

Clearly, from (1.2), the fundamental components of digital filter hardware are simply adders, multipliers, and delay elements. If one were to draw from (1.2) a block diagram composed of adders, multipliers, and
delay elements, the result would look like Fig. 1.1. This is known as the direct form of a digital filter.

But, strictly speaking, (1,2) does not apply to digital filters without modification; what it does describe are discrete-time or sampled-data filters, in which only time is discrete and other quantities are continuous.

(a) Recursive Realization

(b) Non recursive Realization

Fig. 1.1: Typical recursive and non-recursive realizations of digital filters.

When applied to digital filters, it is necessary that all the terms
of (1.2) be expressed with digital words of finite length, i.e. they must be quantized. Were it not because of this requirement, the order in which the arithmetic operations of (1.2) are performed would be immaterial and the direct form of Fig. (1.1) would suffice for all digital filters. Such is not the case, however, and in practice the number of bits of wordlength used to represent various quantities in (1.2) has a strong effect on the amount of hardware required to build the corresponding digital filter.

A digital filter transfer function can often be realized in a variety of ways. Noise and inaccuracies caused by quantization of any practical digital filter implementation are very dependent on the precise digital filter structure.

Another structure for realizing digital filters can be obtained by writing equation (1.2) in the form

\[ H(z) = Y(z) = \frac{Y(z)}{X(z)} = a_0 \prod_{i=1}^{k} H_i(z) \tag{1.5} \]

where \( H_i(z) \) is either a second-order section, shown in Fig. 1.2, i.e.

\[ H_i(z) = \frac{1 + a_{1i} z^{-1} + a_{2i} z^{-2}}{1 + b_{1i} z^{-1} + b_{2i} z^{-2}} \tag{1.6} \]

or a first-order section i.e.

\[ H_i(z) = \frac{1 + a_{1i} z^{-1}}{1 + b_{1i} z^{-1}} \tag{1.7} \]

and \( k \) is the integer part of \((N + 1)/2\).
The realization of equation (1.5) is called the cascade form and is shown in Fig. (1.3)

\[ X(n) \xrightarrow{H_1(z)} H_2(z) \xrightarrow{H_3(z)} \cdots \xrightarrow{H_k(z)} y(n) \]

Fig 1.3: Cascade Form

Yet another structure for realizing digital filters can be obtained by writing Eq.(1.2) in its partial fraction expansion as

\[ H(z) = C + \sum_{i=1}^{k} H_i(z) \]

(1.8)

where \( H_i(z) \) is either a second-order section of the form
\[ H_i(z) = \frac{a_{0i} + a_{1i}z^{-1}}{a + b_{1i}z^{-1} + b_{2i}z^{-1}} \] (1.9)

or a first-order section of the form
\[ H_i(z) = \frac{a_{0i}}{1 + b_{1i}z^{-1}} \] (1.10)

\( k \) is the integer part of \((N + 1)/2\), and \( C = \frac{a_0}{b_N} \) as defined in Eq. (1.2).

![Parallel Form](image)

Figure 1.4 shows a realization of the structure of equation (1.8) which is called the parallel form. The individual first- and second order sections of eq. (1.9) and (1.10) are realized using one of the direct forms.

The filter structures given above are by no means the only available structures. They are, however, the most widely used both in simulations and in digital hardware. The choice among the various structures given above is dictated by their economy of implementation, whether it be hardware or software. This, in turn, depends in many instances on properties of the structures when finite word lengths are
attached to the filter coefficients and the filter variables \([1]\).

As shown above, the second-order section can be used to realize any desired filter by cascading or paralleling the basic sections. The filters so produced have proved in practice to be relatively economical of required coefficient and data word accuracy, and hence of required hardware. Moreover, such filters are not difficult to design since the values of the coefficients can be quickly determined once the pole-zero configuration of the transfer function is known. For these reasons, the basic second-order section has assumed much importance in digital filter work.

1.3 Methods of Designing Digital Filters

The design of digital filters (for realization either in hardware or software) involves the following steps:

(a) The specification of the desired properties of the system.

(b) The approximation of these specifications using a causal discrete-time system and

(c) The realization of the system using finite-precision arithmetic.

The first step is highly dependent on the application, the second step is going to be discussed briefly here and the third step is going to be discussed in another chapter.

1.3.1 FIR Filters Design Techniques:

If all the coefficients, \(b_k\) in Eq. (1.2) are zero, the difference equation becomes

\[
y(n) = \sum_{k=0}^{M} a_k x(n-k)
\]

and

\[
h(n) = a_n \quad 0 \leq n \leq M
\]

\[
= 0 \quad \text{otherwise}
\]
FIR systems have a number of important properties. First, we note that $H(z)$ is a polynomial in $z^{-1}$, and thus $H(z)$ has no non-zero poles, only zeros. Also, FIR systems can have exactly linear phase. This simplifies the approximation problem, in many cases, when one is only interested in designing a filter that approximates an arbitrary magnitude response. Linear Phase Filters are important for applications where frequency dispersion due to nonlinear phase is harmful - e.g., speech processing and data transmission [1]. The penalty that is paid for being able to design filters with an exact linear phase response is that a large impulse response duration is required to adequately approximate sharp cutoff filters.

A number of excellent design methods for linear phase FIR filters have been developed in recent years. Some of these methods are discussed briefly below:

1.3.1.1 Design of FIR Filters Using Windows [1,3,4,5]

One of the earliest attempts at deriving the coefficients of an FIR digital filter in order to approximate an ideal desired frequency response was the method of windowing in which the desired frequency response is expanded in a Fourier Series and truncated to the desired filter length. The resulting filter minimizes the least-squares error between the desired response and the filter response. Instead of simply truncating the infinite Fourier Series, the technique of windowing seeks to reduce the Gibbs phenomenon by multiplying the coefficients of the Fourier Series by a smooth time-limited window. Among the more popular windows are the Kaiser window, the Hamming window, the rectangular window, the Dolph-Chebyshev window, the triangular window, and the Blackman window. One of the attributes of windowing is that it is an analytical technique, whereas, most other FIR design techniques are iterative in nature.
1.3.1.2 Computer-Aided Design of FIR Filters.

There are several iterative design procedures for FIR filters which in general yield better filters than the window method at the expense of greater complexity in the design procedure. Some of these techniques are discussed briefly below.

(a) The Frequency Sampling Method \([1,3,11]\)

The main idea of the frequency sampling design method is that one can approximate a specified frequency response by fixing most of its discrete Fourier transform (DFT) coefficients (the frequency samples) and leaving unspecified those DFT coefficients which lie transition bands. An optimization algorithm is used to choose values for the unspecified coefficients so as to minimize a weighted approximation error over the frequency range of interest.

Simple linear programming techniques may also be used to perform the necessary minimization.

(b) Optimal (Minimax Error) FIR Filter Design \([1,3,11]\)

By considering the linear phase FIR filter design problem as a Chebyshev approximation problem, it is possible to derive a set of conditions for which it can be proved that the solution, is optimal (in the sense that the peak approximation error over the entire interval of approximation is minimized) and unique. In other words, necessary and sufficient conditions for the best Chebyshev approximation could be derived from the classical alternation theorem. The Remez exchange algorithm has been demonstrated to be an effective tool for the computation of these optimal filters. Linear programming offers an alternative method for computing the best Chebyshev approximation. Although linear programming is very flexible and can be used to approximate a wide variety of desired filter shapes, it is comparatively slow and hence the length of the filter it can design is limited.
(c) The Simplex Method of Linear Programming [14, 15]

The simplex method of linear programming can be used to determine the coefficients of the nonrecursive digital filter which minimizes the maximum of the error in the complex response.

1.3.2 IIR Filters Design Techniques

The most popular technique for designing IIR filters is to digitize an analog filter that satisfies the design specifications. The second method for designing IIR digital filters is direct closed form design in the z-plane. Beginning with the desired response of the filter, one can often decide where to place poles and zeros to approximate this response directly. A third way in which IIR filters are often designed is by using optimization procedures to place poles and zeros at appropriate positions in the z-plane to approximate in some sense the desired response [1]. In the following, these techniques will be considered.

1.3.2.1 IIR Digital Filter Design from Continuous-Time Filters

There are many techniques for designing analog prototype filters when the specifications are of the form of a lowpass, bandpass, high pass, or band reject filter. Among the well-known analog filter classes are the Butterworth, Bessel, chebyshev types I and II and Cauer or elliptic filters. There are also several procedures for transforming or digitizing an existing analog filter to an equivalent digital filter. The four most widely used procedures for digitizing analog filters are:

(a) The Method of Mapping of Differentials, [1]

This technique involves the replacement of the differentials in the differential equation of the continuous system with finite differences in order to obtain a difference equation that approximates the given differential equation. The simplest replacement that can be made is to use
a forward or backward difference to replace a first differential. The use of higher-order differences to replace lower-order differentials tends to preserve the two desirable properties of mapping from continuous to discrete space.

(b) The impulse invariant transformation \([1,7]\]

This is a technique in which the impulse response of the derived digital filter is identical to the sampled impulse response of a continuous-time filter. Hence, the impulse response of the digital filter is an aliased version of the frequency response of the corresponding analog filter. This technique is primarily used for narrowband filter designs or else the transformation is applied to the cascade combination of a guard filter and continuous-time transfer function.

(c) The bilinear transformation \([1,3,7,10]\]

This technique uses conformal mapping, from s-plane to the z-plane which eliminates the aliasing problem of the impulse invariant technique. This transformation has the effect of mapping the entire s-plane into the z-plane in such a way that the left-half s-plane maps into the inside of the unit circle and the right-half s-plane maps into the outside of the unit circle. This design technique is most useful in obtaining digital designs of filters whose frequency response can be divided into a number of pass and stop bands in which the response is essentially constant \([7]\).

(d) The matched z-transform technique

This is a technique based on mapping the poles and zeros of the continuous-time filter. That is, the poles of \(H(z)\) will be identical to those obtained by impulse invariant method, however, the zeros will not correspond. The continuous transfer function \(H(s)\) must be in
factored form to apply the matched z-transformation. This technique is easy to apply, although there are many cases when it is not a suitable mapping procedure. For example, if the analog system has zeros with center frequencies greater than half the sampling frequency, their z-plane positions will be greatly aliased. It is also not suitable where the continuous transfer function is an all-pole system. In general, the use of impulse invariant or bilinear transformation is to be preferred over the matched z-transformation [1].

1.3.2.2 Direct Design of IIR Digital Filters

A second possible method for designing IIR digital filters is direct digital design in either the frequency or the time domains. Some of the techniques under this category are:

(a) Magnitude-Squared Function Design [1,10]

In this technique, the filter characteristics are specified via the potential analogy or from a desired squared-magnitude function by using procedures akin to that of the Butterworth or Chebyshev continuous-filter design procedure. In this technique, a suitable rational trigonometric polynomial must be found to provide the desired filtering. Also the magnitude-squared function must be factored to find the poles and zeros. This factorization is generally nontrivial and therefore makes this an undesirable filter design method [1].

(b) Time Domain Design of IIR filters [1,10]

Just as it is possible to design filters to approximate an arbitrary frequency response, it is also possible to design an IIR filter whose impulse response approximates a desired impulse response. The Padé approximation method is used in this technique,
1.3.2.3 Optimization Methods for Designing IIR filters

Many optimization techniques have been used to determine the coefficients for a digital filter having a prescribed response. In this technique, the set of design equations cannot be solved explicitly to give the filter coefficients. Instead a mathematical optimization procedure is used to determine the filter coefficients that minimize some error criterion, subject to the appropriate design equations. The various design techniques already discussed could be taken as an initial estimate for design values and then using different optimization techniques the best filter performance can be achieved. Some optimization design procedures are discussed briefly here.

(a) Minimum Mean Squared Error Design [1,3,12]

This is an IIR Filter design procedure based on minimization of the mean-square error in the frequency domain. The procedure requires that the desired frequency response be prescribed as a discrete set of frequencies. The mean-squared error at these frequencies is defined and minimized as a function of certain parameters by solving a set of nonlinear equations. These equations can be solved algorithmically using, for example, the Fletcher-Powell method. This procedure deals only with the magnitude function.

(b) Minimization of a p-Error Criterion [1,3,10,12]

Deczky [12] has generalized the procedure of the previous section in a number of ways. Instead of minimizing the average squared error, a weighted average of the error raised to the $p^{th}$ power was minimized. Also this technique was applied to both the magnitude and the group delay. When $p = 1$, the approximation is identical to the squared error criterion of the preceding section. As $p \to \infty$, it can be shown that the approximation tends to the minimax of Chebyshev error criterion.
(c) **Least-Squares Inverse Design** [3, 13]

In the two procedures discussed above, the filter was specified in the frequency domain and the resulting set of equations was nonlinear in the filter parameters. An alternative procedure, based on a least-squares approximation to the inverse of the desired filter, leads to a set of linear equations. In this procedure the filter is specified in terms of the first $L$ samples of the desired impulse response. The filter design is based on the criterion that the output of the filter transfer function $H(z)$ must approximate a unit sample when the input is the desired impulse response.

(d) **Optimization Using All-pass Sections in the $w$ plane.** [1]

IIR filters with equiripple passband (or stopband) and arbitrary stopbands (stopbands) can be readily designed by mapping the approximation problem from the $z$ plane to the $w$ plane such that the filter passband in the $z$ plane maps to the entire imaginary axis in the $w$ plane.

An all-pass function is then synthesized whose magnitude is constant along the imaginary axis of the $w$ plane. A simple substitution is used to create a $w$-plane transfer function from the all-pass function whose magnitude is equiripple along the imaginary axis of the $w$ plane independent of the all-pass function coefficients. This optimization procedure was due to Deczky.

(e) **Linear Programming Techniques**

An equiripple approximation to an IIR filter with a prescribed magnitude characteristic can be obtained using linear programming techniques. In this technique, the magnitude-squared function of the filter is evaluated on the unit circle, and linear programming technique is used to find the coefficients of the function such that the absolute value of its magnitude-squared characteristic and also the peak approximation error is minimized.
(f) **Nonlinear Programming Technique** [15]

Both nonrecursive and recursive filters can be designed by nonlinear technique. The nonlinear programming approach resembles the linear programming technique. An error $E_k$ is formed by taking the difference between any previously mentioned response of the digital filter and the ideal values of this response at the frequency $X_k$. It is also possible to define more than one error corresponding to more than one type of response (e.g., amplitude and delay responses) at a single frequency $X_k$.

The nonlinear programming technique begins with a choice for the type and form of the digital filter. The type can be either nonrecursive or recursive. If the recursive type is chosen, the cascade or parallel realization of elementary sections are represented by:

\[
H(z) = g \sum_{i=1}^{s} \frac{a_i z^{-1} + b_i z^{-2}}{1 + c_i z^{-1} + d_i z^{-2}}
\]

or

\[
H(z) = g + \sum_{i=1}^{s} a_i \cdot b_i z^{-1}
\]

The error is chosen to be a differentiable function of all parameters $a_i$, $b_i$, $c_i$, $d_i$ and $g$. The error is defined frequently as the difference between the actual and the ideal values of the square of the amplitude response as in equation,

\[
E_k = |H(\exp(2\pi j x_k))|^2 - |F(2\pi W_k)|^2
\]

The errors $E_k$ are required to satisfy this inequality

\[
L_k \leq E_k \leq U_k, \quad k = 1, 2, \ldots, N
\]

where $L_k$ and $U_k$ are positive and real. In some cases either the lower or the upper limit on $E_k$ will be absent (e.g., $L_k$ will be absent if an ideal value of the amplitude response is zero).

A penalty function such as
\[ Q + \sum_{k=1}^{N} \frac{r}{G_k} + \sum_{k=1}^{N} \frac{r}{H_k} \quad \ldots \ldots \quad (1.17) \]

is formed, where
\[
G_k = \frac{\Delta Q}{U_k - E_k} \quad k = 1, 2, \ldots, N \quad \ldots \ldots \quad (1.18)
\]
\[
H_k = \frac{\Delta Q}{L_k + E_k} \quad Q > 0
\]

\( G_k \) or \( H_k \) are omitted for those values of \( k \) for which no upper or lower bound is specified for the error \( E_k \). A suitable program is used to minimize the penalty function simultaneously with respect to \( Q \) and all the coefficients \( a_i, b_i, c_i, \) and \( d_i \). Other nonlinear programming algorithms can be used to obtain the coefficients of a recursive filter having minimax errors.

\[ \text{(g) Integer linear Programming [15]} \]

Integer programming techniques for solving a linear program in which the variables are integers exist. These techniques can be used to determine the coefficients of digital filters whose coefficients are quantized i.e. expressed in fixed point arithmetic with only a finite number of bits. These techniques determine quantized coefficients which provide minimax errors in the response. This response can be either the frequency response or the time response. The amount of computation time required is only slightly larger than the amount required for ordinary linear programming technique. This is because it differs only from the linear programming technique by the assumption that the coefficients are quantized. A generalization of integer linear programming is nonlinear integer programming, as for example in determining the quantized coefficients of a recursive filter whose response has minimax error.
1.4 Sources of Errors in Digital Filters

The accuracy of a digital filter is limited by the finite word used in its implementation. The finite word length is a consequence of the encoding of all relevant filter parameters in a particular format (mostly binary) and of the fact that the parameters (signals) must be stored in registers, which, of course, have a finite length.

The three common sources of error due to finite word length are

1. the quantization of the input signal \( \{ x_n \} \) into a set of discrete levels.

2. the representation of the filter coefficients \( a_k \) and \( b_k \) by a finite number of bits.

3. the accumulation of round-off errors committed at arithmetic operations.

Because of these errors the actual output will be \( \{ y_n \} \) and will be in general, different from the ideal output \( \{ w_n \} \). One may define

\[
e_n = y_n - w_n
\]  

(1.19)

as the error at the output at the nth sample, and it is important for the designer and the user of a digital filter to be able to determine some measure of the error \( e_n \).

These problems have been investigated extensively, and techniques have been developed to treat the effects of the three common sources of error: input quantization, coefficient inaccuracy, and roundoff accumulation.

In addition to the word length, the accuracy of a digital filter depends on two important factors: the form of realization and the type of arithmetic used. Studies show that for a fixed word length, the accuracy achievable with a direct realization of a high-order filter is considerably less than that with either cascade or parallel realization of the same
filter. Thus a number of studies have concentrated on first and second order filters which are the basic building blocks for higher order filters [21].

1.4.1 Effects of Number Representation on Quantization:

A real number can be represented using a finite number of bits in either the fixed-point form or the floating-point form. Recently a hybrid between these representations was introduced called block floating-point form.

In fixed-point representation it is assumed that the position of the binary point is fixed. The bits to the right represent the fractional part of the number and those to the left represent the integer part. Depending on the way negative numbers are represented, there are three different forms of fixed point arithmetic. They are called sign-magnitude, 2's-complement, and 1's-complement representation. The choice among the three representations above is generally dictated by programming and/or hardware considerations. In digital filtering applications, it is usually necessary to approximate the 2b-bit product of two b-bit numbers by a b-bit result. In integer arithmetic this is difficult. With fractional arithmetic, on the other hand, this can be accomplished by truncating or rounding to the most significant b-bits. For multiplications with fractions, overflow can never occur since the product of two fractions is a fraction. If we add two fixed point fractions overflow can occur. This limitation on the range of numbers that can be represented can be essentially removed by using a floating-point representation.

The floating-point representation can lead to truncation or rounding errors for both addition and multiplication as opposed to the fixed point case where such errors occurred only for multiplication. Overflow is however highly unlikely in the case of floating point addition because of
the very large dynamic range. All of these effects must be considered when comparing fixed-point and floating-point realizations of digital filters.

1.4.2 Effect of Truncation or Rounding

The effect of truncation or rounding depends on whether fixed point or floating point arithmetic is used and how negative numbers are represented. Truncation of a number is accomplished by discarding all bits less significant than the least significant bit that is retained. Rounding of a number of \( b \) bits is accomplished by choosing the rounded result as the \( b \)-bit number closest to the original unrounded quantity. Because rounding is based on the magnitude of the number, the error is independent of the way in which negative numbers are represented. Let us consider the effect of truncation and rounding in the fixed-point case. Let us denote by \( b_1 \) the number of bits to the right of the binary point before truncation or rounding and by \( b \) the number of bits to the right of the binary point after truncation or rounding, with, of course \( b < b_1 \). It is reasonable to assume that \( 2^{-b_1} \ll 2^{-b} \), hence we can summarize the truncation and rounding errors as satisfying the following inequalities [23]:

**Truncation:**

\[
-2^{-b} < E_T < 0 \\
0 < E_T < 2^{-b}
\]

for positive numbers and two's-complement negative numbers \( \ldots \) (1.20a)

for sign and magnitude and one's-complement negative numbers \( \ldots \) (1.20b)

**Rounding**

\[
-\frac{1}{2} \cdot 2^{-b} < E_R < \frac{1}{2} \cdot 2^{-b}
\]

\( \ldots \) (1.21)

where \( E_T \) is the truncation error and \( E_R \) is the rounding error.
In floating-point arithmetic, the truncation or rounding only affects the mantissa. Thus, relative error is more important than absolute error. This means that floating-point errors are multiplicative rather than additive. If $X$ represents the value before rounding or truncation and $Q(X)$ represents the value after, then

$$Q(X) = X(1 + \epsilon) \quad \ldots \quad (1.22)$$

where $\epsilon$ is the relative error for the case of rounding, the error in the mantissa is between $2^{-b}/2$. The rounding error also satisfies the inequality

$$-2^{-b} < \epsilon < 2^{-b} \quad \ldots \quad (1.23)$$

It can be shown [22] that for one's-complement and sign-and-magnitude truncation of the mantissa

$$-2.2^{-b} < \epsilon < 0 \quad \ldots \quad (1.24)$$

and for two's-complement truncation,

$$-2.2^{-b} < \epsilon < 0 \quad X > 0 \quad \ldots \quad (1.25)$$

$$0 < \epsilon < 2.2^{-b} \quad X < 0$$

1.4.3 Effect of Input Quantization

The first place where quantization of the input may take place is at the analog-to-digital converter.

An analog-to-digital (A/D) Converter is a device that operates on the analog waveform to produce a digital output consisting of a sequence of numbers each of which approximates a corresponding sample of the input waveform. Figure 1.5 shows a simple block diagram for the A/D converter that is conceptually represented as a two stage process. In the first stage the sequence $s(n) = s(t)\big|_{t=nT}$ is created where $s(n)$ is expressed to infinite precision. In the second stage the numerical equivalent of each
sample of \( s(n) \) is expressed by a finite number of bits giving the sequence \( s_Q(n) \).

![Block Diagram of A/D Converter](image)

**Fig 1.5** BLOCK DIAGRAM OF A/D CONVERTER

The difference signal \( e(n) = s(n) - s_Q(n) \) is called quantizing noise or A/D conversion noise.

Depending on the detailed way in which \( s(n) \) is quantized, different distributions of quantization noise may be obtained. Quantization error, in the case of rounding satisfies the relation

\[
-\frac{\Delta}{2} < e(n) < \frac{\Delta}{2}
\]

for all \( n \).

where \( \Delta \) is the quantization width, i.e. \( \Delta = 2^{-b} \). It can be shown that the distribution of the error signal is uniform. Thus the probability distribution of the quantization error for rounding is shown in **Fig. 1.6**.

![Probability Density Function](image)

**Fig. 1.6.** Probability density function for roundoff error.
It can also be shown that quantization error resembles a random sequence. Other types of quantization may be used to give $s_Q(n)$. For example, truncation is the rule where the signal is represented by the highest quantization level that is not greater than the signal. Since truncation is equivalent to rounding less one-half a quantization step, the probability distribution of the error signal is shown in Fig. 1.7.

From Fig. 1.6 and Fig. 1.7 it is clear that the quantization error has mean value of 0 for rounding and $\Delta/2$ for truncation, whereas the variance is $\Delta^2/12$ in both cases.

![Fig. 1.7: Probability density function of truncation error.](image)

A third type of signal quantization is often used in digital processing. This type of quantization, called sign-magnitude truncation, is identical to truncation for positive signals; negative signals are approximated by the nearest quantization level that is greater than the signal. Thus, depending on whether $s_Q(n)$ is positive or negative, the distribution of Fig. 1.7 or its mirror image, is used. The mean value of the quantization is zero for sign-magnitude truncation, but the variance is $\Delta^2/3$, or four times that of either rounding or straight truncation. Because of the statistical considerations above, in most practical situations rounding is
to be preferred to the other rules for quantizing a signal [1]. Since the filter is linear, the output is the sum of two components, one due to the signal \( s(n) \) and the other noise \( e(n) \). This is shown in Fig.1.8. Based on linearity, one can process the sequences \( s(n) \) and \( e(n) \) independently. Thus the output sequence (assuming infinite precision processing) can be represented as

\[
y(n) = \frac{s(n) \ast h(n) + e(n) \ast h(n)}{\text{signal}} + \frac{e(n) \ast h(n)}{\text{noise}} \quad \quad (1.27)
\]

where \( h(n) \) is the impulse response of the system.

![Fig. 1.8: Linear Model of quantization noise in an LTI system.](image)

All analog signals are corrupted by some form of noise - i.e. they have a finite signal-to-noise ratio. Thus the use of a large number bits to represent such a signal does not give a digital signal with a higher signal-to-noise ratio than the analog signal. Thus, the word length of an A/D converter is dictated by the signals being converted.

### 1.4.4 Effect of Coefficient Inaccuracy

Another effect of finite word length is coefficient quantization error which occurs when the coefficients of a digital filter, initially specified with unlimited accuracy, are quantized by rounding or truncation. As a consequence the frequency response of the actual filter which is realized deviates from that which would have been obtained with an infinite word length representation. Coefficient length is an important parameter since the amount of required multiplier hardware is more or less proportional to
the number of coefficient bits needed. Different filter forms can differ
substantially in the amount of coefficient accuracy required to achieve
a given filter transfer function within a specified tolerance. In
particular, the direct form suffers in this respect when compared to other
forms. For example, the parallel and cascade forms require the same number
of adders, multipliers and delay elements as the direct form, but they
generally produce filters with shorter coefficient words for the same
transfer function tolerance. Another form which appears to be particularly
thrifty of coefficient word length is the wave digital filter of Fettweis
[24], although this form generally requires more adders and memory elements
than either the parallel or the cascade form. Typical values of coefficient
word length are 8-12 bits for parallel, cascade, and FIR filters and somewhat
less (perhaps as low as 1-4 bits) for the wave digital filters [9].

There are two general approaches to the analysis and synthesis of
digital filters with finite precision coefficients. The first approach is
to treat coefficient quantization errors as intrinsically statistical
quantities. In this case the effects of coefficient quantization can be
represented by a stray transfer function in parallel with the corresponding
ideal filter. Also by making certain assumptions about the coefficient
errors, the expected mean square difference between the frequency responses
of the actual and ideal filters can be readily evaluated.

The second approach to coefficient quantization is to study each
individual filter separately. In this manner one can optimize the finite
precision coefficients to minimize the maximum weighted difference between
the ideal and actual frequency responses. Although one cannot make any
general statements about coefficient quantization, this method has the
benefit of yielding the best finite precision representation of the desired
frequency response.
1.4.5 Limit Cycle Oscillations [25]

If the data sequence to a recursive realization of a digital filter, implemented with finite-register-length arithmetic, consists of constants (e.g. zero) or some other periodically repeating samples, the roundoff or truncation error is periodic and the output may decay into a nonzero amplitude range. This effect is often referred to as limit cycle behaviour and is a consequence of the nonlinear quantizers in the feedback loop of the filter. The limit cycle behaviour of a digital filter is complex and difficult to analyze. The effect is best illustrated by means of an example.

Consider the first order system characterized by the difference equation

\[ y(n) = 0.95 \, y(n-1) + x(n) \ldots \] (1.28)

with input \( x(n) = 0 \) (i.e., the input is turned off) and initial condition \( y(-1) = 13 \). (The values of the variable \( y \) are expressed as integers times the quantization step \( \Delta \)). Assuming rounding of the product, the actual output \( w(n) \) satisfies the nonlinear difference equation

\[ w(n) = \lfloor 0.95 \, w(n-1) \rfloor \cdot x(n) \] (1.29)

where \( \lfloor \cdot \rfloor \) denotes the operation of rounding to the nearest integer with \( x(n) = 0, \, n > 0 \). Thus the variable \( y \) can only take on integer values. The table below shows a comparison between exact values of \( y(n) \) as obtained from Eq. (1.28) using infinite precision arithmetic and rounded values of \( y(n) \) as obtained from finite precision arithmetic [21].
<table>
<thead>
<tr>
<th>n</th>
<th>y(n) - Exact</th>
<th>y(n) - Rounded</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>13,0</td>
<td>13</td>
</tr>
<tr>
<td>0</td>
<td>12,35</td>
<td>12</td>
</tr>
<tr>
<td>1</td>
<td>11,7325</td>
<td>11</td>
</tr>
<tr>
<td>2</td>
<td>11,145875</td>
<td>11</td>
</tr>
<tr>
<td>3</td>
<td>10,58858125</td>
<td>10</td>
</tr>
<tr>
<td>4</td>
<td>10,05915219</td>
<td>10</td>
</tr>
</tbody>
</table>

It is clear that even though the exact value of \( y(n) \) decays exponentially to zero, the rounded value of \( y(n) \) stays trapped at the value 10 and can never get any smaller. This is an example of a limit cycle that occurs at the output of a recursive realization of a digital filter in response to zero input. The amplitude intervals in such limit cycles are confined have been called deadbands by Blackman [25]. The deadband in the example above is the interval \((-10, 10)\). Jackson has studied limit cycles in first- and second-order systems using an "effective value" model — i.e., realizing that limit cycles can only occur if the result of rounding effectively leads to poles on the unit circle. Thus in the case of the first-order difference equation

\[
y(n) = x(n) - [\alpha y(n-1)]',
\]

the deadband in which limit cycles can exist is the region \([-k, k]\) with \( k \) the largest integer satisfying

\[
k < \frac{0.5}{1-|\alpha|}.
\]

If \( \alpha \) is negative, the limit cycle is of constant magnitude and sign. If \( \alpha \) is positive, the limit cycle is of constant magnitude and alternating sign. For all values of \( y(n) \) in the deadband, the multiplier \( \alpha \) has the effective value \( +1 \), i.e., \( [\alpha y(n-1)]' = y(n-1) \). Therefore the difference
equation of (1.31) has an effective pole at \( z = \pm 1 \). For the second-order system with difference equation

\[
y(n) = x(n) - [\beta_1 y(n-1)]' - [\beta_2 y(n-2)]' \quad \cdots \\
\]  

(1.32)

one can consider a deadband in which limit cycles can occur as the region \([-k, k]\) where \( k \) is the largest integer satisfying

\[
k \leq \frac{0.5}{1 - \beta_2} \quad (0 < \beta_2 < 1) \quad \cdots \\
\]  

(1.33)

The effective value of \( \beta_2 \) is 1.0. (Note \( \beta_2 > 0 \) for complex conjugate, stable poles.) The oscillation frequency of the limit cycle is controlled primarily by the value of \( \beta_1 \) but is also somewhat dependent on how the rounding affects the product \( \beta_1 y(n-1) \) in Eq. (1.32). The other possibility in the second-order section is to have a real effective pole at \( z = \pm 1 \).

The condition for a limit cycle of amplitude \( k \) to occur is

\[
[\beta_2 k]' + [\beta_1 k]' = k \quad \cdots \\
\]  

(1.34)

There are two important reasons for studying limit cycles. In a communication environment the idle channel condition can lead to limit cycles that are extremely undesirable in that one would like to hear no signal over a line when no signal is put in. Thus when digital filters are used in the telephone plant, extensive care must be given to this problem. The second important reason for studying limit cycles is their potential application to the design of digital periodic waveform generators. By creating desirable limit cycles in a reliable manner, one can use the limit cycles as a source in digital signal processing,
1.4.6 **Overflow Oscillations** [22]

Overflow occurs when a digital filter computes a number that is too large to be represented in the arithmetic used in that filter. If no compensation is made for the overflow then large errors in the filter output can result either in the form of transients or of overflow oscillations (i.e., the filter output oscillates between the maximum amplitude limits).

A technique used to compensate in part for overflow is saturation Arithmetic where a sum that is too large to be represented is set equal to the largest representable number in the filter. The problem of oscillations caused by overflow is discussed in detail by Ebert et al [22]. Once the overflow oscillation has started, there is no sure way of stopping it short of turning off the power.

We note here that the above discussion was concerned only with zero-input limit cycles, owing to rounding in first-and second-order IIR systems. Although the analysis was somewhat heuristic, the simple formulas that result have been found to be consistent with experimental results and are useful in predicting limit cycle behaviour in IIR digital filters. A similar style of analysis can be used for truncation. In the case of parallel realizations of higher-order systems, the outputs of the individual second-order systems are independent when the input is zero. Thus the previous analysis can be applied directly. In the case of cascade realizations, only the first section has zero input. Succeeding sections may exhibit their own characteristic limit cycle behaviour or they may appear to be simply filtering the limit cycle output of a previous section. For higher-order systems realized by other filter structures, the limit cycle behaviour becomes more complex, as does the analysis thereof.

When the input is nonzero, the quantization effects depend upon the input, and the style of analysis of this section is completely inappropriate.
except for simple inputs such as a unit sample, a unit step, or a sine wave. In other cases, the complexity of the quantization phenomena force us to a statistical model.

The style of analysis developed in this section can also be applied to the study of quantization effects in FIR digital filters. In some respects this analysis is simpler than the analysis of IIR filters. For example, there are no limit cycle effects with non-recursive realizations such as the direct form or cascade form, since these structures have no feedback. However, recursive realizations of FIR systems such as the frequency-sampling structure are subject to the problems discussed in this section, and the analysis of second-order systems applies to the frequency-sampling form. Dynamic range and roundoff noise are important considerations in FIR systems just as they are in IIR systems.
REFERENCES


CHAPTER 2

MODULAR APPROACH TO THE DESIGN OF DIGITAL SIGNAL PROCESSOR

2.1 INTRODUCTION:

The term "digital filter", or "digital processing machine", refers to any device which operates on input number sequence to produce a second sequence of numbers by means of computational algorithm. If a digital filter is part of a signal processing system, the input number sequence is usually the digital version of an analog signal. The output sequence is usually converted to the analog form. The advantages of coding a signal digitally are well known. Briefly, digital representation offers ruggedness, efficient signal regeneration, easy encryption, the possibility of combining transmission and switching functions, and the advantage of a uniform format for different types of signals. The price paid for the benefits is the need for increased bandwidth [1].

Until recently, signal processing has typically been carried out using analog equipment. Because of the flexibility of digital computers, it was often useful to simulate a signal processing system on a digital computer before implementing it in analog hardware. The rapid development in the technology of medium and large scale integrated circuits is also making it possible to construct special-purpose hardware for real-time digital signal processing.

Up till now, higher-order digital filters are normally realized as either a cascade or parallel network of basic second-order section [2] [3], which in the former case, are ordered for minimum round-off noise and have outputs suitably scaled [6], [7]. Also, the actual hardware synthesis, reported in the literature, is usually at the discrete gate level, and structures proposed are mainly for specific configurations.
In this chapter, a new technique for analysis and synthesis of digital signal processor is proposed. This approach is modular, and flexible. The technique is based on the use of a simple mapping (mapping of differentials) procedure to transform a continuous-time filter to its corresponding digital filter.

The characteristic equation of the processor takes a modular form. This gives the possibility of designing a module which can be used to build any higher order digital processor. The output of the module is the first difference of its input, as well as the product of that difference with the coefficient. The modules are connected to realize the difference table which, in turn, satisfies the characteristic equation of the processor. More than one design for the module is considered. A comparison between these different designs from the viewpoints of computation time and equipment is also given. The main advantage of this approach is the possibility of using the same module for building a digital system in which the full sample values, as well as incremental changes are used in processing.

Beside the modular nature of the proposed technique, it has another advantage - the possibility of using it for processing directly a code differential pulse/modulated (DPCM) signal. The importance of this advantage will be clear if we note that in case of using digital filters as an intermediate stage in a communication channel which uses DPCM signal, we must convert at first the DPCM signal to PCM signal, processed it using conventional digital filter, and then to reconvert the resultant PCM signal to its equivalent DPCM.

Using the proposed technique, there will be no need for converters, since the filter will process directly the DPCM signals and its output will be also a difference signal.
In section 2, the design approach using mapping of differentials will be considered. The proposed hardware implementation is the subject of section 3 and section 4. More than one module will be given in this section. A comparison between the different structures is given in section 5. The hardware requirements as well as the computation time is taken as base of the comparison.

2.2. DESIGN APPROACH - MAPPING OF DIFFERENTIALS:

The problem of designing filters is one of finding filter coefficients such that some aspects of the filter's response (e.g., time response, frequency response, group delay, etc.) approximates a desired behaviour in a specified manner. If the approximation problem is solved in the \( z \)-plane, the resulting filter is a digital filter. If it is solved in the \( S \)-plane, the resulting filter is an analog filter. The theory and techniques of approximation in the \( S \)-plane is well established in the literature. Then, instead of redeveloping these theories for digital filters, simple mapping procedures can be used to transform filters from one domain to filters in the other domain. The technique of designing an appropriate continuous-time filter and digitizing the resulting design to give a digital filter is very useful for designing standard filters such as lowpass, bandpass, stoppass, and high-pass filters.

One of the simplest ways of digitizing a continuous system is to replace differentials in the differential equation of the continuous system with finite differences in order to obtain a difference equation that approximates the given differential equation. The differential equation of a filter is:

\[
\sum_{i=0}^{N} a_i \frac{d^i y(t)}{dt^i} = \sum_{i=0}^{M} b_i \frac{d^i x(t)}{dt^i} \quad \ldots \]

\[(2.1)\]
The corresponding difference equation will be:

\[ \sum_{i=0}^{N} a_i \Delta_i [y(n)] = \sum_{i=0}^{M} b_i \Delta_i [x(n)] \]  

where:

- \( x(n) \) is the input sequence to the filter;
- \( y(n) \) is the output sequence;
- \( \Delta_i [w(n)] \) is the \( i \)th difference defined by;

\[ \Delta_{i+1} [w(n)] = \Delta_i [\Delta_i [w(n)]] \]

and

\[ \Delta_1 [w(n)] = \begin{cases} \frac{1}{T}[w(n) - w(n-1)] & \text{for backward difference} \\ \frac{1}{T}[w(n+1) - w(n)] & \text{for forward difference} \end{cases} \]

The two desirable properties of any mapping from continuous to discrete space are:

1. The \( j\Omega \) axis in the \( S \)-plane should be mapped to the unit circle in the \( Z \)-plane;
2. Points in the left-half of the \( S \)-plane \( \{ \text{Re}[s] < 0 \} \) should be mapped inside the unit circle \( \{ |z| < 1 \} \).

The use of backward differences satisfies the second property while the first property is not satisfied [4]. Therefore stable analog filters map into stable digital filters using backward differences but their frequency selection properties are not maintained. Forward difference method also, does not satisfy both properties. The use of higher-order differences to replace low-order differentials tends to preserve the two desirable properties of mapping. For example, consider the case of defining \( \Delta_1 \) of equation (2.4) by the expression;
\[ \Delta_N[w(n)] = \frac{1}{T} \sum_{i=1}^{L} c_i [w(n+i) - w(n-i)] \]  \hspace{1cm} (2.5)

where, \( L \) is the order of difference to be used.

The mapping between S-plane and Z-plane then becomes:

\[ S = \frac{1}{T} \sum_{i=1}^{L} c_i (z^i - z^{-i}) \]  \hspace{1cm} (2.6)

by setting \( Z = \exp(j\omega T) \) in equation (2.6) we obtain

\[ S = \frac{1}{T} \sum_{i=1}^{L} c_i (e^{j\omega Ti} - e^{-j\omega Ti}) \]

\[ = \frac{1}{T} \sum_{i=1}^{L} 2j c_i \sin(\omega iT) = j\beta(w) \]  \hspace{1cm} (2.7)

This result means that, points on the unit circle in the Z-plane \((Z = e^{j\omega T})\) are the mappings of points on the \( j\beta \)-axis in the S-plane \((S = j\beta(w))\). By proper choice of the coefficients \( c_i \), the function \( \beta(w) \) can approximate almost any desired odd function in the S-plane monotonically to the unit circle in the Z-plane. Furthermore, the mapping of equation (2.6) can be shown to be conformal, thereby mapping the left-half of the S-plane to the inside of the unit circle in the Z-plane. For such cases both desirable properties of the mapping are preserved.

The details of selecting the coefficients \( c_i \) that make the function \( \beta(w) \) approximates a given frequency characteristics is out of the scope of this thesis; since we are concerned here with the phase of the hardware implementation of the filters. Some methods of calculating the coefficients are considered in chapter 1; more information are given in ref. [4],[5].

In all the existing works, after calculating the filter coefficients and for the implementation purposes, the difference equation (2.2) is usually converted to the classical non-recursive form:
\[ y(n) = \sum_{i=1}^{N} a_i x_{n-i} \]  
\( (2.8a) \)

or to the recursive form:

\[ y(n) = \sum_{i=0}^{N} a_i x_{n-i} - \sum_{i=1}^{M} b_i y_{n-i} \]  
\( (2.8b) \)

by using either equation (2.4) or (2.5). This is not the method used in this thesis, here, equation (2.2) will be used directly for the hardware implementation for the following reasons:

1. The aim of this thesis is the design of multiplication free digital filter by using mainly the difference signal in processing instead of the full sampled value.

   The multiplication free filter is realized actually (chapter 3) by using one-bit to represent the difference word. Then to reach this limit we discuss at first the case of using any arbitrary word-length for the difference.

2. Equation (2.2) gives the possibility of using modular approach for realizing the digital filter. We can mention here that, if the classical form of the filter equation (Equation 28a) is suitable for processing pulse code modulated (PCM) signals, and if the form discussed in chapter 3 is suitable for processing delta modulated signals, then the form \((2.2)\) is that suitable for implementing a filter whose input is a differential pulse code modulated signal (DPCM).

2.3. Hardware Implementation [8]

Equation (2.2) is the base of the proposed hardware implementation. This equation means that, the implementation of the \(N^{th}\) order digital filter, needs a number of similar modules. The output of the module is the first difference of its input signal. If the input signal is:
\[ I(n) = \Delta_1[w(n)] \quad \ldots \quad (2.9a) \]

the output will be:

\[ O(n) = \Delta_{i+1}[w(n)] \quad \ldots \quad (2.9b) \]

The hardware of the module depends on the method used to calculate the first difference \( \Delta_1 \). In this section, some of the possible modules are examined.

2.3.1 The Use of Backward Difference:

The algorithm of operation of the module is:

\[ O(n) = \Delta_1[I(n)] = [I(n) - I(n-1)] \quad \ldots \quad (2.10) \]

The hardware realization of the module based on the above equation is given in Fig.2.1. The summation in equation (2.2) consists of terms of the form \( a_1 \Delta_1[w(n)] \). To get these terms, the coefficient \( a_1 \) is used as the second input to the module. In turn, the module has two outputs; one of them \( \Delta_{i+1}[w(n)] \) is used in the summation to get the final result, and the other output \( \Delta_{i+1}[w(n)] \) is used as input to the preceding module.

The module consists of a delay element, an adder, and a multiplier.

2.3.2 The Use of Forward Difference

The algorithm of operation of the module is:

\[ O(n) = \Delta_1[I(n)] = [I(n+1) - I(n)] \quad \ldots \quad (2.11) \]

In this case, a prediction is necessary to predict the value of \( I(n+1) \).

Let us first examine how to construct a predictor. Using Taylor series expansion for \( y(t + h) \) we get:

\[ y(t+h) = y(t) + hy' + \frac{1}{2!} h^2 y'' + \ldots \quad (2.12) \]

\[ y(t+h) = a_0 y_0 + a_1 y_1 + a_2 y_2 + a_3 y_3 + \quad (2.13) \]

where \( y_0 = y(t) \)

\( y_1 = y(t-h) \)

\( y_2 = y(t-2h) \) etc. and
Figure 2.1. The Module Using Backward Difference.
\( a_0, a_1, a_2 \ldots \) are the coefficients of the prediction routine. So (2.13) equation becomes:

\[
y(t+h) = a_0 y + \\
+ a_1 (y - hy' + \frac{h^2}{2!} y'' - \frac{1}{3!} h^3 y''' \ldots) + \\
+ a_2 (y - 2hy' + \frac{4h^2}{2!} y'' - \frac{8h^3}{3!} y''' \ldots) + \\
+ a_3 (y - 3hy' + \frac{9h^2}{2!} y'' - \frac{27h^3}{3!} y''' \ldots) + \\
+ \ldots \quad \ldots \quad \ldots \quad (2.14)
\]

To construct a predictor of any chosen order, we simply equate terms in equations (2.12) and (2.14). For instance, zero-order prediction is given by:

\[
y(t+h) = y(t) \quad \ldots \quad (2.15)
\]

A first-order predictor gives:

\[
y + hy' + \frac{h^2}{2!} y'' + \ldots = a_0 y + a_1 (y - hy' + \ldots) \quad (2.16)
\]

and equating terms in \( y \) and \( y' \) in equation (2.16) we have:

\[
\begin{align*}
a_0 + a_1 &= 1 \\
a_1 &= -1 \\
\ldots \quad a_0 &= 2
\end{align*} \quad (2.17)
\]

Thus \( y(t+h) = 2y(t) - y(t-h) \ldots \quad (2.18) \)

The second-order predictor gives:

\[
y + hy' + \frac{1}{2!} h^2 y'' = a_0 y_0 + a_1 y_1 + a_2 y_2 \\
= y(a_0 + a_1 + a_2) + hy' (-a_1 - 2a_2) + \\
+ \frac{1}{2} h^2 y'' (a_1 + 4a_2) \quad \ldots \quad (2.19)
\]
Hence equating terms in equation (2.19)

\[
\begin{align*}
  a_0 + a_1 + a_2 - 1 &= 0 \\
  a_1 - 2a_2 &= 1 \\
  a_1 + 4a_2 &= 1
\end{align*}
\]

therefore

\[
a_2 = 1, \quad a_1 = -3 \quad \text{and} \quad a_0 = +3
\]

i.e.

\[
y(t + h) = 3y(t) - 3y(t - h) + y(t - 2h)
\]  \(\text{(2.21)}\)

Hence in our case, for the first order prediction and using eq. (2.18) we have

\[
I(n+1) = 2I(n) - I(n-1)
\]  \(\text{(2.22)}\)

therefore:

\[
\Delta_1 I(n) = I(n+1) - I(n) = I(n) - I(n-1)
\]  \(\text{(2.23)}\)

This is the same as the backward difference method. The module, in turn, has the same shape as that given in Fig. 2.1.

For the second-order prediction and using equation (2.21) we have:

\[
I(n+1) = 3I(n) - 3I(n-1) + I(n-2)
\]  \(\text{(2.24)}\)

and

\[
\Delta_1 I(n) = 2I(n) - 3I(n - 1) + I(n - 2)
\]  \(\text{(2.25)}\)

The hardware realization of the module, which is based on this equation is given in Fig. 2.2. The module has two inputs \(I(n)\) and \(a\); and two outputs \(\Delta_1 I(n)\) and \(a\Delta_1 I(n)\). Two delay elements are used to get \(I(n-1)\) and \(I(n-2)\); an adder, and \(\times a\) multipliers are also needed to build up the module.
Figure 2.2. The Module Using Forward Difference (second-order prediction)
2.3.3 The Use of Higher Order Difference.

The algorithm of operation of the module is:

\[ 0(n) = \Delta_1 [I(n)] = \frac{1}{L} \sum_{i=1}^{L} c_i [I(n+i) - I(n-i)] \] ........ (2.26)

where \( L \) is the order of difference. Also in this case, a prediction is necessary to predict the value of \( I(n+i) \). The prediction can be for any order.

i - For first order prediction we get:

\[ \Delta_1 [I(n)] = 2 I(n) \sum_{i=1}^{L} c_i - 2 \sum_{i=1}^{L} c_i I(n-i) \] ........ (2.27)

ii - For second order prediction we have:

\[ \Delta_1 [I(n)] = \sum_{i=0}^{2L} k_i I(n-i) \] ........ (2.28)

where, \( k_0 = 3 \sum_{i=0}^{L} c_i \) ........ (2.29)

and the relation between \( k_i \) (i =1,2,3,...) and \( L \) is given in table 2.1

<table>
<thead>
<tr>
<th>L /k_i</th>
<th>k_1</th>
<th>k_2</th>
<th>k_3</th>
<th>k_4</th>
<th>k_5</th>
<th>k_6</th>
<th>k_7</th>
<th>k_8</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-4c_1</td>
<td>c_1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>-4c_1</td>
<td>c_1-4c_2</td>
<td>0</td>
<td>c_2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>-4c_1</td>
<td>c_1-4c_2</td>
<td>-4c_3</td>
<td>c_2</td>
<td>0</td>
<td>c_3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>-4c_1</td>
<td>c_1-4c_2</td>
<td>-4c_3</td>
<td>c_2-4c_4</td>
<td>0</td>
<td>c_3</td>
<td>0</td>
<td>c_4</td>
</tr>
</tbody>
</table>

Table 2.1

The hardware realization of the module for the second order prediction and for \( L = 4 \) is given in Fig. 2.3. It consists of two delay elements, four multipliers, and one adder.
Figure 2.3. The Module Using Higher Order Difference.
2.3.4. **Difference Table Generation Method.**

The right-hand side, as well as the left-hand side of the difference equation (2.2) consists of the differences situated in the upper line of the difference table \[ y(n), \Delta_1 y(n), \Delta_2 y(n), \ldots, \Delta_N y(n) \]. To get the value of the summation at any instant \( n \), the difference table at that instance, must be calculated. As a matter of fact, there is no need to calculate all the terms of the difference table each time we need summation. All the terms at instance \( n \), except those appearing in the upper line, can be generated by simple down shift of the terms at instance \( n-1 \). As a result, at any instant it is enough to calculate the new values of the terms of the upper line. This needs a module which realizes the equation;

\[
O(n) = I(n) - I(n-1)
\]

(2.30)

if \( I(n) = \Delta_{i-1} y(n) \), \( I(n-1) = \Delta_{i-1} y(n-1) \), then

\[
O(n) = \Delta_{i-1} y(n) - \Delta_{i-1} y(n-1)
\]

(2.31)

The hardware realization of the module based on the above equation is given in Fig. 2.4. The module consists of two registers to store the values of \( I(n) \) and \( I(n-1) \). It consists also of one adder and a multiplier.

2.4. **Realization of Higher Order Digital Filter**

The realization of an \( N^{th} \) order digital filter using any of the proposed modules is shown in Fig. 2.5. It needs \( (N + M) \) modules where \( N \) and \( M \) are as in equation (2.2). The decision to use any of the proposed modules is affected by the computation time of the filter and the required hardware comparison of the various implementation methods is given in table 2.2.
Figure 2.4. The Module in Case of Using Table of Differences.
<table>
<thead>
<tr>
<th>Method</th>
<th>Adders</th>
<th>Multipliers</th>
<th>Delays</th>
<th>Computation Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Backward</td>
<td>(M+N+3)</td>
<td>(N+M)</td>
<td>(N+M)</td>
<td>((N+2)T_a + T_m) if (T_a &lt; NT_m) or (2T_a + T_m) if (T_a &gt; NT_m)</td>
</tr>
<tr>
<td>Forward (second order prediction)</td>
<td>(M+N+3)</td>
<td>2(N+M)</td>
<td>2(N+M)</td>
<td>(N(2T_a + T_m) + T_m)</td>
</tr>
<tr>
<td>Higher Order (Second order prediction)</td>
<td>(N+M+3)</td>
<td>2(N+M)</td>
<td>2(N+M)</td>
<td>(N+1)(2T_a + T_m))</td>
</tr>
<tr>
<td>Difference table</td>
<td>(N+M+3)</td>
<td>(N+M)</td>
<td>2(N+M)</td>
<td>((N+2)T_a + T_m)</td>
</tr>
</tbody>
</table>

Table - 2.2.

\(T_a\) - one addition time, \(T_m\) - one multiplication time

The use of the configuration of Fig. 2.5 needs \((N+M)\) multipliers as given in Table 2.2. This makes the given configuration impractical. This is more so if fast multipliers are to be used.

Considering the fact that in any difference table, the formation of the \(n^{th}\) difference \(\Delta_n\) comes only after the formation of the lower order differences, which means that the system is a serial system. As a result, it is possible to take out the multiplier from each module and use only one multiplier for the whole system. As a result the module will consist of only delay elements (or a register) and adders. The number of delay elements and adders depends on the module used. If backward differences are used, the formation of the differences \(\Delta_1(n), \Delta_2(n), \ldots \ldots \Delta_N(n)\) is shown in Fig. 2.6.

The block diagram needs \(N\) delay elements and \([N + (N-1) + (N-2) + \ldots + 2 + 1]\) adders. The number of adders can be reduced to minimum if we take into consideration, as given in paragraph 3.4, that equation (2.2) contains only the
Figure 2.5. The Hardware Realization of $N^{th}$ Order Digital Filter.
Figure 2.6: Formation of the difference table using delay elements and Adders.
upper row of the difference table. As a result the configuration of
Fig. 2.6 can be modified to that given in Fig. 2.7. This system contains
N delay elements and N adders only.

The corresponding digital filter is shown in Fig. 2.8. A memory
unit is used to store the filter coefficients \( a_1, a_2, a_3, \ldots, a_N \); a
multiplier is used to multiply the coefficients with the corresponding
differences and the output of the multiplier is accumulated to get the
final result \( y_n \).

2.5. CONCLUSIONS:

A method for designing a general digital filter has been presented.
The resulting hardware structures are modular, and have uniform inter-
connection patterns. The flexibility of the proposed technique should
make possible, the economical design of special purpose digital filter
hardware for any application including those requiring real-time processing.
Figure 2.7. Generation of the first row of the difference table.
Figure 2.8. The Proposed Hardware Realization of an $N^{th}$ Order Filter.
REFERENCES


CHAPTER 3

DIGITAL SIGNAL PROCESSING USING ONE-BIT WORD

3.1 INTRODUCTION

The use of delta modulators as a source encoders has been emphasized in recent times. Delta modulation has been used by Goodman [1] to convert an analog signal into a pulse-code-modulated (PCM) Signal. Lockhark [2] - [4] and Franks [5] also used delta modulation for filtering an analog signal by hybrid arrangements. Engel [6] and Rafila [7] have presented a direct arithmetic method for processing delta modulated signals and its application in realizing non-recursive digital filters. Their approach and that of Peled and Liu [8] uses Read-only-Memory (ROM) to permit very fast digital filtering. This method does not provide any easy way for designing a multiplier and thus cannot have a general use. The problems of using ROM in such a field of application are discussed in [7] and [9].

The Digital Differential analyser (DDA) is a single-bit digital machine (computing the changes in variables rather than of variables) was originally developed in 1949 in order to provide a relatively inexpensive digital computer for the solution of differential equations [10,11]. Not only the input and output signals of such a machine are single-bit words, processing in the machine also takes place directly using these single-bit word signals. Digital filters using incremental computers [12,13], or numerical integration techniques [14] have been briefly discussed. It has also been shown in [15] that digital filters implemented by digital incremental computers are matched to transfer functions which have all their poles close to \( z = 1 \). Such a filter structure is said to have a low sensitivity, good error characteristics, and simple hardware implementation.

1. Also referred to as Digital incremental computer (DIC).
In this chapter we present a new treatment of the completely digital addition, substraction, and multiplication by a constant, of DM Signals [27]. Beside the theoretical treatment, the hardware implementation for these operations are given. Furthermore, with the suggested multipliers, it is possible to realize multiple-multipliers, which yield simultaneously the product of DM signal with many coefficients. The method suggested also provides information concerning the errors introduced by the operation.

For the hardware implementation we suggest the use of the digital servo and the digital multiplying integrator. These are the two basic elements of a digital differential Analyser. A modified algorithm for the digital servo is proposed in order to reduce to minimum the resulting error. The error will be only one quantization step in adding two delta sequences.

The use of the proposed technique to build up a nonrecursive, as well as recursive digital filter is also discussed. The characteristics of the filter is calculated by simulating the proposed hardware on IBM 370/145 digital computer. The results of the simulation are also given.

In section 2, a short note on the delta modulation system is given. In section 3, the proof of the possibility of using DM in processing is treated. In section 4, some methods of implementing the one-bit word digital filter are given. In section 5, the basic building blocks of a the digital filter using/proposed algorithm are discussed. In section 6, we present a theoretical treatment of the completely digital addition and digital multiplication of DM signals. The method suggested also provides information concerning the errors introduced by the operation. The realization of non-recursive as well as recursive digital filters using the proposed algorithm is presented in section 7 and results and conclusions
are presented in section 8.

3.2. **Delta Modulation (Single-bit word) System**

For the purpose of this chapter, the linear delta modulator is the encoding device which transforms the analogue signal into a suitable binary form for digital filtering. Consider the linear delta modulation of Fig 3.1a, which transforms an analogue input signal \( X(t) \) to the binary sequence

\[
\{ W_n \} = \ldots W_{-1}, W_0, W_1, \ldots
\]  

(3.1)

where \( W_n \) takes the value +1 or -1. These values occur at intervals of \( T \) seconds, as the system sampler is controlled by a clock generator with \( 1/T \) frequency.

The modulator feed-back loop of the system includes an ideal digital integrator. The contents of the integrator will be a step function with step size equals to \( \pm \delta \) (where \( \delta \) is the quantization step). This means that, the integrated feedback output signal \( \tilde{X}(t) \), will be:

\[
\tilde{X}(nT) = \tilde{X}(t) = \delta \sum_{k=-\infty}^{n} W_k
\]

\[ nT \leq t \leq (n+1)T \]

(3.2)

(3.3)

The following relation holds for the system,

\[
X(nT) = \tilde{X}(nT) + \epsilon_1(nT)
\]

(3.4)

where \( \epsilon_1(nT) \) is the error signal introduced by the modulator. A delta demodulator, shown in Fig. 3.1b, is a repetition of the feedback loop circuit of the modulator with a low-pass filter at its output and it transforms the delta sequence \( W_n \) into its corresponding analog signal \( X_1(t) \).
Figure 3.1. Delta Modulator and Demodulator.
3.3. On the Possibility of Using DM in Processing [8].

A nonrecursive digital filter is characterized by an input-output relationship of the form

$$y_n = \sum_{k=0}^{N-1} a_k x_{n-k}$$  \(\text{(3.5)}\)

where \(\{x_n\}\) is the input sequence,

\(\{y_n\}\) the output sequence

\(N-1\)

and \(\{a_k\}\) are the filter coefficients.

Consider the processing of an analog signal by such a filter realized with b-bit word hardware. The conventional implementation of this filter is illustrated in Fig. 3.2. The analog signal is sampled and each sample is quantized and converted to a b-bit word by a PCM. The output sequence of the PCM is fed to a digital processor that computes the output samples \(y_n\) according to (3.5).

The hardware and computational requirements can be easily determined from the block diagram. We need a b-bit PCM, storage for \(N\) filter coefficients, and a digital processor capable of performing addition and multiplication. The processor performs \(N\) multiplications and \(N-1\) additions per output sample.

The realization of the filter using delta modulation technique is shown in Fig. 3.3. The input signal enters a delta modulator whose output is the sequence \(\{w_n\}\) with each \(w_n\) taking on the values +1 or -1. This sequence enters a digital processor that computes

$$\sum_{k=0}^{N-1} a_k w_{n-k}$$

and adds this sum to the content of an accumulator to obtain the output.
Figure 3.2. Realization using PCM

Figure 3.3. Realization using Delta Modulation.
sample $V_n$. This operation is described by

$$V_n = V_{n-1} + \sum_{k=0}^{N-1} a_k W_{n-k}$$  \hspace{1cm} (3.6)

If the error introduced by the delta modulator is negligible, the coefficients $\{a_k\}$ can be chosen so that the output $\{V_n\}$ agrees with $\{y_n\}$.

An error free delta modulator is characterized by the following relation between $W_n$ and samples of the input signal $X(t)$,

$$X(nT) = X[(n-1)T] + \Delta_x W_n$$ \hspace{1cm} (3.7)

where $T$ is the sampling interval, and $\Delta_x$ the step size of the delta modulation. Substituting (3.7) into (3.6), we obtain

$$V_n = \sum_{k=0}^{N-1} a_k X[(n-1)T] + C$$ \hspace{1cm} (3.8)

where $C$ is a constant, which can easily be shown to be zero if we require that the output be zero when no signal is present at the input. With that assumption we immediately see that $V_n$ equals the $y_n$ given by (3.5) provided that

$$a'_k = a_k \Delta_x$$ \hspace{1cm} (3.9)

In the preceding discussion, we have assumed that no roundoff or quantization error occurs in the arithmetic operations. Thus we see that at least in an idealized situation, this implementation scheme has the same capability as the conventional approach of Fig. 3.2

It is also possible to get the same result i.e. equation (3.9) by a completely different approach. This approach will give an idea about the hardware implementation of the one-bit word digital filter. Because of this, in the following the second approach of proving the possibility of using delta modulated signal in processing will be considered.
Consider the linear delta modulator shown in Fig. 3.4. Suppose that the encoder is not slop overloaded by the input signal \( X(t) \) and that the quantization noise is negligible. This requires a high ratio of clock rate \( f_p \) to step-size \( \Delta x \) in the feedback signal \( X(t) \) and the clock rate to be much higher than the Nyquist rate i.e. \( f_p \gg 2f_c \) where \( f_c \) is the highest frequency in \( X(t) \). For these conditions, the reconstructed signal \( X_1(t) \) closely approximates \( X(t) \). If the signal \( W_n \) at the output of the delta modulator is delayed \( rT \) seconds where \( T = 1/f_p \), before being passed to the local decoder i.e. the integrator, then the signal at the output of the integrator is \( X(t-rT) \). Suppose at \( t = nT \), the \( n^{th} \) sampling instant, and that the output of the \( rT \) delay is multiplied by a constant \( a_r \), then the output signal of the integrator is \( a_r X_{n-r} \). The input to the integrator is the signal \( a_r W_{n-r} \), which may be positive or negative and consists of narrow pulses which approximate to impulses of strength \( v_r a_r \) where \( v \) is the magnitude and \( \tau \) the duration of \( W_{n-r} \).

If the output of the delta modulator is applied to a shift register enabling \( W_n, W_{n-1}, W_{n-2}, \ldots, W_{n-N} \) signals to be produced i.e. the shift register has \( N+1 \) outputs spaced by delays of \( T \) seconds, and if each of these outputs are multiplied by coefficients \( a_0, a_1, a_2, \ldots, a_n \) respectively, added together and then integrated, the resulting signal at time \( nT \) is:

\[
V_n = \sum_{r=0}^{N} a_r X_{n-r} \quad \cdots \quad (3.10)
\]

If the values of the coefficients \( a_r \) are suitably selected, a nonrecursive filter having a required frequency response can be obtained. The arrangement of the filter to achieve equation (3.5) is shown in Fig. 3.5.
Figure 3.4. Effect of delaying the binary signal $w_n$ by $rT$ and multiplying by a constant $a_r$. 
Figure 3.5. Non-Recursive filter Realization using One-bit word.
This can also be illustrated by the block diagram shown in Fig. 3.3. In this case, the output sample \( V_n \) is also described by:

\[
V_n = V_{n-1} + \sum_{k=0}^{N-1} a_k W_{n-k}
\]

which is the same as equation (3.6). From this point, the prove of the validity of equation (3.9) is the same as given before.

3.4. Direct-form Realization of One-bit Word Non-recursive Digital Filter (16)

If the weighting sequence \( a_0, a_1, a_2, \ldots, a_N \) of the binary transversal filter shown in Fig. 3.5 is replaced by a sequence having binary coefficients then it can be stored in an \((N + 1)\) bit shift register as shown in Fig. 3.6. The change to binary coefficients means that the weighting sequence can be simply changed by resetting the shift register and reading-in a new set of data. In this way, the weighting sequence is programmable. This digital filter, in addition to being easily programmable, is simple to implement because the weighting coefficients are binary valued and these values are the same as the binary levels \( \pm V \) say, of the signal at the output of the delta modulator. Consequently, the multiplication of \( W_{n-r} \) and \( a_r \) can be accomplished by using an Exclusive-OR circuit which will produce a level \(-V\) if \( W_{n-r} \) and \( a_r \) have the same polarity and \( +V\) if they are different. The outputs of all the Exclusive-ORs are added together and then integrated to give the filtered signal at the output of the binary transversal filter. However, as the binary signal \( W(t) \) presented to the binary transversal filter is generated by a delta modulator a further integration is required to recover the filtered output. This is because \( W(t) \) is a binary representation of differential of \( X(t) \).
Figure 3.6. Programmable binary transversal filter having binary weighting coefficients. SR = Shift Register.
3.4.1 The Use of Recirculating Shift Registers

The parallel processing used in Fig. 3.6 can be replaced by a serial method using recirculating shift registers as shown in Fig. 3.7. The only disadvantage of this method is the reduction in the maximum processing rate. The advantage is that the required hardware is reduced and hence the total cost. We note here that the shift registers must be clocked at $f_R = (N + 1) f_p$ bits/s. At the end of a DM period the clock associated with the shift Register SR_a must be frozen by moving the switch to position 2. All the digits in SR_w are then moved one stage to the right and the longest stored data sample is rejected.

The switch is now in position 2 and the current delta modulator output sample is inserted into SR_w. The switch is now moved back to position 1 and both shift registers have their contents recirculated at $f_R$ bits/sec and multiplication and addition is achieved by the Exclusive OR and the first integrator respectively. The second integrator performs the DM decoding.

The parallel and serial configuration shown in Figs. 3.6 and 3.7 respectively can be incorporated in a hybrid digital filter which is able to exchange the advantages and disadvantages of these configurations in a flexible manner.

3.4.2 Direct-form realization of One-bit word Recursive digital filter

A recursive filter employs feedback to obtain the required filter characteristics. If the analogue output signal $V(t)$ in the non-recursive DM filter is subtracted from the input signal $X(t)$ such that the delta modulator now encodes the difference signal $d(t)$ (Fig. 3.8) then the arrangement is a recursive filter. In order not to restrict the values of the weighting coefficients, it is necessary band-limit $V(t)$ to the same pass-band as $X(t)$, thereby ensuring that $d(t)$ does not contain any frequencies higher than those of $X(t)$. This is achieved by placing a low-pass filter $u(z)$ in the feedback path of the recursive DM filter Fig. 3.8.
Figure 3.7. Programmable binary transversal filter with Serial Processing.
Figure 3.8. DM Recursive Filter at Sampling Instants.
To meet all the required conditions the pulse transfer function of the Low Pass filter (LPF) \( U(z) \) should be of the form:

\[
U(z) = U_o + U_1 (z + z^{-1}) + U_2 (z^2 + z^{-2}) + \cdots + U_n (z^n + z^{-n}). \quad (3.12)
\]

From the above equation it can be seen that the LPF has an impulse response which commences before the impulse has been applied i.e. it has a non-causal response. Fig. 3.8 shows the DM recursive filter as a discrete time network. The delta modulator is assumed to have nullified the integrator. The loop equation is:

\[
X(z) + D(z) G(z) U(z) = D(z) \quad (3.13)
\]

The pulse transfer function of the recursive DM filter is therefore given by:

\[
\begin{align*}
\frac{D(z)}{X(z)} & = \frac{1}{1 + U(z) G(z)} \\
& = \frac{1}{1 + U(z) \sum_{r=0}^{N-1} a_r z^{-r}} \quad (3.14)
\end{align*}
\]

Fig 3.9 illustrates one possible form of DM recursive filter. The signal at the output of the second integrator is a close approximation of the input signal. A recursive filter using recirculating shift registers is also possible.

Higher order digital filters are usually implemented by a network of elementary sections, [26] such a filter section can be constructed for DM filtering using the recirculating shift register technique just described. The arrangement is shown in Fig. 3.10. Where a delta-sigma modulator is used to obtain a delta modulated output thereby avoiding the necessity of using a double integrator. The non-recursive subsection accepts a delta modulated signal, which may be the output from a previous section, and produces a filtered analogue signal which is applied to the recursive
Figure 3.9. DM Recursive Filter.
Figure 3.10. Elementary Filter Section.
section. The data in the shift registers is recirculated in one DM clock period T, and once every T seconds the delta-sigma modulator accepts an analogue sample and outputs a binary sample into the shift register SRc (Fig. 3.10). The elementary filter section is completely digital and the contents of the shift registers SRb and SRd Fig. 3.10; are readily changed since the weighting sequence is programmable. The implementation is simple and the integrator could be replaced by an up-down counter. The filter section requires two Exclusive - ORs and two up-down counters, By increasing the recirculating clock rate generally by a factor of two, only one exclusive - OR and an up-down counter need be used, In the first half of the DM period the Exclusive - OR and the up-down counter service the recirculating shift registers in the non-recursive section; in the second half of the DM period the remaining shift registers in the recursive section are attended to.

3.4.3: Other Methods of Implementing the one-bit word digital filter

Two possible mechanizations of (3.6) are presented in Figs. 3.11 and 3.12. Both configurations will yield exactly the same output, but differ in storage requirements and speed of operation.

In Fig. 3.11, the input analog signal X(t) is passed through a single integration delta modulator whose output is \{Wn\} with each Wn taking on +1 or -1 only, The sequence \{Wn\} enters the arithmetic unit that computes in a straightforward manner the increment of the output according to

\[ \Delta V_n = \sum_{j=0}^{N-1} a_j W_{n-j} \quad \ldots \ldots \quad (3.15) \]

This increment is added to the previous value of the output to form the current output

\[ V_n = V_{n-1} + \Delta V_n \quad \ldots \ldots \quad (3.16) \]
Figure 3.11. Hardware realization of non-recursive digital filter using delta modulation.
The configuration of Fig. 3.12 has no arithmetic unit since the increment \( \Delta V_n \) can take only a finite number \( (2^N) \) of values, depending on the array \( \{w_{n-j}\}_{j=0}^{N-1} \). These \( 2^N \) possible values of \( \Delta U_n \) can be stored in a ROM at addresses that correspond to the binary word obtained from \( \{w_{n-j}\}_{j=0}^{N-1} \) by substituting 0 for -1. The value of the increment \( \Delta V_n \) can be obtained once \( w_n \) is in the shift register. So to obtain the output \( V_n' \), only one addition is needed. The increase in the speed of operation is significant. However, there is a serious shortcoming in this configuration, since the exponential dependence of the storage capacity needed on the number of filter coefficients makes the storage requirement considerable even for a modest number of filter coefficients.

It should be pointed out, that for linear phase filter the coefficients \( \{a_{n-j}\}_{j=0}^{N-1} \) are symmetric. Therefore, only \( 2^{(N+1)/2} \) distinct values of the increment \( V_n \) need to be stored. Some other methods of reducing the volume of storage required to implement a digital filter have also been suggested in the literature [9].

In the remaining sections of this chapter and to avoid the storage capacity, we suggest a new approach to the implementation of a digital filter using delta modulation technique. The basic building block of such a filter is first discussed before treating the filter design itself.


The hardware implementation of the one-bit word digital filter as discussed in the above paragraph faces the problem of the storage capacity or/and that of the computation speed. To avoid this problem we propose here the use of the digital incremental machines (DIM) to implement the one-bit word digital filter. A new way of using the two basic blocks (the digital integrator and the digital servo), of the DIM is used. The proposed way results in a higher accuracy. For this, the digital integrator
Figure 3.12. A very fast ROM realization of a non-recursive digital filter.
and the digital servo will be considered at first and then the use of these blocks for the implementation of the digital filter will be given.

3.5.1 The digital Integrator Unit.

The digital integrator unit (the basic building block of the digital differential analyser DDA) is essentially a fixed-program device employing a simple quadratic scheme. An integrator operates cyclically upon incremental inputs to produce incremental outputs. During any cycle, say the \(i\)th, the integrator receives a primary increment input \(dx_i\), a secondary increment input \(dy_i\), and emits an incremental output \(\Delta z_i\) as shown in Fig. 3.13a. Each integrator consists of two registers, a \(Y\) register and an \(R\) register [10], [11].

The contents of the \(Y\) register is given by:

\[
y_i = \int_{0}^{x_i} dY = y_{i-1} + dy_i
\]

(3.17)

The contents of the \(R\) register is:

\[
R_i = R_{i-1} + y_i \cdot dx_i + \Delta z_i
\]

(3.18)

where the output \(\Delta z_i\) is

\[
\Delta z_i = y_i \cdot dx_i
\]

(3.19)

Equations (3.17), (3.18) and (3.19) define the algorithm of operation of the digital integrator. A symbolic representation of the digital integrator is shown in Fig. 3.13b.

Two special connections are used in the digital differential analyser: the summing integrator (or digital servo) and the multiplying integrator (or scale integrator).

3.5.2 The summing integrator (Digital Servo)

An adder, or summing integrator, is required to sum either the primary or the secondary inputs to an integrator whenever the number of such inputs exceeds the capacity of the input registers or counters.
Figure 3.13. The Digital Integrator.

(a)

Figure 3.14. Special Connection for the Digital Integrator.

(a)
Essentially, an adder is a servo which employs its own output as a secondary input [10]. An adder contains a decision package whose \( \Delta z \) output:

\[
\Delta z_i = \text{Sgn}(y_1) \ dt
\]  
(3.20)

(where \( dt \) is the clock pulse) is connected in the feedback arrangement shown in Fig. 3.14a. An input to the adder causes an output of the same sign that is subtracted on the next cycle from the number held in the \( Y \) register.

The dependent input of the servo will be:

\[
dy_i = \Delta y_1 + \Delta y_2 = \Delta z_{i-1} \quad \ldots.
\]  
(3.21)

and the contents of the \( y \)-register, which are denoted as from now by the symbol \( P_i \), instead of \( y_i \) (equation (3.17)), will be:

\[
P_i = P_{i-1} + (\Delta y_1 + \Delta y_2 - \Delta z_{i-1}) \quad \ldots.
\]  
(3.22)

If the clock pulse \( dt \) occurs at a much greater rate than \( \Delta y_1 \) and \( \Delta y_2 \) pulses, the feedback signal \( \Delta z \) causes \( P_i \) to stay close to zero. The output will then be

\[
\Delta z_i = \Delta y_1 + \Delta y_2 \quad \ldots.
\]  
(3.23)

i.e. represents the sum of the two input sequences. If \( \Delta y_1 \) is the delta sequence \( \{x_n\} \) and \( \Delta y_2 \) is the delta sequence \( \{y_n\} \), then according to [10], the output of the servo \( \Delta z_i \) will be the sequence \( \{S_n\} \) which is the sum of \( \{x_n\} \) and \( \{y_n\} \) where:

\[
S_n = \text{Sgn}(P_n) \cdot dt
\]  
(3.24)

and

\[
P_n = P_{n-1} + x_n + y_n - S_{n-1} \quad \ldots.
\]  
(3.25)

\( n = \ldots, -1, 0, +1, \ldots \)

The error introduced will be \( P_n \).
3.5.3: **The Scale Integrator: (Multiplication by a Constant)**

There are several schemes for the multiplication of a variable by a constant in a digital differential analyser. These schemes are discussed in [10]. In the most straightforward of these, the constant \( c \) is placed in an integrator's \( y \)-register and the variable \( dx \) to be multiplied with this constant provides the primary input (Fig. 3.14b). The output signal \( \Delta z_1 \) is:

\[
\Delta z_1 = C \cdot dx_1
\]  

(3.26)

The contents of the \( R \)-register \( R_i \) is:

\[
R_i = R_{i-1} + C \cdot dx_1 - \Delta z_1
\]  

(3.27)

The constant \( C \) can be any fraction \(|C| < 1\). In the case of using the integrator for changing the scale of the variable \( dx_1 \) (since the digital differential analyser is a fixed-point machine) the constant

\[
C = 2^{-(\alpha - \beta)}
\]  

(3.28)

where \( \alpha \) is the new scale and \( \beta \) is the old scale of \( dx_1 \) [10].

6. **OPERATIONS**

3.6.1 **Addition of Two Delta Modulated Signals**

The sequence \( \{S_n\} \) as defined in (3.24) is used in all the existing digital servos to represent the sum of the analog signals \( X(t) \) and \( Y(t) \) with an error \( \psi(t) \) which depends on the value of \( P_n \). This error can be kept close to zero by making the rate of the clock pulse \( dt \), much more than the rate of \( X_n \) and \( Y_n \) pulses. Even then, the error can take any value. To keep the error in calculating \( \{S_n\} \) within a reasonable defined value (one quantization step for example), expression (3.25) should be modified to contain a sum of difference. In this case, it is possible to apply the "Telescoping Sums" rules. This will keep the error within the required limits using any rate for the clock pulses.
Instead of connecting \(- S_{n-1}\) to the dependent input of the digital servo, we connect signal \(C_n\) defined as:

\[
C_{n-1} = -P_{n-2}
\]

where \(P_{n-2}\) is the contents of the \(Y\)-register at the end of the \((n-2)\)th cycle. Equation (3.25) now takes the form:

\[
P_n = 2^{v-1} \left[ P_{n-1} + Y_n + X_n - P_{n-2} \right]
\]

which contains the telescopic sum, where \(P_{n-1}\) is the contents of the \(Y\)-register at the end of the \((n-1)\)th cycle and:

\[
P_{n-1}, \quad P_{n-2} \in \{ +1, -1 \}
\]

and the sum \(S_n\) will be:

\[
S_n = P_n
\]

If condition (3.31) is satisfied, the terms of \(S_n\) and also that of \(C_n\) take the values \(+1\) or \(-1\), as it is easily deduced from equation (3.30). This in turn realizes equation (3.31). From Equations (3.30) - (3.32) above, Table 3.1 has been constructed which gives the values of \(S_n\) and \(C_n\) as a function of the values \(X_n\), \(Y_n\), and \(C_{n-1}\).

<table>
<thead>
<tr>
<th>(X_n)</th>
<th>(Y_n)</th>
<th>(C_{n-1})</th>
<th>(S_n = P_n)</th>
<th>(C_n)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(-1)</td>
<td>(-1)</td>
<td>1</td>
<td>1</td>
<td>(-1)</td>
</tr>
<tr>
<td>1</td>
<td>(-1)</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>(-1)</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>(-1)</td>
</tr>
<tr>
<td>(-1)</td>
<td>(-1)</td>
<td>1</td>
<td>(-1)</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>(-1)</td>
<td>1</td>
<td>(-1)</td>
<td>(-1)</td>
</tr>
<tr>
<td>(-1)</td>
<td>1</td>
<td>(-1)</td>
<td>(-1)</td>
<td>(-1)</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>(-1)</td>
<td>(-1)</td>
<td>1</td>
</tr>
</tbody>
</table>
**THEOREM:**

The sequence \( \{S_n\} \) as defined in (3.30) and (3.32) represents the delta sequence which is half the sum of the analog signals \( X(t) \) and \( Y(t) \) with an error \( \Phi(t) \), introduced by \( \{S_n\} \), given by the relation:

\[
\Phi(t) = 2^{-1} \delta(C_n - \gamma)
\]

(3.33)

where \( nT \leq t < (n+1)T \)

\( \delta \) is the quantization step and the value of \( \gamma \) will be either \(+1\) or \(-1\).

**PROOF:**

Since the terms of \( \{S_n\} \) take only the values \(+1\) or \(-1\), its integrated signal \( \tilde{S}(t) \) is given by

\[
\tilde{S}(t) = \sum_{k=1}^{n} \delta \sum_{k=1}^{n} \tilde{S}_k
\]

(3.34)

\( nT \leq t < (n+1)T \)

From equation (3.30), the telescopic sum

\[
\sum_{k=m}^{n} \left( C_{k-1} - C_k \right) = C_{m-1} - C_n
\]

(3.35)

\( C_{m-1} \) takes the value \(+1\) or \(-1\) for any value of \( m \) and consequently, for \( m \to -\infty \), (3.35) can be written as

\[
\sum_{m=-\infty}^{n} \left( C_{k-1} - C_k \right) = \gamma - C_n
\]

(3.36)

where \( \gamma \) is a constant equal to \(+1\) or \(-1\).

From (3.30), (3.34) and (3.36) we can derive:

\[
\delta \sum_{k=1}^{n} S_k = 2^{-1} \left[ X(nT) + y(nT) + 2^{-1} \delta(C_n - \gamma) \right]
\]

OR

\[
\tilde{S}(t) = 2^{-1} \left[ X(t) + y(t) + \Phi(t) \right]
\]

(3.37)

\( nT \leq t < (n+1)T \)
where
\[ \phi(t) = 2^{-1} \delta(C_n - \gamma) \]  \hspace{1cm} (3.38)

The only values which \( \phi(t) \) can take are 0 or \(-\delta\) for \( \gamma = 1 \), and 0 or \(+\delta\) for \( \gamma = -1 \) where upon \(|\phi(t)| \leq \delta\). The error can be considered sufficiently small and consequently (3.37) proves the theorem.

We note here that the \( 2^{-1} \) appears in the expressions (3.30) and (3.32), this is as a result of taking the scale of \( s(t) \) to be one half that of \( X(t) \) and \( y(t) \). This is true, since the maximum value \( s(t) \) can reach is the sum of the maximum values of \( X(t) \) and \( y(t) \).

3.6.2: Addition of Several One-bit Sequences.

Consider the analog signals \( X_1(t), X_2(t), \ldots, X_p(t) \) with \( 2^r < p \leq 2^{r+1} \) where \( r = 1, 2, \ldots \). From the one-bit sequence of these signals, one can find the one-bit sequence of the signal
\[ 2^{-(r+1)} [X_1(t) + X_2(t) + \ldots + X_p(t)] \]  as follows:

The one-bit sequence of the signals are separated in pairs. If \( P \) is odd number, one pair will be formed by the use of the idling sequence \( \{I_n\} \) where \( I_{n} = -I_{n-1} \).

We get the sum of each pair using equation (3.30). The same procedure is repeated at a second level for the resulting sequences and so on for \( r + 1 \) levels. As the factor \( 2^{-1} \) is introduced in each level for the corresponding analog signals, the sequence resulting at \( r \) level corresponds to the sum of \( P \) analog signals multiplied by the factor \( 2^{-(r+1)} \). The maximum absolute value of the error due to the successive operations is equal to \( (r+1)\delta \).

3.6.3: Multiplication of DM Signal with a Const.

It is possible to find the delta sequence \( \{M_n\} \) of the product \( a \cdot X(t) \) where \( a \) is a constant with \(|a| < 1\), by using the digital integrator. The constant \( a \) is placed in the \( \gamma \)-register and the delta sequence
\( \{ \chi \}_n \) of the variable \( \chi(t) \) provides the primary input of the integrator, and there is no secondary input. The integrator is now a scale integrator, that have been discussed in 5.3. The algorithm of operation of the integrator is:

\[
R_n = R_{n-1} + a \chi_n - \Delta z_n \quad \ldots \ldots \quad (3.39)
\]

where

\[
M_n = \Delta z_n \quad \ldots \ldots \quad (3.40)
\]

**Theorem:**

The sequence \( \{ M_n \} \) as defined in (3.40), represents the delta sequence of the product \( a \chi(t) \) with an error \( \Phi(t) \) given by the relation:

\[
\Phi(t) \leq \delta \quad \ldots \ldots \quad (3.41)
\]

**Proof:**

Since the digital integrator is a fixed point machine, then we must take into consideration the scales of the various variables, Let:

- \( \gamma \) = the scale factor of \( \chi(t) \)
- \( \varepsilon \) = the scale factor of the product signal \( m(t) \)
- \( \alpha \) = the scale factor of the contents of the \( Y \)-register (in our case).

The relation between the scale factors and the quantization step is:

\[
\delta x = 2^\gamma \quad \text{quantization step of } \chi(t)
\]

\[
\delta y = 2^\alpha \quad \text{quantization step of } y(t)
\]

\[
\delta m = 2^\varepsilon \quad \text{quantization step of } m(t)
\]

The relation between the scale factors is:

\[
\varepsilon = \alpha + \gamma \quad \ldots \ldots \quad (3.42)
\]

Since \( |a| < 1 \), then it is possible to take \( \alpha = 0 \) i.e. the machine value equals the actual value.
From (3.42)

\[ \varepsilon = \gamma \quad \text{and} \quad \delta x = \delta m = \delta \]  

\[ \delta \Sigma k = a, \delta \Sigma k \frac{x}{k} - \delta \Sigma k \frac{R_k - R_{k-1}}{k} \]  

(3.43)

From (3.39) and (3.40) we get

\[ n \delta \Sigma k \frac{M_k}{k} = a, \delta \Sigma k \frac{k}{x} - \delta \Sigma k \frac{R_k - R_{k-1}}{k} \]  

(3.44)

From (3.44) we get

\[ m(t) = a, x(t) + \phi(t) \]  

\[ nT \leq t \leq (n+1)T \]  

where

\[ \phi(t) = \delta \Sigma k \frac{R_{k-1} - R_k}{k} \]  

(3.45)

\[ = \delta \left( R_{-\infty} - R_n \right) \]  

(3.46)

From the theory of the operation of the digital integrator [10], it is possible to keep the round-off error \( R_n \) (n = \(-\infty, 1, 0, +1, \ldots\)) within the range

\[ -0.5 \leq R_n \leq 0.5 \]  

(3.47)

This means that

\[ \phi(t) < \delta \]  

which proofs the theorem.

3.7.1: Digital Filter Realization Using the Proposed Algorithm

The approach proposed in this chapter to realize the non-recursive digital filter equation: \( y_n = \sum_{i=1}^{N} a_i x_{n-i} \) is illustrated in fig. 3.15.

Each term in the equation needs a scale integrator to realize the multiplication of \( a_k \) by the delta sequence \( x_{n-k} \). The outputs of the scale
Figure 3.15. Non-recursive Digital Filter.
integrators are divided into groups of two outputs each. Each group needs a summing integrator to add the two output signals. If $N$ is odd, one of the group contains the "idling" sequence $\{I_n\}$ which is defined as

\[ I_n = \ldots, I_{-1}, I_0, I_1, \ldots \]

with $I_n = -I_{n-1}; \ n = \ldots, -1, 0, +1, \ldots \ldots$.

where $I_i$ takes the values $+1$ or $-1$. This sequence is defined as the delta sequence of the constant function $i(t) = 0$. The outputs of the adders, again are divided into pairs which in turn are added by another system of adders, and so on.

The sequence $X_{n-k}$ ($k = 0, 2, \ldots, N-1$) is stored in a shift register of length $N$. This connection is shown in Fig 3.15a. An alternative connection is given in Fig. 15b. In this configuration, and as a result of the fact that the adder gives half the sum of the inputs, the relation between the coefficients $a_1$ and $a_1'$ is:

\[
\begin{align*}
a_2 &= a_2', 2^{-1} & a_1 &= a_1', 2^{-2} \\
a_0 &= a_0', 2^{-3} & a_3 &= a_3', 2^{-1} \\
a_4 &= a_4', 2^{-2} & a_5 &= a_5', 2^{-3} \\
a_6 &= a_6', 2^{-3} & \end{align*}
\]

where $a_1'$ is the PCM coefficient.

3.7.2: Recursive Digital Filter Realization

The realization of a recursive digital filter using delta modulated sequences is also possible by connecting delta multipliers, adders and a shift register. There is a problem however, with the coefficients as the adders supply output delta sequences which represent half of the analog to their signals of the corresponding input delta sequences. This problem can be solved by multiplying the coefficients by $2^N$, where $N$ is the order of the
filter. As a result, the absolute value of the PCM coefficients must not exceed the value of $2^{-N}$.

The realization of the first and second-order recursive DF using the proposed techniques are shown in Figure 3.16.

3.8. **Conclusion and Results:**

The connection shown in Fig. 15a was used to design a low-pass non-recursive digital filter. The filter is of first order and it has seven coefficients with values

$$a_k = 2^{-k} \quad (k = 0, 1, 2 \ldots \ldots \ldots 6)$$

The network was simulated and the output response of the filter is shown in Fig. 3.17.

We have proposed a new realization of nonrecursive as well as recursive digital filters using delta modulation. The main advantage of the scheme is the simplicity of the hardware, as a result of the elimination of multiplications in the processor, and of a simpler analog to digital conversion method. The proposed technique also results in a modular approach to the hardware implementation of the digital filters. The delta signals are used directly in processing. The resultant error is significantly small for practical use.
Figure 3.16. FIRST AND SECOND ORDER RECURSIVE DF
Figure 3.17. OUTPUT RESPONSE OF LOW-PASS DF.
REFERENCES:


4.1. Introduction.

In the previous chapters, the use of one-bit word for realizing Digital Filters was discussed in detail. As mentioned before, we employed one bit-word in processing in order to get rid of multiplication. The multiplication time is the deciding factor of the speed of the Digital Filters. To reduce the execution time of one output sample, it is necessary either to reduce the multiplication time or to reduce the number of multiplication processes required. The use of ROM as look-up table [26,27] and the use of the incremental changes (the one-bit is the limiting case) of the signal $X_{n-1}$ [28] in processing instead of the full sample value are some forms of reducing the multiplication time. The use of distributed control [1] to get the intermediate entities $I_{in} = A_i X_{n-i}$ and also the use of the incremental changes in processing result in an effectively multiplication free Digital Filter. The use of the incremental changes in processing is discussed in details in the previous two chapters.

The technique proposed in this chapter belongs to the last strategy i.e., to design an effectively multiplication free Digital Filter in spite of the use of full sample values of $X_{n-1}$ in processing. This will be achieved by using distributed arithmetic. A new algorithm is considered in this chapter which will convert the difference equation describing the filter to the sum of the weighted number $C_j 2^j$. This sum can be realized directly using carry save arrays.

The calculation of one output sample using the proposed technique will be achieved in three steps. The first step needs, as will be shown, counters only, while the remaining two steps need the calculation of the sum of the weighted number $C_j 2^j$. This means that the realization of the filter needs only counters with the carry-save arrays.
In section 2 the proposed algorithm is given. The hardware realization of the algorithm is discussed in Section 3. A modified design to allow the use of both positive and negative numbers is discussed in Section 4. The realization of \( n^{\text{th}} \) order non-recursive digital filter, the computation time in terms of gate delays and the required hardware is given in Section 5.

4.2. The proposed algorithm

The non-recursive Digital Filter calculates the \( n^{\text{th}} \) output \( Y_n \) from \( M \) previous inputs \( X_{n-i} \) according to the equation

\[
Y_n = \sum_{i=0}^{M-1} A_i \cdot X_{n-i}
\]

where \( \{X\} \) is the input sequence
\( \{Y\} \) is the output sequence
and \( \{A\} \) filter coefficients.

Let us assume (temporarily) that the data and coefficients satisfy

\[
A_i, X_{n-i} \geq 0
\]

This constrain is to simplify the representation of the algorithm and will be removed later.

In representing \( A_i, X_{n-i} \) in the machine, we assign \( L \) bits to each coefficient and \( N \) bits to each input data word i.e.

\[
A_i = \sum_{k=1}^{L \delta_i} 2^k
\]

\[
X_{n-i} = \sum_{j=1}^{N} X_{n-i,j} \cdot 2^j
\]

Substituting (3) and (4) into (1) and re-arranging the order of the summation, we get
\[ Y_n = \sum_{i=0}^{M-1} \sum_{k=1}^{L} \sum_{j=1}^{N} a_{i,k} \cdot 2^k \cdot X_{n-i,j} \cdot 2^j \]

\[ = \sum_{j=1}^{N} \left( \sum_{k=1}^{L} \left( \sum_{i=0}^{M-1} a_{i,k} \cdot X_{n-i,j} \right) \cdot 2^k \right) \cdot 2^j \]

\[ = \sum_{j=1}^{N} C_j \cdot 2^j \]

Equation (5)

where \( C_j = \sum_{k=1}^{L} S_{k,j} \cdot 2^k \)

Equation (6)

and \( S_{k,j} = \sum_{i=0}^{M-1} a_{i,k} \cdot X_{n-i,j} \)

Equation (7)

\[ M = 0, 1, 2, 3, \ldots, M-1 \]

\[ J = 1, 2, 3, \ldots, N \]

\[ K = 1, 2, 3, \ldots, L \]

\[ a_{i,k}, X_{n-i,j} = 0, 1 \]

Equations (5), (6) and (7) embody the basic strategy of the proposed machine.

To get \( Y_n \), there are three stages: the first is to get the \( N \cdot L \) numbers \( S_{k,j} \) from the input data and the filter coefficients; the second is to use \( S_{k,j} \) to get the \( N \) numbers \( C_j (j = 1, 2, 3, \ldots, N) \); the last stage is to get \( Y_n \) from the \( C_j \) numbers.

These three steps are shown symbolically in Figure 1.

4.3 Hardware Realization of the Proposed Algorithm.

To calculate the output \( y \), two sets of numbers \((S_{k,j}, C_j)\) are to be calculated. The numbers \( S_{k,j} \) are defined by (7). Each element of the sum is a product of two 1-bit entities. Such a product is either 0 or 1 and, practically speaking, no multiplication is involved in its evaluation. A dual input AND gate is all that is needed. Thus, if we design a system in which \( M \) dual input AND gates are fed by the pairs of bits specified in (7) a count of the number of TRUE gates will equal \( S_{k,j} \). Serial implementation
of the above algorithm will produce a machine that is slow and hence not useful in many applications. The two practicable methods of implementing the algorithm are the quasi-parallel and parallel methods.

In the fully-parallel method, the matrix of the bit product (i.e. \( a_{ik} \cdot x_{n-i,j} \) on Figure 1) is generated in a single step using \( M \) AND-Gate matrices each of dimension \( N \times L \). These matrices are reduced to an equivalent \( N \times L \) matrix with a network of counters. The elements of the new matrix are the numbers, \( S_{kj} (k = 1,2,3, \ldots, L \) and \( j = 1,2,3, \ldots, N) \). The \( N \times L \) matrix is summed with a number of single bit full adders arranged in form of carry-save arrays in two separate steps. The first is to get \( N, C_j \) numbers and the second step is to get \( Y_n \).

Although the fully-parallel method is very fast, it does require a considerable hardware that precludes its use for many applications. The quasi-parallel method is somewhat slower than the fully parallel method. In this method, the \( M \) matrices of the bit products are not generated in a single step but \( M \) steps, with one \( N \times L \) array of logic AND gates. All other steps are the same as that of the fully-parallel method. The number of counters needed in both cases is \( N \times L \).

In the following we consider in details each step separately.
Figure 1: Symbolic Representation of the Proposed Algorithm.
4.3.1. Calculation of the $S_{kj}$ Numbers:

The realization of (7) is carried out by using counters that count the number of TRUE Gates of the bit product matrices (i.e., $a_{ik}, x_{n-i,j}$ on Fig. 1.). Serial as well as parallel counters can be used to realize $S_{kj}$. In the following the possibility of using some of these counters are considered.

4.3.1.1. The Use of Asynchronous and Synchronous Counters.

It is possible to implement $S_{kj}$ by using binary up-counter circuit shown in Fig. 2. This form of counter has the disadvantage of introducing delays between successive outputs owing to the signal rippling through the stages. The count sequence is an increasing binary count for each TRUE input signal. This is a very slow method of implementing $S_{kj}$.

One method of overcoming the problem of speed is to use a synchronous binary up-counter circuit shown in Fig. 3. A synchronous counter is one in which all stages are triggered simultaneously. The resulting action of each stage depends on the gating inputs of each respective stage. It is obviously faster than a ripple counter, since higher order stages don't have to wait for lower order changes to occur first, as in a ripple counter. A synchronous binary counter is more complicated than a ripple counter and contains more logic elements, it is also more flexible.

4.3.1.2: The Use of Parallel counters:

Another method of increasing the speed of realizing $S_{kj}$ is to implement it with a parallel counter. A parallel counter [10], [11] is a device that produces at the output, in binary coded form, the number of active (in logic ONE State) inputs. A parallel counter with $u$ inputs and $v$ outputs is called a $(u,v)$ counter [11],

$$v = 1 + \left\lfloor \log_2(u) \right\rfloor \quad ............. \quad (8)$$
Figure 2. Three-Stage binary ripple Counter.

Figure 3. Three-Stage binary Synchronous Counter.

Figure 4. A (U,V) - Counter Schema.
where \( [x] \) denotes the largest integer \( i \) such that \( i \leq x \) (it is equivalent to the functions \( \text{ENTIER}(x) \), \( \text{FLOOR}(x) \), and \( [x] \) used by other authors).

A \((u,v)\) counter schema is shown in Figure 4.

Many methods may be used to implement parallel counters. Some of the more obvious techniques are: two-level gate network [11]; READ-ONLY Memory [6]; full-adder network [10], [19]; full-adder/fast-adder array [10]; and quasi-digital processor [2], [10], [20].

4.3.1.2.1 The Use of Two Level Gate Network [11]

To implement an \( M \)-input counter by using a two-level gate network requires \( 2^M - 1 \) logic AND gates (each with \( M \) inputs) followed by a reduction network that requires \( P \) logic OR gates (each of the order of \( 2^M - 1 \) inputs), where \( P \) is the number of outputs from the counter. This means that the fan-in of the gates required to add \( M \) bits using this technique is:

\[
P \cdot (2^M - 1) + M \cdot (2^M - 1)
\]

where \( P = \lfloor \log_2 M \rfloor + 1 \)

Clearly then the two-level gate network is impractical for most values of \( M \).

4.3.1.2.2. The Use of ROM

The READ-ONLY Memory approach requires a memory with \( 2^M \) words of \( P \)-bits each to implement an \( M \)-input counter. In this case, the \( M \)-bits of the sum describing \( S_{kj} \) are used as an address for the ROM as shown in Fig.5. If the word length of \( S_{kj} \) is \( P \), then the total storage capacity of the ROM will be \( P \cdot 2^M \) bits which is impractical for most values of \( M \).

4.3.1.2.3 The Use of Full-Adder Network

The full-adder network is one of the practical methods of implementing parallel counters. Basically their design procedure involves grouping the counter inputs into sets of three lines each. Each set of three lines is then reduced by a full-adder into a sum output line with weight 2. This reduction results in approximately \( M/3 \) lines of weight 1 and \( M/3 \) lines of weight 2. These lines are then separated into sets of three with equal
Figure 5. The use of ROM to implement $S_{kj}$.

Figure 6a. 7-input binary full adder network parallel counter.

(Where $x_i = a_{i,k} \cdot x_{n-i,j}$)
weight and the reduction continues until only one line of each weight
remains. At most \( M \) full-adders are required to implement an \( M \)-input
counter. If we select the number of inputs to the parallel counter to be
\[
M = 2^k - 1
\]
\[\text{(9)}\]
where \( k \) is an integer, then each full adder will be completely utilized.
For values of \( M \) corresponding to non integer values of \( k \) in (9), some
full-adders will have fewer than three inputs, and are therefore, ineffi-
ciently used. In the case where \( k \) is an integer, the number of
completely utilized full-adders required to implement \( M \)-input parallel
counters is \( 2^k - 1 - k \). Shown in Fig. 6a, is a 7 input parallel counter
implemented with binary full adders (each rectangle represents a full adder).
In [12], it was shown that with the use of four-valued logic "full-adders"
significant reductions are possible in the number of devices required to
realize parallel counters.

4.3.1.2.4 The Use of Full-adder/Fast Adder Array

Another practical method of implementing parallel counters is to
employ full-adder/fast adder array. This may be accomplished by using
fast-adders that are realized with READ-ONLY Memories in place of the ripple-
carry adders of Fig.6a. The design procedure is similar to that which was
used for the full-adder network counter with the exception that fast adders
are used. To show how this is done, consider Fig.6b: \( FA_5 \) and \( FA_6 \) are
replaced by a 2-bit fast adder, \( FA_7 \) and \( FA_8 \) are replaced by a 2-bit fast
adder, and \( FA_9 \), \( FA_{10} \) and \( FA_{11} \) are replaced by a 3-bit fast adder. In this
case the corresponding parallel counter will be as that given in Fig.6b
If the access delay of the ROM "adder" is comparable to the delay of a
full-adder, the counter will be nearly twice as fast. For counters with
under 32 inputs it is a practical method to implement a high-speed counter.
Figure 6b. Synthesis of a 15-input Counter with a full-adder network.
(where $x_i = a_{i,k}, x_{n-i,j}$)
4.3.1.2.5: The use of small Parallel Counters.

The two techniques described in 3.1.2.3 and 3.1.2.4 can be used for small values of M. Otherwise the number of full adders required will be very large. In such cases, the method presented in [11] which enables small parallel counters to generate large ones can be used to implement the parallel counter. The design procedure is based on the use of "(u,v) - counter pyramids" that are reduction networks with only (u,v) - counters as elementary building blocks. Fig. 7 shows the circuit of a (u,v) - counter pyramid. An extremely large number M of inputs can be handled with this method.

4.3.1.2.6 The Use of Quasi-digital Techniques

An alternative practical implementation of parallel counters is the use of quasi-digital techniques. This method uses analog current summing to generate a voltage proportional to the count. This method is described in [10], and it is reported to be potentially much faster than either of the strictly digital approaches, although it is said to be somewhat more complex.

4.3.1.2.7 The Use of the Reordering Technique

Yet, another practical approach to the implementation of parallel counters is the counting by reordering technique presented in [1]. The counting of bits is based on their reordering implemented by a sequential logical circuit.

Let us suppose that we start with a given sequence of bits

\[ x_{ji}, j = 1, 2, \ldots, m \]

for a fixed binary order i, integer

j is the ordering parameter of the sequence. Then if

\[ x_{ji} \geq x_{j+1,i}, \text{ for every } j = 1, 2, \ldots, m-1 \]  \hspace{1cm} (11)

the sequence is ordered so that \( x_{ji} \) never increases with j. A given sequence does not have this property in general, but it is quite easy to reorder without changing the sum.
Figure 7. (U,V)-Counter Pyramid.
\[ S_i = \sum_j x_{ji} \quad \ldots \quad (12) \]

To do this, it is necessary to interchange any two bits which follow each other in the wrong order:

\[ (x_{ji} < x_{j+1,i}) \Rightarrow (x_{ji} = 0, x_{j+1,i} = 1) \]

\[ \Rightarrow (x'_{ji} = x_{j+1,i} = 1, x'_{j+1,i} = x_{ji} = 0) \quad \ldots \quad (13) \]

so that \( x'_{ji} > x'_{j+1,i} \) after the interchange which is repeated until the final sequence is monotonic and obeys (11). An example of the fundamental version of the sequential circuit for reordering is in Figure 8. In this example \( m = 7 \). After the reordering, the last non zero value along the sequence \( x_{ji}, j = 1, 2, \ldots, m \) gives the count \( S_i \) of non zeros in the order \( i \). Logical circuit producing this information is based on the fact that after the reordering there is never more than just one value of \( j \) for which \( x_{ji} > x_{j+1,i} \). Then it is possible to find it by the Boolean condition \( x_{ji} x_{j+1,i} = 1 \). The corresponding logical circuit is located on the right-hand side of Figure 8.

The end of reordering Strobe Signal is easy to derive because the end means that the condition (13) is invalid for every \( j = 1, 2, \ldots, 6 \). Then the NOR gate on the left-hand side of Figure 8 has no input signal and generates the end Strobe. This method has the following properties:

1. It can handle an extremely large number \( m \) of numbers to add.
2. The hardware cost is smaller than with other concepts.
3. The structure is well suited for circuit integration since the control is distributed over the chip.
4. Very small execution times are expected [1].
Figure 8. Sequential Circuit for Reordering of $x_{ji}$.
Any of the above mentioned counters can be used to realize $s_{kj}$.

In Fig. 8-a, the circuit required to calculate $s_{kj}$ using serial counter is given. In Fig. 8-b the corresponding network in the case of using parallel counter is given.

The following methods could be used to realize $s_{kj}$ numbers:

(a) **Serial/serial**: In this case, only one serial counter is required to calculate all the $s_{kj}$ numbers one by one.

(b) **Serial/Parallel**: In this case, only one parallel counter is required to calculate all the $s_{kj}$ numbers one by one.

(c) **Parallel/Serial**: In this case, N.L Serial Counters are required to calculate all the $s_{kj}$ numbers simultaneously.

(d) **Parallel/Parallel**: In this case, N.L parallel counters are required to calculate all the $s_{kj}$ numbers simultaneously.

4.3.2. **Calculation of the $c_j$ Numbers**

The numbers $c_j$ ($j = 1, 2, 3 \ldots, N$) are defined by equation (6).

To explain the technique used to realize it, let us consider at first the problem of getting the product of two numbers $A$ and $B$ which may be given by

$$A \cdot B = A \cdot \sum_{i=1}^{k_k} b_i \cdot 2^i$$

$$= \sum_{i=1}^{k_k} (A \cdot b_i) \cdot 2^i$$

where $k_k$ is the number of bits in $B$.

Define

$$A \cdot b_i = Q_i$$

where $Q_i$ is the $i$th bit of $B$ multiplied by $A$.

Thus

$$A \cdot B = \sum_{i=1}^{k_k} Q_i \cdot 2^i$$

Where $Q_i = [A \text{ for } b_i = 1$

$$0 \text{ for } b_i = 0$$
Figure 8a: Calculation of $S_{kj}$ using Serial Counter.

Figure 8b: Calculation of $S_{kj}$ using Parallel Counter.
Equation (6) has the same structure as (17), but $S_{kj}$ in (6) can assume any value whereas $Q_i$ in (17) can assume only two values (A or 0) as shown above. Hence (6) could be realized by using a modified multiplier circuit.

From a logic design viewpoint, we can divide multipliers into two major categories: clocked multipliers and array multipliers. In both types the product is effectively obtained through successive additions, the difference being that sufficient parallelism exists in the array multiplier so that the final answer is obtained without registering intermediate results. The sequential method used in clocked multipliers is adequate for low-speed multiplication. For high-speed multiplication a combinatorial approach, used in array multipliers is needed.

The array multiplier consists of a two-dimensional array of one-bit full adders. It is a complete memoryless logic net that only requires a prescribed settling time after the application of the input signals before the product can be used. There are many array configurations but they can fairly well categorized according to the method of connecting the adders and according to the way negative numbers are handled. Figure 9a illustrate the interconnection of adders to sum partial products. This is not a very efficient array for multiplying positive numbers. Each little circle is a one bit adder. The speed of multiplication is governed by the carry-propagation delay. If n-bit word is to be multiplied by m-bit word, then the total settling time $τ_t$ is given by [21]

$$τ_t = 2nτ_c + mτ_s$$

(18)

where $τ_c$ is the carry propagation time

$τ_s$ is the sum propagation time. There are several ways of speeding up multiplication.
Figure 9a. Array multiplication of \((x_5x_4x_3x_2x_1x_0)\) by \((y_5y_4y_3y_2y_1y_0)\), both positive numbers.

Figure 9b. Faster array structure for multiplication of Fig. 9a.
The first speeds the addition of partial products by using carry-save adders. This is illustrated in Figure 9.b. In this case, the total settling time is given by [21]

\[
\tau_t = [(n-1) + (n-1)] \tau_c \text{ if } m \tau_c > (n-1) \tau_s \\
= (n-1) (\tau_c + \tau_s) \text{ if } m \tau_c < (n-1) \tau_s 
\] (19)

when \( \tau_s \) is appreciably greater than \( \tau_c \), it becomes desirable to find a structure which reduces the number of \( \tau_s \) times needed. One such structure is shown in Figure 10 which is an \( n \times 16 \) bit multiplier. In this arrangement, only four sum times are necessary. The same number of adders are required as before but more rows of adders are operating in parallel. The total add time for each adder row is really the carry-ripple time or \( (n-1) \tau_c \). Since there are four levels, each of time \( (n-1) \tau_c \), we obtain \( 4(n-1) \tau_c \) as the total time. Thus, this scheme is not so efficient as that of Figure 9.b unless \( \tau_s \) is a very large number. The idea of the tree and the diagonal propagation of the carries can be combined to yield perhaps the fastest configuration. An example is shown in Figure 11.

Additional speed can be gained by the use of a Wallace tree [23]. The Wallace tree is an interconnection of carry-save adders that reduces \( n \) partial products to two operands.

The principle of the Wallace tree is to use a single carry-save adder to reduce three bits of equal weight to two bits, one a sum and the other a carry. An example is shown in Figure 12. In this case, the number of carry-save adder delays necessary to reduce \( n \) partial products to two operands is

\[
\text{CSA delays} = \lceil \log_{3/2} n \rceil - 1 
\] (20)

Thus, in 32 x 32 bit multiplication, 8 CSA delays are needed to reduce the 32 partial products to two operands [18].
Figure 10. Tree idea for array multiplier.

Figure 11. Array multiplier with diagonal carry and tree arrangement for sums.
Figure 12. Example of 3-to-2 tree for 7 input operands.
Another method which could be used to speed-up multiplication time reduces the number of partial products by employing a modified booth Algorithm [18]. This method is not applicable to our case. In general, the best method that can be used out of the above mentioned arrays depends on the I.C. hardware being used and the speed-cost tradeoffs.

In the following, the way of using some of the above mentioned arrays to realize $C_j$ from $S_{kj}$ numbers according to equation (6) will be discussed. As an example, we shall consider here the use of the arrays given in Figures 9a, 9b and 12 to calculate $C_j$ number. To simplify the analysis of the problem, assume that $L = 6$; that is 6-$S_{kj}$ numbers are added to give a $C_j$ number. Assume also that the word length of each $S_{kj}$ number is 4 bits. The bit position of each $S_{kj}$ number is shown in Figure 13.

- Figures 14, 15 and 16 illustrate the interconnection of adders to sum the $S_{kj}$ numbers to produce the $C_j$ number according to figures 9a, 9b and 12 respectively.
Figure 14. Realization of $C_j$ number using full adder arrays.
It is possible to realize all the $N - C_j$ numbers in parallel by using $N$ different arrays. It is also possible to use only one array to realize all the $N - C_j$ numbers sequentially. Parallel implementation is of course faster than the serial method but it requires a lot of hardware and hence costlier.

4.3.3 Calculation of the $Y_n$ Number

The number $Y_n$ is defined by equation (5). Since $Y_n$ is a summation of $N$, $C_j$ numbers, equation (5) has the same structure as equation (6), hence the same array used for the realization of $C_j$ numbers could be used to realize the $Y_n$ number. The array in this case consists of $N$ rows each has a number of adders equals to the word length of $C_j$. The array is shown in Figure 17, for $N = 6$ and with the corresponding word length of $C_j$ which is 4 bits.

4.4. Modified Design to Allow the use of Negative Numbers

Figures 14, 15 and 16 are full-adder array schemes and are applicable and efficient for positive numbers only. Let us consider the general case of dealing with both positive and negative numbers. That is, the constrain for Data and Coefficients' representation given by equation (2) is removed. General purpose computers usually handle negative numbers either by the sign magnitude representation or via the use of the 2's complement. Both methods are quite inappropriate in the present application as they would drastically increase the complexity of the machine.

A possible solution [16] is to apply bias to both data and coefficients so that the numbers presented to the machine are always positive. This method requires corrective measures to compensate for bias.

Another solution is based on a negative radix number representation e.g. base (-2) [17]. This method requires converters to convert numbers from base 2 to base (-2) and vice versa. The two techniques are
Figure 17. Realization of $Y_n = \sum_{j=1}^{N} C_j \cdot 2^j$ using Carry Save arrays.

Figure 18. Three-bit up-down Synchronous Counter.
complicated since the first requires corrective measures to compensate for the bias and the second requires converters to convert from one base to another.

An algorithm that uses "sign-magnitude" and "extended 2's complement" code to represent numbers (data and coefficients) is proposed. By using this new algorithm, the results of calculations on numbers are given directly in the 2's complement form.

The operand \( x_i \), using "sign-magnitude" representation, will be

\[
x_i = (-1)^{\delta x_i} \sum_{j=1}^{N} x_{ij} \cdot 2^j
\]

(21)

where \( \delta x_i \) - the sign bit is 0 for \( x_i > 0 \) and 1 for \( x_i < 0 \).

Also the coefficient \( A_i \) will be

\[
A_i = (-1)^{\delta x_i} \sum_{k=1}^{L} a_{ik} \cdot 2^k
\]

(22)

where \( \delta a_i \) - the sign bit is 0 for \( A_i > 0 \) and 1 for \( A_i < 0 \).

Substituting (21) and (22) into (1) and rearranging the order of the summation, we get

\[
y_n = \sum_{i=0}^{M-1} (-1)^{\delta x_i} \sum_{j=1}^{N} x_{ij} \cdot 2^j \cdot \sum_{k=1}^{L} a_{ik} \cdot 2^k
\]

\[
= \sum_{j=1}^{2^j} \sum_{k=1}^{L} \sum_{i=0}^{N} (-1)^{\delta x_i} (-1)^{\delta a_i} a_{i,k} x_{ij} \cdot 2^k \cdot 2^j
\]

(23)

where

\[
C_j = \sum_{k=1}^{L} 2^j
\]

(24)

and

\[
S_{kj} = \sum_{i=0}^{N} (-1)^{\delta x_i} (-1)^{\delta a_i} a_{i,k} x_{ij}
\]

(25)
Since \( a_{ik} \cdot x_{ij} \in \{-1, 0, +1\} \) then an up-down counter shown in Figure 18 can be used to realize \( S_{k,j} \), which in turn can be either positive or negative. Let us assume that the word length of \( S_{k,j} \) is \( P \) and that \( S_{kj} \) is represented in the 2's complement notation. \( S_{kj} \) will be given by

\[
S_{kj} = \sum_{q=0}^{p-1} S_{kj,q} \cdot 2^q
\]

(26)

where \( S_{kj,p-1} \) represents the sign bit of \( S_{kj} \). If we use equation (26) directly to calculate \( C_j \) and hence \( y_n \), the result of such a calculation will be wrong. This is due to the presence of the weighting factor \( 2^k \) in equation (24), which makes the numbers represented by \( S_{kj} \cdot 2^k \) to be variable wordlength numbers. Note that fixed point notation is used to represent \( S_{kj} \), hence an overflow from the sign bit (bit number \( p-1 \)) of any \( S_{kj} \) number will be added to the unsigned portion of the \( S_{kj+q-1} \) number. This will seriously affect the final result.

The simplest approach to avoid this problem is to use an "extended 2's complement" notation to represent the \( S_{kj} \) numbers. It is well known that the sign bit can be left extended without changing the numerical value of the number in the 2's complement notation. The \( S_{kj} \) numbers, using the extended 2's complement notation, are defined by:

\[
S_{kj} = \sum_{q=0}^{p-2} S_{kj,q} \cdot 2^q + \sum_{q=p-1}^{(p-1)+L-k} S_{kJ,q} \cdot 2^q
\]

(27)

\[
= \sum_{q=0}^{(p-1)+L-k} S_{kj,q} \cdot 2^q
\]

(28)

where \( S_{kj,q} = S_{kj,p-1} \) for \( q > p-1 \)

(29)

By defining the extended 2's complement this way, the numbers \( S_{kj} \cdot 2^k \) of equation (24) will have fixed wordlength of \( L + P \) bits. In this case, an overflow from the sign bit of any \( S_{kj} \) number will propagate through the additional sign bits of the same number.
This will prevent the addition of the overflow to any other number and also lead to a correct answer. As the sign-bit left extension leaves the 2's complement number invariant, the \( C_j \) number is now obtained by adding the \( L \) \( S_{kj} \) extended numbers conventionally including throwing away all carry overflows from the \( (L+P-1) \)th stage. To obtain the \( (L+P) \) -bits of \( C_j \), using the proposed algorithm, equation (24) is to be modified to be the sum of an \( L+P \) numbers that is:

\[
C_j = \sum_{k=0}^{L+P-1} S_{kj} \cdot 2^k
\]  

(30)

where \( S_{kj} = 0 \) for \( K > L \)  

(31)

As an example, let us consider how to obtain the \( C_j \) number by considering a special case when \( L = 6 \) and the wordlength of \( S_{kj} \) is 4 bits. Figure 19 shows the arrangement of the \( S_{kj} \) numbers. In this figure, we dropped the subscript \( j \) since we are going to calculate for certain \( j \). Two ways suggested by commercially available MSI/LSI components of implementing the \( C_j \) number are shown in figures 20 and 21. \( y_n \) can similarly be implemented using the same arrays used for realizing the \( C_j \) number.

4.5. Realization of \( M \)th Order Non-recursive Digital Filter.

A digital filter is characterized by an input-output relationship of the form:

\[
Y_n = \sum_{k=1}^{M} a_k \cdot X_{n-k}
\]

(32)

where \( \{x_n\} \) is the input sequence,

\( \{y_n\} \) the output sequence

and \( \{a_k\} \), is the nonrecursive filter coefficients. The general block-diagram of an \( M \)th order, non-recursive digital filter is shown in Figure 23. Figure 22 shows how to realize the bit product matrix \( a_{i,k} \cdot x_{n-1,j} \)
Figure 19. Bit-position of the $s_{kj}$ numbers. (Arranged to obtain $c_j$ number).
Figure 20. Implementation of $C_j$ number using Carry Save Arrays.
Figure 21. Implementation of $C_j$ number using CSA and CLA.
\[ i = 0, 1, 2, \ldots, M-1 \]

\[ \text{Sign}(X_i) \]

\[ \text{Sign}(A_i) \]

C - Counter Input

**Figure 22.** Realization of the Matrix of the bit product.

\[ a_{i,k} \cdot x_{n-i,j} \text{ using AND-gates.} \]
Figure 23. General block diagram of the BF
using AND-gates. It could be seen from Figure 23 that the proposed technique is ideally suited to modular hardware implementation.

4.5.1 Hardware Requirements and Computation Time:

From the above, it is clear that the time $T_E$ required to compute one output sample $y_n$ is the sum of:

(a) The time $T_{\text{int}}$, required to form the intermediate quantities $a_i x_{n-i}$

(b) The time $T_s$ required to form the $S_{kj}$ words

(c) The time $T_c$ required to get the $C_j$ numbers.

(d) The time $T_y$ required to get the final output from the $C_j$ numbers.

Therefore, the computation time of one output sample:

$$T_E = T_{\text{int}} + T_s + T_c + T_y$$

(33)

The time $T_s$ depends on the type of counters used and also on whether the calculation is done in serial or in parallel mode. The time $T_c$ and $T_y$ depends on the type of carry save array used to implement the $C_j$ number or the final output $y_n$.

The computation time for the four stages of the filter and also for three methods of implementing the filter is given in Table 1 in terms of gate delays.

From Table 1, it is clear that both the fully parallel scheme and the quasi-parallel scheme could be used for real-time digital signal processing. The computation time of the serial scheme will be inadequate for real-time operation.
The required hardware for the three methods of implementing the filter is also given in table 2.

**TABLE 4.1**

<table>
<thead>
<tr>
<th></th>
<th>FULLY PARALLEL OPERATION</th>
<th>QUASI PARALLEL OPERATION</th>
<th>FULLY SERIAL OPERATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. T&lt;sub&gt;int&lt;/sub&gt;</td>
<td>1 gate delay</td>
<td>M gate delays</td>
<td>M gate delays</td>
</tr>
<tr>
<td>2. T&lt;sub&gt;S&lt;/sub&gt;</td>
<td>(a) Synchronous Counters 4xM* gate delays (b) Parallel Counters 6 gate delays</td>
<td>(a) 4xM gate delays (b) 6 gate delays</td>
<td>(a) 4xMxLxN gate delays (b) 6xLxN gate delays</td>
</tr>
<tr>
<td>3. T&lt;sub&gt;C&lt;/sub&gt;</td>
<td>4(L + P - 1) gate delays</td>
<td>4(L + P - 1) gate delays</td>
<td>4N(L + P - 1) gate delays</td>
</tr>
<tr>
<td>4. T&lt;sub&gt;T&lt;/sub&gt;</td>
<td>4(M + q - 1) gate delays</td>
<td>4(M + q - 1) gate delays</td>
<td>4(M + q - 1) gate delays</td>
</tr>
</tbody>
</table>

T<sub>E</sub> - computation time
T<sub>E</sub> = T<sub>int</sub> + T<sub>S</sub> + T<sub>C</sub> + T<sub>T</sub>
S<sub>kj</sub> = p-bits long
C<sub>j</sub> = q-bits long
* T<sub>syn</sub> = T<sub>p</sub> + T<sub>g</sub> = 4 gate delays [25]
T<sub>syn</sub> - speed of operation of a synchronous counter
T<sub>p</sub> - Propagation delay of one Flip-flop
T<sub>g</sub> - Propagation delay of control gating.
** T<sub>parallel</sub> = 6 gate delays [10]
T<sub>parallel</sub> - speed of operation of a parallel counter
### Table 4.2

**Hardware Requirements**

<table>
<thead>
<tr>
<th></th>
<th>Fully Parallel Operation</th>
<th>Quasi-Parallel Operation</th>
<th>Fully Serial Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>1-No. of AND-</strong></td>
<td>L x M x N</td>
<td>L x N</td>
<td>L x N</td>
</tr>
<tr>
<td><strong>gates</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>2-No. of</strong></td>
<td>L x N</td>
<td>L x N</td>
<td>1</td>
</tr>
<tr>
<td><strong>Counters</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>3-No. of</strong></td>
<td>N + 1</td>
<td>N + 1</td>
<td>1</td>
</tr>
<tr>
<td><strong>Carry-save</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>arrays</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
References:


136


CHAPTER 5

5. CONCLUDING REMARKS

We have described the use of differences to realize a multiplication free digital filter. Two different approaches were considered: the mapping of differentials and the use of One-bit word (DM Signals) in processing. These approaches result in a modular structured digital filters. Modularity is one of the basic ways of producing simple and cheap structures. On the other hand, the high speed of operation of the proposed systems is achieved by getting rid of the multiplication processes, as a result of using differences.

The mapping of differentials can be used with DPCM signals which is represented by more than One-bit. While the second technique is suitable for delta modulated signals (One-bit word signals).

The third approach proposed in this thesis results in effectively multiplication free digital filters in spite of using the full sampled values in processing. This is as a result of:

i. Proposing a new algorithm which converts the standard difference equations of a digital filter to the sum of the weighted numbers $c_j \cdot 2^j$.

ii. A new way of using the carry-save arrays. This approach results also in highly modular structure and high speed of operation.

A comparison between the different designs proposed in each approach is given taking the hardware complexity and the time required to get one output sample as a base of comparison.
More work can still be done, especially in the field of using microprocessors in the implementation of the proposed designs.

A lot of work can also be done in the field of using distributed Arithmetic to realize a multiplication free, high speed digital filters, which could be useful in most applications including those involving real-time operations.