Design and Implementation of Odd-Order Wave Digital Lattice Lowpass Filters: From Specifications to Motorola DSP56307EVM Module

By

Yidong Qi

A Thesis

Presented to the Faculty of Graduate Studies

In Partial Fulfillment of the Requirements

For the Degree

MASTER OF SCIENCE

Department of Electrical and Computer Engineering

University of Manitoba

Winnipeg, Manitoba

June 2001

To my grandmother

THE UNIVERSITY OF MANITOBA

FACULTY OF GRADUATE STUDIES

COPYRIGHT PERMISSION

DESIGN AND IMPLEMENTATION OF

ODD-ORDER WAVE DIGITAL LATTICE LOWPASS FILTERS:

FROM SPECIFICATIONS TO MOTOROLA DSP56307EVM MODULE

BY

YIDONG QI

MASTER OF SCIENCE

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.

Yidong Qi

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

Yidong Qi

Contents

Design and Implementation of Odd-Order Wave Digital Lattice Lowpass Filters: From Specifications to Motorola DSP56307EVM Module

Yidong Qi

Abstract

This thesis is dedicated to applying and developing explicit formulas for the design and implementation of odd-order lattice lowpass wave digital filters (WDFs) on a Digital Signal Processor (DSP), such as a Motorola DSP56307EVM (Evaluation Module). The direct design method of Gazsi for filter types such as Butterworfh, Chebyshev, inverse Chebyshev, and Cauer (Elliptic) provides a straightforward method for calculating the coefficients without an extensive knowledge of digital signal processing.

A program package to design and implement odd-order WDFs, including detailed procedures and examples, is presented in this thesis and includes not only the calculations of the coefficients, but also the simulation on a MATLAB platform and an implementation on a Motorola DSP56307EVM board. It is very quick, effective and convenient to obtain the coefficients when the user enters a few parameters according to the general specifications; to verify the characteristics of the designed filter; to simulate the filter on the MATLAB platform; to implement the filter on the DSP board; and to compare the results between the simulation and the implementation.

Acknowledgments

I wish to acknowledge all those who have given me their support, encouragement and assistance in helping me to complete this thesis.

I would like to thank my advisor Dr. G.O. Martens for his constant guidance, encouragement and constructive criticism throughout this study. This thesis would not have been possible without his support. My sincere appreciation is extended to Dr. H. Finlayson and Dr. R.D. McLeod for their reviews.

I would like to express my heartfelt appreciation to my family. I am forever grateful to my parents and my wife, Weimin, whose unconditional love, emotional support and understanding has made this thesis possible.

Chapter 1

Introduction

With digital realizations available for a variety of analog elements, several families of wave digital filters can be obtained by converting classical lattice, ladder, microwave, and other types of analog filters into digital filters.

Wave digital filters (WDFs)[1] are modeled on classical filters, especially in lattice or ladder configurations or generalizations thereof. They have some excellent advantages concerning low coefficient accuracy requirements, dynamic range, and all aspects of stability even under finite-arithmetic conditions.

There are a number of different ways to achieve WDF realizations using various methods. Lajos Gazsi [2] introduced a very convenient and simple approach, named "Explicit Formulas for Lattice Wave Digital Filters", which is to design the most common filter types, such as Butterworth, Chebyshev, inverse Chebyshev, and Cauer (elliptic) filter responses. The design process derives the coefficients directly, and can be of benefit for people who are not familiar with complicated digital signal processing theory, but need to design filters for their own digital signal processing applications, for instance in medical analysis, image processing, speech processing, etc., areas which might not have specialized filter designers.

With developments in science and technology, computer program applications have already been used in almost every area in the world. There are many commercial DSP (Digital Signal Processing) chips available from stock. As mentioned before, people, who lack experience and knowledge in classical network theory and DSP implementation, should be given the means to design filters and implement them on a commercial DSP chip to meet their own specifications.

The main purpose in this thesis is dedicated to develop Gazsi's method to realize DSP applications. In chapter 2, we investigate the basic principles of WD lattice lowpass filters. Sections 2.1-2.3 review the background knowledge required for the derivation of wave lattice digital filers. Section 2.4 describes the structure of the components of allpass functions of degree one and two for the synthesis and realization of lattice configurations. Section 2.5 is focused on the theoretical basis of the explicit formulas for WD lattice filter design.

In chapter 3, we describe the design specifications for digital filters at first, and then review the process of coefficient computations with explicit formulas [2]. Sections 3.3-3.5 describe how to calculate the coefficients with filter types, such as Butterworth, Chebyshev and Cauer responses.

In chapter 4, the general design procedures are highlighted at the beginning point, and then combine this with the design of a 9^{th} order Cauer lowpass filter as an example to demonstrate the whole procedure and results at each step. Basically, we need to make a simulation and a DSP implementation with the same specifications, and compare differences between them to see whether or not the DSP implementation is realizable in terms of the simulation. The procedures include directory establishment, coefficient calculations by user interface, filter characteristics, test vector generation, verification of the simulation, source compilation of the implementation, the implementation on the DSP56307EVM board, data manipulations, and data analysis comparison between Matlab simulation and DSP implementation. Due to a variety of source codes, data files and scripts, we introduce the file management and file name conventions in section 4.13.

In chapter 5, we give the design discussion and state the conclusions.

In the references section, we list all references used in this thesis.

In the appendix section, we provide the results of a 5^{th} order Chebyshev WD lowpass filter in appendix A, the results of a 7^{th} order Butterworth WD lowpass filter in appendix B, the list of the design tools in appendix C, the features of the Motorola DSP56307EVM in appendix D, and the all unduplicated source codes in appendix E for readers. The source codes are split to three groups in terms of the design of the different filter types.

Chapter 2

The Basic Principles of a WD Lattice Lowpass Filter

2.1 The transformation from voltage current network to voltage wave network

In general, a linear time-invariant (LTI) electrical N-port network can be illustrated as in Figure 2.1.

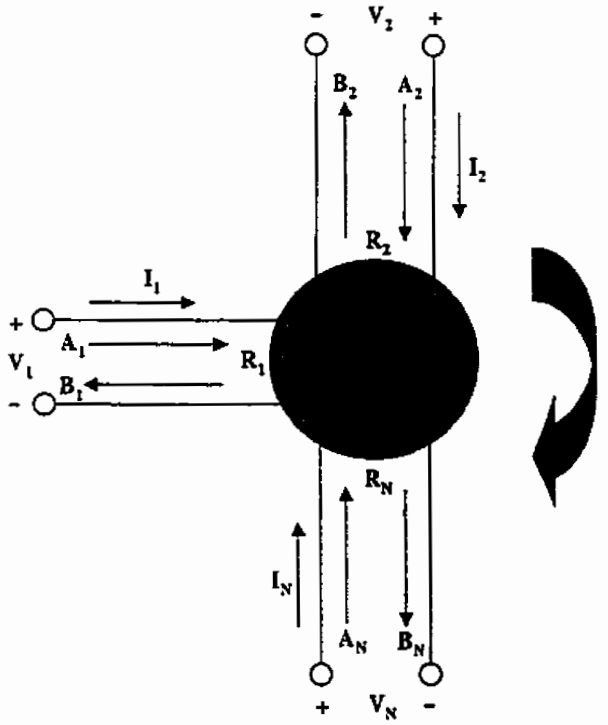


Figure 2.1 An LTI N-port network in the voltage-current domain

The signal quantities at each port consist of a voltage (V_i) and a current (I_i). We can represent a pair of equations for the i^{th} port as follows:

$$A_i = V_i + R_i I_i \qquad (2.1a)$$

$$B_i = V_i - R_i I_i \qquad (2.1b)$$

where A_i and B_i are incident and reflected voltage waves, respectively. The wave quantities (A_i, B_i) for the entire network can be described by a voltage vector **V** and a current vector **I**. We define a vector transformation from the voltage-current vector domain to voltage-wave vector domain. The incident voltage wave vector **A** is given by

$$\mathbf{A} = \mathbf{V} + \mathbf{R}\,\mathbf{I} \qquad (2.2a)$$

and the reflected voltage wave vector **B** is given by

$$\mathbf{B} = \mathbf{V} - \mathbf{R}\,\mathbf{I} \qquad (2.2b)$$

The matrix R is defined to be a real positive definite diagonal matrix. Thus the matrix is of the form

$$\mathbf{R} = diag\,(\,R_1, R_2, \ldots, R_i, \ldots, R_n\,) \qquad R_i > 0,\ i = 1, 2, \ldots, n \qquad (2.3)$$

From (2.2a) and (2.2b), the mapping of voltage current to voltage wave can be rewritten in a matrix form, such as

$$\begin{bmatrix} \mathbf{A} \\ \mathbf{B} \end{bmatrix} = \begin{bmatrix} \mathbf{I}_n & \mathbf{R} \\ \mathbf{I}_n & -\mathbf{R} \end{bmatrix} \begin{bmatrix} \mathbf{V} \\ \mathbf{I} \end{bmatrix} \qquad (2.4)$$

where $\mathbf{I}_n$ is a n by n identity matrix.

R is positive definite by our definition, therefore it guarantees the existence of the inverse matrix of (2.4). The mapping between the voltage current domain and voltage wave domain is linear and one to one. The inverse matrix of (2.4) is

$$\begin{bmatrix} \mathbf{V} \\ \mathbf{I} \end{bmatrix} = \tfrac{1}{2} \begin{bmatrix} \mathbf{I}_n & \mathbf{I}_n \\ \mathbf{R}^{-1} & -\mathbf{R}^{-1} \end{bmatrix} \begin{bmatrix} \mathbf{A} \\ \mathbf{B} \end{bmatrix} \qquad (2.5)$$

Since the matrix **R** is diagonal, the mapping of (2.4) can be rewritten as

$$\begin{bmatrix} \mathbf{A}_j \\ \mathbf{B}_j \end{bmatrix} = \begin{bmatrix} 1 & \mathbf{R}_j \\ 1 & -\mathbf{R}_j \end{bmatrix} \begin{bmatrix} \mathbf{V}_j \\ \mathbf{I}_j \end{bmatrix} \qquad \text{for } j = 1, 2, \ldots, n \qquad (2.6)$$

We know that the transformation from voltage current to voltage wave is the mapping of the signal quantities port by port. The network of Figure 2.1 can be represented in the voltage wave domain as follows:

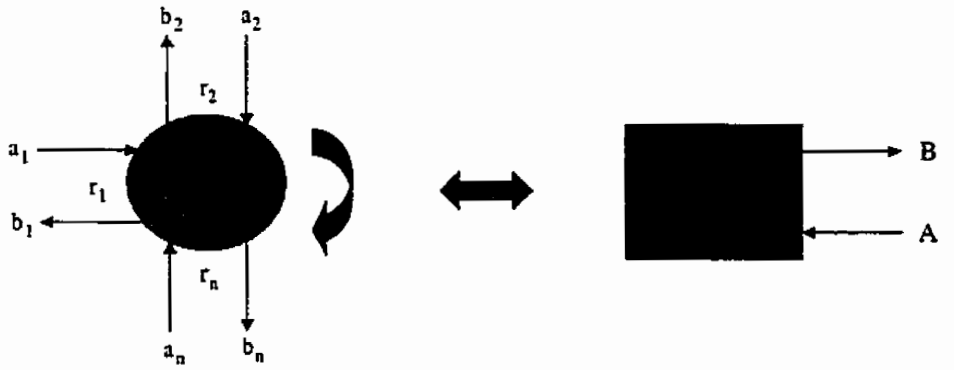


Figure 2.2 An equivalent LTI N-port network in the analog voltage wave domain

2.2 The Interconnection of Network Ports

Consider two isolated 2-port networks cascaded in the voltage current domain shown in Figure 2.3.

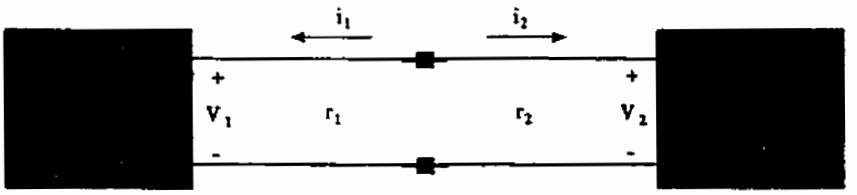


Figure 2.3 The interconnection of two 2-port network in voltage-current domain

We note that this connection results in the constraints

$$v_1 = v_2 \tag{2.7a}$$

and

$$i_1 = -i_2 \tag{2.7b}$$

Now consider the same interconnection of the above in the voltage wave domain. From (2.6), (2.7a) and (2.7b), we have

$$a_1 = v_1 + r_1 i_1 = v_2 - r_1 i_2 \tag{2.8a}$$

and

$$b_1 = v_1 - r_1 i_1 = v_2 + r_1 i_2 \tag{2.8b}$$

If $r_1 = r_2$ is assumed, we can see that

$$a_1 = v_2 - r_2 i_2 = b_2 \tag{2.9a}$$

and

$$b_1 = v_2 + r_2 i_2 = a_2 \tag{2.9b}$$

Therefore we know that the ports can be interconnected as shown in Figure 2.4 if the port resistances are equal.

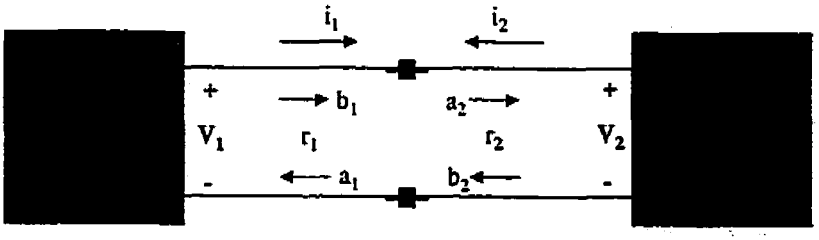


Figure 2.4 The interconnection of two 2-port network in voltage-wave domain

When we take the mapping from the continuous-time voltage wave domain to the discrete-time domain by using a bilinear transformation of the frequency variable, a delay-free directed loop may be created in the interconnection as showed in Figure 2.5. It will cause the discrete-time network to be non-computable and must be avoided in our design [1].

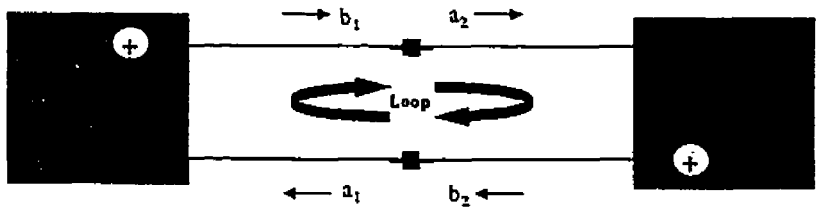


Figure 2.5 A delay-free directed loop in the interconnection of two 2-port network in discrete-time domain

2.3 The derivation of a wave lattice digital filter

An LC resistively terminated filter can be regarded as a combination of some impedances (R, sL, or 1/sC), a source (V or I), and a few 2-port series wire interconnections. Realizing these elements digitally and then substituting for the analogy elements in the LC filter by their digital realizations can synthesize a wave digital filter.

The family of wave lattice filters is based on the lattice networks of Figure 2.6, where Z_A and Z_B are usually canonical, lossless, LC impedances.

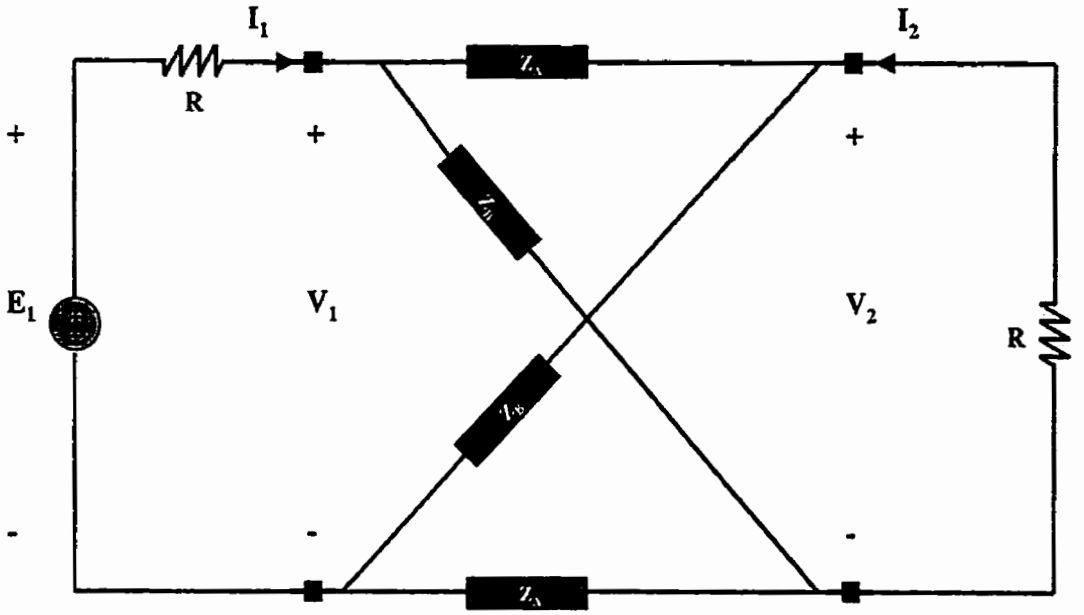


Figure 2.6 An analog lattice network

As any other 2-port network, the lattice network of Figure 2.6 can be translated by the wave characterization as follows:

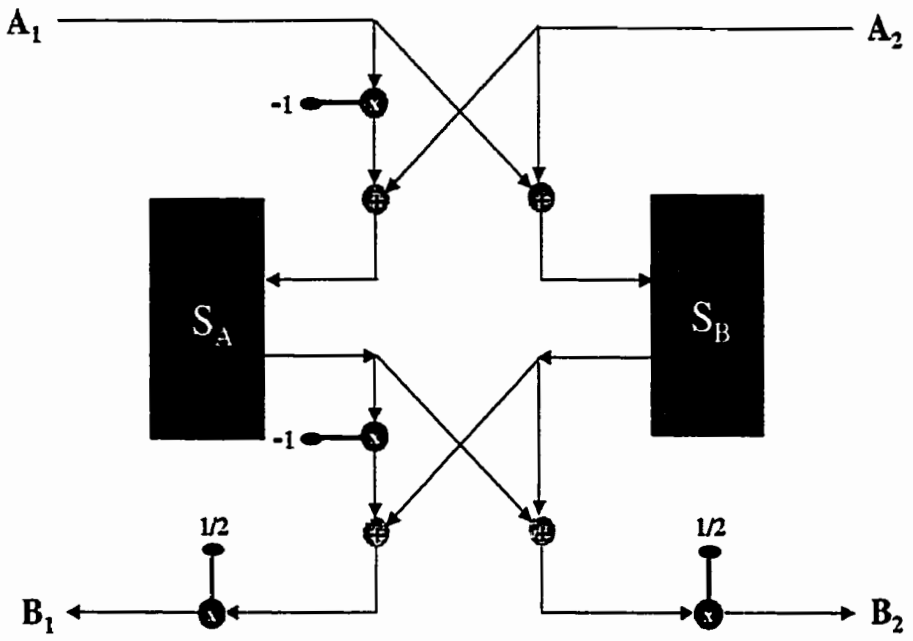


Figure 2.7a An alternative realization in terms of the wave characterization

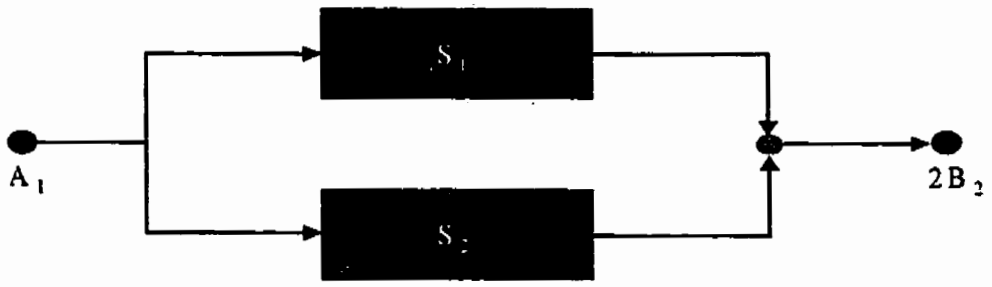


Figure 2.7b A simplified wave configuration ($A_2 = 0$, B_2 is the only output)

If we assign port resistances $R_1 = R_2 = R$ and eliminate I_1, I_2, and V_1, V_2 in (2.1a) and (2.1b) using

$$\begin{bmatrix} V_1 \\ V_2 \end{bmatrix} = \frac{1}{2} \begin{bmatrix} Z_A + Z_B & Z_B - Z_A \\ Z_B - Z_A & Z_A + Z_B \end{bmatrix} \begin{bmatrix} I_1 \\ I_2 \end{bmatrix}$$

from Figure 2.6, we obtain

$$\mathbf{B} = \mathbf{S}\,\mathbf{A} \tag{2.10}$$

where

$$\mathbf{S} = \begin{bmatrix} S_{11} & S_{12} \\ S_{21} & S_{22} \end{bmatrix} \tag{2.11}$$

with $$S_{11} = S_{22} = \tfrac{1}{2}(S_B + S_A) \tag{2.12a}$$

$$S_{12} = S_{21} = \tfrac{1}{2}(S_B - S_A) \tag{2.12b}$$

$$S_A = \frac{Z_A - R}{Z_A + R} \tag{2.13a}$$

$$S_B = \frac{Z_B - R}{Z_B + R} \tag{2.13b}$$

These equations lead to the lattice realization of a WD filter.

2.4 The realization of a wave lattice digital filter

From Figure 2.4, we can rewrite (2.1a) and (2.1b) as follow

$$a_1 = v_1 + r_1 i_1 \tag{2.14a}$$

$$b_1 = v_1 - r_1 i_1 \tag{2.14b}$$

and

$$a_2 = v_2 + r_2 i_2 \tag{2.15a}$$

$$b_2 = v_2 - r_2 i_2 \tag{2.15b}$$

As mentioned in 2.2, (2.7a) and (2.7b) gives the relations of voltages and currents, we can substitute them into (2.15a), and obtain

$$a_2 = v_1 - r_2 i_1 \tag{2.16}$$

From (2.14), (2.15) and (2.7), we obtain

$$i_1 = - \frac{a_2 - a_1}{r_1 + r_2} \qquad (2.17)$$

$$v_1 = \frac{r_2 a_1 + r_1 a_2}{r_1 + r_2} \qquad (2.18)$$

Substituting (2.17) and (2.18) into (2.14b) and (2.15b) using (2.7) respectively, i.e.

$$b_1 = a_2 + \gamma_0 (a_2 - a_1) \qquad (2.19a)$$

$$b_2 = a_1 + \gamma_0 (a_2 - a_1) \qquad (2.19b)$$

Equations (2.19a) and (2.19b) describe a two port known as a two-port adaptor.

Where

$$\gamma_0 = \frac{r_1 - r_2}{r_1 + r_2} \qquad (2.20)$$

Now we move to the discussion of the allpass sections of degree one and degree two for lattice WD filter synthesis using two-port adaptors and delays.

2.4.1 An allpass section of degree one

To realize a degree one allpass section, an adaptor and a delay are required as shown in Figure 2.8.

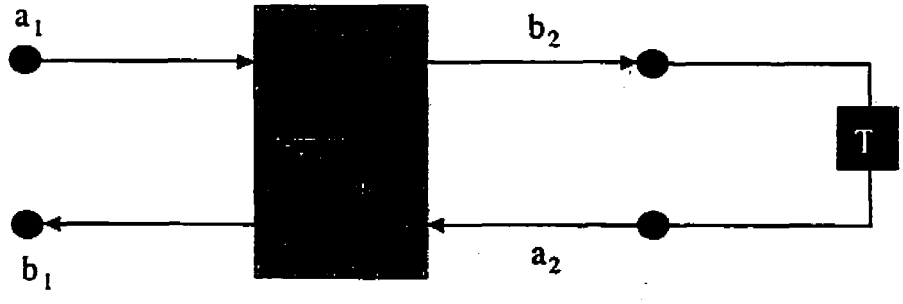


Figure 2.8 Wave-flow diagram of the allpass section of degree one

From Figure 2.8, we have

$$a_2 = z^{-1} b_2 \qquad (2.21)$$

Using (2.19) and (2.20) the reflectance $S = b_1/a_1 = (z^{-1} - \gamma_0)/(1-\gamma_0 z^{-1})$ (2.22)

Furthermore

$$\gamma_0 = \frac{1 - B_0}{1 + B_0} \qquad (2.23)$$

where B_0 is the parameter in the reflectance $S(\psi) = (\psi - B_0)/(\psi + B_0)$. Substituting $\psi = (1-z^{-1})/(1+z^{-1})$ and comparing with (2.22) yields (2.23) [3].

2.4.2 An allpass section of degree two

To realize a degree two allpass section, two two-port adaptors and two delays are required as shown in Figure 2.9.

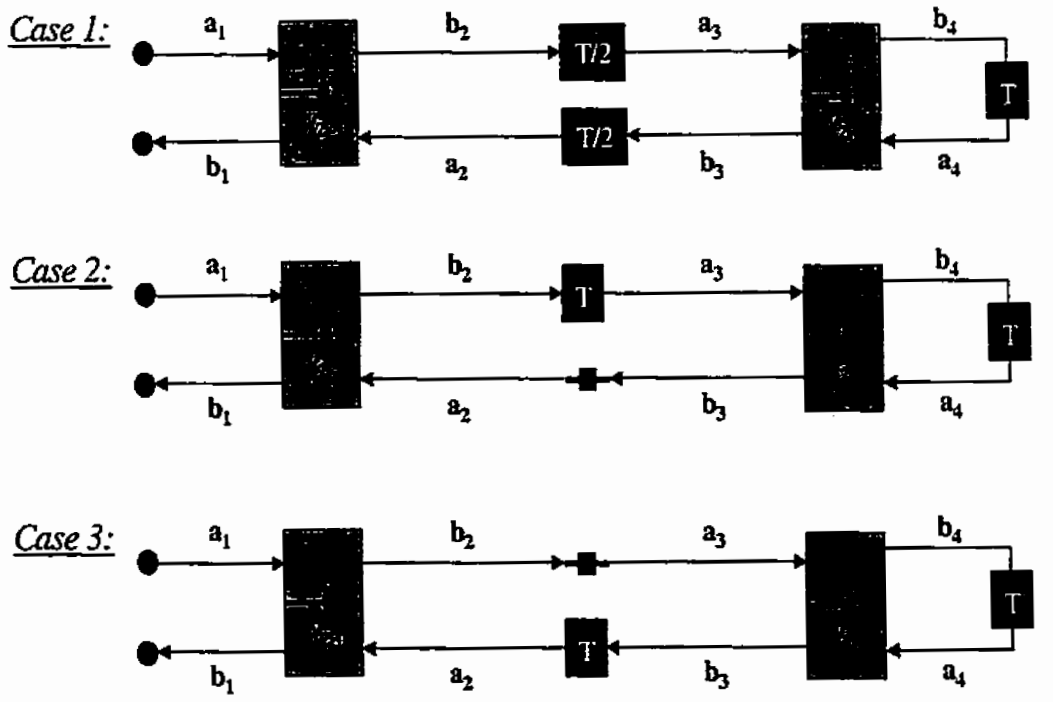


Figure 2.9 Equivalent wave-flow diagrams of the i^{th} second degree allpass section

We apply the results in (2.19a) and (2.19b), we have

$$b_1 = a_2 + \gamma_{2i-1} (a_2 - a_1) \qquad (2.24a)$$

$$b_2 = a_1 + \gamma_{2i-1} (a_2 - a_1) \qquad (2.24b)$$

$$b_3 = a_4 + \gamma_{2i} (a_4 - a_3) \qquad (2.25a)$$

$$b_4 = a_3 + \gamma_{2i} (a_4 - a_3) \qquad (2.25b)$$

Now we discuss the three different wave flow diagrams, respectively.

Case 1: from this diagram, we have

$$a_4 = z^{-1} b_4 \qquad (2.26a)$$

$$a_3 = z^{-1/2} b_2 \qquad (2.26b)$$

$$a_2 = z^{-1/2} b_3 \qquad (2.26c)$$

Substituting (2.25a) into (2.24b)

$$b_4 = \frac{1 - \gamma_{2i}}{1 - \gamma_{2i} z^{-2}} a_3 \qquad (2.27)$$

Substituting (2.27), (2.26a) and (2.26b) into (2.25a)

$$b_3 = \frac{z^{-3/2} - \gamma_{2i} z^{-1/2}}{1 - \gamma_{2i} z^{-1}} b_2 \qquad (2.28)$$

Substituting (2.28) into (2.26c)

$$b_2 = \frac{z - \gamma_{2i}}{z^{-1} - \gamma_{2i}} a_2 \qquad (2.29)$$

Substituting (2.29) into (2.24b)

$$a_2 = \frac{(z^{-1} - \gamma_{2i}) (1 - \gamma_{2i-1})}{z - \gamma_{2i} (1 - \gamma_{2i-1}) - \gamma_{2i-1} z^{-1}} a_1 \qquad (2.30)$$

Substituting (2.30) into (2.24a) and putting it into the form of a transfer function

$$S = \frac{b_1}{a_1} = \frac{- \gamma_{2i-1} - \gamma_{2i} (1 - \gamma_{2i-1}) z^{-1} + z^{-2}}{1 - \gamma_{2i} (1 - \gamma_{2i-1}) z^{-1} - \gamma_{2i-1} z^{-2}} \qquad (2.31)$$

Case 2: from this diagram, we have

$$a_4 = z^{-1} b_4 \tag{2.32a}$$

$$a_3 = z^{-1} b_2 \tag{2.32b}$$

$$a_2 = b_3 \tag{2.32c}$$

We can use the same approach as in Case 1 to derive b_4, b_3, b_2, the results are

$$b_4 = \frac{1 - \gamma_{2i}}{1 - \gamma_{2i} z^{-1}} a_3 \tag{2.33}$$

$$b_3 = \frac{(z^{-1} - \gamma_{2i}) z^{-1}}{1 - \gamma_{2i} z^{-1}} b_2 \tag{2.34}$$

$$b_2 = \frac{z - \gamma_{2i}}{z^{-1} - \gamma_{2i}} a_2 \tag{2.35}$$

Comparing (2.35) with (2.29), we see the same result. The next step is to substitute (2.35) into (2.24b). Obviously, we will get the same relationship between a_1 and a_2 if we recall (2.30) process. Furthermore, the transfer function S is identical to (2.31).

Case 3: from this diagram, we have

$$a_4 = z^{-1} b_4 \tag{2.36a}$$

$$a_3 = b_2 \tag{2.36b}$$

$$a_2 = z^{-1} b_3 \tag{2.36c}$$

We follow the same idea as in the Case 2 discussion, we get

$$b_4 = \frac{1 - \gamma_{2i}}{1 - \gamma_{2i} z^{-1}} a_3 \tag{2.37}$$

$$b_3 = \frac{z^{-1} - \gamma_{2i}}{1 - \gamma_{2i} z^{-1}} b_2 \tag{2.38}$$

From (2.36c), we have

$$b_2 = \frac{z - \gamma_{2i}}{z^{-1} - \gamma_{2i}} a_2 \tag{2.35}$$

(2.39) is equal to (2.35), as discussed in Case 2, (2.39) will result in the same conclusion regarding transfer function S.

Furthermore

$$\gamma_{2i} = \frac{1 - B_i}{1 + B_i} \qquad (2.40)$$

$$\gamma_{2i-1} = \frac{A_i - B_i - 1}{A_i + B_i + 1} \qquad (2.41)$$

where

$$S(\Psi) = \frac{\Psi^2 - A_i\Psi + B_i}{\Psi^2 + A_i\Psi + B_i} \quad \text{and substituting} \quad \Psi = \frac{1 - z^{-1}}{1 + z^{-1}}$$

and comparing with (2.31) yields (2.40) and (2.41) [3].

2.4.3 The synthesis by cascaded allpass sections

Finally, we can build the completed structure of a lattice WD filter for the most common cases, such as Butterworth, Chebyshev and Cauer (elliptic) responses. The block diagram is as follows:

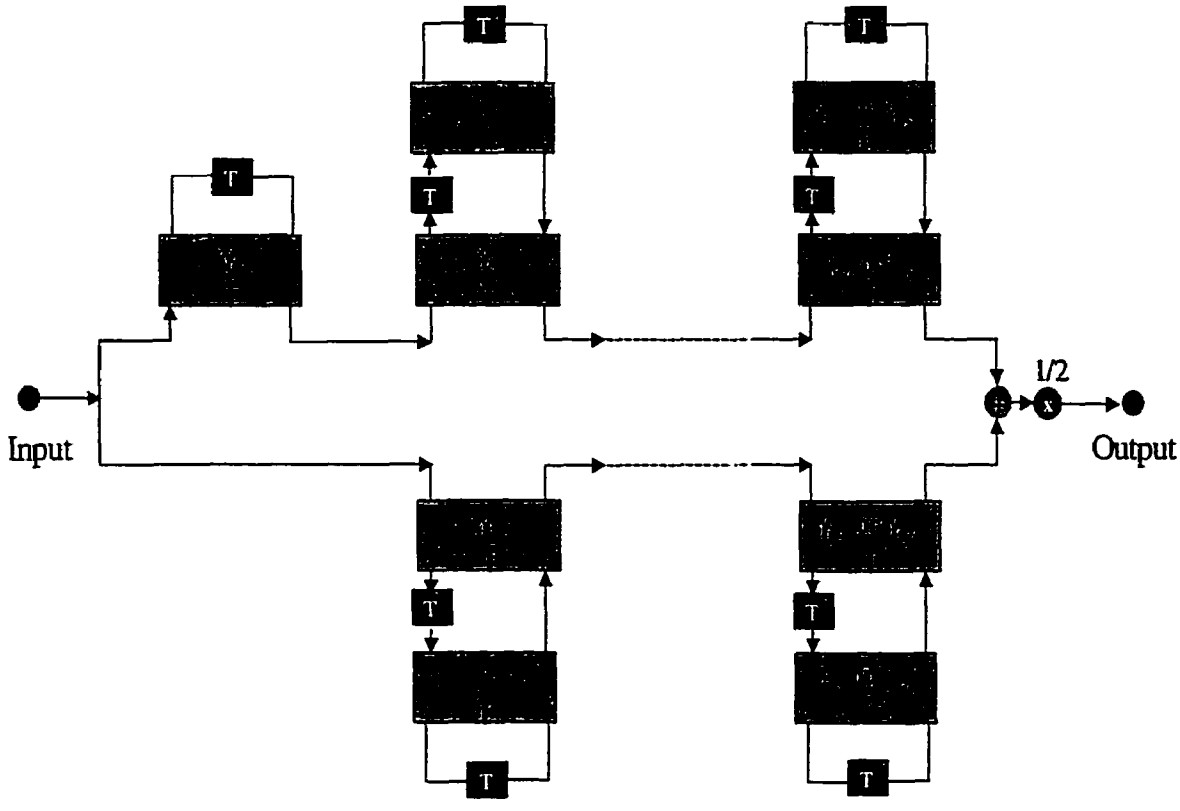


Figure 2.10 **Block diagram of the odd order common WD lattice filter where n = 5, 9, 13, ...as the upper structure**
n = 7, 11, 15, ... as the lower structure

2.5 The principles of the explicit formulas for WD lattice filter design

As we discussed in 2.4, a WD filter is derived from a real lossless reference filter via the voltage wave quantities. For a lattice WD filter, the reference filter is a real symmetric two port.

The reference filter is defined in the Ψ domain. The appropriate choice for Ψ is the bilinear transformation of the z-variable.

$$\Psi = \frac{z - 1}{z + 1} \qquad (2.42)$$

To consider the frequency response, the Laplace variable Ψ is evaluated on the imaginary axis and the z-variable is evaluated on the unit circle. i.e.

$$\Psi = j\varphi \qquad (2.43)$$

$$z = e^{j\omega T} \qquad (2.44)$$

where φ is the analog frequency and ω is the digital frequency.

Substituting (2.43), (2.44) into (2.42)

$$\varphi = \tan(\omega T/2) \tag{2.45}$$

where $T = 1/F$ and F is the sampling frequency. (2.46)

From Figure 2.7b, both lattice branches of the lattice WDF $S_1(\psi)$ and $S_2(\psi)$ are allpass functions. Consequently, we can describe them as follow

$$S_1 = \pm \frac{g_1(-\psi)}{g_1(\psi)} \tag{2.47a}$$

$$S_2 = \frac{g_2(-\psi)}{g_2(\psi)} \tag{2.47b}$$

where $g_1(\psi)$ and $g_2(\psi)$ are Hurwitz polynomials of degree N_1 and N_2, respectively. The corresponding transfer functions can be derived as follows:

$$S_{11} = S_{22} = \frac{S_1 + S_2}{2} = \frac{h(\psi)}{g(\psi)} \tag{2.48a}$$

$$S_{12} = S_{21} = \frac{S_2 - S_1}{2} = \frac{f(\psi)}{g(\psi)} \tag{2.48b}$$

where $h(\psi)$, $f(\psi)$ and $g(\psi)$ are called canonic polynomials.

Substituting (2.47a), (2.47b) into (2.48a) or (2.48b), we get

$$g(\psi) = g_1(\psi)\, g_2(\psi) \tag{2.49}$$

$$h(\psi) = \tfrac{1}{2}\{ g_1(\psi)\, g_2(-\psi) +/- g_1(-\psi)\, g_2(\psi) \} \tag{2.50a}$$

$$f(\psi) = \tfrac{1}{2}\{ g_1(\psi)\, g_2(-\psi) -/+ g_1(-\psi)\, g_2(\psi) \} \tag{2.50b}$$

From the above, we know that g (ψ) is a Hurwitz polynomial of degree N, i.e. the filter order. (It must be an odd number for lowpass filters.) Consequently, a lattice WDF is the sum of the degrees of the two reflectances S_1 and S_2, i.e. $N = N_1 + N_2$.

Chapter 3

The Coefficient Calculations with Explicit Formulas For Lattice Wave Digital Filters

We take advantage of explicit formulas as the basic approach to the design of lattice wave digital filters, and to achieve designs of the most common type of WD filters, such as Butterworth, Chebyshev, and Cauer filters. We write programs for the coefficient computation and frequency response characteristics based on the filter specifications and properties.

First of all, we define some notations in terms of lowpass filter design specifications corresponding to attenuation as shown in Figure 3.1.

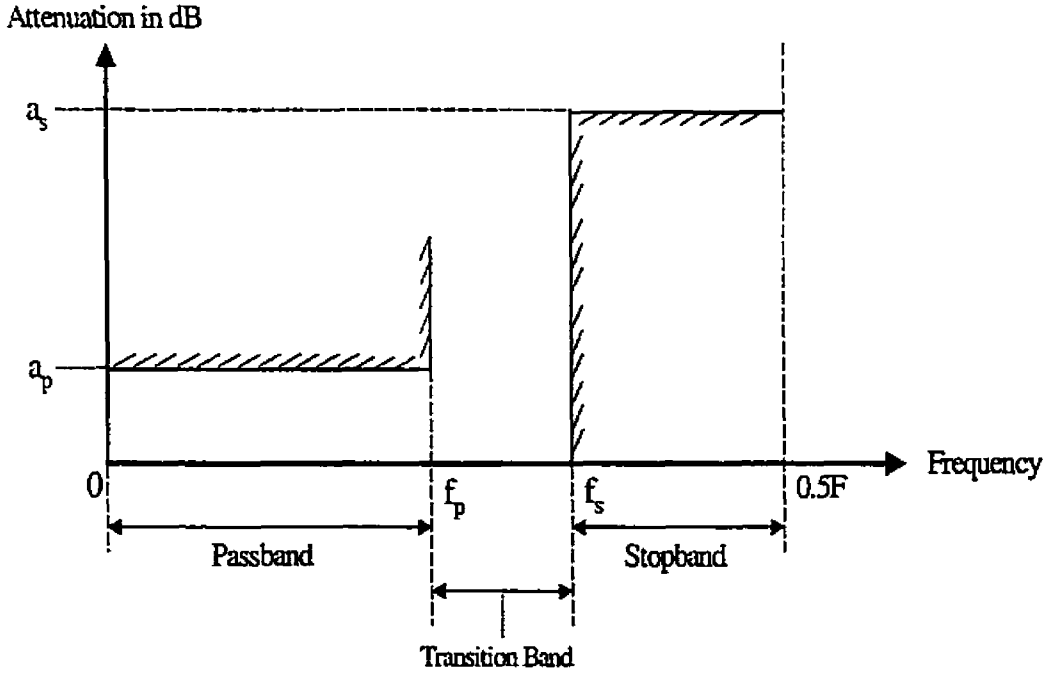


Figure 3.1 A lowpass filter design specifications

In the above figure, we now define the notation as follows:

- a_p: maximum allowable attenuation in the passband
- a_s: specified minimum attenuation in the stopband
- f_p: upper boundary frequency of the passband
- f_s: lower boundary frequency of the stopband
- F: sampling frequency

3.1 The description of the digital lowpass filter specifications

The digital lowpass filter specification used is illustrated in Figure 3.1. The attenuation function of the digital filter $A_D(e^{j\omega})$ is defined by

$$A_D(e^{j\omega}) = -20 \log (| H_D(e^{j\omega}) |) \qquad (3.1)$$

where $H_D(e^{j\omega})$ is the transfer function of the digital filter. The units of the attenuation axis are decibels. The frequency variable ω has units of radians and is related to the z-transform variable by

$$z = e^{j\omega} \qquad (3.2)$$

where $\omega = 2\pi f/F$ and $\varphi = \tan(\omega/2) = \tan(\pi f/F)$

The attenuation function must be within the un-shaded sector shown in Figure 3.1.

The five variables that constitute the specification set (a_p, a_s, f_p, f_s, F) are described as follows. The frequency band (0, 0.5F) is divided into three sub-bands by the variables f_p and f_s. We define the sub-band (0, f_p) to be the passband, (f_p, f_s) to be transition band, and (f_s, 0.5F) to be the stopband. On the attenuation axis we define a reference level whose value is given by the minimum attenuation of the specific transfer function solution in the passband. The variable a_p defines the maximum attenuation relative to the reference level that the attenuation function $A_D(e^{j\omega})$ is allowed within the passband. The variable a_s defines the minimum attenuation relative to the reference level that the attenuation function $A_D(e^{j\omega})$ is allowed within the stopband. There are no restrictions on the attenuation function $A_D(e^{j\omega})$ within the transition band. The design procedure in the following sections is taken from Gazsi[2].

3.2 The coefficient computation

$$\varepsilon_s = \sqrt{10^{as/10} - 1} \tag{3.3}$$

$$\varepsilon_p = \sqrt{10^{ap/10} - 1} \tag{3.4}$$

where ε_s, ε_p are the ripple factors in the stopband and passband, respectively.

$$\varphi_s = \tan(\pi f_s / F) \tag{3.5}$$

$$\varphi_p = \tan(\pi f_p / F) \tag{3.6}$$

where φ_s, φ_p are corresponding analog frequencies determined by the pre-warping effect of the bilinear transformation.

Secondly, we consider the determination of the filter order N according to the above specifications. A minimum value for the lowpass filter order could be estimated by the following approximations

$$n_{min} = \frac{c_1 \ln(c_2 \varepsilon_s/\varepsilon_p)}{\ln(c_3)} \tag{3.7}$$

where c_1, c_2, and c_3 are given in Table 1 with

$$k_0 = \sqrt{\varphi_s / \varphi_p} \tag{3.8}$$

and

$$k_{i+1} = k_i^2 + \sqrt{k_i^4 - 1}, \quad \text{for } i=0,1,2,3. \tag{3.9}$$

Table 1: Parameters for Approximation of the Filter Order

Filter Type	c_1	c_2	c_3
Butterworth	1	1	k_0
Chebyshev	1	2	k_1
Cauer (Elliptic)	8	4	$2k_4$

The value of n_{min} might not be an odd number. Accordingly, we can choose the smallest odd number N as the design filter order satisfying

$$N \geq n_{min} \tag{3.10}$$

The next thing we should do is to calculate the coefficients of the specified filter type. We split it into two steps: the first one is to ensure the design range, the second one is to compute the

coefficients. Now we will show the calculations for the Butterworth, Chebyshev, and Cauer (Elliptic) lowpass filter types, respectively.

3.3 The calculation of the coefficients for Butterworth response

Butterworth filters are "maximally flat " in the passband, and they have the most linear phase response in the passband compared with other types of the filters, such as Chebyshev, inverse Chebyshev, and Cauer (Elliptic) responses. We will show their characteristics combined with design examples in chapter 4.

3.3.1 Determination of the design boundary

Define

$$k_p = \frac{\sqrt[N]{\varepsilon_p^2 - \varphi_p^2}}{\sqrt[N]{\varepsilon_p^2 + \varphi_p^2}} \tag{3.11}$$

$$k_s = \frac{\sqrt[N]{\varepsilon_s^2 - \varphi_s^2}}{\sqrt[N]{\varepsilon_s^2 + \varphi_s^2}} \tag{3.12}$$

where k_p, k_s are the auxiliary parameters.

Now we can choose an arbitrary value for "r" which satisfies the inequalities

$$k_s \leq r \leq k_p \tag{3.13}$$

If $k_s \leq 0$ and $0 \leq k_p$ and $r = 0$ is chosen, it will lead to the bireciprocal case, which we won't discuss in advance.

3.3.2 Determination of the coefficients

When we choose the value in an appropriate manner, the multiplier values (coefficients) can be obtained as follows:

$$\gamma_0 = \frac{1 + r - \sqrt{1 - r^2}}{1 + r + \sqrt{1 - r^2}} \tag{3.14}$$

$$\gamma_{2i-1} = \frac{\sqrt{1 - r^2} \cdot \cos(\pi i/N) - 1}{\sqrt{1 - r^2} \cdot \cos(\pi i/N) + 1} \tag{3.15}$$

$$\gamma_{2i} = r \tag{3.16}$$

3.4 The calculation of coefficients for Chebyshev response

A Chebyshev filter contains a passband ripple and the phase response in the passband is less linear than the Butterworth filter. However, the magnitude response drops off more sharply than the Butterworth filter, i.e., typically, a lower filter order is required. We will show its characteristics combined with design examples later.

3.4.1 Determination of the design boundary

Define:

$$\varepsilon_{p\,min} = \frac{2\varepsilon_s}{k_1^N} \tag{3.17}$$

where $\varepsilon_{p\,min}$ is the smallest possible value of the passband ripple factor;

k_1 is given by (3.7).

We can choose an arbitrary value ε_p^* as an actual passband ripple factor which satisfies

$$\varepsilon_{p\,min} \leq \varepsilon_p^* \leq \varepsilon_p \tag{3.18}$$

As we know, the whole design margin will be allocated to the passband if $\varepsilon_p^* = \varepsilon_{p\,min}$ is selected. This will result in the exact bound with given specifications under the circumstances. We use this condition ($\varepsilon_p^* = \varepsilon_{p\,min}$) in our considerations.

Again, we define additional auxiliary parameters as follows:

$$w = \sqrt{\frac{1}{\varepsilon_p^*} + \sqrt{\frac{1}{\varepsilon_p^*} + 1}} \tag{3.19}$$

and

$$r = \left(w - \frac{1}{w} \right) \cdot \varphi_p \tag{3.20}$$

3.4.2 Determination of the coefficients

$$\gamma_0 = \frac{2 - r}{2 + r} \tag{3.21}$$

$$\gamma_{2i-1} = \frac{A_i - B_i - 1}{A_i + B_i + 1} \tag{3.22}$$

$$\gamma_{2i} = \frac{1 - B_i}{1 + B_i} \tag{3.23}$$

where $A_i = r \cdot \cos(\pi i / N)$ (3.24)

$B_i = \{ w^2 - 1/w^2 - 2 \cdot \cos(2\pi i / N) \}$ (3.25)

with $i = 1, 2, \ldots, (N-1)/2$

3.5 The calculation of the coefficients for Cauer response

The Cauer (elliptic) filter contains a ripple in the passband and stopband. The phase response in the passband is the most non-linear of the common filter types. However, it contains the sharpest drop-off. Therefore, typically a lower order filter is required to meet the specifications.

3.5.1 Determination of the design boundary

Define

$$r_0 = \sqrt{\varepsilon_s/\varepsilon_p} \tag{3.26}$$

and furthermore

$$r_{i+1} = r_i^2 + \sqrt{r_i^2 - 1} \qquad \text{for } i = 0,1 \tag{3.27}$$

$$r_4 = \tfrac{1}{2}\left(\sqrt[N]{r_i^2} \right)^4 \tag{3.28}$$

$$x_{i-1} = \sqrt{\tfrac{1}{2}\left(x_i + 1/x_i \right)} \qquad \text{for } i = 4, 3, 2, 1 \tag{3.29}$$

where r_0, r_{i+1}, r_4, and x_{i-1} are the auxiliary parameters.

Now we define the minimum value of the lower stopband edge frequency, $f_{s\,min}$, in terms of x_o

$$f_{s\,min} = F/\pi \cdot \arctan(\varphi_p x_0^2) \tag{3.30}$$

From (3.28), we can choose the actual value of the lower stop edge frequency, f_p^*, which satisfies the inequalities

$$f_{p\,min} \leq f_p^* \leq f_p \tag{3.31}$$

As we know, $f_p^* = f_{p\,min}$ would allocate the whole design margin to the transition band, or to the passband and to the stopband if $f_p^* = f_p$ chosen.

According to (3.3) and (3.29), we can get

$$\varphi_s^* = \tan(\pi f_s^*/F) \tag{3.32}$$

then

$$q_0 = \sqrt{\varphi_s^* / \varphi_p} \tag{3.33}$$

furthermore

$$q_{i+1} = q_i^2 + \sqrt{q_i^4 - 1} \qquad \text{for } i = 0, 1, 2, 3 \tag{3.34}$$

$$m_4 = \tfrac{1}{2}\left(\sqrt{2q_4}\right)^N \tag{3.35}$$

$$m_{i-1} = \sqrt{\tfrac{1}{2}\left(m_i + 1/m_i\right)} \qquad \text{for } i = 4, 3, 2, 1 \tag{3.36}$$

From the above, we can finally compute m_0, and then we have

$$\varepsilon_{p\,min} = \frac{\varepsilon_s}{m_0^2} \tag{3.37}$$

where $\varepsilon_{p\,min}$ is the smallest possible value of the passband ripple factor.

Similarly, we could choose the actual value ε_p^* which satisfies the inequalities

$$\varepsilon_{p\,min} \le \varepsilon_p^* \le \varepsilon_p \tag{3.38}$$

As we discussed before, this will allocate the entire design margin to the passband if $\varepsilon_p^* = \varepsilon_{pmin}$, and to the entire stopband if $\varepsilon_p^* = \varepsilon_p$ is selected.

3.5.2 Determination of the Coefficients

Define the auxiliary parameters as follows:

$$g_1 = \frac{1}{\varepsilon_p^*} + \sqrt{\frac{1}{\varepsilon_p^{*2}} + 1} \tag{3.39}$$

$$g_{i+1} = m_i g_i^2 + \sqrt{(m_i g_i)^2 - 1} \qquad \text{for } i = 1, 2 \tag{3.40}$$

$$w_5 = \sqrt[N]{\frac{m_3}{g_3} + \sqrt{\left(\frac{m_3}{g_3}\right)^2 + 1}} \tag{3.41}$$

$$w_{i-1} = \frac{1}{2q_{i-1}}\left(w_i - \frac{1}{w_i}\right) \qquad \text{for } i = 5, 4, 3, 2, 1 \tag{3.42}$$

Then the coefficient value of the multiplier, γ_0, can be calculated by

$$\gamma_0 = \frac{1 + w_0 q_0 \varphi_0}{1 - w_0 q_0 \varphi_0} \tag{3.43}$$

Now we define auxiliary parameters for the calculations of the other multiplier values as follows:

$$C_{4,i} = \frac{q4}{\sin(i\pi/N)} \qquad \text{for } i = 1, 2, \ldots, (N-1)/2. \tag{3.44}$$

$$C_{j-1,i} = \frac{1}{2q_{i-1}} \left(C_{j,i} - \frac{1}{C_{j,i}} \right) \qquad \text{for } i = 4, 3, 2, 1, \tag{3.45}$$

$$y_i = 1 / C_{0,i} \tag{3.46}$$

$$B_i = \frac{w_0^2 + y_i^2}{1 + (w_0 y_i)^2} \cdot (q_0 \varphi_p)^2 \tag{3.47}$$

$$A_i = \frac{2 w_0 q_0 \varphi_p}{1 + (w_0 y_i)^2} \cdot \sqrt{1 - \left(q_0^2 + \frac{1}{q_0^2} - y_i^2\right) y_i^2} \tag{3.48}$$

We can get the rest of the multiplier values for $i = 1, 2, \ldots, (N-1)/2$ using

$$\gamma_{2i-1} = \frac{A_i - B_i - 1}{A_i + B_i + 1} \tag{3.49}$$

$$\gamma_{2i} = \frac{1 - B_i}{1 + B_i} \tag{3.50}$$

Chapter 4

The Design Procedures with An Example

In this chapter, the whole design procedures with an example are described step by step. At first, an overview of the common design procedures via a flowchart is present. Secondly, we take a 9th order Cauer (Elliptic) lowpass filter design as an example with the detailed procedures. With the general specifications, the source codes and data files in different directories are specialized for different purposes, such as calculating coefficients, generating a test vector, viewing the characteristics of the filter, loading input and output data for Matlab simulation and define EVM implementation, comparing the difference between simulation and implementation. After the design is completed, a preliminary analysis is present. Based on all types of the filter designs, the analysis of the results in detail will be provided in chapter 5. There are a lot of files (source codes and data files) that need to be dealt, Section 4.13 describes the file management and naming convention. The same procedures could apply to different types of filter design, the results of a 7th order Butterworth lowpass filter design and a 5th order Chebyshev lowpass filter design are present in Appendix A and Appendix B as references.

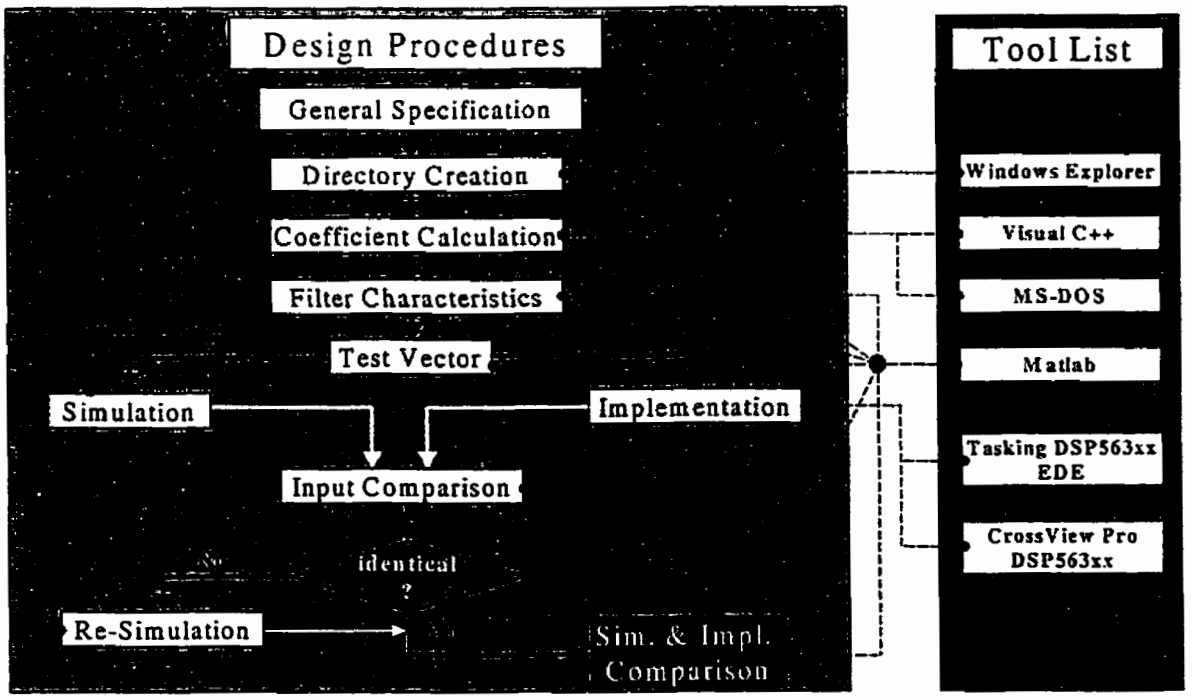


Figure 4.1 The flowchart of design procedures and tools

4.1 General Specifications with an example of a 9th order Cauer lowpass filter

According to the definitions from Figure 3.1, we define the general specifications of a 9th order Cauer lowpass filter as follows:

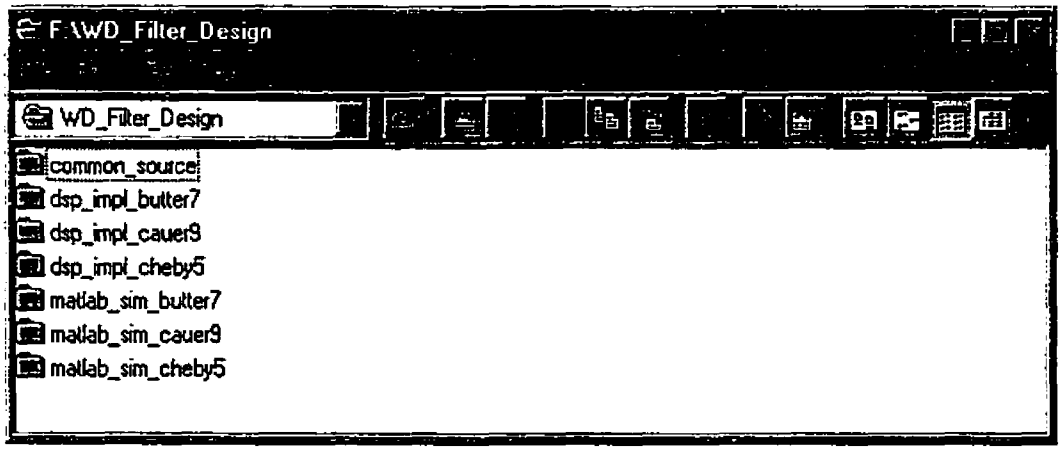
Example: a 9th order Cauer (Elliptic) WD lowpass filter design

General Specifications:

- Passband Frequency: = 3500 Hz
- Passband Attenuation: = 0.3 dB
- Stopband Frequency: = 4000 Hz
- Stopband Attenuation: = 80 dB
- Sampling Frequency: = 16000 Hz

4.2 Directory creation to manage the source codes and data files

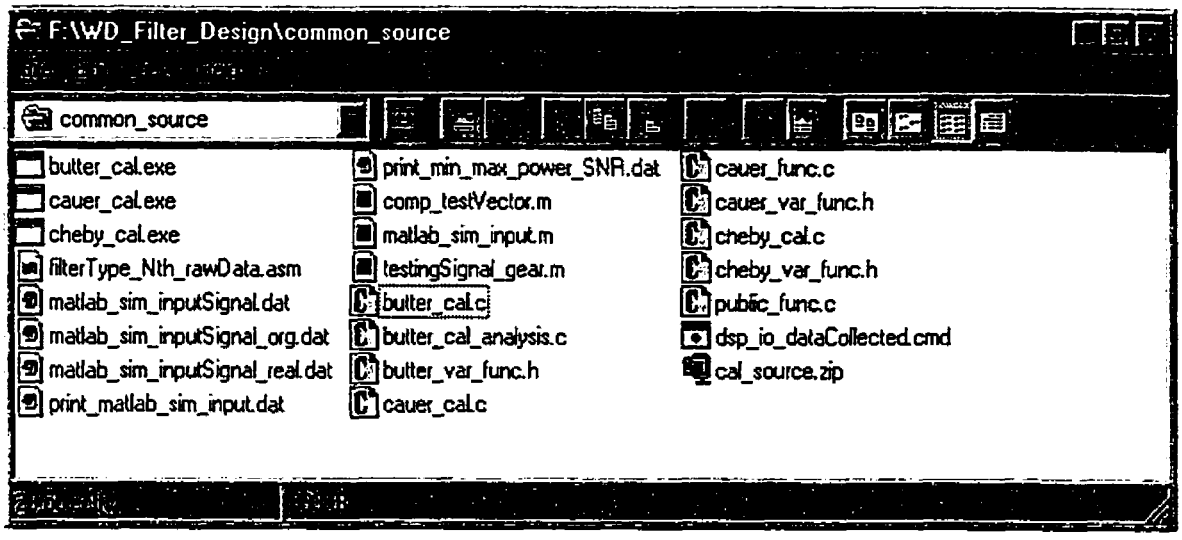
1. Create 1 directory, called **WD_Filter_Design,** and 7 subdirectories under this directory as follows:



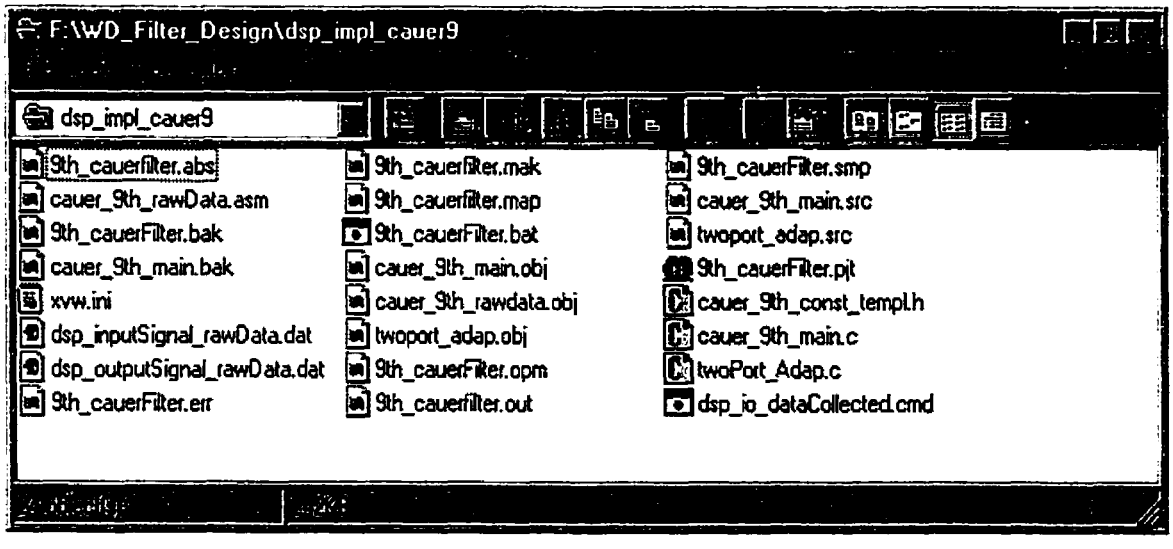
The above 7 directories contain the sources codes and data files with different purposes.

2. At **F:\WD_Filter_Design\common_source,** we put source codes with the filter coefficient calculations, test vector generator and the program to compare the input signal with different platforms. After compiling source codes with MS-Visual C++, we can get executables, such as ***cauer_cal.exe***, for coefficient calculations, and run

testingVetctor_gear.m to obtain data files, which contain test vector data. This directory is similar to the following:



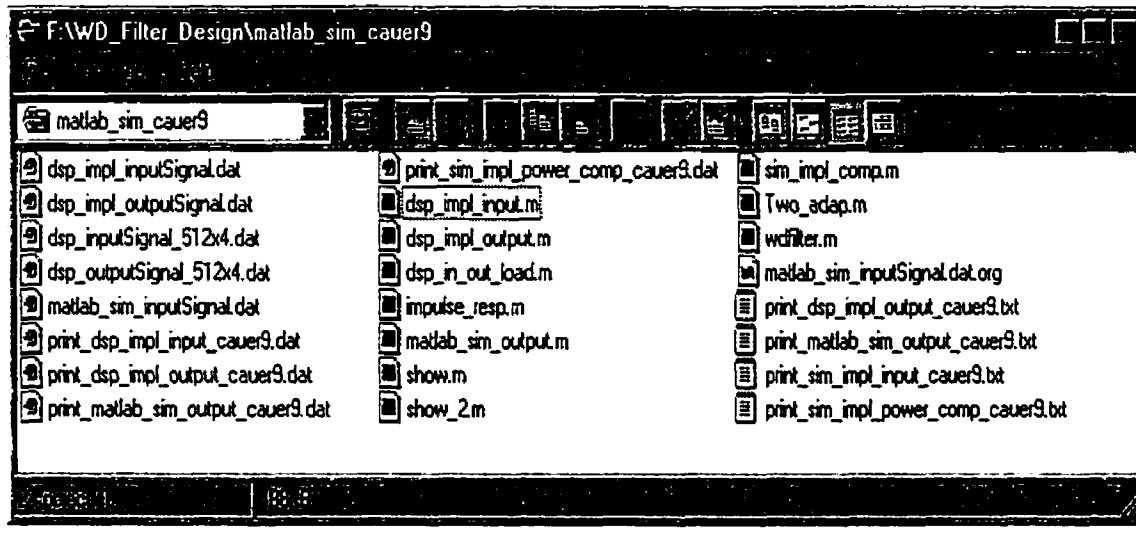
3. At **F:\WD_Filter_Design\dsp_impl_cauer9,** we put source codes with the DSP implementation required. After compilation by TASKING DSP563xx EDE Source Compiler, we will get the executable, ***9th_cauerfilter.abs***, which can be downloaded to the DSP56307EVM board. By using the debugging tool, CrossView Pro DSP56xxx Debugger, we can collect input and output data from the designed filter. The whole set of files in this directory is similar to the following:



When compiling source codes, some files in this directory, which are for debugging only, are automatically generated. The directories **F:\WD_Filter_Design\dsp_impl_cheby5** and

F:\WD_Filter_Design\dsp_impl_butter7, which are focused on the implementations for a 5th order Chebyshev filter and a 7th order Butterworth filter on an DSP56307EVM board are for the same purposes as above, respectively.

4. At **F:\WD_Filter_Design\matlab_sim_cauer9,** we put Matlab source codes and the data files generated, which are for a 9th order Cauer filter analysis of the filter characterization, Matlab simulation, and detailed filter design verifications in both ways, i.e. simulation and EVM implementation; this directory is similar to the following:



The directories **F:\WD_Filter_Design\matlab_sim_cheby5&F:\WD_Filter_Design\matlab_sim_butter7** that are specialized on the design of a 5th order Chebyshev filters and a 7th order Butterworth filters are for the same purposes as above, respectively.

4.3 The compilation and execution of the source codes for the coefficient calculation

At this step, we compile the source codes of the coefficient calculations and execute the executable to obtain the coefficients that satisfy the general specifications:

- At **F:\WD_Filter_Design\common_source**, we load the source files:
  1. ***cauer_func.c***
  2. ***cauer_cal.c***
  3. ***cauer_var_func.h***
  4. ***public_func.c***

Next we use the MS-Visual C++ complier to generate the executable, called ***cauer_cal.exe***, and then run this executable with the MS-DOS window, it will give you the results of the coefficient calculation on the user interface.

Figure 4.2 User interface of the coefficients for the Cauer lowpass filter with the specifications

4.4 The characteristics of 9th order Cauer lowpass filter design

- At **F:\WD_Filter_Design\matlab_sim_cauer9,** we load the Matlab's source files:
  1. ***Two_adap.m***
  2. ***wdfilter.m***
  3. ***impulse_resp.m***

put the coefficients, which are calculated by ***cauer_cal.exe***, into **wdfilter.m,** and run **impulse_resp.m** to check the impulse response characteristics of the 9th order Cauer (Elliptic) WD filer. We also need to declare the coefficients to the implementation source code **cauer_9th_const_templ.h** at **F:\WD_Filter_Design\dsp_impl_cauer9.**

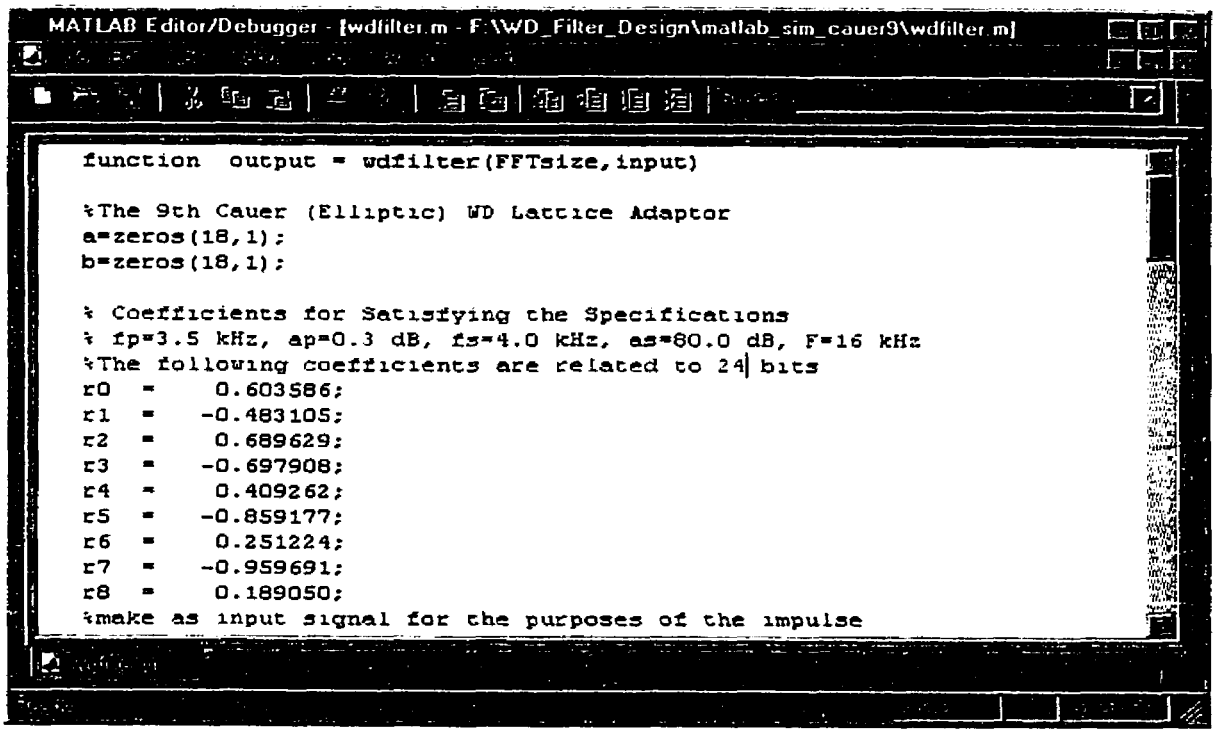


Figure 4.3 The coefficients in **wdfilter.m** for the 9th order Cauer (Elliptic) WD lowpass filter

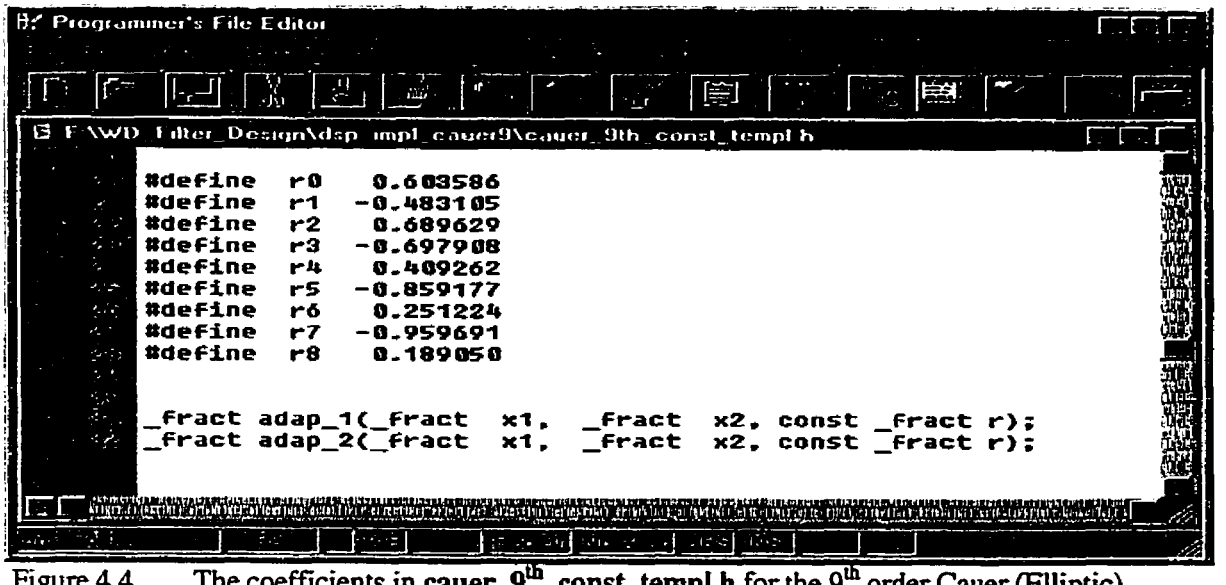


Figure 4.4 The coefficients in **cauer_9th_const_templ.h** for the 9th order Cauer (Elliptic) WD lowpass filter

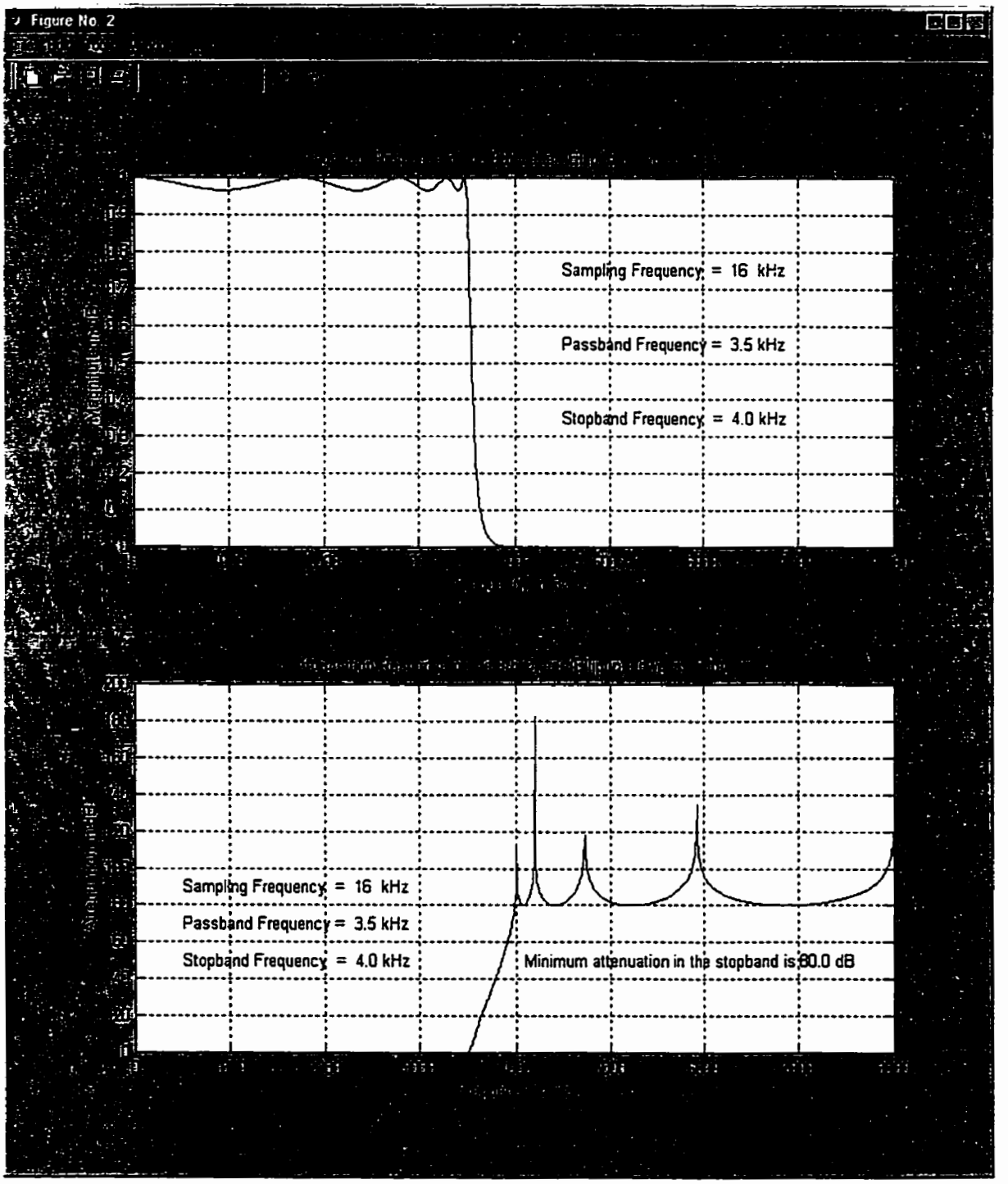


Figure 4.5 The magnitude and attenuation response of the 9th order Cauer (Elliptic) WD lowpass filter

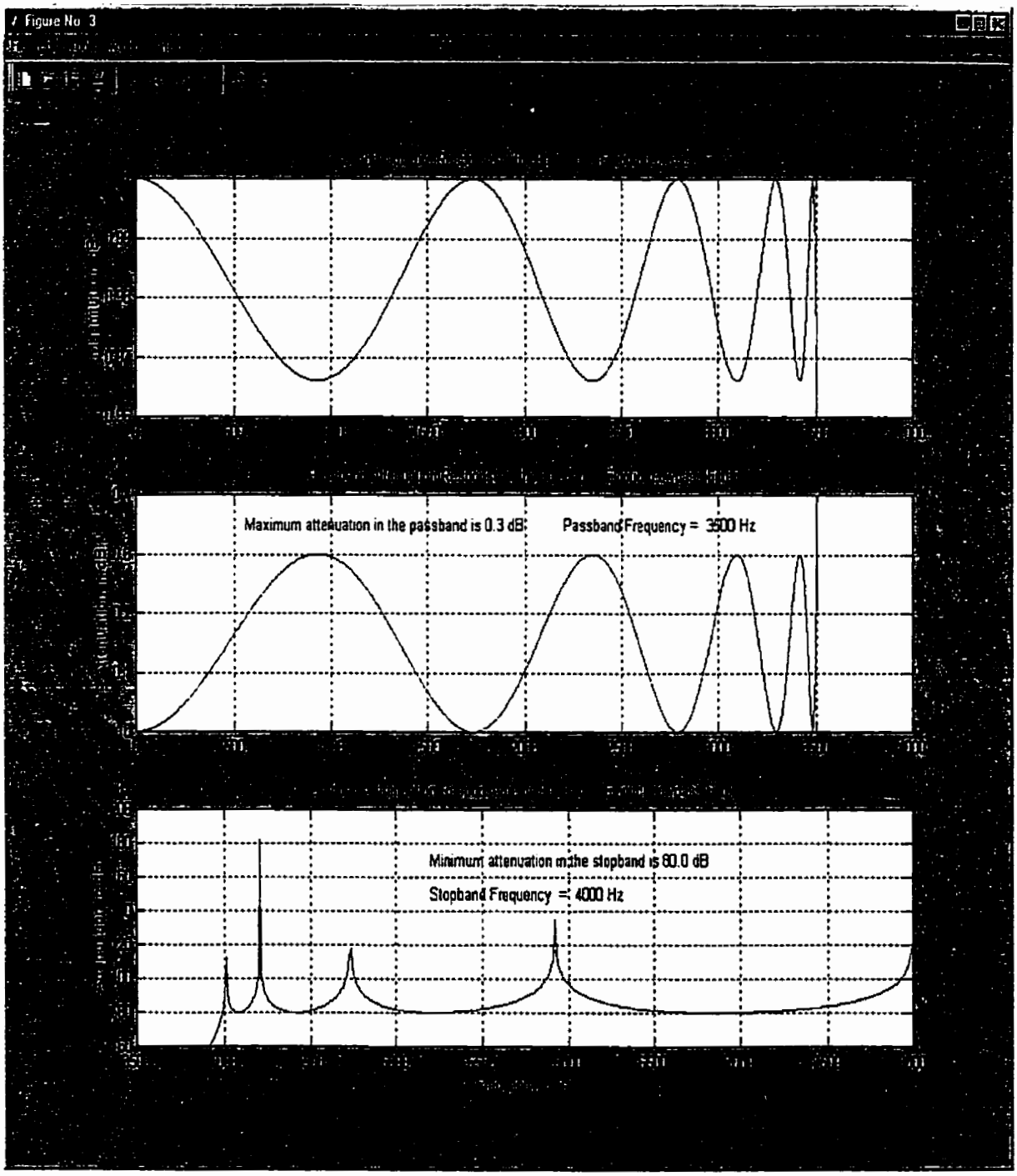


Figure 4.6 The detailed response in the passband and stopband of the 9th order Cauer (Elliptic) WD lowpass filter

4.5 Test vector generation for simulation and implementation

To verify the filter for simulation and implementation, we need to generate a test vector and pass it through different types of filters with the Matlab simulator and DSP56307EVM, respectively.

Test vector description:

- fundamental frequency = 500 Hz
- sampling frequency = 16000 Hz
- test vector size = 2k, i.e. 2048 samples
- low frequency component: $0.070\sin(\omega_0 t) + 0.045\sin(2.2\omega_0 t)$
- high frequency component: $0.041\cos(12.5\omega_0 t) + 0.062\cos(14.2\omega_0 t)$
- noise frequency component: 0.024*random(highFrequencyComponent + noiseComponent)

where ω_0 = fundamentalFrequency / samplingFrequency, t is time

Input Signal = lowFequencyComponent+highFrequencyComponent+ randomFrequencyComponent

Test vector generation:

At **F:\WD_Filter_Design\common_source**, we run **testingSignal_gear.m** and plot out the figures for the input signal in the time and frequency domains for Matlab simulation and DSP56307EVM implementation; this script can also generate data files:

1. ***matlab_sim_inputSignal.dat,***
2. ***print_matlab_sim_input.dat,***
3. ***print_min_max_power_SNR.dat.,***

The file ***print_min_max_power_SNR.dat*** can capture the maximum and minimum amplitudes, which should be within (-1.0, +1.0); this specification is only for the DSP implementation, real input power, noise input power, and signal-to-noise-ratio (SNR) in dB. The data file, ***matlab_sim_inputSignal.dat***, is to collect input signal data in 2048x1 format, which will be applied for the simulation and DSP implementation of 3 different types of filters. The reason we need to collect the data file is that the noise component is a random one. When we run this script at different times, we will get different input data. So the benefit of this data file is to guarantee passing the same vector through simulator and DSP EVM. Also, it is necessary to copy it to the directory, **F:\WD_Filter_Design\matlab_sim_cauer9,**

for simulation, and copy it to the directory **F:\WD_Filter_Design\dsp_impl_cauer9**, for implementation. In this directory we change the data format and save it as the name, ***cauer_9th_rawData.asm*** due to constraints of the DSP tool. The data file, ***print_matlab_sim_input.dat***, is for print (512x4 format) purposes only.

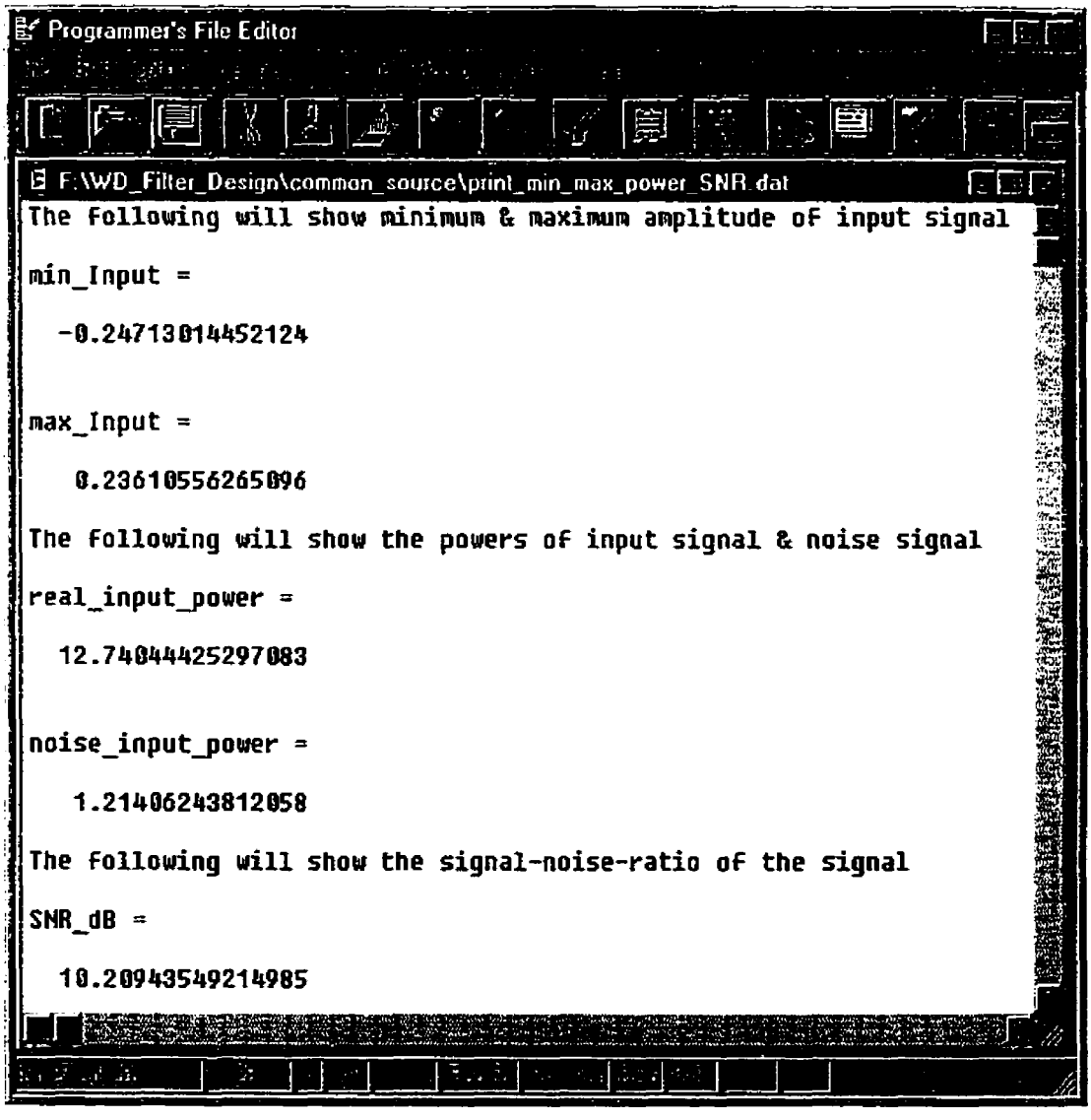


Figure 4.7 The characteristics of the test vector

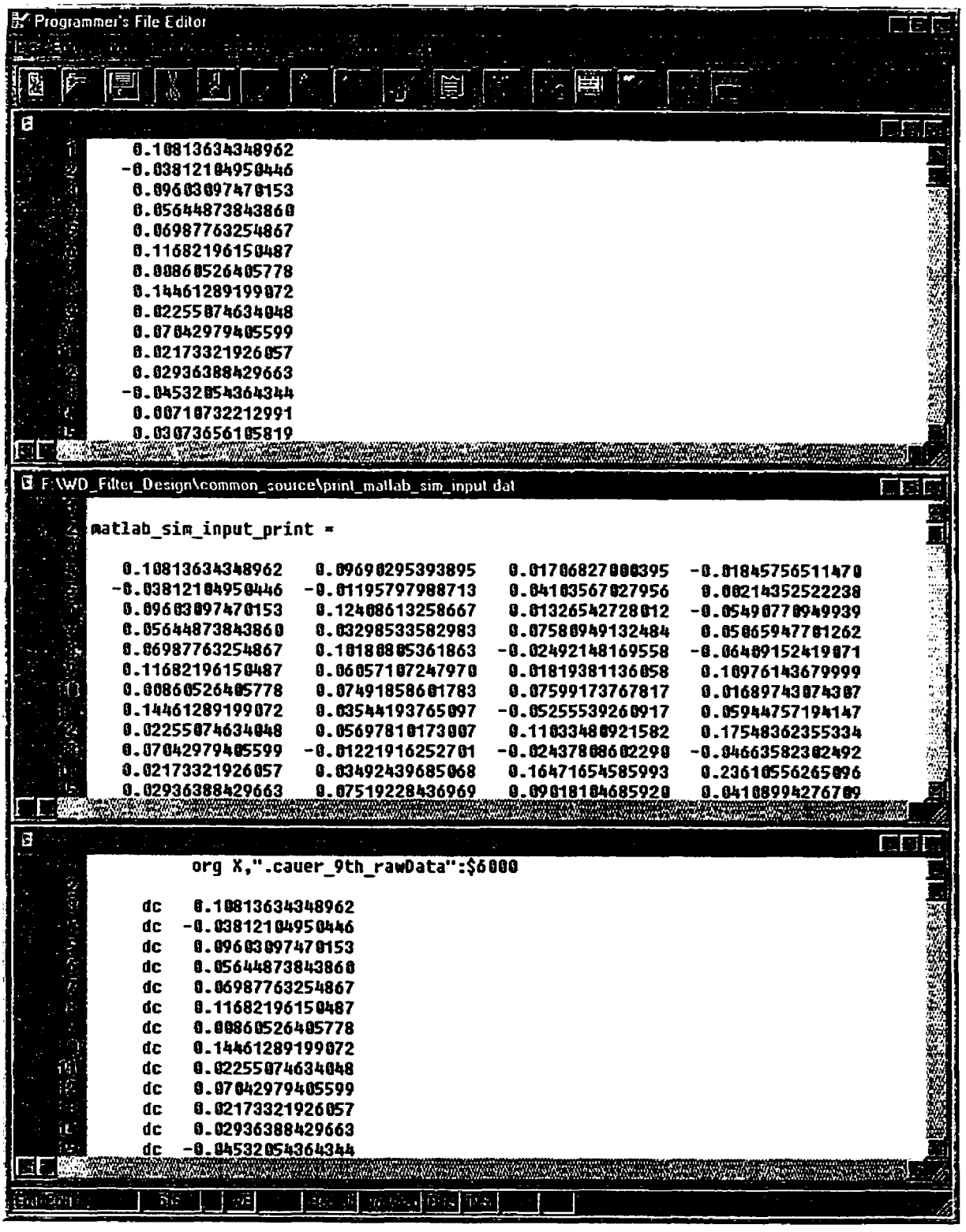


Figure 4.8 The test vector in different formats

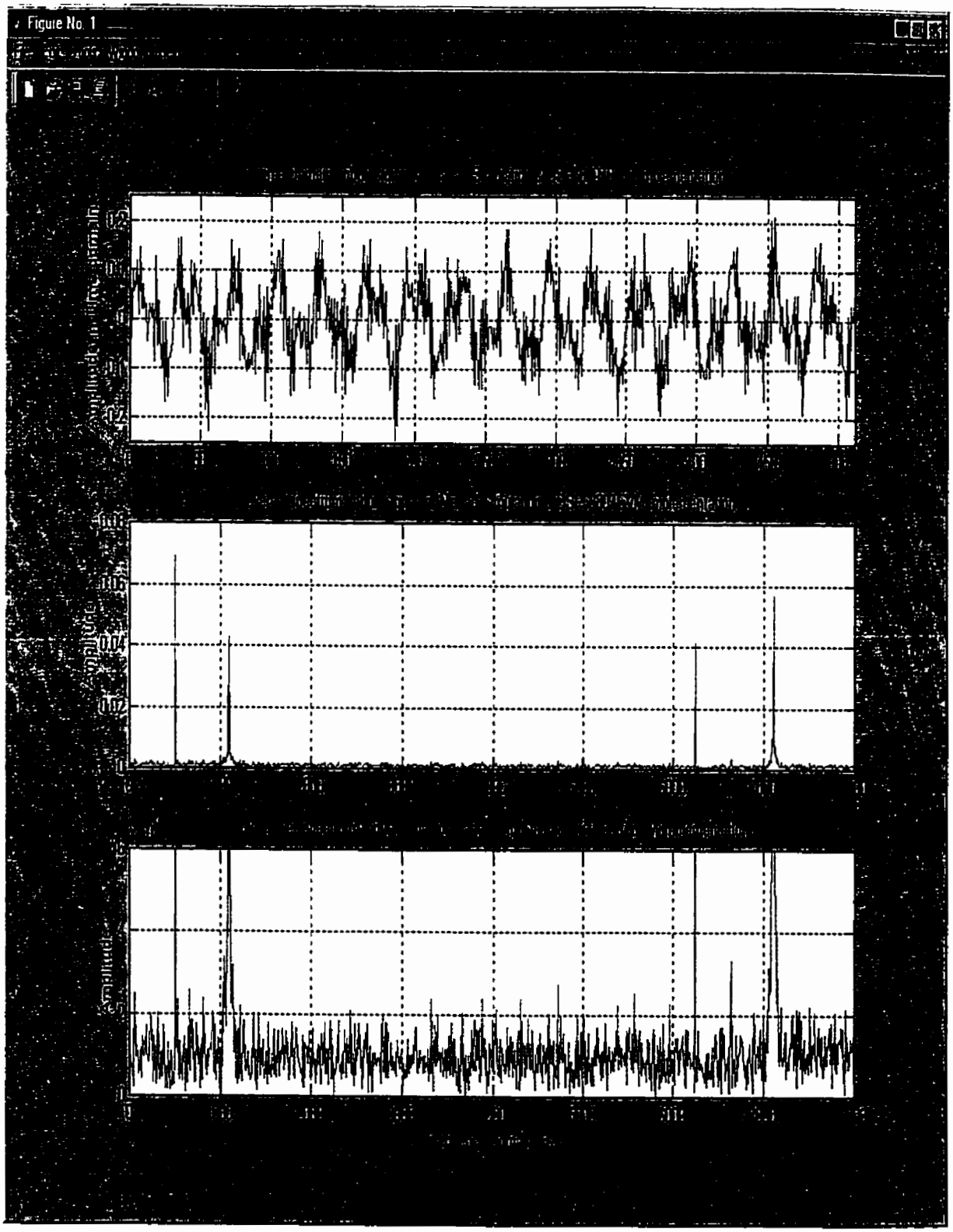


Figure 4.9 The test vector in the time domain and frequency domains

4.6 The simulation with a 9th order Cauer (Elliptic) lowpass filter

- At **F:\WD_Filter_Design\matlab_sim_cauer9,** we can run **matlab_sim_output.m**; it will show the output signal in the time domain and frequency domains graphically by passing the test vector, ***matlab_sim_inputSignal.dat***, to the 9th order Cauer lowpass filter.

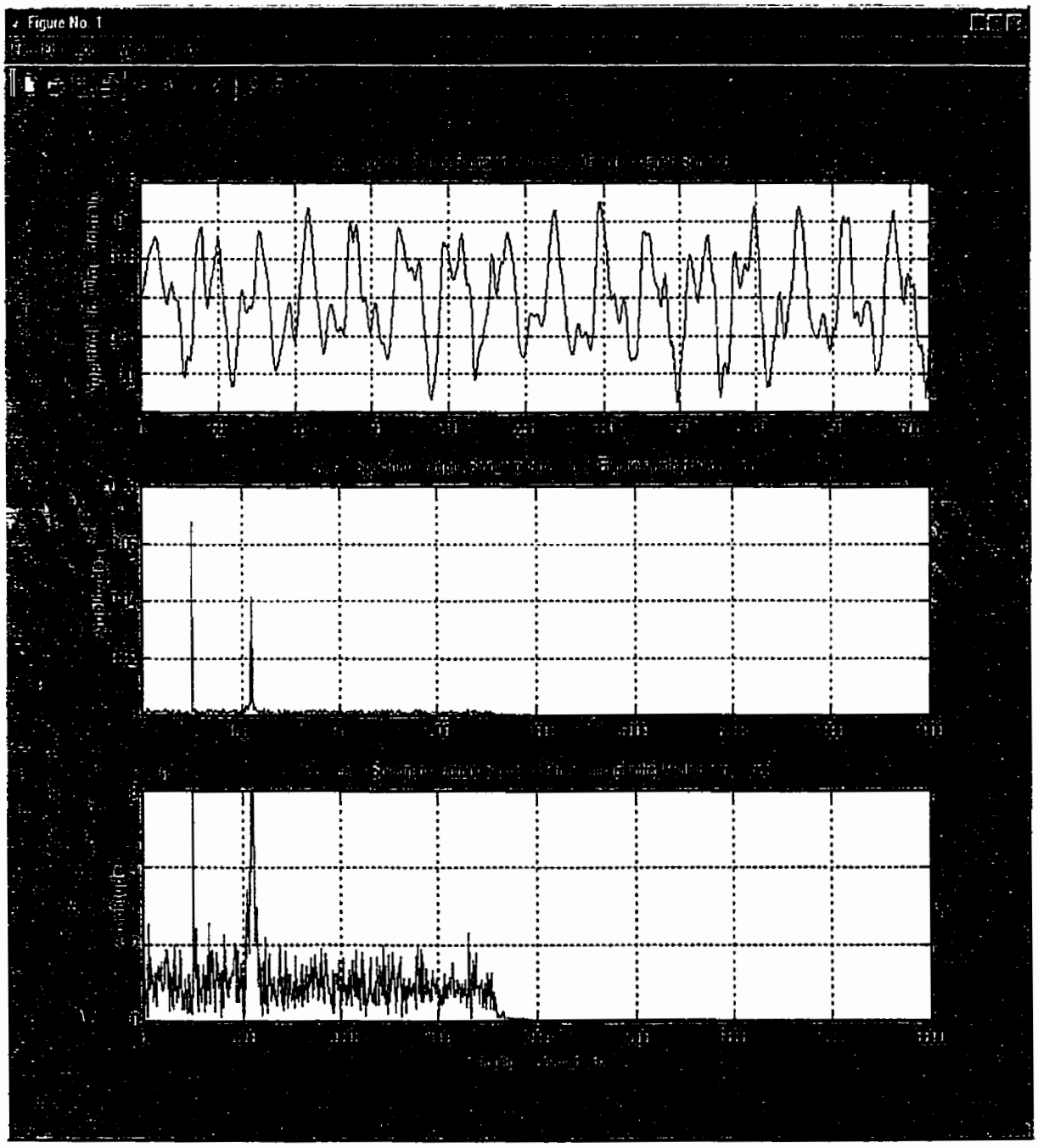


Figure 4.10 Simulation: Output of the 9th order Cauer (Elliptic) lowpass filter for the test vector

4.7 The compilation of the source codes with 9th order Cauer filter implementation

Using the TASKING EDE, we compile the source codes of the 9th order Cauer (Elliptic) lowpass filter to obtain the executable, ***9th_cauerfilter.abs***, which can be downloaded to the DSP56307EVM.

- At **F:\WD_Filter_Design\dsp_impl_cauer9,** we load the source files:
  1. ***cauer_9th_rawData.asm***
  2. **twoPort_Adap.c**
  3. **cauer_9th_const_templ.h**
  4. **cauer_9th_main.c**

Open TASKING DSP563xx EDE Source Compiler, create a project file, called ***9th_cauerFiler.pjt*** and add the above 4 source files to the project. Now we can link and build this project and generate an executable, called ***9th_cauerfilter.abs***, which can be downloaded to DSP56307EVM board for the implementation.

4.8 The implementation of the filter in DSP56307EVM with the test vector

- Open CrossView Pro DSP56xxx Debugger, clean up the memory and download ***9th_cauerfilter.abs*** to the EVM, and run using **GO** in the CrossView command window.
- To collect the input signal data and output signal data at the particular memory addresses and using decimal format, in the **cauer_9th_const_templ.h**, we pre-set the input data at **X:0x6000** and the output data at **X:0x8000**. To do this, we can perform a set of commands in the CrossView command window by using a script file, called **dsp_io_dataCollected.cmd**, as follows:

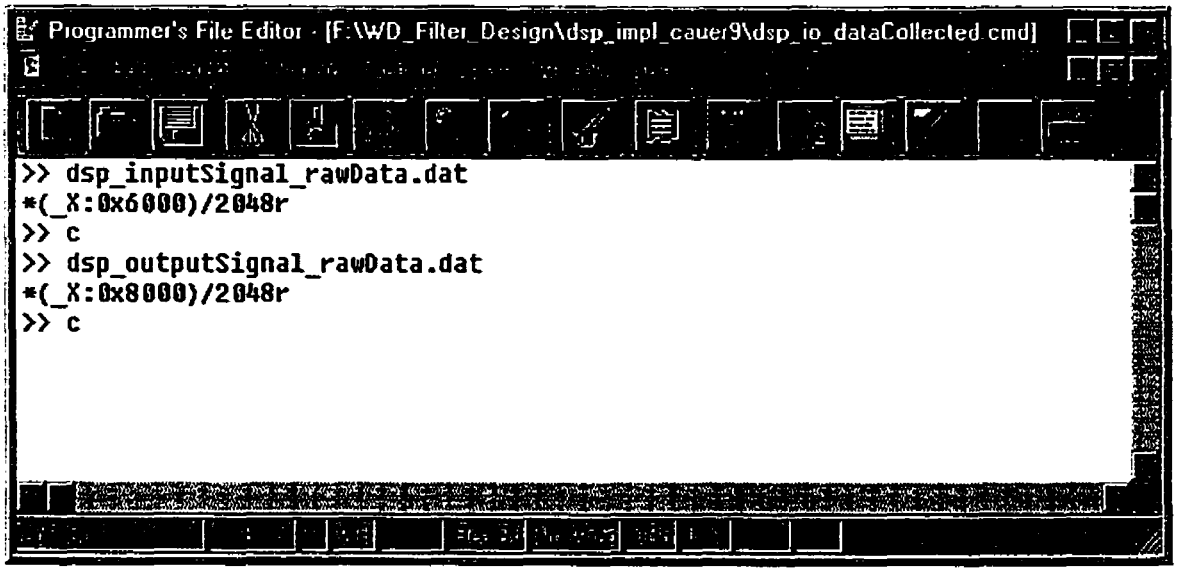


Figure 4.11 The script file, **dsp_io_dataCollected.cmd**, to collect input and output data

- In the command window of the CrossView Pro, send the command "**<dsp_io_dataCollected.cmd**" which will automatically collect input data from X memory X:0x6000 and save it to the file ***dsp_inputSignal_rawData.dat*** and gather output data from X memory X:0x8000 and save it to the file ***dsp_outputSignal_rawData.dat*** both data files have 512x4 format.
- Use MS-Excel to delete the symbols in ***dsp_inputSignal_rawData.dat*** and ***dsp_outputSignal_rawData.dat*** and save as different names, corresponding to ***dsp_inputSiganl_512x4.dat*** and ***dsp_outputSignal_512x4.dat***, copy them to the directory at **F:\WD_Filter_Design\matlab_sim_cauer9.**

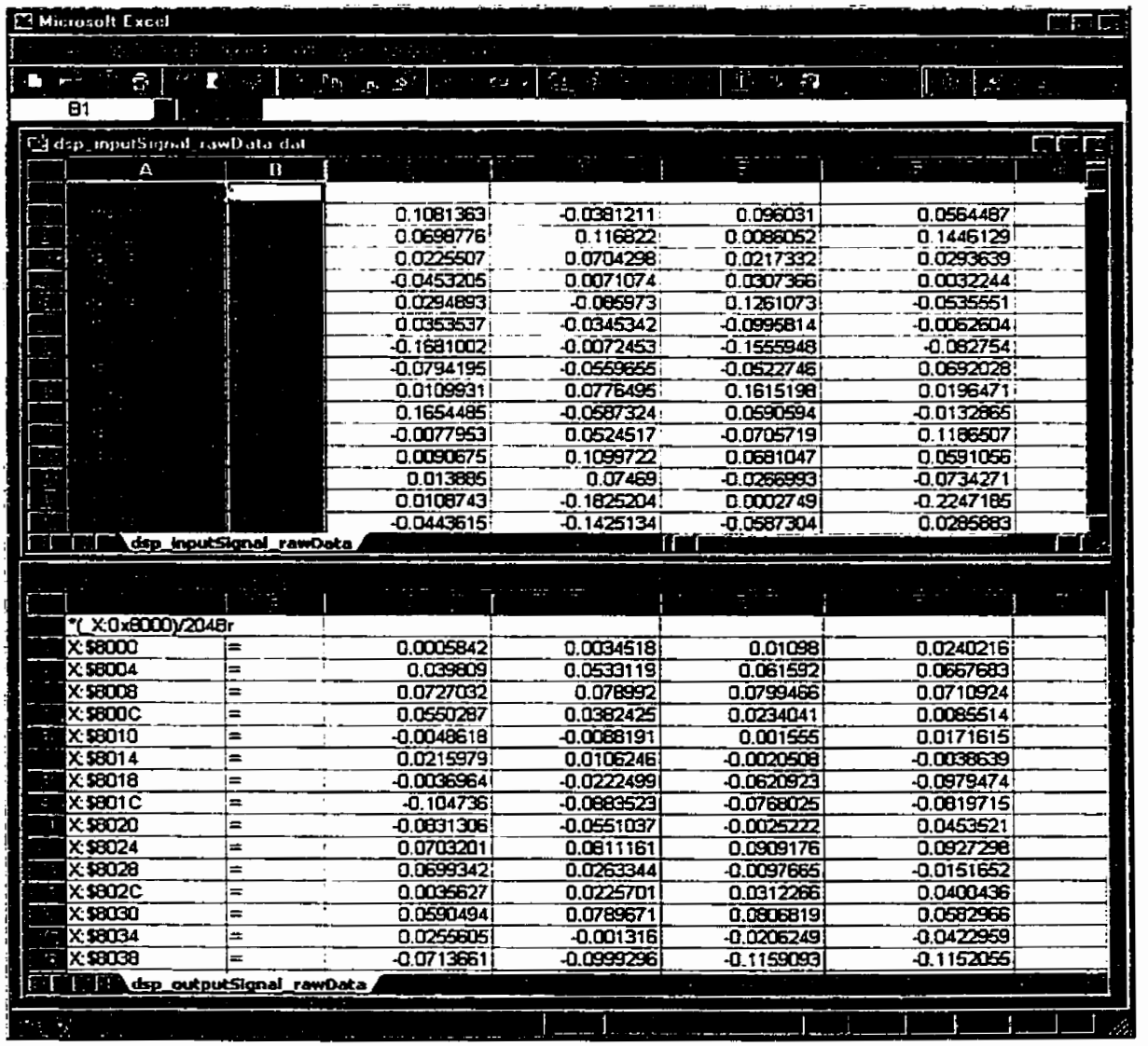


Figure 4.12 Data files: ***dsp_outputSignal_rawData.dat*** and ***dsp_inputSignal_rawData.dat***

- At **F:\WD_Filter_Design\matlab_sim_cauer9,** we could load source code, called **dsp_in_out_load.m,** and run it. This script will automatically generate 2 data files ***dsp_impl_inputSignal.dat*** and ***dsp_impl_outputSignal.dat*** with 2048x1 format. The 2 data files stand for real input and output data implemented on the DSP56307EVM board, respectively.

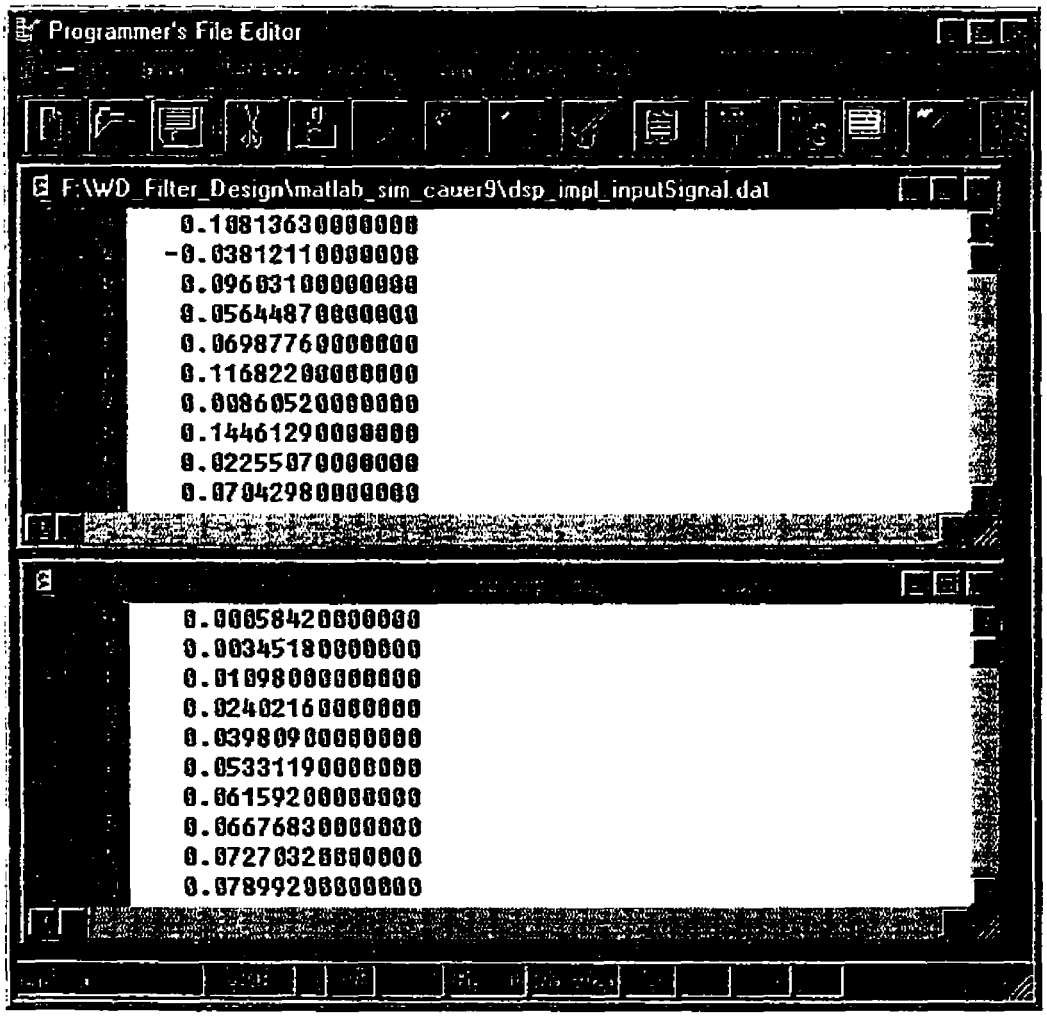


Figure 4.13 Data files: ***dsp_impl_inputSignal.dat*** and ***dsp_impl_outputSignal.dat***

- At **F:\WD_Filter_Design\matlab_sim_cauer9,** we load 2 source codes, **dsp_impl_input.m** and **dsp_impl_output.m,** and run them individually, which will show the real input signal and the output signal of the DSP implementation, graphically.

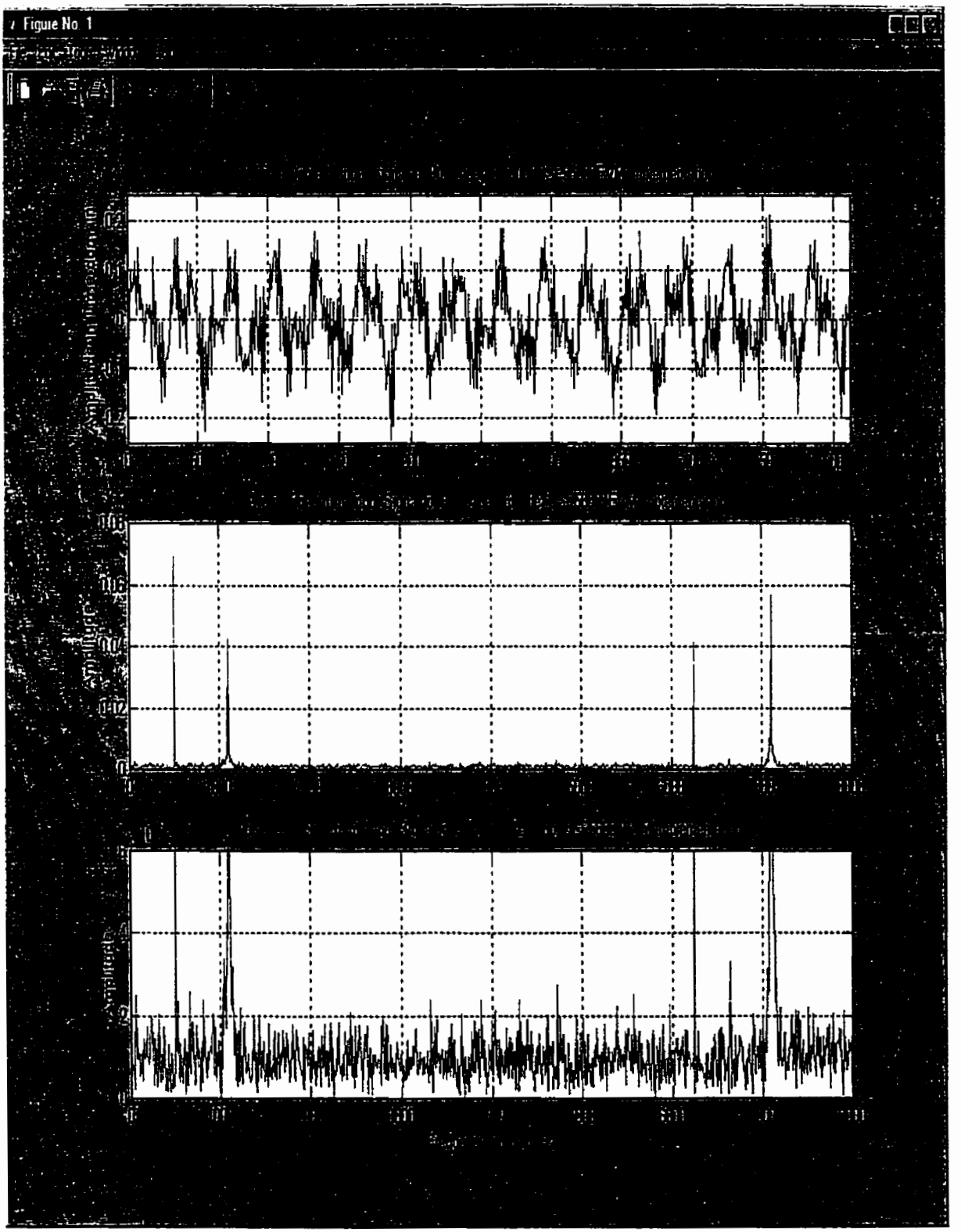


Figure 4.14 Input signal in the time and frequency domains with DSP56307EVM implementation

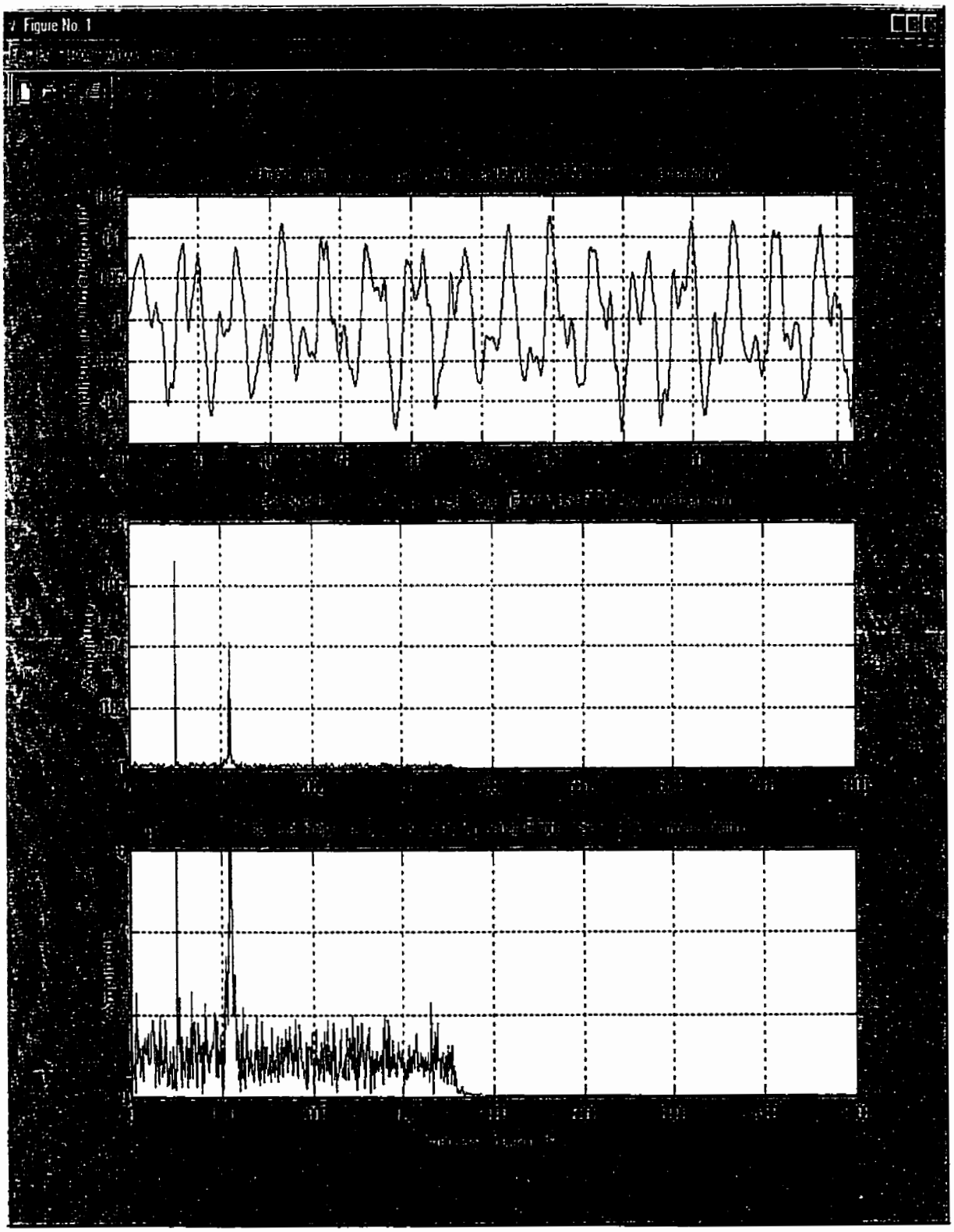


Figure 4.15 Output signal in the time and frequency domains with DSP56307EVM implementation

4.9 The comparison of the input of the test vector with different platforms

Due to the accuracy of the different platforms, such as Matlab platform or CrossView Pro DSP563xx, we need to verify whether or not the data of input signal is identical with different platforms. If not, a test vector from DSP implementation expects a re-simulation to guarantee there is no any contamination from input signal.

- Open a text editor, such as Programmer's File Editor (PFE), and compare the data file ***print_dsp_impl_inputSignal.dat*** from **F:\WD_Filter_Design\matlab_sim_cauer9** with the data file ***print_matlab_sim_input.dat*** from **F:\WD_Filter_Design\common_source.** And run the **comp_testVector.m** at the sub-directory of **common_source** to check the difference.

Figure 4.16 Input of the test vector at different platforms

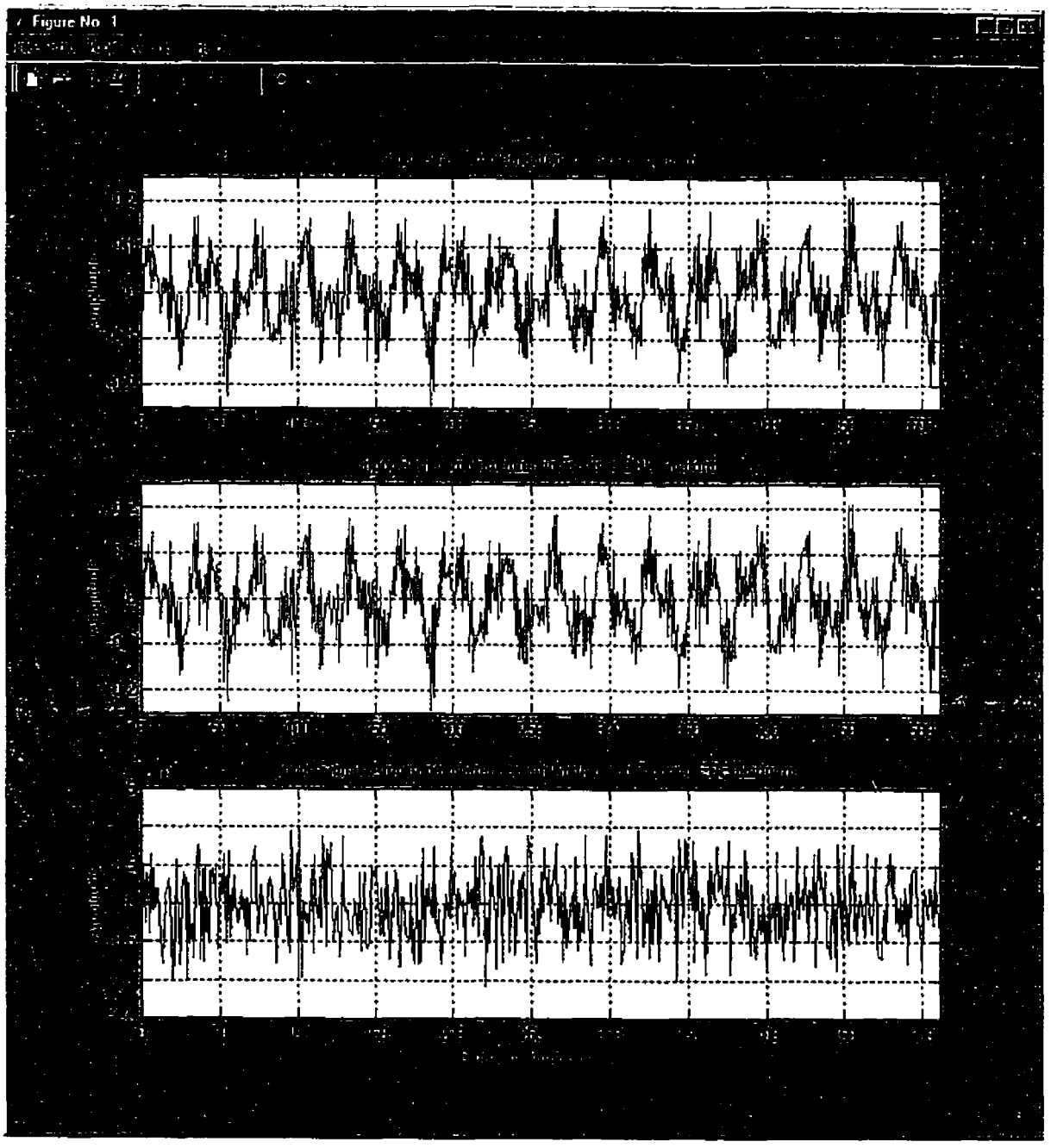


Figure 4.17 Input of the test vector at different platforms by running **comp_testVector.m**

- The input signal is different with different platforms. To minimize the difference to zero, we need to pass the test vector, ***dsp_impl_inputSignal.dat***, which is less accurate than ***matlab_sim_inputSignal.dat***, to the Matlab platform again in order to guarantee that we pass an identical test vector to both of platforms.

4.10 Re-Simulation by passing the test vector, dsp_impl_inputSignal

- Copy the data file, ***dsp_impl_inputSignal.dat***, overwrite the data file, named ***matlab_sim_inputSignal.dat***, and run **matlab_sim_output.m** again.

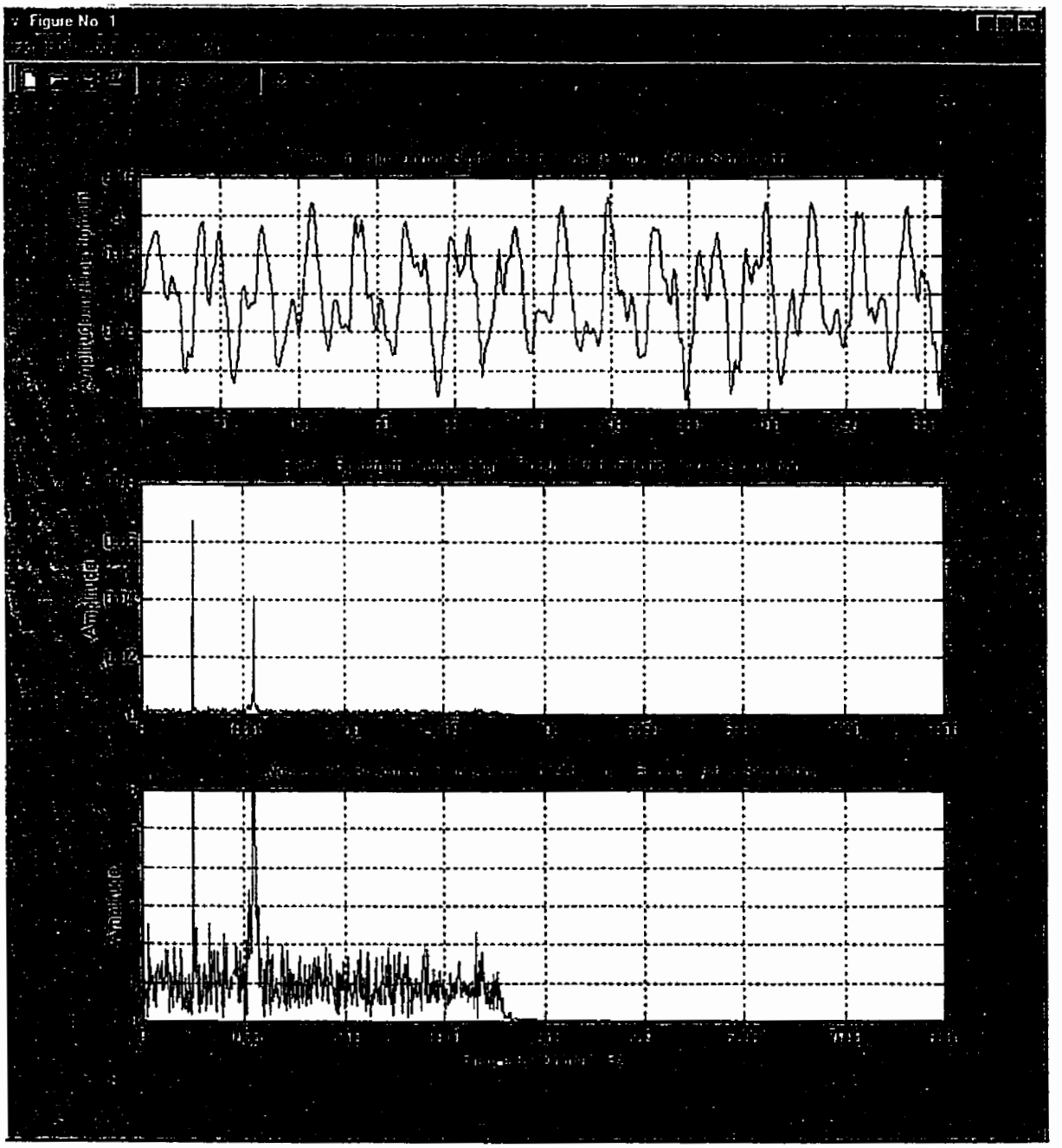


Figure 4.18 Implementation: Output of the 9th order Cauer (Elliptic) lowpass filter for the test vector, ***dsp_impl_inputSignal.dat***

4.11 Data comparison between Matlab simulation and DSP56307EVM implementation

- The final step is to load and run the source code, **dsp_sim_comp.m**. The program is used to compare the Matlab simulation with the real DSP56307EVM implementation in detail. The comparisons are as follows:
    1. Input signal in the time domain. (data files: ***print_sim_impl_input_cauer9.txt,*** figure 4.19).
    2. Output signal in the time domain (data files: ***print_matlab_sim_output_cauer9.txt*** and ***print_dsp_impl_output_cauer9.txt***; figure 4.20).
    3. Output signal in the frequency domain (figure 4.22).
    4. Output signal in the frequency domain with noise amplitude level (figure 4.23).
    5. The spectrum difference between the real and imaginary part of the output signal due to complex numbers in the frequency domain (figure 4.24).
    6. The detailed difference between the real and imaginary part of the output signal in the passband and stopband. The reason is the same as in 5 (figure 4.25).
    7. Power comparison in the time and frequency domains in terms of Parseval's relations for aperiodic signals (data file: ***print_sim_impl_power_comp_cauer9.txt***).

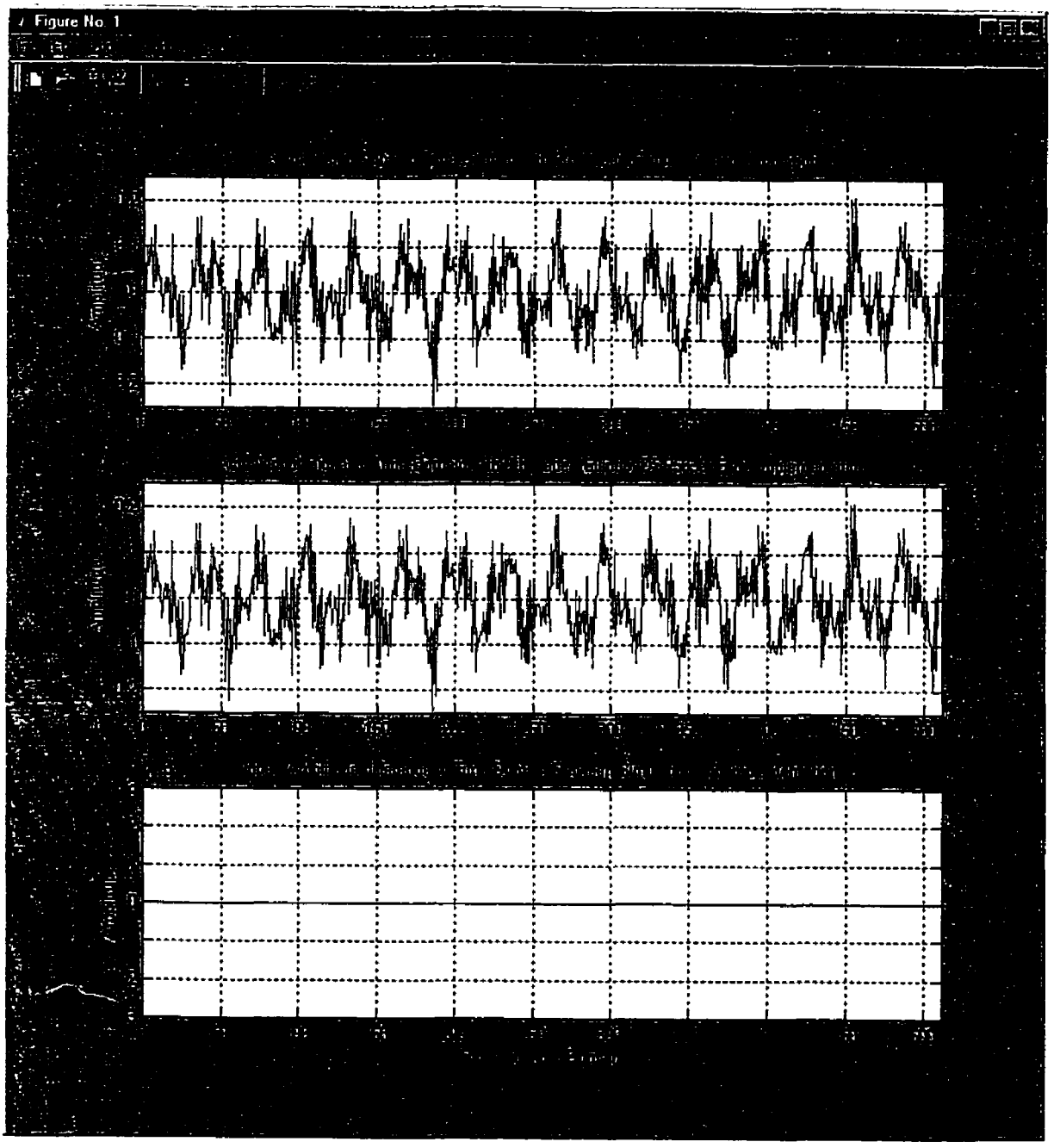


Figure 4.19 Input Signal in the time domain with simulation and implementation

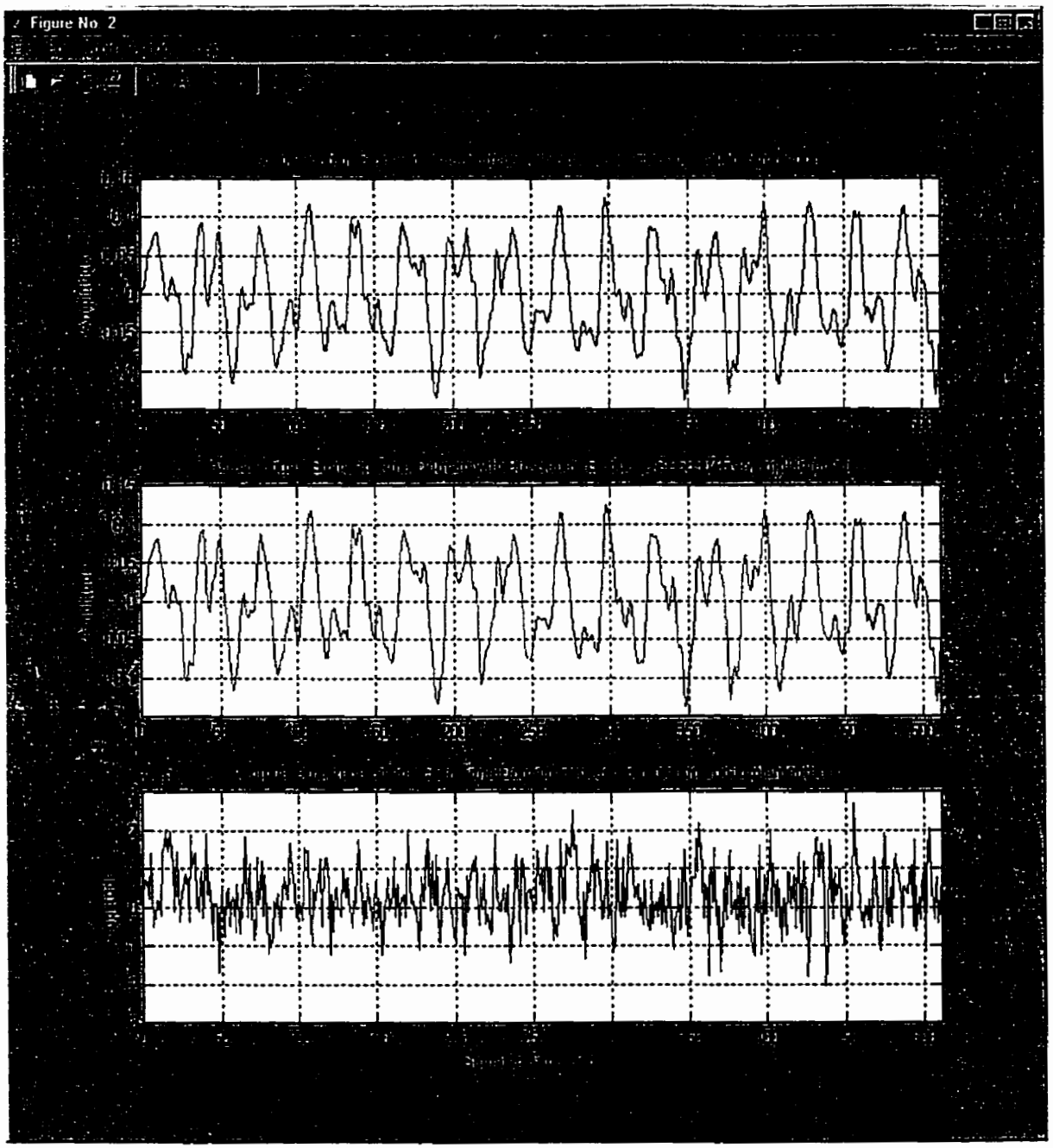


Figure 4.20 Output Signal in the time domain with simulation and implementation

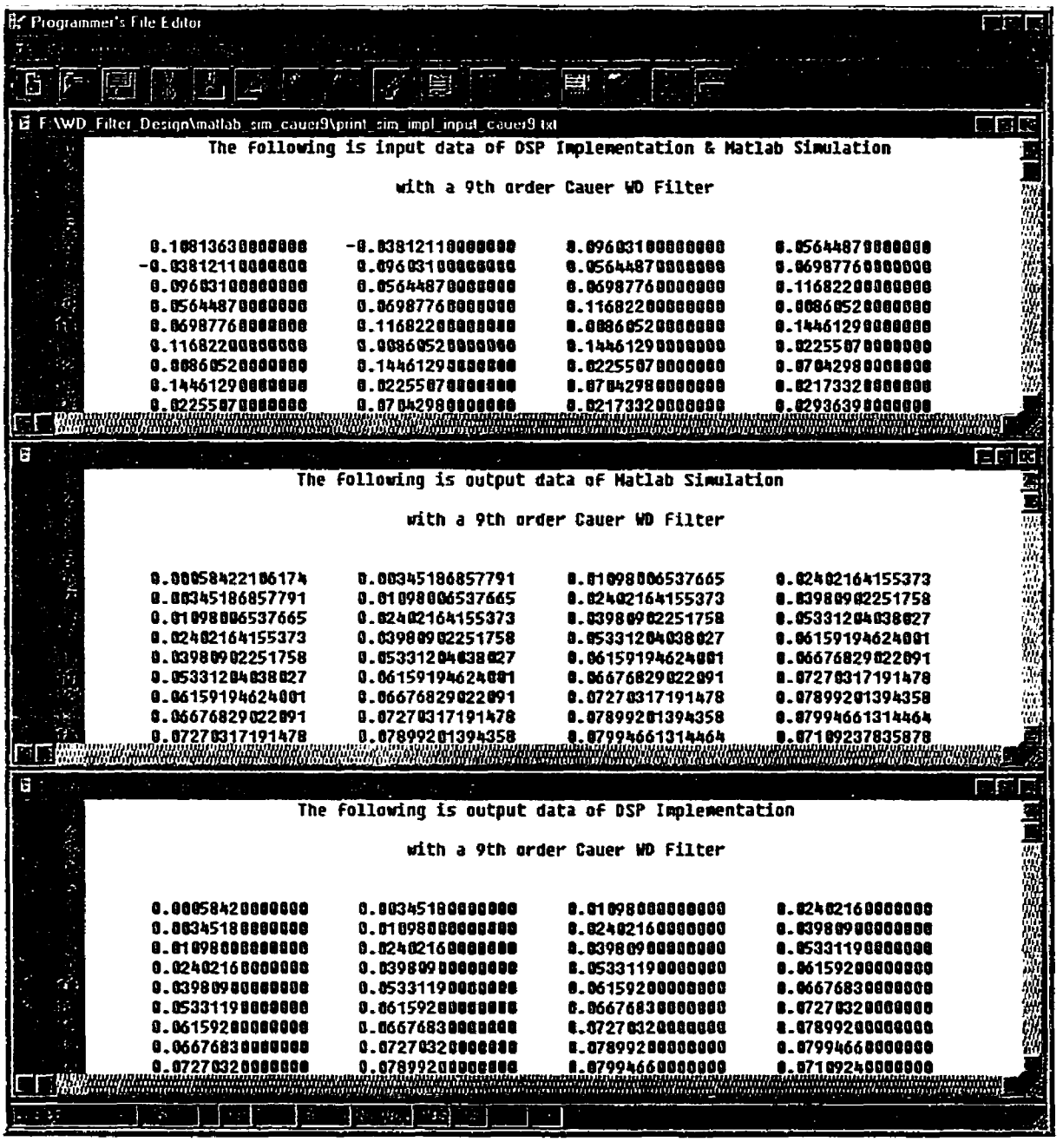


Figure 4.21 The data files of input and output with simulation and implementation

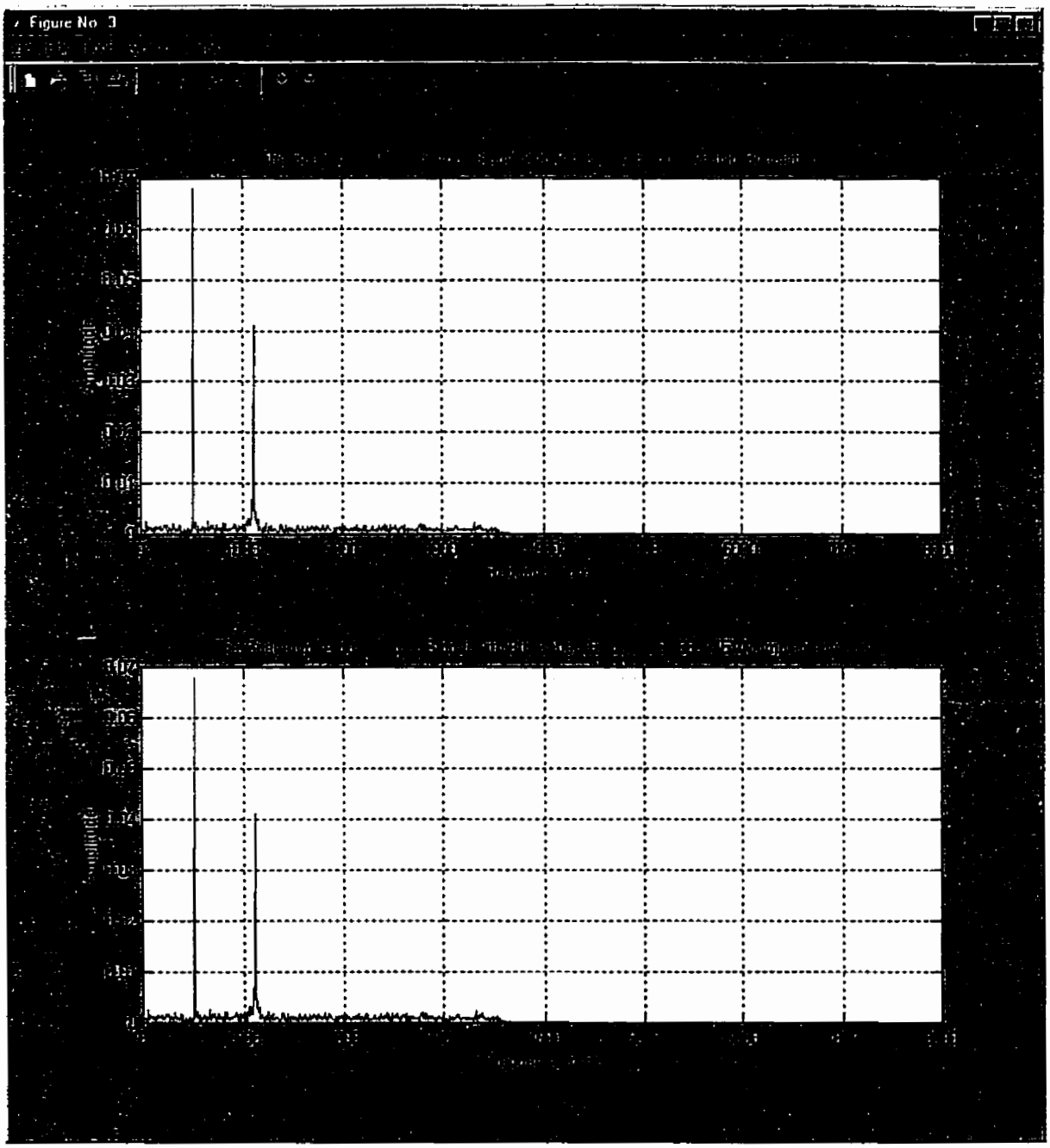


Figure 4.22 Output Signal in the frequency domain with simulation and implementation

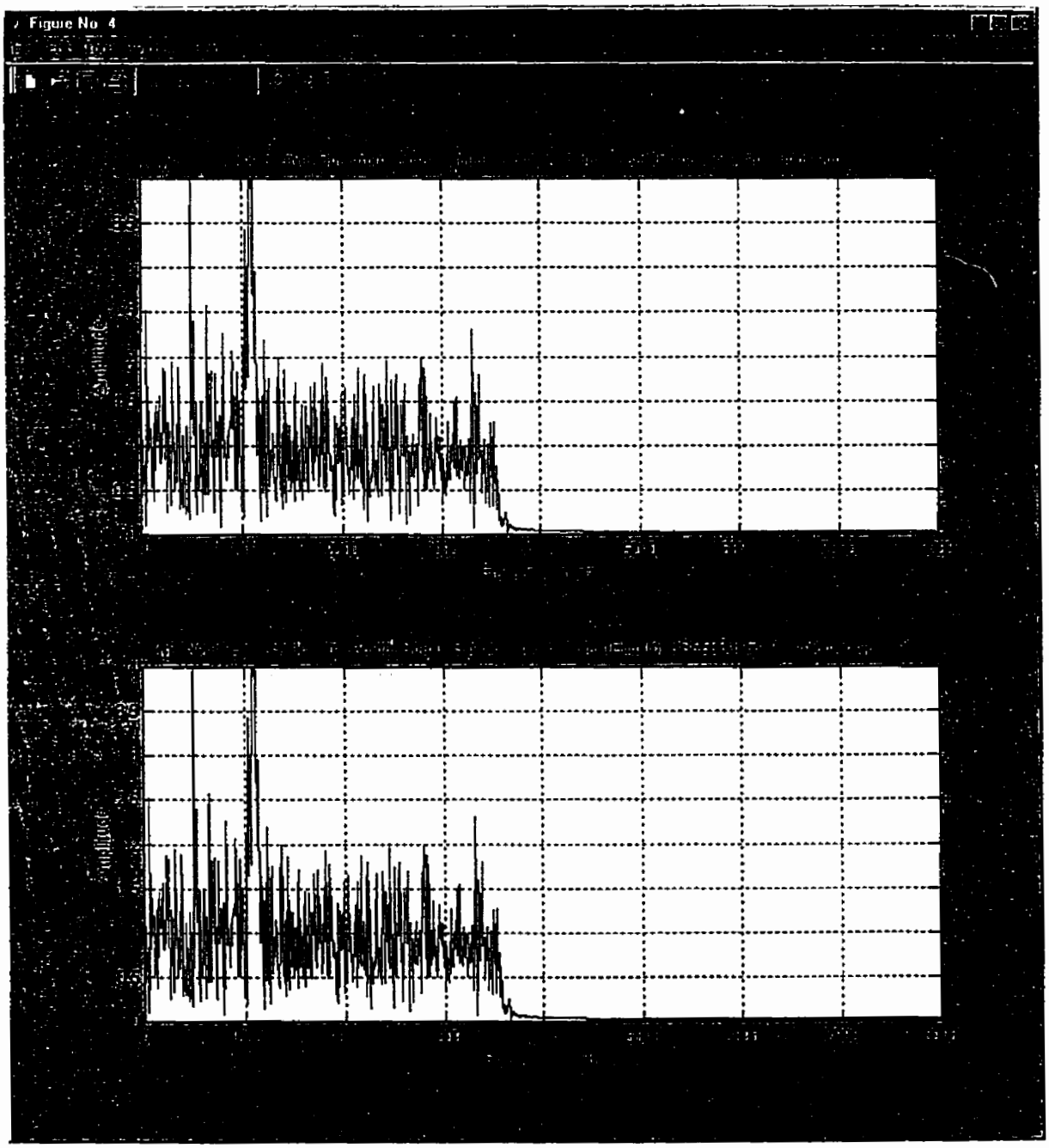


Figure 4.23 Noise level spectrum of the output signal with simulation and implementation

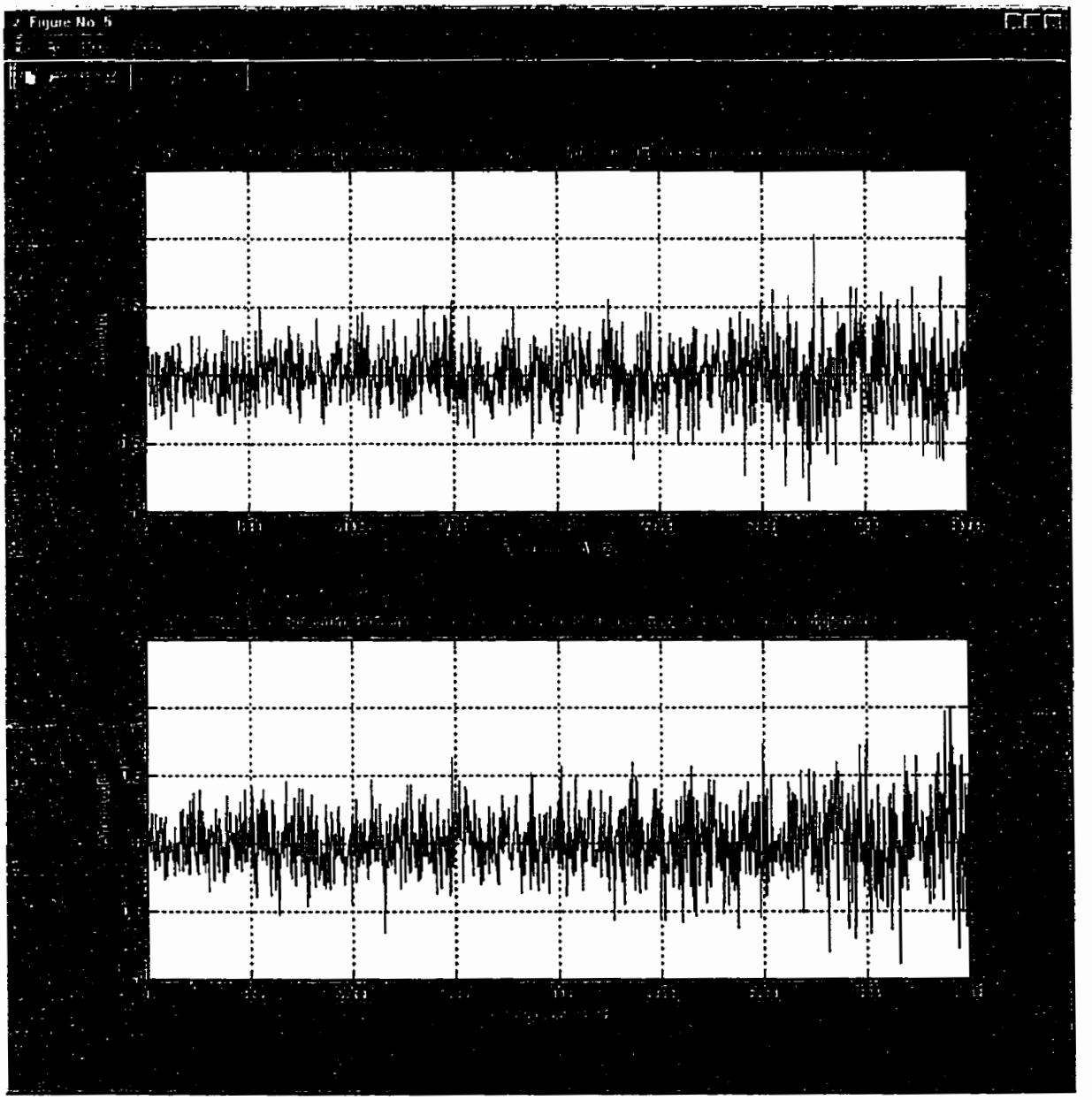


Figure 4.24 Output: Real part and imaginary part difference with simulation and implementation

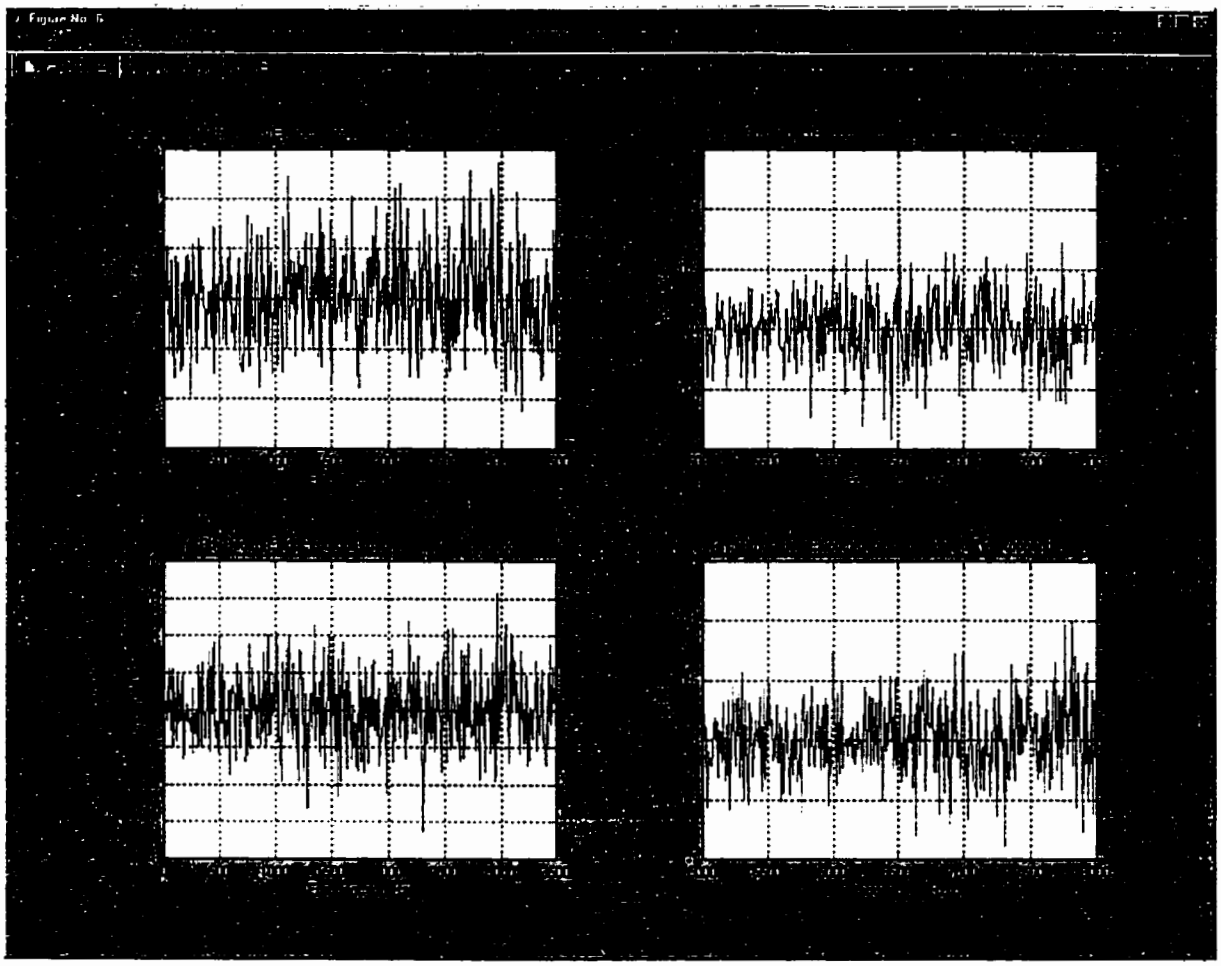


Figure 4.25 Output: Real part and imaginary part difference in the passband and stopband

```
Programmer's File Editor - [F:\WD_Filter_Design\matlab_sim_cauer9\print_sim_impl_power_comp_cauer9.txt]

        Comparison with Matlab simulation & DSP56307EVM implementation

        by a 9th order Cauer WD Filter in terms of Parseval Realation


     Simulation in time domain: Input Signal Power = 13.80498097035438
     Simulation in freq domain: Input Signal Power = 13.80498097035437

  Implementation in time domain: Input Signal Power = 13.80498097035438
  Implementation in freq domain: Input Signal Power = 13.80498097035437

     Simulation in time domain: Output Signal Power =  7.22122048081833
     Simulation in freq domain: Output Signal Power =  7.22122048081834

  Implementation in time domain: Output Signal Power =  7.22121962165052
  Implementation in freq domain: Output Signal Power =  7.22121962165052

        The following is to compare power difference of output signal
              between simulation and DSP implementation

     Output Signal: Power Difference in time domain =  0.00000085916781
     Output Signal: Power Difference in freq domain =  0.00000085916781
```

Figure 4.26 Power comparison: data file, ***print_sim_impl_power_comp_cauer9.txt***

4.12 Preliminary analysis

- The values for the input signal of the simulation and implementation are identical, which guarantees there is no impact of different platforms for the test vector.
- The values for the output signal of the simulation and implementation in the time domain and the frequency are very close, which indicate the signals we gathered are correct with respect to Parseval's relation.
- From a Parseval relation point of view, the power difference in the time or frequency domain between Matlab simulation and DSP56307EVM implementation is about $\mathbf{8.59x10^{-7}}$, which is acceptable for the design.
- Due to complex numbers in the frequency, we compare the output difference between the Matlab simulation and the DSP56307EVM implementation using the real part and imaginary part of the passband and stopband, respectively, and graphically. We can see the difference is within $\mathbf{8.0x10^{-9}}$ ~ $\mathbf{1.5x10^{-8}}$ (Figure 4.25), which is acceptable.

4.13 File management and name conventions

There are a number of files including source codes and data files with a variety of names. To make them more readable for the reader, we introduce the file and directory name conventions and clarify the file managements with the example, 9th order Cauer lowpass filter design. The examples, a 5th order Chebyshev lowpass filter design and a 7th order Butterworth lowpass filter design use the same conventions and file tree. We list four figures from a high-level view to the details to show how the file is managed and explain the name conventions of the directory, source code and data files. We use **bold** font in the names for directories or source codes and ***bold and Italic*** font in the names for the data files or executables.

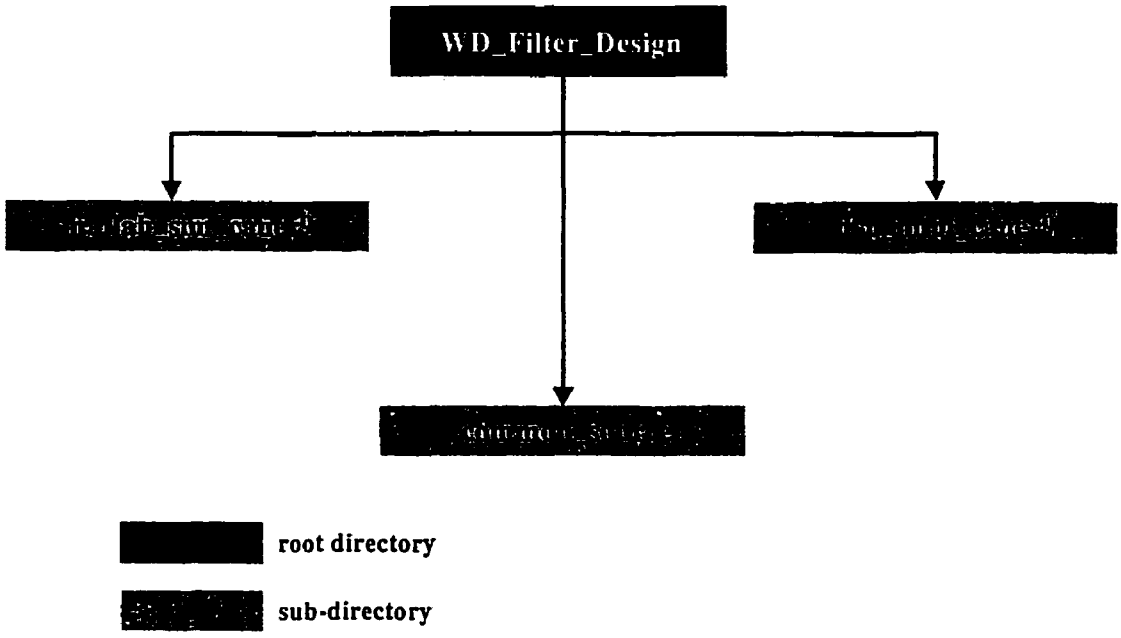


Figure 4.27 9th Order Cauer Filter Design: Source Code and Data File Management High Level View

- **WD_Filter_Design:** directory that is the home directory for the overall filter design. The directory contains 7 sub-directories. In Figure 4.27, we only show 3 sub-directories for the 9th order Cauer lowpass filter design.
- **common_source:** sub-directory that includes all 3 types of source codes and executables for the coefficient calculations of the filter design. Also it covers with source codes and data files for the vector generator as an input signal for the verification.
- **matlab_sim_cauer9:** sub-directory, **_cauer9** stands for 9th order lowpass filter. Since all source codes in this directory are Matlab codes, it is not only for Matlab simulation, but also for an analysis comparison between the Matlab simulation and the DSP implementation.
- **dsp_impl_cauer9:** sub-directory that means 9th order Cauer lowpass filter design for DSP implementation only. It contains all source codes written using C, test vector in assembly format, executable, data files, command script and some auto-generated debugging sources after compilation.

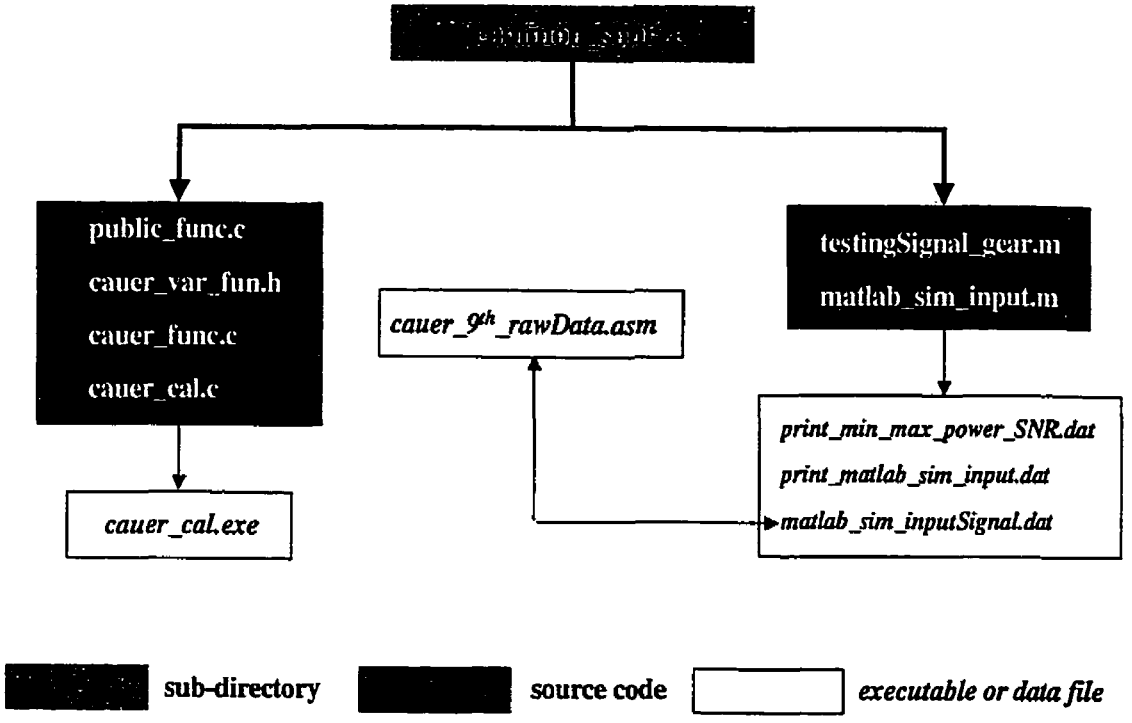


Figure 4.28 9th Order Cauer Filter Design: Source Code and Data File Management
sub-directory: common_source

- **public_func.c:** source code, the name is public functions, which gather all sharable functions for the coefficient calculations with 3 types of filter designs.
- **cauer_var_fun.h:** source code. The name means the declaration of the variables and functions of the coefficient calculations for the 9th order Cauer lowpass filter design.
- **cauer_func.c:** source code which stands for some calculation process of the coefficient calculations for the 9th order Cauer lowpass filter design.
- **cauer_cal.c:** source code that is the main procedure of the coefficient calculations for the 9th order Cauer lowpass filter design.
- **cauer_cal.exe:** executable that can be implemented in a DOS window for the 9th order Cauer lowpass filter design.

- **testingSignal_gear.m:** source code, the name is used as a generator for the test vector. It also generates a data file for the characteristics of the signal if the option is ON when executed.
- ***print_min_max_power_SNR.dat:*** data file. Executing **testingSignal_gear** with print ON option generates it. The name means printing out the minimum and maximum amplitudes real signal power, noise power and SNR in dB. All data files in the next discussions will be enclosed in appendix section if the name contains ***print_*** or ***_print***
- ***matlab_sim_inputSignal.dat:*** data file. It is generated by executing **testingSignal_gear** with print ON option. The data matrix is 2048x1. This data will be as an input signal to pass through all different types of filters with simulation and implementation.
- ***cauer_9th_rawData.asm:*** data file which is exactly the same as ***matlab_sim_inputSignal.dat*** with assembly format for the DSP implementation only. It will be copied to the sub-directory, **dsp_impl_cauer9.** The data matrix is also 2048x1.

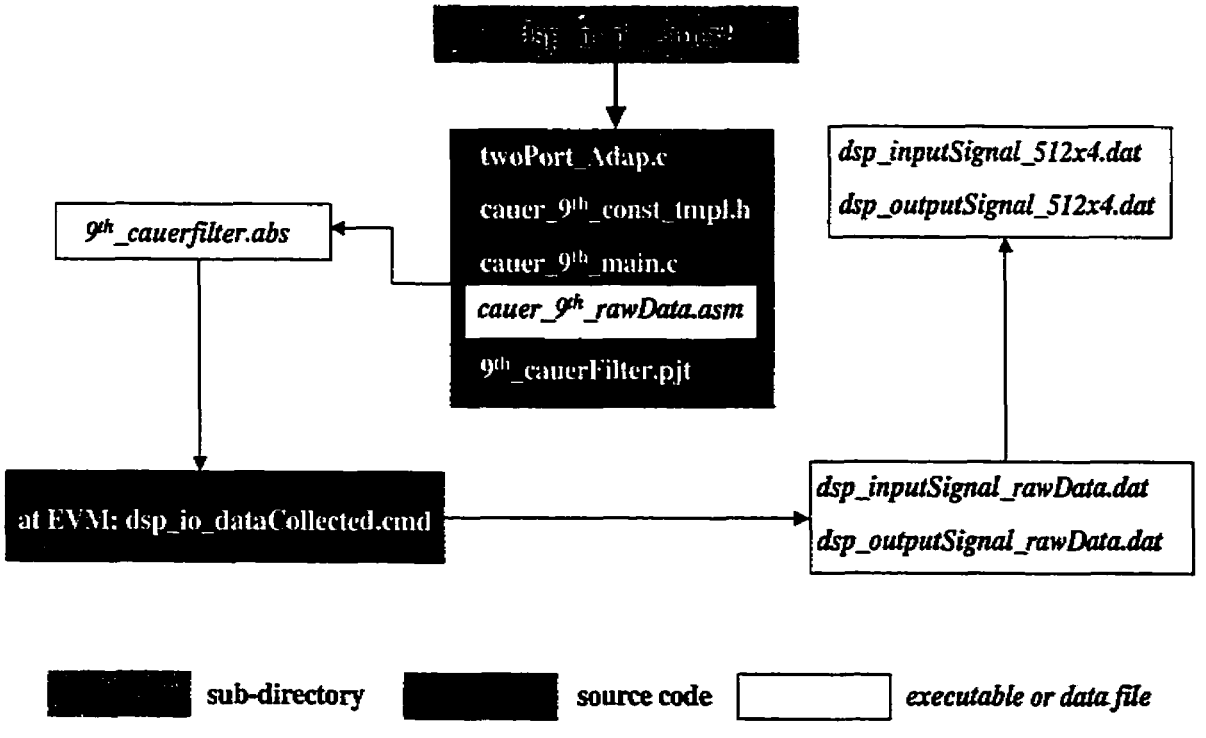


Figure 4.29 **9th Order Cauer Filter Design: Source Code and Data File Management sub-directory: dsp_impl_cauer9**

- **9^{th}_cauerFilter.pjt:** project file. The name means a project for the 9^{th} order Cauer lowpass filter design. It is required by TASKING EDE tool. Under this project, we put all source code and data files in it for easy loading, compiling, linking, and building.
- **twoPort_Adap.c:** source code. This is a basic function to perform a two port adaptive WD filter routine.
- **cauer_9^{th}_const_tmpl.h:** source code. The name is the declaration of the constant templates and basic functions for 9^{th} order Cauer lowpass filter design.
- **cauer_9^{th}_main.c:** source code. This is the main function to execute the procedures for the 9^{th} order Cauer lowpass filter design.
- **9^{th}_cauerFilter.abs:** executable. It automatically be generated by **cauer_9^{th}_main.c** and it is downloadable to the DSP56307EVM board.
- **dsp_io_dataCollected.cmd:** script. After EVM in running mode, executing this script in the command window can auto-collect input data and output data at pre-set memory addresses and auto-generate 2 data files, we explain them as follows.
- **dsp_inputSignal_rawData.dat & dsp_outputSignal_rawData.dat:** data files. They are generated by the above script. The names mean the raw data of the input signal & output signal in the DSP56307EVM implementation. Each file has all data addresses with 512x4 formats.
- **dsp_inputSignal_512x4.dat & dsp_outputSignal_512x4.dat:** data files. The data and size in each file are exactly the same as in the above, correspondingly, but Excel removes the all message addresses. And they will be copied to the sub-directory, **matlab_sim_cauer9.**

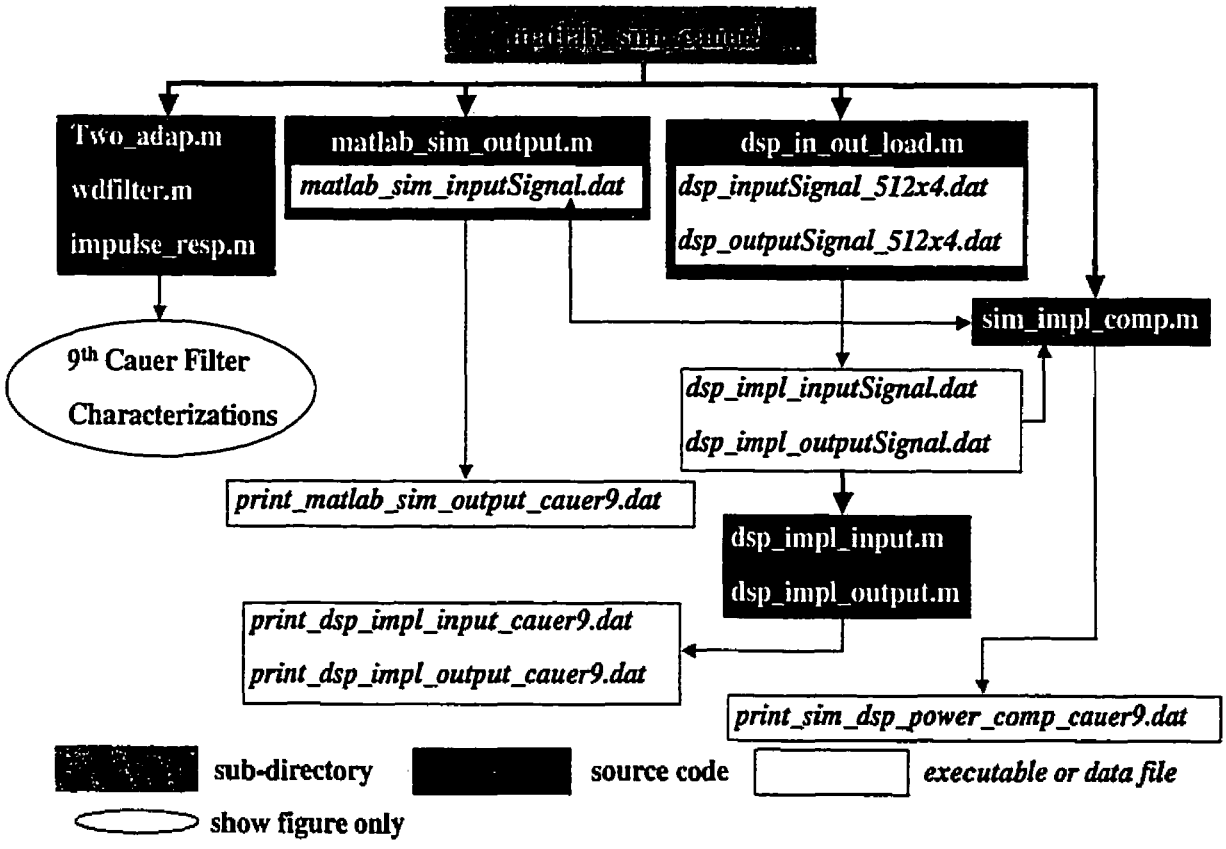


Figure 4.30 **9th Order Cauer Filter Design: Source Code and Data File Management sub-directory: matlab_sim_cauer9**

- **Two_adap.m:** source code. This is a basic function to perform a two port adaptive WD filter routine.
- **wdfilter.m:** source code. This is another basic function to implement the routine for the 9th order WD lowpass filter with defined coefficients.
- **impulse_resp.m:** source code. The name means impulse response. It plots the characteristics of the 9th order Cauer lowpass filter with impulse response.
- **matlab_sim_output.m:** source code. It is only valid when ***matlab_sim_inputSignal.dat*** is loaded in the same sub-directory. It can plot the output signal with the Matlab simulation by passing the test vector to the 9th order Cauer lowpass filter.

- **print_matlab_sim_output_cauer9.dat:** data file. The name means print the output data with the Matlab simulation by passing the 9^{th} order Cauer lowpass filter when **matlab_sim_output** is executed with the print option in the ON mode.
- **dsp_in_out_load.m:** source code. Basically, it is the procedure to load the input and output data from the DSP implementation of the 9^{th} order Cauer lowpass filter. It is executable if ***dsp_inputSignal_512x4.dat*** and ***dsp_outputSignal_512x4.dat*** are loaded in the same sub-directory. It also can generate another 2 data files.
- **dsp_impl_inputSignal.dat:** data file. This is the input data before the DSP implementation of the 9^{th} order Cauer lowpass filter. The data matrix is 2048x1.
- **dsp_impl_outputSignal.dat:** data file. This is the output data after the DSP implementation of the 9^{th} order Cauer lowpass filter. The data matrix is 2048x1.
- **dsp_impl_input.m:** source code. It can plot the input signal and generate a data file for print only with the DSP implementation of the 9^{th} order Cauer lowpass filter.
- **dsp_impl_output.m:** source code. It can plot the output signal and generate a data file for printing only with the DSP implementation of the 9^{th} order Cauer lowpass filter.
- **print_dsp_impl_input_cauer9.dat:** data file. It means printing only the input data with the DSP implementation of the 9^{th} order Cauer lowpass filter. The size of the data is 512x4.
- **print_dsp_impl_output_cauer9.dat:** data file. It means printing only the output data with the DSP implementation of the 9^{th} order Cauer lowpass filter. The size of the data is 512x4.
- **sim_impl_comp.m:** source code. It means the comparison between Matlab simulation and DSP implementation. It compares the simulation with the DSP implementation in the time domain and the frequency domain in detail.
- **print_sim_dsp_power_comp_cauer9.dat:** data file. This is the final data file with the design of the 9^{th} order Cauer lowpass filter. The name means printing the power comparison between the Matlab simulation and the DSP implementation of Parseval's relation, and to check the power difference between 2 different ways to implement the filter.

Chapter 5

The Design Discussion and Conclusions

In this chapter, we will discuss and analyze the whole design procedure described in the above chapters.

5.1 Discussion

- From (3.13) of chapter 3, we have the formula, $k_s \leq r \leq k_p$, this means we can choose any value of r within $[k_s, k_p]$ as a factor to compute the coefficients for the Butterworth lowpass filter design. But this will create a ripple in the frequency response in the stopband. For example, if we use the same general specifications as in Appendix B, and choose r=0.0625, the resulting coefficients are as follows:

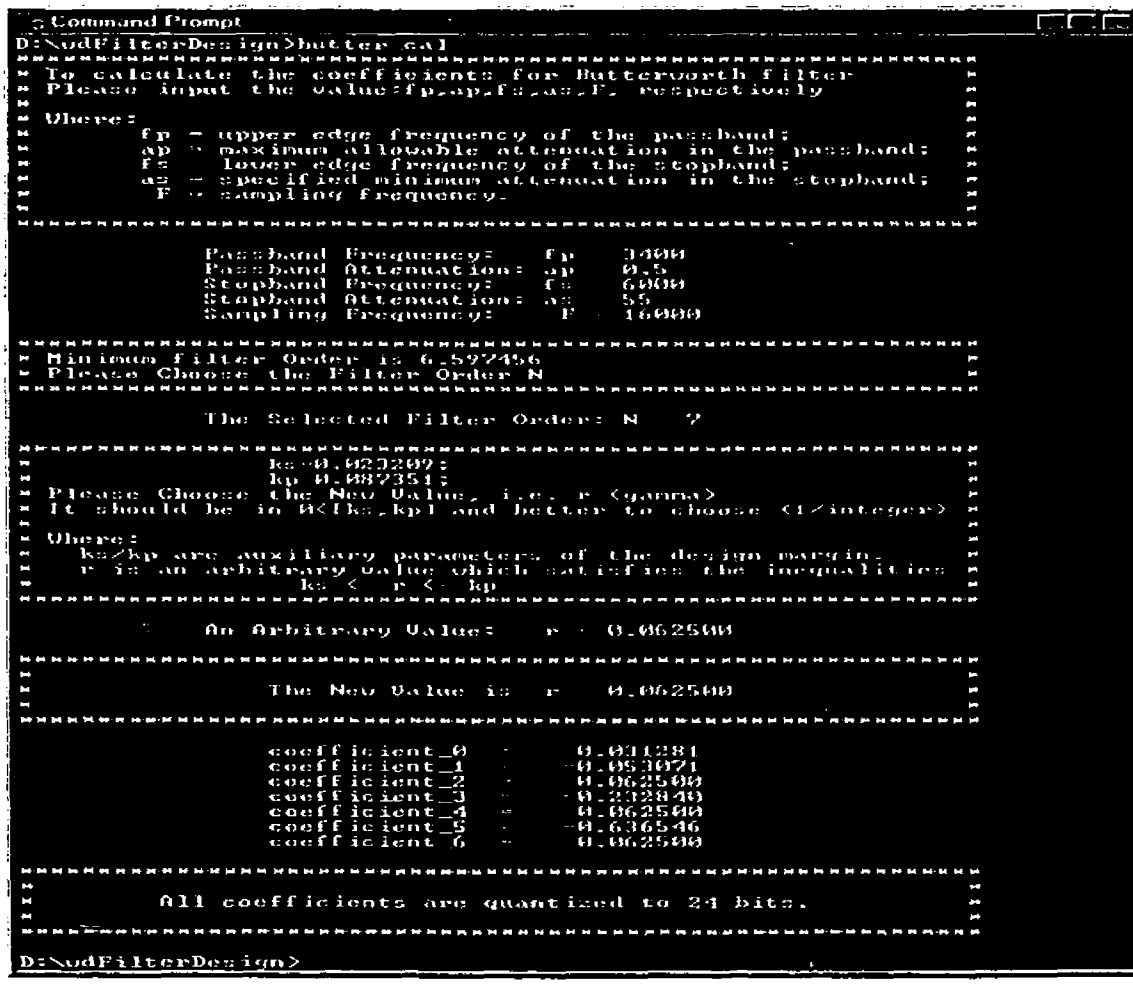


Figure 5.1 The coefficient calculations for a Butterworth lowpass filter

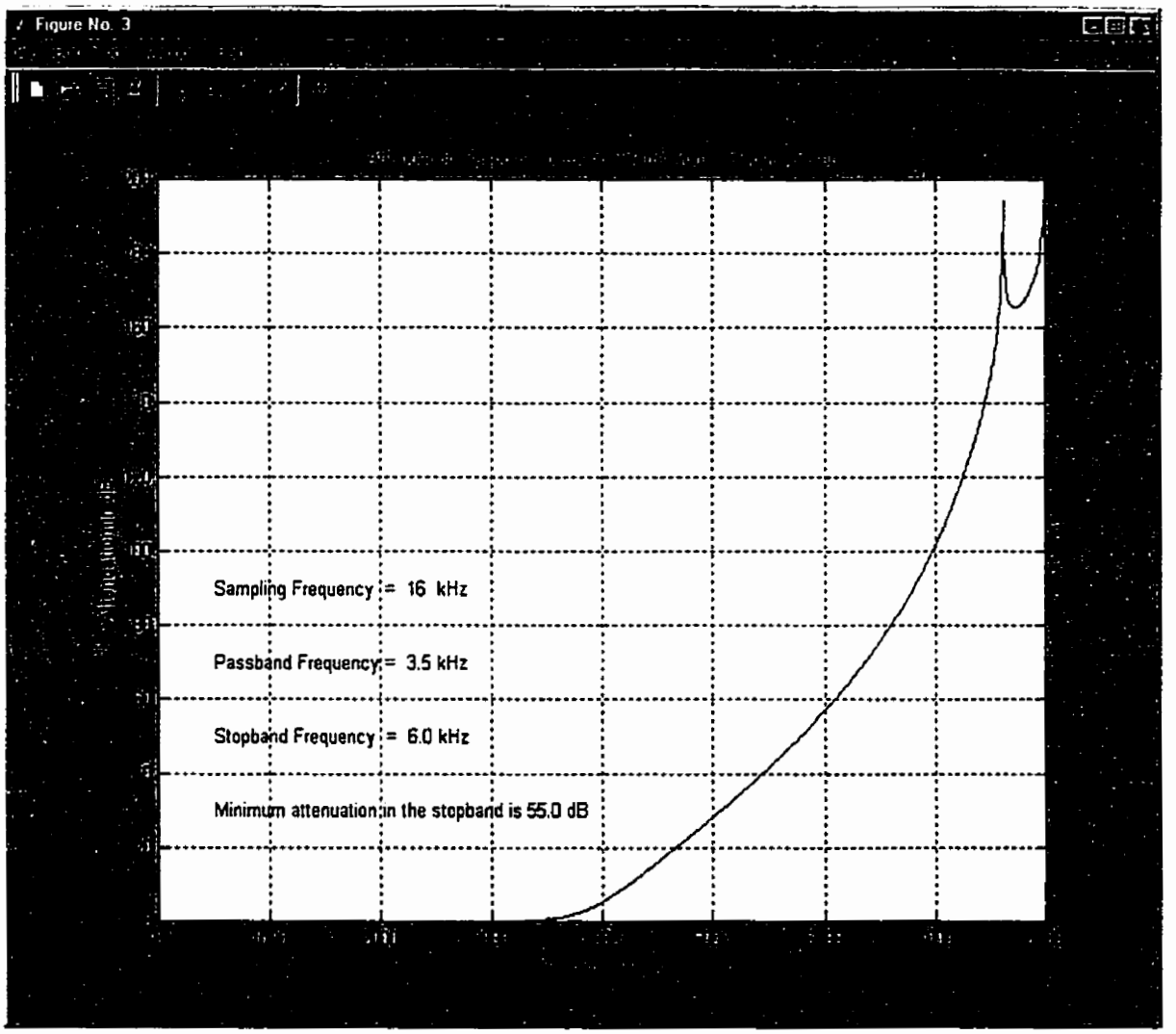


Figure 5.2 The frequency response of the Butterworth filter with r = 0.0062500

For the design in this thesis, we choose $r = k_s$ to avoid it, the reader can see Figure B.2 as a reference. We also choose $\varepsilon_p^* = \varepsilon_{p\ min}$ from (3.18) for the Chebyshev filter design, for the same reason as in the Butterworth design.

- Using Figure 4.26, Figure A.15 and Figure B.13, we have demonstrated that the power in the time domain or in the frequency domain is extremely close between the Matlab simulation and the DSP56307EVM implementation using Parseval's relation, but the powers of the output signals for the different types of filters are different, because the length of each passband and the attenuation in the passband and the stopband of the examples are different.

- From DSP56307EVM implementation point of view, the input signals of different types of the filters are exactly same, but the output signals of different types of the filters are different when we apply an identical test vector to the different types of filters. That means the EVM board doesn't impact input signals at all whichever type of the filter is used. The differences in the output signal are due to different general specifications.

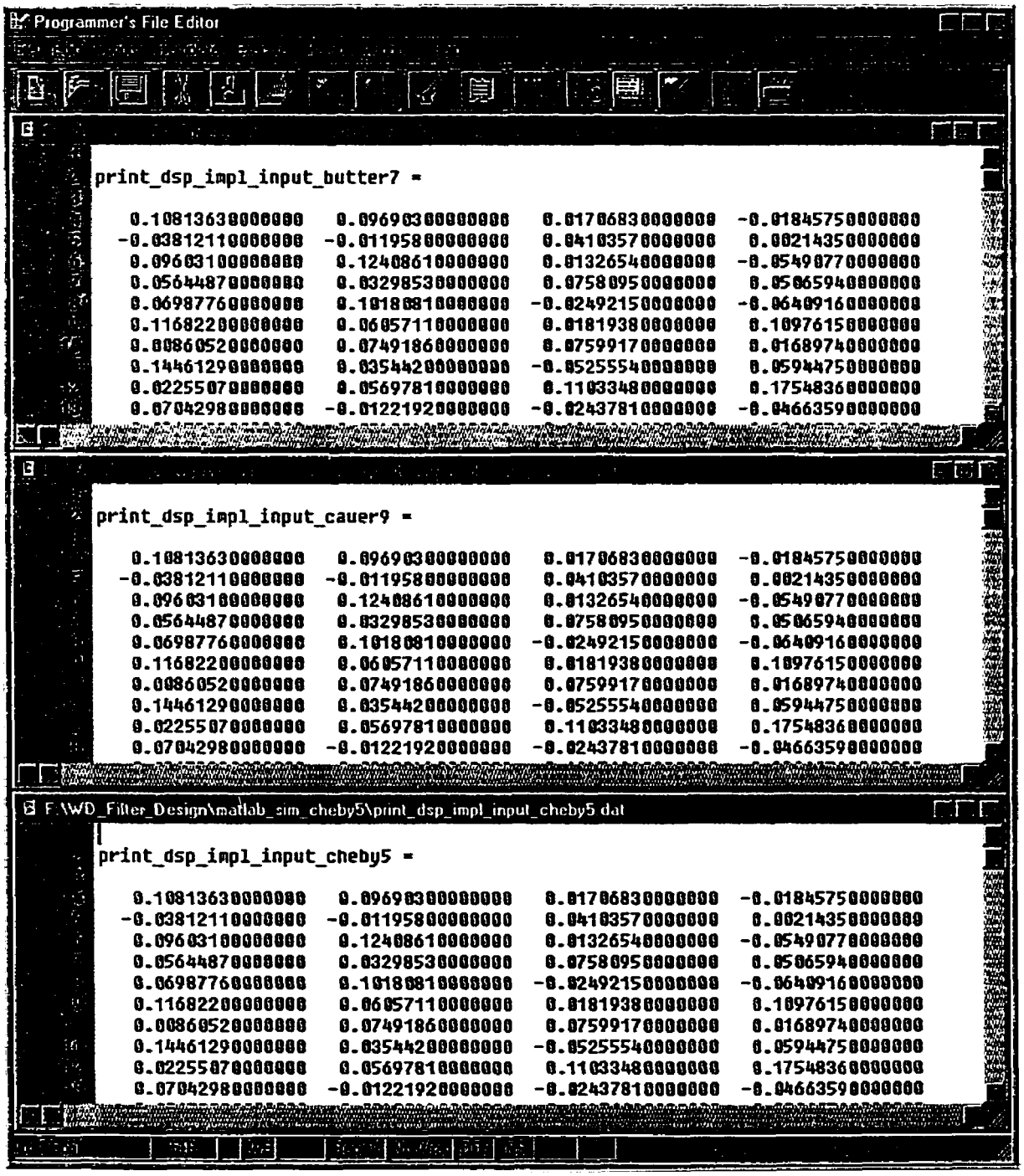


Figure 5.3 The input comparison with 3 different types of filters

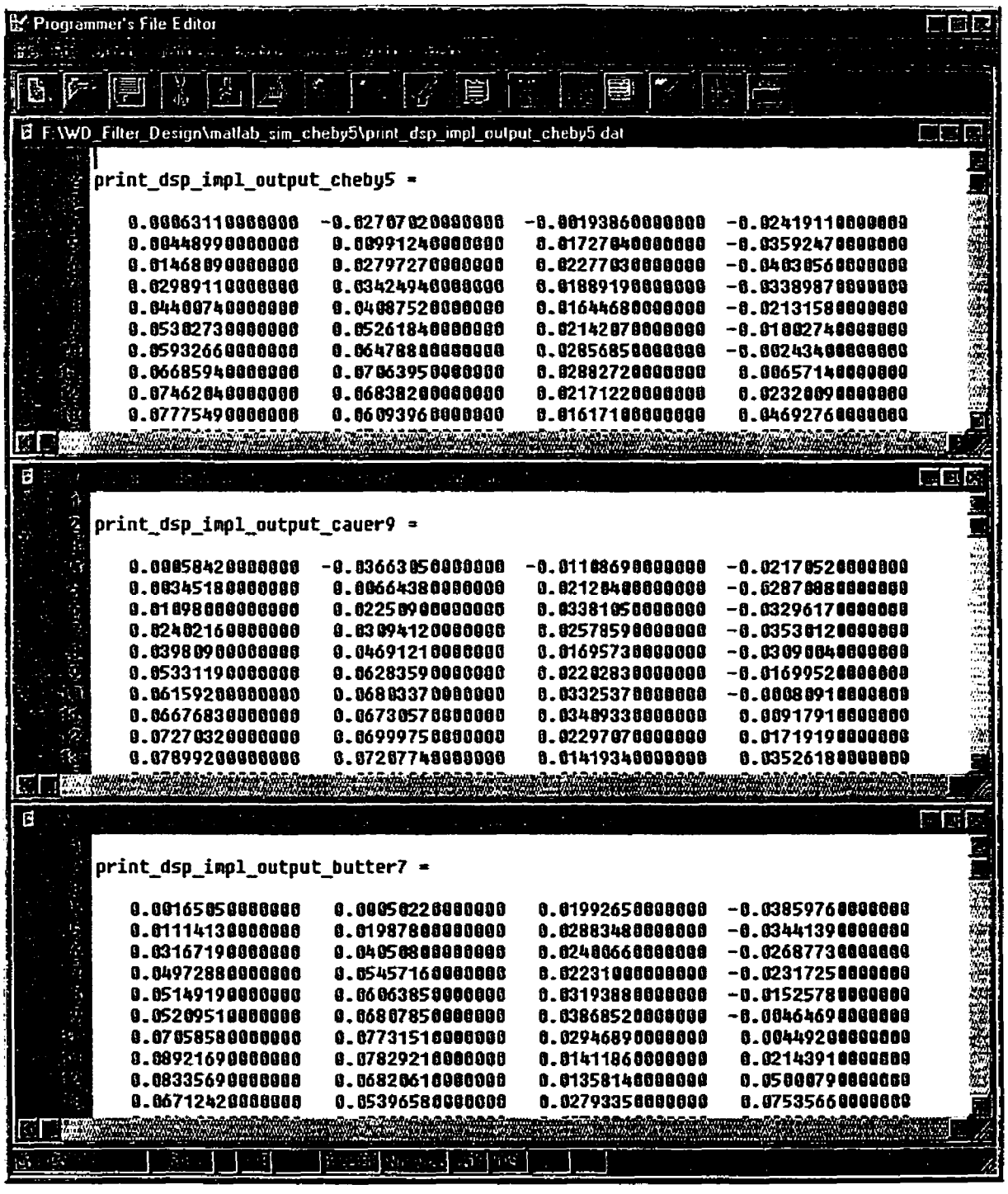


Figure 5.4 The output comparison with 3 different types of filters

- To analyze the difference of the output signal in the frequency domain between Matlab simulation and DSP56307EVM implementation, we take an FFT of the output signals for both methods, and compare the real part and the imaginary part separately due to complex number constraints, we can see the differences are within [$5.0\text{x}10^{-9}$, $1.5\text{x}10^{-8}$], which is

acceptable from a design point of view. This point is demonstrated using Figure 4.25, Figure A.14 and Figure B.12 as references.

5.2 Conclusions

- The explicit formulas developed by Gazsi to design odd-order lattice wave digital lowpass filters provide a very straightforward method for calculating the coefficients from general specifications.
- The computations are fairly tedious, it requires users to repeat the calculations of the coefficients when the specifications need to be changed. In this thesis, we developed a program package and created a friendly user interface for users to easily calculate the coefficients for three different types of filters, i.e. for Butterworth, Chebyshev and Cauer (Elliptic) filters.
- Part of the program package is to verify the characteristics for the different types of filters. We can easily check them by running the scripts, such as **impulse_resp.m** before the simulation and DSP implementation.
- A test vector can be passed to the simulation program and the real DSP implementation to verify whether or not the high frequency components have been filtered out. An appropriate test vector is very important for a successful design.
- We used three examples with the above three different types of the filters, and tested them in a Matlab simulation and in a DSP56307EVM implementation. We verified that the difference in accuracy in the frequency domain is less than 1.5×10^{-8}, which is acceptable for design, and then we concluded that the implementation of the explicit formulas to design lattice wave digital lowpass filters is realizable on a Motorola DSP56307EVM.
- From the DSP design point of view, there are always a number of source codes and data files to be managed properly. We created a method to maintain the files in order in section 4.3.
- Furthermore, a more generic package to design filter types other than those used in this thesis could be developed. For example, computing coefficients would require one program regardless of filter type; i.e. one program would design any odd-order WDF.

References

[1] A. Fettweis, "Wave digital filter: theory and practice". *Proc. IEEEE*, vol 74, pp270-327, Feb., 1986

[2] L. Gazsi, "Explicit formulas for lattice wave digital filters", *IEEEE Trans. Circuits and Sys.*, vol. CAS-32, pp68-88, Jan., 1985

[3] J. Wang, "Design of even-order complex wave digital filters", *M. Sc. Thesis*, University of Manitoba, Winnipeg, Canada, 1996

[4] C. Webb, "Design of digital wave-state-variable lattice filters", *M. Sc. Thesis*, University of Manitoba, Winnipeg, Canada, 1987

[5] A. Fettweis, "Digital filter structure related to classical filter networks", *Arch. Elek. Uebertragung*, vol. 25, pp79-89, 1971

[6] A. Fettweis, H. Levin, and A. Sedlmeyer, "Wave digital lattice filters", *Int. J. Circuit Theory Appl.* Vol. 2 pp163-174, June 1974

[7] A. Fettweis, "Wave digital filters for improved implementation by commercial digital signal processors", *Signal Processing*, North-Holland, vol.16, pp193-207, 1989

[8] A. Fettweis, "Digital circuit and systems" *IEEE Trans. on Circuit and Systems*, vol. Cas-31 No. 1 pp31-48, Jan., 1984

[9] A. H. Gray, and J. D. Markel, "Digital lattice and ladder filter synthesis", *IEEE, Trans. on Audio and Electroacoustics*, vol. AU-21, No. 6 pp491-500, Dec. 1973

[10] A. Fettweis, "Design of orthogonal and related digital filters by network-theory approach", *AEU*, Apr. 1990

[11] M. El-Sharkawy, "Digital signal processing applications with Motorola's DSP56002 processor", *PTR Prentice Hall*, Inc., 1996

[12] "DSP56307 user's manual", *Motorola Inc.*, 1998

[13] A. Oppenheim, and A. Willsky, "Signals and systems", *Prentice Hall*, 1996

[14] "DSP56xxx v2.1 Crossview Pro Debugger User's Guide", *Tasking Inc.*, 1998

[15] Vinay K. Ingle, and John G. Proakis "Digital Signal Processing using MATLAB", *Brooks/Cole*, 2000

[16] "DSP563xx EDE Source Compiler User's Guide", *Tasking Inc.*, 1998

Appendix A

The Results of a 5th order Chebyshev WD Lowpass Filter

Example: A 5th order Chebyshev WD lowpass filter design

General Specifications:

- Passband Frequency: = 3000 Hz
- Passband Attenuation: = 1.0 dB
- Stopband Frequency: = 5000 Hz
- Stopband Attenuation: = 40 dB
- Sampling Frequency: = 16000 Hz

```
Command Prompt

F:\WD_Filter_Design\common_source>cheby_cal
**********************************************************
* To calculate the coefficients for Chebyshev filter     *
* Please input the value:fp,ap,fs,as,F, respectively      *
*                                                        *
* Where:                                                 *
*       fp - upper edge frequency of the passband;       *
*       ap - maximum allowable attenuation in the passband;*
*       fs - lower edge frequency of the stopband;       *
*       as - specified minimum attenuation in the stopband;*
*        F - sampling frequency.                         *
*                                                        *
**********************************************************

            Passband Frequency:   fp - 3000
            Passband Attenuation: ap - 1.0
            Stopband Frequency:   fs - 5000
            Stopband Attenuation: as - 40
            Sampling Frequency:    F - 16000

**********************************************************
* Minimum filter Order is 4.132700                       *
* Please Choose the Filter Order N                       *
**********************************************************

            The Selected Filter Order: N - 5

**********************************************************
*                                                        *
* The actual attenuation in passband is  1.000000 dB     *
* The actual attenuation in stopband is 40.000000 dB     *
*                                                        *
**********************************************************

              coefficient_0   =    0.675837
              coefficient_1   =   -0.583980
              coefficient_2   =    0.678323
              coefficient_3   =   -0.846811
              coefficient_4   =    0.387688

**********************************************************
*                                                        *
*       All coefficients are quantized to 24 bits.       *
*                                                        *
**********************************************************

F:\WD_Filter_Design\common_source>
```

Figure A.1 User interface of the coefficients for the Chebyshev lowpass filter with the specifications

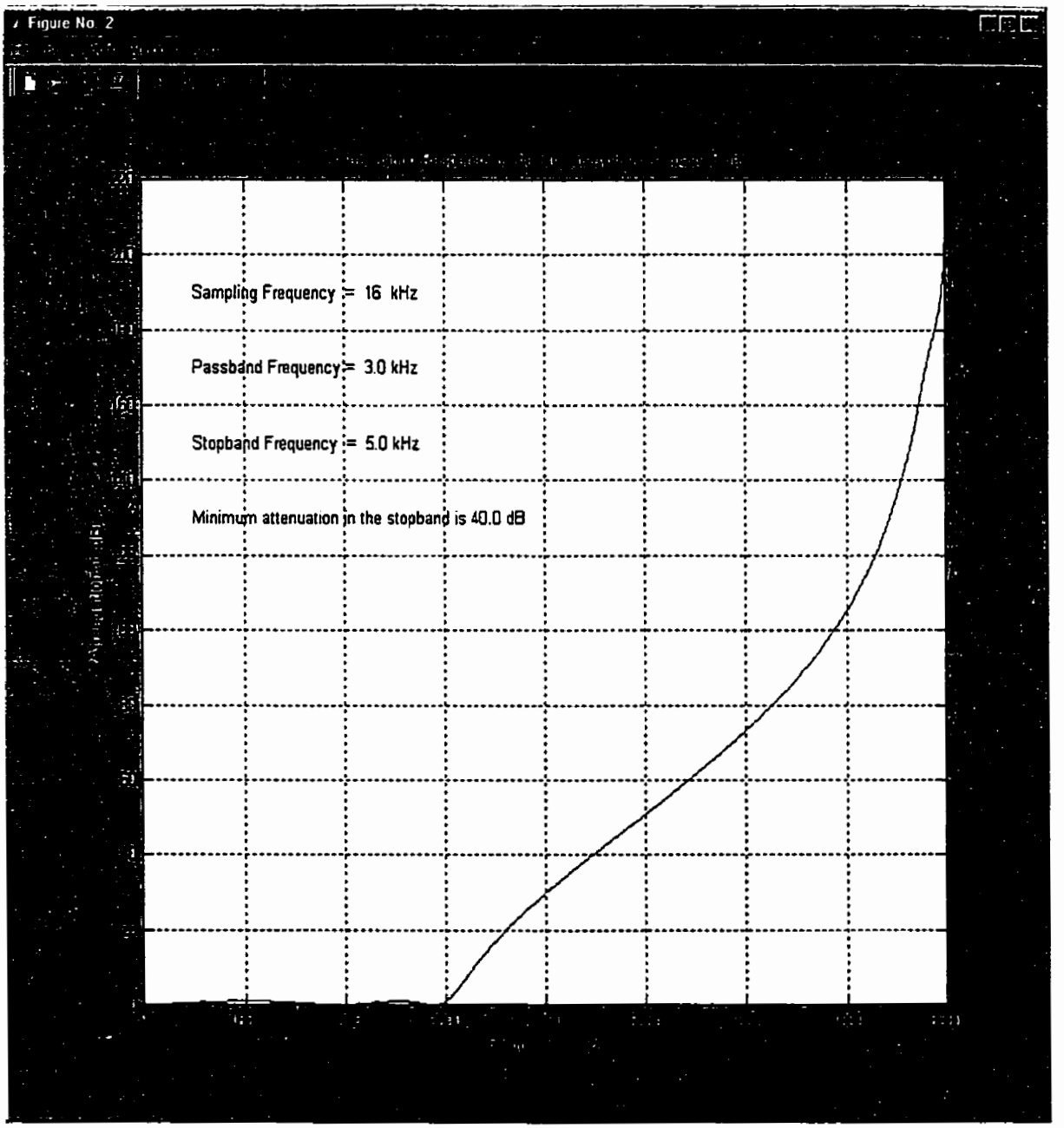


Figure A.2 The attenuation response in the passband and stopband of the 5th order Chebyshev WD lowpass filter

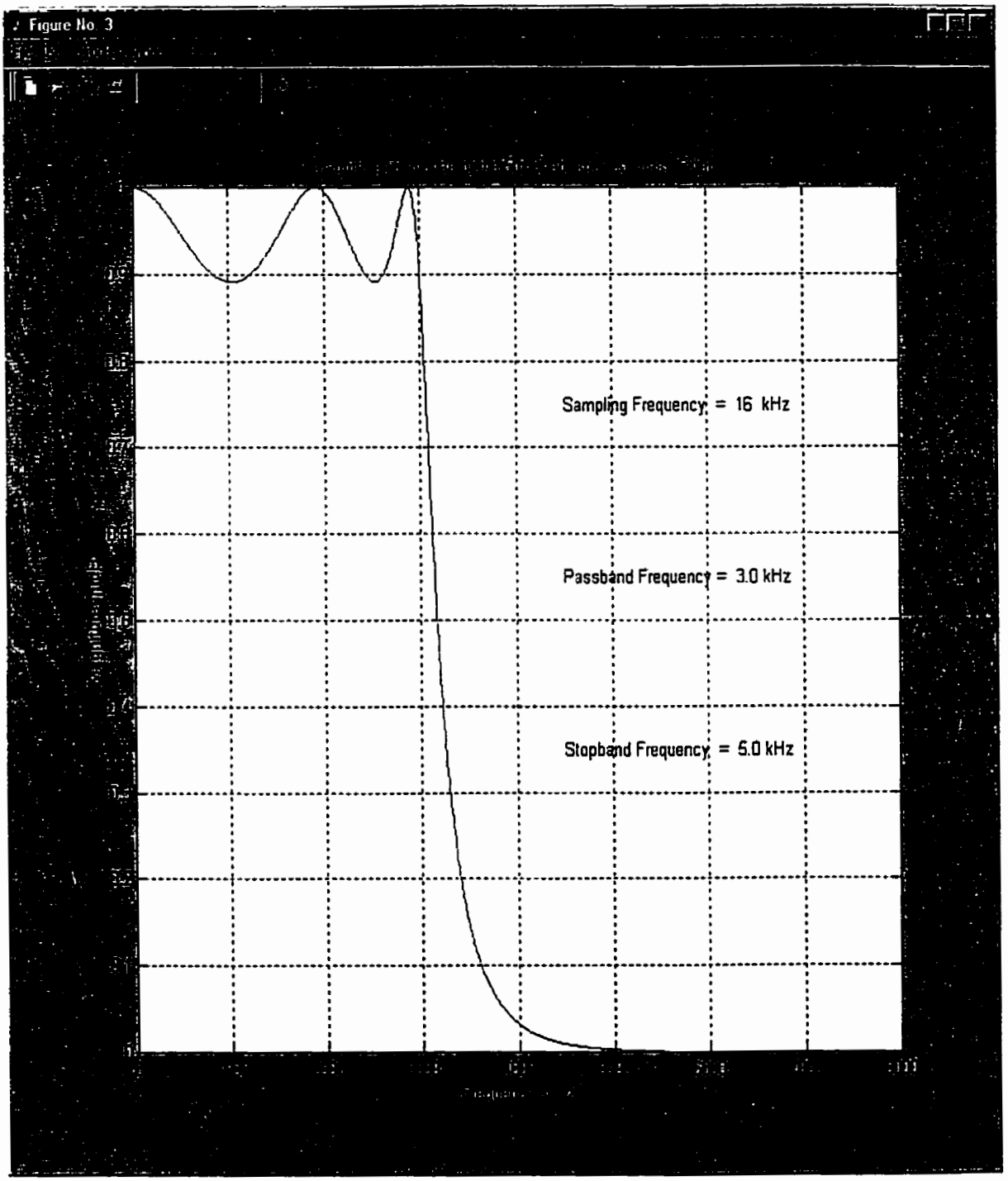


Figure A.3 The magnitude response of the 5th order Chebyshev WD lowpass filter

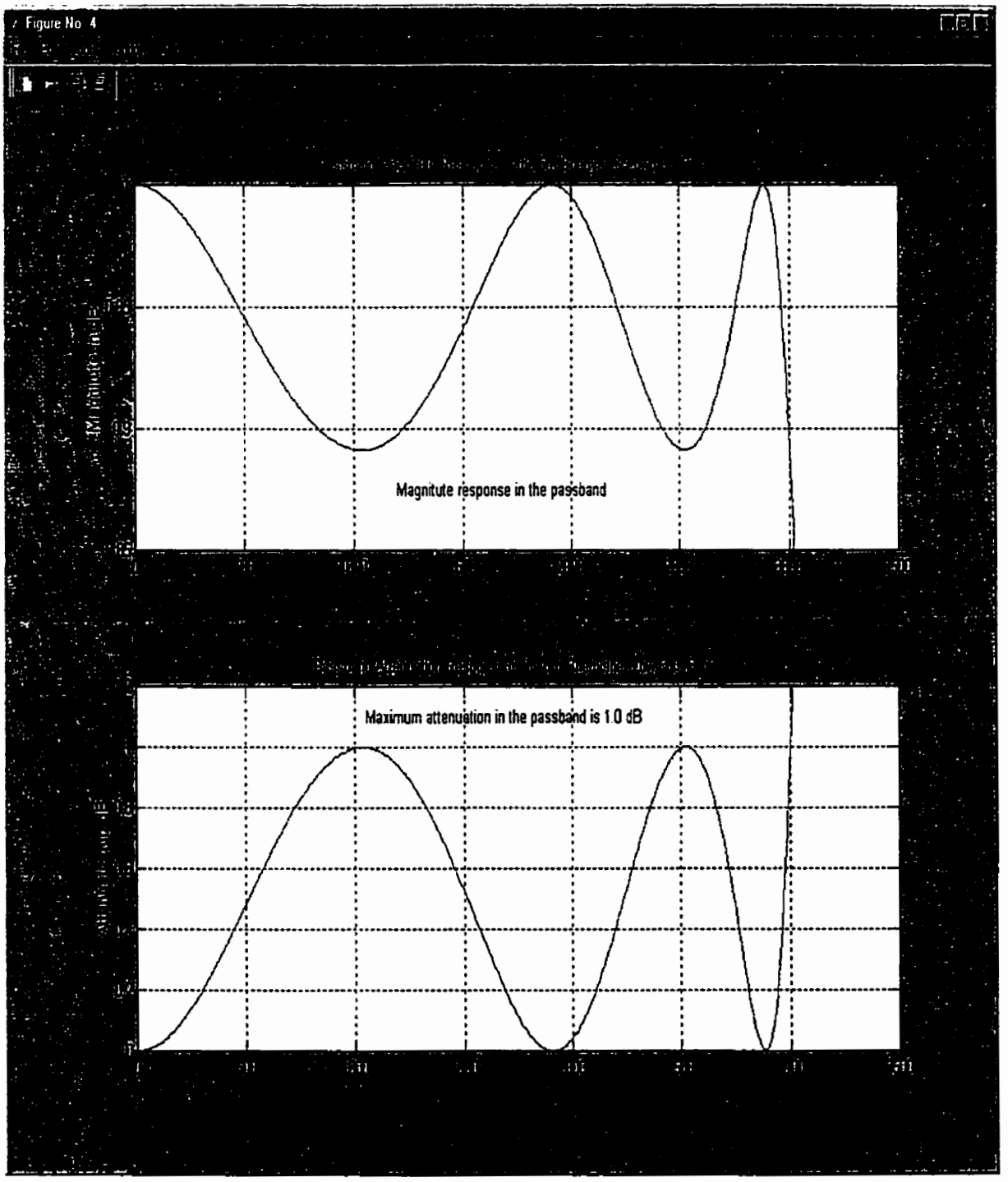


Figure A.4 The detailed response in the passband of the 5th order Chebyshev WD lowpass filter

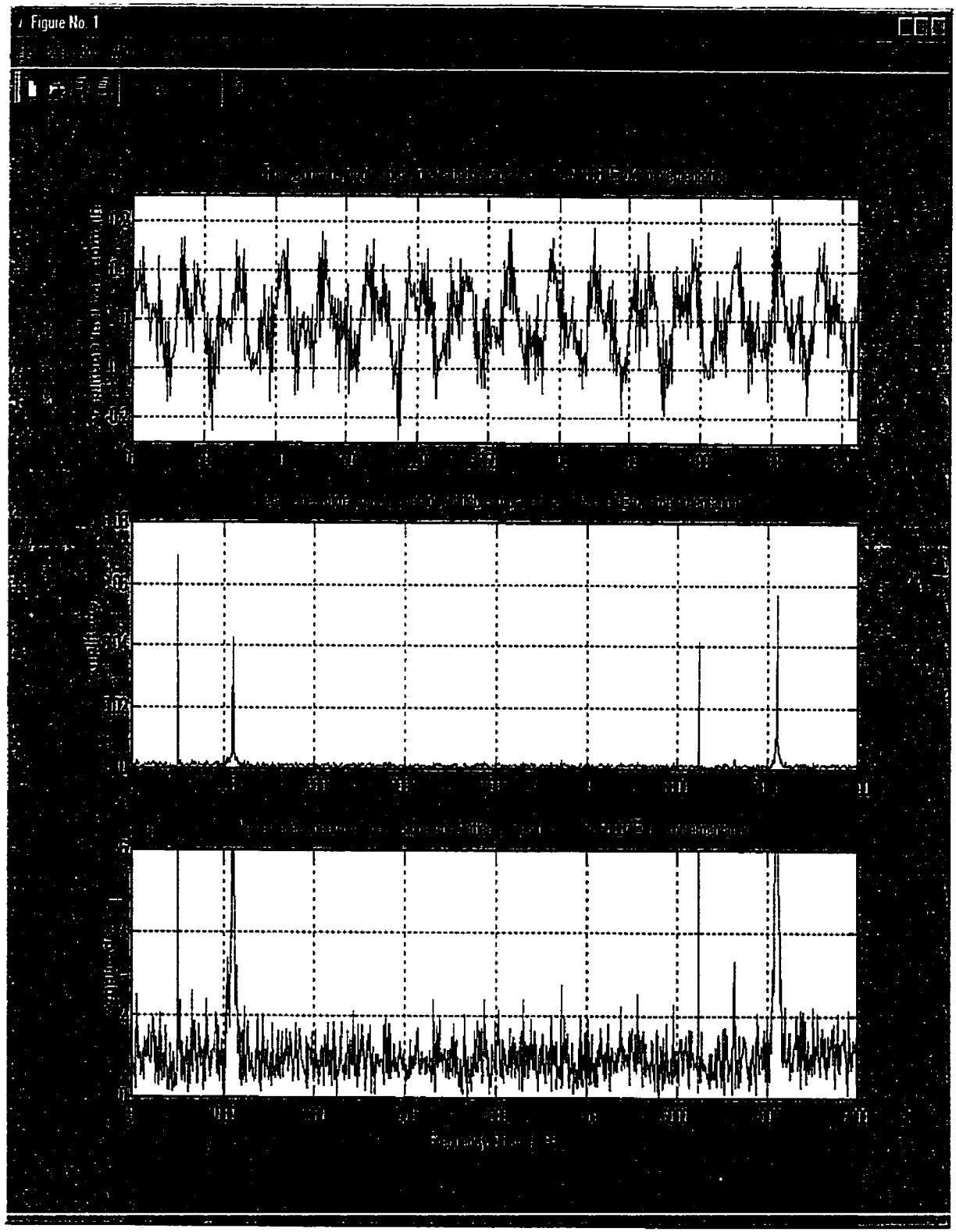


Figure A.5 The test vector in the time domain and frequency domains

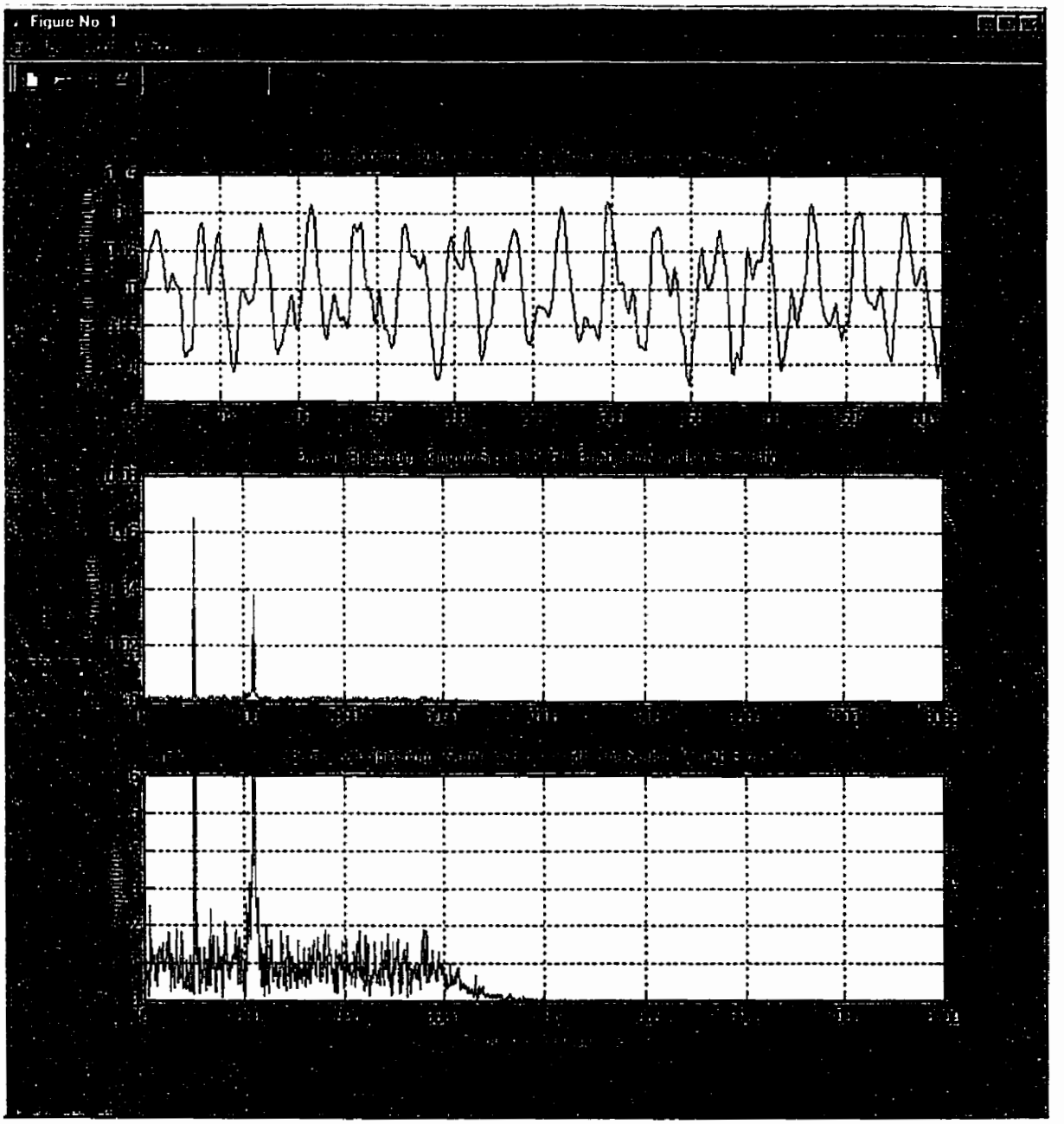


Figure A.6 Simulation: Output of the 5th order Chebyshev lowpass filter for the test vector

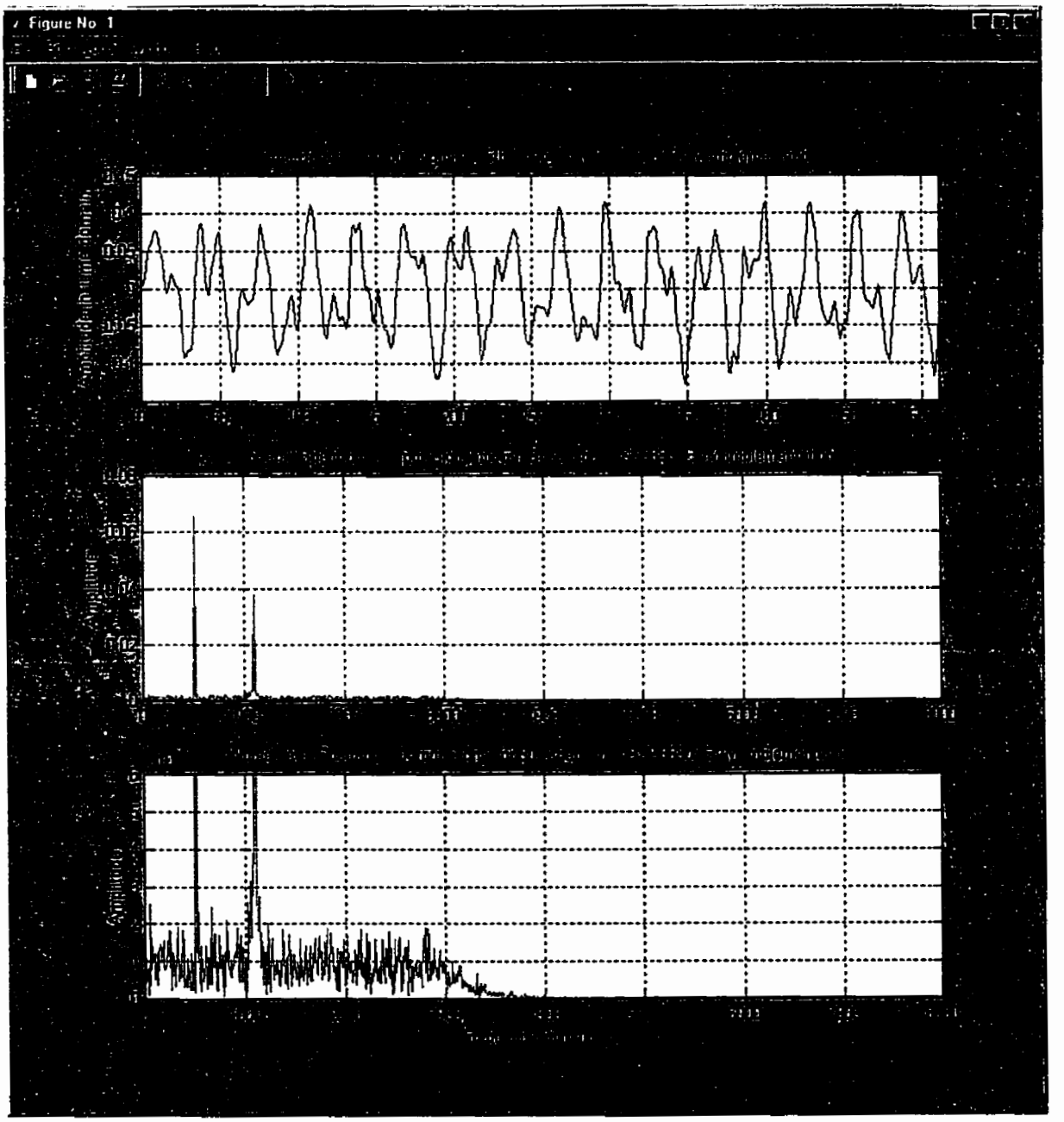


Figure A.7 Implementation: Output of the 5th order Chebyshev lowpass filter for the test vector

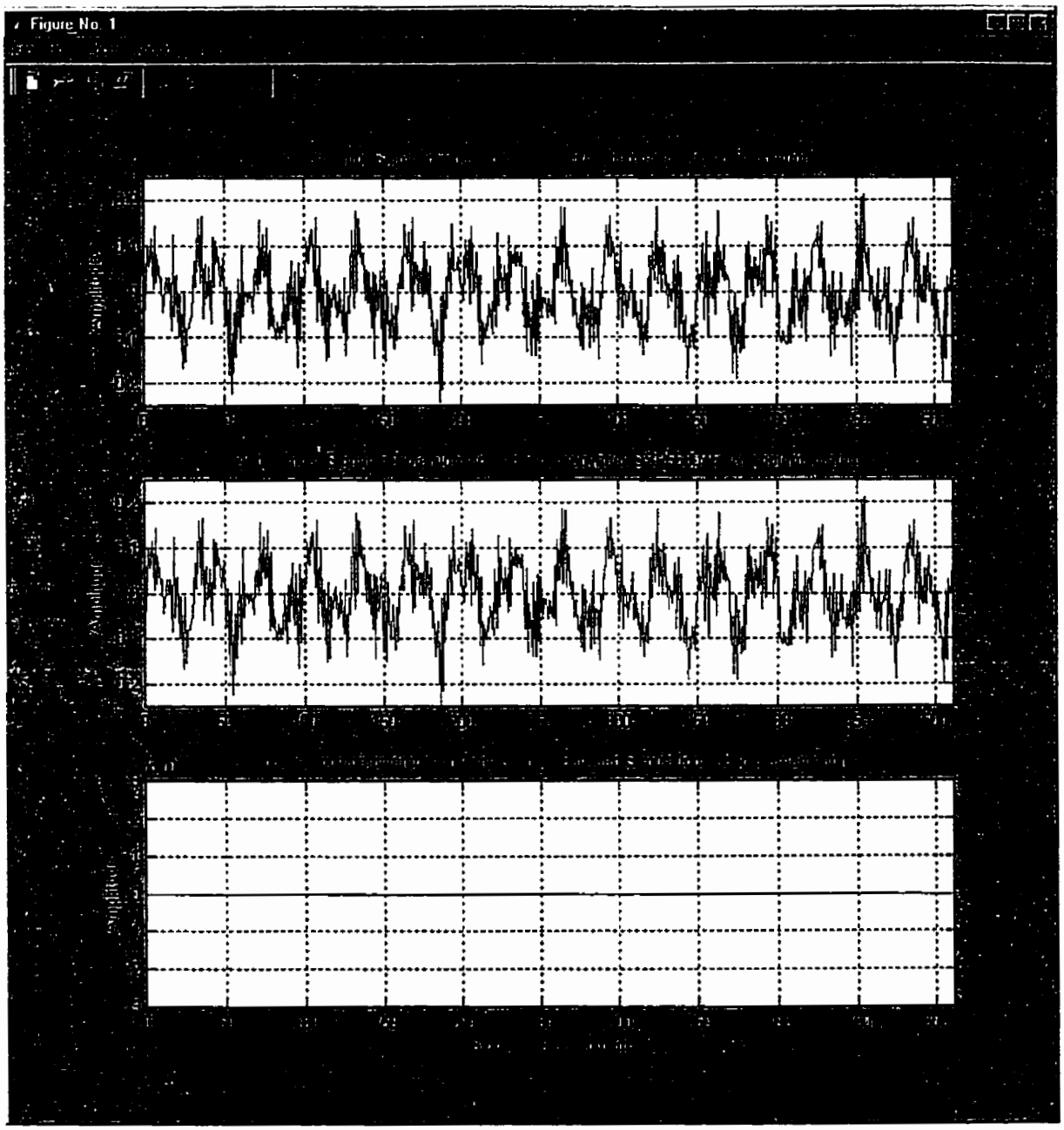


Figure A.8 Input Signal in the time domain with simulation and implementation

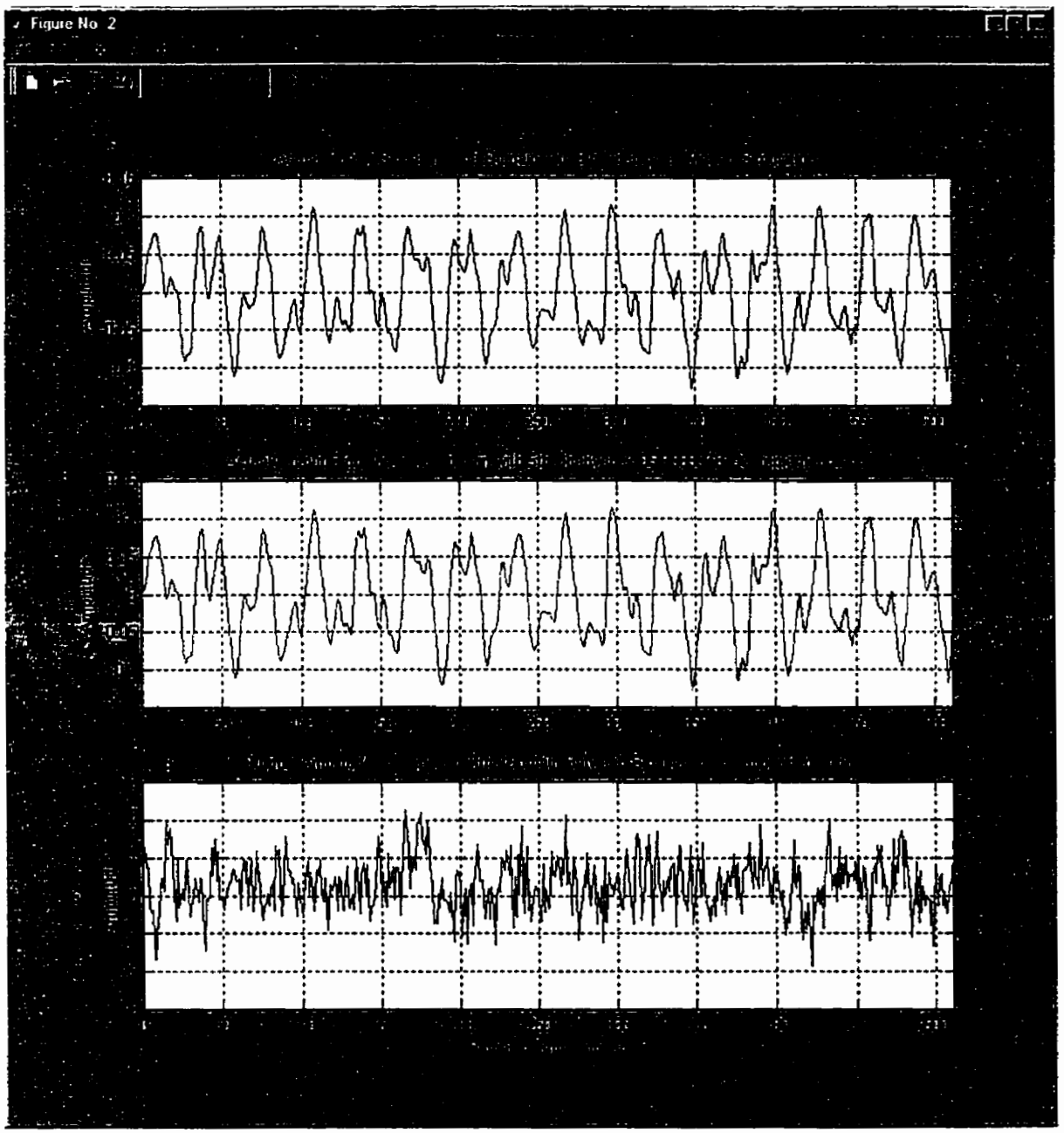


Figure A.9 Output Signal in the time domain with simulation and implementation

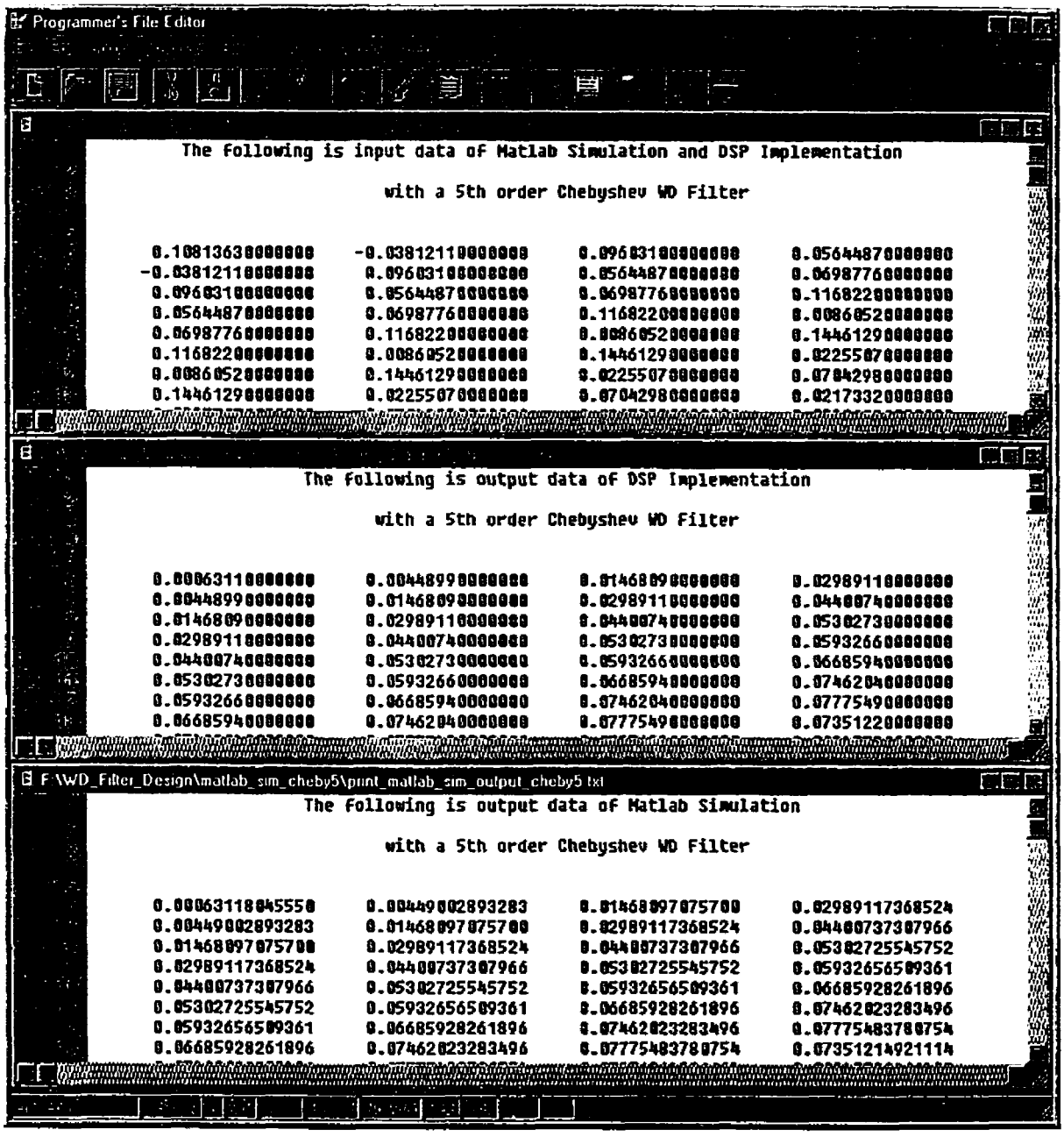


Figure A.10 The data files of input and output with simulation and implementation

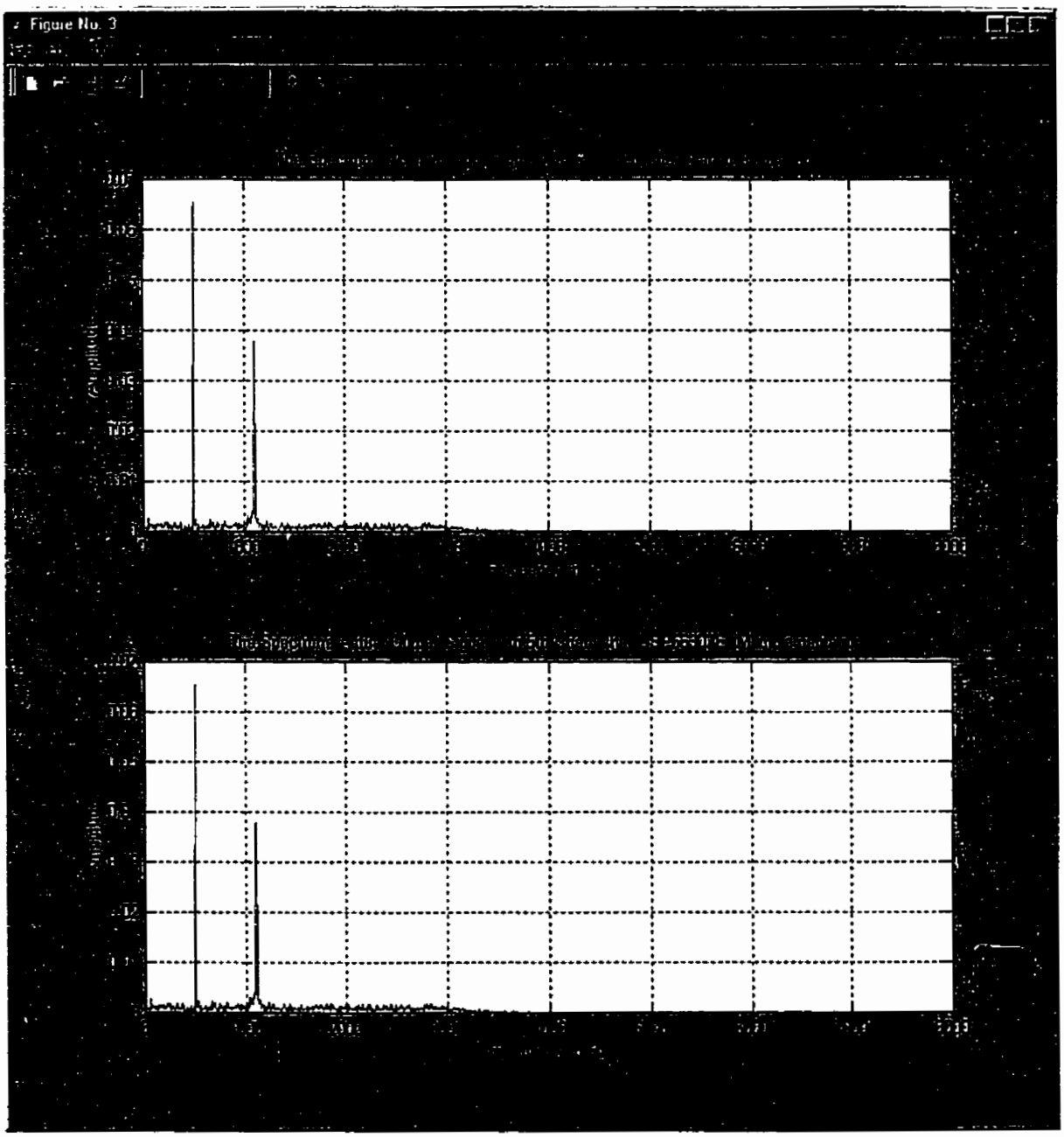


Figure A.11 Output Signal in the frequency domain with simulation and implementation

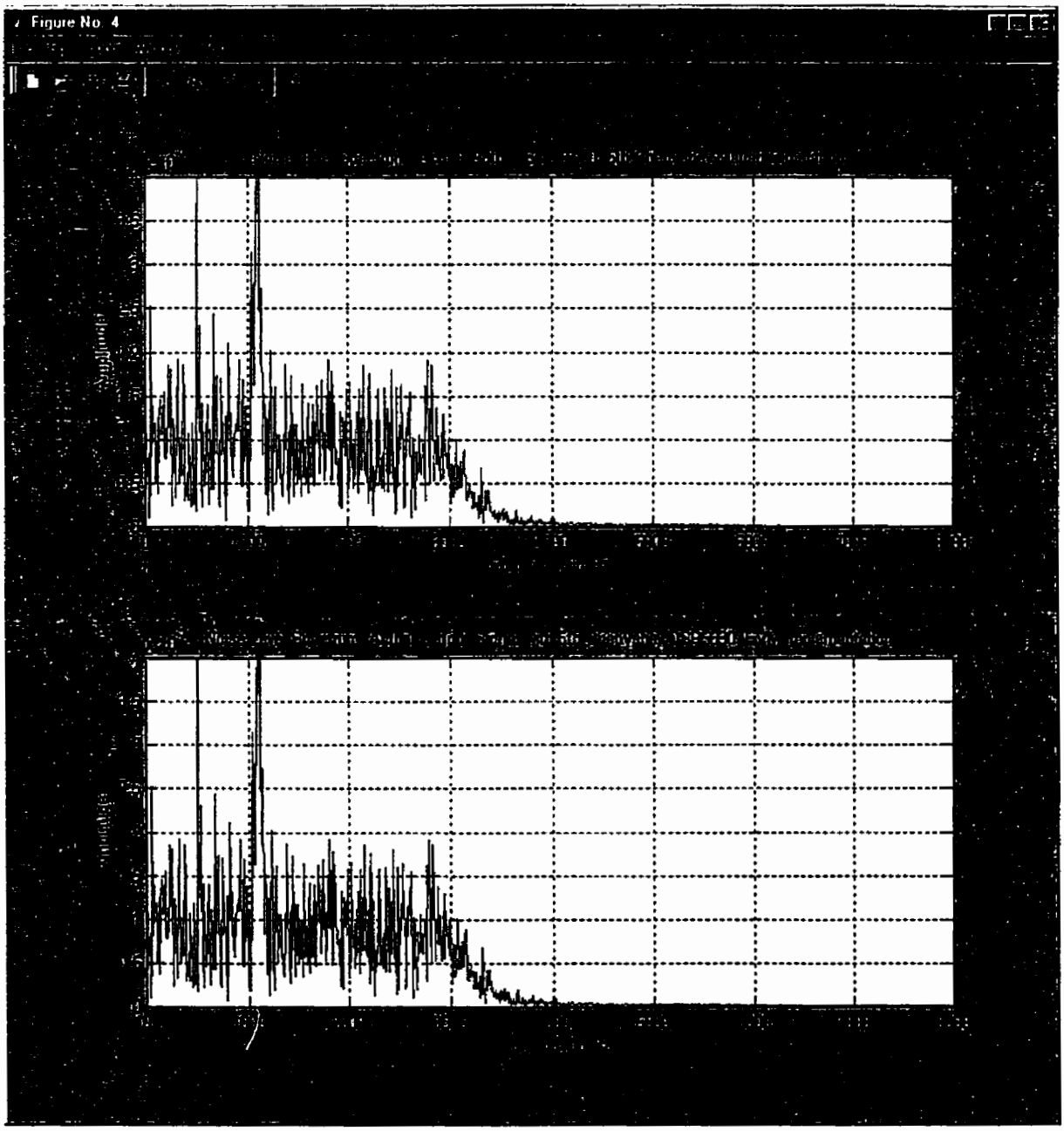


Figure A.12 Noise level spectrum of the output signal with simulation and implementation

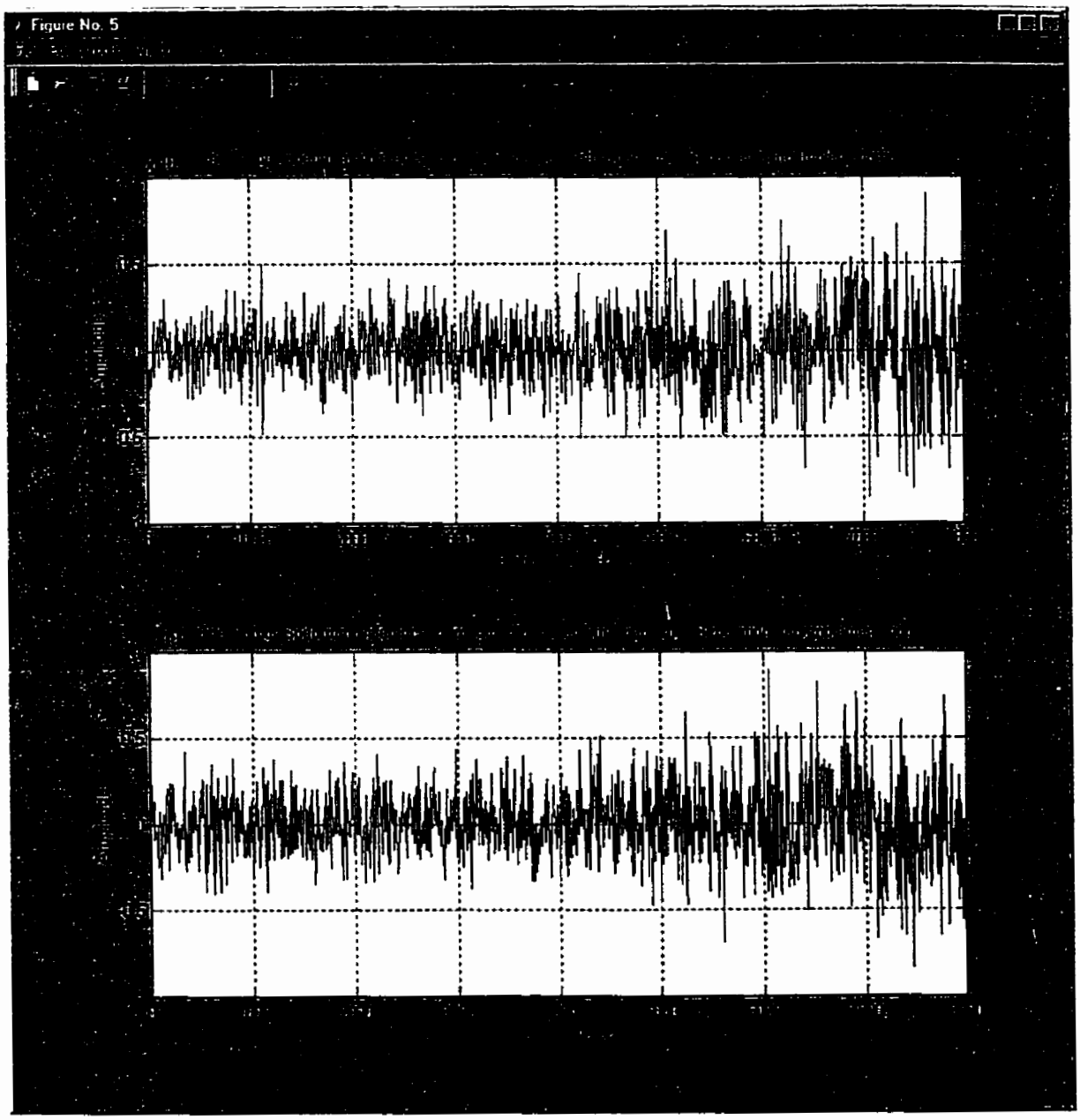


Figure A.13 Output: Real part and imaginary part difference with simulation and implementation

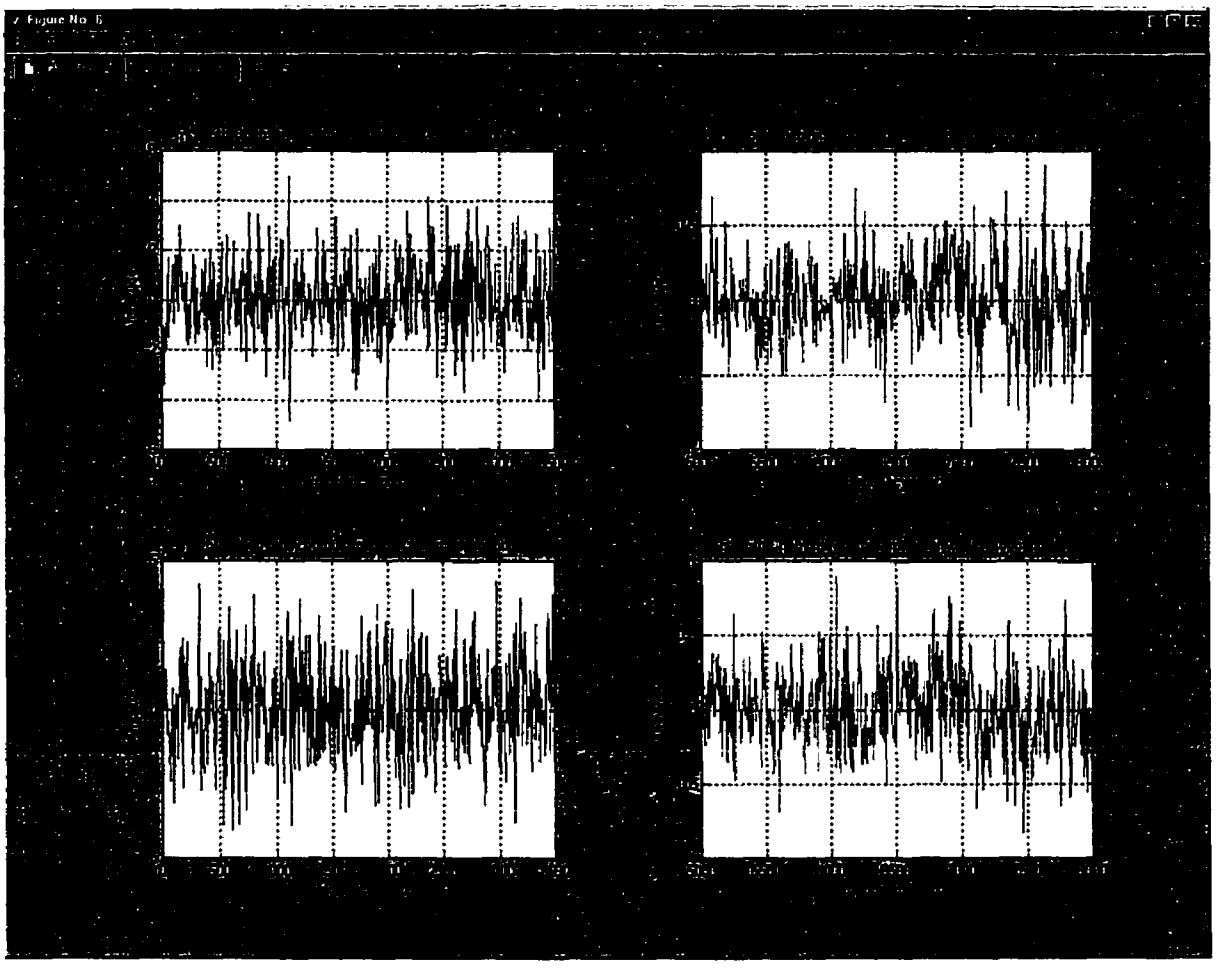


Figure A.14 Output: Real part and imaginary part difference in the passband and stopband

```
Comparison with Matlab simulation & DSP56387EVM implementation

by a 5th order Chebyshev WD Filter in terms of Parseval Relation


     Simulation in time domain: Input Signal Power = 13.80498097035438
     Simulation in freq domain: Input Signal Power = 13.80498097035437

  Implementation in time domain: Input Signal Power = 13.80498097035438
  Implementation in freq domain: Input Signal Power = 13.80498097035437

     Simulation in time domain: Output Signal Power =  6.49919761557241
     Simulation in freq domain: Output Signal Power =  6.49919761557241

  Implementation in time domain: Output Signal Power =  6.49919826111585
  Implementation in freq domain: Output Signal Power =  6.49919826111587

        The following is to compare power difference of output signal
                between simulation and DSP implementation

     Output Signal: Power Difference in time domain =  0.000000064554343
     Output Signal: Power Difference in freq domain =  0.000000064554346
```

Figure A.15 Power comparison: data file, ***print_sim_impl_power_comp_cheby5.txt***

Appendix B

The Results of a 7th order Butterworth WD Lowpass Filter

Example: A 7th order Butterworth WD lowpass filter design

General Specifications:

- Passband Frequency: = 3400 Hz
- Passband Attenuation: = 0.5 dB
- Stopband Frequency: = 6000 Hz
- Stopband Attenuation: = 55 dB
- Sampling Frequency: = 16000 Hz

```
Command Prompt

F:\WD_Filter_Design\common_source>butter_cal
**********************************************************
* To calculate the coefficients for Butterworth filter   *
* Please input the value:fp,ap,fs,as,F, respectively      *
*                                                        *
* Where:                                                 *
*       fp = upper edge frequency of the passband;       *
*       ap = maximum allowable attenuation in the passband;  *
*       fs = lower edge frequency of the stopband;       *
*       as = specified minimun attenuation in the stopband;  *
*        F = sampling frequency.                         *
*                                                        *
**********************************************************

            Passband Frequency:   fp = 3400
            Passband Attenuation: ap = 0.5
            Stopband Frequency:   fs = 6000
            Stopband Attenuation: as = 55
            Sampling Frequency:    F = 16000

**********************************************************
* Minimum filter Order is 6.592456                       *
* Please Choose the Filter Order N                       *
**********************************************************

            The Selected Filter Order: N = 7

**********************************************************

                coefficient_0  =    0.011606
                coefficient_1  =   -0.052229
                coefficient_2  =    0.023209
                coefficient_3  =   -0.232042
                coefficient_4  =    0.023209
                coefficient_5  =   -0.636044
                coefficient_6  =    0.023209

**********************************************************
*                                                        *
*       All coefficients are quantized to 24 bits.       *
*                                                        *
**********************************************************

F:\WD_Filter_Design\common_source>
```

Figure B.1 User interface of the coefficients for the Butterworth lowpass filter with the specifications

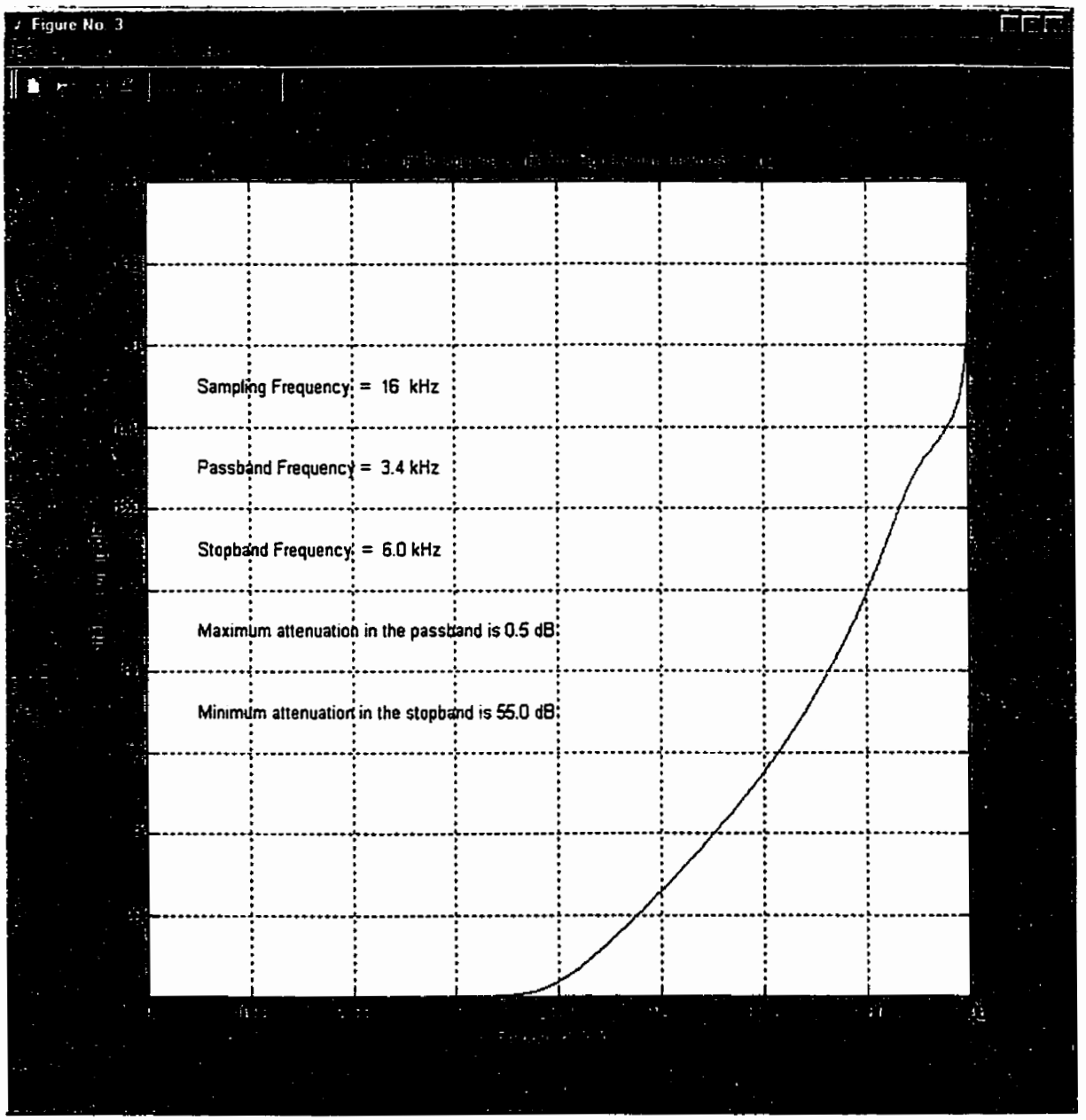


Figure B.2 The magnitude and attenuation response of the 7th order Butterworth WD lowpass filter

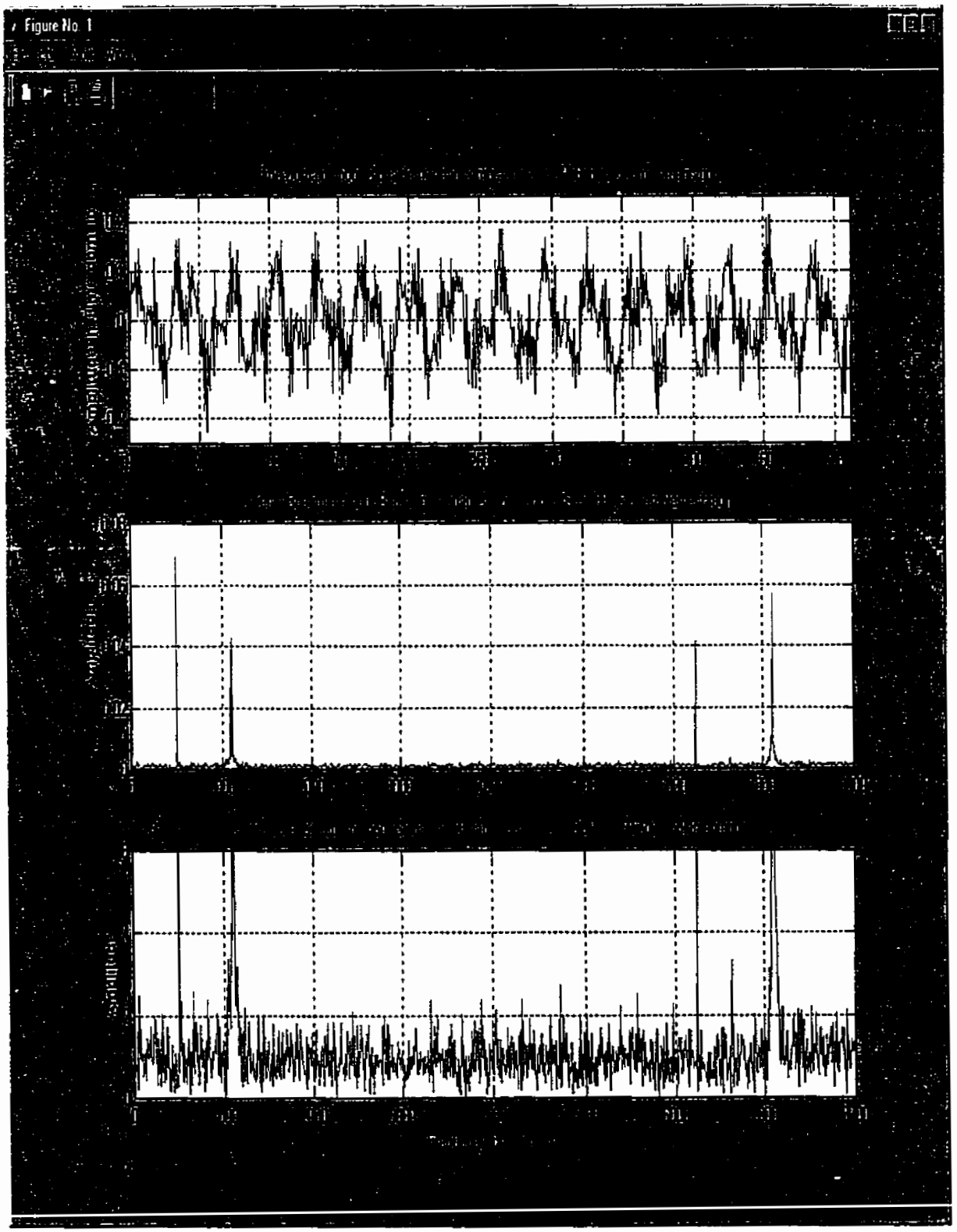


Figure B.3 The test vector in the time domain and frequency domain

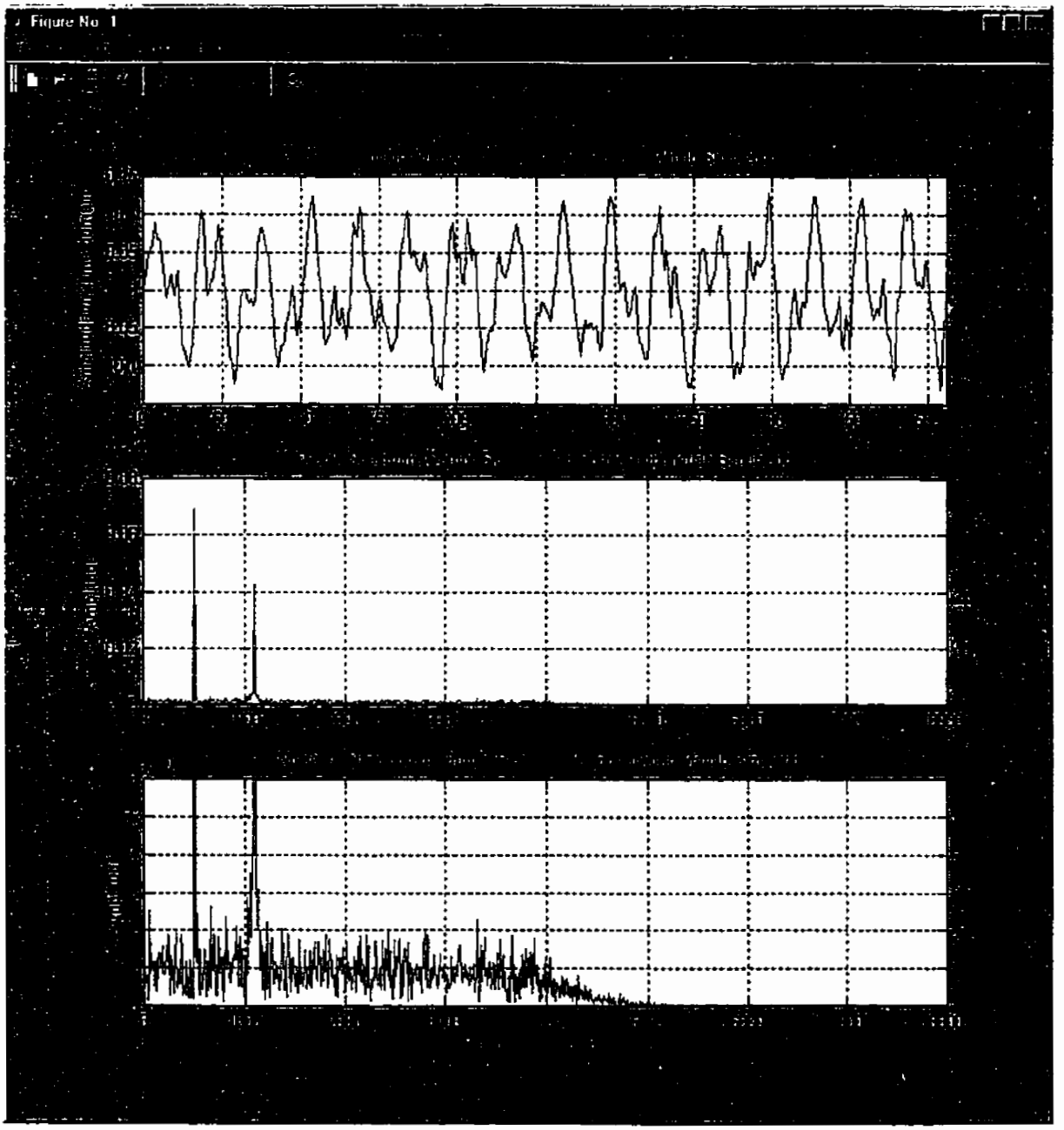


Figure B.4 Simulation: Output of the 7th order Butterworth lowpass filter for the test vector

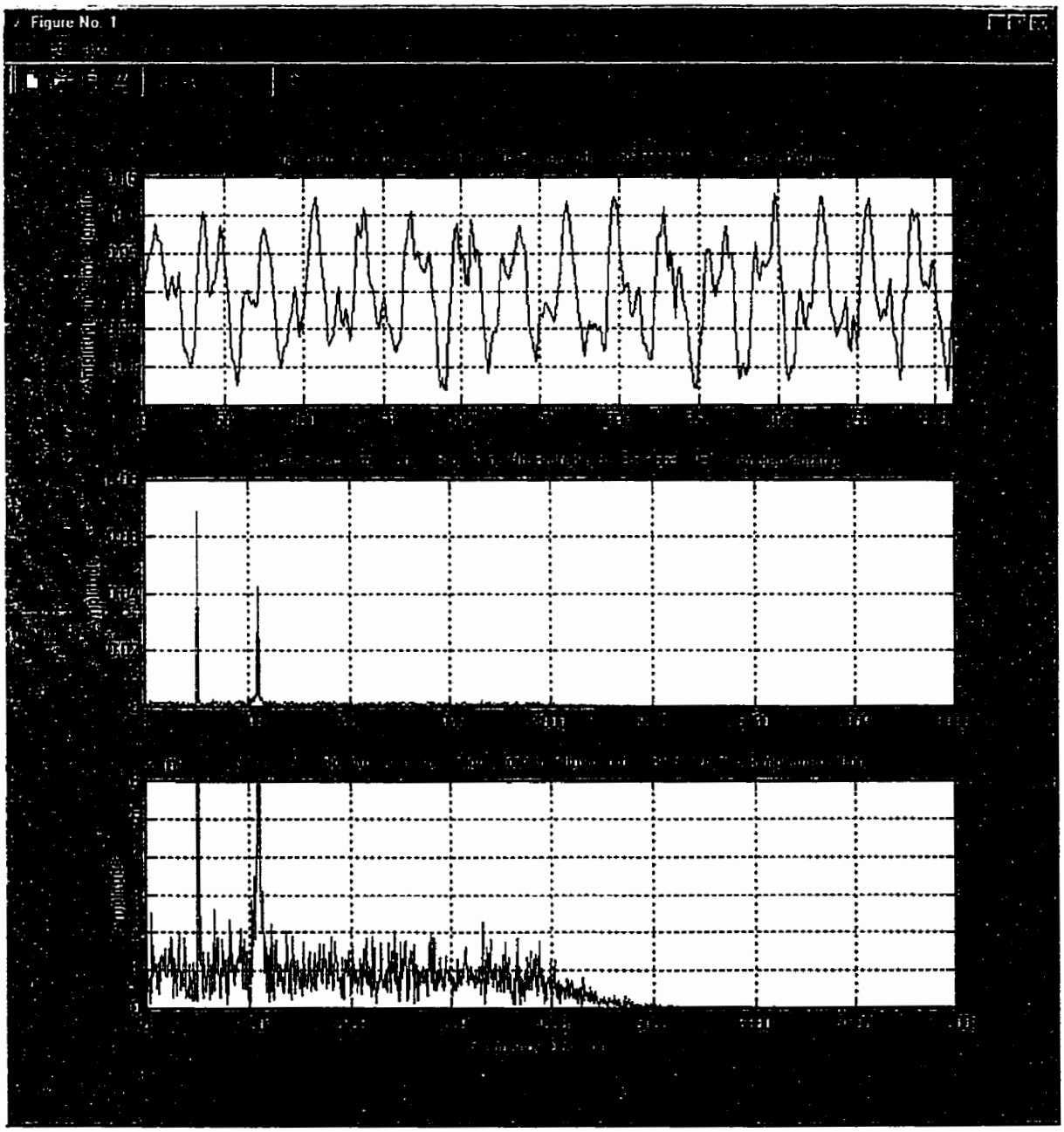


Figure B.5 Implementation: Output of the 7th order Butterworth lowpass filter for the test vector

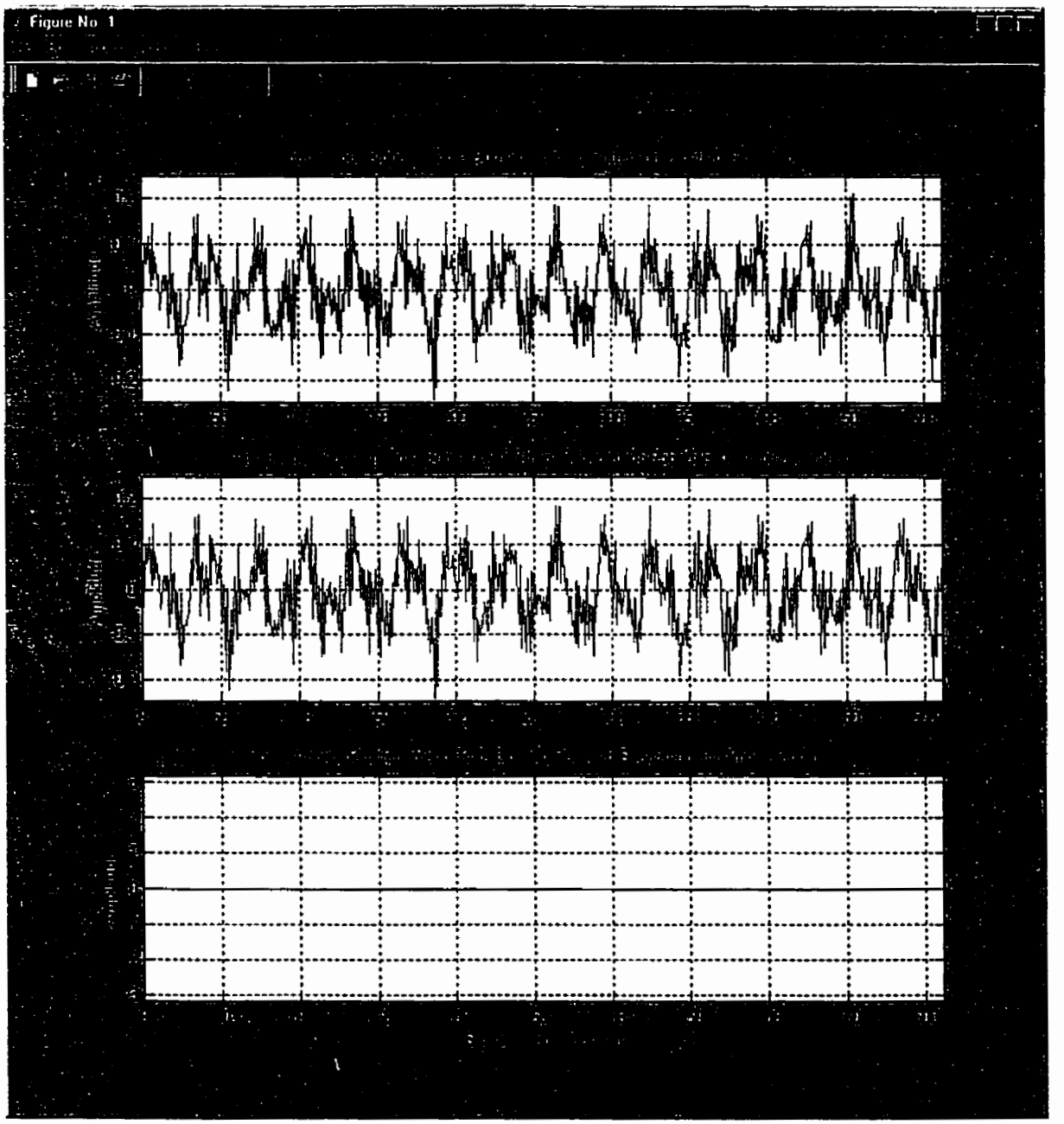


Figure B.6 Input Signal in the time domain with simulation and implementation

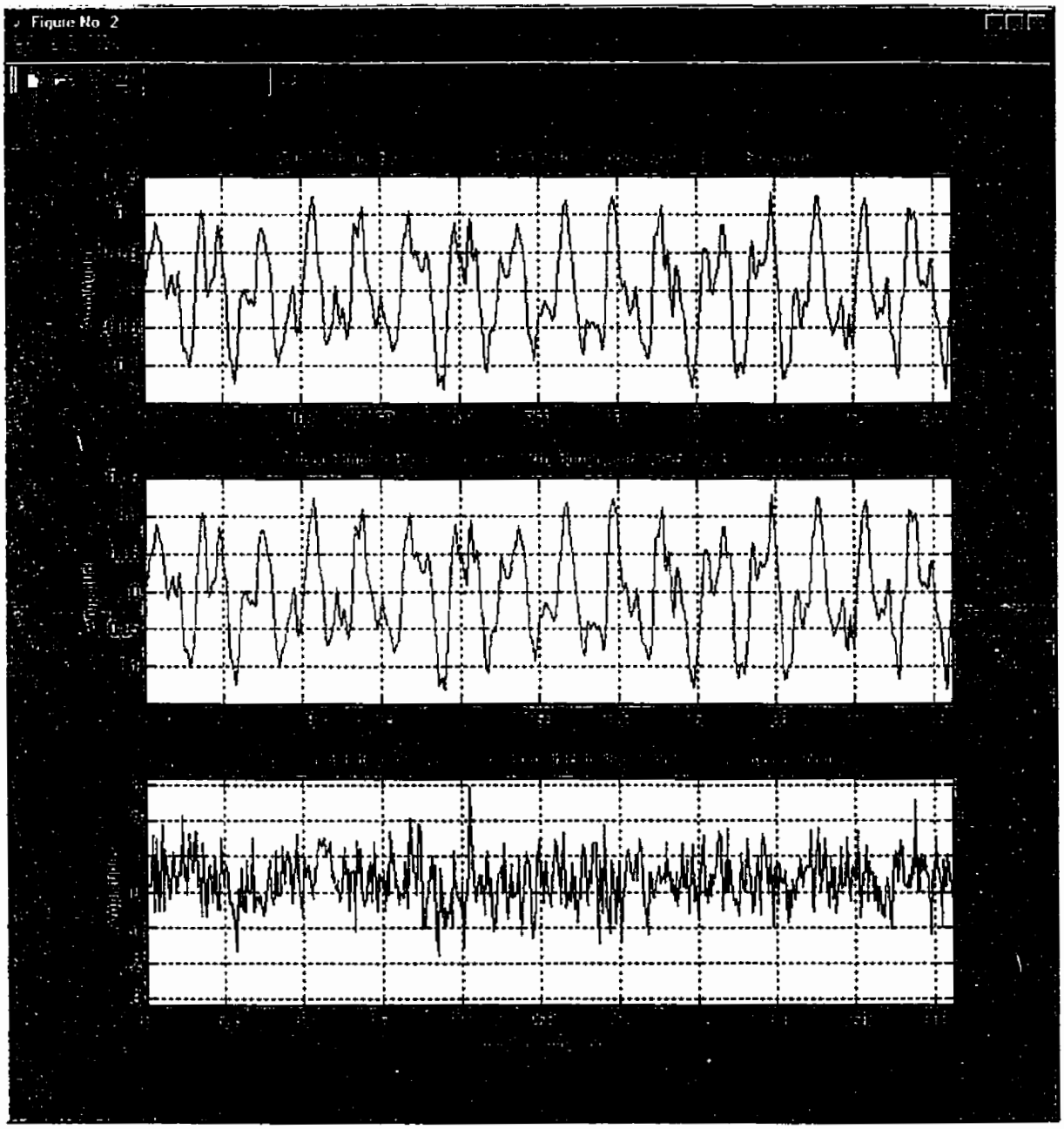


Figure B.7 Output Signal in the time domain with simulation and implementation

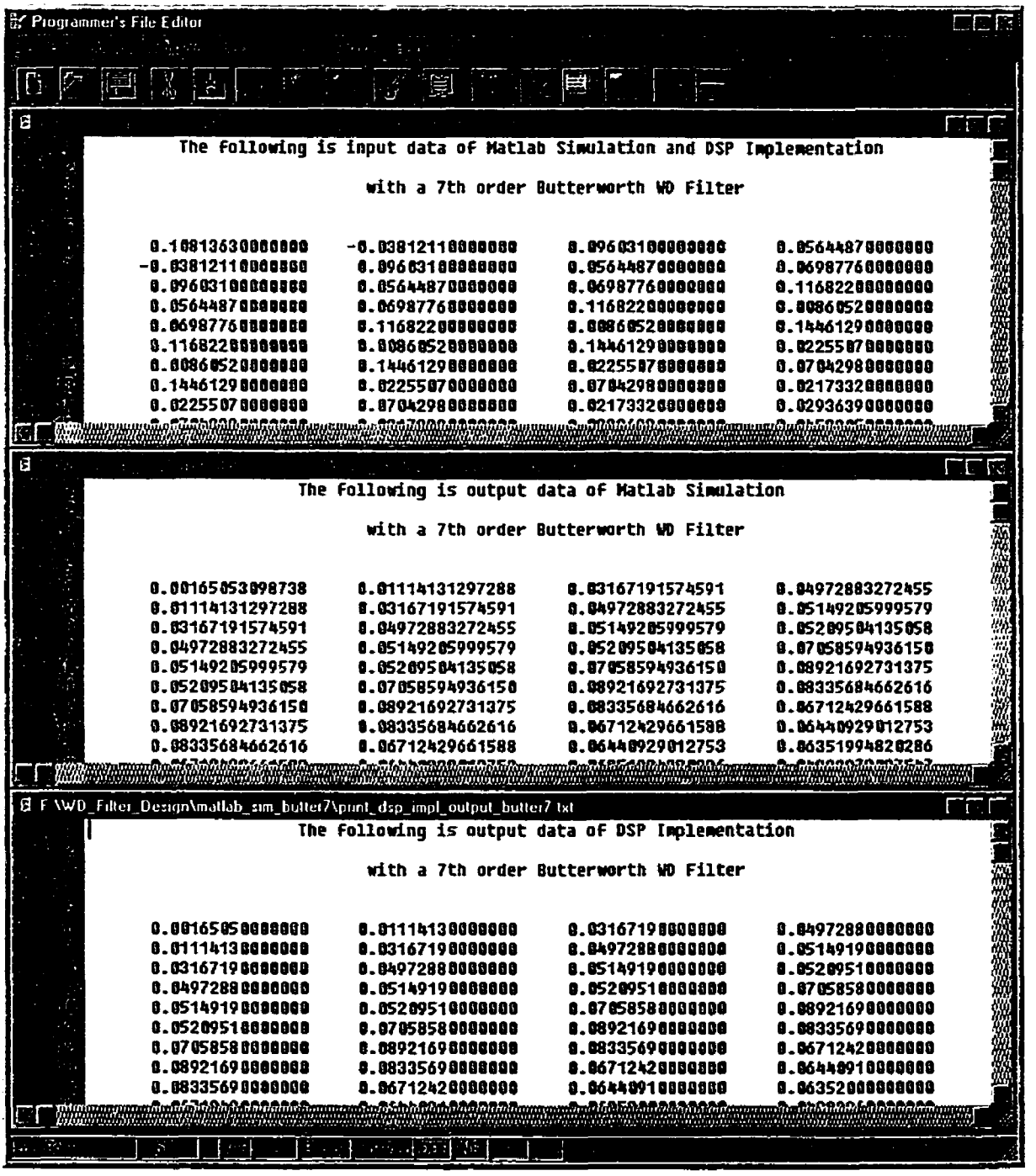


Figure B.8 The data files of input and output with simulation and implementation

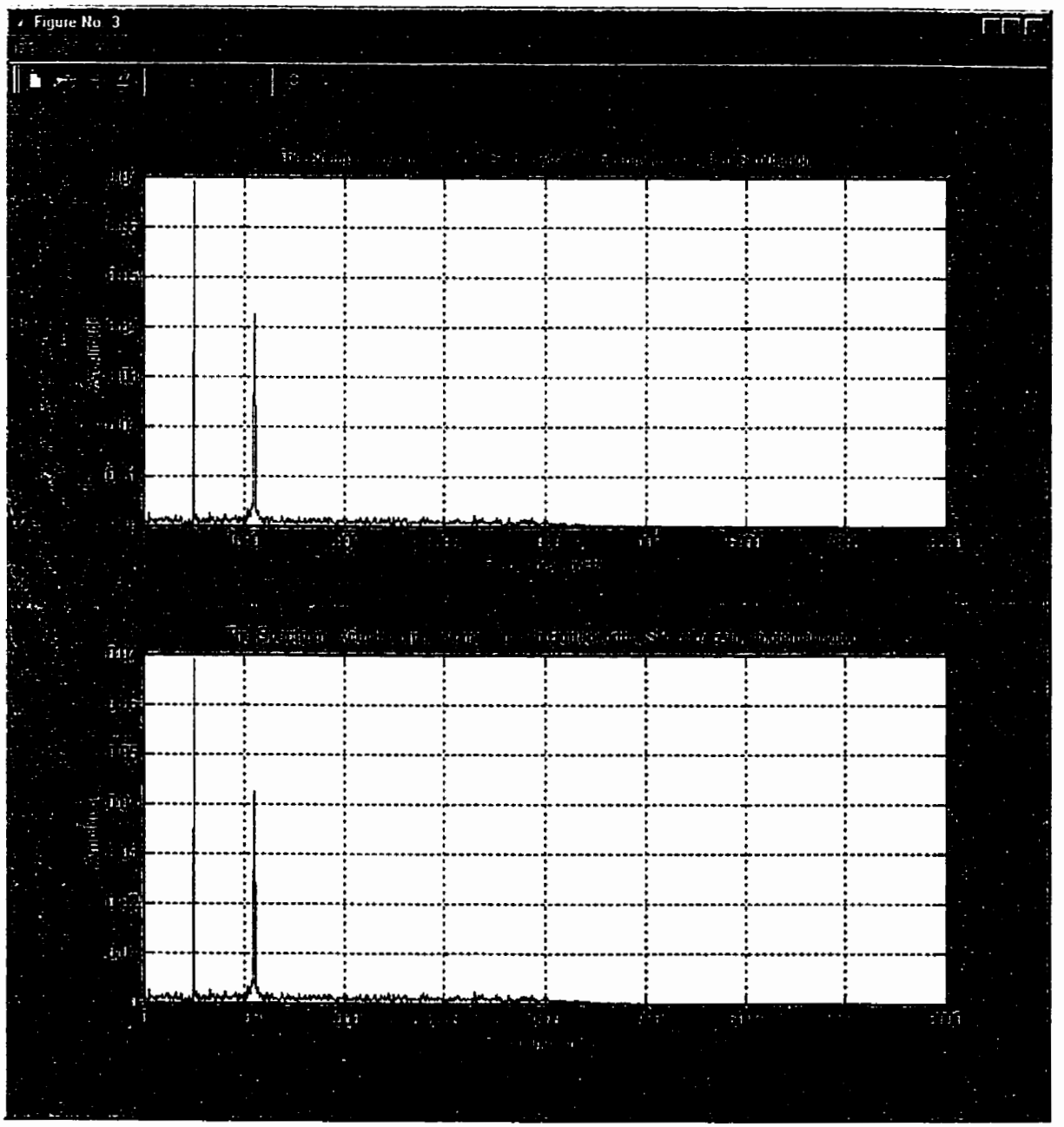


Figure B.9 Output Signal in the frequency domain with simulation and implementation

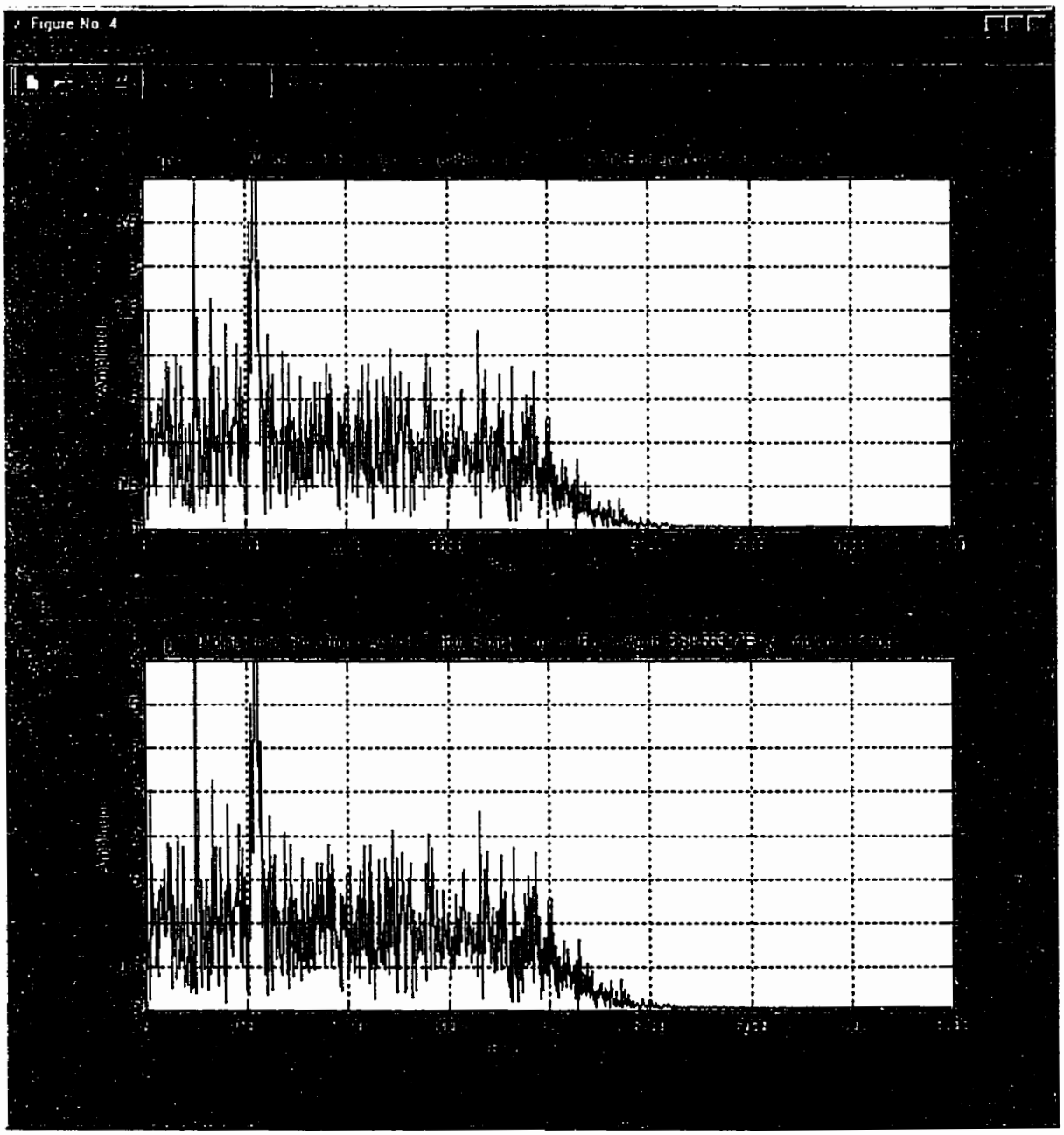


Figure B.10 Noise level spectrum of the output signal with simulation and implementation

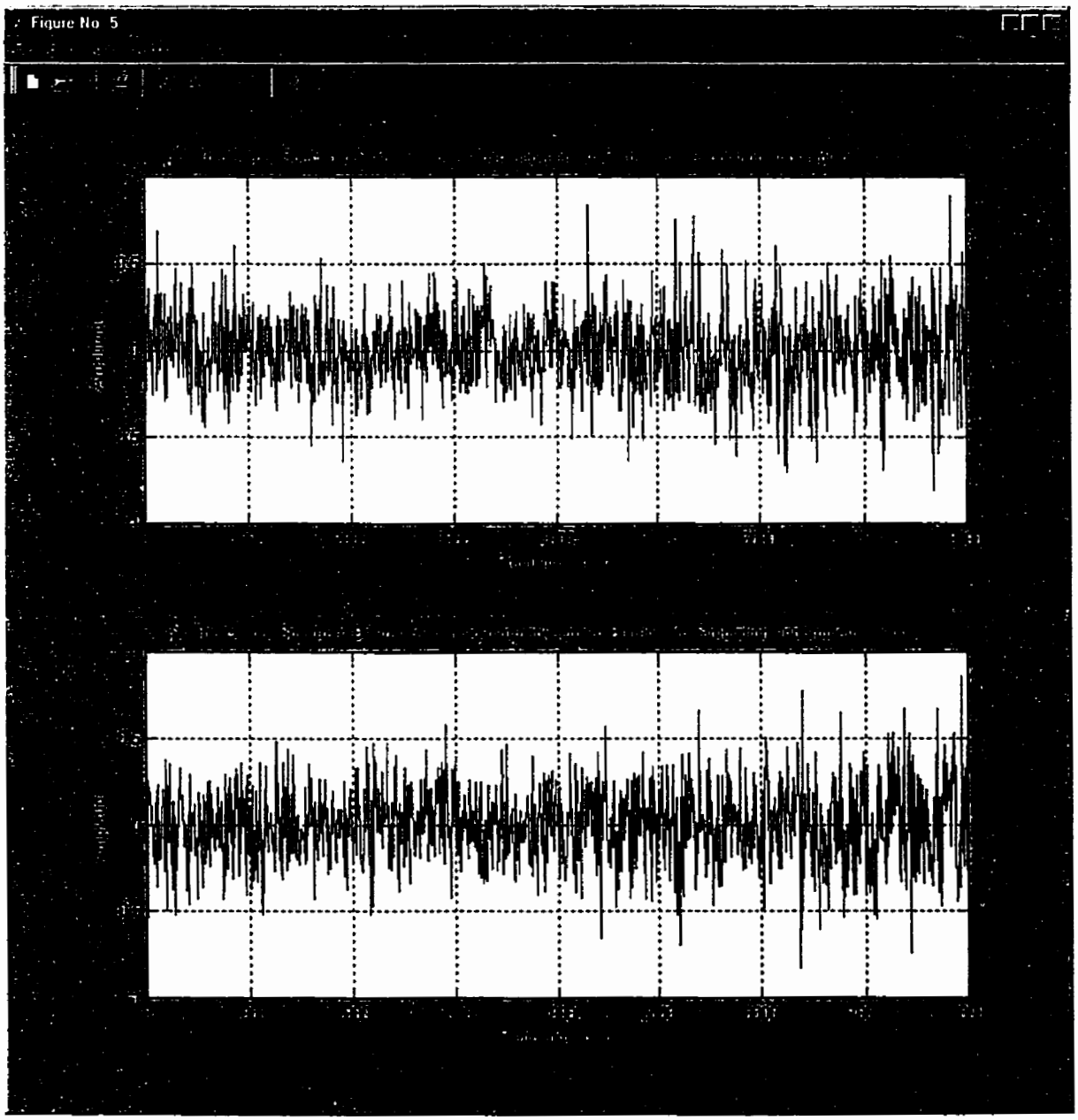


Figure B.11 Output: Real part and imaginary part difference with simulation and implementation

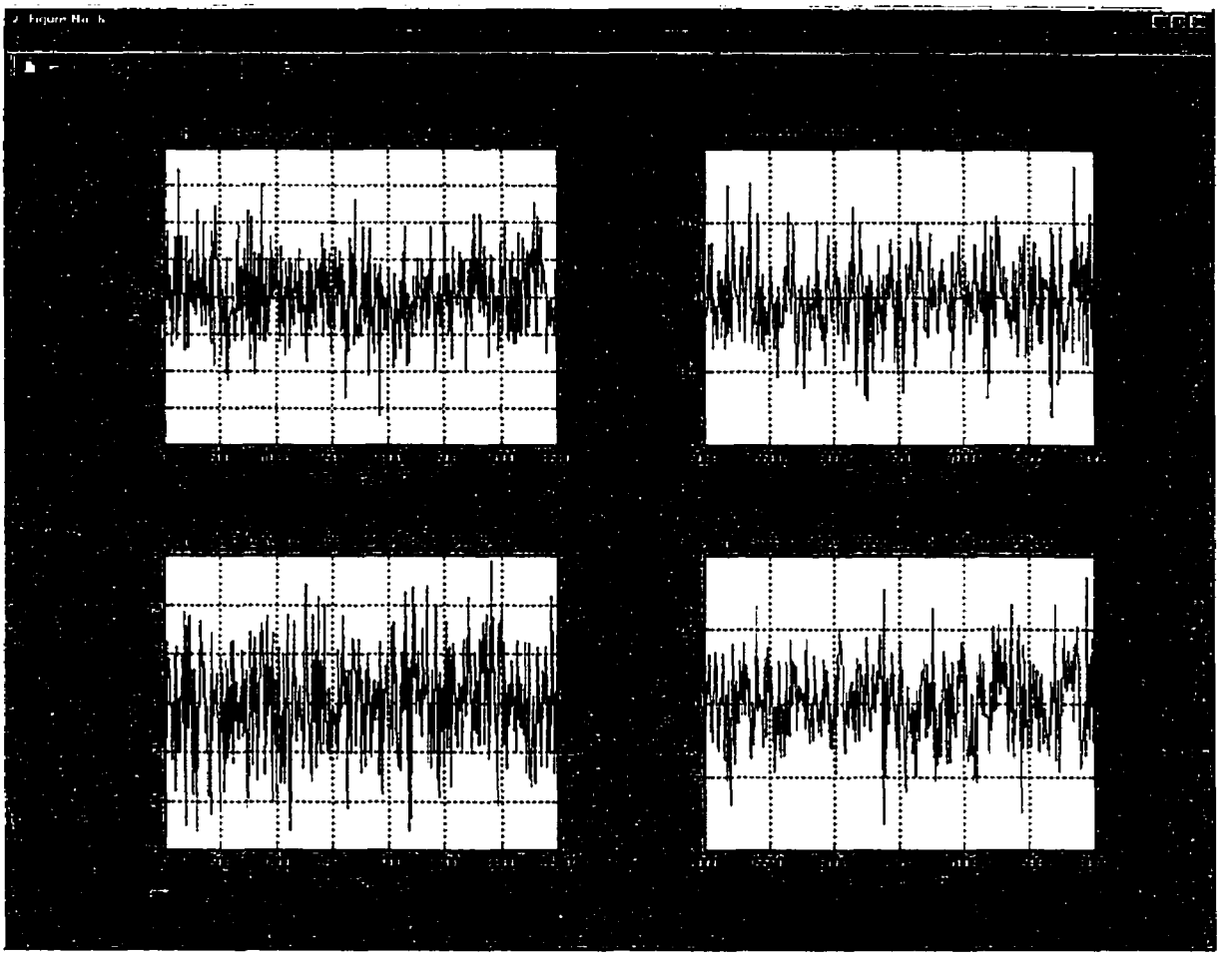


Figure B.12 Output: Real part and imaginary part difference in the passband and stopband

```
Comparison with Matlab simulation & DSP56307EVM implementation

by a 7th order Butterworth WD Filter in terms of Parseval Relation


     Simulation in time domain: Input Signal Power = 13.804980970354 38
     Simulation in freq domain: Input Signal Power = 13.804980970354 37

  Implementation in time domain: Input Signal Power = 13.804980970354 38
  Implementation in freq domain: Input Signal Power = 13.804980970354 37

     Simulation in time domain: Output Signal Power =  7.612789244415 98
     Simulation in freq domain: Output Signal Power =  7.612789244415 97

  Implementation in time domain: Output Signal Power =  7.612709437744 99
  Implementation in freq domain: Output Signal Power =  7.612709437744 99

        The following is to compare power difference of output signal
              between simulation and DSP implementation

     Output Signal: Power Difference in time domain =  0.000080193329 02
     Output Signal: Power Difference in freq domain =  0.000080193329 02
```

Figure B.13 Power comparison: data file, *print_sim_impl_power_comp_butter7.txt*

Appendix C

The List of the Design Tools

- MS-Visual C++ 5.0 Compiler

This tool is used to compile the source codes for coefficient calculation, written in C, it generates an executable file, such as **butter_cal.exe** to calculate the coefficients of a Butterworth type filter.

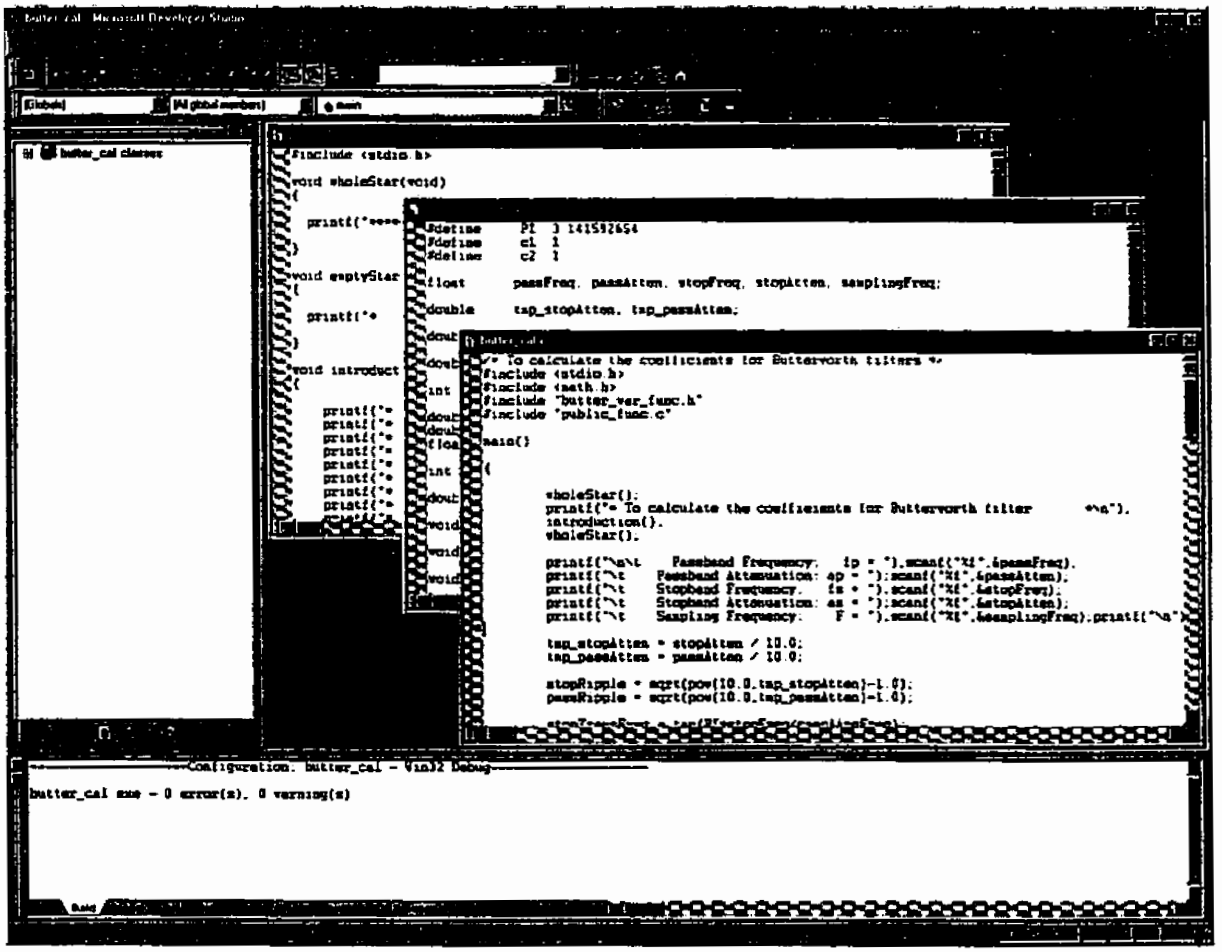


Figure C.1 An example by using Visual C++ Compiler

- MS-DOS

An MS-DOS window is used to implement the calculation of the coefficients for the particular type of lowpass filter designed, which will be run in simulator & EVM.

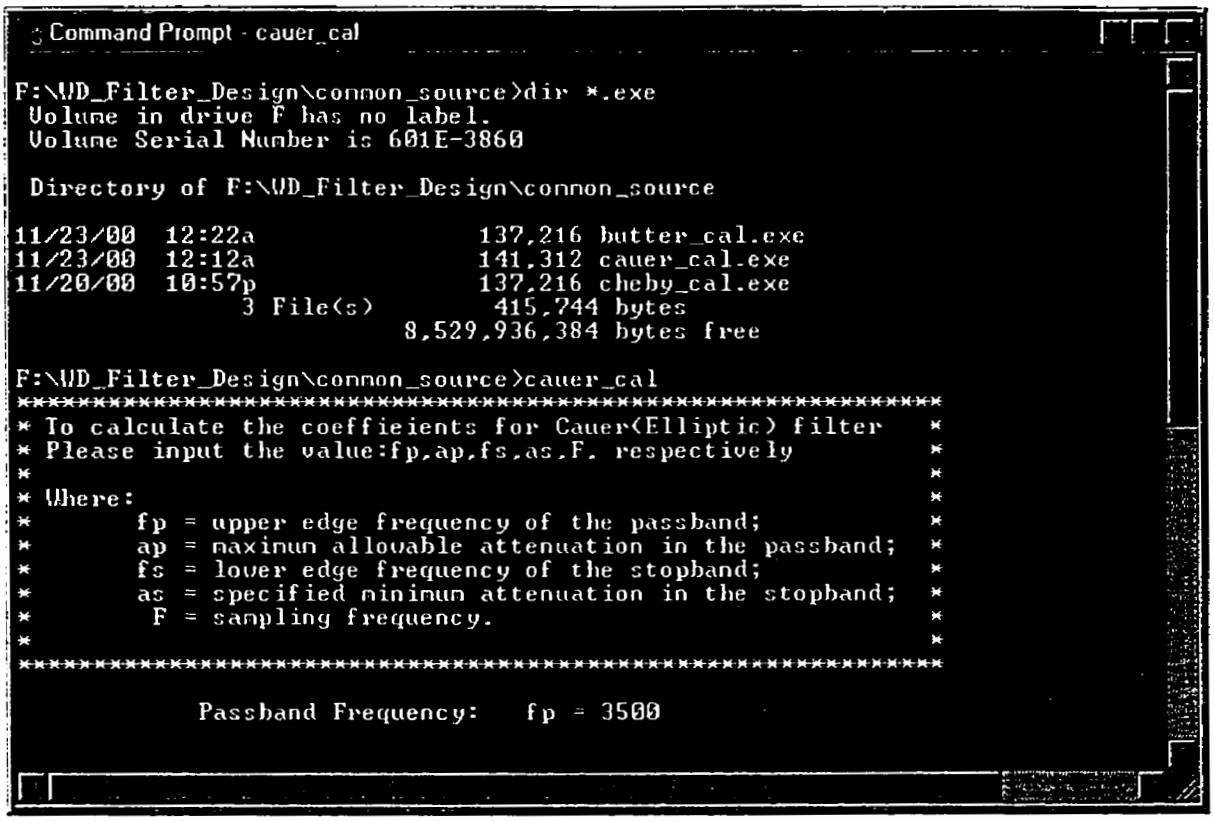


Figure C.2 An example by using MS-DOS window to run **cauer_cal**

- MATLAB 5.30

This tool is used to run the simulation of the filter design, including the filter characteristics, test vector generations, and the analysis of result comparing between the simulation in Matlab & implementation in DSP56307EVM board.

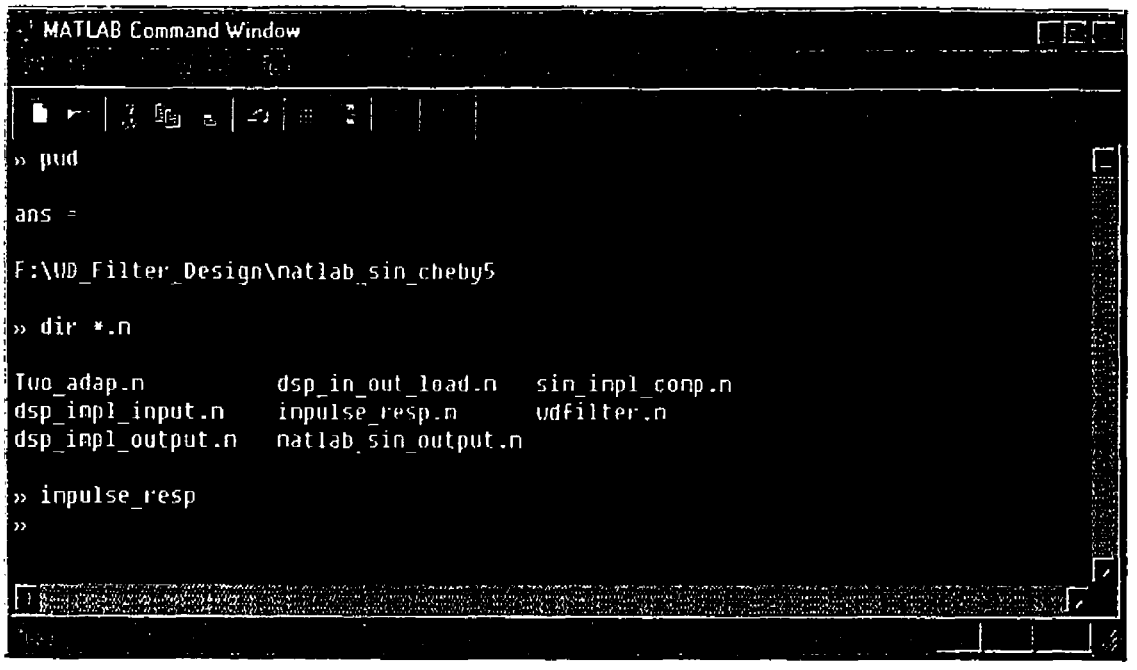


Figure C.3 An example by using Matlab Command Window

- TASKING DSP563xx EDE Source Compiler

The tool is a total programming environment that integrates all the tools a developer needs to create, edit, build, and debug an embedded application, including a language-sensitive editor, easy configuration of compiler, assembler, and linker options, automatic build using MAKE, customizable working and application environment. We compile the source code for implementation with this tool, and generate an executable, such as **9th_cauerfilter.abs**, which is downloadable to the DSP56307EVM board.

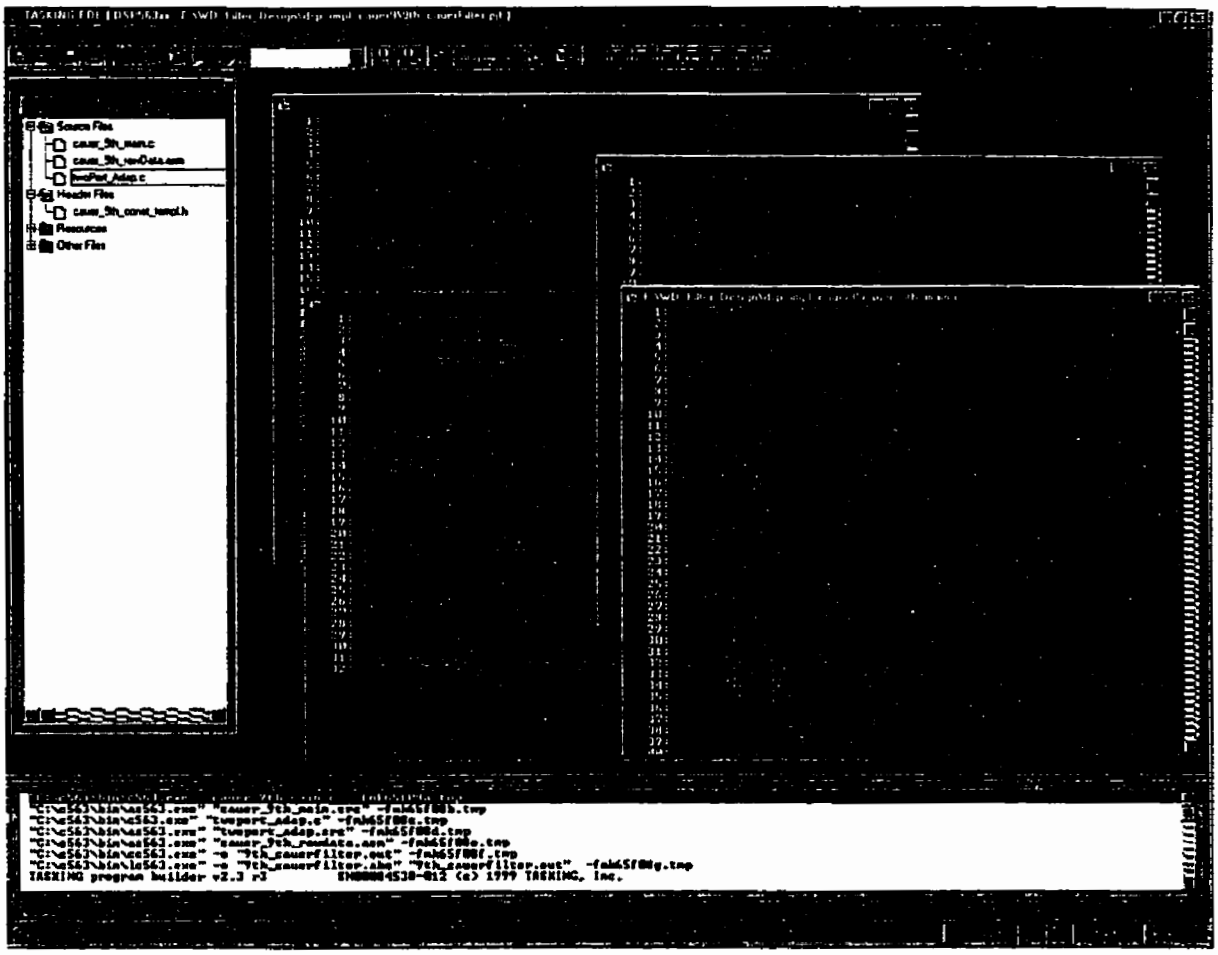


Figure C.4 An example by using TASKING DSP563xx EDE

- CrossView Pro DSP56xxx Debugger

CrossView Pro is a debugger for EVM, it provides features and functionality to help shorten the debug session, determine performance bottlenecks, uncover additional information, and test the application, which has multi-viewing windows, code and software data breakpoints, single stepping without stopping, register grouping, direct memory, C and assembly level trace and stack tracing, etc. We use this tool to execute the executable, such as **9th_cauerfilter.abs**, to run the filter in the EVM board and collect input and output data with a pre-set vector.

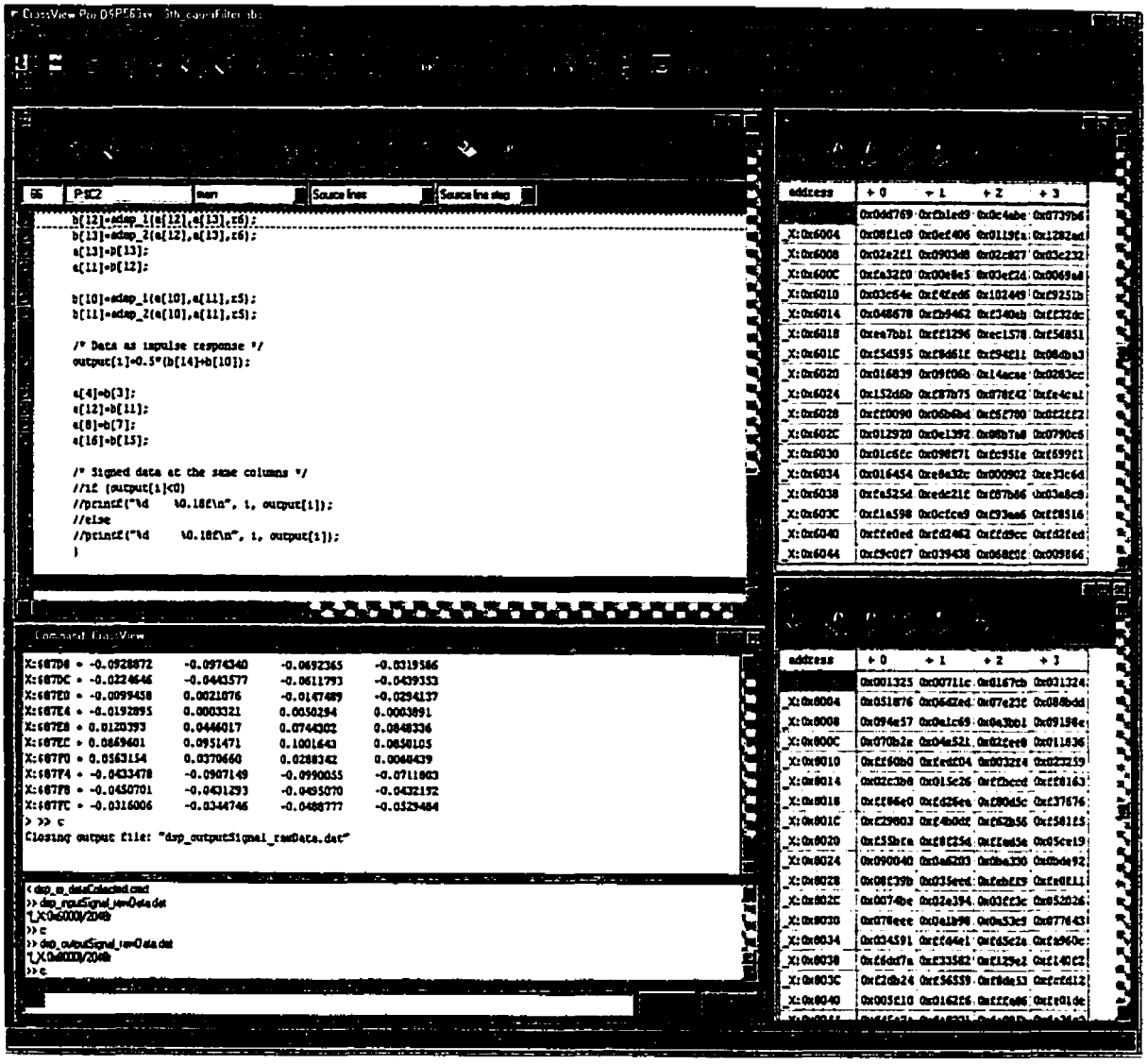


Figure C.5 An example by using CrossView Pro DSP56xxx Debugger

Appendix D

The Features of the Motorola DSP56307EVM

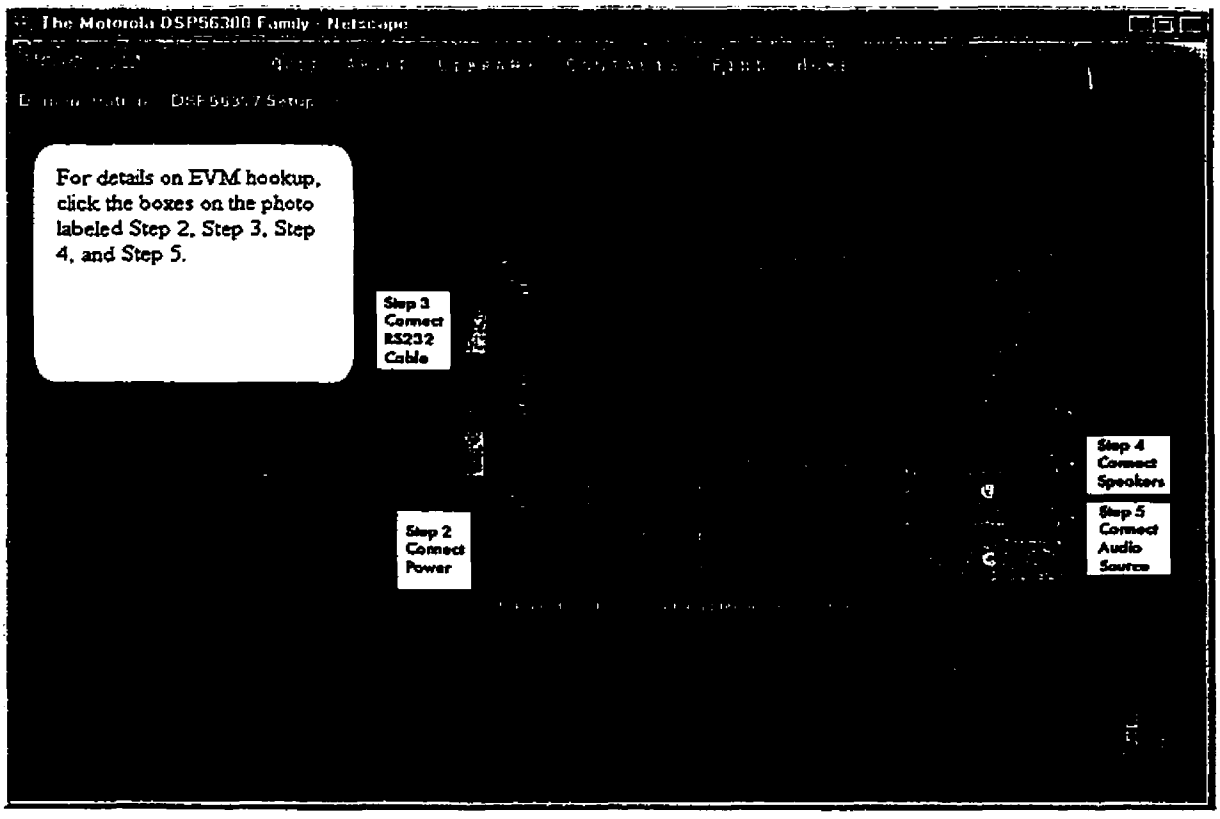


Figure D.1 A Motorola DSP56307EVM processor

The DSP56307 Evaluation Module (DSP56307EVM) is a low-cost platform for developing real-time software and hardware products to support a new generation of wireless, telecommunications, and multimedia applications. The DSP56307EVM targets applications requiring a large amount of on-chip memory such as wireless infrastructure applications. The Enhanced Filter Coprocessor (EFCOP) can accelerate general filtering algorithms, such as finite impulse response (FIR) filters, infinite impulse response (IIR) filters; the EFCOP can also adapt FIR filters as multi-channel filters used in echo cancellation, correlation, and general-purpose convolution-based algorithms. The user can download software to on-chip or on-board RAM, then run and debug it. The user can also connect hardware, such as external memories and analog-to-digital (A/D) or digital-to-analog (D/A) converters, for product development. The 24-bit precision or the DSP56307 Digital Signal Processor

(DSP) combined with the on-board 64K of external SRAM and Crystal Semiconductor's CS4218 stereo, CD-quality, audio codec ideally suits the DSP56307EVM for implementing and demonstrating many communications and audio processing algorithms. It is also an effective tool for learning the architecture and instruction set of the DSP56307 processor.

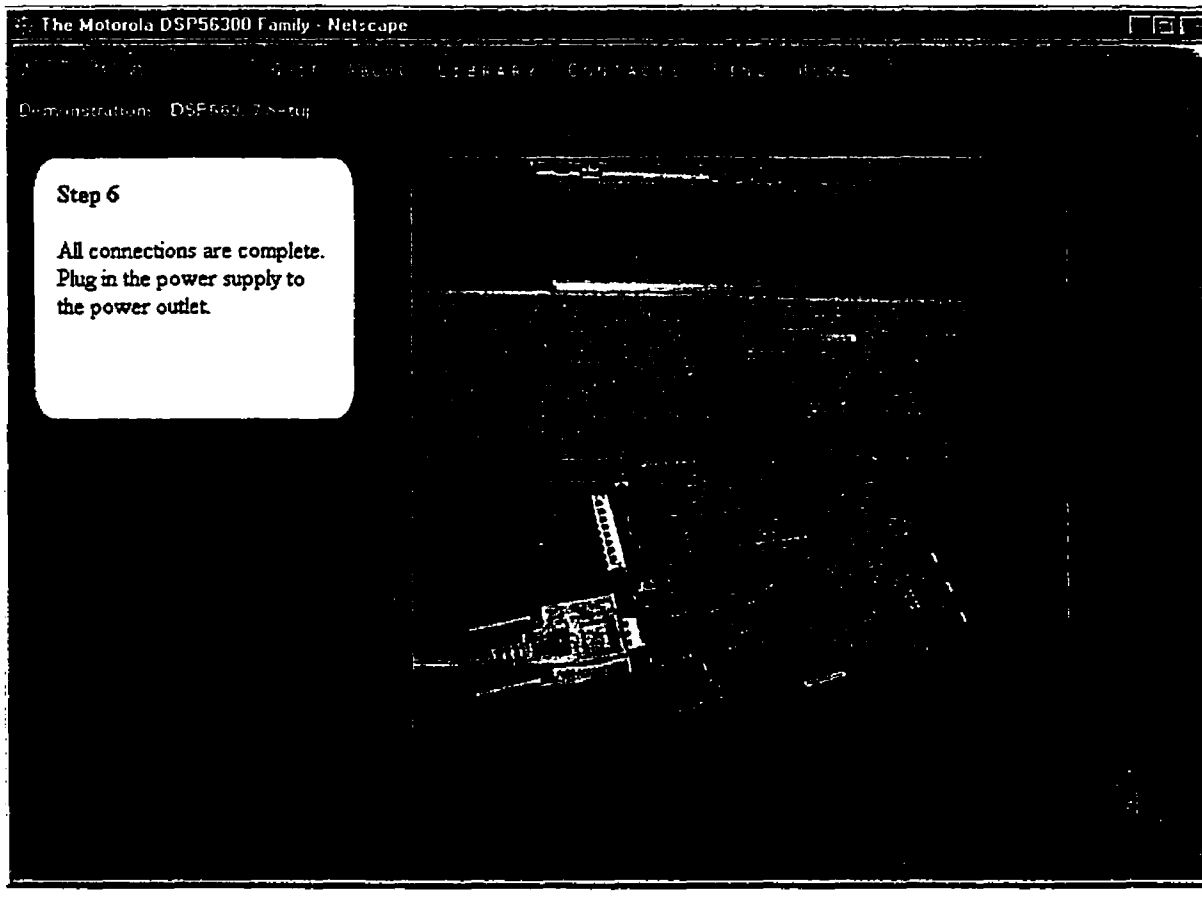


Figure D.2 The connection between DSP56307EVM and working environment

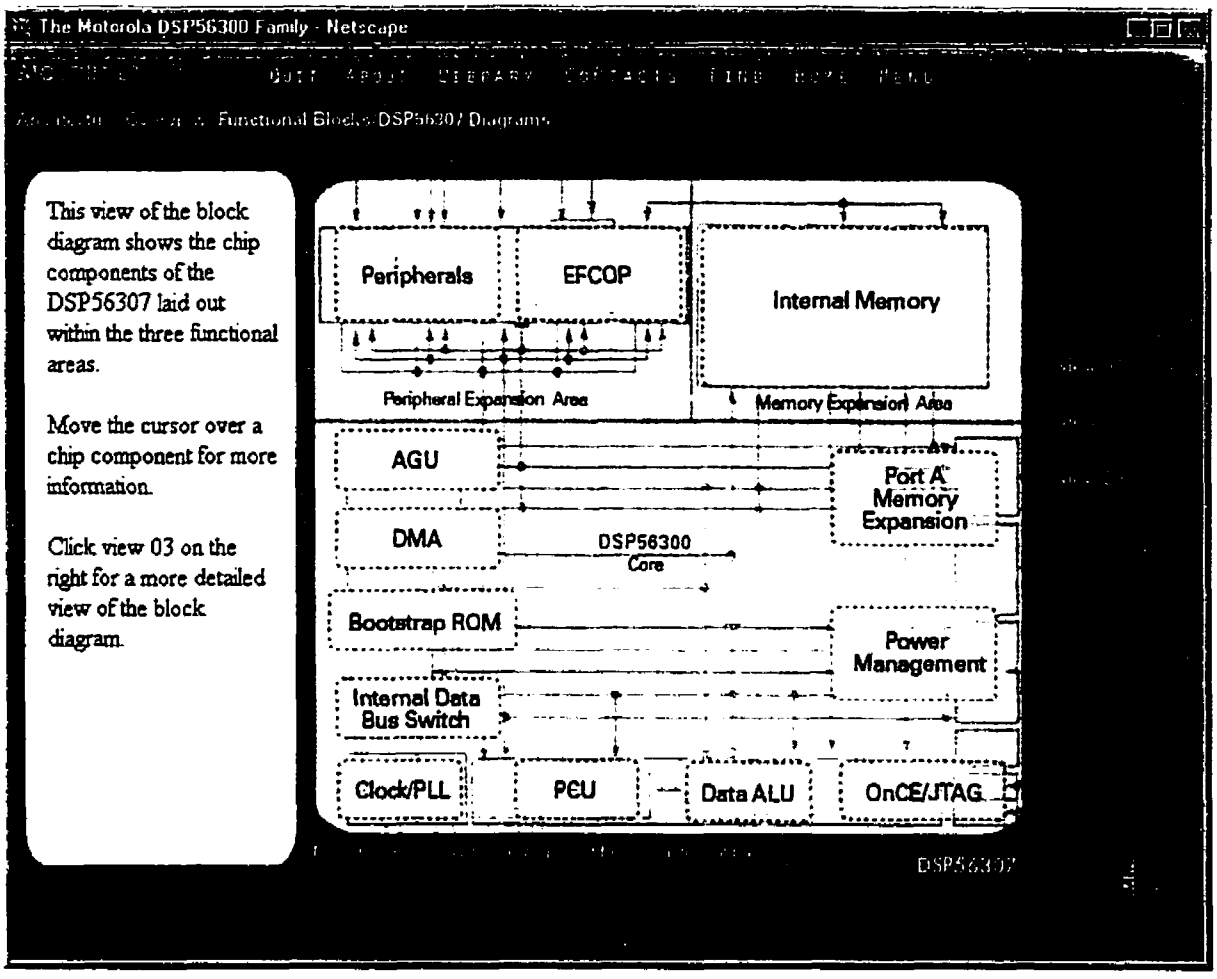


Figure D.3 The functional block diagram of DSP56307EVM

- DSP56307EVM Hardware Features:
  - high performance DSP56307 core: 100 MIPS (Million Instructions Per Second), 100 MHz clock, data ALU (Arithmetic Logic Unit), 6 channel DMA (Data Memory Access) controller, on-chip emulation (OnCE) module
  - enhanced filter coprocessor (EFCOP): on-chip filtering and echo-cancellation coprocessor runs in parallel to the DSP core
  - on-chip memories: 64K on-chip RAM, instruction cache
  - 2 channels of 24-bit A/D conversion
  - 2 channels of 24-bit D/A conversion
  - 2 ESSI (Enhanced Synchronous Serial Interface) connectors
  - SCI (Serial Communications Interface) connectors

Appendix E

The Source Codes for WD Lowpass Filter Design

In the appendix, we enclose all source codes (Matlab's, C's and script's) as references. Some files have the same names, but serve different design purposes. To produce a quality visualization (unconvertible between MS-word editor and text editor) for readers, we use a text file format appended to all of them. Each file has a file name and location in the top bar. We split all files into 3 groups in order, according to 3 types of filter designs as follows:

- Group 1: 9^{th} order Cauer (Elliptic) lowpass WD filter design
- Group 2: 5^{th} order Chebyshev lowpass WD filter design
- Group 3: 7^{th} order Butterworth lowpass WD filter design

Note: All files in the **File Name List (Omitted)** of Group 2 or Group 3 mean that the codes are exactly same as the files in the **File Name List (Presented)** of Group 1, correspondingly.

Group 1: 9^{th} order Cauer (Elliptic) lowpass WD filter design

File Name List (Presented)

- public_func.c
- cauer_func.c
- cauer_var_func.h
- cauer_cal.c
- Two_adap.m
- wdfilter.m
- impulse_resp.m
- testingSignal_geal.m
- matlab_sim_input.m
- matlab_sim_output.m
- twoPort_adap.c
- cauer_9^{th}_const_templ.h
- cauer_9^{th}_main.c
- dsp_io_dataCollected.cmd
- dsp_in_out_load.m
- dsp_impl_input.m
- dsp_impl_output.m
- sim_impl_comp.m

Group 2: 5th order Chebyshev lowpass WD filter design

File Name List (Presented)

- cheby5_var_func.h
- cheby5_cal.c
- wdfilter.m
- impulse_resp.m
- matlab_sim_output.m
- cheby_5th_const_templ.h
- cheby_5th_main.c
- dsp_in_out_load.m
- dsp_impl_input.m
- dsp_impl_output.m
- sim_impl_comp.m

File Name List (Omitted)

- public_func.c
- Two_adap.m
- testingSignal_geal.m
- matlab_sim_input.m
- twoPort_adap.c
- dsp_io_dataCollected.cmd

Group 3: 7^{th} order Butterworth lowpass WD filter design

File Name List (Presented)

- butter7_var_func.h
- butter7_cal.c
- wdfilter.m
- impulse_resp.m
- matlab_sim_output.m
- butter_7^{th}_const_templ.h
- butter_7^{th}_main.c
- dsp_in_out_load.m
- dsp_impl_input.m
- dsp_impl_output.m
- sim_impl_comp.m

File Name List (Omitted)

- public_func.c
- Two_adap.m
- testingSignal_geal.m
- matlab_sim_input.m
- twoPort_adap.c
- dsp_io_dataCollected.cmd

Directory: P:\WD_Filter_Design\common_source

File Name:

1. cauer_cal.c

```
/* To calculate the coefficients for Cauer (Elliptic) filters */

#include <stdio.h>
#include <math.h>
#include "cauer_var_func.h"
#include "cauer_func.c"
#include "public_func.c"

main()

{

        wholeStar();
        printf("* To calculate the coeffieients for Cauer(Elliptic) filter   *\n");
        introduction();
        wholeStar();

        printf("\n\t    Passband Frequency:   fp = ");scanf("%f",&passFreq);
        printf("\t    Passband Attenuation: ap = ");scanf("%f",&passAtten);
        printf("\t    Stopband Frequency:   fs = ");scanf("%f",&stopFreq);
        printf("\t    Stopband Attenuation: as = ");scanf("%f",&stopAtten);
        printf("\t    Sampling Frequency:    F = ");scanf("%f",&samplingFreq);printf("\n");


        tmp_stopAtten = stopAtten / 10.0;
        tmp_passAtten = passAtten / 10.0;

        stopRipple = sqrt(pow(10.0,tmp_stopAtten)-1.0);
        passRipple = sqrt(pow(10.0,tmp_passAtten)-1.0);


        stopTransFreq = tan(PI*stopFreq/samplingFreq);
        passTransFreq = tan(PI*passFreq/samplingFreq);

        aux_k0 = sqrt(stopTransFreq/passTransFreq);
        aux_k4 = cross_k0(aux_k0);

        aux_c3 = 2.0*aux_k4;
        minFilterDegree = (c1*log(c2*stopRipple/passRipple))/log(aux_c3);


        wholeStar();

    if(minFilterDegree < 10.0)

        printf("* Minimum filter Order is %f                          *\n",minFilterDegree);
    else

        printf("* Minimum filter Order is %f                         *\n",minFilterDegree);
        printf("* Please Choose the Filter Order N                        *\n");
        wholeStar();
        printf("\n\t    The Selected Filter Order: N = ");
        scanf("%d",&selectedFilterDegree);printf("\n");
        wholeStar();

    if (selectedFilterDegree < 10)

        printf("* The filter order selected is N = %d                       *\n",selectedFilterDegree);

    else

        printf("* The filter order selected is N = %d                      *\n",selectedFilterDegree);


        aux_R0 = sqrt(stopRipple/passRipple);
        aux_R2 = 2.0*cross_R0(aux_R0);

        tmp_aux_X4 = 4.0/(selectedFilterDegree*1.0);
        aux_X4 = 0.5*pow(aux_R2,tmp_aux_X4);

        aux_X0 = cross_X4(aux_X4);

        minStopFreq = ((samplingFreq*1.0)/PI)*atan(passTransFreq*pow(aux_X0,2.0));

        actualStopFreq = minStopFreq;
    //actualStopFreq = stopFreq;

        actualStopTransFreq = tan((PI*actualStopFreq)/(samplingFreq*1.0));

        aux_Q0 = sqrt(actualStopTransFreq/passTransFreq);
```

```
        aux_Q1 = pow(aux_Q0,2.0) + sqrt(pow(aux_Q0,4.0)-1.0);
        aux_Q2 = pow(aux_Q1,2.0) + sqrt(pow(aux_Q1,4.0)-1.0);
        aux_Q3 = pow(aux_Q2,2.0) + sqrt(pow(aux_Q2,4.0)-1.0);
        aux_Q4 = pow(aux_Q3,2.0) + sqrt(pow(aux_Q3,4.0)-1.0);

        aux_M3 = 0.5*pow(sqrt(2.0*aux_Q4),selectedFilterDegree*1.0);
        aux_M2 = sqrt(0.5*(aux_M3+pow(aux_M3,-1.0)));
        aux_M1 = sqrt(0.5*(aux_M2+pow(aux_M2,-1.0)));
        aux_M0 = sqrt(0.5*(aux_M1+pow(aux_M1,-1.0)));

        minPassRipple = stopRipple/pow(aux_M0,2.0);

        //actualPassRipple = minPassRipple;
        actualPassRipple = passRipple;

        actualStopRipple = actualPassRipple*pow(aux_M0,2.0);

        actualStopAtten = 10.0*log10(1.0+pow(actualStopRipple, 2.0));
        actualPassAtten = 10.0*log10(1.0+pow(actualPassRipple, 2.0));

        printf("* The actual attenuation in passband is  %f dB         *\n",actualPassAtten);
        printf("* The actual attenuation in stopband is %f dB         *\n",actualStopAtten);
        wholeStar();

        tmp_acutualPassRipple = pow(actualPassRipple,-1.0);

        aux_g3 = cross_g1(tmp_acutualPassRipple,aux_M1,aux_M2);

        tmp_selectedFilterDegree = pow(selectedFilterDegree*1.0,-1.0);

        aux_w5 = pow(aux_M3/aux_g3+sqrt(pow(aux_M3/aux_g3,2.0)+1.0),tmp_selectedFilterDegree);
        aux_w0 = cross_w5(aux_w5,aux_Q4,aux_Q3,aux_Q2,aux_Q1,aux_Q0);

        tmp_r0 = (1.0+aux_w0*aux_Q0*passTransFreq)/(1.0-aux_w0*aux_Q0*passTransFreq);

        r_0 = tmp_r0;

        printf("\n");

        printf("\t\tcoefficient_0  =    %f\n",r_0);

for (index=1;index <= (selectedFilterDegree-1)/2;index++)
    {
      aux_C4 = aux_Q4/sin(PI*index*1.0/(selectedFilterDegree*1.0));

      aux_C3 = pow(2.0*aux_Q3,-1.0)*(aux_C4+pow(aux_C4,-1.0));
      aux_C2 = pow(2.0*aux_Q2,-1.0)*(aux_C3+pow(aux_C3,-1.0));
      aux_C1 = pow(2.0*aux_Q1,-1.0)*(aux_C2+pow(aux_C2,-1.0));
      aux_C0 = pow(2.0*aux_Q0,-1.0)*(aux_C1+pow(aux_C1,-1.0));

      aux_y  = pow(aux_C0,-1.0);

      aux_Bi = (pow(aux_w0,2.0)+pow(aux_y,2.0))*pow(aux_Q0*passTransFreq,2.0)/(1.0+pow(aux_w0*aux_y,2.0));

      tmp_aux_Ai = (-2.0*aux_w0*aux_Q0*passTransFreq)/(1.0+pow(aux_w0*aux_y,2.0));
      aux_Ai = tmp_aux_Ai*sqrt(1.0-(pow(aux_Q0,2.0)+1.0/pow(aux_Q0,2.0)-pow(aux_y,2.0))*pow(aux_y,2.0));

      tmp_r1 = (aux_Ai-aux_Bi-1.0)/(aux_Ai+aux_Bi+1.0);

      r_1 = tmp_r1;

      if(2*index-1 < 10)
      {
        if (r_1 > -10000 && r_1 < 0)
        printf("\t\tcoefficient_%d  =   %f\n",2*index-1,r_1);
        else
        printf("\t\tcoefficient_%d  =  %f\n",2*index-1,r_1);
      }
      else
      {
        if (r_1 > -10000 && r_1 < 0)
        printf("\t\tcoefficient_%d =   %f\n",2*index-1,r_1);
        else
        printf("\t\tcoefficient_%d =  %f\n",2*index-1,r_1);
      }


      tmp_r2 = (1.0-aux_Bi)/(1.0+aux_Bi);
      r_2 = tmp_r2;

      if(2*index < 10)
      {
        if(r_2 > 10000)
        printf("\t\tcoefficient_%d  =   %f\n",2*index,r_2);
        else
        printf("\t\tcoefficient_%d  =    %f\n",2*index,r_2);
      }
      else
      {
        if(r_2 > 10000)
        printf("\t\tcoefficient_%d =   %f\n",2*index,r_2);
        else
```

```
            printf("\t\tcoefficient_%d =    %f\n",2*index,r_2);
          }
        }

                printf("\n");
                wholeStar();
                tailPrint();
                wholeStar();

        return 0;

}
```

2. cauer_func.c

```
//#include <stdio.h>
#include <math.h>
#include "cauer_var_func.h"

double cross_k0(double k0)

{

        double k1,k2,k3,k4;

        k1=pow(k0,2.0)+sqrt(pow(k0,4.0)-1.0);
        k2=pow(k1,2.0)+sqrt(pow(k1,4.0)-1.0);
        k3=pow(k2,2.0)+sqrt(pow(k2,4.0)-1.0);
        k4=pow(k3,2.0)+sqrt(pow(k3,4.0)-1.0);

        return (k4);

}

double cross_R0(double R0)

{

         double R1,R2;

        R1=pow(R0,2.0)+sqrt(pow(R0,4.0)-1.0);

        R2=pow(R1,2.0)+sqrt(pow(R1,4.0)-1.0);

        return (R2);

}

double cross_X4(double X4)

{

         double X3,X2,X1,X0;

         X3=sqrt(0.5*(pow(X4,1.0)+pow(X4,-1.0)));
         X2=sqrt(0.5*(pow(X3,1.0)+pow(X3,-1.0)));
         X1=sqrt(0.5*(pow(X2,1.0)+pow(X2,-1.0)));
         X0=sqrt(0.5*(pow(X1,1.0)+pow(X1,-1.0)));

         return (X0);

}

double cross_g1(double e_passStar,double M1,double M2)

{

         double g1,g2,g3;

         g1=e_passStar*1.0+sqrt(pow(e_passStar,2.0)+1.0);

         g2=g1*M1+sqrt(pow(g1*M1,2.0)+1.0);

         g3=g2*M2+sqrt(pow(g2*M2,2.0)+1.0);

         return (g3);

}

double cross_w5(double w5,double Q4,double Q3,double Q2,double Q1,double Q0)

{

         double w4,w3,w2,w1,w0;

         w4=pow(2.0*Q4,-1.0)*(w5-pow(w5,-1.0));
         w3=pow(2.0*Q3,-1.0)*(w4-pow(w4,-1.0));
         w2=pow(2.0*Q2,-1.0)*(w3-pow(w3,-1.0));
         w1=pow(2.0*Q1,-1.0)*(w2-pow(w2,-1.0));
```

```
        w0=pow(2.0*Q0,-1.0)*(w1-pow(w1,-1.0));

        return (w0);

}
```

3. cauer_var_func.h

```
#define PI 3.141592654
#define c1 8.0
#define c2 4.0

float       passFreq, passAtten, stopFreq, stopAtten, samplingFreq;

double      tmp_stopAtten, tmp_passAtten;

double      stopRipple, passRipple, stopTransFreq, passTransFreq;

double      aux_k0, aux_k4, aux_c3, minFilterDegree;
int         selectedFilterDegree;


double      aux_R0, aux_R2, tmp_aux_X4, aux_X4, aux_X0;

double      minStopFreq, actualStopFreq, actualStopTransFreq;

double      aux_Q0, aux_Q1, aux_Q2, aux_Q3, aux_Q4, aux_M3, aux_M2, aux_M1, aux_M0;

double      minPassRipple, actualPassRipple, actualStopRipple;
double      actualStopAtten, actualPassAtten;

double      tmp_acutualPassRipple, aux_g3;
double      tmp_selectedFilterDegree, aux_w5, aux_w0, tmp_r0;


double      r_0, r_1, r_2;
int         index;


double      aux_C4, aux_C3, aux_C2, aux_C1, aux_C0, aux_y;

double      aux_Bi, tmp_aux_Ai, aux_Ai, tmp_r1, tmp_r2;

void        wholeStar(void);

void            introduction(void);

double      cross_k0(double);

double      cross_X4(double);

double      cross_g1(double, double, double);

double      cross_w5(double, double, double, double, double, double);

void        tailPrint(void);
```

4. public_func.c

```
#include <stdio.h>

void wholeStar(void)
{

  printf("*****************************************************************\n");

}

void emptyStar(void)
{

  printf("*                                                               *\n");

}

void introduction(void)
{

        printf("* Please input the value:fp,ap,fs,as,F, respectively         *\n");
        printf("*                                                           *\n");
        printf("* Where:                                                    *\n");
        printf("*       fp = upper edge frequency of the passband,          *\n");
        printf("*       ap = maximum allowable attenuation in the passband; *\n");
        printf("*       fs = lower edge frequency of the stopband;          *\n");
        printf("*       as = specified minimum attenuation in the stopband; *\n");
        printf("*        F = sampling frequency.                            *\n");
```

```
        printf("*                                                              *\n");

//the following code is for debugging purpose only
//printf("* cauer:       3500,0.2,4100,60,16000                          *\n");
//printf("* butterworth: 3400,0.5,6000,55,16000                          *\n");
//printf("* chebyshev:   3000,1.0,5000,40,16000                          *\n");

}

void tailPrint(void)
{

        printf("*                                                              *\n");
        printf("*        All coefficients are quantized to 24 bits.            *\n");
        printf("*                                                              *\n");

}
```

5. dsp_io_dataCollected.cmd

```
>> dsp_inputSignal_rawData.dat
*(_X:0x6000)/2048r
>> c
>> dsp_outputSignal_rawData.dat
*(_X:0x8000)/2048r
>> c
```

6. matlab_sim_input.m

```
%The piece of code is to plot out input signal in Matlab simulation basis

clear;

format long;

load matlab_sim_inputSignal.dat;

matlab_input = matlab_sim_inputSignal;

FFTsize = 2048;
Half_FFTsize = FFTsize/2;
time_interval = linspace(0,FFTsize-1,FFTsize);

index = (0:1023)';
freqAxis = (index/128)*1000;

%-------------------------------------------------------------------------------------------------------------------

%loading input signal for print purpose

disp('Please choose one of options: print_sim_input = 1, show_figure = 2');
which_loop = input('    my option = ');

if which_loop == 1

   diary print_matlab_sim_input.dat;
   inputSignal = matlab_sim_inputSignal';
   matlab_sim_input_print = reshape(inputSignal',512,4)
   diary off;

elseif which_loop == 2

   disp('Only show input signal in time & frequency domain graphically');

else

   disp('Your option is out of the range!');

end

%-------------------------------------------------------------------------------------------------------------------

figure(1)

subplot(3,1,1);plot(time_interval,matlab_input);axis([0 512 -.25 .25]);grid on;
title('Time Domain: Input Signal for Matlab Simulation & DSP56307EVM Implementation');
ylabel('Amplitude in Time domain');

tmp_1 = fft(matlab_input); tmp = abs(fftshift(tmp_1));

subplot(3,1,2);plot(freqAxis,tmp(1025:2048)/Half_FFTsize,'r');grid on;
title('Overall Spectrum: Input Signal for Matlab Simulation & DSP56307EVM Implementation');
ylabel('Amplitude');

subplot(3,1,3);plot(freqAxis,tmp(1025:2048)/Half_FFTsize,'m');axis([0 8000 0 .006]);grid on;
title('Noise Level Spectrum: Input Signal for Matlab Simulation & DSP56307EVM Implementation');
xlabel('Frequency Domain: Hz');ylabel('Amplitude');
```

7. testingSignal_gear.m

```
%This portion is to generate signals as input and use the data through Matlab Simulation & DSP56307EVM
%implementation, which will be as a testing vector to pass different type of the filters designed

clear;

format long;

FFTsize = 2048;
Half_FFTsize = FFTsize/2;
samplingFreq = 16000;
time_interval = linspace(0,FFTsize-1,FFTsize);

fundementalFreq = 500;

Wt_fundemental = 2*pi*fundementalFreq*time_interval/samplingFreq;
wt = Wt_fundemental;

passSignal = 0.070*sin(wt) + 0.045*sin(2.2*wt);

stopSignal = 0.041*cos(12.5*wt) + 0.062*cos(14.2*wt);

noiseSignal = 0.024*randn(size(stopSignal+sin(0.755237*12.5*wt)));

inputSignal = passSignal + stopSignal + noiseSignal;

%load index.dat
index = (0:1023)';
freqAxis = (index/128)*1000;

%-------------------------------------------------------------------------------------------------------------------

%loading signal components for input, i.e. min_max SNR_dB noiseSignal & inputSignal

disp('Please choose one of options: track_MinMaxSNR = 1, print_all_Data = 2');
which_loop = input('     my option = ');

if which_loop == 1

   min_Input = min(inputSignal)
        max_Input = max(inputSignal)

        real_input_power = sum(passSignal.*passSignal + stopSignal.*stopSignal)
        noise_input_power = sum(noiseSignal.*noiseSignal)

        SNR_dB = 10*log10(real_input_power/noise_input_power)

elseif which_loop == 2

        diary matlab_sim_inputSignal.dat;
        matlab_sim_inputSignal = inputSignal'
        diary off;

%-------------------------------------------------------------------------------------------------------------------

   diary print_min_max_power_SNR.dat;

   disp('The following will show minimum & maximum amplitude of input signal');
        min_Input = min(matlab_sim_inputSignal)
        max_Input = max(matlab_sim_inputSignal)

   disp('The following will show the powers of input signal & noise signal');
        real_input_power = sum(passSignal.*passSignal + stopSignal.*stopSignal)
        noise_input_power = sum(noiseSignal.*noiseSignal)

   disp('The following will show the signal-noise-ratio of the signal');
        SNR_dB = 10*log10(real_input_power/noise_input_power)
        diary off;


else

   disp('Your option is out of the range!');

end

%-------------------------------------------------------------------------------------------------------------------

figure(1)

subplot(3,1,1);plot(time_interval,inputSignal);axis([0 512 -.25 .25]);grid on;
title('Time Domain: Input Signal for Matlab Simulation & DSP56307EVM Implementation');
xlabel('Time Domain');ylabel('Amplitude');

tmp_1 = fft(inputSignal); tmp = abs(fftshift(tmp_1));

subplot(3,1,2);plot(freqAxis,tmp(1025:2048)/Half_FFTsize,'r');grid on;
title('Spectrum: Input Signal for Matlab Simulation & DSP56307EVM Implementation');
ylabel('Amplitude');
```

```
FFT_inputSignal = fft(inputSignal); plotFFT_inputSignal = abs(fftshift(FFT_inputSignal));

subplot(3,1,3);plot(freqAxis,plotFFT_inputSignal(1025:2048)/Half_FFTsize,'m');axis([0 8000 0 .006]);grid on;
title('Input Signal at Noise Level Spectrum');
xlabel('Frequency Domain: Hz');ylabel('Amplitude');


8. comp_testVector.m


clear;

format long;

FFTsize = 2048;
Half_FFTsize = FFTsize/2;
time_interval = linspace(0,FFTsize-1,FFTsize);

%load index.dat
index = (0:1023)';
freqAxis = (index/128)*1000;

%-------------------------------------------------------------------------------------------------------------

load matlab_sim_inputSignal_org.dat;
load matlab_sim_inputSignal_real.dat;

inputSignal_org = matlab_sim_inputSignal_org;
inputSignal_rea = matlab_sim_inputSignal_real;

%-------------------------------------------------------------------------------------------------------------
figure(1);

subplot(3,1,1);plot(time_interval,inputSignal_org);axis([0 512 -.25 .25]);grid on;
title('Input Signal implemented in Matlab Platform');
ylabel('Amplitude');

subplot(3,1,2);plot(time_interval,inputSignal_rea,'r');axis([0 512 -.25 .25]);grid on;
title('Input Signal implemented in Tasking EDE Platform');
ylabel('Amplitude');

subplot(3,1,3);plot(time_interval,inputSignal_org-inputSignal_rea);axis([0 512 -1.5e-7 1.5e-7]);grid on;
title('Input Signal: Amplitude difference with Matlab and Tasking EDE platforms');
xlabel('Signal in Time Domain');ylabel('Amplitude');


Directory: F:\WD_Filter_Design\dsp_impl_cauer9
==============================================

File Name:
==========

1. cauer_9th_const_templ.h


#define  FFTSIZE                          2*1024

#define  ADDRESS_CAUER_LOCAL_VAR          0x1000
#define  ADDRESS_SIMULATION_DATA          0x6000
#define  ADDRESS_CAUER_OUTPUT             0x8000


#pragma asm

CAUER_SAMPLE_SIZE_ASM          equ          2048

ADDRESS_CAUER_LOCAL_VAR_ASM    equ          $1000
ADDRESS_CAUER_OUTPUT_ASM       equ          $6000
ADDRESS_SIMULATION_DATA_ASM    equ          $8000

#pragma endasm


#define  r0   0.603586
#define  r1  -0.483105
#define  r2   0.689629
#define  r3  -0.697908
#define  r4   0.409262
#define  r5  -0.859177
#define  r6   0.251224
#define  r7  -0.959691
#define  r8   0.189050


_fract adap_1(_fract  x1,  _fract  x2, const _fract r);
_fract adap_2(_fract  x1,  _fract  x2, const _fract r);


2. cauer_9th_main.c
```

```
/* 9th Cauer(Elliptic) Lattice Lowpass Filter with Specifications*/
/* fp=3500, ap=0.05, fs=4100, as=60, F=16000 */
/* The Following Program is the Implementation for Impulse response*/
#include "cauer_9th_const_templ.h"
//#include "twoPort_Adap.c"

#pragma asm
  include 'cauer_9th_rawData.asm'
#pragma endasm

_internal _fract _X *pointerSimData;
_internal _fract _X output[FFTSIZE]     _at(ADDRESS_CAUER_OUTPUT);

_internal  _fract _X   a[18];
_internal  _fract _X   b[18];

//extern _external  int   _X   i;
_internal _X int i;

void main(void)
{

pointerSimData = (_fract *)ADDRESS_SIMULATION_DATA;


for (i=0; i <= FFTSIZE-1; i++)
    {
      //a[0]=input[i];

      a[0]=*pointerSimData++;

      b[0]=adap_1(a[0],a[1],r0);
      b[1]=adap_2(a[0],a[1],r0);
      a[1]=b[1];
      a[6]=b[0];

      b[8]=adap_1(a[8],a[9],r4);
      b[9]=adap_2(a[8],a[9],r4);
      a[9]=b[9];
      a[7]=b[8];

      b[6]=adap_1(a[6],a[7],r3);
      b[7]=adap_2(a[6],a[7],r3);
      a[14]=b[6];

      b[16]=adap_1(a[16],a[17],r8);
      b[17]=adap_2(a[16],a[17],r8);
      a[17]=b[17];
      a[15]=b[16];

      b[14]=adap_1(a[14],a[15],r7);
      b[15]=adap_2(a[14],a[15],r7);

      /* Input for Lower Section */
      a[2]=a[0];

      b[4]=adap_1(a[4],a[5],r2);
      b[5]=adap_2(a[4],a[5],r2);
      a[5]=b[5];
      a[3]=b[4];

      b[2]=adap_1(a[2],a[3],r1);
      b[3]=adap_2(a[2],a[3],r1);
      a[10]=b[2];

      b[12]=adap_1(a[12],a[13],r6);
      b[13]=adap_2(a[12],a[13],r6);
      a[13]=b[13];
      a[11]=b[12];

      b[10]=adap_1(a[10],a[11],r5);
      b[11]=adap_2(a[10],a[11],r5);

      /* Data as impulse response */
      output[i]=0.5*(b[14]+b[10]);

      a[4]=b[3];
      a[12]=b[11];
      a[8]=b[7];
      a[16]=b[15];

      /* Signed data at the same columns */
      //if (output[i]<0)
      //printf("%d    %0.18f\n", i, output[i]);
      //else
      //printf("%d     %0.18f\n", i, output[i]);
      }

}
```

3. twoPort_Adap.c

```
/* Function's Definition for 1st Element of Two_Port Adaptor */

//float adap_1(float x1, float x2, float r)

_fract adap_1( _fract  x1,  _fract  x2,  const _fract r)

{
   //float t;
   //float y1;

   _fract t;
   _fract y1;

   t=x2-x1;
  y1=x2+r*t;

  return (y1);

}


/* Function's Definition for 2nd Element of Tow_Port Adaptor */

//float adap_2(float x1, float x2, float r)

_fract adap_2( _fract  x1,  _fract  x2,  const _fract r)

{

  // float t;
  // float y2;

   _fract t;
   _fract y2;

   t=x2-x1;
  y2=x1+r*t;

  return (y2);

}
```

Directory: F:\WD_Filter_Design\matlab_sim_cauer9

File Name:

1. dsp_impl_intput.m

```
%The piece of code is to plot out input signal in DSP56307EVM implementation basis

clear;

format long;

load dsp_impl_inputSignal.dat;

dsp_input = dsp_impl_inputSignal;

FFTsize = 2048;
Half_FFTsize = FFTsize/2;
time_interval = linspace(0,FFTsize-1,FFTsize);

index = (0:1023)';
freqAxis = (index/128)*1000;

%-------------------------------------------------------------------------------------------------------------

%loading input signal for print purpose

disp('Please choose one of options: print_sim_input = 1, show_figure = 2');
which_loop = input('    my option = ');

if which_loop == 1

   inputSignal = dsp_impl_inputSignal;

   iTitle_1 = 'print_sim_impl_input_cauer9.txt';
   fid_1 = fopen(iTitle_1,'wr');
   fprintf(fid_1, '\t The following is input data of Matlab Simulation and DSP Implementation\n\n');
   fprintf(fid_1, '\t\t\t\t        with a 9th order Cauer WD Filter\n\n\n');

   for i = 1:512

      if inputSignal(i) < 0
```

```
        fprintf(fid_1,'     %10.14f %20.14f %20.14f %20.14f\n', ...
           inputSignal(i),inputSignal(i+1),inputSignal(i+2),inputSignal(i+3));
     else
        fprintf(fid_1,'      %10.14f %20.14f %20.14f %20.14f\n', ...
           inputSignal(i),inputSignal(i+1),inputSignal(i+2),inputSignal(i+3));
     end

  end

  fclose(fid_1);

elseif which_loop == 2

  disp('Only show input signal in time & frequency domain graphically');

else

  disp('Your option is out of the range!');

end

%-------------------------------------------------------------------------------------------------------------

figure(1)

subplot(3,1,1);plot(time_interval,dsp_input);axis([0 512 -.25 .25]);grid on;
title('Time Domain: Input Signal for 9th Cauer (Elliptic) DSP56307EVM implementation');
ylabel('Amplitude in Time domain');

tmp_1 = fft(dsp_input); tmp = abs(fftshift(tmp_1));

subplot(3,1,2);plot(freqAxis,tmp(1025:2048)/Half_FFTsize,'r');grid on;
title('Overall Spectrum: Input Signal for 9th Cauer (Elliptic) DSP56307EVM implementation');
ylabel('Amplitude');

subplot(3,1,3);plot(freqAxis,tmp(1025:2048)/Half_FFTsize,'m');axis([0 8000 0 .006]);grid on;
title('Noise Level Spectrum: Input Signal for 9th Cauer (Elliptic) DSP56307EVM implementation');
xlabel('Frequency Domain: Hz');ylabel('Amplitude');
```

2. dsp_impl_output.m

```
%The piece of code is to plot out output signal in DSP56307EVM implementation basis

clear;

format long;

load dsp_impl_inputSignal.dat;
load dsp_impl_outputSignal.dat;

dsp_input = dsp_impl_inputSignal;

FFTsize = 2048;
Half_FFTsize = FFTsize/2;
time_interval = linspace(0,FFTsize-1,FFTsize);

index = (0:1023)';
freqAxis = (index/128)*1000;

%-------------------------------------------------------------------------------------------------------------

%loading output signal for print purpose

disp('Please choose one of options: print_sim_output = 1, show_figure = 2');
which_loop = input('    my option = ');

if which_loop == 1

  outputSignal = dsp_impl_outputSignal;

  iTitle_1 = 'print_dsp_impl_output_cauer9.txt';
  fid_1 = fopen(iTitle_1,'wr');
  fprintf(fid_1, '\t\t     The following is output data of DSP Implementation\n\n');
  fprintf(fid_1, '\t\t\t        with a 9th order Cauer WD Filter\n\n\n');

  for i = 1:512

     if outputSignal(i) < 0
        fprintf(fid_1,'     %10.14f %20.14f %20.14f %20.14f\n', ...
          outputSignal(i),outputSignal(i+1),outputSignal(i+2),outputSignal(i+3));
     else
        fprintf(fid_1,'      %10.14f %20.14f %20.14f %20.14f\n', ...
          outputSignal(i),outputSignal(i+1),outputSignal(i+2),outputSignal(i+3));
     end

  end

  fclose(fid_1);

elseif which_loop == 2
```

```
   disp('Only show output signal in time & frequency domain graphically');

else

   disp('Your option is out of the range!');

end

%-------------------------------------------------------------------------------------------------------------

%dsp_output = wdfilter(FFTsize,dsp_input);

dsp_output = dsp_impl_outputSignal;

figure(1)

subplot(3,1,1);plot(time_interval,dsp_output);axis([0 512 -.15 .15]);grid on;
title('Time Domain: Output Signal for 9th Cauer (Elliptic) DSP56307EVM implementation');
ylabel('Amplitude in Time domain');

tmp_1 = fft(dsp_output); tmp = abs(fftshift(tmp_1));

subplot(3,1,2);plot(freqAxis,tmp(1025:2048)/Half_FFTsize,'r');grid on;
title('Overall Spectrum: Output Signal for 9th Cauer (Elliptic) DSP56307EVM implementation');
ylabel('Amplitude');

subplot(3,1,3);plot(freqAxis,tmp(1025:2048)/Half_FFTsize,'m');axis([0 8000 0 .006]);grid on;
title('Noise Level Spectrum: Output Signal for 9th Cauer (Elliptic) DSP56307EVM implementation');
xlabel('Frequency Domain: Hz');ylabel('Amplitude');


3. dsp_in_out_load.m


%The program is to load the data of input and output signal with DSP56307EVM implementation

clear;

format long;

FFTsize = 2048;
Half_FFTsize = FFTsize/2;
time_interval = linspace(0,FFTsize-1,FFTsize);

load dsp_inputSignal_512x4.dat

load dsp_outputSignal_512x4.dat

tmp1 = dsp_inputSignal_512x4';  dsp_inputSignal = tmp1(:);

tmp2 = dsp_outputSignal_512x4'; dsp_outputSignal = tmp2(:);

%load index.dat
index = (0:1023)';
freqAxis = (index/128)*1000;

tmp_1 = fft(dsp_inputSignal);  tmp_11 = abs(fftshift(tmp_1));
tmp_2 = fft(dsp_outputSignal); tmp_22 = abs(fftshift(tmp_2));

figure(1)

subplot(2,1,1);plot(time_interval,dsp_inputSignal,'r');axis([0 512 -.25 .25]);grid on;
title('Input Signal of 9th Cauer (Elliptic) DSP56307EVM Implementation in Time Domain');
ylabel('Amplitude');

subplot(2,1,2);plot(time_interval,dsp_outputSignal);axis([0 512 -.25 .25]);grid on;
title('Output Signal of 9th Cauer (Elliptic) DSP56307EVM Implementation in Time Domain');
xlabel('Signal in Time Domain');ylabel('Amplitude');

figure(2)

subplot(2,1,1);plot(freqAxis,tmp_11(1025:2048)/Half_FFTsize,'r');grid on;
title('Input Signal of 9th Cauer (Elliptic) DSP56307EVM Implementation in Frequency Domain');
ylabel('Amplitude');

subplot(2,1,2);plot(freqAxis,tmp_22(1025:2048)/Half_FFTsize);grid on;
title('Output Signal of 9th Cauer (Elliptic) DSP56307EVM Implementation in Frequency Domain');
xlabel('Signal in Frequency Domain: Hz');ylabel('Amplitude');

figure(3)

subplot(2,1,1);plot(freqAxis,tmp_11(1025:2048)/Half_FFTsize,'r');axis([0 8000 0 .006]);grid on;
title('Noise Level Spectrum: Input Signal of 9th Cauer (Elliptic) DSP56307EVM Implementation');
ylabel('Amplitude');

subplot(2,1,2);plot(freqAxis,tmp_22(1025:2048)/Half_FFTsize);axis([0 8000 0 .006]);grid on;
title('Noise Level Spectrum: Output Signal of 9th Cauer (Elliptic) DSP56307EVM Implementation');
xlabel('Signal in Frequency Domain: Hz');ylabel('Amplitude');

%-------------------------------------------------------------------------------------------------------------
```

```
disp('Please choose one of options: not_load_dsp_io_data = 1, load_dsp_io_data = 2');
which_loop = input('    my option = ');

if which_loop == 1

   disp('Have a nice day!');

elseif which_loop == 2

   diary dsp_impl_inputSignal.dat;
   dsp_inputSignal
   diary off;

   diary dsp_impl_outputSignal.dat;
   dsp_outputSignal
   diary off;

   disp('       <dsp_impl_inputSignal.dat>  is input  data from EVM implementation.');
   disp('       <dsp_impl_outputSignal.dat> is output data from EVM implementation.');
else

   disp('Your option is out of range!');

end
```

4. impulse_resp.m

```
%The charactorizations of the 9th Cauer (Elliptic) lowpass lattice filter

clear;
format long;

FFTsize = 2048;
time_interval = linspace(0,FFTsize-1,FFTsize);

impulseInput(1) = 1;impulseInput(2:FFTsize) = zeros(FFTsize-1,1);
impulseOutput = wdfilter(FFTsize,impulseInput);

impulseOutputFFT = fft(impulseOutput,FFTsize);

%load index.dat
index = [0:1023]';
freqAxis = (index/128)*1000;

%-------------------------------------------------------------------------------------------------------------------------
-

figure(1)

subplot(2,1,1);stem(time_interval,impulseOutput,'r');axis([0,32,-0.2,0.4]);grid on;
title('Impulse Response of the 9th Cauer (Elliptic) Lowpass Filter');

subplot(2,1,2);plot(fftshift(abs(fft((impulseOutput)))));axis([500 1550 0.96 1]);grid on;
title('Impulse Frequency Spectrum of the 9th Cauer (Elliptic) Lowpass Filter');

impulseMagnitute   = abs(impulseOutputFFT(1:1:FFTsize/2));
impulseAttenuation = -20*log10(impulseMagnitute);

%-------------------------------------------------------------------------------------------------------------------------
-

figure(2)

subplot(2,1,1);plot(freqAxis,impulseMagnitute);axis([0,8000,0,1]);grid on;
text(4500,0.75,'Sampling Frequency  =  16  kHz');
text(4500,0.55,'Passband Frequency =  3.5 kHz');
text(4500,0.35,'Stopband Frequency  =  4.0 kHz');
title('Magnitute Response of the 9th Cauer (Elliptic) Lowpass Filter');
xlabel('Frequency in Hz');ylabel('Magnitute in dB');

subplot(2,1,2);plot(freqAxis,impulseAttenuation);axis([0,8000,0,200]);grid on;
text(500,90,'Sampling Frequency  =  16  kHz');
text(500,70,'Passband Frequency =  3.5 kHz');
text(500,50,'Stopband Frequency  =  4.0 kHz');text(4100,50,'Minimum attenuation in the stopband is 80.0 dB');
title('Attenuation Response of the 9th Cauer (Elliptic) Lowpass Filter');
xlabel('Frequency in Hz');ylabel('Attenuation in dB');

%-------------------------------------------------------------------------------------------------------------------------
-

figure(3)

subplot(3,1,1);plot(freqAxis,impulseMagnitute);axis([0 4000 0.96 1]);grid on;
title('Passband Magnitute Response of the 9th Cauer (Elliptic) Lowpass Filter');
ylabel('Magnitute in dB');

subplot(3,1,2);plot(freqAxis,impulseAttenuation);axis([0 4000 0 0.4]);grid on;
text(550,.35,'Maximum attenuation in the passband is 0.3 dB');
text(2200,.35,'Passband Frequency =  3500 Hz');
title('Passband Attenuation Response of the 9th Cauer (Elliptic) Lowpass Filter');
```

```
ylabel('Attenuation in dB');

subplot(3,1,3);plot(freqAxis,impulseAttenuation);axis([3500 8000 60 200]);grid on;
text(5200,170,'Minimum attenuation in the stopband is 80.0 dB');
text(5200,150,'Stopband Frequency  =  4000 Hz');
title('Stopband Attenuation Response of the 9th Cauer (Elliptic) Lowpass Filter');
xlabel('Frequency in Hz');ylabel('Attenuation in dB');
```

5. matlab_sim_output.m

```
%The piece of code is to plot out output signal in Matlab simulation basis

clear;

format long;

load matlab_sim_inputSignal.dat;

matlab_input = matlab_sim_inputSignal;

FFTsize = 2048;
Half_FFTsize = FFTsize/2;
time_interval = linspace(0,FFTsize-1,FFTsize);

index = (0:1023)';
freqAxis = (index/128)*1000;

matlab_output = wdfilter(FFTsize,matlab_input);

%-------------------------------------------------------------------------------------------------------------------

%loading output signal for print purpose

disp('Please choose one of options: print_sim_output = 1, show_figure = 2');
which_loop = input('    my option = ');

if which_loop == 1

   outputSignal = matlab_output';

   iTitle_1 = 'print_matlab_sim_output_cauer9.txt';
   fid_1 = fopen(iTitle_1,'wr');
   fprintf(fid_1, '\t\t     The following is output data of Matlab Simulation\n\n');
   fprintf(fid_1, '\t\t\t        with a 9th order Cauer WD Filter\n\n\n');

   for i = 1:512

      if outputSignal(i) < 0
         fprintf(fid_1,'     %10.14f %20.14f %20.14f %20.14f\n', ...
            outputSignal(i),outputSignal(i+1),outputSignal(i+2),outputSignal(i+3));
      else
         fprintf(fid_1,'      %10.14f %20.14f %20.14f %20.14f\n', ...
            outputSignal(i),outputSignal(i+1),outputSignal(i+2),outputSignal(i+3));
      end

   end

   fclose(fid_1);

elseif which_loop == 2

   disp('Only show output signal in time & frequency domain graphically');

else

   disp('Your option is out of the range!');

end

%-------------------------------------------------------------------------------------------------------------------

figure(1)

subplot(3,1,1);plot(time_interval,matlab_output);axis([0 512 -.15 .15]);grid on;
title('Time Domain: Output Signal for 9th Cauer (Elliptic) Matlab Simulation');
ylabel('Amplitude in Time domain');

tmp_1 = fft(matlab_output); tmp = abs(fftshift(tmp_1));

subplot(3,1,2);plot(freqAxis,tmp(1025:2048)/Half_FFTsize,'r');grid on;
title('Overall Spectrum: Output Signal for 9th Cauer (Elliptic) Matlab Simulation');
ylabel('Amplitude');

subplot(3,1,3);plot(freqAxis,tmp(1025:2048)/Half_FFTsize,'m');axis([0 8000 0 .006]);grid on;
title('Noise Level Spectrum: Output Signal for 9th Cauer (Elliptic) Matlab Simulation');
xlabel('Frequency Domain: Hz');ylabel('Amplitude');
```

6. sim_imp_comp.m

```
%This program is to compare with the simulation & DSP implementation in multi-ways

clear;

format long;

FFTsize = 2048;
Half_FFTsize = FFTsize/2;
time_interval = linspace(0,FFTsize-1,FFTsize);

%load index.dat
index = (0:1023)';
freqAxis = (index/128)*1000;

%-------------------------------------------------------------------------------------------------------------------
%This zone is to load simulation & implementation data

load matlab_sim_inputSignal.dat;
load dsp_impl_inputSignal.dat;
load dsp_impl_outputSignal.dat;

matlab_sim_input = matlab_sim_inputSignal;

%-------------------------------------------------------------------------------------------------------------------

matlab_outputSignal = wdfilter(FFTsize,dsp_impl_inputSignal);

matlab_FFT_outputSignal = fft(matlab_outputSignal);

matlab_plotFFT_outputSignal = abs(fftshift(matlab_FFT_outputSignal));

%-------------------------------------------------------------------------------------------------------------------

dsp_FFT_outputSignal = fft(dsp_impl_outputSignal);

dsp_plotFFT_outputSignal = abs(fftshift(dsp_FFT_outputSignal));

diff_outputSignal = matlab_outputSignal' - dsp_impl_outputSignal;

%-------------------------------------------------------------------------------------------------------------------
%The section is to compare with the real part and imaginary part between Matlab & DSP56307EVM, respectively

matlab_FFT_outputSignal_real = real(fft(matlab_outputSignal));
matlab_FFT_outputSignal_imag = imag(fft(matlab_outputSignal));

dsp_FFT_outputSignal_real = real(fft(dsp_impl_outputSignal));
dsp_FFT_outputSignal_imag = imag(fft(dsp_impl_outputSignal));

diff_realPart_FFT = matlab_FFT_outputSignal_real' - dsp_FFT_outputSignal_real;
diff_imagPart_FFT = matlab_FFT_outputSignal_imag' - dsp_FFT_outputSignal_imag;

%-------------------------------------------------------------------------------------------------------------------
%The section is to compare with the powers in time domain & frequnecy domain, using Parseval's Relation

   matlab_sim_timePower = sum(matlab_sim_input.^2);
   FFT_matlab_sim_real = real(fft(matlab_sim_input)); FFT_matlab_sim_imag = imag(fft(matlab_sim_input));
   matlab_sim_freqPower = sum(FFT_matlab_sim_real.^2 + FFT_matlab_sim_imag.^2)/FFTsize;

   dsp_impl_timePower = sum(dsp_impl_inputSignal.^2);
   FFT_dsp_impl_real = real(fft(dsp_impl_inputSignal)); FFT_dsp_impl_imag = imag(fft(dsp_impl_inputSignal));
   dsp_impl_freqPower = sum(FFT_dsp_impl_real.^2 + FFT_dsp_impl_imag.^2)/FFTsize;

   matlab_outputSignal_timePower = sum(matlab_outputSignal.^2);
   matlab_outputSignal_freqPower = sum(matlab_FFT_outputSignal_real.^2+matlab_FFT_outputSignal_imag.^2)/FFTsize;

   dsp_impl_outputSignal_timePower = sum(dsp_impl_outputSignal.^2);
   dsp_impl_outputSignal_freqPower = sum(dsp_FFT_outputSignal_real.^2 + dsp_FFT_outputSignal_imag.^2)/FFTsize;

   diff_outputSignal_timePower = abs(matlab_outputSignal_timePower - dsp_impl_outputSignal_timePower);
   diff_outputSignal_freqPower = abs(matlab_outputSignal_freqPower - dsp_impl_outputSignal_freqPower);

%-------------------------------------------------------------------------------------------------------------------
figure(1);

subplot(3,1,1);plot(time_interval,matlab_sim_input);axis([0 512 -.25 .25]);grid on;
title('Actual Input Signal in Time Domain with 9th Cauer (Elliptic) Matlab Simulation');
ylabel('Amplitude');

subplot(3,1,2);plot(time_interval,dsp_impl_inputSignal,'r');axis([0 512 -.25 .25]);grid on;
title('Actual Input Signal in Time Domain with 9th Cauer (Elliptic) DSP56307EVM Implementation');
ylabel('Amplitude');

subplot(3,1,3);plot(time_interval,matlab_sim_input-dsp_impl_inputSignal);axis([0 512 -.3e-6 .3e-6]);grid on;
title('Input: Amplitude difference in Time Domain Between Simulation and Implementation');
xlabel('Signal in Time Domain');ylabel('Amplitude');

%-------------------------------------------------------------------------------------------------------------------

figure(2);

subplot(3,1,1);plot(time_interval,matlab_outputSignal);axis([0 512 -.15 .15]);grid on;
```

```
title('Actual Output Signal in Time Domain with 9th Cauer (Elliptic) Matlab Simulation');
ylabel('Amplitude');

subplot(3,1,2);plot(time_interval,dsp_impl_outputSignal,'r');axis([0 512 -.15 .15]);grid on;
title('Actual Output Signal in Time Domain with 9th Cauer (Elliptic) DSP56307EVM Implementation');
ylabel('Amplitude');

subplot(3,1,3);plot(time_interval,matlab_outputSignal'-dsp_impl_outputSignal);axis([0 512 -.3e-6 .3e-6]);grid on;
title('Output: Amplitude difference in Time Domain Between Simulation and Implementation');
xlabel('Signal in Time Domain');ylabel('Amplitude');

%-------------------------------------------------------------------------------------------------------------

figure(3);

subplot(2,1,1);plot(freqAxis,matlab_plotFFT_outputSignal(1025:2048)/Half_FFTsize,'r');
grid on;
title('The Spectrum: Actual Output Signal with 9th Cauer (Elliptic) Matlab Simulation');
xlabel('Frequency in Hz');ylabel('Amplitude');

subplot(2,1,2);plot(freqAxis,dsp_plotFFT_outputSignal(1025:2048)/Half_FFTsize);
grid on;
title('The Spectrum: Actual Output Signal with 9th Cauer (Elliptic) DSP56307EVM Implementation');
xlabel('Frequency in Hz');ylabel('Amplitude');

%-------------------------------------------------------------------------------------------------------------

figure(4);

subplot(2,1,1);plot(freqAxis,matlab_plotFFT_outputSignal(1025:2048)/Half_FFTsize,'r');
axis([0 8000 0 .004]);grid on;
title('Noise Level Spectrum: Actual Output Signal with 9th Cauer (Elliptic) Matlab Simulation');
xlabel('Frequency in Hz');ylabel('Amplitude');

subplot(2,1,2);plot(freqAxis,dsp_plotFFT_outputSignal(1025:2048)/Half_FFTsize);
axis([0 8000 0 .004]);grid on;
title('Noise Level Spectrum: Actual Output Signal with 9th Cauer (Elliptic) DSP56307EVM Implementation');
xlabel('Frequency in Hz');ylabel('Amplitude');

%-------------------------------------------------------------------------------------------------------------

figure(5);

subplot(2,1,1);plot(freqAxis,diff_realPart_FFT(1025:2048)/Half_FFTsize,'r');grid on;
title('The Output Spectrum Difference in real part between 9th Cauer (Elliptic) Simulation and Implementation');
xlabel('Frequency in Hz');ylabel('Amplitude');

subplot(2,1,2);plot(freqAxis,diff_imagPart_FFT(1025:2048)/Half_FFTsize);grid on;
title('The Output Spectrum Difference in imag part between 9th Cauer (Elliptic) Simulation and Implementation');
xlabel('Frequency in Hz');ylabel('Amplitude');

%-------------------------------------------------------------------------------------------------------------

figure(6);

subplot(2,2,1);plot(freqAxis(1:449),diff_realPart_FFT(1025:1473)/Half_FFTsize);
title('9th Cauer (Elliptic) output real part diff in passband');
xlabel('Frequency in Hz');ylabel('Amplitude');grid on;
subplot(2,2,3);plot(freqAxis(1:449),diff_imagPart_FFT(1025:1473)/Half_FFTsize);
title('9th Cauer (Elliptic) output imag part diff in passband');
xlabel('Frequency in Hz');ylabel('Amplitude');grid on;

subplot(2,2,2);plot(freqAxis(641:1024),diff_realPart_FFT(1665:2048)/Half_FFTsize,'r');
title('9th Cauer (Elliptic) output real part diff in stopband');
xlabel('Frequency in Hz');ylabel('Amplitude');grid on;
subplot(2,2,4);plot(freqAxis(641:1024),diff_imagPart_FFT(1665:2048)/Half_FFTsize,'r');
title('9th Cauer (Elliptic) output imag part diff in stopband');
xlabel('Frequency in Hz');ylabel('Amplitude');grid on;

%-------------------------------------------------------------------------------------------------------------

disp('Please choose one of options: track_power_data = 1, print_power_comp = 2');
which_loop = input('    my option = ');

if which_loop == 1

   disp('Comparison with Matlab simulation & DSP56307EVM implementation');
   disp('in terms of Parseval Realation');

   matlab_sim_timePower
   matlab_sim_freqPower

   dsp_impl_timePower
   dsp_impl_freqPower

   matlab_outputSignal_timePower
   matlab_outputSignal_freqPower

   dsp_impl_outputSignal_timePower
   dsp_impl_outputSignal_freqPower

   diff_outputSignal_timePower
```

```
   diff_outputSignal_freqPower

elseif which_loop == 2

   iTitle_1 = 'print_sim_impl_power_comp_cauer9.txt';
   fid_1 = fopen(iTitle_1,'wr');
   fprintf(fid_1, '\t  Comparison with Matlab simulation & DSP56307EVM implementation\n\n');
   fprintf(fid_1, '\t  by a 9th order Cauer WD Filter in terms of Parseval Realation\n\n\n');

   fprintf(fid_1, '        Simulation in time domain: Input Signal Power = %10.14f\n',matlab_sim_timePower);
   fprintf(fid_1, '        Simulation in freq domain: Input Signal Power = %10.14f\n',matlab_sim_freqPower);

   fprintf(fid_1, '\n');

   fprintf(fid_1, '    Implementation in time domain: Input Signal Power = %10.14f\n',dsp_impl_timePower);
   fprintf(fid_1, '    Implementation in freq domain: Input Signal Power = %10.14f\n',dsp_impl_freqPower);

   fprintf(fid_1, '\n');

   fprintf(fid_1, '       Simulation in time domain: Output Signal Power =  %10.14f\n',...
                                    matlab_outputSignal_timePower);
   fprintf(fid_1, '       Simulation in freq domain: Output Signal Power =  %10.14f\n',...
                                    matlab_outputSignal_freqPower);

   fprintf(fid_1, '\n');

   fprintf(fid_1, '   Implementation in time domain: Output Signal Power =  %10.14f\n',...
       dsp_impl_outputSignal_timePower);
   fprintf(fid_1, '   Implementation in freq domain: Output Signal Power =  %10.14f\n',...
       dsp_impl_outputSignal_freqPower);

   fprintf(fid_1, '\n');

   fprintf(fid_1, '\t  The following is to compare power difference of output signal\n');
   fprintf(fid_1, '\t\t  between simulation and DSP implementation\n\n');
   fprintf(fid_1, '       Output Signal: Power Difference in time domain =  %10.14f\n',...
                                    diff_outputSignal_timePower);
   fprintf(fid_1, '       Output Signal: Power Difference in freq domain =  %10.14f\n',...
                                    diff_outputSignal_freqPower);

   fclose(fid_1);

else

   disp('Your option is out of range!');

end

   disp('Have a nice day!');
```

7. Two_adap.m

```
function [b1,b2]=Two_adap(r,a1,a2)

%Two_Port Adaptor
t=a2-a1;
b1=a2+r*t;
b2=a1+r*t;
```

8. wdfilter.m

```
function  output = wdfilter(FFTsize,input)

%The 9th Cauer (Elliptic) WD Lattice Adaptor
a=zeros(18,1);
b=zeros(18,1);

% Coefficients for Satisfying the Specifications
% fp=3.5 kHz, ap=0.3 dB, fs=4.0 kHz, as=80.0 dB, F=16 kHz
%The following coefficients are related to 24 bits
r0  =    0.603586;
r1  =   -0.483105;
r2  =    0.689629;
r3  =   -0.697908;
r4  =    0.409262;
r5  =   -0.859177;
r6  =    0.251224;
r7  =   -0.959691;
r8  =    0.189050;
%make as input signal for the purposes of the impulse
%ImpulseInput(1)=1;%ImpulseInput(2:FFTsize)=zeros(FFTsize-1,1);
%input(1)=1;input(2:FFTsize)=zeros(FFTsize-1,1);

for n=1:FFTsize

% Goes to Upper Section
```

```
a(1) = input(n);

% Calls the function created

[b(1),b(2)]=Two_adap(r0,a(1),a(2));

% Next State

a(2)=b(2);

% Connection Constraint

a(7)=b(1);

[b(9),b(10)]=Two_adap(r4,a(9),a(10));
a(10)=b(10);
a(8)=b(9);

[b(7),b(8)]=Two_adap(r3,a(7),a(8));
a(15)=b(7);

[b(17),b(18)]= Two_adap(r8,a(17),a(18));
a(18)=b(18);
a(16)=b(17);

% Ends Upper Section

[b(15),b(16)]=Two_adap(r7,a(15),a(16));

% Goes to Lower Section

a(3)=a(1);

[b(5),b(6)]=Two_adap(r2,a(5),a(6));

a(6)=b(6);
a(4)=b(5);

[b(3),b(4)]=Two_adap(r1,a(3),a(4));

a(11)=b(3);

[b(13),b(14)]=Two_adap(r6,a(13),a(14));
a(14)=b(14);
a(12)=b(13);

% Ends Lower Section

[b(11),b(12)]=Two_adap(r5,a(11),a(12));

% Output as Response

output(n)=(b(15)+b(11))/2;

% Next State

a(5)=b(4);
a(13)=b(12);
a(9)=b(8);
a(17)=b(16);

end

%figure(1);stem(output);axis([0 70 -0.2 0.4]);grid on;

%make the file, called 'ImpulseOutput.dat'
%MATLAB> who/diary ImpulseOutput.dat/output'/diary off
```

Source Codes - Group 2

Directory: F:\WD_Filter_Design\common_source

File Name:

1. cheby_cal.c

```
/* To calculate the coefficients for Chebyshev filters */
#include <stdio.h>
#include <math.h>
#include "cheby_var_func.h"
#include "public_func.c"

main()
```

```
{
            wholeStar();
            printf("* To calculate the coeffieients for Chebyshev filter         *\n");
            introduction();
            wholeStar();

            printf("\n\t    Passband Frequency:   fp = ");scanf("%f",&passFreq);
            printf("\t    Passband Attenuation: ap = ");scanf("%f",&passAtten);
            printf("\t    Stopband Frequency:   fs = ");scanf("%f",&stopFreq);
            printf("\t    Stopband Attenuation: as = ");scanf("%f",&stopAtten);
            printf("\t    Sampling Frequency:    F = ");scanf("%f",&samplingFreq);printf("\n");

            tmp_stopAtten = stopAtten / 10.0;
            tmp_passAtten = passAtten / 10.0;

            stopRipple = sqrt(pow(10.0,tmp_stopAtten)-1.0);
            passRipple = sqrt(pow(10.0,tmp_passAtten)-1.0);

            stopTransFreq = tan(PI*stopFreq/samplingFreq);
            passTransFreq = tan(PI*passFreq/samplingFreq);

            aux_k0 = sqrt(stopTransFreq/passTransFreq);
            aux_k1 = pow(aux_k0,2) + sqrt(pow(aux_k0,4) - 1);
            aux_c3 = aux_k1;

            minFilterDegree = (c1*log(c2*stopRipple/passRipple))/log(aux_c3);

            wholeStar();

    if  (minFilterDegree < 10.0)

            printf("* Minimum filter Order is %f                              *\n",minFilterDegree);
    else

            printf("* Minimum filter Order is %f                             *\n",minFilterDegree);
            printf("* Please Choose the Filter Order N                              *\n");
            wholeStar();
            printf("\n\t    The Selected Filter Order: N = ");
            scanf("%d",&selectedFilterDegree);printf("\n");

    passRipple_min  = (2*stopRipple)/(pow(aux_k1,selectedFilterDegree));
            passRipple_star = passRipple;

            passAtten_star = 10*log10(1 + pow(passRipple_star,2));
            stopAtten_star = 10*log10(1 + pow(stopRipple, 2));

            wholeStar();emptyStar();
    printf("* The actual attenuation in passband is  %f dB         *\n", passAtten_star);
            printf("* The actual attenuation in stopband is %f dB         *\n", stopAtten_star);
    emptyStar();wholeStar();

            aux_n = pow(selectedFilterDegree,-1);

    aux_w = pow(pow(passRipple_star,-1)+sqrt(pow(passRipple_star,-2)+1),aux_n);

    aux_r = (aux_w-(1/aux_w))*passTransFreq;

            r_0 = (2-aux_r)/(2+aux_r);

            printf("\n");
            printf("\t\tcoefficient_0  =    %f\n",r_0);

    for (index=1;index<=(selectedFilterDegree-1)/2;index++)

{
            aux_A = aux_r*cos(PI*index/selectedFilterDegree);

            aux_B_tmp = (pow(aux_w,2)+pow(1/aux_w,2)-2*cos(2*PI*index/selectedFilterDegree));
            aux_B = aux_B_tmp*pow(passTransFreq,2)/4;

            r_1 = (aux_A - aux_B - 1)/(aux_A + aux_B + 1);

    if(2*index-1 < 10)
    {
      if (r_1 > -10000 && r_1 < 0)
      printf("\t\tcoefficient_%d  =   %f\n",2*index-1,r_1);
      else
      printf("\t\tcoefficient_%d  =  %f\n",2*index-1,r_1);
    }
    else
    {
      if (r_1 > -10000 && r_1 < 0)
      printf("\t\tcoefficient_%d =   %f\n",2*index-1,r_1);
      else
      printf("\t\tcoefficient_%d =  %f\n",2*index-1,r_1);
    }

    r_2 = (1 - aux_B)/(1 + aux_B);

    if(2*index < 10)
    {
```

```
        if(r_2 > 10000)
        printf("\t\tcoefficient_%d  =   %f\n",2*index,r_2);
        else
        printf("\t\tcoefficient_%d  =    %f\n",2*index,r_2);
      }
      else
      {
        if(r_2 > 10000)
        printf("\t\tcoefficient_%d =   %f\n",2*index,r_2);
        else
        printf("\t\tcoefficient_%d =    %f\n",2*index,r_2);
      }

    }

          printf("\n");
          wholeStar();
          tailPrint();
          wholeStar();

    return 0;

}
```

2. cheby_var_func.c

```
#define         PI  3.141592654
#define         c1  1
#define         c2  2

float       passFreq, passAtten, stopFreq, stopAtten, samplingFreq;

double      tmp_stopAtten, tmp_passAtten;

double      stopRipple, passRipple, stopTransFreq, passTransFreq;

double      aux_k0, aux_k1, aux_c3, minFilterDegree;

int         selectedFilterDegree;

double      passRipple_min, passRipple_star;

double      passAtten_star, stopAtten_star;

double      aux_n, aux_w, aux_r;

int         index;

double      r_0, aux_A, aux_B_tmp, aux_B, r_1, r_2;

void        wholeStar(void);

void        emptyStar(void);

void        introduction(void);

void        tailPrint(void);
```

Directory: F:\WD_Filter_Design\dsp_impl_cheby5

File Name:

1. cheby_5th_const_templ.h

```
#define  FFTSIZE                        2*1024

#define  ADDRESS_CHEBY_LOCAL_VAR        0x1000
#define  ADDRESS_SIMULATION_DATA        0x6000
#define  ADDRESS_CHEBY_OUTPUT           0x8000


#pragma asm

CHEBY_SAMPLE_SIZE_ASM          equ          2048

ADDRESS_CHEBY_LOCAL_VAR_ASM    equ          $1000
ADDRESS_CHEBY_OUTPUT_ASM       equ          $6000
ADDRESS_SIMULATION_DATA_ASM    equ          $8000

#pragma endasm

/*The following coefficients are related to 24 bits*/

#define  r0   0.675837
```

```
#define  r1  -0.583980
#define  r2   0.678323
#define  r3  -0.846811
#define  r4   0.387688


_fract adap_1(_fract  x1,  _fract  x2, const _fract r);
_fract adap_2(_fract  x1,  _fract  x2, const _fract r);
```

2. cheby_5th_main.c

```
/* 5th Chebyshev Lattice Lowpass Filter with Specifications*/
/* fp=3000, ap=1.0, fs=5000, as=40, F=16000 */
/* The Following Program is the Implementation with inputSignal*/
#include "cheby_5th_const_templ.h"
//#include "twoPort_Adap.c"

#pragma asm
  include 'cheby_5th_rawData.asm'
#pragma endasm

_internal _fract _X *pointerSimData;
_internal _fract _X output[FFTSIZE]     _at(ADDRESS_CHEBY_OUTPUT);

_internal  _fract _X   a[10];
_internal  _fract _X   b[10];

//extern _external  int   _X   i;
_internal _X int i;

void main(void)
{

pointerSimData = (_fract *)ADDRESS_SIMULATION_DATA;


for (i=0; i <= FFTSIZE-1; i++)
    {
      //a[0]=input[i];

      a[0]=*pointerSimData++;

      b[0]=adap_1(a[0],a[1],r0);
      b[1]=adap_2(a[0],a[1],r0);
      a[1]=b[1];
      a[6]=b[0];

      b[8]=adap_1(a[8],a[9],r4);
      b[9]=adap_2(a[8],a[9],r4);
      a[9]=b[9];
      a[7]=b[8];

      b[6]=adap_1(a[6],a[7],r3);
      b[7]=adap_2(a[6],a[7],r3);

      /* Input for Lower Section */
      a[2]=a[0];

      b[4]=adap_1(a[4],a[5],r2);
      b[5]=adap_2(a[4],a[5],r2);

      a[5]=b[5];
      a[3]=b[4];

      b[2]=adap_1(a[2],a[3],r1);
      b[3]=adap_2(a[2],a[3],r1);

      /* Data as impulse response */
      output[i]=0.5*(b[6]+b[2]);

      a[4]=b[3];

      a[8]=b[7];

      /* Signed data at the same columns */
      //if (output[i]<0)
      //printf("%d    %0.18f\n", i, output[i]);
      //else
      //printf("%d     %0.18f\n", i, output[i]);
      }

}
```

```
Directory: F:\WD_Filter_Design\matlab_sim_cheby5
================================================

File Name:
==========
```

1. dsp_impl_input.m

```
%The piece of code is to plot out input signal in DSP56307EVM implementation basis

clear;

format long;

load dsp_impl_inputSignal.dat;

dsp_input = dsp_impl_inputSignal;

FFTsize = 2048;
Half_FFTsize = FFTsize/2;
time_interval = linspace(0,FFTsize-1,FFTsize);

index = (0:1023)';
freqAxis = (index/128)*1000;

%-------------------------------------------------------------------------------------------------------------------

%loading input signal for print purpose

disp('Please choose one of options: print_sim_input = 1, show_figure = 2');
which_loop = input('    my option = ');

if which_loop == 1

   inputSignal = dsp_impl_inputSignal;

   iTitle_1 = 'print_sim_impl_input_cheby5.txt';
   fid_1 = fopen(iTitle_1,'wr');
   fprintf(fid_1, '\t The following is input data of Matlab Simulation and DSP Implementation\n\n');
   fprintf(fid_1, '\t\t\t     with a 5th order Chebyshev WD Filter\n\n\n');

   for i = 1:512

      if inputSignal(i) < 0
         fprintf(fid_1,'     %10.14f %20.14f %20.14f %20.14f\n', ...
            inputSignal(i),inputSignal(i+1),inputSignal(i+2),inputSignal(i+3));
      else
         fprintf(fid_1,'      %10.14f %20.14f %20.14f %20.14f\n', ...
            inputSignal(i),inputSignal(i+1),inputSignal(i+2),inputSignal(i+3));
      end

   end

   fclose(fid_1);

elseif which_loop == 2

   disp('Only show input signal in time & frequency domain graphically');

else

   disp('Your option is out of the range!');

end

%-------------------------------------------------------------------------------------------------------------------

figure(1)

subplot(3,1,1);plot(time_interval,dsp_input);axis([0 512 -.25 .25]);grid on;
title('Time Domain: Input Signal for 5th Chebyshev DSP56307EVM implementation');
ylabel('Amplitude in Time domain');

tmp_1 = fft(dsp_input); tmp = abs(fftshift(tmp_1));

subplot(3,1,2);plot(freqAxis,tmp(1025:2048)/Half_FFTsize,'r');grid on;
title('Overall Spectrum: Input Signal for 5th Chebyshev DSP56307EVM implementation');
ylabel('Amplitude');

subplot(3,1,3);plot(freqAxis,tmp(1025:2048)/Half_FFTsize,'m');axis([0 8000 0 .006]);grid on;
title('Noise Level Spectrum: Input Signal for 5th Chebyshev DSP56307EVM implementation');
xlabel('Frequency Domain: Hz');ylabel('Amplitude');
```

2. dsp_impl_output.m

```
%The piece of code is to plot out output signal in DSP56307EVM implementation basis

clear;

format long;

load dsp_impl_inputSignal.dat;
load dsp_impl_outputSignal.dat;
```

```
dsp_input = dsp_impl_inputSignal;

FFTsize = 2048;
Half_FFTsize = FFTsize/2;
time_interval = linspace(0,FFTsize-1,FFTsize);

index = (0:1023)';
freqAxis = (index/128)*1000;

%-----------------------------------------------------------------------------------------------------------

%loading output signal for print purpose

disp('Please choose one of options: print_sim_output = 1, show_figure = 2');
which_loop = input('    my option = ');

if which_loop == 1

   outputSignal = dsp_impl_outputSignal;

   iTitle_1 = 'print_dsp_impl_output_cheby5.txt';
   fid_1 = fopen(iTitle_1,'wr');
   fprintf(fid_1, '\t\t     The following is output data of DSP Implementation\n\n');
   fprintf(fid_1, '\t\t\t    with a 5th order Chebyshev WD Filter\n\n\n');

   for i = 1:512

      if outputSignal(i) < 0
         fprintf(fid_1,'     %10.14f %20.14f %20.14f %20.14f\n', ...
            outputSignal(i),outputSignal(i+1),outputSignal(i+2),outputSignal(i+3));
      else
         fprintf(fid_1,'      %10.14f %20.14f %20.14f %20.14f\n', ...
            outputSignal(i),outputSignal(i+1),outputSignal(i+2),outputSignal(i+3));
      end

   end

   fclose(fid_1);
elseif which_loop == 2

   disp('Only show output signal in time & frequency domain graphically');

else

   disp('Your option is out of the range!');

end

%-----------------------------------------------------------------------------------------------------------

%dsp_output = wdfilter(FFTsize,dsp_input);

dsp_output = dsp_impl_outputSignal;

figure(1)

subplot(3,1,1);plot(time_interval,dsp_output);axis([0 512 -.15 .15]);grid on;
title('Time Domain: Output Signal for 5th Chebyshev DSP56307EVM implementation');
ylabel('Amplitude in Time domain');

tmp_1 = fft(dsp_output); tmp = abs(fftshift(tmp_1));

subplot(3,1,2);plot(freqAxis,tmp(1025:2048)/Half_FFTsize,'r');grid on;
title('Overall Spectrum: Output Signal for 5th Chebyshev DSP56307EVM implementation');
ylabel('Amplitude');

subplot(3,1,3);plot(freqAxis,tmp(1025:2048)/Half_FFTsize,'m');axis([0 8000 0 .006]);grid on;
title('Noise Level Spectrum: Output Signal for 5th Chebyshev DSP56307EVM implementation');
xlabel('Frequency Domain: Hz');ylabel('Amplitude');
```

3. dsp_in_out_load.m

```
%The program is to load the data of input and output signal with DSP56307EVM implementation

clear;

format long;

FFTsize = 2048;
Half_FFTsize = FFTsize/2;
time_interval = linspace(0,FFTsize-1,FFTsize);

load dsp_inputSignal_512x4.dat

load dsp_outputSignal_512x4.dat

tmp1 = dsp_inputSignal_512x4';  dsp_inputSignal = tmp1(:);

tmp2 = dsp_outputSignal_512x4'; dsp_outputSignal = tmp2(:);
```

```
%load index.dat
index = (0:1023)';
freqAxis = (index/128)*1000;

tmp_1 = fft(dsp_inputSignal);  tmp_11 = abs(fftshift(tmp_1));
tmp_2 = fft(dsp_outputSignal); tmp_22 = abs(fftshift(tmp_2));

figure(1)

subplot(2,1,1);plot(time_interval,dsp_inputSignal,'r');axis([0 512 -.25 .25]);grid on;
title('Input Signal of 5th Chebyshev DSP56307EVM Implementation in Time Domain');
ylabel('Amplitude');

subplot(2,1,2);plot(time_interval,dsp_outputSignal);axis([0 512 -.25 .25]);grid on;
title('Output Signal of 5th Chebyshev DSP56307EVM Implementation in Time Domain');
xlabel('Signal in Time Domain');ylabel('Amplitude');

figure(2)

subplot(2,1,1);plot(freqAxis,tmp_11(1025:2048)/Half_FFTsize,'r');grid on;
title('Input Signal of 5th Chebyshev DSP56307EVM Implementation in Frequency Domain');
ylabel('Amplitude');

subplot(2,1,2);plot(freqAxis,tmp_22(1025:2048)/Half_FFTsize);grid on;
title('Output Signal of 5th Chebyshev DSP56307EVM Implementation in Frequency Domain');
xlabel('Signal in Frequency Domain: Hz');ylabel('Amplitude');

figure(3)

subplot(2,1,1);plot(freqAxis,tmp_11(1025:2048)/Half_FFTsize,'r');axis([0 8000 0 .006]);grid on;
title('Noise Level Spectrum: Input Signal of 5th Chebyshev DSP56307EVM Implementation');
ylabel('Amplitude');

subplot(2,1,2);plot(freqAxis,tmp_22(1025:2048)/Half_FFTsize);axis([0 8000 0 .006]);grid on;
title('Noise Level Spectrum: Output Signal of 5th Chebyshev DSP56307EVM Implementation');
xlabel('Signal in Frequency Domain: Hz');ylabel('Amplitude');

%-------------------------------------------------------------------------------------------------------------

disp('Please choose one of options: not_load_dsp_io_data = 1, load_dsp_io_data = 2');
which_loop = input('     my option = ');

if which_loop == 1

   disp('Have a nice day!');

elseif which_loop == 2

   diary dsp_impl_inputSignal.dat;
   dsp_inputSignal
   diary off;

   diary dsp_impl_outputSignal.dat;
   dsp_outputSignal
   diary off;

   disp('       <dsp_impl_inputSignal.dat>  is input  data from EVM implementation.');
   disp('       <dsp_impl_outputSignal.dat> is output data from EVM implementation.');
else

   disp('Your option is out of range!');

end
```

4. impulse_resp.m

```
%The charactorizations of the 5th Chebyshev lowpass lattice filter

clear;
format long;

FFTsize = 2048;
time_interval = linspace(0,FFTsize-1,FFTsize);

impulseInput(1) = 1;impulseInput(2:FFTsize) = zeros(FFTsize-1,1);
impulseOutput = wdfilter(FFTsize,impulseInput);

impulseOutputFFT = fft(impulseOutput,FFTsize);

%load index.dat
index = (0:1023)';
freqAxis = (index/128)*1000;

%-------------------------------------------------------------------------------------------------------------

figure(1)

subplot(2,1,1);stem(time_interval,impulseOutput,'r');axis([0,32,-0.2,0.4]);grid on;
title('Impulse Response of the 5th Chebyshev Lowpass Filter');
```

```
subplot(2,1,2);plot(fftshift(abs(fft((impulseOutput)))));grid on;
title('Impulse Frequency Spectrum of the 5th Chebyshev Lowpass Filter');

impulseMagnitute   = abs(impulseOutputFFT(1:1:FFTsize/2));
impulseAttenuation = -20*log10(impulseMagnitute);

%-------------------------------------------------------------------------------------------------------------

figure(2)

plot(freqAxis,impulseAttenuation);axis([0,8000,0,220]);grid on;
text(500,190,'Sampling Frequency  =  16  kHz');
text(500,170,'Passband Frequency =  3.0 kHz');
text(500,150,'Stopband Frequency  =  5.0 kHz');
text(500,130,'Minimum attenuation in the stopband is 40.0 dB');
title('Attenuation Response of the 5th Chebyshev Lowpass Filter');
xlabel('Frequency in Hz');ylabel('Attenuation in dB');

figure(3)

plot(freqAxis,impulseMagnitute,'r');axis([0,8000,0,1]);grid on;
text(4500,0.75,'Sampling Frequency  =  16  kHz');
text(4500,0.55,'Passband Frequency =  3.0 kHz');
text(4500,0.35,'Stopband Frequency  =  5.0 kHz');
title('Magnitute Response of the 5th Chebyshev Lowpass Filter');
xlabel('Frequency in Hz');ylabel('Magnitute in dB');


figure(4)

subplot(2,1,1);plot(freqAxis,impulseMagnitute);axis([0 3500 0.85 1]);grid on;
text(1200,0.875,'Magnitute response in the passband');
title('Passband Magnitute Response of the 5th Chebyshev Lowpass Filter');
ylabel('Magnitute in dB');

subplot(2,1,2);plot(freqAxis,impulseAttenuation);axis([0 3500 0 1.2]);grid on;
text(1050,1.1,'Maximum attenuation in the passband is 1.0 dB');
title('Passband Attenuation Response of the 5th Chebyshev Lowpass Filter');
ylabel('Attenuation in dB');


5. matlab_sim_output.m


%The piece of code is to plot out output signal in Matlab simulation basis

clear;

format long;

load matlab_sim_inputSignal.dat;

matlab_input = matlab_sim_inputSignal;

FFTsize = 2048;
Half_FFTsize = FFTsize/2;
time_interval = linspace(0,FFTsize-1,FFTsize);

index = (0:1023)';
freqAxis = (index/128)*1000;

matlab_output = wdfilter(FFTsize,matlab_input);

%-------------------------------------------------------------------------------------------------------------

%loading output signal for print purpose

disp('Please choose one of options: print_sim_output = 1, show_figure = 2');
which_loop = input('    my option = ');

if which_loop == 1

   outputSignal = matlab_output';

   iTitle_1 = 'print_matlab_sim_output_cheby5.txt';
   fid_1 = fopen(iTitle_1,'wr');
   fprintf(fid_1, '\t\t     The following is output data of Matlab Simulation\n\n');
   fprintf(fid_1, '\t\t\t     with a 5th order Chebyshev WD Filter\n\n\n');

   for i = 1:512

      if outputSignal(i) < 0
         fprintf(fid_1,'     %10.14f %20.14f %20.14f %20.14f\n', ...
            outputSignal(i),outputSignal(i+1),outputSignal(i+2),outputSignal(i+3));
      else
         fprintf(fid_1,'      %10.14f %20.14f %20.14f %20.14f\n', ...
            outputSignal(i),outputSignal(i+1),outputSignal(i+2),outputSignal(i+3));
      end

   end

   fclose(fid_1);
```

```
elseif which_loop == 2

   disp('Only show output signal in time & frequency domain graphically');

else

   disp('Your option is out of the range!');

end

%-------------------------------------------------------------------------------------------------------------

figure(1)

subplot(3,1,1);plot(time_interval,matlab_output);axis([0 512 -.15 .15]);grid on;
title('Time Domain: Output Signal for 5th Chebyshev Matlab Simulation');
ylabel('Amplitude in Time domain');

tmp_1 = fft(matlab_output); tmp = abs(fftshift(tmp_1));

subplot(3,1,2);plot(freqAxis,tmp(1025:2048)/Half_FFTsize,'r');grid on;
title('Overall Spectrum: Output Signal for 5th Chebyshev Matlab Simulation');
ylabel('Amplitude');

subplot(3,1,3);plot(freqAxis,tmp(1025:2048)/Half_FFTsize,'m');axis([0 8000 0 .006]);grid on;
title('Noise Level Spectrum: Output Signal for 5th Chebyshev Matlab Simulation');
xlabel('Frequency Domain: Hz');ylabel('Amplitude');


6. sim_impl_comp.m


%This program is to compare with the simulation & DSP implementation in multi-ways

clear;

format long;

FFTsize = 2048;
Half_FFTsize = FFTsize/2;
time_interval = linspace(0,FFTsize-1,FFTsize);

%load index.dat
index = (0:1023)';
freqAxis = (index/128)*1000;

%-------------------------------------------------------------------------------------------------------------
%This zone is to load simulation & implementation data

load matlab_sim_inputSignal.dat;
load dsp_impl_inputSignal.dat;
load dsp_impl_outputSignal.dat;

matlab_sim_input = matlab_sim_inputSignal;

%-------------------------------------------------------------------------------------------------------------

matlab_outputSignal = wdfilter(FFTsize,dsp_impl_inputSignal);

matlab_FFT_outputSignal = fft(matlab_outputSignal);

matlab_plotFFT_outputSignal = abs(fftshift(matlab_FFT_outputSignal));

%-------------------------------------------------------------------------------------------------------------

dsp_FFT_outputSignal = fft(dsp_impl_outputSignal);

dsp_plotFFT_outputSignal = abs(fftshift(dsp_FFT_outputSignal));

diff_outputSignal = matlab_outputSignal' - dsp_impl_outputSignal;

%-------------------------------------------------------------------------------------------------------------
%The section is to compare with the real part and imaginary part between Matlab & DSP56307EVM, respectively

matlab_FFT_outputSignal_real = real(fft(matlab_outputSignal));
matlab_FFT_outputSignal_imag = imag(fft(matlab_outputSignal));

dsp_FFT_outputSignal_real = real(fft(dsp_impl_outputSignal));
dsp_FFT_outputSignal_imag = imag(fft(dsp_impl_outputSignal));

diff_realPart_FFT = matlab_FFT_outputSignal_real' - dsp_FFT_outputSignal_real;
diff_imagPart_FFT = matlab_FFT_outputSignal_imag' - dsp_FFT_outputSignal_imag;

%-------------------------------------------------------------------------------------------------------------
%The section is to compare with the powers in time domain & frequnecy domain, using Parseval's Relation

   matlab_sim_timePower = sum(matlab_sim_input.^2);
   FFT_matlab_sim_real = real(fft(matlab_sim_input)); FFT_matlab_sim_imag = imag(fft(matlab_sim_input));
   matlab_sim_freqPower = sum(FFT_matlab_sim_real.^2 + FFT_matlab_sim_imag.^2)/FFTsize;

   dsp_impl_timePower = sum(dsp_impl_inputSignal.^2);
```

```
   FFT_dsp_impl_real = real(fft(dsp_impl_inputSignal)); FFT_dsp_impl_imag = imag(fft(dsp_impl_inputSignal));
   dsp_impl_freqPower = sum(FFT_dsp_impl_real.^2 + FFT_dsp_impl_imag.^2)/FFTsize;

   matlab_outputSignal_timePower = sum(matlab_outputSignal.^2);
   matlab_outputSignal_freqPower = sum(matlab_FFT_outputSignal_real.^2+matlab_FFT_outputSignal_imag.^2)/FFTsize;

   dsp_impl_outputSignal_timePower = sum(dsp_impl_outputSignal.^2);
   dsp_impl_outputSignal_freqPower = sum(dsp_FFT_outputSignal_real.^2 + dsp_FFT_outputSignal_imag.^2)/FFTsize;

   diff_outputSignal_timePower = abs(matlab_outputSignal_timePower - dsp_impl_outputSignal_timePower);
   diff_outputSignal_freqPower = abs(matlab_outputSignal_freqPower - dsp_impl_outputSignal_freqPower);

%-------------------------------------------------------------------------------------------------------------
figure(1);

subplot(3,1,1);plot(time_interval,matlab_sim_input);axis([0 512 -.25 .25]);grid on;
title('Actual Input Signal in Time Domain with 5th Chebyshev Matlab Simulation');
ylabel('Amplitude');

subplot(3,1,2);plot(time_interval,dsp_impl_inputSignal,'r');axis([0 512 -.25 .25]);grid on;
title('Actual Input Signal in Time Domain with 5th Chebyshev DSP56307EVM Implementation');
ylabel('Amplitude');

subplot(3,1,3);plot(time_interval,matlab_sim_input-dsp_impl_inputSignal);axis([0 512 -.3e-6 .3e-6]);grid on;
title('Input: Amplitude difference in Time Domain Between Simulation and Implementation');
xlabel('Signal in Time Domain');ylabel('Amplitude');

%-------------------------------------------------------------------------------------------------------------

figure(2);

subplot(3,1,1);plot(time_interval,matlab_outputSignal);axis([0 512 -.15 .15]);grid on;
title('Actual Output Signal in Time Domain with 5th Chebyshev Matlab Simulation');
ylabel('Amplitude');

subplot(3,1,2);plot(time_interval,dsp_impl_outputSignal,'r');axis([0 512 -.15 .15]);grid on;
title('Actual Output Signal in Time Domain with 5th Chebyshev DSP56307EVM Implementation');
ylabel('Amplitude');

subplot(3,1,3);plot(time_interval,matlab_outputSignal'-dsp_impl_outputSignal);axis([0 512 -.3e-6 .3e-6]);grid on;
title('Output: Amplitude difference in Time Domain Between Simulation and Implementation');
xlabel('Signal in Time Domain');ylabel('Amplitude');

%-------------------------------------------------------------------------------------------------------------

figure(3);

subplot(2,1,1);plot(freqAxis,matlab_plotFFT_outputSignal(1025:2048)/Half_FFTsize,'r');
grid on;
title('The Spectrum: Actual Output Signal with 5th Chebyshev Matlab Simulation');
xlabel('Frequency in Hz');ylabel('Amplitude');

subplot(2,1,2);plot(freqAxis,dsp_plotFFT_outputSignal(1025:2048)/Half_FFTsize);
grid on;
title('The Spectrum: Actual Output Signal with 5th Chebyshev DSP56307EVM Implementation');
xlabel('Frequency in Hz');ylabel('Amplitude');

%-------------------------------------------------------------------------------------------------------------

figure(4);

subplot(2,1,1);plot(freqAxis,matlab_plotFFT_outputSignal(1025:2048)/Half_FFTsize,'r');
axis([0 8000 0 .004]);grid on;
title('Noise Level Spectrum: Actual Output Signal with 5th Chebyshev Matlab Simulation');
xlabel('Frequency in Hz');ylabel('Amplitude');

subplot(2,1,2);plot(freqAxis,dsp_plotFFT_outputSignal(1025:2048)/Half_FFTsize);
axis([0 8000 0 .004]);grid on;
title('Noise Level Spectrum: Actual Output Signal with 5th Chebyshev DSP56307EVM Implementation');
xlabel('Frequency in Hz');ylabel('Amplitude');

%-------------------------------------------------------------------------------------------------------------

figure(5);

subplot(2,1,1);plot(freqAxis,diff_realPart_FFT(1025:2048)/Half_FFTsize,'r');grid on;
title('The Output Spectrum Difference in real part between 5th Chebyshev Simulation and Implementation');
xlabel('Frequency in Hz');ylabel('Amplitude');

subplot(2,1,2);plot(freqAxis,diff_imagPart_FFT(1025:2048)/Half_FFTsize);grid on;
title('The Output Spectrum Difference in imag part between 5th Chebyshev Simulation and Implementation');
xlabel('Frequency in Hz');ylabel('Amplitude');

%-------------------------------------------------------------------------------------------------------------

figure(6);

subplot(2,2,1);plot(freqAxis(1:449),diff_realPart_FFT(1025:1473)/Half_FFTsize);
title('5th Chebyshev output real part diff in passband');
xlabel('Frequency in Hz');ylabel('Amplitude');grid on;
subplot(2,2,3);plot(freqAxis(1:449),diff_imagPart_FFT(1025:1473)/Half_FFTsize);
title('5th Chebyshev output imag part diff in passband');
```

```
xlabel('Frequency in Hz');ylabel('Amplitude');grid on;

subplot(2,2,2);plot(freqAxis(641:1024),diff_realPart_FFT(1665:2048)/Half_FFTsize,'r');
title('5th Chebyshev output real part diff in stopband');
xlabel('Frequency in Hz');ylabel('Amplitude');grid on;
subplot(2,2,4);plot(freqAxis(641:1024),diff_imagPart_FFT(1665:2048)/Half_FFTsize,'r');
title('5th Chebyshev output imag part diff in stopband');
xlabel('Frequency in Hz');ylabel('Amplitude');grid on;

%-------------------------------------------------------------------------------------------------------------

disp('Please choose one of options: track_power_data = 1, print_power_comp = 2');
which_loop = input('    my option = ');

if which_loop == 1

   disp('Comparison with Matlab simulation & DSP56307EVM implementation');
   disp('in terms of Parseval Realation');

   matlab_sim_timePower
   matlab_sim_freqPower

   dsp_impl_timePower
   dsp_impl_freqPower

   matlab_outputSignal_timePower
   matlab_outputSignal_freqPower

   dsp_impl_outputSignal_timePower
   dsp_impl_outputSignal_freqPower

   diff_outputSigan1_timePower
   diff_outputSignal_freqPower

elseif which_loop == 2

   iTitle_1 = 'print_sim_impl_power_comp_cheby5.txt';
   fid_1 = fopen(iTitle_1,'wr');
   fprintf(fid_1, '\t  Comparison with Matlab simulation & DSP56307EVM implementation\n\n');
   fprintf(fid_1, '\t  by a 5th order Chebyshev WD Filter in terms of Parseval Realation\n\n\n');

   fprintf(fid_1, '        Simulation in time domain: Input Signal Power = %10.14f\n',matlab_sim_timePower);
   fprintf(fid_1, '        Simulation in freq domain: Input Signal Power = %10.14f\n',matlab_sim_freqPower);

   fprintf(fid_1, '\n');

   fprintf(fid_1, '    Implementation in time domain: Input Signal Power = %10.14f\n',dsp_impl_timePower);
   fprintf(fid_1, '    Implementation in freq domain: Input Signal Power = %10.14f\n',dsp_impl_freqPower);

   fprintf(fid_1, '\n');

   fprintf(fid_1, '       Simulation in time domain: Output Signal Power =  %10.14f\n',....
                                    matlab_outputSignal_timePower);
   fprintf(fid_1, '       Simulation in freq domain: Output Signal Power =  %10.14f\n',....
                                    matlab_outputSignal_freqPower);

   fprintf(fid_1, '\n');

   fprintf(fid_1, '   Implementation in time domain: Output Signal Power =  %10.14f\n',....
       dsp_impl_outputSignal_timePower);
   fprintf(fid_1, '   Implementation in freq domain: Output Signal Power =  %10.14f\n',....
       dsp_impl_outputSignal_freqPower);

   fprintf(fid_1, '\n');

   fprintf(fid_1, '\t  The following is to compare power difference of output signal\n');
   fprintf(fid_1, '\t\t  between simulation and DSP implementation\n\n');
   fprintf(fid_1, '       Output Signal: Power Difference in time domain =  %10.14f\n',....
                                    diff_outputSignal_timePower);
   fprintf(fid_1, '       Output Signal: Power Difference in freq domain =  %10.14f\n',....
                                    diff_outputSignal_freqPower);

   fclose(fid_1);


else

   disp('Your option is out of range!');

end


if which_loop == 1

   disp('Comparison is done. Have a nice day!');

elseif which_loop == 2

   diary off;
   disp('All data are saved at the file, called <print_sim_impl_power_comp_cheby5.dat>');

else
```

```
   disp('Have a nice day!');
end
```

7. wdfilter.m

```
function  output = wdfilter(FFTsize,input)

%The 5th Chebyshev WD Lattice Adaptor

a=zeros(10,1);
b=zeros(10,1);

% Coefficients for Satisfying the Specifications
% fp=3.0 kHz, ap=1.0 dB, fs=5.0 kHz, as=40.0 dB, F=16 kHz
%The following coefficients are related to 24 bits

r0  =    0.675837;
r1  =   -0.583980;
r2  =    0.678323;
r3  =   -0.846811;
r4  =    0.387688;

%make as input signal for the purposes of the impulse
%ImpulseInput(1)=1;%ImpulseInput(2:FFTsize)=zeros(FFTsize-1,1);
%input(1)=1;input(2:FFTsize)=zeros(FFTsize-1,1);

for n=1:FFTsize

% Goes to Upper Section

a(1) = input(n);


% Calls the function, Two_adap.m

[b(1),b(2)]=Two_adap(r0,a(1),a(2));

% Next State

a(2)=b(2);

% Connection Constraint

a(7)=b(1);

[b(9),b(10)]=Two_adap(r4,a(9),a(10));
a(10)=b(10);
a(8)=b(9);

% Ends Upper Section

[b(7),b(8)]=Two_adap(r3,a(7),a(8));

% Goes to Lower Section

a(3)=a(1);

[b(5),b(6)]=Two_adap(r2,a(5),a(6));

a(6)=b(6);
a(4)=b(5);

[b(3),b(4)]=Two_adap(r1,a(3),a(4));

% Ends Lower Section

% Output as Response

output(n)=(b(7)+b(3))/2;

% Next State

a(5)=b(4);
a(9)=b(8);

end

figure(1);stem(output);axis([0 70 -0.2 0.4]);grid on;
```

Source Codes - Group 3

Directory: F:\WD_Filter_Design\common_source

File Name:
==========

1. butter_cal.c

```
/* To calculate the coefficients for Butterworth filters */
#include <stdio.h>
#include <math.h>
#include "butter_var_func.h"
#include "public_func.c"

main()

{

                wholeStar();
                printf("* To calculate the coeffieients for Butterworth filter       *\n");
                introduction();
                wholeStar();

                printf("\n\t    Passband Frequency:   fp = ");scanf("%f",&passFreq);
                printf("\t    Passband Attenuation: ap = ");scanf("%f",&passAtten);
                printf("\t    Stopband Frequency:   fs = ");scanf("%f",&stopFreq);
                printf("\t    Stopband Attenuation: as = ");scanf("%f",&stopAtten);
                printf("\t    Sampling Frequency:    F = ");scanf("%f",&samplingFreq);printf("\n");

                tmp_stopAtten = stopAtten / 10.0;
                tmp_passAtten = passAtten / 10.0;

                stopRipple = sqrt(pow(10.0,tmp_stopAtten)-1.0);
                passRipple = sqrt(pow(10.0,tmp_passAtten)-1.0);

                stopTransFreq = tan(PI*stopFreq/samplingFreq);
                passTransFreq = tan(PI*passFreq/samplingFreq);

                aux_k0 = sqrt(stopTransFreq/passTransFreq);

                aux_c3 = pow(aux_k0,2);
                minFilterDegree = (c1*log(c2*stopRipple/passRipple))/log(aux_c3);

                wholeStar();

        if  (minFilterDegree < 10.0)

                printf("* Minimum filter Order is %f                          *\n",minFilterDegree);
        else

                printf("* Minimum filter Order is %f                         *\n",minFilterDegree);
                printf("* Please Choose the Filter Order N                        *\n");
                wholeStar();
                printf("\n\t    The Selected Filter Order: N = ");
                scanf("%d",&selectedFilterDegree);printf("\n");
                wholeStar();

                aux_n = 2*pow(selectedFilterDegree,-1);

                aux_minus_kp = pow(passRipple,aux_n) - pow(passTransFreq,2);
                aux_plus_kp  = pow(passRipple,aux_n) + pow(passTransFreq,2);

                aux_minus_ks = pow(stopRipple,aux_n) - pow(stopTransFreq,2);
                aux_plus_ks  = pow(stopRipple,aux_n) + pow(stopTransFreq,2);

                aux_kp = aux_minus_kp/aux_plus_kp;
                aux_ks = aux_minus_ks/aux_plus_ks;

                gamma = aux_ks;

                tmp_r0 = sqrt(1-pow(gamma,2));
                r_0 = (1+gamma-tmp_r0)/(1+gamma+tmp_r0);

                printf("\n");
                printf("\t\tcoefficient_0  =    %f\n",r_0);

        for (index=1;index<=(selectedFilterDegree-1)/2;index++)

      {
                tmp_r1 = (sqrt(1-pow(gamma,2)))*(cos(PI*index/selectedFilterDegree));

                r_1    = (tmp_r1 - 1)/(tmp_r1 + 1);

        if(2*index-1 < 10)
        {
          if (r_1 > -10000 && r_1 < 0)
          printf("\t\tcoefficient_%d  =   %f\n",2*index-1,r_1);
          else
          printf("\t\tcoefficient_%d  =  %f\n",2*index-1,r_1);
        }
        else
        {
          if (r_1 > -10000 && r_1 < 0)
          printf("\t\tcoefficient_%d =   %f\n",2*index-1,r_1);
```

```
          else
          printf("\t\tcoefficient_%d =  %f\n",2*index-1,r_1);
        }

        r_2 = gamma;

        if(2*index < 10)
        {
          if(r_2 > 10000)
          printf("\t\tcoefficient_%d  =   %f\n",2*index,r_2);
          else
          printf("\t\tcoefficient_%d  =    %f\n",2*index,r_2);
        }
        else
        {
          if(r_2 > 10000)
          printf("\t\tcoefficient_%d =   %f\n",2*index,r_2);
          else
          printf("\t\tcoefficient_%d =    %f\n",2*index,r_2);
        }

      }

              printf("\n");
              wholeStar();
              tailPrint();
              wholeStar();

        return 0;

}
```

2. butter_var_func.h

```
#define         PI  3.141592654
#define         c1  1
#define         c2  1

float       passFreq, passAtten, stopFreq, stopAtten, samplingFreq;

double      tmp_stopAtten, tmp_passAtten;

double      stopRipple, passRipple, stopTransFreq, passTransFreq;

double      aux_k0, aux_c3, minFilterDegree;

int         selectedFilterDegree;

double      aux_n, aux_minus_kp, aux_plus_kp, aux_minus_ks, aux_plus_ks;
double      aux_kp, aux_ks;
double      gamma;

int         index;

double      tmp_r0, r_0, tmp_r1, r_1, r_2;

void        wholeStar(void);

void        emptyStar(void);

void        introduction(void);

void        tailPrint(void);
```

Directory: F:\WD_Filter_Design\dsp_impl_butter7

File Name:

1. butter_7th_const_templ.h

```
#define  FFTSIZE                          2*1024

#define  ADDRESS_BUTTER_LOCAL_VAR         0x1000
#define  ADDRESS_SIMULATION_DATA          0x6000
#define  ADDRESS_BUTTER_OUTPUT            0x8000


#pragma asm

BUTTER_SAMPLE_SIZE_ASM          equ         2048

ADDRESS_BUTTER_LOCAL_VAR_ASM    equ         $1000
ADDRESS_BUTTER_OUTPUT_ASM       equ         $6000
ADDRESS_SIMULATION_DATA_ASM     equ         $8000
```

```
#pragma endasm

/*The following coefficients are related to 24 bits*/

#define  r0   0.011606
#define  r1  -0.052229
#define  r2   0.023209
#define  r3  -0.232042
#define  r4   0.023209
#define  r5  -0.636044
#define  r6   0.023209


_fract adap_1(_fract  x1,  _fract  x2, const _fract r);
_fract adap_2(_fract  x1,  _fract  x2, const _fract r);
```

2. butter_7th_main.c

```
/* 9th Cauer(Elliptic) Lattice Lowpass Filter with Specifications*/
/* fp=3500, ap=0.2, fs=4100, as=60, F=16000 */
/* The Following Program is the Implementation for Impulse response*/
#include "butter_7th_const_templ.h"
//#include "twoPort_Adap.c"

#pragma asm
  include 'butter_7th_rawData.asm'
#pragma endasm

_internal _fract _X *pointerSimData;
_internal _fract _X output[FFTSIZE]     _at(ADDRESS_BUTTER_OUTPUT);

_internal  _fract _X   a[14];
_internal  _fract _X   b[14];

//extern _external  int   _X   i;
_internal _X int i;

void main(void)
{

pointerSimData = (_fract *)ADDRESS_SIMULATION_DATA;


for (i=0; i <= FFTSIZE-1; i++)
    {
      //a[0]=input[i];

      a[0]=*pointerSimData++;

      b[0]=adap_1(a[0],a[1],r0);
      b[1]=adap_2(a[0],a[1],r0);
      a[1]=b[1];
      a[6]=b[0];

      b[8]=adap_1(a[8],a[9],r4);
      b[9]=adap_2(a[8],a[9],r4);
      a[9]=b[9];
      a[7]=b[8];

      b[6]=adap_1(a[6],a[7],r3);
      b[7]=adap_2(a[6],a[7],r3);

      /* Input for Lower Section */
      a[2]=a[0];

      b[4]=adap_1(a[4],a[5],r2);
      b[5]=adap_2(a[4],a[5],r2);

      a[5]=b[5];
      a[3]=b[4];

      b[2]=adap_1(a[2],a[3],r1);
      b[3]=adap_2(a[2],a[3],r1);
      a[10]=b[2];

      b[12]=adap_1(a[12],a[13],r6);
      b[13]=adap_2(a[12],a[13],r6);
      a[13]=b[13];
      a[11]=b[12];

      b[10]=adap_1(a[10],a[11],r5);
      b[11]=adap_2(a[10],a[11],r5);

      /* Data as impulse response */
      output[i]=0.5*(b[6]+b[10]);

      a[4]=b[3];
      a[12]=b[11];
      a[8]=b[7];
```

```
      /* Signed data at the same columns */
      //if (output[i]<0)
      //printf("%d    %0.18f\n", i, output[i]);
      //else
      //printf("%d     %0.18f\n", i, output[i]);
      }

}
```

Directory: F:\WD_Filter_Design\matlab_sim_butter7
==========

File Name:
==========

1. dsp_impl_input.m

```
%The piece of code is to plot out input signal in DSP56307EVM implementation basis

clear;

format long;

load dsp_impl_inputSignal.dat;

dsp_input = dsp_impl_inputSignal;

FFTsize = 2048;
Half_FFTsize = FFTsize/2;
time_interval = linspace(0,FFTsize-1,FFTsize);

index = (0:1023)';
freqAxis = (index/128)*1000;

%-------------------------------------------------------------------------------------------------------------------

%loading input signal for print purpose

disp('Please choose one of options: print_sim_input = 1, show_figure = 2');
which_loop = input('    my option = ');

if which_loop == 1

   inputSignal = dsp_impl_inputSignal;

   iTitle_1 = 'print_sim_impl_input_butter7.txt';
   fid_1 = fopen(iTitle_1,'wr');
   fprintf(fid_1, '\t The following is input data of Matlab Simulation and DSP Implementation\n\n');
   fprintf(fid_1, '\t\t\t    with a 7th order Butterworth WD Filter\n\n\n');

   for i = 1:512

      if inputSignal(i) < 0
         fprintf(fid_1,'     %10.14f %20.14f %20.14f %20.14f\n', ...
            inputSignal(i),inputSignal(i+1),inputSignal(i+2),inputSignal(i+3));
      else
         fprintf(fid_1,'      %10.14f %20.14f %20.14f %20.14f\n', ...
            inputSignal(i),inputSignal(i+1),inputSignal(i+2),inputSignal(i+3));
      end

   end

   fclose(fid_1);

elseif which_loop == 2

   disp('Only show input signal in time & frequency domain graphically');

else

   disp('Your option is out of the range!');

end

%-------------------------------------------------------------------------------------------------------------------

figure(1)

subplot(3,1,1);plot(time_interval,dsp_input);axis([0 512 -.25 .25]);grid on;
title('Time Domain: Input Signal for 7th Butterworth DSP56307EVM implementation');
ylabel('Amplitude in Time domain');

tmp_1 = fft(dsp_input); tmp = abs(fftshift(tmp_1));

subplot(3,1,2);plot(freqAxis,tmp(1025:2048)/Half_FFTsize,'r');grid on;
title('Overall Spectrum: Input Signal for 7th Butterworth DSP56307EVM implementation');
ylabel('Amplitude');

subplot(3,1,3);plot(freqAxis,tmp(1025:2048)/Half_FFTsize,'m');axis([0 8000 0 .006]);grid on;
```

```
title('Noise Level Spectrum: Input Signal for 7th Butterworth DSP56307EVM implementation');
xlabel('Frequency Domain: Hz');ylabel('Amplitude');
```

2. dsp_impl_output.m

```
%The piece of code is to plot out output signal in DSP56307EVM implementation basis

clear;

format long;

load dsp_impl_inputSignal.dat;
load dsp_impl_outputSignal.dat;

dsp_input = dsp_impl_inputSignal;

FFTsize = 2048;
Half_FFTsize = FFTsize/2;
time_interval = linspace(0,FFTsize-1,FFTsize);

index = (0:1023)';
freqAxis = (index/128)*1000;

%-----------------------------------------------------------------------------------------------------------

%loading output signal for print purpose

disp('Please choose one of options: print_sim_output = 1, show_figure = 2');
which_loop = input('    my option = ');

if which_loop == 1

   outputSignal = dsp_impl_outputSignal;

   iTitle_1 = 'print_dsp_impl_output_butter7.txt';
   fid_1 = fopen(iTitle_1,'wr');
   fprintf(fid_1, '\t\t     The following is output data of DSP Implementation\n\n');
   fprintf(fid_1, '\t\t\t    with a 7th order Butterworth WD Filter\n\n\n');

   for i = 1:512

      if outputSignal(i) < 0
         fprintf(fid_1,'     %10.14f %20.14f %20.14f %20.14f\n', ...
            outputSignal(i),outputSignal(i+1),outputSignal(i+2),outputSignal(i+3));
      else
         fprintf(fid_1,'      %10.14f %20.14f %20.14f %20.14f\n', ...
            outputSignal(i),outputSignal(i+1),outputSignal(i+2),outputSignal(i+3));
      end

   end

   fclose(fid_1);


elseif which_loop == 2

   disp('Only show output signal in time & frequency domain graphically');

else

   disp('Your option is out of the range!');

end

%-----------------------------------------------------------------------------------------------------------

%dsp_output = wdfilter(FFTsize,dsp_input);

dsp_output = dsp_impl_outputSignal;

figure(1)

subplot(3,1,1);plot(time_interval,dsp_output);axis([0 512 -.15 .15]);grid on;
title('Time Domain: Output Signal for 7th Butterworth DSP56307EVM implementation');
ylabel('Amplitude in Time domain');

tmp_1 = fft(dsp_output); tmp = abs(fftshift(tmp_1));

subplot(3,1,2);plot(freqAxis,tmp(1025:2048)/Half_FFTsize,'r');grid on;
title('Overall Spectrum: Output Signal for 7th Butterworth DSP56307EVM implementation');
ylabel('Amplitude');

subplot(3,1,3);plot(freqAxis,tmp(1025:2048)/Half_FFTsize,'m');axis([0 8000 0 .006]);grid on;
title('Noise Level Spectrum: Output Signal for 7th Butterworth DSP56307EVM implementation');
xlabel('Frequency Domain: Hz');ylabel('Amplitude');
```

3. dsp_in_out_load.m

```
%The program is to load the data of input and output signal with DSP56307EVM implementation

clear;

format long;

FFTsize = 2048;
Half_FFTsize = FFTsize/2;
time_interval = linspace(0,FFTsize-1,FFTsize);

load dsp_inputSignal_512x4.dat

load dsp_outputSignal_512x4.dat

tmp1 = dsp_inputSignal_512x4';  dsp_inputSignal = tmp1(:);

tmp2 = dsp_outputSignal_512x4'; dsp_outputSignal = tmp2(:);

%load index.dat
index = (0:1023)';
freqAxis = (index/128)*1000;

tmp_1 = fft(dsp_inputSignal);  tmp_11 = abs(fftshift(tmp_1));
tmp_2 = fft(dsp_outputSignal); tmp_22 = abs(fftshift(tmp_2));

figure(1)

subplot(2,1,1);plot(time_interval,dsp_inputSignal,'r');axis([0 512 -.25 .25]);grid on;
title('Input Signal of 7th Butterworth DSP56307EVM Implementation in Time Domain');
ylabel('Amplitude');

subplot(2,1,2);plot(time_interval,dsp_outputSignal);axis([0 512 -.25 .25]);grid on;
title('Output Signal of 7th Butterworth DSP56307EVM Implementation in Time Domain');
xlabel('Signal in Time Domain');ylabel('Amplitude');

figure(2)

subplot(2,1,1);plot(freqAxis,tmp_11(1025:2048)/Half_FFTsize,'r');grid on;
title('Input Signal of 7th Butterworth DSP56307EVM Implementation in Frequency Domain');
ylabel('Amplitude');

subplot(2,1,2);plot(freqAxis,tmp_22(1025:2048)/Half_FFTsize);grid on;
title('Output Signal of 7th Butterworth DSP56307EVM Implementation in Frequency Domain');
xlabel('Signal in Frequency Domain: Hz');ylabel('Amplitude');

figure(3)

subplot(2,1,1);plot(freqAxis,tmp_11(1025:2048)/Half_FFTsize,'r');axis([0 8000 0 .006]);grid on;
title('Noise Level Spectrum: Input Signal of 7th Butterworth DSP56307EVM Implementation');
ylabel('Amplitude');

subplot(2,1,2);plot(freqAxis,tmp_22(1025:2048)/Half_FFTsize);axis([0 8000 0 .006]);grid on;
title('Noise Level Spectrum: Output Signal of 7th Butterworth DSP56307EVM Implementation');
xlabel('Signal in Frequency Domain: Hz');ylabel('Amplitude');

%-------------------------------------------------------------------------------------------------------------------

disp('Please choose one of options: not_load_dsp_io_data = 1, load_dsp_io_data = 2');
which_loop = input('    my option = ');

if which_loop == 1

   disp('Have a nice day!');

elseif which_loop == 2

   diary dsp_impl_inputSignal.dat;
   dsp_inputSignal
   diary off;

   diary dsp_impl_outputSignal.dat;
   dsp_outputSignal
   diary off;

   disp('       <dsp_impl_inputSignal.dat>  is input  data from EVM implementation.');
   disp('       <dsp_impl_outputSignal.dat> is output data from EVM implementation.');
else

   disp('Your option is out of range!');

end
```

4. impulse_resp.m

```
%The charactorizations of the 7th Butterworth lowpass lattice filter

clear;
format long;

FFTsize = 2048;
```

```
time_interval = linspace(0,FFTsize-1,FFTsize);

impulseInput(1) = 1;impulseInput(2:FFTsize) = zeros(FFTsize-1,1);
impulseOutput = wdfilter(FFTsize,impulseInput);

impulseOutputFFT = fft(impulseOutput,FFTsize);

%load index.dat
index = (0:1023)';
freqAxis = (index/128)*1000;

%-----------------------------------------------------------------------------------------------------------------------
-

figure(1)

subplot(2,1,1);stem(time_interval,impulseOutput,'r');axis([0,32,-0.3,0.5]);grid on;
title('Impulse Response of the 7th Butterworth Lowpass Filter');

subplot(2,1,2);plot(fftshift(abs(fft((impulseOutput)))));grid on;
title('Impulse Frequency Spectrum of the 7th Butterworth Lowpass Filter');

impulseMagnitute   = abs(impulseOutputFFT(1:1:FFTsize/2));
impulseAttenuation = -20*log10(impulseMagnitute);

%-----------------------------------------------------------------------------------------------------------------------
-

figure(2)

plot(freqAxis,impulseMagnitute,'r');axis([0,8500,0,1]);grid on;
text(4500,0.75,'Sampling Frequency  =  16000 Hz');
text(4500,0.55,'Passband Frequency =  3400 Hz');
text(4500,0.35,'Stopband Frequency  =  6000 Hz');
title('FIGURE I -- Magnitute Response of the 7th Butterworth Lowpass Filter');
xlabel('Frequency in Hz');ylabel('Magnitute in dB');

figure(3)
plot(freqAxis,impulseAttenuation);axis([0,8000,0,200]);grid on;
text(500,150,'Sampling Frequency  =  16  kHz');
text(500,130,'Passband Frequency =  3.4 kHz');
text(500,110,'Stopband Frequency  =  6.0 kHz');
text(500,90,'Maximum attenuation in the passband is 0.5 dB');
text(500,70,'Minimum attenuation in the stopband is 55.0 dB');
title('Attenuation Response of the 7th Butterworth Lowpass Filter');
xlabel('Frequency in Hz');ylabel('Attenuation in dB');



5. matlab_sim_output.m


%The piece of code is to plot out output signal in Matlab simulation basis

clear;

format long;

load matlab_sim_inputSignal.dat;

matlab_input = matlab_sim_inputSignal;

FFTsize = 2048;
Half_FFTsize = FFTsize/2;
time_interval = linspace(0,FFTsize-1,FFTsize);

index = (0:1023)';
freqAxis = (index/128)*1000;

matlab_output = wdfilter(FFTsize,matlab_input);

%-----------------------------------------------------------------------------------------------------------------------

%loading output signal for print purpose

disp('Please choose one of options: print_sim_output = 1, show_figure = 2');
which_loop = input('    my option = ');

if which_loop == 1

   outputSignal = matlab_output';

   iTitle_1 = 'print_matlab_sim_output_butter7.txt';
   fid_1 = fopen(iTitle_1,'wr');
   fprintf(fid_1, '\t\t     The following is output data of Matlab Simulation\n\n');
   fprintf(fid_1, '\t\t\t    with a 7th order Butterworth WD Filter\n\n\n');

   for i = 1:512

      if outputSignal(i) < 0
         fprintf(fid_1,'     %10.14f %20.14f %20.14f %20.14f\n', ...
            outputSignal(i),outputSignal(i+1),outputSignal(i+2),outputSignal(i+3));
```

```
      else
         fprintf(fid_1,'      %10.14f %20.14f %20.14f %20.14f\n', ...
            outputSignal(i),outputSignal(i+1),outputSignal(i+2),outputSignal(i+3));
      end

   end

   fclose(fid_1);

elseif which_loop == 2

   disp('Only show output signal in time & frequency domain graphically');

else

   disp('Your option is out of the range!');

end

%-------------------------------------------------------------------------------------------------------------

figure(1)

subplot(3,1,1);plot(time_interval,matlab_output);axis([0 512 -.15 .15]);grid on;
title('Time Domain: Output Signal for 7th Butterworth Matlab Simulation');
ylabel('Amplitude in Time domain');

tmp_1 = fft(matlab_output); tmp = abs(fftshift(tmp_1));

subplot(3,1,2);plot(freqAxis,tmp(1025:2048)/Half_FFTsize,'r');grid on;
title('Overall Spectrum: Output Signal for 7th Butterworth Matlab Simulation');
ylabel('Amplitude');

subplot(3,1,3);plot(freqAxis,tmp(1025:2048)/Half_FFTsize,'m');axis([0 8000 0 .006]);grid on;
title('Noise Level Spectrum: Output Signal for 7th Butterworth Matlab Simulation');
xlabel('Frequency Domain: Hz');ylabel('Amplitude');


6. sim_impl_comp.m


%This program is to compare with the simulation & DSP implementation in multi-ways

clear;

format long;

FFTsize = 2048;
Half_FFTsize = FFTsize/2;
time_interval = linspace(0,FFTsize-1,FFTsize);

%load index.dat
index = (0:1023)';
freqAxis = (index/128)*1000;

%-------------------------------------------------------------------------------------------------------------
%This zone is to load simulation & implementation data

load matlab_sim_inputSignal.dat;
load dsp_impl_inputSignal.dat;
load dsp_impl_outputSignal.dat;

matlab_sim_input = matlab_sim_inputSignal;

%-------------------------------------------------------------------------------------------------------------

matlab_outputSignal = wdfilter(FFTsize,dsp_impl_inputSignal);

matlab_FFT_outputSignal = fft(matlab_outputSignal);

matlab_plotFFT_outputSignal = abs(fftshift(matlab_FFT_outputSignal));

%-------------------------------------------------------------------------------------------------------------

dsp_FFT_outputSignal = fft(dsp_impl_outputSignal);

dsp_plotFFT_outputSignal = abs(fftshift(dsp_FFT_outputSignal));

diff_outputSignal = matlab_outputSignal' - dsp_impl_outputSignal;

%-------------------------------------------------------------------------------------------------------------
%The section is to compare with the real part and imaginary part between Matlab & DSP56307EVM, respectively

matlab_FFT_outputSignal_real = real(fft(matlab_outputSignal));
matlab_FFT_outputSignal_imag = imag(fft(matlab_outputSignal));

dsp_FFT_outputSignal_real = real(fft(dsp_impl_outputSignal));
dsp_FFT_outputSignal_imag = imag(fft(dsp_impl_outputSignal));

diff_realPart_FFT = matlab_FFT_outputSignal_real' - dsp_FFT_outputSignal_real;
diff_imagPart_FFT = matlab_FFT_outputSignal_imag' - dsp_FFT_outputSignal_imag;
```

```
%-------------------------------------------------------------------------------------------------------------
%The section is to compare with the powers in time domain & frequnecy domain, using Parseval's Relation

   matlab_sim_timePower = sum(matlab_sim_input.^2);
   FFT_matlab_sim_real = real(fft(matlab_sim_input)); FFT_matlab_sim_imag = imag(fft(matlab_sim_input));
   matlab_sim_freqPower = sum(FFT_matlab_sim_real.^2 + FFT_matlab_sim_imag.^2)/FFTsize;

   dsp_impl_timePower = sum(dsp_impl_inputSignal.^2);
   FFT_dsp_impl_real = real(fft(dsp_impl_inputSignal)); FFT_dsp_impl_imag = imag(fft(dsp_impl_inputSignal));
   dsp_impl_freqPower = sum(FFT_dsp_impl_real.^2 + FFT_dsp_impl_imag.^2)/FFTsize;

   matlab_outputSignal_timePower = sum(matlab_outputSignal.^2);
   matlab_outputSignal_freqPower = sum(matlab_FFT_outputSignal_real.^2+matlab_FFT_outputSignal_imag.^2)/FFTsize;

   dsp_impl_outputSignal_timePower = sum(dsp_impl_outputSignal.^2);
   dsp_impl_outputSignal_freqPower = sum(dsp_FFT_outputSignal_real.^2 + dsp_FFT_outputSignal_imag.^2)/FFTsize;

   diff_outputSignal_timePower = abs(matlab_outputSignal_timePower - dsp_impl_outputSignal_timePower);
   diff_outputSignal_freqPower = abs(matlab_outputSignal_freqPower - dsp_impl_outputSignal_freqPower);

%-------------------------------------------------------------------------------------------------------------
figure(1);

subplot(3,1,1);plot(time_interval,matlab_sim_input);axis([0 512 -.25 .25]);grid on;
title('Actual Input Signal in Time Domain with 7th Butterworth Matlab Simulation');
ylabel('Amplitude');

subplot(3,1,2);plot(time_interval,dsp_impl_inputSignal,'r');axis([0 512 -.25 .25]);grid on;
title('Actual Input Signal in Time Domain with 7th Butterworth DSP56307EVM Implementation');
ylabel('Amplitude');

subplot(3,1,3);plot(time_interval,matlab_sim_input-dsp_impl_inputSignal);axis([0 512 -.32e-6 .32e-6]);grid on;
title('Input: Amplitude difference in Time Domain Between Simulation and Implementation');
xlabel('Signal in Time Domain');ylabel('Amplitude');

%-------------------------------------------------------------------------------------------------------------

figure(2);

subplot(3,1,1);plot(time_interval,matlab_outputSignal);axis([0 512 -.15 .15]);grid on;
title('Actual Output Signal in Time Domain with 7th Butterworth Matlab Simulation');
ylabel('Amplitude');

subplot(3,1,2);plot(time_interval,dsp_impl_outputSignal,'r');axis([0 512 -.15 .15]);grid on;
title('Actual Output Signal in Time Domain with 7th Butterworth DSP56307EVM Implementation');
ylabel('Amplitude');

subplot(3,1,3);plot(time_interval,matlab_outputSignal'-dsp_impl_outputSignal);axis([0 512 -.32e-6 .32e-6]);grid on;
title('Output: Amplitude difference in Time Domain Between Simulation and Implementation');
xlabel('Signal in Time Domain');ylabel('Amplitude');

%-------------------------------------------------------------------------------------------------------------

figure(3);

subplot(2,1,1);plot(freqAxis,matlab_plotFFT_outputSignal(1025:2048)/Half_FFTsize,'r');
grid on;
title('The Spectrum: Actual Output Signal with 7th Butterworth Matlab Simulation');
xlabel('Frequency in Hz');ylabel('Amplitude');

subplot(2,1,2);plot(freqAxis,dsp_plotFFT_outputSignal(1025:2048)/Half_FFTsize);
grid on;
title('The Spectrum: Actual Output Signal with 7th Butterworth DSP56307EVM Implementation');
xlabel('Frequency in Hz');ylabel('Amplitude');

%-------------------------------------------------------------------------------------------------------------

figure(4);

subplot(2,1,1);plot(freqAxis,matlab_plotFFT_outputSignal(1025:2048)/Half_FFTsize,'r');
axis([0 8000 0 .004]);grid on;
title('Noise Level Spectrum: Actual Output Signal with 7th Butterworth Matlab Simulation');
xlabel('Frequency in Hz');ylabel('Amplitude');

subplot(2,1,2);plot(freqAxis,dsp_plotFFT_outputSignal(1025:2048)/Half_FFTsize);
axis([0 8000 0 .004]);grid on;
title('Noise Level Spectrum: Actual Output Signal with 7th Butterworth DSP56307EVM Implementation');
xlabel('Frequency in Hz');ylabel('Amplitude');

%-------------------------------------------------------------------------------------------------------------

figure(5);

subplot(2,1,1);plot(freqAxis,diff_realPart_FFT(1025:2048)/Half_FFTsize,'r');grid on;
title('The Output Spectrum Difference in real part between 7th Butterworth Simulation and Implementation');
xlabel('Frequency in Hz');ylabel('Amplitude');

subplot(2,1,2);plot(freqAxis,diff_imagPart_FFT(1025:2048)/Half_FFTsize);grid on;
title('The Output Spectrum Difference in imag part between 7th Butterworth Simulation and Implementation');
xlabel('Frequency in Hz');ylabel('Amplitude');

%-------------------------------------------------------------------------------------------------------------
```

```
figure(6);

subplot(2,2,1);plot(freqAxis(1:449),diff_realPart_FFT(1025:1473)/Half_FFTsize);
title('7th Butterworth output real part diff in passband');
xlabel('Frequency in Hz');ylabel('Amplitude');grid on;
subplot(2,2,3);plot(freqAxis(1:449),diff_imagPart_FFT(1025:1473)/Half_FFTsize);
title('7th Butterworth output imag part diff in passband');
xlabel('Frequency in Hz');ylabel('Amplitude');grid on;

subplot(2,2,2);plot(freqAxis(641:1024),diff_realPart_FFT(1665:2048)/Half_FFTsize,'r');
title('7th Butterworth output real part diff in stopband');
xlabel('Frequency in Hz');ylabel('Amplitude');grid on;
subplot(2,2,4);plot(freqAxis(641:1024),diff_imagPart_FFT(1665:2048)/Half_FFTsize,'r');
title('7th Butterworth output imag part diff in stopband');
xlabel('Frequency in Hz');ylabel('Amplitude');grid on;

%-------------------------------------------------------------------------------------------------------------

disp('Please choose one of options: track_power_data = 1, print_power_comp = 2');
which_loop = input('    my option = ');

if which_loop == 1

   disp('Comparison with Matlab simulation & DSP56307EVM implementation');
   disp('in terms of Parseval Realation');

   matlab_sim_timePower
   matlab_sim_freqPower

   dsp_impl_timePower
   dsp_impl_freqPower

   matlab_outputSignal_timePower
   matlab_outputSignal_freqPower

   dsp_impl_outputSignal_timePower
   dsp_impl_outputSignal_freqPower

   diff_outputSiganl_timePower
   diff_outputSignal_freqPower

elseif which_loop == 2

   iTitle_1 = 'print_sim_impl_power_comp_butter7.txt';
   fid_1 = fopen(iTitle_1,'wr');
   fprintf(fid_1, '\t  Comparison with Matlab simulation & DSP56307EVM implementation\n\n');
   fprintf(fid_1, '\t  by a 7th order Butterworth WD Filter in terms of Parseval Realation\n\n\n');

   fprintf(fid_1, '       Simulation in time domain: Input Signal Power = %10.14f\n',matlab_sim_timePower);
   fprintf(fid_1, '       Simulation in freq domain: Input Signal Power = %10.14f\n',matlab_sim_freqPower);

   fprintf(fid_1, '\n');

   fprintf(fid_1, '    Implementation in time domain: Input Signal Power = %10.14f\n',dsp_impl_timePower);
   fprintf(fid_1, '    Implementation in freq domain: Input Signal Power = %10.14f\n',dsp_impl_freqPower);

   fprintf(fid_1, '\n');

   fprintf(fid_1, '       Simulation in time domain: Output Signal Power =  %10.14f\n',...
                                    matlab_outputSignal_timePower);
   fprintf(fid_1, '       Simulation in freq domain: Output Signal Power =  %10.14f\n',...
                                    matlab_outputSignal_freqPower);

   fprintf(fid_1, '\n');

   fprintf(fid_1, '   Implementation in time domain: Output Signal Power =  %10.14f\n',...
       dsp_impl_outputSignal_timePower);
   fprintf(fid_1, '   Implementation in freq domain: Output Signal Power =  %10.14f\n',...
       dsp_impl_outputSignal_freqPower);

   fprintf(fid_1, '\n');

   fprintf(fid_1, '\t  The following is to compare power difference of output signal\n');
   fprintf(fid_1, '\t\t  between simulation and DSP implementation\n\n');
   fprintf(fid_1, '       Output Signal: Power Difference in time domain =  %10.14f\n',...
                                    diff_outputSignal_timePower);
   fprintf(fid_1, '       Output Signal: Power Difference in freq domain =  %10.14f\n',...
                                    diff_outputSignal_freqPower);

   fclose(fid_1);


else

   disp('Your option is out of range!');

end


if which_loop == 1
```

```
   disp('Comparison is done. Have a nice day!');

elseif which_loop == 2

   diary off;
   disp('All data are saved at the file, called <print_sim_impl_power_comp_butter7.dat>');

else

   disp('Have a nice day!');

end
```

7. wdfilter.m

```
function  output = wdfilter(FFTsize,input)

%The 7th Butterworth WD Lattice Adaptor
a=zeros(14,1);
b=zeros(14,1);

% Coefficients for Satisfying the Specifications
% fp=3.4 kHz, ap=0.5 dB, fs=6.0 kHz, as=55.0 dB, F=16 kHz
%The following coefficients are related to 24 bits

r0  =    0.011606;
r1  =   -0.052229;
r2  =    0.023209;
r3  =   -0.232042;
r4  =    0.023209;
r5  =   -0.636044;
r6  =    0.023209;

%make as input signal for the purposes of the impulse
%ImpulseInput(1)=1;%ImpulseInput(2:FFTsize)=zeros(FFTsize-1,1);
%input(1)=1;input(2:FFTsize)=zeros(FFTsize-1,1);

for n=1:FFTsize

% Goes to Upper Section

a(1) = input(n);


% Calls the function created

[b(1),b(2)]=Two_adap(r0,a(1),a(2));

% Next State

a(2)=b(2);

% Connection Constraint

a(7)=b(1);

[b(9),b(10)]=Two_adap(r4,a(9),a(10));
a(10)=b(10);
a(8)=b(9);

% Ends Upper Section

[b(7),b(8)]=Two_adap(r3,a(7),a(8));

% Goes to Lower Section

a(3)=a(1);

[b(5),b(6)]=Two_adap(r2,a(5),a(6));

a(6)=b(6);
a(4)=b(5);

[b(3),b(4)]=Two_adap(r1,a(3),a(4));

a(11)=b(3);

[b(13),b(14)]=Two_adap(r6,a(13),a(14));
a(14)=b(14);
a(12)=b(13);

% Ends Lower Section

[b(11),b(12)]=Two_adap(r5,a(11),a(12));

% Output as Response

output(n)=(b(7)+b(11))/2;

% Next State
```

```
a(5)=b(4);
a(13)=b(12);
a(9)=b(8);

end
```