

City Research Online

City St George's, University of London

Citation: Matharu, P. S. (1988). Architectures for the VLSI implementation of digital filters. (Unpublished Doctoral thesis, The City University)

This is the accepted version of the paper.

This version of the publication may differ from the final published version. To cite this item please consult the publisher's version.

Permanent repository link: https://openaccess.city.ac.uk/id/eprint/36070/

Copyright and Reuse: Copyright and Moral Rights remain with the author(s) and/or copyright holders. Copies of full items can be used for personal research or study, educational, or not-for-profit purposes without prior permission or charge, unless otherwise indicated, provided that the authors, title and full bibliographic details are credited, a hyperlink and/or URL is given for the original metadata page and the content is not changed in any way. For full details of reuse please refer to City Research Online policy.

City Research Online: http://openaccess.city.ac.uk/ publications@citystgeorges.ac.uk

ARCHITECTURES FOR THE VLSI IMPLEMENTATION OF DIGITAL FILTERS

by

P. S. MATHARU

A thesis submitted for the degree of

Doctor of Philosophy

School of Electrical Engineering and
Applied Physics
The Centre for Information Engineering

City University

June 1988

BRITISH LIBRARY

IMAGING SERVICES NORTH

Boston Spa, Wetherby
West Yorkshire, LS23 7BQ
www.bl.uk

BEST COPY AVAILABLE.

VARIABLE PRINT QUALITY

BRITISH LIBRARY

IMAGING SERVICES NORTH

Boston Spa, Wetherby
West Yorkshire, LS23 7BQ
www.bl.uk

PAGE NUMBERING AS ORIGINAL

to Maninder

CONTENTS

list of tables (i)

list of illustrations (ii)

acknowledgements (iii)

declaration (viii)

abstract (ix)

abbreviations (x)

CHAPTER ONE : INTRODUCTION

1.1 Filters 1

1.2 Theory of LTI discrete systems 5

1.3 Introduction to digital filters 10

1.4 Approaches to design of digital filter structures 12

1.5 Approaches to implementation of digital filters 16

1.6 VLSI implementation of digital filters 18

1.7 Objectives in this thesis 21

CHAPTER TWO : COMPARISON OF DIGITAL FILTER STRUCTURES FROM THE FINITE WORDLENGTH EFFECTS VIEWPOINT

2.1 Introduction 32

2.2 Coefficient quantization effects 34

2.3 Signal quantization effects 36

2.4 Simulation software 40

2.5 Comparison of structures based on experimental results 44

2.6 Comparison of structures based on stability aspects 51

2.7 Discussion and conclusions 53

CHAPTER THREE : COMPARISON OF DIGITAL FILTER STRUCTURES FROM THE HARDWARE IMPLEMENTATION VIEWPOINT

3.1 Introduction 84

3.2 Hardware complexity 86

3.3 Multiplexability aspects 88

3.4 Speed aspects 90

3.5 Discussion and conclusions 92

CHAPTER FOUR : THE CASCADED UNIT ELEMENTS WAVE DIGITAL FILTER

4.1	*Introduction*	*106*
4.2	*Theoretical aspects*	*107*
4.3	*A design program based on a heuristic approach*	*110*
4.4	*An architecture for the cascaded unit elements WDF*	*116*
4.5	*Simulation of architecture at functional level*	*122*
4.6	*VLSI implementation aspects*	*124*
4.7	*Discussion and conclusions*	*125*

CHAPTER FIVE : LATTICE WAVE DIGITAL FILTERS

5.1	*Introduction*	*136*
5.2	*Theoretical aspects*	*138*
5.3	*Lattice WDFs based on first and second order all-pass sections*	*139*
5.4	*Lattice WDFs based on cascaded unit elements*	*144*
5.5	*Lattice WDFs based on LC networks*	*147*
5.6	*Discussion and conclusions*	*149*

CHAPTER SIX : SIMULTANEOUS PROCESSING OF TWO SIGNALS USING WDFs BASED ON UNIT ELEMENTS

6.1	*Introduction*	*162*
6.2	*Theoretical aspects*	*163*
6.3	*Implications for digital implementations of discrete systems*	*165*
6.4	*The cascaded unit elements WDF*	*167*
6.5	*Lattice WDFs*	*169*
6.6	*The LC ladder WDF*	*171*
6.7	*Discussion and conclusions*	*173*

CHAPTER SEVEN : A RECONFIGURABLE PROGRAMMABLE ARCHITECTURE FOR CASCADED UNIT ELEMENTS AND LATTICE WDFs

7.1	*Introduction*	*181*
7.2	*Architecture*	*182*
7.3	*Functional level simulation*	*186*
7.4	*Simulation examples*	*188*
7.5	*Discussion and conclusions*	*190*

CHAPTER EIGHT : A RECONFIGURABLE PROGRAMMABLE ARCHITECTURE FOR LC LADDER AND LATTICE WDFs

8.1	*Introduction*	*200*
8.2	*LC ladder and lattice WDF implementations based on a single three port adaptor*	*201*
8.3	*Architecture*	*203*
8.4	*Functional level simulation*	*207*
8.5	*Simulation examples*	*208*
8.6	*Discussion and conclusions*	*210*

CHAPTER NINE : CONCLUSIONS AND COMMENTS

9.1	*Summary*	*218*
9.2	*Further work*	*222*

REFERENCES 226

APPENDICES 231

LIST OF TABLES

		PAGE
Table 2.1	Summary of coefficient wordlength requirements for 5 percent variation in tolerance scheme parameters	56
Table 2.2	Summary of coefficient wordlength requirements for 20 percent variation in tolerance scheme parameters	57
Table 2.3	Summary of signal to noise ratio values for cosine input	58
Table 2.4	Summary of signal to noise ratio values for gaussian distributed random input	59
Table 2.5	Summary of signal to noise ratio values for uniformly distributed random input	60
Table 2.6	Difference in signal to noise ratio values between Gray–Markel normalized and the rest expressed in signal bit equivalents	61
Table 3.1	Element counts for structures under consideration	93
Table 3.2	Gate counts for structures under consideration	94
Table 3.3	Gate counts for seventh order filter	95
Table 3.4	Critical paths for structures under consideration	96
Table 3.5	Critical path complexities in terms of 2–input adder equivalents for structures under consideration	97
Table 5.1	Status of pipeline registers	150
Table 7.1	Clock signals	191
Table 7.2	Control signals	192
Table 7.3	Reconfiguration scheme	193
Table 7.4	Mapping of filter coefficients onto coefficient RAM	194
Table 8.1	Mapping of coefficients onto RAMs for LC ladder operation	212

LIST OF ILLUSTRATIONS

		PAGE
Figure 1.1	Continuous–time signal	24
Figure 1.2	Discrete–time signal	24
Figure 1.3	Representation of a discrete system	24
Figure 1.4	Elements of a discrete system	25
Figure 1.5	Tolerance scheme	25
Figure 1.6	Direct form I digital filter structure	26
Figure 1.7	Direct form II	26
Figure 1.8	Cascaded second order sections	27
Figure 1.9	Parallel second order sections	27
Figure 1.10	Gray–Markel structures	28
Figure 1.11	Wave representation of a capacitor and its digital equivalent	29
Figure 1.12	Wave digital representations of some analogue elements	29
Figure 1.13	n–port parallel adaptor	30
Figure 1.14	Two port representation of series inductor and its wave digital equivalent	31
Figure 2.1	Illustration of tolerance scheme	62
Figure 2.2	Replacement of non–ideal multiplier with ideal multiplier and noise source	62
Figure 2.3	Extraction of tolerance scheme parameters from the amplitude response	63
Figure 2.4	Seventh order cascaded unit elements WDF	63
Figure 2.5	Difsip example	64
Figure 2.6	Difsip example	65
Figure 2.7	coefficient wordlength requirements for five percent variation in tolerance scheme.	67

Figure 2.8	coefficient wordlength requirements for twenty percent variation in tolerance scheme.	68

COSINE INPUT

Figures 2.9	signal to noise ratios for different types of roundoff	69
and 2.10	for each structure.	70
Figure 2.11	signal to noise ratios for rounding for all structures.	71
Figure 2.12	signal to noise ratios for magnitude truncation for all structures.	72
Figure 2.13	signal to noise ratios for 2's complement truncation for all structures.	73
Figure 2.14	maximum amplitude of input signal that does not cause overflow – shown for all structures.	74

GAUSSIAN DISTRIBUTED RANDOM INPUT

Figures 2.15	signal to noise ratios for different types of roundoff	75
and 2.16	for each structure.	76
Figure 2.17	signal to noise ratios for rounding for all structures.	77
Figure 2.18	signal to noise ratios for magnitude truncation for all structures.	78
Figure 2.19	signal to noise ratios for 2's complement truncation for all structures.	79
Figure 2.20	maximum amplitude of input signal that does not cause overflow – shown for all structures.	80
Figure 2.21	RMS of noise signal for gaussian input and rounding – shown for all structures.	81
Figure 2.22	RMS of noise signal for gaussian input and magnitude truncation – shown for all structures.	82
Figure 2.23	RMS of noise signal for gaussian input and 2's complement truncation – shown for all structures.	83
Figure 3.1a	Direct form II	98

Figure 3.1b	Direct form II redrawn to minimize critical path	98
Figure 3.2	Cascaded second order sections	99
Figure 3.3	Parallel second order sections	100
Figure 3.4	Gray–Markel normalized ladder	101
Figure 3.5	Cascaded unit elements WDF	102
Figure 3.6	Lattice WDF based on all–pass sections	103
Figure 3.7	LC ladder WDF	104
Figure 3.8	IVR generalized WDF	105
Figure 4.1	Two port network comprising commensurate length transmission line	126
Figure 4.2	Cascaded transmission line filter	126
Figure 4.3	Fifth order Chebyshev low–pass filter	127
Figure 4.4	Cascaded unit elements WDF	127
Figure 4.5	Typical coefficient curves for cascaded unit elements WDFs	128
Figure 4.6a	Relationsip between coefficient curves and stopband loss	129
Figure 4.6b	Relationsip between coefficient curves and bandwidth	129
Figure 4.7	Equivalent cascaded unit elements WDF structures	130
Figure 4.8a	Pipelining scheme for two port adaptor	131
Figure 4.8b	Cascaded unit elements WDF	131
Figure 4.9	Architecture for cascaded unit elements WDF	132
Figure 4.10a	Control signals	133
Figure 4.10b	Clock signals	134
Figure 4.11	Frequency response of architecture	135
Figure 5.1	The symmetrical lattice filter	151
Figure 5.2	Unbalanced equivalent of symmetric lattice	151

Figure 5.3	Wave flow diagram for lattice filter	152
Figure 5.4	Simplified wave flow diagram for lattice filter	152
Figure 5.5	First and second order all-pass sections based on circulators	153
Figure 5.6	11th order lattice WDF based on chain connection of first and second order all-pass sections	154
Figure 5.7	Modified lattice WDF	154
Figure 5.8	General structure for lattice arm	155
Figure 5.9	Multiplexed architecture for lattice arm	155
Figure 5.10	General structure for a modified lattice arm	156
Figure 5.11	Multiplexed architecture for a modified lattice arm	156
Figure 5.12	Pipelined substructure	157
Figure 5.13	Multilplexed pipelined architecture for modified lattice arm	158
Figure 5.14	Synthesis of reactance function using Richards' procedure	159
Figure 5.15	Lattice WDF based on cascaded unit elements	159
Figure 5.16	Cascaded unit elements WDF	159
Figure 5.17a	Cauer form I LC ladder	160
Figure 5.17b	Corresponding wave digital filter network	160
Figure 5.18	Kuroda's transforms	160
Figure 5.19a	Odd order LC ladder with inserted unit elements	161
Figure 5.19b	Corresponding wave digital network	161
Figure 5.20a	Even order LC ladder with inserted unit elements	161
Figure 5.20b	Corresponding wave digital network	161
Figure 6.1	Equivalent representations of cascaded unit elements WDFs	174
Figure 6.2a	Process simulated by DIFSIP	174
Figure 6.2b	Process simulated by modified version of DIFSIP	174

Figure 6.3	Results for UEWDF	175
Figure 6.4	Equivalent WDF structures	176
Figure 6.5	Results for figure 6.4	177
Figure 6.6	Fifth order lattice WDF based on cascaded unit elements	178
Figure 6.7	Results for figure 6.6	179
Figure 6.8	Equivalent LC ladder WDF structures	178
Figure 6.9	Results for figure 6.8	180
Figure 7.1	A reconfigurable programmable architecture for cascaded unit elements and lattice WDFs	195
Figure 7.2	Structures implemented by architecture	196
Figure 7.3	Results for UEWDF operation	197
Figure 7.4	Results for lattice WDF operation	198
Figure 7.5	Time domain results for UEWDF operation	199
Figure 8.1	LC ladder WDF	213
Figure 8.2a	Symbolic representation of three port parallel adaptor	213
Figure 8.2b	Pipelined three port parallel adaptor	213
Figure 8.3	WDF derived from lattice filter with Cauer synthesis of lattice reactances	214
Figure 8.4	Two port adaptor with one dependent port	214
Figure 8.5	A reconfigurable programmable architecture for LC ladder and lattice WDFs	215
Figure 8.6	Results for LC ladder WDF operation	216
Figure 8.7	Results for lattice WDF operation	217

ACKNOWLEDGEMENTS

I am deeply indebted to Dr Stuart Lawson for his support, guidance and encouragement throughout this project. I note with gratitude his efforts to continue this role over the last few months during which he has been involved with his duties at Warwick University.

I would like to thank the Centre for Information Engineering and, in particular, [redacted] for allowing me the opportunity of working on this project.

I am also indebted to AMSYS (GEC RESEARCH) for allowing me the use of their wordprocessing facilities and the time to finish my work. The help given to me by [redacted] is particularly appreciated.

DECLARATION

I grant powers of discretion to the University Librarian to allow this thesis to be copied in whole or in part without further reference to me. This permission covers only single copies made for study purposes, subject to normal conditions of acknowledgement.

ABSTRACT

Digital Signal Processing (DSP) has a number of well known advantages compared with analogue signal processing. Its main drawback of complexity is becoming more manageable with the advent of VLSI systems. The digital filter is an important DSP function which has wide application and plays some part in most DSP systems. The practicality of DSP is therefore dependent on the efficient implementation of digital filters with VLSI systems. The aim in this thesis is to design architectures for this purpose with an emphasis on identifying features of structures which can be exploited in the design of efficient architectures.

The first step towards this goal is the selection of suitable digital filter structures. This issue is addressed from the viewpoints of finite wordlengths and hardware implementation. Appropriate measures are introduced and used to assess the suitability of a number of structures from the literature ; direct form II, cascaded second order sections, parallel second order sections, Gray–Markel normalized ladder, cascaded unit elements WDF, lattice WDF, LC ladder WDF and IVR generalized WDF. It is shown that of these, the cascaded unit elements WDF and the lattice WDF are best suited to VLSI implementation.

A novel finite wordlength design program for cascaded unit elements WDFs is presented. This program is based on a heuristic approach and exploits a number of properties of cascaded unit elements WDFs which have been identified. Also, programs for the design of lattice WDFs based on cascaded unit elements and LC networks are given.

An architecture for the cascaded unit elements WDF is given which exploits the inherent parallelism of the structure to optimize the speed in a multiplexed scheme. This is achieved by pipelining the basic section. Architectures for lattice WDFs based on cascaded unit elements and all–pass sections are also presented. Simple modifications to the structure, multiplexing and pipelining are used to optimize efficiency.

The relationship between a theoretical discrete system and its synchronous digital realization is explored. This is used to show that synchronous digital realizations of WDFs based on unit elements can be used to process two signals simultaneously without requiring any modifications. It is necessary for the two signals to be skewed by half the sampling period

This property is exploited in the design of a reconfigurable, programmable architecture which can be used to implement two separate UEWDFs, two UEWDFs in cascade or a lattice WDF. Appropriate results are given to demonstrate the efficiency and versatility of this architecture. Similar ideas are used to design an architecture capable of implementing two separate LC ladder WDFs, two LC ladder WDFs in cascade or a lattice WDF.

Closing arguments are given to show that the work presented in this thesis may be used in the development of a CAD system for the VLSI implementation of digital filters which is based on a standard cell approach.

ABBREVIATIONS

A/D	analogue to digital converter
CAD	computer aided design
D/A	digital to analogue converter
DSP	digital signal processing
FIR	finite impulse response
IIR	infinite impule response
LC	inductor capacitor
LTI	linear time invariant
MOS	metal oxide semiconductor
pbej	passband edge frequency
pbrip	passband ripple
sbej	stopband edge frequency
sbgain	maximum gain in stopband
UEWDF	cascaded unit elements WDF
VLSI	very large scale integration
WDF	wave digital filter

CHAPTER ONE

INTRODUCTION

In the past, the complexity of digital signal processing functions made it impractical to implement such functions and this approach was only employed when an analogue alternative was not available. The advances in fabrication technology and the advent of VLSI systems has made it possible to integrate complete digital signal processing functions on single chips. This, coupled with the continuing replacement of analogue systems with digital systems where practical, has led to the current activity in research into VLSI implementation of digital signal processing functions.

The digital filter is an important digital signal processing function with applications in communication systems, control systems, measurement systems etc. Efficient implementations of digital filters with optimum speed performance and silicon area requirements are therefore a desirable goal. To this end, we will consider the design of efficient architectures for the VLSI implementation of digital filters in this thesis.

This chapter begins with a brief outline on filters with an emphasis on their history and the development of digital filters. The digital filter is a linear time–invariant discrete system and the theory of such systems is given in section 1.2. An introduction to digital filters is given in section 1.3 including discussions on the processes involved in the design of digital filters and the choice of digital filter structure. A number of approaches to the design of digital filter structures reported in the literature are outlined in section 1.4. A discussion on approaches to the implementation of digital filters is given in section 1.5 and VLSI implementation aspects are discussed in section 1.6. The chapter concludes with a section on the objectives in this thesis.

1.1 FILTERS

In the early 1900s, the concept of loading transmission lines with lumped inductances in order to reduce attenuation was proposed by Heaviside. The frequency response of

such a loaded transmission line is the same as that of a low pass filter. By reducing the distributed capacitance to lumped capacitances, and distributed inductance to zero in the loaded line, the LC ladder low pass filter is obtained. This type of filter was independently invented during world war I by Campbell and Wagner to whom the birth of filter theory is attributed [1]. Since then, the theory and practice of filter design has advanced considerably with a consequent broadening of the term 'filter' which may now be defined as a network which is required to have a prescribed response for a given excitation. The response requirement may be given in terms of time or frequency [2].

There are three categories of electronic filters; passive, active and digital.

PASSIVE FILTERS

Passive filters evolved from the original work of Campbell and Wagner and comprise either lumped inductances, resistances and capacitances or, for very high frequency applications, distributed inductances, capacitances and resistances.

The problems associated with passive filters are mainly those due to variation in component values due to losses, temperature, ageing etc. At lower frequencies, the bulk of the inductors is a major problem and in general, it is desirable to eliminate the inductor because of the expense and losses associated with it. This can be achieved with active filters. Because of the limitations of active components at high frequencies, such applications will always require passive filters [3].

Of particular interest are passive lossless filters which are inserted between equal resistive terminations. These exhibit a particularly low sensitivity to component variations [4,5]. For this reason, these filters have been used as a basis for theoretical developments in active and digital filters in attempts to emulate this property [3].

ACTIVE FILTERS [6]

The need for inductorless filters and the invention of the transistor in the late 1940s led to the invention of the active filter. In 1953, Linvill showed that a filter comprising resistors, capacitors and an active element was theoretically possible. The emergence of

the monolithic operational amplifier as an off the shelf component in the 1960s and subsequent research meant that by the 1970s, the active RC filter was established as a practical product. The increasing integration of systems onto single chips of silicon and the difficulty of implementing resistors in MOS technology led to the invention of the switched capacitor in the late 1970s. Today, this is the dominant active filter technology.

DIGITAL FILTERS

In the late 1950s, digital filtering was mainly performed in non–real time on computers and was used in cases where filtering with analogue components was not possible. Digital filters implemented in hardware were not practical because of the speed, cost and size of digital components.

The advances in microelectronics since then, in parallel with developments in digital filter theory and design, have made it possible to implement complete digital filters on single chips. With continuing advances in fabrication technology and design methods, digital filters are becoming increasingly practical.

Digital filters have a number of advantages compared with analogue filters.

DIGITAL FILTERS VERSUS ANALOGUE FILTERS

Advantages of digital filtering :

(1) Performance from unit to unit is stable and repeatable.

(2) Arbitrarily high precision is achieved that is limited only by the number of bits carried in memory and the input and output resolution capabilities.

(3) No impedance matching problems exist in the digital domain.

(4) Critical filter break frequencies can be placed without restrictions (influences the precision required).

(5) Component value variation problems normally associated with capacitors,inductors and resistors due to temperature changes or age are non–existent.

(6) Adaptive filtering is possible.

(7) The possibility of time-sharing major implementation sections exists (adders, subtractors, multipliers etc).

(8) Periodic calibration as is required with analogue circuits is eliminated.

(9) Linear phase characteristics are possible.

1.2 THEORY OF LTI DISCRETE SYSTEMS [7,8]

An ideal digital filter is a linear time-invariant (LTI) discrete system. In this section a brief introduction is given to discrete signals and systems.

DISCRETE SIGNALS

A signal can be defined as a function that conveys information, generally about the state or behaviour of a physical system. In electrical systems, signals take the form of voltages or currents.

Signals can be represented mathematically as functions of time. The independent variable of the mathematical representation of a signal may be either continuous or discrete. Continuous-time signals (figure 1.1) are signals that are defined at a continuum of times and thus are represented by continuous variable functions. Discrete-time signals (figure 1.2) are defined at discrete times and thus the independent variable takes on only discrete values, i.e. discrete-time signals are represented as sequences of numbers.

A discrete-time signal may be obtained from a continuous-time signal by the process of uniform sampling. According to the sampling theorem, the two signals are entirely equivalent provided that the sampling rate is at least twice the highest frequency component of the continuous-time signal. If this condition is not met, then aliasing errors will result.

The most elementary discrete-time signal is the unit-sample sequence which is defined as

$$d(n) = 0 \quad \textit{if} \quad n \textit{ is not equal to } 0 \qquad (1.1)$$
$$d(n) = 1 \quad \textit{if} \quad n \textit{ is equal to } 0$$

An arbitrary sequence can be expressed as a sum of scaled delayed unit samples :

$$x(n) = \sum_{k=-\infty}^{\infty} x(k).\, d(n-k) \qquad (1.2)$$

LTI DISCRETE SYSTEMS

A system is defined mathematically as a unique transformation or operation that maps an input sequence $x(n)$ into an output sequence $y(n)$. This is denoted as

$$y(n) = G[x(n)] \qquad (1.3)$$

and is shown diagrammatically in figure 1.3.

A linear discrete system is defined by the principle of superposition. If $y1(n)$ and $y2(n)$ are the responses when $x1(n)$ and $x2(n)$ respectively are the inputs, then a system is linear if and only if

$$G[a.x1(n) + b.x2(n)] = a.G[x1(n)] + b.G[x2(n)] = a.y1(n) + b.y2(n) \qquad (1.4)$$

for arbitrary constants a and b.

A time invariant system is characterized by the property that if $y(n)$ is the response to $x(n)$, then $y(n-k)$ is the response to $x(n-k)$, where k is a positive or a negative integer.

The response of an LTI discrete system to a unit-sample is called the impulse response and this completely characterizes a system. The response of the system to an arbitrary sequence can be obtained by summing weighted impulse responses. This is mathematically described by the convolution sum;

$$y(n) = \sum_{k=-\infty}^{\infty} x(k).h(n-k) \qquad (1.5)$$

where $h(k)$ is the impulse response.

Alternatively, the system may be described by a linear difference equation. This gives the relationship between the input and output sequences in the time domain. In general, for a *pth* order system, the linear difference equation takes the form

$$y(n) = a0.x(n) + a1.x(n-1) + \ldots am.x(n-m) - b1.y(n-1) - b2.y(n-2)\ldots\ldots - bp.y(n-p) \qquad (1.6)$$

where bo, b1...bm, a1, a2.....ap are constants . The values of these constants describe

the dynamical behaviour of the system. p and m are fixed non-negative integers.

Note that the discrete system requires three basic operations ; addition/subtraction, multiplication and delay. Symbolic representations of the operations are shown in figure 1.4. An LTI discrete system can therefore be represented in graphical form as a network of these three elements.

THE Z-TRANSFORM

The Z-transform is used to convert a sequence of numbers into a function of the complex variable Z. The Z-transform of a sequence of numbers $x(n)$ which is identically zero for negative discrete time is defined by

$$Z[x(n)] = X(Z) = \sum_{n=0}^{\infty} x(n).Z^{-n} \qquad (1.7)$$

The Z-transform has a number of properties which are useful in the manipulation of linear difference equations ;

LINEARITY : $Z[x1(n) + x2(n)] = Z[x1(n)] + Z[x2(n)]$ (1.8)

LEFT SHIFT PROPERTY : $Z[x(n-1)] = Z[x(n)].Z^{-1}$ (1.9)

CONVOLUTION -SUM PROPERTY :

$$Z[\sum_{k=-\infty}^{\infty} x(k).h(n-k)] = Z[x(n)].Z[h(n)] \qquad (1.10)$$

THE BILINEAR TRANSFORMATION [7]

The bilinear transformation gives a relationship between the complex frequency variables s and Z where s is the continuous-time complex frequency variable and Z is the discrete-time complex frequency variable;

$$s = \frac{Z - 1}{Z + 1} \qquad (1.11)$$

The bilinear transformation maps the left half of the s-plane onto the inside of the unit circle in the Z-domain, and the right half of the s-plane onto the outside of the unit circle in the Z-domain. The implication of this is that a transfer function that is stable in the s-domain will also be stable in the Z-domain. The mapping of the entire y-axis in the s-domain onto the unit circle in the Z-domain results in a warping of the frequency response and the relationship between frequencies in the s-domain and the Z-domain is as follows;

$$Q = \tan(\omega.T/2) \qquad (1.12)$$

where ω *is the continuous-time frequency,* Q *is the discrete-time frequency and* T *is the sampling period.*

Therefore, in designs using the bilinear transformation, 'prewarping' of the continuous-time frequency response is carried out in order to achieve the desired result in the Z-domain.

THE TRANSFER FUNCTION

An LTI discrete system can be described in the complex frequency domain (Z-domain) with a transfer function. The transfer function is defined as the ratio of the output signal to the input signal in the Z-domain ;

$$i.e.\quad G(Z) = Y(Z)/X(Z) \qquad (1.13)$$

from the earlier definition of a linear difference equation (equation 1.6)

$$G(Z) = \frac{Y(Z)}{X(Z)} = \frac{a0 + a1.Z^{-1} + a2.Z^{-2} + \ldots + am.Z^{-m}}{1 + b1.Z^{-1} + b2.Z^{-2} + \ldots + bp.Z^{-p}} \qquad (1.14)$$

The transfer function can be obtained by taking the Z–transform of the impulse response. The frequency response can be obtained from the transfer function by setting $|Z| = 1$.

STRUCTURE

It was pointed out earlier that a discrete time system can be represented graphically with a network of adders/subtractors, multipliers and delays. The topology of such a network is defined as the STRUCTURE of the discrete system.

It is important to note that an infinite variety of structures may have the same transfer function.

1.3 INTRODUCTION TO DIGITAL FILTERS

The first stage in the design of discrete filters is concerned with obtaining a transfer function satisfying the required response. Transfer functions which comprise ratios of polynomials exhibit impulse responses which are of infinite duration. Such filters are known as IIR filters. On the other hand, transfer functions comprising a polynomial exhibit impulse responses of finite duration. Such filters are known as FIR filters. FIR filters have the advantage of being unconditionally stable and offer the possibility of linear phase response. The disadvantage of FIR filters is that higher orders are required compared with IIR filters to satisfy a given tolerance scheme.

We will only consider IIR filters in this thesis.

DIGITAL REALIZATION OF DISCRETE SYSTEMS

A digital signal is one in which both the independent and dependent variables are discrete. An ideal digital signal (i.e. one in which the number of discrete levels for the dependent variable is infinite) can be used to exactly represent a discrete signal. A digital filter can be described as a discrete filter in which all the signals are digital.

The elements of a discrete system (figure 1.4) can be realized with digital equivalents. Ideal digital adders and multipliers have zero propagation delay. In practice, a finite propagation delay is associated with both these elements. The delay elements may be realized with delay lines or shift registers.

Asynchronous digital realizations are possible with delay lines being used to realize the delay element. In such systems, the delays associated with adders and multipliers have to be considered as part of the design process. The obvious difficulties associated with this approach mean that it is not generally employed in practice.

In practice, synchronous digital realizations are generally employed. In such systems, the shift register is generally employed to realize the delay element and since the system is clocked, the propagation delays associated with multipliers and adders are only considered in determining the maximum clocking frequency.

Since the digital computer is a synchronous system, it can be used to realize a digital filter.

DESIGN OF DIGITAL FILTERS [9,10]

In designing frequency selective filters, the starting point is a specification given in terms of a tolerance scheme (figure 1.5). Such a scheme defines the boundaries within which the actual frequency response must lie.

The first stage of the design process is to find a suitable transfer function which satisfies the tolerance scheme. This is essentially an approximation problem. For IIR filters, the most common approach is to solve the approximation problem in the s-domain by using classical approximation methods such as Butterworth, Chebyshev etc. The Z-domain transfer function is then obtained by means of the bilinear transformation. The approximation problem can also be solved in the Z-domain with the aid of optimization algorithms.

The next stage in the design process is to obtain a structure to realize the transfer function. It has been pointed out that a particular transfer function may be satisfied by an infinite number of structures. The choice of structure is extremely important for a number of reasons.

CHOICE OF DIGITAL FILTER STRUCTURE

In a real digital filter, both the signals and coefficients have to be represented with a finite number of bits. This process is known as quantization. Coefficient quantization causes a deviation in the frequency response of the digital filter. Signal quantization causes quantization noise and may cause limit cycles. Different structures behave differently under these conditions. The digital filter designer is interested in choosing structures which exhibit low sensitivity to quantization.

From the implementation viewpoint, the complexity of the structure is an important consideration. In high speed applications, the degree of inherent parallelism needs to be considered. Also, the amenability of the structure to LSI integration and multiplexability should be considered.

1.4 APPROACHES TO DESIGN OF DIGITAL FILTER STRUCTURES

The general form of a digital filter transfer function is given in equation (1.14). The corresponding linear difference equation is given in equation (1.6). From this equation, the structure of figure 1.6 can be derived. This was the first IIR digital filter structure to be discovered and is known as the Direct form I. Implementation of this structure showed that it is particularly sensitive to coefficient quantization and exhibits high roundoff (quantization) noise. Research over the past two decades has been directed towards finding alternative structures which exhibit low coefficient sensitivity and generate low roundoff noise.

In this section, some approaches to the design of digital filter structures will be described.

DIRECT FORM II [7]

By examination of figure 1.6 and equations (1.6) and (1.14), it can be seen that the left half of figure 1.6 realizes the zeroes of $G(Z)$ whereas the right half realizes the poles of $G(Z)$. It is known that the order in which the zeroes and poles are realized does not matter. Therefore, it is possible to realize $G(Z)$ as shown in figure 1.7.

This type of realization is called direct form II. It can be seen that this has the minimum number of delays necessary to implement $G(Z)$. For this reason, it is also called a canonic form network.

CASCADED SECOND ORDER SECTIONS [7]

The direct form structures were obtained directly from the transfer function $G(Z)$. We can rewrite $G(Z)$ as a product of second order functions as follows:

$$G(Z) = A. \prod_{k=1}^{j} \frac{1 + a1k.Z^{-1} + a2k.Z^{-2}}{1 + b1k.Z^{-1} + b2k.Z^{-2}} \qquad (1.15)$$

where $j = p/2$ for p even and $j = (p+1)/2$ for p odd. For p odd, $a21 = b21 = 0$.

(Note : we have to cascade second order sections because first order sections require complex coefficients to realize complex poles).

The structure is shown in figure 1.8. Each second-order section can be realized with direct form structures.

PARALLEL SECOND ORDER SECTIONS [7]

As an alternative to factorizing the numerator and denominator polynomials of $G(Z)$, we can express $G(Z)$ as a partial fraction expansion in the form

$$G(Z) = B + \sum_{k=1}^{j} \frac{d0k + d1k.Z^{-1}}{1 + a1k.Z^{-1} + a2k.Z^{-2}} \qquad (1.16)$$

The structure is shown in figure 1.9. Each second order section can be realized with direct form structures.

GRAY-MARKEL STRUCTURES

Gray and Markel have developed techniques for synthesizing digital lattice and ladder structures from direct forms [11-14]. These techniques are based on orthogonal polynomial expansion of the direct form numerator and are used to transform the direct form into a cascade of lattice or ladder structures.

The general order of the cascade for a pth order filter is shown in figure 1.10a. The sections G(Z) can be realized in any one of the following forms.

(1) Two multiplier lattice as shown in figure 1.10b.

(2) Three multiplier ladder as shown in figure 1.10c.

(3) Normalized ladder as shown in figure 1.10d.

In addition, the sections may be realized with one multiplier lattices [11].

WAVE DIGITAL FILTER STRUCTURES

WDF structures are derived from passive analogue filters [15–17]. The analogue network is represented in terms of wave variables in the s–domain. Each element of the analogue network is treated separately and the Z–domain equivalents are obtained by application of the bilinear transformation.

Consider, for example, the capacitor shown in figure 1.11. The wave representation of the network is as follows

$$\begin{aligned} A &= V + IR \\ B &= V - IR \end{aligned} \qquad (1.17)$$

where A is the incident voltage wave and B is the reflected voltage wave. The quantity R is known as the port resistance and in the case of the capacitor is given by 1/ C. By replacing s with its bilinear transformation representation in Z and solving the equations, the following relationship is obtained

$$B = A.Z^{-1} \qquad (1.18)$$

The digital network to realize this equation is shown in figure 1.11. The wave digital representation of the most common analogue elements is shown in figure 1.12.

When analogue elements of different port resistances are interconnected in a parallel or series manner, a part of the incident wave will be transmitted and part reflected at the interconnection. This must be taken into account in the digital equivalents. This is done by first mathematically describing the interconnection in terms of wave variables and then obtaining wave digital equivalents in the manner described before. The resulting digital networks are known as adaptors and are either n–port parallel or n–port series adaptors according to the type of interconnection being represented. The main features of the derivation of the n–port parallel adaptor are illustrated in figure 1.13.

Therefore, for a given analogue filter, each element is replaced by its digital equivalent and each interconnection is replaced by an adaptor in order to obtain a WDF.

GENERALIZED WAVE DIGITAL FILTER STRUCTURES

The generalized WDF structures [18–20] are also derived from passive analogue filters. Each element is considered as a two port network and is described in terms of wave variables. In this method, adaptors which are necessary in the one port description (see section on wave digital filter structures) are not needed explicitly but are included as part of the equivalent wave flow diagrams. Since the interconnecting port resistances are identical, wave digital equivalents of the analogue elements can be directly interconnected. As an example, the digital network for a series inductor is shown in figure 1.14.

The wave description of the two port can be viewed as a linear transformation of the transmission matrix [19,20]. This approach is generalized and conditions are given under which a linear transformation when applied to the transmission matrix will result in a realizable digital network. A number of transformations have been identified [19,20] which allow the derivation of a range of digital filter structures from passive analogue filters.

1.5 APPROACHES TO IMPLEMENTATION OF DIGITAL FILTERS

Implementation methods for digital filters may be classified as either software implementation or hardware implementation.

SOFTWARE IMPLEMENTATION

In this case, the digital filter signal flow diagram is viewed as a graphical representation of the algorithm. Software implementation may be real-time or non-real time. Non-real time applications require the signal that is to be processed to be stored in computer memory. Generally, the number representation with this approach is floating point and the availability of a large dynamic range for representation of both signals and coefficients means that the choice of structure is more flexible than in the other methods to be discussed.

Digital filters can be implemented in real-time software on microprocessors or signal processors. Signal processors [21] are special purpose microprocessors geared towards signal processing applications.

HARDWARE IMPLEMENTATION

For hardware implementation, each element in the signal flow diagram can be implemented with a hardware equivalent in bit-parallel form. In general, digital filters are complex digital networks and the requirements for hardware mean that in most cases such an approach is impractical.

Alternatively, each element in the signal flow diagram can be implemented in bit-serial form. Such an approach will require less hardware but will be slower. Clearly, the tradeoff between the bit-serial and bit-parallel approaches is between complexity and speed. It is also worth pointing out that the increase in the signal and coefficient wordlengths will proportionally decrease the speed in the bit-serial approach and, in the bit-parallel approach, the complexity will be proportionally increased. The complexity in the bit -serial approach will also increase due to an increase in the storage requirements.

Early digital filter implementations employed bit–serial techniques because of the limited capabilities of the available ICs [22–24]. More recently, the availability of parallel multipliers and adders in single ICs have made bit–parallel implementations more practical. When considering VLSI implementation, bit–parallel techniques are necessary when high speeds are required. However, in the interests of minimizing silicon area and power requirements, bit–serial techniques [25–30] should be employed when speed requirements permit.

Depending on the structure on which the digital filter is based, multiplexing may be employed to reduce the complexity. Obviously, this will result in a lower speed. Multiplexing may be based on elements such as multipliers and adders or, it may be done at a higher level such that filters of arbitrary order are implemented by time multiplexed filters of lower order.

The multiplier is the most complex component in digital filters. Therefore, alternative methods of implementing the multiplication function in a more efficient manner are of interest. One such method is the distributed–arithmetic approach [31–34]. This technique is based on implementing the sum–of–products using an algorithm called distributed–arithmetic. The hardware implementation is carried out using ROMs and shift–add circuits. Hence, this technique is also referred to as the ROM–accumulator approach.

In the above discussion, some of the major aspects of digital filter implementation have been outlined. Any combination of these, or the many techniques available for digital system design, may be employed in practice. The exact method chosen will be influenced by the sampling frequency requirements, system requirements and the technology available. However, in any implementation, a common goal is to maximize the sampling frequency and minimize the hardware complexity.

1.6 VLSI IMPLEMENTATION OF DIGITAL FILTERS

An ideal architecture for VLSI implementation should possess the following attributes [35].

REGULARITY

An architecture is said to be regular if it is an ordered repetition of a simple cell. A simple cell is one which comprises a small number of gates.

The VLSI design of a regular architecture is then reduced to the design of the cell. This is the best method of managing the complexity of VLSI design.

PARALLELISM

In any digital system, the degree of parallelism will influence the efficiency of the hardware. Some idea of the efficiency of a particular architecture may be gained by evaluating the number of operations carried out by each element (gate, cell etc) over a period of time. In general, parallelism is employed to reduce the complexity of the propagation path in any system or subsystem so as to increase its throughput rate and thereby increase the efficiency of its elements.

Parallelism may be obtained by pipelining.

SHORT INTERCONNECTION LENGTH

Interconnections in VLSI systems should be kept to a minimum because of the silicon area required and the associated delays.

The transmission of signals through interconnections in ICs is governed by the diffusion equation. This means that the relationship between interconnection length and propagation delay takes an exponential form. Therefore, in the interests of speed, interconnection lengths should be kept as short as possible.

Therefore, an ideal architecture from the VLSI viewpoint should comprise an array of simple cells which only require communication with adjoining cells. Furthermore, during normal operation, all the cells should operate in parallel.

Array type architectures which satisfy the above requirements to a large extent are systolic arrays [36–38] and wavefront arrays [39,40].

Systolic arrays are pipelined array architectures that comprise a set of interconnected cells which are all identical and capable of performing some simple operations. Each cell is known as a processing element (PE) and each PE is only connected to adjoining PEs. Data is synchronously pumped in and out of each PE and during normal operation, all PEs operate in parallel. A disadvantage of systolic arrays is that synchronization requires the distribution of a clock signal over the whole array. This problem is circumvented in wavefront arrays by including control functions within each PE to allow exchange of data between adjoining PEs. This leads to a more complex PE.

Although array type architectures are very suitable for VLSI implementation, their suitability for implementing a particular function is dependent on the efficiency with which the function can be mapped onto the array architecture. The massive parallelism inherent in array architectures is obtained through pipelining and implementation of the pipelining function requires storage elements in each PE. The array architectures can only be used to their full potential when the function to be implemented can also be pipelined to the same degree as the array. Otherwise the additional hardware complexity of the array architecture may mean that it is not the most suitable architecture.

For a given function, a more direct approach to VLSI design may be more appropriate. The availability of standard functions (addition, multiplication, etc) as standard cells in most VLSI technologies means that the design process is simplified. For digital filters, the main building blocks required are adders/subtractors, multipliers and shift registers. These are common functions and a variety of methods are generally available for their design and standard cells are also generally available.

The main aim in the VLSI design of digital filters is to maximize the sampling frequency and minimize the silicon area. A useful goal is to design architectures which will take the form of macrocells in cell based systems such that a single cell may be used to implement a digital filter. The complexity of digital filters means that such architectures must employ multiplexing. This approach can be taken a step further by ensuring that the

basic cell architecture is such that a number of these may be used together to satisfy higher sampling frequency requirements.

1.7 OBJECTIVES IN THIS THESIS

The aim in this thesis is to design architectures for the VLSI implementation of digital filters. In the pursuit of this goal, an emphasis is placed on the relationship between structure and architecture and on exploitation of particular aspects of structures to design efficient architectures.

The first step in the design of architectures for digital filters is to identify digital filter structures which are best suited to VLSI implementation. The following structures were selected for detailed study; direct form II, cascaded second order sections, parallel second order sections, Gray–Markel normalized ladder, cascaded unit elements WDF, lattice WDF, LC–ladder WDF and IVR generalized WDF.

The representation of signals and coefficients with a finite number of bits causes a degradation in filter performance. It is well known that the degree of degradation is dependent on the type of filter structure and for a given specification, some structures will require longer wordlengths for signal and coefficient representation. Also, some structures are known to possess better stability properties and will therefore be more suited to certain applications. In order to retain wider application, structures with good stability properties are to be preferred for VLSI implementation. The above structures will be studied in detail from the viewpoint of finite wordlength effects in chapter 2 and comparisons will be made.

Digital filter structures differ in their hardware requirements and topology. From the hardware implementation viewpoint, some structures are more suitable than others. Important aspects to consider are the hardware complexity, multiplexability and speed. The hardware complexity is determined by the number of elements required to implement a particular structure and is influenced by the finite wordlength properties of the structure. In structures where multiplexing is possible, a filter of arbitrary order may be implemented by time multiplexing a subsection of the filter. Multiplexability allows the tradeoff between speed and amount of hardware. The maximum sampling frequency possible is determined by the complexity of the path through which a signal must propagate in one sampling interval. This is influenced by the degree of parallelism inherent in the structure and the parallelism may, in part, be due to pipelining due to the

delay elements. The previously mentioned structures will be studied in detail from the hardware implementation viewpoint in chapter 3 and comparisons will be made.

Through arguments presented in chapters 2 and 3, it will be shown that of the structures considered, the cascaded unit elements and the lattice WDF are the most suitable for VLSI implementation.

The cascaded unit elements WDF will be considered in detail in chapter 4. A novel program for the design of filters based on this structure will be presented. The program is based on a heuristic approach and is intended for designing filters with finite wordlength coefficients. The multiplexability and parallelism properties of the structures are exploited in the design of a novel architecture for VLSI implementation. Pipelining of the multiplexed section is employed to improve the speed and thereby increase the efficiency of the architecture. A program for the functional level simulation of the architecture will be given.

Lattice WDF structures will be considered in detail in chapter 5. Design programs for lattice WDFs based on cascaded unit elements and LC ladder networks will be presented. Lattice WDFs based on all–pass sections will be considered and a number of architectures for this structure will be given. The lattice WDFs based on cascaded unit elements are the simplest from the VLSI implementation viewpoint and architectures for this type of lattice WDF will also be given.

The realization of a discrete system with a synchronous digital system in which the delay elements are realized with shift registers clocked at the sampling frequency only behaves like the discrete system at the clock edges. Realization of delays of half the sampling period requires clocking at twice the sampling rate and the digital system, in effect, behaves like the discrete system twice in a sampling interval. This property can be exploited to process two signals, skewed by half the sampling period, simultaneously. This will be the subject of chapter 6 where it will be shown that WDFs based on unit elements can be used to process two signals simultaneously. No modifications are required. The two signals to be processed in this manner are simply skewed by half the sampling period with respect to one another and added. The combined signal forms the input to the digital

filter and the output of the filter is resolved to obtain the component signals. Simulation results for a number of examples will be given in order to demonstrate this property.

The above property has been exploited in the design of a reconfigurable programmable architecture which may be used to realize two separate cascaded unit elements WDFs, two cascaded unit elements WDFs in cascade, or a lattice WDF based on cascaded unit elements. The architecture is particularly efficient in terms of hardware requirements in that it is based on a single multiplexed two port adaptor and requires little additional hardware to achieve the versatility that it possesses. Furthermore, the efficiency of the hardware is optimum since during normal operation, two samples (of different signals) are processed during each sampling period. This architecture will be discussed in chapter 7 and a program for functional level simulation will be given with appropriate results to demonstrate that the architecture is functionally correct.

The same ideas as above have been used in the design of a reconfigurable, programmable architecture which may be used to implement two separate LC ladder WDFs, two LC ladder WDFs in cascade, or a lattice WDF based on LC ladder networks. This architecture is based on a three port adaptor and is discussed in chapter 8. A program for functional level simulation and appropriate results are given in chapter 8.

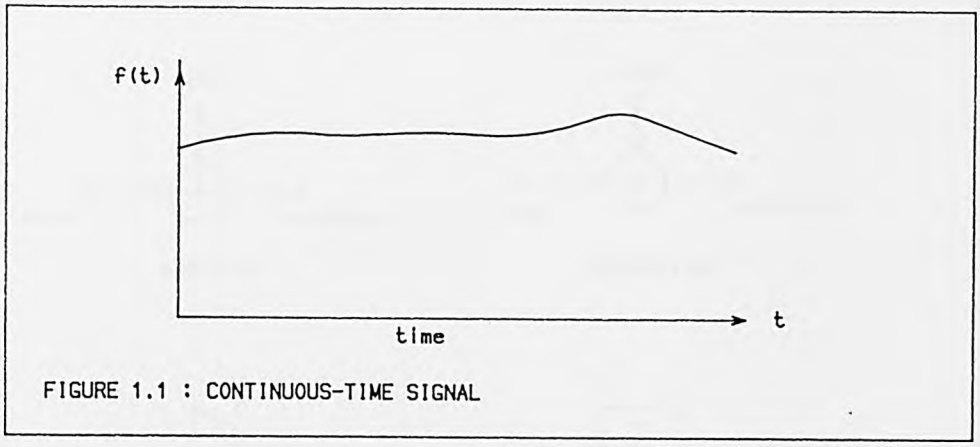


FIGURE 1.1 : CONTINUOUS-TIME SIGNAL

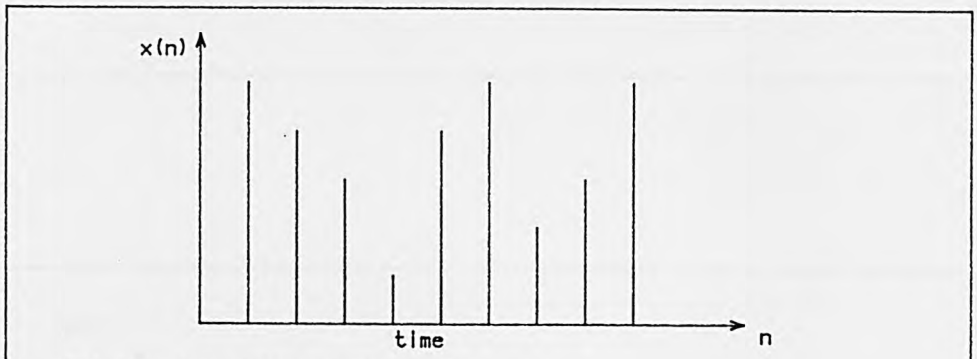


FIGURE 1.2 : DISCRETE-TIME SIGNAL

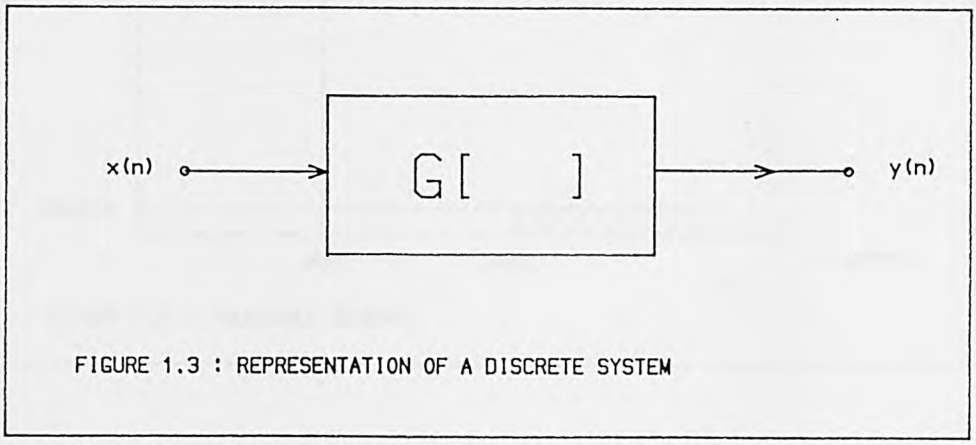


FIGURE 1.3 : REPRESENTATION OF A DISCRETE SYSTEM

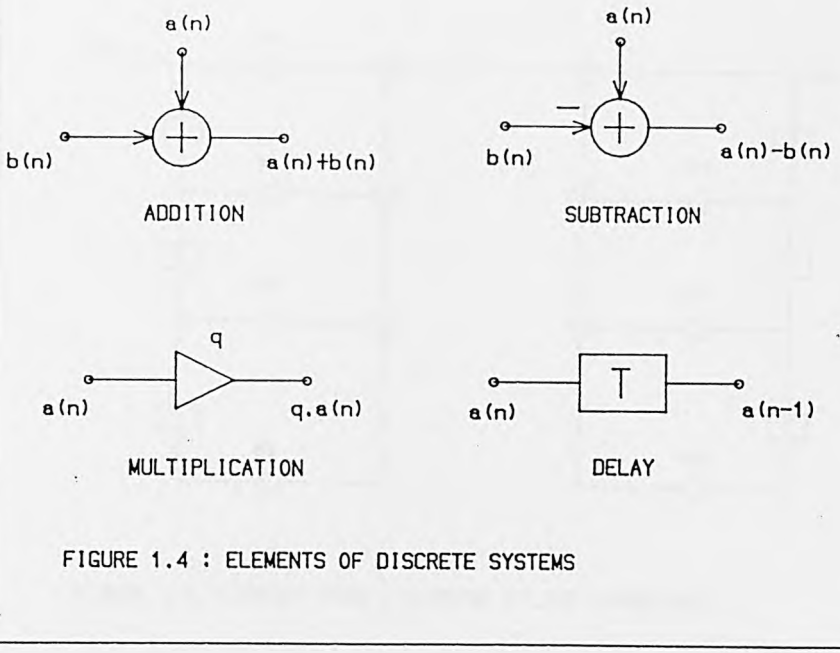


FIGURE 1.4 : ELEMENTS OF DISCRETE SYSTEMS

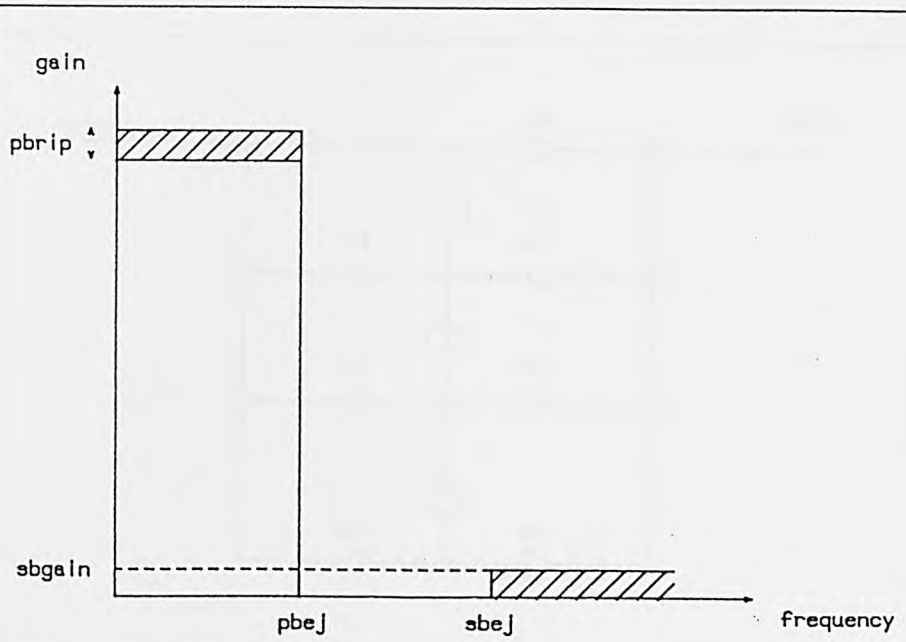


FIGURE 1.5 : TOLERANCE SCHEME

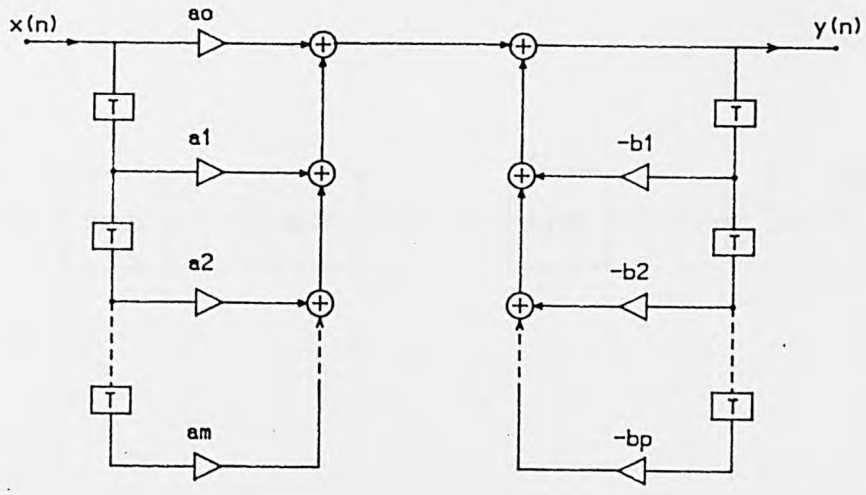


FIGURE 1.6 : DIRECT FORM I DIGITAL FILTER STRUCTURE

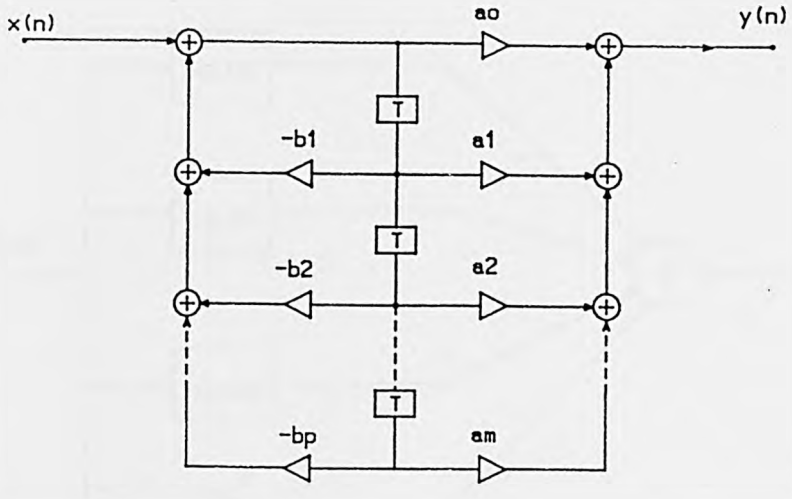


FIGURE 1.7 : DIRECT FORM II

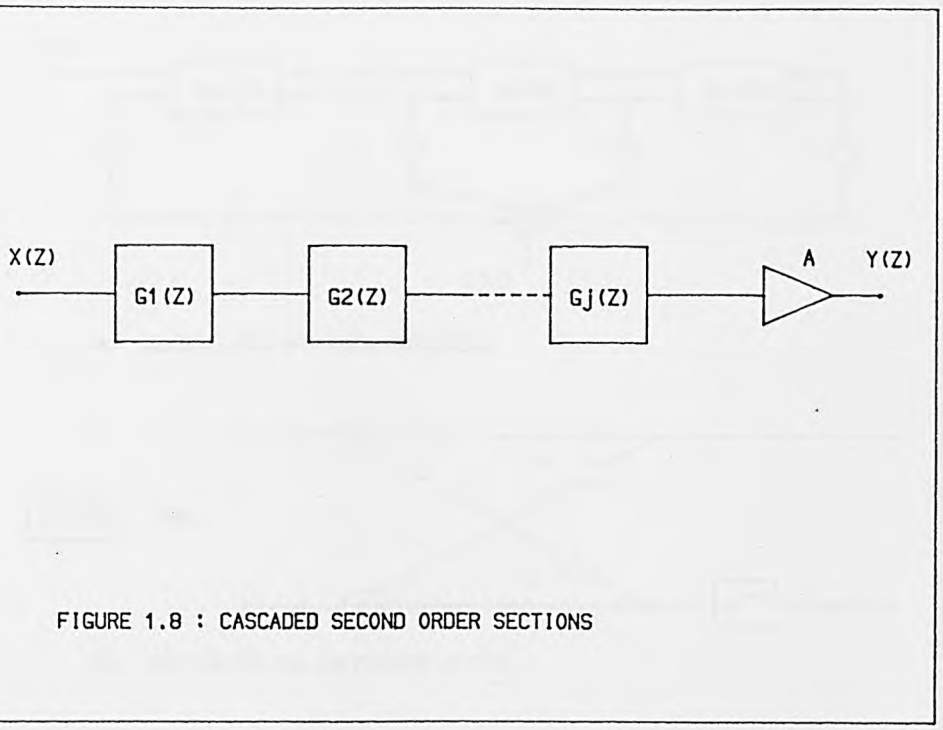


FIGURE 1.8 : CASCADED SECOND ORDER SECTIONS

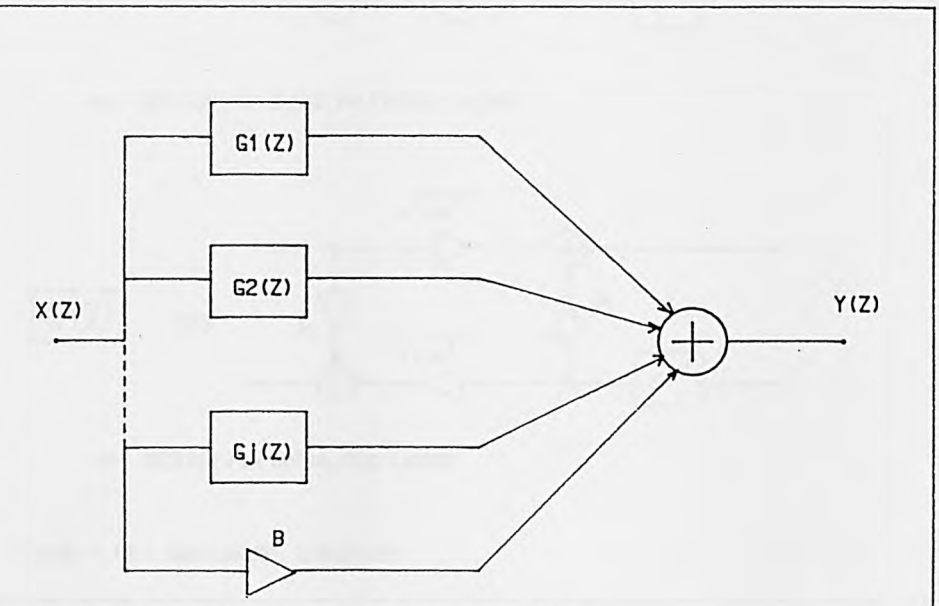
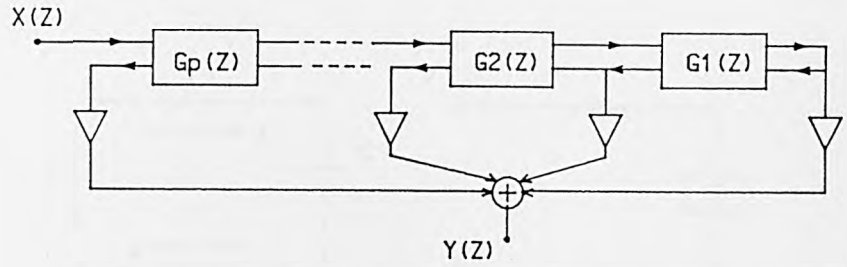
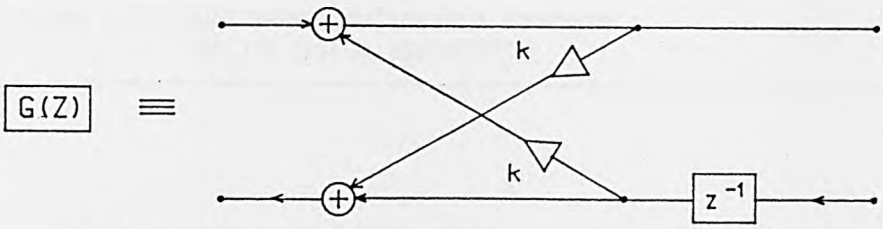


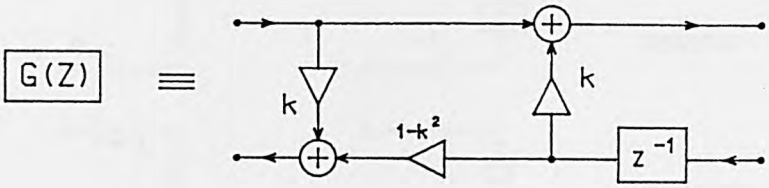
FIGURE 1.9 : PARALLEL SECOND ORDER SECTIONS



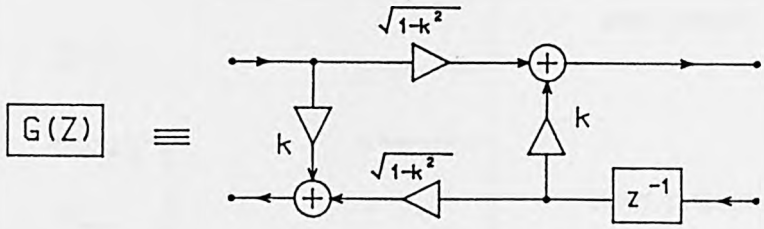
(a) GENERAL FORM OF FILTER STRUCTURE



(b) SECTION FOR TWO MULTIPLIER LATTICE



(c) SECTION FOR THREE MULTIPLIER LADDER



(d) SECTION FOR NORMALIZED LADDER

FIGURE 1.10 : GRAY-MARKEL STRUCTURES

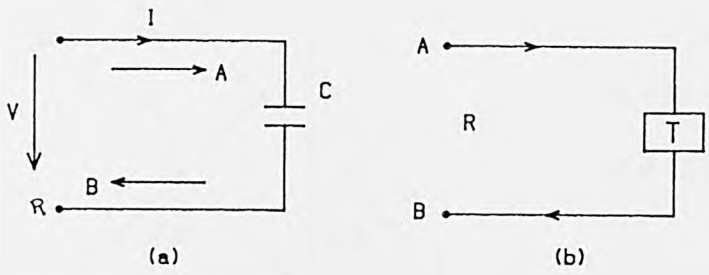


FIGURE 1.11 : WAVE REPRESENTATION OF A CAPACITOR AND ITS DIGITAL EQUIVALENT

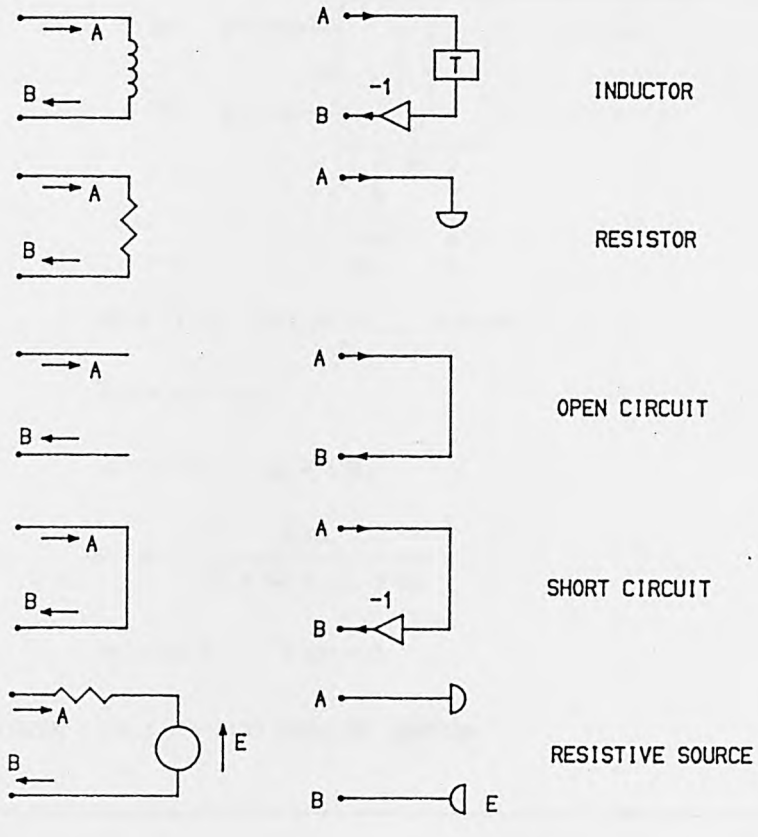
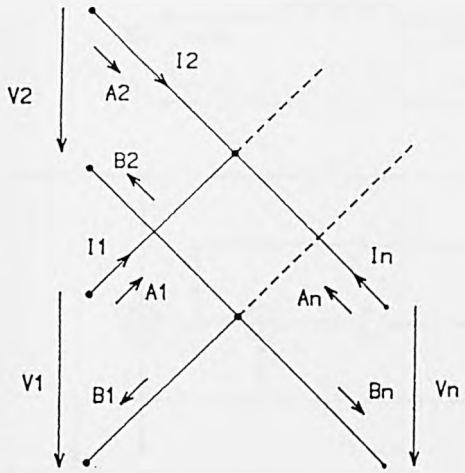
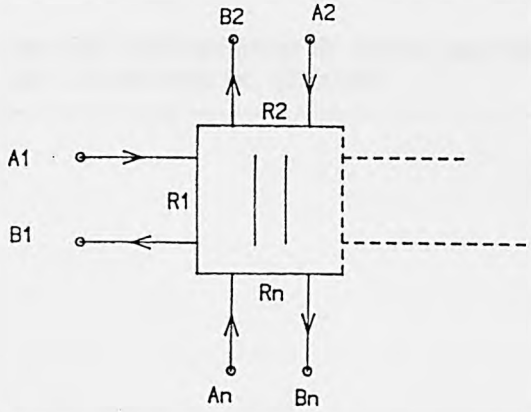


FIGURE 1.12 : WAVE DIGITAL REPRESENTATIONS OF SOME ANALOGUE ELEMENTS



$V1 = V2 = ... = Vn$

$I1 + I2 + ... = 0$



$$A0 = \gamma 1.A1 + \gamma 2.A2 + ... + \gamma n.An$$

$$B.x = a0 - ax$$

$$x = 1 \text{ to } n \text{ , } Gx = 1/Rx$$

$$\gamma x = \frac{2.Gx}{G1 + G2 + ... + Gn}$$

$$\gamma 1 + \gamma 2 + ... + \gamma n = 2$$

FIGURE 1.13 : n - PORT PARALLEL ADAPTOR

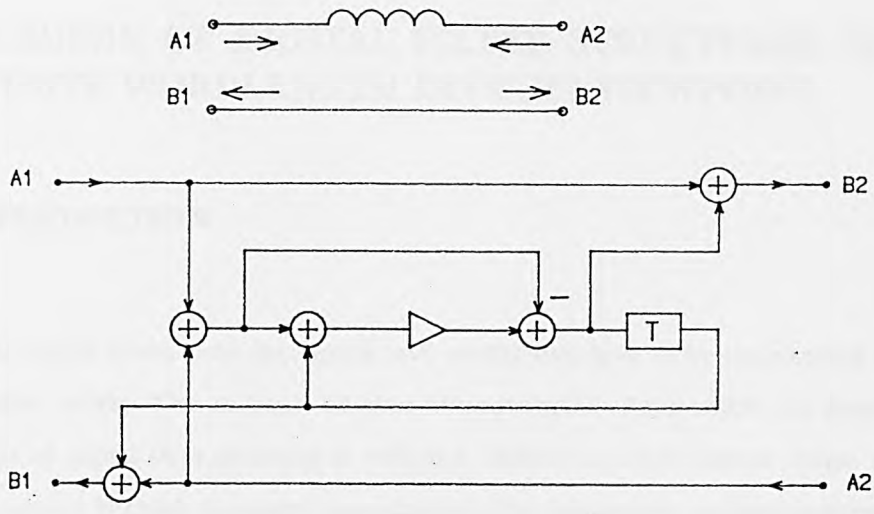


FIGURE 1.14 : TWO PORT REPRESENTATION OF SERIES INDUCTOR AND ITS WAVE DIGITAL EUIVALENT

CHAPTER TWO

COMPARISON OF DIGITAL FILTER STRUCTURES FROM THE FINITE WORDLENGTH EFFECTS VIEWPOINT

2.1 INTRODUCTION

In real digital filters, both the signals and coefficients have to be represented with a finite number of bits. This causes a number of undesirable effects which are dependent on the type of digital filter structure as well as a number of other factors. These effects can be measured in order to express quantitatively the degradation in filter performance. It is known that the degree of degradation is dependent on the wordlengths of the coefficients and signals and these must be determined during the design stage in order to meet given specifications.

For hardware implementation, the coefficient wordlength requirement will influence the complexity of the multipliers and the signal wordlength requirement will dictate the complexity of all the elements in a digital filter. Therefore, it is important to identify those structures which have low sensitivity to variations in coefficient values and exhibit a high signal to noise ratio for a given signal wordlength.

It is the aim of this chapter to study in detail the finite wordlength properties of the structures discussed in chapter 1 and to draw a comparison. The chapter begins with a discussion of coefficient quantization effects and measures for these (section 2.2), followed by a discussion of signal quantization effects and measures for these (section 2.3). Software which has been written for simulating digital filters is outlined in section 2.4 and examples are given to illustrate the main features.

In order to study experimentally some of the finite wordlength properties of the structures, a number of filters based on each type of structure were designed and simulated. Measures of finite wordlength effects were obtained and are detailed in section 2.5. These results are used as a basis for a comparison of the structures and also

manipulated and expressed in terms of number of bits so as to give an indication of the hardware implications.

The stability aspects of the structures and a comparison based on these is given in section 2.6.

2.2 COEFFICIENT QUANTIZATION EFFECTS

In an ideal digital filter, the binary representation of the multiplier coefficients may require an infinite number of bits. In practice, these coefficients have to be approximated with a finite number of bits and this process is generally referred to as coefficient quantization.

In a linear system (i.e infinite signal wordlength), the effect of coefficient quantization is a deviation in the frequency response characteristic from its ideal position. It is known that the degradation in the frequency response characteristic is dependent on the type of digital filter structure, the coefficient wordlength, the type of approximation function used in the design (e.g. Butterworth or Chebyshev), and the tolerance scheme (i.e. the passband, stopband and transition band requirements).

A measure of the degradation due to coefficient quantization may be obtained by measuring, in some way, the deviation of the amplitude response characteristic. One such technique has been proposed by Crochiere [41]. This involves the definition of an error function as follows :

relative error = (hmax–hmin–am)/am for (hmax–hmin)>am

relative error = 0 for (hmax–hmin)<=am

where

hmax = maximum amplitude in the passband (dB)

hmin = minimum amplitude in the passband (dB)

am = originally specified ripple in the passband (dB)

The disadvantage of this method is that only passband behaviour is described.

Other mathematical techniques which describe the error between two functions may also be used but these suffer from the disadvantage of describing overall deviations and not deviations relevant to filter characteristics. With respect to this, a technique is described below which is based on variations in the tolerance scheme.

Filter specifications are generally given in terms of a tolerance scheme (see figure 2.1), i.e. passband ripple in dB, passband edge frequency, maximum stopband gain in dB and stopband edge frequency. Clearly, it would be more useful than the previously described techniques to obtain variations in all these parameters with coefficient wordlength. What is required is a method of extracting the above parameters from a given amplitude response. A method for doing this in software will be described in section 2.4.

The availability of a finite number of bits for the representation of each coefficient means that there is a finite number of values that each coefficient can take. The possibility that a different combination of finite wordlength coefficients may give a better frequency response forms the basis of optimization methods which are generally used at some stage in the finite wordlength design of digital filters [42].

2.3 SIGNAL QUANTIZATION EFFECTS

The arithmetic operations in an ideal digital filter are linear. However, in real digital filters, the arithmetic operations are non-linear due to finite signal wordlength. These non-linearities give rise to several undesirable effects.

2.3.1 QUANTIZATION NOISE

The approximation of an ideal discrete signal (i.e. one which may assume any amplitude value) with a finite number of bits results in an error component which has the statistical properties of white noise for most types of input signal. Consequently, this error is generally called quantization noise and is measured in RMS units and can be expressed as a signal to noise ratio in dB. The signal to noise ratio is a more meaningful expression since it gives a measure of the noise relative to the signal.

In digital filters employing fixed point arithmetic, the sources of quantization error are the analog to digital converter and the multipliers. In systems employing floating point arithmetic, the adders also produce quantization noise.

The signal to noise ratio in a digital filter is dependent on the signal wordlength, the approximation method (i.e. type of roundoff), the type of negative number representation and since a real digital filter is a non-linear system, on the input signal.

There are two types of approximation; truncating or rounding. The errors for an m-bit signal are as follows :

- ROUNDING: the error is in the range $-2^{**}(-m-1)$ to $2^{**}(-m-1)$.

- MAGNITUDE TRUNCATION: if the sign-magnitude negative number representation is used, the error is between 0 and $-2^{**}(-m)$ if the number is positive and between $2^{**}(-m)$ and zero if the number is negative.

- 2's COMPLEMENT TRUNCATION (also known as value truncation): if the 2's complement representation is used, the error is always negative and in the range 0 to $-2^{**}(-m)$.

*(note that the symbol ** denotes 'to the power of').*

Quantization noise can be measured by simulating both the ideal filter (linear)* and the finite signal wordlength filter (non-linear) in the time domain. The same input signal is applied to both filters and the outputs are subtracted to give the noise signal. The RMS values of the input signal and the noise signal are obtained and used to determine the signal to noise ratio. It is important to note that in the above, the initial conditions for both filters must be zero and the coefficients must be quantized to the same values for both filters.

An analytic approach to the determination of quantization noise has been proposed by Jackson [43]. This technique is based on two assumptions. The first assumption is that the errors due to roundoff are uncorrelated from sample to sample and the second assumption is that the errors are uncorrelated from multiplier (or other rounding point) to multiplier. The advantage of assuming uncorrelated errors from one sample to another is that the noise injected into the filter by each rounding point is then 'white', while the advantage of assuming uncorrelated error sources is that the output noise power spectrum may then be computed as simply the superposition of the (filtered) noise spectra due to the separate error sources. The analysis is carried out by replacing each non-ideal multiplier with an ideal multiplier and a noise source (see figure 2.2) and analysing the network in the frequency domain.

A disadvantage of this technique is that the accuracy of the results is dependent on the relevance of the assumptions to a given situation. A second disadvantage is that overflow is not detected with this approach.

The method adopted in this thesis for measuring quantization noise involves time domain simulation as discussed earlier. Overflow detection is included and since the signal to noise ratio is dependent on the type of input signal, a variety of input signals are employed. This is discussed more fully in section 2.4.

* *Note that in the computer simulations described in this thesis, an ideal filter is approximated by using double precision arithmetic.*

2.3.2 OVERFLOW

The addition of two numbers may result in a number which is larger than that which can be represented with the available number of bits. This condition is known as overflow.

Overflow can be avoided by employing scaling such that no signal can ever become larger in magnitude than the largest possible number that can be represented with the available number of bits. However, this type of scaling will lead to an impractically large number of bits for the signal in view of the desired dynamic range. This means that in most practical filters, overflow may occur [44].

If an overflow occurs, a number of different measures can be taken;

- SATURATION: the word that causes the overflow is replaced by a word having the same sign but a magnitude corresponding to the overflow level.

- ZEROING: the word that causes the overflow is replaced with zero.

- 2's COMPLEMENT: discarding the bits that cause the overflow has special advantages with 2's complement arithmetic, since overflows in intermediate results do not cause overflows as long as the final result does not overflow [9].

The important point to note is that in a practical digital filter, overflow is to be expected. It is necessary, therefore, to know the behaviour of the filter should an overflow occur. A practical filter should satisfy the forced response stability condition. Forced–response stability implies that the effects due to overflow die out once the input signal has decreased to a level for which, under linear conditions, the signal values inside of the circuit would at no point exceed the available range [45].

2.3.3 LIMIT CYCLE OSCILLATIONS

Limit cycles are oscillations of fixed amplitude and period that occur in non–linear systems. The occurrence of limit cycles in non–linear systems makes it necessary to define

instability in terms of acceptable magnitudes of oscillation, since a very small non-linear oscillation may not be detrimental to the performance of a system [46].

Two types of limit cycles are known in digital filter; small scale limit cycles (also known as granularity) and large scale limit cycles (also known as overflow oscillations). Small scale limit cycles are generated by quantization error sequences that are highly correlated. Such error sequences arise when the input to a digital filter is zero, constant or other type of signal with a very small amplitude [47].

Small scale limit cycles may be acceptable in some applications but it is necessary to be able to determine bounds on the magnitudes of the limit cycles and how many bits are required for the signal wordlength in order to keep the magnitude of the oscillations within tolerable limits [48]. It is worth noting that limit cycles are more disturbing than quantization noise, and thus in some applications suppressing limit cycles may be more important than reducing quantization noise.

Large scale limit cycles occur in digital filters implemented with 2's complement or 1's complement arithmetic and are due to overflow of adders rather than arithmetic roundoff (small scale limit cycles are due to arithmetic roundoff) [49]. Large scale limit cycles are not acceptable in practice and it is therefore important to eliminate the possibility that they can occur. This can be done by employing adders which saturate when an overflow condition is present.

2.4 SIMULATION SOFTWARE

In this section, a program for the simulation of digital filters will be described. The program is called DIFSIP; an acronym for DIgital Filter SImulation Program. The central time domain simulation algorithm for DIFSIP is taken from the Purdue University digital network analysis program, DINAP [50,51].

DIFSIP is written in fortran 77 and the code is shown in appendix A1.

2.4.1 PROGRAM OUTLINE

DIFSIP can simulate a digital filter so that we can obtain the following information at any node :

- frequency response with infinite precision or finite precision coefficients.

- transient response to different stimuli for infinite precision signals and coefficients, infinite precision signals and finite precision coefficients, and finite precision signals and coefficients.

- the noise signal – i.e. the difference between the finite signal wordlength response and the infinite signal wordlength response to the same input signal – and the signal to noise ratio in dB.

DIFSIP can simulate multipliers and analog to digital converters employing different types of rounding; i.e. rounding, magnitude truncation, 2's complement truncation. DIFSIP can also simulate adders employing different types of overflow constraints; i.e. saturation, zeroing , 2's complement.

DIFSIP can be used to obtain the real (finite signal wordlength) and ideal (infinite signal wordlength) responses of a digital filter to different stimuli. The types of input signal available are ; IMPULSE, RAMP, STEP, SINUSOID, UNIFORMLY DISTRIBUTED RANDOM, NON–NEGATIVE UNIFORMLY DISTRIBUTED RANDOM, GAUSSIAN

DISTRIBUTED RANDOM. The non- deterministic si
subroutines. The real or ideal signals at any node ir
graphic display and a zoom facility is provided.

DIFSIP can be used to perform transient analysis
This is done using the auto-transient analysis option w
types of input signal, range and increment of signal wor
of signal amplitude. Transient analysis is then carried c
results are written to an output file.

All nodes at which overflow occurs during a simulation ı
and the results file.

DIFSIP can be used to obtain the frequency response at any nc
way this is done is to first obtain the impulse response and then perforn
a NAG subroutine. This method of obtaining the frequency response is ι
analysis in the frequency domain because the latter involves dealing with ma.

It has been pointed out in section 2.2 that a useful measure of coefficient sensiı
could be obtained in terms of variations in the tolerance scheme parameters. To this end, a subroutine has been designed for DIFSIP (see appendix A1 – subroutine FRSPAN) which extracts the following information from an amplitude response characteristic :

(i) the maximum gain

(ii) the passband ripple (if a ripple is present) and the highest frequencies at which the amplitude response attains values of −0.1 dB, −0.5 dB, −1.0 dB (relative to maximum gain). This information is useful when no ripple is present in the passband.

(iii) the passband edge

(iv) the highest frequencies at which the amplitude response attains values of −20 dB, −40 dB, −60 dB, −80 dB, −100 dB (relative to maximum gain). This information allows us to gain an insight into the transition band and stopband behaviour of a filter.

(v) the maximum peak in the amplitude response after the passband edge.

The above can be carried out automatically for a range of coefficient wordlengths by using the auto-frequency analysis option in DIFSIP.

2.4.2 SIMULATION EXAMPLE

In order to illustrate some of the features of DIFSIP, the simulation of a seventh order cascaded unit elements WDF (see figure 2.4) will be taken as an example.

The network is described to DIFSIP in terms of a netlist whereby each element is specified in terms of its output and input node numbers. A graphic display of the frequency response and transient response may be obtained at any node in the network.

The frequency response of the filter with a coefficient wordlength of 16 bits is shown in figure 2.5a. A zoom into the passband is shown in figure 2.5b.

The real response to a cosine input of frequency 0.01 radians per second (sampling period = 1 second) and an amplitude of 0.1 volts is shown in figure 2.5c. The signal wordlength in this case was 16 bits and the type of approximation employed by the analog to digital converter and the multipliers was rounding. The corresponding noise signal is shown in figure 2.5d.

The variation in the tolerance scheme of the amplitude response of the filter can be studied with the aid of the auto-frequency analysis option in DIFSIP. This was done on the filter example for the range of coefficient wordlength from 8 to 20 bits and some of the results are illustrated graphically in figures 2.6 a, b and c. (Note that 21 on the x-axis represents infinite precision coefficients). Figure 2.6a shows the variation in passband width, figure 2.6b shows the variation in passband ripple and figure 2.6c shows the variation in stopband parameters; all as a function of coefficient wordlength. Note that these three graphs give a good picture of the variation in amplitude response with coefficient wordlength. However, other parameters are available as a result of auto-frequency analysis (residing in the FRN-OUT file) and may be used if further information is required regarding variation in amplitude response.

The range of analysis that may be performed with the auto-transient analysis option has been discussed before. An example is shown in figure 2.6d which shows the variation in signal to noise ratio with signal wordlength for two different amplitudes. The input signal in this case was the cosine discussed before.

2.5 COMPARISON OF STRUCTURES BASED ON EXPERIMENTAL RESULTS

In this section, the finite wordlength properties of the structures discussed in chapter 1 will be studied through experiments based on simulations with DIFSIP. The results obtained will be used as a basis for a comparison of the structures.

The method used was as follows: for direct form, cascaded second order sections, parallel second order sections, lattice WDFs, LC–ladder WDFs and Gray–Markel normalized structures, filters satisfying the following tolerance schemes were designed :

passband ripple = 0.5 dB

transition width = 0.05 (normalized)

stopband gain = −60 dB

passband edges = 0.05, 0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4 and passband edge = 0.45 with stopband edge = 0.49.

For the generalized WDF, filters satisfying all the above tolerance schemes except the last one were designed and for the cascaded unit elements WDF, only filters up to passband edge 0.3 (remainder of tolerance scheme parameters as above) were possible.

2.5.1 COEFFICIENT SENSITIVITY MEASUREMENTS

An extended version of DIFSIP was used to obtain coefficient sensitivity measures for the all the filters . For a given filter, this program first evaluates the ideal tolerance scheme parameters i.e. it evaluates the amplitude response of the filter for infinite precision coefficients and from this extracts the passband ripple, passband edge, stopband edge, and maximum stopband gain. It then 'relaxes' the tolerance scheme by a certain percentage and then computes the coefficient wordlength required to satisfy the modified tolerance scheme.

The above program was used to obtain the coefficient wordlengths required for 5 percent and 20 percent variation in tolerance scheme parameters for each filter and the

results are shown in figures 2.7 and 2.8. Note that in figures 2.7 and 2.8, the number 17 on the y–axis represents any number greater than 16. Also note that the Gray–Markel normalized filter with the passband width of 0.05 has a design error which has resulted in a passband ripple of 1.5 dB. It will therefore be ignored in the following discussion.

The following observations are made from figure 2.7 :

- The direct form structure gives the worst overall performance. The cascaded second order sections is better than the direct form and in general, the parallel second order sections exhibit a lower sensitivity than the cascaded second order sections.

- The LC–ladder WDF exhibits the lowest sensitivity of all the structures and, in general, all the WDFs are superior to the direct form, parallel and cascaded second order sections as well as the Gray–Markel normalized structure. In general, the lattice WDF exhibits the highest sensitivity of the WDFs.

- The Gray–Markel normalized structure is superior to the direct form, cascaded and parallel second order sections.

- For all structures, the highest sensitivity is exhibited by the narrowest bandwidth filters.

The results of figure 2.7 can be summarized by noting the spread (i.e. the lowest and highest values) of the coefficient wordlength requirements and the mean values as shown in table 2.1. The direct form is ignored in table 2.1 because its coefficient wordlength requirement is nearly always greater than 16.

Note from table 2.1 that the difference in mean values between the least and most sensitive structure is 5 bits and that the cascaded and parallel second order sections require at least 3 more bits than the rest.

The results in figure 2.8 for a 20 percent variation in tolerance scheme parameters follow the same general pattern discussed earlier with respect to figure 2.7 except that lower values of coefficient wordlength are required as expected. The results are summarized in the same way as before and shown in table 2.2. Note that in comparison with table 2.1, all values are reduced by 2 bits except the parallel second order sections and lattice WDF which are reduced by 3 bits.

2.5.2 SIGNAL TO NOISE RATIO MEASUREMENTS

An extended version of DIFSIP was used to perform the following operations on each filter:

(i) set signal wordlength to 16 for simulation run.

(ii) evaluate coefficient wordlength required for 5 percent variation in tolerance scheme and set coefficient wordlength to this value for simulation run.

(iii) select type of roundoff to be rounding.

(iv) select type of input signal as cosine with a frequency that is half the passband edge frequency.

(v) evaluate the maximum amplitude of the input signal that does not cause overflow at any node in the network over 1000 sampling instants (this is the length of a simulation run). Set the input signal amplitude to this value for the simulation run.

(vi) evaluate the signal to noise ratio and write to output file.

(vii) repeat (iv) to (vi) with the type of roundoff in (ii) selected to be magnitude truncation and 2s complement truncation.

(viii) repeat the whole process with the input signal in (iv) selected to be uniformly distributed random and gaussian distributed random.

The results are shown graphically for each type of input signal. For each structure, the signal to noise ratio is shown against the passband width of each filter for the three

different types of roundoff – figures 2.9, 2.10 for cosine and figures 2.15, 2.16 for gaussian distributed random inputs. For each type of roundoff, the signal to noise ratio is shown against the passband width for all the structures – figures 2.11, 2.12, 2.13 for cosine and figures 2.17, 2.18, 2.19 for gaussian distributed random inputs. Note that signal to noise ratio values below 20 dB are not shown in the graphs.

The maximum amplitude that does not cause overflow is shown against the passband width of each filter for all the structures. Since the results obtained showed that the maximum amplitude without overflow for each type of roundoff was the same for a particular input signal, only one graph is shown for each input signal – figure 2.14 for cosine and figure 2.20 for gaussian distributed random inputs.Note that the amplitude is plotted on a log scale.

The RMS value of the noise signal is shown against the passband for each filter for all the structures. This is only shown for the gaussian distributed random input signal – figures 2.21, 2.22, 2.23.

The results for the uniformly distributed random input are included in the appendix A2 and not shown here for reasons which will be discussed later.

It can be seen from figures 2.9 and 2.10 that for all the structures except the direct form, rounding is superior to magnitude truncation and magnitude truncation is superior to 2s complement truncation. In the direct form, the difference between the three types of roundoff is insignificant in most cases. The difference in signal to noise ratios between the three types of roundoff varies from structure to structure and from filter to filter. All the structures give the worst signal to noise ratio for the narrowest bandwidth filter (passband width = 0.05). For most of the structures, the signal to noise ratio increases as the bandwidth increases except for lattice WDFs, parallel second order sections and the direct form for which there is some deterioration for higher bandwidths. It can be seen from figures 2.15 and 2.16 that a similar pattern results for the gaussian distributed random input.

It can be seen from figures 2.11 to 2.13 that for the cosine input, the direct form gives the worst overall performance for all types of roundoff followed by the cascaded

second order sections. The Gray–Markel normalized structure gives the best performance for rounding and the cascaded unit elements WDF gives the best overall performance for truncation although it is bettered by the Gray–Markel normalized structure at lower bandwidths for 2s complement truncation. Of the WDFs, the generalized WDF gives the worst performance and the cascaded unit elements WDF gives the best performance for all types of roundoff.

For the gaussian distributed random input (see figures 2.17 to 2.19), the observations with regard to the direct form and cascaded second order sections are the same as before. The Gray–Markel normalized structure is the best overall for all types of roundoff followed by the cascaded unit elements WDF which gives the best performance of the WDFs. The worst WDF for rounding is the generalized WDF and for truncation it is the lattice WDF.

Note from figure 2.14 that for the cosine input, the direct form and the cascaded second order sections allow the lowest amplitude for the input signal and the Gray–Markel normalized structure allows the highest amplitude. Of the WDFs, the LC–ladder allows the highest amplitude and the lattice WDF allows the lowest amplitude. This pattern is also observed from figure 2.20 for the gaussian distributed random inputs except that of the WDFs, the generalized WDF allows the highest amplitude.

All the results obtained from the signal to noise ratio measurements are summarized in the following manner; for each structure and each type of roundoff, the highest, lowest and mean values of signal to noise ratio are presented in tabular form – table 2.3 for cosine input, table 2.4 for gaussian distributed random input and table 2.5 for uniformly distributed random input.

From tables 2.3, 2.4, 2.5 it can be seen that the signal to noise ratios corresponding to cosine inputs are higher than those corresponding to the gaussian and uniformly distributed random inputs. The signal to noise ratios corresponding to the two random inputs are similar. The RMS noise values for the three input input signals are given in appendix A3. Note that the values for the two random signals are similar and that there is broad similarity between these and the values corresponding to the cosine input. Since the signal to noise ratios are similar for the random signals, it follows that the RMS values of

the output signals will be the same. We may conclude from this that the overflow level for random inputs is dependent on the RMS value of the output signal.

It has been shown that in practice the filters exhibit similar behaviour for gaussian distributed random and uniformly distributed random inputs. Therefore, we only need to consider one of these signals. The gaussian distributed random input will be considered from hereon. From table 2.4 it can be seen that for all types of roundoff, the Gray–Markel normalized structure exhibits the highest signal to noise ratio followed by the cascaded unit elements WDF, and that the worst performances are given by the direct form followed by the cascaded second order sections. For rounding the remaining order is; LC–ladder WDF, lattice WDF, parallel second order sections, generalized WDF, and for magnitude truncation it is; lattice WDF, parallel second order sections, generalized WDF, LC– ladder, and for 2s complement truncation it is; generalized WDF, LC–ladder, parallel second order sections , lattice WDF.

For all the filters, the relationship between input signal amplitude, number of signal bits and the signal to noise ratio were examined using DIFSIP and the results obtained generally followed the pattern shown in figure 2.6b, i.e. in general, the signal to noise ratio increased by 6dB for a single bit increase in signal wordlength. It was also noted that a decrease of amplitude by a factor of 10 caused a fall in the signal to noise ratio of approximately 20dB. Furthermore, it was noted that an increase in the signal wordlength did not affect the minimum input signal amplitude that causes overflow.

From the previous discussion, it may be concluded that for random inputs, the quantization noise generated in the filters under consideration is independent of the type and amplitude of the input signal. The quantization noise is dependent on the type of roundoff and on the signal wordlength on which it has a dependence of 6 dB per signal bit.

It is clear that the best structure in terms of signal to noise ratio is the Gray–Markel normalized structure. It has also been shown that the dependence of signal to noise ratio upon signal wordlength is 6 dB per signal bit. We may therefore express the difference between the signal to noise ratios for the Gray–Markel normalized structure and those for the remaining structures (see table 2.4) in terms of signal bit equivalents. This has been

done for all the values in table 2.4 and the results are given in table 2.6. This table shows the number of additional signal bits that each structure would require in order to achieve a similar signal to noise ratio to the Gray–Markel normalized structure. The important point to note is that a performance equivalent to that of the Gray–Markel normalized structure can be obtained with the other structures by increasing the signal wordlength.

2.6 COMPARISON OF STRUCTURES BASED ON STABILITY ASPECTS

In this section, the stability aspects of the structures will be considered and comparisons will be drawn.

Digital filters may exhibit limit cycles under certain conditions. In applications where limit cycles are acceptable, it is necessary to be able to determine the maximum possible amplitude of the limit cycles during the design stage. Several papers on methods of determining amplitude bounds for direct, cascaded second order sections and parallel second order sections are available including [47,48,53–55]. It is generally accepted however, that these bounds are too pessimistic to be of much practical use.

One of the most important advantages of the WDFs based on the techniques of Fettweis [45] is that requirements for completely avoiding limit cycles have been given. Of the structures under consideration, the cascaded unit elements WDF, lattice WDF and the LC–ladder WDF fall into this category. Obviously, in applications where limit cycles are not acceptable, these WDFs with appropriate modifications as defined by Fettweis [45] can be used.

In all digital filters, overflow oscillations may be avoided by employing overflow constraints with a saturation characteristic.

It was mentioned in a previous section that forced–response stability is an important consideration in practice. While for direct, cascaded second order sections and parallel second order sections, it is not known how to ensure forced–response stability, the problem is completely solved for the WDFs of Fettweis [45].

Stability aspects of the Gray–Markel normalized structure and the generalized WDF have not been reported in the literature. However, it is worth pointing out that the Gray–Markel normalized structure is derived from the direct form whereas the WDFs are based on passive analog filters. It is known that this is an important factor since the passivity properties may be preserved provided certain restrictions are observed [45]. Therefore, it is unlikely that the Gray–Markel structure will possess better stability properties than the WDFs.

In summary, it is noted that the cascaded unit elements WDF, lattice WDF and LC–ladder WDFs possess better stability properties than the direct, cascaded second order sections and the parallel second order sections.

2.7 DISCUSSION AND CONCLUSIONS

The aim in this chapter was to draw a comparison of the structures under consideration on the basis of finite wordlength properties. To this end, a range of filters were designed for each structure and coefficient wordlength sensitivity and quantization noise measurements were made.

The results obtained show that the performance of a particular structure relative to another is not always the same under different conditions (i.e. different passband widths, type of roundoff etc). This leads us to the conclusion that the optimal choice of structure is dependent on the given specification. However, when viewed as a whole, it is clear from the results that some structures are superior to others. It has been shown that quantitative summaries of the results support the observations that have been made regarding the relative performance of the structures.

In terms of coefficient sensitivity, it has been shown that the order of the structures from least sensitive to most sensitive is as follows; LC–ladder WDF, cascaded unit elements WDF, generalized WDF, lattice WDF, Gray–Markel normalized structure, parallel second order sections, cascaded second order sections,direct form.

The results obtained for quantization noise for cosine, uniformly distributed random and gaussian distributed random inputs show that quantization noise is independent of the input signal type and amplitude. The quantization noise is dependent on the type of roundoff employed and it has been shown that rounding is superior to magnitude truncation and magnitude truncation is superior to 2s complement truncation. The signal to noise ratios for the two random inputs were shown to be similar but lower than those for the cosine input. It has also been shown that the signal to noise ratio has a dependence of 6 dB per bit on the signal wordlength. In conclusion, it is noted that the observations regarding quantization noise are in agreement with those of Jackson [43].

It has been shown that the maximum amplitude of the input signal that does not cause overflow is independent of the type of roundoff and signal wordlength for all the filters. In the case of random signals, it has been shown that the RMS value of the input signal determines overflow. It is important to note that the various filters possess different

overflow levels and it is the overflow level which limits the maximum signal to noise ratio that is possible for the given set of conditions. In conclusion, it is noted that structures which generate a low level of quantization noise will not necessarily give the best signal to noise ratio.

In terms of signal to noise ratios, it has been shown that the order of structures from best to worst for the three types of roundoff is :

- ROUNDING: Gray–Markel normalized, cascaded unit elements WDF, LC–ladder WDF, lattice WDF, parallel second order sections, generalized WDF, cascaded second order sections, direct form.

- MAGNITUDE TRUNCATION: Gray–Markel normalized, cascaded unit elements WDF, lattice WDF, parallel second order sections, generalized WDF, LC–ladder WDF, cascaded second order sections, direct form.

- 2's COMPLEMENT TRUNCATION: Gray–Markel normalized, cascaded unit elements WDF, generalized WDF, LC–ladder WDF, parallel second order sections, lattice WDF, cascaded second order sections, direct form.

The stability aspects of the various structures have been examined on the basis of a literature survey and it is noted in conclusion that of the structures under consideration, the LC–ladder WDF, lattice WDF and the cascaded unit elements WDF are the best structures from this point of view.

In most practical applications, stability will be a requirement and since overflow is likely to occur in a practical digital filter, forced–response stability will also be required. The conditions to ensure stability for the cascaded unit elements WDF, LC–ladder WDF, and the lattice WDF are known due to Fettweis [45]. This is a major factor which favours the above WDFs in practical applications.

Finally, note that the structures which require the lowest signal and coefficient wordlengths will not necessarily give the most efficient hardware implementations since

this is also dependent on the hardware complexity of the structure. This aspect is explored in chapter 3.

STRUCTURE	SPREAD (No. OF BITS)	MEAN (No. OF BITS)
CASCADED SECOND ORDER SECTIONS	13 – 17	15
PARALLEL SECOND ORDER SECTIONS	13 – 17	15
GRAY–MARKEL NORMALIZED LADDER	11 – 13	12
CASCADED UNIT ELEMENTS WDF	9 – 13	11
LATTICE WDF	10 – 14	12
LC LADDER WDF	7 – 11	10
GENERALIZED WDF	9 – 13	11

TABLE 2.1 : SUMMARY OF COEFFICIENT WORDLENGTH REQUIREMENTS FOR 5 PERCENT VARIATION IN TOLERANCE SCHEME PARAMETERS

STRUCTURE	SPREAD (No. OF BITS)	MEAN (No. OF BITS)
CASCADED SECOND ORDER SECTIONS	10 – 17	13
PARALLEL SECOND ORDER SECTIONS	10 – 14	12
GRAY–MARKEL NORMALIZED LADDER	9 – 11	10
CASCADED UNIT ELEMENTS WDF	8 – 11	9
LATTICE WDF	8 – 11	9
LC LADDER WDF	6 – 11	8
GENERALIZED WDF	6 – 12	9

TABLE 2.2 : SUMMARY OF COEFFICIENT WORDLENGTH REQUIREMENTS FOR 20 PERCENT VARIATION IN TOLERANCE SCHEME PARAMETERS

STRUCTURE	TYPE OF ROUNDOFF	LOWER (DB)	UPPER (DB)	MEAN (DB)
DIRECT FORM II	ROUND	0.	51.15	23.68
	MAG	0.	51.43	22.54
	2's COMP	0.	49.94	22.51
CASCADED SECOND ORDER SECTIONS	ROUND	0.	83.75	31.12
	MAG	0.	77.10	25.39
	2's COMP	0.	71.96	22.68
PARALLEL SECOND ORDER SECTIONS	ROUND	48.96	79.70	69.04
	MAG	45.42	76.72	64.31
	2's COMP	34.19	68.52	54.74
GRAY-MARKEL NORMALIZED LADDER	ROUND	75.93	85.84	79.52
	MAG	55.25	75.64	63.74
	2's COMP	51.59	73.20	59.25
CASCADED UNIT ELEMENTS WDF	ROUND	65.28	81.10	74.51
	MAG	51.01	75.01	63.80
	2's COMP	45.39	65.52	56.63
LATTICE WDF	ROUND	62.37	82.48	75.40
	MAG	46.67	78.19	67.36
	2's COMP	36.19	68.76	57.57
LC LADDER WDF	ROUND	63.41	86.43	76.56
	MAG	49.27	75.82	64.62
	2's COMP	39.55	71.03	56.72
GENERALIZED WDF	ROUND	49.30	77.93	65.92
	MAG	41.89	73.87	60.21
	2's COMP	39.08	69.39	54.67

ROUND : ROUNDING MAG : MAGNITUDE TRUNCATION
2's COMP : 2's COMPLEMENT TRUNCATION

TABLE 2.3 : SUMMARY OF SIGNAL TO NOISE RATIO VALUES FOR COSINE INPUT

STRUCTURE	TYPE OF ROUNDOFF	LOWER (DB)	UPPER (DB)	MEAN (DB)
DIRECT FORM II	ROUND	0.	44.57	18.42
	MAG	0.	43.69	17.32
	2's COMP	0.	43.80	18.34
CASCADED SECOND ORDER SECTIONS	ROUND	0.	68.00	24.18
	MAG	0.	63.69	21.20
	2's COMP	0.	56.03	17.52
PARALLEL SECOND ORDER SECTIONS	ROUND	43.93	67.03	57.83
	MAG	39.02	60.14	51.71
	2's COMP	28.91	51.05	43.58
GRAY–MARKEL NORMALIZED LADDER	ROUND	70.11	78.30	72.81
	MAG	50.35	64.37	55.42
	2's COMP	46.45	66.00	52.49
CASCADED UNIT ELEMENTS WDF	ROUND	57.53	74.00	66.57
	MAG	41.85	63.12	52.31
	2's COMP	38.56	58.56	49.23
LATTICE WDF	ROUND	53.34	61.76	58.53
	MAG	42.26	57.04	51.22
	2's COMP	27.52	46.40	40.81
LC LADDER WDF	ROUND	55.54	71.93	63.10
	MAG	39.68	59.66	48.11
	2's COMP	32.17	56.72	43.59
GENERALIZED WDF	ROUND	44.64	68.45	57.51
	MAG	35.37	66.20	50.28
	2's COMP	32.96	59.85	45.87

ROUND : ROUNDING MAG : MAGNITUDE TRUNCATION
2's COMP : 2's COMPLEMENT TRUNCATION

TABLE 2.4 : SUMMARY OF SIGNAL TO NOISE RATIO VALUES FOR GAUSSIAN DISTRIBUTED RANDOM INPUT

STRUCTURE	TYPE OF ROUNDOFF	LOWER (DB)	UPPER (DB)	MEAN (DB)
DIRECT FORM II	ROUND	0.	43.35	19.24
	MAG	0.	44.22	17.77
	2's COMP	0.	43.17	19.53
CASCADED SECOND ORDER SECTIONS	ROUND	0.	66.98	24.94
	MAG	0.	62.77	21.36
	2's COMP	0.	55.05	18.10
PARALLEL SECOND ORDER SECTIONS	ROUND	43.19	66.09	57.49
	MAG	39.17	59.22	51.54
	2's COMP	28.47	50.95	43.22
GRAY-MARKEL NORMALIZED LADDER	ROUND	69.52	78.55	72.98
	MAG	49.86	64.36	55.74
	2's COMP	46.18	65.96	52.86
CASCADED UNIT ELEMENTS WDF	ROUND	57.27	73.60	67.13
	MAG	41.11	63.07	52.27
	2's COMP	37.78	58.61	49.55
LATTICE WDF	ROUND	53.26	62.93	58.36
	MAG	42.23	57.73	51.21
	2's COMP	26.97	45.72	40.47
LC LADDER WDF	ROUND	54.42	72.15	63.10
	MAG	37.85	60.00	47.84
	2's COMP	30.38	57.33	43.57
GENERALIZED WDF	ROUND	44.58	68.01	58.19
	MAG	34.88	65.74	50.63
	2's COMP	32.04	59.49	46.31

ROUND : ROUNDING MAG : MAGNITUDE TRUNCATION
2's COMP : 2's COMPLEMENT TRUNCATION

TABLE 2.5 : SUMMARY OF SIGNAL TO NOISE RATIO VALUES FOR UNIFORMLY DISTRIBUTED RANDOM INPUT

STRUCTURE	TYPE OF ROUNDOFF	LOWER (No. OF BITS)	UPPER (No. OF BITS)	MEAN (No. OF BITS)
DIRECT FORM II	ROUND	–	6	9
	MAG	–	4	7
	2's COMP	–	4	6
CASCADED SECOND ORDER SECTIONS	ROUND	–	2	8
	MAG	–	1	6
	2's COMP	–	2	6
PARALLEL SECOND ORDER SECTIONS	ROUND	5	2	3
	MAG	2	1	1
	2's COMP	3	3	2
GRAY–MARKEL NORMALIZED LADDER	ROUND	0	0	0
	MAG	0	0	0
	2's COMP	0	0	0
CASCADED UNIT ELEMENTS WDF	ROUND	3	1	1
	MAG	2	1	1
	2's COMP	2	2	1
LATTICE WDF	ROUND	3	3	3
	MAG	2	2	1
	2's COMP	4	4	2
LC LADDER WDF	ROUND	3	2	2
	MAG	2	1	2
	2's COMP	3	2	2
GENERALIZED WDF	ROUND	5	2	3
	MAG	3	1	1
	2's COMP	3	2	2

ROUND : ROUNDING MAG : MAGNITUDE TRUNCATION
2's COMP : 2's COMPLEMENT TRUNCATION

TABLE 2.6 : DIFFERENCE IN SIGNAL TO NOISE RATIOS BETWEEN GRAY–MARKEL NORMALIZED LADDER AND THE REST EXPRESSED IN SIGNAL BIT EQUIVALENTS

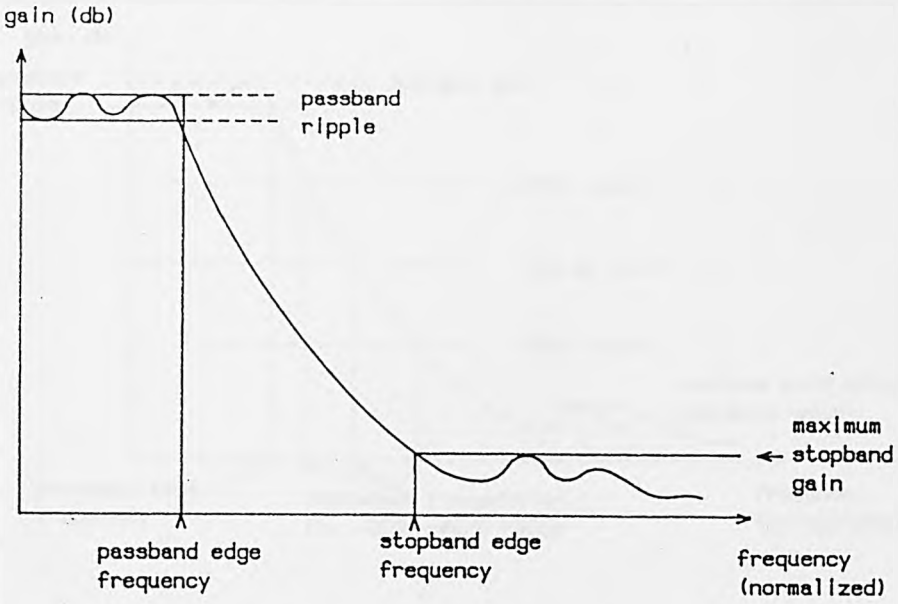


FIGURE 2.1 : ILLUSTRATION OF TOLERANCE SCHEME

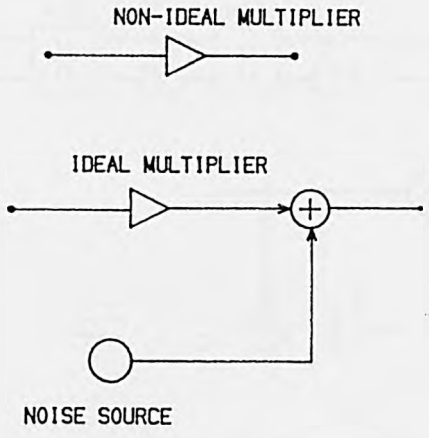


FIGURE 2.2 : REPLACEMENT OF NON-IDEAL MULTIPLIER WITH IDEAL MULTIPLIER AND NOISE SOURCE

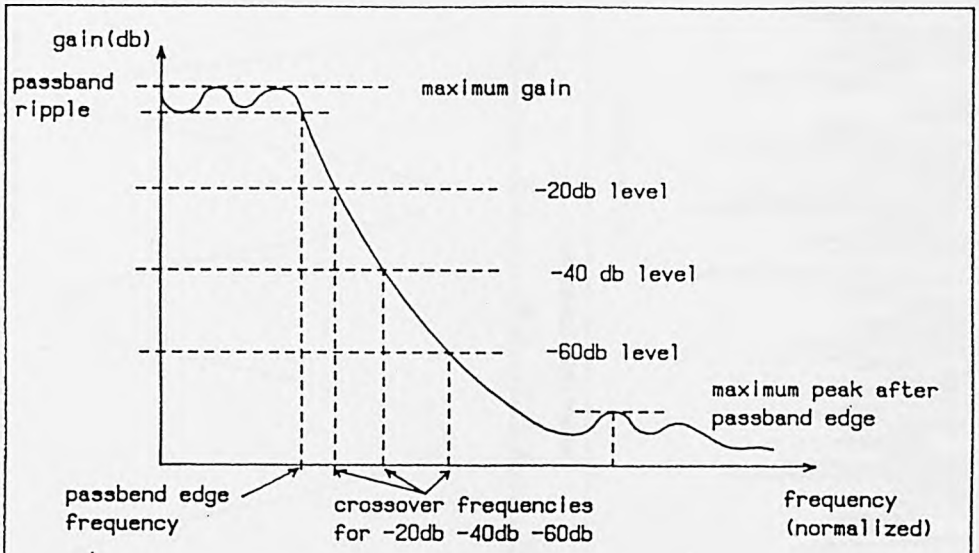


FIGURE 2.3 : EXTRACTION OF TOLERANCE SCHEME PARAMETERS FROM AN AMPLITUDE RESPONSE

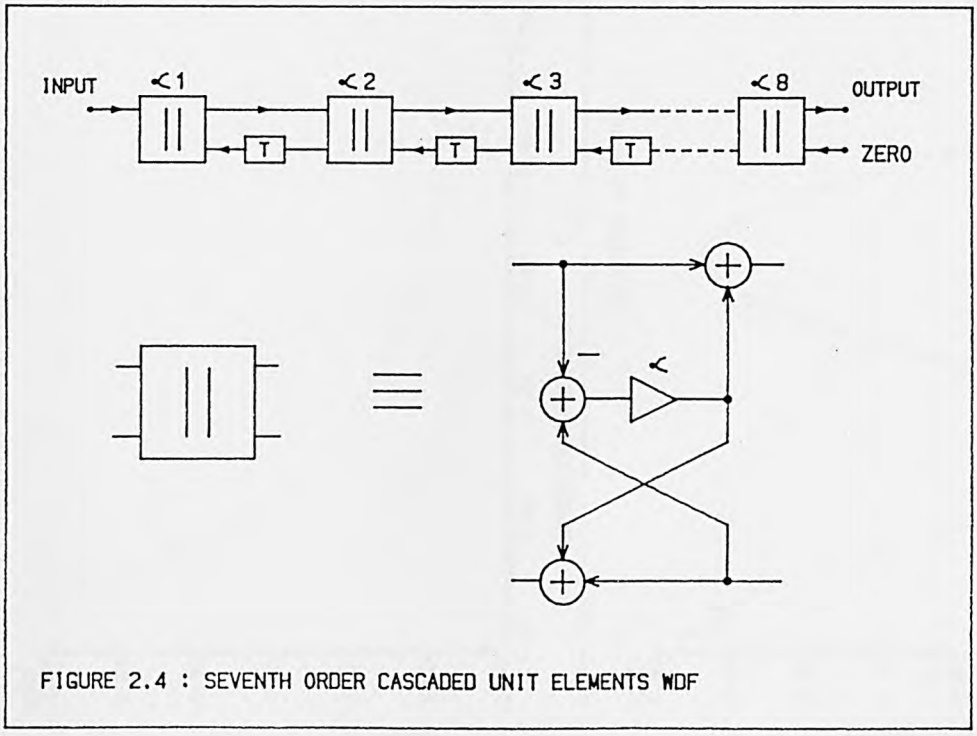
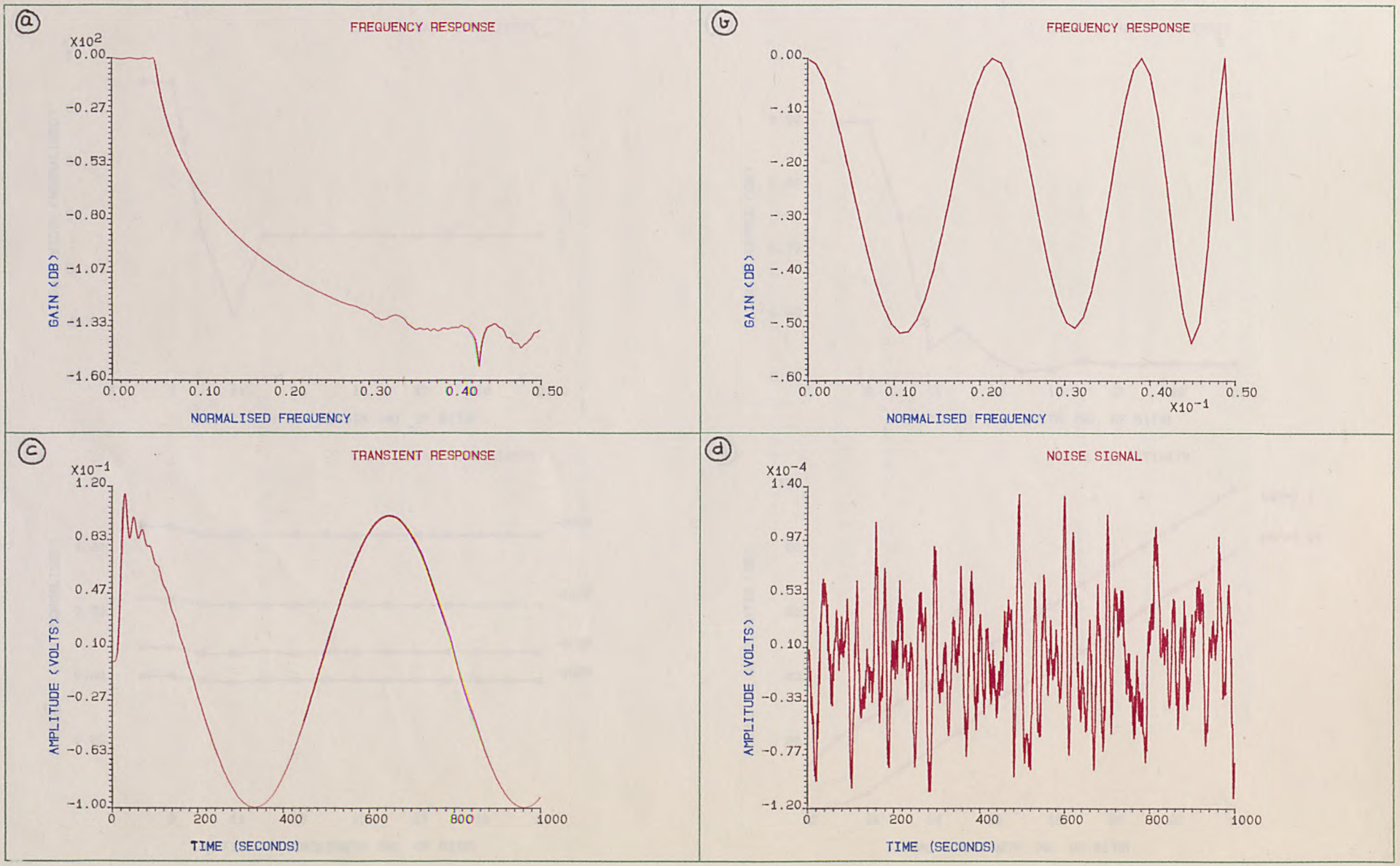


FIGURE 2.4 : SEVENTH ORDER CASCADED UNIT ELEMENTS WDF

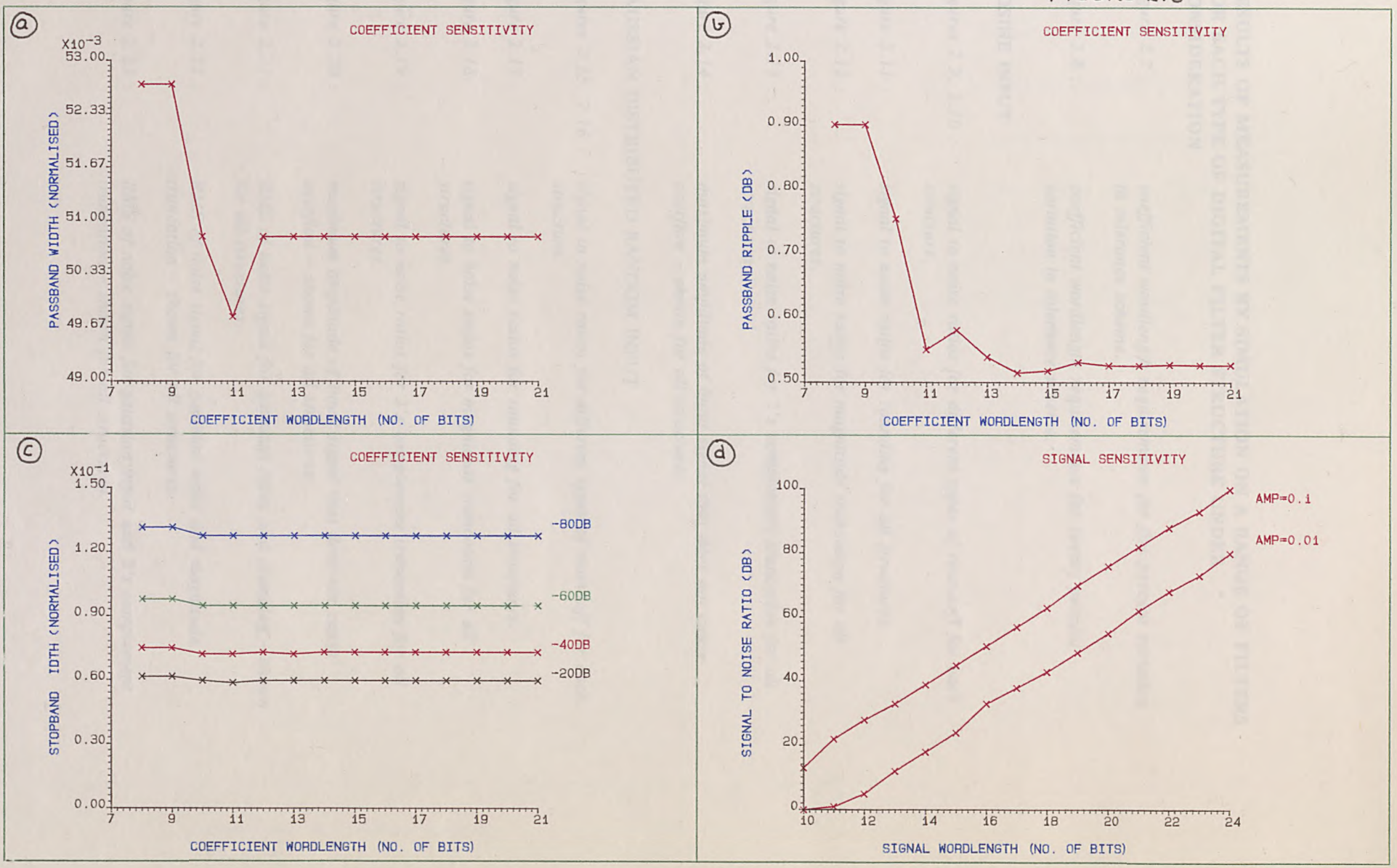
DIFSIP EXAMPLE

FIGURE 2.5



DIFSIP EXAMPLE

FIGURE 2.6



RESULTS OF MEASUREMENTS BY SIMULATION ON A RANGE OF FILTERS FOR EACH TYPE OF DIGITAL FILTER STRUCTURE UNDER CONSIDERATION

Figure 2.7 : *coefficient wordlength requirements for five percent variation in tolerance scheme.*

Figure 2.8 : *coefficient wordlength requirements for twenty percent variation in tolerance scheme.*

COSINE INPUT

Figures 2.9, 2.10 : *signal to noise ratios for different types of roundoff for each structure.*

Figure 2.11 : *signal to noise ratios for rounding for all structures.*

Figure 2.12 : *signal to noise ratios for magnitude truncation for all structures.*

Figure 2.13 : *signal to noise ratios for 2's complement truncation for all structures.*

Figure 2.14 : *maximum amplitude of input signal that does not cause overflow – shown for all structures.*

GAUSSIAN DISTRIBUTED RANDOM INPUT

Figures 2.15, 2.16 : *signal to noise ratios for different types of roundoff for each structure.*

Figure 2.17 : *signal to noise ratios for rounding for all structures.*

Figure 2.18: *signal to noise ratios for magnitude truncation for all structures.*

Figure 2.19 : *signal to noise ratios for 2's complement truncation for all structures.*

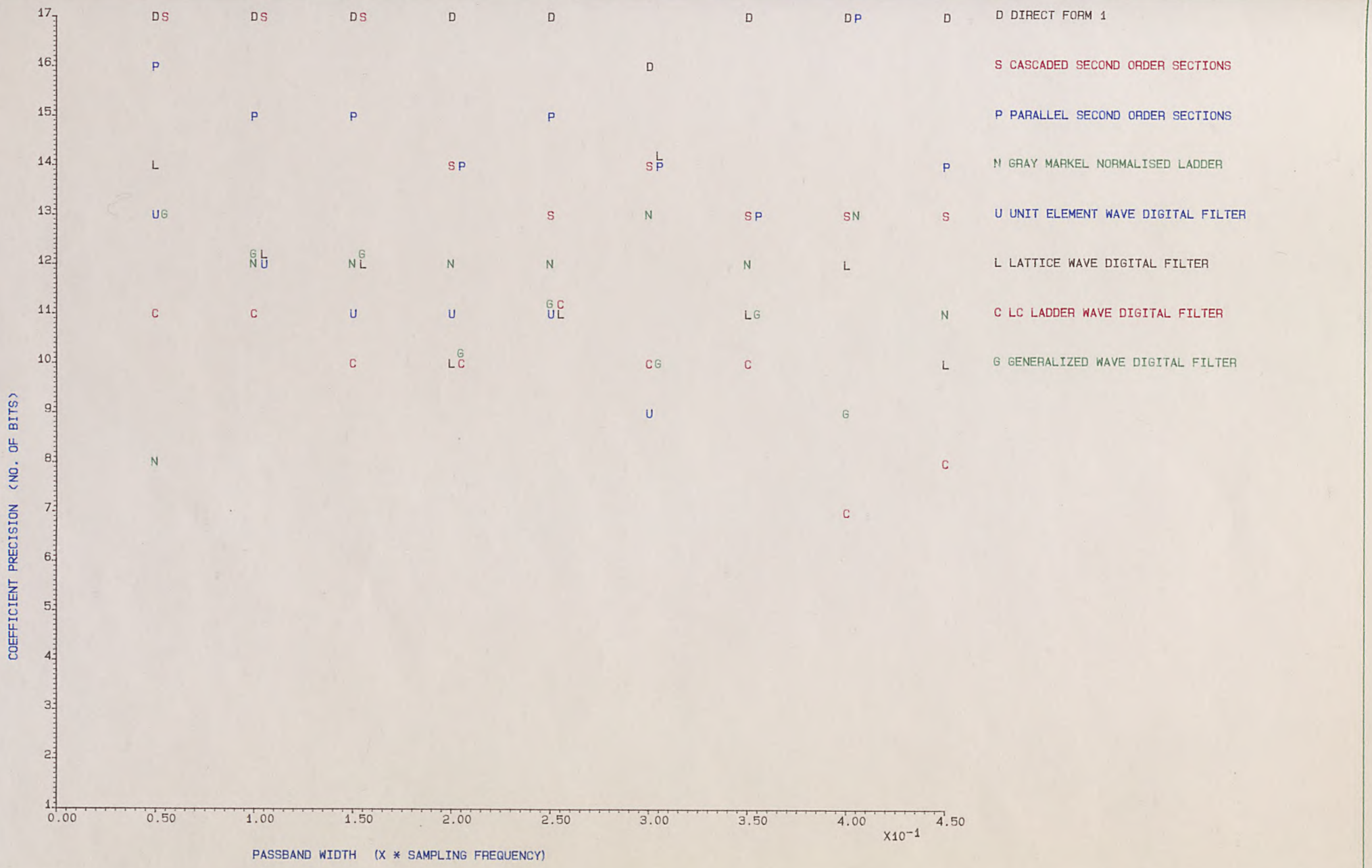
Figure 2.20 : *maximum amplitude of input signal that does not cause overflow – shown for all structures.*

Figure 2.21 : *RMS of noise signal for gaussian input and rounding – shown for all structures.*

Figure 2.22 : *RMS of noise signal for gaussian input and magnitude truncation – shown for all structures.*

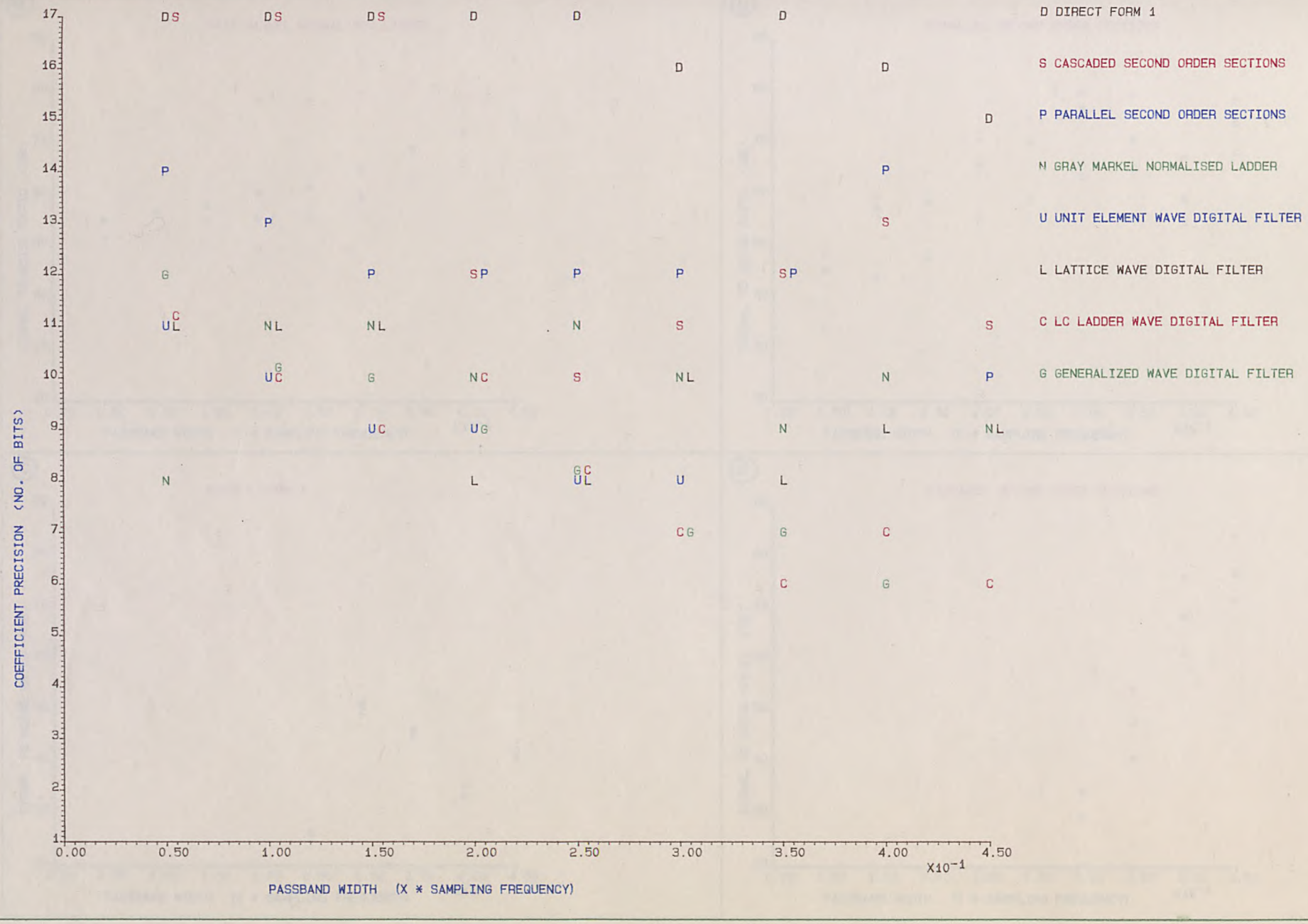
Figure 2.23 : *RMS of noise signal for gaussian input and 2's complement truncation – shown for all structures.*

FIGURE 2.7


COEFFICIENT PRECISION (NO. OF BITS)
PASSBAND WIDTH (X * SAMPLING FREQUENCY)
X10⁻¹
0.00
0.50
1.00
1.50
2.00
2.50
3.00
3.50
4.00
4.50
D DIRECT FORM 1
S CASCADED SECOND ORDER SECTIONS
P PARALLEL SECOND ORDER SECTIONS
N GRAY MARKEL NORMALISED LADDER
U UNIT ELEMENT WAVE DIGITAL FILTER
L LATTICE WAVE DIGITAL FILTER
C LC LADDER WAVE DIGITAL FILTER
G GENERALIZED WAVE DIGITAL FILTER

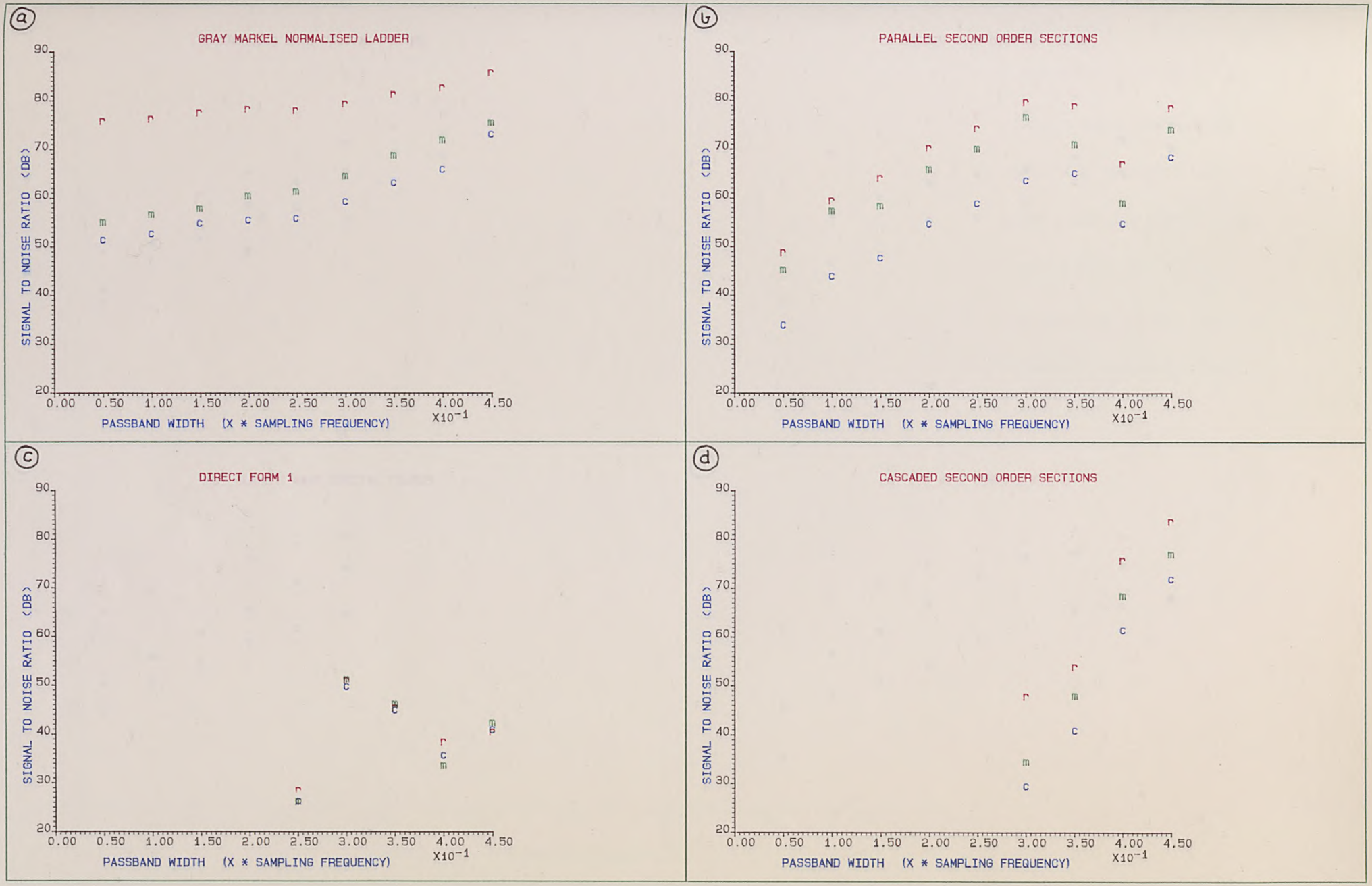
TWENTY PERCENT TOLERANCE SCHEME VARIATION

FIGURE 2.8



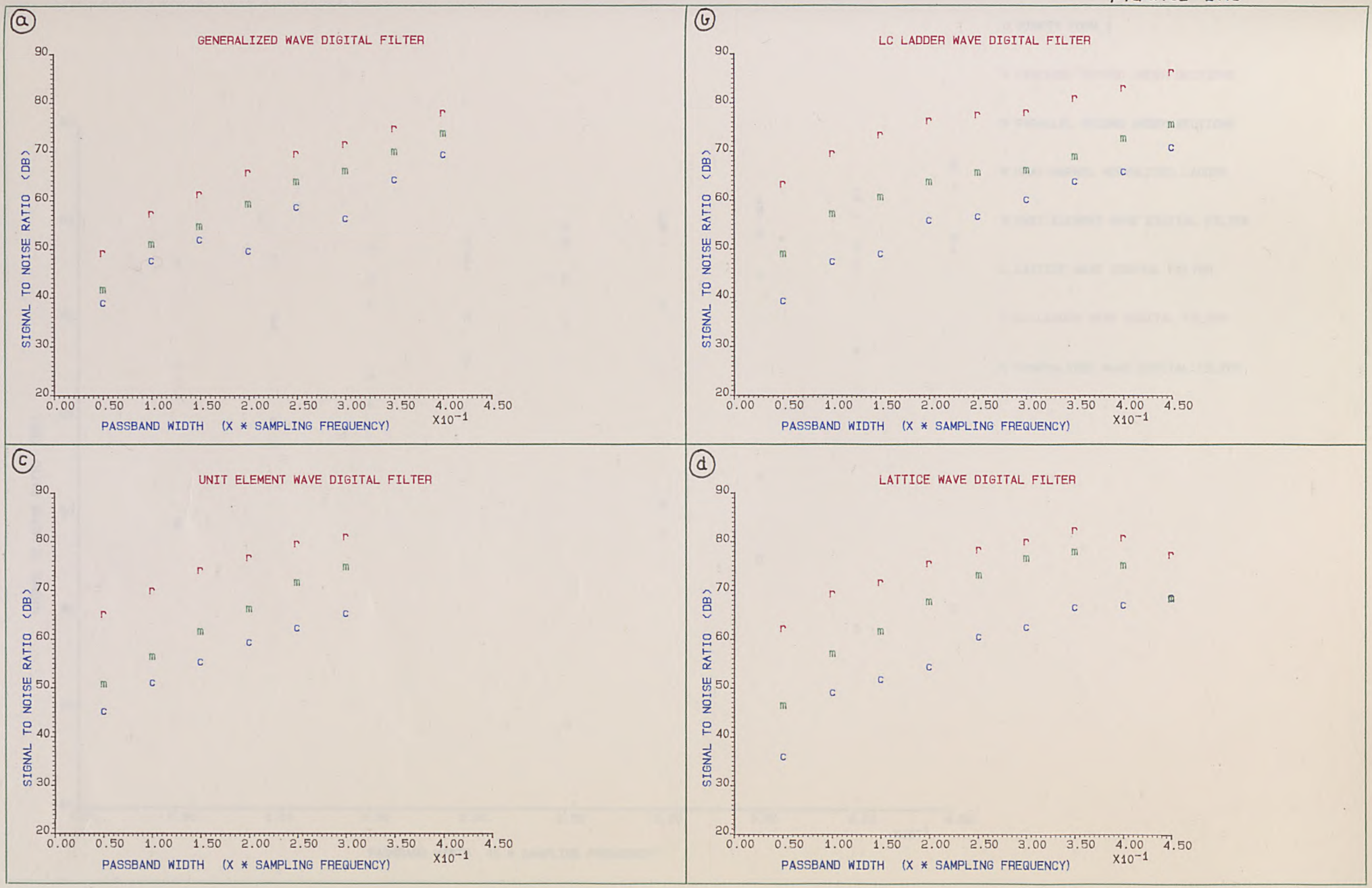
INPUT SIGNAL : COSINE (FREQ=0.5*P.B.WIDTH)

FIGURE 2.9



INPUT SIGNAL : COSINE (FREQ=0.5*P.B.WIDTH)

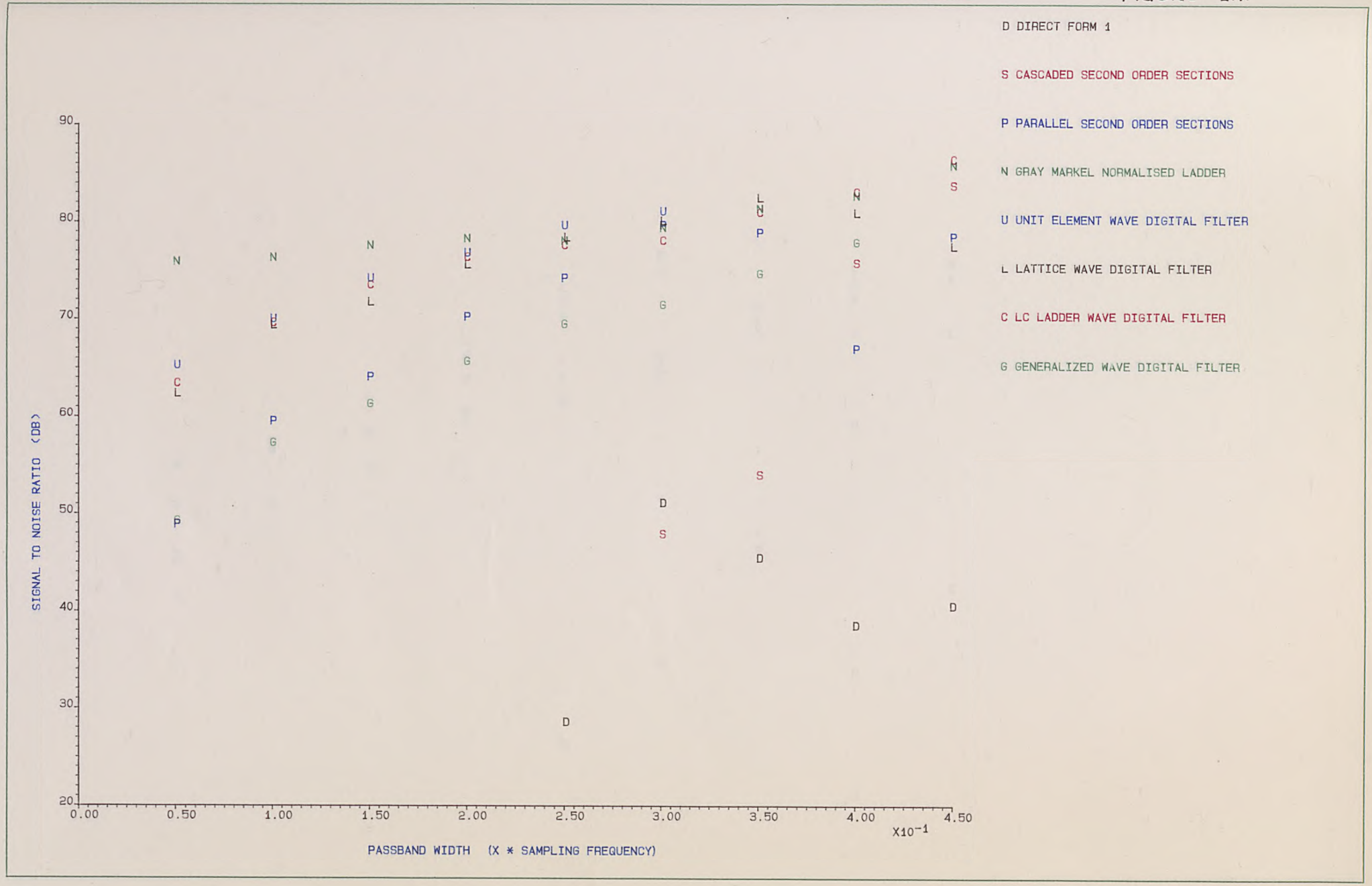
FIGURE 2.10



INPUT SIGNAL : COSINE (FREQ=0.5*P.B.WIDTH)

ROUNDING : ROUNDING

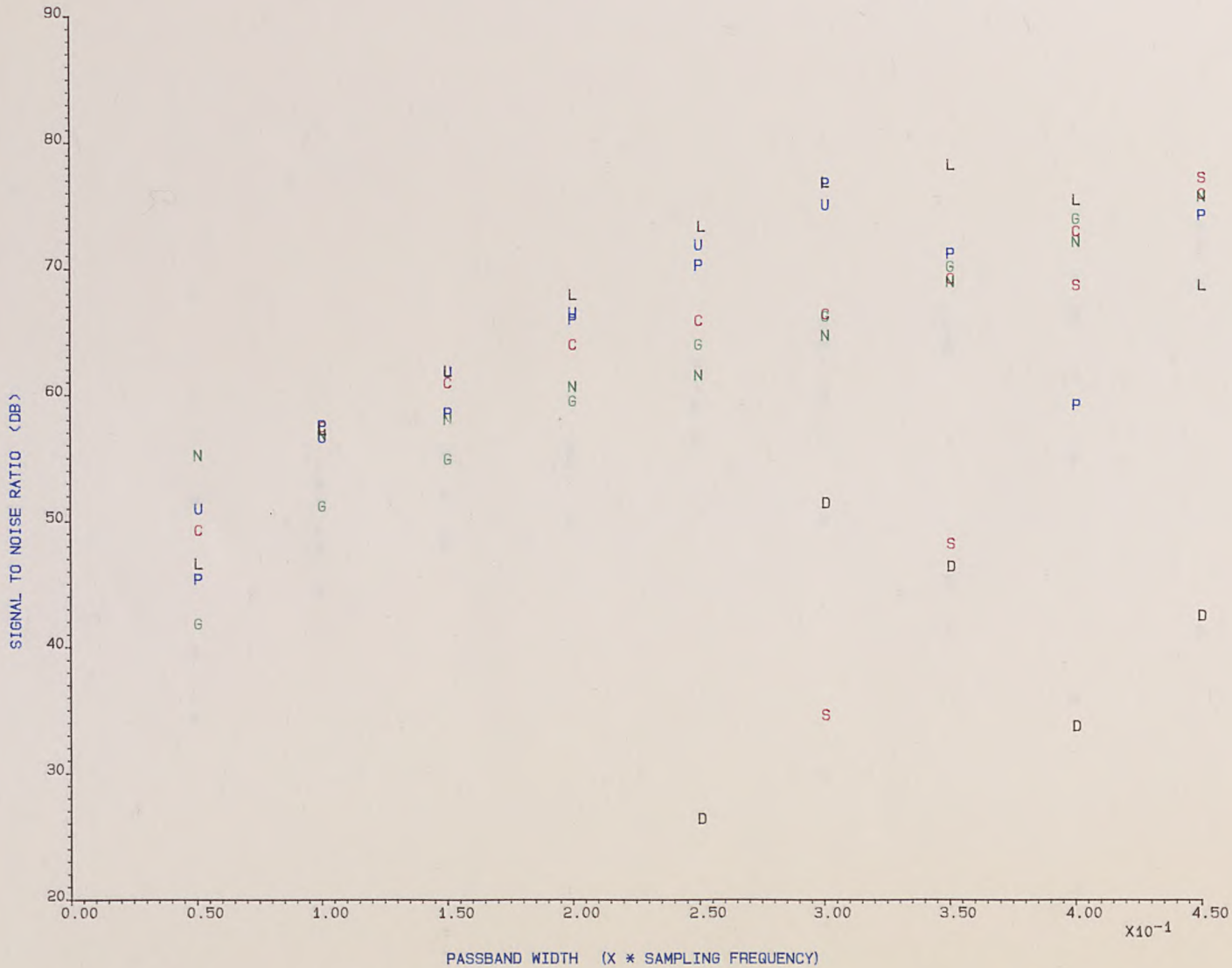
FIGURE 2.11



INPUT SIGNAL : COSINE (FREQ=0.5*P.B.WIDTH)

ROUNDING : MAGNITUDE TRUNCATING

FIGURE 2.12



D DIRECT FORM 1

S CASCADED SECOND ORDER SECTIONS

P PARALLEL SECOND ORDER SECTIONS

N GRAY MARKEL NORMALISED LADDER

U UNIT ELEMENT WAVE DIGITAL FILTER

L LATTICE WAVE DIGITAL FILTER

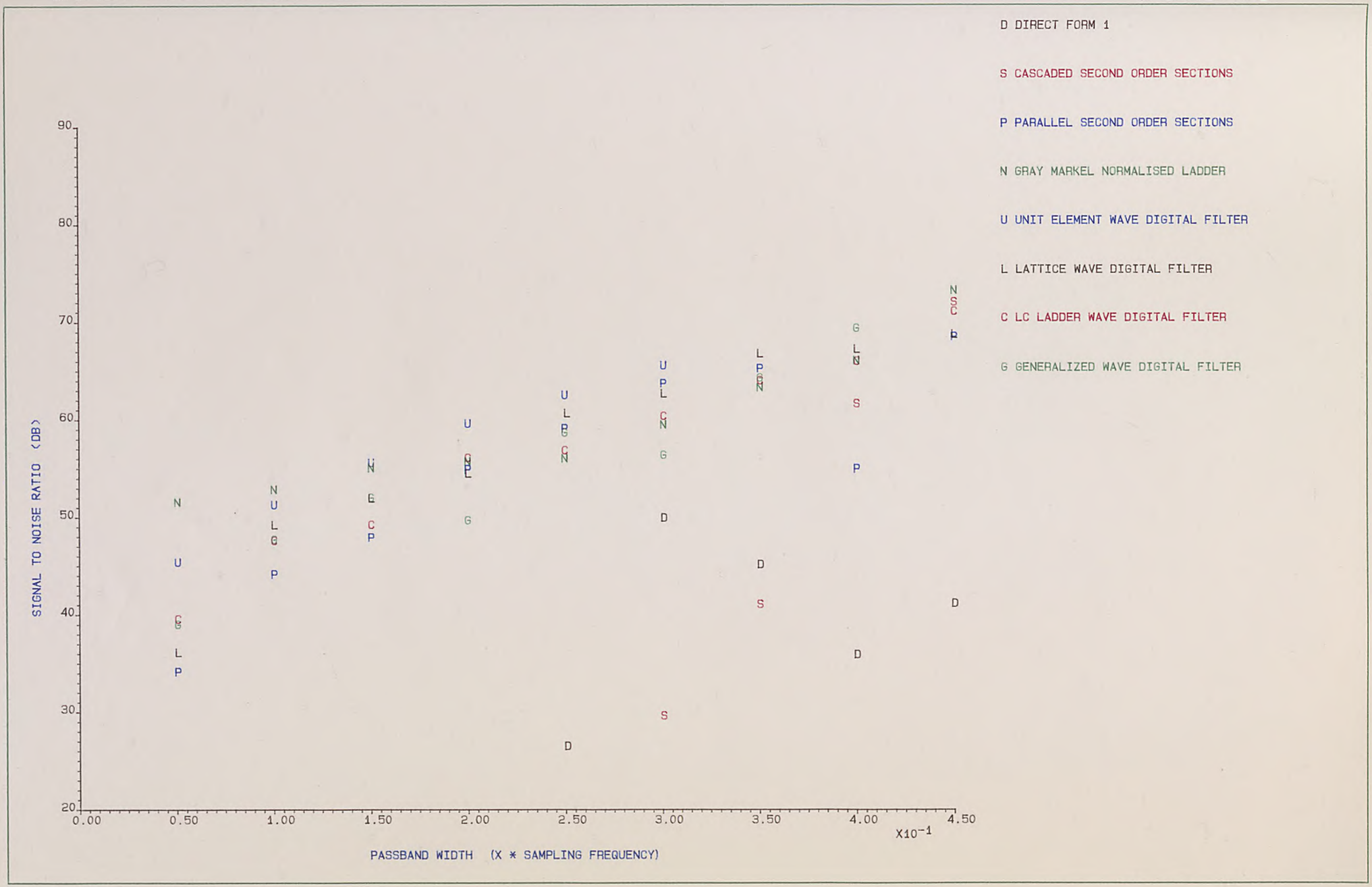
C LC LADDER WAVE DIGITAL FILTER

G GENERALIZED WAVE DIGITAL FILTER

INPUT SIGNAL : COSINE (FREQ=0.5*P.B.WIDTH)

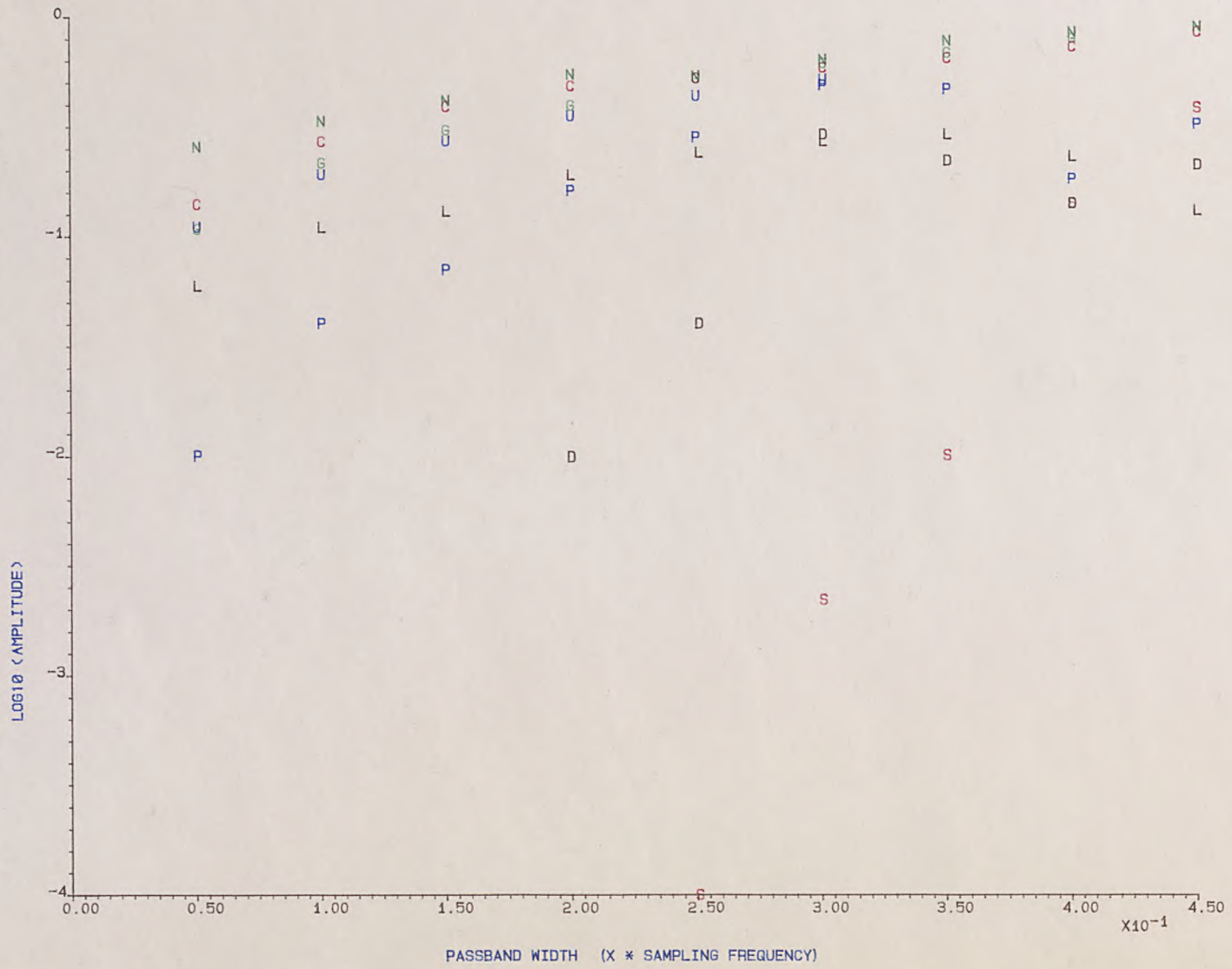
ROUNDING : 2S COMPLEMENT TRUNCATING

FIGURE 2.13



INPUT SIGNAL : COSINE (FREQ=0.5*P.B.WIDTH)

FIGURE 2.14



D DIRECT FORM 1

S CASCADED SECOND ORDER SECTIONS

P PARALLEL SECOND ORDER SECTIONS

N GRAY MARKEL NORMALISED LADDER

U UNIT ELEMENT WAVE DIGITAL FILTER

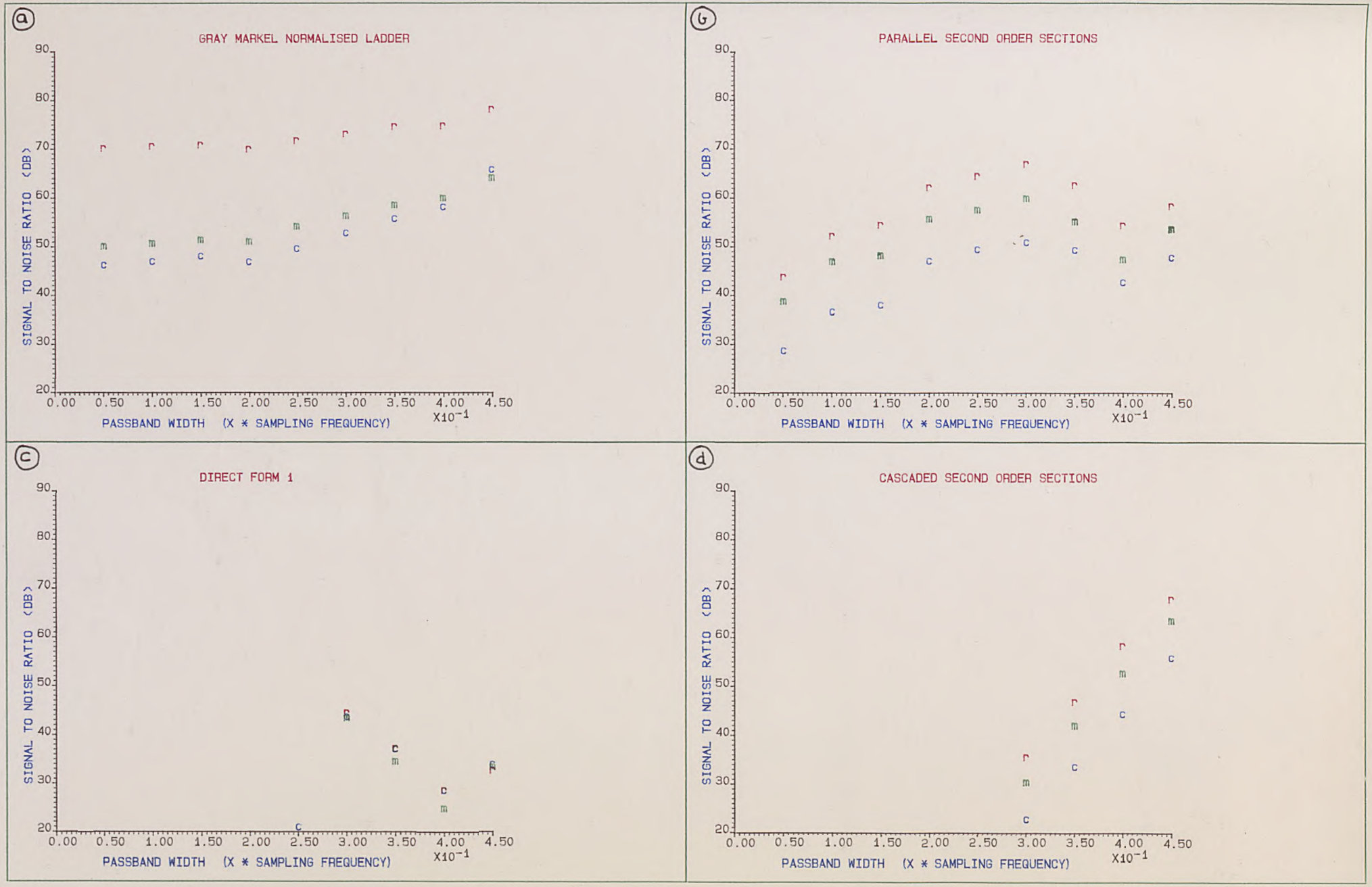
L LATTICE WAVE DIGITAL FILTER

C LC LADDER WAVE DIGITAL FILTER

G GENERALIZED WAVE DIGITAL FILTER

INPUT SIGNAL : GAUSSIAN DISTRIBUTED RANDOM

FIGURE 2.15



INPUT SIGNAL : GAUSSIAN DISTRIBUTED RANDOM

FIGURE 2.16

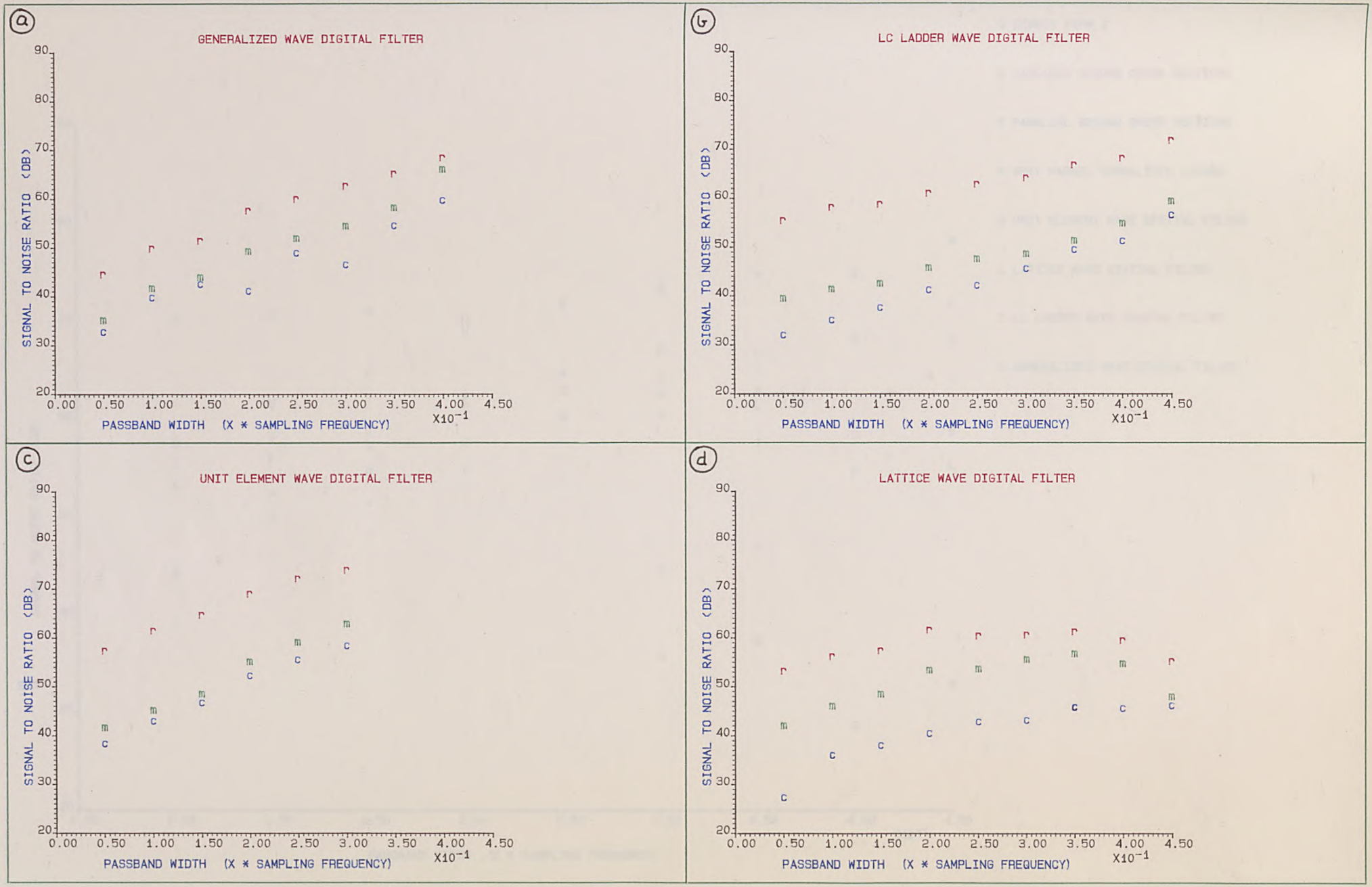
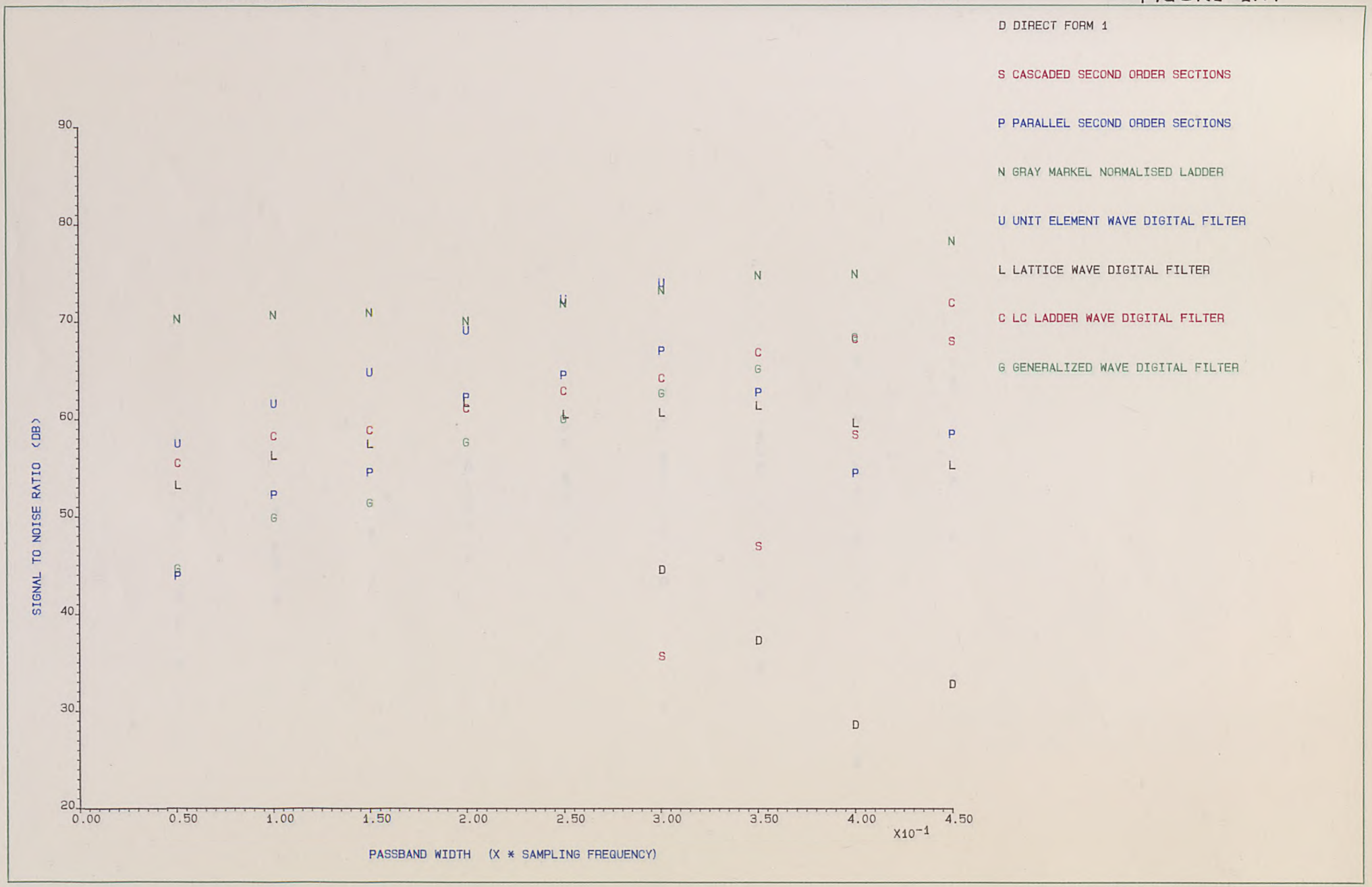


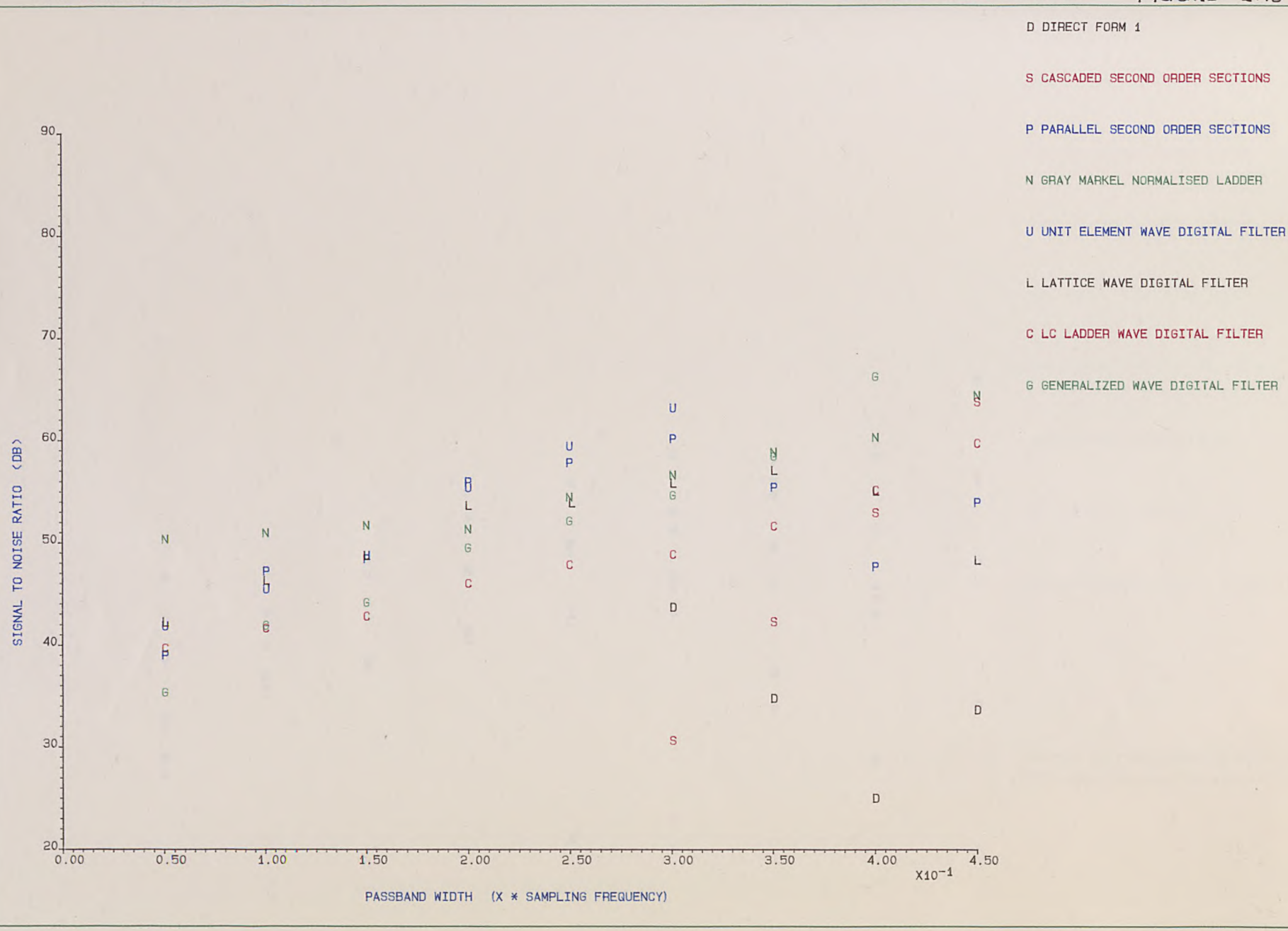
FIGURE 2.17


INPUT SIGNAL : GAUSSIAN DISTRIBUTED RANDOM
ROUNDING : ROUNDING
D DIRECT FORM 1
S CASCADED SECOND ORDER SECTIONS
P PARALLEL SECOND ORDER SECTIONS
N GRAY MARKEL NORMALISED LADDER
U UNIT ELEMENT WAVE DIGITAL FILTER
L LATTICE WAVE DIGITAL FILTER
C LC LADDER WAVE DIGITAL FILTER
G GENERALIZED WAVE DIGITAL FILTER
SIGNAL TO NOISE RATIO (DB)
90
80
70
60
50
40
30
20
0.00
0.50
1.00
1.50
2.00
2.50
3.00
3.50
4.00
4.50
X10-1
PASSBAND WIDTH (X * SAMPLING FREQUENCY)

INPUT SIGNAL : GAUSSIAN DISTRIBUTED RANDOM

ROUNDING : MAGNITUDE TRUNCATING

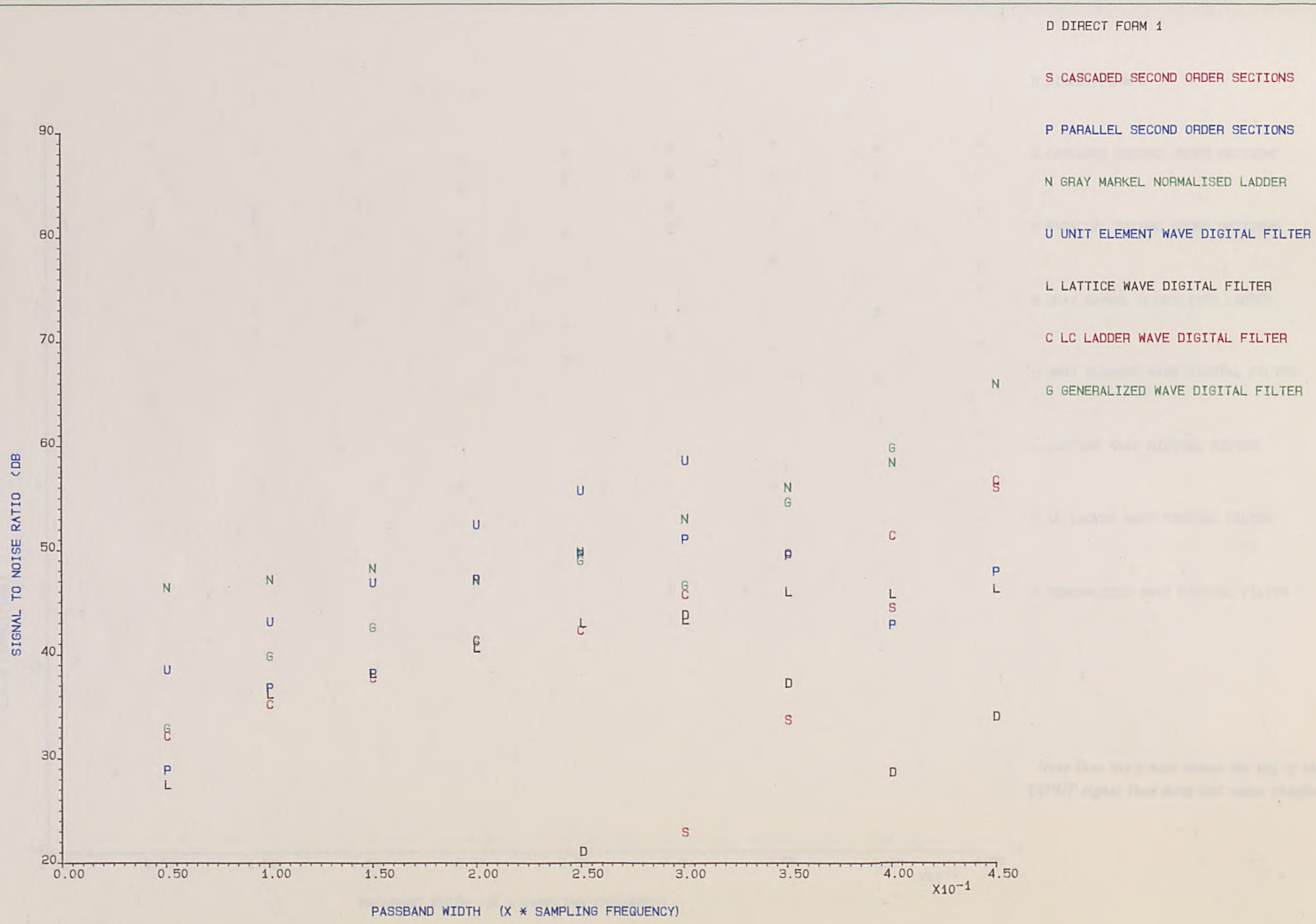
FIGURE 2.18



INPUT SIGNAL : GAUSSIAN DISTRIBUTED RANDOM

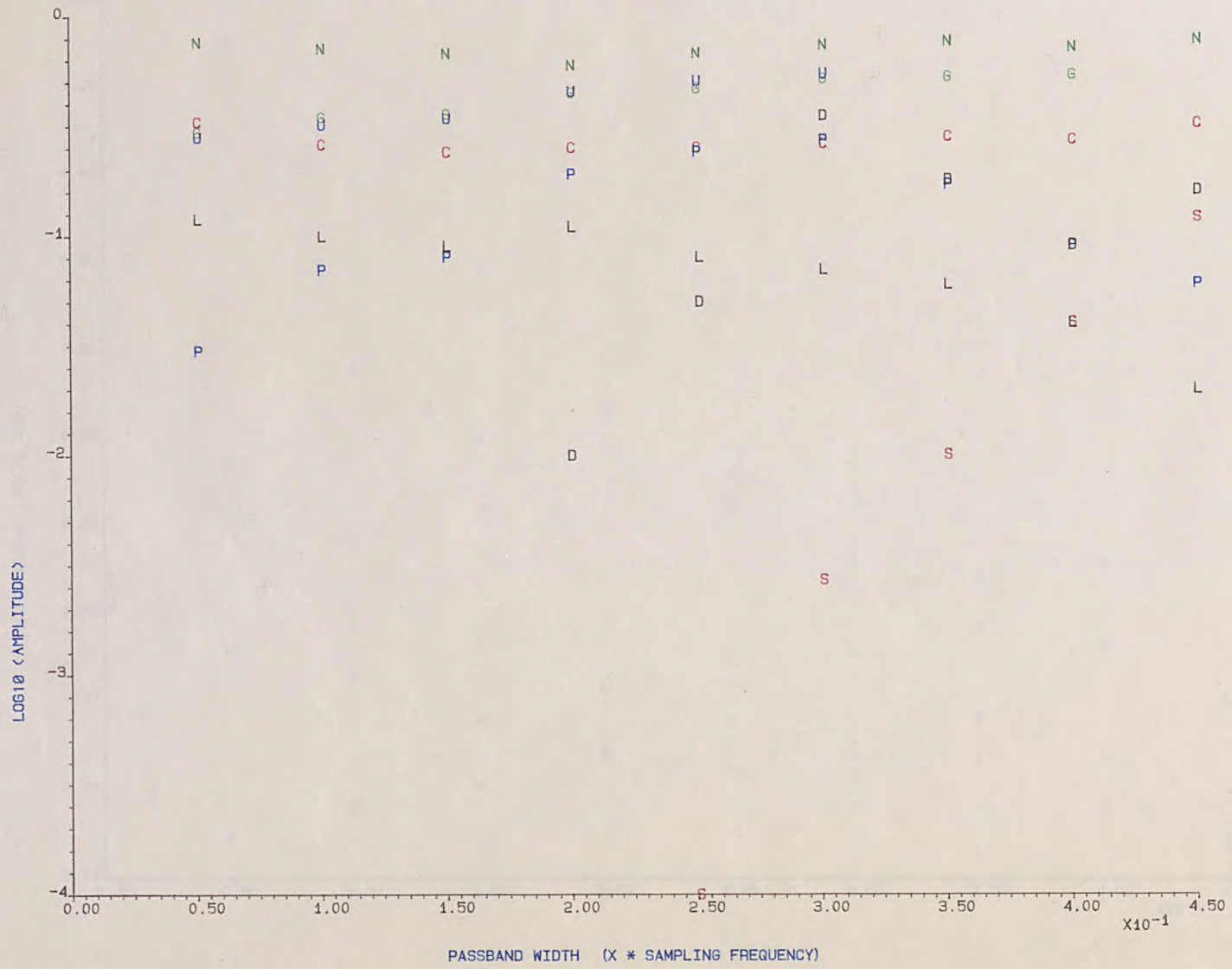
ROUNDING : 2S COMPLEMENT TRUNCATING

FIGURE 2.19



INPUT SIGNAL : GAUSSIAN DISTRIBUTED RANDOM

FIGURE 2.20



D DIRECT FORM 1

S CASCADED SECOND ORDER SECTIONS

P PARALLEL SECOND ORDER SECTIONS

N GRAY MARKEL NORMALISED LADDER

U UNIT ELEMENT WAVE DIGITAL FILTER

L LATTICE WAVE DIGITAL FILTER

C LC LADDER WAVE DIGITAL FILTER

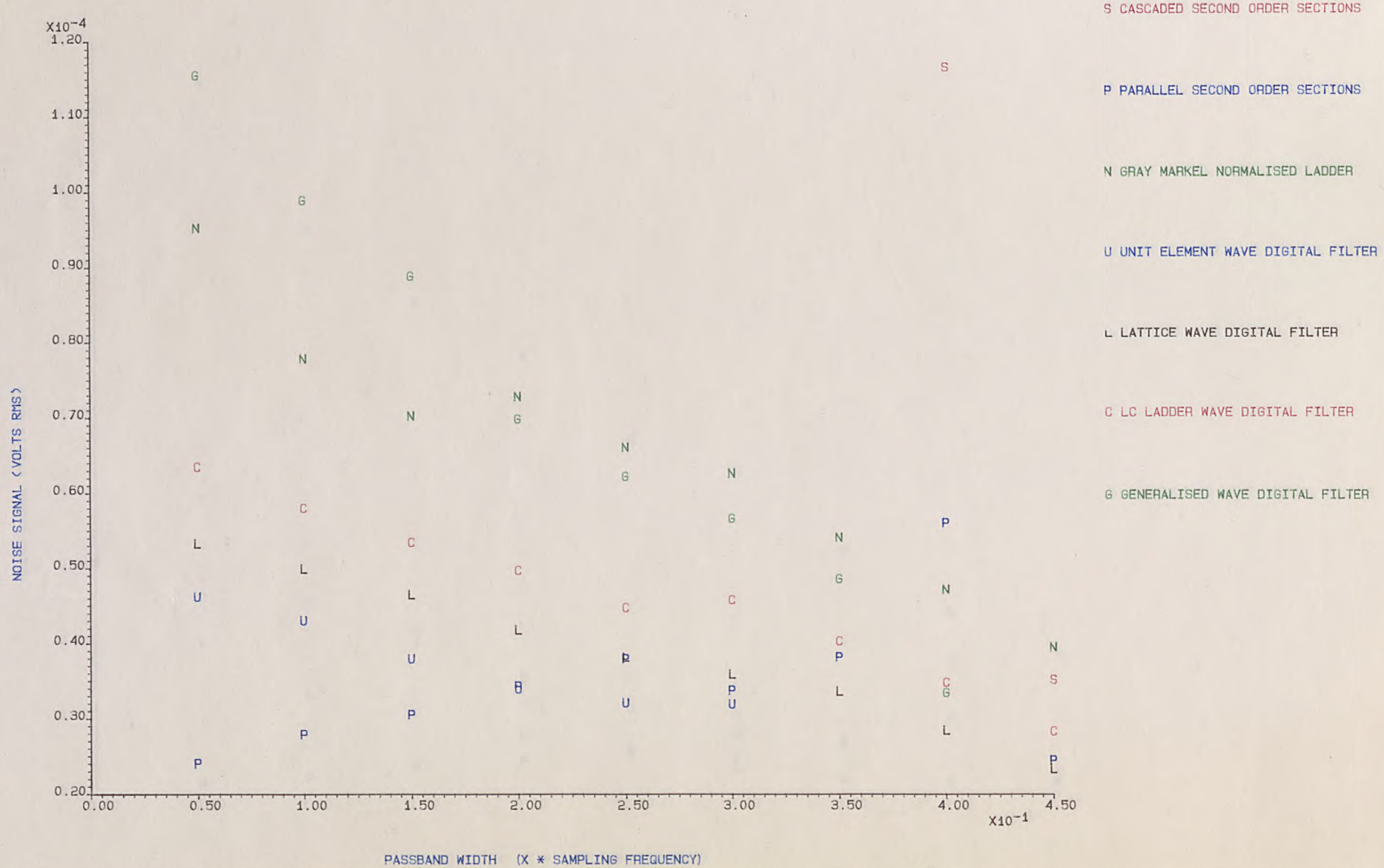
G GENERALIZED WAVE DIGITAL FILTER

Note that the y axis shows the log of the maximum INPUT signal that does not cause overflow

INPUT SIGNAL : GAUSSIAN DISTRIBUTED RANDOM

ROUNDING : ROUNDING

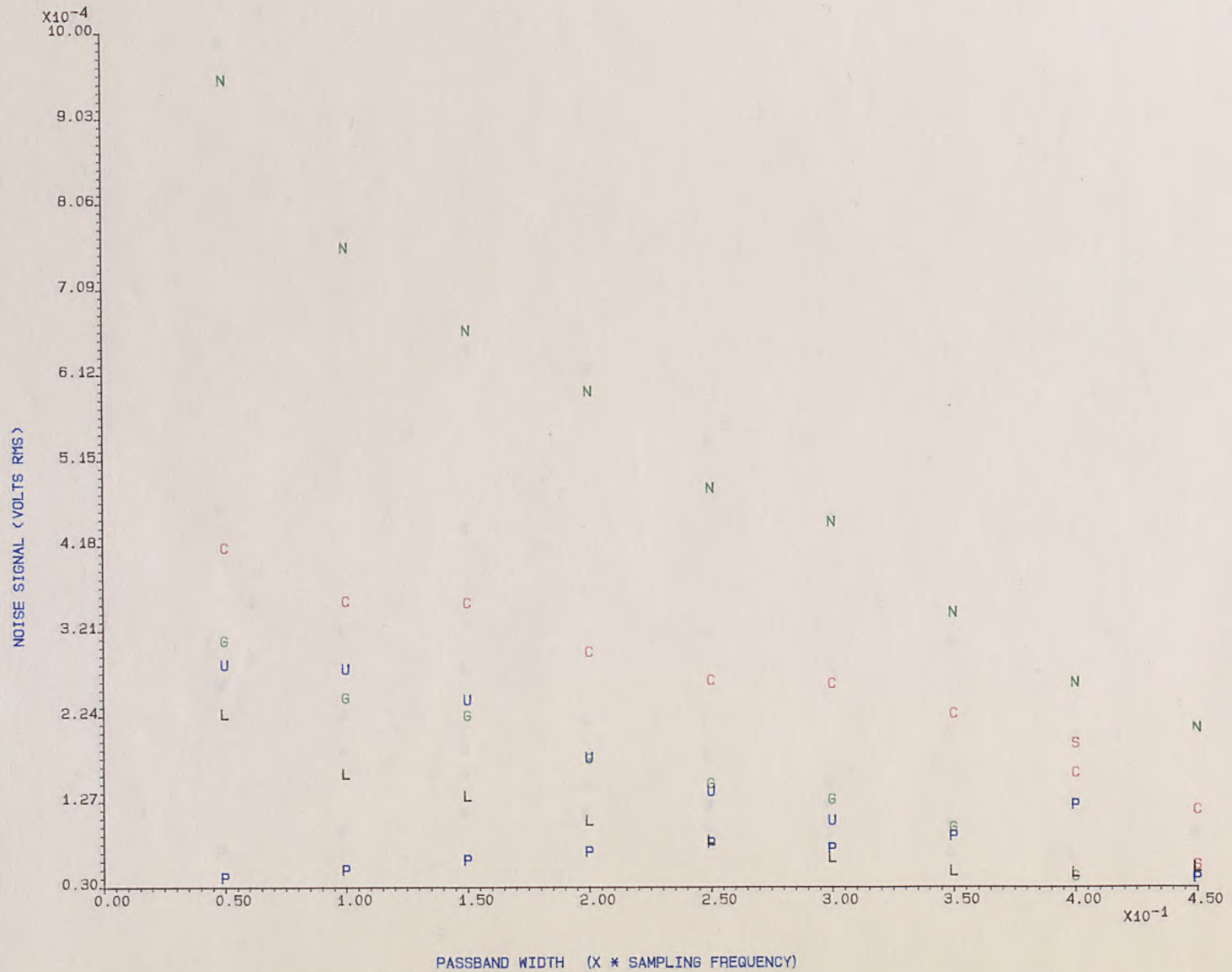
FIGURE 2.21



INPUT SIGNAL : GAUSSIAN DISTRIBUTED RANDOM

ROUNDING : MAGNITUDE TRUNCATING

FIGURE 2.22



S CASCADED SECOND ORDER SECTIONS

P PARALLEL SECOND ORDER SECTIONS

N GRAY MARKEL NORMALISED LADDER

U UNIT ELEMENT WAVE DIGITAL FILTER

L LATTICE WAVE DIGITAL FILTER

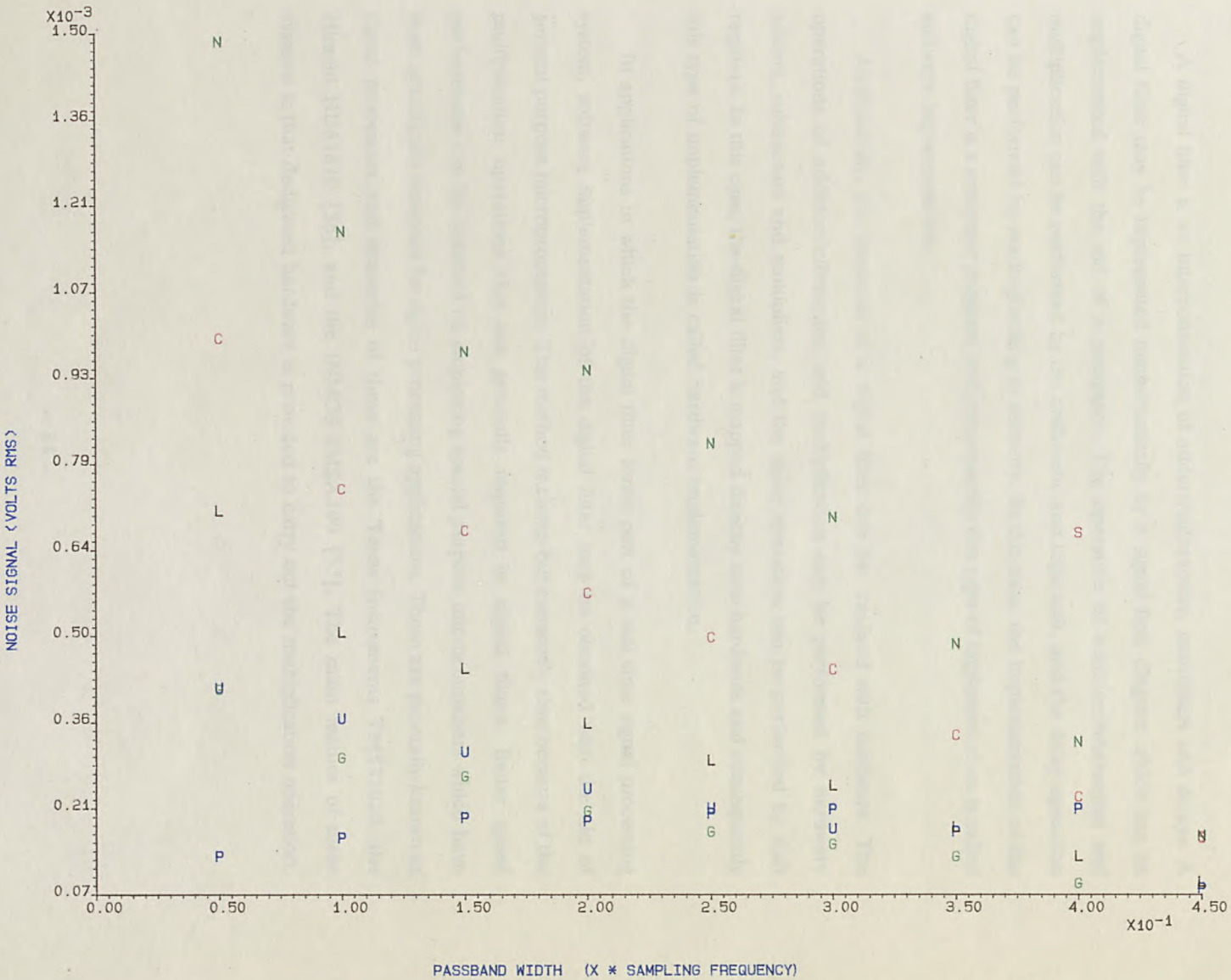
C LC LADDER WAVE DIGITAL FILTER

G GENERALISED WAVE DIGITAL FILTER

INPUT SIGNAL : GAUSSIAN DISTRIBUTED RANDOM

ROUNDING : 2S COMPLEMENT TRUNCATING

FIGURE 2.23



S CASCADED SECOND ORDER SECTIONS

P PARALLEL SECOND ORDER SECTIONS

N GRAY MARKEL NORMALISED LADDER

U UNIT ELEMENT WAVE DIGITAL FILTER

L LATTICE WAVE DIGITAL FILTER

C LC LADDER WAVE DIGITAL FILTER

G GENERALISED WAVE DIGITAL FILTER

CHAPTER THREE

COMPARISON OF DIGITAL FILTER STRUCTURES FROM THE HARDWARE IMPLEMENTATION VIEWPOINT

3.1 INTRODUCTION

A digital filter is an interconnection of adders/subtractors, multipliers and delays. A digital filter may be represented mathematically by a signal flow diagram which can be implemented with the aid of a computer. The operation of addition/subtraction and multiplication can be performed by the arithmetic and logic unit, and the delay operation can be performed by reading/writing to memory. In this case, the implementation of the digital filter is a computer program and consequently this type of implementation is called software implementation.

Alternatively, the elements of a digital filter can be realized with hardware. The operations of addition/subtraction and multiplication can be performed by hardware adders, subtractors and multipliers, and the delay operation can be performed by shift registers. In this case, the digital filter is mapped directly onto hardware and consequently this type of implementation is called hardware implementation.

In applications in which the digital filter forms part of a real time signal processing system, software implementation of the digital filter may be obtained with the aid of general purpose microprocessors. This method is cheap but extremely slow because of the multiplication operations that are generally required in digital filters. Better speed performance can be obtained by employing special purpose microcomputers which have been specifically designed for signal processing applications. These are generally known as signal processors and examples of these are the Texas Instruments TMS32020, the Hitachi HD61810 [56], and the INMOS IMSA100 [57]. The main feature of these systems is that dedicated hardware is provided to carry out the multiplication operation.

In applications in which the above techniques fail to satisfy sampling frequency requirements, it is necessary to employ hardware implementation techniques. Hardware implementation is more expensive but gives better speed performance than software implementation. Since our aim in this thesis is to design VLSI architectures for digital filters, we will concentrate on hardware implementation from hereon.

The aim in this chapter is to draw a comparison of the structures discussed in chapter 1 from the hardware implementation viewpoint. In order to do this, it is first necessary to outline the features which are important from the hardware implementation viewpoint.

The number of elements required to implement a particular filter is an important consideration. The multiplier is the most complex element and therefore it is desirable that the filter possesses a minimal number of these. Ultimately, we are interested in determining the complexity of the hardware required to implement a particular filter. This aspect will be considered in detail in section 3.2 and a measure of hardware complexity will be introduced. A comparison of the structures under consideration based on hardware complexity will also be given.

If a particular digital filter network can be viewed as an interconnected of a simpler network, then the digital filter network is said to possess regularity. Regularity is a requirement for multiplexability which is a desirable feature since it may be used to reduce the amount of hardware required to meet a specification. Clearly, this is limited by sampling frequency requirements. Multiplexability aspects will be discussed in section 3.3 and a comparison of the structures based on multiplexability aspects will be included.

The maximum sampling frequency of a particular filter will be dependent upon the degree of parallelism that is inherent in the network. The maximum sampling frequency is determined by the longest signal propagation time during one sampling period. This signal path is defined as the critical path and it is important to note that for some structures, it is dependent on the filter order. The above aspects of the various structures will be discussed in section 3.4.

3.2 HARDWARE COMPLEXITY

The amount of hardware required to implement a digital filter will determine the cost of implementation. If discrete ICs are used, then the number of elements will determine component costs and power requirements. For such implementations, we may use the number of elements required as a measure of hardware complexity. However, it is important to note that the multiplier is the most complex component, and this fact must be borne in mind when drawing comparisons between various structures.

The element counts for the structures under consideration are shown in table 3.1. It can be seen that the Gray–Markel normalized ladder requires the largest number of multipliers. The lattice WDF requires the smallest number of multipliers and also smallest number of total elements. The cascaded unit elements WDF requires one more multiplier than the lattice WDF for a given order. However, it is known that the cascaded unit elements WDF requires a higher order than the rest of the structures under consideration in order to satisfy a given tolerance scheme. Also note that the LC ladder WDF requires almost twice the number of elements as the lattice WDF.

When considering VLSI implementation, it is more important to know the number of gates required for a particular digital filter than the number of elements. Although this method does not take interconnection complexity into account, it is sufficient for the purposes of comparison since the gate count is directly related to silicon area and power requirements.

We can therefore introduce a measure of hardware complexity in terms of 2–input gate equivalents. The method used here is to represent each adder with a ripple carry adder, each multiplier with a number of ripple carry adders, and each delay with a shift register comprising D–type edge triggered flip flops (see appendix B1 for details). The gate counts for the various structures under consideration are shown as a function of signal wordlength and coefficient wordlength in table 3.2. In order to gain an insight into the gate counts that may be required in practice, it is useful to consider an example.

The gate counts for a seventh order filter with a coefficient wordlength of 6 bits and a signal wordlength of 1 bit are shown in table 3.3. For a signal wordlength of M bits, these

values should be multiplied by M. Note from table 3.3 that the lattice WDF has the lowest gate count followed by the cascaded unit elements WDF. The Gray–Markel normalized ladder requires the largest number of gates and the LC ladder WDF requires almost twice the number of gates compared to the lattice WDF.

It is worth recalling at this stage that the structures with a low coefficient wordlength sensitivity will require less complex multipliers. Therefore, for a given specification, the gate counts will be different to the ones shown. However, for the purposes of a general comparison, the results shown in table 3.3 are adequate.

The order of the structures in terms of hardware complexity as illustrated in table 3.3 will vary depending upon the implementation scheme employed. The definition of a complexity ratio, CR, between the gate requirements of the multiplier and the adder allows us to define conditions from table 3.1 under which changes will occur to the order shown in table 3.3 ;

(i) if CR>8, then the cascaded second order sections requires a larger number of gates than the LC ladder WDF (ignoring gate requirements for the delays).

(ii) if CR>4n−1 (n is the filter order), then the parallel second order sections and the direct form II will require a larger number of gates than the LC ladder WDF (ignoring gate requirements for the delays).

(iii) if 1.5n<CR+1, then the cascaded unit elements WDF will require a larger number of gates than the IVR generalized WDF.

(iv) under all other values of n and CR>2, the order will be as shown in table 3.3.

It is important to note that in table 3.2 a linear dependence of hardware complexity on signal wordlength is assumed which may not necessarily be the case, e.g. in implementations employing carry lookahead adders. In such cases, table 3.2 would have to be modified. However, for a given signal wordlength, the complexity ratio between multipliers and adders may be determined and used in conjunction with table 3.3 and the previously defined conditions to obtain the order in terms of complexity.

In summary, note that the order of the structures in terms of hardware complexity is as shown in table 3.3 with the exception of the LC ladder WDF and the IVR generalized WDF which may take different positions in the order depending on the complexity ratio.

3.3 MULTIPLEXABILITY ASPECTS

By multiplexability, we mean the ability to implement an nth order filter with time multiplexed versions of first or second order filters.

If a digital network can be viewed as interconnected instances of a sub–network, then the network is said to possess regularity. A digital filter structure must possess regularity in order to be multiplexable. However, regularity does not imply multiplexability since multiplexability is also dependent upon the sequence in which the arithmetic operations must be performed.

Multiplexability is important because it may be employed to reduce the amount of hardware required to implement a digital filter. In the simplest example of a first order filter multiplexed to realize an nth order filter, the maximum sampling frequency possible will be 1/n of the maximum sampling frequency possible with a direct implementation, i.e. multiplexability is useful provided sampling frequency requirements can be met.

From the VLSI viewpoint, multiplexability is important because we may then design standard macro cells which implement first or second order filters and use these either in the multiplexed form, or connect a number of these cells together to realize higher order filters. In the multiplexed form, the filter will obviously require a lower silicon area, and this is desirable especially when the filter is part of a larger system that is to be integrated.

The direct form II structure shown in figure 3.1a can be implemented by time multiplexing the sub–structure shown in the dotted area. The cascaded and parallel second order sections shown in figures 3.2 and 3.3 can be implemented by time multiplexing one second order section. The Gray–Markel normalized ladder structure can be implemented by multiplexing the first order section shown in the dotted area.

The cascaded unit elements WDF shown in figure 3.5 is not multiplexable, but by moving the delays in the upper arm to the lower arm, can be made multiplexable. In this case, the basic two port adaptor (shown in the dotted area) is multiplexed to implement the filter. The lattice WDF shown in figure 3.6 can be implemented by multiplexing the

sub–structure shown in the dotted area, but it can be seen that control aspects of the resultant hardware would be complex.

The LC ladder WDF can be implemented by multiplexing the three port adaptor as shown in figure 3.7. The IVR generalized WDF cannot be easily multiplexed.

In summary, we note that all the structures except the IVR generalized WDF can be multiplexed. It is also clear that some structures will result in more complex multiplexed schemes, most notably, the lattice WDF from the examples considered here.

3.4 SPEED ASPECTS

The maximum sampling frequency that can be achieved is particularly important when choosing a filter structure. One reason is that when considering general purpose filters, it is clear that those with higher speed will have wider application. Furthermore, if the structure is multiplexable, then a reduction in hardware requirements can be achieved for a wider range of applications.

For a given filter, the maximum sampling frequency is determined by the longest signal path that must be traversed in one sampling period. This path will be referred to as the critical path from hereon.

The maximum sampling frequency is determined by the complexity of the critical path. By this we mean the number of adders/subtractors or multipliers through which a signal must propagate in one sampling period. It should be obvious that no delays are present in the critical path. Since the multipliers will result in higher time penalties, we are interested in structures with a minimal number of multipliers in the critical path.

The critical paths for the various structures are shown diagramatically in figures 3.1 to 3.8 and in terms of number of adders/subtractors and multipliers in table 3.4. It can be seen that the Gray–Markel normalized ladder has the highest number of multipliers in the critical path and that the number is dependent on the filter order. The interesting point to note is that the cascaded unit elements WDF and the LC ladder WDF have a fixed number of multipliers in the critical path. This is also true for the direct form II and the parallel second order sections. Note also that, although the multipliers in the lattice WDF are dependent on the filter order, the number is lower than that for the Gray– Markel normalized ladder, IVR generalized WDF and cascaded second order sections.

For all the structures except the cascaded unit elements WDF and the LC ladder WDF, the number of adders in the critical path are dependent on the filter order.

If we assume the models for the multipliers as discussed in section 3.3, then we can introduce a measure of critical path complexity in terms of '2 input adder equivalents'. On the basis of this, and for a coefficient wordlength of 6 bits, the complexities for the

structures under consideration are given in table 3.5. An example for seventh order filters is included and it can be seen from this that the Gray–Markel normalized ladder and the IVR generalized WDF will give the worst speed performance. The LC ladder WDF will give the best speed performance followed by the cascaded unit elements WDF.

In summary, we note that the order of the structures in terms of speed performance is as follows; LC ladder WDF, cascaded unit elements WDF, parallel second order sections, lattice WDF, direct form II, cascaded second order sections, IVR generalized WDF, Gray–Markel normalized ladder.

3.5 DISCUSSION AND CONCLUSIONS

Our aim in this chapter has been to compare a number of structures and to identify those which are most suitable for hardware implementation. It has been shown that low hardware complexity, multiplexability and high speed are desirable attributes in a digital filter structure.

It is concluded that the Gray–Markel normalized ladder structure is the least suitable for hardware implementation since it has the highest hardware complexity and gives the worst speed performance. The IVR generalized WDF is not suitable for hardware implementation because it is not multiplexable and gives poor speed performance although it has a low hardware complexity.

The cascaded and parallel second order sections are equally complex in terms of hardware complexity but the parallel second order sections gives a better speed performance. In terms of multiplexability, it can be seen that the cascaded second order sections would require a less complex scheme. The direct form II is comparable with the cascaded and parallel second order sections in terms of hardware complexity, but is superior to the cascaded form and inferior to the parallel form in terms of speed performance. It is worth reiterating here that the direct form is generally not considered for implementation because of its poor sensitivity properties as discussed in chapter 2.

It has been shown that the cascaded unit elements WDF is the second best structure from the viewpoints of hardware complexity and speed performance. The lattice WDF is the best structure in terms of hardware complexity and gives a good speed performance. The LC ladder WDF gives the best speed performance although it is only better than the Gray– Markel normalized ladder in terms of hardware complexity.

The superiority of WDFs from the finite wordlength properties viewpoint has been demonstrated in chapter 2. In this chapter, it has been shown that the cascaded unit elements WDF and the lattice WDF are the most suitable structures for hardware implementation.

TYPE OF FILTER STRUCTURE	n ODD OR EVEN	NUMBER OF TWO INPUT ADDERS / SUBTRACTORS	NUMBER OF MULTIPLIERS	NUMBER OF DELAYS
DIRECT FORM II		2n	2n+1	n
CASCADED SECOND ORDER SECTIONS	ODD EVEN	2n 2n	(5n+1)/2 5n/2	n n
PARALLEL SECOND ORDER SECTIONS		2n	2n+1	n
GRAY-MARKEL NORMALIZED LADDER		3n	4n+1	n
CASCADED UNIT ELEMENTS WDF		3n+1	n+1	n
LATTICE WDF		3n+1	n	n
LC LADDER WDF		6n-1	2n	2n-1
IVR GENRALIZED WDF	ODD EVEN	(9n+3)/2 9n/2	n n	n n

n : filter order

TABLE 3.1 : ELEMENT COUNTS FOR STRUCTURES UNDER CONSIDERATION

TYPE OF FILTER STRUCTURE	NUMBER OF TWO INPUT NAND GATES
DIRECT FORM II	45.M.n + 38.M.N.n + 19.M.N
CASCADED SECOND ORDER SECTIONS	45.M.n + (95/2).M.N.n (n EVEN) 45.M.n + (95/2).M.N.n + (19/2).M.N (n ODD)
PARALLEL SECOND ORDER SECTIONS	45.M.n + 38.M.N.n + 19.M.N
GRAY-MARKEL NORMALIZED LADDER	64.M.n + 76.M.N.n + 19.M.N
CASCADED UNIT ELEMENTS WDF	64.M.n + 19M + 19.M.N.n + 19.M.N
LATTICE WDF	64.M.n + 19M + 19.M.N.n
LC LADDER WDF	128.M.n - 26.M + 38.M.N.n
IVR GENERALIZED WDF	(185/2).M.n + (57/2).M + 19.M.N.n (n ODD) (185/2).M.n + 19.M.N.n (n EVEN)

n order of filter

M signal width in number of bits

N number of non-zero coefficient bits

TABLE 3.2 : GATE COUNTS FOR STRUCTURES UNDER CONSIDERATION

TYPE OF FILTER STRUCTURE	NUMBER OF TWO INPUT NAND GATES
LATTICE WDF	1265
CASCADED UNIT ELEMENTS WDF	1379
IVR GENERALIZED WDF	1474
DIRECT FORM II	2025
PARALLEL SECOND ORDER SECTIONS	2025
CASCADED SECOND ORDER SECTIONS	2082
LC LADDER WDF	2466
GRAY–MARKEL NORMALIZED LADDER	3754

filter order = 7

coefficient precision = 6 bits

signal precision = 1 bit

TABLE 3.3 : GATE COUNTS FOR SEVENTH ORDER FILTER

		CRITICAL PATH	
TYPE OF FILTER STRUCTURE	ODD or EVEN ORDER	NUMBER OF TWO INPUT ADDERS / SUBTRACTORS	NUMBER OF MULTIPLIERS
DIRECT FORM II		n + 1	2
CASCADED SECOND ORDER SECTIONS	ODD EVEN	n + 2 n + 1	(n + 3)/2 (n + 2)/2
GRAY–MARKEL NORMALIZED LADDER		n + 2	n + 1
CASCADED UNIT ELEMENTS WDF		4	2
LC LADDER WDF		7	1
IVR GENERALIZED WDF	ODD EVEN	(7n + 7)/2 (7n + 4)/2	(n + 1)/2 n/2
PARALLEL SECOND ORDER SECTIONS		next higher integer to LOG2(n + 1)	2
LATTICE WDF	n=3 n=5 n=7 n=9 n=11 .	3 5 5 7 7 .	1 2 2 3 3 .

n = filter order

TABLE 3.4 : CRITICAL PATHS FOR STRUCTURES UNDER CONSIDERATION

TYPE OF FILTER STRUCTURE	ODD or EVEN ORDER	CRITICAL PATH: NUMBER OF TWO INPUT ADDER EQUIVALENTS	CRITICAL PATH: SEVENTH ORDER EXAMPLE
DIRECT FORM II		n + 13	20
CASCADED SECOND ORDER SECTIONS	ODD EVEN	4n + 11 4n + 7	39 35
GRAY-MARKEL NORMALIZED LADDER		7n + 8	57
CASCADED UNIT ELEMENTS WDF		16	16
LC LADDER WDF		13	13
IVR GENERALIZED WDF	ODD EVEN	(13n + 13)/2 (13n + 4)/2	52
PARALLEL SECOND ORDER SECTIONS		{next higher integer LOG2(n + 1)} + 12	15
LATTICE WDF	n=3 n=5 n=7 n=9 n=11 .	9 17 17 25 25 .	17

n = filter order

TABLE 3.5 : CRITICAL PATH COMPLEXITIES IN TERMS OF TWO -INPUT ADDER EQUIVALENTS FOR STRUCTURES UNDER CONSIDERATION

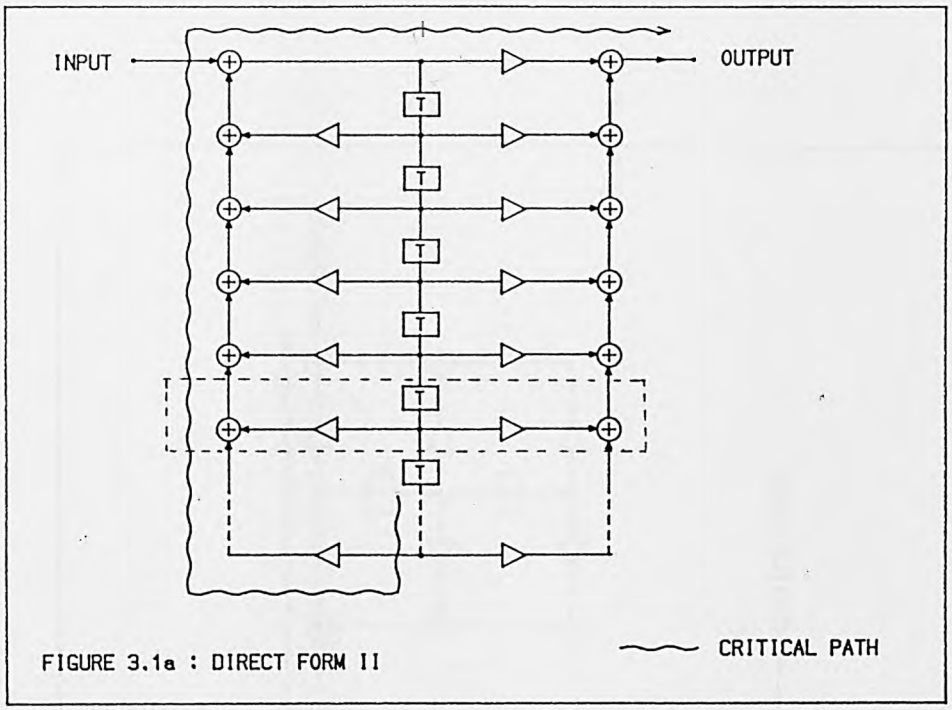


FIGURE 3.1a : DIRECT FORM II

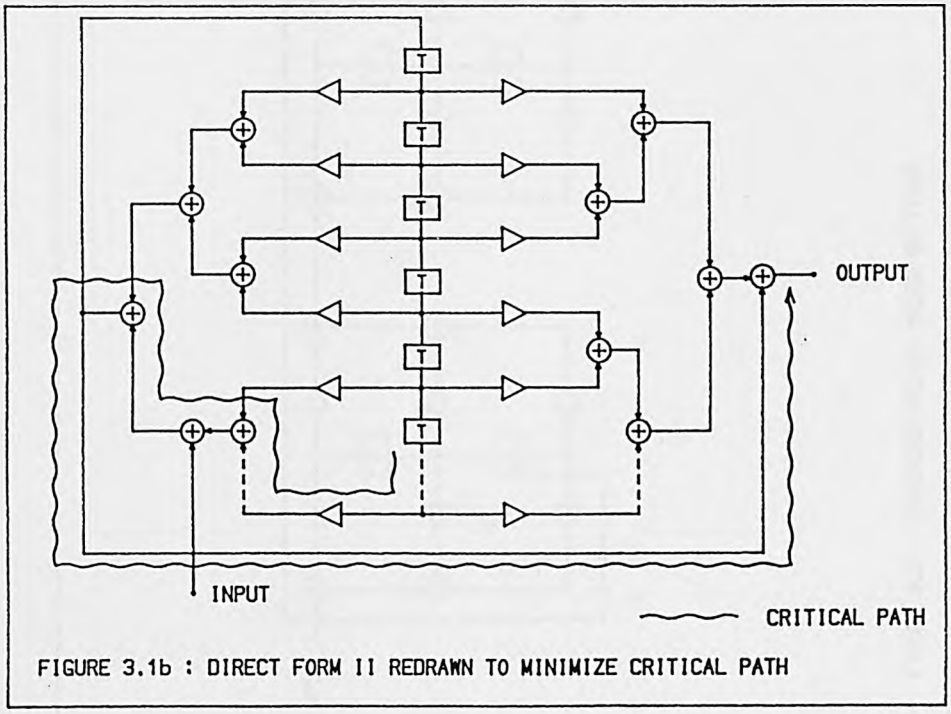


FIGURE 3.1b : DIRECT FORM II REDRAWN TO MINIMIZE CRITICAL PATH

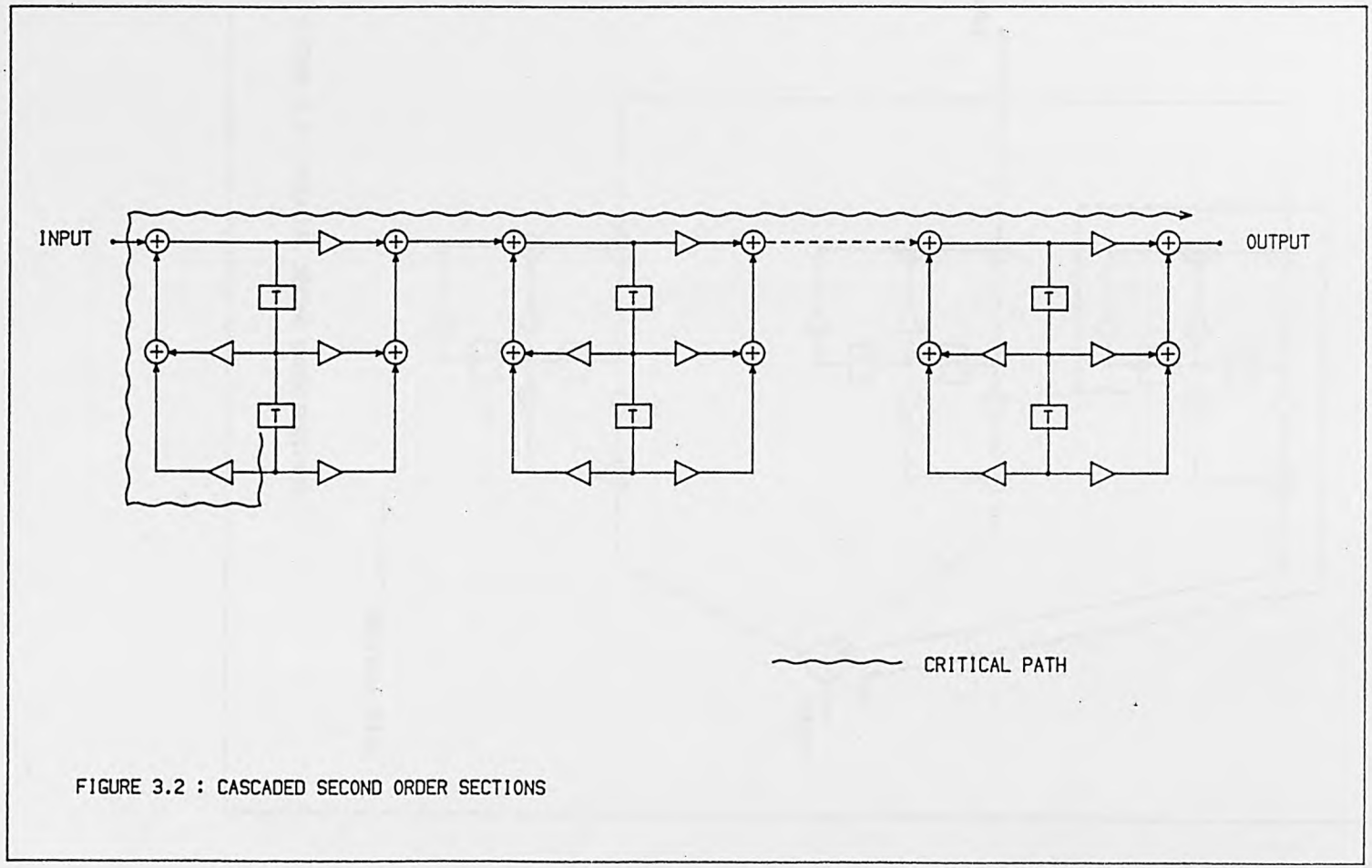


FIGURE 3.2 : CASCADED SECOND ORDER SECTIONS

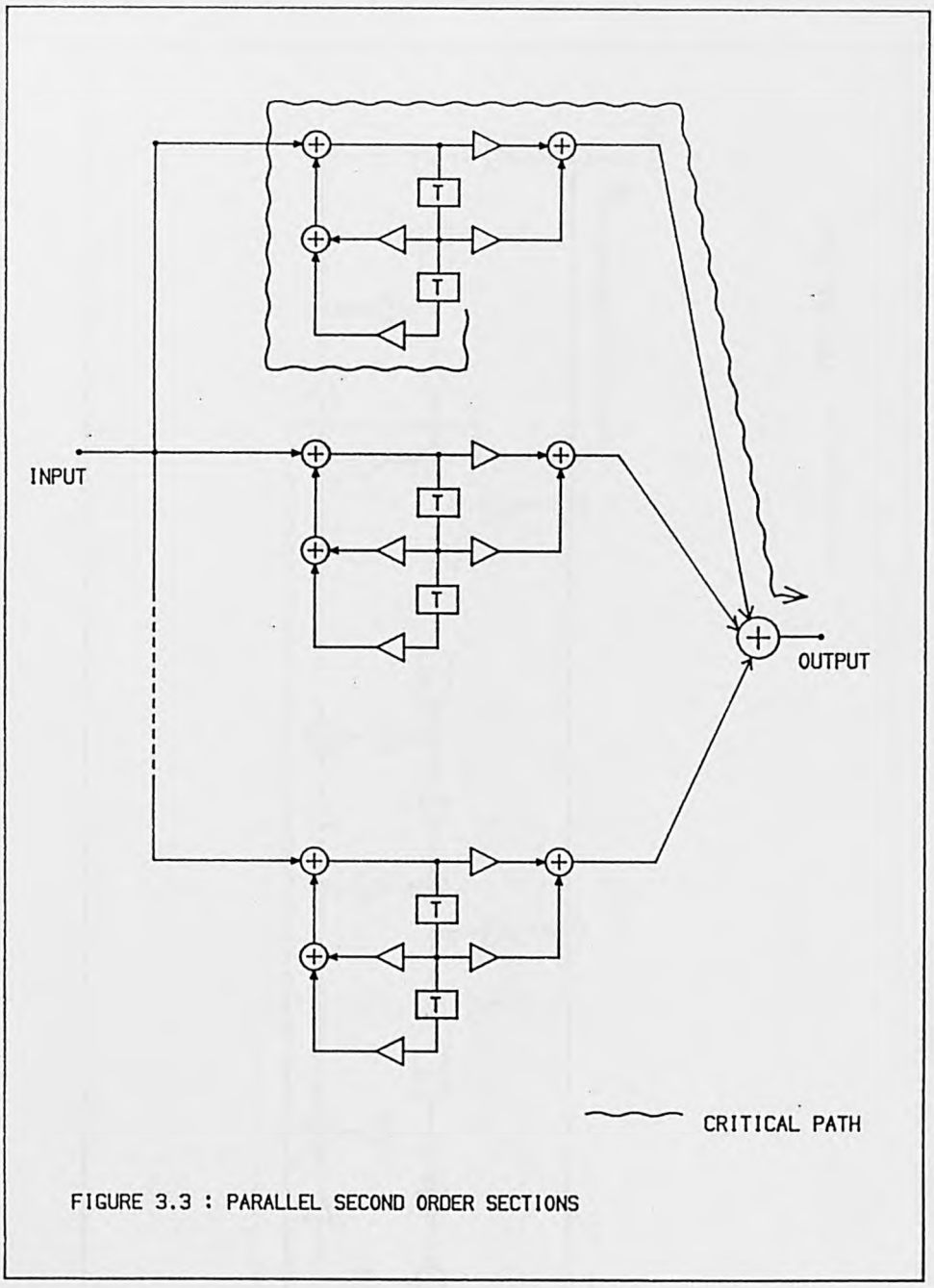


FIGURE 3.3 : PARALLEL SECOND ORDER SECTIONS

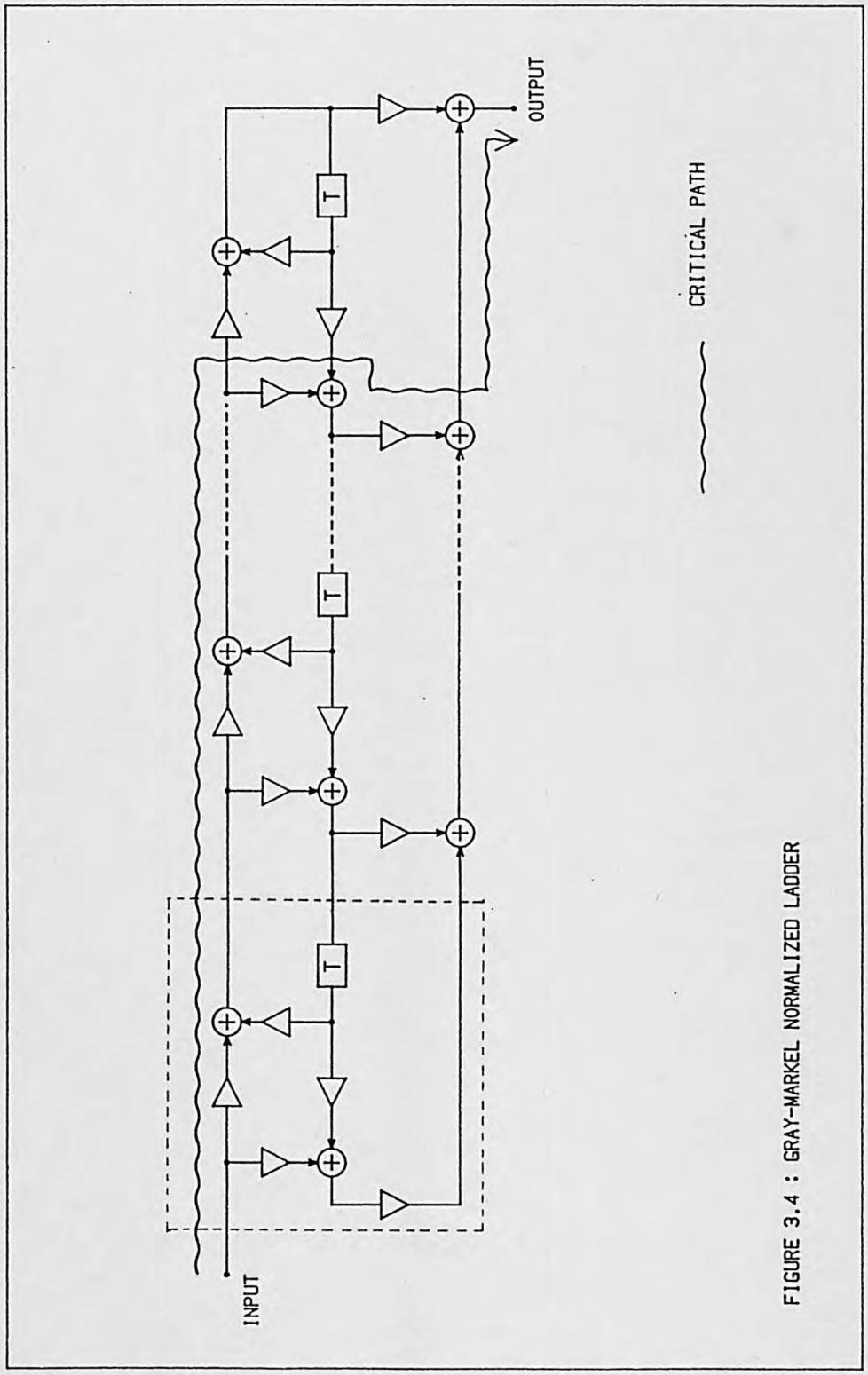


FIGURE 3.4 : GRAY-MARKEL NORMALIZED LADDER

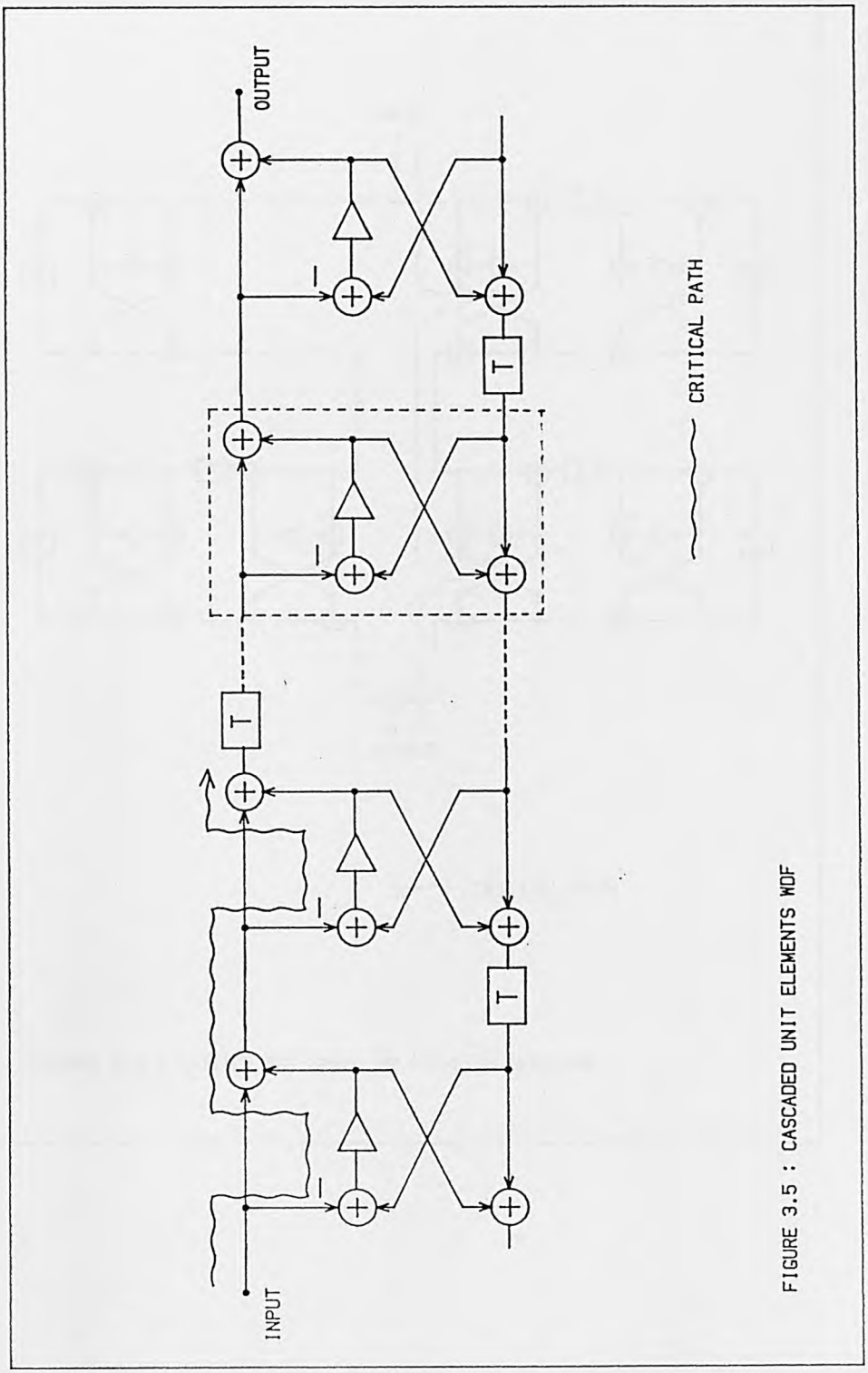


FIGURE 3.5 : CASCADED UNIT ELEMENTS WDF

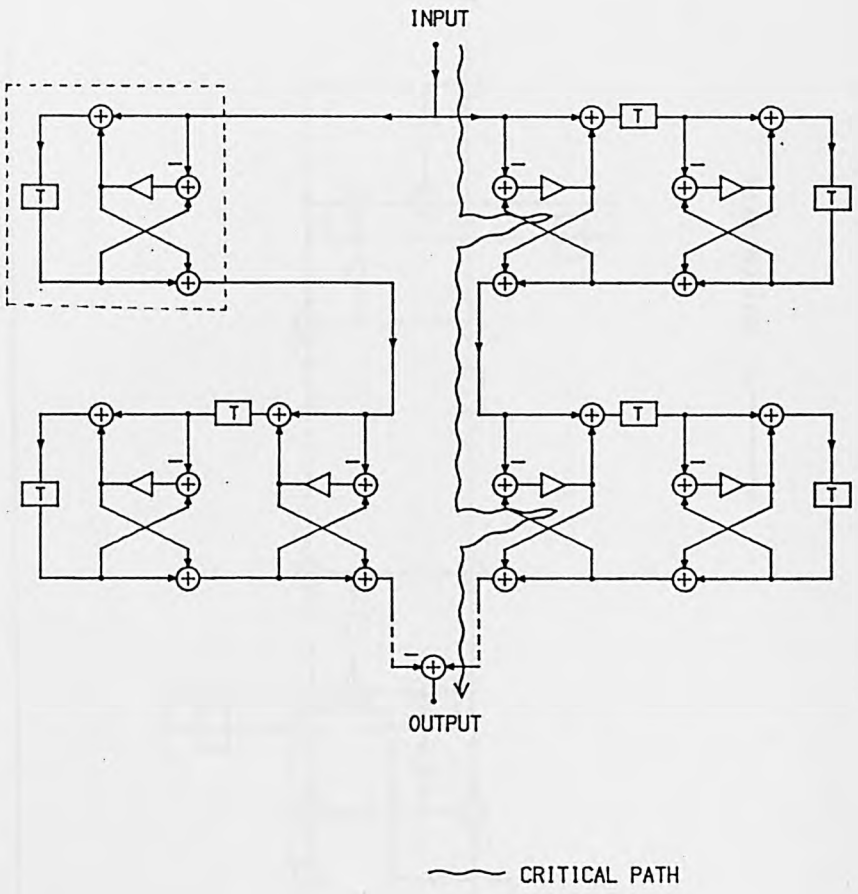


FIGURE 3.6 : LATTICE WDF BASED ON ALL-PASS SECTIONS

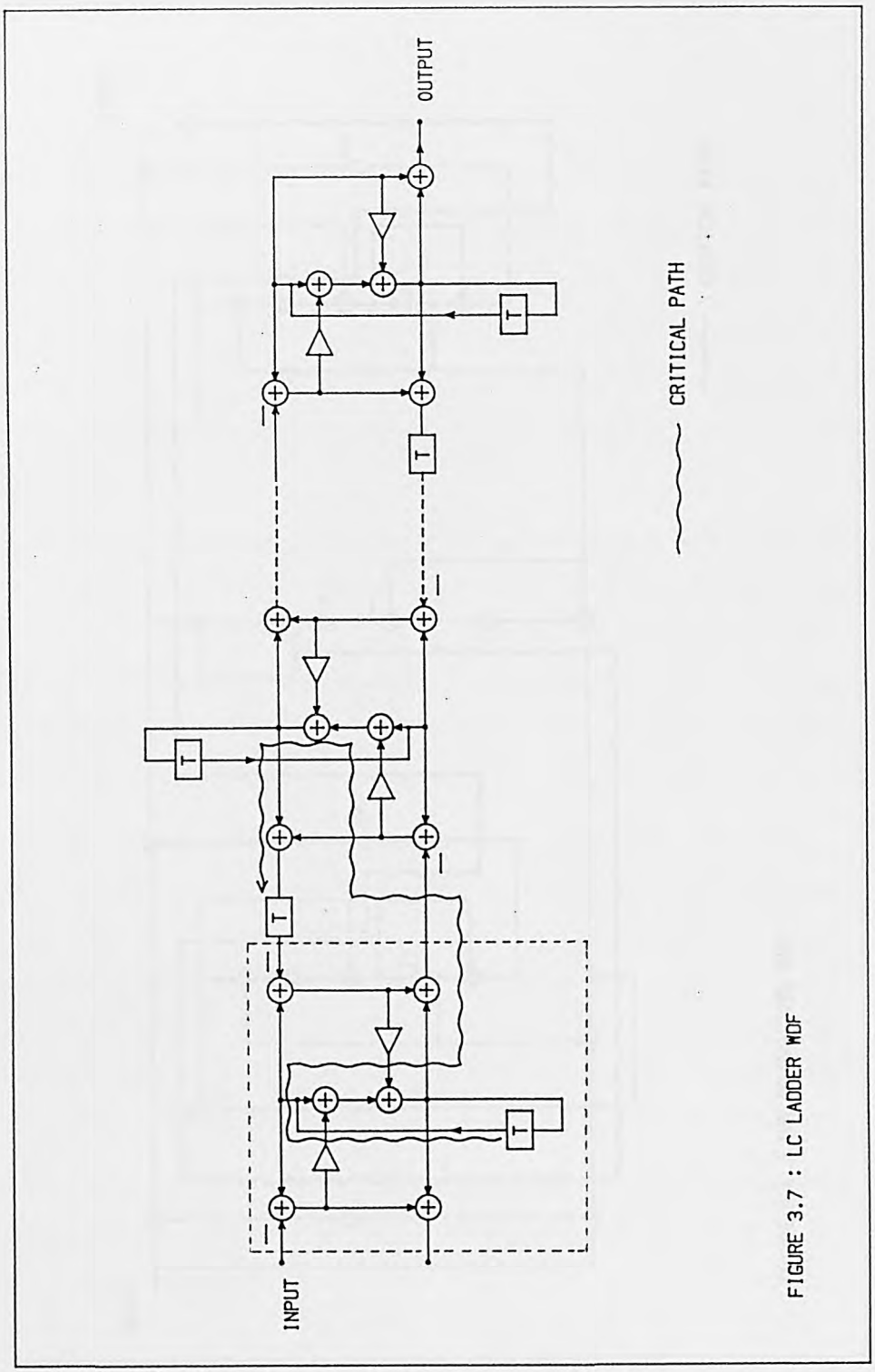


FIGURE 3.7 : LC LADDER WDF

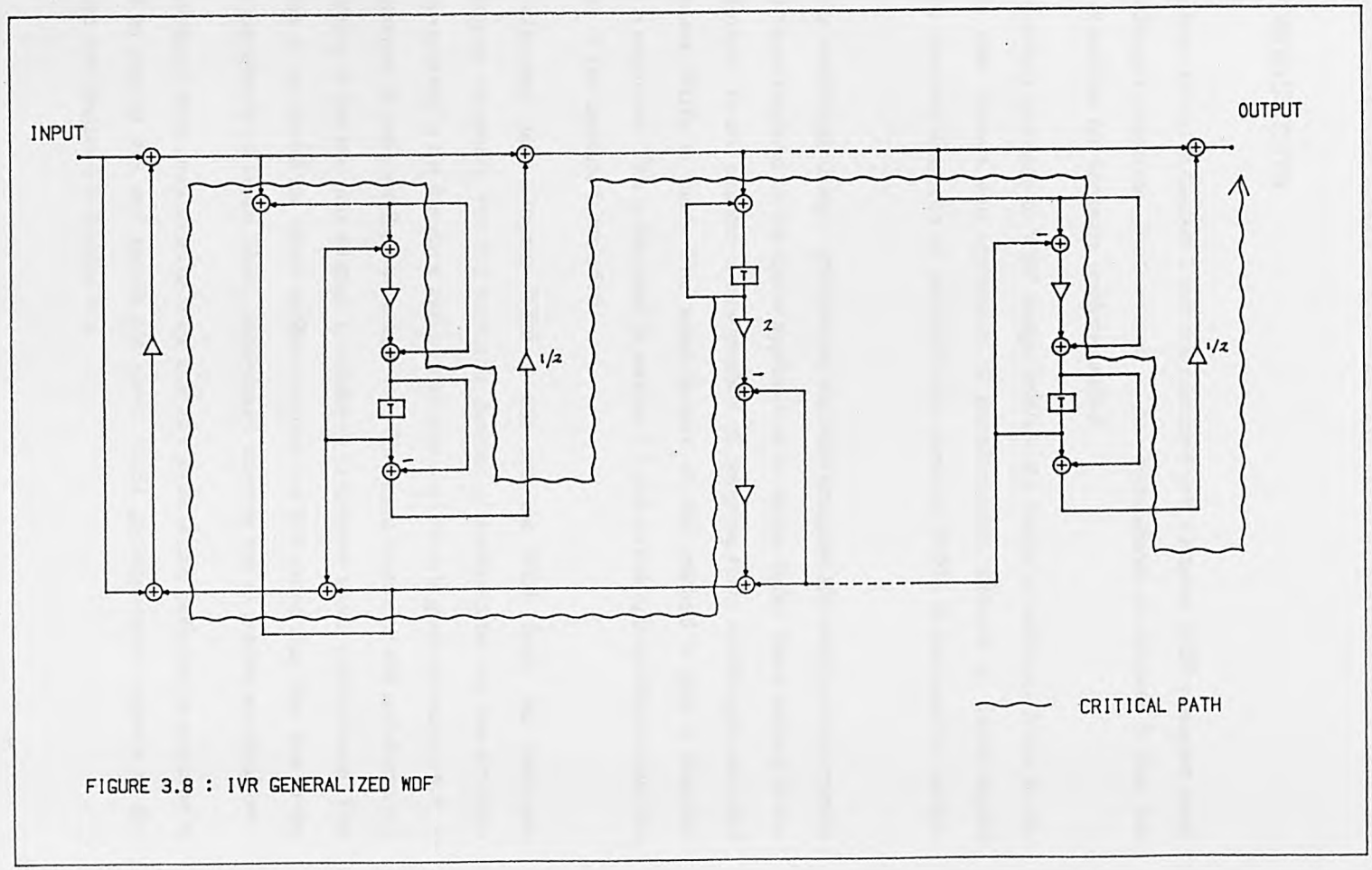


FIGURE 3.8 : IVR GENERALIZED WDF

CHAPTER FOUR

THE CASCADED UNIT ELEMENTS WAVE DIGITAL FILTER

4.1 INTRODUCTION

It has been shown in chapter 2 that the cascaded unit elements WDF possesses good finite wordlength properties. Furthermore, it has been shown in chapter 3 that this structure is suitable for hardware implementation.

The standard approach to WDF design involves the design of reference filters in the continuous time domain and application of transformation methods to derive digital filters. The theoretical aspects of cascaded unit elements WDFs is discussed in section 4.2.

In finite wordlength design, optimization methods are generally employed to improve the digital filters obtained by the above approach or to design digital filters entirely in the discrete domain. In this chapter, a new method of designing finite wordlength cascaded unit elements WDFs is given. The novel feature of this method is that a heuristic approach is employed. This is discussed in section 4.3 and an example to demonstrate the workability of the concept is included.

The cascaded unit elements WDF is the simplest WDF from the hardware implementation viewpoint. For this reason, a number of architectures for this structure have been reported in the literature and a brief review of these is given in section 4.4. A new architecture is also described in section 4.4. The novel feature of this architecture is that pipelining of the two port adaptor is employed to improve speed performance. This architecture is not useful for direct implementations but it is noted that that due to the hardware complexity of digital filters, multiplexed schemes are of greater practical use.

A functional level simulation to show that the architecture performs as required is described in section 4.5 and results are given. VLSI implementation aspects of the architecture are discussed in section 4.6.

4.2 THEORETICAL ASPECTS

4.2.1 THE UNIT ELEMENT

The impedance matrix of a two port network comprising a commensurate length transmission line (see figure 4.1) is given by [2]:

$$\begin{bmatrix} V1 \\ V2 \end{bmatrix} = \begin{bmatrix} -j.Zo.cot\theta & -j.Zo.cosec\theta \\ -j.Zo.cosec\theta & -j.Zo.cot\theta \end{bmatrix} \begin{bmatrix} I1 \\ I2 \end{bmatrix} \qquad (4.1)$$

where Zo is the characteristic impedance of the transmission line.

$\theta = \Omega.l/v$ = *electrical length*

where Ω = *angular frequency in radians per second*

l = *length of shortest commensurate line*

v = *velocity of propagation*

Application of Richards' transformation [58] to the above impedance matrix results in the following impedance matrix:

$$\begin{bmatrix} V1 \\ V2 \end{bmatrix} = Zo/S \cdot \begin{bmatrix} 1 & \sqrt{1-S^2} \\ \sqrt{1-S^2} & 1 \end{bmatrix} \begin{bmatrix} I1 \\ I2 \end{bmatrix} \qquad (4.2)$$

where S is the Richards complex frequency variable.

The two port network element described by this impedance matrix in the S plane is defined as the 'unit element' according to Ozaki and Ishii [59]. In other words, the unit element is the S plane equivalent of a two port network comprising a commensurate length transmission line.

4.2.2 DISTRIBUTED STEPPED IMPEDANCE LOW PASS FILTERS

It is well known from microwave theory that filters can be obtained by cascading transmission lines of commensurate length and different characteristic impedances (see figure 4.2).

Levy [60] has shown that it is possible to select the characteristic impedance of each commensurate line segment in the cascade so as to produce the following low pass Chebyshev magnitude function:

$$|G|^2 = \frac{1}{1 + h^2 . Tn^2 . (\sin\theta/\sin\theta c)} \qquad (4.3)$$

where Tn denotes the Chebyshev function of the first kind of degree n, θ is the electrical length as defined earlier, and θc is the cutoff parameter.

A typical response is shown in figure 4.3 from which it can be seen that the magnitude is periodic with a period of π radians.

Tables of characteristic impedance values required to design distributed low pass filters for a range of bandwidths, passband ripples and stopband losses are given in reference [60].

4.2.3 CASCADED UNIT ELEMENTS WAVE DIGITAL FILTERS

Digital filters can be derived from distributed filters by using the techniques of Fettweis [45]. The digital filter derived in this manner from cascaded transmission line filters are generally referred to as cascaded unit elements WDFs [45] and the structure is shown in figure 4.4. The abbreviation UEWDF will be used to denote cascaded unit elements WDF from hereon.

The relationship between the real frequency variables in the continuous and discrete time domains is as follows [61]:

$$\theta \rightarrow 1/2.\omega.T \qquad (4.4)$$

where ω is the angular frequency in the discrete time domain, θ is the electrical length as defined earlier, and T is the sampling period.

The parameters required to design distributed Chebyshev low pass filters from Levy's tables are; the bandwidth (BW), the ripple voltage standing wave ratio (VSWR), the

order n and the stopband loss. BW and VSWR are related to the digital filter parameters as follows [61]:

$$BW = 4Fo/Fs \qquad (4.5a)$$

where Fo is the cutoff frequency and Fs is the sampling frequency.

$$VSWR = \frac{1+2\sqrt{e^2-e}}{2e-1} \qquad (4.5b)$$

$\eta = 10.log10(e)$ = *passband ripple in dB*

To design a low pass digital filter with Levy's tables then, we first determine the BW and VSWR as shown above and then choose the order n to give the required minimum stopband loss (Lmin). A formula for estimating the value of n is as follows [62]:

$$n > \frac{log(10^{Lmin/10}-1) - log(e-1) + log4}{2.log(2.sin\theta i/sin\theta c)} \qquad (4.6)$$

where $\theta i = \pi.fi/fs$, *fs is the sampling frequency and fi is the stopband edge frequency.*

The digital filter coefficients (see figure 4.4) are then determined from the characteristic impedances (see figure 4.2) of the distributed filter as follows [45]:

$$\alpha k = \frac{Zk-1 - Zk}{Zk-1 + Zk}, \qquad Z0 = Rs, \quad Zn+1 = Rl \qquad (4.7)$$

Rs is the source resistance, Rl is the load resistance and n is the filter order.

Note from figures 4.2 and 4.4 that the transmission line segments are transformed into a pair of T/2 second delays and the interconnections are transformed into two port adaptors [45].

4.3 A DESIGN PROGRAM BASED ON A HEURISTIC APPROACH

In this section, a program for the finite wordlength design of UEWDFs is described. The novel feature of the program is that a heuristic approach is taken which seeks to exploit a number of observations that are made from conventional designs using Levy's tables as described in section 4.2.

4.3.1 CONVENTIONAL APPROACHES TO FINITE WORDLENGTH DESIGN

It has been shown in section 4.2 that low pass UEWDFs can be designed using Levy's tables. For implementation purposes, the multiplier coefficients αk (figure 4.4) must be quantized according to the number of bits that are available. This will result in a deviation in the frequency response as discussed in chapter 2.

For any digital filter, it is possible that slightly different quantized values for multiplier coefficients will lead to a better frequency response. This forms the basis of one approach to finite wordlength design in which a number of processes are involved. First, the ideal multiplier coefficients are obtained by using an appropriate analytical method. Next, the coefficients are quantized and the corresponding frequency response is obtained. This is followed by a process which varies each coefficient value within a narrow range of its initial value and determines the set of coefficients which gives the best response. These processes form the algorithm for the finite wordlength design program.

The availability of a finite number of bits to represent multiplier coefficients means that there is a finite set of values that the coefficients can take. Therefore, it is possible to design a filter entirely in the discrete time domain by examining all possible combinations and determining one that satisfies the frequency response requirements. The number of possible combinations is dependent on the number of bits and the order of the filter. For most practical designs, such an exhaustive search will require large amounts of computer time and is clearly inefficient. In practice, use is made of optimization algorithms to reduce the search space and increase program efficiency [63–69].

Since it is desirable to have a program that implements the entire design process and because of the difficulties in synthesizing cascaded transmission line filters [42,62], the

second approach is preferred for UEWDF design. The design program detailed in this thesis is based on the second approach but does not employ an optimization algorithm.

4.3.2 OBSERVATIONS ON LOW PASS UEWDFs

(a) It is useful, for obvious reasons, to know whether low pass UEWDFs can be designed using only coefficients which are powers of 1/2. In order to answer this question, an exhaustive search was carried out for all combinations of the following coefficient values ; $\pm 1/2$, $\pm 1/4$, $\pm 1/8$, $\pm 1/16$, $\pm 1/32$.

. It was found that for a third order UEWDF, the only combination that resulted in a reasonable low pass response was ; $\alpha 1=-0.25$, $\alpha 2=0.5$, $\alpha 3=-0.5$ $\alpha 4=0.25$.

. In this case the passband ripple was 0.02dB, passband edge was 0.15Fs (Fs is the sampling frequency) and the maximum stopband loss was 8dB.

. For a fifth order UEWDF, it was found that the only combination which resulted in a reasonable low pass response was ; $\alpha 1=-0.25$, $\alpha 2=0.5$, $\alpha 3=-0.5$, $\alpha 4=0.5$, $\alpha 5=-0.5$, $\alpha 6=0.25$.

. In this case the maximum gain was −0.4dB, the passband ripple was 0.8dB, the passband edge was 0.26Fs and the maximum stopband loss was 16dB.

. From the above results, it is concluded that no useful low pass UEWDFs can be designed using only coefficients that are powers of 1/2.

(b) Only UEWDFs which have corresponding cascaded transmission line filters are of interest [45]. Since the characteristic impedances can never be negative, it follows from the relationship between the UEWDF coefficients αk and the characteristic impedances Zk (see section 4.2.3) that αk can only take on values between +1 and −1.

(c) For UEWDFs corresponding to Levy's filters [60], it is known from theory that for all odd order filters, the absolute values of the coefficients are symmetrical about the centre. It was also noted that in all odd order cases, as

we travel along the filter from left to right (see figure 4.4), the sign of the coefficient is negative and the sign alternates for subsequent coefficients, i.e. for an nth order filter, $\alpha k = \alpha n+1-k$ where n is odd and $\alpha 1$is always negative.

. Furthermore, the absolute values of the coefficients always increase towards the centre of the filter. If the values of the coefficients are plotted as shown in figure 4.5, then it is noted that the 'coefficient curve' is always smooth and convex with the exception of the first coefficient in broad band filters.

. The coefficient curves in figure 4.5 are for 7th, 11th and 13th order filters. Viewed from top to bottom, the coefficient curves give UEWDFs of increasing bandwidth.

(d) In practice, it has been found that the optimum low pass UEWDF in terms of frequency response is obtained when the conditions in (c) are satisfied, i.e. the quantized coefficients are related as shown in (c).

. This view is further supported by the results shown in reference [42]. This is important because the method used is not based on Levy's tables but on an optimization procedure.

(e) It is noted from [60] that when n is large, the few central VSWRs are all nearly equal. This means that for the corresponding UEWDFs, the central coefficients are nearly equal.

(f) For all fifth order UEWDFs derived from Levy's filters [60], the coefficient curves were plotted. It was noted that as the stopband loss increases, the coefficient curve occupies a higher region of the graph (see figure 4.6a). This was found to be the case for all odd order UEWDFs derived from Levy's filters.

. It was also noted that as the bandwidth increases, the coefficient curve occupies a lower region of the graph (see figure 4.6b).

4.3.3 PROGRAM OUTLINE

A program to design low pass UEWDFs in the discrete domain can be written using a direct search technique. This would involve searching the entire discrete space for a

solution, i.e. we would have to consider every possible value of $\alpha 1$, $\alpha 2$, $\alpha 3$, etc. For m bit coefficients, this would involve 2^m possible values for each coefficient. For an (n–1)th order UEWDF, the number of possible coefficient sets would then be $(2^m)^n$.

However, if we make use of the observation that the absolute values of the coefficients are symmetrical, the signs of the coefficients alternate and the sign of the first coefficient is always negative, then the number of possible coefficient sets reduces to $(2^{m-1})^{n/2}$. In most practical cases, this value would still be too large to be useful.

We can constrain the search so that only coefficients which lie on a smooth convex coefficient curve (see section 4.2.2(c)) are investigated. On the basis of this, a finite wordlength design program can be written which investigates all curves of this kind in the discrete space until one that satisfies the specification is found.

Taking note of the observations in section 4.2.2(f), we can extract a lower limit to the search area from Levy's tables according to the stopband loss requirement. Similarly, a higher limit may be set according to the bandwidth requirements.

Also, for higher order filters, we can make use of the observation in section 4.2.2(e) by limiting the range of values that the central coefficients can take.

4.3.4 EXAMPLE PROGRAM

A program for the finite wordlength design of low pass UEWDFs with 8 bit coefficients has been designed and implemented. The program is based on the ideas of section 4.3.3 and is shown in appendix C1.

The program receives as input the tolerance scheme parameters ; passband ripple, passband edge frequency, stopband loss and stopband edge frequency. The program first estimates the required order according to the formula given in section 4.2.3 and if the order is even the next higher order is selected. It then extracts the upper and lower limits for the coefficients from Levy's tables. Note that the relevant data is kept in a file called range from which these limits are obtained. The data in the file range for fifth and seventh order UEWDFs is shown in the appendix C1. Note that the lowest values of

coefficient curves for 10dB, 20dB, 30dB,... stopband losses are given. Similarly, the highest values of coefficient curves for 0.05Fs, 0.1Fs,...0.35Fs (Fs is the sampling frequency) passband widths are given.

The main part of the program is concerned with obtaining coefficient values that lie on a curve as discussed earlier and determining whether the specification is met*. For each set of coefficients, the frequency response is evaluated and if it fits the tolerance scheme, the coefficient set is reported. If after examining all possibilities within the limits set above, failure occurs, then the order of the filter is increased by 2 and the whole process is repeated.

As an example, consider the design of a low pass UEWDF which satisfies the following arbitrarily tolerance scheme :

passband edge = 0.1 Fs

passband ripple = 0.1 dB

stopband edge = 0.2 Fs

stopband loss = 60 dB

For this specification, the above program gave the following results :

order = 9

$\alpha 1 = -0.401574 = -\alpha 10$

$\alpha 2 = 0.803148 = -\alpha 9$

$\alpha 3 = -0.913384 = -\alpha 8$

$\alpha 4 = 0.929132 = -\alpha 7$

$\alpha 5 = 0.9370061 = -\alpha 6$

The UEWDF with the above coefficient values was simulated using DIFSIP and it was noted that the amplitude response satisfied the tolerance scheme requirements.

* *for a more complete explanation of the manner in which the coefficients that lie on a convex curve are determined, see appendix C1.*

The run time for this process was 10 minutes of CPU time on a Gould supermini computer. Obviously, this value would be lower for coefficient wordlengths lower than 8 bits and vice versa.

It would be useful to investigate more efficient programming methods in order to reduce the computer time requirements. However, it is clear from above that in principle, the method is of practical value. Furthermore, since this method does not involve any optimization algorithms, elements of it may be used to improve the efficiency of programs based on optimization algorithms. Combinations of the two techniques should be investigated and this is left as an exercise in further work.

4.4 AN ARCHITECTURE FOR THE CASCADED UNIT ELEMENT WDF

The UEWDF is the simplest WDF from the implementation viewpoint. Therefore, it has been widely studied and a number of architectures have been reported in the literature.

In this section, a brief review of the architectures reported in the literature will be given and a new architecture which exploits the inherent parallelism of the UEWDF will be described.

4.4.1 REVIEW

As discussed earlier, the application of Fettweis' techniques to the cascaded transmission lines filter results in the UEWDF of figure 4.4. It can be shown [45] that the pair of half delays corresponding to the transmission line segment can be replaced by a single delay. This delay can be placed in either the upper or lower arms. On the basis of this, two alternative representations of the UEWDF can be obtained as shown as shown in figures 4.7c and 4.7d. The only difference in terms of frequency response between these two structures and the structure of figure 4.7b (note that this is the same as figure 4.4) is a constant delay in the group delay.

Clearly, the structures of figures 4.7c and 4.7d will require less hardware than the structure of figure 4.7b. Therefore, some authors have chosen to design architectures based on these structures. Of these two, it is important to note that multiplexability is only possible when all the delays are either in the upper or lower arms (figure 4.7c). On the other hand, for direct implementations, the structure of figure 4.7d is more suitable because the critical path is then equal to the delay through two adaptors and is independent of the filter order. Therefore, the maximum sampling frequency possible will be higher with the structure of figure 4.7d than with that of figure 4.7c.

The advantage of employing the structure of figure 4.7b is that although it requires more hardware than the other two, it is multiplexable and gives optimum speed performance.

An architecture based on the structure of figure 4.7c has been presented by Lawson [70,71]. This employs a single multiplexed two port adaptor and an implementation of

the architecture with discrete ICs is given in [72]. An architecture suitable for VLSI implementation of this structure and employing systolic arrays has been presented by Mirzai [42]. Sikstrom [73] has proposed an architecture for the LSI implementation of the structure of figure 4.7d and has designed a basic substructure (comprising two adaptors and two delays) which may be cascaded to realize higher order filters.

The structure of figure 4.7b has been used by Cardoso [74] as a basis for an architecture which has the additional feature of being multiplierless. This is achieved by constraining the range of coefficients that can be realized to simple combinations of powers of 1/2. Bearing in mind the discussion of section 4.3.2, it is clear that such an approach will have limited application.

4.4.2 A NEW ARCHITECTURE

A direct implementation of the structure of figure 4.4 would entail a direct mapping of the structure onto hardware. That is, each multiplier, adder/subtractor and delay would be implemented with hardware equivalents. The resulting hardware scheme would then be clocked at twice the sampling frequency where the clock frequency would be limited by the delay through the two port adaptor. Direct implementation requires large amounts of hardware and therefore, multiplexed systems are more useful

The structure of figure 4.4 can be implemented with a single multiplexed two port adaptor. For the purposes of illustration, the structure can be redrawn in the form shown in figure 4.8.

For each two port adaptor in figure 4.8b, we need to compute the values B and D (see figure 4.8a) for the current values of A and C. At the beginning of a half sampling period, the previous values of B and D for a particular adaptor become the current values of A and C for the succeeding adaptor. This is the case for all except the first and last adaptors where the value of A in the first adaptor and C in the last adaptor are the current values of the inputs (normally the input at the last adaptor is set to zero). The current values of B and D for the first and last adaptors give us the two outputs as shown. The sampling frequency in this case is equal to 2n+2 clock cycles (n is the filter order) and the maximum clock frequency is limited by the delay through the two port adaptor.

Note from figure 4.4 that because of the position of the delays, the two port adaptors would operate in parallel for a direct implementation. This fact can be used to advantage in the multiplexed system by pipelining the two port adaptor as shown in figure 4.8b where the dotted lines indicate the breakdown of the pipeline. This means that when the computation process for adaptor 1 (figure 4.8b) progresses to the second stage of the pipeline, the first stage of the pipeline can operate on adaptor 2, and on the next clock cycle the adaptor 1 process progresses to stage 3 of the pipeline, adaptor 2 to stage 2, adaptor 3 to stage 1 and so on. Effectively, the sampling frequency is equal to (2n+2)+2 clock cycles where the clock frequency is limited by the pipeline processes. This clock frequency will obviously be higher than the previous one and consequently the sampling frequency will be higher.

Note that for every adaptor in figure 4.8b, the values B and D can be stored in the same registers as the values A and C since these are no longer required. This process is repeated n+1 times for an nth order UEWDF. It is then necessary to carry out appropriate adjustments so that all A and C values for the next half sampling interval reside in the correct registers. For example, during a half sampling interval, the B and D values for adaptor 2 are evaluated and placed in the same registers as the A and C values for adaptor 2. However, for the next half sampling interval, the B value for adaptor 2 becomes the A value for adaptor 3 and the D value for for adaptor 2 becomes the C value for adaptor 1. This means that a lower sampling frequency than previously suggested will be obtained. This aspect will be discussed further at a later stage.

We will define the processing cycle as the set of processes required by the multiplexed system to compute the response to a single input value. Clearly, there will be two processing cycles per sampling interval and on the second cycle, both input values must be zero.

An architecture based on the above ideas has been designed (see figure 4.9). The two port adaptor implementation comprises a subtractor, multiplier array, two adders, two pipeline registers and two sets of double register stacks (A1,A2 and C1,C2). Note the positions of the pipeline registers in figure 4.9 with respect to the positions indicated in

figure 4.8a. The purpose of the double register stacks is to store the A and C values for use in the third stage of the pipeline.

The delays are implemented with two register stacks and multiplexers are included to enable input values to be fed into the system. The coefficient RAM comprises a circulatory set of registers with two dummy registers. The control system supplies all the necessary clock and control signals and the architecture can be configured for different orders. This is done by making some of the registers in the stacks and RAM transparent.

An important feature of this architecture is that a number of these systems may be used in parallel to achieve higher sampling rates. For example, a seventh order UEWDF may be implemented with two multiplexed systems each of which implements four stages of the UEWDF. The maximum sampling frequency possible will then be twice that of a scheme comprising a single multiplexed system.

SYSTEM OPERATION

Assume that the architecture has been configured for a given order and appropriate coefficient values have been loaded into the RAM. Then the arrival of a sample ready signal (sample clock edge in other words) will trigger the control system which supplies appropriate clock and control signals to complete two processing cycles. This means that the sample clock is independent of the system clock.

For an nth order UEWDF, the number of clock cycles required per sample is $2((n+1)+3)$. The maximum clock frequency possible is therefore $SYSTEM\ CLOCK\ FREQUENCY/(2n+8)$.

CLOCKING AND CONTROL ASPECTS (see figure 4.10)

The system is synchronous and all clock and control signals are derived from the same clock signal (CLK). The frame signal (FRAME) defines the window during which system operation will take place within one sampling interval. The frame width is dependent upon the filter order (frame width = 2n+8 clock cycles where n is the filter order) and this signal is triggered by the sample ready signal (SIGCLK). The system clock

signal (CLK) is not dependent on the sample ready signal and therefore external synchronization problems are avoided. Furthermore, the system may be designed to maximize clock frequency and once set, this clock frequency can be used for all sampling frequencies within the limit.

The frame signal is used to derive the two half frame signals (HALFRM1 and HALFRM2) which define the durations of the two processing cycles that must be completed in one sampling interval. The half frame width is equal to n+4 clock cycles.

The control signals to the multiplexer MUX1 (see figure 4.9) are used to select either input 1 (SIG) or zero or the output of register stack 2. During the first clock cycle of the first processing cycle, input 1 is selected (M11) and similarly for the second processing cycle, zero is selected (M12). At all other times, the output of register stack 2 is selected.

The control signals to the multiplexer MUX2 (see figure 4.9) are used to select either input 2 or zero or the output of register stack 1. During the (n+1)th clock cycle of the first processing cycle, input 2 is selected (M21) and similarly for the second processing cycle, zero is selected (M22). Since input 2 is normally set to zero, a third control signal (M23) is defined which selects zero. At all other times, the output of register stack 1 is selected.

The clock signals required by the subsystems in figure 4.9 are illustrated in figure 4.10b. Note that register stack 1 requires an additional clock edge (CLK2) per processing cycle compared with register stack 2 (CLK1). This additional shift in stack 1 is necessary to place the A and C values in the correct registers as discussed earlier.

Note that the clock signals for the output registers differ for odd and even order filters. The position of the clock pulse within the processing cycle remains unchanged but for even order filters, the output registers are clocked in the first processing cycle whereas for odd order, the registers are clocked in the second processing cycle.

RECONFIGURABILITY AND PROGRAMMABILITY

We may configure the architecture for different order UEWDFs by making some of the registers in the stacks and coefficient RAM transparent. In practice, we may choose to provide individual control lines in order to facilitate transparency control.

For a given order, the architecture may be programmed for different UEWDFs by loading appropriate coefficients into the RAM.

4.5 SIMULATION OF ARCHITECTURE AT FUNCTIONAL LEVEL

FUNCTIONAL LEVEL SIMULATION WITH SEQUENTIAL PROGRAMMING LANGUAGES

Synchronous systems can be modelled with registers and combinational functions (addition, multiplication, etc – i.e. operations which do not involve memory). Such systems can be simulated at the functional level with sequential programming languages.

In modelling a synchronous system with a sequential language, all registers are modelled with real variables and the assignment of values to the variables models the process of clocking a register. The combinational functions are modelled with expressions in terms of the above variables and are evaluated in sequence after register clocking operations have been performed.

This approach is adopted here rather than the use of simulation programs such as HILO [75] because infinite signal wordlengths can be modelled. Furthermore, this approach is more flexible from the viewpoint of input and output signal manipulation.

SIMULATION OF ARCHITECTURE

In order to show that the architecture is functionally correct, a simulation at the functional level was performed with a program written in Fortran 77 (see appendix C2). For simplicity, the architecture was constrained to seventh order UEWDF realizations only.

In the simulation model used here, all registers shown in figure 4.9 are represented with real variables. This includes all stacks, pipeline registers, input/output registers and coefficient RAM.

The control system model used is based on shift registers and all control and clock signals are represented with boolean variables. The control shift registers are represented with boolean arrays. In a real system, shift registers may be employed to implement the control system. However, here we are only concerned with proving that the architecture is functionally correct.

In the program, the subroutine INIT resets all the registers and signal values. The subroutine SIGNAL provides the value of the input (SIG in figure 4.10a) and the signal ready signal (SIGCLK in figure 4.10a). The subroutine PAR performs a parallel load on the control shift registers which are used to generate the control signals. The exact manner in which this is done can be seen from the program. The subroutine DATPTH implements the shift function for the stacks, pipeline registers, input/output registers and coefficient RAM. Similarly, the subroutines FRM, MCON1 and MCON2 implement the shift functions for the control shift registers.

The manner in which the system operation is simulated can be seen from the executive routine. Note that the clock and control signals are derived from the system clock signal (CLK1 in program) and are the same as those shown in figures 4.10a and 4.10b. The simulation is carried out over 700 sampling intervals (this value is arbitrary) and the response to the input signal is reported.

For the coefficient values shown in the program, the impulse response of the system was obtained by simulation and, with the aid of an FFT routine, the frequency response was evaluated. This is shown in figure 4.11 and it can be seen that the system implements a low pass filter as expected.

4.6 VLSI IMPLEMENTATION ASPECTS

The architecture may be implemented with a full custom design. If we choose to implement the subsystems with the logic circuits discussed in chapter 3, then for a system with a 20 bit signal, 8 bit coefficients and a maximum filter order of 15, it is estimated that the gate count would be approximately 12,000 gates. Since gate counts of 100,000 gates are possible on a single VLSI chip, it is clear that the architecture presented would be useful for implementing digital filtering functions in VLSI systems. For higher sampling rates, a number of cells (where a cell is an implementation of the architecture of figure 4.9) may be used in cascade. If low sampling rates are required, then the gate count may be reduced by employing bit–serial techniques.

An important point to note in the architecture is the absence of bidirectional and multiplexed busses. This is a desirable feature from the IC implementation viewpoint because of the complexity in design of such systems.

In practice, a dynamic MOS technology would be a suitable choice because the system is clock driven. This would reduce the complexity of the registers. However, in the interests of speed, more complex systems for multipliers and adders/subtractors than those referred to would be used in practice.

The above design may be used as a macrocell in a standard cell system. One macrocell may then be used for implementing a fully multiplexed UEWDF and multiple cells may be cascaded to implement UEWDFs with higher sampling frequencies.

In the architecture of figure 4.9, the positions of the pipeline registers have been arbitrarily chosen Since the multiplier is the most complex element, the delay through it will determine the system clock frequency. In practice, it may be better to pipeline the multiplier as well or alternatively to use the shift add technique and separate the stages with pipeline registers. For example, we may choose to implement a multiplier with 8 bit coefficients as four sets of pairs of adders separated by pipeline registers. The maximum sampling frequency possible is limited by the system clock frequency divided by 2*(filter order + number of pipeline registers +2). Therefore, in practice we would choose the pipelining scheme that gives us the best sampling frequency.

4.7 DISCUSSION AND CONCLUSIONS

An important part of practical digital filter design is finite wordlength design. It is well known that a combination of analytic and optimization techniques or purely optimization techniques may be used to achieve this.

In this chapter, a number of properties of UEWDFs have been identified through experimentation and study of other researchers' work. Based on these properties, an alternative method for the finite wordlength design of UEWDFs employing a heuristic approach has been presented. An example to show that the technique works in practice has been given. Improvements to the program presented are possible by designing more efficient code.

This technique may be used to improve the efficiency of finite wordlength programs for UEWDFs which employ other methods. Optimization methods may also be combined with the above technique to improve its efficiency.

An architecture which exploits the inherent parallelism of the UEWDF in multiplexed systems has been described. The novel feature of this architecture is that the two port adaptor is pipelined in order to improve maximum sampling rates. A functional level simulation of the architecture has been given in order to show that it is conceptually correct.

VLSI implementation aspects have also been considered. It has been shown that approximately 12,000 gates are required for a fifteenth order system with 8 bit coefficients and 20 bit signals. This architecture may be implemented as a macrocell in a standard cell system allowing single cell implementation of a particular UEWDF or the use of a number of cascaded cells to implement the same UEWDF at a higher sampling rate. If sampling rate requirements permit, bit–serial techniques may be employed to reduce the gate count.

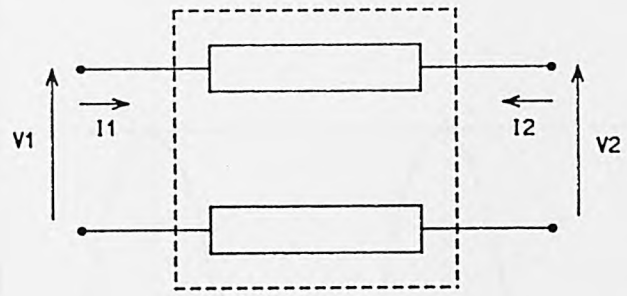


FIGURE 4.1 : TWO PORT NETWORK COMPRISING COMMENSURATE LENGTH TRANSMISSION LINE

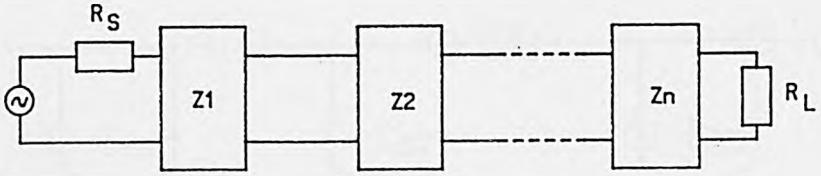


FIGURE 4.2 : CASCADED TRANSMISSION LINE FILTER

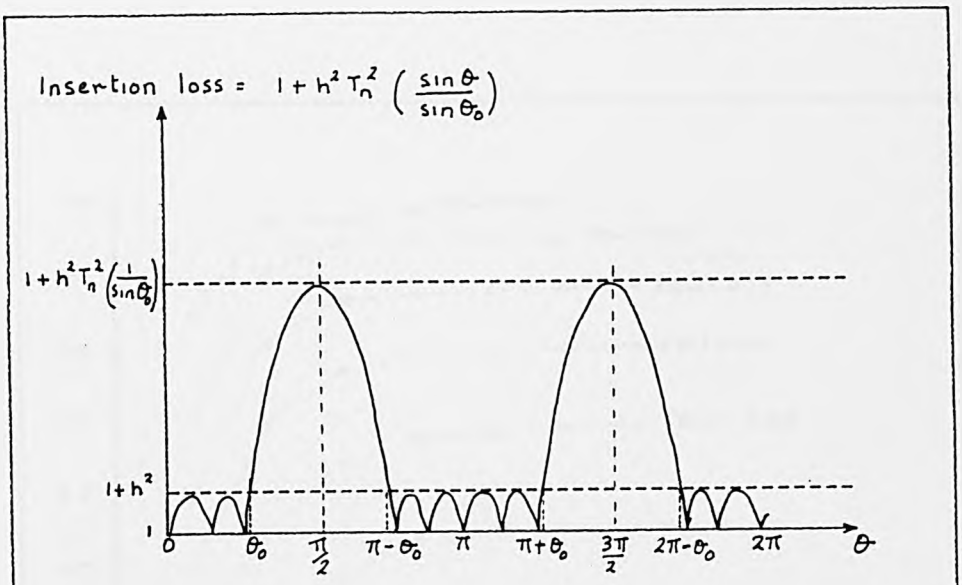


FIGURE 4.3 : FIFTH ORDER CHEBYSEHEV LOW PASS FILTER

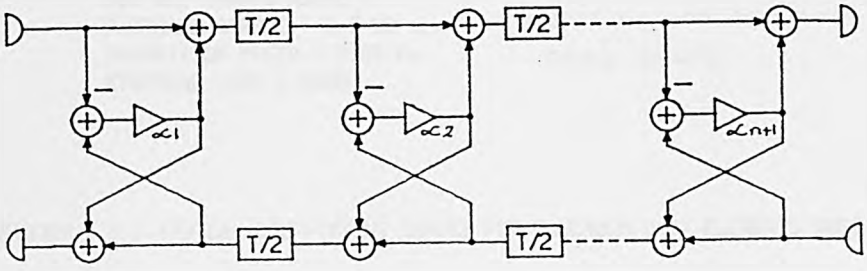
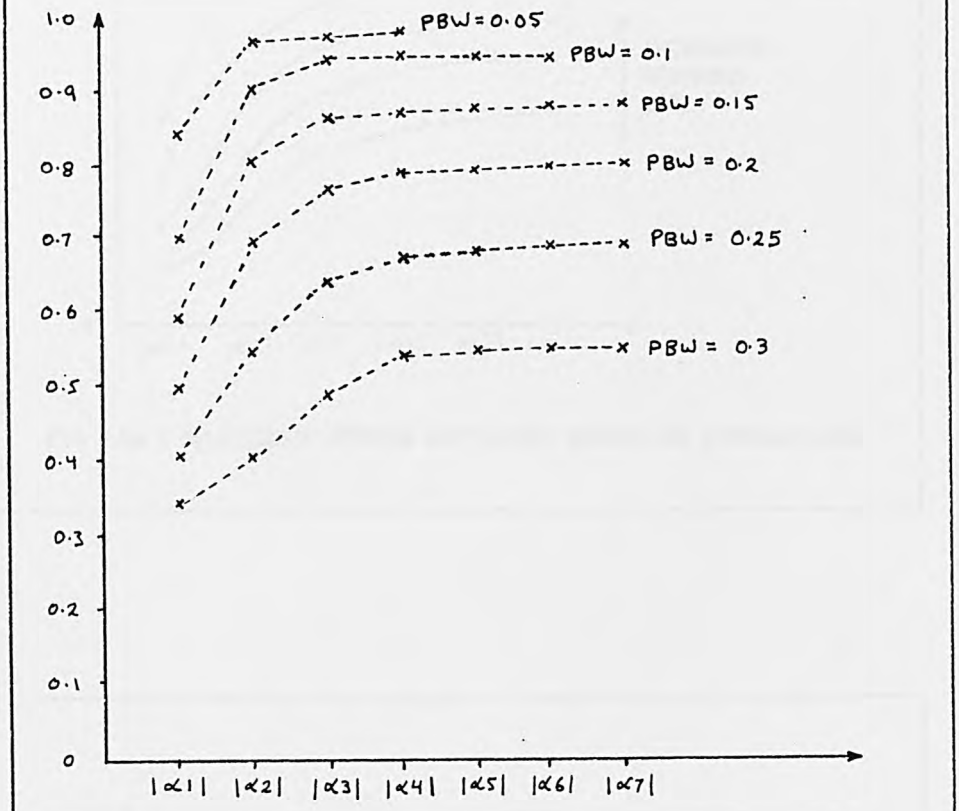


FIGURE 4.4 : CASCADED UNIT ELEMENTS WDF



FOR ALL UEWDFS ABOVE :
PASSBAND RIPPLE = 0.5 DB
TRANSITION WIDTH = 0.05 Fs
STOPBAND LOSS = 60DB

PBW: PASSBAND WIDTH (FRACTION OF SAMPLING FREQUENCY)

FIGURE 4.5 : TYPICAL COEFFICIENT CURVES FOR CASCADED UNIT ELEMENTS WDFS

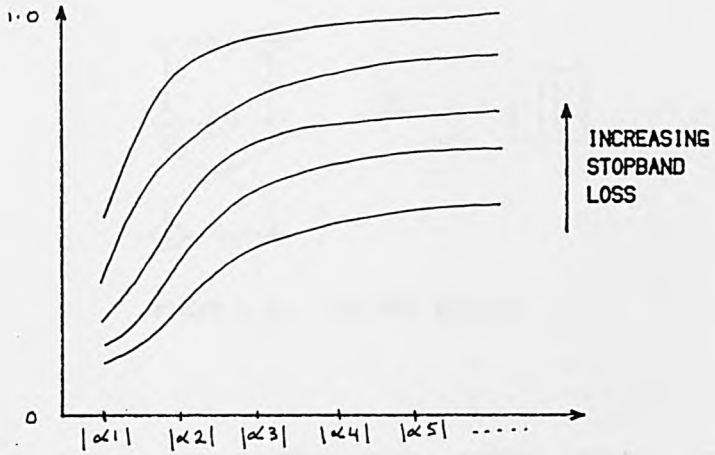


FIG 4.6a : RELATIONSIP BETWEEN COEFFICIENT CURVES AND STOPBAND LOSS

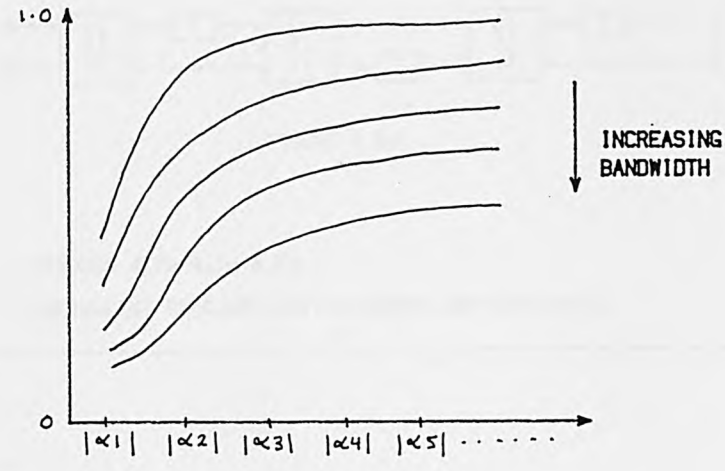


FIGURE 4.6b : RELATIONSHIP BETWEEN COEFFICIENT CURVES AND BANDWIDTH

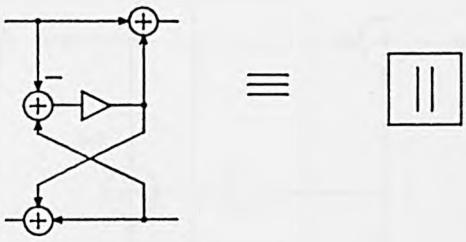


FIGURE 4.7a : TWO PORT ADAPTOR

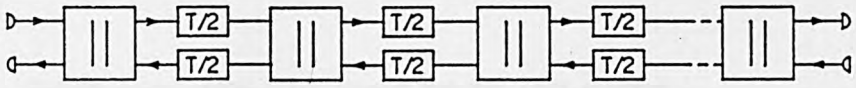


FIGURE 4.7b

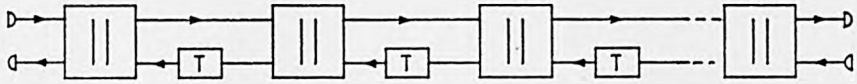


FIGURE 4.7c

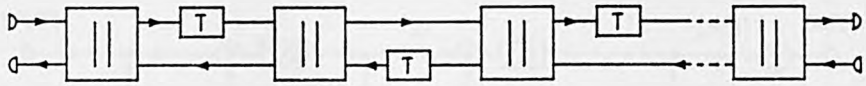


FIGURE 4.7d

FIGURES 4.7b 4.7c 4.7d :

EQUIVALENT CASCADED UNIT ELEMENTS WDF STRUCTURES

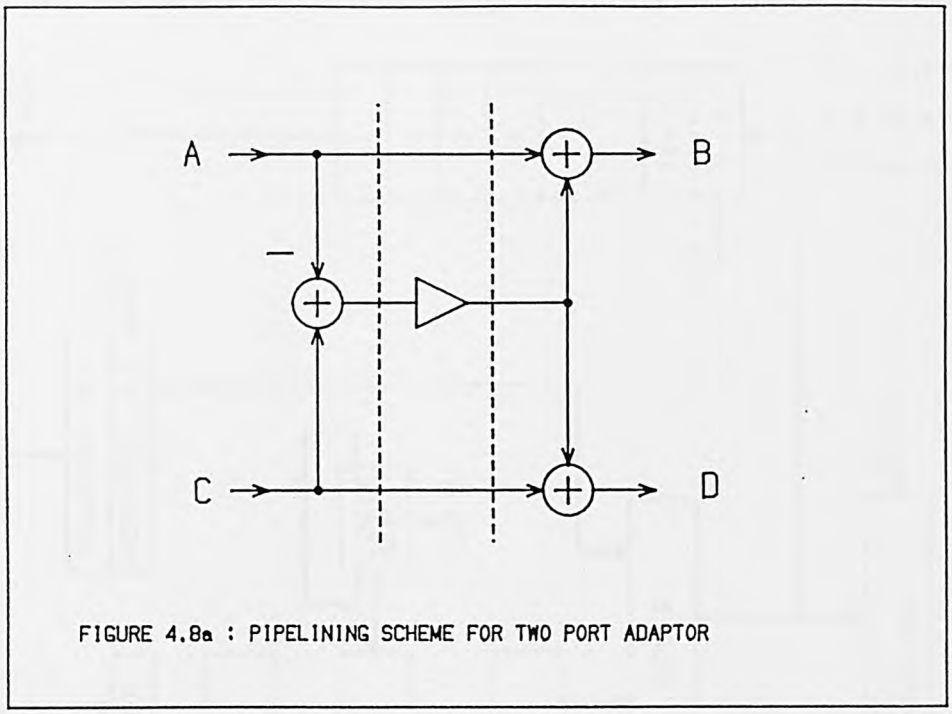


FIGURE 4.8a : PIPELINING SCHEME FOR TWO PORT ADAPTOR

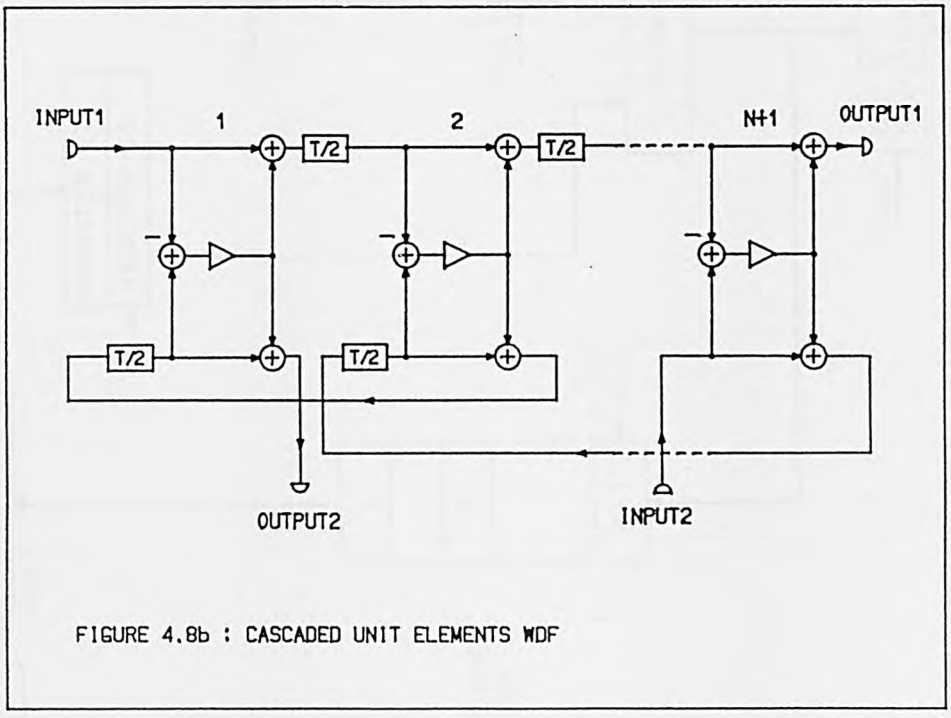


FIGURE 4.8b : CASCADED UNIT ELEMENTS WDF

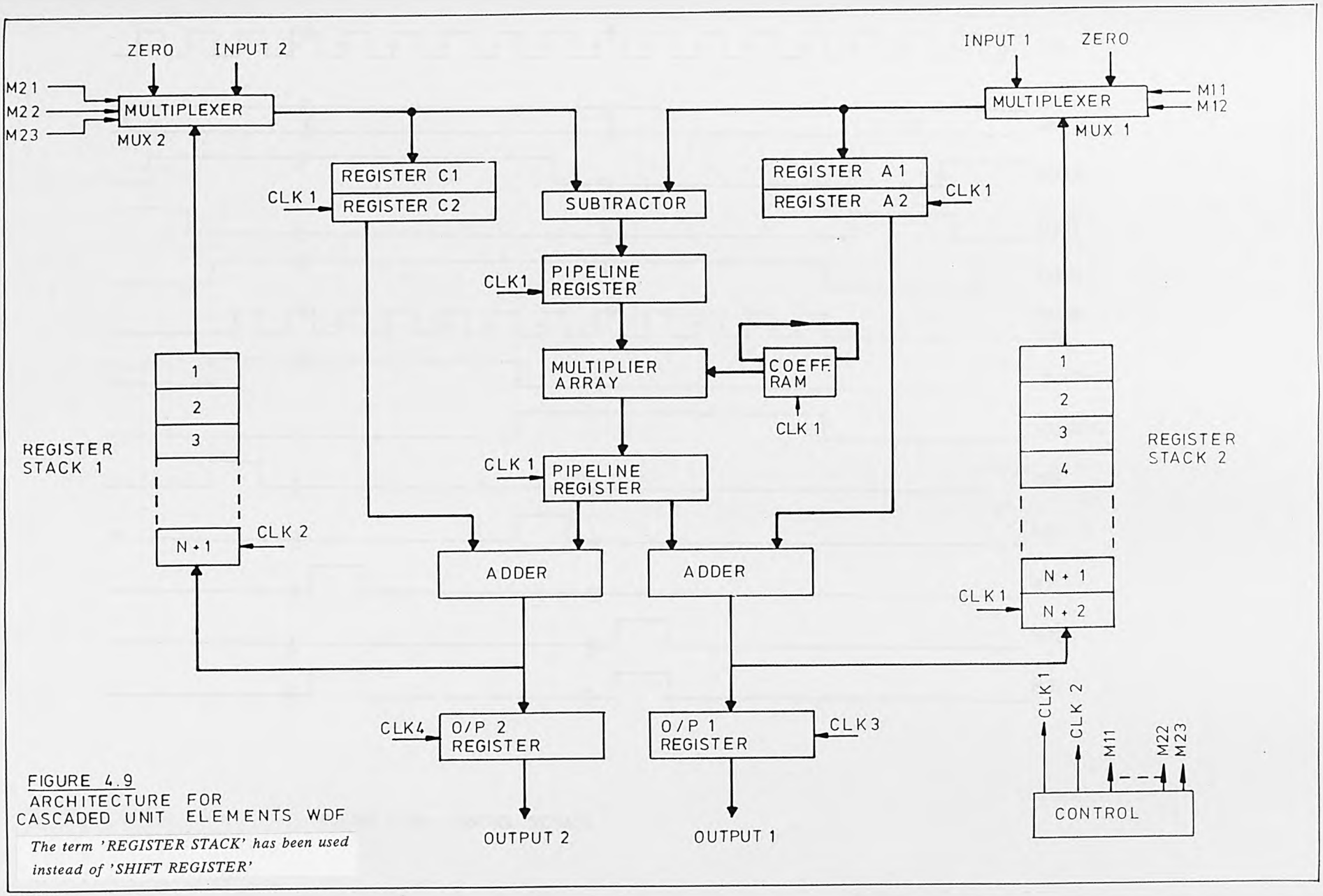


FIGURE 4.9
ARCHITECTURE FOR CASCADED UNIT ELEMENTS WDF

The term 'REGISTER STACK' has been used instead of 'SHIFT REGISTER'

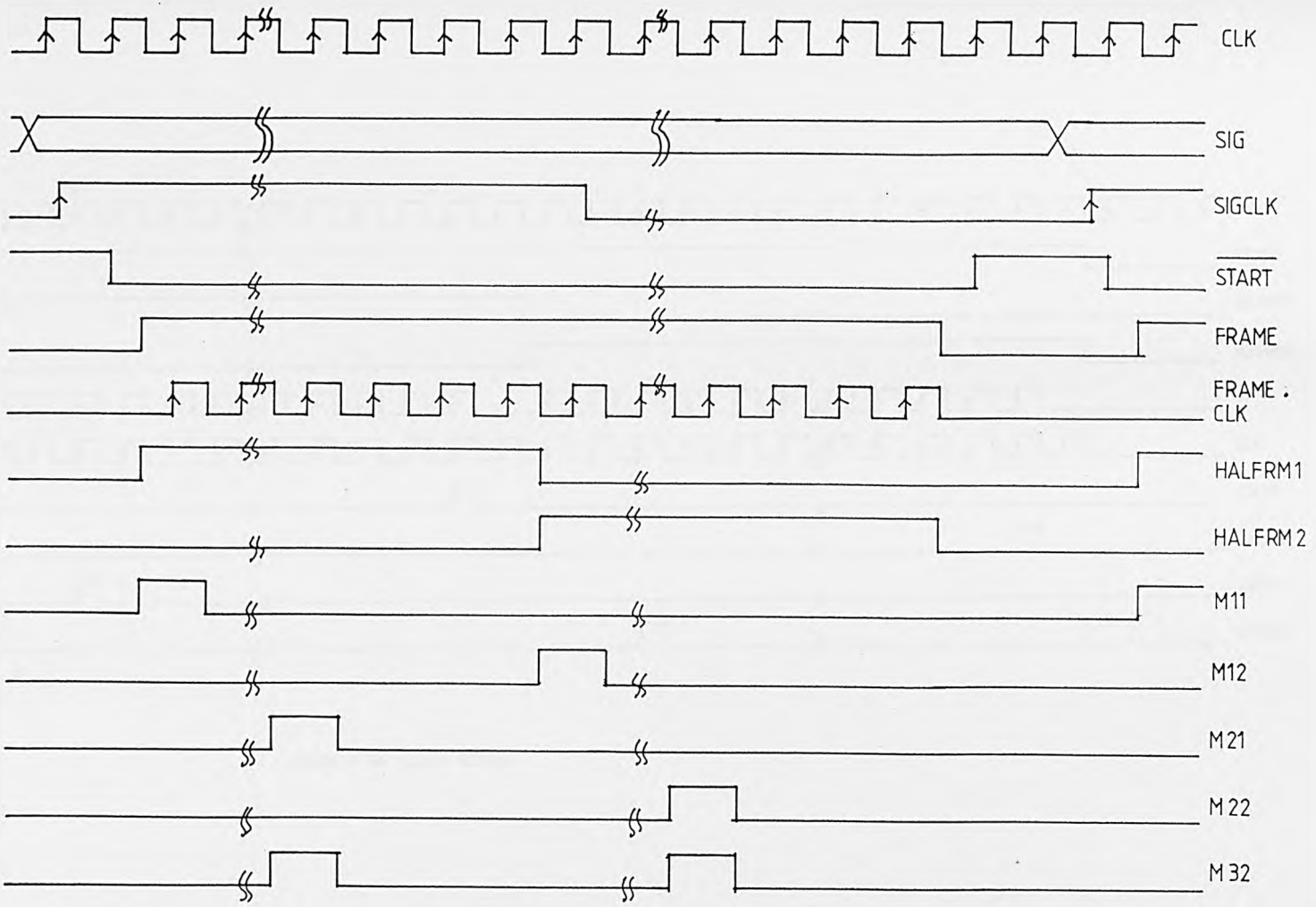


FIGURE 4.10a : CONTROL SIGNALS

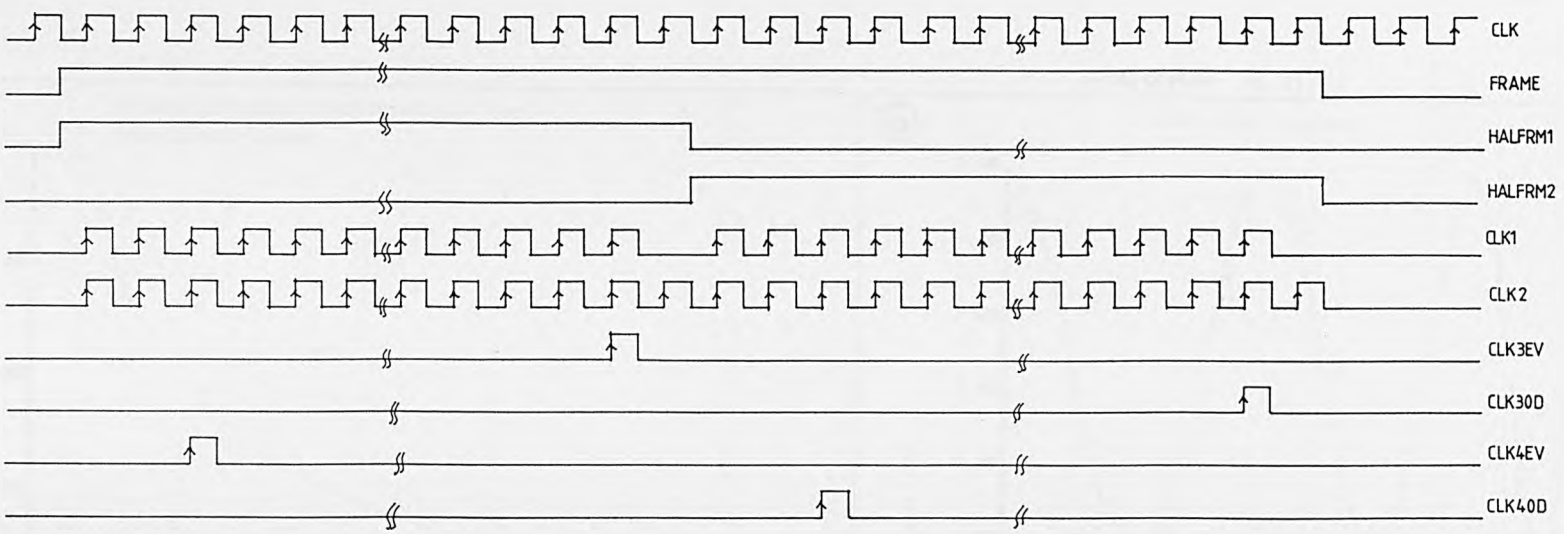
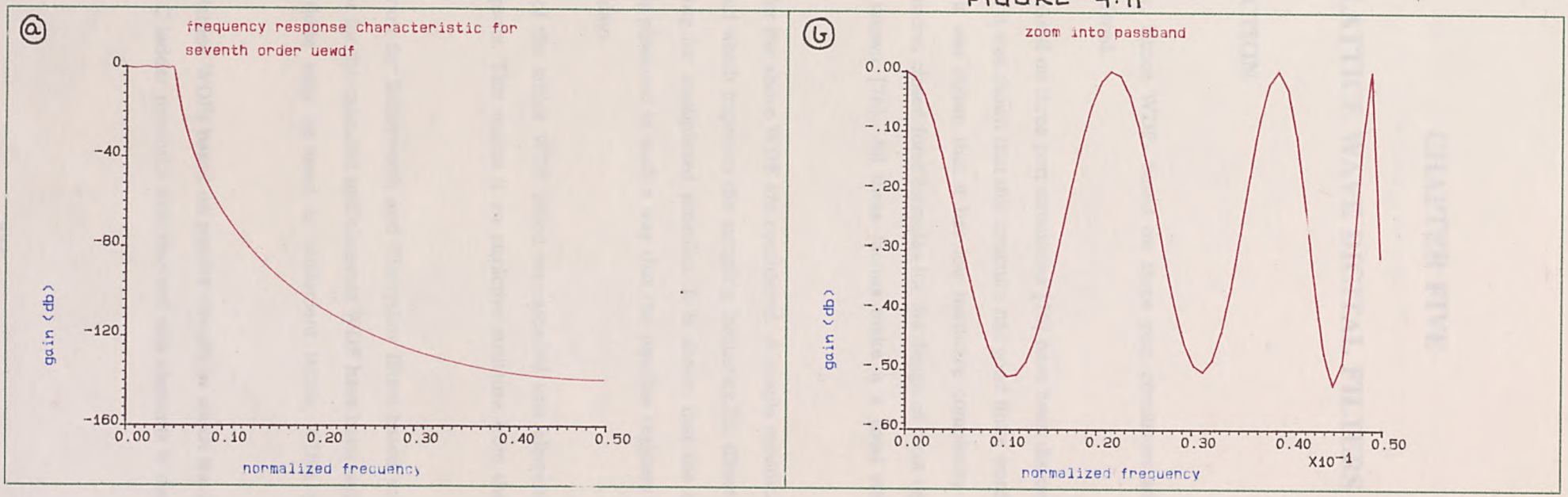


FIGURE 4.10b CLOCK SIGNALS

FIGURE 4.11


(a)
frequency response characteristic for
seventh order uewdf
0
-40
-80
-120
-160
gain (db)
0.00
0.10
0.20
0.30
0.40
0.50
normalized frequency
(b)
zoom into passband
0.00
-.10
-.20
-.30
-.40
-.50
-.60
gain (db)
0.00
0.10
0.20
0.30
0.40
0.50
X10-1
normalized frequency

CHAPTER FIVE

LATTICE WAVE DIGITAL FILTERS

5.1 INTRODUCTION

In this chapter, lattice WDFs based on three port circulators and cascaded unit elements are considered.

Lattice WDFs based on three port circulators [76] have been discussed in chapters 2 and 3. In chapter 2 it was shown that this structure has good finite wordlength properties and in chapter 3 it was shown that it has low hardware complexity relative to other structures. Furthermore, closed form formulas for the design of most types of filters with this structure are known [76]. All these factors make it a good structure for VLSI implementation.

Architectures for the above WDF are considered. A simple modification to the basic structure is proposed which improves the sampling frequency for direct implementations and allows pipelining for multiplexed schemes. It is shown that the basic multiplexed substructure can be pipelined in such a way that the pipeline registers also perform the functions of the delays.

The structure of the lattice WDF based on cascaded unit elements also comprises only two port adaptors. This makes it an attractive structure from the implementation viewpoint.

A design program for Butterworth and Chebyshev filters based on this structure is given. Architectures for the cascaded unit elements WDF have been studied in chapter 4. It is shown that these may be used to implement lattice WDFs with little or no modifications.

The design of lattice WDFs based on passive circuits in which the lattice reactances are realized with LC ladder networks with inserted unit elements is discussed. A design

program is given in which the LC networks are synthesized as Cauer form I [77] networks and the unit elements are inserted using Kuroda's transforms [59]

5.2 THEORETICAL ASPECTS

An analogue lattice filter between resistive terminations [2] is shown in figure 5.1. For equi-terminated lossless filters, R1=R2=R and the impedances Za and Zb are reactances. This type of network is also known as a symmetric network because the electrical behaviour is unaffected when the two ports are interchanged.

The number of components required to implement this filter is twice the theoretical minimum required to implement a particular transfer function [2]. This problem has been resolved by defining equivalent unbalanced forms [2,78] which require fewer components. An unbalanced equivalent of the symmetric lattice is shown in figure 5.2.

The derivation of the wave flow diagram corresponding to figure 5.2 is given in reference [45] and is shown in figure 5.3. Here Sa and Sb are the reflectances (reflection coefficients) of the reactances Za and Zb;

$$Sa = (Za-Ro)/(Za+Ro) \qquad (5.1)$$

$$Sb = (Zb-Ro)/(Zb+Ro)$$

The waveflow diagram of figure 5.3 can be simplified by removing the input A2 and the output B1 since these are not normally required. The resulting wave flow diagram is shown in figure 5.4.

The first step in the design process for lattice WDFs is to evaluate the lattice reactances Za and Zb for a given transfer function. The next step involves the synthesis of the lattice reactances with passive networks including LC ladder networks, cascaded unit elements, chain of circulators etc [45]. The digital realizations of Za and Zb are then obtained by using the techniques of Fettweis and take the place of Sa and Sb in figure 5.4.

The complexity of the lattice WDF is dependent on the type of network used to realize the lattice reactances. From this viewpoint, the simplest lattice WDFs are those derived from reference filters based on circulators and cascaded unit elements. The resulting WDFs require only two port adaptors.

5.3 LATTICE WDFs BASED ON FIRST AND SECOND ORDER ALL–PASS SECTIONS

The reflectance of a reactance is an all–pass function. Therefore, realizing reactances or all–pass functions amounts to the same thing. Mathematically, the simplest method of realizing reactances with all–pass sections is the chain connection of all–pass sections of degree 1 and 2. Such sections can be obtained by means of three port circulators [79] terminated at one port by a capacitance or an inductance or by a parallel or series resonant circuit. Circulators terminated with a capacitance and parallel resonant circuit are shown in figures 5.5a and 5.5b. The corresponding WDF realizations are shown in figures 5.5c and 5.5d. All–pass sections corresponding to circulators terminated with an inductance or series resonant circuit will be the same as figures 5.5c and 5.5d except for possible sign inversions [45].

The lattice reactances Za and Zb may be realized with a chain of circulators as discussed above. The resulting WDF equivalents of reflectances Sa and Sb (see figure 5.4) are then chain connections of first and second order all–pass sections. A typical lattice WDF derived in this manner is shown in figure 5.6.

Explicit formulas for designing most types (from the frequency response viewpoint) of lattice WDFs based on the structure of figure 5.6 have been reported by Gazsi [76]. This eliminates the need for analogue reference filter design. The lattice WDF structure studied in chapters 2 and 3 is that of figure 5.6. Designs for the various tolerance schemes in chapter 2 were carried out using Gazsi's formulas.

It was shown in chapters 2 and 3 that the lattice WDF has good finite wordlength properties and is one of the best structures from the hardware complexity viewpoint. The aim in the remainder of this section is to explore architectures for the hardware implementation of the structure of figure 5.6.

5.3.1 ARCHITECTURES

Clearly, we may choose a direct implementation technique for the structure. In this case, the critical path and therefore the maximum sampling frequency possible, is

dependent on the filter order. We can improve this by simply including additional delays as shown in figure 5.7. It is obvious that the transfer function of the structure will remain unaffected except for a constant delay which is dependent on the additional delays. Note that the number of delays in the two arms must be equal. The maximum sampling frequency possible with this modified structure is limited by the delay through two adaptors and is independent of the filter order.

Consider the general structure for a lattice arm shown in figure 5.8. This may be implemented with a multiplexed section comprising the substructure shown in the dotted area. If A,B,C represent the signal values at various points in the network at a given time, then the computational process for the multiplexed system may be described as follows:

clock cycle 1 : use (input,B1,C1) to evaluate (A2,B1',C1')

clock cycle 2 : use (A2,B2,C2) to evaluate (A3,B2',C2')

clock cycle 3 : use (A3,B3,C3) to evaluate (A4,B3',C3')

clock cycle m : use (Am,Bm,Cm) to evaluate (output,Bm',Cm')

The primes represent corresponding signal values one sampling interval later. The important point to note here is that once the required values of the signals have been evaluated, the initial values may be discarded. Therefore, no intermediate storage is required for the values B and C.

An architecture based on the above scheme is shown in figure 5.9. Detail of the two port adaptor architecture has been omitted, but any suitable architecture may be employed.

Assume that initially B1 resides in register 1 of stack B and all B and C values reside in corresponding registers in this manner. During the first clock cycle, the multiplexer selects the input and at the end of clock cycle 1, the values B1' and C1' are clocked into registers m in the stacks. At the same time, the values B1 and C1 are discarded and the value A2 is loaded into register A. During clock cycle 2, the multiplexer selects the output

of register A and at the end of this clock cycle, the values A3, B2' and C2' are clocked into the appropriate registers. Clearly, m clock cycles are required to evaluate the output and the clock frequency is limited by the delay through two adaptors. Note that at this point, all B and C values are in the correct registers for the next sampling interval.

The architecture may be used to implement each of the lattice arms of figure 5.6. Alternatively, we may use a single architecture to implement the two lattice arms in a time multiplexed fashion. The addition of two output registers and a subtractor to the architecture will allow the implementation of the complete filter.

The maximum sampling frequency possible is given by the clock frequency divided by half the filter order (nearest whole number). The clock frequency is limited by the delay through two adaptors.

The general structure for the modified lattice arm is shown in figure 5.10. As before, the computational process for the multiplexed system may be described as follows :

clock cycle 1 : use (A1,B1,C1) to evaluate (A1',B1',C1')

clock cycle 2 : use (A2,B2,C2) to evaluate (A2',B2',C2')

clock cycle 3 : use (A3,B3,C3) to evaluate (A3',B3',C3')

clock cycle m : use (Am,Bm,Cm) to evaluate (Am',Bm',Cm')

During the next sampling interval, B1',B2'..Bm' become B1, B2...Bm and C1',C2'..Cm' become C1, C2 ..Cm. However, A1' becomes A2, A2' becomes A3 and so on.

An architecture for the modified lattice arm is shown in figure 5.11. Note that the main difference between this and the architecture of figure 5.9 is the register stack A. Also note that this contains one more register than stacks B and C. The purpose of this is to achieve the skew in the A values discussed before. Again, m clock cycles are required to evaluate the output and the clock frequency is limited by the delay through two adaptors.

Clearly, this scheme has no advantage compared with the previous one. In fact, it is inferior because of the additional hardware required. However, note that for a direct

implementation of the structure of figure 5.10, the sections would operate in parallel. We can exploit this property by pipelining the multiplexed section and thereby improve speed performance.

As an example, consider the pipelined substructure shown in figure 5.12. The positions of the pipeline registers are shown marked with crosses. This substructure may be used as the basic section of a multiplexed system realizing a tenth order modified lattice arm as shown in figure 5.13. It is important to note that the functions of register stacks A,B and C (with the exception of one register in stack A) are now performed by the pipeline registers. Also note that the hardware requirement is only slightly greater (two registers more) than an equivalent realization with the architecture of figure 5.11.

During the first clock cycle, the first stage of the pipelined substructure will operate on the first section and at the end of the cycle, the appropriate results will be clocked into registers PIPE1 (figure 5.13). We may describe the operation of the system in these terms. This is shown for the first 15 clock cycles in table 5.1. Here, the results at the end of each clock cycle in the pipeline registers are shown. The first number indicates the section being operated on and the second number indicates the relevant input sample, e.g. 5.1 means the results of section 5 operating on input sample 1. Clearly, one input sample value can be processed every 5 clock cycles. Therefore, the sampling frequency equals 5 clock cycles and the clock frequency is limited by the pipeline processes. Since this clock frequency will be significantly higher than in the previous cases, the pipelined system will give us a significantly better speed performance.

The fact that the registers representing the delays can be effectively used to pipeline the basic section as discussed above is important. It means that for a system of order 2n, the section may be split up into n pipeline stages. If we assume ideal shift registers (zero set and hold times) and similar pipeline stages, then it is clear that the sampling frequency will be independent of n.

Therefore, for a general purpose system, n should be kept as high as possible thereby offering a wide range of filter orders. Lower orders are obtained by simply setting the redundant coefficient values to zero.

The above system may be used as a basis to realize a complete lattice WDF in the same way as discussed previously. It should be clear that this system will give a significantly better speed performance particularly for higher orders.

5.4 LATTICE WDFs BASED ON CASCADED UNIT ELEMENTS

According to Richards [58], any driving point reactance function can be realized with a cascade of unit elements. Therefore, the lattice reactances Za and Zb (figure 5.1) may be realized with a cascade of unit elements. The advantage of this is that each of the lattice arms of the WDF is then a cascade connection of two port adaptors.

5.4.1 DESIGN ASPECTS

SYNTHESIS OF REACTANCE FUNCTIONS WITH CASCADED UNIT ELEMENTS

Short and open circuited transmission lines behave like inductors and capacitors in the Richards complex frequency domain, S. Therefore, any reactance function which can be synthesized with inductors and capacitors in the s domain can be synthesized in a similar manner in the S domain.

More importantly, any reactance function in the S domain can be synthesized with a cascade of unit elements. The basis of this is Richards' theorem [2,58]:

If Z(S) is a positive real function and Z(S)/Z(1) is not identically equal to S or 1/S, then

$$Z'(S) = \frac{(Z(S)-S.Z(1))}{(Z(1)-S.Z(S))} \qquad (5.2)$$

is also a positive real function. Further, if Z(S) is rational in S, then a factor S−1 cancels in the numerator and denominator of the above expression, and Z'(S) is the same degree as Z(S). If Z(1)+Z(−1)=0 as well, a second factor S+1 cancels in the numerator and denominator, and Z'(S) is of lower degree than Z(S).

It follows from this that if Z(S) is a positive real impedance function satisfying Z(S)/Z(1) is not equal to S or 1/S, then a unit element of value Z(1) may always be extracted from Z(S), leaving a positive real impedance Z'(S) in cascade with the unit element. The value of Z'(S) is :

$$Z'(S) = \frac{((Z(S)-S.Z(1))Z(1))}{(Z(1)-S.Z(S))} \qquad (5.3)$$

The impedance Z'(S) is also positive real and of degree equal to Z(S) if Z(1)+Z(−1) is not equal to zero. It is of degree one less than Z(S) if Z(1)+Z(−1)=0. This procedure is illustrated schematically in figure 5.14 and may be used to synthesize an arbitrary reactance function with a cascade of unit elements terminated in an open or short circuit (if Z'(S)=S then short circuit and if Z'(S)=1/S then open circuit).

DESIGN PROGRAM

A program for the design of lattice WDFs based on cascaded unit elements has been written and is shown in appendix D1.

The program essentially performs the following functions:

(a) uses conventional techniques to derive transfer functions based on Chebyshev and Butterworth approximations.

(b) derives lattice reactances Za and Zb (figure 5.1) for the transfer function obtained in (a) using techniques described in reference [2].

(c) synthesizes the lattice reactances with cascaded unit elements using the techniques discussed above.

(d) derives a digital structure from the analogue lattice filter using the techniques of Fettweis [45].

An example of a WDF designed with this program will be given in chapter 7.

5.4.2 ARCHITECTURES

The structure for a lattice WDF based on cascaded unit elements is shown in figure 5.15. It is clear that for a direct implementation of this structure, the maximum sampling frequency possible is limited by the delay through two adaptors, a subtractor and a 0.5

multiplier. This can be improved by adding a delay at the output of each lattice arm. The maximum sampling frequency is then limited by the delay through two adaptors.

Architectures for cascaded unit elements WDFs have been discussed in chapter 4. With slight modifications, any of these may be used to implement each of the lattice arms. Consider the cascaded unit elements WDF structure of figure 5.16. This may be used to realize a kth order (where k is less than or equal to n) lattice arm in the following manner:

The input is applied at input 1, input 2 is set to zero and the output is taken from output 2. All coefficients corresponding to the two port adaptors k+2 to n+1 are set to zero. The coefficient corresponding to the k+1 adaptor is set to 1 if the kth unit element in the lattice arm is short circuited, and to −1 if it is open circuited.

Therefore, any programmable architecture for the cascaded unit elements WDF which gives access to all four inputs and outputs can be used to realize a lattice arm. The architecture discussed in chapter 4 may be used to realize a lattice arm. The sampling frequency with this architecture will equal (2k+8) clock cycles where the clock frequency is limited by the pipeline processes (see chapter 4).

Two such architectures can be used with the addition of a subtractor to realize the complete lattice WDF. The 0.5 multiplier is simply a shift to the right of one bit. It is also possible to realize the two lattice arms in a time multiplexed fashion. Therefore, a single multiplexed two port adaptor may be used to realize the complete lattice WDF. The important point to note is that very little additional control will be required compared with parallel implementation of the two lattice arms.

5.5 LATTICE WDFs BASED ON LC NETWORKS

A driving point reactance function can be realized with a variety of LC networks [78]. One method of synthesis is due to Cauer and is used to synthesize a reactance function with an LC network of the type shown in figure 5.17a. This method of synthesis is generally referred to as Cauer form 1 [77].

A wave digital network derived from the network of figure 5.17a results in a cascade connection of three port parallel and series adaptors which appear alternately in the cascade as shown in figure 5.17b [45]. This digital network is not suitable for hardware implementation because of the long critical path (see chapter 3 for definition of critical path) and because it is not multiplexable. These problems can be overcome by inserting unit elements into the LC ladder using Kuroda's transforms [59].

The Kuroda transform is illustrated in figure 5.18. Effectively, this allows us to insert unit elements at the required positions in the LC one port without affecting the reactance function. The manner in which this is achieved is as follows. A unit element of characteristic impedance one ohm is placed at the input of the one port network and shifted to the required location by repeated application of Kuroda's transforms. This procedure is repeated for the required number of unit elements. For a more detailed discussion of Kuroda's transform, see reference [59].

An odd order LC ladder after insertion of unit elements is shown in figure 5.19a. Note that application of Kuroda's transform has replaced all series inductors with shunt capacitors. The wave digital network corresponding to the above network is shown in figure 5.19b. Note that the unit elements in figure 5.19a have been placed so that all three port adaptors are separated by half delays and only parallel three port adaptors are required. It is clear from the arguments presented in chapter 3 that this network is more suitable for hardware implementation than the network of figure 5.17b.

An even order LC ladder after insertion of unit elements is shown in figure 5.20a. Note that all series inductors have been replaced with shunt capacitors. The corresponding wave digital network is shown in figure 5.20b.

Note that in this case, the last adaptor is a two port. The type of two port adaptor can be selected so that it can be realized with a three port adaptor implementation (this aspect is discussed more fully in chapter 8). Therefore, from the hardware implementation viewpoint, the comments for the odd order case apply.

A program for the design of lattice WDFs based on the above methods has been written and is shown in appendix D2. The program essentially performs the following functions: uses conventional techniques to derive transfer functions based on Chebyshev and Butterworth approximations. derives lattice reactances Za and Zb (figure 5.1) for the transfer function obtained in (a) using techniques described in reference [2]. synthesizes the lattice reactances with Cauer form 1 LC networks. inserts unit elements in the positions described earlier using Kuroda's transforms. derives a digital structure from the analogue lattice filter using the techniques of Fettweis [45].

An example of a WDF designed with this program will be given in chapter 8.

5.6 DISCUSSION AND CONCLUSIONS

WDFs derived from analogue lattice filters in which the lattice reactances are realized with first and second order all–pass sections have been considered. It has been shown that a simple modification to the basic structure improves the sampling frequency and makes it independent of the filter order. Various architectures have been discussed and one which uses pipelining of the basic section in a multiplexed scheme is considered particularly useful. The novel feature of this architecture is that pipeline registers also perform the functions of the delays. The limitation to this method is that the number of pipeline stages has to be half the order of the lattice arm in order to gain full advantage of the pipelining. In cases where this is not so, pipelining is still possible but the control functions will become much more complicated.

Lattice WDFs based on cascaded unit elements have also been considered. A design program for Butterworth and Chebyshev filters based on this structure has been given. It has been shown that the architectures employed for cascaded unit elements WDFs can also be used with little or no modification to implement lattice WDFs based on cascaded unit elements.

Lattice WDFs based on passive networks in which the lattice reactances are realized with LC networks with inserted unit elements have been discussed and a design program for such WDFs has been presented.

	CLOCK CYCLE	PIPE 1	PIPE 2	PIPE 3	PIPE 4	PIPE 5
*	1	1.1	X	X	X	X
	2	2.1	1.1	X	X	X
	3	3.1	2.1	1.1	X	X
	4	4.1	3.1	2.1	1.1	X
*	5	5.1	4.1	3.1	2.1	1.1
	6	1.2	5.1	4.1	3.1	2.1
	7	2.2	1.2	5.1	4.1	3.1
	8	3.2	2.2	1.2	5.1	4.1
†	9	4.2	3.2	2.2	1.2	5.1
*	10	5.2	4.2	3.2	2.2	1.2
	11	1.3	5.2	4.2	3.2	2.2
	12	2.3	1.3	5.2	4.2	3.2
	13	3.3	2.3	1.3	5.2	4.2
†	14	4.3	3.3	2.3	1.3	5.2
*	15	5.3	4.3	3.3	2.3	1.3

X : don't care

* : new input sample value is fed into the system at the start of clock cycle

† : output sample value is available at end of clock cycle

TABLE 5.1 : STATUS OF PIPELINE REGISTERS

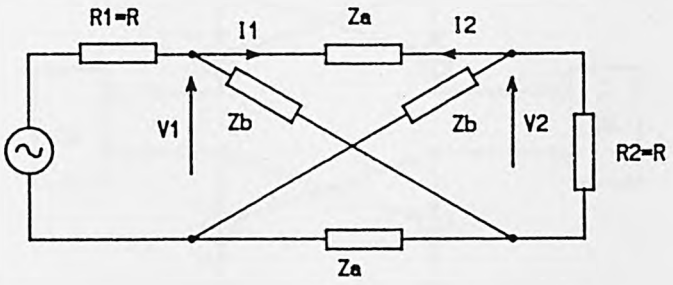


FIGURE 5.1 : THE SYMMETRIC LATTICE FILTER

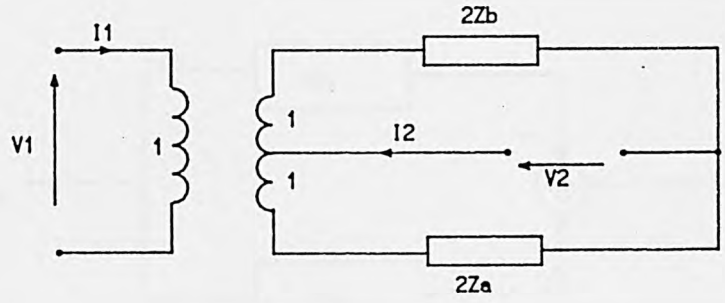


FIGURE 5.2 : UNBALANCED EQUIVALENT OF SYMMETRIC LATTICE

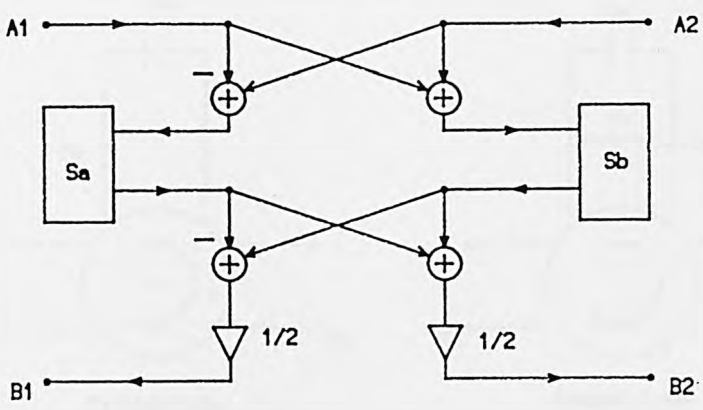


FIGURE 5.3 : WAVE FLOW DIAGRAM FOR LATTICE FILTER

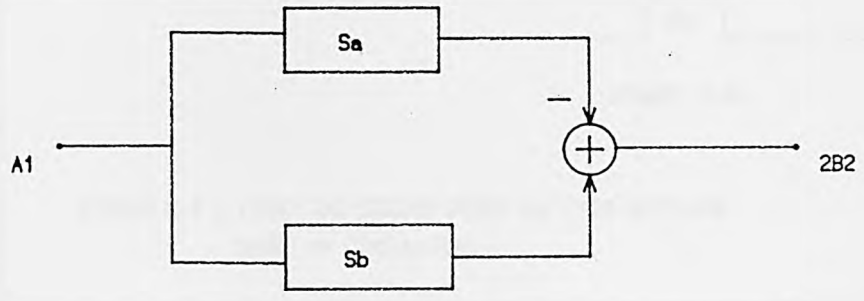


FIGURE 5.4 : SIMPLIFIED WAVE FLOW DIAGRAM FOR LATTICE FILTER

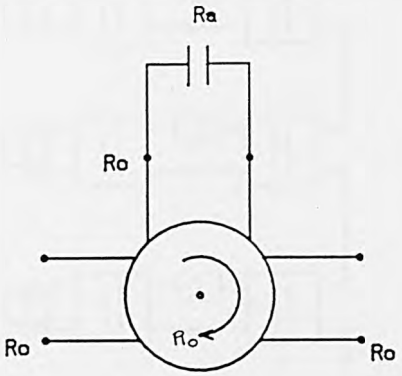


FIGURE 5.5a

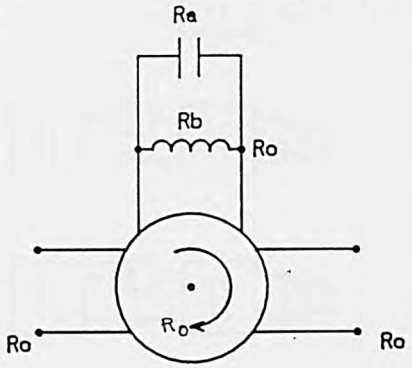


FIGURE 5.5b

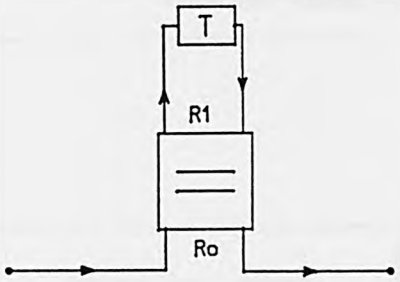


FIGURE 5.5c

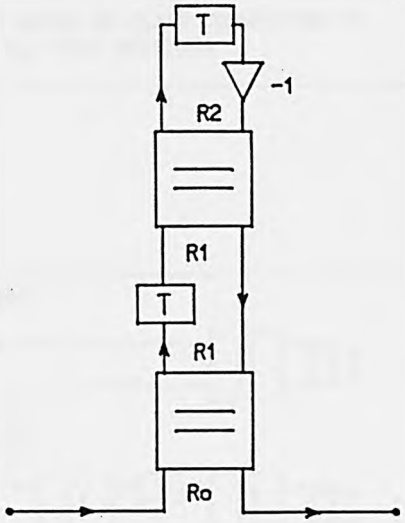


FIGURE 5.5d

FIGURE 5.5 : FIRST AND SECOND ORDER ALL PASS SECTIONS BASED ON CIRCULATORS

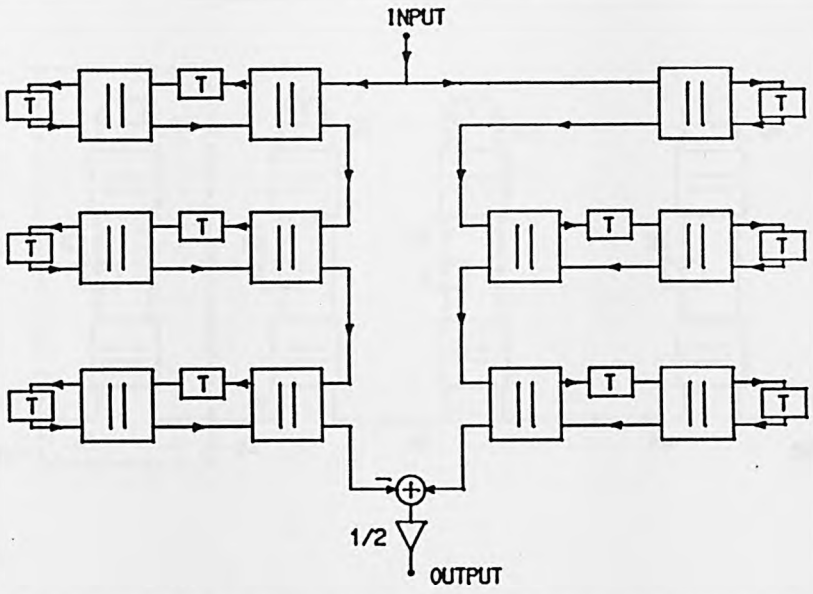


FIGURE 5.6 : 11-th ORDER LATTICE WDF BASED ON CHAIN CONNECTION OF FIRST AND SECOND ORDER ALL PASS SECTIONS

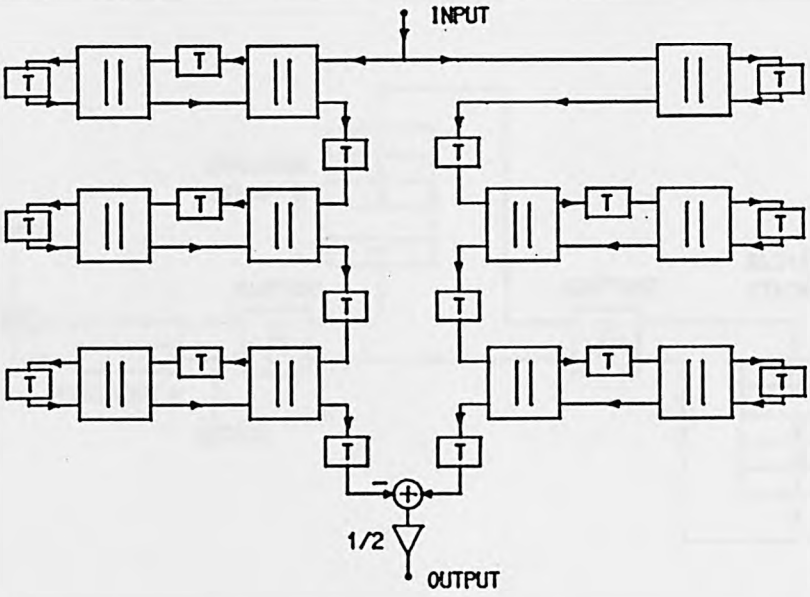


FIGURE 5.7 : MODIFIED LATTICE WDF

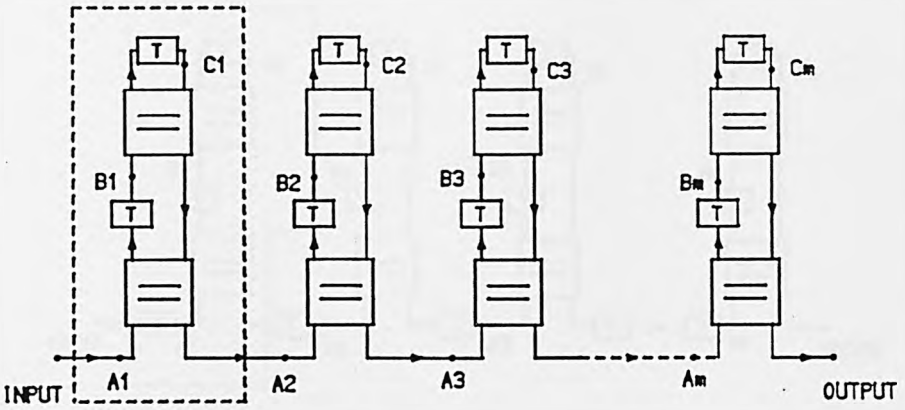


FIGURE 5.8 : GENERAL STRUCTURE FOR LATTICE ARM

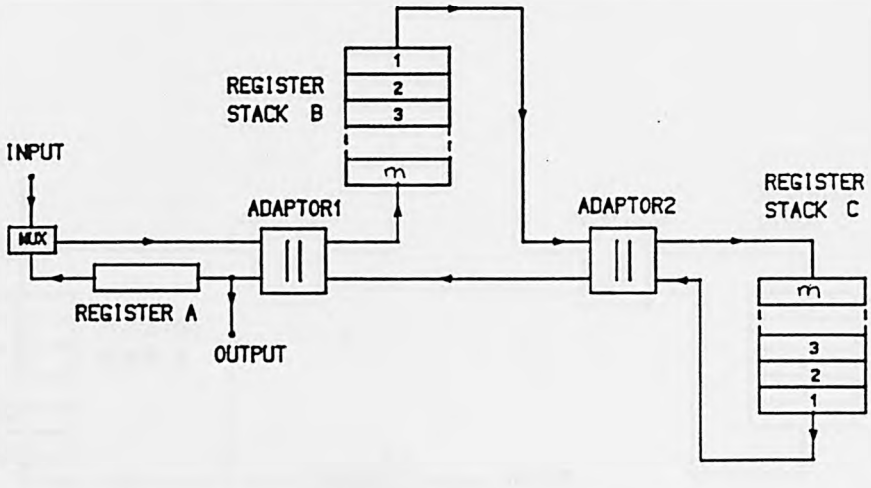


FIGURE 5.9 : MULTIPLEXED ARCHITECTURE FOR LATTICE ARM

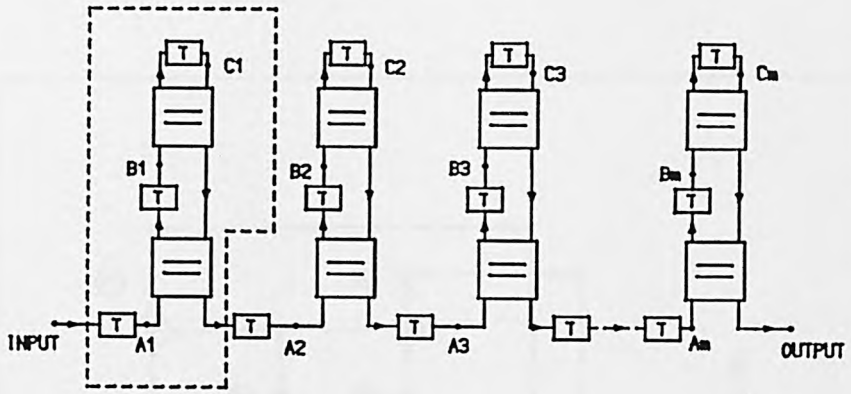


FIGURE 5.10 : GENERAL STRUCTURE FOR A MODIFIED LATTICE ARM

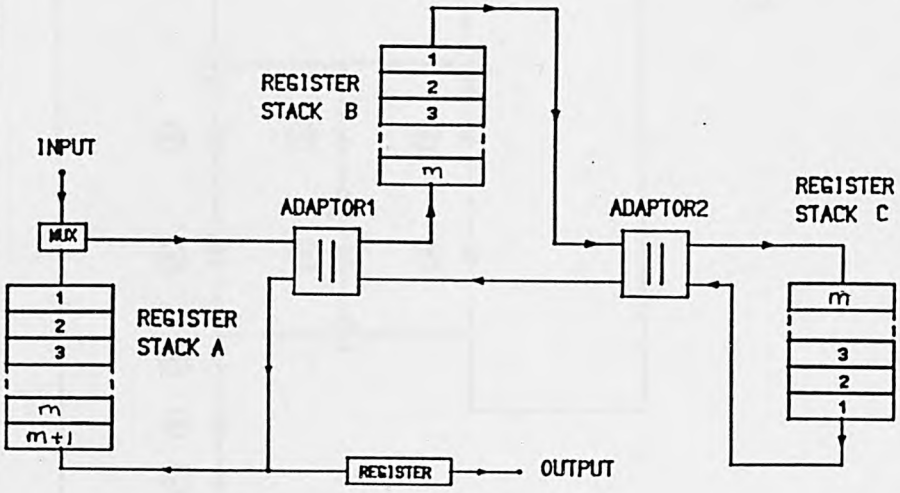


FIGURE 5.11 : MULTIPLEXED ARCHITECTURE FOR MODIFIED LATTICE ARM

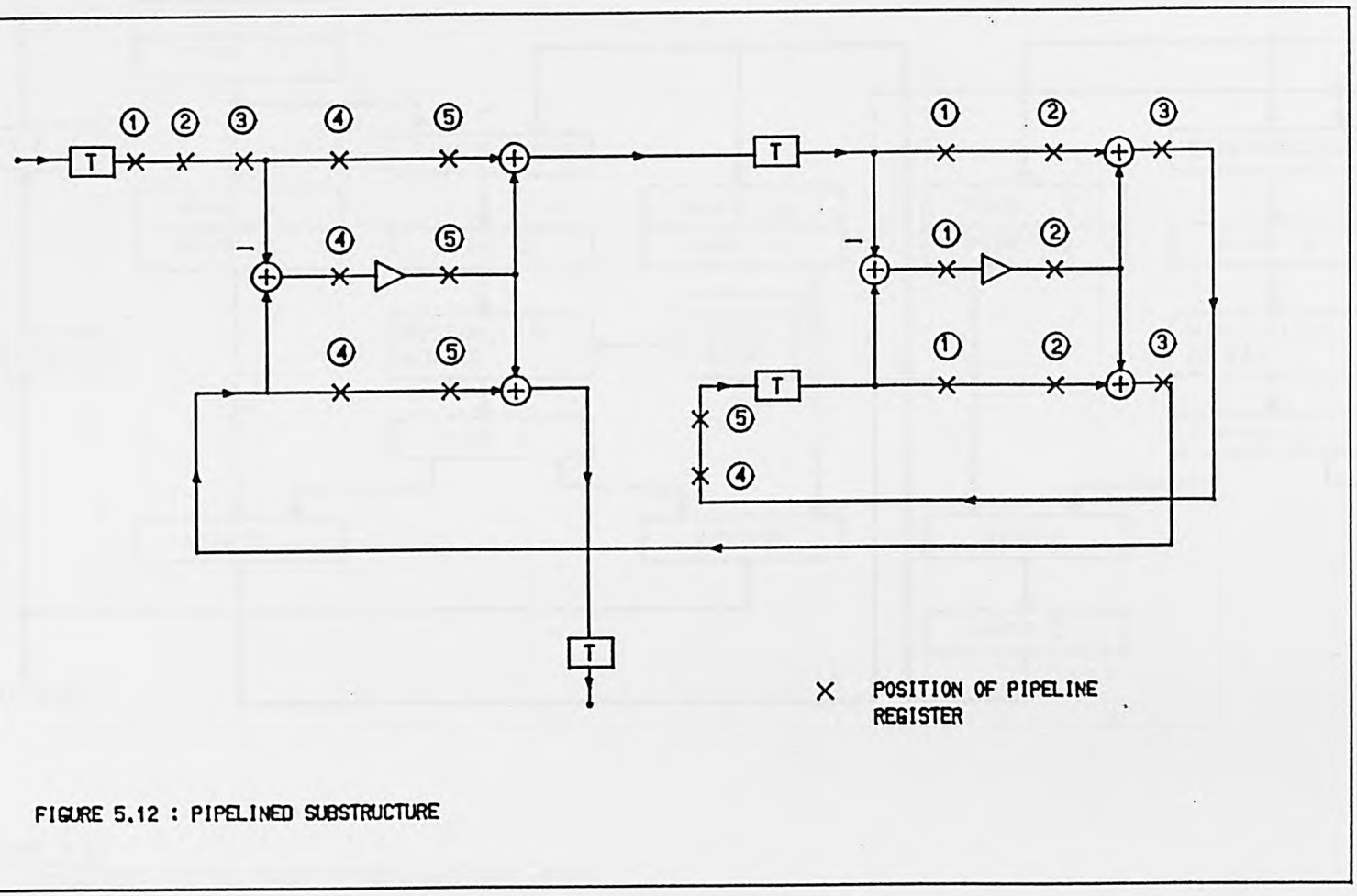


FIGURE 5.12 : PIPELINED SUBSTRUCTURE

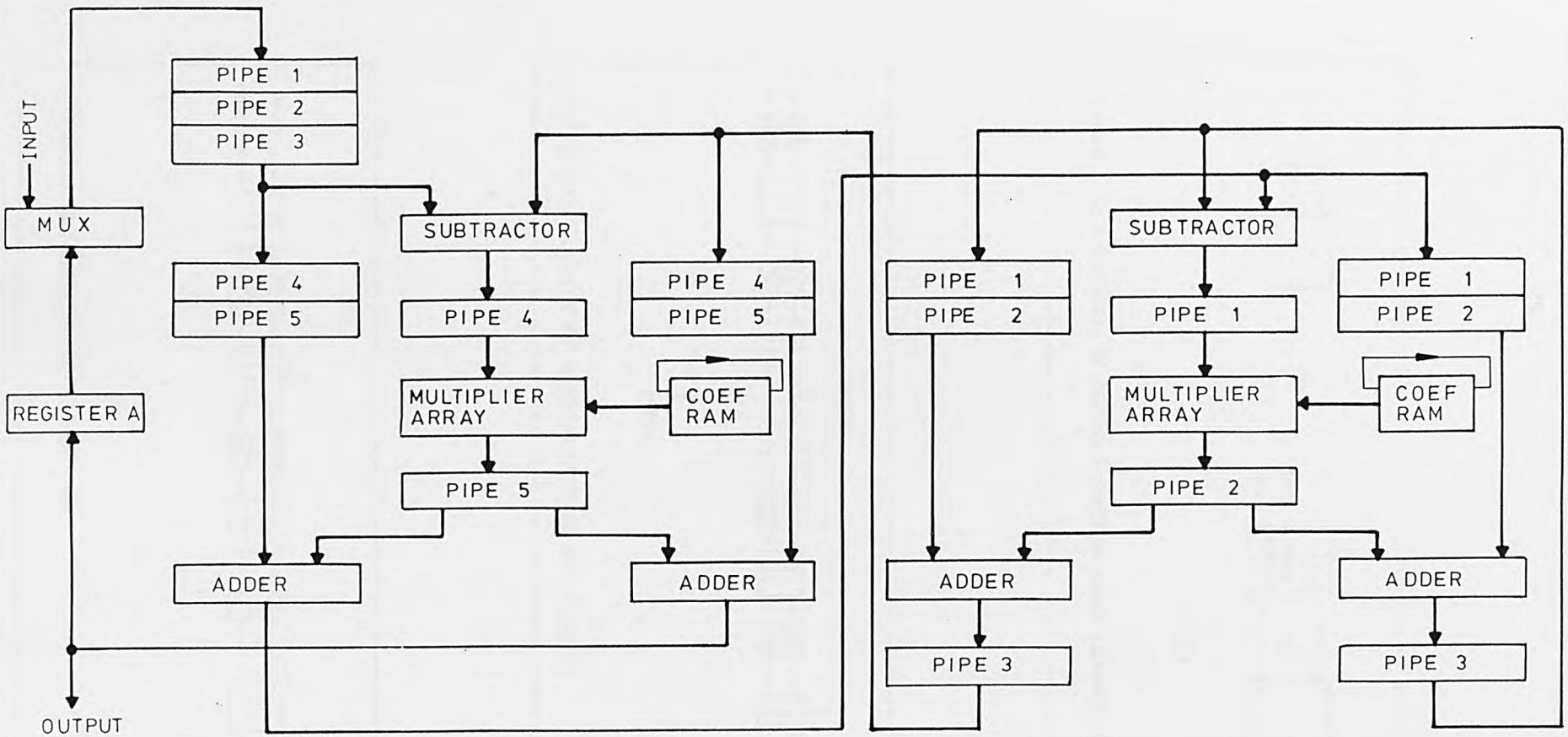


FIGURE 5.13
MULTIPLEXED PIPELINED ARCHITECTURE FOR MODIFIED LATTICE ARM

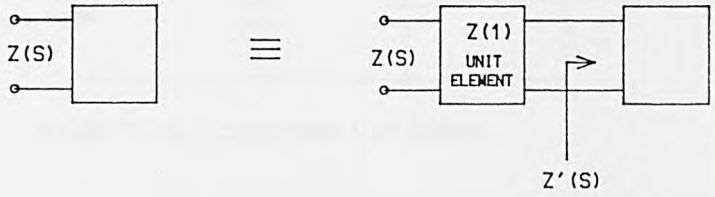


FIGURE 5.14 : SYNTHESIS OF REACTANCE FUNCTION USING RICHARDS' PROCEDURE

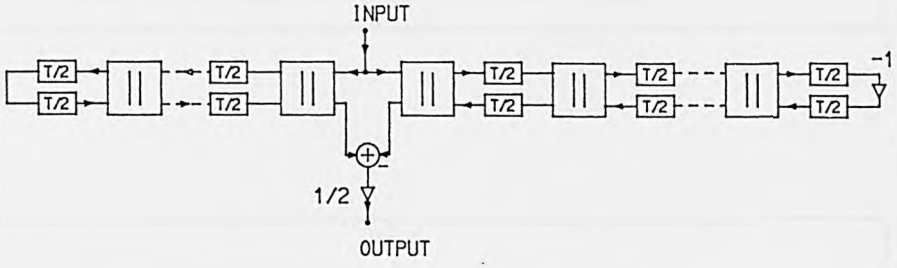


FIGURE 5.15 : LATTICE WDF BASED ON CASCADED UNIT ELEMENTS

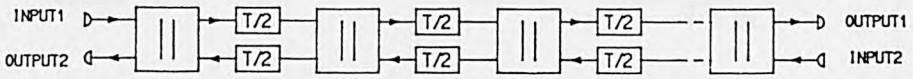


FIGURE 5.16 : CASCADED UNIT ELEMENTS WDF

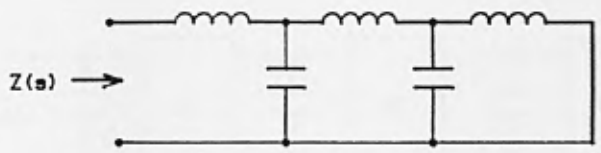


FIGURE 5.17a : CAUER FORM I LC LADDER

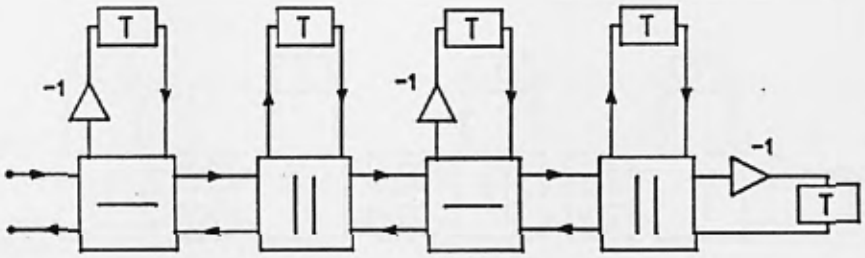


FIGURE 5.17b : CORRESPONDING WAVE DIGITAL NETWORK

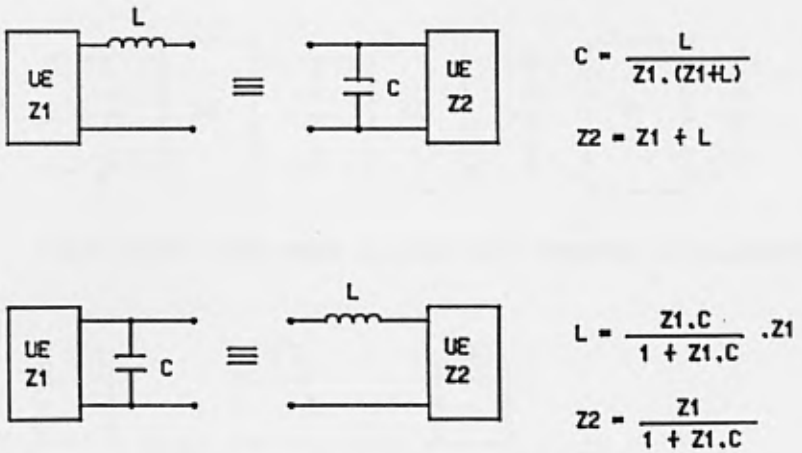


FIGURE 5.18 : KURODA'S TRANSFORMS

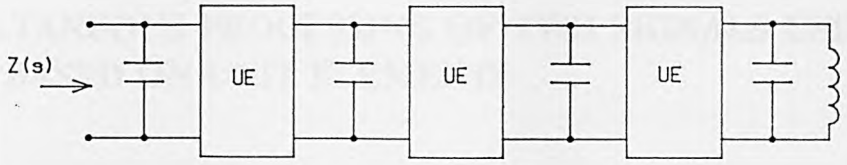


FIGURE 5.19a : ODD ORDER LC LADDER WITH INSERTED UNIT ELEMENTS

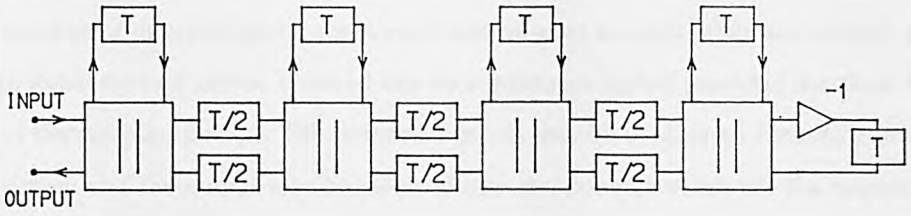


FIGURE 5.19b : CORRESPONDING WAVE DIGITAL NETWORK

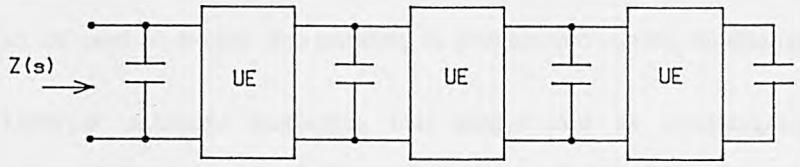


FIGURE 5.20a : EVEN ORDER LC LADDER WITH INSERTED UNIT ELEMENTS

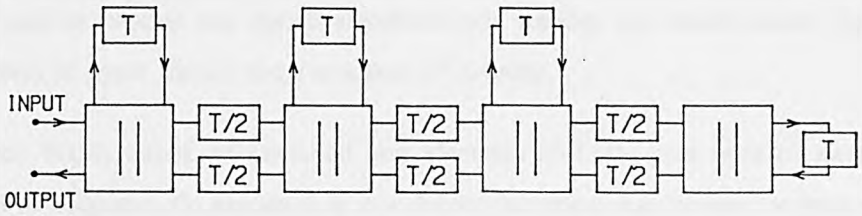


FIGURE 5.20b : CORRESPONDING WAVE DIGITAL NETWORK

CHAPTER SIX

SIMULTANEOUS PROCESSING OF TWO SIGNALS USING WDFs BASED ON UNIT ELEMENTS

6.1 INTRODUCTION

If a number of discrete signals are skewed with respect to each other and added, the composite signal formed can be resolved into its constituent signals provided the skew is a fraction of the sampling period. This composite signal may be used as an input to a linear discrete system and the output may be resolved into components which are the responses of the system to the individual inputs. Effectively, the discrete system operating in this manner processes a number of signals simultaneously. Theoretical arguments to show that this is the case are given in section 6.2.

It is shown in section 6.3 that the simultaneous processing property of theoretical discrete systems is of no use in most digital implementations of discrete systems other than to provide a theoretical basis for designing multiplexed systems. However, digital implementations of WDFs based on unit elements and lattice WDFs based on all– pass sections can be used to exploit this property to process two signals simultaneously.

The UEWDF structure employing half delays and its advantages from the implementation viewpoint have been discussed in chapter 4. It is shown in section 6.4 that this structure can be used to process two signals simultaneously and simulation results are given. In practice, the implication is that any hardware implementation of this structure may be used to process two signals simultaneously without any modifications (ignoring combination of input signals and resolution of outputs).

Lattice WDFs based on cascaded unit elements and all– pass sections have been discussed in chapter 5. In section 6.5, it is shown that these may be used to process two signals simultaneously and simulation results are given. Similar treatment is given to the LC ladder WDF in section 6.6.

6.2 THEORETICAL ASPECTS

Any linear discrete system can be used to simultaneously process a number of signals which are skewed by a fraction of the sampling period. A proof of this is given below.

(I) Any linearly sampled discrete signal can be represented in the Z-domain with the general expression:

$$X(Z) = \frac{\sum_{k=0}^{m} ak.Z^{-k}}{\sum_{k=0}^{n} bk.Z^{-k}} \qquad (6.1)$$

where k, m, n are integers.

(ii) If the signal X(Z) is applied to a linear discrete system with a transfer function T(Z), where:

$$T(Z) = \frac{\sum_{k=0}^{p} ck.Z^{-k}}{\sum_{k=0}^{q} dk.Z^{-k}} \qquad (6.2)$$

where k, p, q are integers.

then it is clear that the output signal Y(z)=T(z).X(z) will be of the form :

$$Y(Z) = \frac{\sum_{k=0}^{m+p} ek.Z^{-k}}{\sum_{k=0}^{n+q} fk.Z^{-k}} \qquad (6.2)$$

where k can only take on integer values. It follows from this that the output signal is defined at t second intervals where t is the sampling period.

(iii) Now, if the signal X(Z) is delayed by a fractional delay $Z^{-1/v}$ (where v is an integer) and applied to T(Z), then the output is given by $Z^{-1/v}.Y(Z)$. That is, the output is the signal Y(Z) delayed by t/v seconds.

(iv) It follows from the superposition theorem and (iii) that for a linear discrete system with a transfer function of the form shown in (ii), any number of signals which are skewed by a fractional delay (with respect to the sampling period) can be simultaneously processed ,i.e. a number of signals can be skewed, added and then used as the input signal to the discrete system. The output signal from the discrete system can be resolved its components by an appropriate demultiplexing process.

A discrete system that operates on a number of signals in this manner can be viewed as a number of similar time multiplexed discrete systems operating on separate signals. An extension of this idea would be to provide facilities for alteration of the transfer function coefficients at the appropriate time instants so that a number of different time multiplexed systems which operate on separate signals could be obtained.

6.3 IMPLICATIONS FOR DIGITAL IMPLEMENTATIONS OF DISCRETE SYSTEMS

In most digital implementations of discrete systems, the delay element is realized with a shift register. The shift registers are clocked every T seconds to realize delays of T seconds. It is clear that this is not a true realization of a delay since the output of the shift register is not a delayed version of its input but simply a sample of this every T seconds. Effectively, the digital system behaves like the corresponding discrete system at every clock cycle edge, i.e. once every T seconds. Clearly, the property discussed in section 6.2 is of no use in such a system.

However, we may realize a delay of T seconds with a cascade of n shift registers each realizing a delay of T/n seconds. The clock frequency in such a case is n times the sampling frequency and more hardware is required in comparison with the above system. Effectively, the digital system behaves like the corresponding discrete system at n time instants during each sampling interval. Therefore, according to the simultaneous processing property discussed in section 6.2, we may process n signals simultaneously or , with appropriate provision for coefficient values, realize n different time multiplexed systems.

Therefore, given a direct implementation of a digital filter, we may obtain a number of time multiplexed filters by replacing each shift register representing a delay with a cascade of shift registers, increasing the clock frequency proportionally and making provisions for different coefficient values for each multiplier. Note that at a given time instant (or clock period) within a sampling period, the coefficients corresponding to one particular filter must be active for all the multipliers. Since each shift register has been replaced with a cascade of shift registers, no multiplexing at the bus level will be necessary. Obviously, we could have obtained this scheme without referring to the simultaneous processing property. However, it is theoretically more complete to arrive at it in this manner.

It is clear from above that when processing a number of signals simultaneously in the manner described, the sampling frequency is proportionally reduced. An interesting exception are WDFs based on unit elements. In these cases, the delays can be split up in

such a way that two signals can be processed simultaneously at the same sampling rate as a single signal. The lattice WDF based on all–pass sections can also be used in this manner.

Although the number of shift registers required in some of these cases is increased, the effective doubling of the processing rate (two signals are processed at the same rate as one signal) will obviously result in a more efficient use of the hardware. Furthermore, some of the WDF structures employing half delays have other advantages from the implementation viewpoint as discussed in chapters 4,5 and 8.

6.4 THE CASCADED UNIT ELEMENTS WDF

6.4.1 SIMULTANEOUS PROCESSING OF TWO SIGNALS

Consider the UEWDF structure of figure 6.1a. A direct digital implementation in which each T/2 delay is realized with a shift register can be used to process two signals (skewed by T/2 seconds) simultaneously. That this is the case can be seen from the arguments presented in sections 6.2 and 6.3.

The WDF realization of a unit element is a delay of T seconds which may be split up between the upper and lower arms as discussed in chapter 4. Therefore, we may redraw the structure of figure 6.1a as shown in figure 6.1b. It follows that a direct digital implementation of this structure can also be used to process two signals simultaneously.

For direct implementations of both structures of figure 6.1, the sampling frequency is equal to half the clock frequency. For the implementation of figure 6.1a, the clock frequency is limited by the delay through two adaptors , whereas for figure 6.1b it is limited by the delay through one adaptor. Therefore, by employing the structure of figure 6.1b, we may process two signals simultaneously without the penalty of lower sampling frequency. This structure has been studied in chapter 4. The important point to note here is that any hardware implementation of this structure can be used to process two signals simultaneously without any modifications (excluding those required to combine the input signals and separate the output signals). Consequently, the number of samples processed per second is doubled.

6.4.2 PROOF BY SIMULATION

In order to show that the UEWDF of figure 6.1b can be used to process two signals simultaneously, a UEWDF satisfying an arbitrary specification was simulated with a modified version of DIFSIP (for a description of DIFSIP see chapter 2).

The modifications made to DIFSIP are illustrated in figure 6.2. The original DIFSIP simulates synchronous digital networks clocked every T seconds whereas in the modified version, the clock period is T/2 seconds. Furthermore, the modified version simulates two signal generators the outputs of which are skewed by T/2 seconds and combined to form

the input to the digital network. In software, this is implemented by two signal generating routines which alternately supply the input sample. The outputs are similarly separated by assigning alternate samples from the digital filter output signal to the two outputs.

A seventh order UEWDF with the following coefficients was simulated with the above program ; $\alpha 1 = -\alpha 8 = -0.83553984$, $\alpha 2 = -\alpha 7 = 0.97769094$, $\alpha 3 = -\alpha 6 = -0.98514007$, $\alpha 4 = -\alpha 5 = 0.98606601$.

The frequency responses for both inputs were obtained and were the same as expected. The frequency response (coefficient wordlength = 16 bits) for input 1 and output 2 is shown in figure 6.3a.

A sinusoidal signal of amplitude 0.001 volts and frequency 0.05 radians per second was applied at input 1 (see figure 6.2b) and a similar signal of frequency 0.1 radians per second was applied at input 2 at the same time. The response to input 1 is obtained at output 2 and is shown in figure 6.3c. The response to input 2 is obtained at output 1 and is shown in figure 6.3b. The coefficient wordlength in both cases was 16 bits and the signal wordlength was 20 bits. Note that the output signals are as expected. Also note that the initial distortion in the sine waves is due to the transient response time.

6.5 LATTICE WDFs

We will consider lattice WDFs based on first and second order all-pass sections and those based on cascaded unit elements.

6.5.1 SIMULTANEOUS PROCESSING OF TWO SIGNALS

Consider the lattice WDF structure shown in figure 6.4a. The reflectance of an all-pass section is unaltered if the delay T is split between the upper and lower arms. This property has been used to derive the equivalent structure of figure 6.4b. Note that the delays used to pipeline the lattice arms are now T/2.

The maximum sampling frequency possible for direct digital implementations of both structures of figure 6.4 will be the same. For figure 6.4a, the sampling frequency is equal to the clock frequency which is limited by the delay through two adaptors. For figure 6.4b, the sampling frequency is twice the clock frequency which is limited by the delay through one adaptor. However, direct digital implementations of figure 6.4b can be used to process two signals simultaneously. The penalty for this is n additional shift registers where n is the filter order.

For the lattice WDF based on cascaded unit elements, the arguments presented in section 6.4.1 apply.

6.5.2 PROOF BY SIMULATION

In order to show that the lattice WDF of figure 6.4b can be used to process two signals simultaneously, a fifth order filter with the following coefficients was simulated with the modified version of DIFSIP; $\alpha 1 = -0.3916527$, $\alpha 2 = -0.6877369$, $\alpha 3 = 0.9000279$, $\alpha 4 = -0.9353002$, $\alpha 5 = -0.9527074$.

The frequency responses for both inputs were obtained and were the same as expected. The frequency response (coefficient wordlength = 16 bits) for input 1 and output 2 is shown in figure 6.5a.

A sinusoidal signal of amplitude 0.001 volts and frequency 0.05 radians per second was applied at input 1 (see figure 6.2b) and a similar signal of frequency 0.1 radians per

second was applied at input 2 at the same time. The response to input 1 is obtained at output 2 and is shown in figure 6.5c. The response to input 2 is obtained at output 1 and is shown in figure 6.5b. The coefficient wordlength in both cases was 16 bits and the signal wordlength was 20 bits.

A fifth order lattice WDF based on unit elements (figure 6.6) with the following coefficients was simulated in the same manner as above; $\alpha 1 = -0.3046126$, $\alpha 2 = 0.2225531$, $\alpha 3 = -0.2206677$, $\alpha 4 = 0.6918970$, $\alpha 5 = 0.0043051$.

The frequency responses for both inputs were obtained and were the same as expected. The frequency response (coefficient wordlength = 16 bits) for input 1 and output 1 is shown in figure 6.7a.

A sinusoidal signal of amplitude 0.001 volts and frequency 0.05 radians per second was applied at input 1 (see figure 6.2b) and a similar signal of frequency 0.1

radians per second was applied at input 2 at the same time. The response to input 1 is obtained at output 1 and is shown in figure 6.3b. The response to input 2 is obtained at output 2 and is shown in figure 6.3c. The coefficient wordlength in both cases was 16 bits and the signal wordlength was 20 bits.

6.6 THE LC LADDER WDF

6.6.1 SIMULTANEOUS PROCESSING OF TWO SIGNALS

Consider the LC ladder WDF structure of figure 6.8a. Since the delay of T seconds separating the three port adaptors is derived from a unit element, it may be split up as discussed before. The structure of figure 6.8a can therefore be redrawn as shown in figure 6.8b.

The maximum sampling frequency possible for direct digital implementations of the structures of figure 6.8 will be the same. For figure 6.8a, the sampling frequency is equal to the clock frequency which is limited by the delay through two adaptors. For figure 6.8b, the sampling frequency is twice the clock frequency which is limited by the delay through one adaptor. However, the direct digital implementation of figure 6.8b can be used to process two signals simultaneously. The penalty for this is 2n–2 additional shift registers where n is the filter order.

Another advantage of the structure of figure 6.8b is that for multiplexed schemes, the three port adaptor can be pipelined. This technique has been discussed in detail for UEWDFs in chapter4. Similar arguments apply to the LC ladder WDF.

6.6.2 PROOF BY SIMULATION

In order to show that the LC ladder WDF of figure 6.8b can be used to process two signals simultaneously, a fifth order filter with the following coefficients was simulated with the modified version of DIFSIP; $\alpha 11 = 1.0$, $\alpha 12 = 0.8701536$, $\alpha 21 = 0.8507776$, $\alpha 22 = 0.8617879$, $\alpha 31 = 0.9003626$, $\alpha 32 = 0.8462301$, $\alpha 41 = 0.7366609$, $\alpha 42 = 0.9895446$, $\alpha 51 = 0.9339637$, $\alpha 52 = 0.8392880$.

The frequency responses for both inputs were obtained and were the same as expected. The frequency response (coefficient wordlength = 16 bits) for input 1 and output 1 is shown in figure 6.9a.

A sinusoidal signal of amplitude 0.001 volts and frequency 0.05 radians per second was applied at input 1 (see figure 6.2b) and a similar signal of frequency 0.1 radians per second was applied at input 2 at the same time. The response to input 1 is obtained at

output 1 and is shown in figure 6.9b. The response to input 2 is obtained at output 2 and is shown in figure 6.9c. The coefficient wordlength in both cases was 16 bits and the signal wordlength was 20 bits.

6.7 DISCUSSION AND CONCLUSIONS

It has been shown that theoretically a discrete system can process any number of signals, skewed by a fraction of the sampling period, simultaneously.

Direct digital implementations of discrete systems which use shift registers clocked every T seconds to realize delays of T seconds behave like the theoretical discrete system at one instant during a sampling period. WDFs based on unit elements employ half delays and therefore direct digital implementations of these employ shift registers clocked twice every sampling period. These systems behave like theoretical discrete systems at two instants during a sampling period. Consequently, they can be used to process two signals, skewed by half the sampling period, simultaneously.

It has been shown that UEWDFs, lattice WDFs based on cascaded unit elements and all–pass sections, and LC ladder WDF can be used to process two signals simultaneously at the same rate as a single signal. Effectively, the efficiency of the hardware implementation is doubled when used in this manner since the number of samples processed per second is doubled.

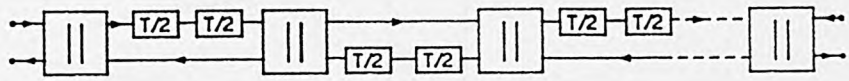


FIGURE 6.1a

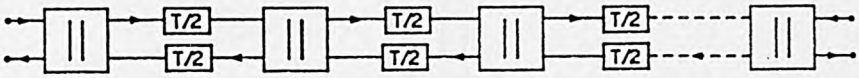


FIGURE 6.1b

FIGURE 6.1 : EQUIVALENT REPRESENTATIONS OF CASCADED UNIT ELEMENTS WDF

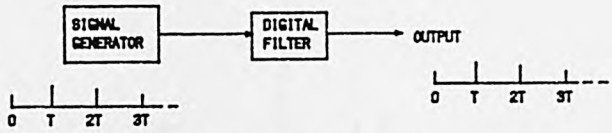


FIGURE 6.2a : PROCESS SIMULATED BY DIFSIP

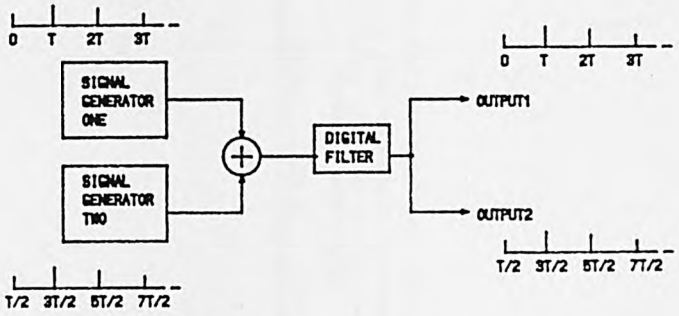


FIGURE 6.2b : PROCESS SIMULATED BY MODIFIED VERSION OF DIFSIP

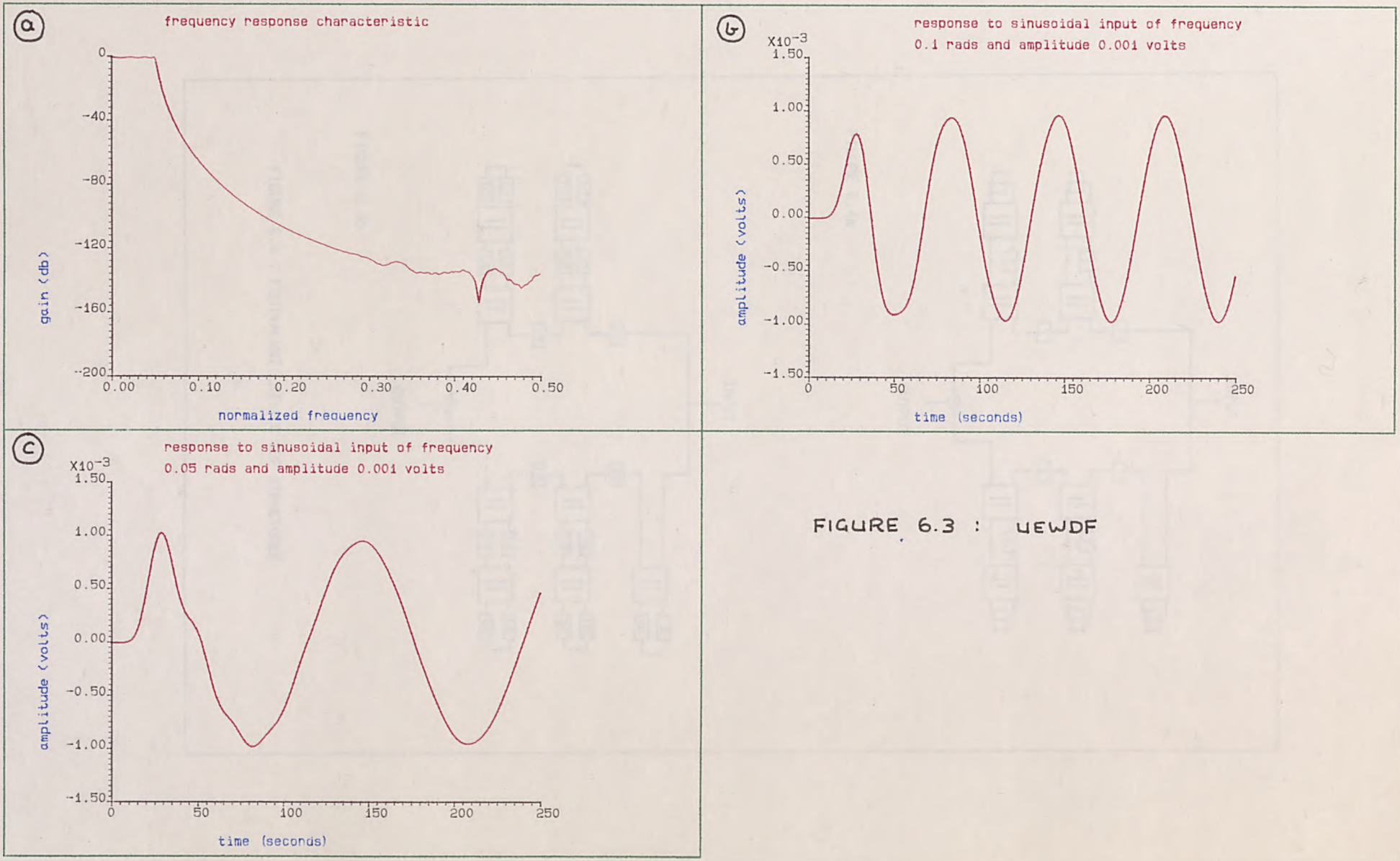


FIGURE 6.3 : UEWDF

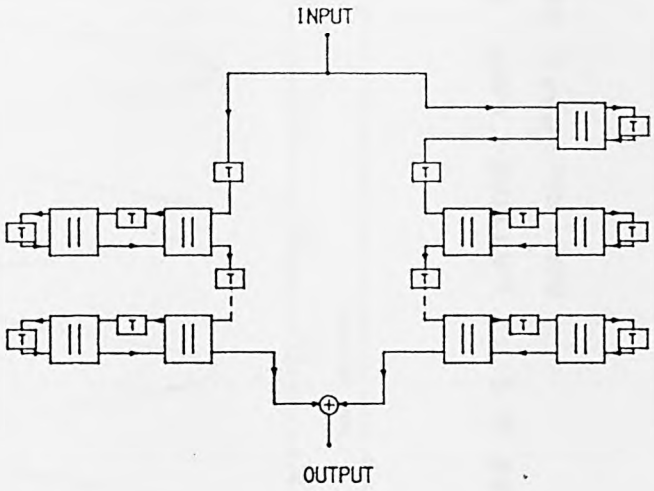


FIGURE 6.4a

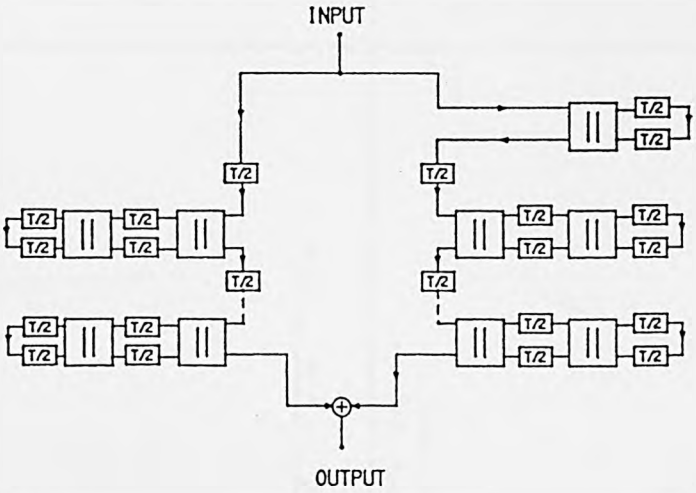


FIGURE 6.4b

FIGURE 6.4 : EQUIVALENT LATTICE WDF STRUCTURES

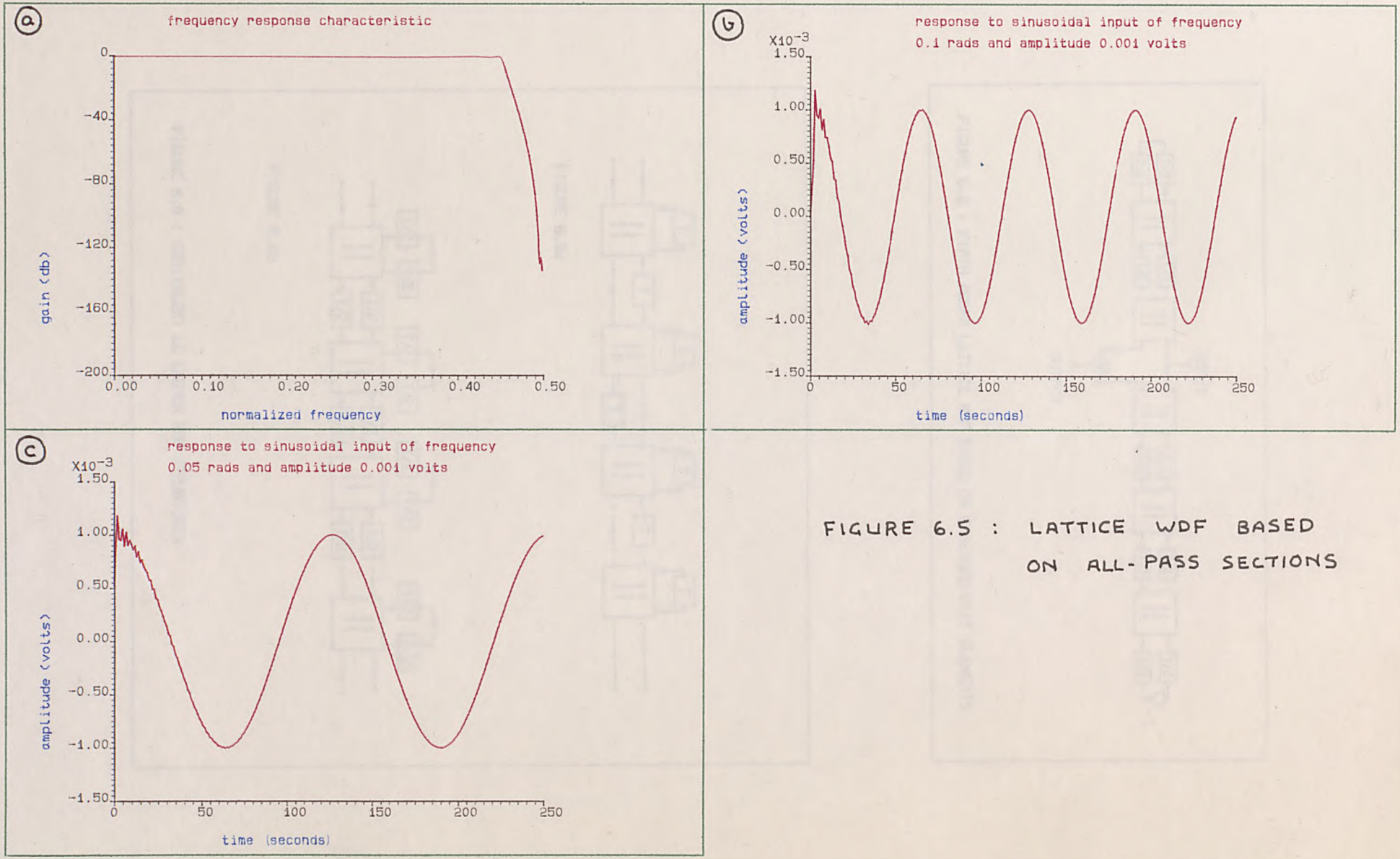


FIGURE 6.5 : LATTICE WDF BASED ON ALL-PASS SECTIONS

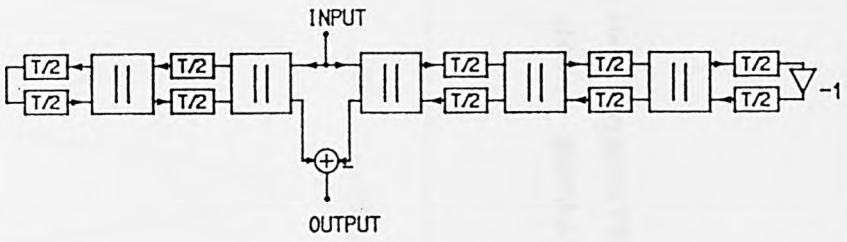


FIGURE 6.6 : FIFTH ORDER LATTICE WDF BASED ON CASCADED UNIT ELEMENTS

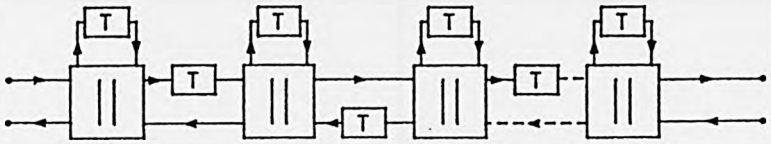


FIGURE 6.8a

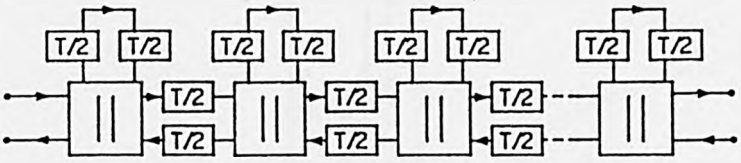


FIGURE 6.8b

FIGURE 6.8 : EQUIVALENT LC LADDER WDF STRUCTURES

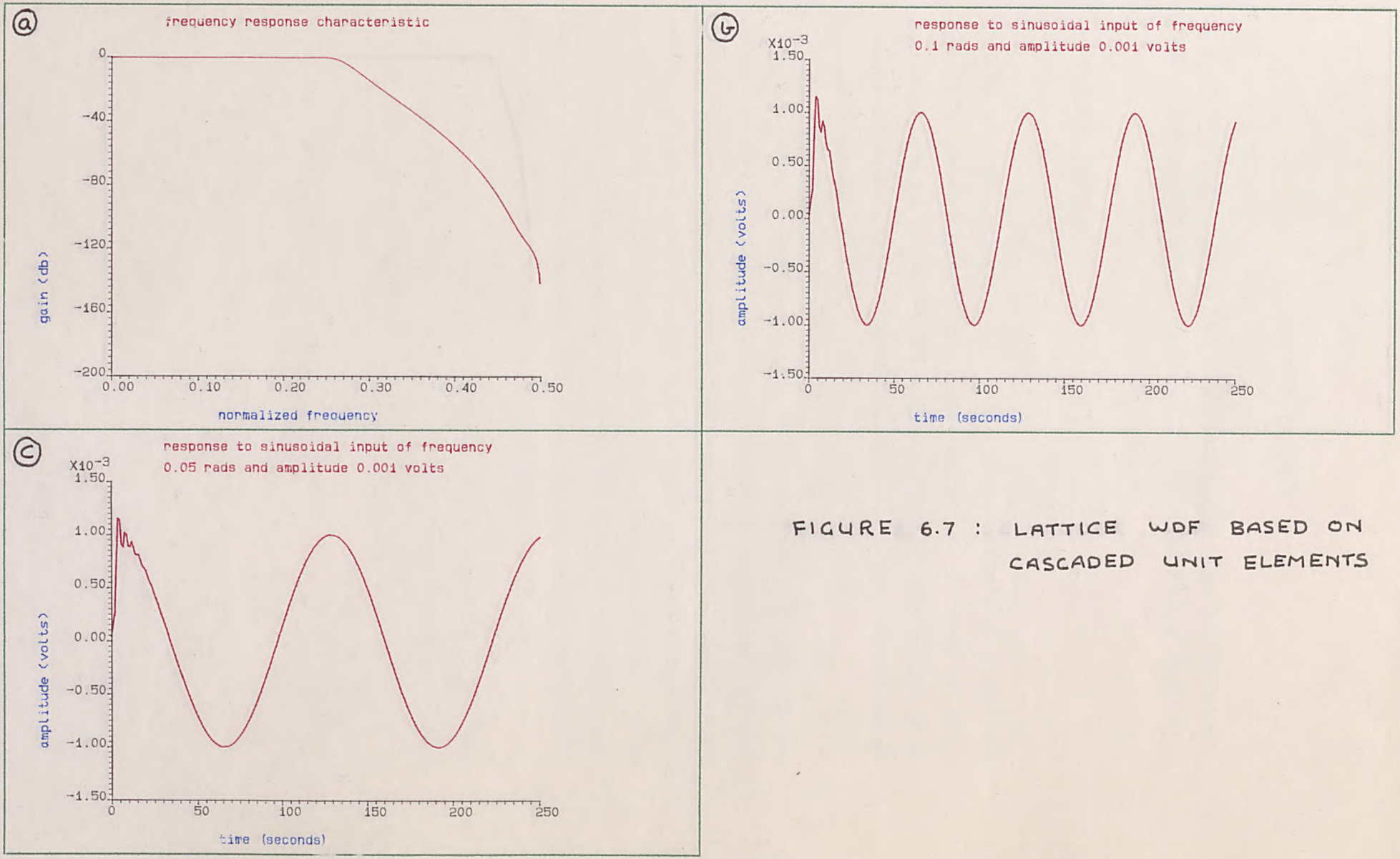


FIGURE 6.7 : LATTICE WDF BASED ON CASCADED UNIT ELEMENTS

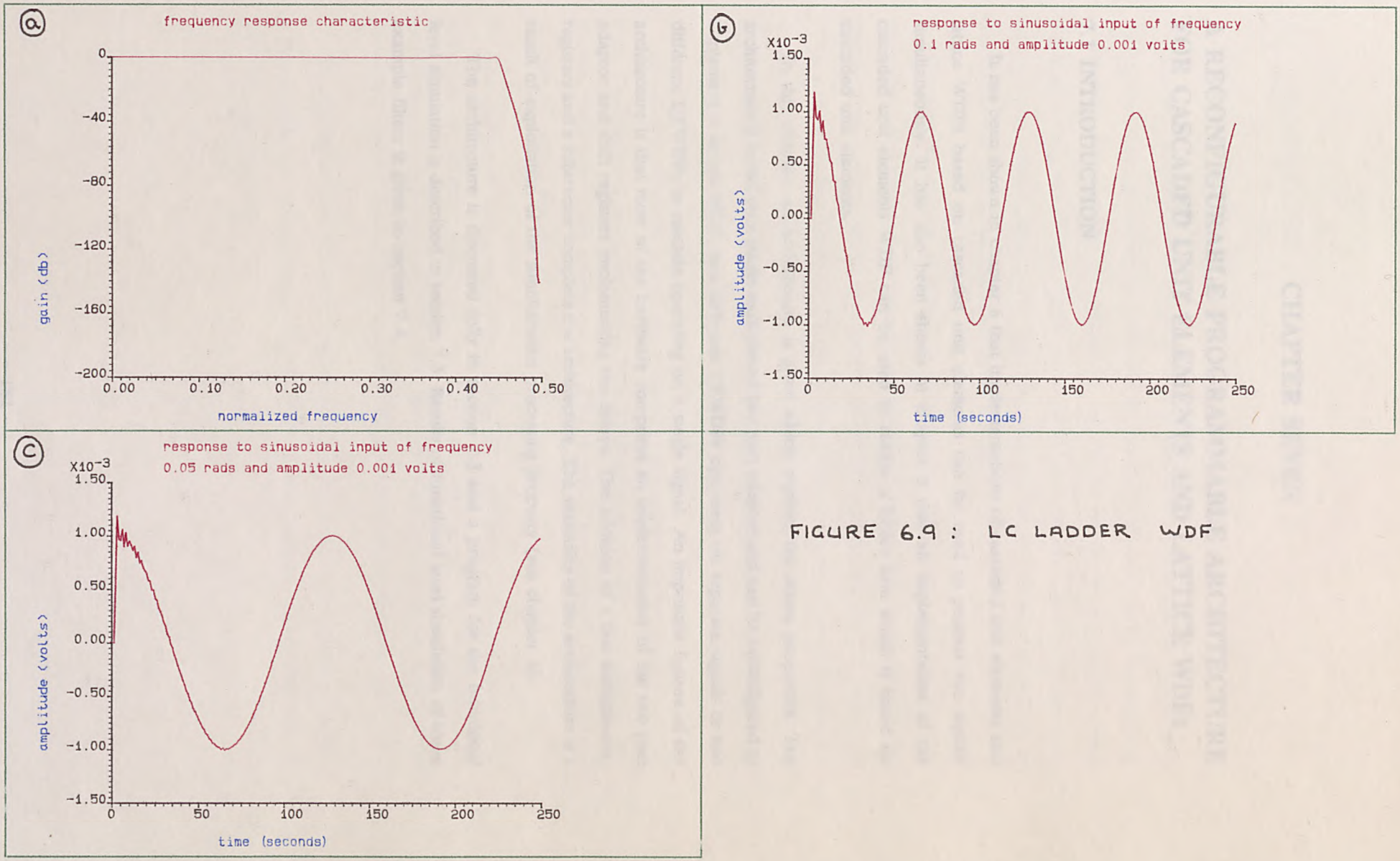


FIGURE 6.9 : LC LADDER WDF

CHAPTER SEVEN

A RECONFIGURABLE PROGRAMMABLE ARCHITECTURE FOR CASCADED UNIT ELEMENTS AND LATTICE WDFs

7.1 INTRODUCTION

It has been shown in chapter 6 that implementations of cascaded unit elements and lattice WDFs based on cascaded unit elements can be used to process two signals simultaneously. It has also been shown in chapter 5 that an implementation of the cascaded unit elements WDF can be used to realize a lattice arm which is based on cascaded unit elements.

In this chapter, an architecture is given which exploits the above properties. The architecture is based on a single multiplexed two port adaptor and can be reconfigured to implement a lattice WDF, two different UEWDFs operating on separate signals or two different UEWDFs in cascade operating on a single signal. An important feature of the architecture is that most of the hardware comprises an implementation of the two port adaptor and shift registers implementing the delays. The addition of a few multiplexers, registers and a subtractor complete the architecture. The versatility of the architecture is a result of exploitation of the simultaneous processing property (see chapter 6).

The architecture is discussed fully in section 7.2 and a program for the functional level simulation is described in section 7.3. Results of functional level simulation of some example filters is given in section 7.4.

7.2 ARCHITECTURE

7.2.1 DESCRIPTION

The architecture is shown in figure 7.1. The pipelined two port adaptor (see chapter 4) implementation comprises the subtractor S1, the two pipeline registers, the multiplier array, the coefficient RAM, the registers A1, A2, C1, C2 and the two adders. The delays are implemented with the two register stacks; stack1 and stack2. Note that for a cascade of n two port adaptors separated by half delays, the number of registers in stack1 is n, in stack2 it is n+1 and in the coefficient RAM it is 2n.

Consider the structure of figure 7.2a. Implementing this structure with a multiplexed two port adaptor involves computing the output values of each of the adaptors in turn (see chapter 4 for a more detailed discussion). For the first adaptor, the input A (see figure 7.2c) is the input signal and for the last adaptor, input C is zero. The multiplexer M3 (figure 7.1) performs the function of selecting zero at the appropriate time and the multiplexer M2 performs the function of selecting the input signal at the appropriate time. At all other times, these multiplexers select the outputs of the register stacks.

The register M is used to store the B value corresponding to the last adaptor in figure 7.2a. The registers K and L are used to store the D value corresponding to adaptors 1 in the two lattice arms of the structure shown in figure 7.2b. The registers Q and R are used to store the output signals.

The multiplexer M4 is used to reconfigure the architecture for either lattice WDF operation or UEWDF operation. The multiplexer M1 is used to reconfigure the architecture for two separate UEWDFs, two UEWDFs in cascade or lattice WDF operation. The subtractor S2 corresponds to the subtractor shown in figure 7.2b.

The two signals to be processed are presented simultaneously. The function of register P is to provide intermediate storage for one of the signals to achieve the required skew of half the sampling period.

7.2.2 CLOCK AND CONTROL SIGNALS

The system is synchronous and all the clock signals are derived from the system clock signal. The relationship between the various clock signals and the system clock signal is shown in table 7.1.

The control signals are also derived from the system clock signal (see table 7.2). The control signal MCON1 is used to switch the output of multiplexer M2 to INPUTA . The control signal MCON2 is used to switch the output of multiplexer M2 to the output of register P. The control signal MCON3 is used to switch the output of multiplexer M3 to zero.

7.2.3 RECONFIGURATION SCHEME

The architecture can be reconfigured to implement three different systems. The reconfiguration scheme is shown in table 7.3.

- In order to implement two different UEWDFs operating on separate signals, the multiplexer M1 is switched to INPUTB and the multiplexer M4 is switched to the output of register M. The two input signals are applied simultaneously to INPUTA and INPUTB and the outputs are obtained from OUTPUTA and OUTPUTB.

- In order to implement a cascade of two different UEWDFs operating on one signal, the multiplexer M1 is switched to OUTPUTA and the multiplexer M4 is switched to the output of register M. The input signal is applied to INPUTA and the output is obtained from OUTPUTA.

- In order to implement a lattice WDF operating on one signal, the multiplexer M1 is switched to INPUTA and the multiplexer M4 is switched to the output of the subtractor. The input signal is applied to INPUTA and the output is obtained from OUTPUTA.

7.2.4 PROGRAMMING

The architecture can be programmed to realize filters of different orders (up to and including N ; n=N+1) and characteristics.

It can be seen from figures 7.2a and 7.2c that in order to realize a UEWDF of order P where P<N, the multiplier coefficients of all adaptors from P+2 to N+1 should be set to zero. It is also clear that to realize a lattice arm of order P (where P<=N), the coefficient of adaptor P+1 is set to +1 if a −1 multiplier is required (see figure 7.2b) and to −1 if a +1 multiplier is required.

The coefficient RAM comprises a circulatory set of 2n registers. The mapping of filter coefficients onto the coefficient RAM is illustrated in table 7.4. Note that for UEWDF implementation, k(1,j) is the jth coefficient for filter 1 and k(2,j) is the jth coefficient for filter 2. For the lattice WDF, k(1,j) is the jth coefficient for one lattice arm and k(2,j) is the jth coefficient for the other lattice arm. The coefficient OS is used to realize the +/− 1 multipliers discussed above.

Note from table 7.4 that the coefficients k(1,j) and k(2,j) have to be interleaved between the two halves of the coefficient RAM.

Assume that the architecture has been reconfigured to implement two separate UEWDFs as explained earlier and that the appropriate coefficients have been loaded into the RAM. The two input signals are applied at INPUTA and INPUTB at the same time. The operation of the system is as follows:

During clock cycle 1 , the INPUTB sample is clocked into register P and the output of multiplexer M2 is switched to INPUTA. The first stage of the pipelined two port adaptor implementation operates on the INPUTA sample. Effectively, during clock cycle 1, the first stage of the pipeline operates on adaptor 1 in figure 7.2a.

During clock cycle 2, the output of multiplexer M2 is switched to the output of stack2. Effectively, during clock cycle 2, the first stage of the pipeline operates on adaptor 2 in figure 7.2a and the second stage of the pipeline operates on adaptor 1. This process is repeated for succeeding adaptors in figure 7.2a.

During clock cycle n−1, the output of multiplexer M3 is switched to zero and during clock cycle n+2, the output sample value is clocked into register M. This value is transferred to registers Q and R at the beginning of alternate half sampling periods, i.e. during the first clock cycle of the first half sampling period, register R is clocked and during the first clock cycle of the second half sampling period, register Q is clocked.

A half sampling period is equal to n+3 clock cycles. From clock cycle n+4 to 2n+6, the processes described above are repeated except that during clock cycle n+4, the output of multiplexer M2 is switched to the output of register P. One sampling period therefore comprises 2n+6 clock cycles where the clock frequency is limited by the pipeline processes.

The operation for two UEWDFs in cascade – assuming the architecture has been appropriately reconfigured and programmed – is the same as above.

Now assume that the architecture has been reconfigured and programmed for lattice WDF operation. This time, a copy of the input sample is stored in register P for use in the second half sampling period. The essential difference between the lattice and UEWDF operation is that the outputs of the two lattice arms are intermediately stored in registers K and L, i.e. during clock cycle 3 of the first half sampling period, register K is clocked and during the third clock cycle of the second half sampling period, register L is clocked. The values stored in these two registers are subtracted and the result is transferred to register R during the first clock cycle of a sampling period.

7.3 FUNCTIONAL LEVEL SIMULATION

A program for the functional level simulation of the architecture under consideration is given in appendix E1. The program is written in fortran 77.

- The system is completely described by the subroutine 'state' in which each register is represented by a real variable and the relationships between the registers are described by mathematical statements. A register is clocked by assigning a new value to it only if the relevant clock signal is active. Note that intermediate variables are used to evaluate the inputs to the registers and that these are evaluated before any of the registers are clocked. This is necessary to ensure that in the software model, the data is correctly shifted through two consecutive registers clocked at the same time.

- All clock and control signals are represented by boolean variables and are derived from the system clock according to tables 7.1 and 7.2. This function is performed by the subroutine 'contrl'.

- During each half system clock cycle, the subroutine 'state' is called to evaluate the state of the registers and the subroutine 'contrl' is called to evaluate the clock and control signal values for that period. The system clock signal is represented by a boolean variable which is toggled after every call to the above two routines.

- The two input signals are supplied by the subroutines 'sigone' and 'sigtwo' at the beginning of each sampling period. Also note that the outputs are available at this time.

- The subroutine 'coflod' is used to program the coefficient RAM.

- The function of the multiplexers is modelled with conditional statements in the subroutine 'state'. Reconfiguration is achieved by appropriate definition of the relevant variables in these statements as can be seen from the program.

- The subroutine 'forier' is used to perform the FFT on the impulse responses of the filters.

7.4 SIMULATION EXAMPLES

The program discussed in the previous section was used to simulate the architecture under the following conditions.

(i) The architecture was reconfigured to implement two separate UEWDFs and programmed for seventh and eleventh order UEWDFs. The coefficients for UEWDF1 were $\alpha1=-\alpha8=$ −0.83553984, $\alpha2=-\alpha7=$ 0.97769094, $\alpha3=-\alpha6=$ −0.98514007, $\alpha4=-\alpha5=$0.98606601. The coefficients for UEWDF2 were $\alpha1=-\alpha12=$−0.70091221, $\alpha2=-\alpha11=$0.91507273, $\alpha3=-\alpha10=$ −0.94217718, $\alpha4=-\alpha9=$ 0.94679689, $\alpha5=-\alpha8$ − 0.948161, $\alpha6=-\alpha7=$ 0.94848824.

. The frequency responses were obtained by performing an FFT on the impulse responses. The frequency response of the eleventh order UEWDF is shown in figure 7.3a and the response of the seventh order UEWDF is shown in figure 7.3b.

(ii) The architecture was reconfigured to implement two UEWDFs in cascade and programmed for the same seventh and eleventh order UEWDFs as in (i).

. The frequency response was obtained as before and is shown in figure 7.3c . Note that theoretically this response should be equal to the sum of the two responses in (i). This is the case except at the lower end where a discrepancy has arisen due to inaccuracy in number representation in the FFT routine. A zoom into the passbands of the frequency responses of figures 7.3a, b, c, is shown in figure 7.3d from which it can be seen that the response in figure 7.3c is equal to the sum of the responses in figures 7.3a and 7.3b.

(iii) The architecture was reconfigured to implement a lattice WDF and programmed for a fifth order lattice WDF. The coefficients for lattice arm 1 were $\alpha1=-0.3046$, $\alpha2=$ 0.2199, and $\alpha3=-1.0$ (note that this implements the +1 multiplier – see section 7.2). The coefficients for lattice arm 2 were $\alpha1=-0.2207$, $\alpha2=$ 0.692, $\alpha3=0.004$ and $\alpha4=1.0$ (note that this implements the −1 multiplier – see section 7.2).

. The frequency response is shown in figure 7.4. The passband gain of 6 dB is due to the omission of a 0.5 multiplier at the output of the lattice WDF.

The above filters were subjected to various signals in the time domain and in each case the result was as expected. An example of this is given in figure 7.5 which shows the response of the cascade of UEWDFs (see (ii) above) to a signal that is a composite of two sinusoids of frequencies 0.2 and 0.5 radians per second. Note that only the sinusoid with the frequency in the passband of the cascade of UEWDFs (see figure 7.3c) is transmitted as expected.

7.5 DISCUSSION AND CONCLUSIONS

A novel architecture has been presented which exploits the simultaneous processing property discussed in chapter 6. The architecture can be reconfigured to implement;

(i) two separate UEWDFs

(ii) two UEWDFs in cascade

(iii) lattice WDF based on cascade of unit elements.

The major part of the architecture is an implementation of the two port adaptor. It has been shown that with little additional hardware, an architecture which is versatile and efficient can be obtained.

A program for the functional level simulation of the architecture has been presented and examples have been given to show that the architecture performs as expected.

A full custom design of the architecture employing the logic circuits discussed in chapter 3 requires approximately 13,000 gates for a fifteenth order system with 8 bit coefficients and 20 bit signals. In VLSI technology, a macrocell of this complexity is possible. The advantages of such a macrocell are that either a single cell or a cascade of cells may be used according to the sampling rate requirements.

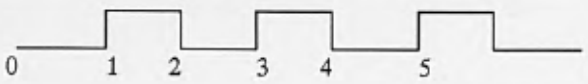
In practice, a dynamic MOS technology would be useful because it allows simple implementation of the registers. The design of adders/subtractors and multipliers will be influenced by the sampling rate requirements. If these are sufficiently low, then bit–serial techniques may be employed to reduce gate counts. For high speed applications, fully parallel techniques must be employed with a consequent increase in silicon area and power requirements.

An important point to note in the architecture is that bidirectional and multiplexed busses have been avoided. This is desirable because of the design complexity of such systems.

For a system of order n, the number of clock cycles required to process one sample is 2n+6. In the pipelining scheme chosen, the multiplier is the most complex block in the

pipeline and will therefore determine the clock frequency. In order to gain an insight into possible sampling frequencies with current technology, consider the multiplier macrocell given by Henlin in reference [95]. This multiplier performs a 16bit X 16bit multiply in 62 nanoseconds and is implemented in 1.5 μm CMOS/BULK technology. The use of such a cell in our scheme would allow a maximum clock frequency of approximately 15 MHz and for a fifteenth order system would allow a maximum sampling frequency of approximately 417 kHz.

It is interesting to note that a comparable 8 stage cascaded second order sections structure implemented on the Hitachi HD61810B signal processor achieves a maximum sampling frequency of approximately 80 kHz.

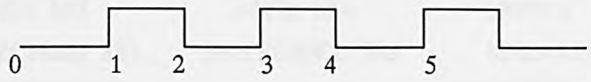


SYSTEM CLOCK SIGNAL

CLOCK SIGNAL	LOW	TOGGLE	LOW	TOGGLE	LOW
CLK1	–	0 : 2n+4	2n+4 : 2n+7	2n+7 : 4n+10	4n+10 : 4n+12
CLK2	–	0 : 4n+12	–	–	–
CLK3	0 : 5	5 : 6	6 : 4n+12	–	–
CLK4	0 : 2n+11	2n+11 : 2n+12	2n+12 : 4n+12	–	–
CLK5	0 : 2n+3	2n+3 : 2n+4	2n+4 : 4n+9	4n+9 : 4n+10	4n+10 : 4n+12
CLK6	0 : 1	1 : 2	2 : 4n+12	–	–
CLK7	0 : 2n+7	2n+7 : 2n+8	2n+8 : 4n+12	–	–
CLK8	0 : 3	3 : 2n+2	2n+2 : 2n+9	2n+9 : 4n+8	4n+8 : 4n+12

n = N+1 where N is the order of UEWDF or lattice arm

TABLE 7.1 : CLOCK SIGNALS



SYSTEM CLOCK SIGNAL

CONTROL SIGNAL	LOW	HIGH	LOW	HIGH	LOW
MCON1	–	0 : 2	2 : 4n+12	–	–
MCON2	0 : 2n+6	2n+6 : 2n+8	2n+8 : 4n+12	–	–
MCON3	0 : 2n–2	2n–2 : 2n	2n : 4n+4	4n+4 : 4n+6	4n+6 : 4n+12

n = N+1 , where N is the filter order

TABLE 7.2 : CONTROL SIGNALS

ARCHITECTURE CONFIGURATION	MUX M1 SWITCHED TO	MUX M4 SWITCHED TO	INPUT SIGNAL ONE	INPUT SIGNAL TWO	OUTPUT SIGNAL ONE	OUTPUT SIGNAL TWO
TWO SEPARATE UEWDFs	INPUT B	REGISTER M	INPUT A	INPUT B	OUTPUT A	OUTPUT B
TWO UEWDFs IN CASCADE	OUTPUT A	REGISTER M	INPUT A	-	OUTPUT B	-
LATTICE WDF	INPUT A	SUBTRACTOR	INPUT A	-	OUTPUT A	-

TABLE 7.3 : RECONFIGURATION SCHEME

	COEFFICIENT VALUES	
COEF RAM LOCATION	UEWDF IMPLEMENTATION	LATTICE WDF IMPLEMENTATION
1	k(1,1)	k(1,1)
2	k(2,2)	k(2,2)
3	k(1,3)	k(1,3)
.	.	.
L	k(1,L)	k(1,L)
L+1	k(2,L+1)	k(2,L+1)
L+2	0	0
.	.	.
M−1	k(2,M−1)	k(2,M−1)
M	0	0
M+1	0	OS
.	.	.
n	0	0
n+1	k(2,1)	k(2,1)
n+2	k(1,2)	k(1,2)
n+3	k(2,3)	k(2,3)
.	.	.
n+L	k(2,L)	k(2,L)
n+L+1	0	OS
n+L+2	k(2,L+2)	k(2,L+2)
.	.	.
n+M−1	0	0
n+M	k(2,M)	k(2,M)
n+M+1	0	0
.	.	.
2n	0	0

L−1 = order of first filter, M−1 = order of second filter, where n>M>L

TABLE 7.4 : MAPPING OF COEFFICIENTS ONTO RAMs FOR LC LADDER WDF OPERATION

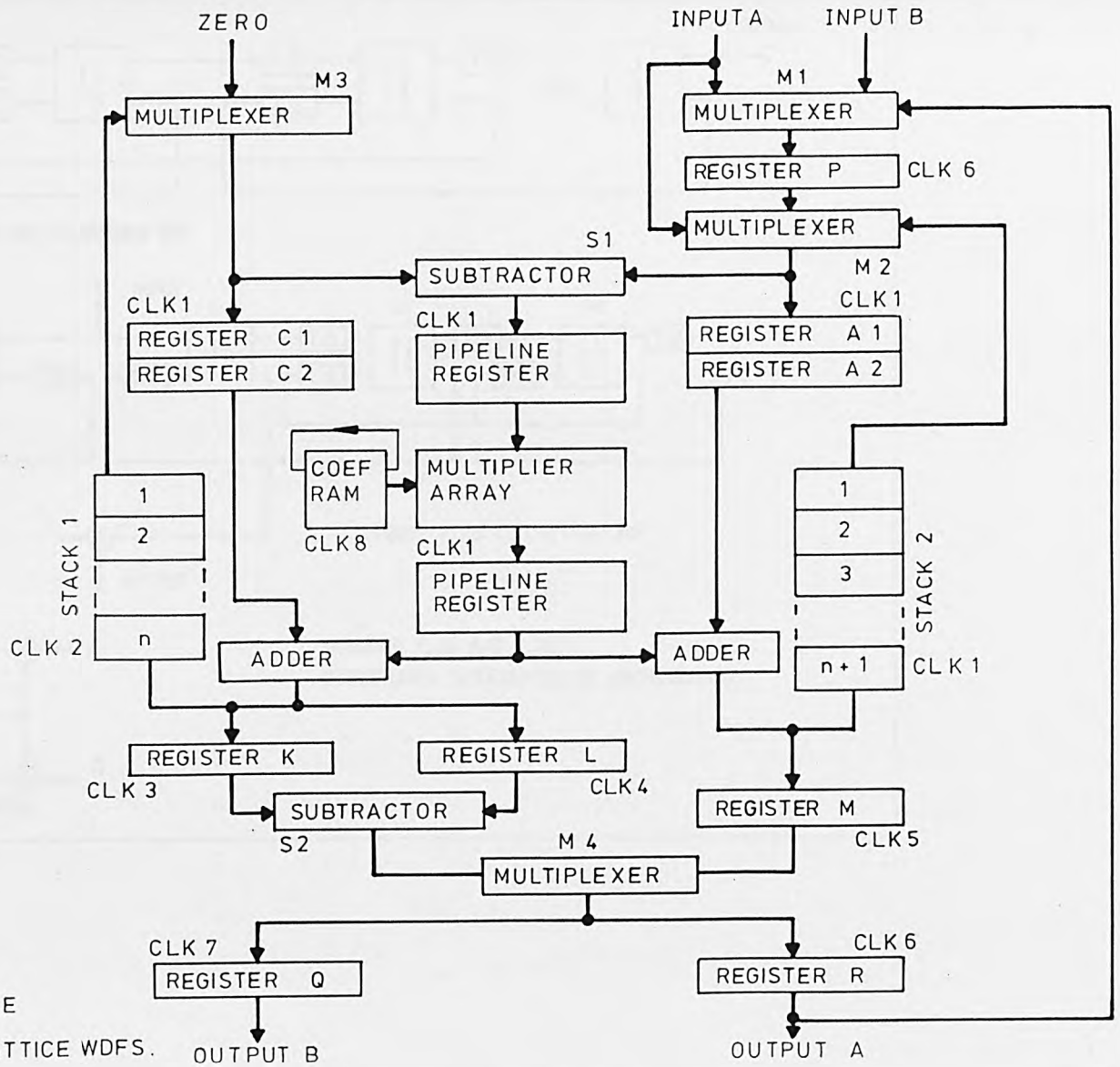


FIGURE 7.1
A RECONFIGURABLE PROGRAMMABLE ARCHITECTURE FOR CASCADED UNIT ELEMENTS AND LATTICE WDFS.

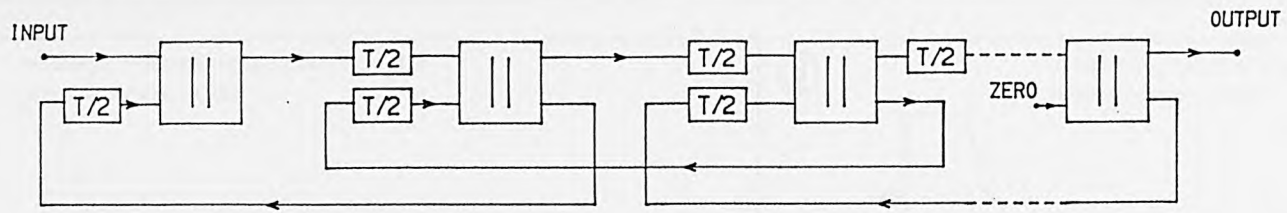


FIGURE 7.2a : CASCADED UNIT ELEMENTS WDF

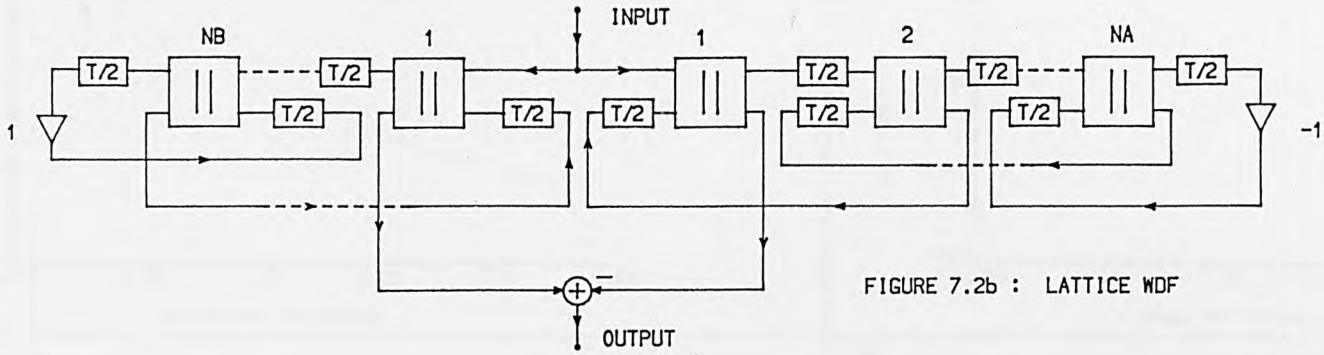


FIGURE 7.2b : LATTICE WDF

FIGURES 7.2a AND 7.2b :
STRUCTURES IMPLEMENTED BY ARCHITECTURE

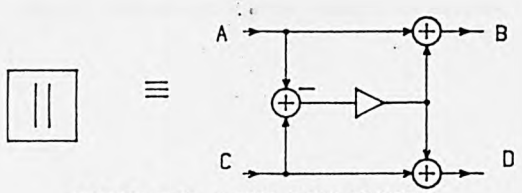


FIGURE 7.2c : TWO PORT ADAPTOR

FIGURE 7.3

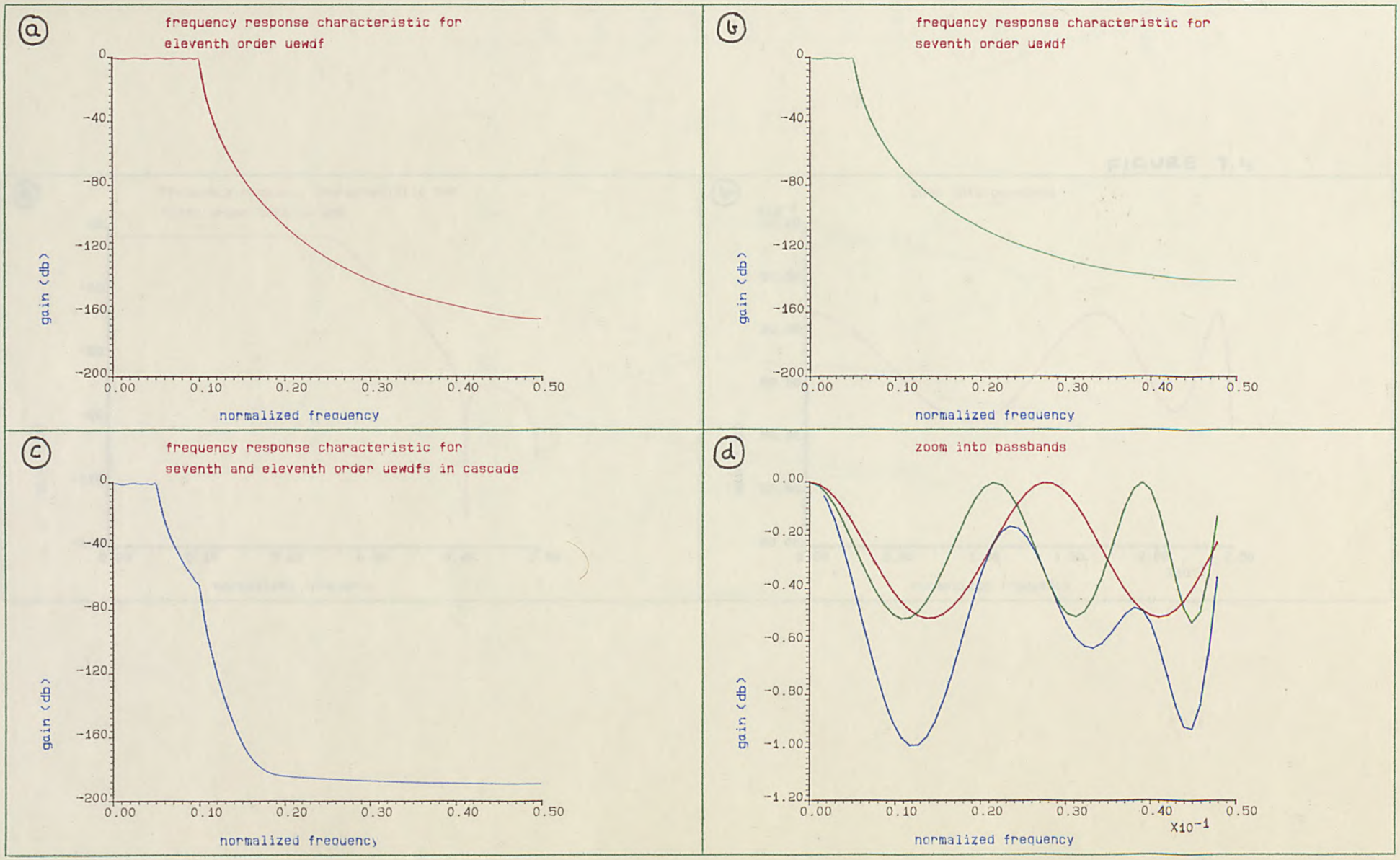
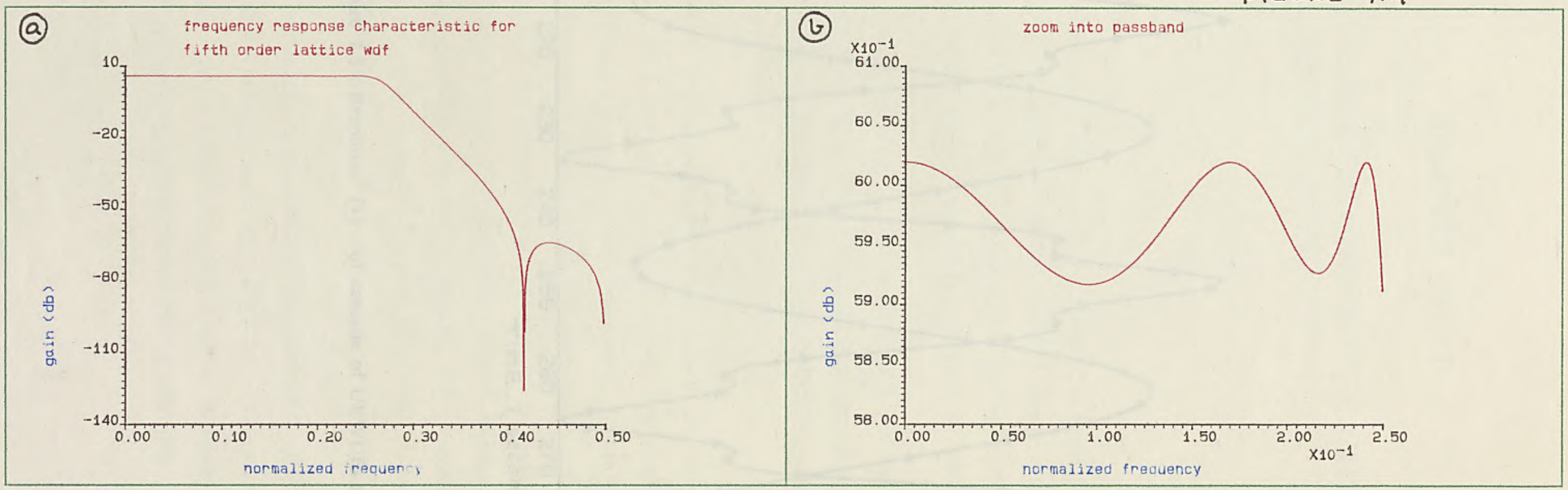

(a)
frequency response characteristic for
eleventh order uewdf
0
-40
-80
-120
-160
-200
gain (db)
0.00
0.10
0.20
0.30
0.40
0.50
normalized frequency
(b)
frequency response characteristic for
seventh order uewdf
0
-40
-80
-120
-160
-200
gain (db)
0.00
0.10
0.20
0.30
0.40
0.50
normalized frequency
(c)
frequency response characteristic for
seventh and eleventh order uewdfs in cascade
0
-40
-80
-120
-160
-200
gain (db)
0.00
0.10
0.20
0.30
0.40
0.50
normalized frequency
(d)
zoom into passbands
0.00
-0.20
-0.40
-0.60
-0.80
-1.00
-1.20
gain (db)
0.00
0.10
0.20
0.30
0.40
0.50
X10-1
normalized frequency

FIGURE 7.4


(a)
frequency response characteristic for
fifth order lattice wdf
10
-20
-50
-80
-110
-140
gain (db)
0.00
0.10
0.20
0.30
0.40
0.50
normalized frequency
(b)
zoom into passband
X10-1
61.00
60.50
60.00
59.50
59.00
58.50
58.00
gain (db)
0.00
0.50
1.00
1.50
2.00
2.50
X10-1
normalized frequency

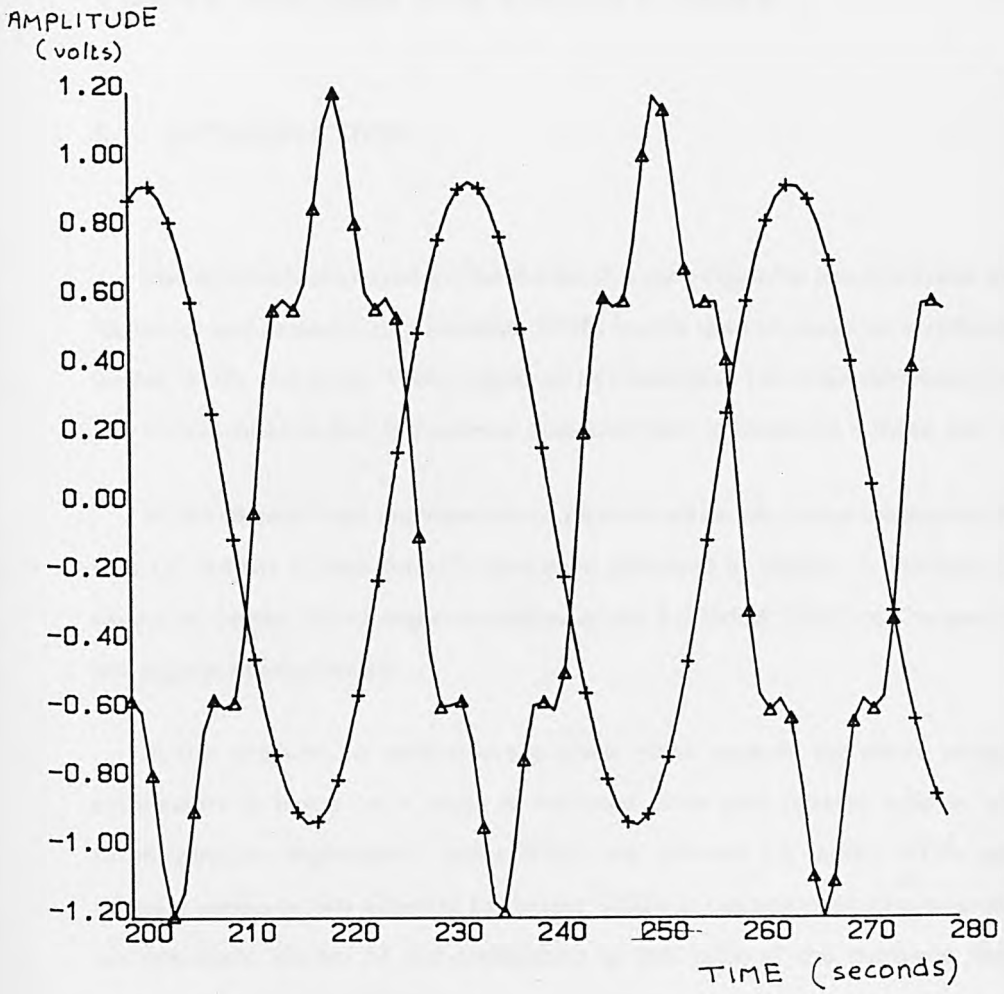


Figure 7.5 : Response (+) of cascade of UEWDFs to input (Δ).

CHAPTER EIGHT

A RECONFIGURABLE PROGRAMMABLE ARCHITECTURE FOR LC LADDER AND LATTICE WDFs

8.1 INTRODUCTION

The approach employed for the design of a reconfigurable programmable architecture for lattice and cascaded unit elements WDFs can be used to design an architecture for LC ladder WDFs and lattice WDFs based on LC networks. The main difference between the two architectures is that the scheme discussed here is based on a three port adaptor.

WDFs derived from analogue lattice filters in which the lattice reactances are realized with LC ladders (Cauer form I) have been discussed in chapter 5. Further, it has been shown in chapter 6 that implementations of the LC ladder WDF can be used to process two signals simultaneously.

In this chapter, an architecture is given which exploits the above properties. The architecture is based on a single multiplexed three port parallel adaptor and can be reconfigured to implement a lattice WDF, two different LC ladder WDFs operating on separate signals or two different LC ladder WDFs in cascade operating on a single signal. An important feature of the architecture is that most of the hardware comprises an implementation of the three port parallel adaptor and shift registers implementing the delays. The addition of a few multiplexers, registers and a subtractor complete the architecture. The versatility of the architecture is a result of exploitation of the simultaneous processing property (see chapter 6).

In section 8.2, a method is described which allows both LC ladder and lattice WDFs to be implemented with a single multiplexed three port adaptor. The architecture is discussed in section 8.3 and a program for functional level simulation is given in section 8.4. Results of functional level simulation of some example filters is given in section 8.5.

8.2 LC LADDER AND LATTICE WDF IMPLEMENTATION BASED ON A SINGLE THREE PORT ADAPTOR

The WDF derived from an LC ladder of Cauer form 1 in which Kuroda's transform has been applied to insert unit elements is shown in figure 8.1 [45].

It has been shown in chapter 3 that this structure is multiplexable and therefore a single multiplexed parallel three port adaptor implementation may be used to realize a filter of arbitrary order.

For a direct implementation of the structure of figure 8.1, all the three port adaptors operate in parallel. This means that we can pipeline the three port adaptor in the multiplexed scheme in order to improve the speed. The pipelined parallel three port adaptor is shown in figure 8.2.

This scheme will form the basis of the architecture to be discussed in section 8.3.

The WDF derived from a lattice filter in which the lattice reactances are realized with one port LC networks is shown in figure 8.3. The lattice reactances are obtained by Cauer synthesis and the unit elements are inserted by using Kuroda's transforms. This type of WDF has been discussed in detail in chapter 5. Note that the WDF is of odd order (figure 8.3) and that the odd order lattice arm comprises only parallel three port adaptors. The even order lattice arm comprises parallel three port adaptors and one two port adaptor. The two port adaptor employed here has one dependent port [80] and is shown in figure 8.4.

The two port adaptor of figure 8.4 can be realized with an implementation of the three port adaptor of figure 8.2 by setting and a3 to zero.

An implementation of the structure of figure 8.1 can be used to realize an mth order lattice arm of figure 8.3 (where m is odd) by setting the coefficients for the mth adaptor to zero. For all subsequent adaptors, $\gamma 1$ is set to -1 and $\gamma 2$ to zero. An even order lattice arm can be realized by setting the $\gamma 2$ coefficient of the mth adaptor to zero and for subsequent adaptors, $\gamma 1$ is set to -1 and $\gamma 2$ to zero. In both cases, the input is applied at INPUT1, INPUT2 is set to zero and the ouput is obtained at OUTPUT2.

It has been shown in chapter 6 that any implementation of the structure of figure 8.1 can be used to process two signals simultaneously. With appropriate modifications, the implementation may be used to realize both the lattice arms. Therefore, a multiplexed scheme based on a single three port adaptor may be used to realize either LC ladder or lattice WDFs.

The above ideas have been used to design a reconfigurable programmable architecture which may be used to implement two separate LC ladder WDFs, two LC ladder WDFs in cascade or a lattice WDF.

8.3 ARCHITECTURE

8.3.1 DESCRIPTION

The architecture is shown in figure 8.5. The pipelined three port parallel adaptor (figure 8.2b) implementation comprises the subtractors S1, S2, the pipeline registers, the multipliers, the adders and the coefficient RAMs. COEF RAM1 holds the coefficients $\gamma 1$ and COEF RAM2 holds the coefficients $\gamma 2$. The multiplier ML3 performs a gating function in that it either multiplies by one or zero. This is necessary in order to implement the condition where a3 is zero. It follows that the COEF RAM3 registers are one bit wide. For a cascade of n three port adaptors separated by half delays (see figure 8.1), the number of registers in the coefficient RAMs is 2n.

The T/2 delays corresponding to the unit elements (see figure 8.1) are implemented with STACK1 and STACK2. The cascaded pair of T/2 delays corresponding to the inductors and capacitors are implemented with STACK3. For a cascade of n three port adaptors separated by half delays, the number of registers in STACK1 is n, in STACK2 it is n+1 and in STACK3 it is 2n+2.

Consider the structure of figure 8.1. Implementing this structure with a multiplexed three port adaptor involves computing the output values of each of the adaptors in turn. For the first adaptor, the input a1 (see figure 8.2) is the input signal and for the last adaptor, input a2 is zero. The multiplexer M3 (see figure 8.5) performs the function of selecting zero at the appropriate time and the multiplexer M2 performs the function of selecting the input signal at the appropriate time. At all other times, the multiplexer M2 selects the output of STACK2 and multiplexer M3 selects the output of STACK1.

The register M is used to store the b2 value corresponding to the last adaptor in figure 8.1. The registers K and L are used to store the b1 value corresponding to adaptors 1 in the two lattice arms of the structure shown in figure 8.3. The registers Q and R are used to store the output signals.

The multiplexer M4 is used to reconfigure the architecture for either lattice WDF or LC ladder WDF operation. The multiplexer M1 is used to reconfigure the architecture for

two separate LC ladder WDFs, two LC ladder WDFs in cascade or lattice WDF operation. The subtractor S3 corresponds to the subtractor shown in figure 8.3.

The two signals to be processed are presented simultaneously. The function of register P is to provide intermediate storage for one of the signals to achieve the required skew of half the sampling period

8.3.2 CLOCK AND CONTROL SIGNALS

The clocking and control schemes used here are the same as in section 7.2.2.

Note that in reading tables 7.1 and 7.2 with respect to the architecture under discussion here, the symbol n represents the order of an LC ladder or a lattice arm.

8.3.3 RECONFIGURATION SCHEME

The reconfiguration scheme used here is the same as in section 7.2.3.

8.3.4 PROGRAMMING

The architecture can be programmed to realize filters of different orders (up to and including n – see figure 8.1) and characteristics.

For an LC ladder WDF of order L where L<n, the number of adaptors and therefore the number of pairs of coefficients ($\gamma 1$ and $\gamma 2$) is L. It can be seen from figure 8.1 that in order to realize such a filter with an nth order system, all $\gamma 1$ coefficients for the (L+1)th adaptor and subsequent adaptors is set to −1. The $\gamma 2$ coefficient for the (L+1)th and subsequent adaptors is set to zero. The output of the Lth order filter is then available at OUTPUT1.

In order to implement two different LC ladder WDFs with the architecture of figure 8.5 where filter 1 is of order L and filter 2 is of order M, the two sets of coefficient values have to be interleaved between the two halves of the coefficient RAMs. This is illustrated in table 8.1. Note that k1(i,j) is the $\gamma 1$ coefficient where i is 1 for filter 1 and 2 for filter 2. The j value represents the number of the adaptor, e.g. k(2,5) is the $\gamma 1$ value of the 5th adaptor of filter 2. Similarly, k2(i,j) represents the $\gamma 2$ value.

8.3.5 OPERATION

Assume that the architecture has been reconfigured to implement two separate LC ladder WDFs as explained earlier and that the appropriate coefficients have been loaded into the RAMs. The two input signals are applied at INPUTA and INPUTB at the same time. The operation of the system is as follows:

During clock cycle 1, the INPUTB sample is clocked into register P and the output of multiplexer M2 is switched to INPUTA. The first stage of the pipelined three port adaptor implementation operates on the INPUTA sample. Effectively, during clock cycle 1, the first stage of the pipeline operates on adaptor 1 in figure 8.1.

During clock cycle 2, the output of multiplexer M2 is switched to the output of STACK2. Effectively, during clock 2, the first stage of the pipeline operates on adaptor 2 in figure 8.1 and the second stage of the pipeline operates on adaptor 1. This process is repeated for succeeding adaptors in figure 8.1.

During clock cycle n−1, the output of multiplexer M3 is switched to zero and during clock cycle n+2, the output sample value is clocked into register M. This value is transferred to registers Q and R at the beginning of alternate half sampling periods, i.e. during the first clock cycle of the first half sampling period, register R is clocked and during the first clock cycle of the second half sampling period, register Q is clocked.

A half sampling period is equal to n+3 clock cycles. From clock cycle n+4 to 2n+6, the processes described above are repeated except that during clock cycle n+4, the output of multiplexer M2 is switched to the output of register P. One sampling period therefore comprises 2n+6 clock cycles where the clock frequency is limited by the pipeline processes.

The operation for two LC ladder WDFs in cascade – assuming the architecture has been appropriately reconfigured and programmed – is the same as above.

Now assume that the architecture has been reconfigured and programmed for lattice WDF operation. This time, a copy of the input sample is stored in register P for use in the second half sampling period. The essential difference between the lattice and LC ladder

WDF operation is that the outputs of the two lattice arms are intermediately stored in registers K and L, i.e. during clock cycle 3 of the first half sampling period, register K is clocked and during the third clock cycle of the second half sampling period, register L is clocked. The values stored in these two registers are subtracted and the result is transferred to register R during the first clock cycle of a sampling period.

8.4 FUNCTIONAL LEVEL SIMULATION

A program for the functional level simulation of the architecture under consideration is given in appendix F1.

The approach taken is the same as in chapter 7 and the discussion of section 7.3 applies here.

8.5 SIMULATION EXAMPLES

The program discussed in the previous section was used to simulate the architecture under the following conditions.

(i) The architecture was reconfigured to implement two separate LC ladder WDFs and programmed for fifth and seventh order LC ladder WDFs. The coefficients are given below.

- 5th order LC ladder WDF :

$\gamma 11 = -1.0$ $\gamma 12 = -0.8701536$

$\gamma 21 = -0.8507776$ $\gamma 22 = -0.8617879$

$\gamma 31 = -0.9003626$ $\gamma 32 = -0.8462301$

$\gamma 41 = -0.7366609$ $\gamma 42 = -0.9895446$

$\gamma 51 = -0.9339637$ $\gamma 52 = -0.8392880$

- 7th order LC ladder WDF

$\gamma 11 = -1.0$ $\gamma 12 = -0.8358278$

$\gamma 21 = -0.8035814$ $\gamma 22 = -0.607597$

$\gamma 31 = -0.5122059$ $\gamma 32 = -0.2721813$

$\gamma 41 = -0.0517625$ $\gamma 42 = -0.0191030$

$\gamma 51 = -0.0146557$ $\gamma 52 = -0.0138348$

$\gamma 61 = -0.0138526$ $\gamma 62 = -0.0147933$

$\gamma 71 = -0.020662$ $\gamma 72 = -0.1653745$

. The frequency responses were obtained by performing an FFT on the impulse responses. The frequency response of the fifth order LC ladder WDF is shown in figure 8.6a and the response for the eleventh order LC ladder WDF is shown in figure 8.6b.

(ii) The architecture was reconfigured to implement two LC ladder WDFs in cascade and programmed for the same seventh and eleventh order LC ladder WDFs as in (i).

. The frequency response was obtained as before and is shown in figure 8.6c . A zoom into the passbands of the frequency responses of figures 8.6a, b, c, is shown in figure 8.6d from which it can be seen that the response in figure 8.6c is equal to the sum of the responses in figures 8.6a and 8.6b.

(iii) The architecture was reconfigured to implement a lattice WDF and programmed for a fifth order lattice WDF. The coefficients for lattice arms are given below

2nd order lattice arm :

$\gamma 11 = -1.0 \quad \gamma 12 = -0.466$

$\gamma 21 = -0.507 \quad \gamma 22 = 0.0$

3rd order lattice arm :

$\gamma 11 = -1.0 \quad \gamma 12 = -0.466$

$\gamma 21 = -0.327 \quad \gamma 22 = -0.711$

. The frequency response is shown in figure 8.7. The passband gain of 6 dB is due to the omission of a 0.5 multiplier at the output of the lattice WDF

8.6 DISCUSSION AND CONCLUSIONS

A novel architecture has been presented which exploits the simultaneous processing property discussed in chapter 6. The architecture can be reconfigured to implement;

(i) two separate LC ladder WDFs

(ii) two LC ladder WDFs in cascade

(iii) lattice WDF based on LC ladder networks.

The major part of the architecture is an implementation of the three adaptor. It has been shown that with little additional hardware, an architecture which is versatile and efficient can be obtained.

A program for the functional level simulation of the architecture has been presented and examples have been given to show that the architecture performs as expected.

A full custom design of the architecture employing the logic circuits discussed in chapter 3 requires approximately 24,000 gates for a fifteenth order system with 8 bit coefficients and 20 bit signals. In VLSI technology, a macrocell of this complexity is possible. The advantages of such a macrocell are that either a single cell or a cascade of cells may be used according to the sampling rate requirements.

In practice, a dynamic MOS technology would be useful because it allows simple implementation of the registers. The design of adders/subtractors and multipliers will be influenced by the sampling rate requirements. If these are sufficiently low, then bit–serial techniques may be employed to reduce gate counts. For high speed applications, fully parallel techniques must be employed with a consequent increase in silicon area and power requirements.

An important point to note in the architecture is that bidirectional and multiplexed busses have been avoided. This is desirable because of the design complexity of such systems.

Note that this architecture requires about twice the number of gates as the architecture presented in chapter 7. However, the LC ladder WDF has lower coefficient

sensitivity than the UEWDF and therefore, in practice, more stringent tolerance scheme requirements can be satisfied with the architecture presented here. Also note from chapter 2 that the LC ladder WDF can be used to satisfy tolerance schemes for which the UEWDF fails. For this reason, both the architecture presented here and that presented in chapter 7 are of practical value since the former can be used where the simpler architecture of chapter 7 fails.

	COEFFICIENT VALUES		
COEF RAM LOCATION	COEF RAM1	COEF RAM2	COEF RAM3
1	k1(1,1)	k2(1,1)	1
2	k1(2,2)	k2(2,2)	1
3	k1(1,3)	k2(1,3)	1
.	.	.	.
L	k1(1,L)	k2(1,L)	1
L+1	k1(2,L+1)	k2(2,L+1)	1
L+2	−1.0	0	0
.	.	.	.
M−1	k1(2,M−1)	k2(2,M−1)	1
M	−1.0	0	0
M+1	−1.0	0	0
.	.	.	.
n	−1.0	0	0
n+1	k1(2,1)	k2(2,1)	1
n+2	k1(1,2)	k2(1,2)	1
n+3	k1(2,3)	k2(2,3)	1
.	.	.	.
n+L	k1(2,L)	k2(2,L)	1
n+L+1	−1.0	0	0
n+L+2	k1(2,L+2)	k2(2,L+2)	1
.	.	.	.
n+M−1	−1.0	0	0
n+M	k1(2,M)	k2(2,M)	1
n+M+1	−1.0	0	0
.	.	.	.
2n	−1.0	0	0

L = order of first filter, M = order of second filter, where n>M>L

TABLE 8.1 : MAPPING OF COEFFICIENTS ONTO RAMs FOR LC LADDER WDF OPERATION

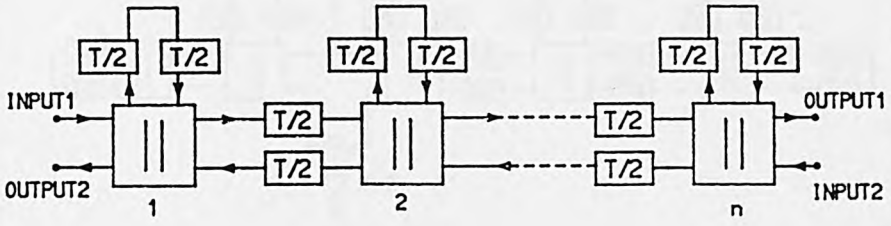


FIGURE 8.1 : LC LADDER WDF

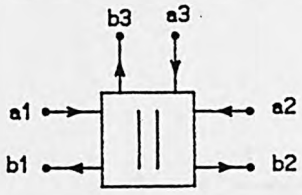


FIGURE 8.2a : SYMBOLIC REPRESENTATION OF 3 PORT PARALLEL ADAPTOR

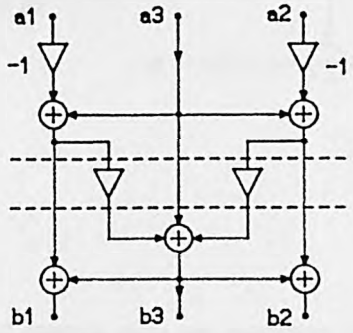


FIGURE 8.2b : PIPELINED 3 PORT PARALLEL ADAPTOR

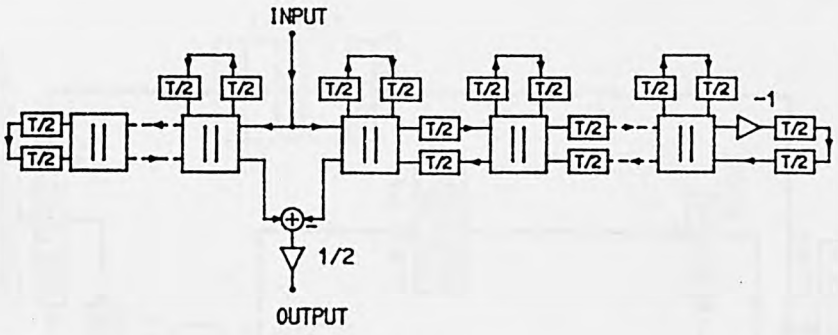


FIGURE 8.3 : WDF DERIVED FROM LATTICE FILTER WITH CAUER SYNTHESIS OF LATTICE REACTANCES

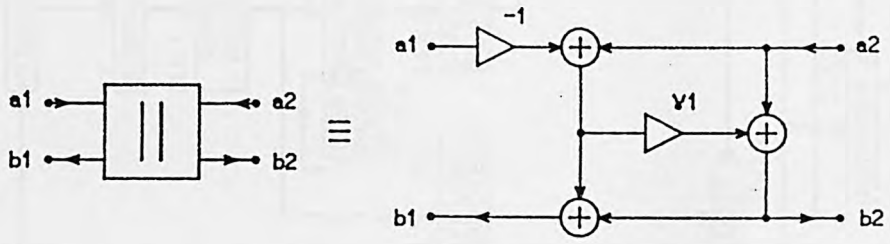


FIGURE 8.4 : TWO PORT ADAPTOR WITH ONE DEPENDENT PORT

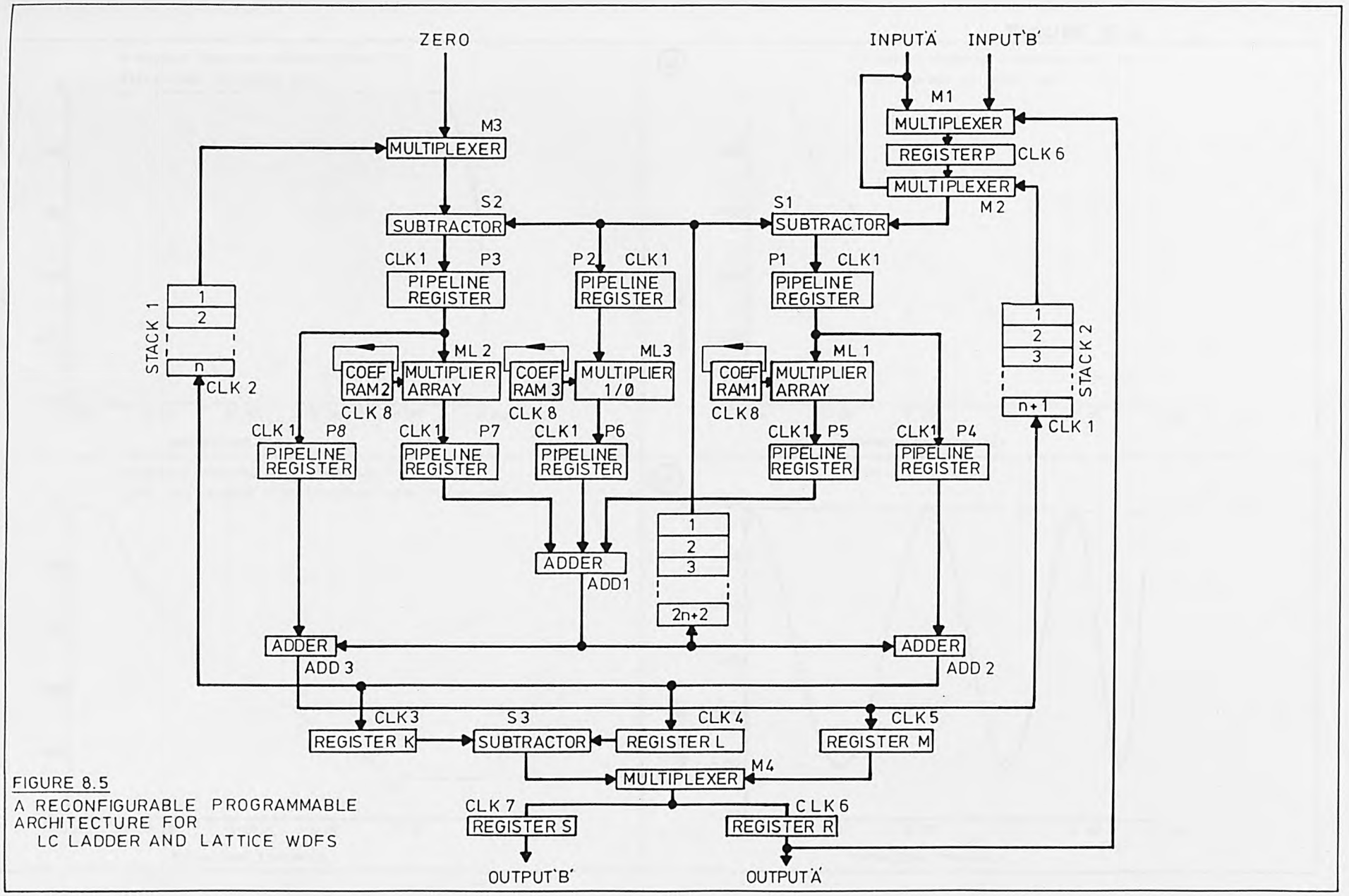


FIGURE 8.5

A RECONFIGURABLE PROGRAMMABLE ARCHITECTURE FOR LC LADDER AND LATTICE WDFS

FIGURE 8.6

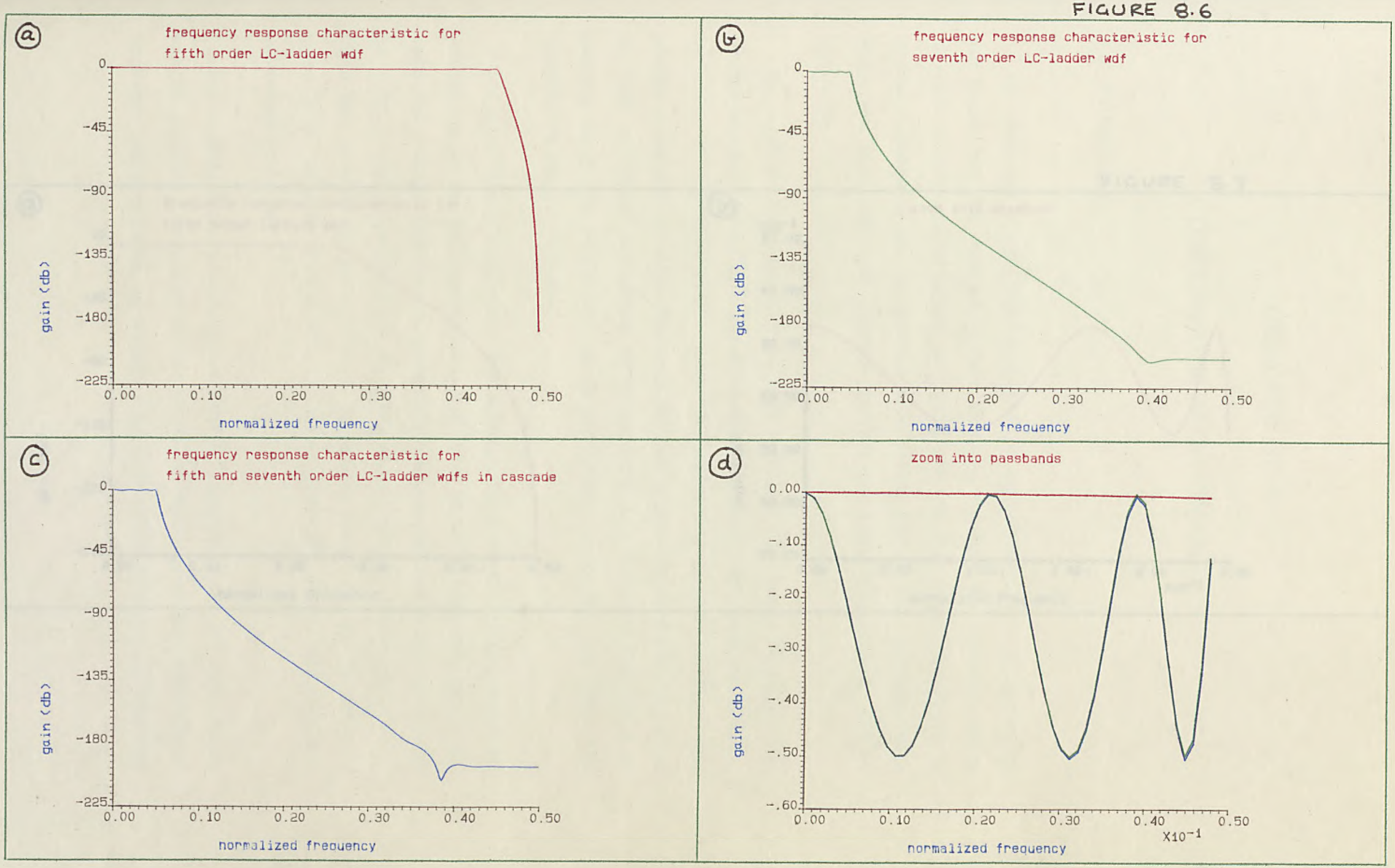
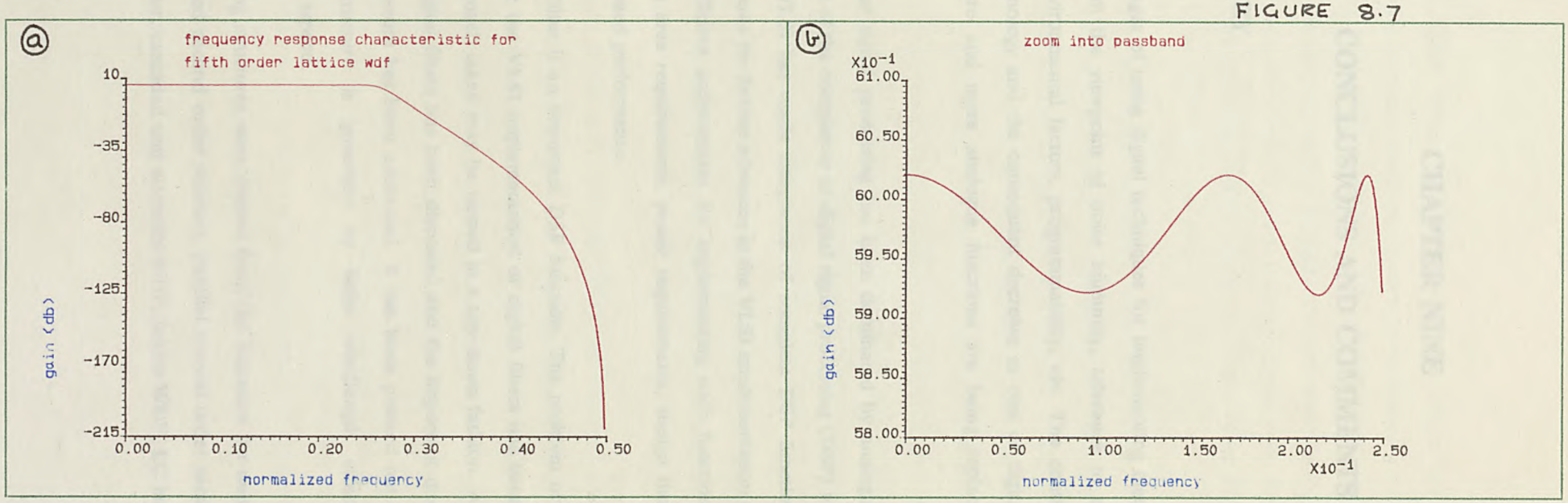

(a)
frequency response characteristic for
fifth order LC-ladder wdf
0
-45
-90
-135
-180
-225
gain (db)
0.00
0.10
0.20
0.30
0.40
0.50
normalized frequency
(b)
frequency response characteristic for
seventh order LC-ladder wdf
0
-45
-90
-135
-180
-225
gain (db)
0.00
0.10
0.20
0.30
0.40
0.50
normalized frequency
(c)
frequency response characteristic for
fifth and seventh order LC-ladder wdfs in cascade
0
-45
-90
-135
-180
-225
gain (db)
0.00
0.10
0.20
0.30
0.40
0.50
normalized frequency
(d)
zoom into passbands
0.00
-.10
-.20
-.30
-.40
-.50
-.60
gain (db)
0.00
0.10
0.20
0.30
0.40
0.50
X10-1
normalized frequency

FIGURE 8.7


(a)
frequency response characteristic for
fifth order lattice wdf
10
-35
-80
-125
-170
-215
gain (db)
0.00
0.10
0.20
0.30
0.40
0.50
normalized frequency
(b)
zoom into passband
X10-1
61.00
60.50
60.00
59.50
59.00
58.50
58.00
gain (db)
0.00
0.50
1.00
1.50
2.00
2.50
X10-1
normalized frequency

CHAPTER NINE

CONCLUSIONS AND COMMENTS

9.1 SUMMARY

The advantages of using digital techniques for implementing electronic systems are well known from the viewpoint of noise immunity, tolerance to component ageing, tolerance to environmental factors, programmability, etc. The continuing advances in fabrication technology and the consequent decrease in cost of digital implementations means that more and more analogue functions are being replaced with digital equivalents.

The realm of signal processing has been dominated by analogue techniques until recently because of the complexity of digital signal processing (DSP) functions. However, the advent of VLSI has made integration of complete DSP functions on single chips possible. A key area for further advances in the VLSI implementation of DSP functions is the study of efficient architectures for implementing such functions. The goal is to minimize silicon area requirements, power requirements, design time and effort, and maximize the speed performance.

The digital filter is an important DSP function. The problem of designing efficient architectures for the VLSI implementation of digital filters has been addressed in this thesis. The approach taken may be viewed in a top-down fashion. At the highest level, the design of digital filters has been discussed and the important question of choice of digital filter structure has been addressed. It has been pointed out that the choice of digital filter structure is governed by finite wordlength effects and hardware implementation aspects.

The following structures were chosen from the literature for detailed study ; direct form II, cascaded second order sections, parallel second order sections, Gray-Markel normalized ladder, cascaded unit elements WDF, lattice WDF, LC ladder WDF and the

IVR generalized WDF. From the finite wordlength effects viewpoint, a detailed comparison of these structures based on experiments performed on a range of filters was given in chapter 2. The experiments were performed by simulating the filters with the aid of a specifically written program and measures for coefficient wordlength sensitivity and signal wordlength sensitivity were obtained. Stability aspects were also considered and it was shown that of the structures considered, the wave digital filters were the most suitable choice from the finite wordlength viewpoint.

The previously mentioned structures were considered from the hardware implementation viewpoint in chapter 3. Measures for hardware complexity were introduced and used to draw a comparison between the structures. The possibility of implementing the structures with multiplexed subsections was considered for each structure because this is an important method of coping with the hardware complexity. The degree of parallelism of each structure was examined and speed aspects were considered. Structures with high speeds are important not only for high speed applications but also because the speed may be traded off against hardware complexity. A measure of the speed performance of the structure was given in terms of the critical path.

Based on the results of chapters 2 and 3, the cascaded unit elements WDF and the lattice WDF were selected for further study and for consideration at the architectural level.

The cascaded unit elements WDF (UEWDF) was discussed in chapter 4. A novel finite wordlength design program based on a heuristic approach was presented. The basis of the program were a number of properties of low–pass UEWDFs which were identified through experimentation and observations drawn from the literature. It was shown with the aid of an arbitrarily chosen example that the technique is of practical value.

An architecture based on a single multiplexed two port adaptor implementation and which is suitable for VLSI implementation was also presented in chapter 4. The novel feature of this architecture is that the inherent parallelism of the UEWDF has been exploited to pipeline the two port adaptor in order to optimize speed performance. It was estimated that a gate count of 12,000 is required for a fifteenth order system with 20 bit signals and 8 bit coefficients. It is thought that a system of this complexity can be used as

a macrocell in a standard cell system. The advantage of the above architecture is that either a single cell, or multiple cascaded cells may be used according to the sampling rate requirements. A functional level simulation of this architecture has been given to show that it is functionally correct.

Lattice WDFs were considered in chapter 5. Design programs for lattice WDFs based on cascaded unit elements and those based on LC networks were given. It was shown that the lattice WDF based on cascaded unit elements can be realized with an implementation of a UEWDF with little or no modifications. The lattice WDF based on all–pass sections was considered at the architectural level and a number of schemes were presented. It was shown that a simple modification to the lattice arm may be used to obtain pipelining. A scheme for this lattice WDF which allows the delay elements to be effectively used in a pipeline in a multiplexed scheme was given. This is thought to be particularly useful for further work.

In chapter 6 arguments were presented to show that the synchronous digital implementation of a discrete system only behaves like the theoretical discrete system at the clocking edge. This argument was developed to show that synchronous digital systems realizing discrete systems with delay elements equal to half the sampling period behave like the theoretical discrete system at two instants within a sampling period. It was shown that this property may be used to process two signals, skewed by half the sampling period, simultaneously. In the case of WDFs employing unit elements, it was shown that these could be used to simultaneously process two signals without requiring any modifications. The implication of this is that the efficiency of the above WDFs can, in effect, be doubled.

The above property was exploited in the design of a reconfigurable, programmable architecture for the implementation of UEWDFs or lattice WDFs and which is suitable for VLSI implementation. This was given in chapter 7. The novel feature of this architecture is that it is based on a single multiplexed two port adaptor which is pipelined for optimum speed performance. The major part of the implementation comprises the two port adaptor and the register stacks for the delays. With little additional hardware, an architecture has been obtained which is capable of implementing two separate UEWDFs, two UEWDFs in

cascade, or a lattice WDF. The gate count for the architecture is approximated at 13,000 gates for a fifteenth order system with 8 bit coefficients and 20 bit signals. A macrocell of this complexity is possible in VLSI. One advantage of such a macrocell is that either a single cell, or a cascade of cells may be used according to sampling rate requirements.

Similar ideas to the above were applied in the design of a reconfigurable, programmable architecture for LC ladder and lattice WDFs. This was discussed in chapter 8. The architecture can be reconfigured to implement two separate LC ladder WDFs, two LC ladder WDFs in cascade, or a lattice WDF. The gate count was approximated at 24,000 for a fifteenth order system with 8 bit coefficients and 20 bit signals. Although the gate count is high, it is still suitable for VLSI implementation. This architecture is useful in practice because the LC ladder WDF is more versatile in the range of tolerance schemes that it can satisfy compared with the UEWDF. It would therefore be useful to have both architectures available.

In the design of all architectures, particular attention was paid to the avoidance of bidirectional and multiplexed busses because of the difficulty in the design of such systems.

Finally, note that all gate counts are based on bit–parallel implementations. The complexity of the logic circuits will be influenced by the speed requirements because high speeds will require fully parallel techniques. However, where speed requirements permit, it is better to use bit–serial techniques in order to optimize silicon area and power requirements.

9.2 FURTHER WORK

The ideas presented in this thesis may be usefully developed further by pursuing the following aspects.

1. The continuing emergence of new digital filter structures means that assessment of these in terms of finite wordlength effects is continually required. The methods used in chapter 2 should be formalized and used to develop software tools to perform a standard set of tests on the structure. A useful addition to the methods in chapter 2 would be a test for limit cycles. A simple method of testing for zero input limit cycles is to stimulate the filter with a unit impulse and observing the response. Failure to settle to zero after the ideal settling time can be taken as indication of a limit cycle. Further practical methods of testing for limit cycles need to be developed. Analytic methods of predicting limit cycles should be incorporated where possible.

   As part of the same set of tools proposed above, the ideas of chapter 3 should be used in a formal way to assess the suitability of a structure for VLSI implementation. In addition to the measures proposed in chapter 3, techniques for assessing the interconnection complexity of a structure need to be developed. The aim of such a measure should be to gain an insight into the silicon area requirements and delays associated with interconnections.

   We can summarize the above discussion by stating that a set of programs may be developed which takes as input a description of the structure and provides as output a set of measures which reflect the suitability of the structure for a particular application and style of implementation.

2. The heuristic program for the design of finite wordlength UEWDFs needs to be developed further. The efficiency of the program presented may be improved by designing more efficient code and incorporating optimization techniques. A further user defined variable representing coefficient wordlength should be incorporated to make the program more versatile from the practical viewpoint.

Optimization procedures should be incorporated into the programs for the design of lattice WDFs based on cascaded unit elements and LC ladder networks in order to make them more suitable for finite wordlength design.

These design programs and design programs for other structures may be incorporated into the software system discussed in 1. A very useful addition to the design suite would be programs for the design of phase equalizers. This would allow the design of linear phase IIR filters by cascading a filter satisfying a particular magnitude response with a filter which has the function of phase equalizing.

3. Detailed VLSI design of the architecture of chapter 7 should be carried out. Optimum pipelinig schemes for the two port adaptor should be examined in the context of a particular method of implementation and performance measures need to be obtained. Application areas should be identified and appropriate cells of varying speeds and complexities should be designed and used to form part of a standard cell system. Similar treatment is required for the architecture of chapter 8.

4. From the preceding discussion and the work presented in this thesis, a framework for the computer aided design of VLSI systems for implementing digital filters can be defined. Essentially, this would comprise two levels. At the higher level, facilities to aid the designer in choosing a particular filter structure would be provided. Once selection is complete, a range of architectures in the form of standard cells would be available to the designer at the lower level. These would then be used to 'build' the required filter. Two novel and efficient architectures which may be used in this way have been presented in this thesis.

Further work is necessary in designing similar architectures for implementing WDFs which are based on passive filters comprising resonant circuits. This would allow the realization of WDFs with elliptic responses. Architectures for other structures which are known to be of practical value should also be incorporated. It is important to ensure that in the design system proposed,

addition of new cells should be a simple process so that the library of cells can be continually updated.

5. In digital filters, the positions of the delays may, in effect, form a pipeline. This is a desirable feature because generally such pipelining ensures a high sampling rate. Pipelining of this form may be obtained by including additional delays as was demonstrated in chapter 5.

In multilplexed schemes, it has been shown in chapters 4,7 and 8 that pipelining of the basic section improves the sampling rate.

A useful area for further study would be to examine direct implementations – i.e. one to one mapping of flow graph onto hardware – of filter structures where the delay element is effectively 'spread' over the other operations. In essence, a number of pipeline registers are spread through the network in such a way that the delay operation becomes implicit.

Clearly, such a highly pipelined architecture will have a high throughput rate but the sampling rate will be slightly lower than in the case where pipelining of this form is not employed. However, note from the discussion of chapter 6 that such an architecture would be able to process a number of signals simultaneously (where the number is determined by the ratio of pipeline registers to number of delays in flowchart). Such a filter would be applicable in situations where the same filtering operation has to be performed on a number of signals, e.g. noise removal on the output of a bank of analogue to digital converters.

The above method was employed in the context of a multiplexed scheme in chapter 5. This technique should be further investigated in a wider context for a range of structures.

6. Digital filters have a wide range of applications [81]. WDFs for a variety of applications in communication systems have been reported particularly in the design of transmultiplexers [82–88].

A requirement for high speed digital filters occurs in sigma–delta modulation systems [89–94]. These systems are used for analogue to digital conversion and are particularly relevant to VLSI systems because the analogue circuitry is simplified at the expense of speed of the digital section. Oversampling at very high speeds is carried out to shape the noise spectrum into a form which is proportional to the frequency squared up to the sampling frequency. The need arises for a high speed digital filter to remove noise at the higher frequencies before decimation of the sampling rate can be carried out. It would be very useful to obtain WDFs which require simple multipliers (powers of half) and provide the necessary rejection at the higher frequencies. High speeds with simple architectures is the main goal and equalization of the passband may be carried out at the lower sampling rate with appropriate WDFs.

REFERENCES

[1] Darlington,S.: 'A history of network synthesis and filter theory for circuits composed of resistors, inductors, and capacitors', IEEE transactions on circuits and systems, vol. CAS-31, pp. 3-13, January 1984.

[2] Temes,G.C., and Mitra, S.K. : 'Modern filter theory and design', John Wiley and sons, 1973.

[3] Hasler,M., and Neirynck, J.: 'Electric filters', Artech house, Inc. 1986.

[4] Orchard,H.J.: 'Inductorless filters', Electronic letters, vol. 2, pp. 224-225, September 1966.

[5] Temes,G.C., and Orchard,H.J: 'First order sensitivity and worst case analysis of doubly terminated reactance two ports', IEEE transactions on circuit theory, vol. CT-20, pp. 567-571, September 1973.

[6] Ghausi,M.S.: 'Analog active filters', IEEE transactions on circuits and systems, vol. CAS-31, pp. 13-31, January 1984.

[7] Oppenheim,A.V., and Schafer,R.W.: 'Digital Signal Processing', Prentice Hall, 1975.

[8] Cadzow,J.A.: 'Discrete-time systems', Prentice Hall 1973.

[9] Antoniou,A.:'Digital filters analysis and design', McGraw-Hill, 1979.

[10] Cappellini,V., Constantinides,A.G., and Emiliani,P.: 'Digital filters and their applications', Academic press, 1978.

[11] Gray,A.H., and Markel,J.D: 'Digital lattice and ladder filter synthesis', IEEE transactions on audio electroacoustics, vol. AU-21, pp. 491-500. December 1973.

[12] - : 'Roundoff noise characteristics of a class of orthogonal polynomial structures', IEEE transactions on acoustics, speech and signal processing, vol. ASSP-23, pp. 473-485, October 1975.

[13] - : 'Fixed point implementation algorithms for a class of orthogonal polynomial filter structures', IEEE transactions on acoustics, speech and signal processing, vol. ASSP-23, pp. 486-495, October 1975.

[14] - : 'A normalized digital filter structure', IEEE transactions on acoustics, speech and signal processing, vol. ASSP-23, pp. 268-277, June 1975.

[15] Fettweis,A.: 'Digital filters related to classical filter networks', Arch. Elecktr. Ubertrag., vol. 25, pp. 79-89, February 1971.

[16] Fettweis,A.: 'Some principles of designing digital filters imitating classical filter structures', IEEE transactions on circuit theory, vol. CT-18, pp. 314-316, March 1971.

[17] Sedlmeyer,A., and Fettweis,A.: 'Digital filters with true ladder configuration', Internaional Journal of circuit theory and applications, vol. 1, pp. 5-10, March 1973.

[18] Lawson,S.S., and Constantinides,A.G.: 'A method for deriving digital filter structures from classical filter networks', proceedings of the IEEE international symposium on circuits and systems, pp. 170-173, 1975.

[19] Lawson,S.S.: 'Digital filter structures from classical analogue networks', Ph.D thesis, University of London, 1975.

[20] Lawson,S.S.: 'On a generalization of the wave digital filter concept', Circuit theory and applications, vol. 6, pp. 107-120, 1978.

[21] Quarmby,D. (editor): 'Signal processor chips', Granada technical books, London, 1984.

[22] Freeny,S.L., Kieburtz,R.B., Mina,K.V., and Tewkesbury,S.K.: 'Design of digital filters for an all digital multiplex-time division multiplex translator', IEEE transactions on circuit theory, vol. CT-18, pp. 702-711, 1971.

[23] Freeny, S.L.: 'Special purpose hardware for digital filtering', Proceedings of the IEEE, pp. 633-648, 1975.

[24] Jackson,L.B., Kaiser,J.F. and McDonald,H.S.: 'An approach to the implementation of digital filters', IEEE transactions on audio and electracoustics, vol. AU-10, pp. 413-421, 1968.

[25] Denyer,P.: 'VLSI signal processing : a bit-serial approach', Addison-Wesley, 1985.

[26] Lyon,R.F.: 'A bit-serial VLSI architectural methodology for signal processing', in VLSI '81, (editor - Gray), Academic press, 1981.

[27] Petrie,N., Mavor,J. and Smith,S.G.: 'General purpose adaptor structure for wave digital filter realization', Electronics letters, vol. 19, pp. 1038-1039, November 1983.

[28] Mavor,J.,Petrie,N., and Lau,C.N.: 'The design of a universal wave filter adaptor using dynamic CMOS techniques', proceedings of the tenth European conference on solid state circuits', Edinburgh, Great Britain, pp. 214-217, September 1984.

[29] Van Ginderdeuren,J.K.J., DeMan,H.J., Goncalves,N.F., and Van Noije,A.M.: 'Compact NMOS building blocks and a methodology for dedicated digital filter applications', IEEE journal of solid state circuits, vol. SC-18, pp. 306-316, June 1983.

[30] Van Ginderdeuren,J.K.J., DeMan,H.J., Catthoor,F., and Beckers,S.: 'A design methodology for compact integration of wave digital filters', proceedings tenth European conference on solid state circuits, Edinburgh, Great Britain, pp. 210-213, September 1984.

[31] Wanhammar,L.: 'Implementation of wave digital filters using distributed arithmetic', Signal Processing, vol. 2, pp. 253-260, July 1980.

[32] Burrus,S.C.: 'Digital filter structures described by distributed arithmetic', IEEE transactions on circuits and systems, vol. CAS-24, pp. 674-680, December 1977.

[33] - : 'Digital filter realization by distributed arithmetic', proceedings IEEE symposium on circuits and systems, Munich, pp.106-109, April 1976.

[34] Peled,A. and Liu,B.: 'A new hardware realization of digital filters', IEEE transactions on acoustics, speech and signal processing, vol. ASSP-22, pp. 456-462, December 1974.

[35] Mead,C.A., and Conway,L.A.: 'Introduction to VLSI systems', Addison-Wesley, 1980.

[36] Kung,H.T.: 'Why systolic arrays?', IEEE computer, vol. C-15, pp. 37-46, 1982.

[37] McCanny,J.V., and McWhirter,J.G.: 'Implementation of signal processing functions using 1-bit systolic arrays', Electronics letters, No. 18, pp. 241-243, 1982.

[38] Fisher, A.L.: 'Special purpose VLSI architectures : general discussion and a case study', in 'VLSI and modern signal processing', edited by Kung,S.Y. et. al., Prentice Hall, pp. 153-169, 1985.

[39] Kung,S.Y.: 'From transversal filter to VLSI wavefront array', Proceedings VLSI '83, Holland, pp. 247-261, 1983.

[40] - : 'On supercomputing with systolic/wavefront array processors', Proceedings of the IEEE, No. 72, pp. 867-884, 1984.

[41] Crochiere,R.E.: 'Digital ladder structures and coefficient sensitivity', IEEE transactions on audio and electroacoustics, vol. AU-20, pp. 240-246, October 1972.

[42] Mirzai,A.R.: 'Finite wordlength design and VLSI implementation of wave digital filters', Ph.D thesis, City University, London, 1986.

[43] Jackson,L.B: 'On the interaction of roundoff noise and dynamic range in digital filters', Digital signal processing, IEEE press, pp.387-412, 1972.

[44] Claasen,T.A.C.M., Mecklenbrauker,W.F.G., and Peek,J.B.H.: 'Effects of quantization and overflow in recursive digital filters', IEEE transactions on acoustics, speech and signal processing, vol. ASSP-24, pp.517-529, December 1976.

[45] Fettweis,A.: 'Wave digital filters : theory and practice', Proceedings of the IEEE, vol. 74, No. 2, pp. 270-327, February 1986.

[46] Shinners,S.M.: 'Modern control system theory and applications', second edition, Addison-Wesley, 1978.

[47] Parker,S.R. and Hess,J.F.: 'Limit-cycle oscillations in digital filters', IEEE transactions on circuit theory, vol. CT-18, pp. 687-397, November 1971.

[48] Jackson,L.B.: 'An analysis of limit cycles due to multiplication rounding digital (sub)filters', Proceedings of the 7th Allerton coference on circuit system theory, pp. 69-78, 1969.

[49] Ebert,P.M., Mazo,J.E., and Taylor,M.G.: 'Overflow oscillations in digital filters', Bell systems technical journal, vol. 48, November 1969, pp. 2999-3020.

[50] Belter,S.E. and Bass,S.C.: 'Computer aided analysis and design of digital filters with arbitrary topology', IEEE transactions on circuits and systems, vol. CAS-22, pp. 810-819, October 1975.

[51] DINAP II user manual, Purdue university, Indiana, U.S.A.

[52] Numerical algoriths group: 'The NAG fortran library', Oxford, England.

[53] Sandberg,I.W. and Kaiser,J.F.: 'A bound on limit cycles in fixed-point implementations of digital filters', IEEE transactions on audio and electroacoustics, vol. AU-20, pp.110-114, June 1972.

[54] Long,J.L. and Trick, T.N.: ' An absolute bound on limit cycles due to roundoff errors in digital filters', IEEE transactions on audio and electroacoustics, vol. AU-21, pp. 27-30, February 1973.

[55] Chang, T.L.: ' A note on upper bounds on limit cycles in digital filters', IEEE tranactions on acoustics, speech and signal processing, vol. ASSP-24, pp.99-100, February 1976.

[56] Hitachi digital signal processor (HSP) HD61810B user's manual, Hitachi.

[57] Hossein,Y.: 'Digital filtering with the IMSA100', IMSA100 application note 1, INMOS.

[58] Richards,P.I.: 'Resistor-transmission line circuits', Proceedings IRE, 36, pp. 217-220, February 1948.

[59] Ozaki,H. and Ishii,J.: 'Synthesis of a class of strip-line filters', IRE transactions on circuit theory, vol. CT-5, pp. 104-109, June 1958.

[60] Levy,R.: 'Tables of element values for the distributed low-pass prototype filter', IEEE transactions on microwave theory and techniques, MTT-13, vol. 5, pp. 514-536, September 1965.

[61] Lawson,S.S.: 'Computer simulation and implementation of a wave digital filter', Proceedings of the IEE colloquium on electronic filters, London, May 1977.

[62] Carlin,H.J.: 'Distributed circuit design with transmission line elements', IEEE Proceedings, 59, pp. 1059-1081, 1971.

[63] Leich,H.: 'On the reduction of the wordlength coefficients for wave digital filters related to LC ladder network', IEEE Proceedings of symposium on circuits and systems, pp. 469-472, 1976.

[64] Wegener,W.: 'Design of wave digital filters with very short coefficient wordlengths' IEEE proceedings of symposium on circuits and systems, pp. 473-476, 1976.

[65] - : 'On the design of wave digital lattice filters with short coefficient wordlengths and optimal dynamic range', IEEE transactions on circuits and systems, vol. CAS-25, No.12, pp. 1091-1098, December 1978.

[66] Charalambous,C. and Best,M.J.: 'Optimization of recursive digital filters with finite wordlengths', IEEE transactions on acoustics, speech and signal processing, ASSP-22, pp. 424-431, 1974.

[67] Avenhaus,E.: 'On the design of digital filters with coefficients of limited wordlength', IEEE transactions on audio and electroacoustics, vol. AU-20, pp. 206-212, August 1972.

[68] Steiglitz,K.: 'Designing short wordlength recursive digital filters', proceedings of the 9th Allerton conference on circuit and system theory, pp. 778-788, October 1971.

[69] Suk,M. and Mitra S.K.: 'Computer aided design of digital filters with finite wordlengths', IEEE transactions on audio and electroacoustics, AU-20, pp.356-363, December 1972.

[70] Lawson,S.S.: 'On the hardware implementation of a multiplexable wave digital filter structure', Proceedings IEEE symposium on circuits and systems, Houston, Texas, pp. 124-127, April 1980.

[71] Lawson,S.S: 'Wave digital filter hardware structure', IEE proceedings, vol.128, Pt. G, No. 6, pp. 307-310, December 1981.

[72] Lawson,S.S. and Terplan,S.: 'High speed single board flexible hardware implementation of a wave digital filter', IEE Proceedings, vol. 131, Pt. G, No. 1, pp. 17-23, February 1984.

[73] Sikstrom,B.: 'Some aspects on the LSI implementation of wave digital filters', Proceedings IEEE symposium on circuits and systems, Rome, vol.3, pp. 780-783, May 1982.

[74] Cardoso,A. and Jabbour,K.: 'High speed multiplierless wave digital filter', 20th Asilomar conference on signals, systems and computers, California, pp. 38-42, November 1986.

[75] HILO III user manual, GENRAD.

[76] Gazsi,L.: 'Explicit formulas for lattice wave digital filters', IEEE transactions on circuits and systems, vol. CAS-32, pp. 68-88 , January 1985.

[77] Sedra,A.S. and Brackett,P.O.: 'Filter theory and design : active and passive', Matrix publishers, 1978.

[78] Guillemin,E.A.: 'Synthesis of passive networks', John Wiley and sons, 1957.

[79] Belevitch,V.: 'Classical network theory', Holden-Day, 1968.

[80] Fettweis,A. and Meerkotter,K.: 'On adaptors for wave digital filters', IEEE transactions on acoustics, speech and signal processing, vol. ASSP, pp. 516-525, December 1975.

[81] Oppenheim,A.V. (editor): 'Applications of digital signal processing', Englewood cliffs, NJ, Prentice-Hall, 1978.

[82] Fettweis,A., Mandeville,G.J. and Kao,C.Y.: 'Design of wave digital filters for communication applications', proceedings IEEE symposium on circuits and systems, Boston, USA, pp. 162-165, April 1975.

[83] Gazsi,L. and Fettweis,A.: 'VLSI realization of wave digital filters for communication systems', proceedings IEEE symposium on circuits and systems, Rome, Italy, pp. 272-275, May 1982.

[84] Scheuermann,H. and Gockler,H.: 'A modular approach to a digital 60-channel transmultiplexer using directional filters', IEEE transactions on communiactions, vol. COM-30, pp. 1598-1613, July 1982.

[85] Fettweis,A.: 'Transmultiplexers with either analog conversion circuits, wave digital filters, or SC filters - A review', IEEE transactions on communications, vol. COM-30, pp. 1575-1586, July 1982.

[86] Gazsi,L.: 'DSP based implementation of a transmultiplexer using wave digital filters', IEEE transactions on communications, vol. COM-30, pp.1587-1597, July 1982.

[87] Gazsi,L.: 'Single chip filter bank with wave digital filters', IEEE transactions on acoustics, speech and signal processing, vol. ASSP-30, pp. 709-718, October 1982.

[88] Nossek,J.A., and Schwartz,H.D.: 'Wave digital lattice filters with applications in communication systems', proceedings IEEE symposium on circuits and systems, Newport beach, CA, May 1983.

[89] Ardalan,S.H. and Paulos,J.J.: 'An analysis of non-linear behaviour in delta-sigma modulators', IEEE transactions on circuits and systems, vol. CAS-34, pp. 593-603, June 1987.

[90] Misawa,T., Iwersen,J.E., Loporcaro,L.J., and Ruch,J.G.: 'Single-chip per channel codec with filters using $\Delta-\Sigma$ modulation', IEEE journal of solid state circuits, vol. SC-16, pp. 333-340, August 1981.

[91] Fox,J.R. and Garrison,G.J.: 'Analog to digital conversion using sigma-delta modulation and digital signal processing', 1982 conference on advanced research in VLSI, M.I.T., pp. 101-112, 1982.

[92] Candy,J.C.: 'Decimation for sigma-delta modulation', IEEE transactions on communications, vol. COM-34, pp. 72-76, January 1986.

[93] - : 'A use of double integration in sigma delta modulation', IEEE transactions on communications, vol. COM-33, pp. 249-258, March 1985.

[94] Matsuya,Y., et. al. : 'A 16-bit oversampling A-to-D conversion technology using triple-integration noise shaping', IEEE journal of solid state circuits, vol. SC-22, pp. 921-929, December 1987.

[95] Henlin,D.A., Fertsch,M.T., Mazin,M.K. and Lewis,E.T.: 'A 16 bit X 16 bit pipelined multiplier macrocell', IEEE journal of solid state circuits, vol. SC-20, pp. 542-547, April 1985.

APPENDIX A1

```
C-----------------------------------------------------------------------
C
C               CENTRE FOR INFORMATION ENGINEERING
C                    CITY UNIVERSITY
C               NORTHAMPTON SQUARE, LONDON EC1V 0HB
C
C
C     DIGITAL FILTER SIMULATION PROGRAM - DIFSIP
C
C
C     AUTHOR: P.S.MATHARU       DATE: FEBRUARY 1986
C
C
C     ACKNOWLEDGEMENT: THE CENTRAL SIMULATION ALGORITHM FOR THIS
C                      PROGRAM IS TAKEN FROM THE PURDUE UNIVERSITY
C                      DIGITAL NETWORK ANALYSIS PROGRAM - DINAP
C
C-----------------------------------------------------------------------
C
      INTEGER SIGTYP,SIMLEN,SIGPCN,T0,T1,TEMP,RANPTR,ADDTYP
      INTEGER TIMSTP,ANTYP,COFSEL,TERMTP
      REAL TRNFP,TRNIP,IPIP
C
      COMMON NETWOR(100,4),COEFF(50,2),COFRND(50),NUMULT,NUMNWK
      COMMON/ONE/NODES(100,3),T0,T1,MAXNUM
      COMMON/TWO/TIME(1024),TRNIP(1024),TRNFP(1024),TRNOIS(1024),SIMLEN
      COMMON/THREE/IPIP(1024),FPIP(1024),SIGPCN,ATDTYP
      COMMON/FOUR/RAND(50),RANPTR,OVFLOW
      COMMON/FIVE/MTYP,ADDTYP,MULTOF,COFSEL,COFPCN,TERMTP,ANTYP,NODOUT
      COMMON/SIX/INTEGR,VOLTAG,SIGTYP,SIGAMP,TIMSTP,RESN,RMPRTE,SIGFR
      COMMON/TEN/NETDPN
C
C
      OPEN(7,FILE='NET')
      OPEN(8,FILE='TRN-NET')
      OPEN(10,FILE='TRN-OUT')
      OPEN(11,FILE='CHECK')
      OPEN(12,FILE='FRN-OUT')
      REWIND(7)
      REWIND(8)
      REWIND(10)
      REWIND(11)
      REWIND(12)
      NETLOD=0
      CALL DEFLT
C
 10   PRINT*,' '
      PRINT*,'                          MAIN MENU'
      PRINT*,'                          ---------'
      PRINT*,'                     1: AUTO FREQUENCY ANALYSIS'
      PRINT*,'                     2: FREQUENCY ANALYSIS'
      PRINT*,'                     3: AUTO TRANSIENT ANALYSIS'
      PRINT*,'                     4: TRANSIENT ANALYSIS'
      PRINT*,'                     5: INPUT NEW NETWORK'
      PRINT*,'                     6: QUIT'
      READ*,MENU1
C
      IF (MENU1.EQ.5) NETDPN=1
      IF (MENU1.EQ.5) CALL NETINP
      IF ((MENU1.NE.5).AND.(NETLOD.EQ.0)) CALL NETINP
      NETLOD=1
      IF (MENU1.EQ.1) CALL FRAUTO
      IF (MENU1.EQ.2) CALL FRNOUT
      IF (MENU1.EQ.3) CALL TRAUTO
      IF (MENU1.EQ.4) CALL TRNOUT
      IF (MENU1.EQ.6) STOP
C
      GO TO 10
C
      END
C
C**********************************************************************
C
C     ANALYSIS MANAGEMENT SUBROUTINE.
C     MANAGEMENT SUBROUTINE FOR CARRYING OUT ANALYSIS. CALLS THE
C     SIGNAL SUBROUTINE TO SUPPLY SIGNAL VALUE AND LOWER LEVEL
C     SIMULATION ROUTINES TO EVALUATE NODE VALUES AT A GIVEN TIME
C     INSTANT. THE PROCESS IS REPEATED OVER SPECIFIED SIMULATION
C     LENGTH
C
C-----------------------------------------------------------------------
C
      SUBROUTINE ANMAN
C
      INTEGER SIGTYP,SIMLEN,SIGPCN,T0,T1,TEMP,RANPTR,ADDTYP
      INTEGER TIMSTP,ANTYP,COFSEL,COFPCN,OVFLOW
      REAL TRNFP,TRNIP,IPIP
      DOUBLE PRECISION ANODES
C
      COMMON NETWOR(100,4),COEFF(50,2),COFRND(50),NUMULT,NUMNWK
      COMMON/ONE/NODES(100,3),T0,T1,MAXNUM
      COMMON/TWO/TIME(1024),TRNIP(1024),TRNFP(1024),TRNOIS(1024),SIMLEN
      COMMON/THREE/IPIP(1024),FPIP(1024),SIGPCN,ATDTYP
      COMMON/FOUR/RAND(50),RANPTR,OVFLOW
      COMMON/FIVE/MTYP,ADDTYP,MULTOF,COFSEL,COFPCN,TERMTP,ANTYP,NODOUT
      COMMON/SIX/INTEGR,VOLTAG,SIGTYP,SIGAMP,TIMSTP,RESN,RMPRTE,SIGFR
      COMMON/NINE/ANODES(100,2)
C
      OVFLOW=0
C
C     INITIALISE ARRAYS NODES AND ANODES
C
      DO 20 LP1=1,100
         NODES(LP1,1)=0
         NODES(LP1,2)=0
         NODES(LP1,3)=0
         ANODES(LP1,1)=0.0D0
         ANODES(LP1,2)=0.0D0
 20   CONTINUE
C
C     INITIALISE NAG RANDOM NUMBER GENERATOR
```

```
C
      CALL G05CBF(0)
C
C     IF INFINITE PRECISION COEFFS LOAD COEFF TO COFRND
C
      IF (COFSEL.EQ.1) GO TO 40
      DO 30 LP2=1,NUMULT
         COFRND(LP2)=COEFF(LP2,1)
 30   CONTINUE
 40   IF (COFSEL.EQ.1) CALL ROUND
C
C     CALCULATE SMALLEST FRACTION THAT CAN BE REPRESENTED
C
      MAXNUM=2**(SIGPCN-1)
      RESN=1/FLOAT(MAXNUM)
C     PRINT*,RESN
C
C     INITIALISE ARRAY POINTERS
C
      T0=1
      T1=2
C
C     TRANSIENT ANALYSIS
C
      DO 100 TIMSTP=1,SIMLEN
C
CC       CALL SIGNAL GENERATING SUBROUTINE
C
         CALL SIGNAL
C
CC       CALL ANALYSIS ROUTINE
C
         CALL TRN(INTEGR,VOLTAG)
C
CC       WRITE OUTPUT NODE VALUES TO ARRAYS TRNFP AND TRNIP
CC       FINITE PRECISION AND INFINITE PRECISION OUTPUT NODE VALUES
C
         TRNFP(TIMSTP) = FLOAT(NODES(NODOUT,T1))*RESN
         TRNIP(TIMSTP) = SNGL(ANODES(NODOUT,T1))
C
CC       SWITCH T0 AND T1
C
         TEMP=T1
         T1=T0
         T0=TEMP
100   CONTINUE
C
      TIME(1)=0.
      DO 110 LP3=2,1024
         TIME(LP3)=FLOAT(LP3)
 110  CONTINUE
C
C
      RETURN
C
      END
```

```
C
C*****************************************************************************
C
C     SUBROUTINE FOR CHANGING COEFFICIENT VALUES.
C
C-----------------------------------------------------------------------------
C
      SUBROUTINE COFCHG
C
      COMMON NETWOR(100,4),COEFF(50,2),COFRND(50),NUMULT,NUMNWK
C
      PRINT*,'MULTIPLIER COEFFICIENT VALUES'
      IPTR=1
      ITEMP1=NUMULT
C
 10   IF (ITEMP1.LT.3) GO TO 20
      WRITE (6,111) IFIX(COEFF(IPTR,2)),COEFF(IPTR,1),
     &              IFIX(COEFF(IPTR+1,2)),COEFF(IPTR+1,1),
     &              IFIX(COEFF(IPTR+2,2)),COEFF(IPTR+2,1)
      ITEMP1=ITEMP1-3
      IPTR=IPTR+3
      GO TO 10
C
 20   IF (ITEMP1.LT.2) GO TO 30
      WRITE (6,121) IFIX(COEFF(IPTR,2)),COEFF(IPTR,1),
     &              IFIX(COEFF(IPTR+1,2)),COEFF(IPTR+1,1)
      GO TO 40
C
 30   WRITE (6,131) IFIX(COEFF(IPTR,2)),COEFF(IPTR,1)
C
 40   PRINT*,'ENTER NODE NUMBER FOLLOWED BY NEW COEFF VALUE'
      PRINT*,'ENTER 0 0 TO END'
 45   READ*,MULNOD,COFNEW
      IF (MULNOD.EQ.0) GO TO 60
      DO 50 LP1=1,NUMULT
         IF (IFIX(COEFF(LP1,2)).NE.MULNOD) GO TO 50
         COEFF(LP1,1) = COFNEW
 50   CONTINUE
      GO TO 45
C
 60   PRINT*,'1: WRITE COEFFS TO TRAN-NET     2: CONTINUE'
      READ*,MENU1
      IF (MENU1.NE.1) RETURN
      DO 70 LP2=1,NUMULT
         WRITE(8,*) IFIX(COEFF(LP2,2)),COEFF(LP2,1)
 70   CONTINUE
C
      RETURN
C
 111  FORMAT (3(2X,I3,2H: ,F10.7,5X))
 121  FORMAT (2(2X,I3,2H: ,F10.7,5X))
 131  FORMAT (2X,I3,2H: ,F10.7)
C
      END
C
C*****************************************************************************
```

```
C
C     SUBROUTINE FOR ASSIGNING DEFAULT VALUES TO PARAMETERS
C
C-----------------------------------------------------------------------
C
      SUBROUTINE DEFLT
C
      INTEGER SIMLEN,SIGPCN,ATDTYP,ADDTYP,COFSEL,COFPCN,SIGTYP
C
      COMMON NETWOR(100,4),COEFF(50,2),COFRND(50),NUMULT,NUMNWK
      COMMON/ONE/NODES(100,3),T0,T1,MAXNUM
      COMMON/TWO/TIME(1024),TRNIP(1024),TRNFP(1024),TRNOIS(1024),SIMLEN
      COMMON/THREE/IPIP(1024),FPIP(1024),SIGPCN,ATDTYP
      COMMON/FOUR/RAND(50),RANPTR,OVFLOW
      COMMON/FIVE/MTYP,ADDTYP,MULTOF,COFSEL,COFPCN,TERMTP,ANTYP,NODOUT
      COMMON/SIX/INTEGR,VOLTAG,SIGTYP,SIGAMP,TIMSTP,RESN,RMPRTE,SIGFR
      COMMON/TEN/NETDPN
C
      NETDPN=2
      SIMLEN=1024
      SIGPCN=24
      ATDTYP=3
      MTYP=3
      MULTOF=2
      ADDTYP=3
      COFSEL=2
      COFPCN=24
      SIGTYP=1
      SIGAMP=1.0
      RMPRTE=0.001
      SIGFR=1.0
C
      RETURN
      END
C
C***********************************************************************
C
C     SUBROUTINE TO EVALUATE FREQUENCY RESPONSE.
C     A NAG SUBROUTINE IS USED TO PERFORM AN FFT ON THE IMPULSE
C     RESPONSE.
C
C-----------------------------------------------------------------------
C
      SUBROUTINE FORIER(RLOSS,NSPT)
C
      REAL RLOSS(1024),NAGWOR(1024)
      DOUBLE PRECISION X(1024),WORK(1024)
      INTEGER IFAIL
      COMPLEX CRLOSS(1024)
C
      COMMON/FRQAN/FREQ(1024),GAIN(1024)
C
      DO 1 LX=1,1024
         RLOSS(LX) = (SQRT(FLOAT(NSPT)))*RLOSS(LX)
         X(LX)=DBLE(RLOSS(LX))
 1    CONTINUE
C     LIMIT SMALLEST VALUE TO PREVENT UNDERFLOW
C
      DO 5 LP1=1,1024
         IF (ABS(RLOSS(LP1)).LT.(0.5E-8)) RLOSS(LP1)=0.0
 5    CONTINUE
C
      TEMP=ALOG(FLOAT(NSPT))/ALOG(2.0)
      M=INT(TEMP+0.5)
      PRINT*,M
C
      FREQ(1)=0.
      DFREQ=1.0/NSPT
      DO 10 I=1,NSPT
         FREQ(I+1)=FREQ(I)+DFREQ
 10   CONTINUE
      IFAIL=0
      CALL C06FAF(X,NSPT,WORK,IFAIL)
      DO 15 LQ=1,1024
         RLOSS(LQ)=SNGL(X(LQ))
 15   CONTINUE
      DO 20 LP2=1,NSPT/2
         CRLOSS(LP2)=CMPLX(RLOSS(LP2),0.0)
         CRLOSS(NSPT+1-LP2)=CMPLX(RLOSS(LP2),0.0)
 20   CONTINUE
C
      CRLOSS(NSPT)=CMPLX(REAL(CRLOSS(NSPT)),0.0)
      DO 25 LP3=NSPT/2+2,NSPT
         CRLOSS(LP3-1)=CMPLX(REAL(CRLOSS(LP3-1)),RLOSS(LP3))
 25   CONTINUE
C
      DO 27 LP4=1,NSPT/2
         CRLOSS(LP4)=CMPLX(REAL(CRLOSS(LP4)),-AIMAG(CRLOSS(NSPT+1-LP4)))
 27   CONTINUE
      DO 30 I=1,NSPT
      TEMP1=(CRLOSS(I)*CONJG(CRLOSS(I)))
      IF (TEMP1.EQ.0.0) TEMP1=100.0
      GAIN(I)=10*ALOG10(TEMP1)
 30   CONTINUE
      RETURN
      END
C
C***********************************************************************
C
C     CALLING ROUTINE FOR AUTOMATIC FREQUENCY RESPONSE ANALYSIS
C
C-----------------------------------------------------------------------
C
      SUBROUTINE FRAUTO
C
      INTEGER COFSEL,COFPCN,COFRNU,COFRNL,COFINC,SIGTYP
C
      COMMON/TWO/TIME(1024),TRNIP(1024),TRNFP(1024),TRNOIS(1024),SIMLEN
      COMMON/FIVE/MTYP,ADDTYP,MULTOF,COFSEL,COFPCN,TERMTP,ANTYP,NODOUT
      COMMON/FRQAN/FREQ(1024),GAIN(1024)
      COMMON/SIX/INTEGR,VOLTAG,SIGTYP,SIGAMP,TIMSTP,RESN,RMPRTE,SIGFR
      COMMON/SEVEN/PBEJFR,SBEJFR,PBGAIN,SBGAIN
```

```
C
 10   PRINT*,'AUTO FREQUENCY ANALYSIS'
      WRITE (6,111) SIGAMP
      PRINT*,'2: CHANGE COEFFICIENT VALUES'
      WRITE (6,121) NUMFFT
      PRINT*,'4: RUN ANALYSIS'
      PRINT*,'5: QUIT'
      READ*,MENU1
C
      GO TO (20,30,40,50,70), MENU1
C
 20   PRINT*,'INPUT IMPULSE AMPLITUDE : 1.0 VOLTS MAX'
      READ*,SIGAMP
      GO TO 10
C
 30   CALL COFCHG
      GO TO 10
C
 40   PRINT*,'INPUT NUMBER OF FFT POINTS'
      READ*,NUMFFT
      GO TO 10
C
 50   PRINT*,'OUTPUT NODE NUMBER?'
      READ*,NODOUT
      SIGTYP=1
      PRINT*,'ENTER COEFF RANGE(LOWER,UPPER) AND COEFF INCREMENT'
      READ*,COFRNL,COFRNU,COFINC
C
      PRINT*,'INPUT SPECIFIED TOLERANCE SCHEME'
      PRINT*,'PBGAIN(DB),SBGAIN(DB),PBEJFR,SBEJFR(NORMALISED)'
      READ*,PBGAIN,SBGAIN,PBEJFR,SBEJFR
C
      WRITE (12,131)
      WRITE (12,141) PBGAIN,SBGAIN,PBEJFR,SBEJFR
      WRITE (12,161)
      WRITE (12,151)
C
C     INFINITE PRECISION ANALYSIS
C
      COFSEL=2
      CALL ANMAN
      CALL FORIER(TRNIP,NUMFFT)
      CALL FRSPAN(NUMFFT)
C
C     FINITE PRECISION ANALYSIS
C
      COFSEL=1
      COFPCN=COFRNU
 60   CALL ANMAN
      CALL FORIER(TRNIP,NUMFFT)
      CALL FRSPAN(NUMFFT)
      COFPCN=COFPCN-COFINC
      IF (COFPCN.LT.COFRNL) GO TO 10
      GO TO 60
C
 70   RETURN
C
 111  FORMAT (('1: CHANGE IMPULSE AMPLITUDE'),5X,('('),F3.1,(')'))
 121  FORMAT (('3: CHANGE NUMBER OF FFT POINTS'),5X,('('),I4,(')'))
 131  FORMAT (('PBGAIN(DB)  SBGAIN(DB)  PBEJFR(NORM)  SBEJFR(NORM)'))
 141  FORMAT (4(6X,F6.2))
 151  FORMAT (('COFPN    GMAX    PGMIN    PGSFR    SGSFR    PFRMG    SF
     &RSG  MKLOW  MKHGH  SGMAX'))
 161  FORMAT (70('-'))
      END
C
C*********************************************************************
C
C     OUTPUT SUBROUTINE FOR FREQUENCY RESPONSE ANALYSIS
C
C---------------------------------------------------------------------
C
      SUBROUTINE FRNOUT
C
      INTEGER COFPCN,COFSEL,SIGTYP
      CHARACTER*2 DOLLAR
C
      COMMON/TWO/TIME(1024),TRNIP(1024),TRNFP(1024),TRNOIS(1024),SIMLEN
      COMMON/FIVE/MTYP,ADDTYP,MULTOF,COFSEL,COFPCN,TERMTP,ANTYP,NODOUT
      COMMON/FRQAN/FREQ(1024),GAIN(1024)
      COMMON/SIX/INTEGR,VOLTAG,SIGTYP,SIGAMP,TIMSTP,RESN,RMPRTE,SIGFR
C
C
      DOLLAR='$$'
 10   PRINT*,' '
      PRINT*,'    FREQUENCY ANALYSIS'
      PRINT*,'    ------------------'
      WRITE (6,111) SIGAMP
      IF (COFSEL.EQ.2) PRINT*,' 2: CHANGE COEEFICIENT PRECISION  ($$)'
      IF (COFSEL.EQ.1) WRITE (6,121) COFPCN
      PRINT*,' 3: CHANGE COEFFICIENT VALUES'
      WRITE (6,131) NUMFFT
      PRINT*,' 5: RUN FREQUENCY ANALYSIS'
      PRINT*,' 6: QUIT'
      READ*,MENU1
C
      GO TO (20,30,40,50,60,70), MENU1
C
 20   PRINT*,'INPUT IMPULSE AMPLITUDE  : 1.0 VOLTS MAX'
      READ*,SIGAMP
      GO TO 10
C
 30   PRINT*,'COEFFS..... 1: FINITE PRECN    2:' INFINITE PRECN'
      READ*,COFSEL
      IF (COFSEL.EQ.2) GO TO 10
      PRINT*,'INPUT COEFFICIENT PRECISION IN NUMBER OF BITS'
      READ*,COFPCN
      GO TO 10
C
 40   CALL COFCHG
      GO TO 10
```

```
C
 50   PRINT*,'INPUT NUMBER OF FFT POINTS : 2,4,8,....,1024'
      READ*,NUMFFT
      GO TO 10
C
 60   PRINT*,'OUTPUT NODE NUMBER?'
      READ*,NODOUT
      SIGTYP=1
      CALL ANMAN
      CALL FORIER(TRNIP,NUMFFT)
      CALL PLOT(FREQ,GAIN,NUMFFT)
      GO TO 10
C
 70   RETURN
C
 111  FORMAT (('  1: CHANGE IMPULSE AMPLITUDE'),5X,('('),F3.1,(')'))
 121  FORMAT ((' 2: CHANGE COEFFICIENT PRECISION'),5X,('('),I2,(')'))
 131  FORMAT (('  4: CHANGE NUMBER OF FFT POINTS'),5X,('('),I4,(')'))
C
      END
C
C*********************************************************************
C
C     FREQUENCY RESPONSE ANALYSIS SUBROUTINE.
C     DETERMINES THE PASSBAND EDGE, PASSBAND RIPPLE, STOPBAND EDGE
C     AND THE STOPBAND GAIN. (THE TRANSITION WIDTH IS USER SPECIFIED).
C
C---------------------------------------------------------------------
C
      SUBROUTINE FRSPAN(NUMPTS)
C
      INTEGER PTR,COFSEL,COFPCN
      character*2 dollar
C
      COMMON/FRQAN/FREQ(1024),GAIN(1024)
      COMMON/FIVE/MTYP,ADDTYP,MULTOF,COFSEL,COFPCN,TERMTP,ANTYP,NODOUT
      COMMON/SEVEN/PBEJFR,SBEJFR,PBGAIN,SBGAIN
C
C
      dollar='$$'
      NFRPTS=NUMPTS/2
      GMAX=GAIN(1)
C
C     INITIALISE POINTERS
C
      PTR=1
      MRK3DB=1
      MRKTRN=1
      MRK01=1
      MRK05=1
      MRK1=1
      MRKPB=1
      MRK20=1
      MRK40=1
      MRK60=1
      MRK80=1
      MRK100=1
      MRKSMX=1
C
C
C     DETERMINE MAXIMUM GAIN
C
      DO 10 PTR=1,NFRPTS-1
 10      GMAX=AMAX1(GMAX,GAIN(PTR+1))
C
C     DETERMINE FINAL -3DB CROSSOVER
C
      GN3DB=GMAX-3.0
      DO 20 PTR=2,NFRPTS-1
 20      IF ((GAIN(PTR-1).GE.GN3DB).AND.(GAIN(PTR+1).LE.GN3DB)) MRK3DB=
     & ptr
C
C     PASSBAND ANALYSIS
C
      GN01=GMAX-0.1
      GN05=GMAX-0.5
      GN1=GMAX-1.0
C
      DO 50 PTR=2,MRK3DB
         IF ((GAIN(PTR-1).GE.GAIN(PTR)).AND.(GAIN(PTR+1).GE.GAIN(PTR)))
     &   GO TO 30
         GO TO 40
 30      IF (GAIN(PTR).LT.GAIN(MRKTRN)) MRKTRN=PTR
 40      IF ((GAIN(PTR-1).GE.GN01).AND.(GAIN(PTR+1).LE.GN01)) MRK01=PTR
         IF ((GAIN(PTR-1).GE.GN05).AND.(GAIN(PTR+1).LE.GN05)) MRK05=PTR
         IF ((GAIN(PTR-1).GE.GN1).AND.(GAIN(PTR+1).LE.GN1)) MRK1=PTR
         IF ((GAIN(PTR-1).GE.GAIN(MRKTRN)).AND.(GAIN(PTR+1).LE.GAIN
     &   (MRKTRN))) MRKPB=PTR
 50   CONTINUE
      PBRIP=GMAX-GAIN(MRKTRN)
C
C     TRANSITION BAND AND STOPBAND ANALYSIS
C
      GN20=GMAX-20.0
      GN40=GMAX-40.0
      GN60=GMAX-60.0
      GN80=GMAX-80.0
      GN100=GMAX-100.0
C
      MRKTRN=1
      DO 60 PTR=MRKPB,NFRPTS-1
         IF (MRKTRN.GT.1) GO TO 60
         IF (GAIN(PTR-1).LT.GAIN(PTR+1)) MRKTRN=PTR
         IF ((GAIN(PTR-1).GE.GN20).AND.(GAIN(PTR+1).LE.GN20)) MRK20=PTR
         IF ((GAIN(PTR-1).GE.GN40).AND.(GAIN(PTR+1).LE.GN40)) MRK40=PTR
         IF ((GAIN(PTR-1).GE.GN60).AND.(GAIN(PTR+1).LE.GN60)) MRK60=PTR
         IF ((GAIN(PTR-1).GE.GN80).AND.(GAIN(PTR+1).LE.GN80)) MRK80=PTR
         IF ((GAIN(PTR-1).GE.GN100).AND.(GAIN(PTR+1).LE.GN100))
     &  MRK100=PTR
 60   CONTINUE
C
      IF (MRKTRN.EQ.1) GO TO 80
```

```
      SGMAX=GAIN(MRKTRN)
      DO 70 PTR=MRKTRN,NFRPTS-1
         IF (GAIN(PTR).LE.SGMAX) GO TO 70
         SGMAX=GAIN(PTR)
         MRKSMX=PTR
   70 CONTINUE
C
C     WRITE RESULTS TO FILE FRN-OUT
C
      GO TO 90
   80 MRKSMX=NFRPTS
   90 continue
      IF (COFSEL.EQ.2) WRITE (12,111) dollar,GMAX,FREQ(MRKPB),
     &PBRIP
      IF (COFSEL.EQ.1) WRITE (12,121) COFPCN,GMAX,FREQ(MRKPB),
     &PBRIP
      WRITE (12,131) FREQ(MRK01),FREQ(MRK05),FREQ(MRK1)
      WRITE (12,141) FREQ(MRK20),FREQ(MRK40),FREQ(MRK60),FREQ(MRK80),
     &              FREQ(MRK100)
      TEMP1=GAIN(MRKSMX)-GMAX
      WRITE (12,151) FREQ(MRKSMX),TEMP1
C
      RETURN
C
  111 FORMAT (A2,3(2X,F9.4))
  121 FORMAT (I2,3(2X,F9.4))
  131 FORMAT (3(2X,F9.4))
  141 FORMAT (5(2X,F9.4))
  151 FORMAT (2(2X,F9.4))
C
      END
C
C*********************************************************************
C
C     FUNCTION FOR FINITE PRECISION ADDITION.
C     PERFORMS FINITE PRECISION ADDITION, OVERFLOW MODELLING AND
C     OVERFLOW DETECTION
C
C---------------------------------------------------------------------
C
      FUNCTION IADD(NETPTR)
C
      INTEGER ADDTYP,T0,T1,OVFLOW
C
      COMMON NETWOR(100,4),COEFF(50,2),COFRND(50),NUMULT,NUMNWK
      COMMON/ONE/NODES(100,3),T0,T1,MAXNUM
      COMMON/FIVE/MTYP,ADDTYP,MULTOF,COFSEL,COFPCN,TERMTP,ANTYP,NODOUT
      COMMON/FOUR/RAND(50),RANPTR,OVFLOW
C
      IADD = NODES(NETWOR(NETPTR,3),T1) + NODES(NETWOR(NETPTR,4),T1)
C
      IF ((IADD.LT.MAXNUM).AND.(IADD.GE.-MAXNUM)) RETURN
      GO TO 10
C
C     BRANCH ON ADDER TYPE.
C
   10 NODES(NETWOR(NETPTR,2),3) = NETWOR(NETPTR,2)
      OVFLOW=1
      GO TO (20,30,40),ADDTYP
C
C     SATURATING ADDER
C
   20 IF (IADD.GT.0) ITEMP1=MAXNUM-1
      IF (IADD.LT.0) ITEMP1=-MAXNUM
      IADD=ITEMP1
      RETURN
C
C     ZEROING ADDER
C
   30 IADD=0
      RETURN
C
C     2s COMPLEMENTING ADDER
   40 IF (IABS(IADD).GE.(2*MAXNUM)) GO TO 50
      GO TO 60
   50 IF (IADD.LT.0) ITEMP2=IADD+(2*MAXNUM)
      IF (IADD.GT.0) ITEMP2=IADD-(2*MAXNUM)
      IADD=ITEMP2
      GO TO 40
C
   60 IF ((IADD.LT.MAXNUM).AND.(IADD.GE.-MAXNUM)) RETURN
      ITEMP3=IABS(IADD)-MAXNUM
      IF (IADD.LT.0) ITEMP4=MAXNUM-ITEMP3
      IF (IADD.GT.0) ITEMP4=-(MAXNUM-ITEMP3)
      IF (ITEMP3.EQ.0) ITEMP4=-MAXNUM
      IADD=ITEMP4
      RETURN
C
C
      END
C
C*********************************************************************
C
C     SUBROUTINE FOR FINITE PRECISION MULTIPLICATION.
C     PERFORMS FINITE PRECISION MULTIPLICATION WITH APPROPRIATE
C     ROUNDING.
C
C---------------------------------------------------------------------
C
      FUNCTION MULT(NETPTR)
C
      INTEGER T0,T1,OVFLOW
      DOUBLE PRECISION DNODE,DCOEFF,DOUT,DTEMP1
C
      COMMON NETWOR(100,4),COEFF(50,2),COFRND(50),NUMULT,NUMNWK
      COMMON/ONE/NODES(100,3),T0,T1,MAXNUM
      COMMON/FIVE/MTYP,ADDTYP,MULTOF,COFSEL,COFPCN,TERMTP,ANTYP,NODOUT
      COMMON/FOUR/RAND(50),RANPTR,OVFLOW
C
      DNODE=DBLE(FLOAT(NODES(NETWOR(NETPTR,3),T1)))
      DCOEFF=DBLE(COFRND(NETWOR(NETPTR,4)))
      DOUT=DNODE*DCOEFF
```

```
C
C     BRANCH ON MULTIPLIER TYPE
C
      GO TO (10,20,30), MTYP
C
C     2s COMPLEMENT TRUNCATION
C
 10   MULT=IDINT(DOUT)
      DTEMP1=DOUT-DBLE(FLOAT(MULT))
      IF (SNGL(DTEMP1).LT.0.) MULT=MULT-1
      IF ((MULT.LT.MAXNUM).AND.(MULT.GE.-MAXNUM)) RETURN
      GO TO 40
C
C     MAGNITUDE TRUNCATION
C
 20   MULT=IDINT(DOUT)
      IF ((MULT.LT.MAXNUM).AND.(MULT.GE.-MAXNUM)) RETURN
      GO TO 40
C
C     ROUNDING
C
 30   IF (FLOAT(DOUT).GT.0) DOUT=DOUT+0.5D0
      IF (FLOAT(DOUT).LT.0) DOUT=DOUT-0.5D0
      MULT=IDINT(DOUT)
      IF ((MULT.LT.MAXNUM).AND.(MULT.GE.-MAXNUM)) RETURN
C
C     MULTIPLIER OVERFLOW
C
 40   NODES(NETWOR(NETPTR,2),3) = NETWOR(NETPTR,2)
      OVFLOW=1
      GO TO (50,80,90), MULTOF
C
 50   IF (IABS(MULT).GE.(2*MAXNUM)) GO TO 60
      GO TO 70
 60   IF (MULT.LT.0) ITEMP1=MULT+(2*MAXNUM)
      IF (MULT.GT.0) ITEMP1=MULT-(2*MAXNUM)
      MULT=ITEMP1
      GO TO 50
C
 70   IF ((MULT.LT.MAXNUM).AND.(MULT.GE.-MAXNUM)) RETURN
      ITEMP2=IABS(MULT)-MAXNUM
      IF (MULT.LT.0) ITEMP3=MAXNUM-ITEMP2
      IF (MULT.GT.0) ITEMP3=-(MAXNUM-ITEMP2)
      IF (ITEMP2.EQ.0) ITEMP3=-MAXNUM
      MULT=ITEMP3
      RETURN
C
C     SATURATING
C
 80   IF (MULT.GT.0) ITEMP4=MAXNUM-1
      IF (MULT.LT.0) ITEMP4=-MAXNUM
      MULT=ITEMP4
      RETURN
C
C     ZEROING
C
 90   MULT=0
      RETURN
C
      END
C
C*********************************************************************
C
C     SUBROUTINE FOR READING AND ORDERING ELEMENT LIST.
C     READS IN ALL ELEMENTS AND SORTS THEM INTO THE APPROPRIATE
C     ORDER FOR SIMULATION
C
C---------------------------------------------------------------------
C
      SUBROUTINE NETINP
C
      INTEGER NETWOR,NETSCH(100,4),COFPTR,PTRSCH,NWKPTR
      INTEGER FOUND1,FOUND2
      REAL COEFF
C
      COMMON NETWOR(100,4),COEFF(50,2),COFRND(50),NUMULT,NUMNWK
      COMMON/TEN/NETDPN
C
C     ASSIGN READ AND WRITE CHANNELS
C
      IF (NETDPN.EQ.1) NRD=5
      IF (NETDPN.EQ.2) NRD=7
C
C     INITIALISE POINTERS AND LOGICAL VALUES
C
      PTRSCH=1
      NWKPTR=1
      COFPTR=1
      FOUND1=0
      FOUND2=0
C
C     READ ALL DELAYS AND STORE IN ARRAY NETWOR
C
      IF (NETDPN.EQ.2) GO TO 10
      PRINT*,'INPUT DELAYS : OUTPUT NODE  INPUT NODE'
      PRINT*,'TYPE 0 0 TO END DELAY INPUT'
C
 10   READ(NRD,*)NOUT,NINP
      IF (NETDPN.EQ.1) WRITE(7,*) NOUT,NINP
      IF (NOUT.EQ.0) GO TO 20
      NETWOR(NWKPTR,1)=3
      NETWOR(NWKPTR,2)=NOUT
      NETWOR(NWKPTR,3)=NINP
      NWKPTR=NWKPTR+1
      GO TO 10
C
 20   NUMNWK=NWKPTR-1
C
C     READ ALL ADDERS
C
      IF (NETDPN.EQ.2) GO TO 30
      PRINT*,'INPUT ADDERS : OUTPUT NODE  INPUT NODE1  INPUT NODE2'
```

```
      PRINT*,'TYPE 0 0 0 TO END ADDER INPUT'
 30   READ(NRD,*)NOUT,NINP1,NINP2
      IF (NETDPN.EQ.1) WRITE(7,*) NOUT,NINP1,NINP2
      IF (NOUT.EQ.0) GO TO 60
C
C     IF ADDER INPUTS PRESENT IN ARRAY NETWOR , STORE ADDER IN NETWOR
C     ELSE STORE IN NETSCH
C
      DO 40 LPCNT2=1,NUMNWK
         IF (NINP1.EQ.(NETWOR(LPCNT2,2))) FOUND1=1
         IF (NINP1.EQ.1) FOUND1=1
         IF (NINP2.EQ.(NETWOR(LPCNT2,2))) FOUND2=1
         IF (NINP2.EQ.1) FOUND2=1
 40   CONTINUE
      IF ((FOUND1.EQ.1).AND.(FOUND2.EQ.1)) GO TO 50
      NETSCH(PTRSCH,1) = 4
      NETSCH(PTRSCH,2)=NOUT
      NETSCH(PTRSCH,3)=NINP1
      NETSCH(PTRSCH,4)=NINP2
      PTRSCH=PTRSCH+1
      FOUND1=0
      FOUND2=0
      GO TO 30
C
 50   NETWOR(NWKPTR,1)=4
      NETWOR(NWKPTR,2)=NOUT
      NETWOR(NWKPTR,3)=NINP1
      NETWOR(NWKPTR,4)=NINP2
      NWKPTR=NWKPTR+1
      FOUND1=0
      FOUND2=0
      GO TO 30
C
C     READ ALL MULTIPLIERS
C
 60   NUMULT=0
      IF (NETDPN.EQ.2) GO TO 70
      PRINT*,'INPUT MUTIPLIERS : OUTPUT NODE  INPUT NODE'
      PRINT*,'TYPE 0 0 TO END MULTIPLIER INPUT'
 70   READ(NRD,*)NOUT,NINP
      IF (NETDPN.EQ.1) WRITE (7,*) NOUT,NINP
      IF (NOUT.EQ.0) GO TO 100
      NUMULT=NUMULT+1
C
C     IF MULTIPLIER INPUT PRESENT IN NETWOR STORE MULT. IN NETWOR
C     ELSE STORE IN NETSCH
C
      DO 80 LPCNT1=1,NUMNWK
         IF (NINP.EQ.NETWOR(LPCNT1,2)) FOUND1=1
         IF (NINP.EQ.1) FOUND1=1
 80   CONTINUE
      IF (FOUND1.EQ.1) GO TO 90
      NETSCH(PTRSCH,1)=-7
      NETSCH(PTRSCH,2)=NOUT
      NETSCH(PTRSCH,3) = NINP
      NETSCH(PTRSCH,4) = COFPTR
      COFPTR = COFPTR+1
      PTRSCH = PTRSCH+1
      GO TO 70
C
 90   NETWOR(NWKPTR,1) = -7
      NETWOR(NWKPTR,2) = NOUT
      NETWOR(NWKPTR,3) = NINP
      NETWOR(NWKPTR,4) = COFPTR
      COFPTR = COFPTR+1
      NUMNWK=NWKPTR
      NWKPTR = NWKPTR+1
      FOUND1 =0
      GO TO 70
C
 100  NUMNWK=NWKPTR-1
      NUMSCH=PTRSCH-1
C
C     SORT OUT ELEMENTS IN NETSCH AND STORE IN CORRECT PLACE IN NETWOR
C
 105  DO 130 PTRSCH=1,NUMSCH
         IF (NETSCH(PTRSCH,1).EQ.0) GO TO 130
         DO 110 LPCNT3=1,NUMNWK
            IF (NETWOR(LPCNT3,2).EQ.NETSCH(PTRSCH,3)) FOUND1=1
            IF (NETSCH(PTRSCH,3).EQ.1) FOUND1=1
            IF (NETWOR(LPCNT3,2).EQ.NETSCH(PTRSCH,4)) FOUND2=1
            IF (NETSCH(PTRSCH,4).EQ.1) FOUND2=1
 110     CONTINUE
         IF ((NETSCH(PTRSCH,1).EQ.4).AND.(FOUND2.EQ.0)) GO TO 120
         IF ((NETSCH(PTRSCH,1).EQ.-7).AND.(FOUND1.EQ.0)) GO TO 120
         IF ((NETSCH(PTRSCH,1).EQ.4).AND.(FOUND1.EQ.0)) GO TO 120
C
C     TRANSFER ELEMENT FROM NETSCH TO NETWOR
C
         NWKPTR=NUMNWK+1
         NETWOR(NWKPTR,1) = NETSCH(PTRSCH,1)
         NETWOR(NWKPTR,2) = NETSCH(PTRSCH,2)
         NETWOR(NWKPTR,3) = NETSCH(PTRSCH,3)
         NETWOR(NWKPTR,4) = NETSCH(PTRSCH,4)
         NETSCH(PTRSCH,1)=0
 120     FOUND1=0
         FOUND2=0
         NUMNWK=NWKPTR
 130  CONTINUE
C
C     CHECK IF NETSCH EMPTY. IF NOT REPEAT SORT PROCESS
C
      DO 140 LPCNT4=1,NUMSCH
         IF(NETSCH(LPCNT4,1).NE.0) GO TO 105
 140  CONTINUE
C
C     WRITE NETLIST TO FILE ON CHANNEL 8
C
      DO 150 LPCNT5=1,NUMNWK
         WRITE (8,*) NETWOR(LPCNT5,1),NETWOR(LPCNT5,2),NETWOR(LPCNT5,
     &   3),NETWOR(LPCNT5,4)
 150  CONTINUE
```

```
C
C     READ MULTIPLIER COEFFICIENT VALUES
C
      IF (NETDPN.EQ.2) GO TO 155
      PRINT*,'INPUT INFINITE PRECISION COEFFICIENT VALUES'
 155  DO 160 LP6=1,NUMNWK
         IF (NETWOR(LP6,1).NE.-7) GO TO 160
         IF (NETDPN.EQ.2) GO TO 157
         PRINT*,NETWOR(LP6,2),NETWOR(LP6,3)
  157    READ(NRD,*) COEFF(NETWOR(LP6,4),1)
         COEFF(NETWOR(LP6,4),2) = FLOAT(NETWOR(LP6,2))
         IF (NETDPN.EQ.1) WRITE(7,*) COEFF(NETWOR(LP6,4),1)
 160  CONTINUE
C
C     WRITE COEFFICIENT VALUES TO FILE ON CHANNEL 8
C
      DO 170 LP7=1,NUMULT
         WRITE (8,*) IFIX(COEFF(LP7,2)),COEFF(LP7,1)
 170  CONTINUE
C
C
      NETDPN=2
C
      RETURN
      END
C
C*********************************************************************
C
C     SUBROUTINE FOR GRAPHIC DISPLAY OF FREQUENCY OR TRANSIENT
C     RESPONSE AT ANY NODE.
C     THIS SUBROUTINE USES GINO ROUTINES.
C
C---------------------------------------------------------------------
C
      SUBROUTINE PLOT(X,Y,NUMPTS)
C
      REAL X(1024),Y(1024),XPLOT(1024),YPLOT(1024)
      INTEGER PTR,TERMTP
C
      COMMON/FIVE/MTYP,ADDTYP,MULTOF,COFSEL,COFPCN,TERMTP,ANTYP,NODOUT
C
      print*,'input any number to continue'
      read*,xxxx
      CALL T4107
      CALL PICCLE
      CALL GRAF(X,Y,NUMPTS)
      CALL CHAMOD
C
 10   READ*,N
      CALL PICCLE
      PRINT*,'1: ZOOM IN     2:QUIT'
      READ*,MENU
      IF (MENU.NE.2) GO TO 15
      CALL PICCLE
      CALL DEVEND
      RETURN
C
 15   IF (MENU.NE.1) GO TO 10
      PRINT*,'INPUT X-AXIS LOWER AND UPPER VALUES'
      READ*,XLOWER,XUPPER
      PTR=1
      DO 30 LP1=1,NUMPTS
         IF ((X(LP1).LE.XUPPER).AND.(X(LP1).GE.XLOWER)) GO TO 20
         GO TO 30
 20      XPLOT(PTR)=X(LP1)
         YPLOT(PTR)=Y(LP1)
         PTR=PTR+1
 30   CONTINUE
C
      NUM=PTR-1
      CALL PICCLE
      CALL GRAF(XPLOT,YPLOT,NUM)
      CALL CHAMOD
      GO TO 10
      END
C
C*********************************************************************
C
C     COEFFICIENT QUANTISATION SUBROUTINE.
C     ROUNDS COEFFICENT VALUES TO REQUIRED NUMBER OF BITS.
C
C---------------------------------------------------------------------
C
      SUBROUTINE ROUND
C
      INTEGER COFSGN(50),COFPTR,COFPCN
      REAL COFVAL(50)
C
      COMMON NETWOR(100,4),COEFF(50,2),COFRND(50),NUMULT,NUMNWK
      COMMON/FIVE/MTYP,ADDTYP,MULTOF,COFSEL,COFPCN,TERMTP,ANTYP,NODOUT
C
C     SPLIT ARRAY COEFF INTO ARRAYS COFVAL AND COFSGN : 0 FOR +VE
C     AND 1 FOR -VE
C
      DO 10 COFPTR=1,NUMULT
         IF (COEFF(COFPTR,1).LT.0) COFSGN(COFPTR)=1
         COFVAL(COFPTR) = ABS(COEFF(COFPTR,1))
 10   CONTINUE
C
C     FIND LARGEST COEFF VALUE ; EXCLUDING MULTIPLE OF 2.0
C
      DO 20 COFPTR=1,NUMULT-1
         TEMP1=COFVAL(COFPTR+1)
         TEMP2=ALOG10(TEMP1)/ALOG10(2.0)
         IF (TEMP2.EQ.INT(TEMP2)) TEMP1=0.0
         TEMP3=COFVAL(COFPTR)
         TEMP4=ALOG10(TEMP3)/ALOG10(2.0)
         IF (TEMP4.EQ.INT(TEMP4)) TEMP3=0.0
         COFLGT = AMAX1(TEMP1,TEMP3)
 20   CONTINUE
      IF (COFLGT.EQ.0.0) COFLGT=1.0
C
```

```
C     FIND NUMBER OF BITS TO LEFT OF BINARY POINT - INCLUDING
C     SIGN BIT
C
      SBITS = ALOG(COFLGT)/ALOG(2.)
      NSBITS = INT(SBITS)
      LEFBIN = NSBITS+1
C
C     FIND SMALLEST NUMBER THAT CAN BE REPRESENTED WITH NUMBER
C     OF BITS
C
      RGTBIN = COFPCN-LEFBIN
      TEMP5 = 2**(RGTBIN)
      VALSMT = 1/TEMP5
C
C     FIND QUANTISED COEFF VALUES
C
      DO 30 COFPTR=1,NUMULT
         TEMP6 = 0.5*VALSMT
         TEMP7 = COFVAL(COFPTR)+TEMP6
         TEMP8 = TEMP7/VALSMT
         COFRND(COFPTR) = INT(TEMP8)*VALSMT
         IF (COFSGN(COFPTR).EQ.1) COFRND(COFPTR)=-COFRND(COFPTR)
C        PRINT*,COFRND(COFPTR)
C
C     REPORT ZERO COEFFS TO VDU
C
      IF (COFRND(COFPTR).NE.0) GO TO 30
      WRITE (6,111) COEFF(COFPTR,1),COFPCN
      PRINT*,'COEFF VALUE SET TO ZERO'
 30   CONTINUE
C
      RETURN
C
 111  FORMAT (11HCOEFFICIENT,2X,F9.7,2X,24HCANNOT BE REPRESENTED BY,2X
     &        I2,2X,4HBITS)
C
      END
C
C*********************************************************************
C
C     SIGNAL GENERATING SUBROUTINE.
C     DEPENDING UPON TIME INSTANT, SUPPLIES VALUE OF INPUT SIGNAL.
C     FOR RANDOM SIGNAL TYPES, NAG SUBROUTINES ARE EMPLOYED.
C     PERFORMS ROUNDING FOR FINITE WORDLENGTH SIGNALS IN ORDER TO MODE
C     REAL A/D CONVERTERS.
C
C---------------------------------------------------------------------
C
      SUBROUTINE SIGNAL
C
      INTEGER SIGTYP,TIMSTP,RANPTR,ATDTYP
      REAL IPIP
C
      COMMON/THREE/IPIP(1024),FPIP(1024),SIGPCN,ATDTYP
      COMMON/FOUR/RAND(50),RANPTR,OVFLOW
      COMMON/SIX/INTEGR,VOLTAG,SIGTYP,SIGAMP,TIMSTP,RESN,RMPRTE,SIGFR
C
      GO TO (10,20,30,40,50,60,70), SIGTYP
C
C
C     SIGNAL GENERATORS
C
C     IMPULSE
C
 10   VOLTAG=SIGAMP
      IF(TIMSTP.GT.1) VOLTAG=0.
      GO TO 80
C
C     STEP
C
 20   VOLTAG=SIGAMP
      GO TO 80
C
C     RAMP
C
 30   VOLTAG = RMPRTE*(TIMSTP-1)
      GO TO 80
C
C     COSINE
C
 40   VOLTAG=COS(SIGFR*(TIMSTP-1))
      VOLTAG=VOLTAG*SIGAMP
      GO TO 80
C
C     UNIFORMLY DISTRIBUTED RANDOM
C
 50   VOLTAG=SIGAMP*SNGL(G05DAF(1.0D0,-1.0D0))
      GO TO 80
C
C     NON-UNIFORMLY DISTRIBUTED RANDOM
C
 60   VOLTAG=SIGAMP*SNGL(G05DAF(0.0D0,1.0D0))
      GO TO 80
C
C     GAUSSIANLY DISTRIBUTED
C
 70   RAN = SNGL(G05DDF(0.0D0,0.3807D0))
      IF ((RAN.GT.1.0).OR.(RAN.LT.-1.0)) GO TO 70
      VOLTAG=SIGAMP*RAN
C
 80   GO TO (90,100,110), ATDTYP
C
C     2S COMPLEMENT TRUNCATION
C
 90   TEMP1=VOLTAG/RESN
      INTEGR=INT(TEMP1)
      TEMP2=TEMP1-INTEGR
      IF (TEMP2.LT.0.) INTEGR=INTEGR-1
      GO TO 120
C
C     MAGNITUDE TRUNCATION
C
```

```
 100  INTEGR=INT(VOLTAG/RESN)
      GO TO 120
C
C     ROUNDING
C
 110  TEMP1=VOLTAG/RESN
      IF (TEMP1.GT.0) TEMP1=TEMP1+0.5
      IF (TEMP1.LT.0) TEMP1=TEMP1-0.5
      INTEGR=INT(TEMP1)
C
C     CREATE INPUT SIGNAL ARRAYS IPIP AND FPIP
C
 120  IPIP(TIMSTP)=VOLTAG
      FPIP(TIMSTP)=FLOAT(INTEGR)*RESN
C
      RETURN
C
      END
C
C*************************************************************
C
C    SUBROUTINE FOR AUTOMATIC TRANSIENT ANALYSIS.
C
C-------------------------------------------------------------
C
      SUBROUTINE TRAUTO
C
      INTEGER SIGTYP,SIMLEN,SIGPCN,SIGRNL,SIGRNU,COFPCN,COFRNU,COFRN
      INTEGER ANTYP,COFSEL,OVFLOW
      INTEGER ADDTYP,ATDTYP,ITITLE(20)
      REAL TRNFP,TRNIP,IPIP,NOSRMS
      CHARACTER*1 ISTAR,IBLANK
      CHARACTER*2 DOLLAR
C
      COMMON/TWO/TIME(1024),TRNIP(1024),TRNFP(1024),TRNOIS(1024),SIM
      COMMON/THREE/IPIP(1024),FPIP(1024),SIGPCN,ATDTYP
      COMMON/FIVE/MTYP,ADDTYP,MULTOF,COFSEL,COFPCN,TERMTP,ANTYP,NODO
      COMMON/SIX/INTEGR,VOLTAG,SIGTYP,SIGAMP,TIMSTP,RESN,RMPRTE,SIGF
      COMMON/FOUR/RAND(50),RANPTR,OVFLOW
C
C
C     INPUT ANALYSIS PARAMETERS
C
      PRINT*,'AUTO TRANSIENT ANALYSIS'
      PRINT*,'ADDER TYPE'
      PRINT*,'1:SATURATING  2:ZEROING  3:2s COMPLEMENT'
      READ*,ADDTYP
      PRINT*,'MULTIPLIER TYPE'
      PRINT*,'1: 2s COMP TRUNC.  2: MAGNITUDE TRUNC. 3:ROUNDING'
      READ*,MTYP
      PRINT*,'MULTIPLIER OVERFLOW TYPE'
      PRINT*,'1:2s COMPLEMENT  2:SATURATING  3:ZEROING'
      READ*,MULTOF
      PRINT*,'1:IMPULSE,2:STEP,3:RAMP,4:COSINE,5:UDR,6:NUDR,7:GAUSS'
      READ*,SIGTYP
      IF (SIGTYP.NE.3) GO TO 10
      PRINT*,'RAMP RATE '
      READ*,RMPRTE
 10   IF (SIGTYP.NE.4) GO TO 20
      PRINT*,'SIGNAL FREQ IN RADS/SEC - SAMPLING FREQ = 2*PI RADS/SEC'
      READ*,SIGFR
 20   PRINT*,'1: FINITE PRECN COEFFS    2:INFINITE PRECN COEFFS'
      READ*,COFSEL
      PRINT*,'SIMULATION LENGTH :  MAX   1024'
      READ*,SIMLEN
      PRINT*,'OUTPUT NODE'
      READ*,NODOUT
      PRINT*,'A/D CONVERTER'
      PRINT*,'1: 2s COMP TRUNC.  2: MAGNITUDE TRUNC. 3:ROUNDING'
      READ*,ATDTYP
C
      PRINT*,'INPUT HEADER FOR PRINTED RESULTS'
      READ(5,311)(ITITLE(I),I=1,20)
      WRITE (10,311) (ITITLE(I),I=1,20)
C
      WRITE (10,111)
      WRITE (10,121)
      WRITE (10,141) ADDTYP
      WRITE (10,161)
      WRITE (10,181) MTYP
      WRITE (10,191)
      WRITE (10,211) MULTOF
      WRITE (10,241)
      WRITE (10,181) ATDTYP
      WRITE (10,111)
      WRITE (10,221)
      WRITE (10,231) SIGTYP
      IF (SIGTYP.EQ.3) WRITE(10,271) RMPRTE
      IF (SIGTYP.EQ.4) WRITE(10,281) SIGFR
      WRITE (10,111)
      WRITE (10,401) SIMLEN
      WRITE (10,411) NODOUT
C
      IF (COFSEL.EQ.2) GO TO 30
      PRINT*,'INPUT COEFF RANGE(LOWER,UPPER) AND INCREMENT'
      READ*,COFRNL,COFRNU,COFINC
 30   PRINT*,'INPUT SIGNAL RANGE(LOWER,UPPER) AND INCREMENT'
      READ*,SIGRNL,SIGRNU,SIGINC
      PRINT*,'INPUT AMPLITUDE RANGE(LOWER,UPPER) AND INCREMENT'
      READ*,AMPRNL,AMPRNU,AMPINC
C
C     INITIALISE ANALYSIS PARAMETERS ACCORDING TO I/P
C
      COFPCN=COFRNU
      SIGPCN=SIGRNU
      SIGAMP=AMPRNU
C
      WRITE(10,111)
      IF (COFSEL.EQ.2) COFPCN=DOLLAR
      IF (COFSEL.EQ.2) COFRNU=DOLLAR
      WRITE (10,171)
      WRITE (10,291)
```

```
      WRITE (10,301)
C
 40   CALL ANMAN
      ITEMP1=IBLANK
      IF (OVFLOW.EQ.1) ITEMP1=ISTAR
C
C     EVALUATE NOISE SIGNAL (TRNOIS) - AND CALCULATE RMS NOISE
C
      TEMP1=0.0
      DO 50 LP1=1,SIMLEN
         TRNOIS(LP1)=TRNIP(LP1)-TRNFP(LP1)
         TEMP2=TRNOIS(LP1)**2
         TEMP1=TEMP1+TEMP2
 50   CONTINUE
      IF (TEMP1.NE.0.0) GO TO 60
      NOSRMS=0.0
      GO TO 70
 60   NOSRMS=SQRT(TEMP1/FLOAT(SIMLEN))
C
C     CALCULATE RMS SIGNAL VOLTAGE
C
 70   TEMP3=0.0
      DO 80 LP2=1,SIMLEN
         TEMP3=TRNFP(LP2)**2 + TEMP3
 80   CONTINUE
      IF (TEMP3.NE.0.0) GO TO 90
      SIGRMS=0.0
      SNRAT=0.0
      GO TO 100
 90   SIGRMS=SQRT(TEMP3/FLOAT(SIMLEN))
C
C     CALCULATE SIGNAL TO NOISE RATIO IN DB
C
      IF (NOSRMS.EQ.0.0) GO TO 100
      SNRAT=20*ALOG10(SIGRMS/NOSRMS)
 100  ISNRAT=INT(SNRAT)
      IF (COFSEL.EQ.2) WRITE (10,201)
     &SIGAMP,COFPCN,SIGPCN,SIGRMS,NOSRMS,ISNRAT,ITEMP1
      IF (COFSEL.EQ.1) WRITE (10,202)
     &SIGAMP,COFPCN,SIGPCN,SIGRMS,NOSRMS,ISNRAT,ITEMP1
      SIGPCN=SIGPCN-SIGINC
      IF (SIGPCN.LT.SIGRNL) GO TO 110
      GO TO 40
C
 110  IF (COFSEL.EQ.2) GO TO 120
      COFPCN=COFPCN-COFINC
      IF (COFPCN.LT.COFRNL) GO TO 120
      SIGPCN=SIGRNU
      GO TO 40
C
 120  SIGAMP=SIGAMP-AMPINC
      IF (SIGAMP.LT.AMPRNL) RETURN
      COFPCN=COFRNU
      SIGPCN=SIGRNU
      GO TO 40
C
```

```
 111  FORMAT(70('-'))
 121  FORMAT (('ADDER OVERFLOW'),47X,('SELECTION'))
 131  FORMAT (53H1:IMPULSE,2:STEP,3:RAMP,4:COSINE,5:UDR,6:NUDR,7:GAUSS,
     &        12X,I1)
 171  FORMAT (6HSIGNAL,5X,11HCOEFFICIENT,2X,6HSIGNAL,7X,6HOUTPUT,5X
     &        ,6HOUTPUT,7X,9HSIGNAL TO)
 291  FORMAT (9HAMPLITUDE,2X,9HPRECISION,4X,9HPRECISION,4X,6HSIGNAL,
     &        5X,5HNOISE,6X,11HNOISE RATIO)
 301  FORMAT (7H(VOLTS),8X,6H(BITS),6X,6H(BITS),3X,11H(VOLTS RMS),3X,
     &        11H(VOLTS RMS),3X,4H(DB))
 201  FORMAT (F4.2,11X,A2,11X,I2,3X,E12.4,3X,E12.4,5X,I3,1X,A1)
 202  FORMAT (F4.2,11X,I2,11X,I2,3X,E12.4,3X,E12.4,5X,I3,1X,A1)
 151  FORMAT (31HINFINITE PRECISION COEFFICIENTS)
C
 161  FORMAT (19HMULTIPLIER ROUNDING,42X,9HSELECTION)
 191  FORMAT (19HMULTIPLIER OVERFLOW,42X,9HSELECTION)
 141  FORMAT (12H1:SATURATING,8X,9H2:ZEROING,7X,15H3:2S COMPLEMENT,14X,
     &        I1)
 181  FORMAT (17H1:2S COMP. TRUNC.,3X,13H2:MAG. TRUNC.,3X,10H3:ROUNDING
     &        ,19X,I1)
 211  FORMAT (15H1:2S COMPLEMENT,5X,12H2:SATURATING,4X,9H3:ZEROING,20X,
     &        I1)
 241  FORMAT (8HA/D TYPE,53X,9HSELECTION)
 221  FORMAT (12HINPUT SIGNAL,49X,9HSELECTION)
 231  FORMAT (53H1:IMPULSE,2:STEP,3:RAMP,4:COSINE,5:UDR,6:NUDR,7:GAUSS,
     &        12X,I1)
 261  FORMAT (18HSIGNAL PRECISION =,1X,I2,2X,4HBITS)
 237  FORMAT (11HCOEFFICIENT,2X,F9.7,2X,24HCANNOT BE REPRESENTED BY,2X,
     &        I3,2X,4HBITS)
 271  FORMAT (11HRAMP RATE =,1X,F9.7,2X,16HVOLTS PER SECOND)
 281  FORMAT (18HSIGNAL FREQUENCY =,1X,F9.7,2X,11HRADIANS/SEC)
 311  FORMAT (20A4)
 401  FORMAT(('SIMULATION LENGTH.....'),I4)
 411  FORMAT(('OUTPUT NODE .....'),I2)
C
      END
C
C*********************************************************************
C
C     SUBROUTINE TO EVALUATE NODE VALUES AT A SAMPLING INSTANT.
C     SWEEPS THROUGH ARRAY NETWOR AND FOR EACH ELEMENT, COMPUTES THE
C     OUTPUT NODE SIGNAL VALUE.
C
C---------------------------------------------------------------------
C
C     SUBROUTINE TRN(INTEGR,VOLTAG)
C
      INTEGER T0,T1
      DOUBLE PRECISION ANODES
C
      COMMON NETWOR(100,4),COEFF(50,2),COFRND(50),NUMULT,NUMNWK
      COMMON/ONE/NODES(100,3),T0,T1,MAXNUM
      COMMON/NINE/ANODES(100,2)
C
C     LOAD SIGNAL VALUE INTO NODES AND ANODES AND EVALUATE NODE VALUES
C
```

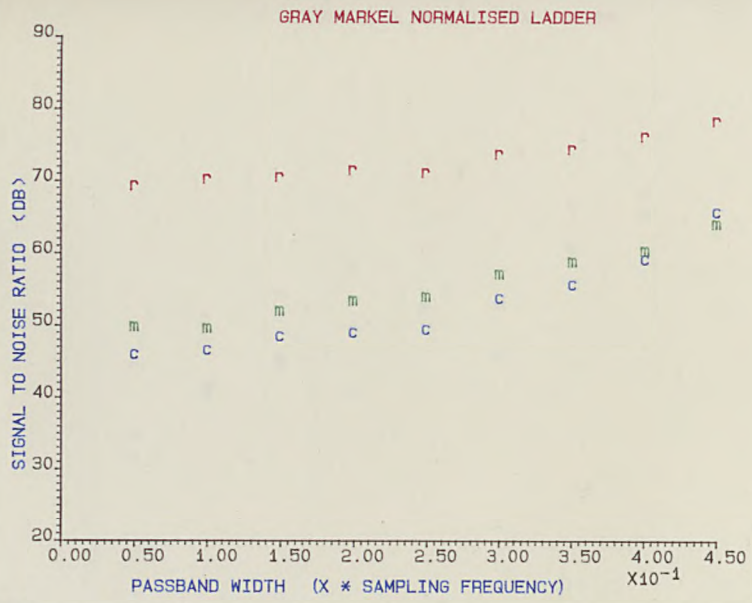
```
      NODES(1,T1)=INTEGR
      ANODES(1,T1)=DBLE(VOLTAG)
      DO 30 LP2=1,(NUMNWK+1)
         IF (NETWOR(LP2,1).NE.3) GO TO 10
         NODES(NETWOR(LP2,2),T1) = NODES(NETWOR(LP2,3),T0)
         ANODES(NETWOR(LP2,2),T1) = ANODES(NETWOR(LP2,3),T0)
 10      IF (NETWOR(LP2,1).NE.4) GO TO 20
         NODES(NETWOR(LP2,2),T1) = IADD(LP2)
         ANODES(NETWOR(LP2,2),T1) = ANODES(NETWOR(LP2,3),T1) +
     &   ANODES(NETWOR(LP2,4),T1)
 20      IF (NETWOR(LP2,1).NE.-7) GO TO 30
         NODES(NETWOR(LP2,2),T1) = MULT(LP2)
         ANODES(NETWOR(LP2,2),T1) = ANODES(NETWOR(LP2,3),T1) *
     &   DBLE(COFRND(NETWOR(LP2,4)))
 30   CONTINUE
C
      RETURN
      END
C
C*********************************************************************
C
C     MAIN  SUBROUTINE FOR TRANSIENT ANALYSIS.
C     PERFORMS ALL NECESSARY INPUT OUTPUT FUNCTIONS FOR TRANSIENT
C     ANALYSIS
C
C---------------------------------------------------------------------
C
      SUBROUTINE TRNOUT
C
      INTEGER SIMLEN,SIGPCN,COFPCN,ADDTYP,SIGTYP,COFSEL,ITITLE(20)
      INTEGER NODOVL(100),ATDTYP
      REAL IPIP,NOSRMS
C
      COMMON NETWOR(100,4),COEFF(50,2),COFRND(50),NUMULT,NUMNWK
      COMMON/ONE/NODES(100,3),T0,T1,MAXNUM
      COMMON/TWO/TIME(1024),TRNIP(1024),TRNFP(1024),TRNOIS(1024),SIM
      COMMON/THREE/IPIP(1024),FPIP(1024),SIGPCN,ATDTYP
      COMMON/FIVE/MTYP,ADDTYP,MULTOF,COFSEL,COFPCN,TERMTP,ANTYP,NODO
      COMMON/SIX/INTEGR,VOLTAG,SIGTYP,SIGAMP,TIMSTP,RESN,RMPRTE,SIGF
C
      PRINT*,' '
      PRINT*,'    TRANSIENT ANALYSIS'
      PRINT*,'    ------------------'
      PRINT*,' '
      PRINT*,'ADDER TYPE'
      PRINT*,'1:SATURATING  2:ZEROING  3:2s COMPLEMENT'
      READ*,ADDTYP
      PRINT*,'MULTIPLIER TYPE'
      PRINT*,'1: 2s COMP TRUNC.  2: MAGNITUDE TRUNC. 3:ROUNDING'
      READ*,MTYP
      PRINT*,'MULTIPLIER OVERFLOW TYPE'
      PRINT*,'1:2s COMPLEMENT  2:SATURATING  3:ZEROING'
      READ*,MULTOF
      PRINT*,'1:IMPULSE,2:STEP,3:RAMP,4:COSINE,5:UDR,6:NUDR,7:GAUSS'
      READ*,SIGTYP
      PRINT*,'SIGAMP : MAX   1.0 VOLTS'
      READ*,SIGAMP
      PRINT*,'SIGNAL QUANTISATION IN BITS'
      READ*,SIGPCN
      IF (SIGTYP.NE.3) GO TO 1
      PRINT*,'RAMP RATE '
      READ*,RMPRTE
 1    IF (SIGTYP.NE.4) GO TO 2
      PRINT*,'SIGNAL FREQ IN RADS/SEC - SAMPLING FREQ = 2*PI RADS/SEC'
      READ*,SIGFR
 2    PRINT*,'1: FINITE PRECN COEFFS    2:INFINITE PRECN COEFFS'
      READ*,COFSEL
      IF (COFSEL.NE.1) GO TO 3
      PRINT*,'COEFFICIENT QUANTISATION IN BITS'
      READ*,COFPCN
 3    PRINT*,'SIMULATION LENGTH :  MAX   1024'
      READ*,SIMLEN
      PRINT*,'OUTPUT NODE'
      READ*,NODOUT
      PRINT*,'A/D CONVERTER'
      PRINT*,'1: 2s COMP TRUNC.  2: MAGNITUDE TRUNC. 3:ROUNDING'
      READ*,ATDTYP
C
      CALL ANMAN
C
C     EVALUATE NOISE SIGNAL (TRNOIS) - AND CALCULATE RMS NOISE
C
      TEMP1=0.0
      DO 10 LP1=1,SIMLEN
         TRNOIS(LP1)=TRNIP(LP1)-TRNFP(LP1)
         TEMP2=TRNOIS(LP1)**2
         TEMP1=TEMP1+TEMP2
 10   CONTINUE
      IF (TEMP1.NE.0.0) GO TO 15
      NOSRMS=0.0
      GO TO 17
 15   NOSRMS=SQRT(TEMP1/FLOAT(SIMLEN))
C
C     CALCULATE RMS SIGNAL VOLTAGE
C
 17   TEMP3=0.0
      DO 20 LP2=1,SIMLEN
         TEMP3=TRNFP(LP2)**2 + TEMP3
 20   CONTINUE
      IF (TEMP3.NE.0.0) GO TO 25
      SIGRMS=0.0
      GO TO 27
 25   SIGRMS=SQRT(TEMP3/FLOAT(SIMLEN))
C
C     CALCULATE SIGNAL TO NOISE RATIO IN DB
C
      IF (NOSRMS.EQ.0.0) GO TO 27
      SNRAT=20*ALOG10(SIGRMS/NOSRMS)
C
C     COPY NODES WHERE OVERFLOW OCCURS TO ARRAY NODOVL
C
 27   IPTR=1
```

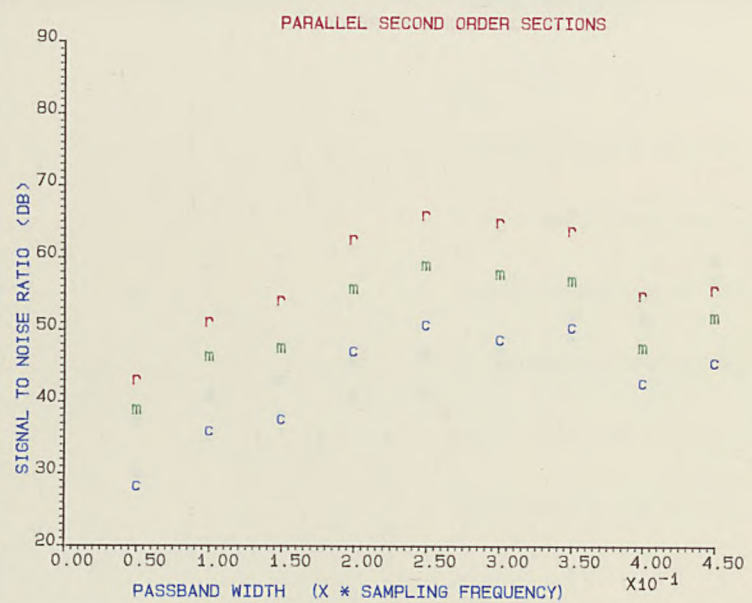
```
      DO 30 LP3=1,100
         IF (NODES(LP3,3).EQ.0) GO TO 30
         NODOVL(IPTR)=NODES(LP3,3)
         IPTR=IPTR+1
 30   CONTINUE
C
      PRINT*,'INPUT HEADER FOR PRINTED RESULTS'
      READ(5,281)(ITITLE(I),I=1,20)
      WRITE (10,281) (ITITLE(I),I=1,20)
C
      WRITE (10,115)
      WRITE (10,121)
      WRITE (10,151) ADDTYP
      WRITE (10,131)
      WRITE (10,161) MTYP
      WRITE (10,141)
      WRITE (10,171) MULTOF
      WRITE (10,175)
      WRITE (10,161) ATDTYP
      WRITE (10,115)
      WRITE (10,181)
      WRITE (10,191) SIGTYP
      WRITE (10,211) SIGAMP
      IF (SIGTYP.EQ.3) WRITE(10,241) RMPRTE
      IF (SIGTYP.EQ.4) WRITE(10,261) SIGFR
      WRITE (10,231) SIGPCN
      WRITE (10,115)
      IF (COFSEL.EQ.1) WRITE(10,235) COFPCN
      DO 32 LP4=1,NUMULT
         IF (COFRND(LP4).NE.0.0) GO TO 32
         WRITE (10,237) COEFF(LP4,1),COFPCN
 32   CONTINUE
      WRITE (10,115)
      WRITE (10,251) SIGRMS,NOSRMS
      WRITE (10,255) SNRAT
      WRITE (10,115)
      IF (NODOVL(1).EQ.0) GO TO 40
      WRITE (10,271)
      DO 35 LP4=1,IPTR
         WRITE (10,*) NODOVL(LP4)
 35   CONTINUE
C
 40   PRINT*,'DISPLAY MENU'
      PRINT*,'1: IDEAL RESPONSE'
      PRINT*,'2: REAL RESPONSE'
      PRINT*,'3: IDEAL INPUT'
      PRINT*,'4: REAL INPUT'
      PRINT*,'5: NOISE'
      PRINT*,'6: OVERFLOW NODES'
      PRINT*,'7: REAL AND IDEAL RESPONSE'
      PRINT*,'8: QUIT'
      READ*,MENU
      IF (MENU.EQ.1) CALL PLOT(TIME,TRNIP,SIMLEN)
      IF (MENU.EQ.2) CALL PLOT(TIME,TRNFP,SIMLEN)
      IF (MENU.EQ.3) CALL PLOT(TIME,IPIP,SIMLEN)
      IF (MENU.EQ.4) CALL PLOT(TIME,FPIP,SIMLEN)
      IF (MENU.EQ.5) CALL PLOT(TIME,TRNOIS,SIMLEN)
      IF (MENU.NE.6) GO TO 50
      DO 45 LP5=1,IPTR-1
         PRINT*,NODOVL(LP5)
 45   CONTINUE
C
 50   IF (MENU.EQ.7) CALL TSTPLT
      IF (MENU.EQ.8) RETURN
      GO TO 40
C
 111  FORMAT(10(4X,I3))
 115  FORMAT(70('-'))
 121  FORMAT (14HADDER OVERFLOW,47X,9HSELECTION)
 131  FORMAT (19HMULTIPLIER ROUNDING,42X,9HSELECTION)
 141  FORMAT (19HMULTIPLIER OVERFLOW,42X,9HSELECTION)
 151  FORMAT (12H1:SATURATING,8X,9H2:ZEROING,7X,15H3:2S COMPLEMENT,14X,
     &        I1)
 161  FORMAT (17H1:2S COMP. TRUNC.,3X,13H2:MAG. TRUNC.,3X,10H3:ROUNDING
     &         ,19X,I1)
 171  FORMAT (15H1:2S COMPLEMENT,5X,12H2:SATURATING,4X,9H3:ZEROING,20X,
     &        I1)
 175  FORMAT (8HA/D TYPE,53X,9HSELECTION)
 181  FORMAT (12HINPUT SIGNAL,49X,9HSELECTION)
 191  FORMAT (53H1:IMPULSE,2:STEP,3:RAMP,4:COSINE,5:UDR,6:NUDR,7:GAUSS,
     &        12X,I1)
 211  FORMAT (18HSIGNAL AMPLITUDE =,1X,F9.7,2X,5HVOLTS)
 221  FORMAT (19HSIMULATION LENGTH =,1X,I3,10X,13HOUTPUT NODE =,1X,I2)
 231  FORMAT (18HSIGNAL PRECISION =,1X,I2,2X,4HBITS)
 235  FORMAT (24HCOEFFICIENT PRECISION = ,I2,2X,4HBITS)
 237  FORMAT (11HCOEFFICIENT,2X,F9.7,2X,24HCANNOT BE REPRESENTED BY,2X,
     &         I3,2X,4HBITS)
 241  FORMAT (11HRAMP RATE =,1X,F9.7,2X,16HVOLTS PER SECOND)
 251  FORMAT (9HSIGNAL = ,E12.4,2X,9HVOLTS RMS,7X,
     &         8HNOISE = ,E12.4,2X,9HVOLTS RMS)
 255  FORMAT (23HSIGNAL TO NOISE RATIO =,1X,E12.4,2X,2HDB)
 261  FORMAT (18HSIGNAL FREQUENCY =,1X,F9.7,2X,11HRADIANS/SEC)
 271  FORMAT (39HOVERFLOW OCCURED AT THE FOLLOWING NODES)
 281  FORMAT (20A4)
C
      END
C
C*********************************************************************
C
C     SUBROUTINE FOR GRAPHIC DISPLAY OF FINITE PRECISION AND INFINITE
C     PRECISION TRANSIENT RESPONSES ON THE SAME AXIS.
C     THIS SUBROUTE USES GINO ROUTINES.
C
C---------------------------------------------------------------------
C
      SUBROUTINE TSTPLT
C
      INTEGER SIMLEN,TERMTP
      REAL IPIP
C
      COMMON/TWO/TIME(1024),TRNIP(1024),TRNFP(1024),TRNOIS(1024),SIMLEN
      COMMON/THREE/IPIP(1024),FPIP(1024),SIGPCN,ATDTYP
```

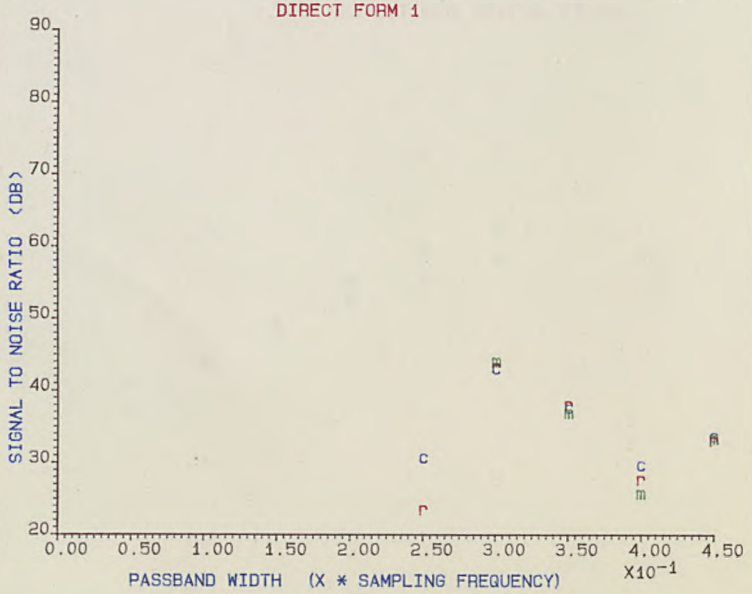
```
      COMMON/FIVE/MTYP,ADDTYP,MULTOF,COFSEL,COFPCN,TERMTP,ANTYP,NODOUT
C
      PRINT*,'PLOT ANY TWO ON SAME AXES'
      PRINT*,'1: FINITE PRECN RESPONSE'
      PRINT*,'2:INFINITE PRECN RESPONSE'
      PRINT*,'3: FINITE PRECN INPUT'
      PRINT*,'4: INFINITE PRECN INPUT'
      READ*,MENU1,MENU2
C
      IF (TERMTP.EQ.1) CALL T4010
      IF (TERMTP.EQ.2) CALL APDS4
      CALL PICCLE
C
      XMAX=FLOAT(SIMLEN)
      CALL AXISCA (1,10,0.0,XMAX,1)
      CALL AXISCA (1,10,-1.0,1.0,2)
      CALL AXIDRA (2,1,1)
      CALL AXIDRA (-2,-1,2)
      IF ((MENU1.EQ.1).OR.(MENU2.EQ.1)) CALL GRAPOL(TIME,TRNFP,SIMLEN)
      IF ((MENU1.EQ.2).OR.(MENU2.EQ.2)) CALL GRAPOL(TIME,TRNIP,SIMLEN)
      IF ((MENU1.EQ.3).OR.(MENU2.EQ.3)) CALL GRAPOL(TIME,FPIP,SIMLEN)
      IF ((MENU1.EQ.4).OR.(MENU2.EQ.4)) CALL GRAPOL(TIME,IPIP,SIMLEN)
      CALL CHAMOD
      READ*,X
      CALL PICCLE
      CALL DEVEND
      RETURN
C
      END
C
C*************************************************************************
```

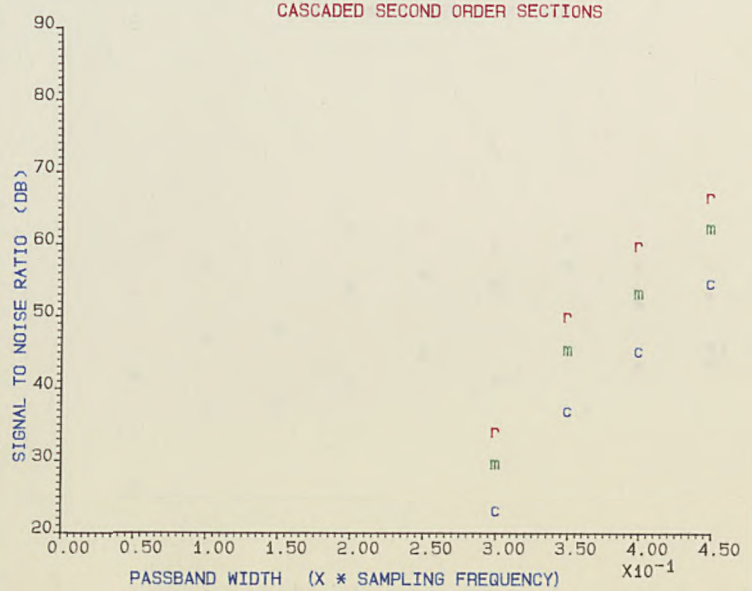
APPENDIX A2

INPUT SIGNAL : UNIFORMLY DISTRIBUTED RANDOM

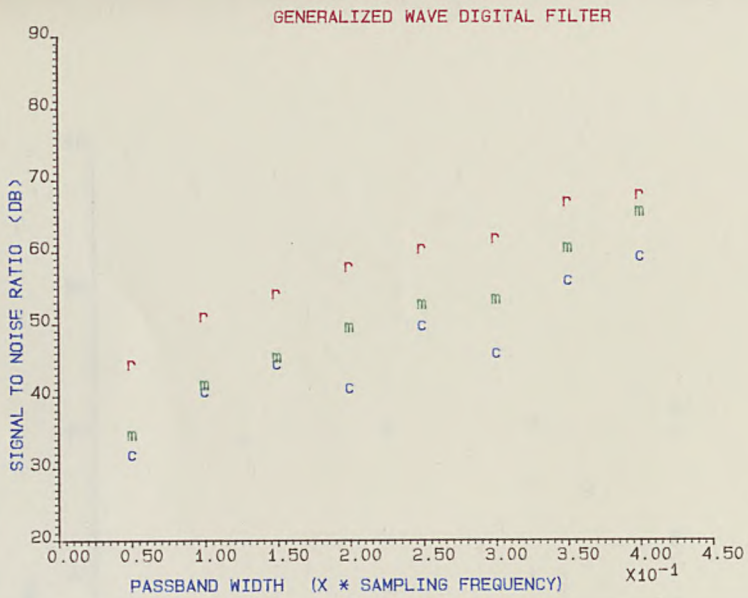


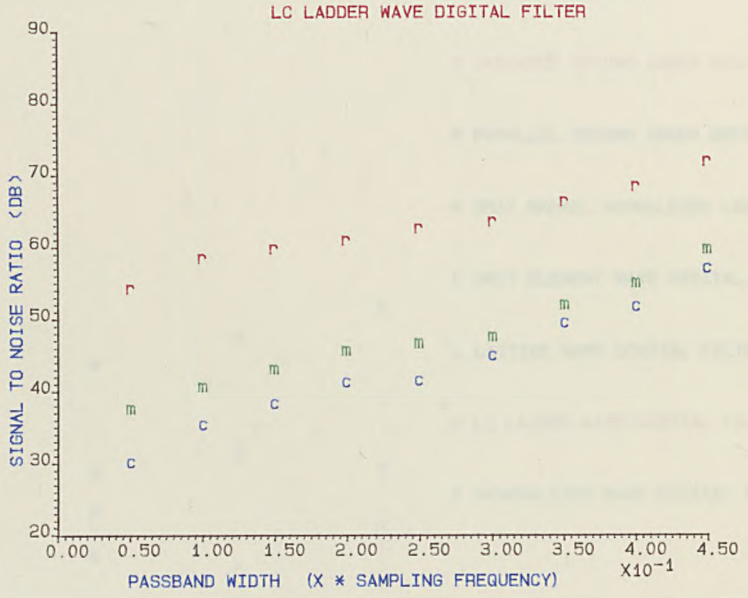


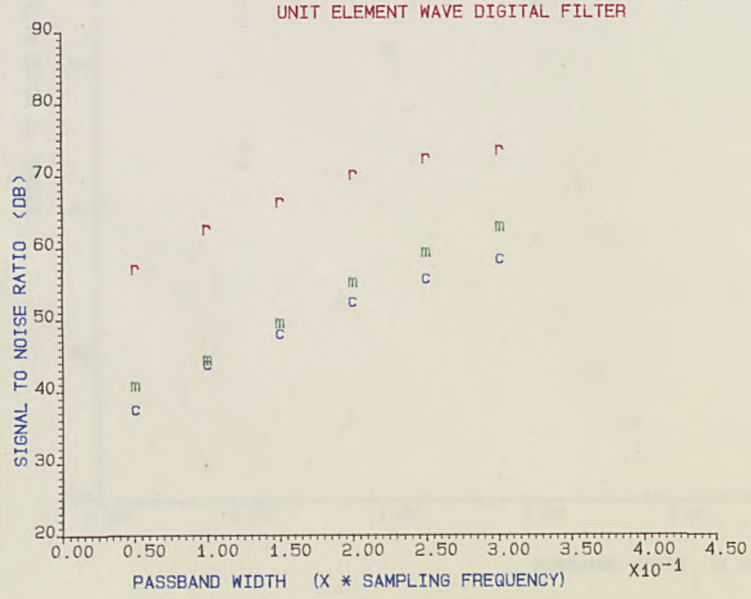


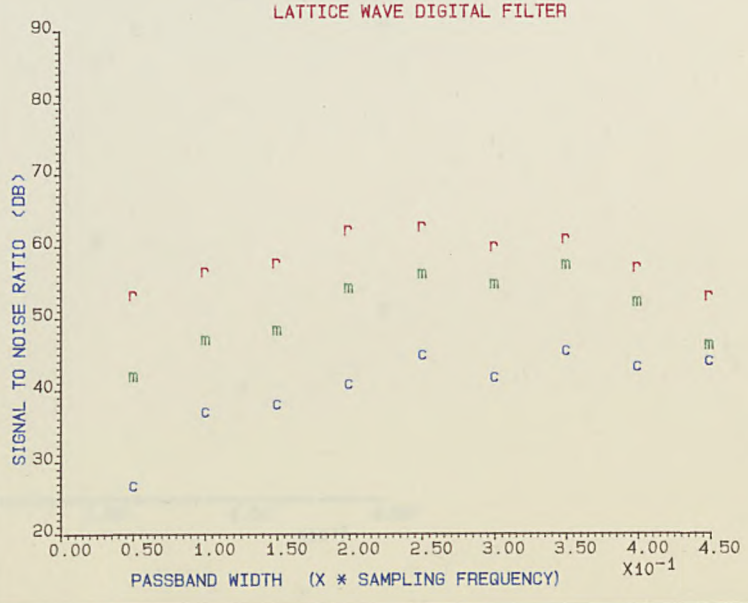


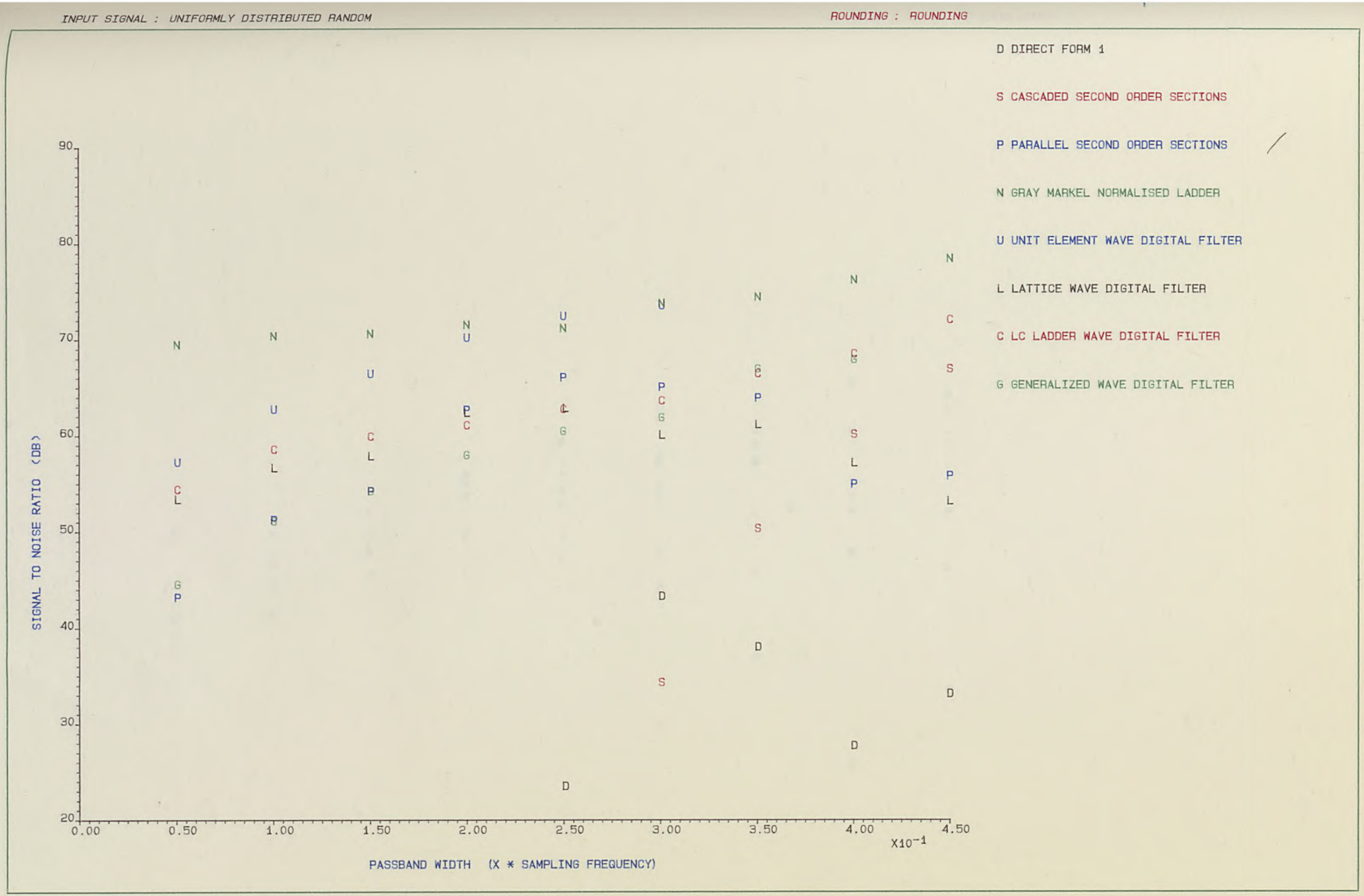
INPUT SIGNAL : UNIFORMLY DISTRIBUTED RANDOM





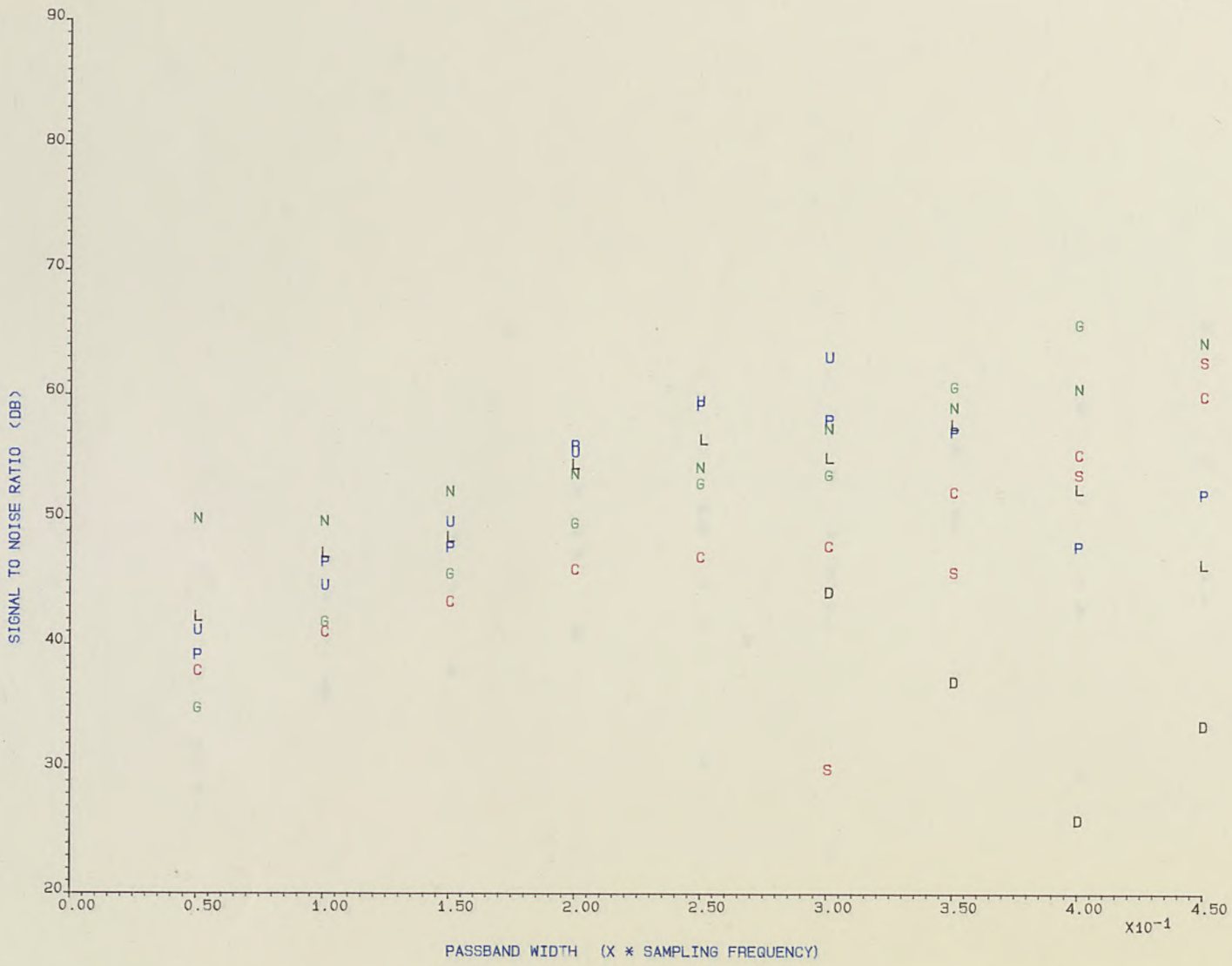





INPUT SIGNAL : UNIFORMLY DISTRIBUTED RANDOM
ROUNDING : ROUNDING
D DIRECT FORM 1
S CASCADED SECOND ORDER SECTIONS
P PARALLEL SECOND ORDER SECTIONS
N GRAY MARKEL NORMALISED LADDER
U UNIT ELEMENT WAVE DIGITAL FILTER
L LATTICE WAVE DIGITAL FILTER
C LC LADDER WAVE DIGITAL FILTER
G GENERALIZED WAVE DIGITAL FILTER
SIGNAL TO NOISE RATIO (DB)
90
80
70
60
50
40
30
20
0.00
0.50
1.00
1.50
2.00
2.50
3.00
3.50
4.00
4.50
X10-1
PASSBAND WIDTH (X * SAMPLING FREQUENCY)

INPUT SIGNAL : UNIFORMLY DISTRIBUTED RANDOM

ROUNDING : MAGNITUDE TRUNCATING



D DIRECT FORM 1

S CASCADED SECOND ORDER SECTIONS

P PARALLEL SECOND ORDER SECTIONS

N GRAY MARKEL NORMALISED LADDER

U UNIT ELEMENT WAVE DIGITAL FILTER

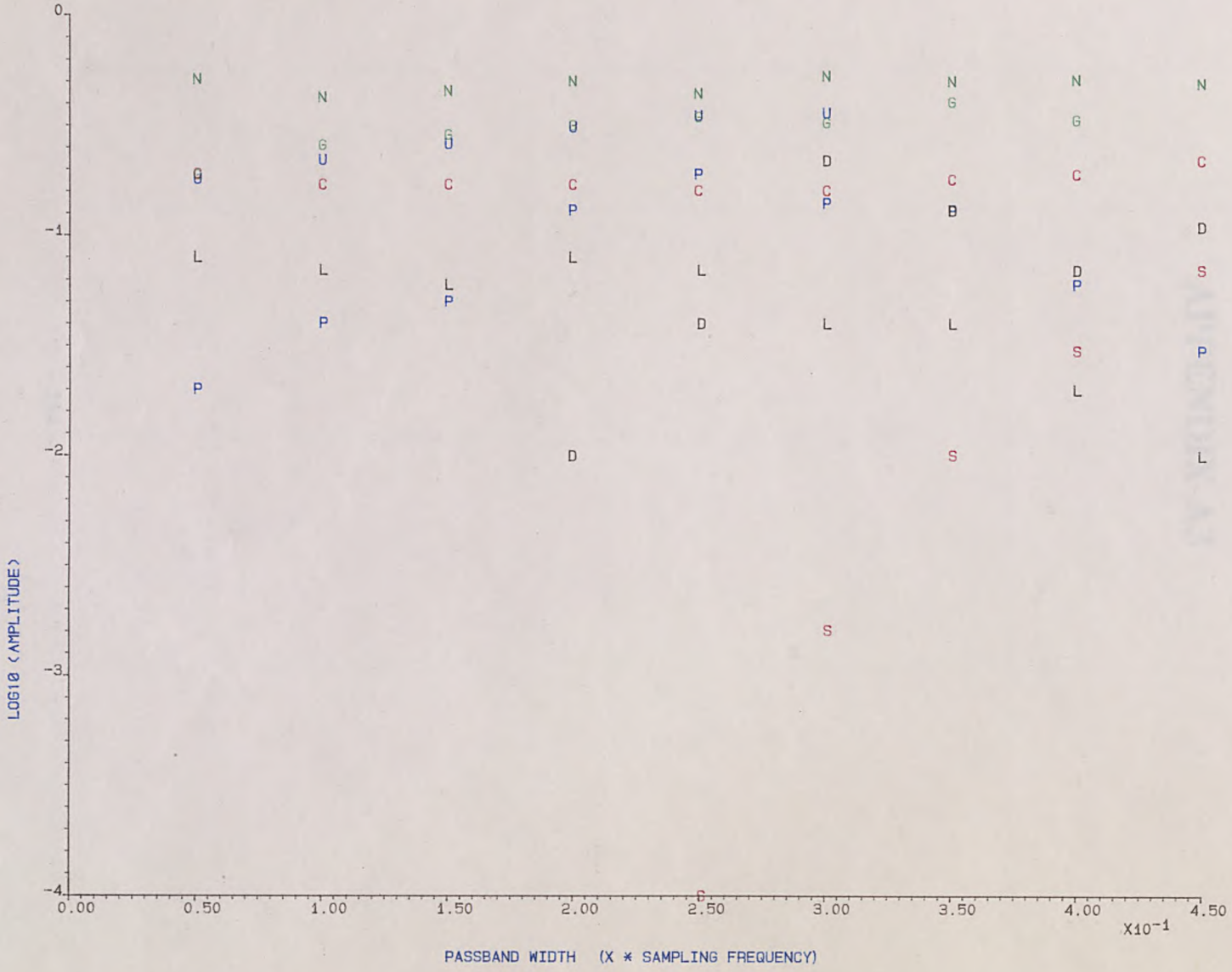
L LATTICE WAVE DIGITAL FILTER

C LC LADDER WAVE DIGITAL FILTER

G GENERALIZED WAVE DIGITAL FILTER


INPUT SIGNAL : UNIFORMLY DISTRIBUTED RANDOM
ROUNDING : 2S COMPLEMENT TRUNCATING
D DIRECT FORM 1
S CASCADED SECOND ORDER SECTIONS
P PARALLEL SECOND ORDER SECTIONS
N GRAY MARKEL NORMALISED LADDER
U UNIT ELEMENT WAVE DIGITAL FILTER
L LATTICE WAVE DIGITAL FILTER
C LC LADDER WAVE DIGITAL FILTER
G GENERALIZED WAVE DIGITAL FILTER
SIGNAL TO NOISE RATIO (DB)
90
80
70
60
50
40
30
20
0.00
0.50
1.00
1.50
2.00
2.50
3.00
3.50
4.00
4.50
X10-1
PASSBAND WIDTH (X * SAMPLING FREQUENCY)

INPUT SIGNAL : UNIFORMLY DISTRIBUTED RANDOM



D DIRECT FORM 1

S CASCADED SECOND ORDER SECTIONS

P PARALLEL SECOND ORDER SECTIONS

N GRAY MARKEL NORMALISED LADDER

U UNIT ELEMENT WAVE DIGITAL FILTER

L LATTICE WAVE DIGITAL FILTER

C LC LADDER WAVE DIGITAL FILTER

G GENERALIZED WAVE DIGITAL FILTER

APPENDIX A3

PARALLEL SECOND ORDER SECTIONS				
PASS BAND WIDTH	SIGNAL TYPE	ROUNDING	MAGNITUDE TRUNCATION	2's COMPLEMENT TRUNCATION
$0.05 f_s$	COS	0.2451 E-4	0.3631 E-4	0.1343 E-3
	UDR	0.2503 E-4	0.3884 E-4	0.1335 E-3
	GAUS	0.2408 E-4	0.4165 E-4	0.1346 E-3
$0.1 f_s$	COS	0.2800 E-4	0.3569 E-4	0.1632 E-3
	UDR	0.2832 E-4	0.4942 E-4	0.1622 E-3
	GAUS	0.2791 E-4	0.5034 E-4	0.1653 E-3
$0.15 f_s$	COS	0.3025 E-4	0.5842 E-4	0.197 E-3
	UDR	0.3075 E-4	0.6298 E-4	0.1988 E-3
	GAUS	0.3053 E-4	0.6166 E-4	0.1988 E-3
$0.2 f_s$	COS	0.3245 E-4	0.5314 E-4	0.1921 E-3
	UDR	0.3252 E-4	0.7060 E-4	0.1923 E-3
	GAUS	0.3430 E-4	0.7080 E-4	0.1930 E-3
$0.25 f_s$	COS	0.3617 E-4	0.5651 E-4	0.2066 E-3
	UDR	0.3731 E-4	0.7925 E-4	0.2059 E-3
	GAUS	0.3797 E-4	0.8100 E-4	0.2059 E-3
$0.3 f_s$	COS	0.3422 E-4	0.4792 E-4	0.2127 E-3
	UDR	0.3341 E-4	0.7476 E-4	0.2142 E-3
	GAUS	0.3374 E-4	0.7466 E-4	0.2126 E-3
$0.35 f_s$	COS	0.3619 E-4	0.9000 E-4	0.1758 E-3
	UDR	0.3783 E-4	0.8281 E-4	0.1752 E-3
	GAUS	0.3816 E-4	0.8872 E-4	0.1757 E-3
$0.4 f_s$	COS	0.5703 E-4	0.1272 E-3	0.2116 E-3
	UDR	0.5411 E-4	0.1221 E-3	0.2148 E-3
	GAUS	0.5599 E-4	0.1243 E-3	0.2134 E-3
$0.45 f_s$	COS	0.2599 E-4	0.4259 E-4	0.8073 E-4
	UDR	0.2568 E-4	0.3913 E-4	0.8090 E-4
	GAUS	0.2446 E-4	0.4122 E-4	0.8164 E-4

CASCADED SECOND ORDER SECTIONS				
PASS BAND WIDTH	SIGNAL TYPE	ROUNDING	MAGNITUDE TRUNCATION	2's COMPLEMENT TRUNCATION
$0.05 f_s$	COS UDR GAUS			
$0.1 f_s$	COS UDR GAUS			
$0.15 f_s$	COS UDR GAUS			
$0.2 f_s$	COS UDR GAUS			
$0.25 f_s$	COS UDR GAUS			
$0.3 f_s$	COS UDR GAUS	0.1224 E-1 0.1108 E-1 0.1123 E-1	0.2375 E-1 0.2081 E-1 0.2068 E-1	0.6991 E-1 0.7174 E-1 0.7145 E-1
$0.35 f_s$	COS UDR GAUS	0.7604 E-3 0.7521 E-3 0.7635 E-3	0.2035 E-2 0.1216 E-2 0.1402 E-2	0.4705 E-2 0.4830 E-2 0.4767 E-2
$0.4 f_s$	COS UDR GAUS	0.1196 E-3 0.1161 E-3 0.1163 E-3	0.2878 E-3 0.1985 E-3 0.1920 E-3	0.6704 E-3 0.6679 E-3 0.6689 E-3
$0.45 f_s$	COS UDR GAUS	0.3462 E-4 0.3505 E-4 0.3503 E-4	0.8737 E-4 0.5442 E-4 0.5419 E-4	0.1648 E-3 0.1656 E-3 0.1646 E-3

GRAY-MARKEL NORMALIZED LADDER

PASS BAND WIDTH	SIGNAL TYPE	ROUNDING	MAGNITUDE TRUNCATION	2's COMPLEMENT TRUNCATION
0.05 f_s	COS	0.8959 E-4	0.9667 E-3	0.1477 E-2
	UDR	1.008 E-3	0.9439 E-3	0.1479 E-2
	GAUS	0.9534 E-4	0.9474 E-3	0.1488 E-2
0.1 f_s	COS	0.8201 E-4	0.7353 E-3	0.1165 E-2
	UDR	0.7691 E-4	0.8174 E-3	0.1170 E-2
	GAUS	0.7792 E-4	0.7566 E-3	0.1173 E-2
0.15 f_s	COS	0.7241 E-4	0.6822 E-3	0.9705 E-3
	UDR	0.7758 E-4	0.6445 E-3	0.9708 E-3
	GAUS	0.7031 E-4	0.6628 E-3	0.9733 E-3
0.2 f_s	COS	0.7360 E-4	0.5798 E-3	0.9405 E-3
	UDR	0.7235 E-4	0.5766 E-3	0.9417 E-3
	GAUS	0.7286 E-4	0.5930 E-3	0.9418 E-3
0.25 f_s	COS	0.6884 E-4	0.4398 E-3	0.8243 E-3
	UDR	0.6835 E-4	0.4863 E-3	0.8194 E-3
	GAUS	0.6611 E-4	0.4837 E-3	0.8203 E-3
0.3 f_s	COS	0.6628 E-4	0.3499 E-3	0.6994 E-3
	UDR	0.6321 E-4	0.4587 E-3	0.6977 E-3
	GAUS	0.6266 E-4	0.4451 E-3	0.6975 E-3
0.35 f_s	COS	0.5519 E-4	0.2697 E-3	0.4826 E-3
	UDR	0.5639 E-4	0.3307 E-3	0.4871 E-3
	GAUS	0.5415 E-4	0.3422 E-3	0.4871 E-3
0.4 f_s	COS	0.4793 E-4	0.1624 E-3	0.3239 E-3
	UDR	0.4593 E-4	0.2762 E-3	0.3226 E-3
	GAUS	0.4723 E-4	0.2628 E-3	0.3252 E-3
0.45 f_s	COS	0.3789 E-4	0.1154 E-3	0.1683 E-3
	UDR	0.3914 E-4	0.1979 E-3	0.1694 E-3
	GAUS	0.3957 E-4	0.2112 E-3	0.1693 E-3

DIRECT FORM II

PASS BAND WIDTH	SIGNAL TYPE	ROUNDING	MAGNITUDE TRUNCATION	2's COMPLEMENT TRUNCATION
$0.05 f_s$	COS			
	UDR			
	GAUS			
$0.1 f_s$	COS			
	UDR			
	GAUS			
$0.15 f_s$	COS			
	UDR			
	GAUS			
$0.2 f_s$	COS			
	UDR			
	GAUS			
$0.25 f_s$	COS	0.1137 E-2	0.1332 E-2	0.1154 E-2
	UDR	0.1234 E-2	0.1343 E-2	0.1130 E-2
	GAUS	0.1202 E-2	0.1466 E-2	0.1446 E-2
$0.3 f_s$	COS	0.1024 E-2	0.1065 E-2	0.1055 E-2
	UDR	0.1268 E-2	0.1131 E-2	0.1101 E-2
	GAUS	0.1073 E-2	0.1175 E-2	0.8671 E-3
$0.35 f_s$	COS	0.7829 E-3	0.8326 E-3	0.8440 E-3
	UDR	0.8068 E-3	0.9324 E-3	0.6750 E-3
	GAUS	0.7248 E-3	0.9714 E-3	0.6704 E-3
$0.4 f_s$	COS	0.1082 E-2	0.2053 E-2	0.1036 E-2
	UDR	0.1171 E-2	0.1672 E-2	0.1152 E-2
	GAUS	0.1108 E-2	0.1745 E-2	0.1204 E-2
$0.45 f_s$	COS	0.2436 E-2	0.2393 E-2	0.2430 E-2
	UDR	0.2679 E-2	0.3069 E-2	0.2470 E-2
	GAUS	0.2437 E-2	0.2844 E-2	0.2273 E-2

CASCADED UNIT ELEMENTS WDF

PASS BAND WIDTH	SIGNAL TYPE	ROUNDING	MAGNITUDE TRUNCATION	2's COMPLEMENT TRUNCATION
$0.05 f_s$	COS	0.3854 E-4	0.2085 E-3	0.4072 E-3
	UDR	0.4764 E-4	0.2812 E-3	0.4121 E-3
	GAUS	0.4620 E-4	0.2830 E-3	0.4129 E-3
$0.1 f_s$	COS	0.4007 E-4	0.1922 E-3	0.3604 E-3
	UDR	0.4350 E-4	0.3362 E-3	0.3583 E-3
	GAUS	0.4303 E-4	0.2783 E-3	0.3631 E-3
$0.15 f_s$	COS	0.4102 E-4	0.1437 E-3	0.3095 E-3
	UDR	0.4082 E-4	0.2552 E-3	0.3060 E-3
	GAUS	0.3793 E-4	0.2431 E-3	0.3078 E-3
$0.2 f_s$	COS	0.3203 E-4	0.1093 E-3	0.2480 E-3
	UDR	0.3257 E-4	0.1782 E-3	0.2481 E-3
	GAUS	0.3391 E-4	0.1778 E-3	0.2462 E-3
$0.25 f_s$	COS	0.3306 E-4	0.7145 E-4	0.2131 E-3
	UDR	0.3261 E-4	0.1431 E-3	0.2132 E-3
	GAUS	0.3208 E-4	0.1374 E-3	0.2142 E-3
$0.3 f_s$	COS	0.3121 E-4	0.6304 E-4	0.1798 E-3
	UDR	0.3203 E-4	0.11 E-3	0.180 E-3
	GAUS	0.3189 E-4	0.1062 E-3	0.180 E-3
$0.35 f_s$	COS			
	UDR			
	GAUS			
$0.4 f_s$	COS			
	UDR			
	GAUS			
$0.45 f_s$	COS			
	UDR			
	GAUS			

IVR GENERALIZED WDF

PASS BAND WIDTH	SIGNAL TYPE	ROUNDING	MAGNITUDE TRUNCATION	2's COMPLEMENT TRUNCATION
$0.05 f_s$	COS	0.1266 E-3	0.2935 E-3	0.4143 E-3
	UDR	0.1112 E-3	0.2960 E-3	0.4116 E-3
	GAUS	0.1156 E-3	0.3106 E-3	0.4123 E-3
$0.1 f_s$	COS	0.9452 E-4	0.1948 E-3	0.2965 E-3
	UDR	0.9770 E-4	0.2661 E-3	0.2978 E-3
	GAUS	0.9898 E-4	0.2456 E-3	0.2988 E-3
$0.15 f_s$	COS	0.8563 E-4	0.1888 E-3	0.2634 E-3
	UDR	0.8637 E-4	0.2340 E-3	0.2625 E-3
	GAUS	0.8890 E-4	0.2253 E-3	0.2680 E-3
$0.2 f_s$	COS	0.7381 E-4	0.1420 E-3	0.2105 E-3
	UDR	0.7213 E-4	0.1708 E-3	0.2100 E-3
	GAUS	0.6988 E-4	0.1762 E-3	0.2102 E-3
$0.25 f_s$	COS	0.5965 E-4	0.1122 E-3	0.1787 E-3
	UDR	0.5827 E-4	0.1507 E-3	0.1780 E-3
	GAUS	0.6828 E-4	0.1481 E-3	0.1758 E-3
$0.3 f_s$	COS	0.5829 E-4	0.9915 E-4	0.1553 E-3
	UDR	0.5528 E-4	0.1431 E-3	0.1541 E-3
	GAUS	0.5667 E-4	0.1307 E-3	0.155 E-3
$0.35 f_s$	COS	0.5487 E-4	0.8721 E-4	0.1337 E-3
	UDR	0.4619 E-4	0.9210 E-4	0.1344 E-3
	GAUS	0.4865 E-4	0.9885 E-4	0.1347 E-3
$0.4 f_s$	COS	0.3588 E-4	0.5442 E-4	0.9064 E-4
	UDR	0.3400 E-4	0.4405 E-4	0.9059 E-4
	GAUS	0.3352 E-4	0.4329 E-4	0.8993 E-4
$0.45 f_s$	COS	0.8563 E-4	0.1888 E-3	0.2634 E-3
	UDR	0.8637 E-4	0.2340 E-3	0.2625 E-3
	GAUS	0.8890 E-4	0.2253 E-3	0.2680 E-3

LATTICE WDF BASED ON ALL-PASS SECTIONS				
PASS BAND WIDTH	SIGNAL TYPE	ROUNDING	MAGNITUDE TRUNCATION	2's COMPLEMENT TRUNCATION
$0.05 f_s$	COS	0.5554 E-4	0.3668E-3	0.6978 E-3
	UDR	0.5564 E-4	0.2191 E-3	0.6918E-3
	GAUS	0.5333 E-4	0.2270 E-3	0.7079 E-3
$0.1 f_s$	COS	0.5135 E-4	0.1997 E-3	0.5058E-3
	UDR	0.5355 E-4	0.1585 E-3	0.5048 E-3
	GAUS	0.4992 E-4	0.1588 E-3	0.5056 E-3
$0.15 f_s$	COS	0.4440 E-4	0.1425 E-3	0.4449 E-3
	UDR	0.4451 E-4	0.1358 E-3	0.4524 E-3
	GAUS	0.4649 E-4	0.1340 E-3	0.4458 E-3
$0.2 f_s$	COS	0.4036 E-4	0.1011 E-3	0.3560 E-3
	UDR	0.4132 E-4	0.103 E-3	0.3547 E-3
	GAUS	0.4179 E-4	0.1060 E-3	0.3546 E-3
$0.25 f_s$	COS	0.3775 E-4	0.6320 E-4	0.2968 E-3
	UDR	0.3742 E-4	0.7971 E-4	0.2987 E-3
	GAUS	0.3817 E-4	0.8338 E-4	0.2933 E-3
$0.3 f_s$	COS	0.3634 E-4	0.5577 E-4	0.2492 E-3
	UDR	0.3663 E-4	0.6111 E-4	0.2540 E-3
	GAUS	0.3589 E-4	0.6390 E-4	0.2512 E-3
$0.35 f_s$	COS	0.3343 E-4	0.5218 E-4	0.1809 E-3
	UDR	0.3189 E-4	0.4564 E-4	0.1829 E-3
	GAUS	0.3362 E-4	0.4849 E-4	0.1823 E-3
$0.4 f_s$	COS	0.2795 E-4	0.5326 E-4	0.1356 E-3
	UDR	0.2810 E-4	0.4699 E-4	0.1354 E-3
	GAUS	0.2847 E-4	0.4690 E-4	0.1340 E-3
$0.45 f_s$	COS	0.2389 E-4	0.6432 E-4	0.9089 E-4
	UDR	0.2371 E-4	0.5026 E-4	0.9253 E-4
	GAUS	0.2339 E-4	0.5327 E-4	0.9183 E-4

LC LADDER WDF

PASS BAND WIDTH	SIGNAL TYPE	ROUNDING	MAGNITUDE TRUNCATION	2's COMPLEMENT TRUNCATION
$0.05 f_s$	COS	0.7246 E-4	0.3181 E-3	0.9903 E-3
	UDR	0.7554 E-4	0.4237 E-3	0.9895 E-3
	GAUS	0.6335 E-4	0.4143 E-3	0.9922 E-3
$0.1 f_s$	COS	0.6038 E-4	0.2355 E-3	0.7366 E-3
	UDR	0.6054 E-4	0.4039 E-3	0.7351 E-3
	GAUS	0.5788 E-4	0.3534 E-3	0.7418 E-3
$0.15 f_s$	COS	0.5128 E-4	0.2266 E-3	0.6743 E-3
	UDR	0.5524 E-4	0.3547 E-3	0.6814 E-3
	GAUS	0.5331 E-4	0.3514 E-3	0.6727 E-3
$0.2 f_s$	COS	0.4882 E-4	0.2044 E-3	0.5666 E-3
	UDR	0.5326 E-4	0.2861 E-3	0.5649 E-3
	GAUS	0.4956 E-4	0.2960 E-3	0.5686 E-3
$0.25 f_s$	COS	0.4482 E-4	0.1811 E-3	0.4947 E-3
	UDR	0.4619 E-4	0.2730 E-3	0.4972 E-3
	GAUS	0.4460 E-4	0.2637 E-3	0.4950 E-3
$0.3 f_s$	COS	0.4918 E-4	0.1912 E-3	0.4441 E-3
	UDR	0.4646 E-4	0.2752 E-3	0.4426 E-3
	GAUS	0.4563 E-4	0.2598 E-3	0.4420 E-3
$0.35 f_s$	COS	0.3887 E-4	0.1569 E-3	0.3353 E-3
	UDR	0.3934 E-4	0.2012 E-3	0.3316 E-3
	GAUS	0.4014 E-4	0.2256 E-3	0.3332 E-3
$0.4 f_s$	COS	0.3749 E-4	0.1116 E-3	0.2317 E-3
	UDR	0.3658 E-4	0.1669 E-3	0.2316 E-3
	GAUS	0.3466 E-4	0.1586 E-3	0.2308 E-3
$0.45 f_s$	COS	0.2793 E-4	0.9431 E-4	0.1621 E-3
	UDR	0.2906 E-4	0.1176 E-3	0.1601 E-3
	GAUS	0.2817 E-4	0.1170 E-3	0.1622 E-3

APPENDIX B1

A MEASURE OF HARDWARE COMPLEXITY FOR DIGITAL FILTERS

For any digital system, a measure of the hardware complexity can be obtained by the number of two-input gate equivalents making up the system.

Digital filters are made up of adders/subtractors, delays and multipliers. The digital filters considered in this thesis use the 2's complement number system.

TWO INPUT GATE EQUIVALENTS FOR THE BUILDING BLOCKS

Note that for our purposes, the standard circuits will be assumed for the building blocks.

DELAYS :-

A delay is a set of edge triggered D-type flip-flops.

Number of D-types = number of signal bits.

Number of two input gates for M-bit signal = 7.M

ADDERS/SUBTRACTORS :-

Adders comprise cascades of full-adders. Boolean expression for full-adder is as follows ;

$SUM = \overline{A}.B.\overline{C} + A.\overline{B}.\overline{C} + \overline{A}.\overline{B}.C + A.B.C$ (14 two input gates)

$CARRY = A.B + A.C + B.C$ (5 two input gates)

The same circuit (direct implementation of the above expression) can be used as either adder or subtractor.

Number of two input gates for M-bit signal = 19.M

MULTIPLIERS :-

The multiplier model used in measuring hardware complexity in chapter 3 is the simple shift and add type and is illustrated below [9]. Here M is the signal wordlength and N is the number of non-zero coefficient bits.

We will assume that the addition of the values in the columns is carried out using ripple carry adders of the type discussed previously. In this case, the number of full-adder stages required are M.N.

Note that in this model, the extra circuitry required to handle the sign bit in 2's complement arithmetic is ignored. The important point is that the multiplier is assumed to be N times more complex than the adder where N is the number of non-zero coefficient bits.

The number of two input gates required for a multiplier with an M-bit signal and N-bit coefficients is therefore 19.M.N.

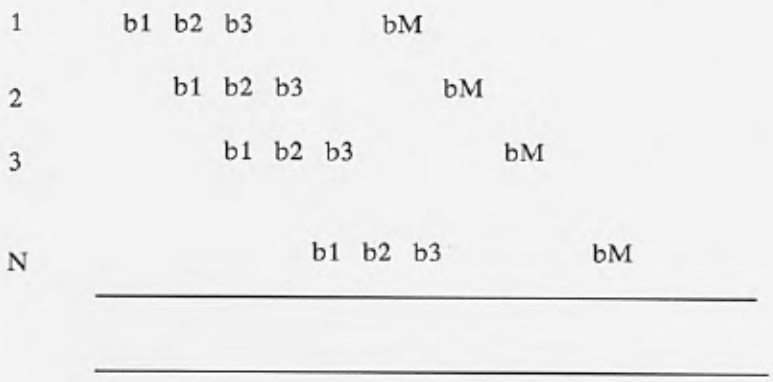


ILLUSTRATION OF SHIFT-ADD MULTIPLIER

APPENDIX C1

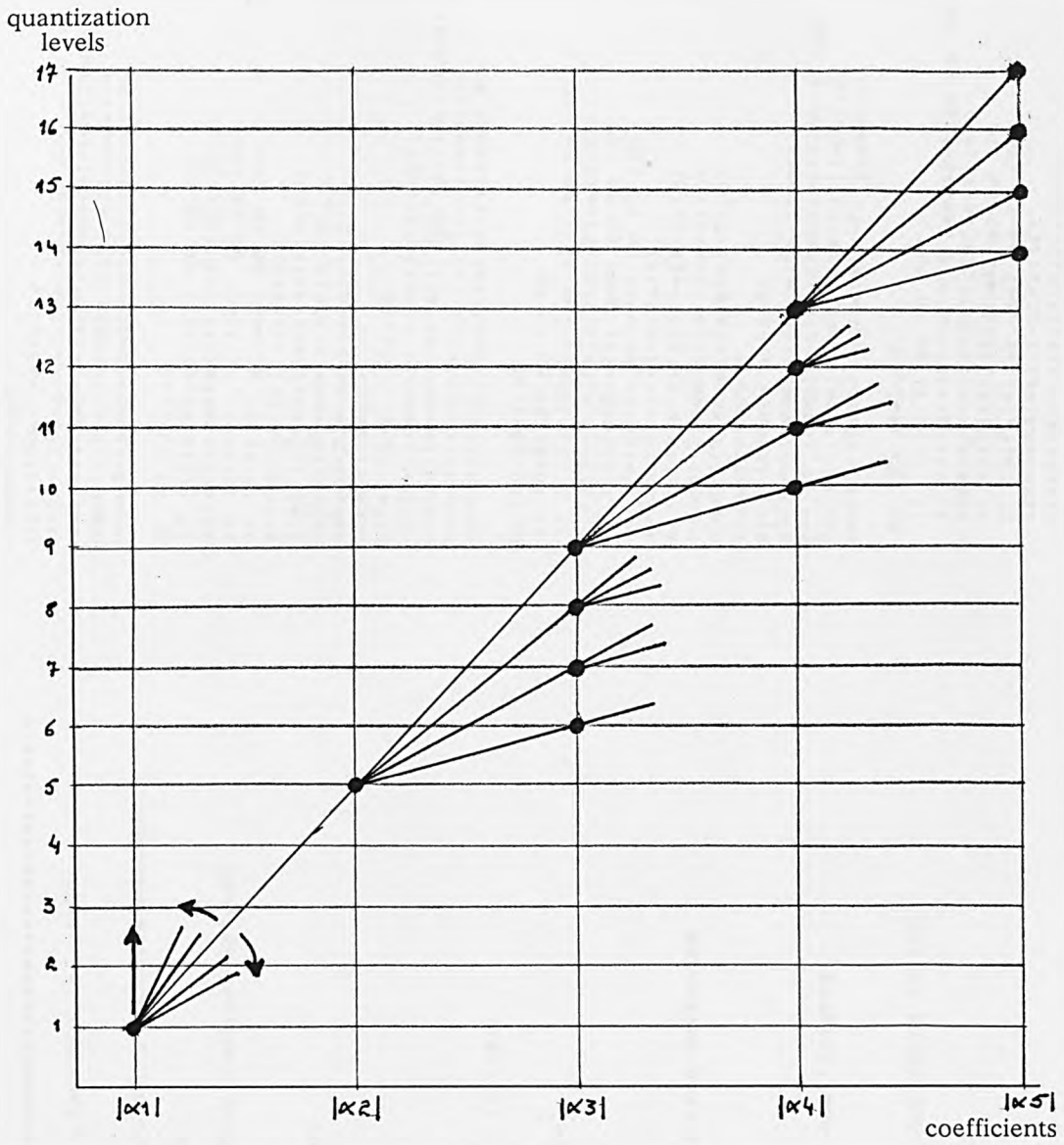


figure C1: generation of coefficients which lie on a convex curve

The method used for generation of coefficient values which lie on a convex curve is illustrated in figure C1. The procedure used is as follows :

(i) select $\alpha 1$ value.

(ii) select $\alpha 2$ value greater than $\alpha 1$ value.

(iii) select $\alpha 3$ value greater than $\alpha 2$ value but less than or equal to $(\alpha 2-\alpha 1)+\alpha 2$.

(iv) select $\alpha 4$ value greater than $\alpha 3$ value but less than or equal to $(\alpha 3-\alpha 2)+\alpha 3$.

(v) select $\alpha 5$ value greater than $\alpha 4$ value but less than or equal to $(\alpha 4-\alpha 3)+\alpha 4$.

The steps (i) to (v) are repeated in sequence for all possible values of $\alpha 1$, $\alpha 2$, $\alpha 3$, $\alpha 4$, $\alpha 5$.

```
      ******************************************************************

      PROGRAM FOR THE FINITE WORDLENGTH DESIGN OF CASCADED UNIT ELEMENTS
      WAVE DIGITAL FILTERS

      ******************************************************************

      real alfa(22),alfinv(22),alalfi(22),loss(50),low1,low3
      real low1,low2,low3,low4,low5
      logical band(50)
      complex dabcd1(22),dabcd3(22),dabcd2,dabcd4,abcd1,abcd2,abcd3
      complex abcd4,eomega(50)
      complex ctemp1,ctemp2,ctemp3,ctemp4
      common pbej,pbdb,sbej,sbdb
      common/one/low1,low2,low3,low4,low5,up1,up2,up3,up4,up5
c
c     input tolerance scheme parameters
c
      call tolsch(eomega,loss,numpts,band)
c
c     estimate required order
c
      pi=3.14159263
      epsilon=(10.0**(pbdb/10.0))
      theta0=pi*pbej
      theta1=pi*sbej
      x1=2.0*alog10((2.0*sin(theta1))/sin(theta0))
      x2=alog10(epsilon-1)
      x3=(10.0**((sbdb)/10.0))-1.0
      ordapx=(alog10(x3)-x2+alog10(4.0))/x1
      n2=int(ordapx+0.5)
      n1=(n2/2)+1
      print*,ordapx,n1
      c1=0.007874
c
c     carry out search for all odd orders from estimated
c     value to 21
c
      do 160 n=n1,11
      nord=(n1**2)-1
c
c     set upper and lower values for the coefficients
c
      call rngset(nord,pbej,sbdb)
c
c     derive values for the coefficients according to the
c     rules given in section 4.3.3
c
      itemp1=(2*n)+1
      itemp2=2*n
      itemp3=(2*n)-1
c
      do 150 j1=1,127
       alfa(1)=-(c1*float(j1))
       if (-alfa(1).lt.low1) go to 150
       if (-alfa(1).gt.up1) go to 160

      alfinv(1)=1/(1-alfa(1))
      alalfi(1)=-alfa(1)*alfinv(1)
      dabcd1(1)=cmplx(alfinv(1),0)
      dabcd3(1)=cmplx(alalfi(1),0)
      itemp=itemp2
      alfa(itemp)=-alfa(1)
      alfinv(itemp)=1/(1-alfa(itemp))
      alalfi(itemp)=-alfa(itemp)*alfinv(itemp)
      dabcd1(itemp)=cmplx(alfinv(itemp),0)
      dabcd3(itemp)=cmplx(alalfi(itemp),0)

      do 140 j2=1,j1
      if ((j1+j2).ge.127) go to 150
      alfa(2)=-alfa(1)+(c1*float(j2))
      if (alfa(2).lt.low2) go to 140
      if (alfa(2).gt.up2) go to 150
      alfinv(2)=1/(1-alfa(2))
      alalfi(2)=-alfa(2)*alfinv(2)
      dabcd1(2)=cmplx(alfinv(2),0)
      dabcd3(2)=cmplx(alalfi(2),0)
      itemp=itemp2-1
      alfa(itemp)=-alfa(2)
      alfinv(itemp)=1/(1-alfa(itemp))
      alalfi(itemp)=-alfa(itemp)*alfinv(itemp)
      dabcd1(itemp)=cmplx(alfinv(itemp),0)
      dabcd3(itemp)=cmplx(alalfi(itemp),0)

      do 130 j3=1,j2
       if (n.lt.3) go to 50
       if (((j1+j2)+j3).ge.127) go to 140
       alfa(3)=-(alfa(2)+(c1*float(j3)))
       if (-alfa(3).lt.low3) go to 130
       if (-alfa(3).gt.up3) go to 140
       alfinv(3)=1/(1-alfa(3))
       alalfi(3)=-alfa(3)*alfinv(3)
       dabcd1(3)=cmplx(alfinv(3),0)
       dabcd3(3)=cmplx(alalfi(3),0)
       itemp=itemp2-2
       alfa(itemp)=-alfa(3)
       alfinv(itemp)=1/(1-alfa(itemp))
       alalfi(itemp)=-alfa(itemp)*alfinv(itemp)
       dabcd1(itemp)=cmplx(alfinv(itemp),0)
       dabcd3(itemp)=cmplx(alalfi(itemp),0)

       do 120 j4=1,j3
        if (n.lt.4) go to 60
        if ((((j1+j2)+j3)+j4).ge.127) go to 130
        alfa(4)=-alfa(3)+(c1*float(j4))
        if (alfa(4).lt.low4) go to 120
        if (alfa(4).gt.up4) go to 130
        alfinv(4)=1/(1-alfa(4))
        alalfi(4)=-alfa(4)*alfinv(4)
        dabcd1(4)=cmplx(alfinv(4),0)
        dabcd3(4)=cmplx(alalfi(4),0)
        itemp=itemp2-3
        alfa(itemp)=-alfa(4)
```

```
      alfinv(itemp)=1/(1-alfa(itemp))
      alalfi(itemp)=-alfa(itemp)*alfinv(itemp)
      dabcd1(itemp)=cmplx(alfinv(itemp),0)
      dabcd3(itemp)=cmplx(alalfi(itemp),0)
c
 60    do 110 j5=1,j4
       if (n.lt.5) go to 75
       if (((((j1+j2)+j3)+j4)+j5).ge.127) go to 120
       alfa(5)=-(alfa(4)+(c1*float(j5)))
       if (-alfa(5).lt.low5) go to 110
       if (-alfa(5).gt.up5) go to 120
       alfinv(5)=1/(1-alfa(5))
       alalfi(5)=-alfa(5)*alfinv(5)
       dabcd1(5)=cmplx(alfinv(5),0)
       dabcd3(5)=cmplx(alalfi(5),0)
       itemp=itemp2-4
       alfa(itemp)=-alfa(5)
       alfinv(itemp)=1/(1-alfa(itemp))
       alalfi(itemp)=-alfa(itemp)*alfinv(itemp)
       dabcd1(itemp)=cmplx(alfinv(itemp),0)
       dabcd3(itemp)=cmplx(alalfi(itemp),0)
       if (n.lt.6) go to 75
c
       temp=1.0
       do 70 lp1=6,n
        alfa(lp1)=temp*(abs(alfa(lp1-1))+c1)
        alfinv(lp1)=1/(1-alfa(lp1))
        alalfi(lp1)=-alfa(lp1)*alfinv(lp1)
        dabcd1(lp1)=cmplx(alfinv(lp1),0)
        dabcd3(lp1)=cmplx(alalfi(lp1),0)
        itemp=itemp1-lp1
        alfa(itemp)=-alfa(lp1)
        alfinv(itemp)=1/(1-alfa(itemp))
        alalfi(itemp)=-alfa(itemp)*alfinv(itemp)
        dabcd1(itemp)=cmplx(alfinv(itemp),0)
        dabcd3(itemp)=cmplx(alalfi(itemp),0)
 70     temp=-temp
c
c      evaluate frequency response
c
 75    do 90 jptr=1,numpts
        abcd1=(1,0)
        abcd2=(0,0)
        abcd3=(0,0)
        abcd4=(1,0)
c
        do 80 lp2=1,itemp3
         dabcd2=alalfi(lp2)*eomega(jptr)
         dabcd4=alfinv(lp2)*eomega(jptr)
         ctemp1=abcd1*dabcd1(lp2)+abcd2*dabcd3(lp2)
         ctemp2=abcd1*dabcd2+abcd2*dabcd4
         ctemp3=abcd3*dabcd1(lp2)+abcd4*dabcd3(lp2)
         ctemp4=abcd3*dabcd2+abcd4*dabcd4
         abcd1=ctemp1
         abcd2=ctemp2
         abcd3=ctemp3
 80      abcd4=ctemp4
c
         ctemp1=abcd1*dabcd1(itemp2)+abcd2*dabcd3(itemp2)
         aloss=sqrt((real(ctemp1)**2)+(aimag(ctemp1)**2))
         if ((aloss.le.loss(jptr)).and.(band(jptr))) go to 90
        if ((aloss.ge.loss(jptr)).and.(.not.band(jptr))) go to 90
        go to 110
 90     continue
c
c       if frequency response fits tolerance scheme print
c       coefficient values
c
        do 100 lp3=1,itemp2
 100     print*,alfa(lp3)
         read*,x
c
 110    if (n.lt.5) go to 120
 120   if (n.lt.4) go to 130
 130  if (n.lt.3) go to 140
 140  continue
 150  continue
 160  continue
c
      stop
      end

*************************************************************************

   Subroutine to input tolerance scheme parameters.
   This subroutine also creates a logical array 'band' which is used
   to make the search more efficient.

*************************************************************************

      subroutine tolsch(eomega,loss,iptr,band)
c
      real freq(50),loss(50)
      logical band(50)
      complex eomega(50)
c
      common pbej,pbdb,sbej,sbdb
c
      print*,'input,pbej,pbloss(db),sbej,sbloss(db)'
      read*,pbej,pbdb,sbej,sbdb
c
      pbloss=10.0**(pbdb/20.0)
      sbloss=10.0**(sbdb/20.0)
      pi=3.14159263
      freq1=pbej
      freq2=sbej
      freq3=0.0
      freq4=0.5
      iptr=0
c
      do 50 lp1=1,50
       if (freq3.gt.freq1) go to 10
       iptr=iptr+1
```

```
      freq(iptr)=freq1
      loss(iptr)=pbloss
      band(iptr)=.true.
10    if (freq2.gt.freq4) go to 20
      iptr=iptr+1
      freq(iptr)=freq2
      loss(iptr)=sbloss
      band(iptr)=.false.
20    if (freq3.ge.freq1) go to 30
      iptr=iptr+1
      freq(iptr)=freq3
      loss(iptr)=pbloss
      band(iptr)=.true.
30    if (freq2.ge.freq4) go to 40
      iptr=iptr+1
      freq(iptr)=freq4
      loss(iptr)=sbloss
      band(iptr)=.false.
40    freq1=freq1-0.01
      freq2=freq2+0.01
      freq3=freq3+0.01
      freq4=freq4-0.01
50    continue
c
      do 60 lp2=1,iptr
       w=2.0*pi*freq(lp2)
       eomega(lp2)=cmplx(cos(w),-sin(w))
60    continue
c
      return
      end

**************************************************************************

     Subroutine to derive lower and upper values of coefficients
     from Levy's tables

**************************************************************************

      subroutine rngset(nord,bw,sbgain)
c
      real bw5al(7,3),bw7al(7,4),bw9al(7,5),bw11al(7,6),bw13al(7,7)
      real bw15al(7,8),bw17al(7,9),bw19al(7,10),bw21al(7,11)
      real sb5al(20,3),sb7al(20,4),sb9al(20,5),sb11al(20,6),sb13al(20,7)
      real sb15al(20,8),sb17al(20,9),sb19al(20,10),sb21al(20,11)
      real low1,low2,low3,low4,low5
c
      common/one/low1,low2,low3,low4,low5,up1,up2,up3,up4,up5
c
      open(1,file='range',status='old')
      rewind(1)
      read(1,*)((sb5al(i,j),j=1,3),i=1,12)
      call sortsb(sb5al,3,12)
      read(1,*)((bw5al(i,j),j=1,3),i=1,7)
      call sortbw(bw5al,3)
      read(1,*)((sb7al(i,j),j=1,4),i=1,20)
      call sortsb(sb7al,4,20)
      read(1,*)((bw7al(i,j),j=1,4),i=1,7)
      call sortbw(bw7al,4)
      read(1,*)((sb9al(i,j),j=1,5),i=1,20)
      call sortsb(sb9al,5,20)
      read(1,*)((bw9al(i,j),j=1,5),i=1,7)
      call sortbw(bw9al,5)
      read(1,*)((sb11al(i,j),j=1,6),i=1,20)
      call sortsb(sb11al,6,20)
      read(1,*)((bw11al(i,j),j=1,6),i=1,7)
      call sortbw(bw11al,6)
      read(1,*)((sb13al(i,j),j=1,7),i=1,20)
      call sortsb(sb13al,7,20)
      read(1,*)((bw13al(i,j),j=1,7),i=1,7)
      call sortbw(bw13al,7)
      read(1,*)((sb15al(i,j),j=1,8),i=1,20)
      call sortsb(sb15al,8,20)
      read(1,*)((bw15al(i,j),j=1,8),i=1,7)
      call sortbw(bw15al,8)
      read(1,*)((sb17al(i,j),j=1,9),i=1,20)
      call sortsb(sb17al,9,20)
      read(1,*)((bw17al(i,j),j=1,9),i=1,7)
      call sortbw(bw17al,9)
      read(1,*)((sb19al(i,j),j=1,10),i=1,20)
      call sortsb(sb19al,10,20)
      read(1,*)((bw19al(i,j),j=1,10),i=1,7)
      call sortbw(bw19al,10)
      read(1,*)((sb21al(i,j),j=1,11),i=1,20)
      call sortsb(sb21al,11,20)
      read(1,*)((bw21al(i,j),j=1,11),i=1,7)
      call sortbw(bw21al,11)
c
c
      isb=int(sbgain/10.0)
      ibw=int(bw/0.05)
      if (isb.eq.0) isb=1
      if (ibw.eq.0) ibw=1
c
      if (nord.ne.5) go to 10
      low1=sb5al(isb,1)
      low2=sb5al(isb,2)
      low3=sb5al(isb,3)
      up1=bw5al(ibw,1)
      up2=bw5al(ibw,2)
      up3=bw5al(ibw,3)
10    if (nord.ne.7) go to 20
      low1=sb7al(isb,1)
      low2=sb7al(isb,2)
      low3=sb7al(isb,3)
      low4=sb7al(isb,4)
      up1=bw7al(ibw,1)
      up2=bw7al(ibw,2)
      up3=bw7al(ibw,3)
      up4=bw7al(ibw,4)
20    if (nord.ne.9) go to 30
      low1=sb9al(isb,1)
```

```
      low2=sb9al(isb,2)
      low3=sb9al(isb,3)
      low4=sb9al(isb,4)
      low5=sb9al(isb,5)
      up1=bw9al(ibw,1)
      up2=bw9al(ibw,2)
      up3=bw9al(ibw,3)
      up4=bw9al(ibw,4)
      up5=bw9al(ibw,5)
30    if (nord.ne.11) go to 40
      low1=sb11al(isb,1)
      low2=sb11al(isb,2)
      low3=sb11al(isb,3)
      low4=sb11al(isb,4)
      low5=sb11al(isb,5)
      up1=bw11al(ibw,1)
      up2=bw11al(ibw,2)
      up3=bw11al(ibw,3)
      up4=bw11al(ibw,4)
      up5=bw11al(ibw,5)
40    if (nord.ne.13) go to 50
      low1=sb13al(isb,1)
      low2=sb13al(isb,2)
      low3=sb13al(isb,3)
      low4=sb13al(isb,4)
      low5=sb13al(isb,5)
      up1=bw13al(ibw,1)
      up2=bw13al(ibw,2)
      up3=bw13al(ibw,3)
      up4=bw13al(ibw,4)
      up5=bw13al(ibw,5)
50    if (nord.ne.15) go to 60
      low1=sb15al(isb,1)
      low2=sb15al(isb,2)
      low3=sb15al(isb,3)
      low4=sb15al(isb,4)
      low5=sb15al(isb,5)
      up1=bw15al(ibw,1)
      up2=bw15al(ibw,2)
      up3=bw15al(ibw,3)
      up4=bw15al(ibw,4)
      up5=bw15al(ibw,5)
60    if (nord.ne.17) go to 70
      low1=sb17al(isb,1)
      low2=sb17al(isb,2)
      low3=sb17al(isb,3)
      low4=sb17al(isb,4)
      low5=sb17al(isb,5)
      up1=bw17al(ibw,1)
      up2=bw17al(ibw,2)
      up3=bw17al(ibw,3)
      up4=bw17al(ibw,4)
      up5=bw17al(ibw,5)
70    if (nord.ne.19) go to 80
      low1=sb19al(isb,1)
      low2=sb19al(isb,2)
      low3=sb19al(isb,3)
      low4=sb19al(isb,4)
      low5=sb19al(isb,5)
      up1=bw19al(ibw,1)
      up2=bw19al(ibw,2)
      up3=bw19al(ibw,3)
      up4=bw19al(ibw,4)
      up5=bw19al(ibw,5)
80    if (nord.ne.21) go to 90
      low1=sb21al(isb,1)
      low2=sb21al(isb,2)
      low3=sb21al(isb,3)
      low4=sb21al(isb,4)
      low5=sb21al(isb,5)
      up1=bw21al(ibw,1)
      up2=bw21al(ibw,2)
      up3=bw21al(ibw,3)
      up4=bw21al(ibw,4)
      up5=bw21al(ibw,5)
90    continue
c
      return
      end


      subroutine sortbw(data,ncol)
      real data(7,11)
c
      i=1
      do 10 j=1,ncol
10      if (data(i,j).eq.1.0) data(i,j)=1.0
      do 20 j=1,ncol
         do 20 i=2,7
            if (data(i,j).eq.0.0) data(i,j)=data(i-1,j)
20    continue
c
      do 30 j=1,ncol
         do 30 i=1,6
            do 30 lp1=i+1,7
               data(i,j)=amax1(data(i,j),data(lp1,j))
30    continue
c
      return
      end


      subroutine sortsb(data,ncol,nrow)
      real data(20,11)
c
      i=1
      do 10 j=1,ncol
10       if (data(i,j).eq.1.0) data(i,j)=0.0
      do 20 j=1,ncol
         do 20 i=2,nrow
            if (data(i,j).eq.1.0) data(i,j)=data(i-1,j)
20    continue
```

```
c
      do 30 j=1,ncol
         do 30 i=1,19
            do 30 lp1=i+1,nrow
               data(i,j)=amin1(data(i,j),data(lp1,j))
 30   continue
c
      return
      end
```

```
**************************************

Example of data held in file 'range'

**************************************

fifth order

alfa1  alfa2  alfa3   stopband loss

0.089  0.305  0.446   10 dB
0.192  0.521  0.597   20 dB
0.270  0.658  0.743   30 dB
0.313  0.749  0.841   40 dB
0.463  0.836  0.890   50 dB
0.523  0.892  0.930   60 dB
0.601  0.936  0.961   70 dB
0.681  0.962  0.979   80 dB
0.708  0.972  0.984   90 dB
0.765  0.984  0.993   100 dB
0.794  0.988  0.995   110 dB
0.851  0.994  0.996   120 dB

                      passband width

0.833  0.977  0.990   0.05 Fs
0.693  0.909  0.937   0.10 Fs
0.576  0.804  0.856   0.15 Fs
0.478  0.671  0.743   0.20 Fs
0.396  0.521  0.597   0.25 Fs
0.007  0.012  0.017   0.30 Fs
0.004  0.005  0.006   0.35 Fs
```

```
seventh order

0.060  0.199  0.252  0.275
0.087  0.297  0.430  0.470
0.123  0.395  0.554  0.585
0.168  0.504  0.673  0.706
0.227  0.620  0.761  0.780
0.307  0.735  0.817  0.831
0.359  0.798  0.866  0.876
0.413  0.840  0.907  0.914
0.490  0.880  0.936  0.942
0.505  0.906  0.959  0.964
0.622  0.945  0.967  0.969
0.674  0.956  0.980  0.983
0.790  0.974  0.984  0.985
0.711  0.975  0.991  0.994
0.736  0.980  0.993  0.994
0.790  0.988  0.995  0.996
1.000  1.000  1.000  1.000
1.000  1.000  1.000  1.000
1.000  1.000  1.000  1.000
1.000  1.000  1.000  1.000


0.836  0.978  0.985  0.986
0.698  0.913  0.940  0.945
0.583  0.812  0.866  0.876
0.486  0.685  0.761  0.780
0.405  0.543  0.629  0.657
0.336  0.395  0.474  0.506
0.277  0.252  0.304  0.329
```

APPENDIX C2

```
      ****************************************************************

      Program for functional level simulation of UEWDF architecture

      ****************************************************************

      INTEGER TIMSTP
      LOGICAL CLK1,CLK2,CLK3,CLK4,CLK3P,CLK4P,FRAME,SIGRDY,START,HFRAM1
      LOGICAL HFRAM2,CONT1,CONT2,M11,M12,M21,M22,M23,CLK5
C
      COMMON STACK1(8),STACK2(9),SAVE1(2),SAVE2(2),PIPE(2),COEFF(10),
     &     FRAME(23),HFRAM1(12),CONT1(3),CONT2(3),SIGVAL,SIGRDY,OUTPUT,
     &     CLK1,CLK2,CLK3,CLK4,CLK3P,CLK4P,START,M11,M12,M21,M22,
     &     TIMSTP,COUNT
      CALL INIT
      DO 10 TIMSTP=1,70000
         CALL SIGNAL
         IF (CLK1.AND.(.NOT.FRAME(1))) START=SIGRDY
         IF ((.NOT.START).AND.(.NOT.CLK1)) CALL PAR
         IF (START.AND.(.NOT.CLK1)) CALL FRM
         HFRAM2=(.NOT.HFRAM1(1)).AND.FRAME(1)
         CLK2=FRAME(1).AND.CLK1
         CLK3=HFRAM1(1).AND.CLK1
         CLK4=HFRAM2.AND.CLK1
C
         IF (CLK3P.AND.(.NOT.CLK3)) CALL MCON1
         IF (CLK4P.AND.(.NOT.CLK4)) CALL MCON2
C
         M11=HFRAM1(1).AND.CONT1(3)
         M12=HFRAM2.AND.CONT2(3)
         M21=HFRAM1(1).AND.(.NOT.HFRAM1(2))
         M22=FRAME(1).AND.(.NOT.FRAME(2))
         M23=M21.OR.M22
         CLK5=CLK2.AND.(.NOT.M23)
C
         IF (CLK5) CALL DATPTH
C
         IF (.NOT.(M23.AND.CLK1)) GO TO 9
         DO 5 LP1=1,7
            STACK1(LP1)=STACK1(LP1+1)
 5       CONTINUE
         STACK1(8)=0.0
 9       CONTINUE
C
         CLK1=.NOT.CLK1
         CLK3P=CLK3
         CLK4P=CLK4
      IF (M22.AND.CLK1) PRINT,STACK2(9)
 10   CONTINUE
C
      STOP
      END
C


      SUBROUTINE INIT
C
      LOGICAL CLK3P,CLK4P,CLK1,SIGRDY,START
C
      COMMON STACK1(8),STACK2(9),SAVE1(2),SAVE2(2),PIPE(2),COEFF(10),
     &     FRAME(23),HFRAM1(12),CONT1(3),CONT2(3),SIGVAL,SIGRDY,OUTPUT,
     &     CLK1,CLK2,CLK3,CLK4,CLK3P,CLK4P,START,M11,M12,M21,M22,
     &     TIMSTP,COUNT
      DO 10 LP1=1,8
         STACK1(LP1)=0.0
         STACK2(LP1)=0.0
 10   CONTINUE
      STACK1(9)=0.0
C
      DO 20 LP2=1,2
         SAVE1(LP2)=0.0
         SAVE2(LP2)=0.0
         PIPE(LP2)=0.0
 20   CONTINUE
C
      SIGRDY=.FALSE.
      START=.FALSE.
      CLK3P=.FALSE.
      CLK4P=.FALSE.
      CLK1=.FALSE.
C
      COEFF(1)=0.0
      COEFF(10)=0.0
      COEFF(9)=0.83553984
      COEFF(2)=-0.83553984
      COEFF(3)=0.97769094
      COEFF(4)=-0.98514007
      COEFF(5)=0.98606601
      COEFF(6)=-0.98606601
      COEFF(7)=0.98514007
      COEFF(8)=-0.97769094
C
      RETURN
      END
C


      SUBROUTINE SIGNAL
C
      INTEGER TIMSTP,COUNT
      LOGICAL SIGRDY
C
      COMMON STACK1(8),STACK2(9),SAVE1(2),SAVE2(2),PIPE(2),COEFF(10),
     &     FRAME(23),HFRAM1(12),CONT1(3),CONT2(3),SIGVAL,SIGRDY,OUTPUT,
     &     CLK1,CLK2,CLK3,CLK4,CLK3P,CLK4P,START,M11,M12,M21,M22,
     &     TIMSTP,COUNT
      SIGVAL=SIN(FLOAT(TIMSTP)/1500.0)
      IF (TIMSTP.LT.10) COUNT=0
      IF (TIMSTP.LT.10) RETURN
      IF (COUNT.EQ.30) GO TO 10
      COUNT=COUNT+1
C     SIGVAL=1.0
```

```
C     IF (TIMSTP.LT.60) SIGVAL=1.0
      RETURN
 10   COUNT=0
      SIGRDY=.NOT.SIGRDY
C     SIGVAL=1.0
C     IF (TIMSTP.LT.60) SIGVAL=1.0
      IF (.NOT.SIGRDY) RETURN
      RETURN
      END
C


      SUBROUTINE PAR
C
      LOGICAL FRAME,HFRAM1,CONT1,CONT2
C
      COMMON STACK1(8),STACK2(9),SAVE1(2),SAVE2(2),PIPE(2),COEFF(10),
     &     FRAME(23),HFRAM1(12),CONT1(3),CONT2(3),SIGVAL,SIGRDY,OUTPUT,
     &     CLK1,CLK2,CLK3,CLK4,CLK3P,CLK4P,START,M11,M12,M21,M22,
     &     TIMSTP,COUNT
      DO 10 LP1=2,23
         FRAME(LP1)=.TRUE.
 10   CONTINUE
      FRAME(1)=.FALSE.
      DO 20 LP2=2,12
         HFRAM1(LP2)=.TRUE.
 20   CONTINUE
      HFRAM1(1)=.FALSE.
C
      CONT1(1)=.FALSE.
      CONT1(2)=.FALSE.
      CONT1(3)=.TRUE.
      CONT2(1)=.FALSE.
      CONT2(2)=.FALSE.
      CONT2(3)=.TRUE.
C
      RETURN
      END
C


      SUBROUTINE FRM
C
      LOGICAL FRAME,HFRAM1
C
      COMMON STACK1(8),STACK2(9),SAVE1(2),SAVE2(2),PIPE(2),COEFF(10),
     &     FRAME(23),HFRAM1(12),CONT1(3),CONT2(3),SIGVAL,SIGRDY,OUTPUT,
     &     CLK1,CLK2,CLK3,CLK4,CLK3P,CLK4P,START,M11,M12,M21,M22,
     &     TIMSTP,COUNT
      DO 10 LP1=1,22
         FRAME(LP1)=FRAME(LP1+1)
 10   CONTINUE
      FRAME(23)=.FALSE.
C
      DO 20 LP2=1,11
         HFRAM1(LP2)=HFRAM1(LP2+1)
 20   CONTINUE
      HFRAM1(12)=.FALSE.
C
      RETURN
      END
C


      SUBROUTINE MCON1
C
      LOGICAL CONT1
C
      COMMON STACK1(8),STACK2(9),SAVE1(2),SAVE2(2),PIPE(2),COEFF(10),
     &     FRAME(23),HFRAM1(12),CONT1(3),CONT2(3),SIGVAL,SIGRDY,OUTPUT,
     &     CLK1,CLK2,CLK3,CLK4,CLK3P,CLK4P,START,M11,M12,M21,M22,
     &     TIMSTP,COUNT
      DO 10 LP1=1,2
         CONT1(LP1)=CONT1(LP1+1)
 10   CONTINUE
      CONT1(3)=.FALSE.
      RETURN
      END
C


      SUBROUTINE MCON2
C
      LOGICAL CONT2
C
      COMMON STACK1(8),STACK2(9),SAVE1(2),SAVE2(2),PIPE(2),COEFF(10),
     &     FRAME(23),HFRAM1(12),CONT1(3),CONT2(3),SIGVAL,SIGRDY,OUTPUT,
     &     CLK1,CLK2,CLK3,CLK4,CLK3P,CLK4P,START,M11,M12,M21,M22,
     &     TIMSTP,COUNT
      DO 10 LP1=1,2
         CONT2(LP1)=CONT2(LP1+1)
 10   CONTINUE
      CONT2(3)=.FALSE.
      RETURN
      END
C


      SUBROUTINE DATPTH
C
      LOGICAL M11,M12,M21,M22,CONT2
C
      COMMON STACK1(8),STACK2(9),SAVE1(2),SAVE2(2),PIPE(2),COEFF(10),
     &     FRAME(23),HFRAM1(12),CONT1(3),CONT2(3),SIGVAL,SIGRDY,OUTPUT,
     &     CLK1,CLK2,CLK3,CLK4,CLK3P,CLK4P,START,M11,M12,M21,M22,
     &     TIMSTP,COUNT
      TEMP1=STACK1(1)
      DO 10 LP1=1,7
         STACK1(LP1)=STACK1(LP1+1)
 10   CONTINUE
      STACK1(8)=PIPE(2)+SAVE1(2)
      SAVE1(2)=SAVE1(1)
```

```
C
      TEMP2=STACK2(1)
      DO 20 LP2=1,8
         STACK2(LP2)=STACK2(LP2+1)
 20   CONTINUE
      STACK2(9)=PIPE(2)+SAVE2(2)
      SAVE2(2)=SAVE2(1)
C
      PIPE(2)=PIPE(1)*COEFF(1)
      TEMP3=COEFF(1)
      DO 30 LP3=1,9
         COEFF(LP3)=COEFF(LP3+1)
 30   CONTINUE
      COEFF(10)=TEMP3
C
      IF (M11) TEMP2=SIGVAL
      IF (M12) TEMP2=0.0
      IF (M21) TEMP1=0.0
      IF (M22) TEMP1=0.0
      SAVE1(1)=TEMP1
      SAVE2(1)=TEMP2
      PIPE(1)=TEMP1-TEMP2
C
C
      L=L+1
      RETURN
      END
```

APPENDIX D1

```
**********************************************************************

    PROGRAM FOR THE DESIGN OF LATTICE WDFs BASED ON
    CASCADED UNIT ELEMENTS

**********************************************************************

      integer ordna,ordda,ordnb,orddb
      double precision na,da,nb,db
c
      common/four/na(33),da(33),nb(33),db(33),ordna,ordda,ordnb,orddb
c
      print*,'1: butterworth'
      print*,'2: chebychev'
      read*,menu1
      call tolsch
      if (menu1.eq.1) call butwth
      if (menu1.eq.2) call chbchv
c
      call reacfn
      call uecas(na,da,ordna,ordda)
      call dfcoef
      call uecas(nb,db,ordnb,orddb)
      call dfcoef
      stop
      end
c     -
c**********************************************************************
c
c     subroutine to evaluate transfer function using Butterworth
c     approximation
c
      subroutine butwth
c
      integer order,ptr,ordfs,ordps
      real temp1,temp2,k1,k,temp3,temp4,apbej,asbej,pbloss,sbloss
      double precision limit1,limit2,pi
      double precision seg,angle,polerl,poleim
      complex poles,fs,ps
c
      common/two/polerl(32),poleim(32),poles(32),fs(33),ps(33)
      common/two1/ordfs,ordps,order
      common/three/apbej,pbloss,asbej,sbloss
c
      pi=3.141592658939d0
      limit1=pi/2.0d0
      limit2=(3.0d0*pi)/2.0d0
c
c
c     initialise
c
      do 10 lp1=1,32
         poles(lp1)=(0,0)
         polerl(lp1)=0.0d0
         poleim(lp1)=0.0d0
 10   continue
      do 20 lp2=1,33
         fs(lp2)=(0,0)
         ps(lp2)=(0,0)
 20   continue
c
c     obtain required order - must be odd
c
      temp1=(10**(pbloss/10))-1
      temp2=(10**(sbloss/10))-1
      k1=sqrt(temp1/temp2)
      k=apbej/asbej
      temp3=alog10(k1)/alog10(k)
      order=int(temp3+1.0)
      temp4=float(int(float(order)/2.0))
      if ((float(order)/2.0).eq.temp4) order=order+1
c
      print*,'lowest odd order.........',order
      print*,'input required order'
      read*,order
c
c     obtain poles and store in double precision arrays polerl poleim
c     and in complex array poles
c
      ptr=0
      seg=pi/(dble(float(order)))
      angle=0.0d0
 30   angle=angle+seg
      if ((angle.le.limit1).or.(angle.ge.limit2)) go to 40
      ptr=ptr+1
      polerl(ptr)=cos(angle)
      poleim(ptr)=sin(angle)
      poles(ptr)=cmplx(sngl(polerl(ptr)),sngl(poleim(ptr)))
 40   if (angle.lt.((3.0d0*pi)/2.0d0)) go to 30
c
c     obtain polynomials fs,ps and ordfs, ordps
c
      ps(1)=(1,0)
      fs(order+1)=(1,0)
      ordps=0
      ordfs=order
c
c     denormalise poles - (normalised w.r.t. apbej)
c
      do 50 lp3=1,order
         poles(lp3)=poles(lp3)*cmplx(apbej,0)
         polerl(lp3)=polerl(lp3)*dble(apbej)
         poleim(lp3)=poleim(lp3)*dble(apbej)
 50   continue
c
c     denormalise polynomials fs and ps - (normalised w.r.t. apbej)
c
      do 60 lp4=1,ordfs+1
         fs(lp4)=fs(lp4)/(cmplx(apbej,0)**(lp4-1))
 60   continue
      do 70 lp5=1,ordps+1
         ps(lp5)=ps(lp5)/(cmplx(apbej,0)**(lp5-1))
```

```
 70   continue
c
      return
      end
c
c*****************************************************************************
c
c     subroutine to evaluate transfer function using Chebyshev
c     approximation
c
      subroutine chbchv
c
      integer order,polptr,k,ordnew
      integer ordfs,ordps
      integer temp6a
      real temp1,temp2,k1,k2,pbloss,sbloss,apbej,asbej,temp3,temp4,temp7
      double precision kp,temp5,w,temp6,u,sigma,omega,polerl
      double precision poleim,pi
      complex poles,fs,ps,fscur(33),fsprev(33)
c
      common/two/polerl(32),poleim(32),poles(32),fs(33),ps(33)
      common/two1/ordfs,ordps,order
      common/three/apbej,pbloss,asbej,sbloss
c
      pi=3.141592658939d0
c
c     initialise
c
      do 10 lp1=1,33
         poles(lp1)=(0,0)
         fs(lp1)=(0,0)
         ps(lp1)=(0,0)
         fscur(lp1)=(0,0)
         fsprev(lp1)=(0,0)
 10   continue
      do 20 lp2=1,32
         polerl(lp2)=0.0d0
         poleim(lp2)=0.0d0
 20   continue
      fsprev(1)=(1,0)
      fscur(2)=(1,0)
c
c     obtain required order - must be odd
c
      temp1=(10**(pbloss/10))-1
      temp2=(10**(sbloss/10))-1
      k1=sqrt(temp1/temp2)
      k2=apbej/asbej
      tempk1=1/k1
      tempk2=1/k2
      tempk3=s11acf(dble(tempk1),ifail)
      tempk4=s11acf(dble(tempk2),ifail)
      temp3=tempk3/tempk4
      order=int(temp3+1.0)
      temp4=float(int(float(order)/2.0))
      if ((float(order)/2.0).eq.temp4) order=order+1
c
      print*,'lowest odd order.......',order
      print*,'input required order'
      read*,order
c
c     obtain poles and store in double precision arrays polerl, poleim
c     and in complex array poles
c
      polptr=1
      kp=dble(sqrt((10**(pbloss/10))-1))
      temp5=dble(float(order))
      temp7=s11abf((1.0d0/kp),ifail)
      w=(1.0d0/temp5)*dble(temp7)
      do 60 k=1,order
         temp6=dble(float((2*k)-1)/float(order))
         u=-(temp6*(pi/2.0d0))
         sigma=sin(u)*sinh(w)
         omega=cos(u)*cosh(w)
         polerl(polptr)=sigma
         poleim(polptr)=omega
         polptr=polptr+1
 60   continue
c
      do 70 polptr=1,order
         poles(polptr)=cmplx(sngl(polerl(polptr)),sngl(poleim(polptr)))
 70   continue
c
c     obtain polynomials fs ps and their orders ordfs ordps
c
      do 100 ordnew=2,order
         fs(1)=-fsprev(1)
         do 80 lp1=2,ordnew
            fs(lp1)=((2.0,0)*fscur(lp1-1))-fsprev(lp1)
 80      continue
         fs(ordnew+1)=(2.0,0)*fscur(ordnew)
         do 90 lp2=1,ordnew+1
            fsprev(lp2)=fscur(lp2)
            fscur(lp2)=fs(lp2)
 90      continue
 100  continue
c
      ps(1)=(1,0)
      ordps=0
      ordfs=order
c
c     denormalise poles - (normalised w.r.t apbej)
c
      do 110 lp3=1,order
         poles(lp3)=poles(lp3)*cmplx(apbej,0)
         polerl(lp3)=polerl(lp3)*dble(apbej)
         poleim(lp3)=poleim(lp3)*dble(apbej)
 110  continue
c
c     denormalise polynomials ps and fs - (normalised w.r.t. apbej)
c
      do 120 lp4=1,ordfs+1
```

```
         fs(lp4)=fs(lp4)/(cmplx(apbej,0)**(lp4-1))
 120   continue
c
       do 130 lp5=1,ordps+1
          ps(lp5)=ps(lp5)/(cmplx(apbej,0)**(lp5-1))
 130   continue
c
c
c      ensure that all terms in fs are positive and multiply fs by kp
c
       do 140 lp6=1,ordfs+1
          if (real(fs(lp6)).lt.0) fs(lp6)=-fs(lp6)
          fs(lp6)=fs(lp6)*cmplx((sngl(kp)),0)
 140   continue
c
       return
       end
c
c*************************************************************************
c
       subroutine cmxmul(rl1,im1,rl2,im2,rl3,im3)
c
c      subroutine for complex multiplication with double precision
c
       double precision rl1,im1,rl2,im2,rl3,im3
c
c      rl3 + jim3=(rl1 + jim1)*(rl2 + jim2)
c
       rl3=(rl1*rl2)-(im1*im2)
       im3=(rl1*im2)+(im1*rl2)
       return
       end
c
c*************************************************************************
c
c      subroutine to evaluate digital filter coefficients
c
       subroutine dfcoef
c
       integer ordue
       double precision multip(32),unel(32)
       character*3 ellast
c
       common/one/unel(32),ordue,ellast
c
       multip(1)=(1.0d0-unel(1))/(1.0d0+unel(1))
       print*,multip(1)
       do 10 lp1=2,ordue
          multip(lp1)=(unel(lp1-1)-unel(lp1))/(unel(lp1-1)+unel(lp1))
          print*,sngl(multip(lp1))
 10    continue
       print*,ellast
c
       return
       end
c
c*************************************************************************
c
       subroutine mltout(rootrl,rootim,rsltrl,rsltim,numrts)
c
       integer row1,row2,row3,row4,row5,col2,col3,col4,col5,col6
       integer ptr1,ptr2,numrts
       double precision onerl(32,2),oneim(32,2),tworl(16,3),twoim(16,3
       double precision threrl(8,5),threim(8,5),fourrl(4,9),fourim(4,9
       double precision fiverl(2,17),fiveim(2,17),sixrl(33),sixim(33).
       double precision rootrl(32),rootim(32),rl1,im1,rl2,im2,rl3,im3
       double precision rsltrl(32),rsltim(32)
c
c      initialise arrays one - note second column = 1
c
       do 10 ptr1=1,32
          onerl(ptr1,1)=0.0d0
          onerl(ptr1,2)=1.0d0
          oneim(ptr1,1)=0.0d0
          oneim(ptr1,2)=0.0d0
 10    continue
c
c      initialise arrays
c
       do 40 ptr1=1,16
          do 30 ptr2=1,3
             tworl(ptr1,ptr2)=0.0d0
             twoim(ptr1,ptr2)=0.0d0
 30       continue
 40    continue
c
       do 60 ptr1=1,8
          do 50 ptr2=1,5
             threrl(ptr1,ptr2)=0.0d0
             threim(ptr1,ptr2)=0.0d0
 50       continue
 60    continue
c
       do 80 ptr1=1,4
          do 70 ptr2=1,9
             fourrl(ptr1,ptr2)=0.0d0
             fourim(ptr1,ptr2)=0.0d0
 70       continue
 80    continue
c
       do 100 ptr1=1,2
          do 90 ptr2=1,17
             fiverl(ptr1,ptr2)=0.0d0
             fiveim(ptr1,ptr2)=0.0d0
 90       continue
 100   continue
c
       do 110 ptr1=1,33
          sixrl(ptr1)=0.0d0
          sixim(ptr1)=0.0d0
 110   continue
c
```

```
c     load arrays root into arrays one. load arrays one first column with 1.0
c     note that roots are negated before loading
c
      do 115 lp1=1,numrts
         oner1(lp1,1)=1.0d0
         oner1(lp1,2)=-rootr1(lp1)
         oneim(lp1,2)=-rootim(lp1)
 115  continue
c
      do 140 row1=1,31,2
         row2=(row1+1)/2
         do 130 ptr1=1,2
            do 120 ptr2=1,2
               col2=ptr1+ptr2-1
               r11=oner1(row1,ptr1)
               im1=oneim(row1,ptr1)
               r12=oner1(row1+1,ptr2)
               im2=oneim(row1+1,ptr2)
               call cmxmul(r11,im1,r12,im2,r13,im3)
               twor1(row2,col2)=twor1(row2,col2)+r13
               twoim(row2,col2)=twoim(row2,col2)+im3
 120        continue
 130     continue
 140  continue
c
      do 170 row2=1,15,2
         row3=(row2+1)/2
         do 160 ptr1=1,3
            do 150 ptr2=1,3
               col3=ptr1+ptr2-1
               r11=twor1(row2,ptr1)
               im1=twoim(row2,ptr1)
               r12=twor1(row2+1,ptr2)
               im2=twoim(row2+1,ptr2)
               call cmxmul(r11,im1,r12,im2,r13,im3)
               threr1(row3,col3)=threr1(row3,col3)+r13
               threim(row3,col3)=threim(row3,col3)+im3
 150        continue
 160     continue
 170   continue
c
      do 200 row3=1,7,2
         row4=(row3+1)/2
         do 190 ptr1=1,5
            do 180 ptr2=1,5
               col4=ptr1+ptr2-1
               r11=threr1(row3,ptr1)
               im1=threim(row3,ptr1)
               r12=threr1(row3+1,ptr2)
               im2=threim(row3+1,ptr2)
               call cmxmul(r11,im1,r12,im2,r13,im3)
               fourr1(row4,col4)=fourr1(row4,col4)+r13
               fourim(row4,col4)=fourim(row4,col4)+im4
 180        continue
 190     continue
 200  continue
c
      do 230 row4=1,3,2
         row5=(row4+1)/2
         do 220 ptr1=1,9
            do 210 ptr2=1,9
               col5=ptr1+ptr2-1
               r11=fourr1(row4,ptr1)
               im1=fourim(row4,ptr1)
               r12=fourr1(row4+1,ptr2)
               im2=fourim(row4+1,ptr2)
               call cmxmul(r11,im1,r12,im2,r13,im3)
               fiver1(row5,col5)=fiver1(row5,col5)+r13
               fiveim(row5,col5)=fiveim(row5,col5)+im3
 210        continue
 220     continue
 230  continue
c
      do 250 ptr1=1,17
         do 240 ptr2=1,17
            col6=ptr1+ptr2-1
            r11=fiver1(1,ptr1)
            im1=fiveim(1,ptr1)
            r12=fiver1(2,ptr2)
            im2=fiveim(2,ptr2)
            call cmxmul(r11,im1,r12,im2,r13,im3)
            sixr1(col6)=sixr1(col6)+r13
            sixim(col6)=sixim(col6)+im3
 240     continue
 250  continue
c
c     store results in arrays rslt in order (s**0),(s**1).....
c
      do 260 lp2=1,33
         rsltr1(lp2)=sixr1(34-lp2)
         rsltim(lp2)=sixim(34-lp2)
 260  continue
c
      return
      end
c
c*****************************************************************************
c
c     subroutine to evaluate lattice reactances
c     reference ; Temes and Mitra
c                 Modern filter theory and design
c
      subroutine reacfn
c
      integer ptrp1,order,ptrfs,ordfs,ptrps,ordps,ptrpla,ptrplb,orda
      integer ordb,ptra,ordda,ordna,ptrb,ordnb,orddb
      real temp3,temp4
      double precision polar1(32),polaim(32),poler1,poleim,polbr1(32)
      double precision polbim(32),nadar1(33),nadaim(33),nbdbr1(33)
      double precision nbdbim(33),na,da,nb,db
      complex fs,ps,poles,sumfs,sumps,temp1,temp2
      logical toggle
```

```
c
      common/two/polerl(32),poleim(32),poles(32),fs(33),ps(33)
      common/two1/ordfs,ordps,order
      common/four/na(33),da(33),nb(33),db(33),ordna,ordda,ordnb,orddb
c
c     initialise
c
      do 10 lp1=1,32
         polarl(lp1)=0.0d0
         polaim(lp1)=0.0d0
         polbrl(lp1)=0.0d0
         polbim(lp1)=0.0d0
 10   continue
      do 20 lp2=1,33
         nadarl(lp2)=0.0d0
         nadaim(lp2)=0.0d0
         nbdbrl(lp2)=0.0d0
         nbdbim(lp2)=0.0d0
         na(lp2)=0.0d0
         nb(lp2)=0.0d0
         da(lp2)=0.0d0
         db(lp2)=0.0d0
 20   continue
      ptrpla=1
      ptrplb=1
c
      do 60 ptrpl=1,order
c
c     initialise sumps and sumfs
c
      sumps=(0,0)
      sumfs=(0,0)
c
c        evaluate f(s)
c
         do 30 ptrfs=1,ordfs+1
            sumfs=sumfs+(fs(ptrfs)*(poles(ptrpl)**(ptrfs-1)))
 30      continue
c
c        evaluate p(s)
c
         do 40 ptrps=1,ordps+1
            sumps=sumps+(ps(ptrps)*(poles(ptrpl)**(ptrps-1)))
 40      continue
c
c        evaluate f(s)+p(s) and f(s)-p(s)
c
         temp1=sumfs+sumps
         temp2=sumfs-sumps
         temp3=abs(real(temp1))
         temp4=abs(real(temp2))
c
c        if pole is zero of f(s)-p(s) store in pola
c
         if (temp3.ge.temp4) go to 50
         polarl(ptrpla)=polerl(ptrpl)
         polaim(ptrpla)=poleim(ptrpl)
         ptrpla=ptrpla+1
c
c        if pole is zero of f(s)+p(s) store in polb
c
 50      if (temp3.le.temp4) go to 60
         polbrl(ptrplb)=polerl(ptrpl)
         polbim(ptrplb)=poleim(ptrpl)
         ptrplb=ptrplb+1
c
 60   continue
      orda=ptrpla-1
      ordb=ptrplb-1
c
c     mutliply out pola and polb to obtain nada and nbdb
c
      call mltout(polarl,polaim,nadarl,nadaim,orda)
      call mltout(polbrl,polbim,nbdbrl,nbdbim,ordb)
c
c     split polynomial nada into odd (na) and even(da) parts
c
      toggle=.true.
      do 80 ptra=1,orda+1
         if (.not.toggle) go to 70
         da(ptra)=nadarl(ptra)
         ordda=ptra-1
 70      if (toggle) go to 75
         na(ptra)=nadarl(ptra)
         ordna=ptra-1
 75      toggle=.not.toggle
 80   continue
c
c     split polynomial nbdb into odd (db) and even (nb) parts
c
      toggle=.true.
      do 100 ptrb=1,ordb+1
         if (.not.toggle) go to 90
         nb(ptrb)=nbdbrl(ptrb)
         ordnb=ptrb-1
 90      if (toggle) go to 95
         db(ptrb)=nbdbrl(ptrb)
         orddb=ptrb-1
 95      toggle=.not.toggle
 100  continue
c
      return
      end
c
c*****************************************************************
c
c     subroutine to input tolerance scheme
c
      subroutine tolsch
c
      real dpbej,pbgain,dsbej,sbgain,asbej,apbej,pbloss,sbloss
c
```

```
      common/three/apbej,pbloss,asbej,sbloss
c
c     input digital domain tolerance scheme
c
      print*,'input pbej,pbgain(db),sbej,sbgain(db)'
      read*,dpbej,pbgain,dsbej,sbgain
c
c     obtain analog domain tolerance scheme
c
      apbej=tan(dpbej*3.1415926)
      asbej=tan(dsbej*3.1415926)
      pbloss=-pbgain
      sbloss=-sbgain
c
      return
      end
c
c*************************************************************************
c
c     subroutine for synthesis of lattice reactances with
c     cascaded unit elements.
c     reference ; Temes and Mitra
c
      subroutine uecas(num,den,ordnum,ordden)
c
      integer ptrun,ordnum,ordden,ordup,ptr1,ptr2,ordue
      double precision numx(32),numy(32),denx(32),deny(32),num(32)
      double precision den(32),numsum,densum,unel,z1
      character*3 ellast
c
      common/one/unel(32),ordue,ellast
c
c     initialise pointer,numsum,densum and arrays numx,numy,denx,deny
c
      ptrun=1
 10   numsum=0.0d0
      densum=0.0d0
      do 20 lp1=1,32
         numx(lp1)=0.0d0
         numy(lp1)=0.0d0
         denx(lp1)=0.0d0
         deny(lp1)=0.0d0
 20   continue
c
c     evaluate z(1) and store in array unel
c
      do 30 lp2=1,ordnum+1
         numsum=numsum+num(lp2)
 30   continue
      do 40 lp3=1,ordden+1
         densum=densum+den(lp3)
 40   continue
      z1=numsum/densum
      unel(ptrun)=z1
      ptrun=ptrun+1
c
c     evaluate numx,negate,remove factor ((s**2)-1) and store result
c     in array numy
c
      ordup=max(ordnum,ordden+1)
      numx(1)=-num(1)
      do 50 lp4=2,ordup+1
         numx(lp4)=(num(lp4)-(z1*den(lp4-1)))*z1
         numx(lp4)=-numx(lp4)
 50   continue
c
      ptr1=ordup+1
      ptr2=ptr1-2
      numy(ptr2)=numx(ptr1)
 60   ptr1=ptr1-2
      ptr2=ptr2-2
      if (ptr2.lt.1) go to 70
      numy(ptr2)=numy(ptr2+2)+numx(ptr1)
      go to 60
 70   continue
      ordnmy=ordup-2
c
c     evaluate denx,negate,remove factor ((s**2)-1) and store result
c     in array deny
c
      ordup=max(ordnum+1,ordden)
      denx(1)=-(z1*den(1))
      do 80 lp5=2,ordup+1
         denx(lp5)=(z1*den(lp5))-num(lp5-1)
         denx(lp5)=-denx(lp5)
 80   continue
c
      ptr1=ordup+1
      ptr2=ptr1-2
      deny(ptr2)=denx(ptr1)
 90   ptr1=ptr1-2
      ptr2=ptr2-2
      if (ptr2.lt.1) go to 100
      deny(ptr2)=deny(ptr2+2)+denx(ptr1)
      go to 90
 100  continue
      orddny=ordup-2
c
      if ((ordnmy.eq.0).and.(orddny.eq.1)) go to 130
      if ((ordnmy.eq.1).and.(orddny.eq.0)) go to 140
c
c     transfer array values in numy,deny to num,den
c
      do 110 lp6=1,ordnmy+1
         num(lp6)=numy(lp6)
 110  continue
      do 115 lp7=ordnmy+2,32
         num(lp7)=0.0d0
 115  continue
      do 120 lp8=1,orddny+1
         den(lp8)=deny(lp8)
 120  continue
```

```
      do 125 lp9=orddny+2,32
         den(lp9)=0.0d0
  125 continue
      ordnum=ordnmy
      ordden=orddny
c
      go to 10
c
c     last element open circuited
c
  130 ellast='o/c'
      unel(ptrun)=numy(1)/deny(2)
      ordue=ptrun
      return
c
c     last element short circuited
c
  140 ellast='s/c'
      unel(ptrun)=numy(2)/deny(1)
      ordue=ptrun
      return
      end
c
c*******************************************************************************
```

APPENDIX D2

```
c
c*****************************************************************
c
c     program for the design of lattice WDFs based on Cauer form 1
c     LC ladders
c
c*****************************************************************
c
      integer ordna,ordda,ordnb,orddb
      double precision na,da,nb,db
c
      common/four/na(33),da(33),nb(33),db(33),ordna,ordda,ordnb,orddb
c
      print*,'1: butterworth'
      print*,'2: chebychev'
      read*,menu1
      call tolsch
      if (menu1.eq.1) call butwth
      if (menu1.eq.2) call chbchv
c
      call reacfn
      call cauer(na,da,ordna,ordda)
      call dfcoef
      call cauer(nb,db,ordnb,orddb)
      call dfcoef
      stop
      end
c
c*****************************************************************
c
c     subroutine to evaluate transfer function using Butterworth
c     approximation
c
      subroutine butwth
c
      this routine is given in appendix E1
c
c
c*****************************************************************
c
c     subroutine to evaluate transfer function using Chebyshev
c     approximation
c
      subroutine chbchv
c
      this routine is given in appendix E1
c
c
c*****************************************************************
c
c     subroutine for complex multiplication with double precision
c
      subroutine cmxmul(r11,im1,r12,im2,r13,im3)
c
      this routine is given in appendix E1
c
c*****************************************************************
c
c     subroutine to evaluate digital filter coefficients
c
      subroutine dfcoef
c
      integer ordlc
      double precision lc, ueimp, temp1, temp2, gamma1, gamma2
c
      common/zzzz/lc(32), ueimp(32)
      common/yyyy/uepos,ordlc,cap
c
c     convert unit element impedance values into admittance values
c
      do 10 lp1=1,ordlc-1
         if (ueimp(lp1).eq.0.0d0) go to 10
         ueimp(lp1)=1.0d0/ueimp(lp1)
 10   continue
c
c     evaluate gamma values and print
c
      temp1=1.0d0 + ueimp(1) +lc(1)
      gamma1= 2.0d0/temp1
      gamma2= (2.0d0*ueimp(1))/temp1
      print*,gamma1,gamma2
c
      temp=float(ordlc)/2.0
      if (temp.ne.(float(int(temp)))) go to 30
      do 20 lp2=2,ordlc-1
         temp1=lc(lp2) + ueimp(lp2-1) + ueimp(lp2)
         gamma1= (2.0d0 * ueimp(lp2-1))/temp1
         gamma2= (2.0d0 * ueimp(lp2))/temp1
         print*, gamma1, gamma2
 20   continue
c
      temp1=1.0d0/(lc(ordlc))
      temp2=1.0d0/(ueimp(ordlc-1))
      gamma1=(2.0d0*temp1)/(temp1+temp2)
      print*,gamma1
c
      return
c
 30   continue
      if (ordlc.le.3) go to 50
      do 40 lp2=2,ordlc-2
         temp1=lc(lp2) + ueimp(lp2-1) + ueimp(lp2)
         gamma1= (2.0d0 * ueimp(lp2-1))/temp1
         gamma2= (2.0d0 * ueimp(lp2))/temp1
         print*, gamma1, gamma2
 40   continue
c
 50   temp1=1.0d0/(lc(ordlc))
      temp2=ueimp(ordlc-2)
      gamma1=(2.0d0*temp2)/(temp1+temp2+lc(ordlc-1))
      gamma2=(2.0d0*temp1)/(temp1+temp2+lc(ordlc-1))
      print*,gamma1,gamma2
```

```
c
      return
      end
c
c*********************************************************************
c
      subroutine mltout(rootrl,rootim,rsltrl,rsltim,numrts)
c
      this routine is given in appendix E1
c
c*********************************************************************
c
c     subroutine to evaluate lattice reactances
c
      subroutine reacfn
c
      this routine is given in appendix E1
c
c*********************************************************************
c
c     subroutine to shift unit element to required position
c
      subroutine shftue
c
      integer uepos
      double precision lc, ueimp, L, C, z
      logical cap
c
      common/zzzz/lc(32),ueimp(32)
      common/yyyy/uepos,ordlc,cap
c
      z=1.0d0
      do 30 lp1=1,uepos
       if (cap) go to 10
         L=lc(lp1)
         C=L/(z*(z+L))
          z=z+L
         lc(lp1)=C
         go to 20
c
 10      C=lc(lp1)
         L=((z*C)*z)/(1.0d0+(z*C))
         z=z/(1.0d0+(z*C))
         lc(lp1)=L
c
 20      continue
         cap = .not.cap
 30   continue
      ueimp(uepos)=z
c
      return
      end

c*********************************************************************
c
c     subroutine to input tolerance scheme
c
      subroutine tolsch
c
      this routine is given in appendix E1
c
c*********************************************************************
c
c     subroutine for synthesis of lattice reactances with Cauer
c     form 1 LC ladders
c
      subroutine cauer(num,den,ordnum,ordden)
c
      integer uepos, ordnum, ordden, lcptr, polyup, ordlc
      double precision num(32), den(32), temp(32), lc,ueimp, L, C, z
      logical inicap
c
      common/zzzz/lc(32),ueimp(32)
      common/yyyy/uepos,ordlc,inicap
c
c     synthesize Cauer form 1 lc ladder
c
c     initialise pointers
c
      lcptr=1
c
      polyup=ordnum+1
c
 10   continue
c
c     remove inductor
c
      lc(lcptr) = num(polyup)/den(polyup-1)
c
c     multiply denominator by s * inductor value
c
      do 20 lp1=1,polyup-1
         temp(lp1+1)=den(lp1)*lc(lcptr)
 20   continue
c
c     subtract above polynomial from numerator polynomial
c
      do 30 lp2=1,polyup
         num(lp2)=num(lp2)-temp(lp2)
 30   continue
c
      polyup=polyup-1
      if (polyup.eq.1) go to 60
      lcptr=lcptr+1
c
c     remove capacitor
c
      lc(lcptr)=den(polyup)/num(polyup-1)
c
c     multiply numerator polynomial by s * capacitor value
c
      do 40 lp3=1,polyup-1
```

```
         temp(lp3+1)= num(lp3)*lc(lcptr)
 40   continue
c
c     subtract above polynomial from denominator polynomial
c
      do 50 lp4=1,polyup
         den(lp4)=den(lp4)-temp(lp4)
 50   continue
c
      polyup=polyup-1
      if (polyup.eq.1) go to 60
      lcptr=lcptr+1
      go to 10
c
 60   continue
c
c     perform Kuroda's tranformation to insert unit elements
c
      ordlc=ordnum
      temp1=float(ordlc)/2.0
      if (temp1.eq.(float(int(temp1)))) go to 70
c
c     for odd order lc ladder, insert one unit element from
c     right hand side
c
      numlc=ordlc-1
      go to 80
 70   numlc=ordlc
 80   continue
c
c     for even order lc ladder, insert unit elements from
c     left hand side
c
      inicap=.false.
      do 90 lp5=1,numlc-1
         uepos=numlc-lp5
         call shftue
         inicap=.not.inicap
 90   continue
c
      return
      end
```

APPENDIX E1

```
*********************************************************
*                                                       *
*   program for the functional level simualtion of a    *
*   reconfigurable programmable architecture for        *
*   cascaded unit elements and lattice WDFs             *
*                                                       *
*********************************************************

      double precision inp1a,inp1b,regp,regq
      double precision impse1(1024),impse2(1024)
      double precision regk,regl,regm,regr
      real out1(1024),out2(1024),time(1024)
      real freq(1024),gain(1024)
      integer filtyp,hlfcyc
      logical clk
c
      common/twoa/regk,regl,regm,regp,regq,regr
      common/three/hlfcyc,clk
      common/four/inp1a,inp1b,cofram(32)
      common/five/volt1,sigfr1,sigtp1
      common/six/volt2,sigfr2,sigtp2
c
      open(10,file='grafs',status='old')
      rewind(10)
      open(11,file='time1',status='old')
      rewind(11)
      open(12,file='time2',status='old')
      rewind(12)
c
      print*,'input filter type'
      print*,'1:uewdf(sep)   2:uewdf(cas)     3:lattice'
      read*,filtyp
c
c     reset registers
c
      call reset
c
c     load coefficients into ram
c
      call coflod(cofram)
c
c     input signal parameters
c
      call sigpar(filtyp)
c
c     run simulation
c
      do 20 isamp=1,1024
         call sigone(isamp)
         inp1a=dble(volt1)
         call sigtwo(isamp)
         inp1b=dble(volt2)
         hlfcyc=0
         clk=.true.
c
         do 10 lp1=1,100
            call contrl
            call state(filtyp)
            clk=.not.clk
 10      continue
c
         impse1(isamp)=regr
         impse2(isamp)=regq
         out1(isamp)=sngl(regr)
         out2(isamp)=sngl(regq)
         write(12,*)float(isamp-1),out2(isamp)
 20   continue
c
      print*,'1: display frequency response'
      print*,'2: display transient response'
      read*,menu1
c
      if (menu1.eq.1) then
       call forier(impse1,freq,gain)
       call plot(freq,gain)
       call forier(impse2,freq,gain)
       call plot(freq,gain)
      endif
c
c     generate array time
c
      do 60 lp2=1,1024
         time(lp2)=float(lp2)
 60   continue
c
      call plot(time,out1)
      call plot(time,out2)
c
      stop
      end
c
c*****************************************************
c
      subroutine plot(x,y)
c
c     insert suitable plot routine
c
c*****************************************************
c
c     subroutine to load coefficient values into
c     coefficient RAM
c
      subroutine coflod(cofrm)
c
      real cofrm(32),alphaa(16),alphab(16)
      integer ordera,orderb
c
      open(07,file='coeffA',status='old')
      open(08,file='coeffB',status='old')
      rewind(07)
      rewind(08)
c
```

```
c     initialise array alpha
c
      do 10 lp1=1,16
         alphaa(lp1)=0.0
         alphab(lp1)=0.0
 10   continue
c
c     input order and multiplier coefficient values
c
c     print*,'input filter order for filter A'
      read(7,*)ordera
      do 20 lp2=1,ordera+1
c        print*,'input alpha..',lp2
         read(7,*)alphaa(lp2)
 20   continue
c
c     print*,'input filter order for filter B'
      read(8,*)orderb
      do 30 lp3=1,orderb+1
c        print*,'input alpha..',lp3
         read(8,*)alphab(lp3)
 30   continue
c
      do 40 lp4=1,15,2
         cofrm(lp4)=alphaa(lp4)
         cofrm(lp4+1)=alphab(lp4+1)
 40   continue
c
      do 50 lp5=1,15,2
         cofrm(lp5+16)=alphab(lp5)
         cofrm(lp5+17)=alphaa(lp5+1)
 50   continue
c     load alpha values into array cofrm
c
      return
      end
c
c*******************************************************
c
c     subroutine to generate clock and control signals
c
      subroutine contrl
c
      integer hlfcyc
      logical clk,clkbar,clk1,clk2,clk3,clk4,clk5,clk6,clk7,clk8
      logical mcon1,mcon2,mcon3a,mcon3b,mcon3
      logical hfrm1,hfrm2,hfrm1a,hfrm2a
      logical frmcfa,frmcfb
c
      common/one/clk1,clk2,clk3,clk4,clk5,clk6,clk7,clk8
      common/onea/mcon1,mcon2,mcon3
      common/three/hlfcyc,clk
      clkbar=.not.clk
      hlfcyc=hlfcyc+1
      if (hlfcyc.le.38) then
         hfrm1=.true.
         else
         hfrm1=.false.
      endif
      if ((hlfcyc.gt.38).and.(hlfcyc.le.76)) then
         hfrm2=.true.
         else
         hfrm2=.false.
      endif
      if ((hlfcyc.eq.1).or.(hlfcyc.eq.2)) then
         mcon1=.true.
         else
         mcon1=.false.
      endif
      if ((hlfcyc.eq.39).or.(hlfcyc.eq.40)) then
         mcon2=.true.
         else
         mcon2=.false.
      endif
      if ((hlfcyc.eq.31).or.(hlfcyc.eq.32)) then
         mcon3a=.true.
         else
         mcon3a=.false.
      endif
      if ((hlfcyc.eq.69).or.(hlfcyc.eq.70)) then
         mcon3b=.true.
         else
         mcon3b=.false.
      endif
      mcon3=mcon3a.or.mcon3b
      if (hlfcyc.le.36) then
         hfrm1a=.true.
         else
         hfrm1a=.false.
      endif
      if ((hlfcyc.gt.38).and.(hlfcyc.le.74)) then
         hfrm2a=.true.
         else
         hfrm2a=.false.
      endif
      clk1=((hfrm1a.and.clkbar).or.(hfrm2a.and.clkbar))
      clk2=((hfrm1.and.clkbar).or.(hfrm2.and.clkbar))
      if (hlfcyc.eq.6) then
         clk3=.true.
         else
         clk3=.false.
      endif
      if (hlfcyc.eq.44) then
         clk4=.true.
         else
         clk4=.false.
      endif
      if ((hlfcyc.eq.36).or.(hlfcyc.eq.74)) then
         clk5=.true.
         else
         clk5=.false.
      endif
```

```
      if (h1fcyc.eq.2) then
         clk6=.true.
         else
         clk6=.false.
      endif
      if (h1fcyc.eq.40) then
         clk7=.true.
         else
         clk7=.false.
      if ((h1fcyc.ge.3).and.(h1fcyc.le.34)) then
         frmcfa=.true.
         else
         frmcfa=.false.
      endif
      if ((h1fcyc.ge.41).and.(h1fcyc.le.72)) then
         frmcfb=.true.
         else
         frmcfb=.false.
      endif
      clk8=((clkbar.and.frmcfa).or.(clkbar.and.frmcfb))
      endif
c
      return
      end
c
c*************************************************************
c
c     subroutine to evaluate frequency response from
c     impulse response
c
      subroutine forier(timrsp,freq,gain)
c
      double precision timrsp(1024),work(1024)
      real rloss(1024),freq(1024),gain(1024)
      integer ifail
      complex crloss(1024)
c
      nspt=1024
c
      do 10 lp1=1,nspt
         temp2=sqrt(float(nspt))
         timrsp(lp1)=dble(temp2)*timrsp(lp1)
 10   continue
c
      freq(1)=0.0
      dfreq=1.0/nspt
      do 20 lp2=1,nspt-1
         freq(lp2+1)=freq(lp2)+dfreq
 20   continue
c
      ifail=0
      call c06faf(timrsp,nspt,work,ifail)
c
      do 30 lp3=1,nspt
         rloss(lp3)=sngl(timrsp(lp3))
 30   continue
c
      do 40 lp4=1,nspt/2
         crloss(lp4)=cmplx(rloss(lp4),0.0)
         crloss(nspt+1-lp4)=cmplx(rloss(lp4),0.0)
 40   continue
c
      crloss(nspt)=cmplx(real(crloss(nspt)),0.0)
      do 50 lp5=nspt/2+2,nspt
         crloss(lp5-1)=cmplx(real(crloss(lp5-1)),rloss(lp5))
 50   continue
c
      do 60 lp6=1,nspt/2
         crloss(lp6)=cmplx(real(crloss(lp6)),-aimag(crloss(nspt+1-lp6)))
 60   continue
c
      do 70 lp7=1,nspt
         temp1=(crloss(lp7)*conjg(crloss(lp7)))
         if (temp1.eq.0.0) temp1=1.0e-20
         gain(lp7)=10*alog10(temp1)
 70   continue
c
      return
      end
c
c*************************************************************
c
c     subroutine to reset registers
c
      subroutine reset
c
      double precision stk1,stk2,stka,stkc,pipe1,pipe2
      double precision regk,regl,regm,regp,regq,regr
c
      common/two/stk1(16),stk2(17),stka(2),stkc(2),pipe1,pipe2
      common/twoa/regk,regl,regm,regp,regq,regr
c
      do 10 lp1=1,17
         stk2(lp1)=0.0d0
 10   continue
      do 20 lp2=1,16
         stk1(lp2)=0.0d0
 20   continue
      stka(1)=0.0d0
      stka(2)=0.0d0
      stkc(1)=0.0d0
      stkc(2)=0.0d0
      pipe1=0.0d0
      pipe2=0.0d0
      regk=0.0d0
      regl=0.0d0
      regm=0.0d0
      regp=0.0d0
      regq=0.0d0
      regr=0.0d0
c
      return
```

```
      end
c
c*********************************************************************
c
c     signal generating subroutine for signal one
c
      subroutine sigone(isamp)
c
      real volt1,sigfr1
      integer isamp,sigtp1
c
      common/five/volt1,sigfr1,sigtp1
c
      goto (10,20,30), sigtp1
 10   temp1=cos(0.20*float(isamp-1))
      temp2=0.2*cos(1.0*float(isamp-1))
      volt1=temp1+temp2
       write(11,*)float(isamp-1),volt1
c10   volt1=cos(sigfr1*float(isamp-1))
      return
 20   if (isamp.eq.1) then
         volt1=1.0
         else
         volt1=0.0
      endif
      return
 30   volt1=0.0
      return
      end
c
c*********************************************************
c
c     signal generating subroutine for signal two
c
      subroutine sigtwo(isamp)
c
      real volt2,sigfr2
      integer isamp,sigtp2
c
      common/six/volt2,sigfr2,sigtp2
c
      goto (10,20,30), sigtp2
 10   volt2=cos(sigfr2*float(isamp-1))
      return
 20   if (isamp.eq.1) then
         volt2=1.0
         else
         volt2=0.0
      endif
      return
 30   volt2=0.0
      return
      end
c
c*********************************************************
c
c     subroutine for evaluating register states
c
      subroutine state(filtyp)
c
      double precision inp1a,inp1b,inpip1,inpip2,m1,m2,m3,m4
      double precision stk1,stk2,stka,stkc,pipe1,pipe2,regk
      double precision regl,regm,regp,regq,regr,oplat,opue,outpt
      double precision b,d
      integer filtyp
      logical clk1,clk2,clk3,clk4,clk5,clk6,clk7,clk8
      logical mcon1,mcon2,mcon3
c
      common/one/clk1,clk2,clk3,clk4,clk5,clk6,clk7,clk8
      common/onea/mcon1,mcon2,mcon3
      common/two/stk1(16),stk2(17),stka(2),stkc(2),pipe1,pipe2
      common/twoa/regk,regl,regm,regp,regq,regr
      common/four/inp1a,inp1b,cofram(32)
c
c     first evaluate all combinational functions
c
      oplat=regk-regl
      opue=regm
      outpt=regr
c
      if (filtyp.eq.3) then
         m1=inp1a
         m4=oplat
      endif
      if (filtyp.eq.2) then
         m1=outpt
         m4=opue
      endif
      if (filtyp.eq.1) then
         m1=inp1b
         m4=opue
      endif
c
      if (mcon1) m2=inp1a
      if (mcon2) m2=regp
      if (.not.(mcon1.or.mcon2)) m2=stk2(1)
c
      if (mcon3) then
         m3=0.0d0
         else
         m3=stk1(1)
      endif
c
      inpip1=m3-m2
      inpip2=pipe1*dble(cofram(1))
      d=pipe2+stkc(2)
      b=pipe2+stka(2)
c
c     on clk1, clock pipeline registers and register stacks a,c,2
c
      if (clk1) then
         do 10 lp1=1,16
```

```
            stk2(lp1)=stk2(lp1+1)
 10       continue
          stk2(17)=b
          stkc(2)=stkc(1)
          stkc(1)=m3
          stka(2)=stka(1)
          stka(1)=m2
          pipe1=inpip1
          pipe2=inpip2
       endif
c
c      on clk2, clock register stack 1
c
       if (clk2) then
          do 20 lp2=1,15
             stk1(lp2)=stk1(lp2+1)
 20       continue
          stk1(16)=d
       endif
c
c      on clk3, clock register k
c
       if (clk3) regk=d
c
c      on clk4, clock register l
c
       if (clk4) regl=d
c
c      on clk5, clock register m
c
       if (clk5) regm=b
c
c      on clk6, clock registers p and r
c
       if (clk6) then
          regp=m1
          regr=m4
       endif
c
c      on clk7, clock register q
c
       if (clk7) regq=m4
c
c      on clk8, clock coefficient ram
c
       if (clk8) then
          temp1=cofram(1)
          do 30 lp3=1,31
             cofram(lp3)=cofram(lp3+1)
 30       continue
          cofram(32)=temp1
       endif
c
       return
       end
```

APPENDIX F1

```
c
*****************************************************************
*                                                               *
*       program for the functional level simulation of a        *
*       reconfigurable programmable architecture for            *
*       LC ladder and lattice WDFs                              *
*                                                               *
*****************************************************************
c
      double precision inp1a,inp1b,regp,regq
      double precision impse1(1024),impse2(1024)
      double precision regk,regl,regm,regr
      real out1(1024),out2(1024),time(1024)
      real freq(1024),gain(1024)
      integer filtyp,hlfcyc
      logical clk
c
      common/twoa/regk,regl,regm,regp,regq,regr
      common/three/hlfcyc,clk
      common/four/inp1a,inp1b,cofrm1(32),cofrm2(32),cofrm3(32)
      common/five/volt1,sigfr1,sigtp1
      common/six/volt2,sigfr2,sigtp2
c
c     input filter type
c
      print*,'input filter type'
      print*,'1:lc ladder(sep)   2:lc ladder(cas)     3:lattice'
      read*,filtyp
c
      open(12,file='results',status='old')
      rewind(12)
c
c     reset registers
c
      call reset
c
c     load coefficients into ram
c
      call cofld1(cofrm1,cofrm3)
      call cofld2(cofrm2)
c     input signal parameters
c
      call sigpar(filtyp)
c
c     run simulation
c
      do 20 isamp=1,1024
         call sigone(isamp)
         inp1a=dble(volt1)
         call sigtwo(isamp)
         inp1b=dble(volt2)
         hlfcyc=0
         clk=.true.
c
         do 10 lp1=1,100
            call contrl
            call state(filtyp)
            clk=.not.clk
 10      continue
c
         impse1(isamp)=regr
         impse2(isamp)=regq
         out1(isamp)=sngl(regr)
         out2(isamp)=sngl(regq)
 20   continue
c
      print*,'1: display frequency response'
      print*,'2: display transient response'
      read*,menu1
c
      if (menu1.eq.1) then
       call forier(impse1,freq,gain)
       call plot(freq,gain)
c
       call forier(impse2,freq,gain)
       call plot(freq,gain)
      endif
c
c     generate array time
c
      do 60 lp2=1,1024
         time(lp2)=float(lp2)
 60   continue
c
      call plot(time,out1)
      call plot(time,out2)
c
      stop
      end
c
c*****************************************************************
c
      SUBROUTINE PLOT(X,Y)
C
      insert suitable plot routine
c
c*****************************************************************
c
c     subroutines to load coefficient values into
c     coefficient RAMs
c
      subroutine cofld1(cofrm,cofrm3)
c
      real cofrm(32),cofrm3(32),alphaa(16),alphab(16)
      real zeroa(16),zerob(16)
      integer ordera,orderb
c
      open(07,file='coeffA',status='old')
      open(08,file='coeffB',status='old')
      rewind(07)
      rewind(08)
c
```

```
c     initialise array alpha
c
      do 10 lp1=1,16
         alphaa(lp1)=-1.0
         alphab(lp1)=-1.0
         zeroa(lp1)=0.0
         zerob(lp1)=0.0
 10   continue
c
c     input order and multiplier coefficient values
c
c     print*,'input filter order for filter A'
      read(7,*)ordera
      do 20 lp2=1,ordera
c        print*,'input alpha..',lp2
         read(7,*)alphaa(lp2)
         zeroa(lp2)=1.0
 20   continue
c
c     print*,'input filter order for filter B'
       read(8,*)orderb
      do 30 lp3=1,orderb
c        print*,'input alpha..',lp3
         read(8,*)alphab(lp3)
         zerob(lp3)=1.0
 30   continue
c
      do 40 lp4=1,15,2
         cofrm(lp4)=alphaa(lp4)
         cofrm(lp4+1)=alphab(lp4+1)
         cofrm3(lp4)=zeroa(lp4)
         cofrm3(lp4+1)=zerob(lp4+1)
 40   continue
c
      do 50 lp5=1,15,2
         cofrm(lp5+16)=alphab(lp5)
         cofrm(lp5+17)=alphaa(lp5+1)
         cofrm3(lp5+16)=zerob(lp5)
         cofrm3(lp5+17)=zeroa(lp5+1)
 50   continue
c
      return
      end
c
c-------------------------------
c
      subroutine cofld2(cofrm)
c
      real cofrm(32),alphaa(16),alphab(16)
      integer ordera,orderb
c
c
c     initialise array alpha
c
      do 10 lp1=1,16
         alphaa(lp1)=0.0
         alphab(lp1)=0.0
 10   continue
c
c     input order and multiplier coefficient values
c
c     print*,'input filter order for filter A'
      read(7,*)ordera
      do 20 lp2=1,ordera
c        print*,'input alpha..',lp2
         read(7,*)alphaa(lp2)
 20   continue
c
c     print*,'input filter order for filter B'
       read(8,*)orderb
      do 30 lp3=1,orderb
c        print*,'input alpha..',lp3
         read(8,*)alphab(lp3)
 30   continue
c
      do 40 lp4=1,15,2
         cofrm(lp4)=alphaa(lp4)
         cofrm(lp4+1)=alphab(lp4+1)
 40   continue
c
      do 50 lp5=1,15,2
         cofrm(lp5+16)=alphab(lp5)
         cofrm(lp5+17)=alphaa(lp5+1)
 50   continue
c     load alpha values into array cofrm
c
      return
      end
c
c*****************************************************************
c
c     subroutine to generate clock and control signals
c
      subroutine contrl
c
      integer hlfcyc
      logical clk,clkbar,clk1,clk2,clk3,clk4,clk5,clk6,clk7,clk8
      logical mcon1,mcon2,mcon3a,mcon3b,mcon3
      logical hfrm1,hfrm2,hfrm1a,hfrm2a
      logical frmcfa,frmcfb
c
      common/one/clk1,clk2,clk3,clk4,clk5,clk6,clk7,clk8
      common/onea/mcon1,mcon2,mcon3
      common/three/hlfcyc,clk
      clkbar=.not.clk
      hlfcyc=hlfcyc+1
      if (hlfcyc.le.38) then
         hfrm1=.true.
         else
         hfrm1=.false.
      endif
      if ((hlfcyc.gt.38).and.(hlfcyc.le.76)) then
```

```
      hfrm2=.true.
      else
      hfrm2=.false.
   endif
   if ((h1fcyc.eq.1).or.(h1fcyc.eq.2)) then
      mcon1=.true.
      else
      mcon1=.false.
   endif
   if ((h1fcyc.eq.39).or.(h1fcyc.eq.40)) then
      mcon2=.true.
      else
      mcon2=.false.
   endif
   if ((h1fcyc.eq.31).or.(h1fcyc.eq.32)) then
      mcon3a=.true.
      else
      mcon3a=.false.
   endif
   if ((h1fcyc.eq.69).or.(h1fcyc.eq.70)) then
      mcon3b=.true.
      else
      mcon3b=.false.
   endif
   mcon3=mcon3a.or.mcon3b
   if (h1fcyc.le.36) then
      hfrm1a=.true.
      else
      hfrm1a=.false.
   endif
   if ((h1fcyc.gt.38).and.(h1fcyc.le.74)) then
      hfrm2a=.true.
      else
      hfrm2a=.false.
   endif
   clk1=((hfrm1a.and.clkbar).or.(hfrm2a.and.clkbar))
   clk2=((hfrm1.and.clkbar).or.(hfrm2.and.clkbar))
   if (h1fcyc.eq.6) then
      clk3=.true.
      else
      clk3=.false.
   endif
   if (h1fcyc.eq.44) then
      clk4=.true.
      else
      clk4=.false.
   endif
   if ((h1fcyc.eq.36).or.(h1fcyc.eq.74)) then
      clk5=.true.
      else
      clk5=.false.
   endif
   if (h1fcyc.eq.2) then
      clk6=.true.
      else
      clk6=.false.
      endif
      if (h1fcyc.eq.40) then
         clk7=.true.
         else
         clk7=.false.
      if ((h1fcyc.ge.3).and.(h1fcyc.le.34)) then
         frmcfa=.true.
         else
         frmcfa=.false.
      endif
      if ((h1fcyc.ge.41).and.(h1fcyc.le.72)) then
         frmcfb=.true.
         else
         frmcfb=.false.
      endif
      clk8=((clkbar.and.frmcfa).or.(clkbar.and.frmcfb))
      endif
c
      return
      end
c
c**********************************************************************
c
c     subroutine to evaluate frequency response from
c     impulse response
c
      subroutine forier(timrsp,freq,gain)
c
      double precision timrsp(1024),work(1024)
      real rloss(1024),freq(1024),gain(1024)
      integer ifail
      complex crloss(1024)
c
      nspt=1024
c
      do 10 lp1=1,nspt
         temp2=sqrt(float(nspt))
         timrsp(lp1)=dble(temp2)*timrsp(lp1)
 10   continue
c
      freq(1)=0.0
      dfreq=1.0/nspt
      do 20 lp2=1,nspt-1
         freq(lp2+1)=freq(lp2)+dfreq
 20   continue
c
      ifail=0
      call c06faf(timrsp,nspt,work,ifail)
c
      do 30 lp3=1,nspt
         rloss(lp3)=sngl(timrsp(lp3))
 30   continue
c
      do 40 lp4=1,nspt/2
         crloss(lp4)=cmplx(rloss(lp4),0.0)
         crloss(nspt+1-lp4)=cmplx(rloss(lp4),0.0)
```

```
 40   continue
c
      crloss(nspt)=cmplx(real(crloss(nspt)),0.0)
      do 50 lp5=nspt/2+2,nspt
         crloss(lp5-1)=cmplx(real(crloss(lp5-1)),rloss(lp5))
 50   continue
c
      do 60 lp6=1,nspt/2
         crloss(lp6)=cmplx(real(crloss(lp6)),-aimag(crloss(nspt+1-lp6)))
 60   continue
c
      do 70 lp7=1,nspt
         temp1=(crloss(lp7)*conjg(crloss(lp7)))
         if (temp1.eq.0.0) temp1=1.0e-20
         gain(lp7)=10*alog10(temp1)
 70   continue
c
      return
      end
c
c*********************************************************************
c
c     subroutine to reset registers
c
      subroutine reset
c
      double precision stk1,stk2,stka,stkc,pipe1,pipe2
      double precision regk,regl,regm,regp,regq,regr
      double precision p1,p2,p3,p4,p5,p6,p7,p8,stk3
c
      common/two/stk1(16),stk2(17),stka(2),stkc(2),pipe1,pipe2
      common/twoa/regk,regl,regm,regp,regq,regr
      common/aaaa/stk3(34),p1,p2,p3,p4,p5,p6,p7,p8
c
      do 10 lp1=1,17
         stk2(lp1)=0.0d0
 10   continue
      do 20 lp2=1,16
         stk1(lp2)=0.0d0
 20   continue
      stka(1)=0.0d0
      stka(2)=0.0d0
      stkc(1)=0.0d0
      stkc(2)=0.0d0
      pipe1=0.0d0
      pipe2=0.0d0
      regk=0.0d0
      regl=0.0d0
      regm=0.0d0
      regp=0.0d0
      regq=0.0d0
      regr=0.0d0
c
      p1=0.0d0
      p2=0.0d0
      p3=0.0d0
      p4=0.0d0
      p5=0.0d0
      p6=0.0d0
      p7=0.0d0
      p8=0.0d0
      do 30 lp3=1,34
         stk3(lp3)=0.0d0
 30   continue
      return
      end
c
c*********************************************************************
c
c     signal generating subroutine for signal one
c
      subroutine sigone(isamp)
c
      real volt1,sigfr1
      integer isamp,sigtp1
c
      common/five/volt1,sigfr1,sigtp1
c
      goto (10,20,30), sigtp1
 10   volt1=cos(sigfr1*float(isamp-1))
      return
 20   if (isamp.eq.1) then
         volt1=1.0
         else
         volt1=0.0
      endif
      return
 30   volt1=0.0
      return
      end
c
c*********************************************************************
c
c     signal generating subroutine for signal two
c
      subroutine sigtwo(isamp)
c
      real volt2,sigfr2
      integer isamp,sigtp2
c
      common/six/volt2,sigfr2,sigtp2
c
      goto (10,20,30), sigtp2
 10   volt2=cos(sigfr2*float(isamp-1))
      return
 20   if (isamp.eq.1) then
         volt2=1.0
         else
         volt2=0.0
      endif
      return
 30   volt2=0.0
```

```
      return
      end
c
c*********************************************************************
c
c     subroutine for evaluating register states
c
      subroutine state(filtyp)
c
      double precision inp1a,inp1b,inpip1,inpip2,m1,m2,m3,m4
      double precision stk1,stk2,stka,stkc,pipe1,pipe2,regk
      double precision regl,regm,regp,regq,regr,oplat,opue,outpt
      double precision b,d
      double precision p1,p2,p3,p4,p5,p6,p7,p8,s1,s2
      double precision add1,add2,add3,ml1,ml2,ml3,stk3
      integer filtyp
      logical clk1,clk2,clk3,clk4,clk5,clk6,clk7,clk8
      logical mcon1,mcon2,mcon3
c
      common/one/clk1,clk2,clk3,clk4,clk5,clk6,clk7,clk8
      common/onea/mcon1,mcon2,mcon3
      common/two/stk1(16),stk2(17),stka(2),stkc(2),pipe1,pipe2
      common/twoa/regk,regl,regm,regp,regq,regr
      common/four/inp1a,inp1b,cofrm1(32),cofrm2(32),cofrm3(32)
      common/aaaa/stk3(34),p1,p2,p3,p4,p5,p6,p7,p8
c
c     first evaluate all combinational functions
c
      oplat=regk-regl
      opue=regm
      outpt=regr
c
      if (filtyp.eq.3) then
         m1=inp1a
         m4=oplat
      endif
      if (filtyp.eq.2) then
         m1=outpt
         m4=opue
      endif
      if (filtyp.eq.1) then
         m1=inp1b
         m4=opue
      endif
c
      if (mcon1) m2=inp1a
      if (mcon2) m2=regp
      if (.not.(mcon1.or.mcon2)) m2=stk2(1)
c
      if (mcon3) then
         m3=0.0d0
         else
         m3=stk1(1)
      endif
c
      inpip1=m3-m2
      inpip2=pipe1*dble(cofrm1(1))
      d=pipe2+stkc(2)
      b=pipe2+stka(2)
      s1=stk3(1)-m2
      s2=stk3(1)-m3
      ml1=p1*dble(cofrm1(1))
      ml2=p3*dble(cofrm2(1))
      ml3=p2*dble(cofrm3(1))
      add1=p5+p6+p7
      add2=p4+add1
      add3=p8+add1
c
c     on clk1, clock pipeline registers and register stacks a,c,2
c
      if (clk1) then
         p4=p1
         p8=p3
         p1=s1
         p2=stk3(1)
         p3=s2
         p5=ml1
         p6=ml3
         p7=ml2
         do 10 lp1=1,16
            stk2(lp1)=stk2(lp1+1)
 10      continue
         stk2(17)=add3
         do 15 lx1=1,33
            stk3(lx1)=stk3(lx1+1)
 15      continue
         stk3(34)=add1
         stkc(2)=stkc(1)
         stkc(1)=m3
         stka(2)=stka(1)
         stka(1)=m2
         pipe1=inpip1
         pipe2=inpip2
      endif
c
c     on clk2, clock register stack 1
c
      if (clk2) then
         do 20 lp2=1,15
            stk1(lp2)=stk1(lp2+1)
 20      continue
         stk1(16)=add2
      endif
c
c     on clk3, clock register k
c
      if (clk3) regk=add2
c
c     on clk4, clock register l
c
      if (clk4) regl=add2
c
```

```
c     on clk5, clock register m
c
      if (clk5) regm=add3
c
c     on clk6, clock registers p and r
c
      if (clk6) then
         regp=m1
         regr=m4
      endif
c
c     on clk7, clock register q
c
      if (clk7) regq=m4
c
c     on clk8, clock coefficient ram
c
      if (clk8) then
         temp1=cofrm1(1)
         temp2=cofrm2(1)
         temp3=cofrm3(1)
         do 30 lp3=1,31
            cofrm1(lp3)=cofrm1(lp3+1)
            cofrm2(lp3)=cofrm2(lp3+1)
            cofrm3(lp3)=cofrm3(lp3+1)
 30      continue
         cofrm1(32)=temp1
         cofrm2(32)=temp2
         cofrm3(32)=temp3
      endif
c
      return
      end
c
c*****************************************************************
c
c     subroutine to input signal parameters
c
      subroutine sigpar(filtyp)
c
      real sigfr1,sigfr2
      integer filtyp,sigtp1,sigtp2
c
      common/five/volt1,sigfr1,sigtp1
      common/six/volt2,sigfr2,sigtp2
c
c     if (filtyp.eq.1) then
         print*,'signal type one'
         print*,'1;cosine     2;impulse    3;zero'
         read*,sigtp1
         print*,'signal two type'
         print*,'1;cosine    2;impulse   3;zero'
         read*,sigtp2
c     endif
c
      if (sigtp1.eq.1) then
         print*,'input freq signal one'
         read*,sigfr1
      endif
      if (sigtp2.eq.1) then
         print*,'input freq signal two'
         read*,sigfr2
      endif
c
      return
      end
```