7 DIGITAL FILTER REALIZATION

7.7 FFT-BASED REALIZATION OF FIR FILTER

FFT-based FIR filter realization is performed blockwise (as opposed to point by point computation in direct realizations). It requires more memory, but it is more efficient when the order of the filter is at least 18 (as we will see in the next analysis), so it is a serious candidate to consider in some applications.

7.7.1 BASIC DFT FEATURES

In the next parts we denote the DFT operation for a length-N signal by $DFT_N\{x(n)\}$. With this notation we can write $DFT_N\{\ \}$ as

$$X(k) = DFT_N\{x(n)\} = \sum_{n=0}^{N-1} x(n) W_N^{-kn}, \quad 0 \le k \le N-1 \tag{7.1}$$

and inverse DFT $IDF_N\{\ \}$ as

$$x(n) = IDFT_N\{X(k)\} = \frac{1}{N}\sum_{k=0}^{N-1} X(k) W_N^{kn}, \quad 0 \le n \le N-1 \tag{7.2}$$

where

$$W_N = e^{j\frac{2\pi}{N}} \tag{7.3}$$

7.7.1.1 ZERO PADDING

The DFT of a length-N sequence is itself a length-N sequence, so it gives the frequency response of the signal at N points. Suppose we are interested in computing the frequency response at M equally spaced frequency points, where $M > N$. A simple device accomplishes this goal: We add $M - N$ zeros at the tail of the given sequence, thus forming a length-M sequence. The DFT of the new sequence has M frequency points.

We prove that the values of the new DFT are indeed samples of the frequency response of the given signal at M equally spaced frequencies. Denote

$$x_a(n) = \begin{cases} x(n), & 0 \le n \le N-1 \\ 0, & N \le n \le M-1 \end{cases} \tag{7.4}$$

The operation of adding zeros to the tail of a sequence is called **zero padding**. The DFT of the zero-padded sequence $x_a(n)$ is given by

$$X_a(k) = \sum_{n=0}^{M-1} x_a(n) e^{-j\frac{2\pi kn}{M}} = \sum_{n=0}^{N-1} x(n) e^{-j\frac{2\pi kn}{M}} = X\left(\theta(k)\right) \tag{7.5}$$

where

$$\theta(k) = \frac{2\pi k}{M}, \quad 0 \le k \le M-1 \tag{7.6}$$

As we see, $X_a(k)$ is indeed a sampling of $X(\theta)$ at M equally spaced frequency points in the range $\langle 0, 2\pi \rangle$. Figure 1 illustrates the zero-padding operation. Part a) shows a signal of length $N = 8$, part b) shows the magnitude of its length-N DFT (note that we interchange the positive- and negative-index halves). Part c) shows the signal obtained by zero padding to length $M = 32$, and part d) shows the magnitude of the length-M DFT of the zero-padded signal.

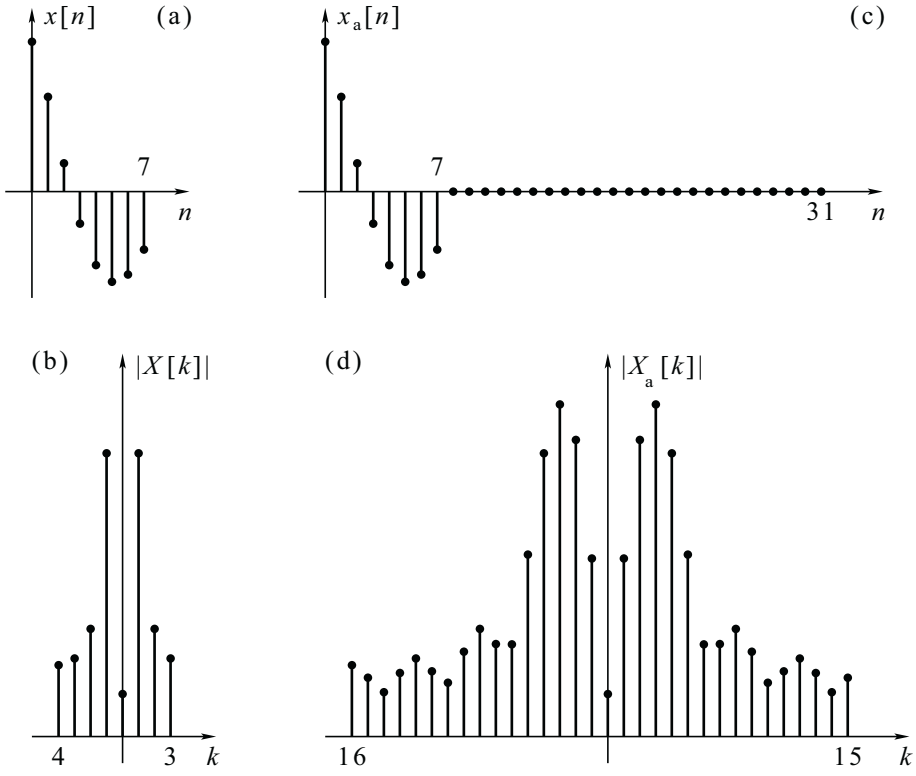

Figure 1 Increasing the DFT length by zero padding

We can interpret the zero-padded DFT $X_a(l)$ as interpolation operation on $X(k)$. Zero padding is typically used for improving the visual continuity of plots frequency

responses. When plotting $X_a(k)$, we typically see more details than when plotting $X(k)$. However, the additional detail do not represent additional information about the signal, since all the information is in the N given the samples of $x(n)$. Indeed, computation of the inverse DFT of $X_a(k)$ gives the zero padded sequence $x_a(n)$, which consists only of the $x(n)$ and zeros.

7.7.1.2 CIRCULAR CONVOLUTION

Let $x(n)$ and $y(n)$ be two finite length sequences, of **equal** length N. We define their **circular convolution** as

$$z(n) = \{x \odot y\}(n) = \sum_{m=0}^{N-1} x(m)\, y\big((n-m) \bmod N\big), \quad 0 \le n \le N-1 \tag{7.7}$$

Other names for this operation are **cyclic convolution** and **periodic convolution**. Another useful way to look at circular convolution is to regard it as partial convolution of the periodic extensions $\{\tilde{x}(n), -\infty < n < \infty\}$, $\{\tilde{y}(n), -\infty < n < \infty\}$ of the two sequences $\{x(n), 0 \le n < N-1\}$, $\{y(n), 0 \le n < N-1\}$. By "partial" we mean that summation is performed only over one period, not from $-\infty$ to ∞.

Circular convolution is, similarly to convolution, a commutative and associative operation, that is

$$x \odot y = y \odot x, \quad (x \odot y) \odot z = x \odot (y \odot z) \tag{7.8}$$

We can now state and prove the convolution and multiplication properties of the DFT.

Convolution theorem

$$z(n) = \{x \odot y\}(n) \Leftrightarrow Z(k) = X(k)Y(k) \tag{7.9}$$

Proof
We have

$$z(n) = \sum_{m=0}^{N-1} x(m)\Big[y\big((n-m) \bmod N\big)\Big] \tag{7.10}$$

Using the linearity and shift property of DFT, we can write

$$Z(k) = \sum_{m=0}^{N-1} x(m) W_N^{-mk} Y(k) = Y(k) \sum_{m=0}^{N-1} x(m) W_N^{-mk} = X(k)Y(k) \tag{7.11}$$

Multiplication theorem

$$z(n) = x(n)\, y(n) \Leftrightarrow Z(k) = \frac{1}{N}\{X \odot Y\}(k) \tag{7.12}$$

The proof of this theorem follows from the duality property of DFT.

We restate the last two theorems, to emphasize their importance[1]:

The DFT of a circular convolution of two sequences is the product of the individual DFTs. The DFT of a product of two sequences is, up to a proportionality constant, the circular convolution of the individual DFTs.

7.7.1.3 LINEAR CONVOLUTION VIA CIRCULAR CONVOLUTION

Suppose we are given two **finite-duration** sequences having different lengths, say $\{x(n), 0 \le n \le N_1 - 1\}$, and $\{y(n), 0 \le n \le N_2 - 1\}$, and we wish to perform their (conventional) discrete-time convolution

$$z(n) = \sum_{m=m_1}^{m_2} x(m)\, y(n-m) \tag{7.13}$$

where $m_1 = \max\{0, n+1-N_2\}$, $m_2 = \min\{N_1 - 1, n\}$.

To avoid confusion, we will henceforth refer to conventional convolution as **linear** convolution.

Let us zero-pad $x(n)$ and $y(n)$ to a length $N = N_1 + N_2 - 1$ and denote the zero-padded sequences by $x_a(n)$, $y_a(n)$, respectively. Now we can express (7.13) as

$$z(n) = \sum_{m=0}^{n} x_a(m)\, y_a(n-m), \quad 0 \le n \le N-1 \tag{7.14}$$

The zero-padded sequences have the same length. Let us compute their circular convolution:

$$\begin{aligned} \{x_a \odot y_a\}(n) &= \sum_{m=0}^{N-1} x_a(m)\left[y_a\left((n-m) \bmod N\right)\right] = \\ &\sum_{m=0}^{n} x_a(m)\, y_a(n-m) + \sum_{m=n+1}^{N-1} x_a(m)\, y_a(n-m+N) \end{aligned} \tag{7.15}$$

In the second sum, the lengths of the two sequences $x(n)$ and $y(n)$ impose the following limits on the summation index m:

$$\begin{aligned} n+1 \le m \le N_1 - 1 \\ N_1 + n \le m \le N + n \end{aligned} \tag{7.16}$$

The intersection of these two limits is empty, so the second term of the right side of (7.15) is zero. The first term is identical to the right side of (7.14), so the conclusion is that

[1] The fact that it is circular, rather than conventional convolution that translates to multiplication in the frequency domain is a common source of confusion and mistakes. The following claim is often made by beginners: "Linear time-invariant filtering is equivalent to multiplication in the frequency domain. Therefore, we can perform linear filtering by computing the DFT of the input signal $X(k)$, multiply by the DFT of the impulse response of the filter $H(k)$ (which is presumably the frequency response of the filter)., and compute the inverse DFT of the product." This claim is wrong, because the DFT is not the frequency response of either the signal or the filter.

$$\{x * y\}(n) = \{x_a \odot y_a\}(n), \quad 0 \le n \le N-1 \tag{7.17}$$

Therefore, we can perform linear convolution of two finite-length sequences by computing the circular convolution of the corresponding zero-padded sequences, provided zero padding is made to the sum of the lengths minus 1. The circular convolution $\{x_a \odot y_a\}$ can be performed by computing DFTs of both sequences, multiplying the resulting vectors point by point, and then computing the inverse DFT of the result. The advantage of performing the convolution this way will become clear when we use the fast Fourier transform (FFT) for speed optimized computation of DFT.

7.7.2 LINEAR CONVOLUTION BY FFT

Now it is known how to perform convolution of two **finite-duration** sequences using circular convolution. The method involves zero padding the two sequences to a length equal to the sum of individual lengths minus 1. Now, equipped with the FFT, we further develop this idea.

Let the two sequences be $\{x(n), 0 \le n \le N_1 - 1\}$ and $\{y(n), 0 \le n \le N_2 - 1\}$. Define N to be the **smallest power of 2** not smaller than $N_1 + N_2 - 1$ and let $x_a(n)$, $y_a(n)$ be the corresponding zero-padded sequences. Then, convolution $\{x * y\}(n)$ can be obtained by performing $\{x_a \odot y_a\}(n)$ and retaining the first $N_1 + N_2 - 1$ elements of the result. Furthermore the convolution property of the DFT, we can obtain the later by the operation

$$z_a(n) = IDFT\{X_a(k) Y_a(k)\} \tag{7.18}$$

The total number of operations, assuming use of a radix-2 FFT, is three times the number of operations for a single N-point FFT (two for the direct DFTs and one for the inverse DFT). This amounts to about $6N \log_2 N$ real multiplications and $9N \log_2 N$ real additions (less if the input sequences are real and we transform them together by $N/2$ complex FFT). We also need N complex multiplications, or $4N$ real multiplications, to compute the product of the FFTs. By comparison, direct computation of the convolution requires about $N_1 N_2$ multiplications and a similar number of additions. Therefore, judging by the number of multiplications, it is preferable to perform the convolution by FFT whenever

$$N_1 N_2 > (N_1 + N_2 - 1)\left[6 \log_2 (N_1 + N_2 - 1) + 4\right] \tag{7.19}$$

In digital signal processing it commonly happens that one of the sequences, say $y(n)$, is known **a priori** and has a **fixed** length N_2, whereas the other sequence $x(n)$, is known only **in real time**, and its length **is not fixed** and is potentially **much larger** than N_2. We concern ourselves with the problem of computing the linear convolution $x * y$ under these conditions. Using zero padding is possible, but may not be advisable, for two reasons:

1. The sequence $y(n)$ will $x(n)$ have to be padded by many zeros, resulting in many unnecessary computations
2. The DFT will have to be performed on very long sequences, which may be inconvenient or impossible.

A better approach is to split the long sequence to segments, each of the length N_1 which is of the same order as N_2, convolve $y(n)$ with each of the segments separately, and properly add up the partial results to form the desired convolution $x * y$. This method is known as **overlap-add convolution**. We now describe its details.

Let us write sequence $x(n)$ as

$$x(n) = \sum_i x_i(n) \tag{7.20}$$

where

$$x_i(n) = \begin{cases} x(n), & N_1 i \le n \le N_1(i+1) - 1 \\ 0, & \text{otherwise} \end{cases} \tag{7.21}$$

We have left range of the index i unspecified on purpose, to emphasize that the length of $x(n)$ need not be known in advance. If this length is not an integer multiple of N_1, we pad $x(n)$ with zeros to make it so. By linearity of the convolution operator we have

$$z(n) = \{x * y\}(n) = \sum_i \{x_i * y\}(n) \tag{7.22}$$

The convolution $\{x_i * y\}$ has length $N_1 + N_2 - 1$, and it is nonzero for the range of time points

$$N_1 i \le n \le N_1 i + N_1 + N_2 - 2 \tag{7.23}$$

We can write $\{x_i * y\}$ as a sum of two sequences, say

$$z_i(n) = \{x_i * y\}(n) = u_i(n) + v_i(n) \tag{7.24}$$

where $u_i(n)$ is nonzero in the range

$$N_1 i \le n \le N_1(i+1) - 1 \tag{7.25}$$

and $v_i(n)$ is nonzero in the range

$$N_1(i+1) \le n \le N_1(i+1) + N_2 - 2 \tag{7.26}$$

The range of definition of $u_i(n)$ coincides with that of $x_i(n)$. On the other hand, the range of definition of $v_i(n)$ coincides with the initial part of the range of $x_{i+1}(n)$. Therefore, the proper sequence of operations, implied by (7.22) and (7.24), is

1. Zero-pad $x_i(n)$ and $y(n)$ to a length $N_1 + N_2 - 1$.
2. Perform the circular convolution $\{x_i \odot y\}$, which is equal to the linear convolution $\{x_i * y\}$. The circular convolution is performed by FFT, as explained earlier.
3. Use $u_i(n)$ as a partial result for the ith stage. Add to its initial part the sequence $v_{i-1}(n)$ from the previous stage. Save $v_i(n)$ for the $(i+1)$ stage.

Figure 2 illustrates the various sequences, their proper timings, and the way they are combined. The signal $y(n)$ in this example has length $N_2 = 4$. The length chosen for the sequences $x_i(n)$ is $N_1 = 5$. Accordingly, the lengths of $z_i(n)$, $u_i(n)$ and $v_i(n)$ are 8, 5, and 3, respectively. The reason for the name overlap-add[2] should be now clear.

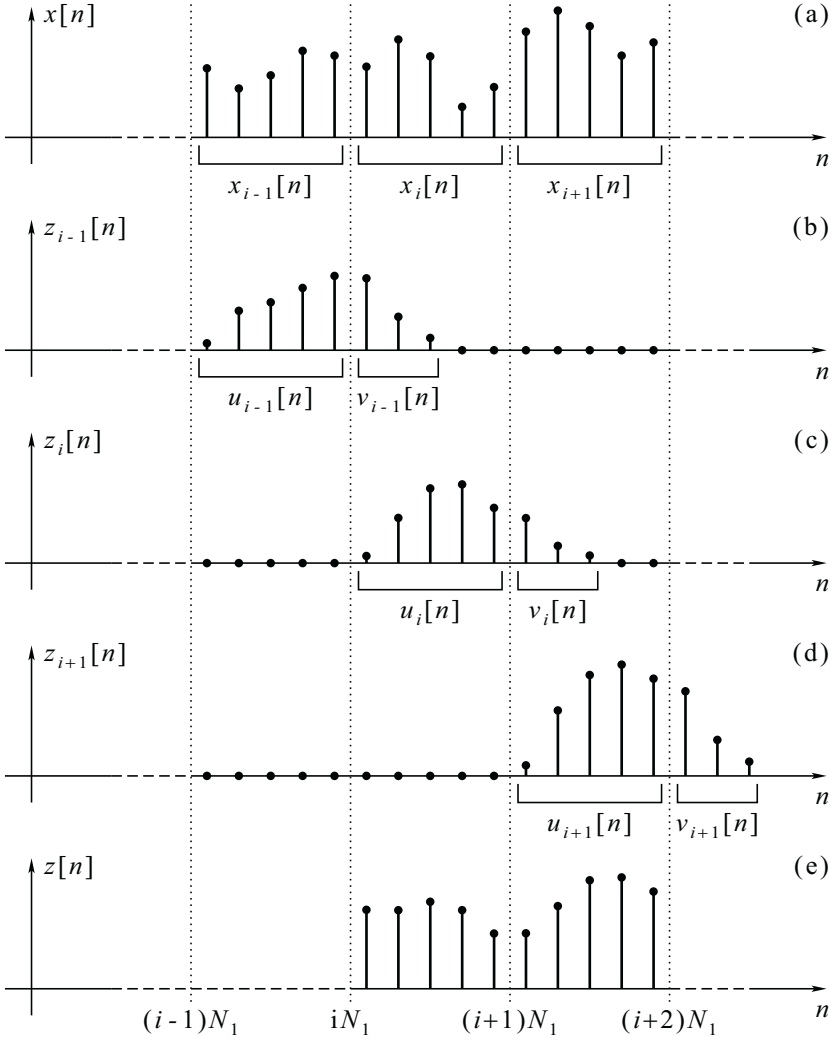

Figure 2 Illustration of overlap-add convolution

The procedure **ola** in Appendix is a Matlab implementation of the overlap-add method. The length of the sequence x is assumed to be much greater than the length of y. The FFT of y is computed first and used throughout. Then the program loops over the segments, and finally handles the tail of the sequence x, which may be shorter than the other segments.

We now discuss the optimal choice of the FFT length N in the overlap-add method. We assume that the sequences in question are real. This enables the simultaneous computation of the FFTs of two consecutive segments. Since we have the freedom to choose N_1, let us choose it such that $N_1 + N_2 - 1$ is a power of 2. The FFT of the zero padded $y(n)$ needs to be computed only once, so we ignore the operations it

[2] Linear convolution of a long sequence $x(n)$ by a fixed-length sequence $y(n)$ can also be performed by a method called **overlap-save**, which is dual to overlap-add.

requires. For each pair of segments of N_1 output points we need $4(N_1+N_2-1)\log_2(N_1+N_2-1)$ multiplications for the two FFTs (a direct one and an inverse one), as well as $4(N_1+N_2-1)$ multiplications for the product of the FFTs. It is convenient, in this application, to count the number of operations per sample of the signal $x(n)$. Using the overlap-add method for convolution is more efficient than direct convolution when the number of operations per sample is smaller. We thus get the criterion

$$2\left(1+\frac{N_2-1}{N_1}\right)\left[1+\log_2(N_1+N_2-1)\right] < N_2 \tag{7.27}$$

Table 1 shows the optimal choice of the parameter N_1+N_2-1 for various values of N_2. The optimum is defined as the value for which the left side of (7.27) is minimal, under the constraint that this parameter is an integer power of 2. For $N_2 < 19$ the inequality (7.27) does not hold, meaning that direct convolution is more efficient than overlap-add.

Table 1 Optimal choice of the segment length for overlap-add method

Range of N_2	Optimal N_1+N_2-1
19-26	128
27-47	256
48-86	512
87-158	1024

7.8 ROBUST DIGITAL FILTER STRUCTURES

Digital filters can be implemented in several ways. The implementation could be a simulation on a general purpose computer, or it could be a program running on a commercial digital signal processing chip (DSP), or it could be a dedicated piece of hardware, for example, a VLSI chip. In any case, the resources such as computational units, time, memory, and chip area are finite. One consequence of this fact is that the external and internal signals involved and the filter coefficients are represented by binary words of finite length. This causes three kinds of errors in the output of the filter:

1. There is coefficient quantization, the effect of which is cause errors in the transfer function being realized. This is a deterministic type of error, and its effect can be evaluated ahead of time.
2. Error due to quantization of signals, particularly the internal state variables (and often nonstate variables). One component of this is a random type of error (called **roundoff noise**) and should accordingly be characterized by stochastic models.
3. The last component is a highly correlated type of error, occurring in the form of periodic oscillations called **limit cycles**. These oscillations can in turn be classified as either **granular** type or **overflow** type. The former are usually of small amplitude and are significant only when the signal level is low, whereas overflow oscillations are very large disturbances

There exist an infinite number of structures to realize a digital filter. The multipliers that appear in the structure are said to be the coefficients of the structure. These structures have different properties when implemented in real hardware. E.g. actual frequency response of the direct form IIR structure is very sensitive to quantization of coefficients, particularly for large filter order N and sharp-cutoff filters.

There are other structures that are less sensitive (i.e., more **robust** to coefficient quantization) and are generally called **low-sensitivity structures**. The importance of finite wordlength effects should not be overlooked. For example, with a commercial 16-bit fixed-point DSP chip, roundoff noise effects can be quite significant for sharp-cutoff IIR filters. In this chapter we present only two such structures. The first is a general overview of State-Space approach and the second is low-sensitivity IIR designs based on structural passivity (parallel connection of two allpass filters). Other techniques [2] e.g. structures with error feedback, wave digital filters are beyond the scope of this subject.

7.8.1 STATE-SPACE APPROACH FOR LOW-NOISE DESIGNS

A powerful tool in the understanding and minimization of finite wordlength effects in digital filters is based on the state-space formalism. This approach can be used to design filters with minimum noise, filters free from limit cycles and often low-sensitivity filters.

Consider again IIR transfer function

$$H(z) = \frac{\sum_{k=0}^{N-1} b(k) z^{-k}}{1 + \sum_{k=1}^{N-1} a(k) z^{-k}} \tag{7.28}$$

A difference equation expresses the present output of the system in terms of past outputs, and present and past inputs. The delay elements in the direct realizations represent the **memory** of the system, in the sense that their inputs must be stored and remembered from one time point to the next. Until now, the outputs of the delay elements were of no interest to us by themselves, only as auxiliary variables.

In this section we study a different way of representing rational LTI systems. In this representation, the output of the delay elements play a central role. Collected together, they are called the **state vector** of the system. Correspondingly, the representations we are going to present are called **state-space representations**. A state-space representation comprises two equations: The **state equation** expresses the time evolution of the state vector as a function of its own past and the input signal, the **output equation** expresses the output signal as a function of the state vector and the input signal.

To motivate the state-space concept, consider again the direct realization shown in Figure 3, introduce the notation

$$s_1(n) = u(n-1), \quad s_2(n) = u(n-2), \quad s_3(n) = u(n-3) \tag{7.29}$$

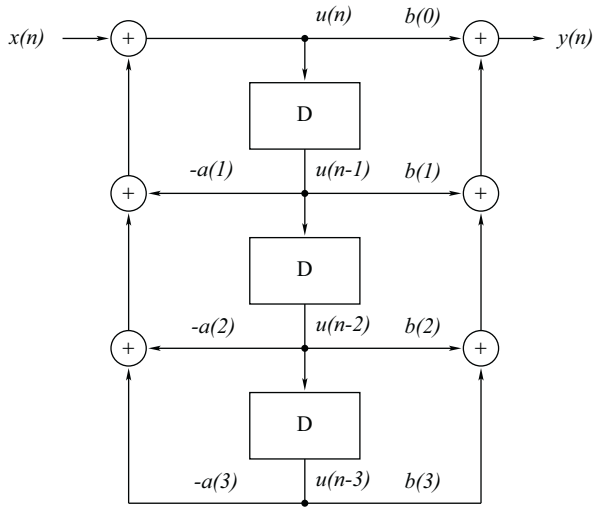

Figure 3 Direct realization (direct form II) of a digital IIR system

In other words, the signal $s_k(n)$ is the output of the k th delay element (starting from the top) at time n. Then we can read the following relationship directly from the figure:

$$\begin{gathered} s_1(n+1) = x(n) - a(1)s_1(n) - a(2)s_2(n) - a(3)s_3(n) \\ s_2(n+1) = s_1(n) \\ s_3(n+1) = s_2(n) \\ y(n) = b(0)x(n) + \left[b(1) - b(0)a(1)\right]s_1(n) + \left[b(2) - b(0)a(2)\right]s_2(n) + \left[b(3) - b(0)a(3)\right]s_3(n) \end{gathered}$$

A compact way of writing is

$$\begin{bmatrix} s_1(n+1) \\ s_2(n+1) \\ s_3(n+1) \end{bmatrix} = \begin{bmatrix} -a(1) & -a(2) & -a(3) \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} s_1(n) \\ s_2(n) \\ s_3(n) \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} x(n) \tag{7.30}$$

$$y(n) = \begin{bmatrix} c(1) & c(2) & c(3) \end{bmatrix} \begin{bmatrix} s_1(n) \\ s_2(n) \\ s_3(n) \end{bmatrix} + b(0)x(n) \tag{7.31}$$

where

$$c(k) = b(k) - b(0)a(k), \quad 1 \le k \le 3 \tag{7.32}$$

These equations can be easily extended to any order N by using matrix notation:

$$\mathbf{s}(n+1) = \mathbf{A}\mathbf{s}(n) + \mathbf{B}x(n) \tag{7.33}$$

$$y(n) = \mathbf{C}\mathbf{s}(n) + Dx(n) \tag{7.34}$$

where $\mathbf{A}$ is $N \times N$ matrix, $\mathbf{B}$ is $N \times 1$ matrix, $\mathbf{C}$ is $1 \times N$ and D is scalar quantity. These matrices are called, respectively, the **state**, **input**, **output** and direct **transmission** matrices. It is clear from the preceding construction that every difference equation has a corresponding state-space representation. The procedure **tf2ss** in the Appendix computes these matrices. The transfer function $H(z)$ is related to the state-space-space parameters by (can be also computed by **ss2tf** procedure in the Appendix)

$$H(z) = D + \mathbf{C}(z\mathbf{I} - \mathbf{A})^{-1}\mathbf{B} \tag{7.35}$$

whereas the impulse response corresponding to $H(z)$ is given by

$$h(n) = \begin{cases} D & \text{for } n = 0 \\ \mathbf{C}\mathbf{A}^{n-1}\mathbf{B} & \text{for } n > 0 \end{cases} \tag{7.36}$$

If a digital filter structure is such that the elements $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$, D of the state space description are also the multiplier coefficients in the structure, then it is called[3] a **state-space implementation**, **state-space realization**, or a **state-space structure**. We can show that the eigenvalues of $\mathbf{A}$ are the poles of the transfer function $H(z)$. Accordingly, a minimal system is stable if and only if $\mathbf{A}$ has all eigenvalues λ_k satisfying $|\lambda_k| < 1$.

Given a state-space description $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$, D, suppose we replace these matrices with $\tilde{\mathbf{A}}$, $\tilde{\mathbf{B}}$, $\tilde{\mathbf{C}}$, $\tilde{D}$, where

$$\tilde{\mathbf{A}} = \mathbf{P}^{-1}\mathbf{A}\mathbf{P}, \quad \tilde{\mathbf{B}} = \mathbf{P}^{-1}\mathbf{B}, \quad \tilde{\mathbf{C}} = \mathbf{C}\mathbf{P}, \quad \tilde{D} = D \tag{7.37}$$

and where $\mathbf{P}$ is any $N \times N$ nonsingular matrix. This leaves the transfer function unchanged. Accordingly, we can derive an equivalent description by using the transformation (7.37), which is called the **similarity transformation**. For a given transfer function, there exist an infinite number of state-space realizations, since $\mathbf{P}$ is arbitrary. However, **some of these realizations have better SNR than others**. Accordingly, it is possible to find the best state-space structure by minimizing the roundoff noise variance at the output (for a given signal level) under scaled conditions. This is the main advantage of the state-space approach – it offers a convenient mathematical framework to minimize a performance measure.

It should be noted that a state-space realization (i.e., a realization of $H(z)$ with multiplier coefficients $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$, D) requires a total of $(N+1)^2$ multipliers, which, for large N, is far in excess of the number of multipliers in a direct form realization. Accordingly, minimum noise state-space realizations are usually restricted to second-order sections, which are then used in a cascade to produce higher order filters.

[3] Note that the direct form is not a state-space implementation, since the elements $\mathbf{C}$ are not the multipliers in the Figure 3.

7.8.2 LOW-SENSITIVITY IIR DESIGNS BASED ON STRUCTURAL PASSIVITY

Perhaps the earliest type of filters known to engineers are passive filters, made of analog electrical components such as inductors, capacitors, and resistors. These filters, if appropriately designed, lead to filters with **low pasband sensitivity**. These features can be applied also to digital filters based on structural passivity losslessness.

7.8.2.1 BASIC REQUIREMENTS FOR LOW SENSITIVITY

Consider a digital filter structure with multiplier coefficients m_i, implementing a transfer function $G(z)$. Assume that m_i are restricted to be in a well-defined range $\mathbb{R}$ (e.g., such as $-1 < m_i < 1$). When m_i are quantized, the response changes in a way depending on the structure. Suppose the structure is such that, as long as the m_i belong to the permissible range $\mathbb{R}$, $\left|G\left(e^{j\omega}\right)\right|$ never exceeds unity. At the frequency ω_k where $\left|G\left(e^{j\omega}\right)\right| = 1$ under ideal conditions, the response can therefore only decrease, no matter how the multiplier coefficient m_i changes. An implementation satisfying the property $\left|G\left(e^{j\omega}\right)\right| \le 1$ for all ω regardless of the values of m_i in a range $\mathbb{R}$ is said to be **structurally bounded** or **structurally passive** in the range $\mathbb{R}$. The special case where a realization is such that $\left|G\left(e^{j\omega}\right)\right| = 1$ for all ω (regardless of values of m_i) is useful in implementing **allpass filters**, which remain allpass despite quantization of m_i. Such an implementation is said to be **structurally lossless**.

A stable transfer function $G(z)$ satisfying $\left|G\left(e^{j\omega}\right)\right| \le 1$ for all ω is said to be **bounded** or **passive**. The term "passive" is motivated by the fact that for such systems, the output energy is less than or equal to the energy of the input sequence. If, in addition, the impulse response of $G(z)$ is real (so that $G(z)$ is real for all real z) we say that $G(z)$ is **bounded real**. Stable allpass transfer functions (which satisfy $\left|G\left(e^{j\omega}\right)\right| = 1$) are also called **lossless** function. For such transfer functions, the energy of the output sequence is equal to that of the input sequence, for every finite energy input. If a lossless function has real impulse response it is said to be **lossless bounded real**. Any stable transfer function can be scaled so that it becomes bouded.

From the above discussion we conclude this: if a bounded transfer function is such that $\left|G\left(e^{j\omega}\right)\right|$ is exactly equal to unity for some passband frequencies, and if it is implemented in a **structurally passive manner**, the resulting system has **low pasband sensitivity**.

7.8.2.2 STRUCTURES BASED ON TWO ALLPASS FUNCTIONS

Suppose $G(z)$ is a transfer function of the form

$$G(z) = \frac{1}{2}\left[A_0(z) + A_1(z)\right] \tag{7.38}$$

where $A_0(z)$ and $A_1(z)$ are stable allpass functions with frequency responses

$$A_0\left(e^{j\omega}\right) = e^{j\phi_0(\omega)}, \quad A_1\left(e^{j\omega}\right) = e^{j\phi_1(\omega)} \tag{7.39}$$

with $\phi_0(\omega)$ and $\phi_1(\omega)$ denoting the phase responses. Clearly, $\left|G\left(e^{j\omega}\right)\right| \le 1$, with equality when $\phi_0(\omega) = \phi_1(\omega) + 2\pi k$ for any integer k. If the allpass functions are implemented such that they remain (stable and) allpass despite coefficient quantization,

then $G(z)$ remains bounded despite quantization. This leads to low passband sensitivity. At the pasband extrema ω_k, where $\left|G\left(e^{j\omega}\right)\right| = 1$, the phases of $A_0\left(e^{j\omega}\right)$ and $A_1\left(e^{j\omega}\right)$ are aligned, whereas at the transmission zeros (in the stopband of $G(z)$), the phases differ by (odd integer multiplies of) π.

It can be shown that classical Butterworth, Chebyshev, and elliptic digital filters (and in fact a much wider class of filters) can be represented in the form (7.38). For lowpass and highpass filters, if the order N is odd, the orders of $A_1(z)$ and $A_0(z)$ are r and $N-r$, respectively, for an appropriate integer r, and these allpass functions have real coefficients. If N is even, then $A_0(z)$ has order $N/2$ and has complex coefficients, and $A_1(z)$ is obtained from $A_0(z)$ by conjugation of coefficients. Accordingly, for even N, the output of $G(z)$ in response to a real input sequence is equal to the real part of the output of $A_0(z)$ in response to the same input.

Example
As an example, consider the third-order elliptic transfer function

$$G(z) = \frac{0.23179 + 0.36021z^{-1} + 0.36021z^{-2} + 0.23179z^{-3}}{1 - 0.38409z^{-1} + 0.70390z^{-2} - 0.13581z^{-3}}$$

It can be verified that this transfer function is decomposable as

$$G(z) = 0.5\left[\frac{-0.20356 + z^{-1}}{1 - 0.20356z^{-1}} + \frac{0.66715 - 0.18053z^{-1} + z^{-2}}{1 - 0.18053z^{-1} + 0.66715z^{-2}}\right]$$

which is a sum of two allpass functions.

APPENDIX - MATLAB PROGRAMS

```
function z = ola(x,y,N);
% Synopsis: z = ola(x,y,N).
% Computes the convolution z = x*y by the overlap-add method.
% Input parameters:
% x: the long input sequence
% y: the short  input sequence
% N: length of the FFT.
% Output parameters:
% z: the output sequence.

N2 = length(y); lx = length(x); y = reshape(y,1,N2);
x = reshape(x,1,lx); z = zeros(1,N2+lx-1);
lz = length(z); rflag = 0; N1 = N-N2+1;
if (any(imag(x)) | any(imag(y))), rflag = 1; end
nframe = floor(lx/N1); ltail = lx - N1*nframe;
Y = fft([y, zeros(1,N-N2)]);
for k = 1:nframe,
  nst = (k-1)*N1;
  temp = ifft(fft([x(1,nst+1:nst+N1), zeros(1,N-N1)]).*Y);
  if (rflag), temp = real(temp); end
  z(1,nst+1:nst+N) = z(1,nst+1:nst+N) + temp;
end
if (ltail > 0),
  nst = nframe*N1+1; temp = [x(1,nst:lx), zeros(1,N-ltail)];
  temp = ifft(fft(temp).*Y);
  if (rflag), temp = real(temp); end
  z(1,nst:lz) = z(1,nst:lz) + temp(1,1:N2+ltail-1);
end
```

```
function [A,B,C,D] = tf2ss(b,a);
% Synopsis: [A,B,C,D] = tf2ss(b,a).
% Converts a transfer function to direct state-space realization.
% Inputs:
% b, a: the numerator and denominator polynomials.
% Outputs:
% A, B, C, D: the state-space matrices

p = length(a)-1; q = length(b)-1; N = max(p,q);
if (N > p), a = [a,zeros(1,N-p)]; end
if (N > q), b = [b,zeros(1,N-q)]; end
A = [-a(2:N+1); [eye(N-1), zeros(N-1,1)]];
B = [1; zeros(N-1,1)];
C = b(2:N+1) - b(1)*a(2:N+1);
D = b(1);
```

```
function [b,a] = ss2tf(A,B,C,D);
% Synopsis: [b,a] = tf2ss(A,B,C,D).
% Converts a state-space realization to a transfer function.
% Inputs:
% A, B, C, D: the state-space matrices
% Outputs:
% b, a: the numerator and denominator polynomials.

a = poly(A); N = length(a)-1; h = zeros(1,N+1); h(1) = D; tmp = B;
for i = 1:N, h(i+1) = C*tmp; tmp = A*tmp; end
b = a*toeplitz([h(1);zeros(N,1)],h);
```

LITERATURE

[1] Porat, B: *A Course in Digital Signal Processing*. John Wiley & Sons, Inc. New York 1997, ISBN 0-471-14961-7.

[2] Mitra, S.K. – Kaiser, S.K.: Handbook for Digital Signal Processing. John Wiley & Sons, inc., New York, 1993, ISBN 0-471-61995-7.