ONE TECHNOLOGY WAY • P.O. BOX 9106 • NORWOOD, MASSACHUSETTS 02062-9106 • 617/329-4700

# Audio Applications of the ADSP Family (IIR Filters)

## INFINITE IMPULSE RESPONSE (IIR) FILTERS

Compared to the FIR filter, an IIR filter can often be much more efficient in terms of attaining certain performance characteristics with a given filter order. This is because the IIR filter incorporates feedback and is capable of realizing both poles and zeros of a system transfer function, whereas the FIR filter is only capable of realizing the zeros (although the FIR filter is still more desirable in many applications, because of features such as stability and the ability to realize exactly linear phase responses).

### Direct Form IIR Filter

The IIR filter can realize both the poles and zeros of a system because it has a rational transfer function, described by polynomials in $z$ in both the numerator and the denominator:

$$H(z) = \frac{\sum_{k=0}^{M} b_k z^{-k}}{1 - \sum_{k=1}^{N} a_k z^{-k}}$$

The difference equation for such a system is described by the following:

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

In most applications, the order of the two polynomials M and N are the same.

The roots of the denominator determine the pole locations of the filter, and the roots of the numerator determine the zero locations. There are, of course, several means of implementing the above transfer function with an IIR filter structure. The "direct form" structure presented in Listing 1 implements the difference equation above.

Note that there is a single delay line buffer for the recursive and nonrecursive portions of the filter (Oppenheim and Schafer's Direct Form II). The sum-of-products of the $a$ values and the delay line values are first computed, followed by the sum-of-products of the $b$ values and the delay line values.

```
.MODULE  diriir_sub;

{
  Direct Form II IIR Filter Subroutine

  Calling Parameters
          MR1 = input sample (x[n])
          MR0 = 0
          I0 → Delay line buffer current location (x[n-1])
          L0 = Filter length
          I5 → Feedback coefficients (a[1], a[2], . . . a[N])
          L5 = Filter length - 1
          I6 → Feedforward coefficients (b[0], b[1], . . . b[N])
          L6 = Filter length
          M0 = 0
          M1,M4 = 1
          CNTR = Filter length - 2
          AX0 = Filter length - 1

  Return Values
          MR1 = output sample (y[n])
          I0 → delay line current location (x[n-1])
          I5 → feedback coefficients
          I6 → feedforward coefficients

  Altered Registers
          MX0,MY0,MR

  Computation Time
          ((N - 2) + (N - 1)) + 10 + 4 cycles (N = M = Filter order)

  All coefficients and data values are assumed to be in 1.15 format.

}

.ENTRY diriir;

diriir:     MX0=DM(I0,M1), MY0=PM(I5,M4);
            DO poleloop UNTIL CE;
poleloop:   MR=MR+MX0*MY0(SS), MX0=DM(I0,M1), MY0=PM(I5,M4);
            MR=MR+MX0*MY0(RND);
            CNTR=AX0;
            DM(I0,M0)=MR1;
            MR=0, MX0=DM(I0,M1), MY0=PM(I6,M4);
            DO zeroloop UNTIL CE;

zeroloop:   MR=MR+MX0*MY0(SS), MX0=DM(I0,M1), MY0=PM(I6,M4);
            MR=MR+MX0*MY0(RND);
            MODIFY (I0,M2);
            RTS;

.ENDMOD;
```

*Listing 1. Direct Form IIR Filter*

**Cascaded Biquad IIR Filter**

A second-order biquad IIR filter section is shown on Figure 1. Its transfer function in the z-domain is:

$$H(z) = Y(z)/X(z) = (B_0 + B_1 z^{-1} + B_2 z^{-2})/(1 + A_1 z^{-1} + A_2 z^{-2})$$

where $A_1$, $A_2$, $B_0$, $B_1$ and $B_2$ are coefficients that determine the desired impulse response of the system H(z). Furthermore, the corresponding difference equation for a biquad section is:

$$Y(n) = B_0 X(n) + B_1 X(n-1) + B_2 X(n-2) - A_1 Y(n-1) - A_2 Y(n-2)$$

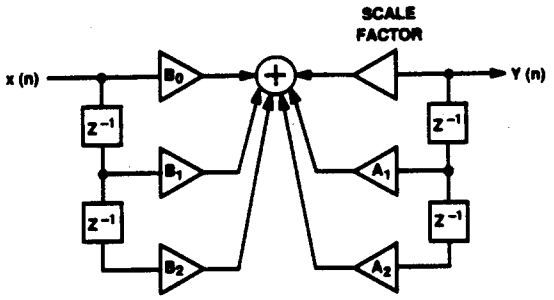


*Figure 1. Second-Order Biquad IIR Filter Section*

Higher-order filters can be obtained by cascading several biquad sections with appropriate coefficients. Another way to design higher-order filters is to use only one complicated single section. This approach is called the direct form implementation. The biquad implementation executes slower but generates smaller numerical errors than the direct form implementation. The biquads can be scaled separately and then cascaded to minimize the coefficient quantization and the recursive accumulation errors. The coefficients and data in the direct form implementation must be scaled all at once, which gives rise to larger errors. Another disadvantage of the direct form implementation is that the poles of such single-stage high-order polynomials get increasingly sensitive to quantization errors. The second-order polynomial sections (i.e., biquads) are less sensitive to quantization effects.

An ADSP-2100 subroutine that implements a high-order filter is shown in Listing 2. The subroutine is arranged as a module and is labeled *biquad_sub*. There are a number of registers that need to be initialized in order to execute this subroutine. It may be sufficient to do this initialization only once (e.g., at power-up) if other executed algorithms do not need these registers. In most typical cases, however, some of these registers may need to be set every time the *biquad_sub* routine is called. It may sometimes be beneficial, from a modular software point of view, always to initialize all the setup registers as a part of this subroutine.

The *biquad_sub* routine takes its input from the SR1 register. This register must contain the 16-bit input X(n). X(n) is assumed to be already computed before this subroutine is called. The output of the filter is also made available in the SR1 register.

After the initial design of a high-order filter, all coefficients must be scaled down separately in each biquad stage. This is necessary in order to conform to the 16-bit fixed-point fractional number format as well as to ensure that overflows will not occur in the final multiply-accumulate operations in each stage. The scaled-down coefficients are the ones that get stored in the processor's memory. The operations in each biquad are performed with scaled data and coefficients and are eventually scaled up before being output to the next one. The choice of a proper scaling factor depends greatly on the design objectives, and in some cases it may even be unnecessary. The filter coefficients are usually designed with a commercial software package in higher than 16-bit precision arithmetic. System performance deviates from ideal when such high precision coefficients are quantized to 16 bits and further scaled down. In systems that require stringent specifications, careful simulations of quantization and scaling effects must be performed.

During the initialization of the *biquad_sub* routine, the index register I0 points to the data memory buffer that contains the previous error inputs and the previous biquad section outputs. This buffer must be initialized to zero at powerup unless some nonzero initial condition is desired. The index register I1 points to another buffer in data memory that contains the individual scale factors for each biquad. The buffer length register L1 is set to zero if the filter has only one biquad section. In the case of multiple biquads, L1 is initialized with the number of biquad sections. The index register I4, on the other hand, points to the circular program memory buffer that contains the scaled biquad coefficients. These coefficients are stored in the order: $B_2$, $B_1$, $B_0$, $A_2$ and $A_1$ for each biquad. All of the individual biquad coefficient groups must be stored in the same order in which the biquads are cascaded, such as: $B_2$, $B_1$, $B_0$, $A_2$, $A_1$, $B_2$*, $B_1$*, $B_0$*, $A_2$*, $A_1$*, $B_2$**, etc. The buffer length register L4 must be set to the value of (5 × number of biquad sections). Finally, the loop counter register CNTR must be set to the number of biquad sections, since the filter code will be executed as a loop.

The core of the *biquad_sub* routine starts its execution at the *biquad* label. The routine is organized in a looped fashion where the end of the loop is the instruction labeled *sections*. Each iteration of the loop executes the computations for one biquad. The number of loops to be executed is determined by the CNTR register contents. The SE register is loaded with the appropriate scaling factor for the particular biquad at the beginning of each loop iteration. After this operation, the coefficients and the data values are fetched from memory in the sequence in which they have been stored. These numbers are multiplied and accumulated until all of the values for a particular biquad have been accessed. The result of the last multiply/accumulate is rounded to 16 bits and upshifted by the scaling value. At this point, the *biquad* loop is executed again, or the filter computations are completed by doing the final update to the delay line.

The delay lines for data values are always being updated within the *biquad* loop as well as outside of it.

The filter coefficients must be scaled appropriately so that no overflows occur after the upshifting operation between the biquads. If this is not ensured by design, it may be necessary to include some overflow checking between the biquads.

The execution time for an Nth order *biquad_sub* routine can be calculated as follows (assuming that the appropriate registers have been initialized and N is a power of 2):

ADSP-2101/2102: $(8 \times N/2) + 4$ processor cycles
ADSP-2100/2100A: $(8 \times N/2) + 4 + 5$ processor cycles

It may take up to a maximum of 12 cycles to initialize the appropriate registers every time the filter is called, but typically this number will be lower.

```
.MODULE biquad_sub;

{ Nth order cascaded biquad filter subroutine

  Calling Parameters:

        SR1 = input X(n)
        I0 → delay line buffer for X(n-2), X(n–1),
             Y(n-2), Y(n-1)
        L0 = 0
        I1 → scaling factors for each biquad section
        L1 = 0 (in the case of a single biquad)
        L1 = number of biquad sections
             (for multiple biquads)
        I4 → scaled biquad coefficients
        L4 = 5 × [number of biquads]
        M0, M4 = 1
        M1 = –3
        M2 = 1 (in the case of multiple biquads)
        M2 = 0 (in the case of a single biquad)
        M3 = (1 – length of delay line buffer)

  Return Value:
        SR1 = output sample Y(n)

  Altered Registers:
        SE, MX0, MX1, MY0, MR, SR

  Computation Time (with N even):
        ADSP-2101/2102: (8 × N/2) + 5 cycles
        ADSP-2100/2100A: (8 × N/2) + 5 + 5 cycles

  All coefficients and data values are assumed to be in 1.15 format

}

.ENTRY biquad;

biquad: CNTR = number_of_biquads
        DO sections UNTIL CE;
          SE=DM(I1,M2);
          MX0=DM(I0,M0), MY0=PM(I4,M4);
          MR=MX0*MY0(SS), MX1=DM(I0,M0), MY0=PM(I4,M4);
          MR=MR+MX1*MY0(SS), MY0=PM(I4,M4);
          MR=MR+SR1*MY0(SS), MX0=DM(I0,M0), MY0=PM(I4,M4);
          MR=MR+MX0*MY0(SS), MX0=DM(I0,M1), MY0=PM(I4,M4);
          DM(I0,M0)=MX1, MR=MR+MX0*MY0(RND);
sections: DM(I0,M0)=SR1, SR=ASHIFT MR1 (HI);
        DM(I0,M0)=MX0;
        DM(I0,M3)=SR1;
        RTS;

.ENDMOD;
```

*Listing 2. Cascaded Biquad IIR Filter*