7 DIGITAL FILTER REALIZATION

7.1 INTRODUCTION

In the preceding lessons we learned how to design Linear Time Invariant (LTI) digital filters, both IIR and FIR. The end result of the design was the transfer function $H(z)$ of the filter or, equivalently, the difference equation it represents. We thus far looked at a filter as a black box, whose input-output relationships are well defined, but whose internal structure is ignored. Now it is time to look more closely at possible internal structures of digital filters, and to learn how to build such filters. It is convenient to break the task of building a digital filter into two stages:

1. Construction of a block diagram of the filter. Such a block diagram is called a **realization** of the filter. Realization of a filter at a block-diagram level is essentially a flow graph of the signals in the filter. It includes operations such as delays, additions, and multiplications of signals by constant coefficients. It ignores ordering of operations, accuracy, scaling and the like. A given filter can be realized in infinitely many ways. Different realizations differ in their properties, and some are better than others.
2. **Implementation** of the realization, either in hardware or in software. At this stage we must concern ourselves with problems neglected during the realization stage: order of operations, signal scaling, accuracy of signal values, accuracy of coefficients, accuracy of arithmetic operations. We must analyze the effect of such imperfections on the performance of the filter. Finally, we must build the filter – either the hardware or the program code (or both, if the filter is a specialized combination of hardware and software).

In this lesson we cover the most common filter realizations. We describe each realization by its block diagram and by a representative MATLAB code.

7.2 BASIC BUILDING BLOCKS OF DIGITAL FILTERS

Any digital system that is linear, time invariant, rational and causal can be realized using three basic types of element:

Unit delay
The purpose of this element is to hold its input for a unit of time (physically equal to the sampling interval T) before it is delivered to the output. Mathematically, it performs the operation

$$y(n) = x(n-1) \tag{7.1}$$

Unit delay is depicted schematically in Figure 1a. The letter "D," indicating delay, sometimes is replaced by z^{-1}, which is the delay operator in the Z domain. Unit delay can be implemented in hardware by a data register, which moves its input to the output when clocked. In software, it is implemented by a storage variable, which changes its value when instructed by the program.

Adder
The purpose of this element is to add two or more signals appearing at the input at a specified time. Mathematically, it performs the operation

$$y(n) = x_1(n) + x_2(n) + \ldots \tag{7.2}$$

An adder is depicted schematically in Figure 1b.

Multiplier
The purpose of this element is to multiply a signal (a varying quantity) by a constant number. Mathematically,

$$y(n) = a\,x(n) \tag{7.3}$$

A multiplier is depicted schematically in Figure 1c. We do not use a special graphical symbol for it, but simply put the constant factor above (or beside) the signal line. A standard[1] physical multiplier (in hardware or software) can multiply two signal equally easily, but such an operation is not needed in LTI filters.

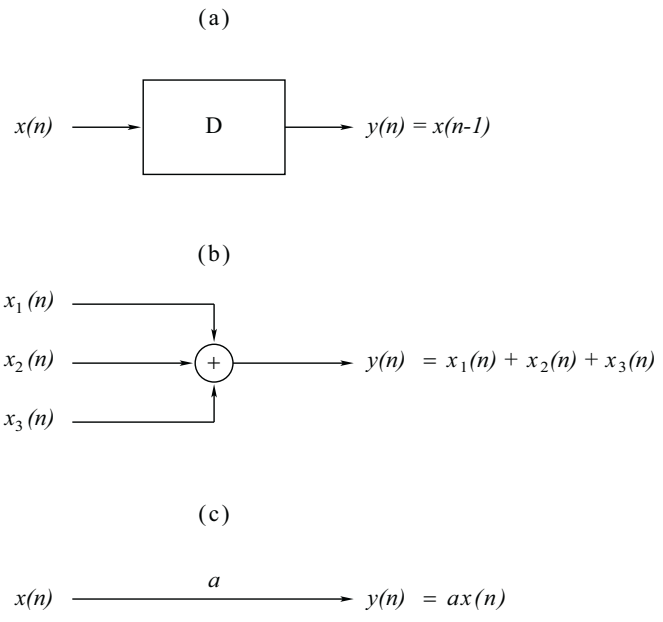

Figure 1 Basic building blocks for digital filter realizations: a) unit delay; b) adder; c) multiplier by a constant

[1] Simpler modified multiplier can also perform multiplication by a constant factor. Implementation of such multiplier can be much simpler and it is used in special ASIC chips.

Example 1 *Consider the first-order FIR filter with the transfer function* $H_{FIR}(z)=b(0)+b(1)z^{-1}$. *Figure 2a shows a realization of this FIR filter using one delay element, two multipliers, and one adder. The input to the delay element at a time n is* $x(n)$, *and its output is then* $x(n-1)$. *The output of the realization is therefore* $y(n)=b(0)x(n)+b(1)x(n-1)$ *and this is exactly time-domain expression for* $H_{FIR}(z)$.

Example 2 *Consider the first-order IIR filter with the transfer function* $H_{IIR}(z)=1/(1+a(1)z^{-1})$. *Figure 2b shows a realization of this IIR filter using one delay element, one multiplier, and one adder. The input to the delay element at a time n is then* $y(n)$, *and its output is then* $y(n-1)$. *The output of the realization is therefore* $y(n)=-a(1)y(n-1)+x(n)$ *and this is exactly time-domain expression for* $H_{IIR}(z)$. *This realization is* ***recursive****. It builds the present value of* $y(n)$ *from its own past values and the input signal. Put another way, the realization uses* ***feedback****.*

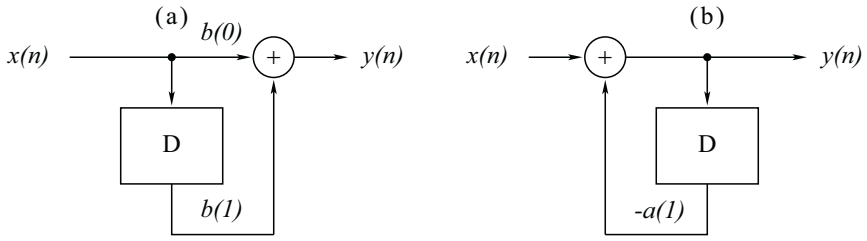

Figure 2 Realizations of first-order filters: a) FIR; b) IIR

7.3 DIRECT REALIZATIONS OF IIR FILTERS

Let $H(z)$ be a rational, causal, stable transfer function. We assume, for convenience, that the orders of the numerator and denominator polynomials of the filter transfer function are equal[2]. Thus $H(z)$ is given by

$$\frac{Y(z)}{X(z)}=H(z)=\frac{B(z)}{A(z)}=\frac{\sum_{k=0}^{N}b(k)z^{-k}}{1+\sum_{k=1}^{N}a(k)z^{-k}} \tag{7.4}$$

7.3.1 DIRECT FORMS I AND II

Let us introduce an auxiliary signal $u(n)$, related to the input and output signals $x(n)$ and $y(n)$ through

$$u(n)=-a(1)u(n-1)-\ldots-a(N)u(n-N)+x(n) \tag{7.5}$$

[2] There is no loss of generality in this assumption since it is always possible to extend numerator or denominator polynomials by adding zero-valued coefficients.

$$y(n) = b(0)u(n) + b(1)u(n-1) + \ldots + b(N)u(n-N) \tag{7.6}$$

or, in the Z domain,

$$U(z) = \frac{1}{A(z)} X(z), \quad Y(z) = B(z)U(z) \tag{7.7}$$

Figure 3 shows a realization of (7.5) using the three types of building block (in the figure we have used $N = 3$ as an example). By passing $u(n)$ through a chain of N delay elements, we get the signals $\{u(n-1), u(n-2), \ldots, u(n-N)\}$. Then we can form $u(n)$ as a linear combination of the delayed signals, plus the input signal $x(n)$, as is expressed in (7.5). We need N multipliers for the coefficients $\{-a(1), \ldots, -a(N)\}$, and N two-input adders to form the sum. The realization uses feedback.

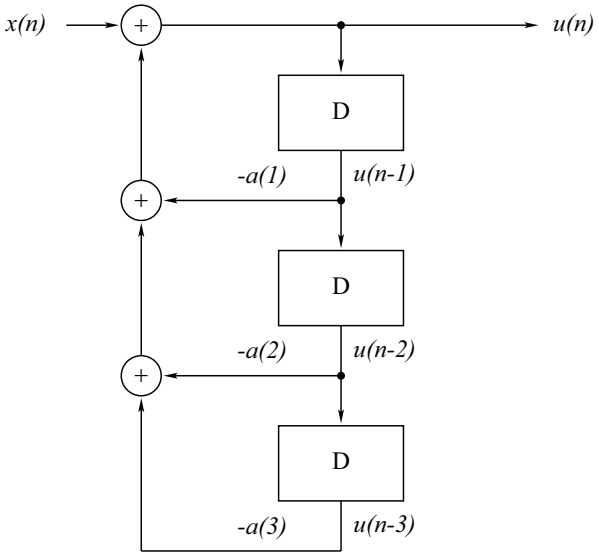

Figure 3 Realization of the auxiliary signal $u(n)$

We can now use (7.6) for generating the output signal $y(n)$ from the auxiliary signal $u(n)$ and its delayed values. We do this by augmenting Figure 3 with $N+1$ multipliers for the coefficients $\{b(0), \ldots, b(N)\}$ and N adders. This results in the realization (called **direct form II**) shown in Figure 4. Note that it is **not necessary to increase** the number of delay elements and such realization is minimal (**canonical**) realization of IIR system.

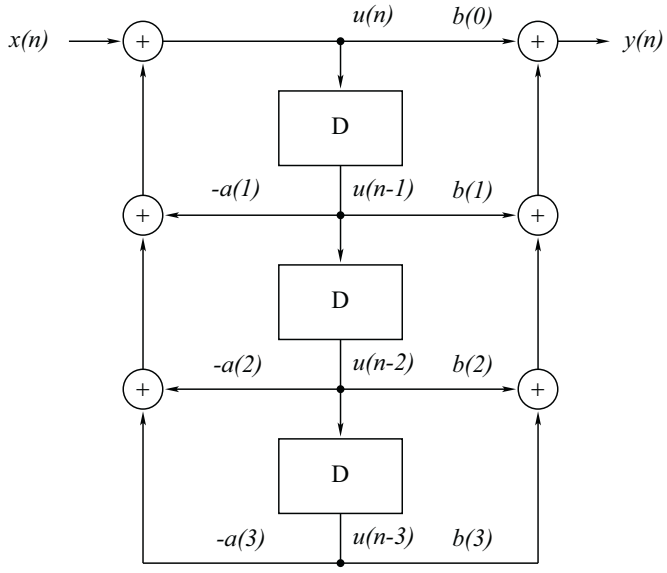

Figure 4 Direct realization (direct form II) of a digital IIR system

It has the following properties:

1. The number of delay elements N is the maximum of the orders of polynomials $A(z)$ and $B(z)$. Assuming that these polynomials have no common factor, this is the **minimum possible number of delays** in any realization of $H(z)$. The set of values at the outputs of the delay elements is called the **state** of the system.
2. There are $2N+1$ multipliers and $2N$ adders. However, since some coefficients may be zero, there **may be a smaller** number of adders and multipliers.
3. The realization is **recursive**, since it generates present value from past values of these signals. It can be shown that any realization of an IIR system must be recursive if it is required to include only a finite number of delay elements.
4. Assuming that the input signal $x(n)$ is causal. It is necessary **to initialize the state** components before feeding the input signal to the system. The state is usually initialized to zero. However, initialization to nonzero values is sometimes required, depending on the application.

Another direct realization (**direct form I**) of N-th order IIR filter can be constructed using $2N$ delay elements[3], N for the input signal $x(n)$, and N for the output signal $y(n)$. It is based on the auxiliary signal $v(n)$, related to the input and output signals $x(n)$ and $y(n)$ through

$$v(n) = b(0)x(n) + b(1)x(n-1) + \ldots + b(N)x(n-N) \tag{7.8}$$

$$y(n) = -a(1)y(n-1) - \ldots - a(N)y(n-N) + v(n) \tag{7.9}$$

or, in the Z domain,

[3] It is non-minimal structure since it uses more than N delay elements.

$$V(z) = B(z)X(z), \quad Y(z) = \frac{1}{A(z)}V(z) \tag{7.10}$$

and it is shown in Figure 5.

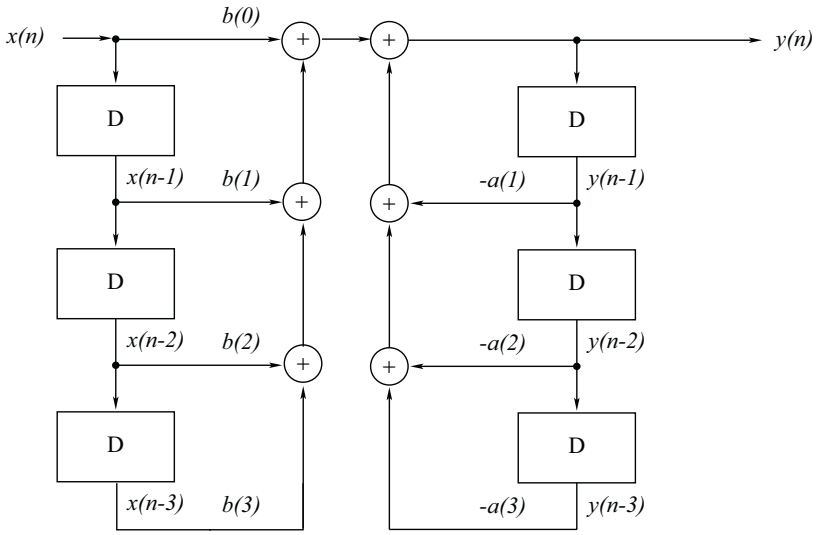

Figure 5 Direct realization (direct form I) of a digital IIR system

7.3.2 TRANSPOSED DIRECT FORM II

We now explore an alternative to the direct form realizations. Figure 6 shows a realization of (7.9) using the three types of building block (in the figure we again use $N = 3$ as an example). The present value of $y(n)$ is built from its own past values and the auxiliary signal $v(n)$. To generate $y(n-N)$, we need N delay elements. These elements can be used for generating all intermediate delays, as shown in the figure. This realization effectively computes (7.9) in the Z domain form

$$Y(z) = \left(\ldots\left(-a(N)z^{-1} - a(N-1)\right)z^{-1} - \ldots - a(1)\right)z^{-1}Y(z) + V(z) \tag{7.11}$$

The state of this realization does not consist of pure delays of a single signal, but of linear combinations of different delays. As before, we need N multipliers for the coefficients $\{-a(1),\ldots,-a(N)\}$ and N two-input adders.

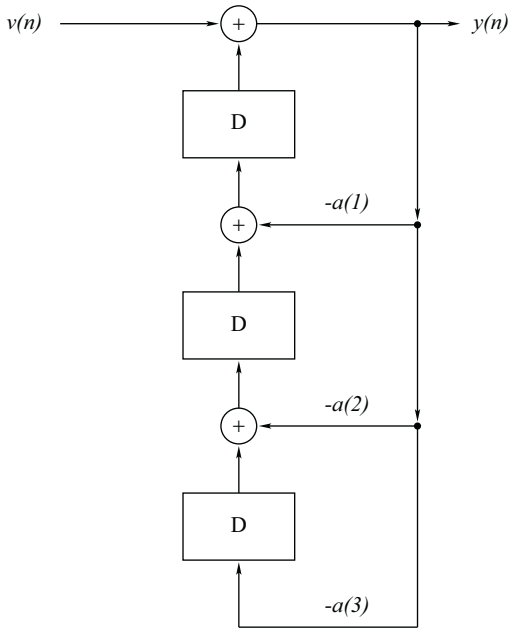

Figure 6 Realization of $y(n)$ from the auxiliary signal $v(n)$

We can now use (7.8) for generating the auxiliary signal $v(n)$ from the input signal $x(n)$ and its delayed values. We do this by augmenting Figure 6 with $N+1$ multipliers for the coefficients $\{b(0),\ldots,b(N)\}$ and N adders. This results in the realization shown in Figure 7. Note that it is **not necessary to increase** the number of delay elements[4], since the existing elements can include of the necessary delay of the input signal.

The realization shown in Figure 7 is known as a transposed direct realization (or **transposed direct form II**), for reasons explained next. The transposed direct form II shares the main properties of the direct form II. In particular, it has the same number of delays, multipliers, and two-input adders[5]. However, the two realizations **have different states[6]**.

[4] This is also canonical realization of IIR filter.

[5] Note that, in Figure 7, there are $N-1$ three-input adders and 2 two-input adders, which are equivalent to $2N$ two-input adders.

[6] As long as the state is initialized to zero, this difference is iconsequential. However, when initialization to a nonzero state is necessary, the two realizations require different computations of the initial state.

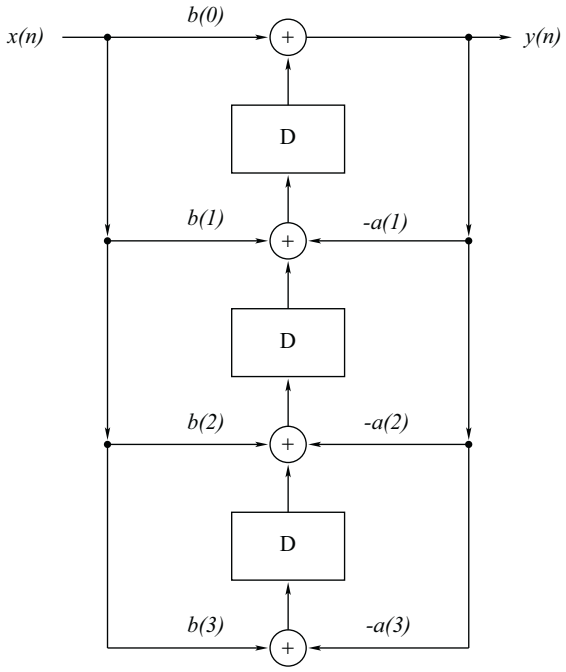

Figure 7 Transposed direct form II realization of a digital IIR filter

Comparison of Figures 4 and 7 reveals that the later can be obtained from the former by the following sequence of operations:

1. Reversal of the signal flow direction in all lines (i.e., reversal of all arrows).
2. Replacing all adders by contact points, and all contact points by adders.
3. Interchange the input and output.

This sequence of operation is called **transposition** of the realization. A known theorem in the network theory states that transposition of a given realization leaves the transfer function of the realization invariant. Transposition of the transposed realization obviously gives back the original realization. Two such realizations are said to be **dual** to each other.

The procedure **direct** in the Appendix implements the two direct realizations of an IIR filter. The MATLAB function **filter** performs the same computation more efficiently, since it is coded internally. Therefore, this program is intended for educational purposes, rather than for serious use.

7.4 DIRECT REALIZATIONS OF FIR FILTERS

Realizations of digital FIR filters can be obtained from IIR filter realizations by specializing these realizations to the case $A(z)=1$. However since FIR filters are usually either symmetric or anti-symmetric, we can save about half the number of multiplications by exploiting symmetry.

For an FIR filter with odd number of coefficients M [7] we can write the filter's output as

$$y(n)=h\left(\frac{M-1}{2}\right)x\left(n-\frac{M-1}{2}\right)+\sum_{k=0}^{\left(\frac{M-1}{2}\right)-1}h(k)\left[x(n-k)\pm x(n-M+k+1)\right] \quad (7.12)$$

whereas for an even number of coefficients we put

$$y(n)=\sum_{k=0}^{\left(\frac{M}{2}\right)-1}h(k)\left[x(n-k)\pm x(n-M+k+1)\right] \quad (7.13)$$

The plus sign is for a symmetric filter, and the minus sign for an anti-symmetric one. Figure 8 shows a realization of (7.12) for odd [8] M . This realization can be regarded as a specialization of the direct form II shown in Figure 4 to an FIR filter. As we see, symmetry (or anti-symmetry) helps reducing the number of multiplications from M to $\lceil M/2 \rceil$, the number of additions, however, is still $M-1$.

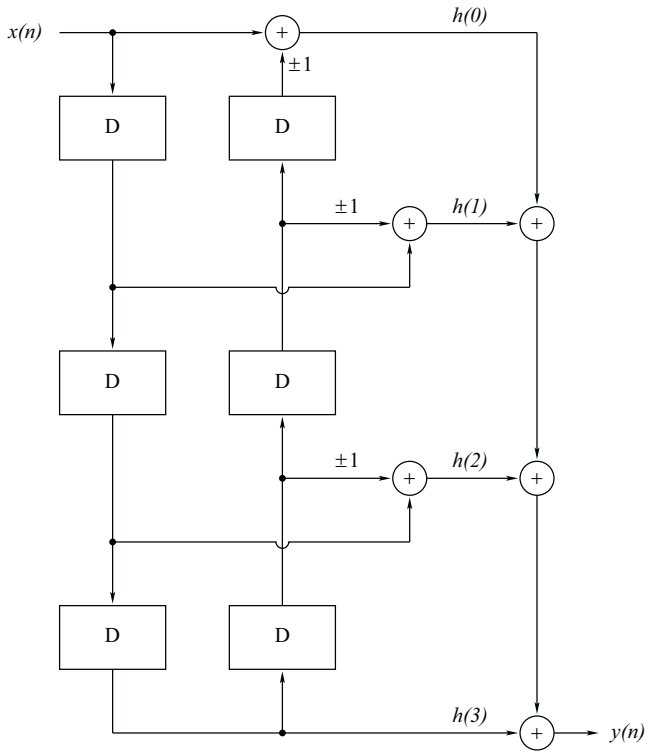

Figure 8 Direct realization of a symmetric or anti-symmetric FIR filter with odd M

By exploiting the rules of transposition of realizations, we get from Figure 8 the transposed direct realization shown in Figure 9.

[7] For FIR filters with M coefficients we use the same symbols as in previous lessons that is $H(z)=\sum_{k=0}^{M-1}h(k)x(n-k)$

[8] Realizations for even length M require modifications according to (7.13)

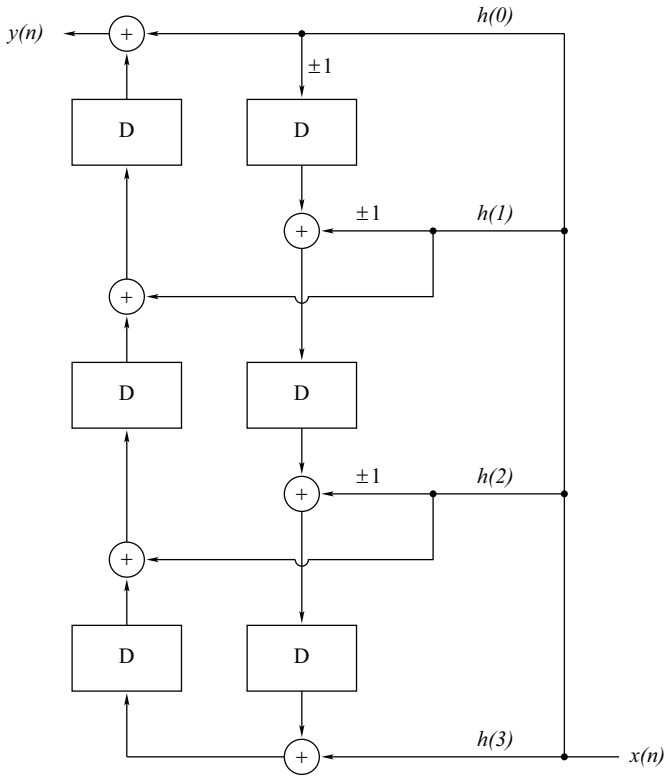

Figure 9 Transposed direct realization of a symmetric or anti-symmetric FIR filter with odd M

7.5 PARALLEL REALIZATION

Recall the partial fraction decomposition of a general rational causal transfer function whose poles are simple:

$$H(z)=\frac{\sum_{k=0}^{N} b(k)\,z^{-k}}{1+\sum_{k=1}^{M} a(k)\,z^{-k}}=c(0)+\ldots+c(N-M)\,z^{-(N-M)}+\sum_{k=1}^{M}\frac{A_k}{1-p_k z^{-1}} \tag{7.14}$$

For most practical digital IIR filters $N \le M$, so the right side of (7.14) contains only coefficient $c(0)$. Also, if p_k is complex then A_k is complex as well, and the conjugate fraction (* denotes complex conjugate)

$$\frac{A_k^*}{1-p_k^* z^{-1}} \tag{7.15}$$

also appears on the right side. The two terms can be combined together under common denominator, yielding the real fraction

$$\frac{\left(A_k + A_k^*\right) - \left(A_k p_k^* + A_k^* p_k\right) z^{-1}}{1 - \left(p_k + p_k^*\right) z^{-1} + p_k p_k^* z^{-2}} \tag{7.16}$$

Let us denote the order of the filter by M. Also, let M_1 be the number of real poles and $2M_2$ the number of complex poles, so

$$M = M_1 + 2M_2 \tag{7.17}$$

After joining complex conjugate fractions, we can bring (7.14) to the form

$$H(z) = c(0) + \sum_{k=1}^{M_1} \frac{f(k)}{1 + e(k) z^{-1}} + \sum_{k=1}^{M_2} \frac{f\left(M_1 + 2k - 1\right) + f\left(M_1 + 2k\right) z^{-1}}{1 + e\left(M_1 + 2k - 1\right) z^{-1} + e\left(M_1 + 2k\right) z^{-2}} \tag{7.18}$$

where the real numbers $\left\{e(k), f(k), 1 \le k \le M\right\}$ depend on $\left\{A_k, p_k, 1 \le k \le M\right\}$. The sum of the right side of (7.18) corresponds to a parallel connection of the individual terms. Each of the first and second order terms can be implemented by a direct realizations described in the previous parts. The constant term $c(0)$ is realized by simple multiplication. Figure 10 illustrates the result for $M_1 = 1$, $M_2 = 1$, $M = 3$. The realization thus obtained is called **parallel realization**.

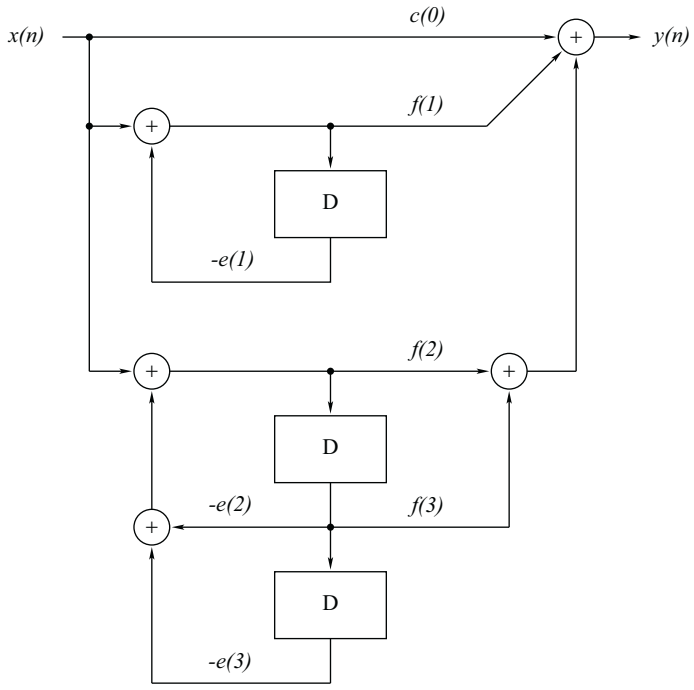

Figure 10 Parallel realization of digital IIR system

Parallel realization requires the same number of delay elements as the direct realizations. If $N = M$, it also requires the same amount of additions and multiplications. If $N < M$, the direct realizations are more economical, since in this case they require only $N + M + 1$ multiplications and additions, whereas parallel

realization still requires $2M+1$ operations of each kind[9]. As we have said, the parallel realization is limited to systems whose poles are simple. It can be extended to the case of multiple poles, but then a parallel realization is rarely used[10]. The advantage of parallel realization over direct realizations is the **lower sensitivity** of the frequency response of the filter to **finite word length**.

The procedure **tf2rpf** in the Appendix computes the parallel decomposition (7.18) of a digital IIR filter.

7.6 CASCADE REALIZATION

7.6.1 BASIC PRINCIPLE AND FEATURES

Let us assume again that $M = N$ for the digital filter in question. Assume for now that N is even. Recall the pole-zero factorization of the transfer function

$$H(z) = b(0)\frac{\prod_{k=1}^{N}\left(1 - z_k z^{-1}\right)}{\prod_{k=1}^{N}\left(1 - p_k z^{-1}\right)} \tag{7.19}$$

Since N is even, we can always rewrite (7.19) as

$$H(z) = b(0)\prod_{k=1}^{N/2}\frac{1 + h(2k-1)z^{-1} + h(2k)z^{-2}}{1 + g(2k-1)z^{-1} + g(2k)z^{-2}} \tag{7.20}$$

The second order factors in the numerator and the denominator are obtained by expanding conjugate pairs of zeros or poles, hence the coefficients $\{g(k), h(k), 1 \le k \le N\}$ are real. In general, some poles or zeros may be real. Since we have assumed that N is even, their number must be even, so we can expand them in pairs as well. Thus in general, the second order terms in (7.20) may correspond to either real or complex pairs of poles or zeros.

A product of transfer functions represents a cascade connection of the factors. Therefore we can implement (7.20) as a **cascade** connection of $N/2$ sections, each of order 2. Each section can be realized by either of the direct realizations. Figure 11 illustrates the connection for $N = 4$. This is a **cascade realization**. Note that constant gain $b(0)$ can appear anywhere along the cascade (in Figure 11 it appears in the middle between two sections). The second order sections are also called **bi-quads**.

[9] The reason is that all the coefficients $f(k)$ will be nonzero in general.

[10] Parallel realization of a system with multiple poles has a mixed parallel/series structure.

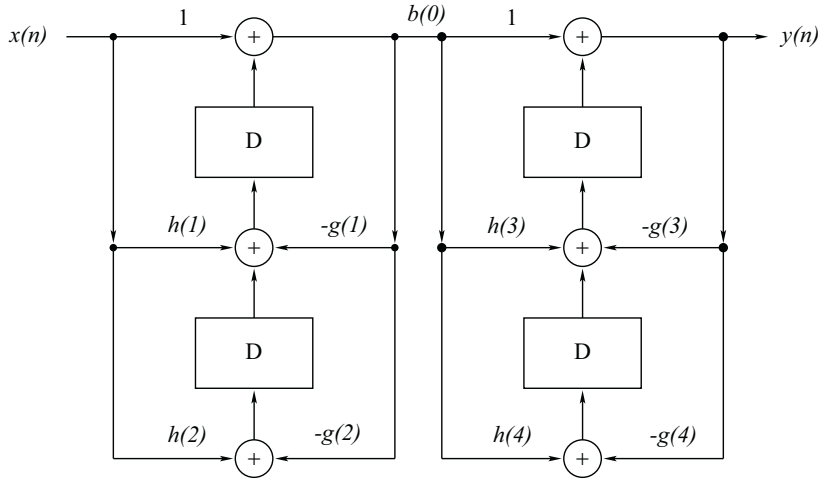

Figure 11 Cascade realization of a digital IIR system

Remarks for cascade realization

1. Although we have assumed that N is even, the realization can be easily extended to the case of odd N. In this case there is an extra first-order term, so we must add a first order section in cascade.
2. Although we have assumed that $N = M$, this condition is not necessary. Extra poles can be represented by section with zero values for the $\{h(k)\}$ coefficients, whereas extra zeros can be represented by sections with zero values for the $\{g(k)\}$ coefficients.
3. The realization is minimal in terms of number of delays, additions and multiplications (with the understanding that zero-valued coefficients save the corresponding multiplications and additions).
4. The realization is **nonunique**, since:
    a) There are multiple ways of pairing each second-order term in the denominator with one in the numerator. In the next section we discuss the pairing problem in detail.
    b) There are multiple ways of ordering the sections in the cascade connection.
    c) There are multiple ways of inserting the constant gain factor $b(0)$.
5. Contrary to the parallel realization, the cascade realization is not limited to simple poles. Moreover, it does not require condition $N \le M$. Cascade realization is applicable to FIR filters, although its use for such filters is relatively uncommon.

7.6.2 PAIRING IN CASCADE REALIZATION

When cascade realization is implemented in **floating point** and at a high precision (such as in MATLAB), the pairing of poles and zeros to second order sections is of little importance. However, in **fixed point** implementations and short word lengths, it is advantageous to pair poles and zeros to produce a frequency response for each section that is as flat as possible (i.e., such that the ratio of the maximum to the minimum magnitude response is close to unity) . We now describe a pairing procedure that approximately achieves this goal. We consider only digital filters obtained from one of

the four standard filter types (Butterworth, Chebyshev-I, Chebyshev-II, elliptic) through an analog frequency transformation followed by a bilinear transform. Such filters satisfy the following properties:

1. The number of zeros is equal to the number of poles. If the underlying analog filter has more poles than zeros, the extra zeros of the digital filter are all at $z = -1$.
2. The number of complex poles is newer smaller than the number of complex zeros.
3. The number of real poles is not larger than 2. A low-pass filter has one real pole if its order is odd, and this pole may be transformed to two real poles or to a pair of complex poles by either a low-pass to band-pass or low-pass to band-stop transformation. Except for those, all poles of the analog filter are complex, hence so are poles of the digital filter.

The basic idea is to pair each pole with a zero as close to it as possible. This makes the magnitude response of the pole-zero pair as flat as possible. The pairing procedure starts at the pair of complex poles nearest to the unit circle (i.e., those with the largest absolute value) and pair them with the nearest complex zeros. It then removes these two pairs from the list and proceeds according to the same rule. When all the complex zeros are exhausted, pairing continues with the real zeros according to the same rule. Finally, there may be left up to two real poles, and these are paired with the remaining real zeros.

The procedure **pairpz** in the Appendix implements this algorithm. It receives the vectors of poles and zeros, supplied by the program **iirdes** and supplies arrays of second order numerator and denominator polynomials (a first order pair, if any, is represented as a second order pair with zero coefficient of z^{-2}). The routine **cplxpair** is a built-in MATLAB function that orders the poles (or zeros) in conjugate pairs, with real ones (if any) at the end. The program then selects one representative of each conjugate pair and sorts them in decreasing order of magnitude. Next the program loops over the complex poles and, for each one, finds the nearest complex zero. Every paired zero is removed from the list. The polynomials of the corresponding second order section are computed and stored. When the complex zeros are exhausted, the remaining complex poles are paired with the real zeros using the same procedure. Finally, the real poles are paired with the remaining real zeros.

The procedure **cascade** in the Appendix implements the cascade realization of digital IIR filter. It accepts the parameters computed by the program pairpz. The input sequence is fed to the first section, the output is fed to the second section, and so forth. Finally, the result is multiplied by the constant gain.

The cascade realization is usually considered as the best of all those we have discussed[11], therefore it is the most widely used.

[11] There are some other realizations e.g. state space realizations of digital filters that have very good numerical properties.

APPENDIX - MATLAB PROGRAMS

The MATLAB software is from the book [1] and available by anonymous file transfer protocol (ftp) from:
ftp.wiley.com/public/college/math/matlab/bporat
ftp.technion.ac.il/pub/supported/ee/Signal_processing/B_Porat

```
function y = direct(typ,b,a,x);
% Synopsis: direct(typ,b,a,x).
% Direct realizations of rational transfer functions.
% Input parameters:
% typ: 1 for direct realization, 2 for transposed
% b, a: numerator and denominator polynomials
% x: input sequence.
% Output:
% y: output sequence.

p = length(a)-1; q = length(b)-1; pq = max(p,q);
a = a(2:p+1); u = zeros(1,pq); % u: the internal state
if (typ == 1),
  for i = 1:length(x),
    unew = x(i)-sum(u(1:p).*a);
    u = [unew,u];
    y(i) = sum(u(1:q+1).*b);
    u = u(1:pq);
  end
elseif (typ == 2),
  for i = 1:length(x),
    y(i) = b(1)*x(i)+u(1);
    u = [u(2:pq),0];
    u(1:q) = u(1:q) + b(2:q+1)*x(i);
    u(1:p) = u(1:p) - a*y(i);
  end
end
```

```
function [c,nsec,dsec] = tf2rpf(b,a);
% Synopsis: [c,nsec,dsec] = tf2rpf(b,a).
% Real partial fraction decomposition of b(z)/a(z). The polynomials
% are in negative powers of z. The poles are assumed distinct.
% Input parameters:
% a, b: the input polynomials
% Output parameters:
% c: the free polynomial; empty if deg(b) < deg(a)

nsec = []; dsec = []; [c,A,alpha] = tf2pf(b,a);
while (length(alpha) > 0),
  if (imag(alpha(1)) ~= 0),
    dsec = [dsec; [1,-2*real(alpha(1)),abs(alpha(1))^2]];
    nsec = [nsec; [2*real(A(1)),-2*real(A(1)*conj(alpha(1)))]];
    alpha(1:2) = []; A(1:2) = [];
  else,
    dsec = [dsec; [1,-alpha(1),0]]; nsec = [nsec; [real(A(1)),0]];
    alpha(1) = []; A(1) = [];
  end
end
```

```
function y = parallel(c,nsec,dsec,x);
% Synopsis: y = parallel(c,nsec,dsec,x).
% Parallel realization of an IIR digital filter.
% Input parameters:
% c: the free term of the filter.
% nsec, dsec: numerators and denominators of second-order sections
% x: the input sequence.
% Output:
% y: the output sequence.

 [n,m] = size(dsec); dsec = dsec(:,2:3);
u = zeros(n,2); % u: the internal state
for i = 1:length(x),
   y(i) = c*x(i);
   for k = 1:n,
      unew = x(i)-sum(u(k,:).*dsec(k,:)); u(k,:) = [unew,u(k,1)];
      y(i) = y(i) + sum(u(k,:).*nsec(k,:));
   end
end
```

```
function y = cascade(C,nsec,dsec,x);
% Synopsis: y = cascade(C,nsec,dsec,x).
% Cascade realization of an IIR digital filter.
% Input parameters:
% C: the constant gain of the filter.
% nsec, dsec: numerators and denominators of second-order sections
% x: the input sequence.
% Output:
% y: the output sequence.

 [n,m] = size(dsec);
u = zeros(n,2); % u: the internal state
dsec = dsec(:,2:3); nsec = nsec(:,2:3);
for i = 1:length(x),
   for k = 1:n,
      unew = x(i)-sum(u(k,:).*dsec(k,:));
      x(i) = unew + sum(u(k,:).*nsec(k,:));
      u(k,:) = [unew,u(k,1)];
   end
   y(i) = C*x(i);
end
```

```
function [nsec,dsec] = pairpz(v,u);
% Synopsis: [nsec,dsec] = pairpz(v,u).
% Pole-zero pairing for cascade realization.
% Input parameters:
% v, u: the vectors of poles and zeros, respectively.
% Output parameters:
```

```
% nsec: matrix of numerator coefficients of 2nd-order sections
% dsec: matrix of denom. coefficients of 2nd-order sections.

if (length(v) ~= length(u)),
  error('Different numbers of poles and zeros in PAIRPZ'); end
u = reshape(u,1,length(u)); v = reshape(v,1,length(v));
v = cplxpair(v); u = cplxpair(u);
vc = v(find(imag(v) > 0)); uc = u(find(imag(u) > 0));
vr = v(find(imag(v) == 0)); ur = u(find(imag(u) == 0));
[temp,ind] = sort(abs(vc)); vc = vc(fliplr(ind));
[temp,ind] = sort(abs(vr)); vr = vr(fliplr(ind));
nsec = []; dsec = [];
for n = 1:length(vc),
   dsec = [dsec; [1,-2*real(vc(n)),abs(vc(n))^2]];
   if (length(uc) > 0),
      [temp,ind] = min(abs(vc(n)-uc)); ind = ind(1);
      nsec = [nsec; [1,-2*real(uc(ind)),abs(uc(ind))^2]];
      uc(ind) = [];
   else,
      [temp,ind] = min(abs(vc(n)-ur)); ind = ind(1);
      tempsec = [1,-ur(ind)]; ur(ind) = [];
      [temp,ind] = min(abs(vc(n)-ur)); ind = ind(1);
      tempsec = conv(tempsec,[1,-ur(ind)]); ur(ind) = [];
      nsec = [nsec; tempsec];
   end
end
if (length(vr) == 0), return
elseif (length(vr) == 1),
   dsec = [dsec; [1,-vr,0]]; nsec = [nsec; [1,-ur,0]];
elseif (length(vr) == 2),
   dsec = [dsec; [1,-vr(1)-vr(2),vr(1)*vr(2)]];
   nsec = [nsec; [1,-ur(1)-ur(2),ur(1)*ur(2)]];
else
   error('Something wrong in PAIRPZ, more than 2 real zeros');
end


function [b,a,v,u,C] = iirdes(typ,band,theta,deltap,deltas);
% Synopsis: [b,a,v,u,C] = iirdes(typ,band,theta,deltap,deltas).
% Designs a digital IIR filter to meet given specifications.
% Input parameters:
% typ: the filter type: 'but', 'ch1', 'ch2', or 'ell'
% band: 'l' for LP, 'h' for HP, 'p' for BP, 's' for BS
% theta: an array of band-edge frequencies, in increasing
%        order; must have 2 frequencies if 'l' or 'h',
%        4 if 'p' or 's'
% deltap: pass-band ripple/s (possibly 2 for 's')
% deltas: stop-band ripple/s (possibly 2 for 'p')
% Output parameters:
% b, a: the output polynomials
% v, u, C: the output poles, zeros, and constant gain.

% Prewarp frequencies (with T = 1)
omega = 2*tan(0.5*theta);
% Transform specifications
if (band == 'l'), wp = omega(1); ws = omega(2);
elseif (band == 'h'), wp = 1/omega(2); ws = 1/omega(1);
elseif (band == 'p'),
   wl = omega(2); wh = omega(3); wp = 1;
   ws = min(abs((omega([1,4]).^2-wl*wh) ...
        ./((wh-wl)*omega([1,4]))));
elseif (band == 's'),
   wl = omega(2); wh = omega(3); ws = 1;
   wp = 1/min(abs((omega([1,4]).^2-wl*wh) ...
        ./((wh-wl)*omega([1,4]))));
end
% Get low-pass filter parameters
[N,w0,epsilon,m] = lpspec(typ,wp,ws,min(deltap),min(deltas));
% Design low-pass filter
[b,a,v1,u1,C1] = analoglp(typ,N,w0,epsilon,m);
% Transform to the required band
ww = 1; if (band == 'p' | band == 's'), ww = [wl,wh]; end
[b,a,v2,u2,C2] = analogtr(band,v1,u1,C1,ww);
% Perform bilinear transformation
[b,a,v,u,C] = bilin(v2,u2,C2,1);
```

LITERATURE

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