31455 9778

Digital Filter Design Package (DFP) Extension

by

Hongbo Sun

A Project
Presented to Ryerson University
in partial fulfillment of the
requirements for the degree of

Master of Engineering

in the department of
Electrical and Computer Engineering

Toronto, Ontario, Canada, 2003

Author's Declaration:

I hereby declare that I am the sole author of this Research Paper.

I authorize Ryerson University to lend this Research Paper to other institutions or individuals for the purpose of scholarly research.

I further authorize Ryerson University to reproduce this Research Paper 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.

Borrow List

Digital Filter Package (DFP) Extension

Abstract

In this study, we present an integrated approach to the design of digital filters using the Digital Filter Package (DFP). We added four new features to DFP: multi-band filter design, adaptive filter design, two-dimensional filter design and TMS320 code generation. We discuss each of these new features from design consideration to realization. We present design examples and demonstrate the design results generated by DFP. We also discuss GUI design in MATLAB, the DFP GUI design structure and the data transition structure from one DFP function to another. We show that DFP can allow efficient design of the following filter structures: low pass filter, high pass filter, band pass filter, band stop filter, multi band filter, differentiator, Hilbert Transformer, adaptive and 2D filters. DFP allows the use of a variety of FIR and IIR design methods as well as hardware implementation structures; it provides a useful tool to quickly examine a variety of filters and understand the tradeoffs involved in varying the characteristics of the filter.

Keywords: digital signal processing, DFP, adaptive filter design, two-dimensional digital filter design, multi-band digital filter design, TMS320 code generation

Acknowledgement

I would like to thank Mehmet Zeytinoglu, my supervisor, for his patient support and advice.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In many applications, it is desirable that the frequency spectrum of a signal to be modified, reshaped, or manipulated according to a desired specification. The process may include attenuating a range of frequency components and rejecting or isolating one specific frequency component. Any system or network that exhibits such frequency-selective characteristics is called a filter. Several types of filters can be identified: low-pass filter (LPF) that passes only low frequencies, high-pass filter (HPF) that passes high frequencies, band-pass filter (BPF) that passes a band of frequencies, and band-stop filter (BSF) that rejects certain frequencies. Filters are used in a variety of applications, such as removing noise from a signal, removing signal distortion due to the transmission channel, separating two or more distinct signals that were mixed in order to maximize communication channel utilization, demodulating signals, and converting discrete-time signal into continuous-time signals.

Digital filters are used extensively in applications, such as digital image processing, pattern recognition, and spectrum analysis. [1]

In many signal- processing applications, it is advantageous to use digital filters in place of analog filters. Digital filters can meet tight specifications on magnitude and phase characteristics and eliminate voltage drift, temperature drift, and noise problems associated with analog filter components. Another important advantage of digital filters when implemented with a programmable processor is the ease of changing filter parameters to modify the filter characteristics. This feature allows the design engineer to effectively and easily update the characteristics of the designed filter due to changes in the application environment.

Quantization is a natural outgrowth of digital filtering and digital signal processing development. Also, there is a growing need for fixed-point filters that meet power, cost, and size restrictions. In applications where power limitations and size constraints drive the filter design, we use fixed-point filters. When we convert filter coefficients from floating-point to fixed-point, we use quantization to perform the conversion [2].

The Digital Filter Package (DFP) [27] is a user- friendly GUI front-end filter design toolbox implemented in the MATLAB environment. In its original implementation it allows the design of LPF, HPF, BPF, BSF, Differentiator and Hilbert Transform filters. Filter design methods include FIR (Remez equiripple FIR design, windowed linear phase FIR digital filter design and least squares linear phase FIR filter design) and IIR (Butterworth, Chebychev-I, Chebyshev-II and Elliptic design). DFP also extends the basic digital filter design functionality of MATLAB in two important ways: (1) The filter coefficients can be quantized. This feature is of particular importance if the filter will eventually be implemented on a fixed point DSP. (2) DFP can also generate assembler code for the Motorola DSP56k family of fixed-point processors.

The desirable features of an adaptive filter, namely, the ability to operate satisfactorily in an unknown environment and also track time variations of input statistics, make the adaptive filter a powerful device for signal processing and control applications [3]. By integrating adaptive filter design methods into DFP, the designer can use the DFP graphic user interface to design adaptive filter with Least Mean Square (LMS) algorithm and Recursive Least Square (RLS) algorithm instead of trying to design these complicated algorithms by themselves. Two- dimensional digital filters are also widely used in many fields. Therefore, extending DFP to include 2D filter design capability makes DFP stand out from other digital filter design packages by providing an easy way to design 2D filters. We also enhanced the digital filter design ability of DFP by adding multi-band filter design capability and code generation feature for the TMS320 family of digital signal processors.

The design of digital filters involves the following steps: approximation, realization, study of arithmetic errors, and hardware implementation steps. Approximation is the process of generating a transfer function that satisfies a set of desired specifications, which may involve the time-domain response, frequency-domain response, or some combination of both responses of the filter. Realization consists of the conversion of the desired transfer function into a filter network. Approximation and realization assume an infinite-precision device for implementation. However, implementation is concerned with the actual hardware or software coding of the filter using a programmable processor. Since many practical devices are of finite precision, it is necessary to study the effects of arithmetic errors on the filter response.

The organization of this project report is as follows. Chapter 2 introduces digital filter design principles including specification analysis, approximation methods, realization structures, and implementation in hardware and software. Chapter 3 illustrates multi-band filter design results. In Chapter 4 we present the application of adaptive filters and adaptive filter realization algorithm (LMS and RLS) design procedures, then introduce how to design an adaptive filter using the DFP. Chapter 5 presents the separable two- dimensional digital filter design theory and design examples. We also discuss how to design multi-band two-dimensional digital filters. Chapter 6 addresses DFP GUI design and TMS320 code generation. Chapter 7 summarizes the contributions of this project and discusses future directions of research.

Chapter 2

Digital Filter Design Principles

The design of a digital filter involves the following five steps:

1. Specification of the filter requirements.
2. Selection of the approximation methods.
3. Realization of the filter by a suitable structure.
4. Analysis of finite wordlength effects on the filter performance.
5. Implementation of filter in software and/or hardware.

2.1 Digital Filter Specifications

The frequency responses of the four popular types of ideal digital filters with real impulse response coefficients are shown in Figure 2.1. All frequency values in this figure are normalized frequency values, i.e., suppose f_s is defined as sampling frequency, then the normalized upper frequency limit $f_s/2$ becomes 0.5. The frequencies f_c, f_{c1}, f_{c2} are called the cutoff frequencies of their respective filters. An ideal filter has a magnitude response equal to unity in the pass-band and zero in the stop-band everywhere [1]. Table 2.1 shows the pass-band and stop-band region for LPF, HPF, BPF and BSF.

	Pass-band Region	Stop-band Region
LPF	$0 \le \| f \| \le f_c$	$f_c \le \| f \| \le 0.5$
HPF	$f_c \le \| f \| \le 0.5$	$0 \le \| f \| \le f_c$
BPF	$f_{c1} \le \| f \| \le f_{c2}$	$0 \le \| f \| \le f_{c1}$ and $f_{c2} \le \| f \| \le 0.5$
BSF	$0 \le \| f \| \le f_{c1}$ and $f_{c2} \le \| f \| \le 0.5$	$f_{c1} \le \| f \| \le f_{c2}$

Table 2.1: Pass-band and Stop-band Region for LPF, HPF, BPF, and BSF

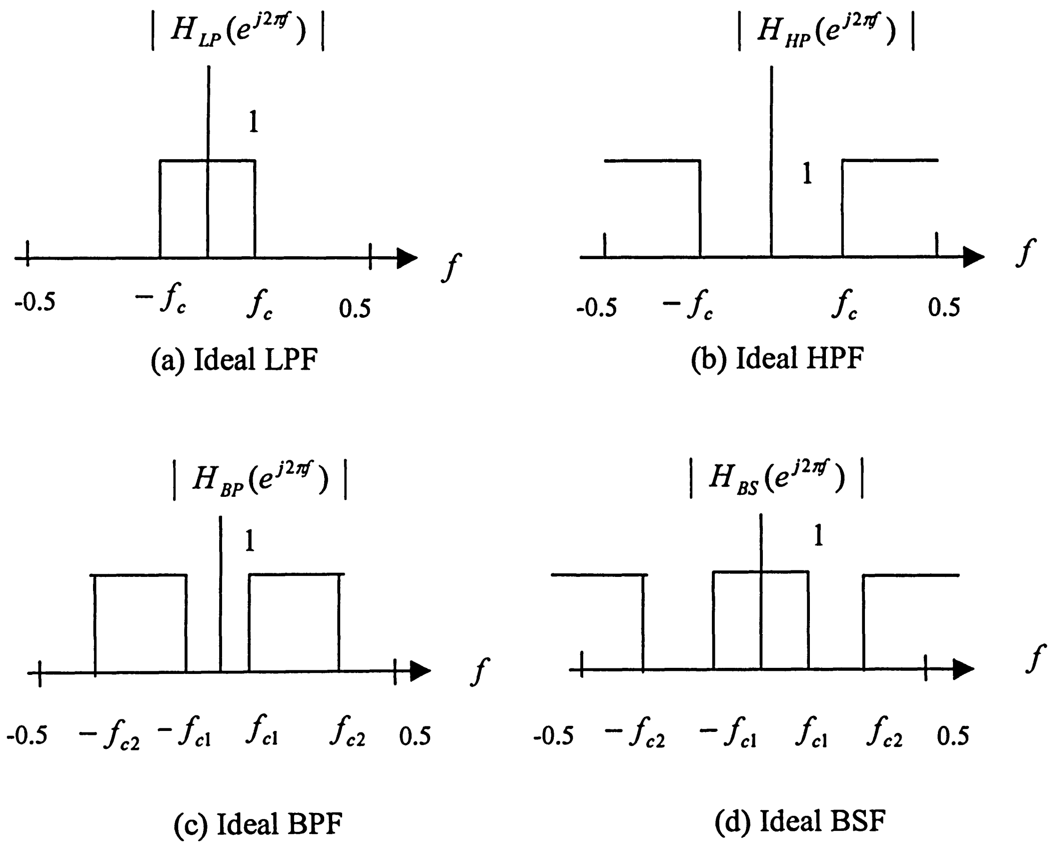


Figure 2.1: Four Types of Ideal Filters: LPF, HPF, BPF, and BSF

In practice, it is impossible to realize a finite dimensional Linear Time Invariant (LTI) filter with the ideal "brick wall" characteristics of Figure 2.1 as the corresponding impulse response is not causal and is of doubly infinite length. Moreover, the impulse response is not absolutely summable, and hence, the corresponding transfer function is not Bounded-Input, Bounded-Output (BIBO) stable.

In order to develop stable and realizable transfer functions, the ideal frequency response specifications of Figure 2.1 are relaxed by including a transition band between the pass-band and the stop-band to permit the magnitude response to decay more gradually from its maximum value in the pass-band to the zero value in the stop-band. Moreover, the magnitude response is allowed to vary by a small amount both in the pass-band and the stop-band. Typical magnitude specifications used for the design of LPF, HPF, BPF, and BSF are shown in Figure 2.2. Note the magnitude response used in this figure is logarithm.

The cutoff frequencies are listed in Table 2.2. α and β are pass-band and stop-band attenuation in dB. Max and Min are maximum and minimum target region limit in dB.

	Pass-band Cutoff Frequency	Stop-band Cutoff Frequency
LPF	f_1	f_2
HPF	f_4	f_3
BPF	Lower cutoff frequency is f_6 Upper cutoff frequency is f_7	1st: Upper cutoff frequency is f_5 2nd: Lower cutoff frequency is f_8
BSF	1st: Upper cutoff frequency is f_9 2nd: Lower cutoff frequency is f_{12}	Lower cutoff frequency is f_{10} Upper cutoff frequency is f_{11}

Table 2.2: Pass-band and Stop-band Region for LPF, HPF, BPF and BSF

For multi-band digital filter specifications and target regions, we can extend the specifications and target regions of BPF and BSF. Most multi-band filter design techniques depend on the specification of these parameters: pass-band and stop-band cutoff frequencies, pass-band and stop-band attenuation factors.

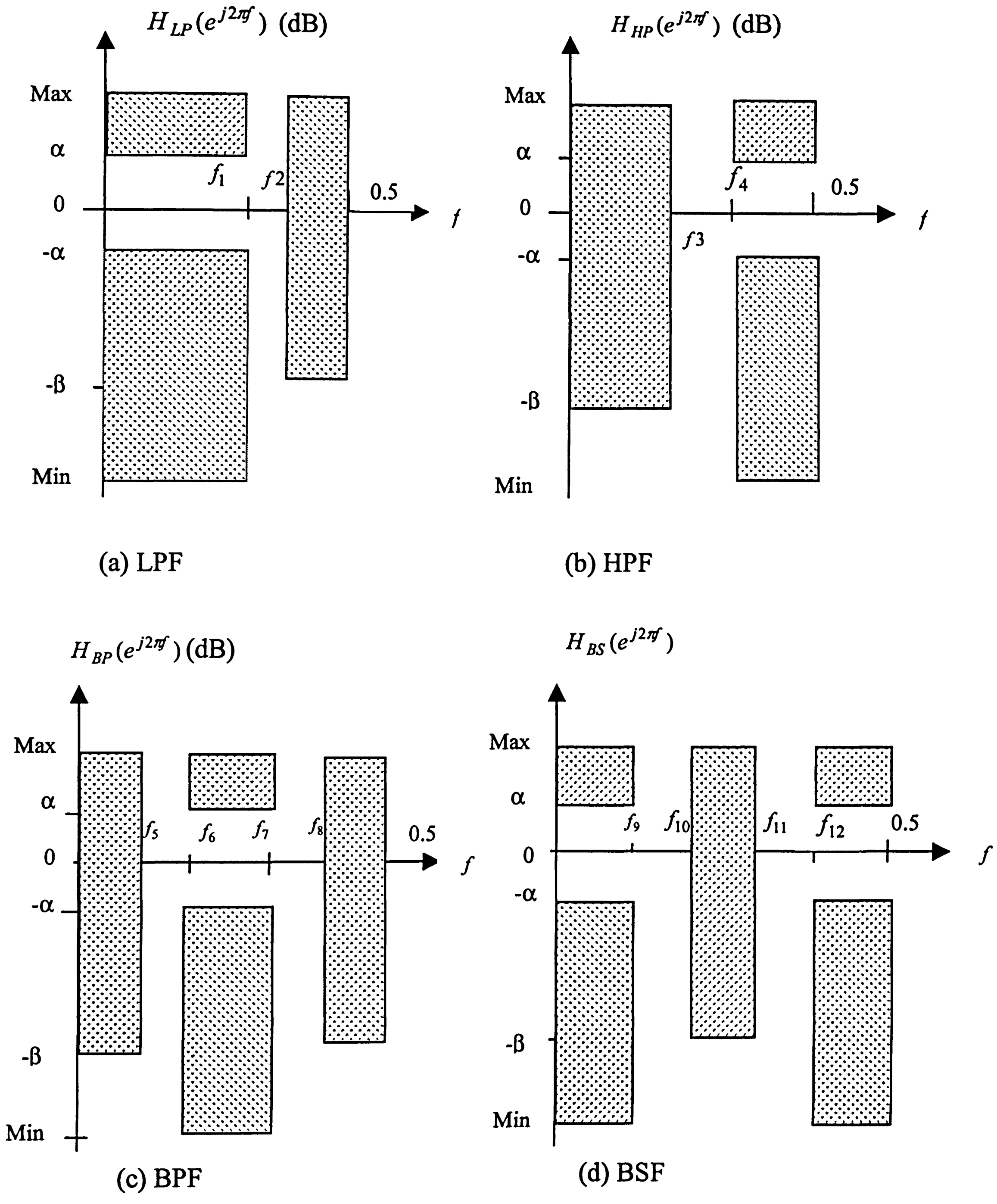


Figure 2.2: Target Regions of LPF, HPF, BPF, and BSF

2.2 Digital Filter Approximation Method

Digital filter design requires the use of both frequency domain and time domain techniques. This is because filter design specifications are often given in the frequency domain, but filters are usually implemented in the time domain. Typically, frequency domain analysis is done using the Z-transform and the discrete-time Fourier Transform (DTFT).

In general, a linear and time-invariant digital filter with input $x(n)$ and output $y(n)$ may be specified by its difference equation [4].

$$y(n)=\sum_{i=0}^{M} b_i x(n-i)+\sum_{k=1}^{N} a_k y(n-k) \tag{2.1}$$

where b_i and a_k are coefficients which parameterize the filter. This filter is said to have M zeros and N poles. Each new value of the output signal, *y (n)*, is determined by past values of the output, and by the present and past values of the input. There are two general classes of digital filters: infinite impulse response (IIR) and finite impulse response (FIR). The FIR case occurs when a_k=0, for all *k*. Such a filter is said to have no poles, only zeros. In the case where $a_k \neq 0$, the difference equation usually represents an IIR filter.

There are several FIR and IIR filter approximation methods. The most frequently used approximation methods for FIR filter design are the Remez exchange approximation method (using the Parks McClellan algorithm), the windowing method and the least-mean-square approximation method [2]. Approximation methods for IIR filter design include the widely used transformation of analogue filters design methods such as the Butterworth approximation method, the Chebyshev approximation method, and the elliptic approximation method. Solving the approximation problem using IIR filters require smaller order than using FIR filters. The penalty is non-linear phase response and potential instability.

2.3 Realization Structure of FIR and IIR Filters

The Z-transform is the major tool used for analyzing frequency response of filters. The realization of the filter by a suitable structure corresponding to a selected finite length representation, is the next step before implementation. For hardware realization, the most commonly used structures are the direct, cascade and parallel forms. For FIR filters the most widely used structure is the direct form. Figure 2.3 (1) shows the transversal or direct form structure realization [18]. The *Z*-transform of FIR non-recursive filter is shown as equation (2.3) where *X(z)* and *Y(z)* are the *Z* transforms of *x(n)* and *y(n)* [5].

$$^{(1)}Y(z) = \sum_{i=0}^{M} b(i) z^{-i} X(z) \tag{2.3}$$

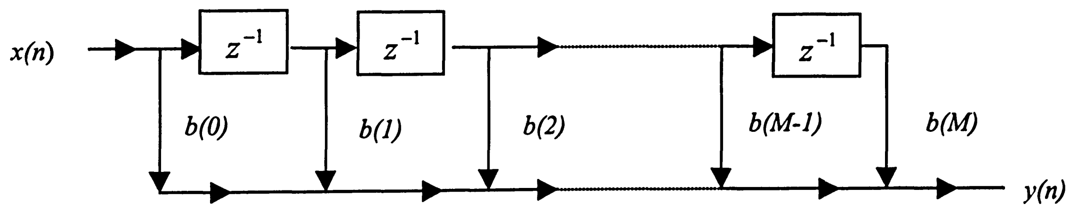


Figure 2.3: Direct Form FIR Filter Structure

Another FIR filter structure is cascade structure. The cascade structure converts the transfer function into a product of second-order functions, as shown in equation (2.4) [17]:

$$H(z) = b_0 \prod_{k=1}^{N/2} (1 + b_{k,1} z^{-1} + b_{k,2} z^{-2}) \tag{2.4}$$

This leads to the cascade form shown in Figure 2.4 [18].

(1) In the figures of this report we use the convention that two branches merging at a node represents that the node output is the sum of the branch signal.

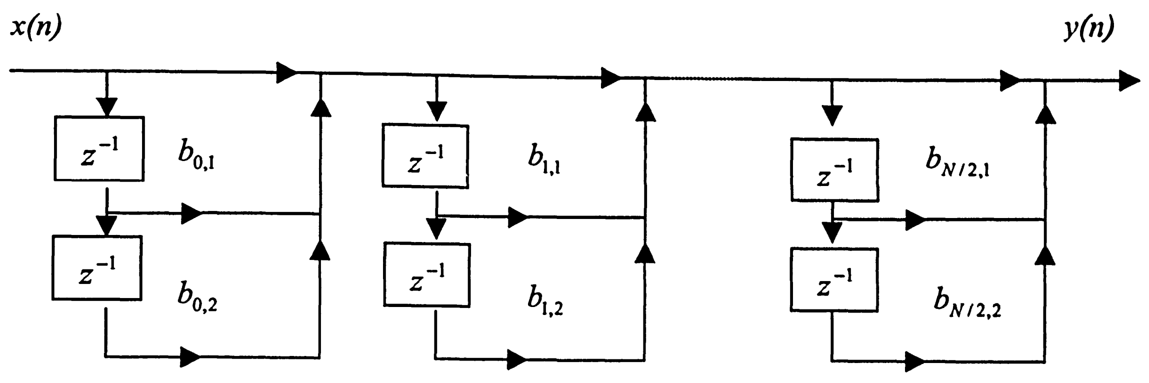


Figure 2.4: FIR Filter Cascade Structure

There are many different methods for implementing recursive IIR filters. Equation (2.5) shows the Z-transform of the impulse response of an IIR filter, where *H(z), Y(z),* and *X(z)* are the Z-transforms of *h(n), y(n),* and *x(n),* respectively.

$$H(z) = \frac{Y(z)}{X(z)} = \frac{\sum_{i=0}^{M} b_i z^{-i}}{1 - \sum_{k=1}^{N} a_k z^{-k}} \tag{2.5}$$

Three different structures often used to implement digital filters are the direct form (I and II), the cascade form and the lattice form. Figure 2.5 and Figure 2.6 show the system diagrams known as direct form I and direct form II implementation, respectively [18]. Direct form II structure is also called Canonical form. Note that the left half of Figure 2.5 implements the numerator (zeros) of *H(z),* while the right half implements the denominator (poles) of *H(z).* When the order of the numerator and the order of the denominator are the same *(N=M),* the delay lines can be combined in a single one as shown in Figure 2.6. The two forms require the same number of arithmetic operations, but the direct form II can require as few as half the number of memory registers for storing the past values of the inputs and outputs. Although direct form I and direct form II have the same transfer function *H(z)*, the corresponding difference equations are not the same. The difference equation of the direct form I structure is as equation (2.6), while that of direct form II is as equation (2.7).

$$y(n) = \sum_{i=0}^{M} b_i x(n-i) + \sum_{k=1}^{N} a_k y(n-k) \tag{2.6}$$

$$d(n) = \sum_{k=1}^{N} a_k d(n-k) + x(n)\,;\; y(n) = \sum_{i=0}^{M} b_i d(n-i) \tag{2.7}$$

The implementation of a cascade-form IIR filter is an extension of the results of the implementation of the direct-form IIR filter. The Z-transform equation of the impulse response of an IIR filter (2.5) may be written in the equivalent form (2.8) where the filter is realized as a biquads [5].

$$H(z) = \prod_{k=1}^{N/2} \frac{\beta_{0k} + \beta_{1k} z^{-1} + \beta_{2k} z^{-2}}{1 - \alpha_{1k} z^{-1} - \alpha_{2k} z^{-1}} \tag{2.8}$$

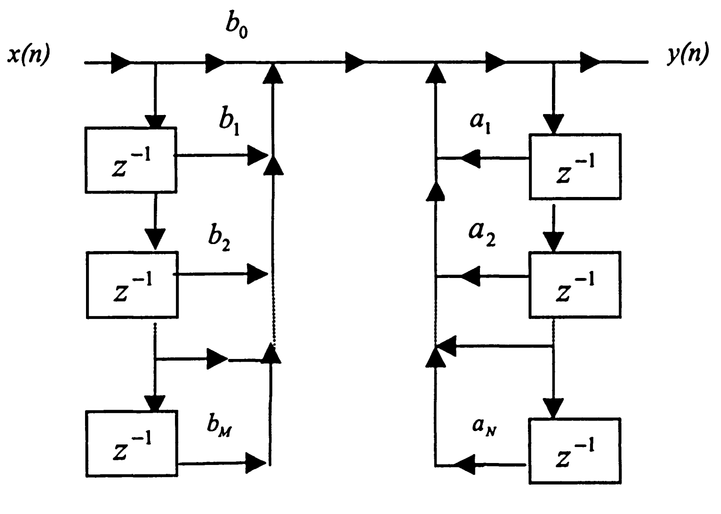


Figure 2.5: Direct Form I Structure of IIR filter

Therefore, this realization is referred to as the cascade form. Figure 2.7 shows the IIR filter implemented in cascade structure [18], where each sub-block corresponds to one of the terms in the product (2.8). Note that any single cascade section is identical to the second-order direct-form II IIR filter.

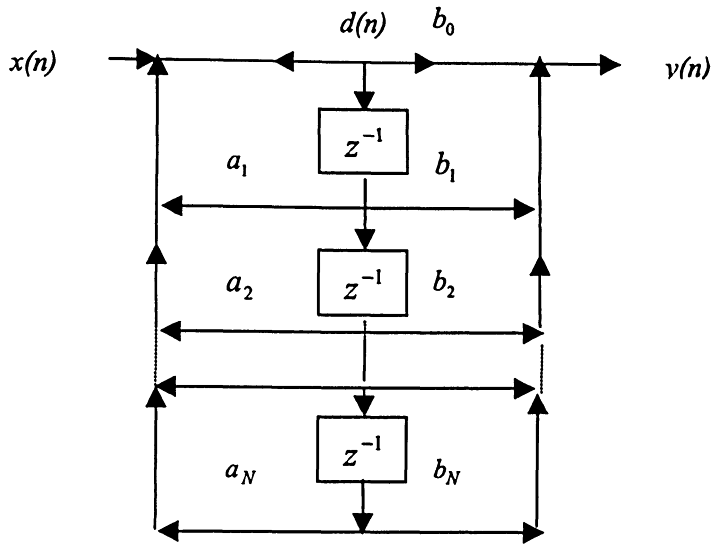


Figure 2.6: Direct Form II Structure of IIR filter

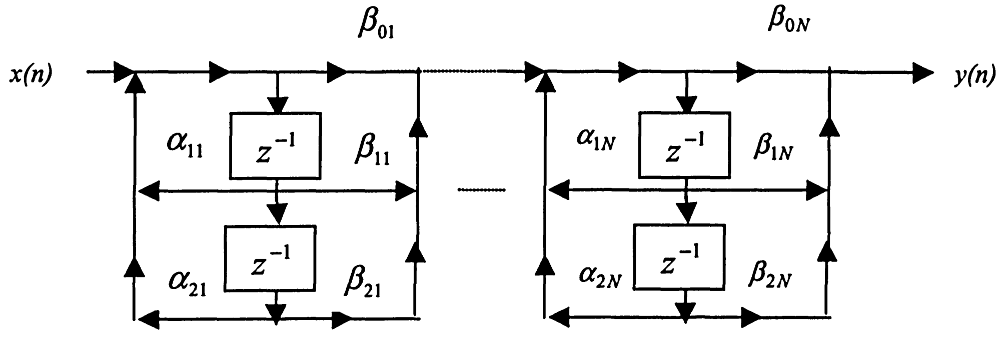


Figure 2.7: Cascade Second Order Structure of IIR filter

In the direct form realization shown in Figure 2.6, the variation of one parameter will affect the locations of all the poles of *H(z)*. In a cascade realization shown in Figure 2.7, the variation of one parameter will affect only poles in that section. Therefore the cascade realization is less sensitive to parameter variation (due to coefficient quantization, etc.) than the direct form structure. In practical implementations of digital IIR filters, the cascade form is preferred.

Equation (2.9) displays the relations between f(n) and g(n) for the m^{th} stage of an FIR lattice filter, where *Km, f(n)* and *g(n)* represent the reflection coefficient, the forward and backward prediction residuals respectively.

$$\begin{aligned} f_m(n) &= f_{m-1}(n) + K_m g_{m-1}(n-1), \qquad & m = 1,2,3,..M-1 \\ g_m(n) &= K_m f_{m-1}(n) + g_{m-1}(n-1), \qquad & m = 1,2,3..M-1 \end{aligned} \tag{2.9}$$

Figure 2.8 shows the basic section of an FIR lattice filter.

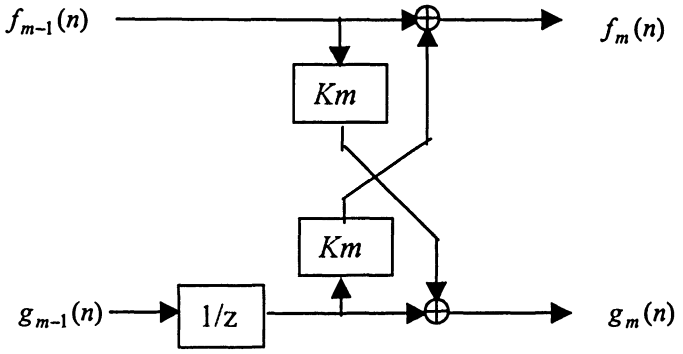


Figure 2.8: The basic section of a FIR lattice filter

The general form of an IIR lattice filter is shown in Figure 2.9.

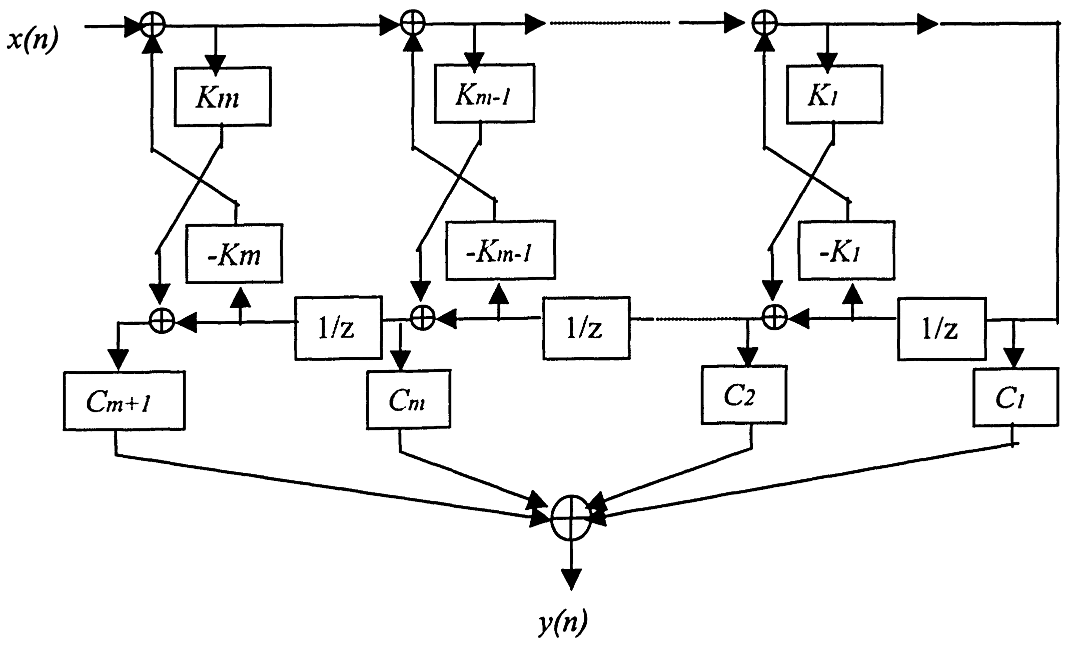


Figure 2.9: The general form of the IIR lattice filter

The K_m and C_m coefficients can be obtained from a direct form filter by using MATLAB function: [K,C] = tf2latc(B,A) where B, A are direct form coefficients . Lattice structure is less sensitive to parameter variation and has the advantage of easily expending the filter order without fully re-design the lattice filter.

2.4 Implementation Considerations

The cascade form is most often employed in practical applications for reasons concerning quantization effects and DSP implementation. There are four types of quantization effects in digital filters - input quantization, coefficient quantization, roundoff errors, and overflow [4].

Representing instantaneous values of a continuous-time signal in digital form introduces errors that are associated with I/O quantization. Input signals are subject to A/D quantization noise while output signals are subject to D/A quantization noise. The input A/D quantization noise is the more dominant factor due to the fact that input noise circulates within IIR filters and can be regenerative while output noise normally just propagates off-stage.

Digital filters are designed with the assumption that the filter will be implemented on an infinite precision device. However, since all processors are of finite precision, it is necessary to approximate the ideal filter coefficients. This approximation introduces coefficient quantization error. The effect of coefficient quantization is highly dependent on the structure of the filter and the wordlength of the implementation hardware. Since the poles and zeros of a filter implemented with finite wordlength arithmetic are not necessarily the same as the poles and zeros of a filter implemented on an infinite precision device, the difference may affect the performance of the filter.

Truncation or rounding off the products formed within the digital filter is referred to as correlated roundoff noise. The result of correlated roundoff noise, including overflow oscillations, is that filters suffer from “limit-cycle effect”. For system with adequate coefficient wordlength and dynamic range, this problem is usually negligible. Overflows are generated by additions resulting in undesirable large amplitude oscillations. Both limit cycles and overflow oscillations force the digital filter into nonlinear operations. Although limit cycles are difficult to eliminate, saturation arithmetic can be used to reduce overflow oscillations. The overflow mode of operation on the TMS320 family is accomplished with set overflow mode instruction, which sets the accumulator to the largest representable 32-bit positive or negative number according to the direction of overflow.

2.5 DFP Digital Filter Design Capability

DFP is developed using Graphic User Interface (GUI) design technique. It integrates all steps needed to design a digital filter, i.e., defining filter specifications (different filter type corresponding to different specifications), choosing approximation methods (FIR and IIR), selecting realization structures (Direct form, Cascade Second-Order Section, Lattice), DFP quantization consideration (fixed point or floating point) and generating DSP assembly code working with the Motorola 56000, TMS32010/TMS32020, TMS320C54x, TMS320C67x DSP families. The DFP GUI structure is shown in Figure 2.10.

In Figure 2.8, the IMPLEMENTATION branch is for selecting realization structures (Direct form, Cascade Second-Order Section form, Lattice form), while the OUTPUT FORMAT branch is to generate assembly code for Motorola56000, TMS32010/TMS32020, TMS320C54x, TMS320C67x series of DSP processors.

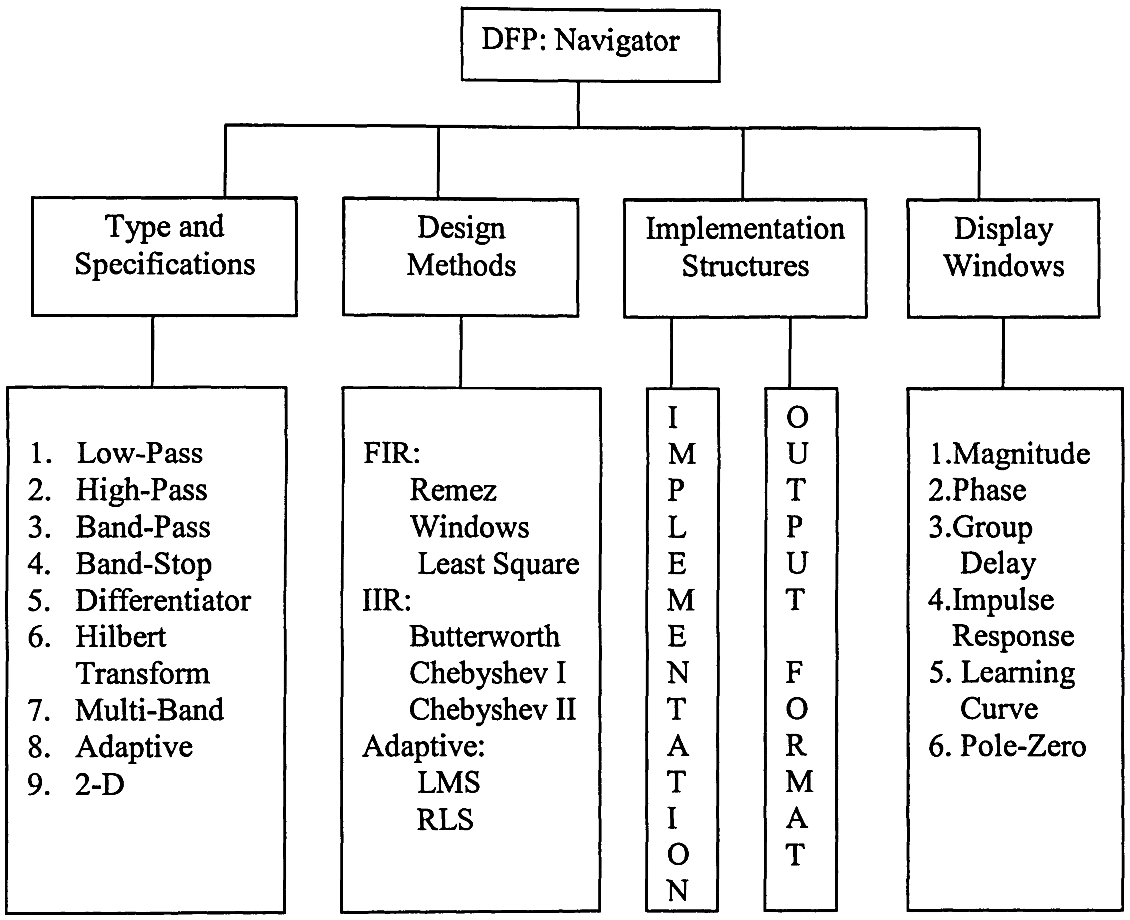


Figure 2.10: DFP GUI Structure

Chapter 3

Multi-Band Filter Design

The goal of adding multi-band filter design to DFP is to increase the band design ability to more than three. A multi-band filter has the combined functions of LPF, HPF, BPF, BSF and exhibits more than three filter bands. In multi-band filter design, we consider the following questions:

- How to define multi-band digital filter specifications?
- How to design target region to meet the required specifications?
- How are the data information passed from one window to the other?

3.1 Filter Specifications

Table 3.1 and Table 3.2 list the specifications needed to enter DFP for designing two bands digital filters such as LPF and HPF and three bands digital filters such as BPF and BSF.

LPF	HPF
Sampling Frequency	Sampling Frequency
Pass-band Frequency Edge	Stop-band Frequency Edge
Pass-band Attenuation (dB)	Stop-band Attenuation (dB)
Stop-band Frequency Edge	Pass-band Frequency Edge
Stop-band Attenuation (dB)	Pass-band Attenuation (dB)

Table 3.1: Specifications for LPF and HPF

BPF	BSF
Sampling Frequency	Sampling Frequency
First Stop-band Frequency Edge	First Pass-band Frequency Edge
First Stop-band Attenuation (dB)	First Pass-band Attenuation (dB)
First Pass-band Frequency Edge	First Stop-band Frequency Edge
Pass-band Attenuation (dB)	Stop-band Attenuation (dB)
Second Pass-band Frequency Edge	Second Stop-band Frequency Edge
Second Stop-band Frequency Edge	Second Pass-band Frequency Edge
Second Stop-band Attenuation (dB)	Second Pass-band Attenuation (dB)

Table 3.2: Specifications for BPF and BSF

By analyzing Tables 3.1 and 3.2, we observe the need to increase the number of design parameters if we want to extend the design capability of DFP to beyond 2 filter bands. We consider a vector-input idea so that design specifications can be entered into the DFP workspace in a more efficient way. In the multi-band *Type and Specification Window*, we specify the *frequency-edge*, the *weight* and the *attenuation* vectors as the multi-band filter design parameters.

In DFP we support three input modes: the normalized frequency input, the frequency input in Hz and strings. If we choose to use strings to specify the frequency parameter, this string has to be defined in the MATLAB workspace.

3.2 Target Region Design

The objective of designing the target region is to make DFP possess self-test ability. Original DFP has designed the target regions for LPF, HPF, BPF and BSF, Differentiator, and Hilbert Transform filter. Using filter specification parameters, i.e., frequency edge vector (fre_vector), weight vector (wgt_vector), and attenuation vector (dev_vector), we can determine the target region coordinate matrix and plot the corresponding target region in DFP.

There are two possibilities for the design of multi-band target region: the first band is pass-band or stop-band. Figure 3.1 shows the program flow chart to design multi-band target region. Figure 3.2 shows the coordinate diagram to design target region when the first band is pass-band. When the first band is stop-band, the target region coordinate diagram is shown in Figure 3.3. In these two figures, we use a,b,c,d to determine each rectangular target region. *Ymax* and *Ymin* represent maximum and minimum target region limit values. A_1 and A_2 represent the pass-band and the stop-band attenuation values in Figure 3.2, and the stop-band and the pass-band attenuation values in Figure 3.3. Figure 3.4 shows part of design code in MATLAB when the first band is pass-band as shown in Figure 3.2. Table 3.3 summarizes the specification vectors corresponding to Figure 3.2 and Figure 3.3.

	The First Band is Pass-band	The First Band is Stop-band
fre_vector	$[f_1\ f_{21}\ f_{22}\ f_{31}\ f_{32}\ \ldots.]$	$[f_1\ f_{21}\ f_{22}\ f_{31}\ f_{32}\ \ldots.]$
wgt_vector	[1 0 1 0 1 ....]	[0 1 0 1 ....]
dev_vector	$[A_1\ A_2\ A_1\ A_2\ A_1\ \ldots.]$	$[A_1\ A_2\ A_1\ A_2\ A_1\ \ldots.]$

Table 3.3: Specification Vectors for Multi-band Filter

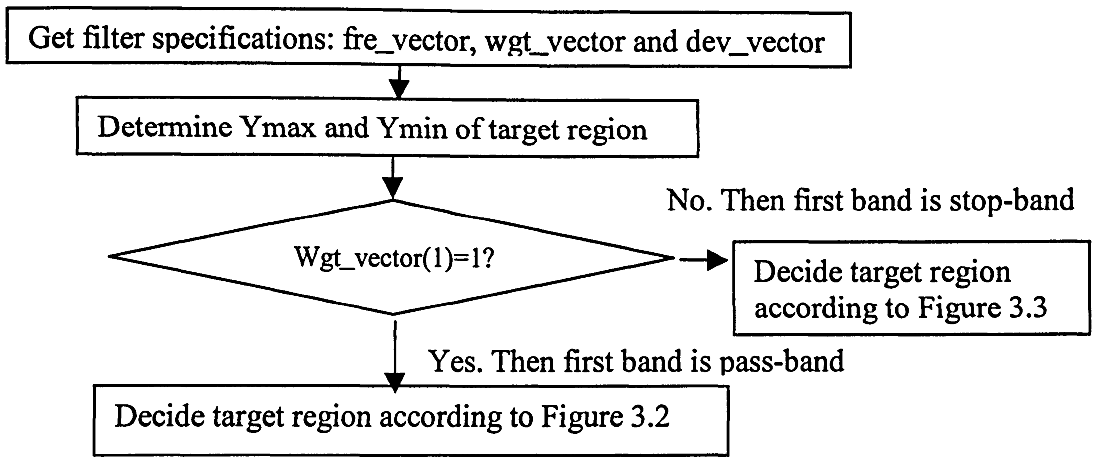

Get filter specifications: fre_vector, wgt_vector and dev_vector
Determine Ymax and Ymin of target region
Wgt_vector(1)=1?
No. Then first band is stop-band
Decide target region according to Figure 3.3
Yes. Then first band is pass-band
Decide target region according to Figure 3.2

Figure 3.1: Flow chart of target region design program

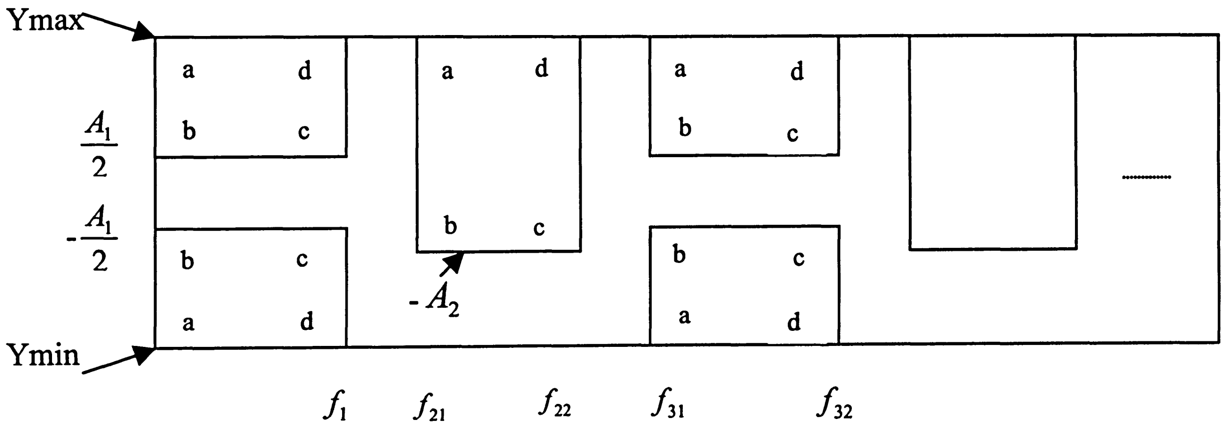


Figure 3.2: Target Region if the First Band is a Pass-band.

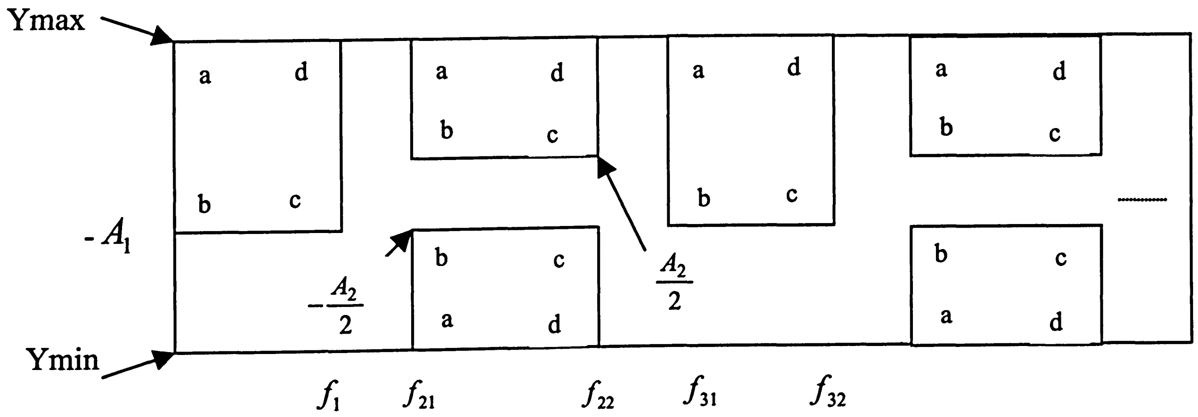


Figure 3.3: Target Region if the First Band is a Stop-band

```
elseif ( FILTtypeNumber == 7 ) % Multi-band Case

    % Determine Ymin
   Ymin1 = -fix(abs(-dev_vector(1) + DFP_Targetmin)/10)*10;
   for i = 1: length(dev_vector)
      dev_vector1(i) = -fix(abs(-dev_vector(i)+DFP_Targetmin)/10)*10;
      Ymin1 = min (Ymin1, dev_vector1(i));
   end
   Ymin = Ymin1;

    % Determine Ymax

   Ymax1 = fix(abs(dev_vector(1) + DFP_Targetmax)/10)*10 + dev_vector(1)/2;
   for i = 1: length(dev_vector)
      dev_vector1(i) = fix(abs(dev_vector(i)+DFP_Targetmax)/10)*10 + dev_vector(i)/2;
      Ymax1 = max( Ymax1, dev_vector1(i));
   end
   Ymax = Ymax1;

  if (wgt_vector(1) == 1)            % This is the case of first band is pass band%

   for i = 1:length(dev_vector)
      bandn = length(dev_vector);   % How many band it has? %

     if (rem(bandn,2)==1)
        ynumber = (3*bandn +1)/2; % number of column in x vector (x coordinate)

   % Initial matrix  ( x and y coordinate)
      x1 = zeros(1,ynumber); % a
      x2 = zeros(1,ynumber); % b
      x3 = zeros(1,ynumber); % c
      x4 = zeros(1,ynumber); % d
      x  = zeros(4,ynumber);           % x coordinate matrix of target region

      yLOG1 = zeros(1,ynumber);
      yLOG2 = zeros(1,ynumber);
      yLOG3 = zeros(1,ynumber);
      yLOG4 = zeros(1,ynumber);
      yyLOG = zeros(4,ynumber);       % y coordinate of target region

    for i = 3:ynumber
      if ( rem(i,3)==0 )
           k = i/3*2;
           yLOG1(i) = Ymax;
           yLOG4(i) = Ymax;
           yLOG2(i) = -dev_vector(k);
           yLOG3(i) = -dev_vector(k);
      elseif (rem(i,3) ==1)
           k = (((i+2)/3)*2)-1;
           yLOG1(i) = Ymax;
           if ( i==(ynumber-1) )
              yLOG4(ynumber-1) = dev_vector(bandn)/2;
           else
```

```
            yLOG4(i) = Ymax;
         end
          yLOG2(i) = dev_vector(k)/2;
          yLOG3(i) = dev_vector(k)/2;

    elseif (rem(i,3) ==2)
         k = (((i+1)/3)*2)-1;
         yLOG1(i) = Ymin;

         if ( i == ynumber )
            yLOG4(ynumber) = -dev_vector(bandn)/2;
         else
            yLOG4(i) = Ymin;
         end
     end

      x1(i) = fre_vector(2*(k-2)+2);
     x2(i) = fre_vector(2*(k-2)+2);
     if ( ( i == ynumber )|( i == (ynumber-1) ))
        x3(i) = fs/2;x4(i) = fs/2;
     else
       x3(i) = fre_vector(2*(k-2)+3);
       x4(i) = fre_vector(2*(k-2)+3);
     end

     x1(1) = 0;
     x1(2) = 0;
     x2(1) = fre_vector(1);
     x2(2) = fre_vector(1);
     x3(1) = fre_vector(1);
     x3(2) = fre_vector(1);
     x4(1) = fre_vector(1);
     x4(2) = fre_vector(1);

     x = [x1;x2;x3;x4];                          % x coordinate of target region

     yLOG1(1) = dev_vector(1)/2;
     yLOG1(2) = -dev_vector(1)/2;
     yLOG2(1) = dev_vector(1)/2;
     yLOG2(2) = -dev_vector(1)/2;
     yLOG3(1) = Ymax;
     yLOG3(2) = Ymin;
     yLOG4(1) = Ymax;
     yLOG4(2) = Ymin;

     yyLOG= [yLOG1;yLOG2;yLOG3;yLOG4];    % y coordinate of target region
     yLOG = yyLOG;
end
```

Figure 3.4: Part of Target Region Design Code

3.3 Multi-band Data Structure and Design Example

We have introduced multi-band specifications in DFP and target region determination. In DFP, data is entered using the *DFP-Type and Specifications Window.* After multi-band specifications values are entered, they are passed to the *DFP-Design Method Window* and are processed based on the various filter approximation algorithms (FIR Remez, FIR Windows, FIR Least Square, IIR Butterworth, IIR Chebyshev and IIR Elliptic). DFP adopts the *Userdata* object property associated with MATLAB GUI elements to transfer data among DFP windows. The *Userdata* of the *DFP-Design Method Window* combines all filter specifications. This data is first parsed, then used to estimate filter order, normalize the input and transfer the newly computed *Userdata* to the *DFP-Display Window.* The results of different approximation algorithms become the *Userdata* transferred to the *DFP-Display Window.* Finally, the *DFP-Display Window* receives this *Userdata* and plots the corresponding filter magnitude response, phase/group delay/impulse responses, displays the filter coefficients resulting from the current design, the pole/zero diagram and the target region.

Table 3.4 shows the data structure in the *DFP-Type and Specifications Window,* the *DFP-Design Method Window* and the *DFP-Display Window.* It also shows the data flow direction among the DFP windows.

	Input Data	Output Data
"Type and Specifications Window"	● Frequency vector: fre_vector ● Weight vector: wgt_vector ● Attenuation vector: dev_vector ● Sampling frequency: fs	Userdata1 = [fs, fre_vector, wgt_vector, dev_vector] Userdata1 Output
"Design Method Window"	Userdata1 Input Received Userdata1. Need to do following process: ● Separate Userdata1 ● Order estimation: N ● Data normalization. Suppose normalization results: N, f, w, d ● Calculate target region coordinate matrix	Userdata2 = remez (N, f, w, d) or = fir1(N, f, w, d) or =firls (N, f, w, d) or =butter(N, f, w, d) or =cheby1(N, f, w, d) or =cheby2(N, f, w, d) or =ellip(N, f, w, d) Userdata2 Output
"Display Window"	Userdata2 Input Filter coefficients are determined according to Userdata2	Display the following results: ● Magnitude Response ● Phase Response ● Coefficients ● Target Region

Table 3.4: DFP data structure.

Example: Target region if the first band is a pass-band

Suppose filter specifications are given as following:

Sampling Frequency: 48000(Hz)
Frequency Edge Vector: [8000 10000 13000 15000 18000 20000]
Weight Vector: [1 0 1 0]
Attenuation Vector: [3 50 3 50](dB)

Figure 3.5 shows the target region generated by DFP.

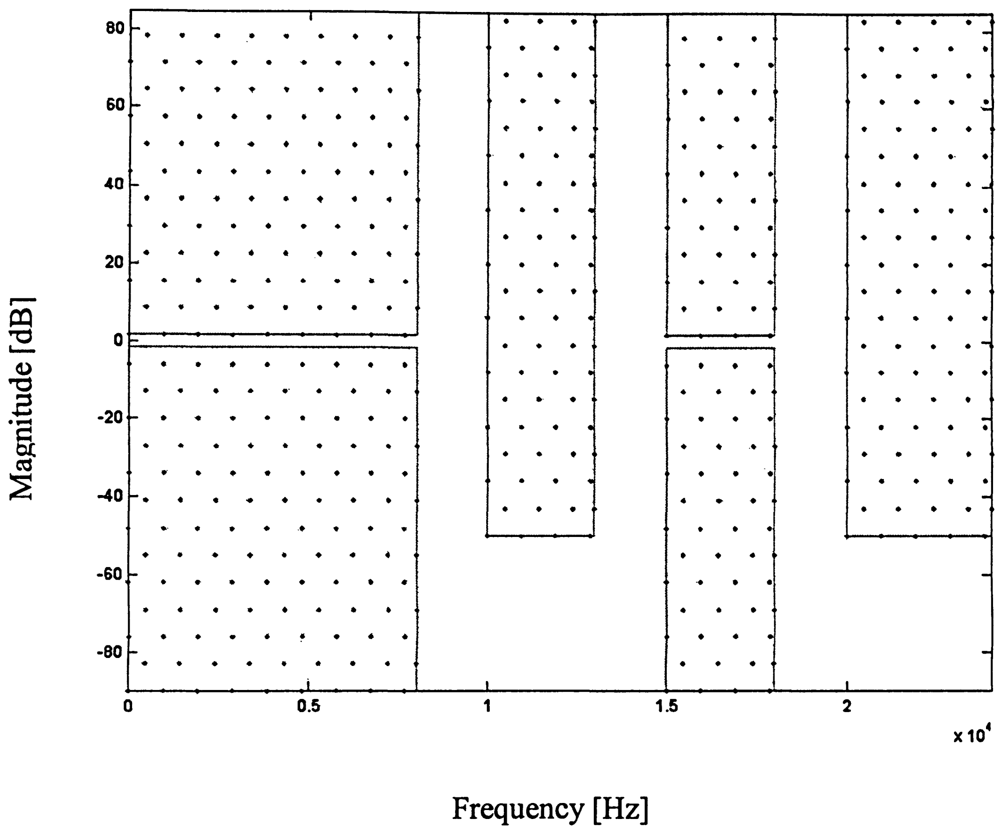


Figure 3.5: Target Region of Logarithm Case

In the *DFP-Design Method* window, if we choose Remez FIR digital filter design algorithm to realize this specification, the filter order estimate is 36. Figure 3.6 displays the corresponding filter magnitude response. Analyzing the target region and the filter design, we observe that this filter design result meets target requirements. If the filter design result does not meet the target requirements, we can interactively change the filter order until we obtain a filter design that satisfies target requirements.

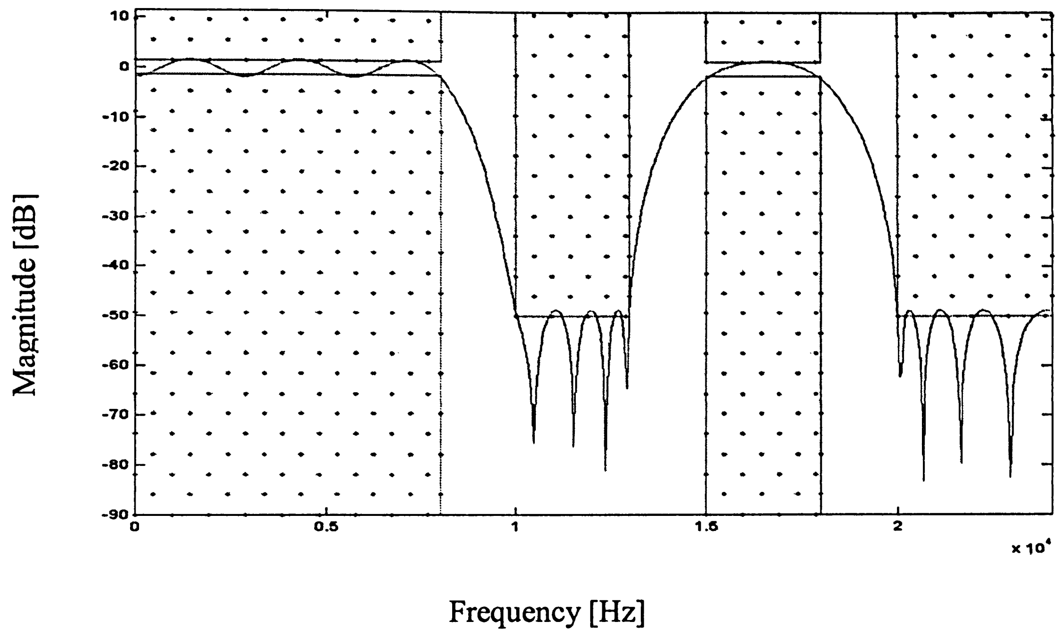


Figure 3.6: Filter Design Result of Logarithm Case if First Band is a Pass-band

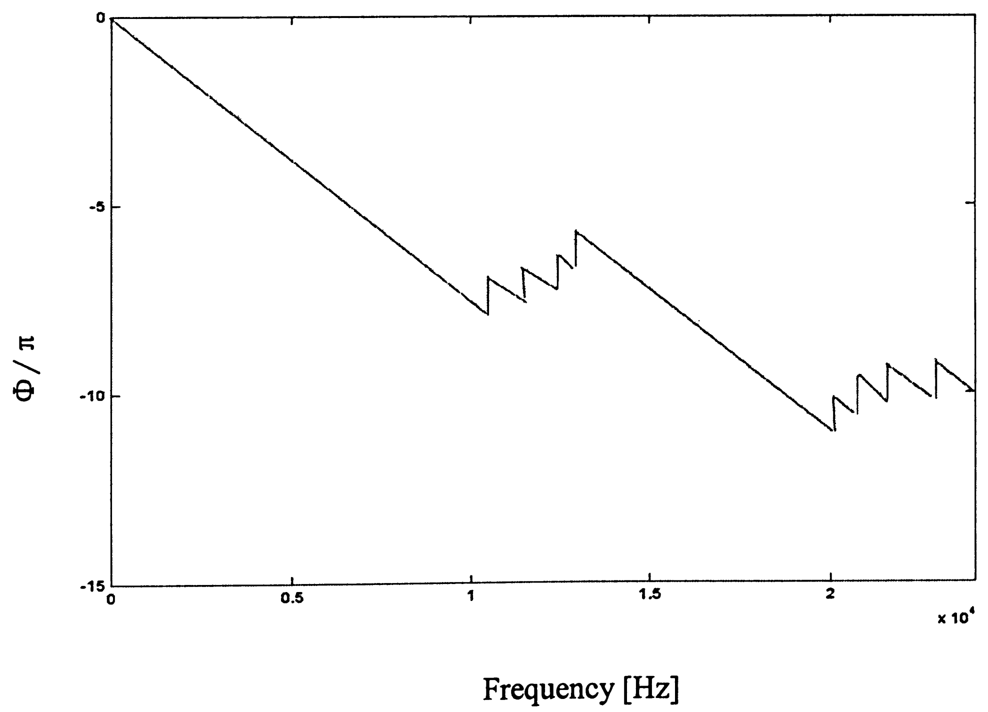


Figure 3.7: Filter Design Result of Phase Response

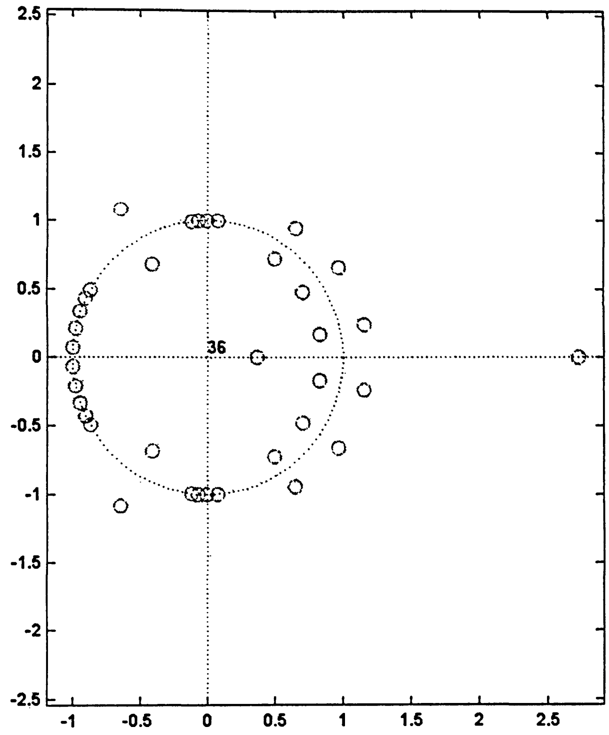


Figure 3.8: Pole/Zero Distribution of the Designed Multiband Filter

Chapter 4

Adaptive Digital Filter Design

Filtering refers to the linear process designed to alter the spectral content of an input signal in a specified manner. Conventional FIR and IIR filters are time-invariant. They perform linear operations on an input signal to generate an output signal based on constant coefficient values. Adaptive filters are time varying, their characteristics such as bandwidth and frequency response change with time. The coefficients of the adaptive filter are adjusted automatically by an adaptive algorithm based on the incoming signal. This feature enables adaptive filters to be used in areas where the requirements of the filtering operation are unknown or non-stationary [23].

Adaptive filtering problems do not have unique solutions. There are a variety of recursive algorithms, each of which offers desirable features and respective limitations. One approach is based on the Wiener filter theory which uses the Least-Mean-Square (LMS) algorithm. The LMS algorithm is simple and capable of achieving satisfactory performance under right conditions. Its major limitations are a relatively slow rate of convergence and sensitivity to variations in the number of the correlation matrix of the tap inputs.

Another approach based on the Kalman filter theory which uses the Recursive-Least-Square (RLS) algorithm. The RLS algorithm can provide a much faster rate of convergence than that attainable by the LMS algorithm. The RLS algorithm is robust in the sense that its rate of convergence is essentially insensitive to the eigenvalue-spread problem. Moreover, we may utilize the Kalman filtering algorithm to deal with non-stationary environments. The basic limitation of this RLS algorithm is its computational complexity [3].

4.1 Least Mean Square Algorithm (LMS)

There are 5 steps in the implementation of the LMS algorithm. Let *W(n)* represent the filter tap weight vector, *x(n)* the input signal, *y(n)* the filter output signal, *d(n)* the desired signal, *e(n)* the error signal, μ the step size of LMS algorithm [26].

1. Select an initial tap-weight vector at time *n*=0 using $W(0) = 0$;
2. Receive an input sequence *x(n)* and for each time sample $n = 1, 2, 3 \ldots N$ compute the filter output.

   $y(n) = W^T(n)x(n)$ or $y(n) = x(n)W^T(n)$
3. Compute the estimation error defined as the difference of the desired value and the filter output value.

   $e(n) = d(n) - y(n)$
4. Update the tap-weight vector

   $W(n+1) = W(n) + \mu e(n)x(n)$
5. Repeat steps 2 through 4 until $n = N$ (sample length).

The choice of step size μ must make algorithm always convergence. To guarantee this, μ must be selected on the basis of maximum eigenvalue of autocorrelation matrix. Let $\lambda_{\max}$ represent the maximum eignvalue of the autocorrelation matrix $x^T(n)x(n)$ such that

$$0 < \mu \le \frac{1}{\lambda_{\max}} \le \frac{1}{x^T(n)x(n)} \tag{4.1}$$

In some instances, the denominator in equation (4.1) can equal zero due to machine precision or signal fading. To avoid this a small positive constant ζ is added to the denominator as shown in equation (4.2).

$$\mu \le \frac{1}{\zeta + x^T(n)x(n)} \tag{4.2}$$

4.2 Recursive-Least-Square Algorithm (RLS)

Let $\Phi(n)$ represent the time-averaged autocorrelation matrix of the input signal as defined in Equation (4.3).

$$\Phi(n) = \sum_{i=1}^{n} x^T(i)x(i) \tag{4.3}$$

Let $P(n) = \Phi^{-1}(n)$. The implementation of the RLS algorithm realization consists of the following steps [26].

1. Compute Kalman gain vector $K(n)$.

$$K(n) = \frac{P(n)x(n)}{1 + x^T(n)P(n-1)x(n)} \tag{4.4}$$

2. Compute $P(n)$

$$P(n) = P(n-1) - K(n)x^T(n)P(n-1) \tag{4.5}$$

3. Compute the *a priori* error $\alpha(n) = d(n) - W^T(n-1)x(n)$ (4.6)

4. Update the tap-weight vector $W(n) = W(n-1) + K(n)\alpha(n)$ for n = 1,2 ...

The initial conditions for RLS algorithm realization are:

1. $K(1) = \dfrac{P(0)x(1)}{1 + x^T(1)P(0)x(1)}$

2. $P(1) = P(0) - K(1)x^T(1)P(0)$

3. $\alpha(1) = d(1) - W^T(0)x(1)$

4. $W(1) = W(0) + K(1)\alpha(1)$ with *W(0)* = 0 and $P(0) = \Phi^{-1}(0) = \delta^{-1}I$ where $\delta \ll 1$ and I is the identity matrix.

4.3 Designing Adaptive Filters Using DFP

In DFP we first wrote the M-functions `adaptlms.m` and `adaptrls.m` which implement the LMS and RLS algorithms, respectively. In the *DFP-Type and Specification* window we have three input variables: input signal vector, desired signal vector and filter order. Input signal vector and desired signal vector should be designed in the MATLAB workspace based on the application requirements. In this section, we provide several application examples. After entering the specifications of the adaptive filter, we proceed to the method selection window to choose either one of LMS and RLS algorithms. In the *DFP-Display* window, we can monitor and compare the learning curves from the LMS and RLS algorithms and the frequency response of the adaptive filter. We can also monitor the resulting filter impulse response and transfer adaptive filter coefficients to the MATLAB workspace.

Example1: System Identification

In this class of applications, an adaptive filter is used as a linear model that represents the best fit to an unknown system. The unknown system and the adaptive filter are driven by the same input *x(n)*. The unknown system supplies the desired response for the adaptive filter. Figure 4.1 presents this approach. [1]

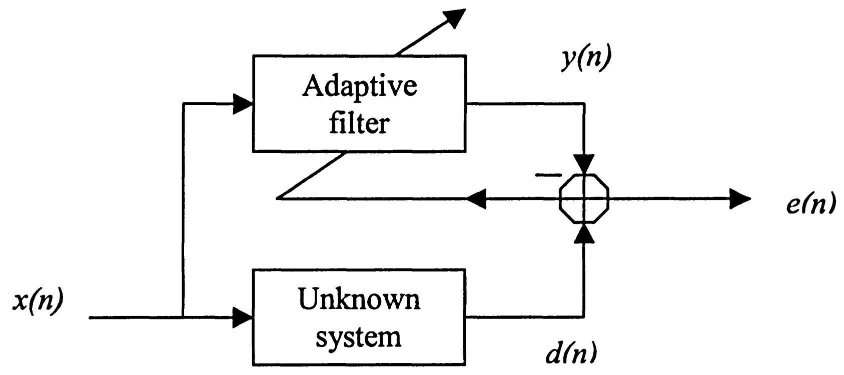


Figure4.1: System Identification

We observe that when *e(n)* is very small, the adaptive filter response is close to the response of the unknown system.

To use the adaptive filter functions in DFP, we need to provide three design parameters. First we need to generate the signal input *x(n)* to both the unknown system and the adaptive filter as shown in Figure 4.1. Then we generate the desired signal *d(n)*. Finally we specify the tap length N of the adaptive filter.

In this example, we have 500 points of the random input signal *x(n)*. The desired signal *d(n)* is the output from a known LPF. If the tap length N of adaptive filter we want to design is the same as known LPF order 13, then after we enter the three design parameters (tap length, input signal and desired signal) into the *DFP-Adaptive Filter Design* window, DFP generates as one output the adaptive filter coefficients. According to Figure 4.1 system identification analysis, the filter coefficients of this unknown system should be the same as designed adaptive filter coefficients. In this way, we can identify unknown system characteristic in term of known system characteristic. Another output from DFP is the squared error. Using this parameter, we can plot the learning curve of the adaptive filter.

The following MATLAB sample code is used to generate the input and the desired signal vectors in the base MATLAB workspace.

```
input_signal = 0.1 * randn(1,500);

b = remez(12,[0 0.4 0.5 1], [1 1 0 0], [1 0.2]); % b is LPF coefficients

desired_signal = filter(b,1,input_signal);
```

Figure 4.2 shows the LMS algorithm learning curve. Figure 4.3 shows the RLS algorithm learning curve. From the learning curve displayed in DFP, we observe that the RLS algorithm has a faster convergence speed than the LMS algorithm.

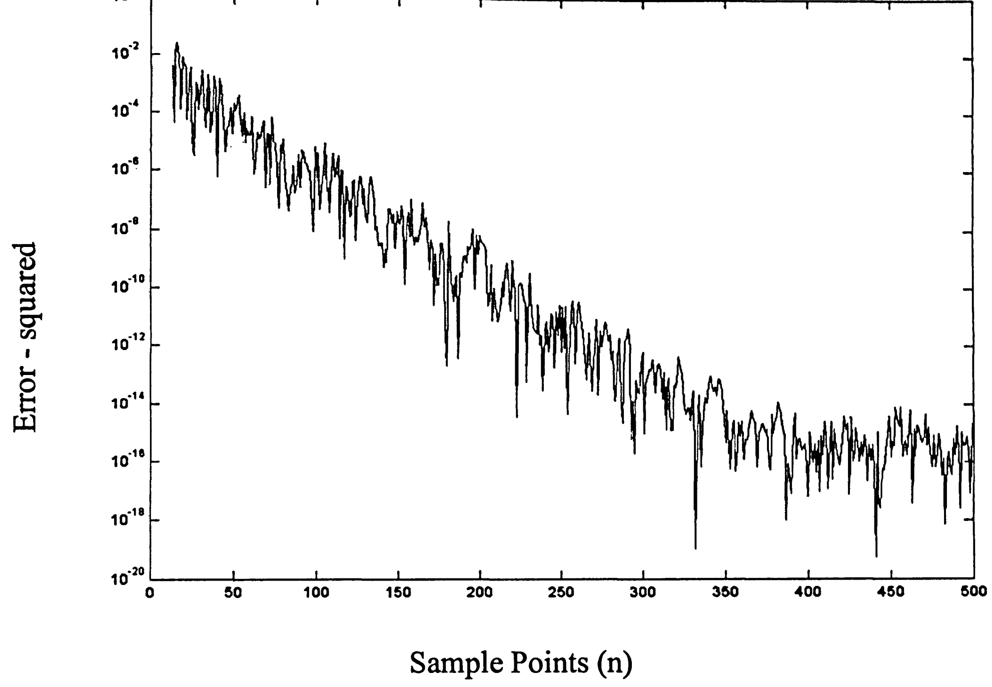


Figure 4.2: LMS Learning Curve for the System Identification Example

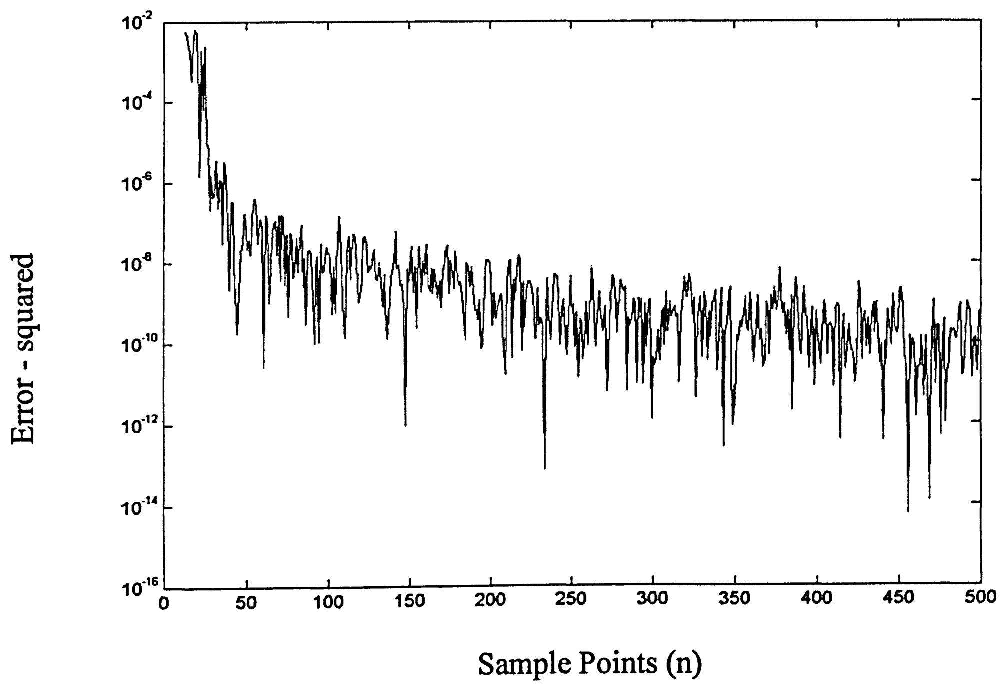


Figure 4.3: RLS Learning Curve for the System Identification Example

Example 2: Inverse System Identification

In this class of applications, the function of the adaptive filter is to provide an inverse model that represents the best fit to an unknown system. Ideally, the inverse model has a transfer function equal to the reciprocal of the unknown system's transfer function. A delayed version of the system input constitutes the desired response for the adaptive filter. Figure 4.4 represents this approach. Let

x(n) = input applied to the adaptive filter

y(n) = output of the adaptive filter

d(n) = desired response

e(n) = *d(n) – y(n)* = estimation error

s(n) = system input

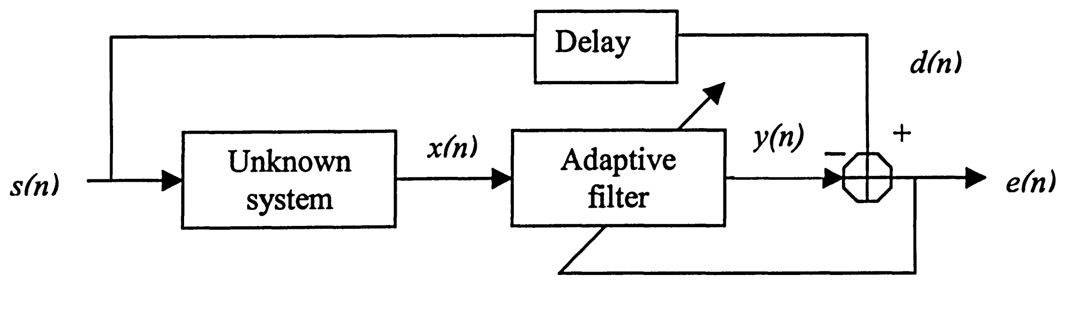


Figure4.4: Inverse System Identification

Figure 4.4 shows that the process requires a delay element inserted in the desired signal path to keep the data at the summation point synchronized. The delay element keeps the system causal. Without the delay element, the adaptive filter algorithm tries to match the output from the adaptive filter *y(n)* to input data *x(n)* that has not yet reached the adaptive elements because it is passing through the unknown system. Therefore, the filter cannot compensate for the unknown system response. Including a delay equal to the delay path caused by the unknown system prevents this condition.

Adaptive channel equalization is an application example of the inverse system identification problem. Let us consider a digital communication channel. *s(n)* is an uncorrelated random sequence of binary values –1 and +1 with zero mean. In this example, the unknown system is the transmission channel. The additive channel noise is uncorrelated with mean 0 and variance 0.001. Therefore, *x(n)* is the sum of *s(n)* and the noise. In order to eliminate Inter-Symbol Interference (ISI), a seven order adaptive filter is used. The following MATLAB sample code is used to generate *x(n)* and *d(n)* in the base MATLAB workspace.

```
W=2.9;                                                  % Channel parameter
nvari = 0.001;                                          % Variance of noise
s = fix(rand(1,700)+0.5)*2-1;                           % System input
v = fix(randn(1,700)+0.5)*2*sqrt(nvari)-sqrt(nvari);    % Random Noise
for k=1:3
   h(k) = 1/2*(1+cos(2*pi/W*(k-2)));                    % Channel Impulse Response
end
xx = conv(s,h);                                         % Channel output
inp = xx(1:700)+v;                                      % Adapt filter input
ref = [0 0 0 s(1:697)];                                 % Desired signal
                          % as contrast, if we input not delay variable.
inp1 = xx(1:700)+v(1:700);                              % Adapt filter input
ref1 = s;                                               % Desired signal
                          % Then it is difficult to converge.
```

We first enter the input *x(n)* and *d(n)* vectors and the adaptive filter order into DFP, we then obtain the adaptive filter coefficients and the learning curves from the LMS and RLS algorithms. Figures 4.5 and 4.6 show respectively the learning curves corresponding to the LMS and the RLS algorithms.

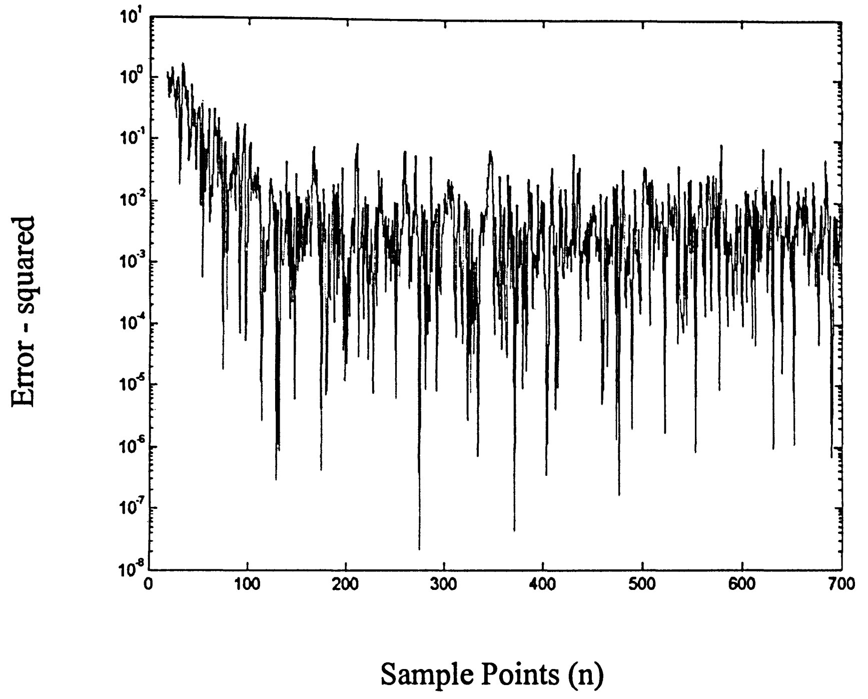


Figure 4.5: LMS Learning Curve for Inverse System Identification

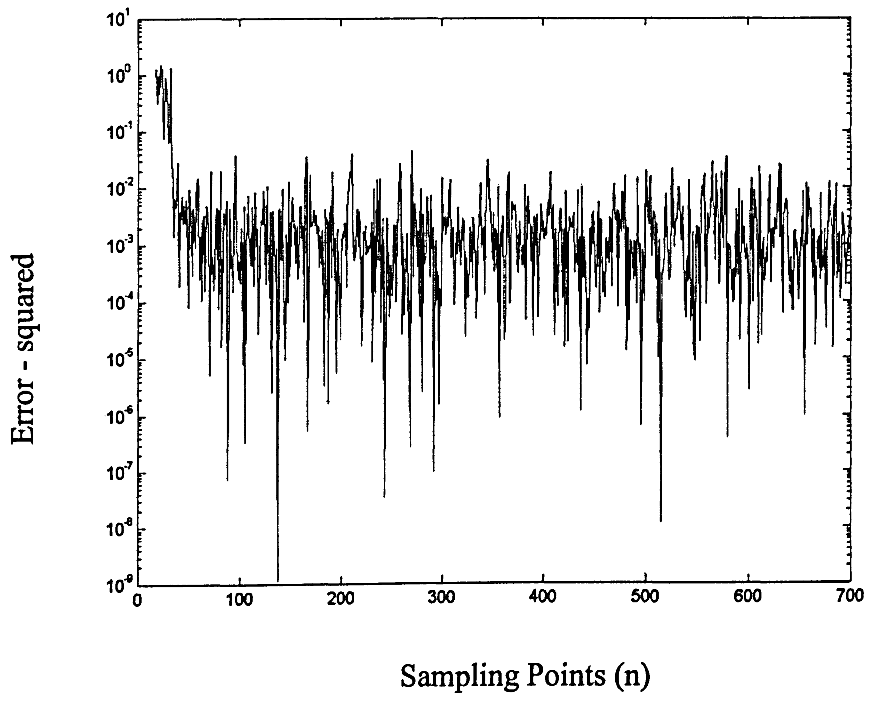


Figure 4.6: RLS Learning Curve for Inverse System Identification

Example 3: Noise Interference Cancellation

In this example, the adaptive filter is used to cancel the interference contained in the primary signal. Here we use the same notation as in the previous example. The primary signal *s(k)+n(k)* serves as the target response of the adaptive filter. An auxiliary noise signal *n'(k)* is the input to the adaptive filter. As long as the input noise to the filter remains correlated with the unwanted noise accompanying the target signal, the adaptive filter adjusts its coefficients to reduce the difference between *y(k)* and *d(k)*, thus removing the noise and generating the *clean* signal in *e(k)*. Notice that in this application, the error signal actually converges to the input data signal, rather than converging to zero.

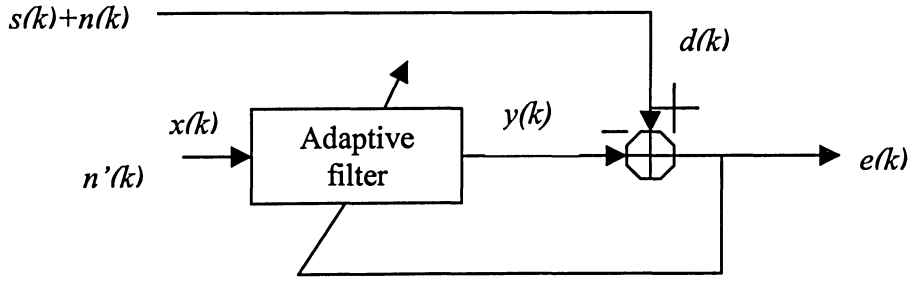


Figure 4.7: Noise Cancellation

To realize this application in DFP, we use the following sample code. We use the notation where: the system signal *s(k)* is a sinusoid, *n'(k)* is the noise to be eliminated. In adaptive noise cancellation system, *n'(k)* is the signal input *x(k)* to the adaptive filter, *d(k)= s(k)+n(k)* is the target signal for the adaptive filter desired. In order to make *n(k)* correlated with *n'(k),* we pass *n'(k)* through a low-pass FIR filter. Then, we define *n(k)* as the output of this low-pass FIR filter.

Once we enter the input and the target signal vector to DFP, we obtain the de-noised signal and the adaptive filter coefficients. De-noised signal results obtained using the LMS and RLS algorithm are shown in Figures 4.8 and 4.9. Adaptive filter coefficients using LMS algorithm is shown in Figure 4.10.

```
s = sin(2*pi*0.055*[0:1000-1]');          % System Input Signal
x = randn(1,1000);                         % Adaptive Filter Input Signal.
nfilt = firl(11,0.4);                      % LPF Coefficients
n = filter(nfilt,1, x);                    % Correlated Noise Data.
d = s.'+ n;                                % Adaptive Filter Desired Signal
```

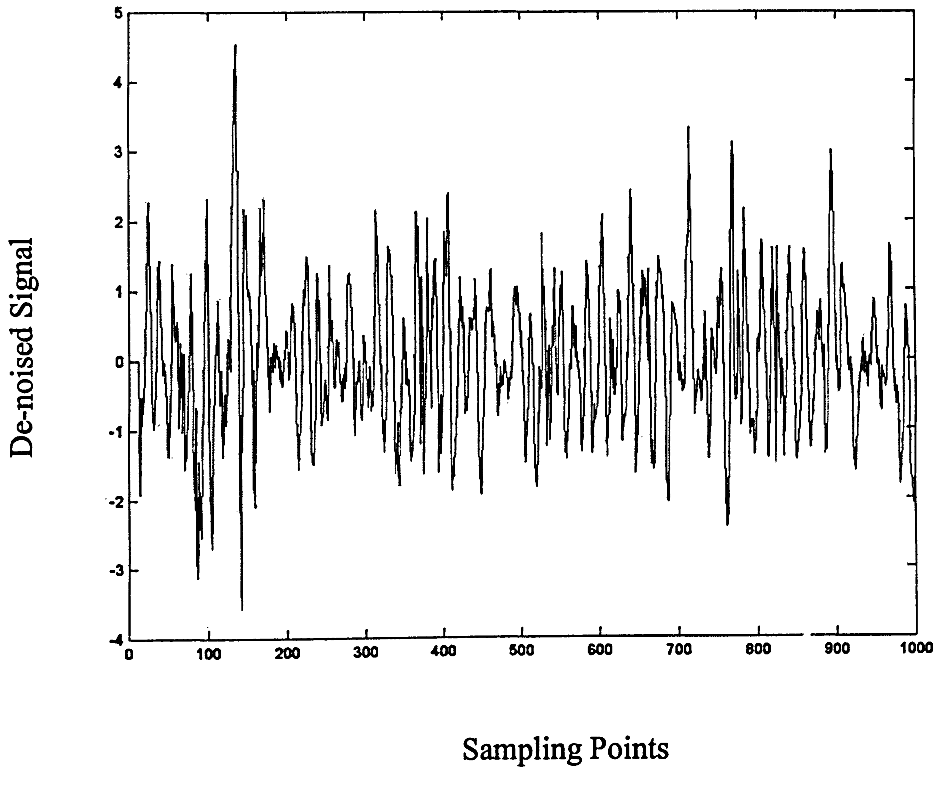


Figure 4.8: De-noised Signal Using LMS Algorithm

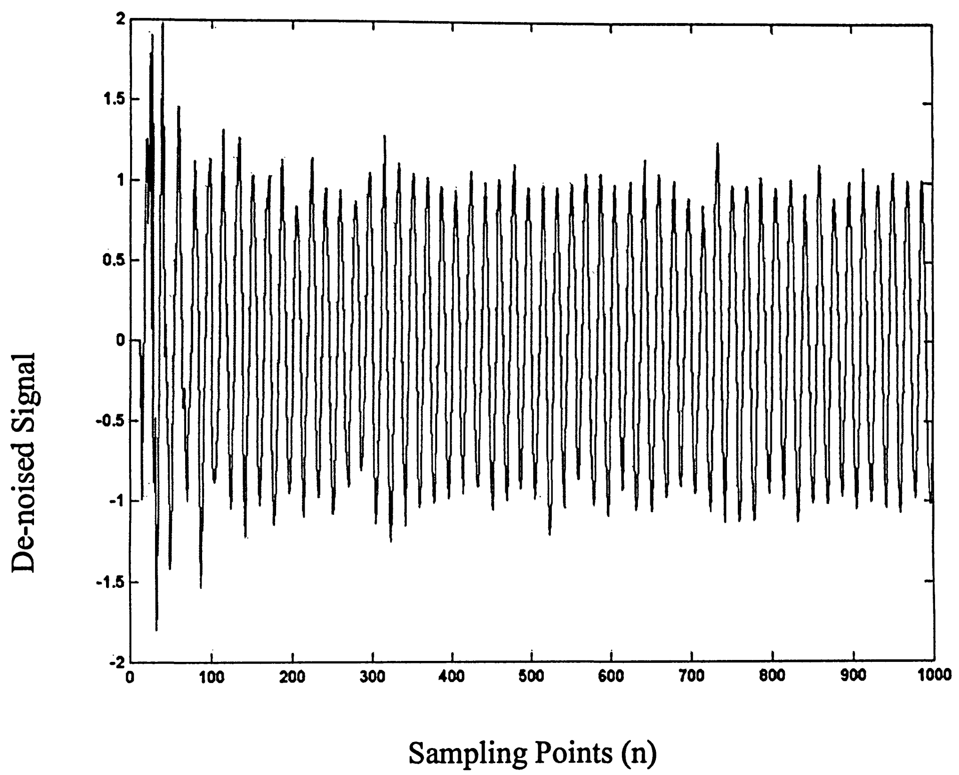


Figure 4.9: De-noised Signal Using RLS Algorithm

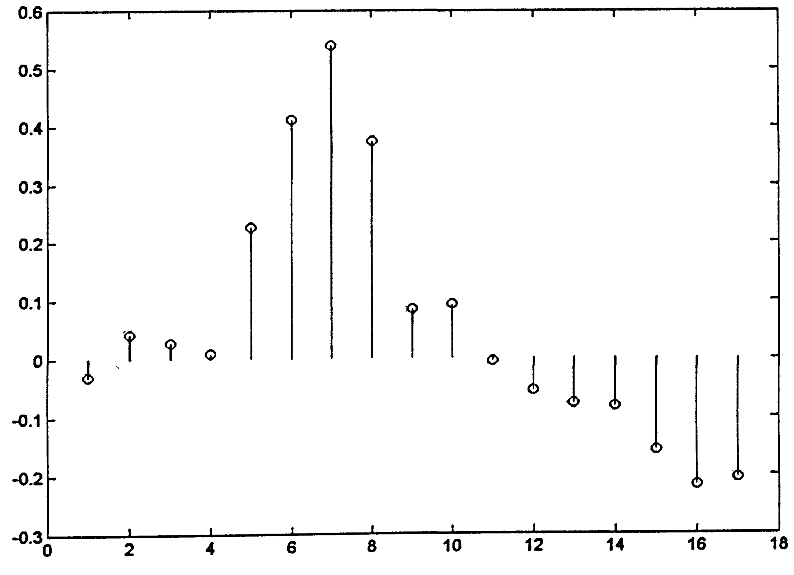


Figure 4.10: Adaptive Filter Coefficients Using LMS Algorithm

Chapter 5

2-Dimensional Digital Filter Design

As a result of rapid increase in the demand and development of video consumer products, high performance 2-D digital filters have become an important component of digital signal processing. 2-D digital filters can be found in diverse applications such as image/video processing, image restoration and noise reduction. They represent one of the most basic and important processing techniques in image and data processing. Typically, a recursive 2DDF (2D Digital Filter) has better magnitude response than a nonrecursive 2DDF of the same order. However, since a non-recursive, i.e., an FIR, filter is stable, realizable and always can be designed to have linear phase response, non-recursive 2DDF has been proved to be much easier for hardware implementation.

The main approach used in this project is based on separable 2-D digital filter design method. The separable-denominator 2DDF is a very important filter class because its design and analysis are easy and any arbitrary frequency response can be approximated [6].

5.1 Separable-Denominator 2DDF

Equation (5.1) gives the transfer function of a separable-denominator 2DDF.

$$H(z_1,z_2)=\frac{N(z_1,z_2)}{D_1(z_1)D_2(z_2)} \tag{5.1}$$

If the numerator polynomial $N(z_1,z_2)$ can be factored as

$$N(z_1,z_2)=N_f(z_1)N_g(z_2) \tag{5.2}$$

then $H(z_1,z_2)$ can be decomposed into a product of two 1-DDF transfer functions $H_f(z_1)$ and $H_g(z_2)$ as shown

$$H(z_1,z_2)=H_f(z_1)\cdot H_g(z_2) \tag{5.3}$$

where

$$H_f(z_1)=\frac{N_f(z_1)}{D_1(z_1)} \tag{5.4a}$$

$$H_g(z_2)=\frac{N_g(z_2)}{D_2(z_2)} \tag{5.4b}$$

The flow-chart for designing separable-denominator 2DDF based on the reduced-dimensional decomposition is shown in Figure 5.1. By means of the reduced-dimensional decomposition, the design problem of the separable 2DDF is simply reduced to the problem of designing two 1DDF.

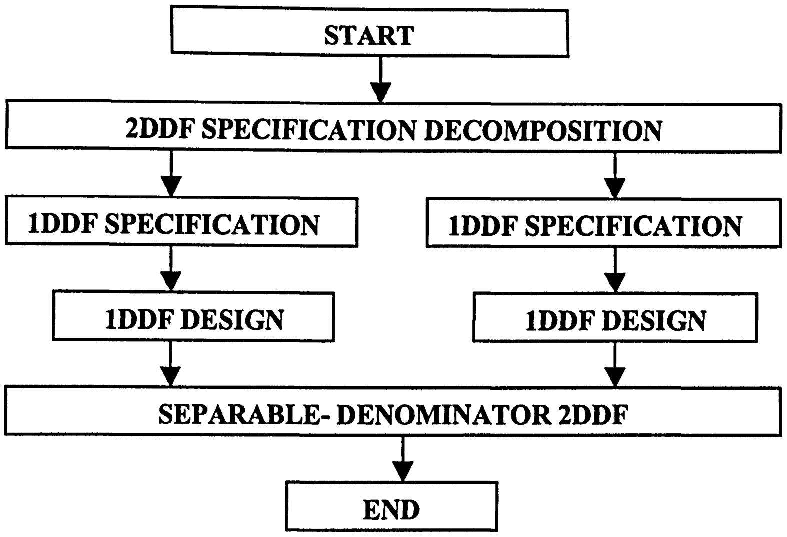


Figure 5.1: Separable-Denominator 2DDF Design Flow-Chart

5.2 2DDF Design in DFP

The MATLAB Image Processing Toolbox provides several two dimensional digital filter design methods such as 1-D window method, 2-D window method, two-dimensional frequency response method, and frequency transformation method.

FWIND1 and FWIND2 2DDF design functions in the Image Processing Toolbox can be used to design an approximately circularly symmetric 2-D FIR filter by using 1-D window method. If we enter just one 1DDF specification, then we obtain the resulting 2DDF, which assumes that the two-dimensional filter will have the same specifications in both dimensions. The only difference between FWIND1 and FWIND2 is that FWIND1 works with one-dimensional windows, while FWIND2 works with two-dimensional windows. By using this method we cannot design 2DDF with two distinct one-dimensional specifications.

FTRANS2 designs a 2-D FIR filter using frequency transformation. It is based on the same assumptions as FWIND1 and FWIND2. They all cannot design 2DDF with distinct one-dimensional specifications. FSAMP2 designs two-dimensional FIR filters according to a desired two-dimensional frequency response. For those users who cannot access Image Processing Toolbox, they cannot design 2-D digital filters directly by using these 2-D design methods.

In Chapter 3, we introduced the design of one-dimensional multi-band digital filters in DFP. For the 2DDF case, we use the separable 2DDF design mechanism. Our design objective for the 2DDF is to have flexible multi-band two- dimensional digital filter design ability. So the idea of designing 2D filters in DFP is that we do not want to depend on built-in 2D filter design functions in the Image Processing Toolbox. We integrated separable 2D filter design method into DFP package. The advantage over using Image Processing Toolbox 2D filter design functions lies in that we can design multi-band 2D filter, which can have different 1DDF specifications. The 2DDF we designed can have one of the four possibilities listed in Table 5.1.

		Dimension1 Specification	
		LPF	HPF
Dimension2 Specification	LPF	**2-D L-L**	**2-D L-H**
	HPF	**2-D H-L**	**2-D H-H**

Table 5.1: DFP Design Type of 2DDF

The display window provides a 3-dimensional frequency and impulse responses of the 2-D filter. We can interactively change the filter order we design and obtain the desired filter design. If we want to identify two 1DDFs, which are design elements of 2DDF, we can use

1DDF design display windows to verify whether the filter order can meet the target requirements.

Example 1: Design L-L mode 2DDF

Figure 5.2 presents the resulting design corresponding to the following 2DDF specifications.

DFP-Type and Specification

x: Frequency Edge Vector: [12000 15000]
Weight Vector: [1 0]
Attenuation Vector: [3 50]
y: Frequency Edge Vector: [12000 15000]
Weight Vector: [1 0]
Attenuation Vector: [3 50]
Sampling Frequency: 48000

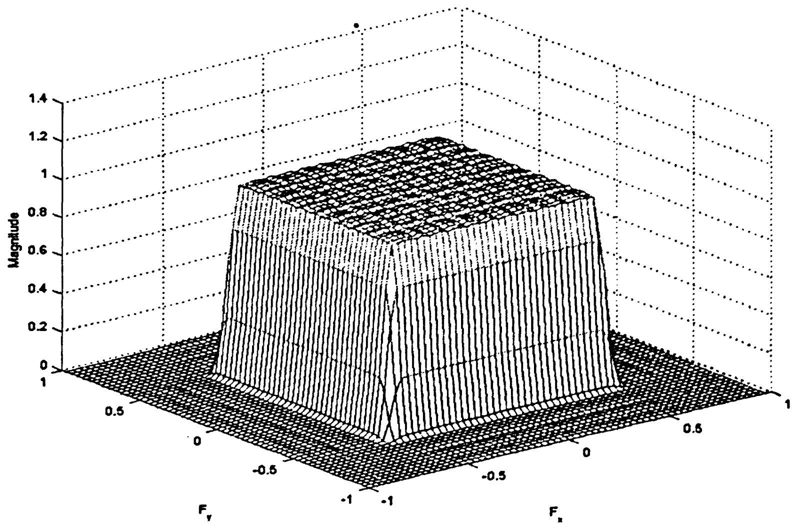


Figure 5.2: 2DDF with two low-pass 1DDF specifications

Example 2: Design L-H mode 2DDF

Figure 5.3 presents the resulting design corresponding to the following 2DDF specifications.

DFP-Type and Specification

x: Frequency Edge Vector: [12000 15000]

Weight Vector: [1 0]

Attenuation Vector: [3 50]

y: Frequency Edge Vector: [12000 15000]

Weight Vector: [0 1]

Attenuation Vector: [50 3]

Sampling Frequency: 48000

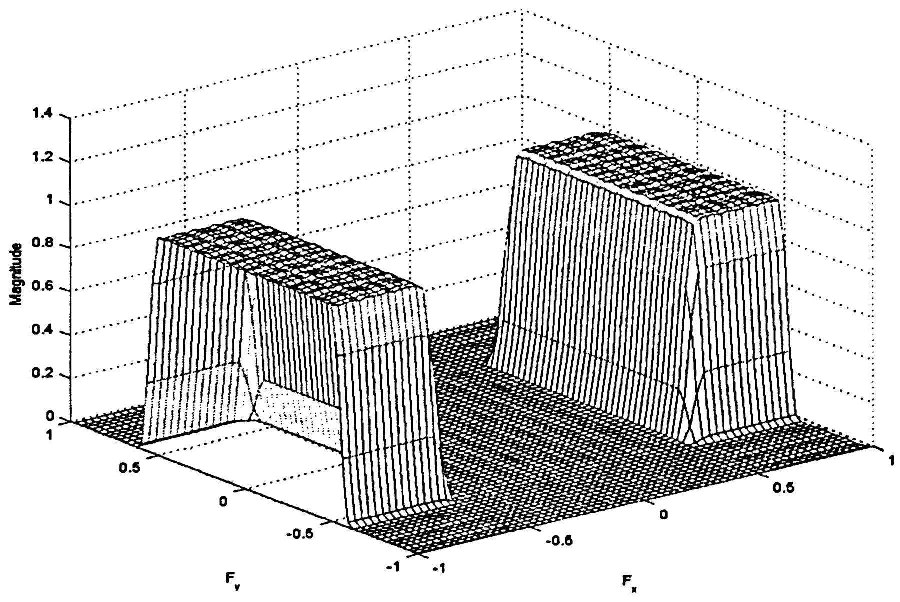


Figure 5.3: 2DDF with low-pass and high-pass 1DDF specifications

Example 3: Design H-L mode 2DDF

Figure 5.4 presents the resulting design corresponding to the following 2DDF specifications.

DFP-Type and Specification

x: Frequency Edge Vector: [12000 15000]

Weight Vector: [0 1]

Attenuation Vector: [50 3]

y: Frequency Edge Vector: [12000 15000]

Weight Vector: [1 0]

Attenuation Vector: [3 50]

Sampling Frequency: 48000

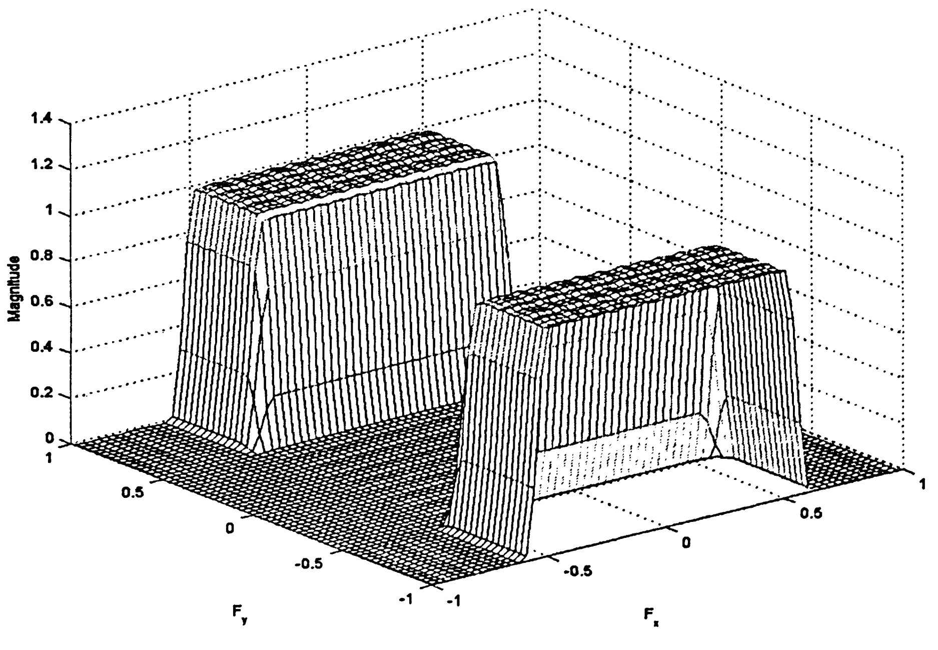


Figure 5.4: 2DDF with high-pass and low-pass 1DDF specifications

Example 4: Design H-H mode 2DDF

Figure 5.5 presents the resulting design corresponding to the following 2DDF specifications.

DFP-Type and Specification

x: Frequency Edge Vector: [12000 15000]
  Weight Vector: [0 1]
  Attenuation Vector: [50 3]
y: Frequency Edge Vector: [12000 15000]
  Weight Vector: [0 1]
  Attenuation Vector: [50 3]
Sampling Frequency: 48000

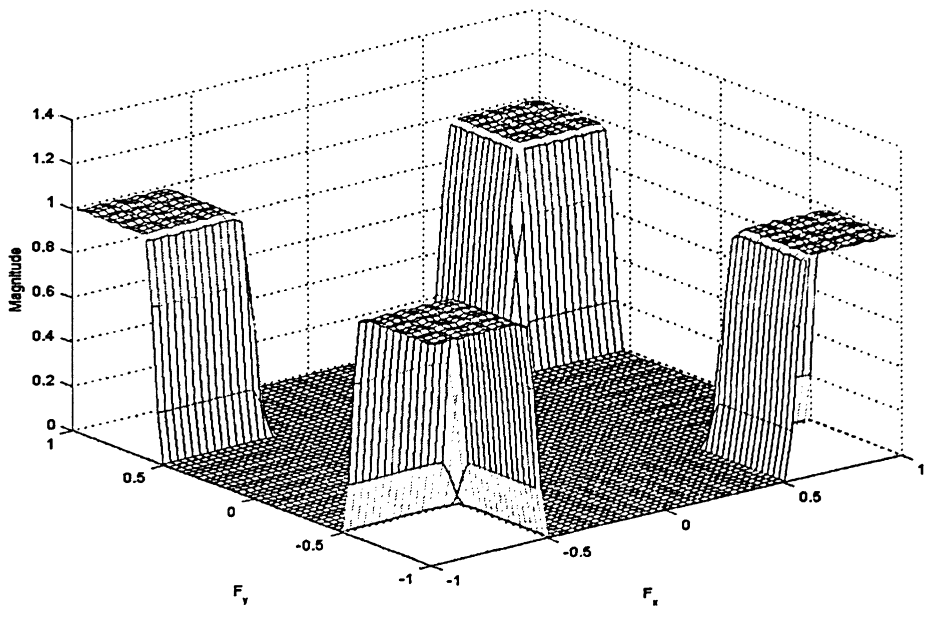


Figure 5.5 2DDF with two high-pass 1DDF specifications

Example 5: Design multi-band mode 2DDF

Figure 5.6 presents the resulting design corresponding to the following 2DDF specifications.

DFP-Type and Specification

x: Frequency Edge Vector: [10000 11000 13000 14000 16000 17000]

Weight Vector: [1 0 1 0]

Attenuation Vector: [3 50 3 50]

y: Frequency Edge Vector: [12000 13000 15000 16000 18000 19000]

Weight Vector: [1 0 1 0]

Attenuation Vector: [3 50 3 50]

Sampling Frequency: 48000

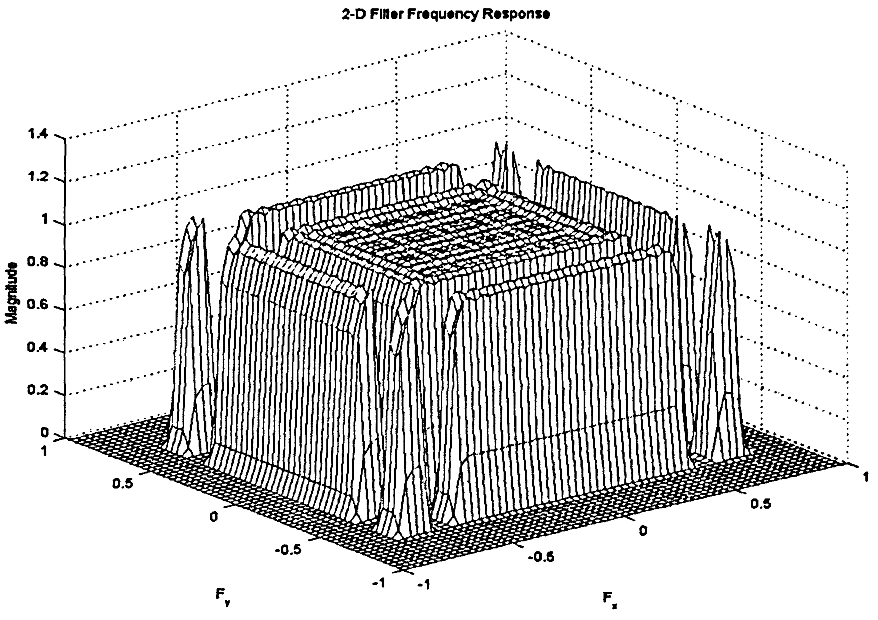


Figure 5.6: 2DDF with multi-band 1DDF specifications

Chapter 6

DFP GUI Design and TMS320 Code Generation

Graphical User Interface (GUI) is a user interface built with graphical objects, such as buttons, text fields, sliders, and menus. In general, these objects already have meanings to most computer users. For example, when you move a slider, a value changes; when you press a **CLOSE** button, your settings are applied and the dialog box is dismissed [10].

Applications that provide GUIs are generally easier to learn and use. The action that results from a particular user action can be made clear by the design of the interface.

6.1 DFP GUI Structure and Design

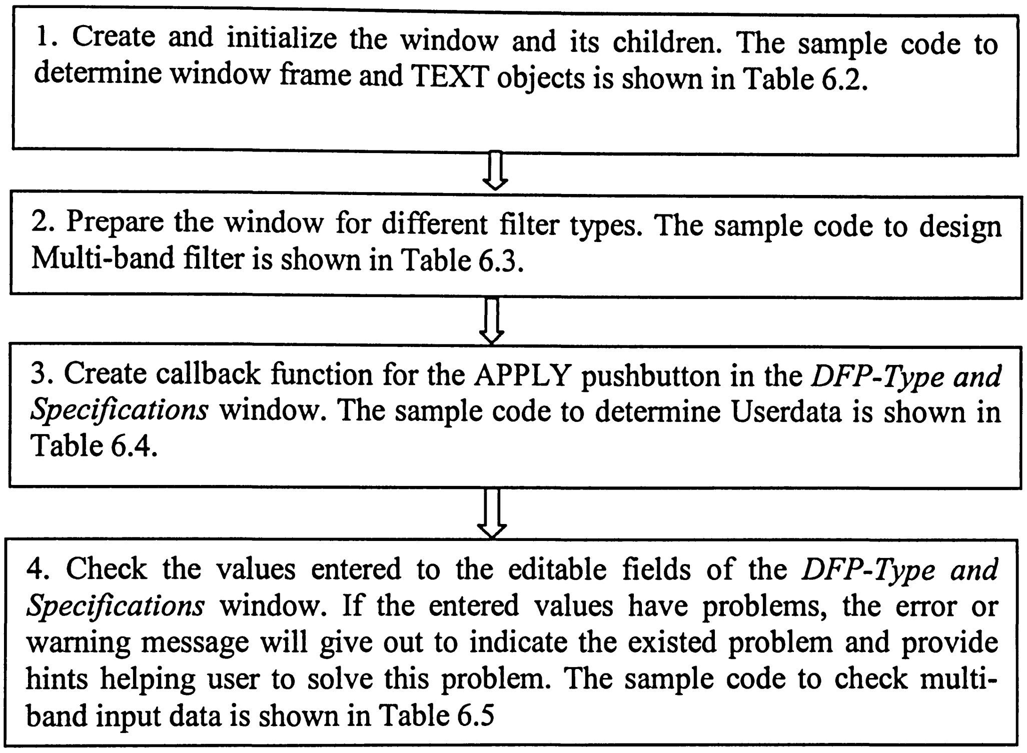
The original DFP package has 14 windows. Table 6.1 provides an overview of the DFP windows. These windows are designed using GUI objects built into MATLAB.

Original DFP Windows	New Windows
DFP: Navigator	'DFP: Display Window' Note: This window is used to plot adaptive filter learning curve, adaptive/2-D filter magnitude response, impulse response and display filter coefficients.
DFP: Type and Specifications	
DFP: Design Method	
DFP: Display	
DFP: Implementation	
DFP: Output Format	'DFP: Magnitude15 Axes' Note: This is the snapshot window for 2-D filter magnitude response
DFP: File I/O	
DFP: Properties	
DFP: About	
DFP: Magnitude Axes	
DFP: Phase Axes	'DFP: Learning Curve Axes' Note: This is the snapshot window for adaptive filter learning curve and magnitude response
DFP: Group Delay	
DFP: Impulse Response	
DFP	

Table 6.1: DFP Windows

6.1.1 Types and Specifications Window

For this part, the design programs were modified in order to extend the DFP design capability to include multi-band, adaptive and 2-D filters. The main modifications lie in the size and the position modifications for the frame object, text object and editable text object so that the layout is suitable for all filter types. The design procedure for the *DFP-Types and Specifications* window is shown in Figure 6.1. Tables 6.2–6.5 provide commented code samples.



1. Create and initialize the window and its children. The sample code to determine window frame and TEXT objects is shown in Table 6.2.

2. Prepare the window for different filter types. The sample code to design Multi-band filter is shown in Table 6.3.

3. Create callback function for the APPLY pushbutton in the *DFP-Type and Specifications* window. The sample code to determine Userdata is shown in Table 6.4.

4. Check the values entered to the editable fields of the *DFP-Type and Specifications* window. If the entered values have problems, the error or warning message will give out to indicate the existed problem and provide hints helping user to solve this problem. The sample code to check multi-band input data is shown in Table 6.5

Figure 6.1: Design Procedures for the *DFP-Type and Specifications* window

Table 6.2: Sample Code to Determine Window Frame and TEXT Objects

```
% FRAME objects

% Upper program has defined FRx, FRy1, FRy2, FRy3, FRy31, Fryp. Dim1 = 1;Dim2 = 1;Dim5=0;
% FRx is x coordinate of frame. FRy1 is y coordinate with one frame wide. FRy2 is y coordinate with
% two frame wide. FRy3 is y coordinate with three frame wide. FRy31 is y coordinate with three frame
% wide  for 2-D filter. FRyp is y coordinate with push button wide. Dim8 = 0;

% Then we can determine Frame Position in "Type and Specifications Window"

FRpos=[Dim1  Dim2+FRyp+Dim8+2*FRy2+3*Dim5+FRy3     FRx  FRy1;
       Dim1  Dim2+FRyp+Dim8+FRy2+2*Dim5+FRy3       FRx  FRy2;
       Dim1 Dim2+FRyp+Dim8+FRy2+Dim5+FRy3          FRx  FRy31;
       Dim1  Dim2+FRyp+Dim8+FRy2+Dim5+FRy3-FRy2    FRx  FRy2;
       Dim1  Dim2+FRyp+Dim8+FRy2+Dim5               FRx  FRy3;
       Dim1  Dim2+FRyp+Dim8                         FRx  FRy2;
       Dim1  Dim2                                   FRx  FRyp;   ];

% The first two dimensions in each row indicate the frame x and y coordinate position relative to left
% lowest window point. The next two dimensions indicate the frame length and wide. Here we defined
% 7 frame positions.

% TEXT objects
% This part defines text object size and position. TXx is text object wide, TXy is text object height.
% Dim 3 is button to left/right Frame edge. Dim4 is button to bottom/top Frame edge. Dim7 is the
% separation between buttons within the Frame

TXpos = [
     ...
     FRpos(1,1)+Dim3  FRpos(1,2)+Dim4            TXx  TXy;
     ...
     FRpos(2,1)+Dim3  FRpos(2,2)+Dim4+ TXy+ Dim7  TXx  TXy;
     FRpos(2,1)+Dim3  FRpos(2,2)+Dim4            TXx  TXy;
     ...
     FRpos(5,1)+Dim3  FRpos(5,2)+Dim4+2*TXy+2*Dim7  TXx  TXy;
     FRpos(5,1)+Dim3  FRpos(5,2)+Dim4+ TXy+ Dim7  TXx  TXy;
     FRpos(5,1)+Dim3  FRpos(5,2)+Dim4            TXx  TXy;
     ...
     FRpos(6,1)+Dim3  FRpos(6,2)+Dim4+ TXy+ Dim7  TXx  TXy;
     FRpos(6,1)+Dim3  FRpos(6,2)+Dim4            TXx  TXy;
  ];
 TXstr = str2mat( '',    ...
          '', '', '', '', '', '', '', '');                        % Text string matrix

% POP-UP MENU  objects

Dim11 = (FRx-PUx)/2; % PUx is popup menu x coordinate. PUy is y coordinate.
PUpos = [ Dim11  FRpos(1,2)+FRpos(1,4)+3*Dim21  PUx  PUy ];
PUstr = [ 'Low Pass|High Pass|Band Pass|Band Stop|Differentiator|'
         'Hilbert Transform|Multiband|Adaptive|2-D' ];  % Popup menu string
```

Table 6.3: Sample Code to Prepare Window Frame and TEXT Objects

```
% This part code is used to define which frame and text objects are active according to sample code in
% Table 6.1
elseif ( FiltTypeNumber == 7 )  % Multiband

  FRindex = [ 1 2 0 4 0 0 7 ];   % Refer to FRpos, the 1,2,4,7 frames are active
  TXindex = [ 1 2 3 4 0 0 0 0 ]; % Refer to TXpos, the 1,2,3, 4 text position are active
  ETindex = [ 1 2 3 4 0 0 0 0 ]; % The 1,2,3,4 editable text position are active
  TXstr   = str2mat( ...
            'Sampling Frequency ',
            'Frequency Edge Vector',
            'Weight Vector', .
            'Attenuation Vector(db)'        ...
            );                    % Define the text string name
```

Table 6.4: Sample code to create callback function for the APPLY pushbutton

```
elseif ( FILTtypeNumber == 7 )
 % Multiband call back function sample code
% This code is used to prepare the Userdata to transfer to "Design Method Window"
   ss = get(EThan(1),'String');  % get the first editable text object string value
   SpecData1 = str2num( ss );
   size1 = length(SpecData1);

   ss = get(EThan(2),'String');  % get the second editable text object string value
   if ( isempty(str2num(ss)) )  % this indicates array name input
     SpecData2 = evalin( 'base', ss );
   else
     SpecData2 = str2num( ss );
   end
   size2 = length(SpecData2);

   ss = get(EThan(3),'String');   % get the third editable text object string value
   if ( isempty(str2num(ss)) )   %this indicates array name input
     SpecData3 = evalin( 'base', ss );
   else
     SpecData3 = str2num( ss );
   end
   size3 = length(SpecData3);

   ss = get(EThan(4),'String');   % get the fourth editable text object string value
   if ( isempty(str2num(ss)) )   %this indicates array name input
     SpecData4 = evalin( 'base', ss );
   else
     SpecData4 = str2num( ss );
   end
   size4 = length(SpecData4);
   SpecData = [SpecData1 SpecData2 SpecData3 SpecData4]; % Defined userdata
   NNN = size1 + size2 + size3 + size4;
```

Table 6.5: Sample Code to Check Multi-band Input Data

```
elseif   ( FILTtypeNumber == 7 )  % for multi-band

    totallength = length(SpecData);  % Separate and restore input data according to Userdata
    bands = (totallength + 1)/4;
    frelength = 2*bands -2;

    fs  = SpecData(1);
    fre_vector = [(SpecData(2:(frelength+1)))];
    wgt_vector = [(SpecData((frelength+2):(frelength+1+bands)))];
    dev_vector = [(SpecData((frelength+2+bands):totallength))];

    for i = 1:length(dev_vector)
      j = rem(i,2);
      if ( j ==1 )
```

```
          dev1(i) = dev_vector(i); % odd
          wgt1(i) = wgt_vector(i);
       else
          dev2(i) = dev_vector(i); % even
          wgt2(i) = wgt_vector(i);
       end
    end

  % Any intermediate frequencies exceeding (fs/2) ?
  if ( any( fre_vector >= fs/2 ) )
    d_error( 'ET-02' )
    flag = 0;   % Give error information and hint to solve this error
  end

  % Are intermediate frequencies non-decreasing?
  if ( any( diff([0 fre_vector]) <= 0 ) )
    d_error( 'ET-03' )
    flag = 0;
 end

  % Are Attenuation parameters non-negative ?
  if ( any( dev_vector <= 0 ) )
    d_error( 'ET-04' )
    flag = 0;
  end

  % Are Stopband/Passband Attenuation parameters comparable ?
 for i = 2:length(wgt_vector)
    j = rem(i,2);
 if ( ( j == 1 ) & ( wgt1(i) == 1 ) & ( dev1(i) >= dev2(i-1)) )
    d_error( 'ET-05' )
    flag = 0;
 elseif ( ( j == 0 ) & ( wgt2(i) == 1 ) & ( dev2(i) >= dev1(i-1)) )
    d_error( 'ET-05' )
    flag = 0;
 end
```

6.1.2 The *DFP-Design Method* Window

In order to design adaptive filter in DFP, I generated adaptive design methods: LMS and RLS algorithms and integrated into DFP. For multi-band and 2-D filter design, I modified the original design program and solved the following problems:

1. How to translate the Userdata coming from *DFP-Types and Specifications* window?
2. How to prepare the Userdata to be passed to *DFP-Display* window.

Tables 6.6 provides commented code sample used for designing adaptive filters.

```
elseif (FILTtypeNumber == 8)  % Adaptive Filter Design Method
totallength = length(FILTdata); % FILTdata is received Userdata from "Types and Specifications
ban = (totallength-1)/2;         % Window"
NN1 = FILTdata(1);               % First Userdata (It is Filter Order.)
input_vector = FILTdata(2:(1+ban));          % Restored input signal;
ref_vector  = FILTdata((2+ban):totallength); % Restored reference signal;
RBusr = str2mat(...
  '', ...
  '', ...
  '', ...
  '', ...
  '', ...
  '', ...
  '', ...
['adaptlms([' sprintf('%s ',input_vector) '],[' sprintf('%s ', ref_vector) '],[' sprintf('%g ',NN1) '])' ], ...
['adaptrls([' sprintf('%s ',input_vector) '],[' sprintf('%s ', ref_vector) '],[' sprintf('%g ',NN1) '])' ]);

for ii=[8 9]
  % set(TXhan(ii), 'String', sprintf('%i', NN1));
  set( RBhan(ii), 'Userdata' , deblank(RBusr(ii,:))); %Prepare the Userdata to "Display Window"
  set( RBhan(ii), 'Userdata' , deblank(RBusr(ii,:)), 'Enable', 'on' );
  %set( RBhan(8), 'Userdata' , '' , 'Enable', 'on' );
  %set( RBhan(9), 'Userdata' , '' , 'Enable', 'on' );
end
  for ii=[1:7]
  set(TXhan(ii), 'String', '' );
  set(EDhan(ii), 'String', '', 'Enable', 'off');
  set(RBhan(ii), 'Userdata', '', 'Enable', 'off');

end
set( RBhan(10), 'Userdata' , '' , 'Enable', 'off' );
```

Table 6.6: Sample Code for Adaptive Filter Design Method

6.1.3 *DFP-Display* window

The learning curve is the preferred tool for monitoring results from adaptive filter design techniques. Similarly, a three-dimensional filter response display window is the preferred method of displaying the results from the 2-D digital filter design. Because it is difficult to use original display window to plot adaptive and 2-D filter design results, we designed a stand-alone display window for adaptive and 2-D digital filters. The new window design is based on the original display window. Some work was needed to prevent transition problems among the new and original *DFP-Display* windows. Figures 6.2 and 6.3 show respectively the layout of the original and the new display windows.

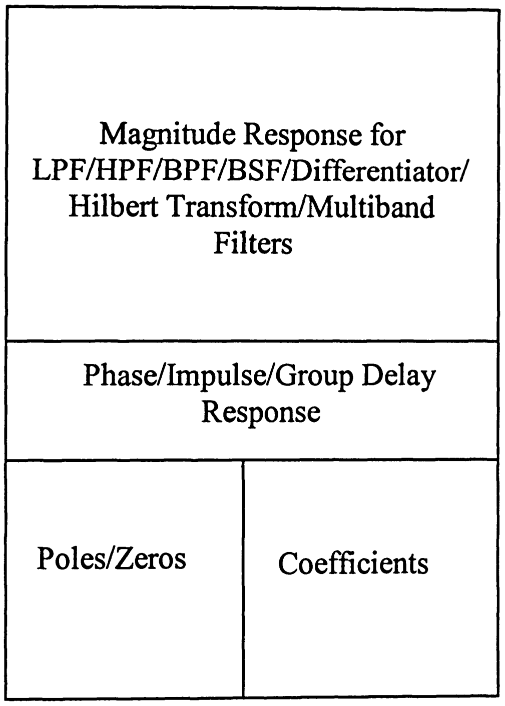

Magnitude Response for LPF/HPF/BPF/BSF/Differentiator/ Hilbert Transform/Multiband Filters
Phase/Impulse/Group Delay Response
Poles/Zeros
Coefficients

Figure6.2: Display Window Layout of Original

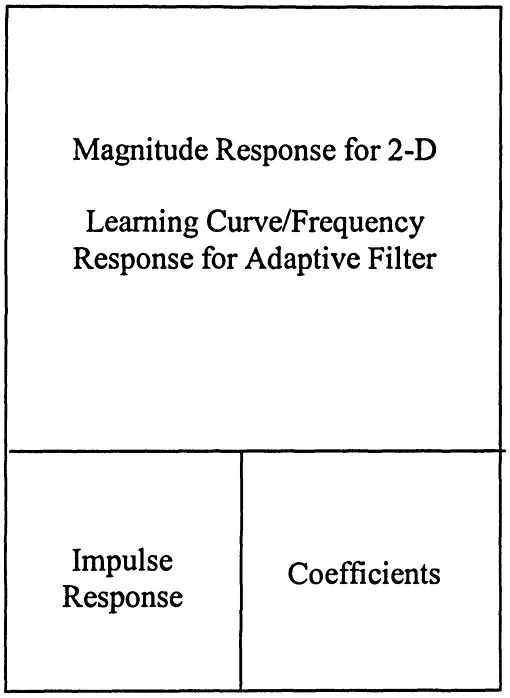

Magnitude Response for 2-D
Learning Curve/Frequency Response for Adaptive Filter
Impulse Response
Coefficients

Figure 6.3: Display Window Layout of New Display

The *DFP-Display* window receives its *Userdata* from the *DFP-Design Method* window and plots the magnitude, phase/impulse/group-delay responses, and the learning curve. Filter coefficients are also displayed in the display window.

6.2 TMS320 Code Generation

TMS32010 is the first fixed-point DSP in the TMS320 family. Today, TMS320 DSP family consists of three supported DSP platforms: TMS320C2000, TMS320C5000, and

TMS320C6000. The TMS320C2000 DSP platform is frequently used in the digital control industry. This generation of DSPs deliver power and control advantages that allow designers to develop modern and cost-efficient control systems. The TMS320C5000 DSP platform is optimized for the mobile Internet and its convergence with other consumer electronics. The TMS320C6000 DSP platform is optimized for highest performance and ease-of-use in high-level language programming.

The TMS320 implementation codes integrated with the DFP are collected from a variety of Texas Instrument standard documents [11,12,13, 19, 20, 21, 22, 23, 24, 25] and the T.I. BBS site. After classifying and analyzing these codes, we integrated them into the DFP according to different filter design type and realization structures. The current DFP release provides code generation for FIR/IIR direct/lattice structures and IIR Second Order Section (SOS) cascade structures for the TMS32067x processor; and FIR/IIR direct structures, IIR SOS cascade structures for TMS32010/TMS32020/TMS32054x processors.

As the code for all different T.I. DSP families are not fully compatible, we were not able to test all TMS320 codes integrated in DFP. The installed TMS320C67x DSK can only test the TMS320C67x code. In order to test the FIR/IIR Direct Form implementation code, we programmed test files writing in C program language. For other implementation structures (FIR/IIR lattice structure and IIR SOS), we only provided the complete subroutine assembly codes. Table 6.7 compares the TMS32010/TMS32020, TMS320C54x and TMS320C67x DSP implementation codes.

	TMS32010/ TMS32020	TMS320C54x	TMS320C67x
Data Format of Filter Coefficients	16 bit Hexadecimal word without the header 0x, such as FFC2.	16 bit Hexadecimal word without the header 0x, such as FFC2.	16 bit Hexadecimal word with the header 0x, such as 0xFFC2.
Section Coefficient Representation for SOS Structure	The k^{th} section: $[b_{1k}\ b_{2k}\ b_{3k}\ \text{-}a_{2k}\ \text{-}a_{3k}]$	The k^{th} section: $[-a_{3k}\ -a_{2k}/2\ b_{3k}\ b_{2k}\ b_{1k}]$	The k^{th} section: $[\text{-}a_{3k}\ \text{-}a_{2k}\ b_{3k}\ b_{2k}\ b_{1k}]$
Addressing Mode	• Direct Mode • Indirect Mode	Circular Mode	Circular Mode
Software Pipeline	No	Yes	Yes
Coding Constraints	Need large program memory for long filter coefficients.	• Code for FIR/IIR Direct Structure with 32*16-bit. (Input data are 32 bits, filter coefficients are 16 bits.) • Code for IIR SOS structure with 16*16-bit.	• Number of filter coefficients must be a multiple of 4. • The size of the block must be a multiple of 2.

Table 6.7 Comparisons of Code Generation Considerations

The Z-transform of the unit-sample response of an IIR filter with a SOS has been introduced in Chapter 2 as

$$H(z)=\prod_{k=1}^{N/2}\frac{b_{1k}+b_{2k}z^{-1}+b_{3k}z^{-2}}{1+\alpha_{2k}z^{-1}+\alpha_{3k}z^{-1}} \qquad (6.1)$$

The difference equation corresponding to equation (6.1) is shown by equation (6.2), where k represents the section number.

$$y(n)=\prod_{k=1}^{N/2}[\sum_{i=1}^{3}b_{ik}x(n-i)-\sum_{i=2}^{3}a_{ik}y(n-i)] \qquad (6.2)$$

In Table 6.7 the representation of section coefficients for the SOS structure is based on Equation (6.2) yet has a separate format for different T.I families.

There are direct addressing mode and indirect addressing mode for TMS32010/TMS32020. Direct addressing mode uses LTD/MPY instruction pair to implement the multiplications and shifts. If there is a long length of the filter coefficients, the coding program becomes very long for direct addressing mode. We need to use indirect addressing mode to reduce program memory size. Using either of the auxiliary registers along with the auto-increment or auto-decrement feature, the program can be rewritten in looped form or repeat instruction RPTK and MACD (multiply and accumulate with data move) pair. TMS32010/TMS32020 code generated by DFP uses indirect addressing mode.

TMS320C54x and TMS320C67x use circular buffer addressing mode to conserve memory and minimize software overhead. Circular addressing uses pointer manipulation to add the new samples to the buffer by overwriting the oldest available samples hence reusing the memory buffer.

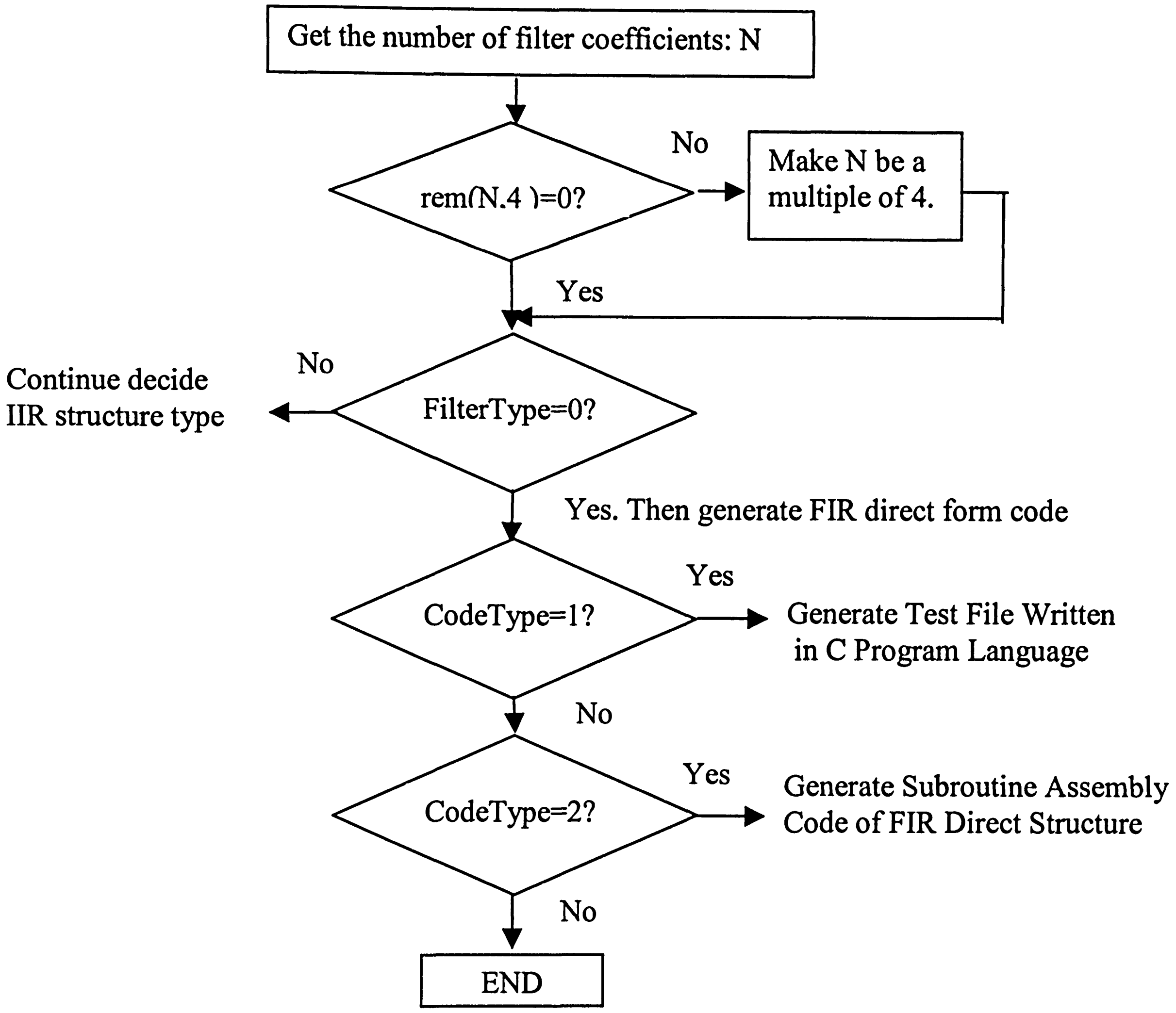
The pipeline technique has been used in TMS320C54x and TMS320C67x DSP to improve processor performance and reduce the overall instruction execution time. The pipeline execution breaks a sequence of operations into small segments and executes these small pieces in parallel.

In DFP, TMS32010/TMS32020, TMS320C54x and TMS320C67x families have their own code generation programs. These code generation programs are also based on the implementation of Motorola 56000 DSP code generator. Table 6.8 shows the sample program for generating TMS320C67x test file and the subroutine assembly code for the FIR direct form.

Table 6.8: Sample Program for TMS320C67x Code Generation

```
% Now start generating the filter code

if ( FilterType == 0 )   %==== FIR Direct Form ===

  NN = Options(2);          % if NN is not the number multiple of 4, we make it become multiple of 4
  if ( rem(NN,4) == 1 )
     NN = NN + 3;  strr = str2num(str);
     strrr = [strr;0;0;0]; str = num2str(strrr);
  elseif ( rem(NN,4) ==2 )
     NN = NN +2;   strr = str2num(str);
     strrr = [strr;0;0]; str = num2str(strrr);
  elseif ( rem(NN,4) ==3 )
     NN = NN +1;   strr = str2num(str);
     strrr = [strr;0]; str = num2str(strrr);
  end

  NUM_SAMP = NN+4;

  if ( CodeType == 1 )       % This is for generate main test file

  s = [ DFP_TemplatesDir2 'fir_d00.asm' ]; % Determine whether fir_d00.asm template exist.
  if ( exist(s) )
      d_append( Fid, s );
  else
      d_error('EO-06', s, DFP_TemplatesDir2 );
      return;
  end

  TOT_SAMP= 100;         % Input number
  fprintf( Fid,'//Define Circular Block Size (BUF_LEN), Number of Coefficients (NUM_TAPS) and \n' );
  fprintf( Fid,'//Block FIR Size (NUM_SAMP). BUF_LEN is defined in bytes \n' );
  fprintf( Fid,'#define BUF_LEN 128  \n' );
  fprintf( Fid,'#define TOT_SAMP 100  \n' );
  fprintf( Fid,'#define NUM_TAPS %i  \n', NN );
  fprintf( Fid,'#define NUM_SAMP %i  \n', NUM_SAMP );
  fprintf( Fid,'short out_array[TOT_SAMP];  \n' );
  fprintf( Fid,'short in_array[BUF_LEN/2];  \n' );
  fprintf( Fid,'short inp_samp[TOT_SAMP+NUM_TAPS-1];  \n' );
  fprintf( Fid,'static void dataIO() \n' );
  fprintf( Fid,'{  \n' );
  fprintf( Fid,'   /* do data I/O */ \n' );
  fprintf( Fid,'   return;  \n' );
  fprintf( Fid,'}  \n' );
  N2 = NN/2;
  fprintf( Fid,'short coeff_array[NUM_TAPS] = {  %s , %s, \n', str(1,:) , str(2,:));
  for ii = 2:N2-1
    jj = 2*ii-1; jj1 = 2*ii;
  fprintf( Fid,'                    %s , %s, \n', str(jj,:) , str(jj1,:));
  end
  fprintf( Fid,'                   %s , %s}; \n', str(2*N2-1,:) , str(2*N2,:));
  fprintf( Fid,'extern void fir_circ_asm(short *y, short *x, int n, short *h, int s, int m, int size,int indexex);
\n' );
  fprintf( Fid,'void readdata(short *y, short *x, int n, int m); \n' );
  fprintf( Fid,'main() \n' );
  fprintf( Fid,'{ \n' );
  fprintf( Fid,' int scale_factor=15; \n' );
  fprintf( Fid,' int index=0; \n\n' );
  fprintf( Fid,'dataIO();  \n\n' );
  callnum = floor(TOT_SAMP/NUM_SAMP)-1;  % determine calling subroutine times
  indexx = NUM_SAMP+NN-1; % the first time input sample length
  fprintf( Fid,'readdata(inp_samp, in_array, 0, %i); \n\n', indexx );
  fprintf( Fid,'//Call Block FIR algorithm \n' );
  fprintf(      Fid,'fir_circ_asm(out_array,      in_array,      NUM_TAPS,      coeff_array,
scale_factor,NUM_SAMP,BUF_LEN,index); \n\n' );
  fprintf( Fid,'//Compute next INDEX value based on the old INDEX and BLOCK FIR (nor circular \n' );
  fprintf( Fid,'//buffer block) size BUFLEN/2 is used since the pointer points to 16 bit data \n\n' );

  for ii = 1:callnum

  fprintf( Fid,'index = (index+NUM_SAMP)-(BUF_LEN/2)*floor((index+NUM_SAMP)/(BUF_LEN/2));
\n\n' );
  indexx1 = indexx + (ii-1)*NUM_SAMP; indexx2 = indexx + (ii)*NUM_SAMP;
  fprintf( Fid,'readdata(inp_samp, in_array, %i, %i); \n\n',indexx1, indexx2 );
  fprintf(   Fid,'fir_circ_asm(&out_array[%i*NUM_SAMP],   in_array,   NUM_TAPS,   coeff_array,
scale_factor,NUM_SAMP,BUF_LEN,index); \n\n',ii );

  end
  fprintf( Fid,'void readdata(short init_values[], short array[], int n, int m) \n' );
  fprintf( Fid,'{ \n' );
  fprintf( Fid,'  int i, temp; \n' );
  fprintf( Fid,'  for (i=n;i<m;i++) \n' );
  fprintf( Fid,'  { \n', NN );
  fprintf( Fid,'   temp=i-(BUF_LEN/2)*floor(i/(BUF_LEN/2)); \n' );
  fprintf( Fid,'   array[temp]=init_values[i];\n' );
  fprintf( Fid,'  } \n' );
  fprintf( Fid,' }  \n' );

  elseif ( CodeType == 2 )
  s = [ DFP_TemplatesDir2 'fir_d02.asm' ];
    if ( exist(s) )
      d_append( Fid, s );
    else
      d_error('EO-06', s, DFP_TemplatesDir2 );
    return;
    end
  end
```

The program flow-chart is shown in Figure 6.4.



Get the number of filter coefficients: N
rem(N.4)=0?
No
Make N be a multiple of 4.
Yes
No
Continue decide IIR structure type
FilterType=0?
Yes. Then generate FIR direct form code
CodeType=1?
Yes
Generate Test File Written in C Program Language
No
CodeType=2?
Yes
Generate Subroutine Assembly Code of FIR Direct Structure
No
END

Figure 6.4: Program Flow-Chart to Generate Test File and Subroutine Assembly Code

In this flow-chart, Filter Type is used to define which structure code to be generated. Code Type is used to define output code type (test file or subroutine assembly code). Table 6.9 lists the values of these two parameters for different T.I. families.

		Filter Type	Code Type
TMS32010/TMS32020 TMS320C54x	FIR Direct Form	0	1: Macro Code 2:Full Code
	IIR Direct Form	2	
	IIR SOS	4	
TMS320C67x	FIR Direct Form	0	1: Test File in C Language 2: Subroutine Assembly Code
	FIR Lattice Form	1	
	IIR Direct Form	2	
	IIR Lattice Form	3	
	IIR SOS	4	

Table 6.9 Filter Type and Code Type Value

After we generate test file and subroutine assembly code, we can test these codes on DSP320C6711 DSK and Code Compose Studio software development using the following test procedure.

1. Turn on the power of the DSK. It goes through a Power On Self-Test (POST) procedure first in which the three LEDs count 1-7, and then all LEDs blinks to show that the tests completed successfully.
2. Create a new project called fir.pjt.
3. Create a new DSP/BIOS configuration file called fir.cdb using the dsk6711.cdb template. Save this file in the same subdirectory.
4. Add the configuration file to the project. This also automatically adds the file fircfg.s62.
5. Also add the linker command file fircfg.cmd automatically generated when we generate DSP/BIOS configuration file.
6. Building the project: In the compiler window, add the new generated FIR subroutine code using assembly language and circular addressing mode as well as parallel instruction execution technique. We also add the C test code consists of filter coefficient array designed from DFP package, circular addressing parameters and a probe point to read

input data from computer. After finishing compiling the source files, we build the project by choosing rebuild all. If there is no error occur, an output file is generated.

7. Load and run the program, we can either use an oscilloscope to the DSK output or we use Time/Frequency Graph Display method provided in C6711 DSK to monitor the output waveform.

Chapter 7

Conclusions and Future Research

In this study we added multi-band, adaptive and 2-D filter design and some TMS320C54x/TMS320C67x/TMS32010/TMS32020 code generation capability to the DFP. These new features make DFP a powerful digital filter design tool. These new features also make DFP distinct from other digital filter design tools. Table 7.1 provides an overview of the new DFP features developed in this study.

Filter Types	**Filter Design Methods**	**Code Generation**	**Display Option**
Multiband	*Adaptive LMS*	*TMS32010/TMS32020*	*Additional Display Window for Adaptive and 2-D Filter*
Adaptive	*Adaptive RLS*	*TMS320C54x*	
2-D		*TMS320C67x*	*Learning Curve Display*

Table 7.1: New DFP features.

The current DFP depends on significant number of global variables to allow various modules to communicate with each other. Since MATLAB has introduced data structures as part of its programming language, converting DFP such that all global variables are replaced with appropriate structure will be a very desirable enhancement. This will make the entire

DFP software package easier to maintain and will allow efficiency in the further development of its capabilities.

As part of further development we consider adding interactive pole-zero placement method to be used as a teaching tool. In addition there are other useful digital filter types such as comb filter, notch filter, etc., that can be integrated into DFP.

Bibliography

[1]. Mohamed EL-Sharkawy, *Real Time Digital Signal Processing Applications with Motorola's DSP56000 Family*, Prentice-Hall, Englewood Cliffs, New Jersey 07632.

[2]. Bogner, R.E., *Introduction to Digital Filtering*, New York, Wiley, London, 1975.

[3]. Simon Haykin, *Adaptive Filter Theory*, Prentice Hall Inc., 1984.

[4]. J.G. Proakis and D.G. Manolakis, *Digital Signal Processing: Principles, Algorithms, and Applications*, Prentice-Hall, 1996.

[5]. Sen M. Kuo and Bob H. Lee, *Real-Time Digital Signal Processing: Implementations, Applications, and Experiments with the TMS320C55x*, John Wiley & Sons, Inc. 2001.

[6]. Tian-Bo Deng and Masayuki Kawamata, "Design of Separable-Denominator Two-Dimensional Digital Filters Based on the REDUCED-Dimensional Decomposition of Frequency Domain Specifications", *Electronics and Communications in Japan*, Part 3, Vol. 73, No. 9,1990.

[7]. Y.Kamp and J.P. Thiran, "Chebyshev Approximation for Two-Dimensional Nonrecursive Digital Filters", *IEEE Transactions on Circuits and Systems*, vol. Cas-22, No.3, March 1975.

[8]. D.G. Manolakis, V.K. Ingle, and S.M. Kogon, *Statistical and Adaptive Signal Processing*, McGraw Hill, 2000.

[9]. *Filter Design Toolbox for use with MATLAB*, The Mathworks.

[10]. *MATLAB GUIs*, The Mathworks.

[11]. Todd Anderson, "The TMS320C2xx Sum-of-Products Methodology", *Texas Instruments*, SPRA068, May 1996.

[12]. "TMS320C4x General-Purpose Applications User's Guide, Digital Signal Processing Solutions," *Texas Instruments.*

[13]. "TMS320C54x DSP Reference Set," Volume 2: Mnemonic Instruction Set, Texas Instrument.

[14]. E.C. Ifeachor and B.W. Jervis, *Digital Signal Processing: A Practical Approach*, Addison Wesley, 1993.

[15]. Bozic, S. M. (Svetozar Mile), *Digital and Kalman filtering: an introduction to discrete-time filtering and optimum linear estimation*, London: E. Arnold, 1979.

[16]. Hamming, R. W. (Richard Wesley), *Digital filters*, Englewood Cliffs, N.J.: Prentice-Hall, 1977.

[17]. Naim Dahnoun, *DSP implementation using the TMS3206x processors*, New York: Prentice Hall, 2000.

[18]. *Digital Signal Processing Applications with the TMS320 Family – Theory, Algorithms, and Implementations*, Texas Instruments, 1986.

[19]. "TMS320C6000 Code Composer Studio Tutorial", Literature Number: SPRU301C, February 2000.

[20]. "TMS320C6000 Assembly Language Tools User's Guide", Literature Number: SPRU1861, April 2001.

[21]. "TMS320C6000 Programmer's Guide", Literature Number: SPRU198G, August 2002.

[22]. "TMS320 DSP Algorithm Standard Rules and Guidelines", SPRU352D, January 2001.

[23]. "Extended Precision IIR Filter Design on the TMS320C54x DSP", Literature Number: SPRU454, Texas Instruments Europe, June 1998.

[24]. "Circular Buffering on TMS320C6000", Literature Number: SPRU645A, April 2001.

[25]. *Developer's Kit for Texas Instruments DSP – for Use with Real – Time Workshop*, The Mathworks.

[26]. *Adaptive Digital Signal Processing* course notes, Dr. S. Krishnan, Ryerson University, Toronto, Canada.

[27]. *DFP Program*, Dr. M. Zeytinoglu, Ryerson University, Toronto, Canada.

[28]. WWW.TI.COM.