Realizations of Digital Filters

3. 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 \( M \), we can write the filter’s output as:

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

(1)

whereas for an even number of coefficients we put

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

(2)

The plus sign is for a symmetric filter and the minus sign for an anti-symmetric one. Figure 8 shows a realization of (eq. 1) for odd \( M \). This realization can be regarded as a specialization of the direct form II shown in (Figure 2 Lect. 12) to an FIR filter. As we see, symmetry (or anti-symmetry) helps reducing the number of multiplications from \( M \) to \( M/2 \) the number of additions, however, is still \( M - 1 \).
Figure 1: 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 1 the transposed direct realization shown in Figure 2.
Figure 2: Transposed direct realization of a symmetric or anti-symmetric FIR filter with odd M.
4. PARALLEL REALIZATION:

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

\[
H(z) = \frac{\sum_{k=0}^{N} b(k) z^{-k}}{1 + \sum_{k=1}^{M} a(k) z^{-k}} = c(0) + \ldots + c(N-M) z^{-(N-M)} + \sum_{k=1}^{M} \frac{A_k}{1 - p_k z^{-1}}
\]  \hspace{1cm} (3)

For most practical digital IIR filters \( N \leq M \), so the right side of (eq. 3) 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)

\[
\frac{A_k^*}{1 - p_k^* z^{-1}}
\]  \hspace{1cm} (4)

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

\[
\frac{\left( A_k + A_k^* \right) - \left( A_k p_k^* + A_k^* p_k \right) z^{-1}}{1 - \left( p_k + p_k^* \right) z^{-1} + p_k p_k^* z^{-2}}
\]  \hspace{1cm} (5)

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

\[
M = M_1 + 2M_2
\]  \hspace{1cm} (6)

After joining complex conjugate fractions, we can bring (eq. 3) to the form

\[
H(z) = c(0) + \sum_{k=1}^{M_1} \frac{f(k)}{1 + e(k) z^{-1}} + \sum_{k=1}^{M_2} \frac{f(M_1 + 2k - 1) + f(M_1 + 2k) z^{-1}}{1 + e(M_1 + 2k - 1) z^{-1} + e(M_1 + 2k) z^{-2}}
\]  \hspace{1cm} (7)

By: Zeyad Al-Hamdany
Where the real numbers \( \{e(k), f(k), 1 \leq k \leq M\} \) depend on \( \{A_k, p_k, 1 \leq k \leq M\} \). The sum of the right side of (eq. 7) 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 is realized by simple multiplication. Figure 3 illustrates the result for \( M1=1, M2=1, M=3 \). The realization thus obtained is called **parallel realization**.

![Parallel realization of digital IIR system.](image)

**Figure 3: 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 (*The reason is that all the coefficients \( f(k) \) will be nonzero in general*). 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 (*Parallel realization of a system with multiple poles has a mixed parallel/series structure*). 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 (eq. 7) of a digital IIR filter.

5. **CASCADE REALIZATION:**

5.1 **Basic Principle and Features.**

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

\[
H(z) = b(0) \frac{\prod_{k=1}^{N} (1 - z_k z^{-1})}{\prod_{k=1}^{N} (1 - p_k z^{-1})}
\]  

(8)

Since \( N \) is even, we can always rewrite (eq. 8) as

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

(9)
The second order factors in the numerator and the denominator are obtained by expanding conjugate pairs of zeros or poles, hence the coefficients \( \{g(k), h(k), 1 \leq k \leq N\} \) 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 (eq.9) 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 (eq.9) as a \textit{cascade} connection of \( N/2 \) sections, each of order 2. Each section can be realized by either of the direct realizations. Figure 4 illustrates the connection for \( N=4 \). This is a \textit{cascade realization}. Note that constant gain \( g(0) \) can appear anywhere along the cascade (in Figure 4 it appears in the middle between two sections). The second order sections are also called \textit{bi-quads}.

![Figure 4: Cascade realization of a digital IIR system.](image-url)
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 $N \leq M$. Cascade realization is applicable to FIR filters, although its use for such filters is relatively uncommon.

5.2 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.
**Note:** The cascade realization is usually considered as the best of all those we have discussed therefore it is the most widely used.

**Note:** There are some other realizations e.g. state space realizations of digital filters that have very good numerical properties.

**Design Codes in Matlab (In Appendix):**

- 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.
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 [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
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);
v = v(find(imag(v) > 0)); u = u(find(imag(u) > 0));
v = v(find(imag(v) == 0)); u = 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(vc) > 0),
        [temp,ind] = min(abs(vc(n)-vc)); ind = ind(1);
        nsec = [nsec, [1,-2*real(vc(ind)),abs(vc(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
end

By: Zeyad Al-Hamdany
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) ...
            ./((wl-wl)*omega([1,4]))));
elseif (band == 's'),
    wp = 1/min(abs((omega([1,4]).^2-wl*wh) ...
            ./((wl-wl)*omega([1,4]))));
end
% Get low-pass filter parameters
[N,w0,epsilon,m] = Ipspec(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);

Reference