TIME DOMAIN SIMULATION AND MODELLING OF WAVE DIGITAL FILTERS

by
Randall K. Smith

A thesis
presented to the University of Manitoba
in partial fulfilment of the
requirements for the degree of
Master of Science
In
Electrical and Computer Engineering

Winnipeg, Manitoba, 1990

© Randall K. Smith

National Library of Canada

Bibliothèque nationale du Canada

Canadian Theses Service

Service des thèses canadiennes

Ottawa, Canada
K1A 0N4

The author has granted an irrevocable non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of his/her thesis by any means and in any form or format, making this thesis available to interested persons.

L'auteur a accordé une licence irrévocable et non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de sa thèse de quelque manière et sous quelque forme que ce soit pour mettre des exemplaires de cette thèse à la disposition des personnes intéressées.

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

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

ISBN 0-315-63338-7

TIME DOMAIN SIMULATION AND MODELLING

OF WAVE DIGITAL FILTERS

BY

RANDALL K. SMITH

A thesis submitted to the Faculty of Graduate Studies of the University of Manitoba in partial fulfillment of the requirements of the degree of

MASTER OF SCIENCE

© 1990

Permission has been granted to the LIBRARY OF THE UNIVERSITY OF MANITOBA to lend or sell copies of this thesis. to the NATIONAL LIBRARY OF CANADA to microfilm this thesis and to lend or sell copies of the film, and UNIVERSITY MICROFILMS to publish an abstract of this thesis.

The author reserves other publication rights, and neither the thesis nor extensive extracts from it may be printed or otherwise reproduced without the author's written permission.

I hereby declare that I am the sole author of this thesis.

I authorize the University of Manitoba to lend this thesis to other institutions or individuals for the purpose of scholarly research.

Randall K. Smith

I further authorize the University of Manitoba to reproduce this thesis by photocopying or by other means, in whole or in part, at the request of other institutions or individuals for the purpose of scholarly research.

Randall K. Smith

The University of Manitoba requires the signature of all persons using or photocopying this thesis. Please sign below, and give your address and the date.

ABSTRACT

Simulation of digital circuits is becoming an increasingly popular and important engineering design tool, particulary for digital filters where quantization may have detrimental effects on the desired operation of the filter. To examine the effects of quantization, and for initial design verification, it is useful to have a simulation system that can simulate the filter operating under nominal and fixed-precision conditions. This thesis presents the development of such a simulation system for a certain class of filters known as *wave digital*. Wave digital filters (WDFs), originally introduced by Fettweis in 1971, have many desirable properties that other classes of digital filters lack, making them excellent choices for digital filtering applications.

For modelling purposes, WDFs are viewed as a topological interconnection of WD building blocks. Modelling of the WD building blocks is done in an efficient, yet general enough way to allow for easy expansion. It is found that modelling of the network topology requires that a reference basis be developed for the port numbering scheme of the individual building blocks.

An algorithm that is independent of any type of arithmetic has been developed for the time domain simulation. It is shown that this algorithm may be used to compute the frequency response with the modification of only one step. A sub-algorithm, and the heart of the simulation system, is the ControlLoop algorithm. An important task of the ControlLoop algorithm is the detection of any non-computable filters. An efficient method to calculate the group delay, one of the frequency response characteristics, from the impulse response has been derived.

The results of the simulation are presented in a graphical manner in the form of plots. In order to perform the plotting, dynamic data structures and algorithms including an autoscaling algorithm were developed. The number of plots that can exist at a single time is limited only by the amount of memory that the machine has on which the program is running.

Conditions to ensure passivity is maintained when quantizing the multipliers of the various WD building blocks are defined. Quantization algorithms that have experimentally been found to work well are presented. Methods introduced by Pepe and Rogers for the simulation of various types of two's complement fixed-point binary arithmetic are employed to allow for the simulation of filters operating under finite wordlength conditions.

Procedures for performing multiple modulus residue arithmetic, as well as all forward and inverse mappings, have been developed. An algorithm for performing the simulation of WDFs based on multiple modulus residue number systems has been implemented. Such realizations have applications in filters where high-throughput is the principal design

criterion.

The simulation system has been implemented in LightSpeed Pascal running on the Macintosh computer. It is well structured, highly modular, and is composed of fourteen separate units. Like most Macintosh applications, the simulator takes full advantage of the Macintosh Toolbox routines in its man-machine interface. The combination of pull-down menus, dialog boxes, graphics, and windowing makes the simulator very user-friendly.

The program has been tested by simulating the responses of several WDFs that have appeared in previous literature. No discrepancies could be found in a comparison of the two sets of results.

ACKNOWLEDGEMENTS

The author wishes to express his sincere gratitude to Prof. G.O. Martens for his guidance and encouragement during the course of this work. He also wishes to thank his colleagues M. Jarmasz, G. Scarth, and L. Leung for many informative discussions. Financial support from the Natural Sciences and Engineering Research Council in the form of a Post-Graduate Scholarship is greatly appreciated.

TABLE OF CONTENTS

page

ABSTRACT iv
ACKNOWLEDGEMENTS vi
TABLE OF CONTENTS vii

Chapter

I. INTRODUCTION 1

1.1 General Introduction 1
1.2 Introduction to Wave Digital Filters 3

II. OVERVIEW 5

III. MODELLING OF WDF STRUCTURES 12

3.1 WDF Building Blocks 12
3.1.1 Elements and Sources 12
3.1.2 Interconnections and Adaptors 14
3.1.3 Elementary Two-Port Sections 17
3.2 The **Network** Data Structure 29
3.3 Forming the Internal Connections 30

IV. GENERAL SIMULATION 32

4.1 Time Domain Simulation 32
4.1.1 General Time Domain Simulation Algorithm 32
4.1.2 The ControlLoop Algorithm 34
4.1.3 The **Updators** Unit 35
4.2 Frequency Responses 38
4.2.1 Calculation of the Frequency Response 38
4.2.2 Calculation of the Narrowband Response 38
4.2.3 Calculation of the Group Delay 39
4.3 The **Simulation** Record 41

4.4 Plotting the Results 43
4.4.1 Plotting Data Structures 43
4.4.2 The **Plotting** Unit 45

V. SIMULATION WITH QUANTIZATION 47

5.1 Quantizing Multipliers 47
5.1.1 Quantizing Parallel and Series Adaptors 47
5.1.2 Quantizing Inverse Multipliers 48
5.1.3 Quantizing Normalized Two-Port Adaptors 49
5.2 Two's Complement Fixed-Point Binary Arithmetic 50
5.3 Quantized Simulation of Elementary Two-Port Sections
Based on Voltage Wave Adaptors 52

VI. SIMULATION OF WDFS IN RESIDUE NUMBER SYSTEMS 58

6.1 Basic Concepts 58
6.2 The Autoscale Residue Multiplier [16] 60
6.3 The Inverse Mappings 62
6.4 RNS Simulation 63

VII. MAN-MACHINE INTERFACE 65

7.1 Mouse Events 65
7.1.1 Menu Commands 67
7.1.2 Other Mouse Events 71
7.2 Key Events 73
7.3 Activate Events 74
7.4 Update Events 75

VIII. SIMULATION EXAMPLES 76

8.1 Example 1: Fifth-Order Lowpass Filter [26] 76
8.2 Example 2: Seventh-Order Lowpass Filter [26] 81
8.3 Example 3: Seventh-Order Lowpass Filter [27,4] 85
8.4 Example 4: Eighth-Order Bandpass Filter [29,4] 90
8.4.1 Example 4 Realized as a Relectance 90
8.4.2 Example 4 Realized as a Transmittance 93

IX. CONCLUSIONS AND RECOMMENDATIONS 96

REFERENCES 99

Appendix

A: WDFSim Program Listing 102
B: Example Program to Generate a File Containing a Sequence of Signal Sample Values 185
C: Programs to Generate the Pictures for WDFSim and Save them in a Resource File as Resource Type 'PICT' 187
D: Program and Text File to Generate the **Connections** Data Structure and Store it in a Resource File as Resource Type 'PORT' 249
E: Program to Read a PICT File and Save it in a Resource File as Resource Type 'PICT' 255

I. INTRODUCTION

1.1 General Introduction

A digital filter designer, starting with a set of design specifications, first selects a filter prototype that meets the specifications. Through the use of such design techniques as frequency scaling, lowpass-to-bandpass transformations, etc., a nominal design is obtained. By nominal, we mean floating point (perhaps infinite precision) representation for the filter coefficients. For implementation, either special purpose hardware or digital signal processor (DSP) based, the designer is faced with the problem of finding the optimal binary representation (number of bits) to use. A solution may be found through the use of simulation.

A simulation system that can show the filter operating in finite precision and compare it directly to the highest precision available (nominal) would be required. The designer could test different quantization schemes and observe the behavior of the filter, getting an immediate indication of the performance of a given design. Typical performance measurements include frequency response characteristics (particularily attenuation) and stability. Simulation with nominal values would also be useful for initial design verification.

This thesis presents the development of such a simulation system for wave digital filters. The simulation program, called WDFSim, is intended to be used by the WDF designer. The user of WDFSim (and the reader of this thesis) is expected to have a basic knowledge of WDFs and WDF design procedures. WDFSim can simulate the response of WDFs with nominal coefficients as well as quantized coefficients with varying wordlengths. Simulation results can be compared immediately in order to arrive at an optimal binary solution. As well, the program can simulate the response of a WDF that has been implemented based on residue number systems. Such realizations have applications in filters where high throughput is the principle design criterion.

WDFSim has been implemented in LightSpeed Pascal running on the Macintosh computer. Like most Macintosh applications, it is highly graphically oriented and is designed to be very user-friendly. In Chapter II, an overview of WDFSim is presented. The overview is intended to introduce the reader to the features of WDFSim as well as some of the terminology used throughout the remainder of this thesis.

In Chapter III, the WDF building blocks implemented in WDFSim are introduced. These include many of the elements and adaptors presented in Fettweis [2] as well as eleven elementary two-port sections from Jarmasz [4]. Data structures that define the computer models for both the individual building blocks and the network topology are described.

General simulation algorithms for both time domain and frequency responses and the related data structures are described in Chapter IV. By the term general, we mean algorithms that are independent of the type of number system chosen (nominal, binary, or residue). Also, plotting algorithms and data structures are defined.

In Chapter V, algorithms and conditions to ensure passivity is maintained when quantizing the multipliers of the different types of building blocks are described. Simulation of various types of two's complement fixed-point binary arithmetic is explained. A method for reducing the number of sources of passivity introduced when quantizing the multipliers of the elementary two-port sections is presented.

Single modulus and multiple modulus residue number systems are defined in Chapter VI. It is found that certain choices for the moduli set lead to efficient scaling and decoding implementations. Scaling and decoding algorithms for a residue number system based on the moduli set $P = \{2^n - 1, 2^n, 2^n + 1\}$ are described. Procedures for performing the simulation of WDFs based on residue number systems are presented.

In Chapter VII, the man-machine interface of WDFSim is described. Like most Macintosh applications, WDFSim is event driven. Four types of events are of interest to WDFSim: mouse events, key events, update events, and activate events. The procedures taken by WDFSim in response to each of these event types are also described in Chapter VII.

Four simulation examples that have appeared in the literature are presented in Chapter VIII. Finally, conclusions and some recommendations for further work are given in Chapter IX. References and appendices follow.

1.2 Intoduction to Wave Digital Filters

Wave digital filters (WDFs), originally introduced by Fettweis [1], represent a class of digital filters that are closely related to classical filter networks [3]. To every WDF, there corresponds an analog *reference filter* from which it is derived. In the frequency domain, the correspondence between a WDF and its reference filter is defined by the change of frequency variable and the bilinear transform

$$\psi = \tanh\left(\frac{sT}{2}\right) = \frac{z-1}{z+1} \ , \ z = e^{sT} \ , \tag{1.1}$$

where s is the actual complex frequency (Laplace variable), z is the digital domain variable, and T is the sampling period. The additional complex frequency variable ψ is introduced so that rational transfer functions in the reference domain (ψ domain) map to rational functions in the digital domain, and vice versa. The mapping (1.1), hereafter denoted by the symbol $\leftrightarrow$, has some additional properties:

1. The imaginary axis $\psi = j\phi$ is mapped onto the imaginary axis $s = j\omega$ and one-to-one onto the unit circle $z = e^{j\omega T}$, where $\phi = \tan\left(\frac{\omega T}{2}\right)$.

2. The Nyquist range $0 \leq \omega \leq \frac{\pi}{T}$ is mapped one-to-one and onto the range $0 \leq \phi \leq \infty$.

3. Stable filters in the reference domain are mapped to stable and causal filters in the digital domain.

In the time domain, the analogy between a WDF and its corresponding reference filter is based, not on voltages and currents as signal variables, but on so-called *wave quantities* [3]. Consider the analog port shown in Fig. 1.1, where R is an arbitrary positive port reference. The wave quantities are defined by

$$A = V + RI \ , \ B = V - RI \ , \tag{1.2}$$

where A and B are known as the *incident* and *reflected* (voltage) waves, respectively. Wave quantities are used rather than voltages and currents as signal variables to ensure realizability [1,2].

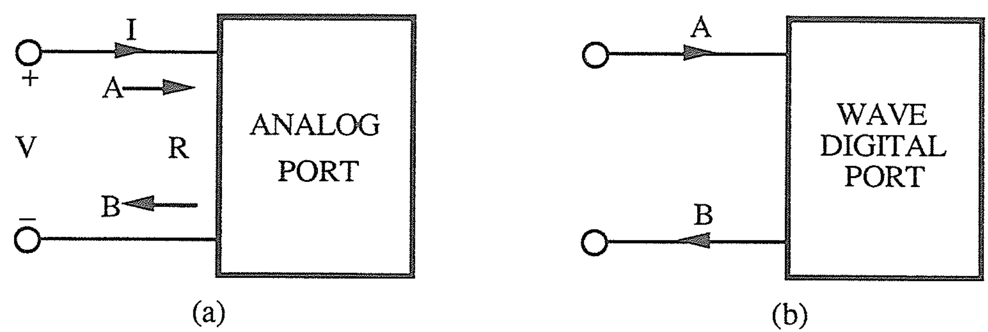


Fig. 1.1: Wave quantities as port variables in (a) analog and
(b) wave digital domains.

WDFs inherit may desirable properties from their analog reference filters, all of which make them a viable choice for digital filtering applications. In particular, the class of WDFs derived from doubly-terminated lossless analog filters exhibit the features of passivity and low sensitivity to parameter variation. Another important feature of WDFs is the suppression of zero-input parasitic oscillations (limit cycles) and forced-response stability by using only magnitude truncation and saturation overflow arithmetic. Also, WDFs provide two complementary transfer functions, the transmittance and the reflectance, making them ideal for branching filters.

II. OVERVIEW

After launching WDFSim by double-clicking on its icon, a new file can be created by selecting the **New** command from the **File** menu. A window similar to the one shown in Fig. 2.1 appears. This window is called the *File Window* and is initially empty. The File Window contains sixty cells (or nodes) arranged in six horizontal rows of ten cells each. Each node is assigned a number as outlined in Fig. 2.1. Only one file (and hence File Window) may be open at one time.

The WDF structure to be simulated is entered into the File Window in a graphical manner as follows: Click the mouse button with the cursor in a cell where you want to put one of the WDF building blocks. Note that the cell reverses color, what was black becomes white and what was white becomes black. This process is called activating a node, and the reversed node is called the *Active Node.* Now, with the desired node active, select one of the WDF building block types from either the **Element** or the **Adaptor** menus. A dialog box similar to the one shown in Fig. 2.2 will appear. Click on the desired orientation of the building block and then click the dialog's **OK** button. If the building block contains multipliers, another dialog box, similar to the one shown in Fig. 2.3, will appear asking for the multiplier values. Type them in the appropriate places and then click the dialog's **OK** button. The dialog disappears and the File Window is redrawn with the new building block in place. Continue on in this manner until all the building blocks of the filter have been entered.

Untitled

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60

Fig. 2.1: The File Window and node numbering system.

If a building block has been entered into the structure and is no longer required, it can be removed by a two step process: first, activate the desired node; and second, press the **delete** key. It is possible to examine (and modify) the multipliers within a building block that already exists in the File Window. Holding down the **option** key while clicking to activate the node causes a dialog box similar to the one shown in Fig. 2.4 to appear. Note that this is the same dialog box as in Fig. 2.3, but the existing multiplier values (which can now be modified) are filled in. Click the dialog's **OK** button to record any changes and to dispose of the box.

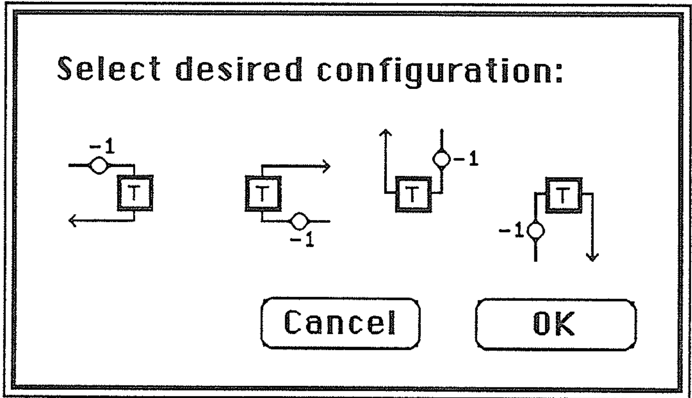


Fig. 2.2: An orientation dialog box.

Once the entire filter has been entered into the File Window, WDFSim must be told to form the interconnections between the building blocks in its internal representation of the network. This process is invoked by selecting the **Build** command from the **File** menu. Once the program has successfully built its internal representation, the **Simulate** menu becomes active and the filter's response may be simulated.

Simulation is invoked by selecting one of the simulation types from the **Simulation** menu. A dialog box, identical to the one shown in Fig. 2.5, then appears asking for the desired type of number system to be used for the time domain simulation. Three types of number systems are available:

1. **Nominal** - full precision (96-bit floating point) representation for signals and coefficients;
2. **Quantized** - b-bit binary numbers for coefficients and full precision signals or b-bit binary numbers for signals and coefficients; and
3. **MMRNS** - multiple modulus residue number system representation for signals and coefficients with moduli set $\{2^n - 1, 2^n, 2^n + 1\}$.

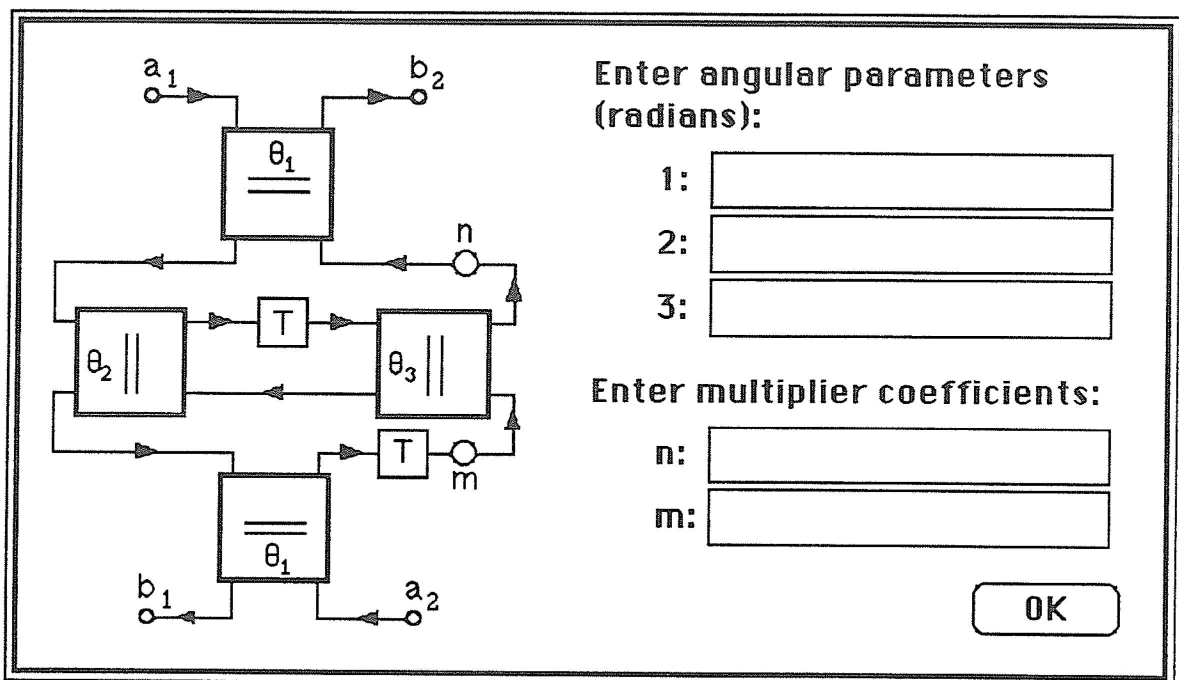


Fig. 2.3: A typical multiplier dialog box.

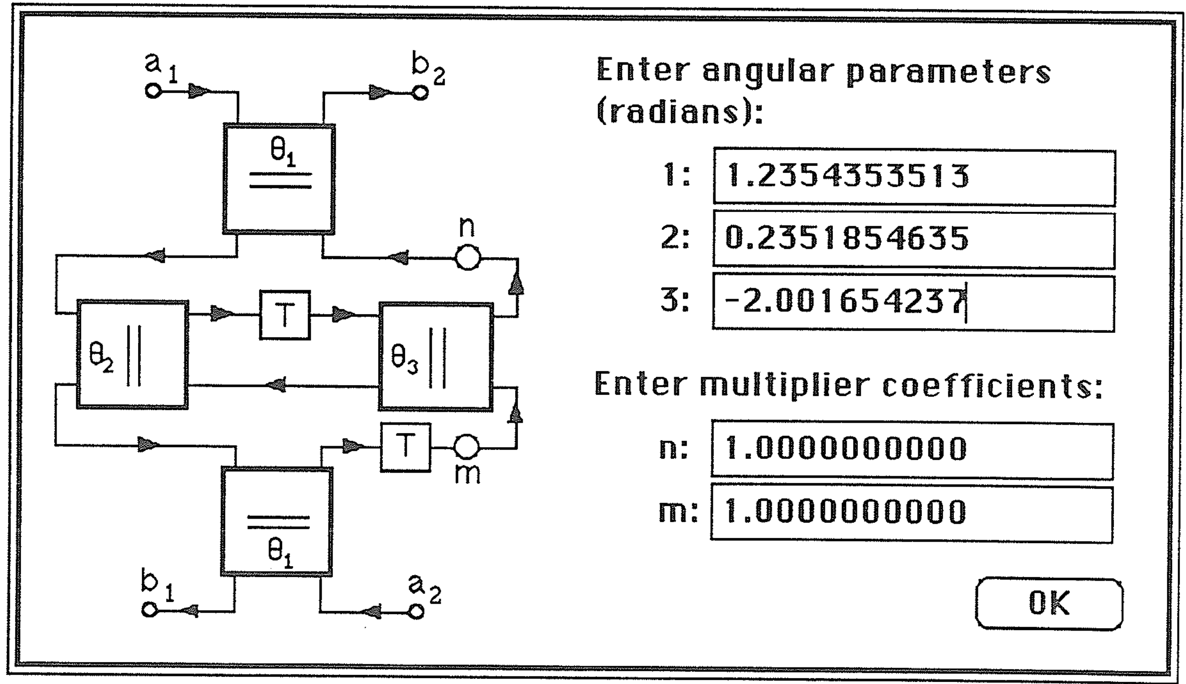


Fig. 2.4: The multiplier dialog of Fig. 2.3 after option-click.

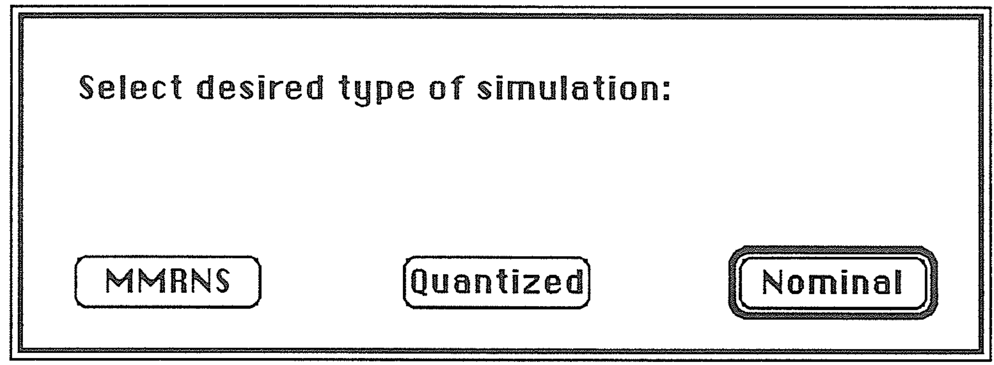


Fig. 2.5: Number system dialog box.

Choosing the **Quantized** option causes the dialog box shown in Fig. 2.6 to appear. Quantization of just multipliers or of both multipliers and signals may be selected. If the latter is chosen, the arithmetic section of the dialog box becomes active and the two's complement fixed-point binary arithmetic scheme may be configured. The possible choices are: overflow or saturation addition; and rounding, truncation, or magnitude truncation multiplication. The two other parameters that must be entered are b - the total number of bits, and c - the number of bits to the left of the decimal point (including sign bit). Note that c must be greater than or equal to one. Once the quantization scheme has been fully configured, click the dialog's **OK** button.

If the **MMRNS** number system is chosen for the simulation, the dialog box shown in Fig. 2.7 appears. The parameters that must be entered to configure the MMRNS are n - the exponent in the moduli set, and c - the number of bits (including sign bit) to the left of the decimal point in the two's complement representation of coefficients and signals before they are mapped into the MMRNS. The MMRNS has a dynamic range of approximately $3n$ bits. Click the dialog's **OK** button once the MMRNS has been fully configured.

After selecting the **Nominal** number system, or after the quantization/MMRNS scheme has been configured, a dialog box appears prompting the user to select a file in which to save the filter's output. An identical dialog box appears for every node in the File Window that contains a sink. If the **user** simulation type is selected, a dialog box appears asking for an input file for every node that contains a source *before* the output files are requested.

Once the input and output files have been specified a dialog box appears asking for the number of samples to be used in calculating the time domain response. If **frequency** is the chosen simulation type, the number of points in the frequency response will be the smallest power of two greater than or equal to the number entered. If **narrowband** is the chosen simulation type, the number entered is the number of points used in the impulse response. Further dialog boxes then appear requesting the minimum and maximum frequencies, and the number of frequency domain points to be calculated.

Select Desired Quantization Scheme:

Quantize: ◉ Multipiers only

○ Multipiers and signals

Addition:

○ Overflow

○ Saturation

Multiplication:

○ Magnitude Truncation

○ Truncation

○ Rounding

Bit Pattern:

[ ] - total number of bits

[ ] - number of integer bits including sign bit (must be >= 1)

OK

Fig. 2.6: Quantization configuration dialog box.

Select Desired MMRNS Scheme:

Moduli Set: {K - 1, K, K + 1} where $K = 2^n$

Dynamic range of MMRNS is approximately 3n

[ ] - n

[ ] - number of integer bits including sign bit (must be >= 1)

OK

Fig. 2.7: MMRNS configuration dialog box.

The results of the simulation are presented in a graphical manner. Time responses are plotted amplitude versus sample number using 'needle' graphs (see Fig. 2.8). Frequency responses are plotted as magnitude, phase, attenuation, and group delay all versus normalized frequency, i.e., $\omega_n = \frac{\omega T}{\pi}$ where ω_n is the normalized frequency, ω is the actual digital frequency, and T is the sampling interval. Note that $T = 1\ s$ is assumed with no loss of generality. See Fig. 2.9 for an example of a plot of an attenuation response.

Response plots may be manipulated using the commands under the **Plot** menu. Options include the ability to change the ranges of either the independent or dependent variables, to set the number of horizontal or vertical divisions, to turn a grid on or off, and to save the plot as a PICT file which may then be printed by applications such as MacDraw, MacPaint, or SuperPaint.

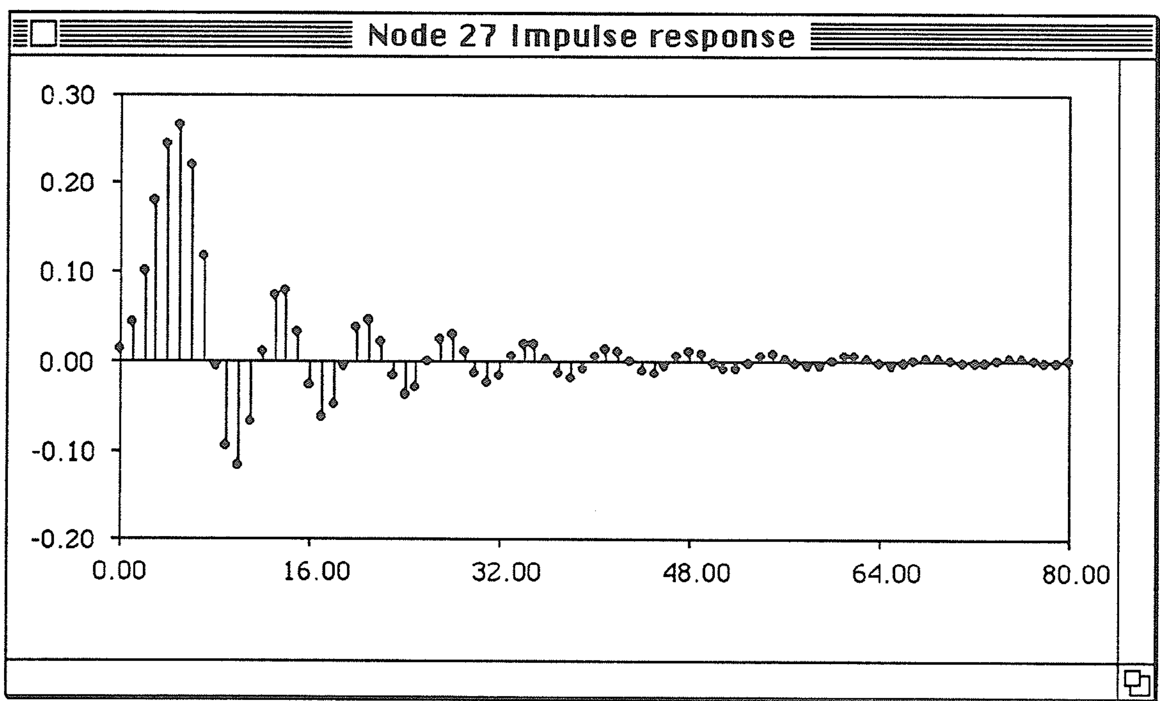


Fig. 2.8: Impulse response of a sample filter.

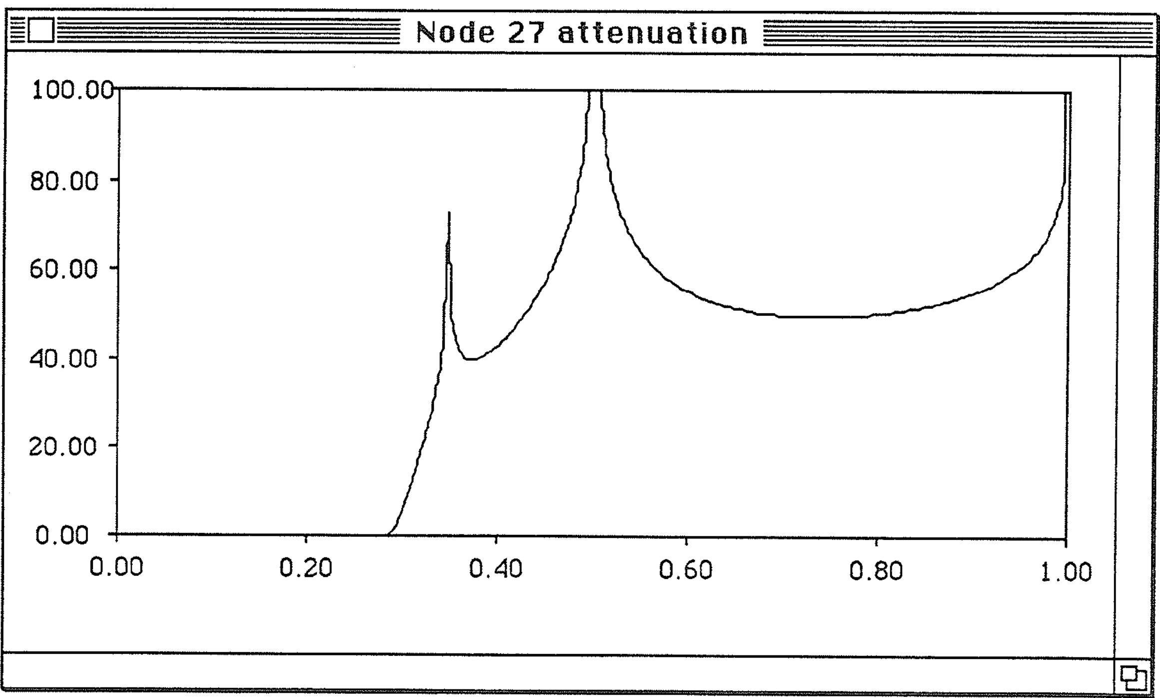


Fig. 2.9: Attenuation response of a sample filter.

III. MODELLING OF WDF STRUCTURES

Before any type of simulation can be performed, computer models of the physical system must be designed and implemented. For modelling purposes, wave digital filters may be considered to be a topological connection of WD building blocks. Two types of modelling are therefore required: modelling of the individual building blocks, and modelling of the network topology.

3.1 WDF Building Blocks

In this section the characteristic equations which describe the input-output relationships of the WDF building blocks and their analog counterparts are presented. No derivation of these relationships is attempted or implied. For complete derivations the reader is referred to [2] for Sections 3.1.1 and 3.1.2 and to [4] for Section 3.1.3.

3.1.1 Elements and Sources

3.1.1a One-Port Terminations

Four one-port terminations have been implemented as building blocks in the WD domain. They are: a delay, a delay with an inverter, a sink, and a sink with a source, which correspond to a capacitance, an inductance, a resistance, and a resistive source, respectively, in the ψ domain. Table 3.1 summarizes the one-port mappings and their characteristic equations in both the ψ and WD domains.

3.1.1b Ideal Transformer

An ideal transformer and its WD equivalents for two special cases of port resistances are shown in Table 3.2. For the case $R_1 = n^2R_2$ the ideal transformer translates to a pair of inverse multipliers in the WD domain. For the case $R_1 = R_2$ the WD equivalent of the ideal transformer is the normalized two-port adapter. See [4] for a derivation of both relationships.

3.1.1c Three-Port and Four-Port Circulators

Wave digital equivalents of three- and four-port circulators are also implemented as filter building blocks. Table 3.3 summarizes the ψ to WD domain mappings and the WD

characteristic equations.

Table 3.1: One-port elements and sources.

	Capacitance	Inductance	Resistance	Resistive Source
ψ Domain	I, +, V, -, $C=\frac{1}{R}$	I, +, V, -, L=R	I, +, V, -, R	I, +, V, -, R, E
	$V=IR/\psi$	$V=IR\,\psi$	$V=IR$	$V=IR+E$
WD Domain	A, R, T, B	A, R, T, B, -1	A, R, B, 0	A, R, B, E
	$B=z^{-1}A$	$B=-z^{-1}A$	$B=0$	$B=E$

Table 3.2: Ideal transformer.

ψ Domain	$R_1=n^2R_2\equiv R$	$R_1=R_2\equiv R$
I_1, I_2, +, +, V_1, n, 1, V_2, –, –, R_1, R_2	A_1, 1/n, B_2, R, n, R, B_1, A_2	A_1, B_2, θ, B_1, A_2, $\cos\theta=\frac{1-n^2}{1+n^2}$, $\sin\theta=\frac{2n}{1+n^2}$
$V_1=nV_2,\quad I_2=-nI_1$	$B_1=nA_2,\quad B_2=\frac{1}{n}A_1$	$\begin{bmatrix}B_1\\B_2\end{bmatrix}=\begin{bmatrix}-\cos\theta & \sin\theta\\ \sin\theta & \cos\theta\end{bmatrix}\begin{bmatrix}A_1\\A_2\end{bmatrix}$

Table 3.3: Three- and four-port circulators.

	Three-Port	Four-Port
ψ Domain		
WD Domain		
	$B_1 = A_3, B_2 = A_1, B_3 = A_2$	$B_1 = A_4, B_2 = A_1, B_3 = A_2, B_4 = A_3$

3.1.2 Interconnections and Adaptors

Adaptors are the WD equivalent of interconnections (topological connections) in the analog domain. The role of adaptors is to preserve the equivalent of Kirchhoff's voltage and current laws in the WD domain. Two types of adaptors are implemented as filter building blocks: parallel adaptors and series adaptors, both of which may or may not have a reflection-free port. Reflection-free ports have the property that the reflected wave is independent of the incident wave at that port, and are required for computability [2]. Parallel (series) adaptors with and without a reflection-free port are described in Table 3.4 (3.5).

Adaptors without a reflection-free port have been implemented for n = 2, 3, and 4, where n is the number of ports. In all cases port n is chosen as the dependent port. Adaptors with a reflection-free port have been implemented for n = 3 and 4. In all cases port n is chosen to be reflection-free and port n - 1 is chosen as the dependent port.

Table 3.4: Parallel adaptors.

ψ Domain	W/O Reflection-Free Port	W/ Reflection-Free Port
$V_1 = V_2 = V_3 = \cdots = V_n$ $I_1 + I_2 + I_3 + \cdots + I_n = 0$	$G_k = \frac{1}{R_k}, \; k = 1,2,\ldots,n$ $G_0 = G_1 + G_2 + \cdots + G_n$ $\gamma_k = \frac{2G_k}{G_0}, \; k = 1,2,\ldots,n\text{-}1$ $B_n = A_n - \sum_{k=1}^{n-1} \gamma_k (A_n - A_k)$ $B_k = B_n + A_n - A_k,$ $k=1,2,\ldots n\text{-}1$	$G_k = \frac{1}{R_k}, \; k = 1,2,\ldots,n$ $G_0 = G_1 + G_2 + \cdots + G_{n\text{-}1}$ $\gamma_k = \frac{G_k}{G_0}, \; k = 1,2,\ldots,n\text{-}2$ $B_0 = -\sum_{k=1}^{n-2} \gamma_k (A_{n-1} - A_k)$ $B_{n\text{-}1} = B_0 + A_n, \; B_n = B_0 + A_{n\text{-}1}$ $B_k = B_{n\text{-}1} + A_{n\text{-}1} - A_k,$ $k=1,2,\ldots n\text{-}2$

Table 3.5: Series adaptors.

ψ Domain	W/O Reflection-Free Port	W/ Reflection-Free Port
$V_1 + V_2 + \cdots + V_n = 0$ $I_1 = I_2 = \cdots = I_n$	$R_0 = R_1 + R_2 + \cdots + R_n$ $\gamma_k = \frac{2R_k}{R_0}, \; k=1,2,\ldots,n\text{-}1$ $A_0 = A_1 + A_2 + \cdots + A_n$ $B_k = A_k - \gamma_k A_0, \; k=1,2,\ldots,n\text{-}1$ $B_n = -(B_1 + \cdots + B_{n\text{-}1} + A_0)$	$R_0 = R_1 + R_2 + \cdots + R_{n\text{-}1}$ $\gamma_k = \frac{2R_k}{R_0}, \; k=1,2,\ldots,n\text{-}2$ $B_n = -(A_1 + A_2 + \cdots + A_{n\text{-}1})$ $A_0 = A_n - B_n$ $B_k = A_k - \gamma_k A_0, \; k=1,2,\ldots,n\text{-}2$ $B_{n\text{-}1} = -(B_1 + \cdots + B_{n\text{-}2} + A_n)$

3.1.3 Elementary Two-Port Sections

First- and second-order reciprocal and non-reciprocal sections as well as a third-order quasi-lattice section, all of which have been derived in [4], are also available as building blocks in WDFSim. All the elementary two-port sections are constructed using only normalized two-port adaptors, inverters, and delays. In [4] (and here), all two-port sections are characterized by three canonic polynomials f, g, and h and a unimodular constant σ. Such a characterization is known as *Belevitch's representation* and is described fully in [3], however, a brief summary follows.

The coefficients of the canonic polynomials in the ψ domain are expressed in terms of the triplet $\{\psi_i, \rho(\psi_i), d(\psi_i)\}$, where ψ_i is the location of the transmission zero that the section realizes, $\rho(\psi_i)$ is the value of the reflectance evaluated at the transmission zero, and $d(\psi_i)$ is the value of the return group delay at the transmission zero. In the WD domain, the corresponding triplet is $\{z_i, \rho(z_i), \delta(z_i)\}$, where $z_i = (1 + \psi_i)/(1 - \psi_i)$, $\rho(z_i) = \rho(\psi_i)$, and $d(\psi_i)$ and $\delta(z_i)$ are related by a constant [4].

3.1.3a Belevitch's Representation Of Lossless Two-Port Networks

Consider the resistively terminated lossless two-port network and its WD equivalent shown in Fig. 3.1. Such a network may be characterized using scattering variables, i.e.,

$$\begin{bmatrix} B_1 \\ B_2 \end{bmatrix} = \begin{bmatrix} S_{11} & S_{12} \\ S_{21} & S_{22} \end{bmatrix} \begin{bmatrix} A_1 \\ A_2 \end{bmatrix} = \mathrm{S} \begin{bmatrix} A_1 \\ A_2 \end{bmatrix} \tag{3.1}$$

where

$$A_k = \frac{V_k + R_k I_k}{2\sqrt{R_k}}, \quad B_k = \frac{V_k - R_k I_k}{2\sqrt{R_k}}, \quad k=1,2 \tag{3.2}$$

are known as the incident and reflected *power* waves, respectively, and the 2x2 matrix S is called the *scattering matrix* [3]. Note that Jarmasz [4] chooses to use power rather than voltage wave quantities to describe the scattering parameters.

In [3], Belevitch showed that the scattering matrix S in (3.1) can be expressed using only three polynomials and a unimodular constant. In particular, S takes on the form

$$\mathrm{S} = \frac{1}{g} \begin{bmatrix} h & \sigma f_* \\ f & -\sigma h_* \end{bmatrix}, \tag{3.3}$$

where the polynomials f, g, and h must satisfy the following conditions for realizability:

1. Polynomials f, g, and h are real in the complex frequency variable $\psi \leftrightarrow z^{-1}$ and

the lower asterisk denotes paraconjugation. For real polynomials paraconjugation is defined in the ψ domain by

$$f_*(\psi) \equiv f(-\psi) \text{ and } h_*(\psi) \equiv h(-\psi),$$

and in the WD domain by

$$f_*(z^{-1}) \equiv z^{-m} f(z) \text{ and } h_*(z^{-1}) \equiv z^{-m} h(z),$$

where $m = \deg g(z)$.

2. The polynomial $g(\psi)$ has all its zeros strictly in the left-half plane. Correspondingly, the zeros of $g(z)$ lie strictly within the unit circle.

3. For real two-ports the unimodular constant σ is either +1 or −1.

4. Polynomials f, g, and h must satisfy the Feldkeller equation

$$\left|\frac{f}{g}\right|^2 + \left|\frac{h}{g}\right|^2 = 1,$$

for $\psi = j\phi$ where $\phi = \tan\left(\frac{\omega T}{2}\right)$ in the ψ domain and for $z^{-1} = e^{-j\omega T}$ in the WD domain.

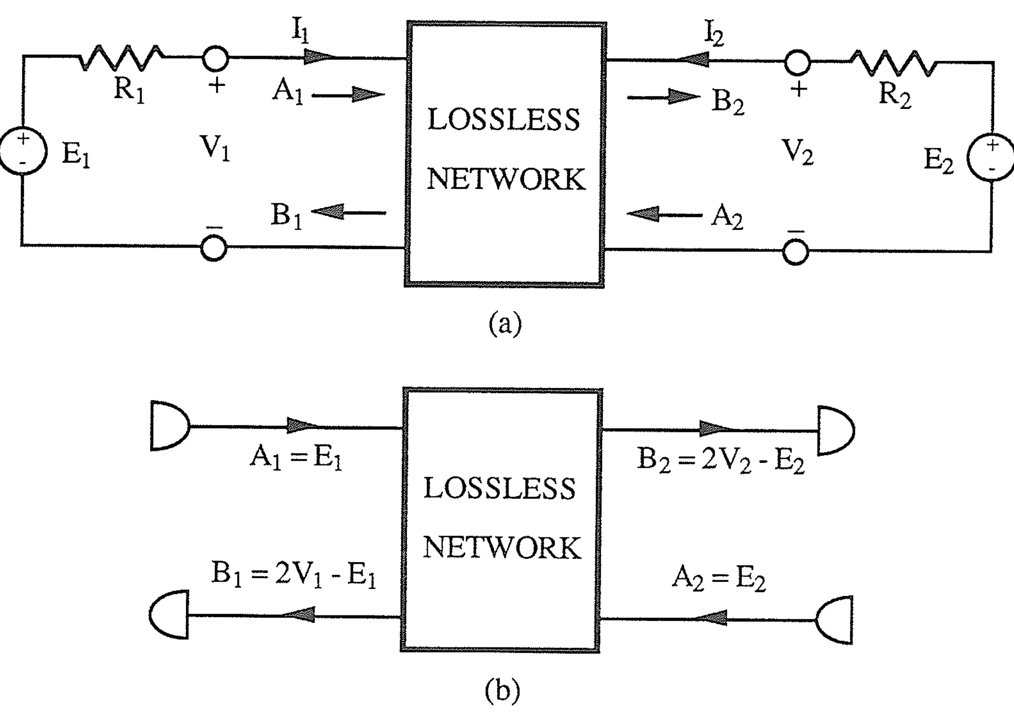


Fig. 3.1: (a) A lossless analog two-port network connected between resistive terminations, and (b) its wave digital equivalent.

3.1.3b First-Order Sections

Four reciprocal first-order sections (types **1A**, **1B**, **1C**, and **1D**) and one non-reciprocal first-order section (type **1E**) have been implemented as building blocks. The four reciprocal sections realize transmission zeros at $\psi = 0 \leftrightarrow z^{-1} = 1$ or at $\psi = \infty \leftrightarrow z^{-1} = -1$ and have a reflectance $\rho = \pm 1$ at the transmission zero. The non-reciprocal section **1E** realizes a transmission zero at $\psi = -r \leftrightarrow z^{-1} = -\varphi$ with a reflectance ρ at the zero. Tables 3.6 to 3.10 summarize the structure and the characteristic polynomials of the five first-order sections. Note that for all first-order sections port 2 has been chosen to be reflection-free.

3.1.3c Second-Order Sections

Four reciprocal second-order sections (types **2A**, **2B**, **2C**, and **2D**) and one non-reciprocal second-order section (type **2E**) have been implemented as building blocks. Sections **2A** and **2B** realize a pair of transmission zeros at $\psi = \pm j\phi_0 \leftrightarrow z = e^{\pm j\omega_0 T}$ and have a reflectance $\rho = \pm 1$ at the transmission zero. Section **2C** realizes a pair of transmission zeros at $\psi = \pm r \leftrightarrow z^{-1} = \varphi^{\pm 1}$ with a reflectance $\pm\rho^{\pm 1}$. Section **2D** realizes a pair of transmission zeros at $\psi = \pm j\phi_0 \leftrightarrow z = e^{\pm j\omega_0 T}$ with a reflectance $\rho = e^{\pm j\alpha}$. The non-reciprocal section **2E** realizes a transmission zero at $\psi = r\, e^{\pm j\phi_0} \leftrightarrow z^{-1} = k\, e^{\pm j\omega_0 T}$ with a reflectance $\rho = \beta\, e^{\pm j\alpha}$ at the transmission zero. Tables 3.11 to 3.16 summarize the structure and the characteristic polynomials of the five second-order sections. Note that for all second-order sections port 2 has been chosen to be reflection-free.

3.1.3c Third-Order Sections

Only one third-order section, a quasi-lattice section (types **3QL**), has been implemented. Its structure and characteristic polynomials are given in Table 3.17 (for the WD domain only). Note that section **3QL** has no reflection-free port.

Table 3.6: First-order reciprocal section **1A**.

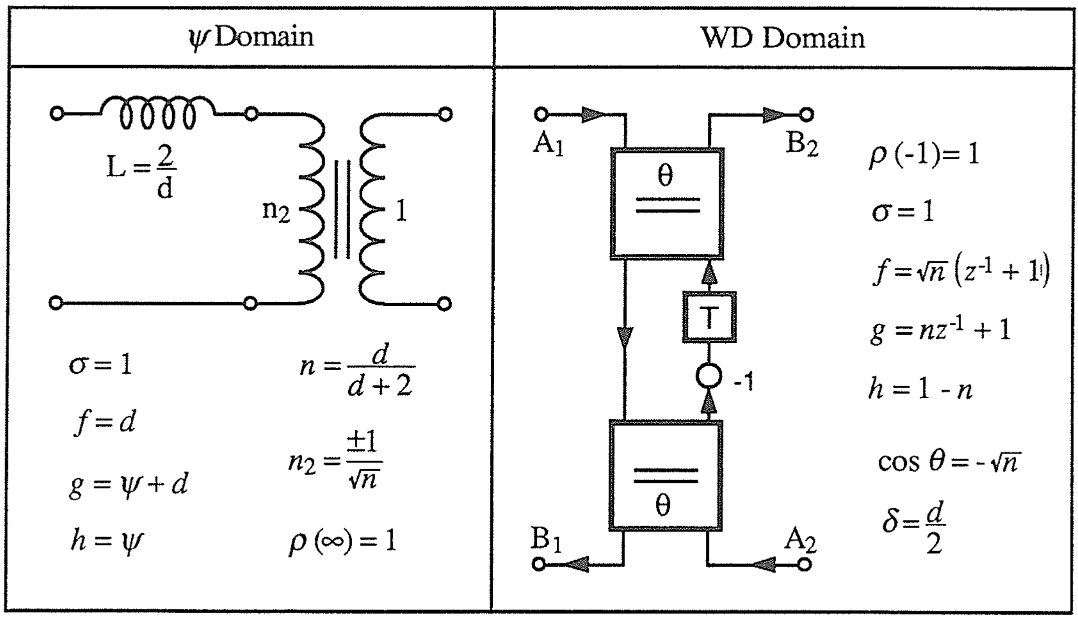


Table 3.7: First-order reciprocal section **1B**.

ψ Domain		WD Domain
$C = \frac{2}{d}$, n_2 : 1		A_1, B_2, B_1, A_2, θ, T, -1
$\sigma = 1$	$n = \frac{d}{d+2}$	$\rho(-1) = -1$
$f = d$	$n_2 = \pm\sqrt{n}$	$\sigma = 1$
$g = \psi + d$		$f = \sqrt{n}\left(z^{-1} + 1\right)$
$h = -\psi$	$\rho(\infty) = -1$	$g = nz^{-1} + 1$
		$h = n - 1$
		$\cos\theta = -\sqrt{n}$
		$\delta = \frac{d}{2}$

Table 3.8: First-order reciprocal section **1C**.

ψ Domain	WD Domain
$C = \frac{d}{2}$ n_2 1 $\sigma = -1$ $f = d\psi$ $g = d\psi + 1$ $h = 1$ $n = \frac{d}{d+2}$ $n_2 = \frac{\pm 1}{\sqrt{n}}$ $\rho(0) = 1$	A_1 B_2 θ T θ B_1 A_2 $\rho(1) = 1$ $\sigma = -1$ $f = \sqrt{n}\left(z^{-1} - 1\right)$ $g = nz^{-1} - 1$ $h = n - 1$ $\cos\theta = -\sqrt{n}$ $\delta = \frac{d}{2}$

Table 3.7: First-order reciprocal section **1D**.

ψ Domain	WD Domain
$L = \frac{d}{2}$ n_2 1 $\sigma = -1$ $f = d\psi$ $g = d\psi + 1$ $h = -1$ $n = \frac{d}{d+2}$ $n_2 = \pm\sqrt{n}$ $\rho(0) = -1$	A_1 B_2 θ -1 T -1 θ B_1 A_2 $\rho(1) = -1$ $\sigma = -1$ $f = \sqrt{n}\left(z^{-1} - 1\right)$ $g = nz^{-1} - 1$ $h = 1 - n$ $\cos\theta = -\sqrt{n}$ $\delta = \frac{d}{2}$

Table 3.10: First-order non-reciprocal section 1E.

ψ Domain	WD Domain
$C = \frac{1}{Rr}$, R, n_2, 1	A_1, B_2, θ_2, T, θ_1, B_1, A_2
$\sigma = -1$	$\frac{h}{g}(-\varphi) = \rho$
$f = \psi + r$	$\cos \gamma_2 = \frac{-\rho(1+\varphi)}{1+\rho^2\varphi}$
$g = \psi + \frac{r(\rho^2+1)}{\rho^2 - 1}$	$\cos \theta_1 = \sqrt{\frac{1-\rho^2}{1-\varphi^2\rho^2}}$
$h = \frac{2r\rho}{\rho^2 - 1}$	$\sin \theta_2 = \sqrt{\frac{1-\varphi^2}{1-\varphi^2\rho^2}}$
$R = \frac{\rho+1}{\rho-1}$	$\sin \theta_1 = \rho \sin \theta_2$
$\frac{h}{g}(-r) = \rho$	$\cos \theta_2 = \varphi \cos \theta_1$
$\varphi = \frac{r+1}{r-1}$	$f = (\cos \theta_1)\, z^{-1} + \cos \theta_2$
$n_2 = \tan^2\left(\frac{\gamma_2}{2}\right)$	$g = (\cos \theta_1)(\cos \theta_2)\, z^{-1} + 1$
	$\sigma = -1$
	$h = (\sin \theta_1)(\sin \theta_2)$

Table 3.11: Second-order reciprocal section 2A.

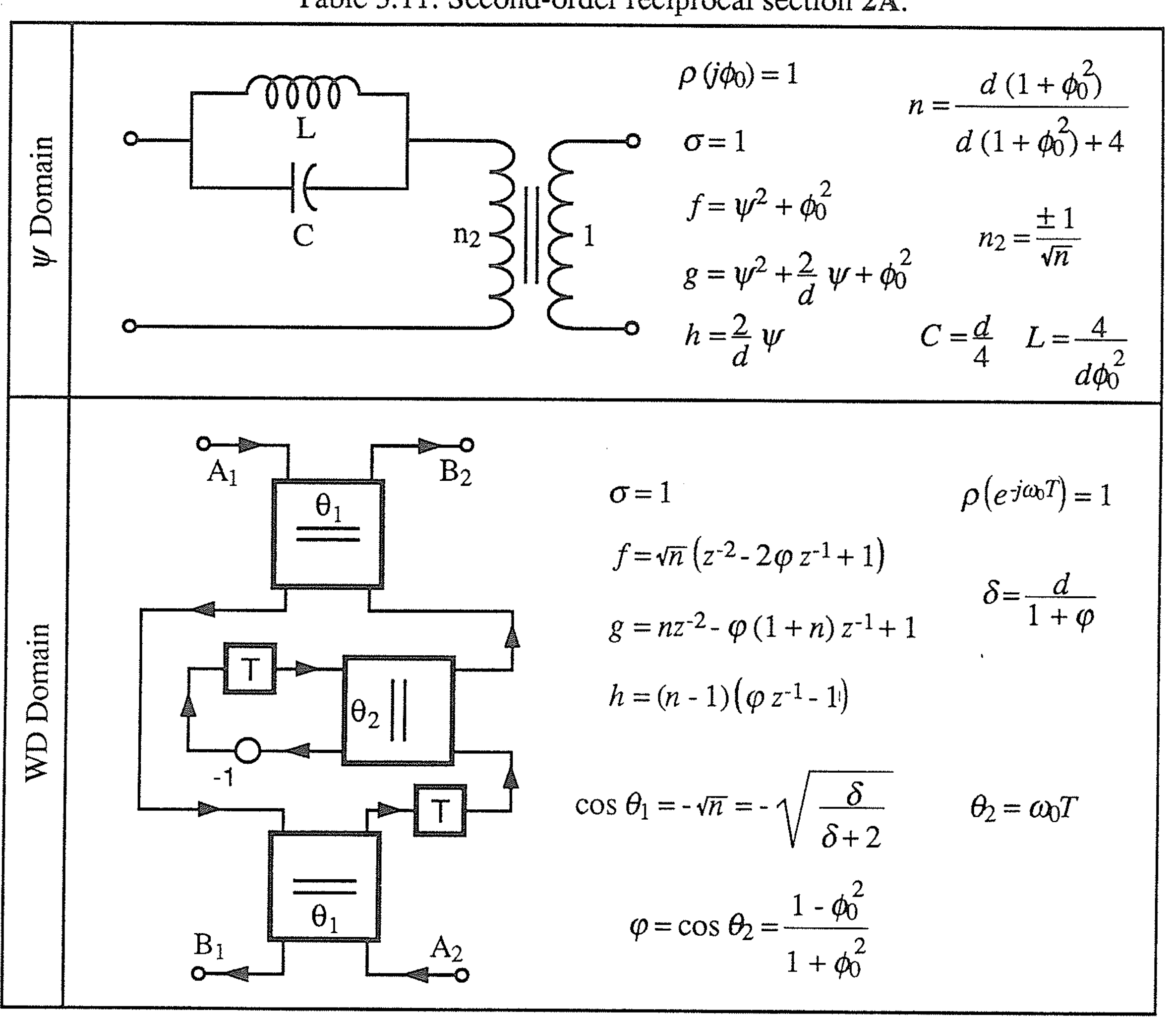


Table 3.12: Second-order reciprocal section 2B.

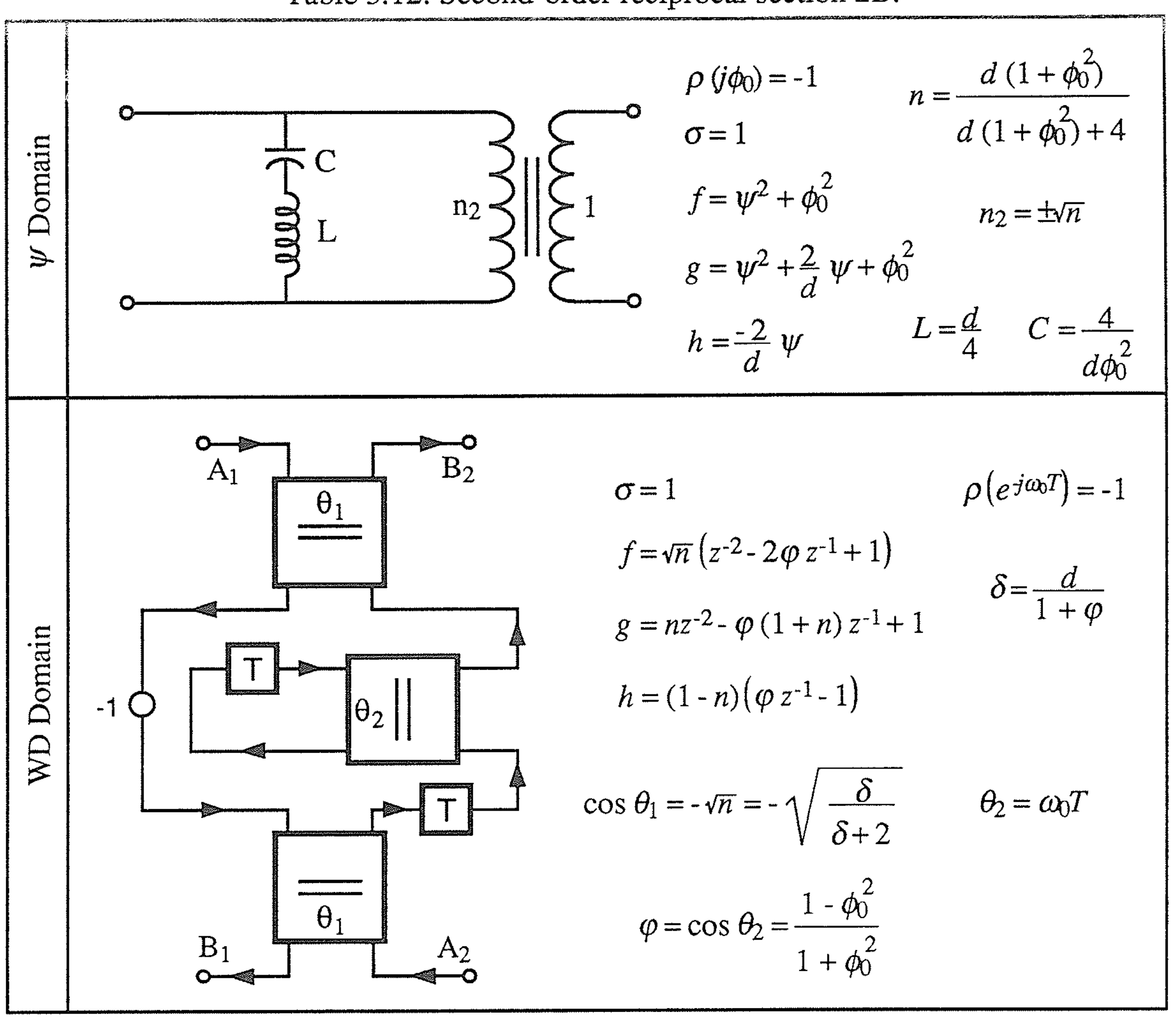


Table 3.13: Second-order reciprocal section **2C**.

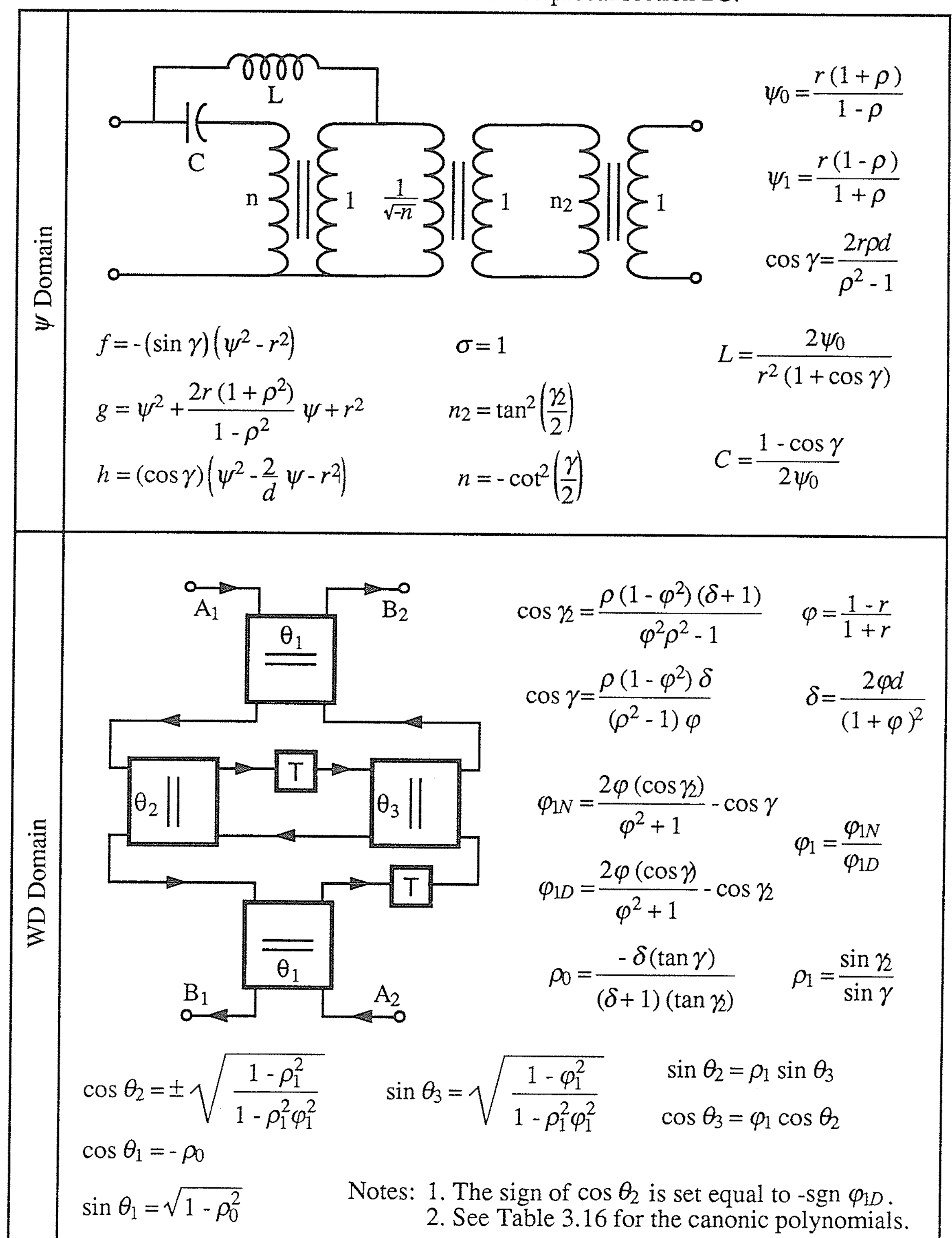


Table 3.14: Second-order reciprocal section **2D**.

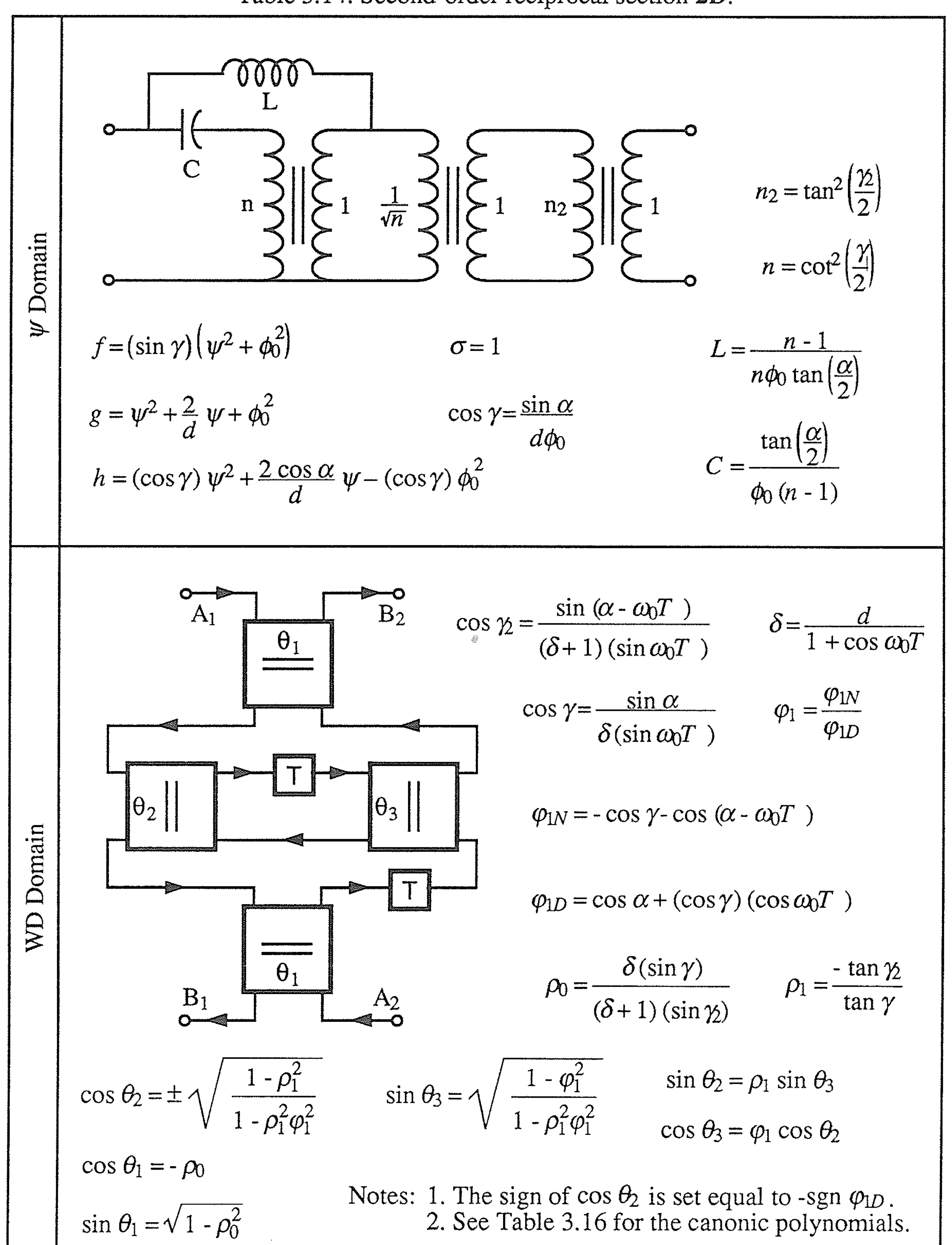
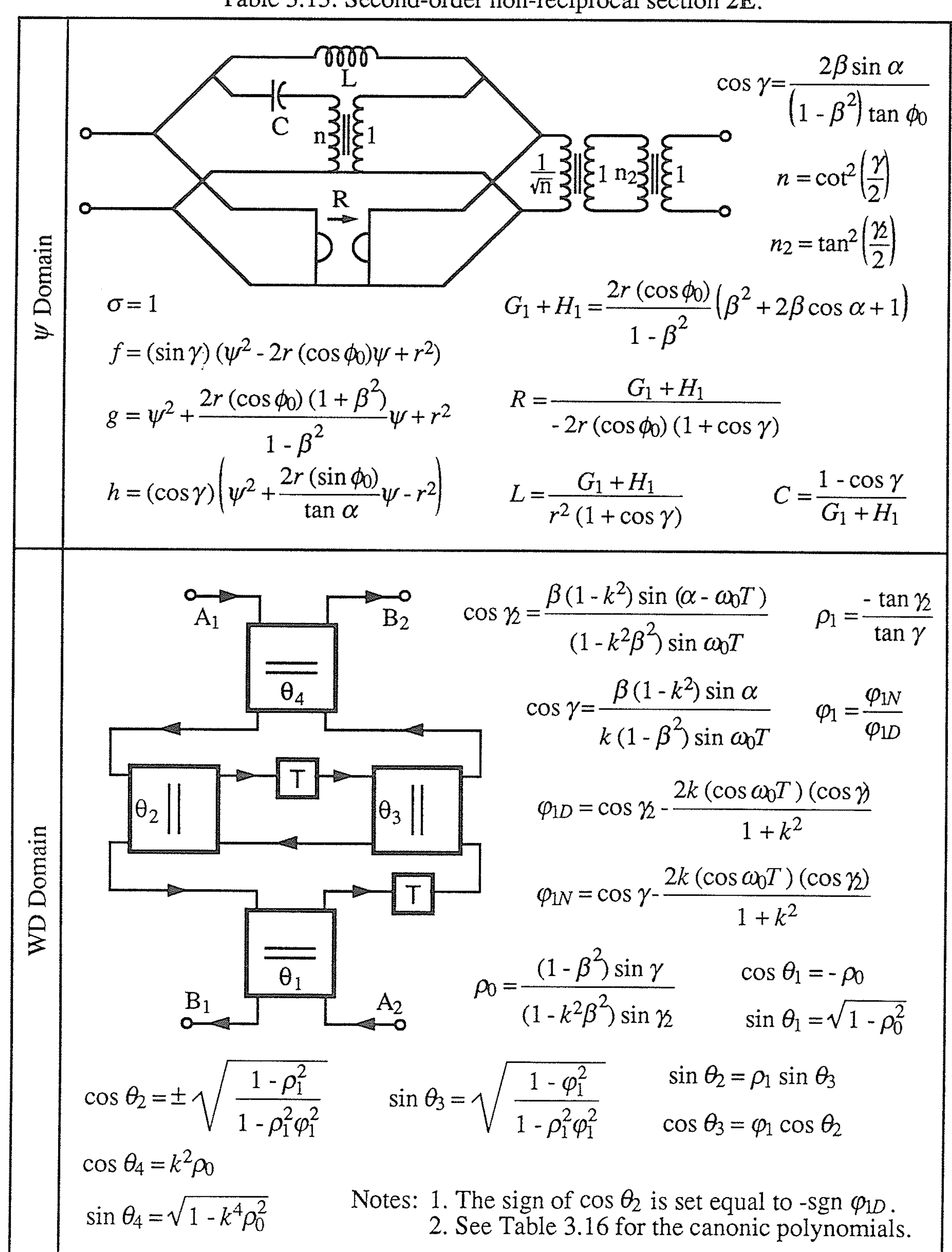


Table 3.15: Second-order non-reciprocal section **2E**.



ψ Domain

$$\cos\gamma = \frac{2\beta\sin\alpha}{\left(1-\beta^2\right)\tan\phi_0}$$

$$n = \cot^2\left(\frac{\gamma}{2}\right)$$

$$n_2 = \tan^2\left(\frac{\gamma_2}{2}\right)$$

$$\sigma = 1$$

$$G_1 + H_1 = \frac{2r(\cos\phi_0)}{1-\beta^2}\left(\beta^2 + 2\beta\cos\alpha + 1\right)$$

$$f = (\sin\gamma)\left(\psi^2 - 2r(\cos\phi_0)\psi + r^2\right)$$

$$g = \psi^2 + \frac{2r(\cos\phi_0)(1+\beta^2)}{1-\beta^2}\psi + r^2$$

$$R = \frac{G_1 + H_1}{-2r(\cos\phi_0)(1+\cos\gamma)}$$

$$h = (\cos\gamma)\left(\psi^2 + \frac{2r(\sin\phi_0)}{\tan\alpha}\psi - r^2\right)$$

$$L = \frac{G_1 + H_1}{r^2(1+\cos\gamma)} \qquad C = \frac{1-\cos\gamma}{G_1 + H_1}$$

WD Domain

$$\cos\gamma_2 = \frac{\beta(1-k^2)\sin(\alpha-\omega_0 T)}{(1-k^2\beta^2)\sin\omega_0 T} \qquad \rho_1 = \frac{-\tan\gamma_2}{\tan\gamma}$$

$$\cos\gamma = \frac{\beta(1-k^2)\sin\alpha}{k(1-\beta^2)\sin\omega_0 T} \qquad \varphi_1 = \frac{\varphi_{1N}}{\varphi_{1D}}$$

$$\varphi_{1D} = \cos\gamma_2 - \frac{2k(\cos\omega_0 T)(\cos\gamma)}{1+k^2}$$

$$\varphi_{1N} = \cos\gamma - \frac{2k(\cos\omega_0 T)(\cos\gamma_2)}{1+k^2}$$

$$\rho_0 = \frac{(1-\beta^2)\sin\gamma}{(1-k^2\beta^2)\sin\gamma_2} \qquad \cos\theta_1 = -\rho_0 \qquad \sin\theta_1 = \sqrt{1-\rho_0^2}$$

$$\cos\theta_2 = \pm\sqrt{\frac{1-\rho_1^2}{1-\rho_1^2\varphi_1^2}} \qquad \sin\theta_3 = \sqrt{\frac{1-\varphi_1^2}{1-\rho_1^2\varphi_1^2}}$$

$$\sin\theta_2 = \rho_1\sin\theta_3 \qquad \cos\theta_3 = \varphi_1\cos\theta_2$$

$$\cos\theta_4 = k^2\rho_0$$

$$\sin\theta_4 = \sqrt{1-k^4\rho_0^2}$$

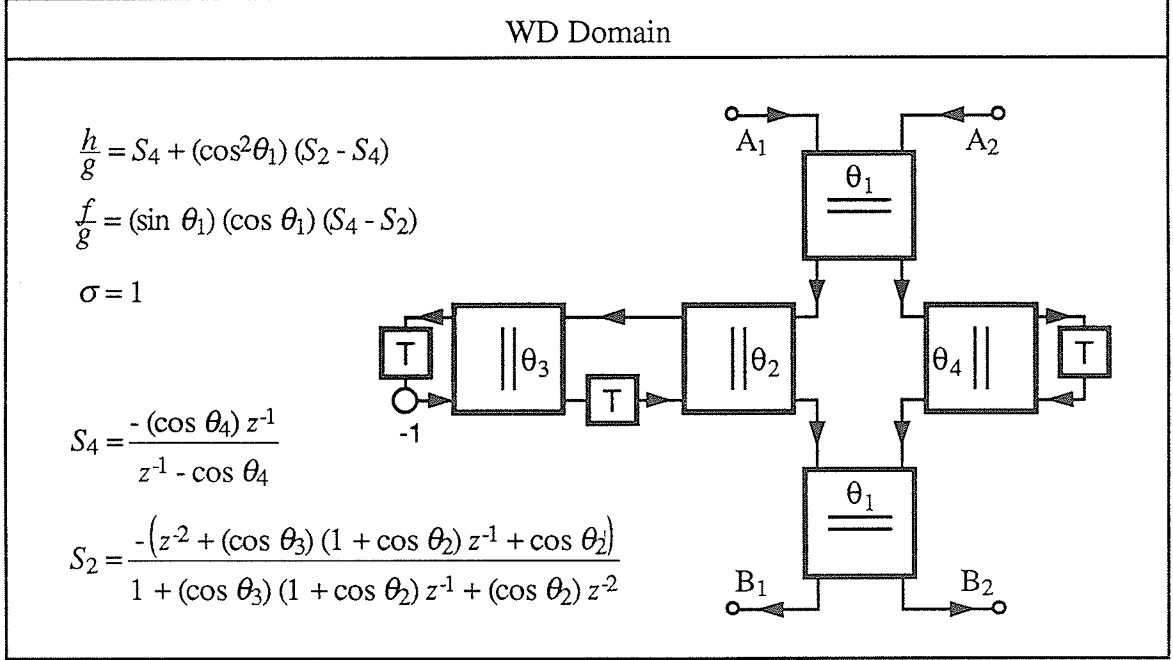
Notes: 1. The sign of $\cos\theta_2$ is set equal to $-\text{sgn}\,\varphi_{1D}$.
2. See Table 3.16 for the canonic polynomials.

Table 3.16: WD canonic polynomials for sections **2C**, **2D**, and **2E**.

$f = -(\cos\theta_1)\, z^{-2} + \frac{a-b}{2}\, z^{-1} + \cos\theta_4$	$\sigma = 1$
$g = -(\cos\theta_1)(\cos\theta_4)\, z^{-2} + \frac{a+b}{2}\, z^{-1} + 1$	$a = (1 - \cos\theta_1)(1 + \cos\theta_4)\cos(\theta_2 - \theta_3)$
$h = -(\sin\theta_1)(\sin\theta_4)(z^{-1}\cos\theta_3 + \cos\theta_2)$	$b = (1 + \cos\theta_1)(1 - \cos\theta_4)\cos(\theta_2 + \theta_3)$

Note: For reciprocal sections **2C** and **2D**, set $\theta_4 = \pi - \theta_1$.

Table 3.17: Third-order quasi-lattice section **3QL**.



$$\frac{h}{g} = S_4 + (\cos^2\theta_1)(S_2 - S_4)$$

$$\frac{f}{g} = (\sin\theta_1)(\cos\theta_1)(S_4 - S_2)$$

$$\sigma = 1$$

$$S_4 = \frac{-(\cos\theta_4)\, z^{-1}}{z^{-1} - \cos\theta_4}$$

$$S_2 = \frac{-\left(z^{-2} + (\cos\theta_3)(1 + \cos\theta_2)\, z^{-1} + \cos\theta_2\right)}{1 + (\cos\theta_3)(1 + \cos\theta_2)\, z^{-1} + (\cos\theta_2)\, z^{-2}}$$

3.2 The Network Data Structure

All information pertaining to one node (or WDF building block) in the network (WDF) is held in the **Node** record structure shown in Fig. 3.2. An array of **Nodes** (called a **Network**) is used to represent the entire structure in the File Window. The information held in each of the particular fields of the **Node** Record is described below.

> *n* - The number of ports the WDF building block has. Note that $n = 0$ for an empty node.
>
> *pict* - The number of the picture that is drawn in the cell. All pictures are stored in the array Picts which is initialized during startup. Note that $pict = 0$ for an empty node.
>
> *nodeType* - The type of WDF building block in the cell. Note that *nodeType* = **none** for an empty node.
>
> *gamma* - The multipliers (if any).
>
> *aval*[*k*] - The input to port *k*. For $k <= 0$, *aval*[*k*] represents the contents of an internal state.
>
> *bval*[*k*] - The output from port *k*. For $k <= 0$, *bval*[*k*] represents the contents of an internal state.
>
> *astat*[*k*] - A boolean flag. Set to true once *aval*[*k*] has been updated since the last sample.
>
> *bstat*[*k*] - A boolean flag. Set to true once *bval*[*k*] has been calculated.
>
> *bnode*[*k*] - The node number to which port *k* is connected.
>
> *bport*[*k*] - The port number of *bnode*[*k*] that port *k* is connected to.

The fields *n*, *pict*, *nodeType*, and *gamma* are filled in as the network is entered to the File Window. Once the entire structure is assembled and the **Build** command is invoked, the fields *bnode* and *bport* are filled in. The fields *aval*, *bval*, *astat*, and *bstat* are used during simulation and are described further in Chapter IV.

```
Gammas = array [1..16] of extended;
Node = record
        n, pict: integer;
        nodeType: (parallel, serial, parallelRF, serialRF, capacitor,
        inductor, IOport, circulator, A0, A1, B1, C1, D1, E1, A2,
        B2, C2, D2, E2, QL3, transformer, none);
        bnode, bport: array [1..Nports] of integer;
        astat, bstat: array [0..Nports] of boolean;
        aval, bval: array [-6..Nports] of extended;
        gamma: Gammas;
end;
Network = array [1..Nnodes] of Node;
```

Fig. 3.2: The **Node** and **Network** type declarations.

3.3 Forming the Internal Connections

In order to form the internal connections (fill in the fields *bnode* and *bstat* in the **Node** records), the program must know the port numbering scheme for every building block. Since each type of building block may have many different orientations (see Fig. 3.3) a reference basis for the port numbers must be used. Consider an arbitrary empty node in Fig. 2.1. Each node has four sides (except edge nodes) where adjoining building blocks may be attached. Label these four sides north (N), south (S), east (E), and west (W) where north is the top of the window. With respect to this basis, the ports of the WDF building blocks shown in Fig. 3.3 may be uniquely coded as shown in Fig. 3.4. (Note that zero is used to indicate that a building block has no port in the given direction).

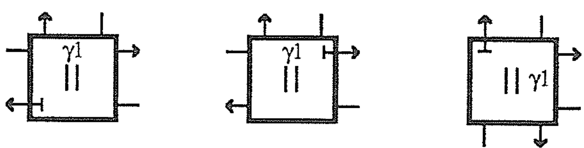


Fig. 3.3: Three possible orientations for a three-port parallel adaptor.

For each element k in the *Picts* array there is a corresponding element k in an array called the *PortCons* array that holds the port numbers with respect to the basis {N, S, E, W}. The array *PortCons* is of type **Connections**, which is defined in Fig. 3.5. During startup, the *PortCons* data structure is read from the resource fork of WDFSim into main memory where it is easily accessed.

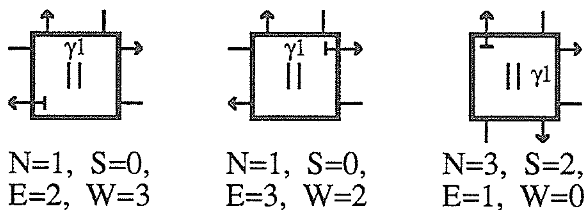


Fig. 3.4: Port numbering scheme with respect to the basis {N, S, E, W}.

The routines responsible for parsing the File Window (network) and building the internal connections are all contained in the **Builders** unit. The **Builders** unit, whose hierarchy is shown in Fig. 3.6, exports one procedure - BuildFile. BuildFile calls function AddNode for every non-empty node in the network. AddNode then calls ConnectE, ConnectW, ConnectS, and ConnectN to form the port connections in each of the four directions. AddNode returns a boolean **true** if the node has been successfully connected.

```
Connection = record
      N, S, E, W: integer;
end;
Connections = array [1..NPicts] of Connection;
```

Fig. 3.5: The **Connection** and **Connections** type declarations.

Once the entire network has been parsed error-free, BuildFile sets the global variable *Built* to **true** and activates the **Simulate** menu. If an error (port mismatch) occurs during the building process, AddNode returns a boolean **false** which tells BuildFile to take the following actions: 1) activate the offending node; 2) beep twice; and 3) terminate the parse.

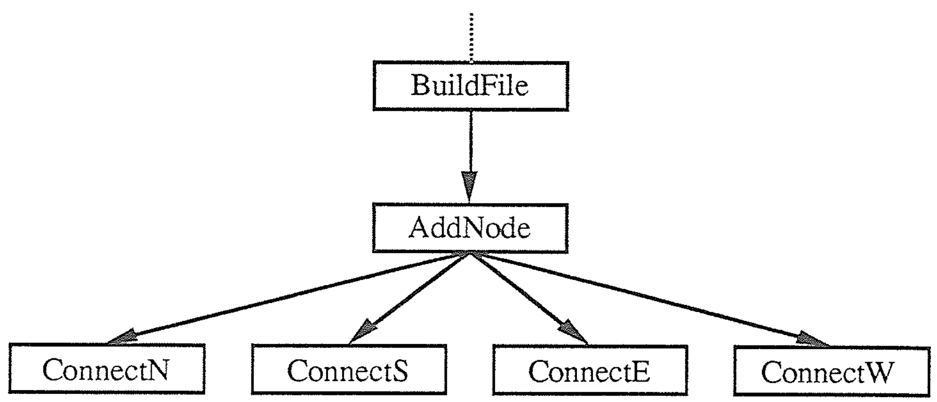


Fig. 3.6: Hierarchy of the **Builders** unit.

IV. GENERAL SIMULATION

Once the filter structure has been successfully parsed by the Build procedure as described in Section 3.3, its response may be simulated. Simulation is invoked by selecting one of the five simulation types under the **Simulate** menu. Since all modelling of WDF building blocks is done based on time domain input/output relationships, all simulations are performed in the time domain. Once completed, the results of the simulation are plotted. All routines used during the simulation are held in a unit called **Simulators**, which uses two other units: **Updators** and **SimTools**. The hierarchy of the **Simulators** unit is shown in Fig. 4.1.

4.1 Time Domain Simulation

Three types of time domain simulations may be selected from the **Simulation** menu: **impulse, step**, and **user**. Simulation types **impulse** and **step** calculate the response of the filter to the standard impulse and step functions. Simulation type **user** allows the user to specify any input signal which is stored as a sequence of samples in a file (see Appendix B for an example).

4.1.1 General Time Domain Simulation Algorithm

All time domain simulations are performed with the same algorithm, the variables being the input files and the number of samples. Furthermore, as will be shown in Section 4.2, the identical algorithm may be used to compute the frequency responses with the modification of only one step. The general simulation algorithm follows.

1. Make a duplicate of the network.

2. Adjust the parameters of the network according to which type of number system has been chosen; i.e. quantize the multipliers if necessary.

3. Get the number of sample points to be used (N).

4. If simulation type **user** was chosen, get the name(s) of the input file(s); otherwise, create file(s) containing either the impulse or step function.

5. Get the name(s) of the output file(s).

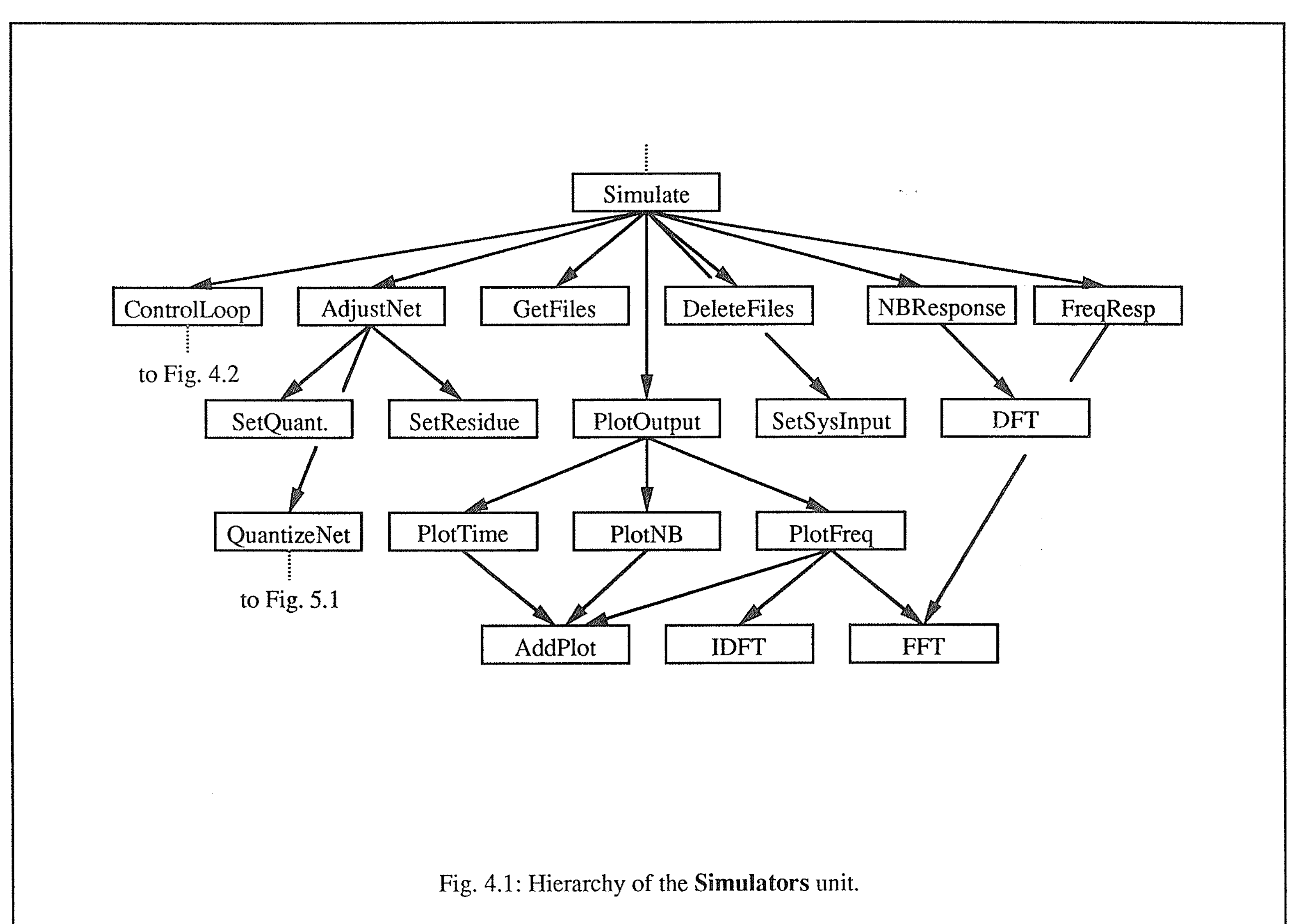


Fig. 4.1: Hierarchy of the **Simulators** unit.

6. Call function ContolLoop to perform the simulation. See Section 4.1.2 for a detailed description of this step.

7. Delete any files created in step 4.

8. If ControlLoop returned **true** (no errors during simulation), plot the results.

9. Restore the network from the copy created in step 1.

4.1.2 The ContolLoop Algorithm

Given the input and output files and the number of samples (N), it is the responsibility of function ControlLoop to perform the simulation. ControlLoop is also required to detect any non-computable structures, in which case it will return boolean **false**. ControlLoop will return boolean **true** once the simulation has successfully been completed. The hierarchy of function ControlLoop and its associated routines is shown in Fig. 4.2. The ControlLoop algorithm follows.

1. Initialize all internal states to zero.

2. Reset all *astat* and *bstat* flags in each non-empty cell's **Node** record.

3. Set $oldCnt = 4000$ (an arbitrarily large number).

4. Call procedure UpdateNet. This step is described in more detail in Section 4.1.3.

5. Count the number of *bstat* flags that remain **false** and set *newCnt* equal to this number.

6. If $oldCnt = newCnt$ the filter is deemed uncomputable. In this case, give a warning message, terminate the simulation, and return **false**.

7. If $newCnt > 0$, set $oldCnt = newCnt$ and go to step 4. Repeat until $newCnt = 0$.

8. Save the output of the filter; i.e., for every node that contains a sink, write *aval*[1] to the corresponding output file.

9. If all the samples have been processed, terminate and return **true**; otherwise, go to step 2.

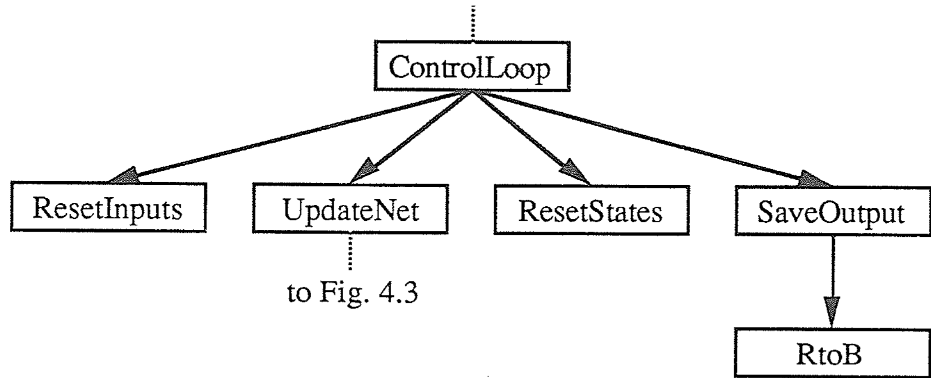


Fig. 4.2: The hierarchy of function ControlLoop and its related routines.

4.1.3 The Updators Unit

The **Updators** unit, whose hierarchy is shown in Fig. 4.3, exports one procedure - UpdateNet. Unit **Updators** also contains a procedure that defines the time domain input-output relationships for every building block described in Section 3.1. When called, UpdateNet traverses once through the network calling the appropriate procedures for each non-empty node. Each procedure is then responsible for:

1. Computing all possible port outputs (*bval*[*k*]s) that have not already been calculated given the available inputs (*aval*[*k*]s); and

2. For every *bval*[*k*] that has been calculated in step 1, do the following:
   (a) Set *bstat*[*k*] = **true**;
   (b) Set *Net*[*bnode*[*k*]].*aval*[*bport*[*k*]] = *bstat*[*k*]; i.e., pass the computed signal to the connecting node; and
   (c) Set *Net*[*bnode*[*k*]].*astat*[*bport*[*k*]] = **true**; i.e., set the *astat* flag of the connecting node.

As an example, consider the cascade of two-port sections **2D** and **1B** as shown in Fig. 4.4(a). Let the **2D** section be in node 14, and the **1B** section be in node 15. The perttinent fields of the **Node** records are shown before updating in Fig. 4.4(b), and after one call to update the **2D** section in Fig. 4.4(c). The Up2D procedure determines that it can calculate *bval*[2] since *bstat*[2] is **false** and *astat*[1] is **true**. After calculating *bval*[2] = 0.735, it sets *bstat*[2] to **true**, *astat*[1] of node 15 to **true**, and *aval*[1] of node 15 to 0.735.

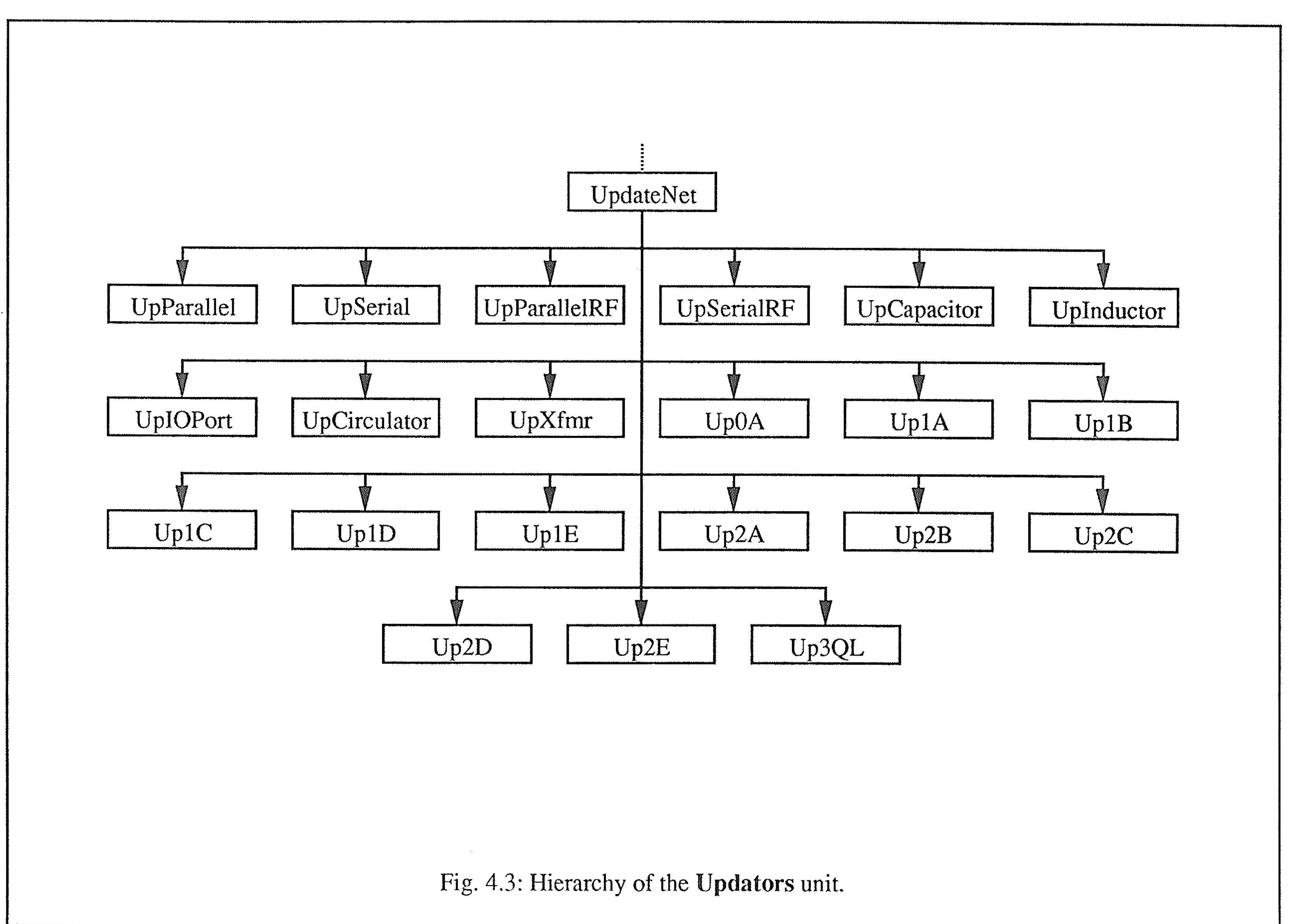


Fig. 4.3: Hierarchy of the **Updators** unit.

In order to comply with the different number systems that may be used during the simulation, all update procedures use the generic addition, multiplication, and negation functions SimSum, SimProd, and SimNeg, respectively. These generic functions, which are held in the **SimTools** unit, determine the appropriate action based on the type of arithmetic being used. The **SimTools** unit also contains the procedures AdjustNet, SetQuantization, and SetResidue (Fig. 4.1) which contol and read the information from the dialog boxes shown in Figures 2.5, 2.6, and 2.7, respectively.

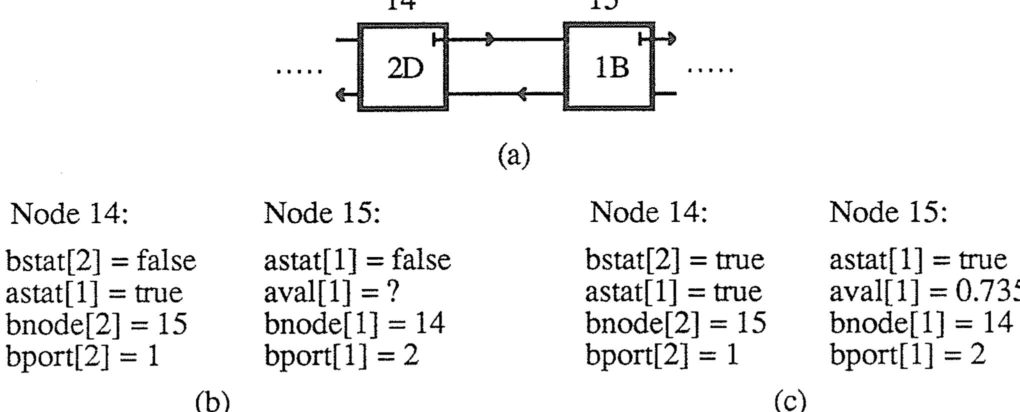


Fig. 4.4: (a) Cascade of two-port sections **2D** and **1B**, (b) fields of the **Node** records before and (c) after updating the **2D** section.

4.2 Frequency Responses

Two types of frequency responses may be selected from the **Simulation** menu: **Frequency** and **Narrowband.** Simulation type **Frequency** calculates the frequency response over the entire normalized frequency range of 0 to 1 (0 to π/T radians). Simulation type **Narrowband** allows a certain frequency range $[\omega_1, \omega_2]$ to be examined with greater resolution by allowing ω_1, ω_2, and M - the number of frequency domain points, to be specified numbers. This feature is useful for obtaining accurate plots of the passband of filters with very narrow bandwidths.

4.2.1 Calculation of the Frequency Response

To compute the **Frequency** response, the algorithm described in Section 4.1.1 is used to compute the N-point impulse response $h[n]$ and step 8 is replaced by the following.

8a. Compute the N-point fast-Fourier transform (FFT) $H(\omega_n)$ of $h[n]$. If N is not a power of two, $h[n]$ will be zero padded to the lowest power of two greater than N before the FFT is computed.

8b. Compute and plot:

(a) the magnitude of $H(\omega_n) \equiv |H(\omega_n)| = \sqrt{\mathrm{Re}^2\{H(\omega_n)\} + \mathrm{Im}^2\{H(\omega_n)\}}$;

(b) the phase of $H(\omega_n) \equiv \theta(\omega_n) = \tan^{-1}\left(\frac{\mathrm{Im}\,\{H(\omega_n)\}}{\mathrm{Re}\,\{H(\omega_n)\}}\right)$;

(c) the attenuation of $H(\omega_n) \equiv \alpha(\omega_n) = -\,20 \log |H(\omega_n)|$; and

(d) the group delay $d(\omega_n)$ (see Section 4.2.3).

4.2.2 Calculation of the Narrowband Response

To compute the **Narrowband** response, the algorithm described in Section 4.1.1 is used to compute the N-point impulse response $h[n]$ and step 8 is replaced by the following.

8a. Get ω_1, ω_2, and M.

8b. Compute the M-point discrete Fourier transform (DFT) $H(\omega_m)$, $m = 0,1,\ldots, M\text{-}1$, of $h[n]$; i.e.,

$$H(\omega_m) = \sum_{n=0}^{N-1} h[n]\, e^{-jn\omega_m} \text{ where } \omega_m = \omega_1 + m\left(\frac{\omega_2 - \omega_1}{M - 1}\right). \tag{4.1}$$

8c. Compute and plot:

(a) the magnitude of $H(\omega_m) \equiv |H(\omega_m)| = \sqrt{\mathrm{Re}^2\{H(\omega_m)\} + \mathrm{Im}^2\{H(\omega_m)\}}$;

(b) the phase of $H(\omega_m) \equiv \theta(\omega_m) = \tan^{-1}\left(\frac{\mathrm{Im}\{H(\omega_m)\}}{\mathrm{Re}\{H(\omega_m)\}}\right)$; and

(c) the attenuation of $H(\omega_m) \equiv \alpha(\omega_m) = -\,20 \log |H(\omega_m)|$.

4.2.3 Calculation of the Group Delay

Calculation of the magnitude, phase, and attenuation of $H(\omega)$ is straightforward; however, computation of the group delay is nontrivial. The group delay $d(\omega)$ is defined by

$$d(\omega) \equiv -\frac{\mathrm{d}}{\mathrm{d}\omega}\,\theta(\omega), \tag{4.2}$$

where $\theta(\omega)$ is the phase spectrum of $H(\omega)$. Rather than trying to perform numerical differentiation, consider the following approach to calculating the group delay. Starting with

$$\frac{H(\omega)}{H^*(\omega)} = e^{2j\,\theta(\omega)}, \tag{4.3}$$

differentiate both side with respect to ω to get

$$\frac{\mathrm{d}}{\mathrm{d}\omega}\left(\frac{H(\omega)}{H^*(\omega)}\right) = 2j\, e^{2j\,\theta(\omega)}\,\frac{\mathrm{d}}{\mathrm{d}\omega}\big(\theta(\omega)\big). \tag{4.4}$$

Solving for $\frac{\mathrm{d}}{\mathrm{d}\omega}\,\theta(\omega)$ gives

$$d(\omega) = -\frac{\mathrm{d}}{\mathrm{d}\omega}\,\theta(\omega) = \frac{-1}{2j}\,\frac{H^*(\omega)}{H(\omega)}\,\frac{\mathrm{d}}{\mathrm{d}\omega}\left(\frac{H(\omega)}{H^*(\omega)}\right), \tag{4.5}$$

which may be written as

$$d(\omega) = \frac{-1}{2j}\,\frac{\mathrm{d}}{\mathrm{d}\omega}\left\{\ln\left(\frac{H(\omega)}{H^*(\omega)}\right)\right\} = \frac{-1}{2j}\,\frac{\mathrm{d}}{\mathrm{d}\omega}\big(\ln H(\omega) - \ln H^*(\omega)\big) \tag{4.6}$$

by applying the (inverse) rule of differentiation of natural logarithms. Applying the linearity property of the differential operator allows equation (4.6) to be expressed as

$$d(\omega) = \frac{-1}{2j}\left(\frac{1}{H(\omega)}\frac{\mathrm{d}}{\mathrm{d}\omega}H(\omega) - \frac{1}{H^*(\omega)}\frac{\mathrm{d}}{\mathrm{d}\omega}H^*(\omega)\right). \tag{4.7}$$

Substitute

$$H(\omega) = \sum_{n=0}^{N-1} h[n]\, e^{-j\omega n}, \tag{4.8}$$

(i.e., the DFT of $h[n]$) and differentiate with respect to ω getting

$$d(\omega) = \frac{-1}{2j}\left(\frac{1}{H(\omega)}\sum_{n=0}^{N-1} -jn\, h[n]\, e^{-j\omega n} - \frac{1}{H^*(\omega)}\sum_{n=0}^{N-1} jn\, h[n]\, e^{j\omega n}\right). \tag{4.9}$$

Now, let

$$G(\omega) = \mathrm{DFT}\,\{n\, h[n]\} = \sum_{n=0}^{N-1} n\, h[n]\, e^{-j\omega n}\,, \tag{4.10}$$

equation (4.9) becomes

$$d(\omega) = \frac{1}{2}\left(\frac{G(\omega)}{H(\omega)} + \frac{G^*(\omega)}{H^*(\omega)}\right) = \mathrm{Re}\left\{\frac{G(\omega)}{H(\omega)}\right\}. \tag{4.11}$$

Hence, to find the group delay from the impulse response $h[n]$ we: 1) compute the sequence $g[n] = n\, h[n]$; 2) compute the FFTs of $h[n]$ and $g[n]$; and 3) compute the delay using equation (4.11).

4.3 The Simulation Record

All parameters that define a distinct type of simulation are held in a global variable of the record type **Simulation** defined in Fig. 4.5.

```
Simulation = record
    simType: (user, impulse, step, frequency, narrowband);
    addType: (Afloating, overflow, saturation, Aresidue);
    multType: (Mfloating, rounding, truncation, magtruncation,
    Mresidue);
    nSamples: integer;
    wmin, wmax: extended;
    RNb, RNbc: longint;
    binary: boolean;
    p1, p2, p3: integer;
    P, scale: longint;
end;
```

Fig. 4.5: The **Simulation** record type declarations.

The information contained in each particular field of the **Simulation** record is described below.

simType - The simulation type.

addType - The type of addition used in the time domain response.

multType - The type of multiplication used in the time domain response.

nSamples - The number of sample points to be computed.

wmin, wmax - the minimum and maximum normalized frequencies (used only in simulation type **NarrowBand**).

RNb - 2^b where b is the number of bits in the quantized simulation.

RNbc - 2^{b-c} where b and c are the total and the integer number of bits, respectively, in the quantized simulation.

binary - A boolean flag set to **true** if two's complement fixed-point binary arithmetic

is to be used during the simulation.

p1, p2, p3 - the moduli set of the MMRNS simulation where $\{p_1, p_2, p_3\} = \{2^n - 1, 2^n, 2^n + 1\}$.

P - $p_1 p_2 p_3 = 2^n (2^{2n} - 1)$, used only in the MMRNS simulation.

scale - The scaling factor in the MMRNS simulation where $scale = P / 2^c$.

4.4 Plotting the Results

The results of the simulation are presented in a graphical manner in the form of plots. For time domain responses, 'needle' plots are used (Fig. 2.8) and for frequency responses, continuous curves are used (Fig 2.9).

4.4.1 Plotting Data Structures

All information pertaining to a particular plot is held in a record of type **PlotRec**, which is defined in Fig. 4.6.

```
PlotPtr = ^PlotRec
PlotRec = record
        ymin, ymax, xmin, xmax: real;
        ydiv, xdiv: integer;
        window: WindowPtr;
        xpts, ypts: PlotArray;
        discrete, grid: boolean;
        next: PlotPtr;
end;
```

Fig. 4.6: The **PlotRec** type declarations.

In particular, the information contained within each field of the **PlotRec** record is described below.

ymin, ymax - The minimum and maximum values to be plotted along the vertical axis. All values outside this range will be 'chopped' off (see Fig. 4.7).

xmin, xmax - The minimum and maximum values to be plotted along the horizontal axis (see Fig. 4.7).

xdiv, ydiv - The number of divisions along the horizontal and vertical axes, respectively. If *xdiv* is set to less than one, it will automatically default to five. If *ydiv* is set less than one, the automatic scaling is enabled (see Fig. 4.7).

window - A window pointer to the window the plot is drawn in.

xpts, ypts - The arrays containing the horizontal and vertical values, respectively, of

the points to be plotted.

discrete - A boolean flag. If *discrete* is **true**, the plot will be drawn using 'needle' points, otherwise a continuous curve is drawn.

grid - A boolean flag. If *grid* is **true,** a grid will be drawn on the plot.

next - A pointer to the next plot in the plot list.

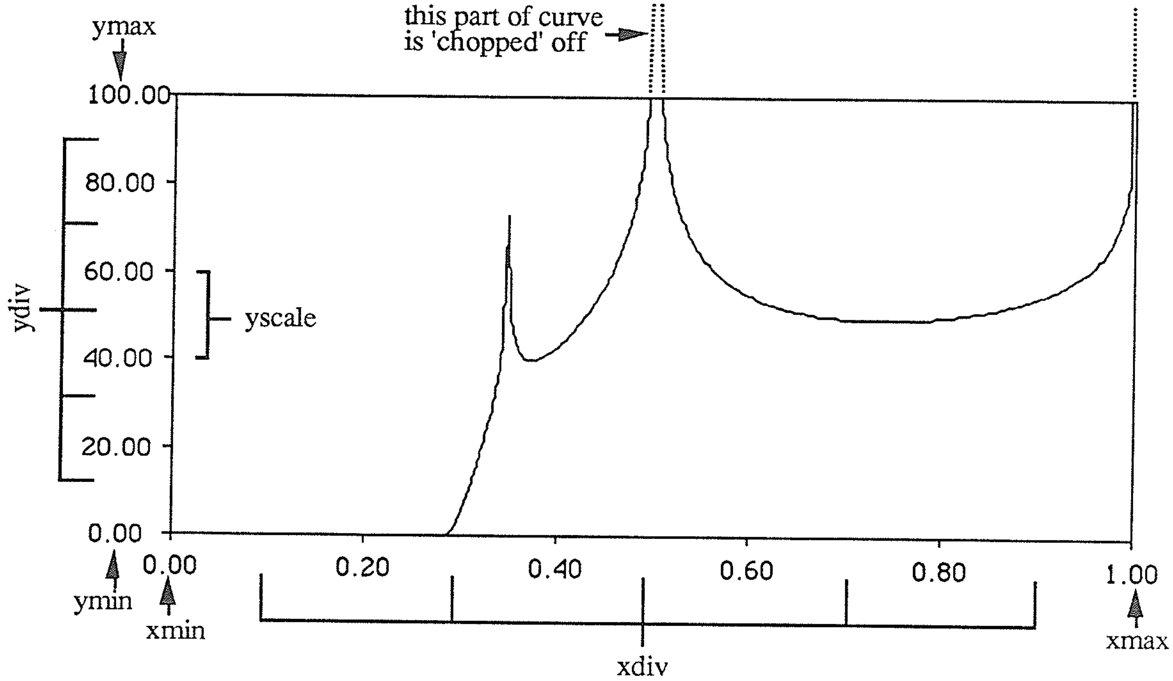


Fig. 4.7: Some of the plotting parameters.

The number of plots that can exist at a single time is limited only by the memory of the computer WDFSim is running on. The plots are organized as shown in Fig. 4.8. All plots belonging to a particular node are held in a linear linked list. A global array called *Plots* contains a top node for each node's linked plot list (i.e. an array of linked lists).

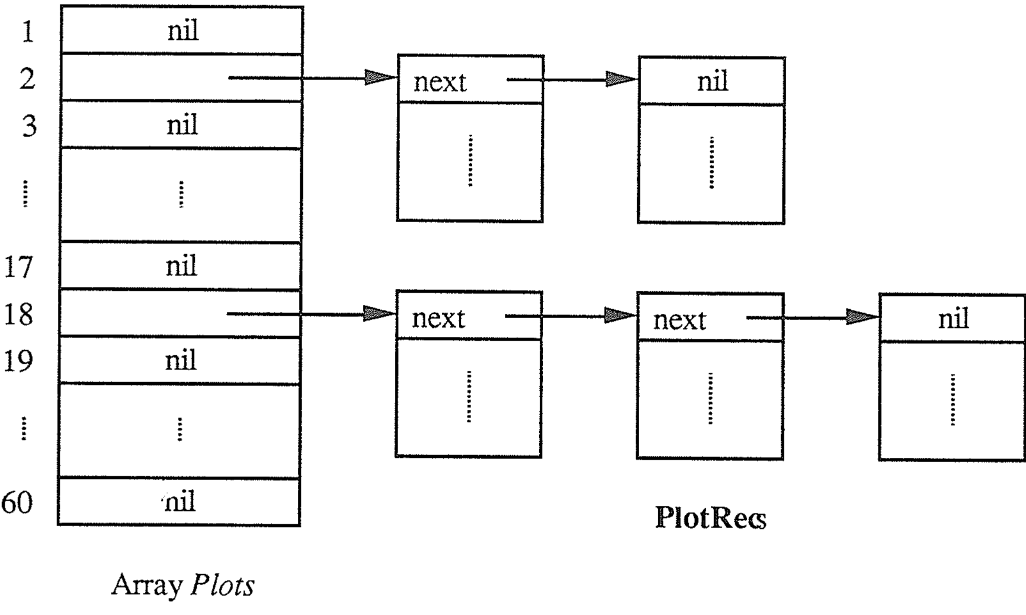


Fig. 4.8: Plot organization.

4.4.2 The Plotting Unit

All routines required to perform the plotting, and to manage the plotting data structure described in Fig. 4.8 are contained in the **Plotting** unit. The routines used to manage the data structure are the procedures InitPlots, AddPlot, RemovePlot, ErasePlots, and the functions FindPlot and SetPlotParam. Procedure InitPlots initializes the array *Plots* and is called only during startup. Procedure AddPlot adds a **PlotRec** to the end of the plot list of node k. Procedure RemovePlot removes a plot from the plot list of node k. Procedure ErasePlots removes all plots from the *Plots* array. Function FindPlot finds the plot in the plot list of node k that contains window pointer w in its *window* field and returns a pointer to the plot. Function SetPlotParm is used to control and retrieve information from the dialog boxes that appear when changing the plotting parameters.

The routines used to perform the plotting are the procedures XLabel, YLabel, PlotCurve, and the functions StartPt, EndPt, and Scale, which are arranged in the heirarchy shown in Fig. 4.9. Procedure PlotCurve accepts a **PlotRec** as an argument and oversees the plotting operation. Function StartPt finds and returns the first index k into the *xpts* array such that $xpts[k] \geq xmin$. Similarily, function EndPt finds and returns the first index k into the *xpts* array such that $xpts[k] \geq xmax$. Procedures XLabel and YLabel draw the labels on the horizontal and vertical axes, respectively. Function Scale performs the autoscaling and returns *yscale* (see Fig. 4.7) to PlotCurve.

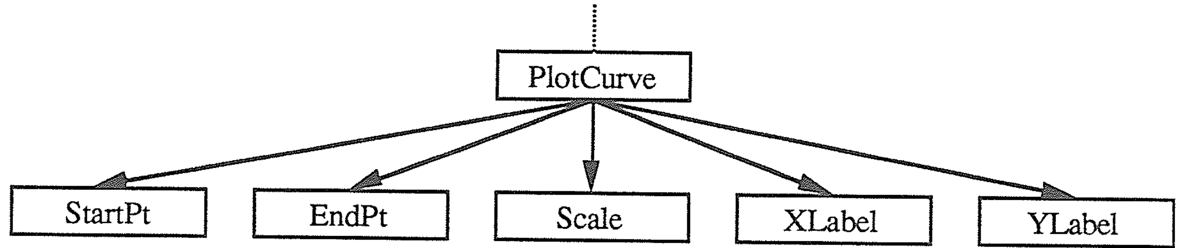


Fig. 4.9: Heirarchy of the plotting routines.

The following algorithm is used for autoscaling.

1. Call functions StartPt and EndPt which return values of n_1 and n_2, respectively.

2. Find

$$ymax = \max\,(plot.ypts[k],\ \ k = n_1,...,n_2)$$

and

$$ymin = \min\,(plot.ypts[k],\ \ k = n_1,...,n_2).$$

3. Set $ymax = \max\,(plot.ymax,\ ymax)$ and $ymin = \min\,(plot.ymin,\ ymin)$.

4. Compute $plotmax = \max\,(|ymax|,\ |ymin|)$.

5. Compute

$$t = \begin{cases} \lfloor \log\,(plotmax\,) \rfloor - 1 & , 0 \le plotmax < 1 \\ \lfloor \log\,(plotmax\,) \rfloor & , plotmax \ge 1 \end{cases},$$

where $\lfloor \cdot \rfloor$ denotes truncation to the next lower integer.

6. Compute $yscale = 10^t$.

7. Define new plotting maxima and minima given by

$$ymax = \left[\frac{ymax}{yscale} + 0.49\right] yscale$$
$$ymin = \left[\frac{ymin}{yscale} - 0.49\right] yscale\ ,$$

where $[\cdot]$ denotes rounding to the nearest integer.

8. If $plot.ydiv > 0$, then set $yscale = (ymax - ymin)\,/\,ydiv$.

V. SIMULATION WITH QUANTIZATION

Three distinct types of simulation may be performed using quantization: 1) quantization of multipliers (filter parameters) only; 2) quantization of multipliers and signals; and 3) quantization of multipliers only of filters consisting of two-port sections based on voltage rather than power wave adaptors. Of the three types of simulation, the first two may be selected by choosing the **Quantized** option from the dialog box shown in Fig. 2.5. The third type of quantized simulation is not so straightforward and requires that the quantizations be performed manually.

5.1 Quantizing Multipliers

Choosing the Quantized option from the dialog box shown in Fig. 2.5 causes procedure AdjustNet (Fig. 4.1) to call procedure QuantizeNet. The function of QuantizeNet is to quantize all multipliers before simulation commences. Procedure QuantizeNet and its related routines (see Fig. 5.1) are held in a unit called **Quantizers**. Quantization is performed in a manner to ensure the network remains passive (i.e. $|H(\omega) \leq 1|$). Hence, for different building blocks, different quantization algorithms are required.

Given the two's complement binary representation selected in the Quantization dialog box (Fig. 2.6) with b total bits and c integer bits (including sign bit), the maximum and minimum representable numbers are $2^{c-1} - 2^{c-b}$ and -2^{c-1}, respectively. As the nominal multiplier values are quantized, their values are checked to ensure they are within the range $[-2^{c-1}, 2^{c-1} - 2^{c-b}]$. If a multiplier falls outside this range, its value is saturated to either -2^{c-1} or $2^{c-1} - 2^{c-b}$ and a warning message is displayed.

5.1.1 Quantizing Parallel and Series Adaptors

Since parallel and series adaptors are inherently lossless [2], no special quantization algorithm is required to ensure that passivity is achieved. Let $Q[\cdot]$ denote the operation of quantization. The quantized multipliers of parallel and series adaptors are then given by

$$Q[\gamma_k] = \frac{[\gamma_k 2^{b-c}]}{2^{b-c}}, \qquad (5.1)$$

where $[\cdot]$ denotes rounding to the nearest integer.

5.1.2 Quantizing Inverse Multipliers

To ensure passivity is maintained, a pair of inverse multipliers n and $\frac{1}{n}$, must be quantized such that

$$Q\left[\frac{1}{n}\right] \leq \frac{1}{n} \text{ and } Q[n] \leq n. \tag{5.2a}$$

Hence, the quantized pair of inverse multipliers are given by

$$Q[n] = \frac{\lfloor n\, 2^{b-c} \rfloor}{2^{b-c}} \text{ and } Q\left[\frac{1}{n}\right] = \frac{\lfloor \frac{1}{n}\, 2^{b-c} \rfloor}{2^{b-c}}, \tag{5.2b}$$

where $\lfloor \cdot \rfloor$ denotes truncation to the next lower integer.

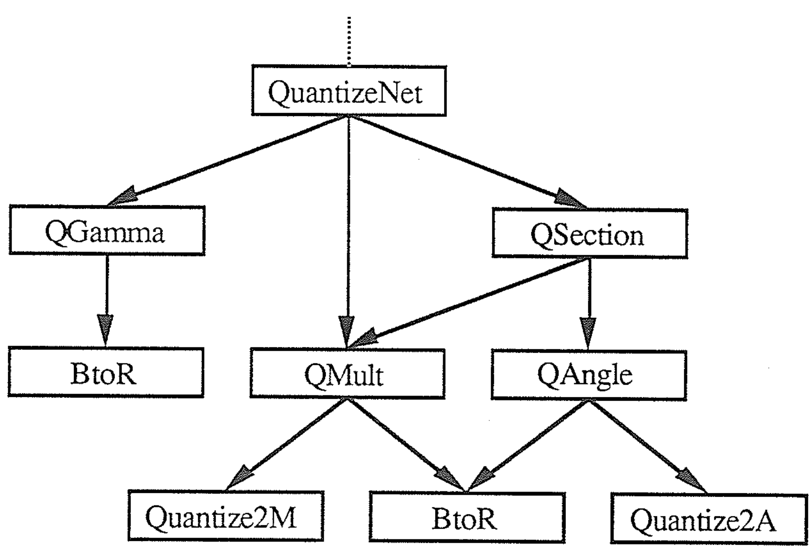


Fig. 5.1: Hierarchy of the **Quantizers** unit.

5.1.3 Quantizing Normalized Two-Port Adaptors

Normalized two-port adaptors contain two sets of the pair of multipliers $\{\cos\theta, \sin\theta\}$, and are parameterized by the single value θ. Recall that the normalized two-port adaptor is the WD equivalent of the ideal transformer with equal port resistances (Table 3.2). In order to ensure passivity, the pair of multipliers $\{\cos\theta, \sin\theta\}$ must be quantized such that

$$\left(Q\left[\cos\theta\right]\right)^2 + \left(Q\left[\sin\theta\right]\right)^2 \leq 1. \tag{5.3}$$

Experimentally, the following algorithm has been found to work well.

1. Compute

   $$x_1 = \frac{\left[2^{b-c}\cos\theta\right]}{2^{b-c}}, \text{ and } y_1 = \frac{\pm\left\lfloor 2^{b-c}\sqrt{1-x_1^2}\right\rfloor}{2^{b-c}},$$

   where the sign of y_1 is set equal to sgn($\sin\theta$).

2. Compute

   $$y_2 = \frac{\left[2^{b-c}\sin\theta\right]}{2^{b-c}}, \text{ and } x_2 = \frac{\pm\left\lfloor 2^{b-c}\sqrt{1-y_2^2}\right\rfloor}{2^{b-c}},$$

   where the sign of x_2 is set equal to sgn($\cos\theta$).

3. Compute $ss_1 = x_1^2 + y_1^2 \leq 1$ and $ss_2 = x_2^2 + y_2^2 \leq 1$.

4. If $ss_1 > ss_2$ then $Q\left[\cos\theta\right] = x_1$ and $Q\left[\sin\theta\right] = y_1$, otherwise $Q\left[\cos\theta\right] = x_2$ and $Q\left[\sin\theta\right] = y_2$.

5.2 Two's Complement Fixed-Point Binary Arithmetic

If quantization of signals is also selected from the quantization scheme configuration dialog box (Fig. 2.6), two's complement fixed-point binary arithmetic is used during the time domain simulation. Options include: overflow or saturation addition; and rounding, truncation, or magnitude truncation multiplication.

The binary arithmetic routines, all of which are contained in the **BinaryMath** unit, are based on previous work by Pepe and Rogers [5]. All signals and multipliers, which must be from the set

$$\mathbb{F} = \left\{ x \in \mathbb{R} \,\middle|\, -2^{c-1} \leq x \leq 2^{c-1} - 2^{c-b} \right\}, \tag{5.4}$$

are mapped into the finite set

$$\mathbb{Z}_{2^b} = \left\{ 0,\ 1,\ 2,\ \ldots,\ 2^b - 1 \right\} \tag{5.5}$$

by the mapping $\Psi \,|\, \mathbb{F} \rightarrow \mathbb{Z}_{2^b}$ defined by

$$\Psi(x) = \begin{cases} \lfloor x\, 2^{b-c} \rceil, & x \geq 0 \\ \lfloor x\, 2^{b-c} + 2^b \rceil, & x < 0 \end{cases} . \tag{5.6}$$

In general, the mapping $\Psi \,|\, \mathbb{F} \rightarrow \mathbb{Z}_{2^b}$ is many to one. The simulation proceeds with the elements of $\mathbb{Z}_{2^b}$ as parameters.

If the *binary* flag in the **Simulation** record is set to **true** (indicating that binary arithmetic is to be used during the time domain simulation), the quantization routines call function BtoR (see Fig. 5.1) which implements the forward mapping $\Psi(x)$. The forward mapping must also be applied to the elements of the input file(s) as they are read in during simulation. The inverse mapping $\Phi \,|\, \mathbb{Z}_{2^b} \rightarrow \mathbb{F}$ is given by

$$\Phi(x) = \begin{cases} \dfrac{x}{2^{b-c}}, & x < 2^{b-1} \\ \dfrac{x - 2^b}{2^{b-c}}, & x \geq 2^{b-1} \end{cases} , \tag{5.7}$$

which is implemented in function RtoB. The inverse mapping is applied to the filter's output before the values are written to the output file(s).

Simulation of fixed-point two's complement binary addition is performed by function BSum. If the *sat* flag is passed as **true**, saturation addition is used, otherwise, overflow addition is performed. Simulation of fixed-point two's complement binary multiplication is

performed by functions BProd and MagTrunc. If BProd's *rnd* flag is passed as **true**, BProd simulates rounding multiplication, otherwise BProd performs truncation multiplication. Function MagTrunc simulates magnitude truncation multiplication. Function BNeg returns the two's complement of the passed argument. For a description of the specific algorithms used, the reader is referred to [5].

5.3 Quantized Simulation of Elementary Two-Port Sections Based on Voltage Wave Adaptors

As discussed in Section 5.1.3, passivity is introduced when quantizing the multiplier set $\{\cos\theta, \sin\theta\}$ of normalized (power wave) two-port adaptors. This is due to the fact that, in general,

$$\left(Q[\cos\theta]\right)^2 + \left(Q[\sin\theta]\right)^2 < 1 \tag{5.8}$$

when $Q[\cos\theta]$ and $Q[\sin\theta]$ are binary numbers. Thus, in the two-port elementary sections described in Section 3.1.3, $2n$ sources of passivity are introduced, where n is the number of normalized two-port adaptors within the section.

As pointed out in [4], by replacing each power wave adaptor with its voltage wave equivalent shown in Fig. 5.2, the number of sources of passivity can be reduced. This is especially evident is the case of reciprocal sections, where two of the extra pairs of multipliers cancel each other out.

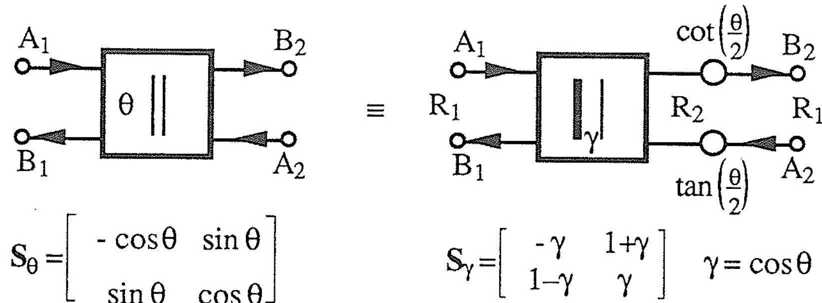


Fig. 5.2: Voltage wave equivalent of a power wave two-port adaptor.

First consider reciprocal sections **1A**, **1B**, **1C**, and **1D**. The general WD structure of these sections is shown in Fig. 5.3(a), where $S_4 = \pm 1$ and $S_2 = \pm z^{-1}$. By applying the equivalence shown in Fig. 5.2 to all power wave adaptors, the structure shown in Fig. 5.3(b) is obtained. The two pairs of multipliers cancel each other out and all sources of passivity are eliminated. The final voltage wave equivalents of sections **1A**, **1B**, **1C**, and **1D** are shown in Fig. 5.3(c).

By applying the same procedure to sections **2A** and **2B**, the sequence of equivalences shown in Fig 5.4 are obtained. Again, $S_4 = \pm 1$ and $S_2 = \pm z^{-1}$, and all multipliers cancel each other out. For sections **2C** and **2D** (see Fig. 5.5), not all multiplier pairs cancel each other out, however, they may be grouped into a single pair. Thus, the number of sources of passivity has been reduced from eight to one.

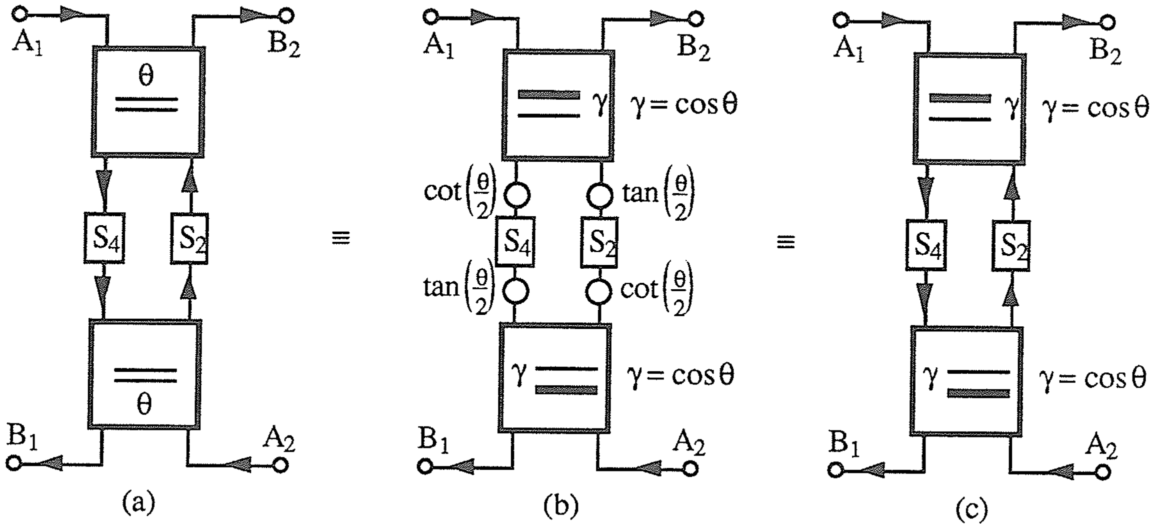


Fig. 5.3: (a) General structure of first order power wave reciprocal sections, (b) intermediate step, and (c) voltage wave equivalent.

It is possible to simulate first- and second-order reciprocal sections that are realized with voltage wave adaptors using the power wave structures already implemented (with a slight modification to sections **2C** and **2D**). To see how this is accomplished, let the voltage wave multipliers be quantized values, i.e.,

$$\gamma'_k = \mathrm{Q}[\gamma_k] = \mathrm{Q}[\cos\ \theta_k]\ . \tag{5.9}$$

Next, apply the equivalence shown in Fig. 5.6 to each quantized voltage wave adaptor. By following the steps outlined in Figures 5.3 and 5.4 (in reverse), it can be shown that all additional multipliers introduced cancel each other out for sections **1A**, **1B**, **1C**, **1D**, **2A**, and **2B**. For sections **2C** and **2D**, however, this is not the case. The steps to obtain a power wave equivalent of these sections is shown explicitly in Fig. 5.7. Note that two multipliers m and n had to be added into the structure of sections **2C** and **2D** (for power wave simulations set $m = n = 1$).

The simulation of voltage wave elementary two-port sections must be done in the **Nominal** mode, and hence, quantizations must be done manually. The simulation algorithm follows.

1. Compute the quantized voltage wave adaptor multipliers

   $$\gamma'_k = \mathrm{Q}[\gamma_k] = \mathrm{Q}[\cos\ \theta_k]\ .$$

   for each two-port adaptor.

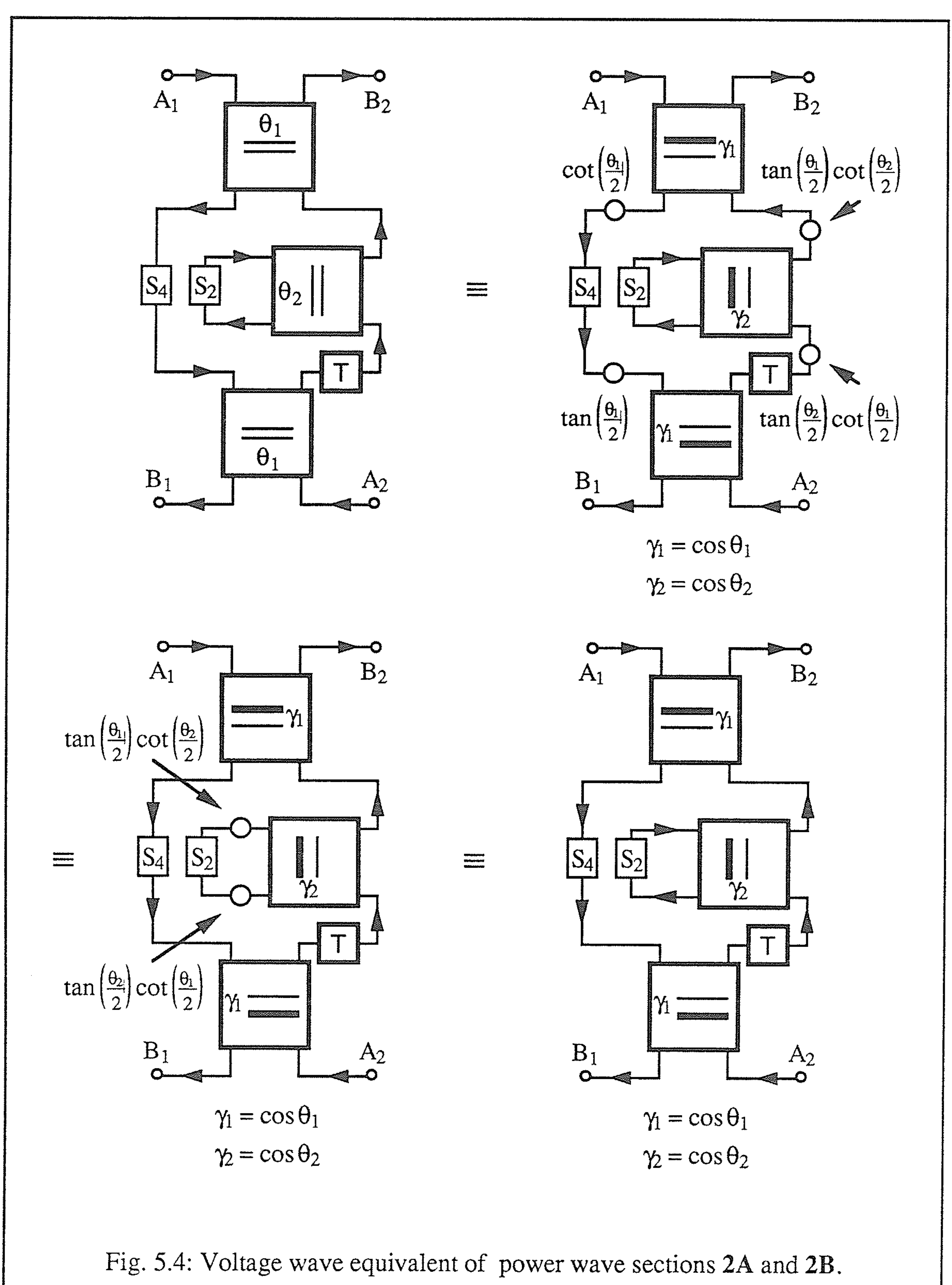


Fig. 5.4: Voltage wave equivalent of power wave sections **2A** and **2B**.

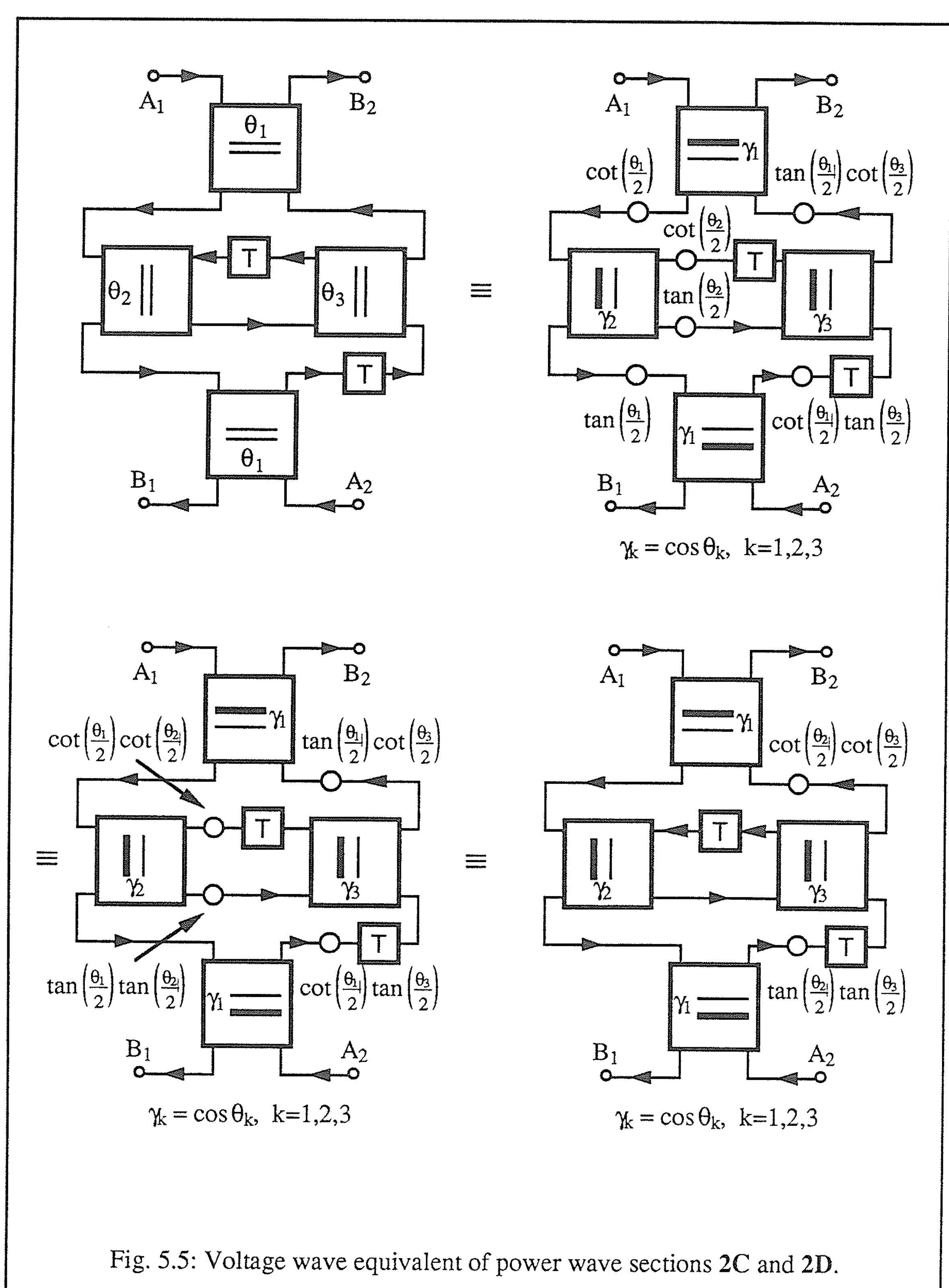


Fig. 5.5: Voltage wave equivalent of power wave sections **2C** and **2D**.

2. Compute the power wave adaptor rotation angles

$$\widehat{\theta}_k = \mathrm{Cos}^{-1}\, \gamma'_k$$

where $\widehat{\theta}_k$ is chosen to lie in the same quadrant as θ_k .

3. For sections **2C** and **2D** compute the multipliers

$$n' = \mathrm{Q}\left[\cot\left(\frac{\widehat{\theta}_2}{2}\right)\cot\left(\frac{\widehat{\theta}_3}{2}\right)\right] \text{ and } m' = \mathrm{Q}\left[\tan\left(\frac{\widehat{\theta}_2}{2}\right)\tan\left(\frac{\widehat{\theta}_3}{2}\right)\right]$$

following the quantization algorithm in Section 5.1.2 and then the multipliers

$$n = n' \tan\left(\frac{\widehat{\theta}_2}{2}\right)\tan\left(\frac{\widehat{\theta}_3}{2}\right) \text{ and } m = m' \cot\left(\frac{\widehat{\theta}_2}{2}\right)\cot\left(\frac{\widehat{\theta}_3}{2}\right).$$

4. Perform the simulation in the **Nominal** mode using parameters $\widehat{\theta}_k$, m, and n.

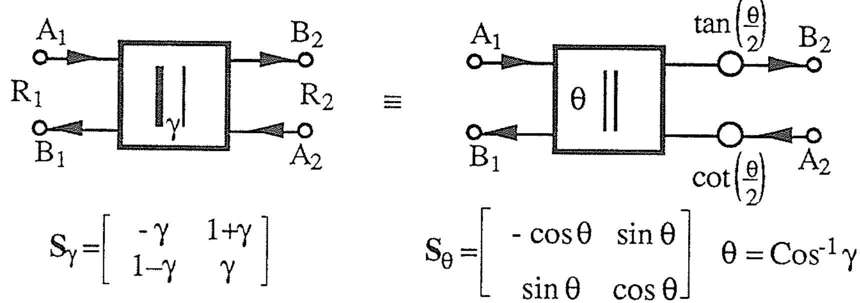


Fig. 5.6: Power wave equivalent of a voltage wave two-port adaptor.

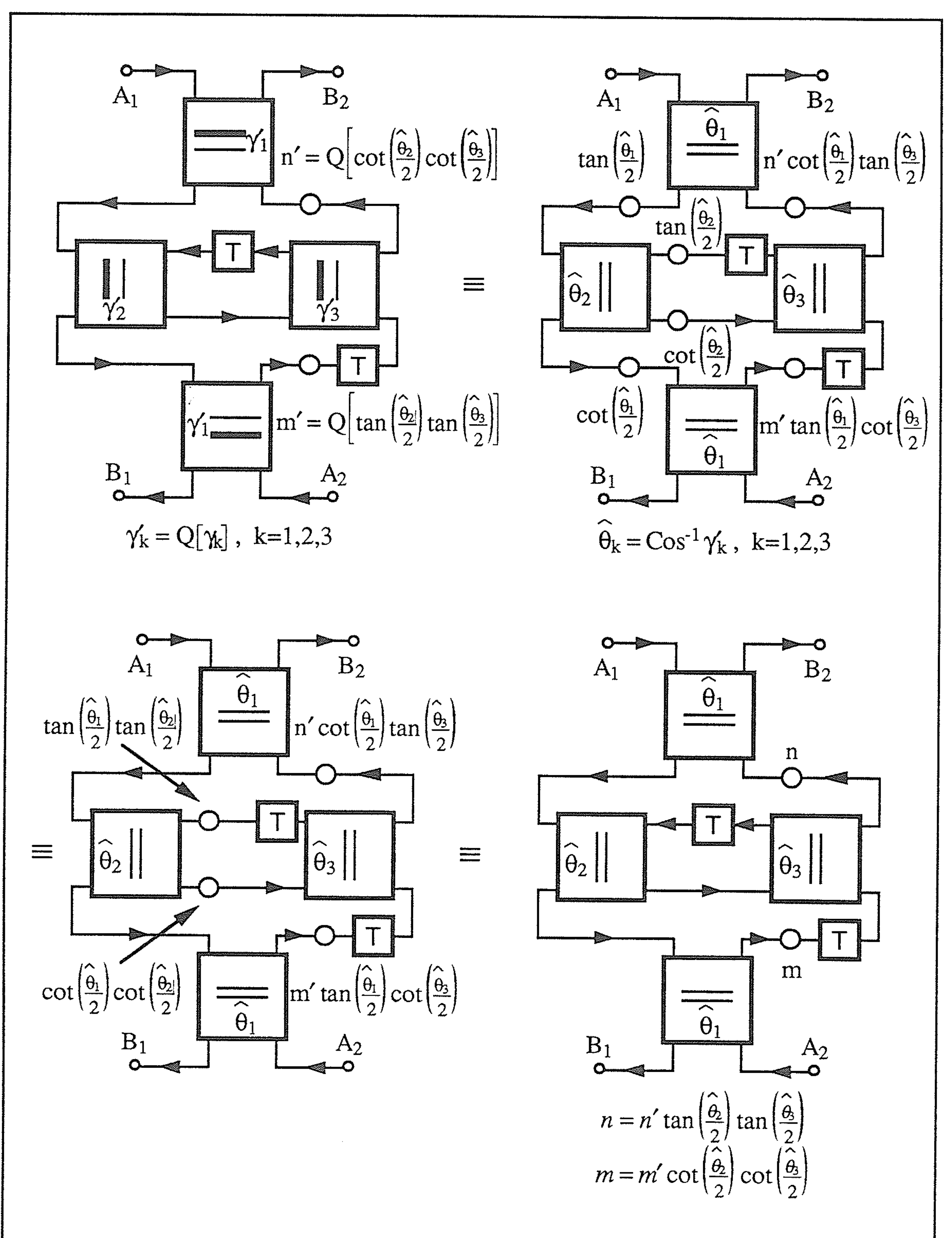


Fig. 5.7: Power wave equivalent of quantized voltage wave sections **2C** and **2D**.

VI. SIMULATION OF WDFS IN RESIDUE NUMBER SYSTEMS

The residue number system (RNS) is an ancient Chinese form of integer mathematics that dates back to the third century [6]. The RNS has been successfully applied to the implementation of several digital signal processing applications at high speed. In particular, finite impulse response (FIR) filtering [7], infinite impulse response (IIR) filtering [8], number theoretic transforms (NTTs) [9], and more recently, structurally passive digital filters consisting of only planar rotators and delays [10]. The planar rotator is closely related to the normalized two-port adaptor (see [4] for an explanation).

6.1 Basic Concepts

In general, the set

$$\mathbb{Z}_M = \{0, 1, 2, \ldots, M - 1\} \tag{6.1}$$

forms a finite ring under modulo M addition and multiplication. If M is prime, $\mathbb{Z}_M$ forms a finite field, or *Galois Field*, which is more commonly denoted by $GF(M)$. The ring $\mathbb{Z}_M$ may be called a *single modulus residue number system* (SMRNS) [12], where M is the modulus. Every integer x in the set

$$\mathbb{S}_M = \begin{cases} \left\{\frac{-M}{2}, \ldots, -1, 0, 1, \ldots, \frac{M}{2} - 1\right\} & \text{, if } M \text{ even} \\ \left\{\frac{-(M-1)}{2}, \ldots, -1, 0, 1, \ldots, \frac{M-1}{2}\right\} & \text{, if } M \text{ odd} \end{cases} \tag{6.2}$$

may be uniquely mapped into the ring $\mathbb{Z}_M$ by the mapping $\Lambda \mid \mathbb{S}_M \rightarrow \mathbb{Z}_M$, where

$$\Lambda(x) = \begin{cases} x & , x \geq 0 \\ x + M & , x < 0 \end{cases} . \tag{6.3}$$

Now, suppose $M = \prod_{i=1}^{L} P_i$ where the P_i are all relatively prime. The ring $\mathbb{Z}_M$ can be made isomorphic to the product ring [12]

$$\mathbb{Z}_{P_1} \times \mathbb{Z}_{P_2} \times \ldots \times \mathbb{Z}_{P_L} . \tag{6.4}$$

The product ring in (6.4) may be called a *multiple modulus residue number system*

(MMRNS), where P_i is the ith modulus. Every integer X in $\mathbb{Z}_M$ may be uniquely mapped to its corresponding member $\langle X_1, X_2, ..., X_L \rangle$ of the product ring by the mapping $\Gamma | \mathbb{Z}_M \rightarrow \mathbb{Z}_{P_1} \times \mathbb{Z}_{P_2} \times ... \times \mathbb{Z}_{P_L}$, where

$$\Gamma(X) = \langle X \bmod P_1, X \bmod P_2, ..., X \bmod P_L \rangle . \tag{6.5}$$

The set $P = \{P_1, P_2, ..., P_L\}$ of relatively prime moduli is called the *moduli set* of the MMRNS.

The principal attraction of the MMRNS is the way in which arithmetic is performed. If $X \rightarrow \langle X_1, X_2, ..., X_L \rangle$ and $Y \rightarrow \langle Y_1, Y_2, ..., Y_L \rangle$, then

$$Z = X \circ Y \rightarrow \langle Z_1, Z_2, ..., Z_L \rangle \tag{6.6}$$

where $Z_i = (X_i \circ Y_i) \bmod P_i$, and $\circ$ denotes addition, subtraction, or multiplication. (Note that the symbol $\rightarrow$ is used to symbolize the mapping from $\mathbb{S}_M$ to the product ring.) Thus, the MMRNS digits can be computed concurrently with no carry propagation. Typically, $Z_i = (X_i \circ Y_i) \bmod P_i$ is computed via high speed table look-up, leading to systems with very high throughput.

A member of the product ring may be mapped back to its unique member of $\mathbb{Z}_M$ via either: 1) the Chinese Remainder Theorem (CRT), or 2) the mixed-radix number representation (MRNR). The CRT is described in Blauht [11], and the MRNR is explained in Taylor [6] and Gregory and Krishnamurthy [12]. The MRNR is generally accepted as being superior to the CRT [6,12], either in terms of computational speed or because special purpose hardware implementations are more efficient.

One major disadvantage of the RNS is its ability (or lack of it) to efficiently manage dynamic range (register) overflow. Unlike weighted number systems, where overflow can be efficiently handled by saturation, rounding, or truncation arithmetic, RNS procedures are complex and time consuming. Various scaling routines have been published [8,13-15] based on the CRT and the MRNR, however, all require a significant investment in hardware. In [16], an efficient scaling algorithm based on the tri-moduli set $P = \{2^n - 1, 2^n, 2^n + 1\}$ was developed which is suitable for digital filtering. Other moduli sets that lead to efficient scaling operations are reported in [17]. However, these are beyond the scope of this work and will not be considered here.

6.2 The Autoscale Residue Multiplier [16]

In order to see why scaling is necessary in RNS based digital filters, consider the first-order IIR filter

$$y(n) = a\,x(n) + b\,y(n-1). \tag{6.7}$$

In general, the signals y and x, and the multipliers a and b are fractional or mixed fractional numbers. Fractional coefficients and signals must be multiplied by a fixed constant S in order to bring the parameters into the the set $\mathbb{S}_M$. Note that no actual multiplication need be performed, the only operation implied is the interpretation of the radix point (e.g. $0.110 \equiv 6$).

Now, let uppercase letters denote RNS quantities while lowercase letters represent the corresponding fixed point quantity (e.g. $A = aS$). Equation (6.7) becomes

$$Y(n) = \frac{1}{S}\{A\,X(n) + B\,Y(n-1)\} \tag{6.8}$$

in the RNS. After the expression in the braces $\{\cdot\}$ of (6.8) has been computed, the result must be divided by S (and quantized) so that $Y(n)$ is available for further iteration of the recursive relationship. Division by a fixed constant is known as *scaling*.

The efficient autoscale residue multiplier (ARM) of Taylor and Huang [16] has been implemented in WDFSim as an integrated RNS multiplier and scaler. The ARM, shown in Fig. 6.1, computes $Z = [cX/V]$ (c and V are fixed constants) in the MMRNS defined over the moduli set $P = \{2^n - 1, 2^n, 2^n + 1\}$, where $[\cdot]$ denotes rounding to the nearest integer. The dynamic range of this MMRNS is $M = 2^{3n} - 2^n$, or approximately $3n$ bits.

The ARM uses a form of data compression to calculate $\overline{X} \rightarrow \langle 0, \overline{X}_2, \overline{X}_3 \rangle$ where

$$\overline{X}_i = (X_i - X_1) \bmod P_i, \;\; i = 1,2,3 \tag{6.9}$$

which is used to address three tables (ROMs usually) that lookup $[c\overline{X}/V]$ mod P_i, $i = 1,2,3$. Obviously, the maximum dynamic range of a realizable RNS is limited by the address space of the ROMs. Suppose the largest available ROMs have an address space of q bits. Without any data compression, the dynamic range is limited by $n \le q/3$, however with data compression , the limiting factor becomes $n \le q/2$.

In the data compression scheme (6.9), P_1 is known as the *truncation moduli*, which in fact, may be chosen arbitrarily. With truncation moduli P_i, the error variance for the ARM was reported in [16] to be

$$\sigma^2 = \frac{1}{12}\left(\frac{P_i^2\, c^2}{M^2}\right). \tag{6.10}$$

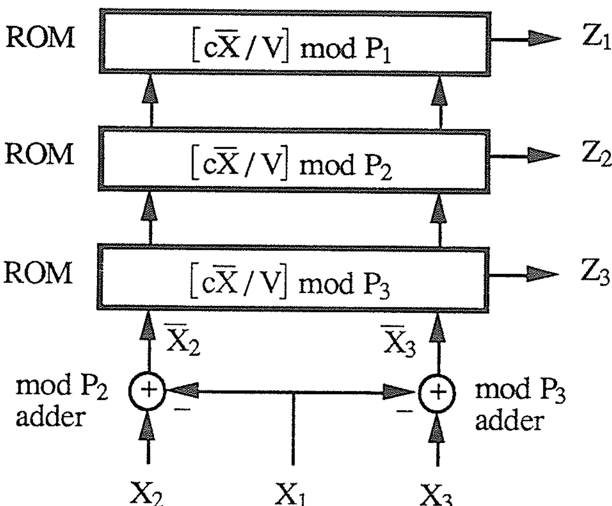


Fig. 6.1: The ARM architecture.

6.3 The Inverse Mappings

Of the two inverse mappings, $\Theta \,|\, \mathbb{Z}_M \rightarrow \mathbb{S}_M$ and $\Xi \,|\, \mathbb{Z}_{P_1} \times \mathbb{Z}_{P_2} \times \ldots \times \mathbb{Z}_{P_L} \rightarrow \mathbb{Z}_M$, the first is straightforward and is given by

$$\Theta(x) = \begin{cases} x & , x < \frac{M}{2} \\ x - M, & x \geq \frac{M}{2} \end{cases} . \tag{6.11}$$

The latter, however, is more difficult and must be considered in more detail.

As pointed out in Section 6.1, the inverse mapping $\Xi \,|\, \mathbb{Z}_{P_1} \times \mathbb{Z}_{P_2} \times \ldots \times \mathbb{Z}_{P_L} \rightarrow \mathbb{Z}_M$ may be carried out via either the CRT or the MRNR. However, for the tri-moduli set $P = \{2^n - 1, 2^n, 2^n + 1\}$, more efficient procedures have been developed [18-20]. The algorithm adopted for use in WDFSim was originally presented by Bernardson [18]. For hardware implementations, Bernardson's procedure may be realized using only binary adders and offers a 40ns 36-bit conversion time. The algorithm to restore X from its residues $\langle X_1, X_2, X_3 \rangle$ follows [18].

1. Compute $\widehat{X} = \left(2^{n-1}\left[2^n (X_1 + X_3) + (X_1 - X_3)\right]\right) \bmod 2^{2n} - 1$.

2. Compute $X = \widehat{X} + 2^{2n}\left[\left(\widehat{X} \bmod 2^n - X_2\right) \bmod 2^n\right] - \left[\left(\widehat{X} \bmod 2^n - X_2\right) \bmod 2^n\right]$.

6.4 RNS Simulation

Simulation of WDFs based on residue number systems is invoked by selecting the **MMRNS** option from the number system dialog box (Fig. 2.5). The dialog box shown in Fig. 2.7 then appears prompting for two parameters: n - the exponent in the moduli set $P = \{2^n - 1, 2^n, 2^n + 1\}$, and c - the number of bits reserved for the integer portion (including the sign bit) of the data words.

Ideally, the mappings $\mathbb{R} \rightarrow \mathbb{S}_M \rightarrow \mathbb{Z}_M \rightarrow \mathbb{Z}_{P_1} \times \mathbb{Z}_{P_2} \times \mathbb{Z}_{P_3}$ would be applied to the WDF multiplier values prior to simulation and then the simulation would proceed with the members of the product ring (MMRNS) as parameters. The problem is that members of the product ring are defined as Pascal type

MMRN = **array** [1..3] **of** longint

and multipliers are defined to be Pascal type **extended** (Section 3.2). Hence, multipliers cannot be assigned values of the product ring. To overcome this problem, the mapping $\Omega \mid \mathbb{R} \rightarrow \mathbb{S}_M$ is performed initially and the simulation proceeds with the members of $\mathbb{S}_M$ as parameters. Every time the operation $C = A \circ B$ is required, the procedure followed is:

1. Perform the forward mappings $\Gamma(\Lambda(A)) = \langle A_1, A_2, A_3 \rangle$ and $\Gamma(\Lambda(B)) = \langle B_1, B_2, B_3 \rangle$;

2. Compute $\langle C_1, C_2, C_3 \rangle = \langle A_1 \circ B_1, A_2 \circ B_2, A_3 \circ B_3 \rangle$; and

3. Perform the inverse mapping $\Theta(\Xi(\langle C_1, C_2, C_3 \rangle)) = C$.

Since the mappings $\Lambda(x)$, $\Gamma(x)$, $\Xi(x)$, and $\Theta(x)$ are all isomorphisms, members of one set map exactly to a unique member of the other sets. Hence, no errors are introduced by the above procedure for MMRNS simulation. The only drawback is an increase in the computation time.

On the other hand, the mapping $\Omega \mid \mathbb{R} \rightarrow \mathbb{S}_M$ defined by

$$\Omega(x) = [x\, S\,] \tag{6.12}$$

where

$$x \in \left\{ \mathbb{R} \;\middle|\; -2^{c-1} \leq x \leq 2^{c-1}\left(1 - \tfrac{2}{M}\right) \right\}, \tag{6.13}$$

and

$$S = \frac{M}{2^c} = 2^{n-c}\left(2^{2n} - 1\right) \tag{6.14}$$

is the *scale factor*, is many-to-one and is a form of quantization. The mapping $\Omega(x)$, like other quantizations, must be performed in such a manner that the network remains passive. The **Quantizers** unit, described in Section 5.1, is used to perform the mapping $\Omega(x)$ on the filter parameters prior to simulation. Each 2^{b-c} term in the quantization algorithms described in Section 5.1 is replaced with S from (6.14). After quantization, the multipliers of the WDF will be within the set

$$\left\{ \frac{a}{S} \,\middle|\, a \in \mathbb{S}_M \right\}. \tag{6.15}$$

The multiplier values are then simply multiplied by S to bring all filter parameters into the set $\mathbb{S}_M$.

The **ResidueMath** unit contains all the routines required for arithmetic in the product ring, as well as routines to perform all the mappings defined in this chapter. In particular, the arithmetic routines are functions RSum and AutoScale, which perform addition and multiplication (Section 6.2), respectively. Functions BtoSM, SMtoMM, MMtoSM, and SMtoB implement the mappings $\Lambda | \mathbb{S}_M \rightarrow \mathbb{Z}_M$, $\Gamma | \mathbb{Z}_M \rightarrow \mathbb{Z}_{P_1} \times \mathbb{Z}_{P_2} \times \mathbb{Z}_{P_3}$, $\Xi | \mathbb{Z}_{P_1} \times \mathbb{Z}_{P_2} \times \mathbb{Z}_{P_3} \rightarrow \mathbb{Z}_M$, and $\Theta | \mathbb{Z}_M \rightarrow \mathbb{S}_M$, respectively. The sets and mappings defined in this chapter are summarized in Fig. 6.2.

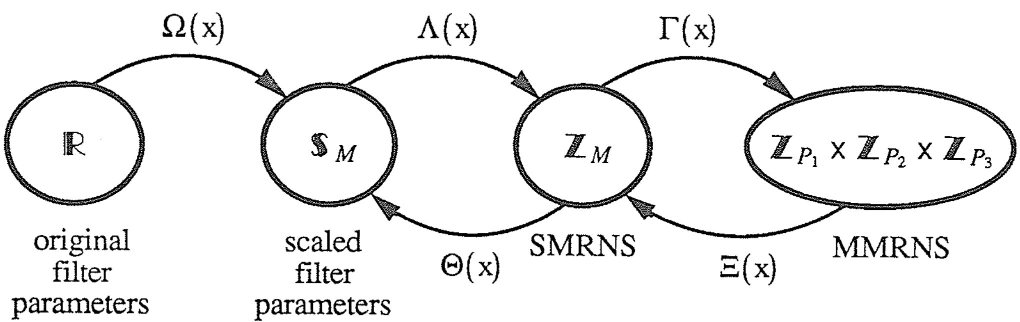


Fig. 6.2: Sets and mappings associated with RNS simulation.

VII. MAN-MACHINE INTERFACE

WDFSim, like most Macintosh applications, is *event driven*. That is, the program waits in a loop, called the *event loop*, for some event (e.g. mouse click) to occur. When an event occurs, the program takes the appropriate actions and then returns to the event loop where it waits for the next event to occur.

The Macintosh Toolbox[1] routine GetNextEvent waits for the system to detect the occurrence of an event, and then returns a record of type **EventRecord**. The *what* field of the **EventRecord** contains the event type. Four event types are of interest to WDFSim: mouse events, key events, activate events, and update events.

The event loop is contained in the unit **Main**, along with some initialization routines. All routines related to the man-machine interface, including routines to perform all menu functions (excluding the **Simulate** and **Build** commands), are contained in the unit **UserInterface**. Unit **UserInterface** exports four procedures: MouseEvent, KeyEvent, ActEvent, and UpEvent, which respond to the four event types listed in the preceding paragraph (see Fig. 7.1).

7.1 Mouse Events

By far, most of the actions taken by WDFSim are invoked through use of the mouse. By using the Toolbox routine FindWindow and the *where* field of the **EventRecord**, the exact location (and in which window) at which the mouse event took place is determined. The seven possibilities, and the actions taken are:

1. in a system window → call Toolbox routine SystemClick;

2. in the menu bar → call procedure MenuCmnd;

3. in the drag bar of a window → call Toolbox procedure DragWindow;

4. in the window content region → if the window is not the front window, make it the front window, otherwise, if the window is the File Window, call procedure ActivateNode;

---

1 Throughout this section of WDFSim, the built-in Macintosh Toolbox routines and data structures are extensively used. For a complete description of the Toolbox, see [21-25].

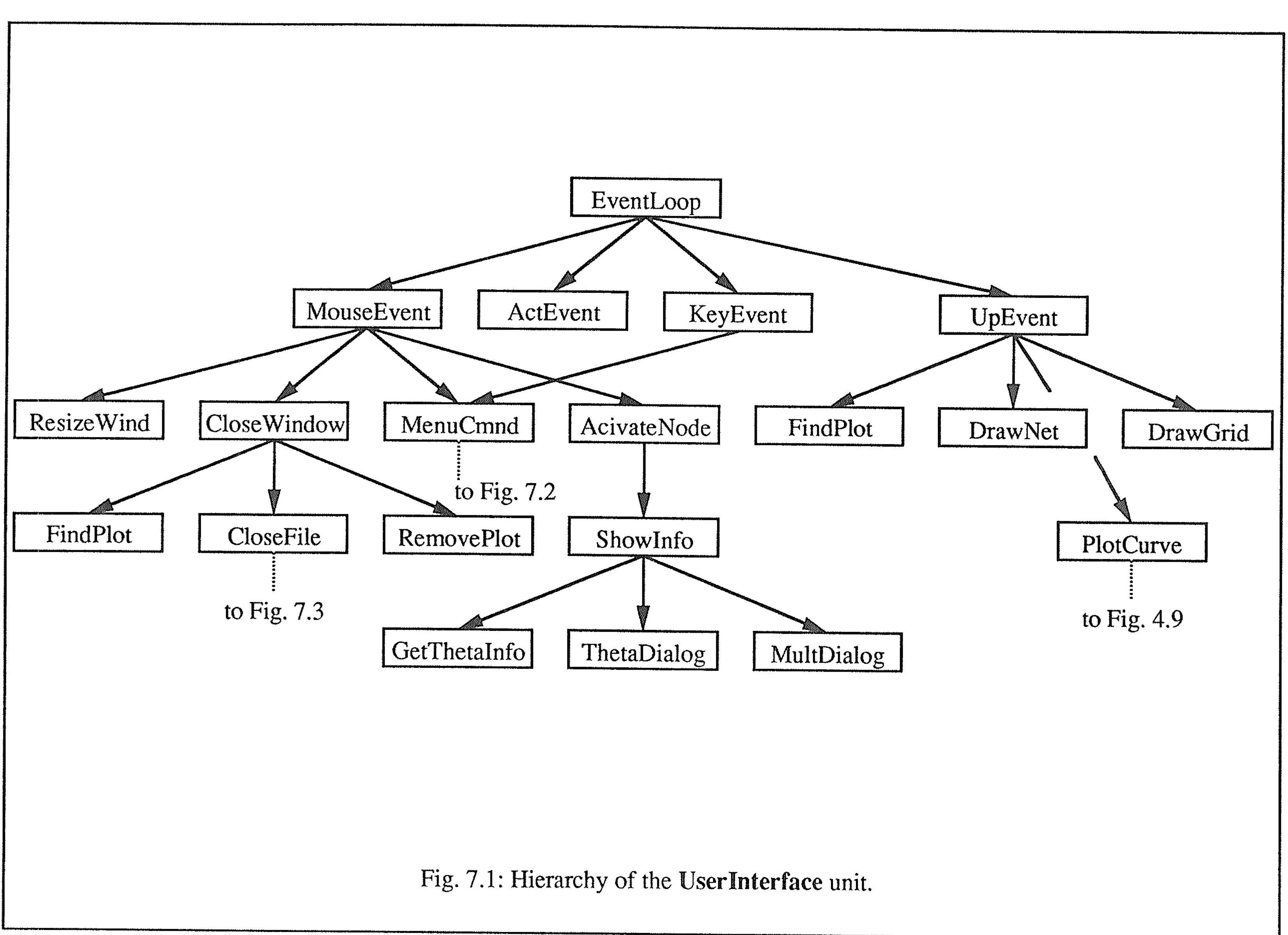


Fig. 7.1: Hierarchy of the **UserInterface** unit.

5. in the go away box of a window → call procedure CloseWindow;

6. in the grow box of a window → call procedure ResizeWindow; and

7. in the desk → do nothing.

7.1.1 Menu Commands

If it has been determined that a mouse event has occurred in the menu bar, the Toolbox routine MenuSelect is used to convert the *where* field of the **EventRecord** to a Pascal **longint** containing both the menu number and the menu item number. The routines HiWord and LoWord are then used to get the menu and the menu item numbers, respectively, from the **longint**. Procedure MenuCmnd then calls the routine which handles the particular menu that has been selected (see Fig. 7.2).

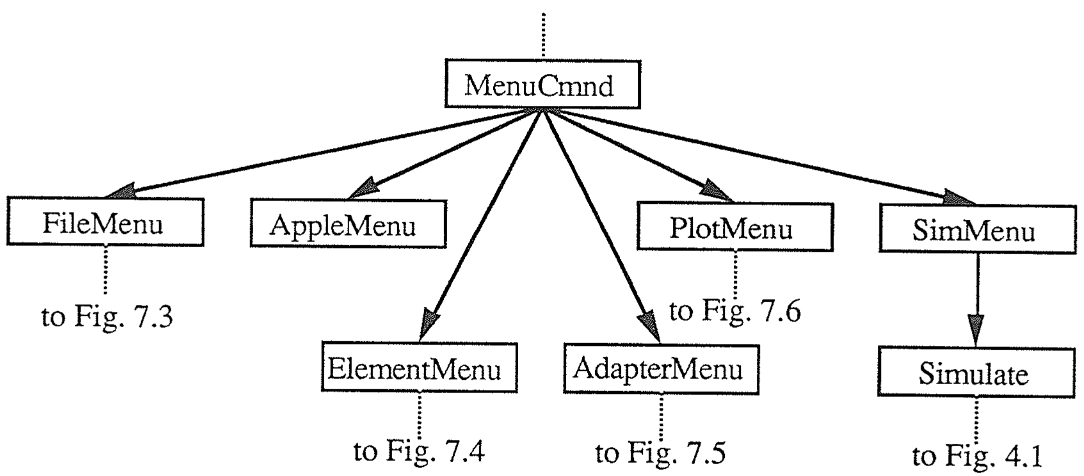


Fig. 7.2: Hierarchy of procedure MenuCmnd and its associated routines.

Two of the menu procedures, AppleMenu and SimulateMenu, are fairly simple. Procedure AppleMenu responds to the command **About WDFSim** by displaying a dialog box containing some information about the program. For any other item selected from the menu (desk accessories), the Toolbox routine OpenDeskAcc is called. Procedure SimulateMenu sets the *SimType* field of the **Simulation** record according to which item has been chosen, and then calls procedure Simulate (see Chapter IV). The other menu procedures, which are more complex, are individually described in the following sections.

7.7.1a The File Menu

There are eight file maintenance commands under the **File** menu: 1) **New**, 2) **Open**, 3) **Save**, 4) **Save As**, 5) **Close**, 6) **Build**, 7) **Save As PICT**, and 8) **Quit**. Of these commands, the purposes of the first five and last one are obvious. The **Build** command is described in Section 3.3. The **Save As PICT** option saves the contents of the File Window (minus the grid) as a PICT file [25] which may later be printed by applications such as MacDraw or MacPaint.

WDFSim saves the filter files as file type FILT with the creator signature WDFS [23]. The only type of files that can be opened by WDFSim are FILT files. Therefore, any program generating a filter file to be used by WDFSim must save it as type FILT, preferrably with the creator signature ???? or WDFS. The hierarchy of procedure FileMenu and its related routines is shown in Fig. 7.3.

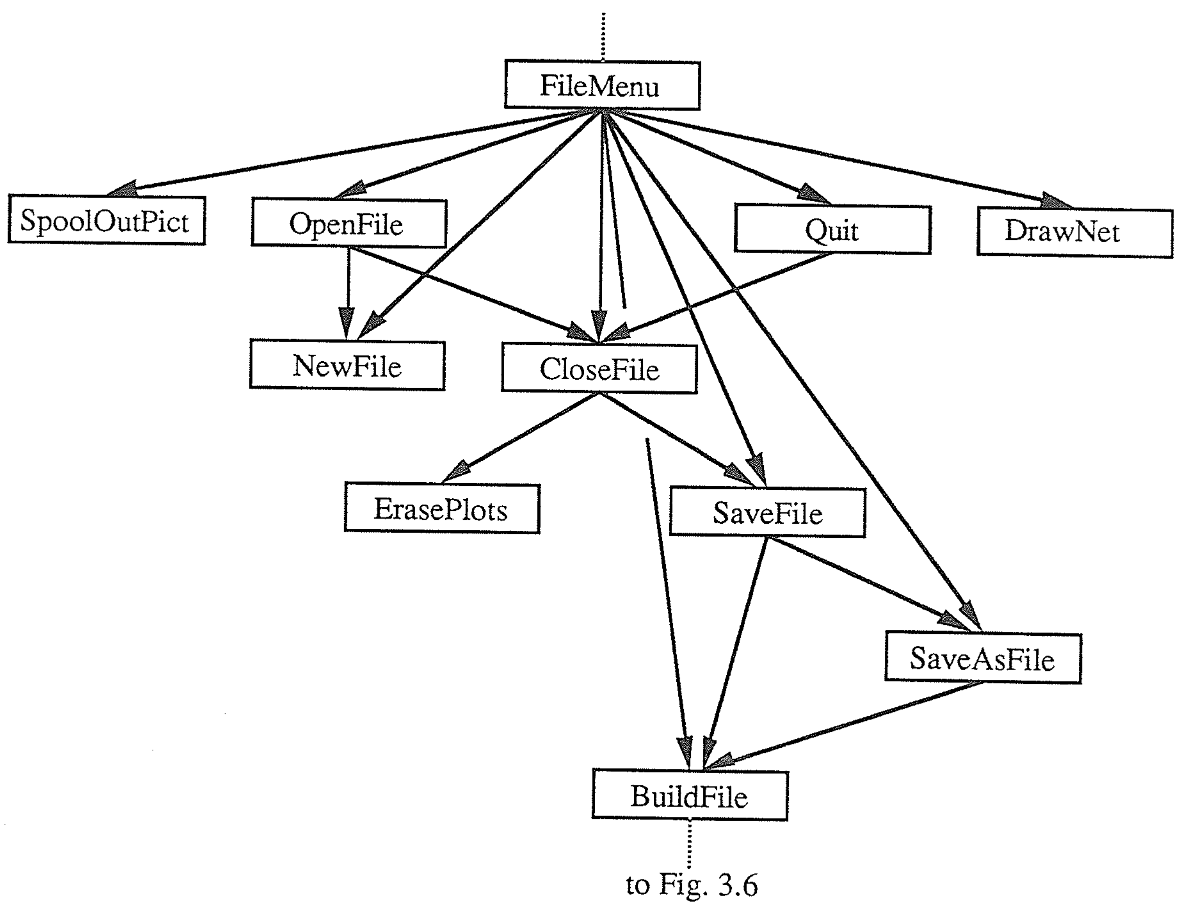


Fig. 7.3: Hierarchy of procedure FileMenu and its related routines.

7.7.1b The Element and Adaptor Menus

Combined, the **Element** and the **Adaptor** menus contain a selection for every type of WDF building block described in Section 3.1. For each building block type, there is an 'item' procedure that sets the *n*, *pict*, *nodeType*, and *gamma* fields of the active node's **Node** record (see Section 3.2). Each item procedure uses function ItemDialog which controls and reads the information from the orientation dialog boxes (Fig. 2.2). Function MultDialog is used to control and get the information from the multiplier dialog boxes. Function ThetaDialog is used to control and read the rotation angles from dialog boxes which display the inner structure of the elementary two-port sections (Fig. 2.3). The hierarchy of procedures ElementMenu and AdaptorMenu and their related routines is shown in Figures 7.4 and 7.5, respectively.

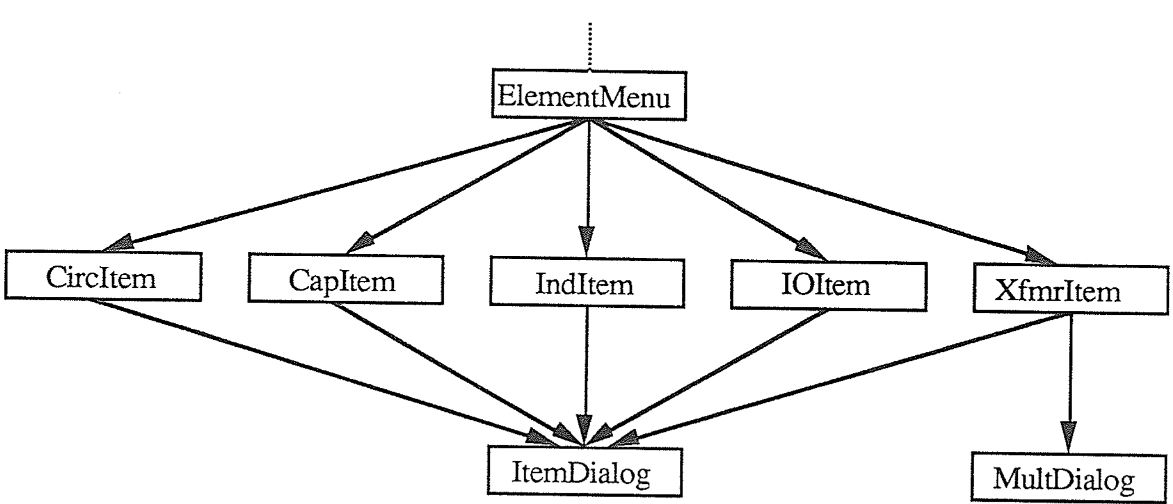


Fig. 7.4: Hierarchy of procedure ElementMenu and its related routines.

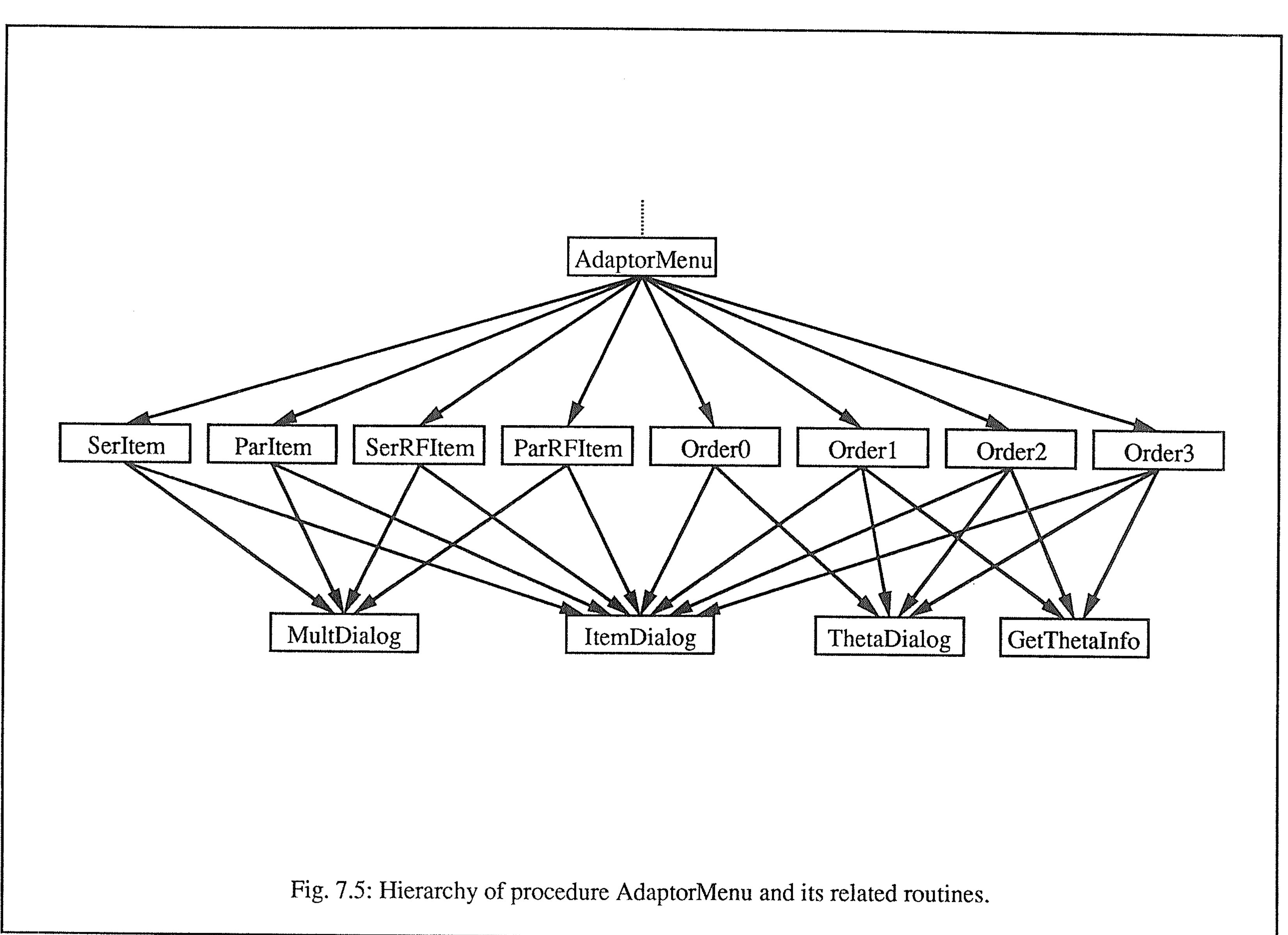


Fig. 7.5: Hierarchy of procedure AdaptorMenu and its related routines.

7.7.1c The Plot Menu

There are nine commands under the **Plot** menu: 1) **Set ymax**, 2) **Set ymin**, 3) **Set xmax**, 4) **Set xmin**, 5) **Set ydiv**, 6) **Set xdiv**, 7) **Toggle Grid**, 8) **Save As PICT**, and 9) **Save Data**. The first seven of these commands are used to set the plot parameters that are described in Section 4.4.1. The last two commands are used to save the information in the plot in two different formats. The command **Save As PICT** saves the plot graphically as a PICT file [25], which may then be printed and/or modified (e.g. label axes) by other Macintosh applications such as MacDraw or MacPaint. The command **Save Data** saves the raw data points (**PlotRec** record fields *xpts* and *ypts*) as a TEXT file, which may then be imported to other graphing packages such as CricketGraph via the Clipboard. The hierarchy of procedure PlotMenu and its associated routines is shown in Fig. 7.6.

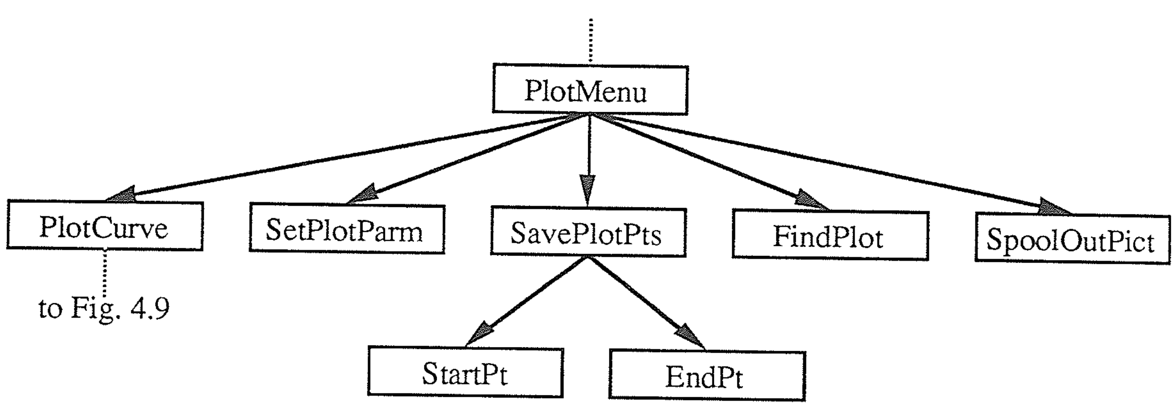


Fig. 7.6: Hierarchy of procedure PlotMenu and its associated routines.

7.1.2 Other Mouse Events

Besides invoking menu commands, the other mouse events of interest to WDFSim are the following: resizing windows, closing windows, and activating nodes in the File Window. Procedure ResizeWindow, through the use of Toolbox routines GrowWindow and SizeWindow, accomplishes the task of window resizing. Once the window has been resized, its contents must be redrawn. This is done in a two step process: 1) call Toolbox routine InvalRect which tells the system to generate an update event for the window; and 2) respond to the update event as described in Section 7.4.

When closing windows, procedure CloseWindow must first determine if it is a plot window or the File Window being closed (see Section 7.3). If a plot window is being closed, function FindPlot is first called to get a pointer to the window's **PlotRec** record. Procedure RemovePlot is then called to remove the plot from the plot list (see Section

4.4.1) and erase the window. If the File Window, and hence the file, is being closed, function CloseFile is called (see Fig. 7.3).

If the mouse button was clicked in the content region of the File Window, procedure ActivateNode is called by MouseEvent. The process of activating a node includes setting the global variables *ActiveNode* and *ActiveRect*, as well as highlighting the active node. If the **option** key was held down while clicking to activate a node, procedure ShowInfo is called. ShowInfo uses routines MultDialog and ThetaDialog to display the multiplier dialog boxes with the existing values filled in.

7.2 Key Events

There are two types of key events supported by WDFSim: deletion of nodes from the File Window by pressing the **delete** key, and key equivalents of menu commands. Using the *message* and *modifiers* fields of the **EventRecord**, as well as the BitAnd Toolbox routine, the exact combination of key(s) pressed is determined. If the **command** key was pressed, KeyEvent calls procedure MenuCmnd (see Section 7.1.1). If the **delete** key was pressed, the active node is deleted form the File Window. Any other combination of key presses is ignored.

7.3 Activate Events

Activate events are generated when a window is becoming active or inactive. WDFSim uses this type of event to control which menu items are active and which are inactive. From the *message* field of the **EventRecord**, a pointer to the window is created. The *modifiers* field of the **EventRecord** is used to determine if the window is becoming active or inactive.

Since different actions must be taken for different types of windows, some method is needed for distinguishing between them. To accomplish this, WDFSim uses the *refcon* field of the window's **WindowRecord** to hold a code according to the scheme shown in Table 7.1.

Table 7.1: Window types and corresponding *refcon* values.

Window Type	Value of *refcon* field
File Window	100
Plot Window of Node k	k
System Window	0

If any type of window is becoming inactive, procedure ActEvent disables all menus except for the **New** and **Open** commands under the **File** menu. If a plot window is becoming active, the **Plot** menu is enabled and all other menus, except for the **File** menu, are disabled. If the File Window is becoming active, all menus are enabled except the **Plot** and **Edit** menus, which are disabled. If a system window (desk accessory) is becoming active, all menus are disabled except the **Edit** menu, which is enabled.

7.4 Update Events

Update events are generated when a part of a window needs to be redrawn. For simplicity, when an update event occurs, WDFSim redraws the entire contents of the window. Procedure UpEvent first determines which window needs updating by using the *message* field of the **EventRecord** to get a pointer to the window. Next, the entire contents of the window is erased. If the window is the FileWindow, procedures DrawNet and DrawGrid are used to draw its contents. If the window is a plot window, function FindPlot is used to get a pointer to the **PlotRec**, and then procedure PlotCurve is called to draw the window's contents.

VIII. SIMULATION EXAMPLES

In this chapter, four examples that have appeared in previous literature are presented with the aim of demonstrating: 1) the correctness of the simulation program; 2) the different types of arithmetic available in WDFSim; and 3) the advantage of using voltage rather than power wave two-port adaptors in the elementary two-port sections.

8.1 Example 1: Fifth-Order Lowpass Filter [26]

The first example is a fifth-order elliptic lowpass filter taken from Meerkötter [26]. The analog reference filter is shown in Fig. 8.1, where

$$\begin{aligned} &R_1 = 1 \quad, R_2 = \frac{2}{3} \quad, R_3 = \frac{12}{5} \;, R_4 = \frac{4}{3} \quad, R_5 = \frac{12}{11} \;. \\ &R_6 = \frac{4}{13} \quad, R_7 = \frac{1}{3} \quad, R_8 = \frac{1}{5} \quad, R_9 = \frac{2}{5} \end{aligned} \tag{8.1}$$

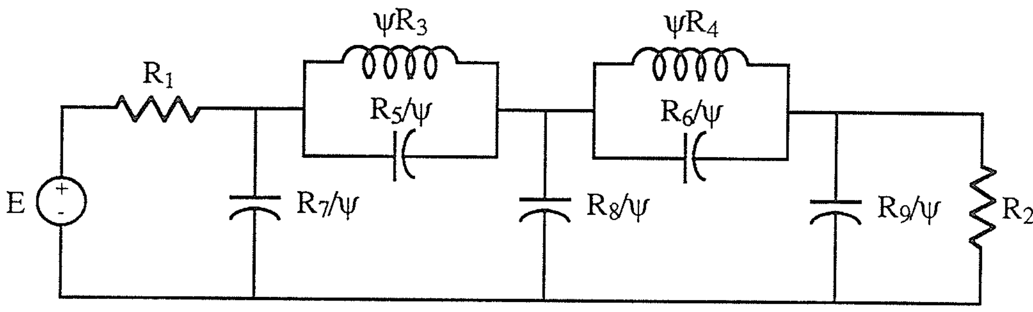


Fig. 8.1: Reference filter for Example 8.1.

The WD translation of Fig. 8.1 is shown in Fig. 8.2, where the additional port references are given by

$$\begin{aligned} &R_{11} = \frac{1}{G_3 + G_5} = \frac{3}{4} \quad, R_{12} = \frac{1}{G_4 + G_6} = \frac{1}{4} \;, R_{13} = \frac{1}{G_1 + G_7} = \frac{1}{4} \;. \\ &G_{14} = \frac{1}{R_{11} + R_{13}} = 1 \quad, G_{15} = \frac{1}{R_{12} + R_{16}} = 2, \; R_{16} = \frac{1}{G_9 + G_2} = \frac{1}{4} \end{aligned} \tag{8.2}$$

(Note that $G_k = 1/R_k$).

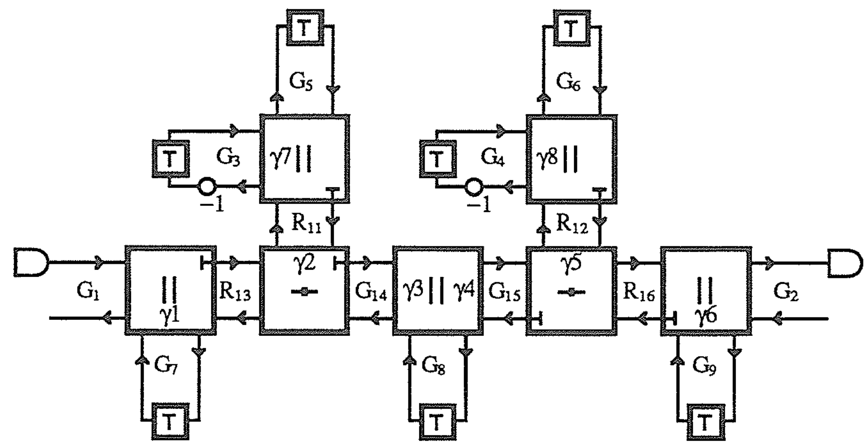


Fig. 8.2: The WDF for Example 8.1.

The multiplier values are calculated to be

$$\begin{aligned}
\gamma_1 &= \frac{G_7}{G_1 + G_7} = \frac{3}{4} \quad , \quad \gamma_2 = \frac{R_{11}}{R_{11} + R_{13}} = \frac{3}{4} \quad , \quad \gamma_3 = \frac{2G_{14}}{G_{14} + G_8 + G_{15}} = \frac{1}{4} \\
\gamma_4 &= \frac{2G_{15}}{G_{14} + G_8 + G_{15}} = \frac{1}{2} \quad , \quad \gamma_5 = \frac{R_{12}}{R_{12} + R_{16}} = \frac{1}{2} \quad , \quad \gamma_6 = \frac{G_9}{G_9 + G_2} = \frac{5}{8} \\
\gamma_7 &= \frac{G_3}{G_3 + G_5} = \frac{5}{16} \quad , \quad \gamma_8 = \frac{G_4}{G_4 + G_6} = \frac{3}{16}
\end{aligned} \tag{8.3}$$

The WDF shown in Fig. 8.2 was simulated using nominal values (which are actually 4-bit binary numbers) only. The results of the frequency response are shown in Fig. 8.3. The attenuation responses (Figures 8.3c and 8.3d) are identical to the results shown in [26, pp. 102].

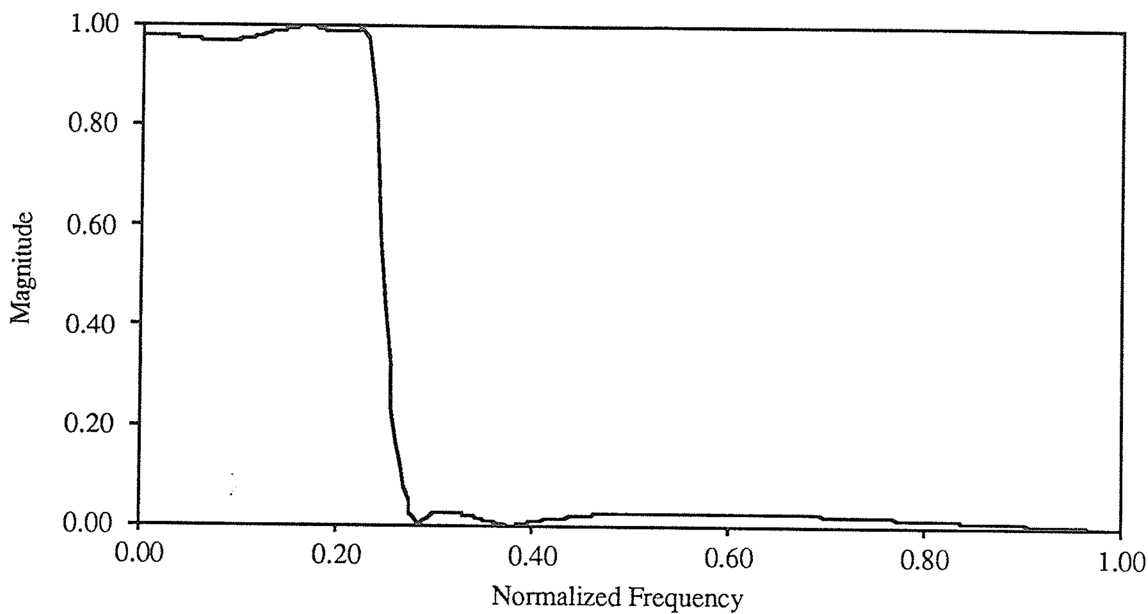


Fig. 8.3 (a): Magnitude response of Example 8.1.

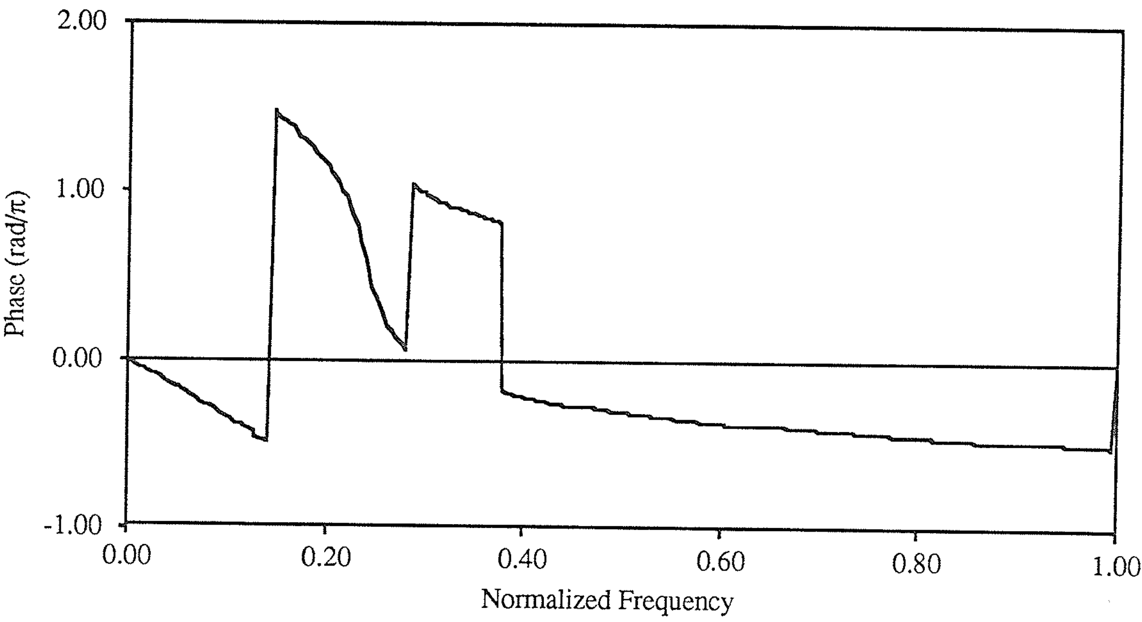


Fig. 8.3 (b): Phase response of Example 8.1.

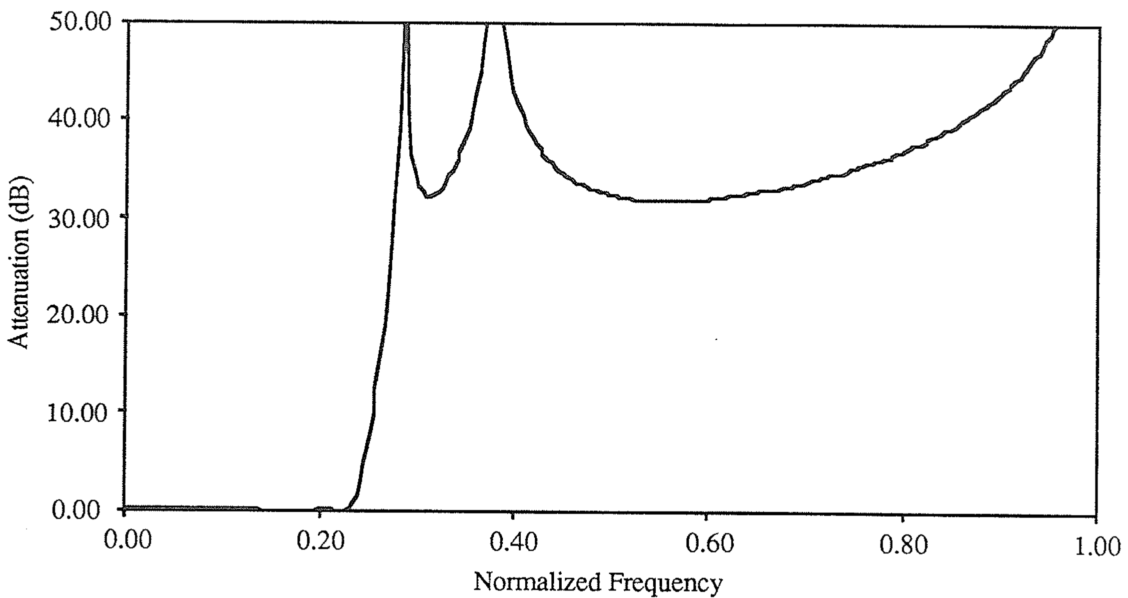


Fig. 8.3 (c): Fullband attenuation response of Example 8.1.

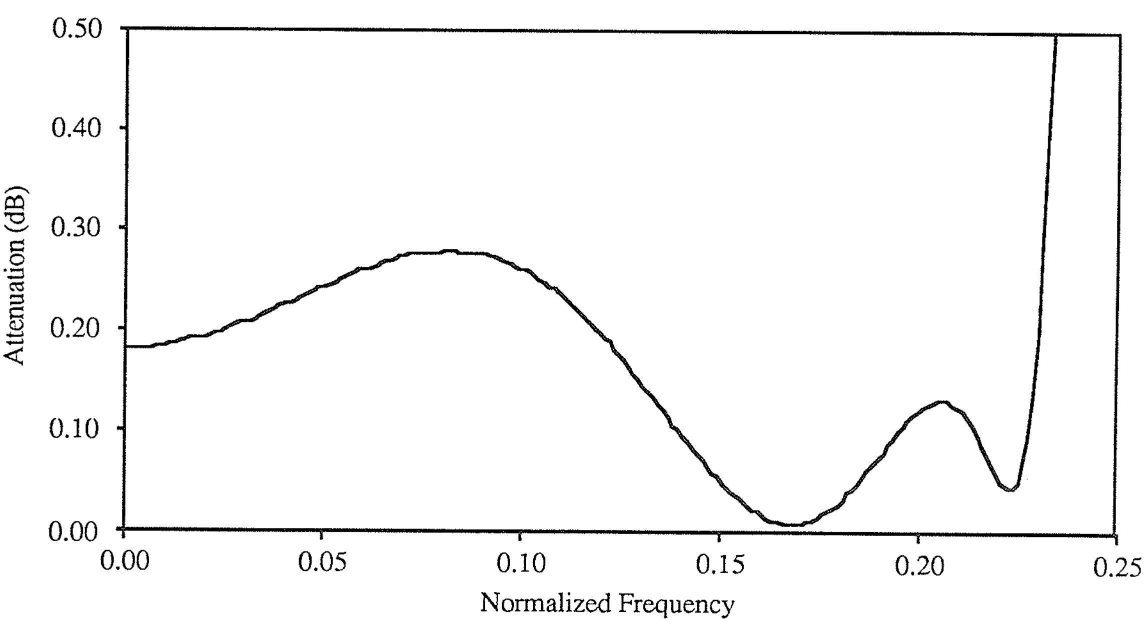


Fig. 8.3 (d): Passband attenuation response of Example 8.1.

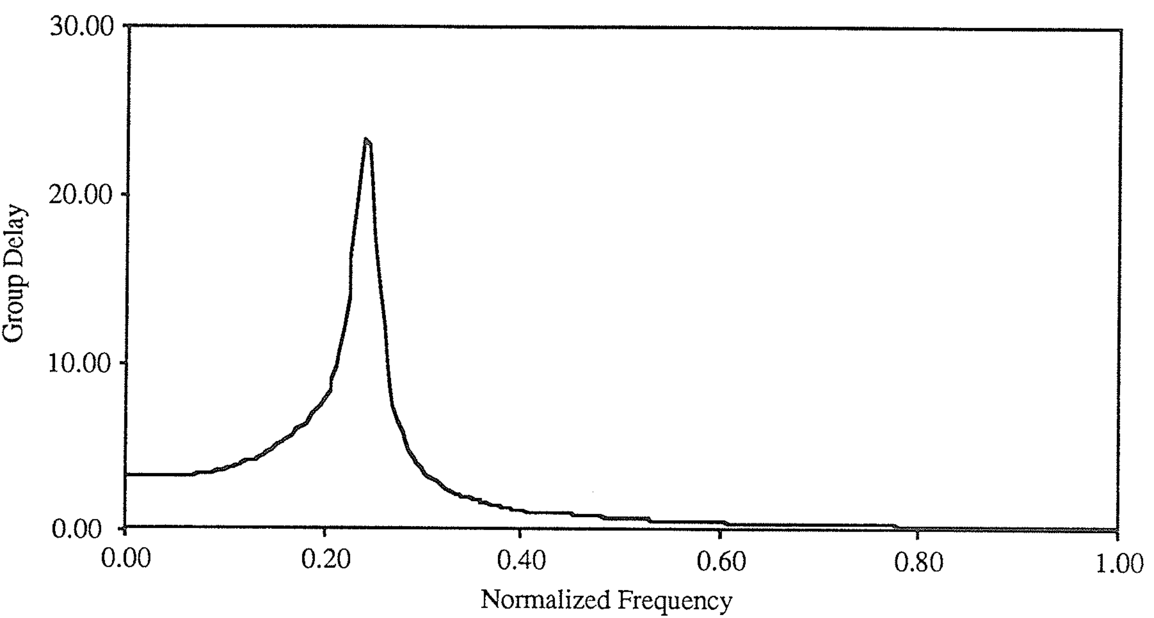


Fig. 8.3 (e): Group delay response of Example 8.1.

8.2 Example 2: Seventh-Order Lowpass Filter [26]

The second example, a seventh-order elliptic lowpass filter, also originates from Meerkötter [26]. The analog reference filter is shown in Fig. 8.4, where the element values are

$$R_1 = R_2 = R_3 = R_{12} = 1 \quad , \; R_4 = R_5 = R_7 = R_8 = R_9 = 3$$
$$R_6 = R_{10} = \frac{1}{3} \quad , \; R_{11} = \frac{2}{7} \quad . \tag{8.4}$$

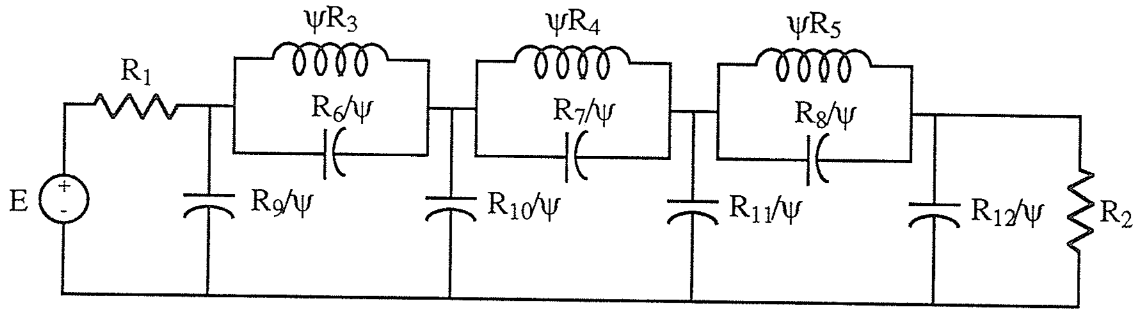


Fig. 8.4: Reference filter for Example 8.2.

The WD translation of Fig. 8.4 is shown in Fig. 8.5, where the additional port references are given by

$$R_{13} = \frac{1}{G_1 + G_9} = \frac{3}{4} \quad , \; G_{14} = \frac{1}{R_{19} + R_{13}} = 1 \quad , \; R_{15} = \frac{1}{G_{10} + G_{14}} = \frac{1}{4}$$
$$R_{16} = \frac{1}{G_{11} + G_{17}} = \frac{1}{4} \quad , \; G_{17} = \frac{1}{R_{18} + R_{21}} = \frac{1}{2} \quad , \; R_{18} = \frac{1}{G_{12} + G_2} = \frac{1}{2} \tag{8.5}$$
$$R_{19} = \frac{1}{G_3 + G_6} = \frac{1}{4} \quad , \; R_{20} = \frac{1}{G_7 + G_4} = \frac{3}{2} \quad , \; R_{21} = \frac{1}{G_5 + G_8} = \frac{3}{2}$$

The multiplier values are calculated to be

$$\gamma_1 = \frac{G_9}{G_1 + G_9} = \frac{1}{4} \quad , \; \gamma_2 = \frac{R_{19}}{R_{19} + R_{13}} = \frac{1}{4} \quad , \; \gamma_3 = \frac{G_{10}}{G_{14} + G_{10}} = \frac{3}{4}$$
$$\gamma_4 = \frac{2R_{15}}{R_{15} + R_{20} + R_{16}} = \frac{1}{4}, \; \gamma_5 = \frac{2R_6}{R_{15} + R_{20} + R_{16}} = \frac{1}{4}, \; \gamma_6 = \frac{G_{11}}{G_{11} + G_{17}} = \frac{7}{8}$$
$$\gamma_7 = \frac{R_{21}}{R_{21} + R_{18}} = \frac{3}{4} \quad , \; \gamma_8 = \frac{G_{12}}{G_{12} + G_2} = \frac{1}{2} \quad , \; \gamma_9 = \frac{G_3}{G_3 + G_6} = \frac{1}{4} \tag{8.6}$$
$$\gamma_{10} = \frac{G_4}{G_4 + G_7} = \frac{1}{2} \quad , \; \gamma_{11} = \frac{G_5}{G_5 + G_8} = \frac{1}{2}$$

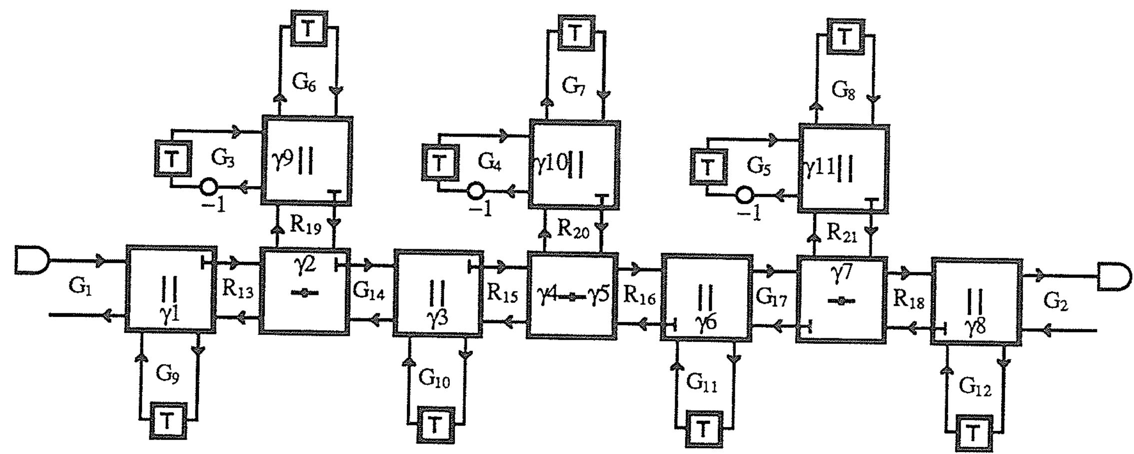


Fig. 8.5: The WDF for Example 8.2.

The WDF specified by Fig. 8.5 and equation (8.6) was simulated using nominal values (which are actually 3-bit binary numbers). The results of the frequency response are shown in Fig. 8.6. Again, the attenuation responses (Figures 8.3c and 8.3d) are identical (except for a scaling factor) to Meerkötter's results [26, pp. 105].

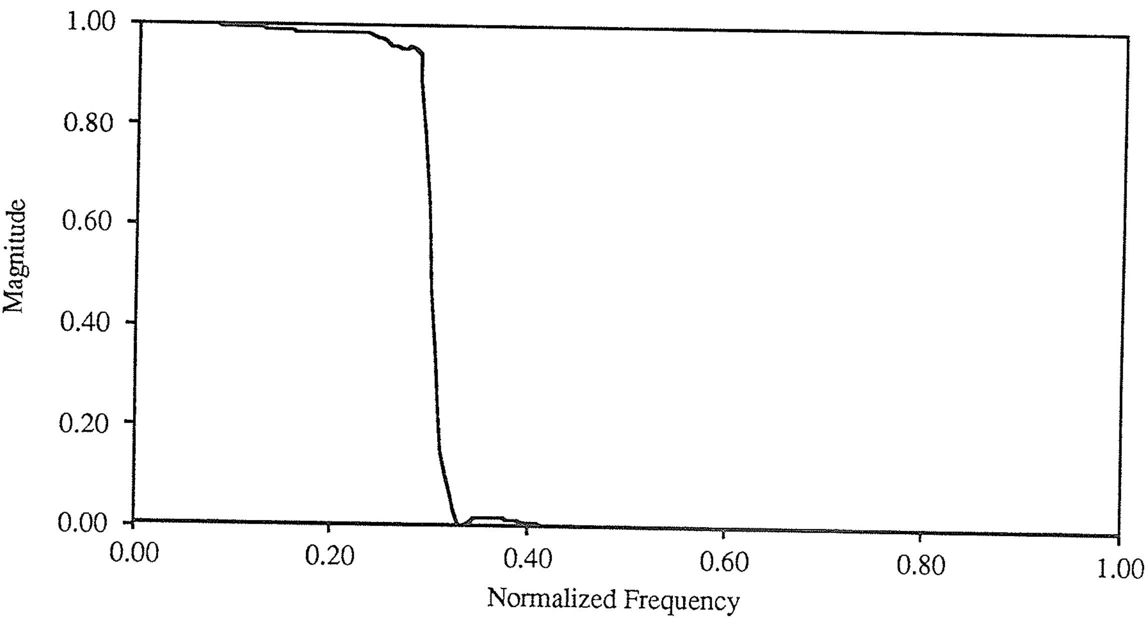


Fig. 8.6 (a): Magnitude response of Example 8.2.

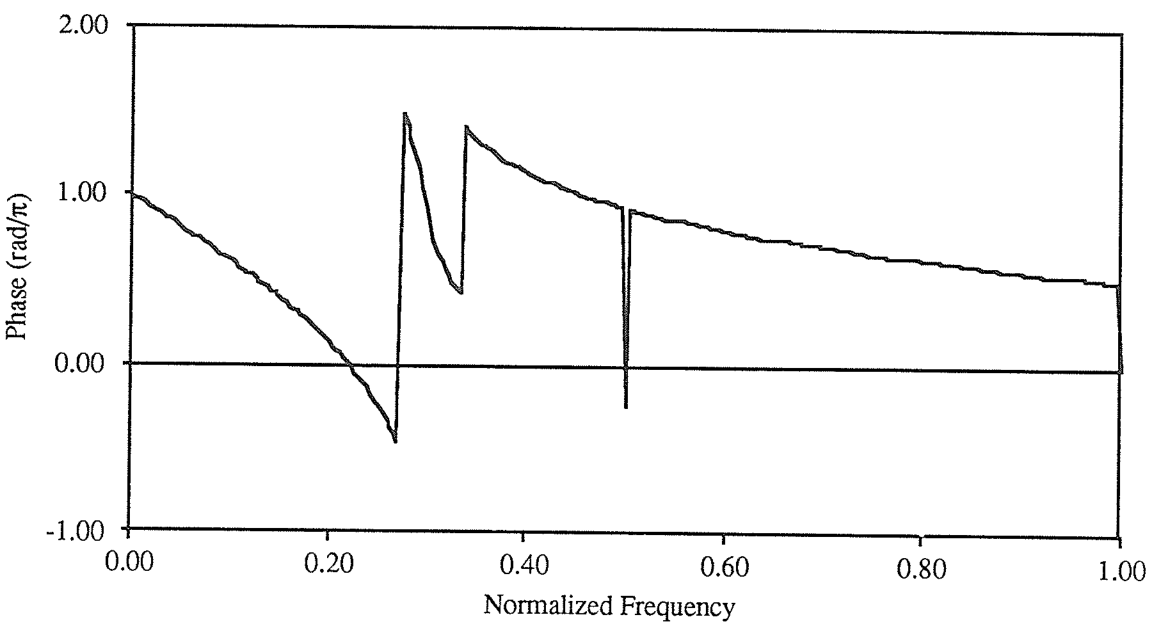


Fig. 8.6 (b): Phase response of Example 8.2.

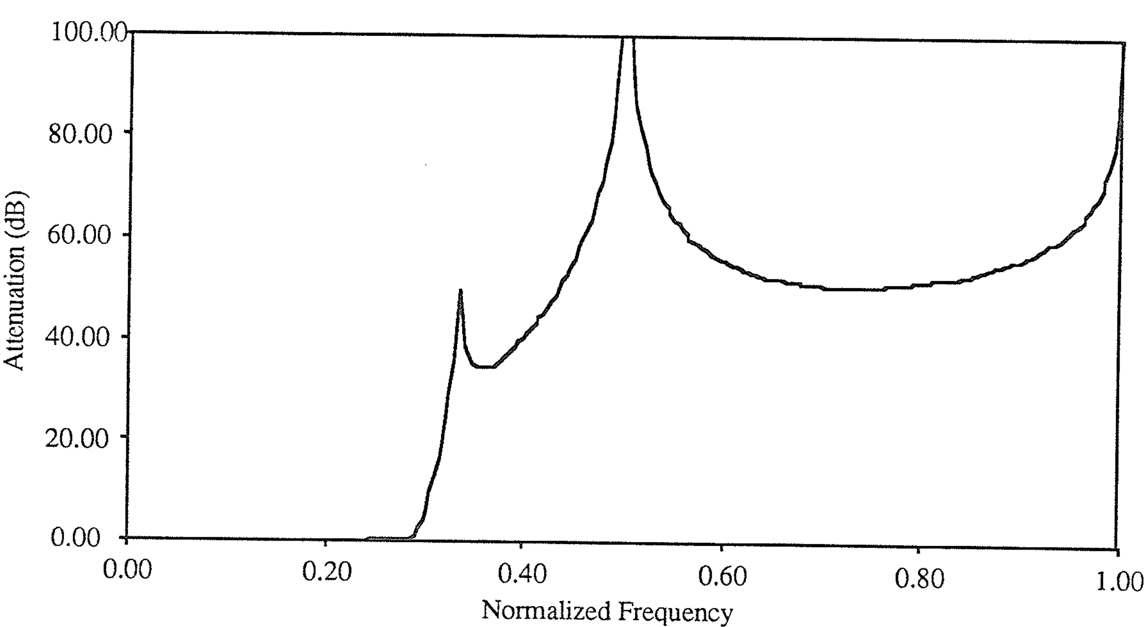


Fig. 8.6 (c): Fullband attenuation response of Example 8.2.

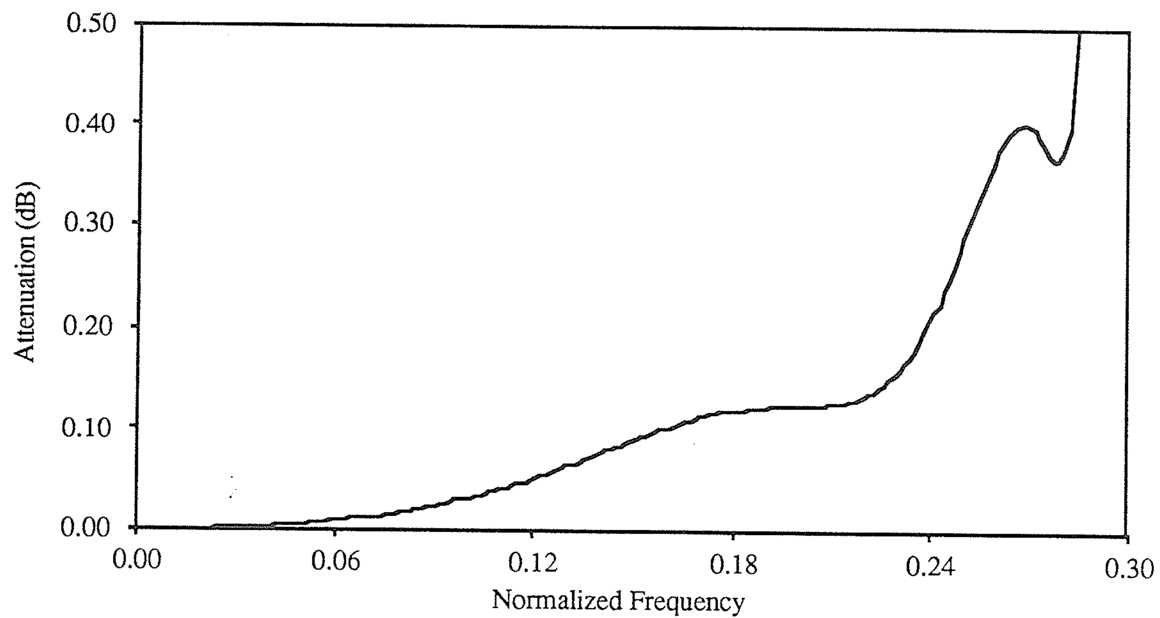


Fig. 8.6 (d): Passband attenuation response of Example 8.2.

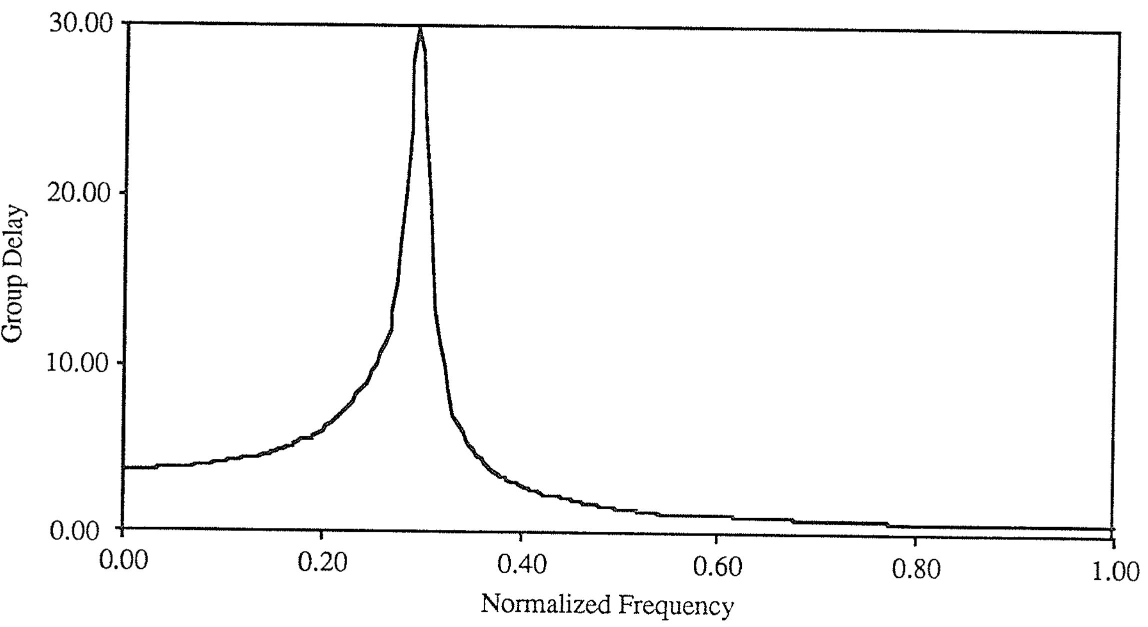


Fig. 8.6 (e): Group delay response of Example 8.2.

8.3 Example 3: Seventh-Order Lowpass Filter [27,4]

The third example is also a seventh-order elliptic lowpass filter, originally appearing as a bandpass filter in [27, Ex.7] and later in [4, Ex.5.3]. The canonic analog reference filter is shown in Fig. 8.7, where

$$R_1 = R_2 = \frac{1}{4} \; , \; R_3 = R_5 = \frac{3}{4}, \; R_4 = \frac{23}{36} \; , \; R_6 = R_8 = \frac{3}{16}$$
$$R_7 = \frac{46}{417} \; , \; R_9 = \frac{3}{58} \; , \; n_1 = n_2 = \frac{1}{4} \; , \; n_3 = \frac{1}{2} \; . \tag{8.7}$$

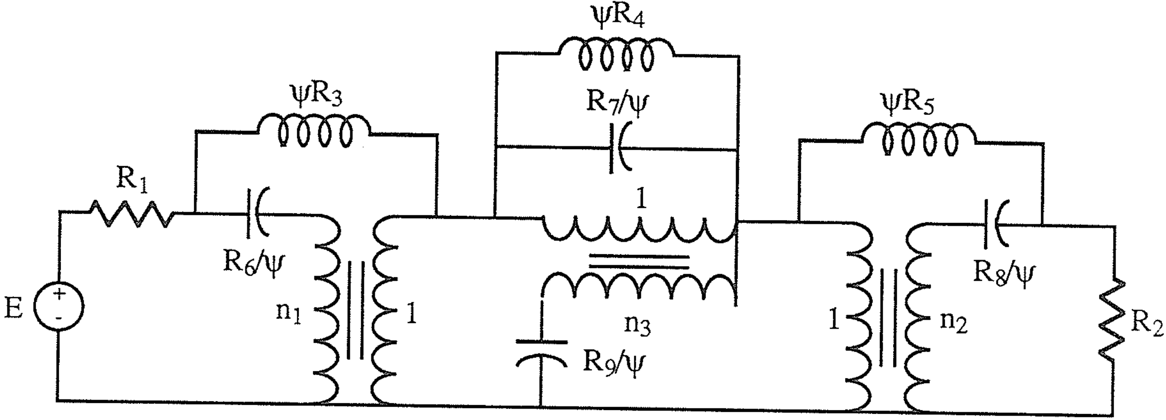


Fig. 8.7: Reference filter for Example 8.3.

The filter was realized in the WD domain by a cascade of two **2D** sections and a **3QL** section as shown in Fig. 8.8. The rotation angles [28] are given in Table 8.1, and the simulated attenuation response (nominal) is shown in Fig. 8.9.

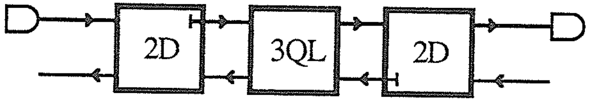


Fig. 8.8: The WDF for Example 8.3.

Table 8.1: The rotation angles for the WDF in Fig. 8.8.

Section	θ_1	θ_2	θ_3	θ_4
2D	-1.0471975512	1.5707963268	3.1415926536	
3QL	0.7853981634	0.4271352472	2.3539550307	0.6223684886
2D	-1.0471975512	1.5707963268	-3.1415926536	

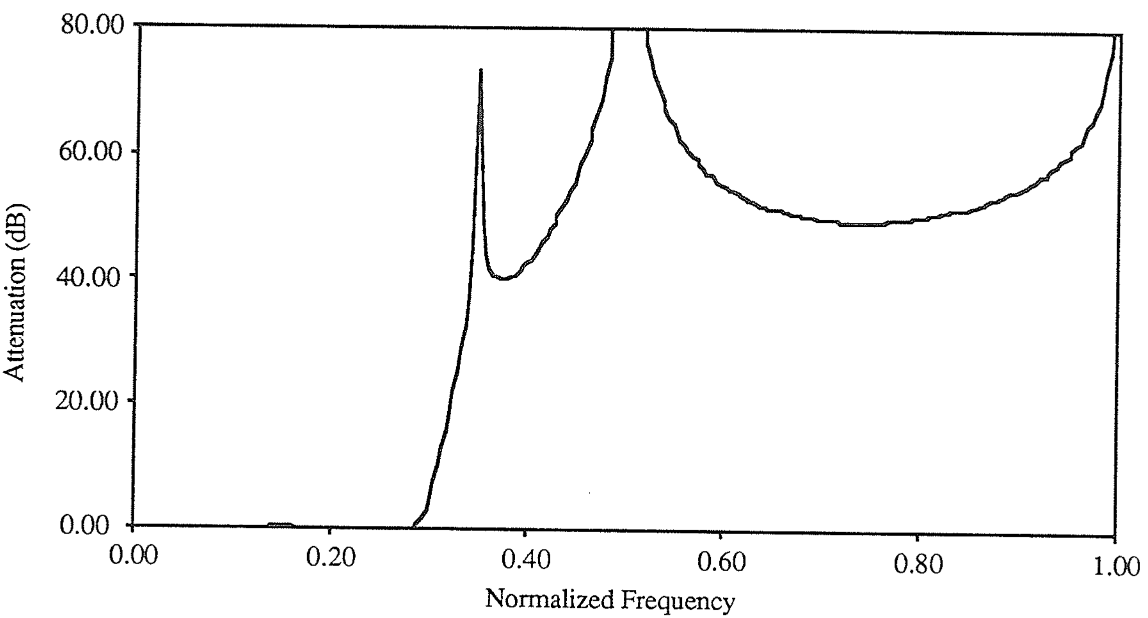


Fig. 8.9 (a): Fullband attenuation response for the WDF of Example 8.3.

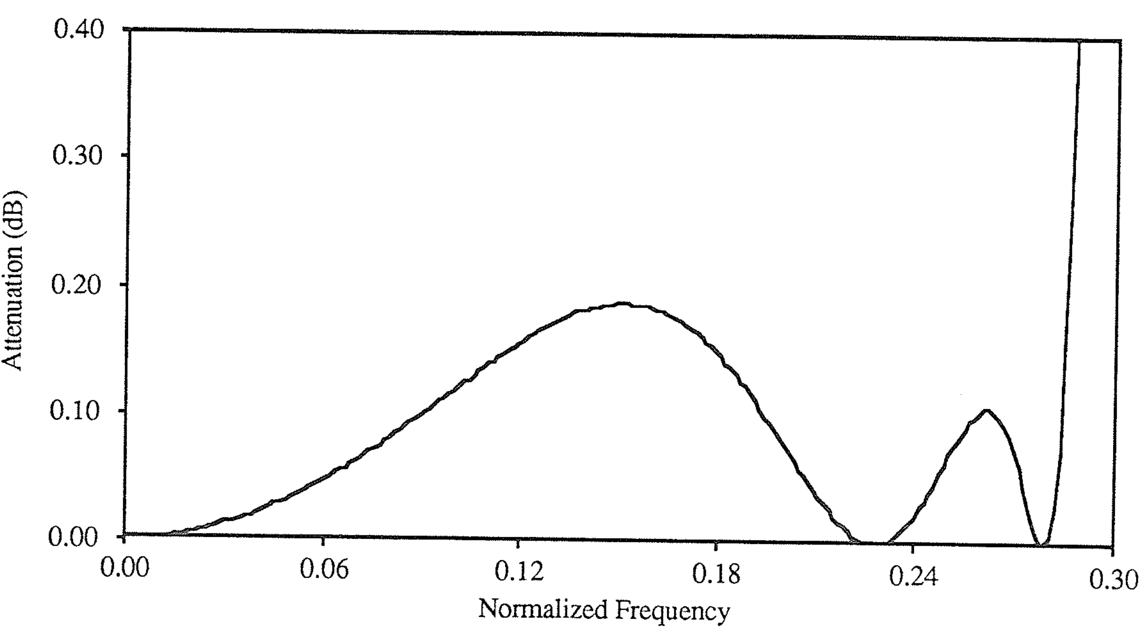


Fig. 8.9 (b): Passband attenuation response for the WDF of Example 8.3.

To demonstrate the different types of arithmetic that WDFSim can simulate, consider the rectangular pulse shown in Fig. 8.10a as input to the filter. The rectangular signal consists of 40 samples of value 0, followed by 41 samples of value 0.8, followed by 120 samples of value 0. The nominal response of the filter to this input signal is shown in Fig. 8.10b. The output of the filter was calculated using several combinations of binary arithmetic with 10 total bits and 2 integer bits. The results for truncation, rounding, and magnitude truncation multiplication with overflow addition are shown in Fig. 8.11. No noticeable differences could be detected using saturation, rather than overflow addition. As expected, magnitude truncation provided the best results since both other cases broke into limit cycles. The filter was also simulated using RNS arithmetic for two sets of moduli. These results, shown in Fig. 8.12, appear to resemble the results obtained using magnitude truncation multiplication.

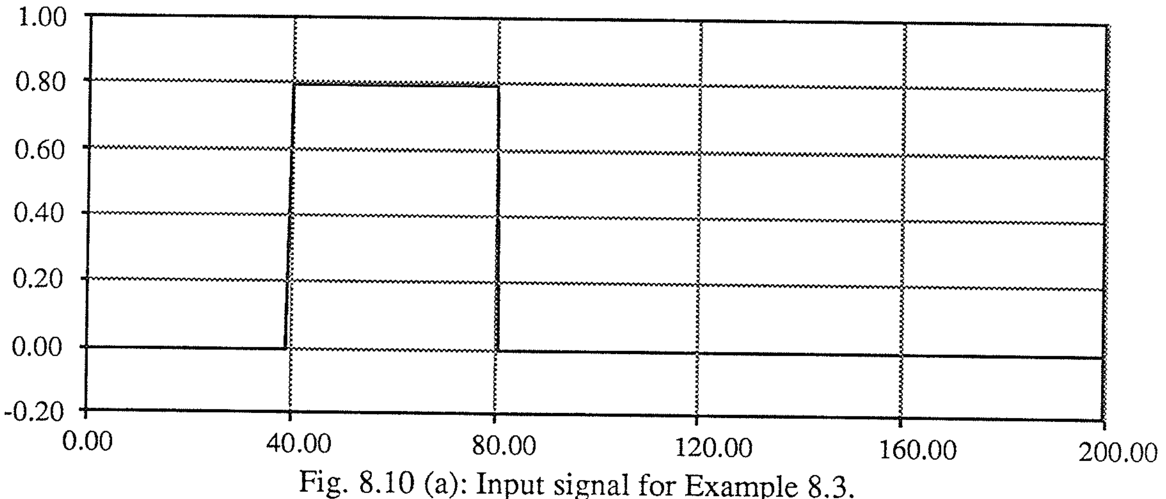


Fig. 8.10 (a): Input signal for Example 8.3.

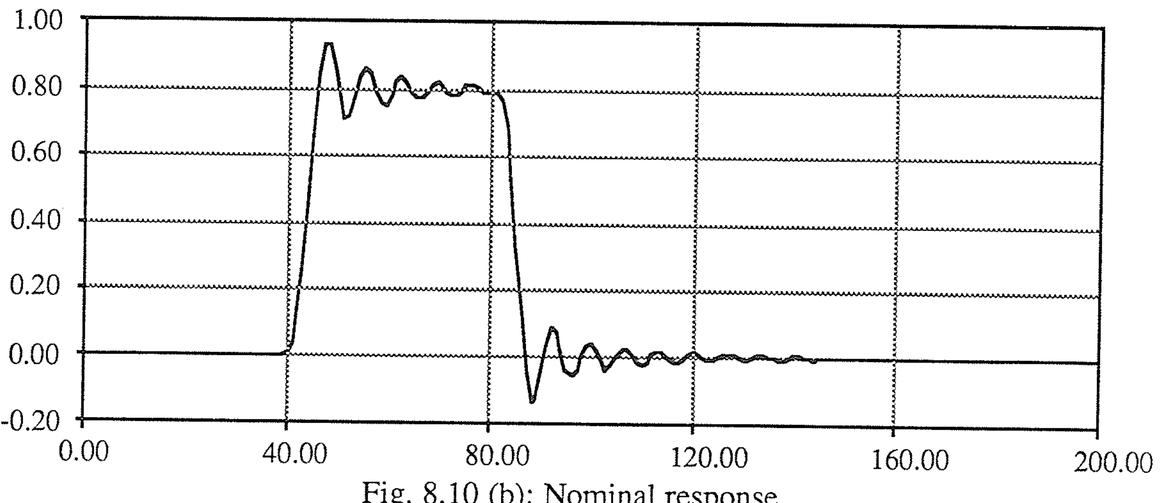


Fig. 8.10 (b): Nominal response.

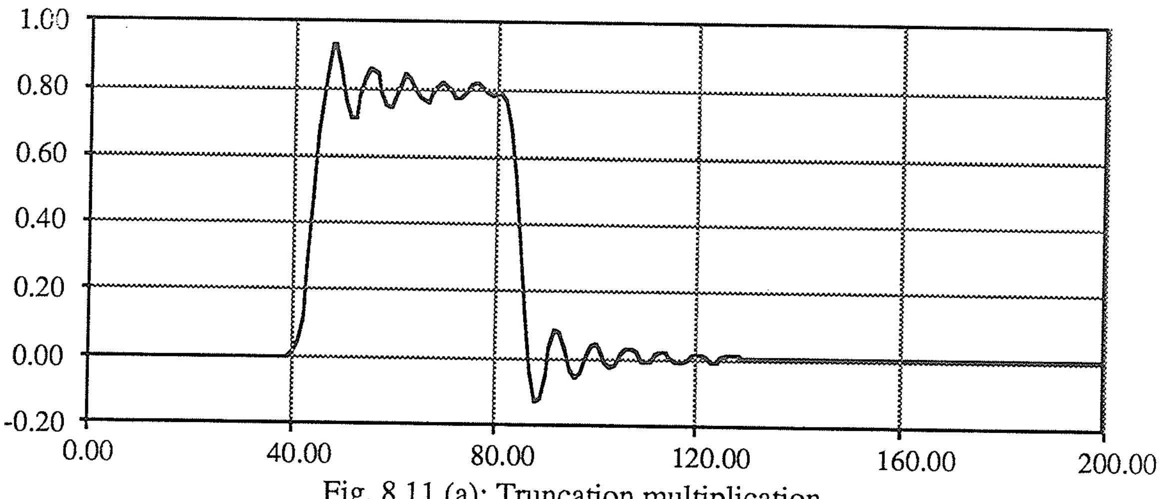


Fig. 8.11 (a): Truncation multiplication.

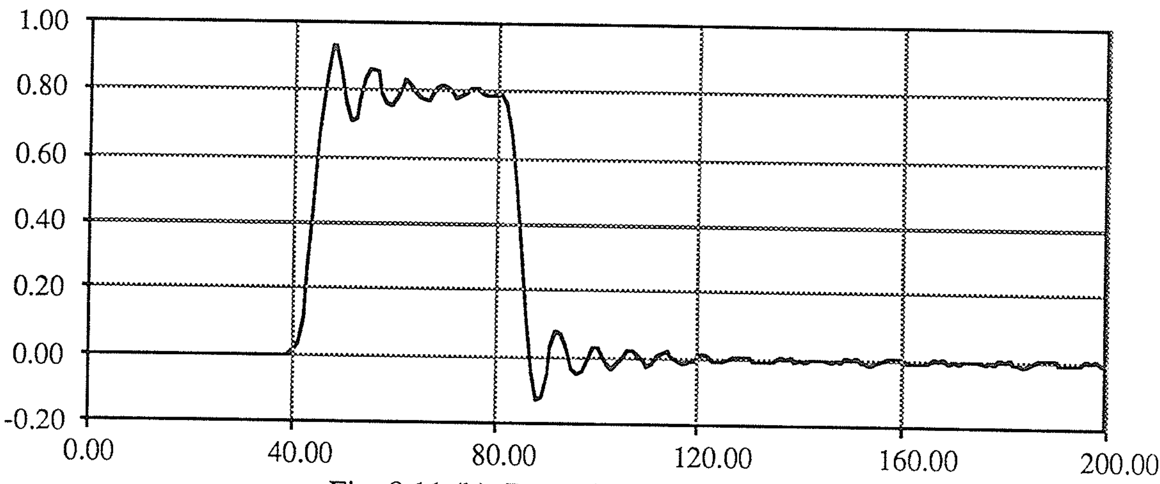


Fig. 8.11 (b): Rounding multiplication.

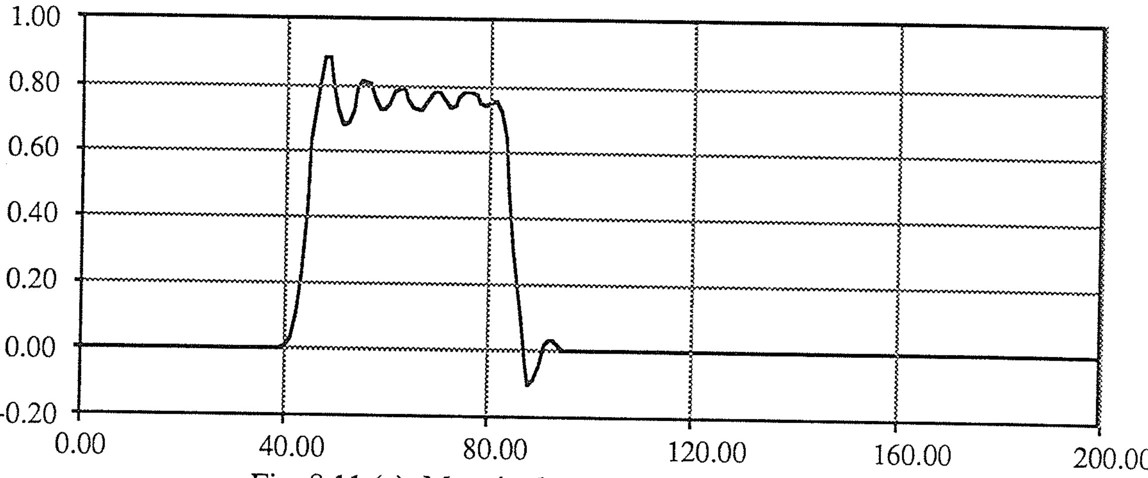


Fig. 8.11 (c): Magnitude truncation multiplication.

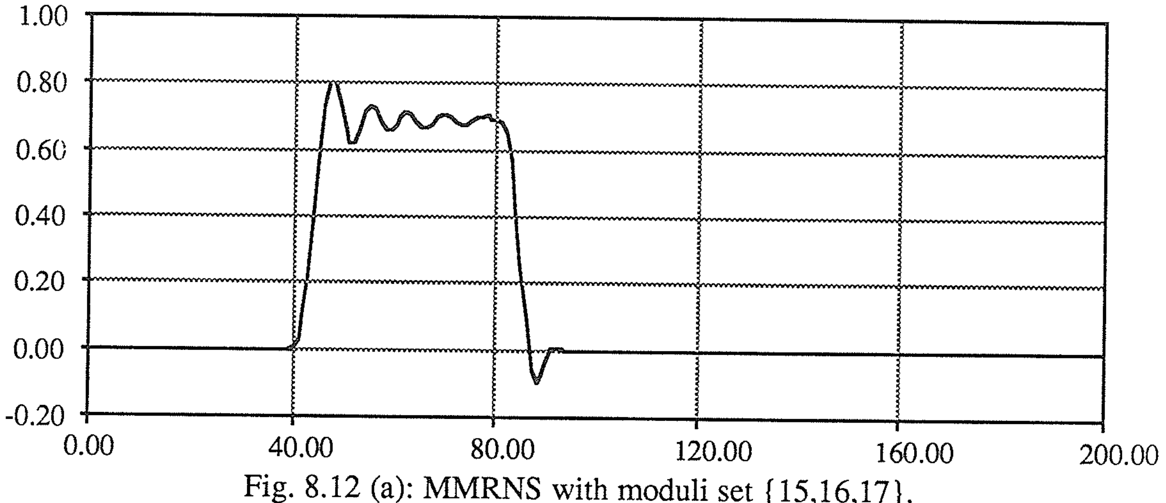


Fig. 8.12 (a): MMRNS with moduli set {15,16,17}.

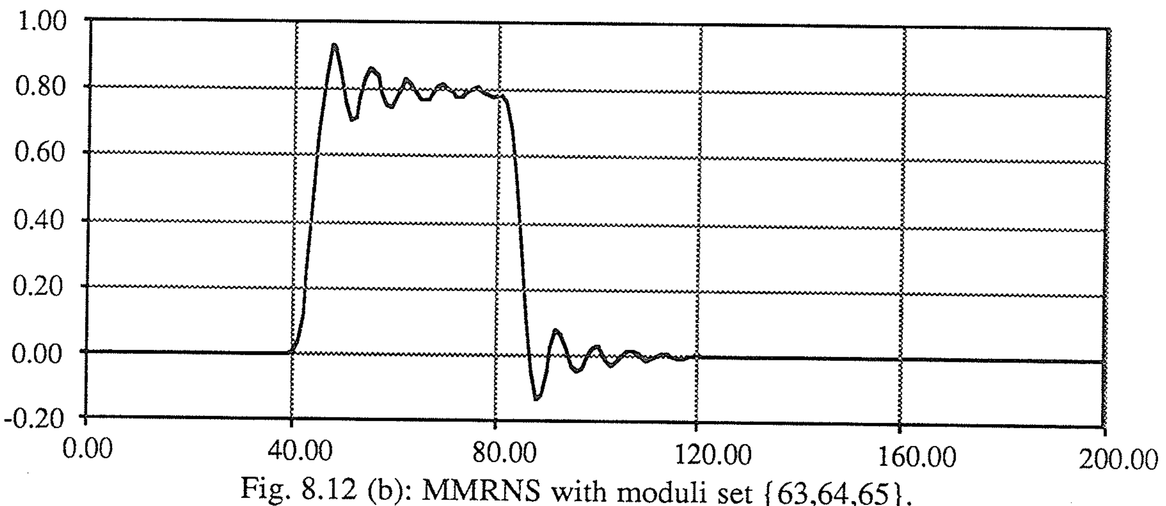


Fig. 8.12 (b): MMRNS with moduli set {63,64,65}.

8.4 Example 4: Eighth-Order Bandpass Filter [29,4]

The fourth, and final example is an eighth-order bandpass filter that originally appeared in [29, Ex.7.1], and more recently in [4, Ex. 5.2]. Since WDFs provide two complementary transfer functions, it is possible to realize a given transfer function in either of two ways: as a reflectance, or as a transmittance. WDFs realized as a reflectance are less sensitive to multiplier quantization in the passband than WDFs realized as a transmittance. Conversely, WDFs realized as a transmittance are less sensitive to multiplier quantization in the stopband than filters realized as a reflectance. This property of WDFs is demonstrated in this example.

8.4.1 Example 4 Realized as a Reflectance

The eighth-order bandpass filter for this example was realized in the WD domain as a reflectance by the cascade of four **2D** elementary sections shown in Fig. 8.13. The nominal rotation angles [4] are given in Table 8.2.

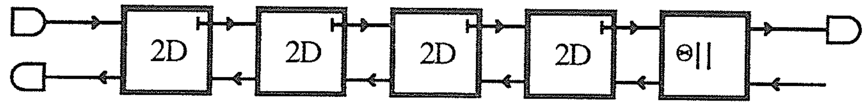


Fig. 8.13: The WDF for Example 8.4 realized as a reflectance.

Table 8.2: The nominal rotation angles for the WDF in Fig. 8.13.

Section	θ_1	θ_2	θ_3
2D	-0.1304969843	1.3440695213	2.9025334491
2D	-0.2138635968	1.8040203957	0.1904923373
2D	-0.2150724367	-1.2039394241	3.1354473920
2D	-0.1312700773	-2.1275196879	0.2372413538
Final Two-Port	1.571701537		

In order to compare the effects of multiplier quantization on the frequency response, three types of simulations were performed: 1) nominal response; 2) power wave with multipliers quantized to 8 bits; and 3) voltage wave with multipliers quantized to 8 bits. The procedure described in Section 5.3 yielded the parameters given in Table 8.3 for the voltage wave simulation. Note that $\hat{\theta}_3$ for the third **2D** section is equal to π, which has a decoupling effect. The resulting structure is similiar to the **2B** section and the multipliers m and n cancel each other out .

Table 8.3: Parameters for the voltage wave simulation.

Section	$\widehat{\theta_1}$	$\widehat{\theta_2}$	$\widehat{\theta_3}$	n	m
2D	-0.1250815236	1.3422493741	2.9072028921	1.00117056	0.99846502
2D	-0.2169314605	1.8033557002	0.1979654592	1.00460000	0.99542106
2D	-0.2169314605	-1.2030982632	3.1415926536	1.00000000	1.00000000
2D	-0.1250815236	-2.1262675683	0.2343897615	1.00281278	0.99667777
Final Two-Port	1.5707963268				

The attenuation responses resulting from the three simulations are shown in Fig. 8.14. As expected, the results from the quantized voltage wave sections were superior to those of the power wave sections. However, both quantized versions deteriorated significantly in the stopband.

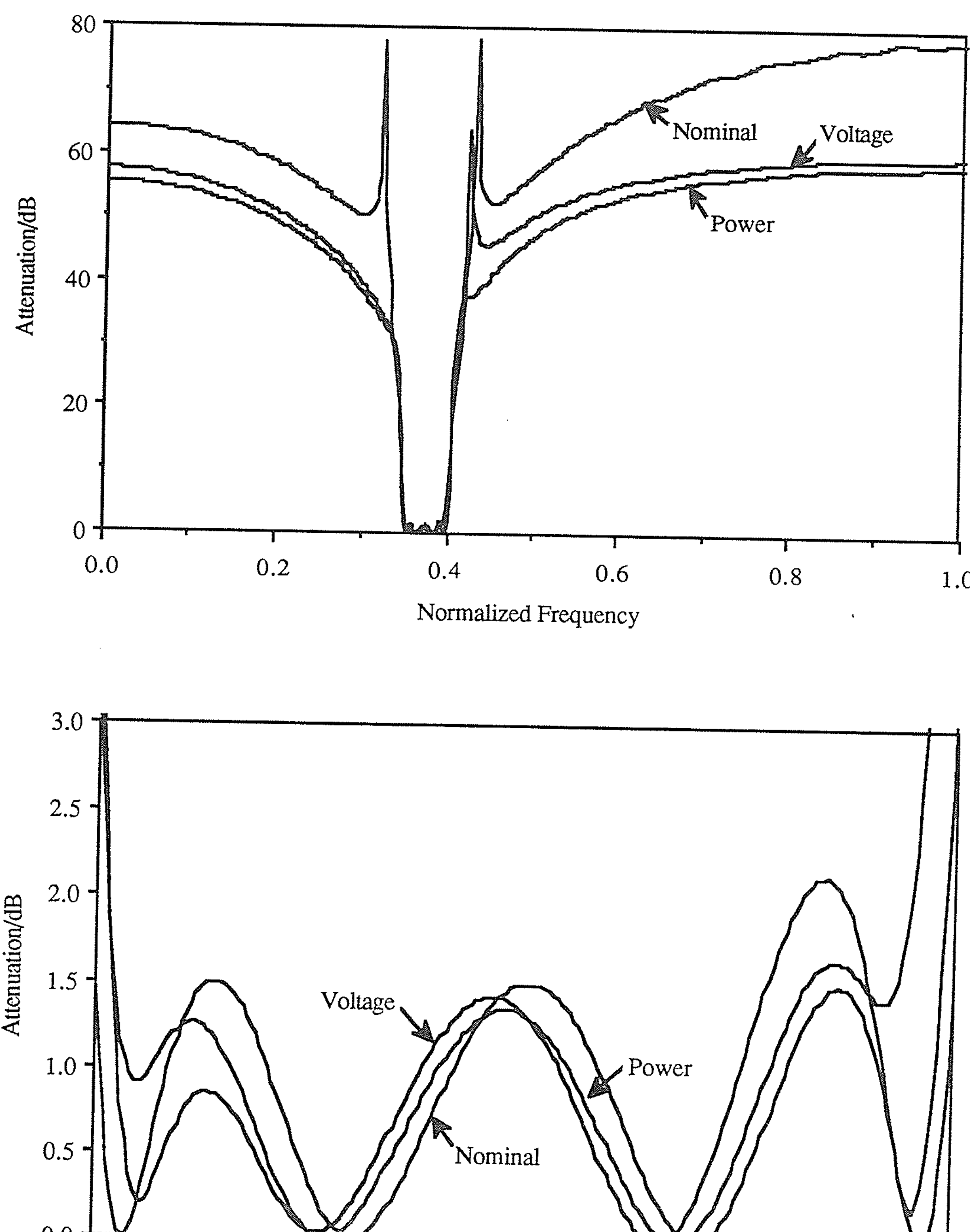


Fig. 8.14: Stopband and passband attenuation responses for Example 8.4 realized as a reflectance.

8.4.2 Example 4 Realized as a Transmittance

The WDF for Example 8.4 was realized as a transmittance by the cascade of **2D** and **1E** elementary sections shown in Fig. 8.15. The nominal rotation angles [4] for this realization are given in Table 8.4.

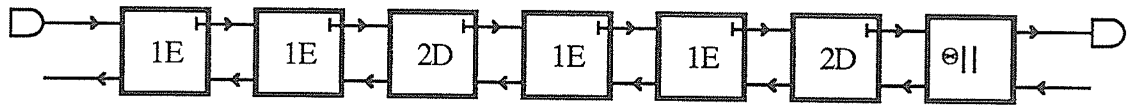


Fig. 8.15: The WDF for Example 8.4 realized as a transmittance.

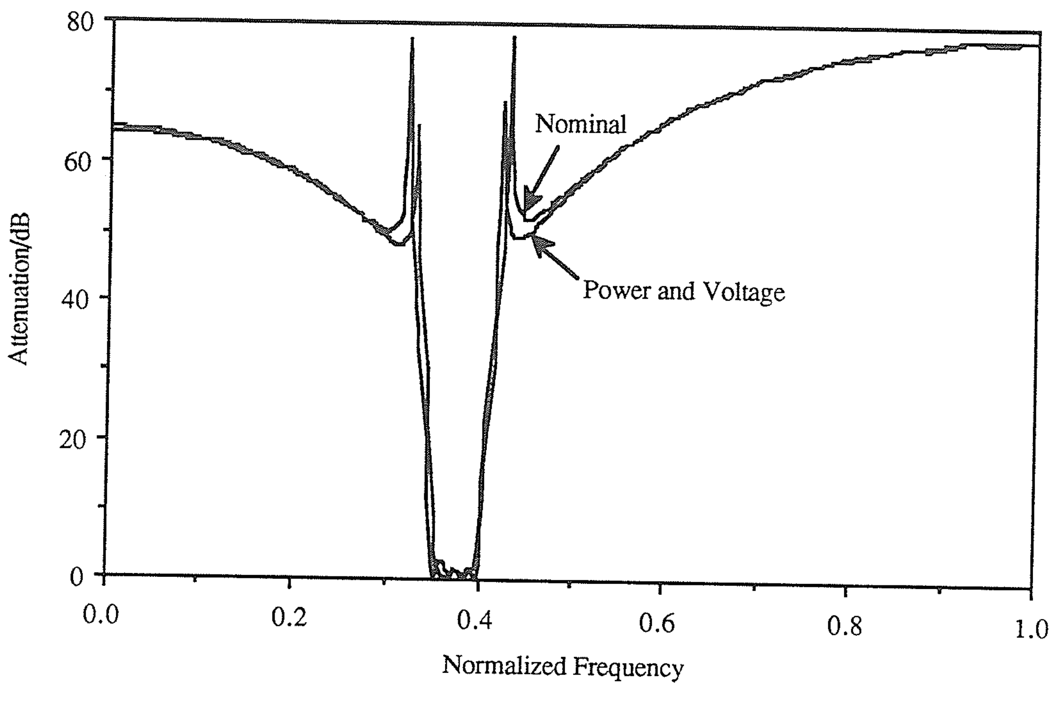
Table 8.4: The nominal rotation angles for the WDF in Fig. 8.15.

Section	θ_1	θ_2	θ_3
1E	1.9235002470	1.5707963268	
1E	2.7495953317	1.5707963268	
2D	-1.2639903262	3.2438692236	1.1702487683
1E	1.6610724096	1.5707963268	
1E	2.7558031796	1.5707963268	
2D	-1.2393643680	3.0436480777	1.1780649673
Final Two-Port	2.7888887334		

Again, the three types of simulations described in the preceeding section were performed. The parameters for the voltage wave simulation are given in Table 8.5. The results of the three simulations are shown in Fig. 8.16. Again, the results of the voltage wave realization were superior in the passband. Any differences between the two quantized simulations were indistinguishable in the stopband. Upon examining Figures 8.14 and 8.16, we can see the expected differences in the sensitivities in the passband and the stopband of the two types of realizations (reflectance and transmittance).

Table 8.5: Parameters for the voltage wave simulation.

Section	$\widehat{\theta_1}$	$\widehat{\theta_2}$	$\widehat{\theta_3}$	n	m
1E	1.9262175285	1.5707963268		1.00000000	1.00000000
1E	2.7487538661	1.5707963268		1.00000000	1.00000000
2D	-1.2652845947	-3.0531755084	1.1694858898	0.99370324	1.00632130
1E	1.6592134720	1.5707963268		1.00000000	1.00000000
1E	2.7571958791	1.5707963268		1.00000000	1.00000000
2D	-1.2406105770	3.0531755084	1.1779575393	1.00287387	0.99700914
Final Two-Port	2.7861714519				



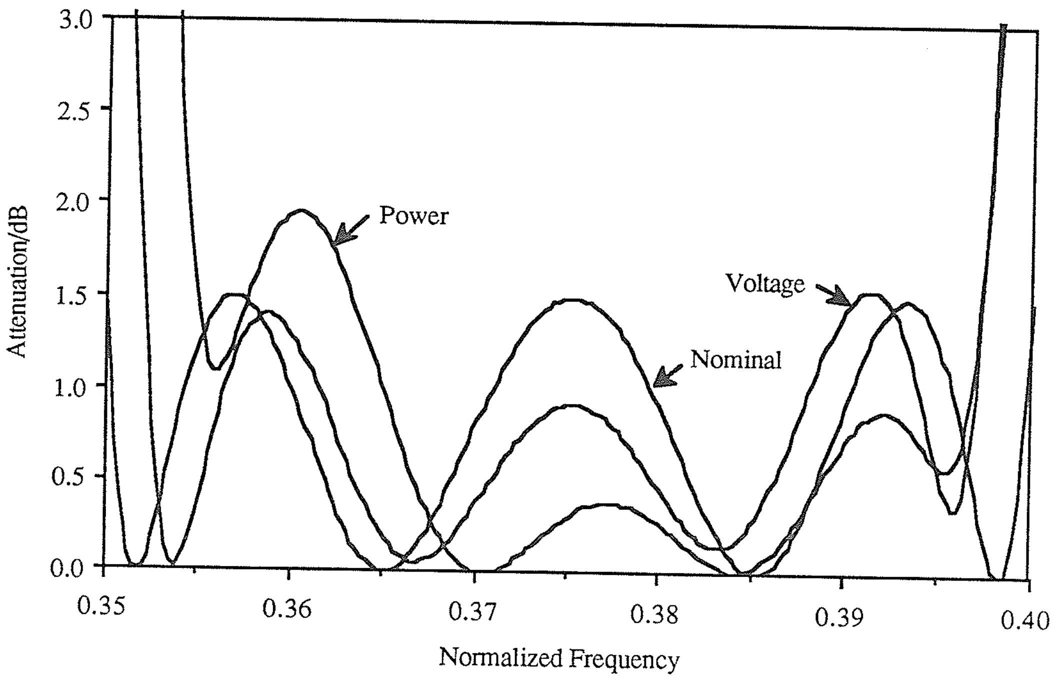


Fig. 8.16: Stopband and passband attenuation responses for Example 8.4 realized as a transmittance.

IX. CONCLUSIONS AND RECOMMENDATIONS

A program, called WDFSim, that simulates the operation of a WDF in the time domain has been designed, implemented, and tested. WDFSim was developed on the Macintosh computer using LightSpeed Pascal. Like most Macintosh applications, WDFSim takes full advantage of the Macintosh Toolbox routines in its man-machine interface. The combination of pull-down menus, dialog boxes, graphics, and windowing makes WDFSim very user-friendly.

Simulation may be performed using full precision (nominal) values in order to verify WDF designs. Once a design has been determined to be correct, quantization may be introduced in varying amounts in order to determine the optimal number of bits required in a realization. In filtering applications where high-throughput is the principal design criterion, residue number system based WDFs may be a solution. WDFSim can simulate the response of a WDF based on the RNS defined over the set $P = \{2^n - 1, 2^n, 2^n + 1\}$ of relatively prime moduli. This particular moduli set has been shown to lead to efficient scaling and decoding hardware realizations.

Filters to be simulated may be composed of several types of WDF building blocks. Building blocks that have thus far been implemented are the following: WD equivalents of a capacitance, an inductance, a resistance, a resistive source, an ideal transformer, and three- and four-port circulators; parallel and series adaptors - both of which may or may not have a reflection-free port; and elementary two-port sections including four first-order reciprocal sections, four second-order reciprocal sections, one first-order non-reciprocal section, one second-order non-reciprocal section, and a third-order quasi-lattice section.

WDFSim has been tested using several wave digital filters from published documents. No discrepancies could be found between the published results and the output of WDFSim. Based on the work done thus far, the following recommendations are made.

1. Use of the **Edit** menu (and key equivalents) to cut, copy, and paste nodes in the File Window. The edit command **Clear** could be used to erase the entire File Window contents.

2. Use of the keyboard arrow keys $(\leftarrow, \uparrow, \rightarrow, \text{and } \downarrow)$ to move the Active Node around.

3. Eliminate the need to keep the filter parameters in $\mathbb{S}_M$ during the RNS simulation. This could possibly be accomplished through the use of generic pointers (i.e. LightSpeed Pascal type **Ptr**).

4. Allow for different types of quantization schemes. For example, different numbers of bits for multipliers than for signals and/or mantissa, rather than fixed point binary representations.

5. Better error handling procedures are required. In particular, if a multiplier dialog box's **OK** button is clicked before all fields have been filled in, a system crash occurs.

6. More WDF building block types could be added to WDFSim. Some possibilities include WD equivalents of: a short circuit, an open circuit, a voltage source, a unit element, a QUARL, and a gyrator. Also, the 'symmetric' two-port adaptor in [2] (pp. 279-280) may deserve some consideration.

These are just a few of the possible enhancements that could be made to WDFSim. Undoubtably, many more could be added to the list. Of the suggested recommendations, the first two should be trivial. Items 3-5 however, are more difficult and require some careful thought. Addition of new types of building blocks is straightforward, although a fair amount of work is required (mostly user-interface oriented). In particular, the following must be done.

1. Create **pictures** defining all the possible orientations of the new building block and add them to the resource file. Use the programs in Appendices C or E as required.

2. Replace the old **Connections** data structure in the resource file (type PORT) with a new one that includes the new building block type. Use the program and files in Appendix D as required.

3. In unit **Declarations**, add the number of new pictures created to the *Npicts* constant and add the new building block type to the *nodeType* enumerated type list of the **Node** record.

4. Add the new type to either the **Element** or the **Adaptor** menus in the resource file. Correspondingly, add the new type to either procedure ElementMenu or procedure AdaptorMenu.

5. Create an 'orientation' dialog in the resource file and write the 'item' procedure required in step 4.

6. If the new building block contains multipliers, create a 'multiplier' dialog box in the resource file and add the new type to procedure ShowInfo. Also, write (or use existing) quantization routine(s) and add the new building block type to procedure QuantizeNet.

7. Write an 'update' procedure defining the time domain input-output relationship of the new building block and add its type to procedure UpdateNet.

REFERENCES

[1] A. Fettweis, 'Digital filter structures related to classical filter networks', *Arch. Elek. Ubertragung*, vol. AEU-25, 1971, pp. 79-89.

[2] A. Fettweis, 'Wave digital filters: Theory and practice', *Proc. IEEE*, vol. 74, no. 2, Feb. 1986, pp. 270-327.

[3] V. Belevitch, *Classical Network Theory*, San Francisco, CA: Holden-Day, 1968.

[4] M. Jarmasz, 'A simplified synthesis of lossless two-port wave digital and analog filters', *Ph.D. Thesis*, University of Manitoba, Winnipeg, Canada, 1990.

[5] R. Pepe and J. Rogers, 'Simulation of fixed-point operations with high-level languages', *IEEE Trans. on Acoust., Speech, and Signal Proc.*, vol. ASSP-35, no. 1, Jan. 1987, pp. 116-118.

[6] F. Taylor, 'Residue arithmetic: A tutorial with examples', *IEEE Computer Mag.*, May 1984.

[7] W. Jenkins and B. Leon, 'The use of residue number systems in the design of finite impulse response digital filters', *IEEE Trans. on Circ. and Syst.*, vol. CAS-24, no. 4, April 1977, pp. 191-201.

[8] W. Jenkins, 'Recent advances in residue number techniques for recursive digital filtering', *IEEE Trans. on Acoust., Speech, and Signal Proc.*, vol. ASSP-27, no. 1, Feb. 1979, pp. 19-30.

[9] A. Baraniecka and G. Jullian, 'Residue number system implementations of number theoretic transforms in complex residue rings', *IEEE Trans. on Acoust., Speech, and Signal Proc.*, vol. ASSP-28, no. 3, June 1980, pp. 285-291.

[10] G. Cardarilli, R. Lojacono, G. Martinelli, and M. Salerno, 'Structurally passive digital filters in residue number systems', *IEEE Trans. on Circ. and Syst.*, vol. CAS-35, no. 2, Feb. 1988, pp. 149-158.

[11] R. Blahut, *Fast Algorithms for Digital Signal Processing*, Reading, Massachusetts: Addison-Wesley, 1985.

[12] R. Gregory and E. Krishnamurthy, *Methods and Applications of Error-Free Computation*, New York, New York: Springer-Verlag, 1984.

[13] R. Thun, 'On residue number system decoding', *IEEE Trans. on Acoust., Speech, and Signal Proc.*, vol. ASSP-34, no. 5, Oct. 1986, pp. 1346-1347.

[14] N. Chakraborti, J. Soundararajan, and A. Reddy, 'An implementation of mixed-radix conversion for residue number applications', *IEEE Trans. on Comp.*, vol. C-35, no. 8, Aug. 1986, pp. 762-764.

[15] T. Vu, 'Efficient implementations fo the Chinese Remainder Theorem for sign detection and residue decoding', *IEEE Trans. on Comp.*, vol. C-34, no. 7, July 1985, pp. 646-651.

[16] F. Taylor and C. Huang, 'An autoscale residue multipier', *IEEE Trans. on Comp.*, vol. C-31, no. 4, April 1982, pp. 321-325.

[17] M. Etzel and W. Jenkins, 'The design of specialized residue classes for efficient recursive digital filter realization', *IEEE Trans. on Acoust., Speech, and Signal Proc.*, vol. ASSP-30, no. 3, June 1982, pp. 370-380.

[18] P. Bernardson, 'Fast memoryless, over 64 bits, residue-to-binary convertor', *IEEE Trans. on Circ. and Syst.*, vol. CAS-32, no. 3, March 1985, pp. 298-300.

[19] K. Ibrahim and S. Saloum, 'An efficient residue to binary convertor design', *IEEE Trans. on Circ. and Syst.*, vol. CAS-35, no. 9, Sept. 1988, pp. 1156-1158.

[20] S. Andraos and H. Ahmad, 'A new efficient memoryless residue to binary convertor', *IEEE Trans. on Circ. and Syst.*, vol. CAS-35, no. 11, Nov. 1988, pp. 1441-1444.

[21] Apple Computer Inc., *Inside Macintosh Volume I*, Reading, Massachusetts: Addison-Wesley, 1988.

[22] Apple Computer Inc., *Inside Macintosh Volume II*, Reading, Massachusetts: Addison-Wesley, 1988.

[23] Apple Computer Inc., *Inside Macintosh Volume III*, Reading, Massachusetts: Addison-Wesley, 1988.

[24] Apple Computer Inc., *Inside Macintosh Volume IV*, Reading, Massachusetts: Addison-Wesley, 1988.

[25] Apple Computer Inc., *Inside Macintosh Volume V*, Reading, Massachusetts: Addison-Wesley, 1988.

[26] K. Meerkötter, 'Beiträge zur theorie der wellandigitalfilter', *Ph.D. Thesis*, Ruhr University, Bochum, Federal Republic of Germany, 1979.

[27] M. Jarmasz and G. Martens, 'Design of canonic wave digital filters using Brune and matched 4-port adaptors', *IEEE Trans. on Circ. and Syst.*, vol. CAS-34, no. 5, May 1987, pp. 480-496.

[28] M. Jarmasz - private communication.

[29] P. Vaidyanathan and S. Mitra, 'Low passband sensitivity digital filters: A generalized viewpoint and synthesis procedures', *Proc. IEEE*, vol. 72, no. 4, April 1984, pp. 404-423.

APPENDIX A

THE WDFSIM PROGRAM LISTING

(pages 102-184)

```
unit Declarations;

interface

  const
    Pi = 3.14159265358979323846264338322;
    Nnodes = 60;
    Npicts = 131;
    Nports = 4;
    Ngammas = 16;
    MaxPts = 1024;
    LPtoBP = false;

  type
    Gammas = array[1..Ngammas] of extended;
    Node = record
        n, pict: integer;
        nodeType: (parallel, serial, parallelRF, serialRF, capacitor, inductor, IOport, circulator, A0,
      A1, B1, C1, D1, E1, A2, B2, C2, D2, E2, QL3, none, transformer);
        bnode, bport: array[1..Nports] of integer;
        astat, bstat: array[0..Nports] of boolean;
        aval, bval: array[-6..Nports] of extended;
        gamma: Gammas;
      end;
    Network = array[1..Nnodes] of Node;
    NetPtr = ^Network;
    FileRec = record
        name: Str255;
        vRefNum, fRefNum: integer;
      end;
    FileRecArray = array[1..Nnodes] of FileRec;
    Connection = record
        N, S, E, W: integer;
      end;
    Connections = array[0..Npicts] of Connection;
    Simulation = record
        simType: (User, Impulse, Step, Frequency, Narrowband);
        addType: (AFloating, Overflow, Saturation, AResidue);
        multType: (MFloating, Rounding, Truncation, MagTruncation, MResidue);
        nSamples: integer;
        wmin, wmax: extended;
        RNb, RNbc: longint;
        binary: boolean;
        p1, p2, p3: integer;
        P, scale: longint;
      end;
    Complex = record
        re, im: extended;
      end;
    FFTArrayPtr = ^FFTArray;
    FFTArray = array[0..MaxPts] of Complex;
    PlotArray = array[0..MaxPts] of real;
    PlotPtr = ^PlotRec;
```

```
    PlotRec = record
        ymin, ymax, xmin, xmax: real;
        ydiv, xdiv: integer;
        window: WindowPtr;
        xpts, ypts: PlotArray;
        discrete, grid: boolean;
        next: PlotPtr;
      end;

  var
    Wind: WindowPtr;
    Done, Built, Saved: boolean;
    Net: NetPtr;
    Sim: Simulation;
    FilterFile: FileRec;
    DragRect, ActiveRect: Rect;
    ActiveNode, k: integer;
    Menus: array[0..6] of MenuHandle;
    Picts: array[0..Npicts] of PicHandle;
    PortCons: Connections;
    Plots: array[1..Nnodes] of PlotPtr;

implementation

end.
```

```
unit Complex_Math;

interface

   uses
      Declarations;

   function ArkTan (x, y: extended): extended;
   function Cabs (x: Complex): extended;
   function Carg (x: Complex): extended;
   function Carg0 (x: Complex): extended;
   function Cmplx (a, b: extended): Complex;
   function Cadd (a, b: Complex): Complex;
   function Csub (a, b: Complex): Complex;
   function Cmult (a, b: Complex): Complex;
   function Cdiv (a, b: Complex): Complex;
   function Conjg (x: Complex): Complex;
   function Atten (x: Complex): extended;

implementation

   function ArkTan;
      var
         a: extended;
   begin
      if x = 0 then begin
         if y = 0 then
            a := 0;
         if y < 0 then
            a := -Pi / 2;
         if y > 0 then
            a := Pi / 2;
      end
      else begin
         a := arctan(y / x);
         if x < 0 then
            a := a + Pi;
{     if a < 0 then}
{     a := a + 2 * Pi;}
      end;
      ArkTan := a;
   end;

   function Cabs;
   begin
      Cabs := sqrt(sqr(x.re) + sqr(x.im));
   end;

   function Carg0;
   begin
      if abs(x.re) < 0.0001 then
         x.re := 0;
      if abs(x.im) < 0.0001 then
```

```
      x.im := 0;
    Carg0 := ArkTan(x.re, x.im);
  end;

  function Carg;
  begin
    Carg := ArkTan(x.re, x.im);
  end;

  function Cmplx;
  begin
    Cmplx.re := a;
    Cmplx.im := b;
  end;

  function Cadd;
  begin
    Cadd.re := a.re + b.re;
    Cadd.im := a.im + b.im;
  end;

  function Csub;
  begin
    Csub.re := a.re - b.re;
    Csub.im := a.im - b.im;
  end;

  function Cmult;
  begin
    Cmult.re := a.re * b.re - a.im * b.im;
    Cmult.im := a.re * b.im + a.im * b.re;
  end;

  function Cdiv;
    var
      den: extended;
  begin
    den := sqr(b.re) + sqr(b.im);
    Cdiv.re := (a.re * b.re + a.im * b.im) / den;
    Cdiv.im := (a.im * b.re - a.re * b.im) / den;
  end;

  function Conjg;
  begin
    Conjg.re := x.re;
    Conjg.im := -x.im;
  end;

  function Atten;
    var
      c: extended;
  begin
    c := Cabs(x);
```

```
    if c < 1.0e-20 then
      Atten := +INF
    else
      Atten := -20 * ln(c) / ln(10);
  end;

end.
```

```
unit BinaryMath;

interface

   function BSum (t1, t2: extended; RN: longint; sat: boolean): extended;
   function BProd (t1, t2: extended; RN, RNter1: longint; rnd: boolean): extended;
   function MagTrunc (t1, t2: extended; RN, RNter1: longint): extended;
   function BNeg (x: extended; RN, RNter1: longint): extended;
   function BtoR (x: extended; RN, RNter1: longint): extended;
   function RtoB (x: extended; RN, RNter1: longint): extended;

implementation

   procedure Saturate (var x: extended; RN: longint; ovcnt: integer);
   begin
      if ovcnt < 0 then
         x := RN div 2;
      if ovcnt > 0 then
         x := (RN div 2) - 1;
   end;

   function BSum;
      var
         sum1, rn2: extended;
         sgnsum, sgnt1, sgnt2, ovcnt: integer;
   begin
      ovcnt := 0;
      sum1 := t1 + t2;
      sum1 := sum1 - RN * trunc(sum1 / RN);
      rn2 := RN div 2;
      sgnsum := 1 - 2 * trunc(sum1 / rn2);
      sgnt1 := 1 - 2 * trunc(t1 / rn2);
      sgnt2 := 1 - 2 * trunc(t2 / rn2);
      if (sgnt1 = sgnt2) and (sgnt1 <> sgnsum) then
         ovcnt := ovcnt + sgnt1;
      if sat then
         Saturate(sum1, RN, ovcnt);
      BSum := sum1;
   end;

   function BProd;
      var
         a: extended;
         prod1, aux: extended;
         im: integer;
   begin
      prod1 := t1 * t2;
      if t1 >= (RN div 2) then
         prod1 := prod1 + RN * (RN - t2);
      if t2 >= (RN div 2) then
         prod1 := prod1 + RN * (RN - t1);
      prod1 := prod1 / RN;
      prod1 := prod1 / RNter1;
```

```
    prod1 := RN * (prod1 - trunc(prod1));
    im := 1;
    a := trunc(prod1);
    if (a = (RN div 2) - 1) or (not rnd) then
      im := 0;
    BProd := trunc(prod1 + im * 0.5);
  end;

  function MagTrunc;
    var
      sign: integer;
      magtrunc1, aux: extended;
      magtrunc2: extended;
  begin
    if (t1 >= RN div 2) and (t2 >= RN div 2) then begin
      sign := 1;
      magtrunc1 := (RN - t1) * (RN - t2);
    end;
    if (t1 < RN div 2) and (t2 < RN div 2) then begin
      sign := 1;
      magtrunc1 := t1 * t2;
    end;
    if (t1 >= RN div 2) and (t2 < RN div 2) then begin
      sign := -1;
      magtrunc1 := (RN - t1) * t2;
    end;
    if (t1 < RN div 2) and (t2 >= RN div 2) then begin
      sign := -1;
      magtrunc1 := t1 * (RN - t2);
    end;
    magtrunc1 := magtrunc1 / RN;
    magtrunc1 := magtrunc1 / RNter1;
    magtrunc1 := RN * (magtrunc1 - trunc(magtrunc1));
    magtrunc2 := trunc(magtrunc1);
    if sign = -1 then
      magtrunc2 := RN - magtrunc2;
    MagTrunc := magtrunc2;
  end;

  function BNeg;
  begin
    BNeg := RN - x;
  end;

  function RtoB;
  begin
    if x >= RN div 2 then
      RtoB := -(RN - x) / RNter1
    else
      RtoB := x / RNter1;
  end;

  function BtoR;
```

```
  begin
    if x >= 0 then
      BtoR := round(x * RNter1)
    else
      BtoR := RN + round(x * RNter1);
  end;

end.
```

```
unit ResidueMath;

interface

  uses
    Declarations;

  type
    MMRN = array[1..3] of longint;
    SMRN = longint;

  function BtoMM (x: extended): MMRN;
  function MMtoB (x: MMRN): extended;
  function SMtoMM (x: SMRN): MMRN;
  function MMtoSM (x: MMRN): SMRN;
  function BtoSM (x: extended): SMRN;
  function SMtoB (x: SMRN): extended;
  function RNeg (x: MMRN): MMRN;
  function RSum (x, y: MMRN): MMRN;
  function AutoScale (x: MMRN; c: SMRN; v: longint): MMRN;

implementation

  function BtoSM;
  begin
    if x >= 0 then
      BtoSM := round(x * Sim.scale)
    else
      BtoSM := Sim.P + round(x * Sim.scale);
  end;

  function SMtoB;
  begin
    if x < (Sim.P div 2) then
      SMtoB := x / Sim.scale
    else
      SMtoB := (x - Sim.P) / Sim.scale;
  end;

  function SMtoMM;
  begin
    SMtoMM[1] := x mod Sim.p1;
    SMtoMM[2] := x mod Sim.p2;
    SMtoMM[3] := x mod Sim.p3;
  end;

  function MMtoSM;
    var
      r, t: longint;
  begin
    r := ((Sim.p2 div 2) * (Sim.p2 * (x[1] + x[3]) + x[1] - x[3])) mod (Sim.p2 * Sim.p2 - 1);
    t := ((r mod Sim.p2) - x[2]) mod Sim.p2;
    if t < 0 then
```

```
      t := Sim.p2 + t;
    MMtoSM := r + Sim.p2 * Sim.p2 * t - t;
  end;

  function BtoMM;
  begin
    BtoMM := SmtoMM(BtoSM(x));
  end;

  function MMtoB;
  begin
    MMtoB := SMtoB(MMtoSM(x));
  end;

  function RNeg;
  begin
    RNeg := BtoMM(-MMtoB(x));
  end;

  function RSum;
  begin
    RSum[1] := (x[1] + y[1]) mod Sim.p1;
    RSum[2] := (x[2] + y[2]) mod Sim.p2;
    RSum[3] := (x[3] + y[3]) mod Sim.p3;
  end;

  function AutoScale;
    var
      x1: MMRN;
      y1: SMRN;
      zz: extended;
      sign: integer;
  begin
    sign := 1;
    zz := MMtoB(x);
    if zz < 0 then begin
      sign := -sign;
      x := BtoMM(-zz);
    end;
    zz := SMtoB(c);
    if zz < 0 then begin
      sign := -sign;
      c := BtoSM(-zz);
    end;
    x1[1] := 0;
    x1[2] := (x[2] - x[1]) mod Sim.p2;
    if x1[2] < 0 then
      x1[2] := Sim.p2 + x1[2];
    x1[3] := (x[3] - x[1]) mod Sim.p3;
    if x1[3] < 0 then
      x1[3] := Sim.p3 + x1[3];
    y1 := MMtoSM(x1);
    y1 := round(c * y1 / v);
```

```
      x1 := SMtoMM(y1);
      if sign < 0 then
         x1 := Rneg(x1);
      AutoScale := x1;
   end;

end.
```

```
unit FFTs;

interface

  uses
    Declarations, Complex_Math;

  const
    r = 2;                  { transform radix }

  var
    N: integer;

  procedure IDFT (x: FFTArrayPtr; var nn: integer);
  procedure FFT (x: FFTArrayPtr; var nn: integer);
  function DFT (x: FFTArrayPtr; w: extended; npts: integer): Complex;

implementation

  procedure termA2 (x: FFTArrayPtr);
    var
      k, N2: integer;
      u, v: complex;
  begin
    N2 := N div 2;
    for k := 0 to (N2 - 1) do begin
      u.re := x^[k].re + x^[k + N2].re;
      u.im := x^[k].im + x^[k + N2].im;
      v.re := x^[k].re - x^[k + N2].re;
      v.im := x^[k].im - x^[k + N2].im;
      x^[k] := u;
      x^[k + N2] := v;
    end;
  end;

  procedure termA4 (x: FFTArrayPtr);
    var
      k, N4: integer;
      s, t, u, v: complex;
  begin
    N4 := N div 4;
    for k := 0 to (N4 - 1) do begin
      s.re := x^[k].re + x^[k + N4].re + x^[2 * N4 + k].re + x^[3 * N4 + k].re;
      s.im := x^[k].im + x^[k + N4].im + x^[2 * N4 + k].im + x^[3 * N4 + k].im;
      t.re := x^[k].re + x^[k + N4].im - x^[2 * N4 + k].re - x^[3 * N4 + k].im;
      t.im := x^[k].im - x^[k + N4].re - x^[2 * N4 + k].im + x^[3 * N4 + k].re;
      u.re := x^[k].re - x^[k + N4].re + x^[2 * N4 + k].re - x^[3 * N4 + k].re;
      u.im := x^[k].im - x^[k + N4].im + x^[2 * N4 + k].im - x^[3 * N4 + k].im;
      v.re := x^[k].re - x^[k + N4].im - x^[2 * N4 + k].re + x^[3 * N4 + k].im;
      v.im := x^[k].im + x^[k + N4].re - x^[2 * N4 + k].im - x^[3 * N4 + k].re;
      x^[k] := s;
      x^[k + N4] := t;
      x^[k + 2 * N4] := u;
```

```
      x^[k + 3 * N4] := v;
    end;
  end;

  procedure termA (x: FFTArrayPtr);
  begin
    case r of
      2:
        termA2(x);
      4:
        termA4(x);
    end;
  end;

  procedure termB (x: FFTArrayPtr; i: integer);
    var
      Nri, ri, rir, j, k, l, p, q: integer;
      a, b: extended;
      w, u: complex;
  begin
    ri := round(exp(i * ln(r)));
    Nri := N div ri;
    rir := ri div r;
    b := -2 * pi / ri;
    for j := 0 to (ri - 1) do begin
      p := j div rir;
      q := j mod rir;
      a := b * p * q;
      w.re := cos(a);
      w.im := sin(a);
      for l := 0 to (Nri - 1) do begin
        k := l + j * Nri;
        u.re := x^[k].re * w.re - x^[k].im * w.im;
        u.im := x^[k].re * w.im + x^[k].im * w.re;
        x^[k] := u;
      end;
    end;
  end;

  procedure termC (x: FFTArrayPtr; i: integer);
    var
      Nri, ri, j, k, l, p, q, Pj: integer;
      t: FFTArrayPtr;
  begin
    New(t);
    ri := round(exp(i * ln(r)));
    Nri := N div ri;
    for j := 0 to (ri - 1) do begin
      p := j div r;
      q := j mod r;
      Pj := (ri div r) * q + p;
      for l := 0 to (Nri - 1) do begin
        k := l + j * Nri;
```

```
          t^[k] := x^[Pj * Nri + l];
        end;
      end;
      for j := 0 to (N - 1) do
        x^[j] := t^[j];
      Dispose(t);
  end;

  procedure FFT;
    var
      n1, i: integer;
  begin
    N := 2;
    while N < nn do
      N := N * 2;
    n1 := round(ln(N) / ln(r));
    for i := nn to (N - 1) do begin
      x^[i].re := 0;
      x^[i].im := 0;
    end;
    for i := n1 downto 1 do begin
      termA(x);
      termB(x, i);
      termC(x, i);
    end;
    nn := N;
  end;

  procedure IDFT;
    var
      k: integer;
  begin
    for k := 0 to nn do
      x^[k].im := -x^[k].im;
    FFT(x, nn);
    for k := 0 to nn do begin
      x^[k].re := x^[k].re / nn;
      x^[k].im := -x^[k].im / nn;
    end;
  end;

  function DFT;
    var
      t, c: Complex;
      k: integer;
  begin
    t := Cmplx(0, 0);
    for k := 0 to (npts - 1) do begin
      c := Cmplx(cos(w * k), -sin(w * k));
      t := Cadd(t, Cmult(x^[k], c));
    end;
    DFT := t;
  end;
```

```
end.
```

```
unit Plotting;

interface

  uses
    Declarations;

  procedure InitPlots;
  procedure ErasePlots;
  procedure AddPlot (var plot: PlotPtr; k: integer);
  procedure RemovePlot (var plot: PlotPtr; k: integer);
  function FindPlot (var w: WindowPtr; k: integer): PlotPtr;
  function StartPt (var x: PlotArray; xmin: real): integer;
  function EndPt (var x: PlotArray; xmax: real): integer;
  function SetPlotParam (s: string): real;
  procedure PlotCurve (var plot: PlotRec);

implementation

  procedure InitPlots;
    var
      k: integer;
  begin
    for k := 1 to Nnodes do
      Plots[k] := nil;
  end;

  procedure AddPlot;
    var
      curr: PlotPtr;
  begin
    if Plots[k] = nil then
      Plots[k] := plot
    else begin
      curr := Plots[k];
      while curr^.next <> nil do
        curr := curr^.next;
      curr^.next := plot;
    end;
  end;

  procedure RemovePlot;
    var
      curr, prev: PlotPtr;
  begin
    prev := nil;
    curr := Plots[k];
    while curr <> plot do begin
      prev := curr;
      curr := curr^.next;
    end;
    if prev = nil then
      Plots[k] := curr^.next
```

```
    else
      prev^.next := curr^.next;
    DisposeWindow(curr^.window);
    Dispose(curr);
  end;

  function FindPlot;
    var
      curr: PlotPtr;
  begin
    curr := Plots[k];
    while curr^.window <> w do
      curr := curr^.next;
    FindPlot := curr;
  end;

  procedure ErasePlots;
    var
      k: integer;
  begin
    for k := 1 to Nnodes do begin
      while Plots[k] <> nil do
        RemovePlot(Plots[k], k);
    end;
  end;

  function SetPlotParam;
    var
      item, kind: integer;
      str: Str255;
      dlog: DialogPtr;
      box: Rect;
      hand: Handle;
      t: real;
  begin
    ParamText(s, '', '', '');
    dlog := GetNewDialog(208, nil, pointer(-1));
    ModalDialog(nil, item);
    GetDItem(dlog, 3, kind, hand, box);
    GetIText(hand, str);
    ReadString(str, t);
    DisposDialog(dlog);
    SetPlotParam := t;
  end;

  function StartPt;
    var
      k: integer;
  begin
    StartPt := MaxPts - 1;
    k := 0;
    while k < MaxPts do begin
      if x[k] >= xmin then begin
```

```
        StartPt := k;
        k := MaxPts;
      end
      else
        k := k + 1;
    end;
  end;

  function EndPt;
    var
      k: integer;
  begin
    EndPt := 0;
    k := 0;
    while k < MaxPts do begin
      if x[k] >= xmax then begin
        EndPt := k;
        k := MaxPts
      end
      else
        k := k + 1;
    end;
  end;

  function Scale (var ymin, ymax: real): real;
    var
      plotMax, plotScale, ymax0, ymin0: real;
      t, k: integer;
  begin
    if abs(ymin) > abs(ymax) then
      plotMax := abs(ymin)
    else
      plotMax := abs(ymax);
    plotScale := 1;
    if plotMax < 1 then begin
      t := Trunc(ln(plotMax) / ln(10)) - 1;
      for k := 1 to (-t) do
        plotScale := plotScale / 10;
    end
    else begin
      t := Trunc(ln(plotMax) / ln(10));
      for k := 1 to t do
        plotScale := plotScale * 10;
    end;
    ymax0 := Round(ymax / plotScale + 0.49);
    ymax := ymax0 * plotScale;
    ymin0 := Round(ymin / plotScale - 0.49);
    ymin := ymin0 * plotScale;
    Scale := plotScale;
  end;

  procedure XLabel (rec: Rect; xmin, xmax: real; grid: boolean; xdiv: integer);
    var
```

```
        k, hpos: integer;
        xDelta: real;
        s: Str255;
  begin
    if xdiv <= 0 then
      xdiv := 5;
    xDelta := (xmax - xmin) / xdiv;
    for k := 0 to xdiv do begin
      hpos := Round(rec.left + k * (rec.right - rec.left) / xdiv);
      MoveTo(hpos, rec.bottom + 3);
      LineTo(hpos, rec.bottom);
      if grid then begin
        PenPat(gray);
        LineTo(hpos, rec.top);
        PenPat(black);
      end;
      s := StringOf((k * xDelta + xmin) : 4 : 2);
      MoveTo(hpos - 10, rec.bottom + 18);
      DrawString(s);
    end;
  end;

  procedure YLabel (rec: Rect; ymin, ymax, yscale: real; grid: boolean);
    var
      k, vpos, ndiv: integer;
      s: Str255;
  begin
    ndiv := Round((ymax - ymin) / yscale);
    for k := 0 to ndiv do begin
      vpos := Round(rec.bottom - k * (rec.bottom - rec.top) / ndiv);
      MoveTo(rec.left - 3, vpos);
      LineTo(rec.left, vpos);
      if grid then begin
        PenPat(gray);
        LineTo(rec.right, vpos);
        PenPat(black);
      end;
      s := StringOf((ymin + k * yscale) : 5 : 2);
      MoveTo(rec.left - 35, vpos + 5);
      DrawString(s);
    end;
  end;

  procedure PlotCurve;
    var
      hpos, vpos, xaxis, i, n1, n2: integer;
      hscale, vscale, ymin, ymax, yscale: real;
      rec, rec2: Rect;
  begin
    TextFont(times);
    TextSize(12);
    rec.top := thePort^.portRect.top + 15;
    rec.bottom := thePort^.portRect.bottom - 65;
```

```
rec.left := thePort^.portRect.left + 45;
rec.right := thePort^.portRect.right - 35;
ymin := +INF;
ymax := -INF;
n1 := StartPt(plot.xpts, plot.xmin);
n2 := EndPt(plot.xpts, plot.xmax);
hscale := (rec.right - rec.left) / (n2 - n1);
vscale := rec.bottom - rec.top;
for i := n1 to n2 do begin
  if plot.ypts[i] > ymax then
    ymax := plot.ypts[i];
  if plot.ypts[i] < ymin then
    ymin := plot.ypts[i];
end;
if ymax > plot.ymax then
  ymax := plot.ymax;
if ymin < plot.ymin then
  ymin := plot.ymin;
yscale := Scale(ymin, ymax);
if plot.ydiv > 0 then
  yscale := (ymax - ymin) / plot.ydiv;
xaxis := rec.top + round(vscale * ymax / (ymax - ymin));
MoveTo(rec.left, xaxis);
LineTo(rec.right, xaxis);
if not plot.discrete then begin
  vpos := xaxis - round(vscale * plot.ypts[n1] / (ymax - ymin));
  MoveTo(rec.left, vpos);
  for i := (n1 + 1) to n2 do begin
    hpos := rec.left + round((i - n1) * hscale);
    if plot.ypts[i] = +INF then
      vpos := 32767
    else if plot.ypts[i] = -INF then
      vpos := -32767
    else
      vpos := xaxis - round(vscale * plot.ypts[i] / (ymax - ymin));
    LineTo(hpos, vpos);
  end;
  SetRect(rec2, 0, 0, thePort^.portRect.right - 15, rec.top);
  FillRect(rec2, white);
  SetRect(rec2, 0, rec.bottom, thePort^.portRect.right - 15, thePort^.portRect.bottom - 15);
  FillRect(rec2, white);
end
else begin
  for i := n1 to n2 do begin
    hpos := rec.left + round((i - n1) * hscale);
    if plot.ypts[i] <= plot.ymax then begin
      vpos := xaxis - round(vscale * plot.ypts[i] / (ymax - ymin));
      PenPat(black);
    end
    else begin
      vpos := rec.top;
      PenPat(white);
    end;
```

```
          MoveTo(hpos, xaxis);
          LineTo(hpos, vpos);
          PaintCircle(hpos, vpos, 2);
        end;
      end;
      XLabel(rec, plot.xpts[n1], plot.xpts[n2], plot.grid, plot.xdiv);
      YLabel(rec, ymin, ymax, yscale, plot.grid);
      rec.right := rec.right + 1;
      rec.bottom := rec.bottom + 1;
      FrameRect(rec);
    end;

end.
```

```
unit Utilities;

interface

  uses
    Declarations;

  procedure PrintFilter;
  function NodeToRect (node: integer): Rect;
  function PtToNode (pt: Point): integer;
  procedure GetThetaInfo (var dNo, nAngles, nMults: integer);
  function InvCos (x, y: extended): extended;

implementation

  procedure PrintFilter;
    var
      i, k: integer;
  begin
    showtext;
    writeln;
    for i := 1 to Nnodes do begin
      if Net^[i].nodeType <> none then begin
        writeln('Node :', i);
        writeln('n :', Net^[i].n);
        writeln('pict :', Net^[i].pict);
        writeln('type :', Net^[i].nodeType);
        writeln('gammas :');
        for k := 1 to Ngammas do
          writeln('    ', k : 1, ' :', Net^[i].gamma[k] : 25 : 20);
        writeln('port connections:');
        for k := 1 to Nports do
          writeln('    ', k : 1, ' :', Net^[i].bnode[k], Net^[i].bport[k]);
        writeln;
      end;
    end;
  end;

  function NodeToRect;
    var
      i, j: integer;
  begin
    i := (node - 1) div 10;
    j := (node - 1) mod 10;
    NodeToRect.left := j * 50;
    NodeToRect.right := (j + 1) * 50;
    NodeToRect.top := i * 50;
    NodeToRect.bottom := (i + 1) * 50;
  end;

  function PtToNode;
    var
      i, j: integer;
```

```
begin
  i := pt.v div 50;
  j := pt.h div 50;
  if pt.h > 499 then
    j := 9;
  if pt.v > 299 then
    i := 5;
  PtToNode := (i * 10) + j + 1;
end;

procedure GetThetaInfo;
begin
  nMults := 0;
  case Net^[ActiveNode].nodetype of
    A0:
      begin
      dNo := 499;
      nAngles := 1;
    end;
    A1:
      begin
      dNo := 500;
      nAngles := 1;
    end;
    B1:
      begin
      dNo := 501;
      nAngles := 1;
    end;
    C1:
      begin
      dNo := 502;
      nAngles := 1;
    end;
    D1:
      begin
      dNo := 503;
      nAngles := 1;
    end;
    E1:
      begin
      dNo := 504;
      nAngles := 2;
      nMults := 2;
    end;
    A2:
      begin
      dNo := 505;
      nAngles := 2;
    end;
    B2:
      begin
      dNo := 506;
```

```
          nAngles := 2;
        end;
        C2:
          begin
          dNo := 507;
          nAngles := 3;
          nMults := 2;
        end;
        D2:
          begin
          dNo := 508;
          nAngles := 3;
          nMults := 2;
        end;
        E2:
          begin
          dNo := 509;
          nAngles := 4;
          nMults := 2;
        end;
        QL3:
          begin
          dNo := 510;
          nAngles := 4;
          nMults := 0;
        end;
    end;
  end;

  function InvCos;
  begin
    if y >= 0 then
      InvCos := ArcCos(x);
    if y < 0 then
      InvCos := -ArcCos(x);
  end;

end.
```

```
unit Builders;

interface

  uses
    Declarations, Utilities;

  procedure BuildFile;

implementation

  function ConnectWest (k: integer): boolean;
    var
      p1, p2: integer;
  begin
    ConnectWest := true;
    p1 := PortCons[Net^[k].pict].W;
    if p1 > 0 then begin
      ConnectWest := false;
      if (k mod 10) <> 1 then begin
        p2 := PortCons[Net^[k - 1].pict].E;
        if p2 > 0 then begin
          ConnectWest := true;
          Net^[k].bport[p1] := p2;
          Net^[k].bnode[p1] := k - 1;
        end;
      end;
    end;
  end;

  function ConnectEast (k: integer): boolean;
    var
      p1, p2: integer;
  begin
    ConnectEast := true;
    p1 := PortCons[Net^[k].pict].E;
    if p1 > 0 then begin
      ConnectEast := false;
      if (k mod 10) <> 0 then begin
        p2 := PortCons[Net^[k + 1].pict].W;
        if p2 > 0 then begin
          ConnectEast := true;
          Net^[k].bport[p1] := p2;
          Net^[k].bnode[p1] := k + 1;
        end;
      end;
    end;
  end;

  function ConnectNorth (k: integer): boolean;
    var
      p1, p2: integer;
  begin
```

```
      ConnectNorth := true;
      p1 := PortCons[Net^[k].pict].N;
      if p1 > 0 then begin
        ConnectNorth := false;
        if k > 10 then begin
          p2 := PortCons[Net^[k - 10].pict].S;
          if p2 > 0 then begin
            ConnectNorth := true;
            Net^[k].bport[p1] := p2;
            Net^[k].bnode[p1] := k - 10;
          end;
        end;
      end;
    end;

    function ConnectSouth (k: integer): boolean;
      var
        p1, p2: integer;
    begin
      ConnectSouth := true;
      p1 := PortCons[Net^[k].pict].S;
      if p1 > 0 then begin
        ConnectSouth := false;
        if k < 51 then begin
          p2 := PortCons[Net^[k + 10].pict].N;
          if p2 > 0 then begin
            ConnectSouth := true;
            Net^[k].bport[p1] := p2;
            Net^[k].bnode[p1] := k + 10;
          end;
        end;
      end;
    end;

    function AddNode (k: integer): boolean;
      var
        f: boolean;
    begin
      f := true;
      f := f and ConnectNorth(k);
      f := f and ConnectSouth(k);
      f := f and ConnectEast(k);
      f := f and ConnectWest(k);
      if not f then begin
        InvertRect(ActiveRect);
        ActiveNode := k;
        ActiveRect := NodeToRect(k);
        InvertRect(ActiveRect);
        SysBeep(20);
        SysBeep(20);
      end;
      AddNode := f;
    end;
```

```
  procedure BuildFile;
    var
      k: integer;
      f: boolean;
  begin
    f := true;
    k := 1;
    while (k <= Nnodes) and f do begin
      f := f and AddNode(k);
      k := k + 1;
    end;
    if f then begin
      Built := true;
      EnableItem(Menus[5], 0);
      DrawMenuBar;
    end;
  end;

end.
```

```
unit Quantizers;

interface

   uses
      Declarations, BinaryMath, Utilities;

   procedure SetQuantization;
   procedure QuantizeNet;

implementation

   procedure SetQuantization;
      var
         dlog: DialogPtr;
         item: Handle;
         box: Rect;
         text: Str255;
         bbits, cbits, itemHit, itemType, k: integer;
   begin
      dlog := GetNewDialog(210, nil, pointer(-1));
      GetDItem(dlog, 4, itemType, item, box);
      SetCtlValue(ControlHandle(item), 1);
      for k := 8 to 12 do begin
         GetDItem(dlog, k, itemType, item, box);
         HiliteControl(ControlHandle(item), 255);
      end;
      ShowWindow(WindowPtr(dlog));
      itemHit := 0;
      while itemHit <> 1 do begin
         ModalDialog(nil, itemHit);
         case itemHit of
            4:
               begin
               GetDItem(dlog, 5, itemType, item, box);
               SetCtlValue(ControlHandle(item), 0);
               GetDItem(dlog, 4, itemType, item, box);
               SetCtlValue(ControlHandle(item), 1);
               for k := 8 to 12 do begin
                  GetDItem(dlog, k, itemType, item, box);
                  SetCtlValue(ControlHandle(item), 0);
                  HiliteControl(ControlHandle(item), 255);
               end;
            end;
            5:
               begin
               GetDItem(dlog, 4, itemType, item, box);
               SetCtlValue(ControlHandle(item), 0);
               GetDItem(dlog, 5, itemType, item, box);
               SetCtlValue(ControlHandle(item), 1);
               for k := 8 to 12 do begin
                  GetDItem(dlog, k, itemType, item, box);
                  HiliteControl(ControlHandle(item), 0);
```

```
        end;
        GetDItem(dlog, 8, itemType, item, box);
        SetCtlValue(ControlHandle(item), 1);
        GetDItem(dlog, 10, itemType, item, box);
        SetCtlValue(ControlHandle(item), 1);
      end;
      8:
        begin
        GetDItem(dlog, 9, itemType, item, box);
        SetCtlValue(ControlHandle(item), 0);
        GetDItem(dlog, 8, itemType, item, box);
        SetCtlValue(ControlHandle(item), 1);
      end;
      9:
        begin
        GetDItem(dlog, 8, itemType, item, box);
        SetCtlValue(ControlHandle(item), 0);
        GetDItem(dlog, 9, itemType, item, box);
        SetCtlValue(ControlHandle(item), 1);
      end;
      10:
        begin
        GetDItem(dlog, 11, itemType, item, box);
        SetCtlValue(ControlHandle(item), 0);
        GetDItem(dlog, 12, itemType, item, box);
        SetCtlValue(ControlHandle(item), 0);
        GetDItem(dlog, 10, itemType, item, box);
        SetCtlValue(ControlHandle(item), 1);
      end;
      11:
        begin
        GetDItem(dlog, 10, itemType, item, box);
        SetCtlValue(ControlHandle(item), 0);
        GetDItem(dlog, 12, itemType, item, box);
        SetCtlValue(ControlHandle(item), 0);
        GetDItem(dlog, 11, itemType, item, box);
        SetCtlValue(ControlHandle(item), 1);
      end;
      12:
        begin
        GetDItem(dlog, 10, itemType, item, box);
        SetCtlValue(ControlHandle(item), 0);
        GetDItem(dlog, 11, itemType, item, box);
        SetCtlValue(ControlHandle(item), 0);
        GetDItem(dlog, 12, itemType, item, box);
        SetCtlValue(ControlHandle(item), 1);
      end;
      otherwise
    end;
  end;
  GetDItem(dlog, 4, itemType, item, box);
  if GetCtlValue(ControlHandle(item)) = 1 then begin
    Sim.addType := AFloating;
```

```
      Sim.multType := MFloating;
      Sim.binary := false;
    end
    else begin
      Sim.binary := true;
      GetDItem(dlog, 8, itemType, item, box);
      if GetCtlValue(ControlHandle(item)) = 1 then
        Sim.addType := Overflow;
      GetDItem(dlog, 9, itemType, item, box);
      if GetCtlValue(ControlHandle(item)) = 1 then
        Sim.addType := Saturation;
      GetDItem(dlog, 10, itemType, item, box);
      if GetCtlValue(ControlHandle(item)) = 1 then
        Sim.multType := MagTruncation;
      GetDItem(dlog, 11, itemType, item, box);
      if GetCtlValue(ControlHandle(item)) = 1 then
        Sim.multType := Truncation;
      GetDItem(dlog, 12, itemType, item, box);
      if GetCtlValue(ControlHandle(item)) = 1 then
        Sim.multType := Rounding;
    end;
    GetDItem(dlog, 14, itemType, item, box);
    GetIText(item, text);
    ReadString(text, bbits);
    GetDItem(dlog, 15, itemType, item, box);
    GetIText(item, text);
    ReadString(text, cbits);
    Sim.RNb := round(exp(bbits * ln(2)));
    Sim.RNbc := round(exp((bbits - cbits) * ln(2)));
    DisposDialog(dlog);
  end;

  procedure CheckMax (var x: extended);
    var
      bmax: extended;
      dummy: integer;
  begin
    bmax := (Sim.RNb div 2 - 1) / Sim.RNbc;
    if x > bmax then begin
      ParamText(StringOf(x : 10 : 7), StringOf(bmax : 10 : 7), '', '');
      dummy := CautionAlert(212, nil);
      x := bmax;
    end;
  end;

  procedure CheckMin (var x: extended);
    var
      bmin: extended;
      dummy: integer;
  begin
    bmin := -(Sim.RNb div 2) / Sim.RNbc;
    if x < bmin then begin
      ParamText(StringOf(x : 10 : 7), StringOf(bmin : 10 : 7), '', '');
```

```
      dummy := CautionAlert(212, nil);
      x := bmin;
    end;
  end;

  procedure Quantize2A (var x, y: extended);
    var
      x1, y1: extended;
  begin
    x1 := round(x * Sim.RNbc) / Sim.RNbc;
    y1 := trunc(sqrt(1 - sqr(x1)) * Sim.RNbc) / Sim.RNbc;
    x := x1;
    if y < 0 then
      y := -y1
    else
      y := y1;
  end;

  procedure QuantizeAngle (var x, y: extended);
    var
      x1, x2, y1, y2, ss1, ss2: extended;
  begin
    x1 := x;
    x2 := x;
    y1 := y;
    y2 := y;
    Quantize2A(x1, y1);
    Quantize2A(y2, x2);
    ss1 := sqr(x1) + sqr(y1);
    ss2 := sqr(x2) + sqr(y2);
    if ss1 > ss2 then begin
      x := x1;
      y := y1;
    end
    else begin
      x := x2;
      y := y2;
    end;
    if Sim.binary then begin
      x := BtoR(x, Sim.RNb, Sim.RNbc);
      y := BtoR(y, Sim.RNb, Sim.RNbc);
    end;
  end;

  procedure QuantizeGamma (var x: extended);
  begin
    CheckMin(x);
    CheckMax(x);
    x := round(x * Sim.RNbc) / Sim.RNbc;
    if Sim.binary then
      x := BtoR(x, Sim.RNb, Sim.RNbc);
  end;
```

```
procedure Quantize2M (var x, y: extended);
begin
  x := round(x * Sim.RNbc) / Sim.RNbc;
  y := trunc(Sim.RNbc / x) / Sim.RNbc;
end;

procedure QuantizeMult (var x, y: extended);
  var
    x1, x2, y1, y2: extended;
begin
  CheckMin(x);
  CheckMax(x);
  CheckMin(y);
  CheckMax(y);
  x1 := x;
  x2 := x;
  y1 := y;
  y2 := y;
  Quantize2M(x1, y1);
  Quantize2M(y2, x2);
  if (x1 * y1) > (x2 * y2) then begin
    x := x1;
    y := y1;
  end
  else begin
    x := x2;
    y := y2;
  end;
  if Sim.binary then begin
    x := BtoR(x, Sim.RNb, Sim.RNbc);
    y := BtoR(y, Sim.RNb, Sim.RNbc);
  end;
end;

procedure QuantizeSection (i: integer);
  var
    dNo, nMults, nAngles, temp, k: integer;
begin
  temp := ActiveNode;
  ActiveNode := i;
  GetThetaInfo(dNo, nAngles, nMults);
  ActiveNode := temp;
  for k := 1 to nAngles do
    QuantizeAngle(Net^[i].gamma[2 * k - 1], Net^[i].gamma[2 * k]);
  for k := 1 to (nMults div 2) do
    QuantizeMult(Net^[i].gamma[2 * (nAngles + k) - 1], Net^[i].gamma[2 * (nAngles + k)]);
end;

procedure QuantizeNet;
  var
    i, k: integer;
begin
  for i := 1 to Nnodes do begin
```

```
        case Net^[i].nodeType of
          parallel, serial:
            for k := 1 to (Net^[i].n - 1) do
              QuantizeGamma(Net^[i].gamma[k]);
          parallelRF, serialRF:
            for k := 1 to (Net^[i].n - 2) do
              QuantizeGamma(Net^[i].gamma[k]);
          A0, A1, B1, C1, D1, E1, A2, B2, C2, D2, E2, QL3:
            QuantizeSection(i);
          transformer:
            QuantizeMult(Net^[i].gamma[1], Net^[i].gamma[2]);
          otherwise
        end;
      end;
    end;

end.
```

```
unit SimTools;

interface

  uses
    Declarations, BinaryMath, ResidueMath, Quantizers;

  function SimSum (x, y: extended): extended;
  function SimProd (x, y: extended): extended;
  function SimNeg (x: extended): extended;
  procedure AdjustNet;

implementation

  function SimNeg;
  begin
    case Sim.addType of
      AFloating:
        SimNeg := -x;
      Overflow, Saturation:
        SimNeg := BNeg(x, Sim.RNb, Sim.RNbc);
      AResidue:
        SimNeg := MMtoB(BtoMM(-x));
    end;
  end;

  function SimSum;
  begin
    case Sim.addType of
      AFloating:
        SimSum := x + y;
      Overflow:
        SimSum := BSum(x, y, Sim.RNb, false);
      Saturation:
        SimSum := BSum(x, y, Sim.RNb, true);
      AResidue:
        SimSum := MMtoB(RSum(BtoMM(x), BtoMM(y)));
    end;
  end;

  function SimProd;
  begin
    case Sim.multType of
      MFloating:
        SimProd := x * y;
      Rounding:
        SimProd := BProd(x, y, Sim.RNb, Sim.RNbc, true);
      Truncation:
        SimProd := BProd(x, y, Sim.RNb, Sim.RNbc, false);
      MagTruncation:
        SimProd := MagTrunc(x, y, Sim.RNb, Sim.RNbc);
      MResidue:
        SimProd := MMtoB(AutoScale(BtoMM(x), BtoSM(y), Sim.scale));
```

```
      end;
    end;

    procedure SetResidue;
      var
        dlog: DialogPtr;
        item: Handle;
        box: Rect;
        text: Str255;
        cbits, itemHit, itemType, k: integer;
    begin
      dlog := GetNewDialog(216, nil, pointer(-1));
      ModalDialog(nil, itemHit);
      GetDItem(dlog, 5, itemType, item, box);
      GetIText(item, text);
      ReadString(text, k);
      GetDItem(dlog, 6, itemType, item, box);
      GetIText(item, text);
      ReadString(text, cbits);
      Sim.addType := AResidue;
      Sim.multType := MResidue;
      Sim.p1 := k - 1;
      Sim.p2 := k;
      Sim.p3 := k + 1;
      Sim.P := Sim.p1;
      Sim.P := Sim.P * Sim.p2;
      Sim.P := Sim.P * Sim.p3;
      Sim.scale := Sim.P div round(exp(cbits * ln(2)));
      Sim.binary := false;
      Sim.RNb := Sim.P;
      Sim.RNbc := Sim.scale;
      DisposDialog(dlog);
    end;

    procedure AdjustNet;
      var
        item: integer;
    begin
      item := Alert(213, nil);
      case item of
        1:
          begin
          Sim.addType := AFloating;
          Sim.multType := MFloating;
        end;
        2:
          begin
          SetQuantization;
          QuantizeNet;
        end;
        3:
          begin
          SetResidue;
```

```
      QuantizeNet;
    end;
    otherwise
  end;
end;

end.
```

```
unit Updators;

interface

  uses
    Declarations, BinaryMath, Utilities, SimTools;

  var
    ovcnt: integer;

  procedure UpdateNet (var SysInFiles: FileRecArray);

implementation

  procedure Up0A (i: integer);
  begin
    with Net^[i] do begin
      if astat[1] and astat[2] and (not bstat[1]) then begin
        bval[1] := SimSum(SimProd(gamma[2], aval[2]), SimNeg(SimProd(gamma[1], aval[1])));
        bval[2] := SimSum(SimProd(gamma[2], aval[1]), SimProd(gamma[1], aval[2]));
        Net^[bnode[1]].astat[bport[1]] := true;
        Net^[bnode[1]].aval[bport[1]] := bval[1];
        bstat[1] := true;
        Net^[bnode[2]].astat[bport[2]] := true;
        Net^[bnode[2]].aval[bport[2]] := bval[2];
        bstat[2] := true;
      end;
    end;
  end;

  procedure Up1A (i: integer);
  begin
    with Net^[i] do begin
      if astat[1] and (not bstat[2]) then begin
        aval[-2] := SimSum(SimProd(gamma[2], aval[1]), SimProd(gamma[1], aval[-1]));
        bval[2] := SimSum(SimProd(gamma[2], aval[-1]), SimNeg(SimProd(gamma[1], aval[1])));
        Net^[bnode[2]].astat[bport[2]] := true;
        Net^[bnode[2]].aval[bport[2]] := bval[2];
        bstat[2] := true;
      end;
      if astat[2] and bstat[2] and (not bstat[1]) then begin
        aval[-1] := SimNeg(SimSum(SimProd(gamma[2], aval[2]), SimProd(gamma[1], aval[-2])));
        bval[1] := SimSum(SimProd(gamma[2], aval[-2]), SimNeg(SimProd(gamma[1], aval[2])));
        Net^[bnode[1]].astat[bport[1]] := true;
        Net^[bnode[1]].aval[bport[1]] := bval[1];
        bstat[1] := true;
      end;
    end;
  end;

  procedure Up1B (i: integer);
  begin
    with Net^[i] do begin
```

```
      if astat[1] and (not bstat[2]) then begin
        aval[-2] := SimNeg(SimSum(SimProd(gamma[2], aval[1]), SimProd(gamma[1], aval[-1])));
        bval[2] := SimSum(SimProd(gamma[2], aval[-1]), SimNeg(SimProd(gamma[1], aval[1])));
        Net^[bnode[2]].astat[bport[2]] := true;
        Net^[bnode[2]].aval[bport[2]] := bval[2];
        bstat[2] := true;
      end;
      if astat[2] and bstat[2] and (not bstat[1]) then begin
        aval[-1] := SimSum(SimProd(gamma[2], aval[2]), SimProd(gamma[1], aval[-2]));
        bval[1] := SimSum(SimProd(gamma[2], aval[-2]), SimNeg(SimProd(gamma[1], aval[2])));
        Net^[bnode[1]].astat[bport[1]] := true;
        Net^[bnode[1]].aval[bport[1]] := bval[1];
        bstat[1] := true;
      end;
    end;
  end;

  procedure Up1C (i: integer);
  begin
    with Net^[i] do begin
      if astat[1] and (not bstat[2]) then begin
        aval[-2] := SimSum(SimProd(gamma[2], aval[1]), SimProd(gamma[1], aval[-1]));
        bval[2] := SimSum(SimProd(gamma[2], aval[-1]), SimNeg(SimProd(gamma[1], aval[1])));
        Net^[bnode[2]].astat[bport[2]] := true;
        Net^[bnode[2]].aval[bport[2]] := bval[2];
        bstat[2] := true;
      end;
      if astat[2] and bstat[2] and (not bstat[1]) then begin
        aval[-1] := SimSum(SimProd(gamma[2], aval[2]), SimProd(gamma[1], aval[-2]));
        bval[1] := SimSum(SimProd(gamma[2], aval[-2]), SimNeg(SimProd(gamma[1], aval[2])));
        Net^[bnode[1]].astat[bport[1]] := true;
        Net^[bnode[1]].aval[bport[1]] := bval[1];
        bstat[1] := true;
      end;
    end;
  end;

  procedure Up1D (i: integer);
  begin
    with Net^[i] do begin
      if astat[1] and (not bstat[2]) then begin
        aval[-2] := SimNeg(SimSum(SimProd(gamma[2], aval[1]), SimProd(gamma[1], aval[-1])));
        bval[2] := SimSum(SimProd(gamma[2], aval[-1]), SimNeg(SimProd(gamma[1], aval[1])));
        Net^[bnode[2]].astat[bport[2]] := true;
        Net^[bnode[2]].aval[bport[2]] := bval[2];
        bstat[2] := true;
      end;
      if astat[2] and bstat[2] and (not bstat[1]) then begin
        aval[-1] := SimNeg(SimSum(SimProd(gamma[2], aval[2]), SimProd(gamma[1], aval[-2])));
        bval[1] := SimSum(SimProd(gamma[2], aval[-2]), SimNeg(SimProd(gamma[1], aval[2])));
        Net^[bnode[1]].astat[bport[1]] := true;
        Net^[bnode[1]].aval[bport[1]] := bval[1];
        bstat[1] := true;
```

```
      end;
    end;
  end;

  procedure Up1E (i: integer);
  begin
    with Net^[i] do begin
      if astat[1] and (not bstat[2]) then begin
        aval[-2] := SimProd((SimSum(SimProd(gamma[4], aval[1]), SimNeg(SimProd(gamma[3],
      aval[-1])))), gamma[5]);
        bval[2] := SimSum(SimProd(gamma[4], aval[-1]), SimProd(gamma[3], aval[1]));
        Net^[bnode[2]].astat[bport[2]] := true;
        Net^[bnode[2]].aval[bport[2]] := bval[2];
        bstat[2] := true;
      end;
      if astat[2] and bstat[2] and (not bstat[1]) then begin
        aval[-1] := SimProd((SimSum(SimProd(gamma[2], aval[2]), SimProd(gamma[1],
      aval[-2]))), gamma[6]);
        bval[1] := SimSum(SimProd(gamma[2], aval[-2]), SimNeg(SimProd(gamma[1], aval[2])));
        Net^[bnode[1]].astat[bport[1]] := true;
        Net^[bnode[1]].aval[bport[1]] := bval[1];
        bstat[1] := true;
      end;
    end;
  end;

  procedure Up2A (i: integer);
    var
      t1, t2: extended;
  begin
    with Net^[i] do begin
      if astat[1] and (not bstat[2]) then begin
        t1 := SimNeg(SimSum(SimProd(gamma[4], aval[-2]), SimNeg(SimProd(gamma[3],
      aval[-1]))));
        t2 := SimSum(SimProd(gamma[4], aval[-1]), SimProd(gamma[3], aval[-2]));
        aval[-1] := t1;
        aval[-3] := SimSum(SimProd(gamma[2], aval[1]), SimProd(gamma[1], t2));
        bval[2] := SimSum(SimProd(gamma[2], t2), SimNeg(SimProd(gamma[1], aval[1])));
        Net^[bnode[2]].astat[bport[2]] := true;
        Net^[bnode[2]].aval[bport[2]] := bval[2];
        bstat[2] := true;
      end;
      if astat[2] and bstat[2] and (not bstat[1]) then begin
        aval[-2] := SimSum(SimProd(gamma[2], aval[2]), SimProd(gamma[1], aval[-3]));
        bval[1] := SimSum(SimProd(gamma[2], aval[-3]), SimNeg(SimProd(gamma[1], aval[2])));
        Net^[bnode[1]].astat[bport[1]] := true;
        Net^[bnode[1]].aval[bport[1]] := bval[1];
        bstat[1] := true;
      end;
    end;
  end;

  procedure Up2B (i: integer);
```

```
   var
      t1, t2: extended;
 begin
   with Net^[i] do begin
      if astat[1] and (not bstat[2]) then begin
         t1 := SimSum(SimProd(gamma[4], aval[-2]), SimNeg(SimProd(gamma[3], aval[-1])));
         t2 := SimSum(SimProd(gamma[4], aval[-1]), SimProd(gamma[3], aval[-2]));
         aval[-1] := t1;
         aval[-3] := SimNeg(SimSum(SimProd(gamma[2], aval[1]), SimProd(gamma[1], t2)));
         bval[2] := SimSum(SimProd(gamma[2], t2), SimNeg(SimProd(gamma[1], aval[1])));
         Net^[bnode[2]].astat[bport[2]] := true;
         Net^[bnode[2]].aval[bport[2]] := bval[2];
         bstat[2] := true;
      end;
      if astat[2] and bstat[2] and (not bstat[1]) then begin
         aval[-2] := SimSum(SimProd(gamma[2], aval[2]), SimProd(gamma[1], aval[-3]));
         bval[1] := SimSum(SimProd(gamma[2], aval[-3]), SimNeg(SimProd(gamma[1], aval[2])));
         Net^[bnode[1]].astat[bport[1]] := true;
         Net^[bnode[1]].aval[bport[1]] := bval[1];
         bstat[1] := true;
      end;
   end;
end;

procedure Up2C (i: integer);
   var
      t1, t2, t3: extended;
begin
   with Net^[i] do begin
      if astat[1] and (not bstat[2]) then begin
         t1 := SimProd((SimSum(SimProd(gamma[6], aval[-1]), SimProd(gamma[5], aval[-2]))),
      gamma[7]);
         t2 := SimSum(SimProd(gamma[6], aval[-2]), SimNeg(SimProd(gamma[5], aval[-1])));
         bval[2] := SimSum(SimProd(gamma[2], t1), SimNeg(SimProd(gamma[1], aval[1])));
         t3 := SimSum(SimProd(gamma[2], aval[1]), SimProd(gamma[1], t1));
         Net^[bnode[2]].astat[bport[2]] := true;
         Net^[bnode[2]].aval[bport[2]] := bval[2];
         bstat[2] := true;
         aval[-3] := SimSum(SimProd(gamma[4], t2), SimNeg(SimProd(gamma[3], t3)));
         aval[-1] := SimNeg(SimSum(SimProd(gamma[4], t3), SimProd(gamma[3], t2)));
      end;
      if astat[2] and bstat[2] and (not bstat[1]) then begin
         aval[-2] := SimNeg(SimProd((SimSum(SimProd(gamma[2], aval[2]), SimProd(gamma[1],
      aval[-3]))), gamma[8]));
         bval[1] := SimSum(SimProd(gamma[2], aval[-3]), SimNeg(SimProd(gamma[1], aval[2])));
         Net^[bnode[1]].astat[bport[1]] := true;
         Net^[bnode[1]].aval[bport[1]] := bval[1];
         bstat[1] := true;
      end;
   end;
end;

procedure Up2D (i: integer);
```

```
    var
      t1, t2, t3: extended;
  begin
    with Net^[i] do begin
      if astat[1] and (not bstat[2]) then begin
{t1 := SimSum(SimProd(gamma[6], aval[-1]), SimProd(gamma[5], aval[-2]));}
        t1 := SimProd((SimSum(SimProd(gamma[6], aval[-1]), SimProd(gamma[5], aval[-2]))),
     gamma[7]);
        t2 := SimSum(SimProd(gamma[6], aval[-2]), SimNeg(SimProd(gamma[5], aval[-1])));
        bval[2] := SimSum(SimProd(gamma[2], t1), SimNeg(SimProd(gamma[1], aval[1])));
        t3 := SimSum(SimProd(gamma[2], aval[1]), SimProd(gamma[1], t1));
        Net^[bnode[2]].astat[bport[2]] := true;
        Net^[bnode[2]].aval[bport[2]] := bval[2];
        bstat[2] := true;
        aval[-3] := SimSum(SimProd(gamma[4], t2), SimNeg(SimProd(gamma[3], t3)));
        aval[-1] := SimSum(SimProd(gamma[4], t3), SimProd(gamma[3], t2));
      end;
      if astat[2] and bstat[2] and (not bstat[1]) then begin
{aval[-2] := SimSum(SimProd(gamma[2], aval[2]), SimProd(gamma[1], aval[-3]));}
        aval[-2] := SimProd((SimSum(SimProd(gamma[2], aval[2]), SimProd(gamma[1],
     aval[-3]))), gamma[8]);
        bval[1] := SimSum(SimProd(gamma[2], aval[-3]), SimNeg(SimProd(gamma[1], aval[2])));
        Net^[bnode[1]].astat[bport[1]] := true;
        Net^[bnode[1]].aval[bport[1]] := bval[1];
        bstat[1] := true;
      end;
    end;
  end;

  procedure Up2E (i: integer);
    var
      t1, t2, t3: extended;
  begin
    with Net^[i] do begin
      if astat[1] and (not bstat[2]) then begin
        t1 := SimProd((SimSum(SimProd(gamma[6], aval[-1]), SimProd(gamma[5], aval[-2]))),
     gamma[9]);
        t2 := SimSum(SimProd(gamma[6], aval[-2]), SimNeg(SimProd(gamma[5], aval[-1])));
        bval[2] := SimSum(SimProd(gamma[8], t1), SimProd(gamma[7], aval[1]));
        t3 := SimSum(SimProd(gamma[8], aval[1]), SimNeg(SimProd(gamma[7], t1)));
        Net^[bnode[2]].astat[bport[2]] := true;
        Net^[bnode[2]].aval[bport[2]] := bval[2];
        bstat[2] := true;
        aval[-3] := SimSum(SimProd(gamma[4], t2), SimNeg(SimProd(gamma[3], t3)));
        aval[-1] := SimSum(SimProd(gamma[4], t3), SimProd(gamma[3], t2));
      end;
      if astat[2] and bstat[2] and (not bstat[1]) then begin
        aval[-2] := SimProd((SimSum(SimProd(gamma[2], aval[2]), SimProd(gamma[1],
     aval[-3]))), gamma[10]);
        bval[1] := SimSum(SimProd(gamma[2], aval[-3]), SimNeg(SimProd(gamma[1], aval[2])));
        Net^[bnode[1]].astat[bport[1]] := true;
        Net^[bnode[1]].aval[bport[1]] := bval[1];
        bstat[1] := true;
```

```
      end;
    end;
  end;

  procedure Up2EE (i: integer);
    var
      t4, t5, t6: extended;
  begin
    with Net^[i] do begin
      if astat[1] and (not bstat[2]) then begin
        aval[-3] := aval[1] * gamma[1] + aval[-1] * gamma[2];
        bval[2] := -aval[1] * gamma[2] + aval[-1] * gamma[1];
        Net^[bnode[2]].astat[bport[2]] := true;
        Net^[bnode[2]].aval[bport[2]] := bval[2];
        bstat[2] := true;
      end;
      if astat[2] and bstat[2] and (not bstat[1]) then begin
        t4 := aval[-2] * gamma[7] + aval[2] * gamma[8];
        t5 := -aval[-2] * gamma[8] + aval[2] * gamma[7];
        bval[1] := aval[-3] * gamma[3] + t4 * gamma[4];
        t6 := -aval[-3] * gamma[4] + t4 * gamma[3];
        aval[-2] := -t5 * gamma[5] + t6 * gamma[6];
        aval[-1] := t5 * gamma[6] + t6 * gamma[5];
        Net^[bnode[1]].astat[bport[1]] := true;
        Net^[bnode[1]].aval[bport[1]] := bval[1];
        bstat[1] := true;
      end;
    end;
  end;

  procedure Up3QL (i: integer);
    var
      t1, t2, t3, t4, t5, t6, t7: extended;
      k: integer;
  begin
    with Net^[i] do begin
      if astat[1] and astat[2] and (not bstat[1]) then begin
        t1 := SimSum(SimProd(aval[2], gamma[2]), SimNeg(SimProd(aval[1], gamma[1])));
        t2 := SimSum(SimProd(aval[1], gamma[2]), SimProd(aval[2], gamma[1]));
        t6 := SimSum(SimProd(aval[-1], gamma[8]), SimNeg(SimProd(t2, gamma[7])));
        t7 := SimSum(SimProd(t2, gamma[8]), SimProd(aval[-1], gamma[7]));
        aval[-1] := t7;
        t5 := SimSum(SimProd(aval[-2], gamma[4]), SimNeg(SimProd(t1, gamma[3])));
        t3 := SimSum(SimProd(t1, gamma[4]), SimProd(aval[-2], gamma[3]));
        aval[-2] := SimSum(SimProd(aval[-3], gamma[6]), SimNeg(SimProd(t3, gamma[5])));
        t4 := SimSum(SimProd(t3, gamma[6]), SimProd(aval[-3], gamma[5]));
        aval[-3] := SimNeg(t4);
        bval[1] := SimSum(SimProd(t6, gamma[2]), SimNeg(SimProd(t5, gamma[1])));
        bval[2] := SimSum(SimProd(t5, gamma[2]), SimProd(t6, gamma[1]));
        for k := 1 to 2 do begin
          Net^[bnode[k]].astat[bport[k]] := true;
          Net^[bnode[k]].aval[bport[k]] := bval[k];
          bstat[k] := true;
```

```
          end;
        end;
      end;
    end;

    procedure UpIOPort (var SysInFiles: FileRecArray; i: integer);
      var
        e: integer;
        s: longint;
        r: extended;
    begin
      with Net^[i] do begin
        if (not bstat[1]) then begin
          if SysInFiles[i].name <> '' then begin
            r := bval[1];
            s := SizeOf(r);
            e := FSRead(SysInFiles[i].fRefNum, s, @bval[1]);
            if e > 0 then begin
              ParamText(SysInFiles[i].name, '', '', '');
              e := Alert(149, nil);
              bval[1] := 0;
            end;
          end
          else
            bval[1] := 0;
          case Sim.addType of
            Overflow, Saturation:
              bval[1] := BtoR(bval[1], Sim.RNb, Sim.RNbc);
            otherwise
          end;
          bstat[1] := true;
          Net^[bnode[1]].astat[bport[1]] := true;
          Net^[bnode[1]].aval[bport[1]] := bval[1];
        end;
      end;
    end;

    procedure UpInductor (i: integer);
    begin
      with Net^[i] do begin
        if (not bstat[1]) then begin
          bval[1] := SimNeg(aval[1]);
          bstat[1] := true;
          Net^[bnode[1]].astat[bport[1]] := true;
          Net^[bnode[1]].aval[bport[1]] := bval[1];
        end;
      end;
    end;

    procedure UpCapacitor (i: integer);
    begin
      with Net^[i] do begin
        if (not bstat[1]) then begin
```

```
          bval[1] := aval[1];
          bstat[1] := true;
          Net^[bnode[1]].astat[bport[1]] := true;
          Net^[bnode[1]].aval[bport[1]] := bval[1];
        end;
      end;
    end;

  procedure UpCirculator (i: integer);
    var
      k: integer;
  begin
    with Net^[i] do begin
      if astat[n] and (not bstat[1]) then begin
        bstat[1] := true;
        Net^[bnode[1]].astat[bport[1]] := true;
        Net^[bnode[1]].aval[bport[1]] := aval[n];
      end;
      for k := 1 to (n - 1) do begin
        if astat[k] and (not bstat[k + 1]) then begin
          bstat[k + 1] := true;
          Net^[bnode[k + 1]].astat[bport[k + 1]] := true;
          Net^[bnode[k + 1]].aval[bport[k + 1]] := aval[k];
        end;
      end;
    end;
  end;

  procedure UpTransformer (i: integer);
    var
      k: integer;
  begin
    with Net^[i] do begin
      if astat[1] and (not bstat[2]) then begin
        bstat[2] := true;
        Net^[bnode[2]].astat[bport[2]] := true;
        Net^[bnode[2]].aval[bport[2]] := SimProd(gamma[1], aval[1]);
      end;
      if astat[2] and (not bstat[1]) then begin
        bstat[1] := true;
        Net^[bnode[1]].astat[bport[1]] := true;
        Net^[bnode[1]].aval[bport[1]] := SimProd(gamma[2], aval[2]);
      end;
    end;
  end;

  procedure UpSerial (i: integer);
    var
      f: boolean;
      t: extended;
      k: integer;
  begin
    with Net^[i] do begin
```

```
        f := true;
        for k := 1 to n do
          f := f and astat[k];
        if f and (not astat[0]) then begin
          aval[0] := 0;
          for k := 1 to n do
            aval[0] := SimSum(aval[0], aval[k]);
          astat[0] := true;
          t := 0;
          for k := 1 to (n - 1) do begin
            bval[k] := SimSum(aval[k], SimNeg(SimProd(gamma[k], aval[0])));
            bstat[k] := true;
            Net^[bnode[k]].astat[bport[k]] := true;
            Net^[bnode[k]].aval[bport[k]] := bval[k];
            t := SimSum(t, bval[k]);
          end;
          bval[n] := SimNeg(SimSum(aval[0], t));
          bstat[n] := true;
          Net^[bnode[n]].astat[bport[n]] := true;
          Net^[bnode[n]].aval[bport[n]] := bval[n];
        end;
      end;
    end;

    procedure UpParallel (i: integer);
      var
        f: boolean;
        k: integer;
    begin
      with Net^[i] do begin
        f := true;
        for k := 1 to n do
          f := f and astat[k];
        if f and (not astat[0]) then begin
          bval[0] := 0;
          for k := 1 to (n - 1) do
            bval[0] := SimSum(bval[0], SimProd(gamma[k], SimSum(aval[n], SimNeg(aval[k]))));
          bval[0] := SimNeg(bval[0]);
          bstat[0] := true;
          bval[n] := SimSum(aval[n], bval[0]);
          writeln('node ', i : 2, '   bval ', n : 1, '      ', bval[n] : 20);
          bstat[n] := true;
          Net^[bnode[n]].astat[bport[n]] := true;
          Net^[bnode[n]].aval[bport[n]] := bval[n];
          for k := 1 to (n - 1) do begin
            bval[k] := SimSum(SimSum(bval[n], aval[n]), SimNeg(aval[k]));
            bstat[k] := true;
            writeln('node ', i : 2, '   bval ', k : 1, '      ', bval[k] : 20);
            Net^[bnode[k]].astat[bport[k]] := true;
            Net^[bnode[k]].aval[bport[k]] := bval[k];
          end;
        end;
      end;
```

```
end;

procedure UpSerialRF (i: integer);
  var
    f: boolean;
    t: extended;
    k: integer;
begin
  with Net^[i] do begin
    f := true;
    for k := 1 to (n - 1) do
      f := f and astat[k];
    if f and (not bstat[n]) then begin
      bval[n] := 0;
      for k := 1 to (n - 1) do
        bval[n] := SimSum(bval[n], aval[k]);
      bval[n] := SimNeg(bval[n]);
      writeln('node ', i : 2, '   bval ', n : 1, '      ', bval[n] : 20);
      bstat[n] := true;
      Net^[bnode[n]].astat[bport[n]] := true;
      Net^[bnode[n]].aval[bport[n]] := bval[n];
    end;
    if astat[n] and bstat[n] and (not astat[0]) then begin
      aval[0] := SimSum(aval[n], SimNeg(bval[n]));
      astat[0] := true;
      t := 0;
      for k := 1 to (n - 2) do begin
        bval[k] := SimSum(aval[k], SimNeg(SimProd(gamma[k], aval[0])));
        bstat[k] := true;
        writeln('node ', i : 2, '   bval ', k : 1, '      ', bval[k] : 20);
        Net^[bnode[k]].astat[bport[k]] := true;
        Net^[bnode[k]].aval[bport[k]] := bval[k];
        t := SimSum(t, bval[k]);
      end;
      bval[n - 1] := SimNeg(SimSum(aval[n], t));
      bstat[n - 1] := true;
      writeln('node ', i : 2, '   bval ', n - 1 : 1, '      ', bval[n - 1] : 20);
      Net^[bnode[n - 1]].astat[bport[n - 1]] := true;
      Net^[bnode[n - 1]].aval[bport[n - 1]] := bval[n - 1];
    end;
  end;
end;

procedure UpParallelRF (i: integer);
  var
    f: boolean;
    k: integer;
begin
  with Net^[i] do begin
    f := true;
    for k := 1 to (n - 1) do
      f := f and astat[k];
    if f and (not bstat[0]) then begin
```

```
        bval[0] := 0;
        for k := 1 to (n - 2) do
          bval[0] := SimSum(bval[0], SimProd(gamma[k], SimSum(aval[n - 1],
      SimNeg(aval[k]))));
        bval[0] := SimNeg(bval[0]);
        bstat[0] := true;
        bval[n] := SimSum(bval[0], aval[n - 1]);
        writeln('node ', i : 2, '   bval ', n : 1, '     ', bval[n] : 20);
        bstat[n] := true;
        Net^[bnode[n]].astat[bport[n]] := true;
        Net^[bnode[n]].aval[bport[n]] := bval[n];
      end;
      if bstat[0] and astat[n] and (not bstat[n - 1]) then begin
        bval[n - 1] := SimSum(bval[0], aval[n]);
        bstat[n - 1] := true;
        Net^[bnode[n - 1]].astat[bport[n - 1]] := true;
        Net^[bnode[n - 1]].aval[bport[n - 1]] := bval[n - 1];
        for k := 1 to (n - 2) do begin
          bval[k] := SimSum(SimSum(bval[n - 1], aval[n - 1]), SimNeg(aval[k]));
          bstat[k] := true;
          writeln('node ', i : 2, '   bval ', k : 1, '     ', bval[k] : 20);
          Net^[bnode[k]].astat[bport[k]] := true;
          Net^[bnode[k]].aval[bport[k]] := bval[k];
        end;
      end;
    end;
end;

procedure UpdateNet;
  var
    k: integer;
begin
  for k := 1 to Nnodes do begin
    case Net^[k].nodeType of
      A0:
        Up0A(k);
      A1:
        Up1A(k);
      B1:
        Up1B(k);
      C1:
        Up1C(k);
      D1:
        Up1D(k);
      E1:
        Up1E(k);
      A2:
        Up2A(k);
      B2:
        Up2B(k);
      C2:
        Up2C(k);
      D2:
```

```
          Up2D(k);
        E2:
          Up2EE(k);
        QL3:
          Up3QL(k);
        parallel:
          UpParallel(k);
        serial:
          UpSerial(k);
        parallelRF:
          UpParallelRF(k);
        serialRF:
          UpSerialRF(k);
        capacitor:
          UpCapacitor(k);
        inductor:
          UpInductor(k);
        circulator:
          UpCirculator(k);
        IOPort:
          UpIOPort(SysInFiles, k);
        transformer:
          UpTransformer(k);
        otherwise
      end;
    end;
  end;

end.
```

```
unit Simulators;

interface

  uses
    Declarations, Complex_Math, BinaryMath, FFTs, Plotting, Utilities, SimTools, Updators;

  procedure Simulate;

implementation

  procedure ResetInputs;
    var
      i, k: integer;
  begin
    for k := 1 to Nnodes do begin
      if Net^[k].nodeType <> none then begin
        for i := -6 to Net^[k].n do
          Net^[k].aval[i] := 0;
      end;
    end;
  end;

  procedure ResetStates;
    var
      i, k: integer;
  begin
    for k := 1 to Nnodes do begin
      if Net^[k].nodeType <> none then begin
        for i := 0 to Net^[k].n do begin
          Net^[k].astat[i] := false;
          Net^[k].bstat[i] := false;
        end;
      end;
    end;
  end;

  procedure SaveOutput (var SysOutFiles: FileRecArray);
    var
      e, k: integer;
      s: longint;
      r: extended;
  begin
    for k := 1 to Nnodes do begin
      if SysOutFiles[k].name <> '' then begin
{   writeln('node ', k : 3, Net^[k].aval[1] : 12 : 8);}
        case Sim.addType of
          AFloating, AResidue:
            r := Net^[k].aval[1];
          Overflow, Saturation:
            r := RtoB(Net^[k].aval[1], Sim.RNb, Sim.RNbc);
{   r := RtoB(Net^[k].aval[1], Sim.RNb, Sim.RNbc) / RtoB((Sim.RNb div 2) - 1, Sim.RNb,
      Sim.RNbc);}
```

```
         end;
         s := SizeOf(r);
         e := FSWrite(SysOutFiles[k].fRefNum, s, @r);
         if e > 0 then begin
            ParamText(SysOutFiles[k].name, '', '', '');
            e := Alert(141, nil);
         end;
      end;
   end;
end;

procedure OpenFiles (var SysInFiles, SysOutFiles: FileRecArray);
   var
      e, k: integer;
begin
   for k := 1 to Nnodes do begin
      if SysInFiles[k].name <> '' then begin
         e := FSOpen(SysInFiles[k].name, SysInFiles[k].vRefNum, SysInFiles[k].fRefNum);
         e := e + SetFPos(SysInFiles[k].fRefNum, 1, 0);
         if e > 0 then begin
            ParamText(SysInFiles[k].name, '', '', '');
            e := Alert(142, nil);
         end;
      end;
      if SysOutFiles[k].name <> '' then begin
         e := FSOpen(SysOutFiles[k].name, SysOutFiles[k].vRefNum, SysOutFiles[k].fRefNum);
         e := e + SetFPos(SysOutFiles[k].fRefNum, 1, 0);
         if e > 0 then begin
            ParamText(SysOutFiles[k].name, '', '', '');
            e := Alert(142, nil);
         end;
      end;
   end;
end;

procedure CloseFiles (var SysInFiles, SysOutFiles: FileRecArray);
   var
      e, k: integer;
begin
   for k := 1 to Nnodes do begin
      if SysInFiles[k].name <> '' then begin
         e := FSClose(SysInFiles[k].fRefNum);
         if e > 0 then begin
            ParamText(SysInFiles[k].name, '', '', '');
            e := Alert(141, nil);
         end;
      end;
      if SysOutFiles[k].name <> '' then begin
         e := FSClose(SysOutFiles[k].fRefNum);
         if e > 0 then begin
            ParamText(SysOutFiles[k].name, '', '', '');
            e := Alert(141, nil);
         end;
```

```
      end;
    end;
  end;

  procedure SetSysInput (var SysInFiles, SysOutFiles: FileRecArray);
    var
      i, k, e: integer;
      r, r1: extended;
      s: longint;
  begin
    OpenFiles(SysInFiles, SysOutFiles);
    for k := 1 to Nnodes do begin
      if SysInFiles[k].name <> '' then begin
        e := SetEof(SysInFiles[k].fRefNum, 0);
        for i := 0 to (Sim.nSamples - 1) do begin
          case Sim.simType of
            impulse, frequency, narrowband:
              if i = 0 then
                r := 1.0
              else
                r := 0.0;
            step:
              r := 1.0;
          end;
          s := SizeOf(r);
          e := e + FSWrite(SysInFiles[k].fRefNum, s, @r);
        end;
        if e > 0 then begin
          ParamText(SysInFiles[k].name, '', '', '');
          e := Alert(141, nil);
        end;
      end;
    end;
    CloseFiles(SysInFiles, SysOutFiles);
  end;

  procedure PlotTime (var SysOutFiles: FileRecArray; i: integer);
    var
      k, e: integer;
      r: extended;
      s: longint;
      title: str255;
      plot: PlotPtr;
  begin
    s := SizeOf(r);
    New(plot);
    for k := 0 to (Sim.nSamples - 1) do begin
      e := FSRead(SysOutFiles[i].fRefNum, s, @r);
      plot^.ypts[k] := r;
      plot^.xpts[k] := k;
    end;
    plot^.window := GetNewWindow(130, nil, pointer(-1));
    plot^.discrete := true;
```

```
    plot^.grid := false;
    plot^.xmax := Sim.nSamples - 1;
    plot^.xmin := 0;
    plot^.ymax := +INF;
    plot^.ymin := -INF;
    plot^.ydiv := 0;
    plot^.xdiv := 5;
    plot^.next := nil;
    SetWRefCon(plot^.window, i);
    title := Concat('Node ', StringOf(i : 2), ' ', StringOf(Sim.simType), ' response');
    SetWTitle(plot^.window, title);
    AddPlot(plot, i);
    ShowWindow(plot^.window);
  end;

  procedure PlotFrequency (var SysOutFiles: FileRecArray; i: integer);
    var
      j, k, e, N2: integer;
      s: longint;
      deltaw, x1, y1: extended;
      c: complex;
      h, g: FFTArrayPtr;
      title: str255;
      plot: array[1..4] of PlotPtr;
  begin
    New(h);
    New(g);
    deltaw := (Sim.wmax - Sim.wmin) / (Sim.nSamples - 1);
    s := SizeOf(c);
    for k := 0 to (Sim.nSamples - 1) do begin
      e := FSRead(SysOutFiles[i].fRefNum, s, @c);
      h^[k] := c;
    end;
    for k := 1 to 4 do begin
      New(plot[k]);
      plot[k]^.window := GetNewWindow(130, nil, pointer(-1));
      plot[k]^.discrete := false;
      plot[k]^.grid := false;
      plot[k]^.xmax := 1;
      plot[k]^.xmin := 0;
      plot[k]^.ymax := +INF;
      plot[k]^.ymin := -INF;
      plot[k]^.ydiv := 0;
      plot[k]^.xdiv := 5;
      plot[k]^.next := nil;
      SetWRefCon(plot[k]^.window, i);
    end;
    title := Concat('Node ', StringOf(i : 2), ' magnitude');
    SetWTitle(plot[1]^.window, title);
    title := Concat('Node ', StringOf(i : 2), ' phase');
    SetWTitle(plot[2]^.window, title);
    title := Concat('Node ', StringOf(i : 2), ' attenuation');
    SetWTitle(plot[3]^.window, title);
```

```
        plot[3]^.ymax := 100;
        plot[3]^.ymin := -100;
        title := Concat('Node ', StringOf(i : 2), ' group delay');
        SetWTitle(plot[4]^.window, title);
        plot[4]^.ymax := 100;
        plot[4]^.ymin := -100;
        g^ := h^;
        IDFT(g, Sim.nSamples);
        for k := 0 to (Sim.nSamples - 1) do
          g^[k].re := g^[k].re * k;
        FFT(g, Sim.nSamples);
        for k := 0 to (Sim.nSamples - 1) do begin
{writeln(k:6, '    ', h[k].re : 10 : 8, '    ', h[k].im : 10 : 8, '      ', g[k].re : 10 : 8, '      ', g[k].im : 10 :
          8);}
          plot[1]^.ypts[k] := Cabs(h^[k]);
          plot[2]^.ypts[k] := Carg(h^[k]) / pi;
          plot[3]^.ypts[k] := Atten(h^[k]);
          if Cabs(h^[k]) > 0 then begin
            c := Cdiv(g^[k], h^[k]);
            plot[4]^.ypts[k] := c.re
          end
          else
            plot[4]^.ypts[k] := +INF;
          for j := 1 to 4 do
            plot[j]^.xpts[k] := Sim.wmin + k * deltaw;
        end;
        if LPtoBP then begin
          N2 := Sim.nSamples div 2;
          plot[3]^.xpts[k] := 1;
          plot[3]^.ypts[Sim.nSamples] := plot[3]^.ypts[N2];
          for k := N2 to (Sim.nSamples - 1) do begin
            x1 := plot[3]^.xpts[k];
            y1 := plot[3]^.ypts[k];
            plot[3]^.xpts[k] := (plot[3]^.xpts[k - N2] + 1) / 2;
            plot[3]^.ypts[k] := plot[3]^.ypts[k - N2];
            plot[3]^.xpts[k - N2] := (x1 - 1) / 2;
            plot[3]^.ypts[k - N2] := y1;
          end;
        end;
        for k := 1 to 3 do begin
          AddPlot(plot[k], i);
          MoveWindow(plot[k]^.window, k * 20, 30 + k * 20, false);
          ShowWindow(plot[k]^.window);
          SelectWindow(plot[k]^.window);
        end;
        Dispose(h);
        Dispose(g);
      end;

  procedure PlotNarrowband (var SysOutFiles: FileRecArray; i: integer);
    var
      j, k, e: integer;
      s: longint;
```

```
        deltaw: extended;
        c: complex;
        h: FFTArrayPtr;
        title: str255;
        plot: array[1..3] of PlotPtr;
        xxpts, yypts: PlotArray;
  begin
     New(h);
     deltaw := (Sim.wmax - Sim.wmin) / (Sim.nSamples - 1);
     s := SizeOf(c);
     for k := 0 to (Sim.nSamples - 1) do begin
        e := FSRead(SysOutFiles[i].fRefNum, s, @c);
        h^[k] := c;
     end;
     for k := 1 to 3 do begin
        New(plot[k]);
        plot[k]^.window := GetNewWindow(130, nil, pointer(-1));
        plot[k]^.discrete := false;
        plot[k]^.grid := false;
        plot[k]^.xmax := Sim.wmax / Pi;
        plot[k]^.xmin := Sim.wmin / Pi;
        plot[k]^.ymax := +INF;
        plot[k]^.ymin := -INF;
        plot[k]^.ydiv := 0;
        plot[k]^.xdiv := 5;
        plot[k]^.next := nil;
        SetWRefCon(plot[k]^.window, i);
     end;
     title := Concat('Node ', StringOf(i : 2), ' magnitude');
     SetWTitle(plot[1]^.window, title);
     title := Concat('Node ', StringOf(i : 2), ' phase');
     SetWTitle(plot[2]^.window, title);
     title := Concat('Node ', StringOf(i : 2), ' attenuation');
     SetWTitle(plot[3]^.window, title);
     plot[3]^.ymax := 100;
     plot[3]^.ymin := -100;
     for k := 0 to (Sim.nSamples - 1) do begin
{writeln(k:6, '    ', h[k].re : 10 : 8, '    ', h[k].im : 10 : 8, '    ', g[k].re : 10 : 8, '    ', g[k].im : 10 :
   8);}
        plot[1]^.ypts[k] := Cabs(h^[k]);
        plot[2]^.ypts[k] := Carg(h^[k]) / pi;
        plot[3]^.ypts[k] := Atten(h^[k]);
        for j := 1 to 3 do
           plot[j]^.xpts[k] := (Sim.wmin + k * deltaw) / pi;
     end;
     if LPtoBP then begin
        for k := 0 to (Sim.nSamples - 1) do begin
           xxpts[k + Sim.nSamples] := (plot[3]^.xpts[k] + 1) / 2;
           xxpts[Sim.nSamples - k - 1] := (1 - plot[3]^.xpts[k]) / 2;
           yypts[k + Sim.nSamples] := plot[3]^.ypts[k];
           yypts[Sim.nSamples - k - 1] := plot[3]^.ypts[k];
        end;
        plot[3]^.xpts := xxpts;
```

```
        plot[3]^.ypts := yypts;
        plot[3]^.xmin := xxpts[0];
        plot[3]^.xmax := xxpts[2 * Sim.nSamples - 1];
      end;
      for k := 1 to 3 do begin
        AddPlot(plot[k], i);
        MoveWindow(plot[k]^.window, k * 20, 30 + k * 20, false);
        ShowWindow(plot[k]^.window);
        SelectWindow(plot[k]^.window);
      end;
      Dispose(h);
    end;

  procedure PlotOutput (var SysInFiles, SysOutFiles: FileRecArray);
    var
      i: integer;
  begin
    OpenFiles(SysInFiles, SysOutFiles);
    for i := 1 to Nnodes do begin
      if SysOutFiles[i].name <> '' then begin
        case Sim.simType of
          user, impulse, step:
            PlotTime(SysOutFiles, i);
          narrowband:
            PlotNarrowband(SysOutFiles, i);
          frequency:
            PlotFrequency(SysOutFiles, i);
        end;
      end;
    end;
    CloseFiles(SysInFiles, SysOutFiles);
  end;

  procedure DeleteFiles (var SysInFiles: FileRecArray);
    var
      e, k: integer;
  begin
    for k := 1 to Nnodes do begin
      if Copy(SysInFiles[k].name, 1, 9) = 'SysInput.' then begin
        e := FSDelete(SysInFiles[k].name, SysInFiles[k].vRefNum);
        SysInFiles[k].name := '';
      end;
    end;
  end;

  function ControlLoop (var SysInFiles, SysOutFiles: FileRecArray): boolean;
    var
      i, k, l, oldCnt, newCnt: integer;
  begin
    ControlLoop := true;
    OpenFiles(SysInFiles, SysOutFiles);
    ResetInputs;
    l := 1;
```

```
    while l <= Sim.nSamples do begin
      ResetStates;
      newCnt := 1;
      oldCnt := 4000;
      while newCnt > 0 do begin
        UpdateNet(SysInFiles);
        newCnt := 0;
        for k := 1 to Nnodes do begin
          if Net^[k].nodeType <> none then begin
            for i := 1 to Net^[k].n do begin
              if (not Net^[k].bstat[i]) then
                newCnt := newCnt + 1;
            end;
          end;
        end;
        if oldCnt = newCnt then begin
          k := CautionAlert(215, nil);
          ControlLoop := false;
          l := Sim.nSamples;
          newCnt := 0;
        end;
        oldCnt := NewCnt;
      end;
      SaveOutput(SysOutFiles);
      l := l + 1;
    end;
    CloseFiles(SysInFiles, SysOutFiles);
end;

function GetUserResp (s: string): extended;
  var
    item, kind: integer;
    str: Str255;
    dlog: DialogPtr;
    box: Rect;
    hand: Handle;
    t: extended;
begin
  ParamText(s, '', '', '');
  dlog := GetNewDialog(145, nil, pointer(-1));
  ModalDialog(nil, item);
  GetDItem(dlog, 3, kind, hand, box);
  GetIText(hand, str);
  ReadString(str, t);
  DisposDialog(dlog);
  GetUserResp := t;
end;

procedure NBResponse (var SysInFiles, SysOutFiles: FileRecArray);
  var
    i, k, e, npts: integer;
    x, vec: FFTArrayPtr;
    s: longint;
```

```
      w, r, deltaw: extended;
      c: Complex;
      dlog: DialogPtr;
      itemType: integer;
      item: Handle;
      box: Rect;
  begin
    New(x);
    New(vec);
    npts := Sim.nSamples;
    Sim.nSamples := Round(GetUserResp('Enter number of frequency domain points:'));
    Sim.wmin := GetUserResp('Enter minimum normalized frequency:') * Pi;
    Sim.wmax := GetUserResp('Enter maximum normalized frequency:') * Pi;
    dlog := GetNewDialog(200, nil, pointer(-1));
    GetDItem(dlog, 1, itemType, item, box);
    SetIText(item, 'Computing the narrowband response ...');
    deltaw := (Sim.wmax - Sim.wmin) / (Sim.nSamples - 1);
    OpenFiles(SysInFiles, SysOutFiles);
    for k := 1 to Nnodes do begin
      if SysOutFiles[k].name <> '' then begin
        s := SizeOf(r);
        for i := 0 to (npts - 1) do begin
          e := FSRead(SysOutFiles[k].fRefNum, s, @r);
          x^[i].re := r;
          x^[i].im := 0;
        end;
        i := 0;
        w := Sim.wmin;
        while w <= Sim.wmax do begin
          vec^[i] := DFT(x, w, npts);
          w := w + deltaw;
          i := i + 1;
        end;
        Sim.nSamples := i;
        e := SetFPos(SysOutFiles[k].fRefNum, 1, 0);
        s := SizeOf(c);
        for i := 0 to (Sim.nSamples - 1) do begin
          c := vec^[i];
          e := FSWrite(SysOutFiles[k].fRefNum, s, @c);
        end;
      end;
    end;
    Dispose(x);
    Dispose(vec);
    CloseFiles(SysInFiles, SysOutFiles);
    DisposDialog(dlog);
end;

procedure FreqResponse (var SysInFiles, SysOutFiles: FileRecArray);
  var
    i, k, e, tSamples: integer;
    vec: FFTArrayPtr;
    s: longint;
```

```
      r: extended;
      c: Complex;
      dlog: DialogPtr;
      itemType: integer;
      item: Handle;
      box: Rect;
  begin
    dlog := GetNewDialog(200, nil, pointer(-1));
    GetDItem(dlog, 1, itemType, item, box);
    SetIText(item, 'Computing the frequency response ...');
    New(vec);
    Sim.wmin := 0;
    Sim.wmax := 2;
    OpenFiles(SysInFiles, SysOutFiles);
    for k := 1 to Nnodes do begin
      if SysOutFiles[k].name <> '' then begin
        tSamples := Sim.nSamples;
        s := SizeOf(r);
        for i := 0 to (tSamples - 1) do begin
          e := FSRead(SysOutFiles[k].fRefNum, s, @r);
          vec^[i].re := r;
          vec^[i].im := 0;
        end;
        FFT(vec, tSamples);
        e := SetFPos(SysOutFiles[k].fRefNum, 1, 0);
        s := SizeOf(c);
        for i := 0 to (tSamples - 1) do begin
          c := vec^[i];
          e := FSWrite(SysOutFiles[k].fRefNum, s, @c);
        end;
      end;
    end;
    Dispose(vec);
    Sim.nSamples := tSamples;
    CloseFiles(SysInFiles, SysOutFiles);
    DisposDialog(dlog);
  end;

procedure GetFiles (var SysInFiles, SysOutFiles: FileRecArray);
  var
    k, e: integer;
    s, ss: string;
    reply: SFReply;
    where: Point;
    typeList: SFTypeList;
begin
  where.v := 45;
  typeList[0] := 'TEXT';
  for k := 1 to Nnodes do begin
    SysInFiles[k].name := '';
    SysOutFiles[k].name := '';
    if Net^[k].nodeType = IOPort then begin
      if Net^[k].pict <= 70 then begin
```

```
            if Sim.simType = user then begin
               where.h := 82;
               s := Concat('Select node ', StringOf(k : 2), ' input file:');
               SFGetFile(where, s, nil, 1, typeList, nil, reply);
               if reply.good then begin
                  SysInFiles[k].name := reply.fName;
                  SysInFiles[k].vRefNum := reply.vRefNum;
               end;
            end
            else begin
               SysInFiles[k].name := Concat('SysInput.', StringOf(k));
               SysInFiles[k].vRefNum := 0;
               e := Create(SysInFiles[k].name, 0, '????', 'TEXT');
            end;
         end;
         if (Net^[k].pict <= 6) or (Net^[k].pict >= 71) then begin
            where.h := 104;
            s := Concat('Select node ', StringOf(k : 2), ' output file:');
            ss := Concat('Output', StringOf(k : 2));
            SFPutFile(where, s, ss, nil, reply);
            if reply.good then begin
               SysOutFiles[k].name := reply.fName;
               SysOutFiles[k].vRefNum := reply.vRefNum;
               e := Create(reply.fName, reply.vRefNum, '????', 'TEXT');
            end;
         end;
      end;
   end;
end;

procedure Simulate;
   var
      dlog: DialogPtr;
      itemType: integer;
      item: Handle;
      box: Rect;
      SysInFiles, SysOutFiles: FileRecArray;
      netCopy: NetPtr;
      simulated: boolean;
begin
   New(netCopy);
   netCopy^ := Net^;
   AdjustNet;
   GetFiles(SysInFiles, SysOutFiles);
   Sim.nSamples := Round(GetUserResp('Enter number of sample points for simulation:'));
   SetCursor(GetCursor(4)^^);
   dlog := GetNewDialog(200, nil, pointer(-1));
   GetDItem(dlog, 1, itemType, item, box);
   case Sim.simType of
      frequency:
         begin
         SetIText(item, 'Computing the impulse response ...');
         SetSysInput(SysInFiles, SysOutFiles);
```

```
          simulated := ControlLoop(SysInFiles, SysOutFiles);
          DeleteFiles(SysInFiles);
          DisposDialog(dlog);
          FreqResponse(SysInFiles, SysOutFiles);
        end;
        narrowband:
          begin
          SetIText(item, 'Computing the impulse response ...');
          SetSysInput(SysInFiles, SysOutFiles);
          simulated := ControlLoop(SysInFiles, SysOutFiles);
          DeleteFiles(SysInFiles);
          DisposDialog(dlog);
          NBResponse(SysInFiles, SysOutFiles);
        end;
        user:
          begin
          SetIText(item, 'Computing the user defined response ...');
          simulated := ControlLoop(SysInFiles, SysOutFiles);
          DisposDialog(dlog);
        end;
        impulse, step:
          begin
          SetIText(item, Concat('Computing the ', StringOf(Sim.simType), ' response ...'));
          SetSysInput(SysInFiles, SysOutFiles);
          simulated := ControlLoop(SysInFiles, SysOutFiles);
          DeleteFiles(SysInFiles);
          DisposDialog(dlog);
        end;
      end;
      SetCursor(Arrow);
      if simulated then
        PlotOutput(SysInFiles, SysOutFiles);
      Net^ := netCopy^;
      Dispose(netCopy);
    end;

end.
```

```
unit User_Int;

interface

  uses
    Declarations, BinaryMath, Utilities, Plotting, Builders, Simulators;

  procedure UpEvent (event: EventRecord);
  procedure KeyEvent (event: EventRecord);
  procedure MouseEvent (event: EventRecord);
  procedure ActEvent (event: EventRecord);

implementation

  procedure DrawGrid;
    var
      k: integer;
  begin
    setport(Wind);
    PenPat(gray);
    for k := 1 to 5 do begin
      MoveTo(0, k * 50);
      LineTo(512, k * 50);
    end;
    for k := 1 to 9 do begin
      MoveTo(k * 50, 0);
      LineTo(k * 50, 300);
    end;
    PenPat(black);
  end;

  procedure DrawNet;
    var
      k: integer;
      pict: PicHandle;
      rec: Rect;
  begin
    for k := 1 to Nnodes do begin
      rec := NodeToRect(k);
      pict := Picts[Net^[k].pict];
      DrawPicture(pict, rec);
    end;
  end;

  procedure SaveAsFile;
    var
      where: Point;
      reply: SFReply;
      error, refNum, dummy: integer;
      size: longint;
  begin
    where.h := 104;
    where.v := 45;
```

```
    if (not Built) then
      BuildFile;
    if Built then begin
      if FilterFile.name = '' then
        SFPutFile(where, 'Save current filter as:', 'Untitled', nil, reply)
      else
        SFPutFile(where, 'Save current filter as:', FilterFile.name, nil, reply);
      if reply.good then begin
        size := SizeOf(Network);
        error := Create(reply.fName, reply.vRefNum, 'WDFS', 'FILT');
        error := FSOpen(reply.fName, reply.vRefNum, refNum);
        error := error + SetFPos(refNum, 1, 0);
        error := error + FSWrite(refNum, size, Ptr(Net));
        error := error + FSClose(refNum);
        FilterFile.name := reply.fName;
        FilterFile.vRefNum := reply.vRefNum;
        FilterFile.fRefNum := refNum;
        if error <> 0 then
          dummy := Alert(141, nil)
        else begin
          Saved := true;
          SetWTitle(Wind, reply.fName);
        end;
      end;
    end;
  end;

  procedure SaveFile;
    var
      error, dummy: integer;
      size: longint;
  begin
    if FilterFile.name = '' then
      SaveAsFile;
    if (not Built) then
      BuildFile;
    if Built then begin
      size := SizeOf(Network);
      error := FSOpen(FilterFile.name, FilterFile.vRefNum, FilterFile.fRefNum);
      error := error + SetFPos(FilterFile.fRefNum, 1, 0);
      error := error + FSWrite(FilterFile.fRefNum, size, Ptr(Net));
      error := error + FSClose(FilterFile.fRefNum);
      if error <> 0 then
        dummy := Alert(141, nil)
      else
        Saved := true;
    end;
  end;

  function CloseFile: boolean;
    var
      i, k: integer;
  begin
```

```
      CloseFile := true;
      if Wind <> nil then begin
        CloseFile := false;
        if Saved then begin
          ErasePlots;
          DisposeWindow(Wind);
          Wind := nil;
          CloseFile := true;
        end
        else begin
          k := Alert(140, nil);
          if k <= 2 then begin
            if k = 1 then
              SaveFile;
            ErasePlots;
            DisposeWindow(Wind);
            Wind := nil;
            CloseFile := true;
          end;
        end;
      end;
      for k := 3 to 5 do
        DisableItem(Menus[k], 0);
      for k := 3 to 8 do
        DisableItem(Menus[1], k);
      DrawMenuBar;
    end;

function ThetaDialog (dNo, nAngles, nMults: integer; show: boolean): Gammas;
  var
    k, itemType: integer;
    dlog: DialogPtr;
    item: Handle;
    text: str255;
    box: Rect;
    temp: Gammas;
    angle, mult: extended;
begin
  if nAngles > 0 then begin
    dlog := GetNewDialog(dNo, nil, pointer(-1));
    if show then begin
      for k := 1 to nAngles do begin
        angle := InvCos(Net^[ActiveNode].gamma[2 * k - 1], Net^[ActiveNode].gamma[2 * k]);
        text := StringOf(angle : 12 : 10);
        GetDItem(dlog, k + 2, itemType, item, box);
        SetIText(item, text);
      end;
      for k := 1 to nMults do begin
        text := StringOf(Net^[ActiveNode].gamma[2 * nAngles + k] : 12 : 10);
        GetDItem(dlog, k + 2 + nAngles, itemType, item, box);
        SetIText(item, text);
      end;
    end;
```

```
          ModalDialog(nil, k);
          for k := 1 to nAngles do begin
            GetDItem(dlog, k + 2, itemType, item, box);
            GetIText(item, text);
            ReadString(text, angle);
            temp[2 * k - 1] := cos(angle);
            temp[2 * k] := sin(angle);
          end;
          for k := 1 to nMults do begin
            GetDItem(dlog, k + 2 + nAngles, itemType, item, box);
            GetIText(item, text);
            ReadString(text, temp[2 * nAngles + k]);
          end;
          for k := (2 * nAngles + nMults + 1) to Ngammas do
            temp[k] := 0;
          DisposDialog(dlog);
          ThetaDialog := temp;
        end;
      end;

      function MultDialog (pic, nCoeff: integer; show: boolean): Gammas;
        var
          k, itemType: integer;
          dlog: DialogPtr;
          item: Handle;
          text: str255;
          box: Rect;
          temp: gammas;
      begin
        if nCoeff > 0 then begin
          dlog := GetNewDialog(140 - nCoeff, nil, pointer(-1));
          GetDItem(dlog, 2, itemType, item, box);
          SetDItem(dlog, 2, itemType, Handle(Picts[pic]), box);
          if show then begin
            for k := 1 to nCoeff do begin
              text := StringOf(Net^[ActiveNode].gamma[k] : 12 : 10);
              writeln(text);
              GetDItem(dlog, k + 2, itemType, item, box);
              SetIText(item, text);
            end;
          end;
          ModalDialog(nil, k);
          for k := 1 to nCoeff do begin
            GetDItem(dlog, k + 2, itemType, item, box);
            GetIText(item, text);
            ReadString(text, temp[k]);
          end;
          for k := (nCoeff + 1) to Ngammas do
            temp[k] := 0;
          DisposDialog(dlog);
          MultDialog := temp;
        end;
      end;
```

```
function ItemDialog (dlogNo, hiBound: integer): integer;
  var
    itemHit, lastItem, itemType, temp: integer;
    itemHandle: Handle;
    box: Rect;
    dlog: DialogPtr;
begin
  temp := 0;
  itemHit := 99;
  lastItem := 0;
  dlog := GetNewDialog(dlogNo, nil, pointer(-1));
  SetPort(WindowPtr(dlog));
  while itemHit > 2 do begin
    ModalDialog(nil, itemHit);
    if itemHit = 2 then begin
      temp := 0;
    end;
    if (itemHit >= 3) and (itemHit <= hiBound) then begin
      if lastItem > 0 then begin
        GetDItem(dlog, lastItem, itemType, itemHandle, box);
        InvertRect(box);
      end;
      GetDItem(dlog, itemHit, itemType, itemHandle, box);
      InvertRect(box);
      temp := itemHit;
    end;
    lastItem := itemHit;
  end;
  DisposDialog(dlog);
  SetPort(Wind);
  if temp > 0 then begin
    Built := false;
    DisableItem(Menus[5], 0);
    Saved := false;
  end;
  ItemDialog := temp;
end;

procedure ParRFItem;
  var
    item: integer;
begin
  item := ItemDialog(135, 22);
  case item of
    3..6:
      begin
      Net^[ActiveNode].n := 2;
      Net^[ActiveNode].pict := item + 12;
      Net^[ActiveNode].nodeType := ParallelRF;
    end;
    7..18:
      begin
```

```
        Net^[ActiveNode].n := 3;
        Net^[ActiveNode].pict := item + 22;
        Net^[ActiveNode].nodeType := ParallelRF;
      end;
      19..22:
        begin
        Net^[ActiveNode].n := 4;
        Net^[ActiveNode].pict := item + 39;
        Net^[ActiveNode].nodeType := ParallelRF;
      end;
      otherwise
    end;
    if (item >= 3) and (item <= 22) then
      Net^[ActiveNode].gamma := MultDialog(Net^[ActiveNode].pict, Net^[ActiveNode].n - 2,
     false);
  end;

  procedure SerRFItem;
    var
      item: integer;
  begin
    item := ItemDialog(134, 22);
    case item of
      3..6:
        begin
        Net^[ActiveNode].n := 2;
        Net^[ActiveNode].pict := item + 18;
        Net^[ActiveNode].nodeType := SerialRF;
      end;
      7..18:
        begin
        Net^[ActiveNode].n := 3;
        Net^[ActiveNode].pict := item + 38;
        Net^[ActiveNode].nodeType := SerialRF;
      end;
      19..22:
        begin
        Net^[ActiveNode].n := 4;
        Net^[ActiveNode].pict := item + 44;
        Net^[ActiveNode].nodeType := SerialRF;
      end;
      otherwise
    end;
    if (item >= 3) and (item <= 22) then
      Net^[ActiveNode].gamma := MultDialog(Net^[ActiveNode].pict, Net^[ActiveNode].n - 2,
     false);
  end;

  procedure ParItem;
    var
      item: integer;
  begin
    item := ItemDialog(133, 9);
```

```
      case item of
        3..4:
          begin
          Net^[ActiveNode].n := 2;
          Net^[ActiveNode].pict := item + 10;
          Net^[ActiveNode].nodeType := Parallel;
        end;
        5..8:
          begin
          Net^[ActiveNode].n := 3;
          Net^[ActiveNode].pict := item + 20;
          Net^[ActiveNode].nodeType := Parallel;
        end;
        9:
          begin
          Net^[ActiveNode].n := 4;
          Net^[ActiveNode].pict := item + 48;
          Net^[ActiveNode].nodeType := Parallel;
        end;
        otherwise
      end;
      if (item >= 3) and (item <= 9) then
        Net^[ActiveNode].gamma := MultDialog(Net^[ActiveNode].pict, Net^[ActiveNode].n - 1,
       false);
  end;

  procedure SerItem;
    var
      item: integer;
  begin
    item := ItemDialog(132, 9);
    case item of
      3..4:
        begin
        Net^[ActiveNode].n := 2;
        Net^[ActiveNode].pict := item + 16;
        Net^[ActiveNode].nodeType := Serial;
      end;
      5..8:
        begin
        Net^[ActiveNode].n := 3;
        Net^[ActiveNode].pict := item + 36;
        Net^[ActiveNode].nodeType := Serial;
      end;
      9:
        begin
        Net^[ActiveNode].n := 4;
        Net^[ActiveNode].pict := item + 53;
        Net^[ActiveNode].nodeType := Serial;
      end;
      otherwise
    end;
    if (item >= 3) and (item <= 9) then
```

```
        Net^[ActiveNode].gamma := MultDialog(Net^[ActiveNode].pict, Net^[ActiveNode].n - 1,
      false);
  end;

  procedure XfmrItem;
    var
      item: integer;
  begin
    item := ItemDialog(214, 4);
    case item of
      3..4:
        begin
        Net^[ActiveNode].n := 2;
        Net^[ActiveNode].pict := item + 127;
        Net^[ActiveNode].nodeType := Transformer;
      end;
      otherwise
    end;
    if (item >= 3) and (item <= 4) then begin
      Net^[ActiveNode].gamma := MultDialog(Net^[ActiveNode].pict, 1, false);
      Net^[ActiveNode].gamma[2] := 1 / Net^[ActiveNode].gamma[1];
    end;
  end;

  procedure CapItem;
    var
      item: integer;
  begin
    item := ItemDialog(130, 6);
    case item of
      3..6:
        begin
        Net^[ActiveNode].n := 1;
        Net^[ActiveNode].pict := item + 2;
        Net^[ActiveNode].nodeType := Capacitor;
      end;
      otherwise
    end;
  end;

  procedure CircItem;
    var
      item: integer;
  begin
    item := ItemDialog(209, 11);
    case item of
      3..6:
        Net^[ActiveNode].n := 2;
      7..10:
        Net^[ActiveNode].n := 3;
      11:
        Net^[ActiveNode].n := 4;
      otherwise
```

```
    end;
    Net^[ActiveNode].pict := item + 116;
    Net^[ActiveNode].nodeType := Circulator;
  end;

  procedure IOItem;
    var
      item: integer;
  begin
    item := ItemDialog(129, 14);
    case item of
      3..6:
        begin
        Net^[ActiveNode].n := 1;
        Net^[ActiveNode].pict := item - 2;
        Net^[ActiveNode].nodeType := IOPort;
      end;
      7..14:
        begin
        Net^[ActiveNode].n := 1;
        Net^[ActiveNode].pict := item + 60;
        Net^[ActiveNode].nodeType := IOPort;
      end;
      otherwise
    end;
  end;

  procedure IndItem;
    var
      item: integer;
  begin
    item := ItemDialog(131, 6);
    case item of
      3..6:
        begin
        Net^[ActiveNode].n := 1;
        Net^[ActiveNode].pict := item + 6;
        Net^[ActiveNode].nodeType := Inductor;
      end;
      otherwise
    end;
  end;

  procedure Order0_2Port;
    var
      item: integer;
  begin
    item := ItemDialog(207, 6);
    case item of
      3..6:
        begin
        Net^[ActiveNode].nodeType := A0;
        Net^[ActiveNode].n := 2;
```

```
          Net^[ActiveNode].pict := item + 112;
          Net^[ActiveNode].gamma := ThetaDialog(499, 1, 0, false);
        end;
        otherwise
      end;
    end;

    procedure Order1_2Port;
      var
        item, dNo, nMults, nAngles: integer;
    begin
      item := ItemDialog(201, 22);
      case item of
        3..6:
          Net^[ActiveNode].nodeType := A1;
        7..10:
          Net^[ActiveNode].nodeType := B1;
        11..14:
          Net^[ActiveNode].nodeType := C1;
        15..18:
          Net^[ActiveNode].nodeType := D1;
        19..22:
          Net^[ActiveNode].nodeType := E1;
        otherwise
      end;
      if (item >= 3) and (item <= 22) then begin
        GetThetaInfo(dNo, nAngles, nMults);
        Net^[ActiveNode].n := 2;
        Net^[ActiveNode].pict := item + 72;
        Net^[ActiveNode].gamma := ThetaDialog(dNo, nAngles, nMults, false);
      end;
    end;

    procedure Order2_2Port;
      var
        item, dNo, nMults, nAngles: integer;
    begin
      item := ItemDialog(202, 22);
      case item of
        3..6:
          Net^[ActiveNode].nodeType := A2;
        7..10:
          Net^[ActiveNode].nodeType := B2;
        11..14:
          Net^[ActiveNode].nodeType := C2;
        15..18:
          Net^[ActiveNode].nodeType := D2;
        19..22:
          Net^[ActiveNode].nodeType := E2;
        otherwise
      end;
      if (item >= 3) and (item <= 22) then begin
        GetThetaInfo(dNo, nAngles, nMults);
```

```
      Net^[ActiveNode].n := 2;
      Net^[ActiveNode].pict := item + 92;
      Net^[ActiveNode].gamma := ThetaDialog(dNo, nAngles, nMults, false);
    end;
  end;

  procedure Order3_2Port;
    var
      item, dNo, nMults, nAngles: integer;
  begin
    item := ItemDialog(211, 4);
    case item of
      3..4:
        Net^[ActiveNode].nodeType := QL3;
      otherwise
    end;
    if (item >= 3) and (item <= 4) then begin
      GetThetaInfo(dNo, nAngles, nMults);
      Net^[ActiveNode].n := 2;
      Net^[ActiveNode].pict := item + 125;
      Net^[ActiveNode].gamma := ThetaDialog(dNo, nAngles, nMults, false);
    end;
  end;

  procedure Quit;
  begin
    if CloseFile then
      Done := true;
  end;

  procedure NewFile;
    var
      k, i: integer;
  begin
    if CloseFile then begin
      FilterFile.name := '';
      FilterFile.vRefNum := 0;
      FilterFile.fRefNum := 0;
      Wind := GetNewWindow(128, nil, pointer(-1));
      SetPort(Wind);
      for k := 1 to Nnodes do begin
        Net^[k].nodeType := none;
        Net^[k].pict := 0;
        Net^[k].n := 0;
        for i := 1 to Nports do begin
          Net^[k].bport[i] := 0;
          Net^[k].bnode[i] := 0;
        end;
        for i := 1 to Ngammas do
          Net^[k].gamma[i] := 0;
      end;
      Saved := true;
      Built := false;
```

```
      DisableItem(Menus[5], 0);
      ActiveNode := 1;
      ActiveRect := NodeToRect(1);
    end;
  end;

  procedure OpenFile;
    var
      where: Point;
      reply: SFReply;
      error, refNum, dummy, k: integer;
      typeList: SFTypeList;
      size: longint;
  begin
    if CloseFile then begin
      where.h := 82;
      where.v := 45;
      typeList[0] := 'FILT';
      SFGetFile(where, 'Select filter file from disk:', nil, 1, typeList, nil, reply);
      if reply.good then begin
        NewFile;
        size := SizeOf(Network);
        error := FSOpen(reply.fName, reply.vRefNum, refNum);
        error := error + SetFPos(refNum, 1, 0);
        error := error + FSRead(refNum, size, Ptr(Net));
        error := error + FSClose(refNum);
        FilterFile.name := reply.fName;
        FilterFile.vRefNum := reply.vRefNum;
        FilterFile.fRefNum := refNum;
        if error <> 0 then begin
          DisposeWindow(Wind);
          dummy := Alert(142, nil);
        end
        else begin
          Built := true;
          EnableItem(Menus[5], 0);
          Saved := true;
          SetWTitle(Wind, reply.fName);
        end;
      end;
    end;
  end;

  procedure ShowInfo;
    var
      dNo, nMults, nAngles, pic: integer;
  begin
    case Net^[ActiveNode].nodeType of
      A0, A1, B1, C1, D1, E1, A2, B2, C2, D2, E2, QL3:
        begin
        GetThetaInfo(dNo, nAngles, nMults);
        Net^[ActiveNode].gamma := ThetaDialog(dNo, nAngles, nMults, true);
        Saved := false;
```

```
        end;
        parallel, serial:
          begin
          Net^[ActiveNode].gamma := MultDialog(Net^[ActiveNode].pict, Net^[ActiveNode].n - 1,
      true);
          Saved := false;
        end;
        parallelRF, serialRF:
          begin
          Net^[ActiveNode].gamma := MultDialog(Net^[ActiveNode].pict, Net^[ActiveNode].n - 2,
      true);
          Saved := false;
        end;
        transformer:
          begin
          Net^[ActiveNode].gamma := MultDialog(Net^[ActiveNode].pict, 1, true);
          Saved := false;
        end;
        otherwise
      end;
  end;

  procedure SpoolOutPICT (var picHand: picHandle);
    var
      where: Point;
      reply: SFReply;
      longZero, byteCount: longint;
      err, fRefNum, k: integer;
  begin
    where.h := 104;
    where.v := 45;
    SFPutFile(where, 'Save the PICT as:', '', nil, reply);
    if reply.good then begin
      err := Create(reply.fName, reply.vRefNum, '????', 'PICT');
      err := FSOpen(reply.fName, reply.vRefNum, fRefNum);
      longZero := 0;
      byteCount := SizeOf(longZero);
      for k := 1 to (512 div byteCount) do
        err := FSWrite(fRefNum, byteCount, @longZero);
      byteCount := picHand^^.picSize;
      err := FSWrite(fRefNum, byteCount, Ptr(picHand^));
    end;
    KillPicture(picHand);
  end;

  procedure FileMenu (item: integer);
    var
      dummy: boolean;
      pic: PicHandle;
      rec: Rect;
  begin
    case item of
      1:
```

```
          NewFile;
        2:
          OpenFile;
        3:
          SaveFile;
        4:
          SaveAsFile;
        5:
          dummy := CloseFile;
        7:
          BuildFile;
        8:
          begin
          rec := thePort^.portRect;
          pic := OpenPicture(rec);
          DrawNet;
          ClosePicture;
          SpoolOutPICT(pic);
        end;
        10:
          Quit;
        otherwise
      end;
  end;

  procedure AdapterMenu (item: integer);
  begin
    case item of
      1:
        SerItem;
      2:
        ParItem;
      4:
        SerRFItem;
      5:
        ParRFItem;
      7:
        Order0_2Port;
      8:
        Order1_2Port;
      9:
        Order2_2Port;
      10:
        Order3_2Port;
      otherwise
    end;
    InvertRect(ActiveRect);
    EraseRect(ActiveRect);
    DrawPicture(Picts[Net^[ActiveNode].pict], ActiveRect);
    DrawGrid;
    InvertRect(ActiveRect);
  end;
```

```
procedure ElementMenu (item: integer);
begin
   case item of
      1:
         CapItem;
      2:
         IndItem;
      3:
         IOItem;
      4:
         XfmrItem;
      5:
         CircItem;
   end;
   InvertRect(ActiveRect);
   EraseRect(ActiveRect);
   DrawPicture(Picts[Net^[ActiveNode].pict], ActiveRect);
   DrawGrid;
   InvertRect(ActiveRect);
end;

procedure SimulateMenu (item: integer);
begin
   case item of
      1:
         Sim.simType := impulse;
      2:
         Sim.simType := step;
      3:
         Sim.simType := user;
      5:
         Sim.simType := frequency;
      6:
         Sim.simType := narrowband;
   end;
   Simulate;
end;

procedure SavePlotPts (var plot: PlotRec);
   var
      n1, n2, k: integer;
      fName: string;
      f: text;
begin
   fName := NewFileName('Enter file name:');
   Rewrite(f, fName);
   n1 := StartPt(plot.xpts, plot.xmin);
   n2 := EndPt(plot.xpts, plot.xmax);
   for k := n1 to n2 do
      writeln(f, plot.xpts[k] : 12 : 8);
   writeln(f);
   for k := n1 to n2 do
      writeln(f, plot.ypts[k] : 12 : 8);
```

```
    Close(f);
  end;

  procedure PlotMenu (item: integer);
    var
      ref: longint;
      plot: PlotPtr;
      w: WindowPtr;
      pic: PicHandle;
      rec: Rect;
  begin
    w := FrontWindow;
    SetPort(w);
    ref := GetWRefCon(w);
    plot := FindPlot(w, ref);
    case item of
      1:
        plot^.ymax := SetPlotParam('Y-max');
      2:
        plot^.ymin := SetPlotParam('Y-min');
      3:
        plot^.xmax := SetPlotParam('X-max');
      4:
        plot^.xmin := SetPlotParam('X-min');
      6:
        plot^.ydiv := Round(SetPlotParam('Y divisions'));
      7:
        plot^.xdiv := Round(SetPlotParam('X divisions'));
      9:
        plot^.grid := not plot^.grid;
      10:
        plot^.discrete := not plot^.discrete;
      11:
        begin
        rec := thePort^.portRect;
        pic := OpenPicture(rec);
        PlotCurve(plot^);
        ClosePicture;
        SpoolOutPICT(pic);
      end;
      12:
        SavePlotPts(plot^);
      otherwise
    end;
    if item <= 10 then
      InvalRect(thePort^.portRect);
  end;

procedure AppleMenu (item: integer);
  var
    temp: integer;
    name: str255;
begin
```

```
    if item = 1 then
      temp := Alert(143, nil)
    else begin
      GetItem(Menus[0], item, name);
      temp := OpenDeskAcc(name);
    end;
  end;

  procedure ResizeWindow (w1: WindowPtr; pt: Point);
    var
      rec: Rect;
      l: longint;
  begin
    SetPort(w1);
    with screenbits.bounds do
      SetRect(rec, 200, 100, right, bottom - 20);
    l := GrowWindow(w1, pt, rec);
    SetRect(rec, 0, 0, LoWord(l), HiWord(l));
    InvalRect(rec);
    SizeWindow(w1, LoWord(l), HiWord(l), true);
  end;

  procedure CloseWindow (w: WindowPtr);
    var
      f: boolean;
      ref: longint;
      plot: PlotPtr;
  begin
    ref := GetWRefCon(w);
    case ref of
      100:
        f := CloseFile;
      1..Nnodes:
        begin
        plot := FindPlot(w, ref);
        RemovePlot(plot, ref);
      end;
      otherwise
        DisposeWindow(w);
    end;
  end;

  procedure ActivateNode (event: EventRecord);
  begin
    InvertRect(ActiveRect);
    GlobalToLocal(event.where);
    ActiveNode := PtToNode(event.where);
    ActiveRect := NodeToRect(ActiveNode);
    InvertRect(ActiveRect);
    if BitAnd(event.modifiers, optionKey) <> 0 then
      ShowInfo;
  end;
```

```
procedure MenuCmnd (result: longint);
  var
    dummy: boolean;
    item, menu: integer;
begin
  menu := HiWord(result);
  if menu > 0 then begin
    item := LoWord(result);
    case menu of
      128:
        AppleMenu(item);
      129:
        FileMenu(item);
      130:
        dummy := SystemEdit(item - 1);
      131:
        ElementMenu(item);
      132:
        AdapterMenu(item);
      133:
        SimulateMenu(item);
      134:
        PlotMenu(item);
    end;
    HiliteMenu(0);
  end;
end;

procedure UpEvent;
  var
    w: WindowPtr;
    ref: longint;
    plot: PlotPtr;
begin
  w := WindowPtr(event.message);
  ref := GetWRefCon(w);
  SetPort(w);
  BeginUpdate(w);
  EraseRect(thePort^.portRect);
  case ref of
    100:
      begin
      DrawNet;
      DrawGrid;
      InvertRect(ActiveRect);
    end;
    1..Nnodes:
      begin
      DrawGrowIcon(w);
      plot := FindPlot(w, ref);
      PlotCurve(plot^);
    end;
    otherwise
```

```
    end;
    EndUpdate(w);
  end;

  procedure KeyEvent;
    var
      acode: integer;
  begin
    acode := BitAnd(event.message, charCodeMask);
    if BitAnd(event.modifiers, cmdKey) <> 0 then
      MenuCmnd(MenuKey(chr(acode)));
    if acode = 8 then begin
      Net^[ActiveNode].n := 0;
      Net^[ActiveNode].pict := 0;
      Net^[ActiveNode].nodeType := none;
      InvertRect(ActiveRect);
      EraseRect(ActiveRect);
      InvertRect(ActiveRect);
      DrawGrid;
    end;
  end;

  procedure MouseEvent;
    var
      w1: WindowPtr;
  begin
    case FindWindow(event.where, w1) of
      inSysWindow:
        SystemClick(event, w1);
      inMenuBar:
        MenuCmnd(MenuSelect(event.where));
      inDrag:
        DragWindow(w1, event.where, DragRect);
      inContent:
        if w1 <> FrontWindow then
          SelectWindow(w1)
        else if w1 = Wind then
          ActivateNode(event);
      inGoAway:
        CloseWindow(w1);
      inGrow:
        ResizeWindow(w1, event.where);
      inDesk:
        ;
    end;
  end;

  procedure ActEvent;
    var
      w: WindowPtr;
      k: integer;
  begin
    if BitAnd(event.modifiers, activeFlag) <> 0 then begin
```

```
        w := WindowPtr(event.message);
        SetPort(w);
        case GetWRefCon(w) of
          1..Nnodes:
            begin
            DrawGrowIcon(w);
            for k := 2 to 5 do
              DisableItem(Menus[k], 0);
            EnableItem(Menus[6], 0);
          end;
          100:
            begin
            for k := 3 to 8 do
              EnableItem(Menus[1], k);
            EnableItem(Menus[3], 0);
            EnableItem(Menus[4], 0);
            DisableItem(Menus[2], 0);
            DisableItem(Menus[6], 0);
            DisableItem(Menus[1], 6);
            if Built then
              EnableItem(Menus[5], 0);
          end;
          otherwise
            begin
            EnableItem(Menus[2], 0);
            DisableItem(Menus[1], 0);
            DisableItem(Menus[3], 0);
            DisableItem(Menus[4], 0);
            DisableItem(Menus[5], 0);
            DisableItem(Menus[6], 0);
          end;
        end;
      end
      else begin
        for k := 2 to 6 do
          DisableItem(Menus[k], 0);
        for k := 3 to 8 do
          DisableItem(Menus[1], k);
      end;
      DrawMenuBar;
    end;

end.
```

```
program WDF_Sim;

  uses
    Declarations, Utilities, Plotting, User_Int;

  var
    event: EventRecord;

  procedure SetPicts;
    type
      connectPtr = ^Connections;
      connectHand = ^ConnectPtr;
    var
      k: integer;
      cHand: connectHand;
      cPtr: connectPtr;
      rec: rect;
  begin
    for k := 0 to Npicts do
      Picts[k] := GetPicture(k + 128);
    cHand := connectHand(GetResource('PORT', 128));
    cPtr := cHand^;
    PortCons := cPtr^;
  end;

  procedure SetMenus;
    var
      k: integer;
  begin
    for k := 0 to 6 do
      Menus[k] := GetMenu(128 + k);
    AddResMenu(Menus[0], 'DRVR');
    for k := 0 to 6 do
      InsertMenu(Menus[k], 0);
    for k := 2 to 6 do
      DisableItem(Menus[k], 0);
    for k := 3 to 8 do
      DisableItem(Menus[1], k);
    DrawMenuBar;
  end;

begin
  InitGraf(@thePort);
  InitFonts;
  FlushEvents(everyEvent, 0);
  InitWindows;
  InitMenus;
  TEInit;
  InitDialogs(nil);
  SetMenus;
  SetPicts;
  InitPlots;
  with screenbits.bounds do
```

```
    SetRect(DragRect, 4, 24, right - 4, bottom - 4);
  Done := false;
  New(Net);
  Wind := nil;
  InitCursor;
  Built := false;
  repeat
    SystemTask;
    if GetNextEvent(everyEvent, event) then begin
      case event.what of
        mouseDown:
          MouseEvent(event);
        keyDown:
          KeyEvent(event);
        activateEvt:
          ActEvent(event);
        updateEvt:
          UpEvent(event);
        otherwise
      end;
    end;
  until Done;
  Dispose(Net);
end.
```

APPENDIX B

EXAMPLE PROGRAM TO GENERATE A FILE CONTAINING A SEQUENCE OF SIGNAL SAMPLE VALUES

(pages 185-186)

```
program SetInput;

   var
      where: Point;
      reply: SFReply;
      error, refNum, dummy, k: integer;
      size: longint;
      zero, one: extended;

begin
   zero := 0;
   one := 1;
   where.h := 104;
   where.v := 45;
   SFPutFile(where, 'Save data file as:', 'Untitled', nil, reply);
   if reply.good then begin
      size := SizeOf(zero);
      error := Create(reply.fName, reply.vRefNum, '????', 'TEXT');
      error := FSOpen(reply.fName, reply.vRefNum, refNum);
      error := error + SetFPos(refNum, 1, 0);

{ Here is the definition of the input signal }

      for k := 0 to 39 do
         error := error + FSWrite(refNum, size, @zero);
      for k := 40 to 80 do
         error := error + FSWrite(refNum, size, @one);
      for k := 81 to 200 do
         error := error + FSWrite(refNum, size, @zero);

      error := error + FSClose(refNum);
   end;
end.
```

APPENDIX C

PROGRAMS TO GENERATE THE PICTURES FOR WDFSIM AND SAVE THEM IN A RESOURCE FILE AS RESOURCE TYPE 'PICT'

(pages 187-248)

```
program PictRefs;

  var
    pic: PicHandle;
    r1, r2: Rect;
    ref: integer;

  procedure Box;
  begin
    PenSize(2, 2);
    MoveTo(8, 8);
    LineTo(40, 8);
    LineTo(40, 40);
    LineTo(8, 40);
    LineTo(8, 8);
    PenSize(1, 1);
  end;

  procedure NPort;
  begin
    MoveTo(14, 0);
    LineTo(14, 7);
    MoveTo(35, 0);
    LineTo(35, 7);
    MoveTo(12, 2);
    LineTo(14, 0);
    LineTo(16, 2);
  end;

  procedure SPort;
  begin
    MoveTo(14, 42);
    LineTo(14, 49);
    MoveTo(35, 42);
    LineTo(35, 49);
    MoveTo(33, 47);
    LineTo(35, 49);
    LineTo(37, 47);
  end;

  procedure EPort;
  begin
    MoveTo(42, 14);
    LineTo(49, 14);
    MoveTo(42, 35);
    LineTo(49, 35);
    MoveTo(47, 12);
    LineTo(49, 14);
    LineTo(47, 16);
  end;

  procedure WPort;
  begin
```

```
    MoveTo(0, 14);
    LineTo(7, 14);
    MoveTo(0, 35);
    LineTo(7, 35);
    MoveTo(2, 33);
    LineTo(0, 35);
    LineTo(2, 37);
  end;

  procedure NRFree;
  begin
    MoveTo(12, 13);
    LineTo(16, 13);
    MoveTo(14, 10);
    LineTo(14, 12);
  end;

  procedure SRFree;
  begin
    MoveTo(33, 36);
    LineTo(37, 36);
    MoveTo(35, 37);
    LineTo(35, 39);
  end;

  procedure ERFree;
  begin
    MoveTo(36, 12);
    LineTo(36, 16);
    MoveTo(37, 14);
    LineTo(39, 14);
  end;

  procedure WRFree;
  begin
    MoveTo(13, 33);
    LineTo(13, 37);
    MoveTo(10, 35);
    LineTo(12, 35);
  end;

  procedure Parallel;
  begin
    MoveTo(22, 20);
    LineTo(22, 29);
    MoveTo(26, 20);
    LineTo(26, 29);
  end;

  procedure Serial;
  begin
    MoveTo(20, 25);
    LineTo(29, 25);
```

```
    MoveTo(24, 24);
    LineTo(25, 24);
    MoveTo(24, 26);
    LineTo(25, 26);
  end;

  procedure NGamma (s: Str255);
  begin
    TextFont(symbol);
    TextSize(9);
    MoveTo(20, 17);
    DrawString(s);
  end;

  procedure SGamma (s: Str255);
  begin
    TextFont(symbol);
    TextSize(9);
    MoveTo(20, 37);
    DrawString(s);
  end;

  procedure WGamma (s: Str255);
  begin
    TextFont(symbol);
    TextSize(9);
    MoveTo(11, 27);
    DrawString(s);
  end;

  procedure EGamma (s: Str255);
  begin
    TextFont(symbol);
    TextSize(9);
    MoveTo(30, 27);
    DrawString(s);
  end;

  procedure Delay;
  begin
    PenSize(2, 2);
    MoveTo(18, 18);
    LineTo(30, 18);
    LineTo(30, 30);
    LineTo(18, 30);
    LineTo(18, 18);
    PenSize(1, 1);
    MoveTo(22, 22);
    LineTo(26, 22);
    MoveTo(24, 23);
    LineTo(24, 27);
  end;
```

```
    procedure NCap;
    begin
        Delay;
        NPort;
        MoveTo(32, 24);
        LineTo(35, 24);
        LineTo(35, 8);
        MoveTo(17, 24);
        LineTo(14, 24);
        LineTo(14, 8);
    end;

    procedure SCap;
    begin
        Delay;
        SPort;
        MoveTo(32, 24);
        LineTo(35, 24);
        LineTo(35, 42);
        MoveTo(17, 24);
        LineTo(14, 24);
        LineTo(14, 42);
    end;

    procedure ECap;
    begin
        Delay;
        EPort;
        MoveTo(24, 17);
        LineTo(24, 14);
        LineTo(42, 14);
        MoveTo(24, 32);
        LineTo(24, 35);
        LineTo(42, 35);
    end;

    procedure WCap;
    begin
        Delay;
        WPort;
        MoveTo(24, 17);
        LineTo(24, 14);
        LineTo(8, 14);
        MoveTo(24, 32);
        LineTo(24, 35);
        LineTo(8, 35);
    end;

begin
    InitGraf(@thePort);
    InitFonts;
    FlushEvents(everyEvent, 0);
    InitWindows;
```

```
InitMenus;
TEInit;
InitDialogs(nil);
InitCursor;
SetRect(r1, 0, 0, 50, 50);
ref := OpenResFile('Thesis.res');
writeln(ref);

pic := OpenPicture(r1);
PenPat(white);
PaintRect(r1);
PenPat(black);
ClosePicture;
AddResource(Handle(pic), 'PICT', 128, 'Picture 0');

pic := OpenPicture(r1);
SetRect(r2, 22, 9, 33, 20);
FrameArc(r2, 0, 180);
MoveTo(27, 9);
LineTo(20, 9);
LineTo(20, 19);
LineTo(27, 19);
MoveTo(19, 14);
LineTo(8, 14);
SetRect(r2, 20, 30, 31, 41);
FrameArc(r2, 0, -180);
MoveTo(25, 30);
LineTo(32, 30);
LineTo(32, 40);
LineTo(25, 40);
MoveTo(19, 35);
LineTo(8, 35);
WPort;
ClosePicture;
AddResource(Handle(pic), 'PICT', 129, 'Picture 1');

pic := OpenPicture(r1);
SetRect(r2, 19, 9, 30, 20);
FrameArc(r2, 0, 180);
MoveTo(24, 9);
LineTo(17, 9);
LineTo(17, 19);
LineTo(24, 19);
MoveTo(30, 14);
LineTo(42, 14);
SetRect(r2, 17, 30, 28, 41);
FrameArc(r2, 0, -180);
MoveTo(22, 30);
LineTo(29, 30);
LineTo(29, 40);
LineTo(22, 40);
MoveTo(30, 35);
LineTo(42, 35);
```

```
EPort;
ClosePicture;
AddResource(Handle(pic), 'PICT', 130, 'Picture 2');

pic := OpenPicture(r1);
SetRect(r2, 9, 20, 20, 31);
FrameArc(r2, 0, -90);
FrameArc(r2, 0, 90);
MoveTo(9, 25);
LineTo(9, 32);
LineTo(19, 32);
LineTo(19, 25);
MoveTo(14, 19);
LineTo(14, 8);
SetRect(r2, 30, 22, 41, 33);
FrameArc(r2, 90, 180);
MoveTo(30, 27);
LineTo(30, 20);
LineTo(40, 20);
LineTo(40, 27);
MoveTo(35, 19);
LineTo(35, 8);
NPort;
ClosePicture;
AddResource(Handle(pic), 'PICT', 131, 'Picture 3');

pic := OpenPicture(r1);
SetRect(r2, 9, 17, 20, 28);
FrameArc(r2, 0, -90);
FrameArc(r2, 0, 90);
MoveTo(9, 22);
LineTo(9, 29);
LineTo(19, 29);
LineTo(19, 22);
MoveTo(14, 30);
LineTo(14, 42);
SetRect(r2, 30, 19, 41, 30);
FrameArc(r2, 90, 180);
MoveTo(30, 24);
LineTo(30, 17);
LineTo(40, 17);
LineTo(40, 24);
MoveTo(35, 30);
LineTo(35, 42);
SPort;
ClosePicture;
AddResource(Handle(pic), 'PICT', 132, 'Picture 4');

pic := OpenPicture(r1);
WCap;
ClosePicture;
AddResource(Handle(pic), 'PICT', 133, 'Picture 5');
```

```
pic := OpenPicture(r1);
ECap;
ClosePicture;
AddResource(Handle(pic), 'PICT', 134, 'Picture 6');

pic := OpenPicture(r1);
NCap;
ClosePicture;
AddResource(Handle(pic), 'PICT', 135, 'Picture 7');

pic := OpenPicture(r1);
SCap;
ClosePicture;
AddResource(Handle(pic), 'PICT', 136, 'Picture 8');

pic := OpenPicture(r1);
WCap;
SetRect(r2, 8, 11, 15, 18);
PenPat(white);
PaintRect(r2);
PenPat(black);
FrameOval(r2);
TextFont(symbol);
TextSize(9);
MoveTo(8, 10);
DrawString('-1');
ClosePicture;
AddResource(Handle(pic), 'PICT', 137, 'Picture  9');

pic := OpenPicture(r1);
ECap;
SetRect(r2, 35, 32, 42, 39);
PenPat(white);
PaintRect(r2);
PenPat(black);
FrameOval(r2);
TextFont(symbol);
TextSize(9);
MoveTo(35, 45);
DrawString('-1');
ClosePicture;
AddResource(Handle(pic), 'PICT', 138, 'Picture  10');

pic := OpenPicture(r1);
NCap;
SetRect(r2, 32, 8, 39, 15);
PenPat(white);
PaintRect(r2);
PenPat(black);
FrameOval(r2);
TextFont(symbol);
TextSize(9);
MoveTo(40, 14);
```

```
DrawString('-1');
ClosePicture;
AddResource(Handle(pic), 'PICT', 139, 'Picture  11');

pic := OpenPicture(r1);
SCap;
SetRect(r2, 11, 35, 18, 42);
PenPat(white);
PaintRect(r2);
PenPat(black);
FrameOval(r2);
TextFont(symbol);
TextSize(9);
MoveTo(1, 41);
DrawString('-1');
ClosePicture;
AddResource(Handle(pic), 'PICT', 140, 'Picture  12');

pic := OpenPicture(r1);
Box;
Parallel;
WGamma('g1');
WPort;
EPort;
ClosePicture;
AddResource(Handle(pic), 'PICT', 141, 'Picture  13');

pic := OpenPicture(r1);
Box;
Parallel;
SGamma('g1');
NPort;
SPort;
ClosePicture;
AddResource(Handle(pic), 'PICT', 142, 'Picture  14');

pic := OpenPicture(r1);
Box;
Parallel;
WRFree;
WPort;
EPort;
ClosePicture;
AddResource(Handle(pic), 'PICT', 143, 'Picture  15');

pic := OpenPicture(r1);
Box;
Parallel;
SRFree;
NPort;
SPort;
ClosePicture;
AddResource(Handle(pic), 'PICT', 144, 'Picture  16');
```

```
pic := OpenPicture(r1);
Box;
Parallel;
ERFree;
WPort;
EPort;
ClosePicture;
AddResource(Handle(pic), 'PICT', 145, 'Picture 17');

pic := OpenPicture(r1);
Box;
Parallel;
NRFree;
NPort;
SPort;
ClosePicture;
AddResource(Handle(pic), 'PICT', 146, 'Picture 18');

pic := OpenPicture(r1);
Box;
Serial;
WGamma('g1');
WPort;
EPort;
ClosePicture;
AddResource(Handle(pic), 'PICT', 147, 'Picture 19');

pic := OpenPicture(r1);
Box;
Serial;
SGamma('g1');
NPort;
SPort;
ClosePicture;
AddResource(Handle(pic), 'PICT', 148, 'Picture 20');

pic := OpenPicture(r1);
Box;
Serial;
WRFree;
WPort;
EPort;
ClosePicture;
AddResource(Handle(pic), 'PICT', 149, 'Picture 21');

pic := OpenPicture(r1);
Box;
Serial;
SRFree;
NPort;
SPort;
ClosePicture;
```

```
AddResource(Handle(pic), 'PICT', 150, 'Picture 22');

pic := OpenPicture(r1);
Box;
Serial;
ERFree;
WPort;
EPort;
ClosePicture;
AddResource(Handle(pic), 'PICT', 151, 'Picture 23');

pic := OpenPicture(r1);
Box;
Serial;
NRFree;
NPort;
SPort;
ClosePicture;
AddResource(Handle(pic), 'PICT', 152, 'Picture 24');

pic := OpenPicture(r1);
Box;
Parallel;
WGamma('g1');
EGamma('g2');
WPort;
EPort;
NPort;
ClosePicture;
AddResource(Handle(pic), 'PICT', 153, 'Picture 25');

pic := OpenPicture(r1);
Box;
Parallel;
WGamma('g1');
EGamma('g2');
WPort;
EPort;
SPort;
ClosePicture;
AddResource(Handle(pic), 'PICT', 154, 'Picture 26');

pic := OpenPicture(r1);
Box;
Parallel;
SGamma('g1');
NGamma('g2');
NPort;
SPort;
EPort;
ClosePicture;
AddResource(Handle(pic), 'PICT', 155, 'Picture 27');
```

```
pic := OpenPicture(r1);
Box;
Parallel;
SGamma('g1');
NGamma('g2');
NPort;
SPort;
WPort;
ClosePicture;
AddResource(Handle(pic), 'PICT', 156, 'Picture  28');

pic := OpenPicture(r1);
Box;
Parallel;
NGamma('g1');
ERFree;
NPort;
EPort;
WPort;
ClosePicture;
AddResource(Handle(pic), 'PICT', 157, 'Picture  29');

pic := OpenPicture(r1);
Box;
Parallel;
SGamma('g1');
ERFree;
SPort;
EPort;
WPort;
ClosePicture;
AddResource(Handle(pic), 'PICT', 158, 'Picture  30');

pic := OpenPicture(r1);
Box;
Parallel;
EGamma('g1');
NRFree;
SPort;
EPort;
NPort;
ClosePicture;
AddResource(Handle(pic), 'PICT', 159, 'Picture  31');

pic := OpenPicture(r1);
Box;
Parallel;
WGamma('g1');
NRFree;
SPort;
WPort;
NPort;
ClosePicture;
```

```
AddResource(Handle(pic), 'PICT', 160, 'Picture  32');

pic := OpenPicture(r1);
Box;
Parallel;
NGamma('g1');
WRFree;
EPort;
WPort;
NPort;
ClosePicture;
AddResource(Handle(pic), 'PICT', 161, 'Picture  33');

pic := OpenPicture(r1);
Box;
Parallel;
SGamma('g1');
WRFree;
EPort;
WPort;
SPort;
ClosePicture;
AddResource(Handle(pic), 'PICT', 162, 'Picture  34');

pic := OpenPicture(r1);
Box;
Parallel;
EGamma('g1');
SRFree;
EPort;
NPort;
SPort;
ClosePicture;
AddResource(Handle(pic), 'PICT', 163, 'Picture  35');

pic := OpenPicture(r1);
Box;
Parallel;
WGamma('g1');
SRFree;
WPort;
NPort;
SPort;
ClosePicture;
AddResource(Handle(pic), 'PICT', 164, 'Picture  36');

pic := OpenPicture(r1);
Box;
Parallel;
WGamma('g1');
NRFree;
WPort;
NPort;
```

```
EPort;
ClosePicture;
AddResource(Handle(pic), 'PICT', 165, 'Picture  37');

pic := OpenPicture(r1);
Box;
Parallel;
WGamma('g1');
SRFree;
WPort;
SPort;
EPort;
ClosePicture;
AddResource(Handle(pic), 'PICT', 166, 'Picture  38');

pic := OpenPicture(r1);
Box;
Parallel;
NGamma('g1');
ERFree;
NPort;
SPort;
EPort;
ClosePicture;
AddResource(Handle(pic), 'PICT', 167, 'Picture  39');

pic := OpenPicture(r1);
Box;
Parallel;
NGamma('g1');
WRFree;
NPort;
SPort;
WPort;
ClosePicture;
AddResource(Handle(pic), 'PICT', 168, 'Picture  40');

pic := OpenPicture(r1);
Box;
Serial;
WGamma('g1');
EGamma('g2');
WPort;
EPort;
NPort;
ClosePicture;
AddResource(Handle(pic), 'PICT', 169, 'Picture  41');

pic := OpenPicture(r1);
Box;
Serial;
WGamma('g1');
EGamma('g2');
```

```
WPort;
EPort;
SPort;
ClosePicture;
AddResource(Handle(pic), 'PICT', 170, 'Picture  42');

pic := OpenPicture(r1);
Box;
Serial;
SGamma('g1');
NGamma('g2');
NPort;
SPort;
EPort;
ClosePicture;
AddResource(Handle(pic), 'PICT', 171, 'Picture  43');

pic := OpenPicture(r1);
Box;
Serial;
SGamma('g1');
NGamma('g2');
NPort;
SPort;
WPort;
ClosePicture;
AddResource(Handle(pic), 'PICT', 172, 'Picture  44');

pic := OpenPicture(r1);
Box;
Serial;
NGamma('g1');
ERFree;
NPort;
EPort;
WPort;
ClosePicture;
AddResource(Handle(pic), 'PICT', 173, 'Picture  45');

pic := OpenPicture(r1);
Box;
Serial;
SGamma('g1');
ERFree;
SPort;
EPort;
WPort;
ClosePicture;
AddResource(Handle(pic), 'PICT', 174, 'Picture  46');

pic := OpenPicture(r1);
Box;
Serial;
```

```
EGamma('g1');
NRFree;
SPort;
EPort;
NPort;
ClosePicture;
AddResource(Handle(pic), 'PICT', 175, 'Picture  47');

pic := OpenPicture(r1);
Box;
Serial;
WGamma('g1');
NRFree;
SPort;
WPort;
NPort;
ClosePicture;
AddResource(Handle(pic), 'PICT', 176, 'Picture  48');

pic := OpenPicture(r1);
Box;
Serial;
NGamma('g1');
WRFree;
EPort;
WPort;
NPort;
ClosePicture;
AddResource(Handle(pic), 'PICT', 177, 'Picture  49');

pic := OpenPicture(r1);
Box;
Serial;
SGamma('g1');
WRFree;
EPort;
WPort;
SPort;
ClosePicture;
AddResource(Handle(pic), 'PICT', 178, 'Picture  50');

pic := OpenPicture(r1);
Box;
Serial;
EGamma('g1');
SRFree;
EPort;
NPort;
SPort;
ClosePicture;
AddResource(Handle(pic), 'PICT', 179, 'Picture  51');

pic := OpenPicture(r1);
```

```
Box;
Serial;
WGamma('g1');
SRFree;
WPort;
NPort;
SPort;
ClosePicture;
AddResource(Handle(pic), 'PICT', 180, 'Picture  52');

pic := OpenPicture(r1);
Box;
Serial;
WGamma('g1');
NRFree;
WPort;
NPort;
EPort;
ClosePicture;
AddResource(Handle(pic), 'PICT', 181, 'Picture  53');

pic := OpenPicture(r1);
Box;
Serial;
WGamma('g1');
SRFree;
WPort;
SPort;
EPort;
ClosePicture;
AddResource(Handle(pic), 'PICT', 182, 'Picture  54');

pic := OpenPicture(r1);
Box;
Serial;
NGamma('g1');
ERFree;
NPort;
SPort;
EPort;
ClosePicture;
AddResource(Handle(pic), 'PICT', 183, 'Picture  55');

pic := OpenPicture(r1);
Box;
Serial;
NGamma('g1');
WRFree;
NPort;
SPort;
WPort;
ClosePicture;
AddResource(Handle(pic), 'PICT', 184, 'Picture  56');
```

```
pic := OpenPicture(r1);
Box;
Parallel;
NPort;
SPort;
WPort;
EPort;
WGamma('g1');
NGamma('g2');
EGamma('g3');
ClosePicture;
AddResource(Handle(pic), 'PICT', 185, 'Picture  57');

pic := OpenPicture(r1);
Box;
Parallel;
NPort;
SPort;
WPort;
EPort;
WGamma('g1');
EGamma('g2');
SRFree;
ClosePicture;
AddResource(Handle(pic), 'PICT', 186, 'Picture  58');

pic := OpenPicture(r1);
Box;
Parallel;
NPort;
SPort;
WPort;
EPort;
WGamma('g1');
EGamma('g2');
NRFree;
ClosePicture;
AddResource(Handle(pic), 'PICT', 187, 'Picture  59');

pic := OpenPicture(r1);
Box;
Parallel;
NPort;
SPort;
WPort;
EPort;
NGamma('g1');
SGamma('g2');
ERFree;
ClosePicture;
AddResource(Handle(pic), 'PICT', 188, 'Picture  60');
```

```
pic := OpenPicture(r1);
Box;
Parallel;
NPort;
SPort;
WPort;
EPort;
NGamma('g1');
SGamma('g2');
WRFree;
ClosePicture;
AddResource(Handle(pic), 'PICT', 189, 'Picture  61');

pic := OpenPicture(r1);
Box;
Serial;
NPort;
SPort;
WPort;
EPort;
WGamma('g1');
NGamma('g2');
EGamma('g3');
ClosePicture;
AddResource(Handle(pic), 'PICT', 190, 'Picture  62');

pic := OpenPicture(r1);
Box;
Serial;
NPort;
SPort;
WPort;
EPort;
WGamma('g1');
EGamma('g2');
SRFree;
ClosePicture;
AddResource(Handle(pic), 'PICT', 191, 'Picture  63');

pic := OpenPicture(r1);
Box;
Serial;
NPort;
SPort;
WPort;
EPort;
WGamma('g1');
EGamma('g2');
NRFree;
ClosePicture;
AddResource(Handle(pic), 'PICT', 192, 'Picture  64');

pic := OpenPicture(r1);
```

```
    Box;
    Serial;
    NPort;
    SPort;
    WPort;
    EPort;
    NGamma('g1');
    SGamma('g2');
    ERFree;
    ClosePicture;
    AddResource(Handle(pic), 'PICT', 193, 'Picture  65');

    pic := OpenPicture(r1);
    Box;
    Serial;
    NPort;
    SPort;
    WPort;
    EPort;
    NGamma('g1');
    SGamma('g2');
    WRFree;
    ClosePicture;
    AddResource(Handle(pic), 'PICT', 194, 'Picture  66');

    CloseResFile(ref);
end.
```

```
program PictRefs2;

  var
    pic: PicHandle;
    r1, r2: Rect;
    ref: integer;

  procedure Box;
  begin
    PenSize(2, 2);
    MoveTo(8, 8);
    LineTo(40, 8);
    LineTo(40, 40);
    LineTo(8, 40);
    LineTo(8, 8);
    PenSize(1, 1);
  end;

  procedure NPort;
  begin
    MoveTo(14, 0);
    LineTo(14, 7);
    MoveTo(35, 0);
    LineTo(35, 7);
    MoveTo(12, 2);
    LineTo(14, 0);
    LineTo(16, 2);
  end;

  procedure SPort;
  begin
    MoveTo(14, 42);
    LineTo(14, 49);
    MoveTo(35, 42);
    LineTo(35, 49);
    MoveTo(33, 47);
    LineTo(35, 49);
    LineTo(37, 47);
  end;

  procedure EPort;
  begin
    MoveTo(42, 14);
    LineTo(49, 14);
    MoveTo(42, 35);
    LineTo(49, 35);
    MoveTo(47, 12);
    LineTo(49, 14);
    LineTo(47, 16);
  end;

  procedure WPort;
  begin
```

```
    MoveTo(0, 14);
    LineTo(7, 14);
    MoveTo(0, 35);
    LineTo(7, 35);
    MoveTo(2, 33);
    LineTo(0, 35);
    LineTo(2, 37);
  end;

  procedure NRFree;
  begin
    MoveTo(12, 13);
    LineTo(16, 13);
    MoveTo(14, 10);
    LineTo(14, 12);
  end;

  procedure SRFree;
  begin
    MoveTo(33, 36);
    LineTo(37, 36);
    MoveTo(35, 37);
    LineTo(35, 39);
  end;

  procedure ERFree;
  begin
    MoveTo(36, 12);
    LineTo(36, 16);
    MoveTo(37, 14);
    LineTo(39, 14);
  end;

  procedure WRFree;
  begin
    MoveTo(13, 33);
    LineTo(13, 37);
    MoveTo(10, 35);
    LineTo(12, 35);
  end;

  procedure Parallel;
  begin
    MoveTo(22, 20);
    LineTo(22, 29);
    MoveTo(26, 20);
    LineTo(26, 29);
  end;

  procedure Serial;
  begin
    MoveTo(20, 25);
    LineTo(29, 25);
```

```
    MoveTo(24, 24);
    LineTo(25, 24);
    MoveTo(24, 26);
    LineTo(25, 26);
  end;

  procedure NGamma (s: Str255);
  begin
    TextFont(symbol);
    TextSize(9);
    MoveTo(20, 17);
    DrawString(s);
  end;

  procedure SGamma (s: Str255);
  begin
    TextFont(symbol);
    TextSize(9);
    MoveTo(20, 37);
    DrawString(s);
  end;

  procedure WGamma (s: Str255);
  begin
    TextFont(symbol);
    TextSize(9);
    MoveTo(11, 27);
    DrawString(s);
  end;

  procedure EGamma (s: Str255);
  begin
    TextFont(symbol);
    TextSize(9);
    MoveTo(30, 27);
    DrawString(s);
  end;

  procedure Delay;
  begin
    PenSize(2, 2);
    MoveTo(18, 18);
    LineTo(30, 18);
    LineTo(30, 30);
    LineTo(18, 30);
    LineTo(18, 18);
    PenSize(1, 1);
    MoveTo(22, 22);
    LineTo(26, 22);
    MoveTo(24, 23);
    LineTo(24, 27);
  end;
```

```
procedure NCap;
begin
  Delay;
  NPort;
  MoveTo(32, 24);
  LineTo(35, 24);
  LineTo(35, 8);
  MoveTo(17, 24);
  LineTo(14, 24);
  LineTo(14, 8);
end;

procedure SCap;
begin
  Delay;
  SPort;
  MoveTo(32, 24);
  LineTo(35, 24);
  LineTo(35, 42);
  MoveTo(17, 24);
  LineTo(14, 24);
  LineTo(14, 42);
end;

procedure ECap;
begin
  Delay;
  EPort;
  MoveTo(24, 17);
  LineTo(24, 14);
  LineTo(42, 14);
  MoveTo(24, 32);
  LineTo(24, 35);
  LineTo(42, 35);
end;

procedure WCap;
begin
  Delay;
  WPort;
  MoveTo(24, 17);
  LineTo(24, 14);
  LineTo(8, 14);
  MoveTo(24, 32);
  LineTo(24, 35);
  LineTo(8, 35);
end;

begin
  InitGraf(@thePort);
  InitFonts;
  FlushEvents(everyEvent, 0);
  InitWindows;
```

```
InitMenus;
TEInit;
InitDialogs(nil);
InitCursor;
SetRect(r1, 0, 0, 50, 50);
ref := OpenResFile('Thesis.res');

pic := OpenPicture(r1);
MoveTo(19, 14);
LineTo(8, 14);
SetRect(r2, 20, 30, 31, 41);
FrameArc(r2, 0, -180);
MoveTo(25, 30);
LineTo(32, 30);
LineTo(32, 40);
LineTo(25, 40);
MoveTo(19, 35);
LineTo(8, 35);
WPort;
ClosePicture;
AddResource(Handle(pic), 'PICT', 195, 'Picture 67');

pic := OpenPicture(r1);
SetRect(r2, 19, 9, 30, 20);
FrameArc(r2, 0, 180);
MoveTo(24, 9);
LineTo(17, 9);
LineTo(17, 19);
LineTo(24, 19);
MoveTo(30, 14);
LineTo(42, 14);
MoveTo(30, 35);
LineTo(42, 35);
EPort;
ClosePicture;
AddResource(Handle(pic), 'PICT', 196, 'Picture 68');

pic := OpenPicture(r1);
SetRect(r2, 9, 20, 20, 31);
FrameArc(r2, 0, -90);
FrameArc(r2, 0, 90);
MoveTo(9, 25);
LineTo(9, 32);
LineTo(19, 32);
LineTo(19, 25);
MoveTo(14, 19);
LineTo(14, 8);
MoveTo(35, 19);
LineTo(35, 8);
NPort;
ClosePicture;
AddResource(Handle(pic), 'PICT', 197, 'Picture 69');
```

```
pic := OpenPicture(r1);
MoveTo(14, 30);
LineTo(14, 42);
SetRect(r2, 30, 19, 41, 30);
FrameArc(r2, 90, 180);
MoveTo(30, 24);
LineTo(30, 17);
LineTo(40, 17);
LineTo(40, 24);
MoveTo(35, 30);
LineTo(35, 42);
SPort;
ClosePicture;
AddResource(Handle(pic), 'PICT', 198, 'Picture 70');

pic := OpenPicture(r1);
SetRect(r2, 22, 9, 33, 20);
FrameArc(r2, 0, 180);
MoveTo(27, 9);
LineTo(20, 9);
LineTo(20, 19);
LineTo(27, 19);
MoveTo(19, 14);
LineTo(8, 14);
MoveTo(19, 35);
LineTo(8, 35);
WPort;
ClosePicture;
AddResource(Handle(pic), 'PICT', 199, 'Picture 71');

pic := OpenPicture(r1);
MoveTo(30, 14);
LineTo(42, 14);
SetRect(r2, 17, 30, 28, 41);
FrameArc(r2, 0, -180);
MoveTo(22, 30);
LineTo(29, 30);
LineTo(29, 40);
LineTo(22, 40);
MoveTo(30, 35);
LineTo(42, 35);
EPort;
ClosePicture;
AddResource(Handle(pic), 'PICT', 200, 'Picture 72');

pic := OpenPicture(r1);
MoveTo(14, 19);
LineTo(14, 8);
SetRect(r2, 30, 22, 41, 33);
FrameArc(r2, 90, 180);
MoveTo(30, 27);
LineTo(30, 20);
LineTo(40, 20);
```

```
    LineTo(40, 27);
    MoveTo(35, 19);
    LineTo(35, 8);
    NPort;
    ClosePicture;
    AddResource(Handle(pic), 'PICT', 201, 'Picture 73');

    pic := OpenPicture(r1);
    SetRect(r2, 9, 17, 20, 28);
    FrameArc(r2, 0, -90);
    FrameArc(r2, 0, 90);
    MoveTo(9, 22);
    LineTo(9, 29);
    LineTo(19, 29);
    LineTo(19, 22);
    MoveTo(14, 30);
    LineTo(14, 42);
    MoveTo(35, 30);
    LineTo(35, 42);
    SPort;
    ClosePicture;
    AddResource(Handle(pic), 'PICT', 202, 'Picture 74');

    CloseResFile(ref);
end.
```

```
program PictRefs3;

  var
    pic: PicHandle;
    r1, r2: Rect;
    ref: integer;

  procedure Box;
  begin
    PenSize(2, 2);
    MoveTo(8, 8);
    LineTo(40, 8);
    LineTo(40, 40);
    LineTo(8, 40);
    LineTo(8, 8);
    PenSize(1, 1);
  end;

  procedure NPort;
  begin
    MoveTo(14, 0);
    LineTo(14, 7);
    MoveTo(35, 0);
    LineTo(35, 7);
    MoveTo(12, 2);
    LineTo(14, 0);
    LineTo(16, 2);
  end;

  procedure SPort;
  begin
    MoveTo(14, 42);
    LineTo(14, 49);
    MoveTo(35, 42);
    LineTo(35, 49);
    MoveTo(33, 47);
    LineTo(35, 49);
    LineTo(37, 47);
  end;

  procedure EPort;
  begin
    MoveTo(42, 14);
    LineTo(49, 14);
    MoveTo(42, 35);
    LineTo(49, 35);
    MoveTo(47, 12);
    LineTo(49, 14);
    LineTo(47, 16);
  end;

  procedure WPort;
  begin
```

```
    MoveTo(0, 14);
    LineTo(7, 14);
    MoveTo(0, 35);
    LineTo(7, 35);
    MoveTo(2, 33);
    LineTo(0, 35);
    LineTo(2, 37);
 end;

 procedure NRFree;
 begin
    MoveTo(12, 13);
    LineTo(16, 13);
    MoveTo(14, 10);
    LineTo(14, 12);
 end;

 procedure SRFree;
 begin
    MoveTo(33, 36);
    LineTo(37, 36);
    MoveTo(35, 37);
    LineTo(35, 39);
 end;

 procedure ERFree;
 begin
    MoveTo(36, 12);
    LineTo(36, 16);
    MoveTo(37, 14);
    LineTo(39, 14);
 end;

 procedure WRFree;
 begin
    MoveTo(13, 33);
    LineTo(13, 37);
    MoveTo(10, 35);
    LineTo(12, 35);
 end;

 procedure Parallel;
 begin
    MoveTo(22, 20);
    LineTo(22, 29);
    MoveTo(26, 20);
    LineTo(26, 29);
 end;

 procedure Serial;
 begin
    MoveTo(20, 25);
    LineTo(29, 25);
```

```
    MoveTo(24, 24);
    LineTo(25, 24);
    MoveTo(24, 26);
    LineTo(25, 26);
  end;

  procedure NGamma (s: Str255);
  begin
    TextFont(symbol);
    TextSize(9);
    MoveTo(20, 17);
    DrawString(s);
  end;

  procedure SGamma (s: Str255);
  begin
    TextFont(symbol);
    TextSize(9);
    MoveTo(20, 37);
    DrawString(s);
  end;

  procedure WGamma (s: Str255);
  begin
    TextFont(symbol);
    TextSize(9);
    MoveTo(11, 27);
    DrawString(s);
  end;

  procedure EGamma (s: Str255);
  begin
    TextFont(symbol);
    TextSize(9);
    MoveTo(30, 27);
    DrawString(s);
  end;

  procedure Center (s: Str255);
  begin
    TextFont(times);
    TextSize(12);
    MoveTo(18, 29);
    DrawString(s);
  end;

  procedure Delay;
  begin
    PenSize(2, 2);
    MoveTo(18, 18);
    LineTo(30, 18);
    LineTo(30, 30);
    LineTo(18, 30);
```

```
    LineTo(18, 18);
    PenSize(1, 1);
    MoveTo(22, 22);
    LineTo(26, 22);
    MoveTo(24, 23);
    LineTo(24, 27);
end;

procedure NCap;
begin
    Delay;
    NPort;
    MoveTo(32, 24);
    LineTo(35, 24);
    LineTo(35, 8);
    MoveTo(17, 24);
    LineTo(14, 24);
    LineTo(14, 8);
end;

procedure SCap;
begin
    Delay;
    SPort;
    MoveTo(32, 24);
    LineTo(35, 24);
    LineTo(35, 42);
    MoveTo(17, 24);
    LineTo(14, 24);
    LineTo(14, 42);
end;

procedure ECap;
begin
    Delay;
    EPort;
    MoveTo(24, 17);
    LineTo(24, 14);
    LineTo(42, 14);
    MoveTo(24, 32);
    LineTo(24, 35);
    LineTo(42, 35);
end;

procedure WCap;
begin
    Delay;
    WPort;
    MoveTo(24, 17);
    LineTo(24, 14);
    LineTo(8, 14);
    MoveTo(24, 32);
    LineTo(24, 35);
```

```
    LineTo(8, 35);
  end;

begin
  InitGraf(@thePort);
  InitFonts;
  FlushEvents(everyEvent, 0);
  InitWindows;
  InitMenus;
  TEInit;
  InitDialogs(nil);
  InitCursor;
  SetRect(r1, 0, 0, 50, 50);
  ref := OpenResFile('Thesis.res');

  pic := OpenPicture(r1);
  Box;
  EPort;
  WPort;
  WRFree;
  Center('1A');
  ClosePicture;
  AddResource(Handle(pic), 'PICT', 203, 'Picture 75');

  pic := OpenPicture(r1);
  Box;
  NPort;
  SPort;
  SRFree;
  Center('1A');
  ClosePicture;
  AddResource(Handle(pic), 'PICT', 204, 'Picture 76');

  pic := OpenPicture(r1);
  Box;
  EPort;
  WPort;
  ERFree;
  Center('1A');
  ClosePicture;
  AddResource(Handle(pic), 'PICT', 205, 'Picture 77');

  pic := OpenPicture(r1);
  Box;
  NPort;
  SPort;
  NRFree;
  Center('1A');
  ClosePicture;
  AddResource(Handle(pic), 'PICT', 206, 'Picture 78');

  pic := OpenPicture(r1);
  Box;
```

```
EPort;
WPort;
WRFree;
Center('1B');
ClosePicture;
AddResource(Handle(pic), 'PICT', 207, 'Picture 79');

pic := OpenPicture(r1);
Box;
NPort;
SPort;
SRFree;
Center('1B');
ClosePicture;
AddResource(Handle(pic), 'PICT', 208, 'Picture 80');

pic := OpenPicture(r1);
Box;
EPort;
WPort;
ERFree;
Center('1B');
ClosePicture;
AddResource(Handle(pic), 'PICT', 209, 'Picture 81');

pic := OpenPicture(r1);
Box;
NPort;
SPort;
NRFree;
Center('1B');
ClosePicture;
AddResource(Handle(pic), 'PICT', 210, 'Picture 82');

pic := OpenPicture(r1);
Box;
EPort;
WPort;
WRFree;
Center('1C');
ClosePicture;
AddResource(Handle(pic), 'PICT', 211, 'Picture 83');

pic := OpenPicture(r1);
Box;
NPort;
SPort;
SRFree;
Center('1C');
ClosePicture;
AddResource(Handle(pic), 'PICT', 212, 'Picture 84');

pic := OpenPicture(r1);
```

```
Box;
EPort;
WPort;
ERFree;
Center('1C');
ClosePicture;
AddResource(Handle(pic), 'PICT', 213, 'Picture 85');

pic := OpenPicture(r1);
Box;
NPort;
SPort;
NRFree;
Center('1C');
ClosePicture;
AddResource(Handle(pic), 'PICT', 214, 'Picture 86');

pic := OpenPicture(r1);
Box;
EPort;
WPort;
WRFree;
Center('1D');
ClosePicture;
AddResource(Handle(pic), 'PICT', 215, 'Picture 87');

pic := OpenPicture(r1);
Box;
NPort;
SPort;
SRFree;
Center('1D');
ClosePicture;
AddResource(Handle(pic), 'PICT', 216, 'Picture 88');

pic := OpenPicture(r1);
Box;
EPort;
WPort;
ERFree;
Center('1D');
ClosePicture;
AddResource(Handle(pic), 'PICT', 217, 'Picture 89');

pic := OpenPicture(r1);
Box;
NPort;
SPort;
NRFree;
Center('1D');
ClosePicture;
AddResource(Handle(pic), 'PICT', 218, 'Picture 90');
```

```
pic := OpenPicture(r1);
Box;
EPort;
WPort;
WRFree;
Center('1E');
ClosePicture;
AddResource(Handle(pic), 'PICT', 219, 'Picture 91');

pic := OpenPicture(r1);
Box;
NPort;
SPort;
SRFree;
Center('1E');
ClosePicture;
AddResource(Handle(pic), 'PICT', 220, 'Picture 92');

pic := OpenPicture(r1);
Box;
EPort;
WPort;
ERFree;
Center('1E');
ClosePicture;
AddResource(Handle(pic), 'PICT', 221, 'Picture 93');

pic := OpenPicture(r1);
Box;
NPort;
SPort;
NRFree;
Center('1E');
ClosePicture;
AddResource(Handle(pic), 'PICT', 222, 'Picture 94');

pic := OpenPicture(r1);
Box;
EPort;
WPort;
WRFree;
Center('2A');
ClosePicture;
AddResource(Handle(pic), 'PICT', 223, 'Picture 95');

pic := OpenPicture(r1);
Box;
NPort;
SPort;
SRFree;
Center('2A');
ClosePicture;
AddResource(Handle(pic), 'PICT', 224, 'Picture 96');
```

```
pic := OpenPicture(r1);
Box;
EPort;
WPort;
ERFree;
Center('2A');
ClosePicture;
AddResource(Handle(pic), 'PICT', 225, 'Picture 97');

pic := OpenPicture(r1);
Box;
NPort;
SPort;
NRFree;
Center('2A');
ClosePicture;
AddResource(Handle(pic), 'PICT', 226, 'Picture 98');

pic := OpenPicture(r1);
Box;
EPort;
WPort;
WRFree;
Center('2B');
ClosePicture;
AddResource(Handle(pic), 'PICT', 227, 'Picture 99');

pic := OpenPicture(r1);
Box;
NPort;
SPort;
SRFree;
Center('2B');
ClosePicture;
AddResource(Handle(pic), 'PICT', 228, 'Picture 100');

pic := OpenPicture(r1);
Box;
EPort;
WPort;
ERFree;
Center('2B');
ClosePicture;
AddResource(Handle(pic), 'PICT', 229, 'Picture 101');

pic := OpenPicture(r1);
Box;
NPort;
SPort;
NRFree;
Center('2B');
ClosePicture;
```

```
AddResource(Handle(pic), 'PICT', 230, 'Picture 102');

pic := OpenPicture(r1);
Box;
EPort;
WPort;
WRFree;
Center('2C');
ClosePicture;
AddResource(Handle(pic), 'PICT', 231, 'Picture 103');

pic := OpenPicture(r1);
Box;
NPort;
SPort;
SRFree;
Center('2C');
ClosePicture;
AddResource(Handle(pic), 'PICT', 232, 'Picture 104');

pic := OpenPicture(r1);
Box;
EPort;
WPort;
ERFree;
Center('2C');
ClosePicture;
AddResource(Handle(pic), 'PICT', 233, 'Picture 105');

pic := OpenPicture(r1);
Box;
NPort;
SPort;
NRFree;
Center('2C');
ClosePicture;
AddResource(Handle(pic), 'PICT', 234, 'Picture 106');

pic := OpenPicture(r1);
Box;
EPort;
WPort;
WRFree;
Center('2D');
ClosePicture;
AddResource(Handle(pic), 'PICT', 235, 'Picture 107');

pic := OpenPicture(r1);
Box;
NPort;
SPort;
SRFree;
Center('2D');
```

```
ClosePicture;
AddResource(Handle(pic), 'PICT', 236, 'Picture 108');

pic := OpenPicture(r1);
Box;
EPort;
WPort;
ERFree;
Center('2D');
ClosePicture;
AddResource(Handle(pic), 'PICT', 237, 'Picture 109');

pic := OpenPicture(r1);
Box;
NPort;
SPort;
NRFree;
Center('2D');
ClosePicture;
AddResource(Handle(pic), 'PICT', 238, 'Picture 110');

pic := OpenPicture(r1);
Box;
EPort;
WPort;
WRFree;
Center('2E');
ClosePicture;
AddResource(Handle(pic), 'PICT', 239, 'Picture 111');

pic := OpenPicture(r1);
Box;
NPort;
SPort;
SRFree;
Center('2E');
ClosePicture;
AddResource(Handle(pic), 'PICT', 240, 'Picture 112');

pic := OpenPicture(r1);
Box;
EPort;
WPort;
ERFree;
Center('2E');
ClosePicture;
AddResource(Handle(pic), 'PICT', 241, 'Picture 113');

pic := OpenPicture(r1);
Box;
NPort;
SPort;
NRFree;
```

```
  Center('2E');
  ClosePicture;
  AddResource(Handle(pic), 'PICT', 242, 'Picture 114');

  CloseResFile(ref);
end.
```

```
program PictRefs4;

  var
    pic: PicHandle;
    r1, r2: Rect;
    ref: integer;

  procedure Box;
  begin
    PenSize(2, 2);
    MoveTo(8, 8);
    LineTo(40, 8);
    LineTo(40, 40);
    LineTo(8, 40);
    LineTo(8, 8);
    PenSize(1, 1);
  end;

  procedure NPort;
  begin
    MoveTo(14, 0);
    LineTo(14, 7);
    MoveTo(35, 0);
    LineTo(35, 7);
    MoveTo(12, 2);
    LineTo(14, 0);
    LineTo(16, 2);
  end;

  procedure SPort;
  begin
    MoveTo(14, 42);
    LineTo(14, 49);
    MoveTo(35, 42);
    LineTo(35, 49);
    MoveTo(33, 47);
    LineTo(35, 49);
    LineTo(37, 47);
  end;

  procedure EPort;
  begin
    MoveTo(42, 14);
    LineTo(49, 14);
    MoveTo(42, 35);
    LineTo(49, 35);
    MoveTo(47, 12);
    LineTo(49, 14);
    LineTo(47, 16);
  end;

  procedure WPort;
  begin
```

```
    MoveTo(0, 14);
    LineTo(7, 14);
    MoveTo(0, 35);
    LineTo(7, 35);
    MoveTo(2, 33);
    LineTo(0, 35);
    LineTo(2, 37);
  end;

  procedure NRFree;
  begin
    MoveTo(12, 13);
    LineTo(16, 13);
    MoveTo(14, 10);
    LineTo(14, 12);
  end;

  procedure SRFree;
  begin
    MoveTo(33, 36);
    LineTo(37, 36);
    MoveTo(35, 37);
    LineTo(35, 39);
  end;

  procedure ERFree;
  begin
    MoveTo(36, 12);
    LineTo(36, 16);
    MoveTo(37, 14);
    LineTo(39, 14);
  end;

  procedure WRFree;
  begin
    MoveTo(13, 33);
    LineTo(13, 37);
    MoveTo(10, 35);
    LineTo(12, 35);
  end;

  procedure Parallel;
  begin
    MoveTo(22, 20);
    LineTo(22, 29);
    MoveTo(26, 20);
    LineTo(26, 29);
  end;

  procedure Serial;
  begin
    MoveTo(20, 25);
    LineTo(29, 25);
```

```
      MoveTo(24, 24);
      LineTo(25, 24);
      MoveTo(24, 26);
      LineTo(25, 26);
   end;

   procedure NGamma (s: Str255);
   begin
      TextFont(symbol);
      TextSize(9);
      MoveTo(22, 17);
      DrawString(s);
   end;

   procedure SGamma (s: Str255);
   begin
      TextFont(symbol);
      TextSize(9);
      MoveTo(22, 39);
      DrawString(s);
   end;

   procedure WGamma (s: Str255);
   begin
      TextFont(symbol);
      TextSize(9);
      MoveTo(13, 27);
      DrawString(s);
   end;

   procedure EGamma (s: Str255);
   begin
      TextFont(symbol);
      TextSize(9);
      MoveTo(32, 27);
      DrawString(s);
   end;

   procedure Center (s: Str255);
   begin
      TextFont(times);
      TextSize(12);
      MoveTo(18, 29);
      DrawString(s);
   end;

   procedure Delay;
   begin
      PenSize(2, 2);
      MoveTo(18, 18);
      LineTo(30, 18);
      LineTo(30, 30);
      LineTo(18, 30);
```

```
    LineTo(18, 18);
    PenSize(1, 1);
    MoveTo(22, 22);
    LineTo(26, 22);
    MoveTo(24, 23);
    LineTo(24, 27);
  end;

  procedure NCap;
  begin
    Delay;
    NPort;
    MoveTo(32, 24);
    LineTo(35, 24);
    LineTo(35, 8);
    MoveTo(17, 24);
    LineTo(14, 24);
    LineTo(14, 8);
  end;

  procedure SCap;
  begin
    Delay;
    SPort;
    MoveTo(32, 24);
    LineTo(35, 24);
    LineTo(35, 42);
    MoveTo(17, 24);
    LineTo(14, 24);
    LineTo(14, 42);
  end;

  procedure ECap;
  begin
    Delay;
    EPort;
    MoveTo(24, 17);
    LineTo(24, 14);
    LineTo(42, 14);
    MoveTo(24, 32);
    LineTo(24, 35);
    LineTo(42, 35);
  end;

  procedure WCap;
  begin
    Delay;
    WPort;
    MoveTo(24, 17);
    LineTo(24, 14);
    LineTo(8, 14);
    MoveTo(24, 32);
    LineTo(24, 35);
```

```
      LineTo(8, 35);
    end;

  begin
    InitGraf(@thePort);
    InitFonts;
    FlushEvents(everyEvent, 0);
    InitWindows;
    InitMenus;
    TEInit;
    InitDialogs(nil);
    InitCursor;
    SetRect(r1, 0, 0, 50, 50);
    ref := OpenResFile('Thesis.res');

    pic := OpenPicture(r1);
    Box;
    Parallel;
    EPort;
    WPort;
    WGamma('Q');
    ClosePicture;
    AddResource(Handle(pic), 'PICT', 243, 'Picture 115');

    pic := OpenPicture(r1);
    Box;
    Parallel;
    NPort;
    SPort;
    SGamma('Q');
    ClosePicture;
    AddResource(Handle(pic), 'PICT', 244, 'Picture 116');

    pic := OpenPicture(r1);
    Box;
    Parallel;
    EPort;
    WPort;
    EGamma('Q');
    ClosePicture;
    AddResource(Handle(pic), 'PICT', 245, 'Picture 117');

    pic := OpenPicture(r1);
    Box;
    Parallel;
    NPort;
    SPort;
    NGamma('Q');
    ClosePicture;
    AddResource(Handle(pic), 'PICT', 246, 'Picture 118');

    CloseResFile(ref);
end.
```

```
program PictRefs4;

  var
    pic: PicHandle;
    r1, r2: Rect;
    ref: integer;

  procedure Box;
  begin
    PenSize(2, 2);
    MoveTo(8, 8);
    LineTo(40, 8);
    LineTo(40, 40);
    LineTo(8, 40);
    LineTo(8, 8);
    PenSize(1, 1);
  end;

  procedure NPort;
  begin
    MoveTo(14, 0);
    LineTo(14, 7);
    MoveTo(35, 0);
    LineTo(35, 7);
    MoveTo(12, 2);
    LineTo(14, 0);
    LineTo(16, 2);
  end;

  procedure SPort;
  begin
    MoveTo(14, 42);
    LineTo(14, 49);
    MoveTo(35, 42);
    LineTo(35, 49);
    MoveTo(33, 47);
    LineTo(35, 49);
    LineTo(37, 47);
  end;

  procedure EPort;
  begin
    MoveTo(42, 14);
    LineTo(49, 14);
    MoveTo(42, 35);
    LineTo(49, 35);
    MoveTo(47, 12);
    LineTo(49, 14);
    LineTo(47, 16);
  end;

  procedure WPort;
  begin
```

```
    MoveTo(0, 14);
    LineTo(7, 14);
    MoveTo(0, 35);
    LineTo(7, 35);
    MoveTo(2, 33);
    LineTo(0, 35);
    LineTo(2, 37);
  end;

  procedure NRFree;
  begin
    MoveTo(12, 13);
    LineTo(16, 13);
    MoveTo(14, 10);
    LineTo(14, 12);
  end;

  procedure SRFree;
  begin
    MoveTo(33, 36);
    LineTo(37, 36);
    MoveTo(35, 37);
    LineTo(35, 39);
  end;

  procedure ERFree;
  begin
    MoveTo(36, 12);
    LineTo(36, 16);
    MoveTo(37, 14);
    LineTo(39, 14);
  end;

  procedure WRFree;
  begin
    MoveTo(13, 33);
    LineTo(13, 37);
    MoveTo(10, 35);
    LineTo(12, 35);
  end;

  procedure Parallel;
  begin
    MoveTo(22, 20);
    LineTo(22, 29);
    MoveTo(26, 20);
    LineTo(26, 29);
  end;

  procedure Serial;
  begin
    MoveTo(20, 25);
    LineTo(29, 25);
```

```
    MoveTo(24, 24);
    LineTo(25, 24);
    MoveTo(24, 26);
    LineTo(25, 26);
  end;

  procedure NGamma (s: Str255);
  begin
    TextFont(symbol);
    TextSize(9);
    MoveTo(22, 17);
    DrawString(s);
  end;

  procedure SGamma (s: Str255);
  begin
    TextFont(symbol);
    TextSize(9);
    MoveTo(22, 39);
    DrawString(s);
  end;

  procedure WGamma (s: Str255);
  begin
    TextFont(symbol);
    TextSize(9);
    MoveTo(13, 27);
    DrawString(s);
  end;

  procedure EGamma (s: Str255);
  begin
    TextFont(symbol);
    TextSize(9);
    MoveTo(32, 27);
    DrawString(s);
  end;

  procedure Center (s: Str255);
  begin
    TextFont(times);
    TextSize(12);
    MoveTo(18, 29);
    DrawString(s);
  end;

  procedure Delay;
  begin
    PenSize(2, 2);
    MoveTo(18, 18);
    LineTo(30, 18);
    LineTo(30, 30);
    LineTo(18, 30);
```

```
    LineTo(18, 18);
    PenSize(1, 1);
    MoveTo(22, 22);
    LineTo(26, 22);
    MoveTo(24, 23);
    LineTo(24, 27);
  end;

  procedure NCap;
  begin
    Delay;
    NPort;
    MoveTo(32, 24);
    LineTo(35, 24);
    LineTo(35, 8);
    MoveTo(17, 24);
    LineTo(14, 24);
    LineTo(14, 8);
  end;

  procedure SCap;
  begin
    Delay;
    SPort;
    MoveTo(32, 24);
    LineTo(35, 24);
    LineTo(35, 42);
    MoveTo(17, 24);
    LineTo(14, 24);
    LineTo(14, 42);
  end;

  procedure ECap;
  begin
    Delay;
    EPort;
    MoveTo(24, 17);
    LineTo(24, 14);
    LineTo(42, 14);
    MoveTo(24, 32);
    LineTo(24, 35);
    LineTo(42, 35);
  end;

  procedure WCap;
  begin
    Delay;
    WPort;
    MoveTo(24, 17);
    LineTo(24, 14);
    LineTo(8, 14);
    MoveTo(24, 32);
    LineTo(24, 35);
```

```
    LineTo(8, 35);
  end;

begin
  InitGraf(@thePort);
  InitFonts;
  FlushEvents(everyEvent, 0);
  InitWindows;
  InitMenus;
  TEInit;
  InitDialogs(nil);
  InitCursor;
  SetRect(r1, 0, 0, 50, 50);
  ref := OpenResFile('Thesis.res_B');

  pic := OpenPicture(r1);
  MoveTo(0, 14);
  LineTo(14, 14);
  LineTo(14, 0);
  MoveTo(0, 35);
  LineTo(35, 35);
  LineTo(35, 0);
  WPort;
  NPort;
  ClosePicture;
  AddResource(Handle(pic), 'PICT', 247, 'Picture 119');

  pic := OpenPicture(r1);
  MoveTo(0, 14);
  LineTo(35, 14);
  LineTo(35, 49);
  MoveTo(0, 35);
  LineTo(14, 35);
  LineTo(14, 49);
  WPort;
  SPort;
  ClosePicture;
  AddResource(Handle(pic), 'PICT', 248, 'Picture 120');

  pic := OpenPicture(r1);
  MoveTo(14, 0);
  LineTo(14, 35);
  LineTo(49, 35);
  MoveTo(35, 0);
  LineTo(35, 14);
  LineTo(49, 14);
  EPort;
  NPort;
  ClosePicture;
  AddResource(Handle(pic), 'PICT', 249, 'Picture 121');

  pic := OpenPicture(r1);
  MoveTo(14, 49);
```

```
LineTo(14, 14);
LineTo(49, 14);
MoveTo(35, 49);
LineTo(35, 35);
LineTo(49, 35);
EPort;
SPort;
ClosePicture;
AddResource(Handle(pic), 'PICT', 250, 'Picture 122');

pic := OpenPicture(r1);
MoveTo(0, 14);
LineTo(14, 14);
LineTo(14, 0);
MoveTo(35, 0);
LineTo(35, 14);
LineTo(49, 14);
MoveTo(0, 35);
LineTo(49, 35);
WPort;
EPort;
NPort;
ClosePicture;
AddResource(Handle(pic), 'PICT', 251, 'Picture 123');

pic := OpenPicture(r1);
MoveTo(0, 35);
LineTo(14, 35);
LineTo(14, 49);
MoveTo(35, 49);
LineTo(35, 35);
LineTo(49, 35);
MoveTo(0, 14);
LineTo(49, 14);
SPort;
EPort;
WPort;
ClosePicture;
AddResource(Handle(pic), 'PICT', 252, 'Picture 124');

pic := OpenPicture(r1);
MoveTo(0, 14);
LineTo(14, 14);
LineTo(14, 0);
MoveTo(0, 35);
LineTo(14, 35);
LineTo(14, 49);
MoveTo(35, 0);
LineTo(35, 49);
WPort;
SPort;
NPort;
ClosePicture;
```

```
    AddResource(Handle(pic), 'PICT', 253, 'Picture 125');

    pic := OpenPicture(r1);
    MoveTo(35, 0);
    LineTo(35, 14);
    LineTo(49, 14);
    MoveTo(35, 49);
    LineTo(35, 35);
    LineTo(49, 35);
    MoveTo(14, 0);
    LineTo(14, 49);
    EPort;
    SPort;
    NPort;
    ClosePicture;
    AddResource(Handle(pic), 'PICT', 254, 'Picture 126');

    pic := OpenPicture(r1);
    MoveTo(0, 14);
    LineTo(14, 14);
    LineTo(14, 0);
    MoveTo(0, 35);
    LineTo(14, 35);
    LineTo(14, 49);
    MoveTo(35, 0);
    LineTo(35, 14);
    LineTo(49, 14);
    MoveTo(35, 49);
    LineTo(35, 35);
    LineTo(49, 35);
    EPort;
    WPOrt;
    SPort;
    NPort;
    ClosePicture;
    AddResource(Handle(pic), 'PICT', 255, 'Picture 127');

    CloseResFile(ref);
end.
```

```
program PictRefs6;

  var
    pic: PicHandle;
    r1, r2: Rect;
    ref: integer;

  procedure Box;
  begin
    PenSize(2, 2);
    MoveTo(8, 8);
    LineTo(40, 8);
    LineTo(40, 40);
    LineTo(8, 40);
    LineTo(8, 8);
    PenSize(1, 1);
  end;

  procedure NPort;
  begin
    MoveTo(14, 0);
    LineTo(14, 7);
    MoveTo(35, 0);
    LineTo(35, 7);
    MoveTo(12, 2);
    LineTo(14, 0);
    LineTo(16, 2);
  end;

  procedure SPort;
  begin
    MoveTo(14, 42);
    LineTo(14, 49);
    MoveTo(35, 42);
    LineTo(35, 49);
    MoveTo(33, 47);
    LineTo(35, 49);
    LineTo(37, 47);
  end;

  procedure EPort;
  begin
    MoveTo(42, 14);
    LineTo(49, 14);
    MoveTo(42, 35);
    LineTo(49, 35);
    MoveTo(47, 12);
    LineTo(49, 14);
    LineTo(47, 16);
  end;

  procedure WPort;
  begin
```

```
    MoveTo(0, 14);
    LineTo(7, 14);
    MoveTo(0, 35);
    LineTo(7, 35);
    MoveTo(2, 33);
    LineTo(0, 35);
    LineTo(2, 37);
end;

procedure NRFree;
begin
    MoveTo(12, 13);
    LineTo(16, 13);
    MoveTo(14, 10);
    LineTo(14, 12);
end;

procedure SRFree;
begin
    MoveTo(33, 36);
    LineTo(37, 36);
    MoveTo(35, 37);
    LineTo(35, 39);
end;

procedure ERFree;
begin
    MoveTo(36, 12);
    LineTo(36, 16);
    MoveTo(37, 14);
    LineTo(39, 14);
end;

procedure WRFree;
begin
    MoveTo(13, 33);
    LineTo(13, 37);
    MoveTo(10, 35);
    LineTo(12, 35);
end;

procedure Parallel;
begin
    MoveTo(22, 20);
    LineTo(22, 29);
    MoveTo(26, 20);
    LineTo(26, 29);
end;

procedure Serial;
begin
    MoveTo(20, 25);
    LineTo(29, 25);
```

```
    MoveTo(24, 24);
    LineTo(25, 24);
    MoveTo(24, 26);
    LineTo(25, 26);
  end;

  procedure NGamma (s: Str255);
  begin
    TextFont(symbol);
    TextSize(9);
    MoveTo(20, 17);
    DrawString(s);
  end;

  procedure SGamma (s: Str255);
  begin
    TextFont(symbol);
    TextSize(9);
    MoveTo(20, 37);
    DrawString(s);
  end;

  procedure WGamma (s: Str255);
  begin
    TextFont(symbol);
    TextSize(9);
    MoveTo(11, 27);
    DrawString(s);
  end;

  procedure EGamma (s: Str255);
  begin
    TextFont(symbol);
    TextSize(9);
    MoveTo(30, 27);
    DrawString(s);
  end;

  procedure Center (s: Str255);
  begin
    TextFont(times);
    TextSize(12);
    MoveTo(15, 29);
    DrawString(s);
  end;

  procedure Delay;
  begin
    PenSize(2, 2);
    MoveTo(18, 18);
    LineTo(30, 18);
    LineTo(30, 30);
    LineTo(18, 30);
```

```
     LineTo(18, 18);
     PenSize(1, 1);
     MoveTo(22, 22);
     LineTo(26, 22);
     MoveTo(24, 23);
     LineTo(24, 27);
  end;

  procedure NCap;
  begin
     Delay;
     NPort;
     MoveTo(32, 24);
     LineTo(35, 24);
     LineTo(35, 8);
     MoveTo(17, 24);
     LineTo(14, 24);
     LineTo(14, 8);
  end;

  procedure SCap;
  begin
     Delay;
     SPort;
     MoveTo(32, 24);
     LineTo(35, 24);
     LineTo(35, 42);
     MoveTo(17, 24);
     LineTo(14, 24);
     LineTo(14, 42);
  end;

  procedure ECap;
  begin
     Delay;
     EPort;
     MoveTo(24, 17);
     LineTo(24, 14);
     LineTo(42, 14);
     MoveTo(24, 32);
     LineTo(24, 35);
     LineTo(42, 35);
  end;

  procedure WCap;
  begin
     Delay;
     WPort;
     MoveTo(24, 17);
     LineTo(24, 14);
     LineTo(8, 14);
     MoveTo(24, 32);
     LineTo(24, 35);
```

```
    LineTo(8, 35);
  end;

begin
  InitGraf(@thePort);
  InitFonts;
  FlushEvents(everyEvent, 0);
  InitWindows;
  InitMenus;
  TEInit;
  InitDialogs(nil);
  InitCursor;
  SetRect(r1, 0, 0, 50, 50);
  ref := OpenResFile('Thesis.res_B');

  pic := OpenPicture(r1);
  Box;
  EPort;
  WPort;
  Center('3QL');
  ClosePicture;
  AddResource(Handle(pic), 'PICT', 256, 'Picture 128');

  pic := OpenPicture(r1);
  Box;
  NPort;
  SPort;
  Center('3QL');
  ClosePicture;
  AddResource(Handle(pic), 'PICT', 257, 'Picture 129');

  CloseResFile(ref);
end.
```

```
program PictRefs7;

  var
    pic: PicHandle;
    r1, r2: Rect;
    ref: integer;

  procedure Box;
  begin
    PenSize(2, 2);
    MoveTo(8, 8);
    LineTo(40, 8);
    LineTo(40, 40);
    LineTo(8, 40);
    LineTo(8, 8);
    PenSize(1, 1);
  end;

  procedure NPort;
  begin
    MoveTo(14, 0);
    LineTo(14, 7);
    MoveTo(35, 0);
    LineTo(35, 7);
    MoveTo(12, 2);
    LineTo(14, 0);
    LineTo(16, 2);
  end;

  procedure SPort;
  begin
    MoveTo(14, 42);
    LineTo(14, 49);
    MoveTo(35, 42);
    LineTo(35, 49);
    MoveTo(33, 47);
    LineTo(35, 49);
    LineTo(37, 47);
  end;

  procedure EPort;
  begin
    MoveTo(42, 14);
    LineTo(49, 14);
    MoveTo(42, 35);
    LineTo(49, 35);
    MoveTo(47, 12);
    LineTo(49, 14);
    LineTo(47, 16);
  end;

  procedure WPort;
  begin
```

```
    MoveTo(0, 14);
    LineTo(7, 14);
    MoveTo(0, 35);
    LineTo(7, 35);
    MoveTo(2, 33);
    LineTo(0, 35);
    LineTo(2, 37);
  end;

  procedure NRFree;
  begin
    MoveTo(12, 13);
    LineTo(16, 13);
    MoveTo(14, 10);
    LineTo(14, 12);
  end;

  procedure SRFree;
  begin
    MoveTo(33, 36);
    LineTo(37, 36);
    MoveTo(35, 37);
    LineTo(35, 39);
  end;

  procedure ERFree;
  begin
    MoveTo(36, 12);
    LineTo(36, 16);
    MoveTo(37, 14);
    LineTo(39, 14);
  end;

  procedure WRFree;
  begin
    MoveTo(13, 33);
    LineTo(13, 37);
    MoveTo(10, 35);
    LineTo(12, 35);
  end;

  procedure Parallel;
  begin
    MoveTo(22, 20);
    LineTo(22, 29);
    MoveTo(26, 20);
    LineTo(26, 29);
  end;

  procedure Serial;
  begin
    MoveTo(20, 25);
    LineTo(29, 25);
```

```
    MoveTo(24, 24);
    LineTo(25, 24);
    MoveTo(24, 26);
    LineTo(25, 26);
  end;

  procedure NGamma (s: Str255);
  begin
    TextFont(symbol);
    TextSize(9);
    MoveTo(20, 17);
    DrawString(s);
  end;

  procedure SGamma (s: Str255);
  begin
    TextFont(symbol);
    TextSize(9);
    MoveTo(20, 37);
    DrawString(s);
  end;

  procedure WGamma (s: Str255);
  begin
    TextFont(symbol);
    TextSize(9);
    MoveTo(11, 27);
    DrawString(s);
  end;

  procedure EGamma (s: Str255);
  begin
    TextFont(symbol);
    TextSize(9);
    MoveTo(30, 27);
    DrawString(s);
  end;

  procedure Center (s: Str255);
  begin
    TextFont(times);
    TextSize(12);
    MoveTo(15, 29);
    DrawString(s);
  end;

  procedure Delay;
  begin
    PenSize(2, 2);
    MoveTo(18, 18);
    LineTo(30, 18);
    LineTo(30, 30);
    LineTo(18, 30);
```

```
      LineTo(18, 18);
      PenSize(1, 1);
      MoveTo(22, 22);
      LineTo(26, 22);
      MoveTo(24, 23);
      LineTo(24, 27);
  end;

  procedure NCap;
  begin
      Delay;
      NPort;
      MoveTo(32, 24);
      LineTo(35, 24);
      LineTo(35, 8);
      MoveTo(17, 24);
      LineTo(14, 24);
      LineTo(14, 8);
  end;

  procedure SCap;
  begin
      Delay;
      SPort;
      MoveTo(32, 24);
      LineTo(35, 24);
      LineTo(35, 42);
      MoveTo(17, 24);
      LineTo(14, 24);
      LineTo(14, 42);
  end;

  procedure ECap;
  begin
      Delay;
      EPort;
      MoveTo(24, 17);
      LineTo(24, 14);
      LineTo(42, 14);
      MoveTo(24, 32);
      LineTo(24, 35);
      LineTo(42, 35);
  end;

  procedure WCap;
  begin
      Delay;
      WPort;
      MoveTo(24, 17);
      LineTo(24, 14);
      LineTo(8, 14);
      MoveTo(24, 32);
      LineTo(24, 35);
```

```
    LineTo(8, 35);
  end;

begin
  InitGraf(@thePort);
  InitFonts;
  FlushEvents(everyEvent, 0);
  InitWindows;
  InitMenus;
  TEInit;
  InitDialogs(nil);
  InitCursor;
  ShowDrawing;
  SetRect(r1, 0, 0, 50, 50);
  ref := OpenResFile('Thesis.res_B');

  pic := OpenPicture(r1);
  EPort;
  WPort;
  MoveTo(0, 14);
  LineTo(49, 14);
  MoveTo(0, 35);
  LineTo(49, 35);
  SetRect(r2, 20, 10, 30, 20);
  PenPat(white);
  PaintRect(r2);
  PenPat(black);
  FrameOval(r2);
  SetRect(r2, 20, 30, 30, 40);
  PenPat(white);
  PaintRect(r2);
  PenPat(black);
  FrameOval(r2);
  TextFont(helvetica);
  TextSize(9);
  MoveTo(23, 8);
  DrawString('n');
  MoveTo(20, 48);
  DrawString('1/n');
  ClosePicture;
  AddResource(Handle(pic), 'PICT', 258, 'Picture 130');

  pic := OpenPicture(r1);
  NPort;
  SPort;
  MoveTo(14, 0);
  LineTo(14, 49);
  MoveTo(35, 0);
  LineTo(35, 49);
  SetRect(r2, 10, 20, 20, 30);
  PenPat(white);
  PaintRect(r2);
  PenPat(black);
```

```
    FrameOval(r2);
    SetRect(r2, 30, 20, 40, 30);
    PenPat(white);
    PaintRect(r2);
    PenPat(black);
    FrameOval(r2);
    TextFont(helvetica);
    TextSize(9);
    MoveTo(3, 28);
    DrawString('n');
    MoveTo(42, 26);
    DrawString('1');
    MoveTo(43, 32);
    DrawString('n');
    MoveTo(41, 25);
    LineTo(47, 25);
    ClosePicture;
    AddResource(Handle(pic), 'PICT', 259, 'Picture 131');

    CloseResFile(ref);
end.
```

APPENDIX D

PROGRAM AND TEXT FILE TO GENERATE THE CONNECTIONS DATA STRUCTURE AND STORE IT IN A RESOURCE FILE AS RESOURCE TYPE 'PORT'

(pages 249-254)

```
program SetPortCons;

 const
  Npicts = 131;

 type
  Connection = record
    N, S, E, W: integer;
   end;

 var
  inFile: text;
  PortCons: array[0..Npicts] of Connection;
  e, ref, k: integer;
  hand: Handle;

begin
 reset(inFile, 'PortConsFile');
 for k := 0 to Npicts do
  begin
   with PortCons[k] do
    readln(inFile, N, S, E, W);
  end;
 close(inFile);
 ref := OpenResFile('Thesis.res_B');
 e := PtrToHand(@PortCons, hand, SizeOf(PortCons));
 AddResource(hand, 'PORT', 128, 'Port Connections');
 CloseResFile(ref);
end.
```

The contents of the PortConsFile TEXT file:

```
0 0 0 0
0 0 0 1
0 0 1 0
1 0 0 0
0 1 0 0
0 0 0 1
0 0 1 0
1 0 0 0
0 1 0 0
0 0 0 1
0 0 1 0
1 0 0 0
0 1 0 0
0 0 2 1
2 1 0 0
0 0 1 2
1 2 0 0
0 0 2 1
2 1 0 0
0 0 2 1
2 1 0 0
0 0 1 2
1 2 0 0
0 0 2 1
2 1 0 0
3 0 2 1
0 3 2 1
2 1 3 0
2 1 0 3
1 0 3 2
0 1 3 2
3 2 1 0
3 2 0 1
1 0 2 3
0 1 2 3
2 3 1 0
2 3 0 1
```

3 0 2 1
0 3 2 1
1 2 3 0
1 2 0 3
3 0 2 1
0 3 2 1
2 1 3 0
2 1 0 3
1 0 3 2
0 1 3 2
3 2 1 0
3 2 0 1
1 0 2 3
0 1 2 3
2 3 1 0
2 3 0 1
3 0 2 1
0 3 2 1
1 2 3 0
1 2 0 3
2 4 3 1
3 4 2 1
4 3 2 1
1 2 4 3
1 2 3 4
2 4 3 1
3 4 2 1
4 3 2 1
1 2 4 3
1 2 3 4
0 0 0 1
0 0 1 0
1 0 0 0
0 1 0 0
0 0 0 1
0 0 1 0
1 0 0 0
0 1 0 0
0 0 1 2

1 2 0 0
0 0 2 1
2 1 0 0
0 0 1 2
1 2 0 0
0 0 2 1
2 1 0 0
0 0 1 2
1 2 0 0
0 0 2 1
2 1 0 0
0 0 1 2
1 2 0 0
0 0 2 1
2 1 0 0
0 0 1 2
1 2 0 0
0 0 2 1
2 1 0 0
0 0 1 2
1 2 0 0
0 0 2 1
2 1 0 0
0 0 1 2
1 2 0 0
0 0 2 1
2 1 0 0
0 0 1 2
1 2 0 0
0 0 2 1
2 1 0 0
0 0 1 2
1 2 0 0
0 0 2 1
2 1 0 0
0 0 1 2
1 2 0 0
0 0 2 1
2 1 0 0
0 0 2 1

2 1 0 0
0 0 1 2
1 2 0 0
2 0 0 1
0 2 0 1
1 0 2 0
0 2 1 0
2 0 3 1
0 3 2 1
2 3 0 1
1 3 2 0
2 4 3 1
0 0 2 1
1 2 0 0
0 0 2 1
2 1 0 0

APPENDIX E

PROGRAM TO READ A PICT FILE (CREATED BY MACDRAW, MACPAINT, ECT.)
AND SAVE IT IN A RESOURCE FILE AS RESOURCE TYPE 'PICT'

(pages 255-256)

```
program PICT_to_Res;

  var
    globalRef, resRef, resNum, picSize: integer;
    err: OSErr;
    byteCount: longint;
    PICTHandle: PicHandle;

  function GetPICTFile: boolean;
    var
      wher: Point;
      reply: SFReply;
      fileTypes: SFTypeList;
      err: OSErr;
  begin
    wher.h := 82;
    wher.v := 45;
    fileTypes[0] := 'PICT';
    SFGetFile(wher, 'Select PICT file:', nil, 1, fileTypes, nil, reply);
    GetPICTFile := reply.good;
    if reply.good then
      err := FSOpen(reply.fName, reply.vRefNum, globalRef);
  end;

begin
  resRef := OpenResFile('Thesis.res_B');
  resNum := 650;
  while GetPICTFile do begin
    err := SetFPos(globalRef, fsFromStart, 512);
    byteCount := SizeOf(picSize);
    err := err + FSRead(globalRef, byteCount, @picSize);
    writeln('picSize ', picSize);
    err := SetFPos(globalRef, fsFromMark, -byteCount);
    byteCount := picSize;
    PICTHandle := PicHandle(NewHandle(byteCount));
    err := err + FSRead(globalRef, byteCount, Ptr(PICTHandle^));
    writeln('err ', err);
    if err = 0 then begin
      AddResource(Handle(PICTHandle), 'PICT', resNum, '');
      resNum := resNum + 1;
    end;
    err := FSClose(globalRef);
  end;
  CloseResFile(resRef);
end.
```